

## 第1章 概 述

在过去的三个世纪中,每一个世纪都有一种占主导地位的技术。18世纪伴随着工业革命到来的是大型机械系统的时代;19世纪是蒸汽机的时代;而在20世纪的发展历程中,关键的技术是信息收集、处理和分发。在其他的发展方面,我们还可以看到:遍布全球的电话网络建立起来了;无线电广播和电视出现了;计算机工业诞生了,并且以超乎想象的速度在增长;另外,通信卫星也发射上天了。

技术快速发展的一个直接结果是,这些领域正在快速地融合,信息收集、传输、存储和处理之间的差别正在迅速地消失。对于具有数百个办公室的大型组织来说,尽管这些办公室分布在广阔的地理区域中,但未来期望的情景是,工作人员只要单击一下按钮,就可以查看到最远处分部的状态。随着信息收集、处理和分发能力的不断提高,我们对于更加复杂的信息处理技术的需求也增长得更快。

与其他的工业(比如汽车和航空运输业)相比,计算机工业还非常年轻,尽管如此,计算机技术却在很短的时间内有了惊人的进展。在计算机诞生之初的20年间,计算机系统是高度集中化的,通常位于一个很大的房间中。该房间通常配有玻璃墙,参观的人透过玻璃墙可以欣赏到里边伟大的电子奇迹。中等规模的公司或者大学可能会有一台或者两台计算机,而大型的研究机构最多也就几十台计算机。要在20年内生产出大量同样功能但是体积比邮票还小的计算机,在当时的人们看来纯属科学幻想。

计算机和通信的结合对于计算机系统的组织方式产生了深远的影响。把一台大型的计算机放在一个单独的房间中,然后用户带着他们的处理任务去房间里上机,这种“计算机中心”的概念现在已经完全过时了。由一台计算机来处理整个组织中所有的计算需求,这种老式的模型已经被新的模型所取代,在新的模型下,由大量独立的、但相互连接起来的计算机来共同完成计算任务。这些系统称为**计算机网络(computer networks)**。如何设计这些网络,并且将这些网络组织起来,这是本书的主题。

在本书中,我们将使用术语“计算机网络”来表示通过同一种技术相互连接起来的一组自主计算机的集合。如果两台计算机能够交换信息,则称这两台计算机是相互连接的(interconnected)。两台机器之间的连接不一定要通过铜线,光纤、微波、红外线和通信卫星也可以用来建立连接。以后我们将会看到,网络可以有不同的大小、形状和形式。Internet或者万维网(World Wide Web)都不是计算机网络,可能很多人对此会觉得很奇怪。到本书末尾的时候,你就会明白其中的原因。现在给出一个简单的答案:Internet并不是一个单一的网络,而是一个由许多个网络构成的网络;Web是一个分布式系统,它运行在Internet之上。

在一些文献中,计算机网络和分布式系统(distributed system)这两个概念容易使人混

---

淆。两者的关键差别在于：在一个分布式系统中，一组独立的计算机展现给用户的的是一个统一的整体，就好像是一个系统似的。通常，对用户来说，分布式系统只有一个模型或范型。在操作系统之上有一层软件中间件(middleware)负责实现这个模型。一个著名的分布式系统的例子是万维网(World Wide Web)，在万维网中，所有的一切看起来就好像是一个文档(Web 页面)一样。

在计算机网络中，这种统一性、模型以及其中的软件都不存在。用户看到的是实际的机器，计算机网络并没有使这些机器看起来是统一的，或者使它们的行为是统一的。如果这些机器有不同的硬件或者不同的操作系统，那么，这些差异对于用户来说都是完全可见的。如果一个用户希望在一台远程机器上运行一个程序，那么，他<sup>①</sup>必须登录到远程机器上，然后在那台机器上运行该程序。

实际上，分布式系统是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。因此，网络与分布式系统之间的区别更多地在于软件(特别是操作系统)，而不是硬件。

然而，这两个主题之间也有许多重合的地方。例如，分布式系统和计算机网络都需要移动文件。不同之处在于是谁来发起移动操作，是系统还是用户？虽然本书的焦点主要在于网络，但是讨论到的许多话题在分布式系统中也是很重要的。有关于分布式系统的更多信息，请参考(Tanenbaum and Van Steen, 2002)。

## 1.1 计算机网络的应用

在开始讨论技术细节之前，首先值得花一点时间来说明为什么人们对于计算机网络很感兴趣，以及计算机网络可用来做些什么事情。毕竟，如果没有人对计算机网络感兴趣的话，那就不会建立这么多计算机网络了。我们首先讨论针对公司和个人的传统用法，然后再转移到最新的一些发展动向，包括针对移动用户和家庭网络的应用上来。

### 1.1.1 商业应用

许多公司都具有相当数量的计算机。例如，一家公司可能用一些单独的计算机来监视生产过程、记录库存，以及管理工资的发放工作。最初的时候，这些计算机都是独立工作的，但是后来管理部门决定将这些计算机连接起来，以便将有关整个公司的信息关联起来，并且可以随时访问这些信息。

将这个公司的情形更加泛化一点，这里涉及到的问题是资源共享(resource sharing)，其目标是，让每一个人都可以访问所有的程序、设备和特殊的数据，并且做到跟这些资源和用户的物理位置无关。一个既显然又非常普遍的例子是，让一个办公室里的所有工作人员共用同一台打印机。公司没有必要为每一个工作人员都配备一台个人打印机，而且，一台高性能的网络打印机通常比一大批独立的打印机更加便宜，打印速度更快，而且也更容易维护。

---

<sup>①</sup> 在本书中，“他”应该被看作“他或她”。



然而,比共享物理资源(比如打印机、扫描仪和 CD 刻录机)更重要的是共享信息。每一个大型的或中等规模的公司和许多小型的公司都越来越依赖于计算机化的信息。大多数公司都有顾客记录、库存信息、收到的账单记录、财务报告、缴税信息以及其他更多的在线信息。如果一家银行的所有计算机都不能工作了,那么这家银行可能坚持不了 5 分钟。如果一个现代化的生产车间使用了计算机控制的装配线,那么计算机崩溃后也不可能继续工作。现在,即使是很小规模的旅行社,甚至只有三个人规模的律师事务所也与计算机网络有着密切的联系,通过计算机网络,雇员们可以即时地访问有关的信息和文档。

对于小公司而言,可能所有的计算机都在一个办公室里,或者位于同一个建筑物内,但是对于大型的公司,计算机和雇员们可能分散在许多个办公室中,甚至分散在不同国家的多个分支机构中。然而,纽约的一个销售员有时候需要访问新加坡的产品库存数据库。换句话说,一个用户离他要访问的数据相隔 15 000 公里,但是他仍然要访问这些数据,就好像这些数据存放在本地一样。简而言之,计算机网络的这个目标可以定义为:全图打破“地理位置的束缚(tyranny of geography)”。

按照最简单的形式,你可以把一个公司的信息系统想象成:由一个或者多个数据库,以及许多需要远程访问这些数据库的雇员们组成的。在这个模型中,数据存储在性能较强大的计算机上,称为服务器(server)。通常这些服务器集中在同一个场所,并且由系统管理员对它们进行维护。相反,雇员们的桌子上有一些简单的机器,称为客户(client),通过这些客户,雇员可以访问远程的数据,例如,他们可以访问远程的电子表格。(有时候,我们也把客户机器的使用者称作“客户”,但是根据上下文环境,你应该可以判断出到底是指机器,还是指机器的用户。)客户和服务器通过网络连接起来,如图 1.1 所示。请注意,这里我们只是用一个简单的椭圆来表示一个网络,而没有表达其中的任何细节。当我们从抽象意义上来表达一个网络的时候,就使用这种形式。当有需要的时候,我们也会提供更多的细节。

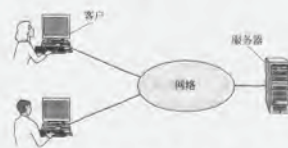


图 1.1 一个网络,包含两台客户机和一台服务器

这个结构称为“客户-服务器模型(client-server model)”。这种模型有很广泛的应用,它也是许多网络应用的基础。当客户和服务器位于同一个建筑物内(比如,属于同一个公司)的时候,这种模型是适用的;当客户和服务器相隔很远的时候,这种模型也是适用的。例如,当一个人在家里访问一个 Web 页面的时候,就使用了这种模型;其中,远程的 Web 服务器就是模型中的服务器,用户的个人计算机是模型中的客户。在大多数情况

下,一台服务器可以处理许多客户的请求。

如果更加仔细地看一看客户-服务器模型,我们就可以看到,该模型涉及到了两个进程,一个位于客户机器上,另一个位于服务器机器上。其中的通信形式是这样的:客户进程通过网络将一个消息发送给服务器进程。然后客户进程等待应答消息。当服务器进程获得了请求消息之后,它执行所请求的工作,或者查到客户所要的数据,然后送回一个应答消息。图 1.2 显示了这些消息。

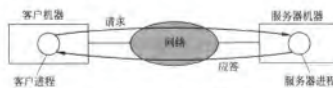


图 1.2 客户-服务器模型涉及到了请求和应答消息

建立计算机网络的第二个目标是为了满足人的需要,而不是信息或者计算机的要求。计算机网络可以为雇员们提供一个功能强大的通信媒介(communication medium)。现在,几乎每一家公司,只要有两台以上的计算机,就会有一个电子邮件(electronic mail, E-mail)系统,雇员们利用该系统来进行大量日常的通信。实际上,这样带来的一个问题是,每个人每天必须处理大量的 E-mail,而且,其中有一些 E-mail 是没有意义的,因为公司的老板们发现,他们只需点一下按钮就可以将同样的 E-mail 消息(通常并没有多少实际内容)发送给所有的下属。

但是, E-mail 并不是计算机网络带来的惟一一种新型的通信形式。有了网络之后,两个或者多个相距甚远的人可以联合起来写一份报告,这是非常容易做到的。当一个工作人员修改了一份在线文档的时候,其他的人立即就可以看到文档被改动了,根本不用为一封信而等上几天。这样的速度使得那些分散在各地的工作组协同工作非常容易,而这在以前是根本不可能的。

另一种计算机辅助通信的形式是视频会议。利用这项技术,即使雇员们处于相距甚远的不同地点上,他们也可以开会,相互之间看得到、听得到,甚至还可以在一个共享的虚拟黑板上写写画画。视频会议是一个功能强大的工具,它消除了以前的差旅开销和路途上所需的时间。有时候你可能会听到这样的说法,通信和运输在进行一场竞争,不管哪一方赢了,都会淘汰掉另一方。

第三个目标是,越来越多的公司与其他公司进行电子商务活动,特别是跟供应商和客户公司之间的商务活动。例如,汽车、飞机和计算机的制造商需要从各种不同的供应商购买子系统,然后再将这些部件装配起来。这些制造商利用计算机网络,就可以根据需求以电子的方式下订单。这种实时(即根据需要)下订单的能力大大降低了对于大库存量的需求,并且也提高了效率。

第四个目标正在变得越来越重要,那就是通过 Internet 与客户做各种交易。航空公司、书店和音乐零售商发现,许多顾客喜欢在家里购买他们的商品或者服务。因此,许多公司提供了在线查询商品和服务的功能,并且允许在线下订单。计算机网络的这种用途

有望在将来得到进一步的发展。这就是电子商务 (electronic commerce, e-commerce)。

### 1.1.2 家庭应用

1977 年的时候, Ken Olsen 是 DEC(Digital Equipment Corporation)公司的总裁,当时该公司名列全球第二大计算机厂商(仅次于 IBM)。当有人问起“为什么 DEC 不大胆地进军个人计算机市场”的时候,他回答说“没有任何理由让每个人家里都拥有一台计算机。”历史证明这是错误的,所以 DEC 公司不再存在了。为什么人们要为了家用而购买计算机呢?最初,是为了文字处理和游戏,但是,最近几年情况发生了很大的变化。现在最主要的理由可能是为了访问 Internet。对于家庭用户而言,Internet 的几个最流行的用途是:

- (1) 访问远程信息。
- (2) 个人之间的通信。
- (3) 交互式娱乐。
- (4) 电子商务。

访问远程信息可以有多种方式。浏览 Web 页面可以是为了获取远程信息,也可以纯粹为了娱乐。获取到的信息多种多样,包括艺术、商务、烹饪、政府、健康、历史、爱好、娱乐、科学、运动、旅游,等等。通过 Internet 进行娱乐的方式有很多是可以提上台面的,然而,还有一些娱乐方式最好不提为妙。

许多新闻报纸已经变成在线形式了,并且可以针对个人进行定制。例如,你可以告诉一份报纸,你希望阅读有关腐败政客、火灾、名人丑闻,以及流行疾病方面的信息,而不需要足球信息。有时候,甚至还可能在睡觉的时候,这些被选出来的文章自动下载到你的硬盘上,或者在早餐前输出到打印机上。这种趋势继续发展的结果是,一大批 12 岁报童们将会失业,但是报社喜欢这样,因为报纸发行是整个产品链中最薄弱的环节。

报纸(以及杂志和科学期刊)变成在线形式以后,更进一步就是在线数字图书馆了。许多专业组织,比如 ACM([www.acm.org](http://www.acm.org))和 IEEE Computer Society([www.computer.org](http://www.computer.org))已经将许多期刊和会议论文集放到了网上。其他的组织也正在快速跟上。随着书本大小的笔记本计算机的价格、尺寸和重量越来越为大众所接受,印刷图书慢慢就会变得过时了。那些对这种趋势抱有怀疑态度的人应该注意到,印刷出版社对于中世纪时期的手抄业带来的影响其实也是这样一种重要的进展。

所有以上这些应用都涉及到了个人与远程信息数据库之间的交互过程。第二类广泛的网络应用是个人对个人的通信,基本上是 21 世纪对 19 世纪的电话的回应。E-mail 已经成为全世界成千百万人日常工作和生活的基础,并且它的使用还在快速地增长。这一类应用中也包括音频和视频,以及文本和图片。下面再进一步介绍。

十几岁的青少年特别容易沉溺于**即时消息(instant messaging)**应用。这种应用是从 UNIX 上的 talk 程序(差不多从 1970 年以来一直在使用)演变过来的,它允许两个人相互之间实时地输入消息。同样是这种思想,一个多人参与的版本是**聊天室(chat room)**应用。在聊天室应用中,一组人可以同时输入消息,并且所有的人都可以看到这些消息。

全球范围的新闻组常常是一组特定相关人员的公共活动场所,在这里大家可以畅所

欲言。这种现象正在继续发展,参与的人数越来越多。在新闻组的讨论中,每个人都可以张贴消息,所有其他订阅该新闻组的人都可以读到这条消息。新闻组的讨论范围极其广泛,从幽默小事到煽动激情的大事件,什么都有。与聊天室不同的是,新闻组不是实时的,其中的消息将被保存起来,所以,当你度假回来时,仍然可以阅读到所有在度假期间张贴的消息。

另一种个人对个人的通信类型通常被冠以“对等通信(peer-to-peer communication)”的名字,以便与客户-服务器模型区分开(Parameswaram et al., 2001)。在这种通信形式中,所有单独的个体构成了一个松散的组,这些个体可以与组中的其他个体进行通信,如图 1.3 所示。原则上,每个人都可以与其他的一个或者多个人进行通信,所以,这里并不存在固定的客户和服务角色。



图 1.3 在对等系统中,没有固定的客户和服务角色

对等通信真正引起人们关注是在 2000 年的时候,一个称为 Napster 的服务在巅峰时刻有超过 5 千万的乐迷在交换音乐,这可能是有史以来规模最大的侵犯版权活动(Lam and Tan, 2001; Macedonia, 2000)。Napster 的思想非常简单,成员们把他们硬盘上保存的音乐注册到一个中心数据库中,这个中心数据库位于 Napster 的服务器上。如果某一个成员想要一首歌,那么,他只要检查中心数据库,看一看谁拥有这首歌,然后直接到这个人的机器上去取。Napster 的机器上并没有真正保存任何音乐,所以他们辩解说自己并没有侵犯任何人的版权。法庭并不支持这种说法,所以最终还是关掉了 Napster。

但是,下一代对等系统不再需要中心数据库,而是让每个用户在本机维护自己的数据库,并且提供一个列表,列表中包含了邻近的、也属于该系统的成员。然后,新的用户可以到一个已有的成员那里看一看他那里有些什么音乐,并且从他的邻近成员列表中了解到更多的音乐和更多的名字。这个查找过程可以无限重复,从而建立起一个庞大的本地数据库。这种行为要是让人来完成则是非常枯燥乏味,但是对于计算机来说则是再合适不过了。

合法的对等通信应用也是存在的。例如,有的乐迷喜欢收集公益音乐或者样本音乐片段(比如新的乐队为了引起公众注意),然后拿出来跟其他人共享;有些家庭喜欢共享照片、电影和家谱信息;有些十多岁的青少年喜欢玩多人在线游戏。实际上,最流行的 Internet 应用之一,即 E-mail,本质上就是对等的。这种通信形式有望在将来获得更大的发展。

电子犯罪已经不再局限于版权法了。另一个热点领域是电子赌博。这几十年来计算机一直在模拟各种事物,那么,为什么不模拟老虎机、轮盘、二十一点发牌器以及更加高级的赌博设备呢?是的,不应该模拟这些赌博设备,因为在许多地方赌博是非法的。然而麻烦在于,在有些地方(比如英格兰)赌博是合法的,那里的娱乐场主们已经抓住了 Internet 赌博的潜在优势。如果参与赌博的人和设立赌局的娱乐站点位于不同的国家,并且两个国家的法律是有冲突的,那该怎么办呢?这是一个很好的问题。

其他面向通信的应用包括:用 Internet 来承载电话呼叫、用 Internet 来实现可视电话,以及 Internet 无线电台,这三种应用的发展都非常快。另一个应用是远程学习(telelearning),这就是说,你根本不用起床就可以参加早晨 8 点钟的课。从长远来看,通过网络可以增强人与人之间的通信,这可能比其他任何一种网络应用更加重要。

第三类网络应用是娱乐,娱乐业是一个巨大的、并且还在快速增长的行业。这里的杀手级应用(killer application,即可以驱动其他所有应用的关键应用)是视频点播。因此,大约再过 10 年左右,你可能可以选择任何一部电影,或者任何一个国家的电视节目,然后让它立即显示在你的屏幕上。新的电影可能变成交互式的,看电影的时候用户偶尔可以提示一下故事的发展方向(麦克白应该杀死邓肯,还是应该等待时机?),电影片子为各种可能的选择提供了相应的场景。直播电视也可能会变成交互式的,观众可以参与智力竞赛节目,选择参赛对手,等等。

另一方面,杀手级应用可能不是视频点播。也许游戏应用将来反而会成为关键应用。我们已经有了多人实时模拟游戏,比如在虚拟城堡中玩捉迷藏,以及飞行模拟游戏(一组玩家尽力消灭掉另一组玩家)。如果我们能够戴上专门的防护眼镜,并且在玩的时候配上三维实时的、高质量的移动图像,那么我们就拥有了一个全球范围的虚拟现实空间。

第四类网络应用是广义的电子商务。家庭购物已经很流行了,它允许用户在家里浏览上千家公司的在线商品目录。对于有些商品,你只要在商品的名字上单击一下鼠标,就可以看到一段关于该商品的录像。如果顾客以电子方式购买了一种商品,但是不清楚如何使用该商品,那么,他甚至还可以获得在线技术支持。

在电子商务应用中另一个领域是允许直接访问金融机构。许多人已经通过电子方式来支付账单、管理银行的账户、处理他们的投资。随着计算机网络变得越来越安全,这种业务肯定会继续增长。

另一个几乎人人没有预料到的领域是电子跳蚤市场(e-flea?)。二手货物的在线拍卖已经变成了一个巨大的行业。传统的电子商务使用了客户-服务器模型,而在线拍卖更多地是一种对等系统,也是一种顾客对顾客的系统。由于“to”与“2”在英文中有同样的发音,所以,有些电子商务形式有了专门的名称,其中一些最流行的名称如图 1.4 所示。

毫无疑问,在将来,计算机网络的使用范围还会持续增长,甚至这种增长的态势是现在所无法预测的。毕竟,在 1990 年的时候又有多少人会预见到,10 年之后那些坐在公共汽车上频频发送冗长短消息的青少年会给电话公司带来巨额的利润。但事实上,短消息服务确实是非常盈利的。



专用名称	全 名	例 子
B2C	Business-to-consumer(商家对消费者)	在线订购图书
B2B	Business to business(商家对商家)	汽车制造商从轮胎供应商处订购轮胎
G2C	Government-to-consumer(政府对消费者)	政府通过电子方式发行税单
C2C	Consumer-to-consumer(消费者对消费者)	在线拍卖二手商品
P2P	Peer-to-peer(对等系统)	文件共享

图 1.4 电子商务的一些形式

对于那些地理位置极其不方便的人来说,计算机网络可能会变得非常重要,因为计算机网络使得他们在访问远程服务的时候与那些住在市中心的人们同样方便。远程学习(telelearning)可能会极大地影响到教育领域,有些高等院校可能会因此而变成全国性的,或者是国际性的。远程医学现在还只是刚刚起步(例如,远程病人监控),但是将来可能会变得非常重要。但也许杀手级应用也许是某一个非常普通的应用,比如在冰箱中使用一个支持 Web 的小组件,这样你就可以确定是否需要在下班的路上购买一些牛奶。

### 1.1.3 移动用户

可移动的计算机,比如笔记本电脑和 PDA(personal digital assistants,个人数字助理),是计算机工业中增长最为迅猛的分支之一。大多数这些计算机的所有者在办公室或家里还拥有桌面计算机,而且他们往往希望当自己不在家的时候,或者在路上的时候,还可以连接到家里的机器上。由于在汽车上或者在飞机上通过有线连接是不可能的,所以,无线网络便成了许多人的研究热点。在这一小节中,我们来看一下无线网络的一些用途。

人们为什么需要无线网络呢?一个常见的理由是移动办公。行走在路上的人们往往希望使用移动设备来发送或者接收电话、传真和电子邮件,或者浏览 Web 页面、访问远程文件、登录到远程机器上。他们希望在陆、海、空的任何一个地方都能够做到这一点。例如,在如今的计算机会议上,组织者通常在会议区域中建立一个无线网络。与会的人员只要有一个笔记本电脑和一个无线调制解调器,就可以打开计算机并连接到 Internet 上,其效果就如同他们将计算机接入到有线网络中一样。类似地,有些大学的校园里也安装了无线网络,所以,学生们坐在树下就可以查看图书馆的资料目录,或者阅读电子邮件。

无线网络对于运货车队、出租车、快递专车以及修理工与他们的后方或者家庭保持联络有特别的意义。例如,在许多城市中,出租车司机是个体户,而不是受雇于某一家出租车公司;在某些这样的城市中,出租车上有一个专门供司机观看的显示器,当有顾客呼叫的时候,中心调度室就会输入该顾客的上车地点和目的地。于是,这份信息就会显示在司机所看到的显示器上,并且还会有提示音。第一个在显示器上按下按钮的司机就会接到呼叫。

无线网络对于军事也很重要。如果你要在匆忙之间准备一场战争的话,那么,指望使用局域网络设施来进行通信可能不是一个好主意。最好你能够自己带一套网络。

虽然无线网络和移动计算常常被联系在一起,但它们不是一回事,如图 1.5 所示。这

里,我们可以看到**固定无线(fixed wireless)**和**移动无线(mobile wireless)**之间的区别。笔记本计算机有时候也可以通过有线网络来实现移动性。例如,旅游的客人可以把笔记本计算机接入到旅馆房间中的电话插孔上,这样即使没有无线网络,他也可以移动工作。

无线	移动	应    用
否	否	办公室中的桌面计算机
否	是	旅馆房间中使用的笔记本计算机
是	否	老式的、没有布线的建筑物中的网络
是	是	可移动的办公室;用 PDA 来管理商店库存

图 1.5 无线网络和移动计算的组合

另一方面,有些无线计算机并不是移动的。比如这样一个典型的例子:一个公司的大楼很老,以前并没有布置好网线,现在这家公司要将计算机连接起来。安装一个无线网络是比较省事的做法,它只需购买一些电子设备,并根据一定的规格将这些设备安装好即可。这种方案比重新让工人在整个大楼内铺设电缆要合算得多。

当然,也有一些真正移动的无线应用,从移动办公室到“在仓库周围走动的人用 PDA 来管理库存”,这些都是真正的无线移动应用。在许多繁忙机场的停车场,负责被租赁汽车归还工作的雇员是通过无线移动计算机来工作的。他们输入归还汽车的牌照号码,于是移动计算机向主计算机发出请求,以获取该汽车的租用信息,然后通过内置的打印机当场打印出账单。

随着无线技术的不断普及,大量其他的应用也在不断地涌现出来。现在我们来概略地看一下其中一些可能的应用。无线停车计时器无论对于用户,还是对于城管部门都很有优势。计时器可以接受信用卡,也可以接受借记卡,通过无线网络立刻就可以验证这些卡的有效性。当计时器设定的时间到了之后,它可以检查是否还停有汽车(通过回射一个信号),并且向警察报告停车超时。据估计,通过这种方式,仅仅美国的城管部门就可以增加 100 亿美元的收入(Harte et al., 2000)。而且,好的停车秩序也有助于改善环境,因为当司机们知道非法停车一定会被抓住之后,有些司机可能就会使用公共交通。

食品、饮料和其他的自动售货机已经随处可见。然而,食品当然不会自动地进入到机器中。所以,定期会有人将食品送来并装到机器中。如果自动售货机通过无线网络每天发送一次库存状态报告,那么运货车司机就知道哪些机器需要服务,并且知道哪种食品需要带多少。利用这样的信息,调度中心或者司机可以安排更加有效的运输路线。当然,通过标准的电话线来发送这样的信息也是有可能的,但是,为每一台自动售货机配备一个固定的电话连接,并且每天只呼叫一次,相对于每个月的月租费而言,这样做太昂贵了。

利用无线网络来节省开销的另一个领域是仪表阅读服务。如果每个家庭中的电表、天然气表、水表以及其他的仪表都通过无线网络来报告用量,那就根本不需要再有抄表员出来工作了。类似地,无线的烟雾检测器可以直接呼叫消防队,而不用发出那么大的噪音了(而且,如果没有人在家的话,这种噪音根本无济于事)。随着无线设备和发射时间这两者成本的不断降低,越来越多的测量和报告手段都将可以通过无线网络来完成。

无线网络的另一个截然不同的应用是期望将蜂窝电话和 PDA 合并到微型的无线计算机内部。最开始的尝试是微型的无线 PDA,它可以在一个非常小的屏幕上显示被简化之后的 Web 页面。这样的系统称为 WAP 1.0(Wireless Application Protocol,无线应用协议),但是它失败了,主要的原因是太微型的屏幕、低的带宽和太差的服务。不过,随着 WAP 2.0 的出现,新的设备和服务应该会更好。

这些设备的一个适用领域是 m-commerce (mobile-commerce, 移动商务) (Senn, 2000)。其中的推动力来自于一些既生产无线 PDA,又经营网络服务的厂商,它们试图分得电子商务市场的一杯羹。他们的一个希望是,利用 PDA 来使用银行的服务,以及实现网上购物。一种思路是,将无线 PDA 用作一种电子钱包,在商场里完成支付过程,从而代替现金和信用卡。然后,这些费用将出现在移动电话的账单上。从商场的角度来看,这种方案可能会节省下信用卡公司的绝大部分费用(这部分费用可能有几个百分点)。当然,这个计划也可能会适得其反,因为商场中的顾客在购买商品之前,可以使用他们的 PDA 来查看其他商场的商品价格。更糟糕的是,电话公司提供的 PDA 可能带有条码阅读器的功能,所以,顾客扫描了商品之后,马上就可以得到详细的报告信息,包括在其他哪些商场可以购买到这种商品,以及相应的价格信息。

由于网络运营商知道用户所在的区域,所以,有些服务可以将区域位置作为一种因素考虑进去。例如,用户可以请求查找最近的书店,或者最近的中国餐馆。移动地图也是一种可以考虑的服务。非常细致的天气预报也很合适(比如说,用户可以这样问“我家后院什么时候会停止下雨?”)。毫无疑问,随这种设备而来的许多其他应用会越来越广泛。

移动商务的一个很大优势在于,移动电话用户习惯于付费服务(Internet 用户正好与之相反,他们期望什么都是免费的)。如果 Internet 上的一个 Web 站点需要顾客使用信用卡支付一定的费用才允许访问,那么,用户一定会对此抱怨不已。如果一个移动电话运营商允许人们使用电话来支付商场中的物品,然后针对这项便利服务收取一定的费用,那么,这种方式极有可能被大众所接受。不管怎么样,时间会证明这一切。

另一个稍微远一点的应用是个人区域网络和可穿戴计算机。IBM 已经开发出了一种手表,它运行 Linux(包括 X11 窗口系统),并且通过无线网络连接到 Internet 上以便发送和接收 E-mail(Narayanaswami et al., 2002)。将来,人们只要相互之间向对方露出手表就可以交换名片了。可穿戴的无线计算机将允许人们访问安全的房间,就如同今天磁条卡所起的作用那样(可能跟 PIN 码或者生物测量手段结合起来)。这些手表也能够获取到与该用户所在位置相关的信息(比如本地的餐馆)。像这样潜在的用途是没有止境的。

具有无线发射功能的智能手表自从 1946 年出现在 Dick Tracy 的连环画中以来,已经成为我们精神世界的一部分。但是智能尘埃呢(smart dust)? Berkeley 的研究者们已经将一台无线计算机封装到一个边长为 1mm 的立方体中(Warneke et al., 2001)。潜在的应用包括跟踪库存、包裹,甚至小鸟、小的啮齿动物和昆虫等。

#### 1.1.4 社会问题

网络的广泛应用已经导致了新的社会、伦理和政治问题。下面我们仅仅简要地列举

出一些这样的问题,如果要完全深入地探讨这些问题,则至少需要一整本书的篇幅。许多网络中一个共同的特征是新闻组或者布告板,趣味相投的人们通过新闻组或者布告板交换消息。只要所涉及的话题仅限于技术或者如园艺之类的爱好方面,则不会有太多的问题。

当新闻组所讨论的主题真正是人们所关心的一些话题,比如政治、宗教或者性的时候,麻烦就来了。在这种新闻组中张贴出来的一些观点可能会攻击到某一些人,更糟糕的是,这些观点有可能在政治上是不正确的。而且,这些消息不一定只是文本,在计算机网络上传输高分辨率的彩色图片或者简短的视频片断也是非常容易的。有的人采取宽容和容忍的态度,但有的人则认为,张贴某些具有特定含义的材料(比如对于某些国家或者宗教的攻击,或者色情图片等)是不可接受的,这些材料应该被审查之后才能张贴。在这些领域中,不同的国家有不同的法律,有的甚至是相互冲突的。于是,论战就爆发了。

人们控告网络运营商,声称他们应该对网络上传递的内容负责,就好像报社和杂志社所做的那样。网络运营商对此的反应自然是,网络就像一个电话公司或者邮局一样,它不可能成为用户们所期望的警察。更有甚者,如果网络运营商要是审查消息的话,他们可能会删除任何有可能导致自己被指控的消息,哪怕只有一点点被指控的可能性,这样的做法必然会侵犯用户的言论自由权利。可以这样说,这种争论还会持续一段时间。

另一个有意思的话题是雇员的权利与雇主的权利。许多人在工作中需要读写邮件。许多雇主已经声明了他们有权阅读雇员的消息,并且可能还要对消息的内容进行审查,即使在下班之后从家里的计算机发送出来的消息也要审查。然而,并不是所有的雇员都同意这种观点。

即使雇主们拥有这样的权利,那么,大学学校和学生之间也具有这样的关系吗?中学学校和学生之间呢?1994年,卡内基-梅隆大学决定关闭几个涉及到性的新闻组的进入消息流,因为校方已经意识到这样的材料不适合于未成年人(比如,那些未满18岁的学生们)。这次事件的影响持续了好几年才平息下来。

另一个关键的话题是政府和老百姓。FBI已经在许多Internet服务供应商处安装了一个系统,以便监听到所有进入和出去的邮件,他们这样做当然有各种目的(Blaze and Belloc, 2000; Sobel 2001; Zacks, 2001)。该系统最初称为 **Carnivore**(食肉动物),但是,它被公开之后的反响很不好,所以又换了一个好听一点的名字 DCS1000。但是,它的目标仍然是监听成千上万人的邮件,以期望找到有关非法活动的信息。不幸的是,美国宪法的第四次修正案明确规定了:如果没有搜查许可的话,禁止政府执行任何搜查工作。这段写于18世纪的话在21世纪还会有分量吗?不管怎么样,它会让法庭一直忙到22世纪去了。

威胁到人们隐私的不仅仅只有政府,有一些私人信息存储区也同样会泄露个人隐私。例如,一种称为 cookie 的小文件是 Web 浏览器在用户计算机上保存信息而使用的,它允许 Web 服务公司记录下用户在计算机上所做的各种操作,也可能会将信用卡号码、社会安全号码和其他保密信息泄露到 Internet 上(Beighel, 2001)。

计算机网络也提供了发送匿名消息的能力。在有些情况下,这种能力可能是非常理想的。例如,通过这种匿名邮件的方式,学生、士兵、雇员和老百姓可以检举教授、长官、上



级、政府官员的非法行为,而不用担心会遭到报复。另一方面,在美国以及大多数其他的民主国家中,法律明确规定了,被告有权利要求在法庭上与原告当面对质。匿名指控不能用作证据。

简而言之,计算机网络就像 500 年前的印刷出版业一样,它允许普通的老百姓也能够以不同的方式向不同的听众发表他们的观点,而这在以前则是不可能的。这种新兴的自由也带来了许多尚未解决的社会、政治和道德问题。

计算机网络在带来好处的同时也必然会导致一些问题。Internet 使得人们有可能快速地找到想要的信息,但是有许多信息是过期的、误导的或者完全错误的。你从 Internet 上收集来的医疗建议有可能来自于一位诺贝尔奖得主,也可能来自于一位连中学都没有读完的学生。计算机网络也已经导致了一些新的危害社会秩序甚至犯罪的行为。电子垃圾邮件(spam)已经成为许多人生活的一部分,因为有人收集了上百万的电子邮件地址,然后通过 CD-ROM 将这些地址卖给那些做市场的人。那些包含了活动内容(主要是程序,或者可在接收者机器上运行的宏)的电子邮件消息有可能包含病毒(为了发泄不满,或者报复)。

身份偷窃也成了一个日益严重的问题,因为窃贼可以收集到受害人足够的信息,利用受害人的名字获得信用卡号和其他的文档。最后,通过网络可以很方便地传输数字音乐和视频,这打开了侵犯版权的大门,而且这种版权侵犯还难以捕捉和制约。

如果计算机工业界认真考虑了安全性的话,那么,大多数这样的问题都能够解决。如果所有的消息都是加密和认证的,那么,要想犯这样的错误也并不容易。这些技术已经非常成熟了,我们将在第 8 章中详细讨论。问题是,硬件和软件制造商们知道,加入这些安全特性需要很高的代价,而且他们的顾客并没有要求这样的安全特性。另外,有相当数量的问题是由于软件的错误而引起的,因为软件制造商们不断地在他们的程序中加入新的功能,从而导致代码量越来越大,因而错误也就越来越多。由于新特征而增加收入当然是很好的,但是这可能会影响几个季度的销售情况。针对这些有缺陷软件的退款可能是好事,只不过,在刚开始的年头里,这可能会使整个软件工业破产。

## 1.2 网络硬件

现在我们把注意力从计算机网络的应用和社会问题(这些都比较有趣)转移到与网络设计有关的技术问题(这些就比较专业了)上来。关于计算机网络,没有一种被普遍接受的分类方法,但是有两个因素是非常重要的:传输技术和距离尺度。下面我们依次进行讨论。

一般来说,目前普遍使用的传输技术有两种,它们是:

- (1) 广播式链接;
- (2) 点到点链接。

**广播网络(broadcast networks)**只有一个通信信道,网络上所有的机器都共享该信道。在机器之间传递的是短消息(在有些上下文环境中称为分组或包,packet),任何一台机器发送的短消息都可以被其他所有的机器接收到。在分组中有一个地址域,指明了该分组



的目标接收者。一台机器收到了一个分组以后,它检查地址域。如果该分组正是发送给它的,那么它就处理该分组;如果该分组是发送给其他机器的,那么它就忽略该分组。

比方说,考虑这样的情形:有一条走廊连着很多房间,然后,有一个人站在走廊的头上大声喊“Watson,到这里来,我要见你。”虽然许多人都收到了(听到了)这个分组,但是,只有 Watson 会应答。其他人都忽略了。另外一个可以类比的例子是,机场广播站请所有乘坐 644 航班的旅客都立即到第 12 号登机口上飞机。

广播系统往往也允许将一个分组发送给所有的目标机器,一般的做法是在地址域中使用一个特殊的编码。如果被传输的分组带有这样的地址编码,那么网络中的每一台机器都将会接收该分组,并进行处理。这种操作模式称为广播(broadcasting)。有些广播系统也支持传输给一组机器,即所有机器的一个子集,这种模式称为多播(multicasting)。一种可能的方案是,在地址域中专门用一位来表示多播传输,剩下的  $n-1$  位用于存放组的编号。每台机器都可以“订阅”一个组,甚至“订阅”所有的组。当一个分组被发送给某个特定的组的时候,它将被递交给所有订阅了该组的机器。

与此相反,点到点(point-to-point)网络则是由许多连接构成的,每一个连接对应于一对机器。在这种网络中,为了将一个分组从源端传送到目的地,该分组可能首先要经过一台或者多台中间机器。通常有可能存在多条不同长度的路径,所以,找到一条好的路径对于点到点网络是非常重要的。一般的原则(尽管也有很多例外)是,越是小的、地理位置局部化的网络倾向于使用广播传输模式,而大的网络通常使用点到点传输模式。只有一个发送方和一个接收方的点到点传输模式有时候也称为单播(unicasting)。

分类网络的另一个准则是网络的距离尺度。在图 1.6 中,我们按照物理范围来划分多处理器系统。最上面的是个人区域网络(personal area network),其含义是仅仅供一个人使用的网络。例如,一个无线网络将一台计算机与它的鼠标、键盘和打印机连接起来,这样的无线网络就是一个个人区域网络。用 PDA 来控制用户的助听器或者起搏器也属于这一类网络。除了个人区域网络以外,其他的都是跨越较长距离的网络。这些网络可以分为局域网、城域网和广域网。最后,两个或者多个网络连接起来之后称为互联网。世界范围的 Internet 是一个最有名的互联网例子。距离作为一种分类的度量是非常重要的,因为不同的距离尺度将会使用不同的技术。本书中我们将会讨论所有这些尺度上的网络。下面我们简要地介绍一下网络硬件。

处理器之间的距离	处理器所在的位置	例子
1m	一米见方的范围内	个人区域网
10m	同一房间	局域网
100m	同一建筑物内	
1km	同一校园	
10km	同一城市	城域网
100km	同一国家	广域网
1000km	同一个洲	
10 000km	同一个行星	
		Internet

图 1.6 对相互连接的多处理器系统按照距离进行分类

### 1.2.1 局域网

**局域网 (local area network, LAN)** 是专有网络, 通常位于一个建筑物内, 或者一个校园内, 也可以远到几千米的范围。局域网通常用来将公司办公室或者工厂中的个人计算机和 workstation 连接起来, 以便共享资源 (比如打印机) 和交换信息。相比其他类型的网络, 局域网有三个不同的特征: (1) 范围; (2) 传输技术; (3) 拓扑结构。

LAN 的覆盖范围是有限制的, 这意味着最差情况下的传输时间是有限的, 而且可以预先知道其上界。知道了最差的传输时间之后, 我们就有可能使用某种特定的设计, 若不知道此上界就无法使用这样的设计。而且, 这样也简化了网络的管理。

LAN 有可能使用这样一种传输技术: 所有的机器都连接到同一根电缆上, 就好像电话公司过去在农村地区使用的线路一样。传统 LAN 的运行速度在 10Mbps~100Mbps 之间, 其延迟很低 (微秒或者纳秒量级), 而且很少有传输错误。新型的 LAN 可以达到 10Gbps 的速度。在本书中, 我们将遵照传统的做法, 使用 Mbps (每秒百万位, 即 1Mbps = 1 000 000 位/秒) 和 Gbps (1Gbps = 1 000 000 000 位/秒) 作为线速度的度量单位。

广播式 LAN 也可能有不同的拓扑结构。图 1.7 显示了两种拓扑结构。在总线型 (比如一条直线电缆) 网络中, 任何一个时刻, 至多只有一台机器是主机器, 并且只有它才可以传送数据。其他所有的机器都被禁止发送数据。当两台或者多台机器同时想传送数据的时候, 需要有一种仲裁机制来解决冲突问题。这种仲裁机制可以是集中式的, 也可以是分布式的。例如, 通常称为以太网 (Ethernet) 的 IEEE 802.3 就是一种非集中控制的、基于总线的广播式网络, 其工作速度往往在 10Mbps~10Gbps 之间。以太网上的计算机在任何时候都可以传送数据; 如果两个或者多个分组发生碰撞的话, 每台计算机只是等待一段随机的时间, 然后再次尝试发送数据。

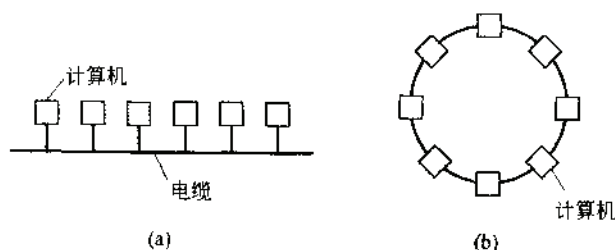


图 1.7 两种广播式网络  
(a) 总线型; (b) 环型

第二种广播式系统是环型网络。在环中, 每一位都沿着自己的路径独立向前传播, 而不需要等待它所属分组的其他位。通常情况下, 每一位环绕一周所需要的时间也就相当于传送几个位的时间, 常常在整个分组被传送出去之前, 大多数的位已经走完一周了。如同所有其他的广播式系统一样, 为了避免并发访问同一个环, 制定一些规则也是有必要的。有许多种方法已经在实践中使用了, 比如让所有的机器都按照既定的顺序来发送数据。IEEE 802.5 (IBM 令牌环) 是一种基于环的 LAN, 其工作速度是 4Mbps 和 16Mbps。FDDI 是另一种环网的例子。

广播式网络进一步可以分为静态和动态的,划分的准则是信道的分配方式。一种典型的静态分配方案是,将时间分成离散的间隔,并且使用一种轮循算法,每一台机器只有在它自己的时槽(time slot)到达的时候才可以广播数据。如果一台机器在它所分配到的时槽中不需要发送数据的话,那么,这种静态分配方案就浪费了信道的传输容量,所以,大多数的系统都会采用动态分配信道(即按需分配)。

公用信道的动态分配方法也可以分为集中式的,或者非集中式的。在集中式的信道分配方法中,有一个独立的实体,比如总线仲裁单元,由它决定谁是下一个发送方。它的工作方式可能是这样的:首先接受大家的请求,然后根据某一个内部算法来作出决定。在非集中式的分配方法中不存在这样的中心实体,每台机器必须自己来决定是否传递数据。你可能会认为这种做法总是会导致混乱,但事实上不是这样的。有许多种算法可以用来避免这种潜在的混乱,本书后面我们将学习这些算法。

### 1.2.2 城域网

城域网(metropolitan area network, MAN)覆盖了一个城市。最有名的城域网例子是有线电视网,许多城市都有这样的网络。这种系统是由早期的社区天线系统(用于那些条件较差的,通过天线来接收电视的用户)发展起来的。在这些早期的系统中,常常有一个很大的天线放在附近的小山上,然后通过它将电视信号转发到订阅者的家里。

最初的时候,这都是一些局部范围内设计的专用(ad hoc)系统。然后商业公司开始参与其中,他们从城市管理部门拿到筹建整个城市电视网的合同。接下来是电视节目的编排,以及专门针对这种电缆而设计的各种信道。通常这些信道各司其职,比如有的信道传递新闻,有的信道传递体育运动节目,有的传递烹饪节目,有的传递园艺节目,等等。但是,从初期建设网络一直到九十年代后期,这些信道都仅仅专用于电视节目的传递和接收。

由于Internet吸引了大量的用户,从这时起,有线电视网络运营商开始意识到,只需对他们原有的系统做一些变化,他们就可以利用原来尚未使用的频谱来提供双向Internet服务。从这时候起,有线电视系统不再仅仅传递电视节目,它开始变成一个城域网。大致上讲,一个城域网(MAN)看上去跟图1.8中所示的系统非常类似。在该图中,



图 1.8 一个基于有线电视的城域网

我们可以看到,电视信号和 Internet 都送到一个集中控制的头端(head end),然后再送到居民的家庭中。在第 2 章中我们将再回到这个主题的细节上来。

有线电视网络不是唯一的 MAN。在高速无线 Internet 接入领域中的最新发展导致了另一个 MAN,并且它已经被标准化了(IEEE 802.16)。我们将在第 2 章讨论这个话题。

### 1.2.3 广域网

广域网(wide area network, WAN)跨越了一个很大的地理区域,通常是一个国家或者一个洲。它包含了大量的机器,在这些机器上可以运行用户的程序(也就是应用程序)。我们将按照传统的用法,将这些机器称作主机(host)。这些主机通过通信子网(communication subnet,或仅仅简称为子网,subnet)连接起来。主机是用户所有的(比如,个人计算机),而通信子网则往往是由电话公司或者 Internet 服务提供商所有的。子网的任务是承载消息,它将消息从一台主机发送到另一台主机,这就好比电话公司承载了语音,将语音从说话者一方传递到接听者一方。把网络(子网)纯粹通信的方面与应用的方面(主机)分离开,这有助于简化网络的整体设计。

在大多数广域网中,子网是由两个独立的部分组成的:传输线和交换单元。传输线(transmission line)用于在机器之间传送数据位。它们可以由铜线、光纤,甚至无线电链路构成的。交换单元(switching element)是指一种特殊的计算机,它们连接了三条或者更多条传输线。当数据在一条进线上到达的时候,交换单元必须选择一条出线,以便将数据转发出去。这些交换计算机在过去被冠以多个不同的名字,其中“路由器(router)”是目前最为普遍使用的名字。然而,不幸的是,有的人将它发音成“rooter”,还有一些人则发音成“doubter”。到底哪个发音是正确的,这就留给读者作为练习吧。(注意,你认为正确的答案可能要取决于你所在的地区。)

在如图 1.9 所示的模型中,每一台主机往往连接到一个 LAN 上,在 LAN 上会有一个路由器,但是,在有些情况下,主机也可以直接连接到一个路由器上。通信线和路由器(不包含主机)的集合构成了子网。

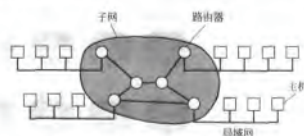


图 1.9 LAN 上的主机和子网之间的关系

关于“子网”的另一个简短说明也有必要在这里提一下。最初的时候,子网的含义仅仅是指路由器和通信线的集合,它用来将分组从源主机传送到目标主机上。然而,几年之后,它就有了第二层含义,即包含了与网络编址相关的含义(我们将在第 5 章中讨论网络编址)。不幸的是,针对它最初的含义,还没有一个广泛可用的替换词,所以,尽管犹豫再

三,我们还是在两种意义下都使用这个术语。从上下文读者总是可以清楚地辨别出它的实际含义。

在大多数 WAN 中,网络包含了大量的传输线,每条传输线将一对路由器连接起来。如果两个路由器之间并没有共享一条传输线,而它们又希望进行通信的话,那么,它们必须间接地通过其他的路由器来实现通信。当一个分组从一个路由器,经过一个或者多个中间路由器,被发送到另一个路由器上的时候,每一个中间路由器都会完整地接收到该分组,然后将它保存起来,直到所要求的输出线路空闲,才将它转发出去。根据这种原则组织起来的子网称为“存储-转发(store-and-forward)”或者“分组交换(packet-switched)”子网。几乎所有的广域网(除了那些使用卫星的广域网以外)都包含存储-转发子网。当分组很小,并且所有的分组大小相同时,它们通常称为信元(cell)。

分组交换型 WAN 的原则非常重要,因此有必要作进一步的说明。一般而言,当某一台主机上的一个进程要发送一条消息给另一台主机上的进程的时候,发送主机首先要把这条消息切割成一个一个的分组,每个分组都包含一个序号。然后,这些分组就快速连续地送入网络,一次送入一个。这些分组在网络上是单独传输的,然后陆续堆积到接收主机中,在接收主机上,它们又重新装配成原始的消息,然后递交给接收进程。从某一原始消息到分组流的过程如图 1.10 所示。

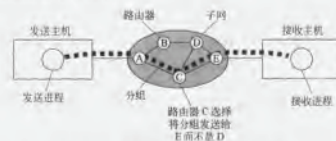


图 1.10 从发送方到接收方之间的分组流

在图 1.10 中,所有的分组都沿着路由器 ACE 传送,而不是 ABDE 或者 ACDE。在有些网络中,同一条消息的所有分组必须沿着同样的路径;而在其他的网络中,每个分组可以单独路由。当然,如果 ACE 是最佳的路由途径,那么,即使每个分组单独路由,这些分组仍然会沿着 ACE 传送。

路由决策是在路由器本地进行的。当一个分组到达路由器 A 的时候,A 可以自行决定该分组应该被送到连接 B 的传输线上,还是连接 C 的传输线上。A 做出这个决定的过程称为路由算法(routing algorithm)。路由算法非常多,我们将在第 5 章详细地讨论其中的一些。

并不是所有的 WAN 都是分组交换的,另一种可能的 WAN 是卫星系统。每一个路由器都有一个天线,通过该天线可以发送或者接收数据。所有的路由器都得到来自卫星的输出;在某些情况下,它们也能够得到其他路由器上传到卫星的信息。有时候,这些路由器也连接到一个实际的点到点子网上,并且,该子网中只有一部分路由器才具有卫星天线。卫星网络本身就是广播式的,当广播特性非常重要时它尤其有用。



#### 1.2.4 无线网络

数字无线通信并不是一种新的思想。早在 1901 年的时候,意大利物理学家 Guglielmo Marconi 演示了从轮船向海岸发送无线电报的试验,在试验中他使用了 Morse Code(莫尔斯编码,用点和划来表示二进制数字)。现代的数字无线系统的性能更好了,但是基本的思想并没有变化。

大致上,无线网络可以分成以下三大类:

- (1) 系统互连;
- (2) 无线 LAN;
- (3) 无线 WAN。

系统互连是指通过短距离的无线电,将一台计算机的各个部件连接起来。几乎所有的计算机都有一个监视器、键盘、鼠标和打印机,它们通过电缆连接到主机单元上。所以,许多新的用户刚开始的时候都很难将所有这些电缆连接到正确的插口上(尽管这些插口有彩色的标志),因而,大多数计算机销售商都提供了上门服务,让工程师到用户家里帮助安装所有这些电缆。因此,有一些公司联合起来,设计了一种称为蓝牙(Bluetooth)的短距离无线网络,将这些部件以无线的方式连接起来。通过蓝牙也可以将数码相机、耳机、扫描仪和其他的设备连接到计算机上,只要保证它们在一定的距离范围内即可。不需要电缆,也不需要安装驱动程序,只要把它们放到一起,然后打开开关,它们就可以工作了。对于许多人而言,如此简单的操作自然再合适不过了。

在最简单的形式下,系统互连网络使用了主-从模式,如图 1.11(a)所示。系统单元往往是主部分,从部分是鼠标、键盘等,主部分与从部分进行通话。主部分告诉从部分:应该使用什么地址,什么时候它们可以广播,它们可以传送多长时间,它们可以使用哪个频段,等等。我们将在第 4 章详细地讨论蓝牙技术。



图 1.11 (a) 蓝牙配置; (b) 无线 LAN

无线网络的下一步发展是无线 LAN(WLAN)。在无线 LAN 中有许多计算机,每台计算机都有一个无线电调制解调器和一个天线,通过该天线,它可以与其他的系统进行通信。通常在天花板上也有一个天线,所有的机器都可以与它通话,如图 1.11(b)所示。然而,如果系统之间的距离足够近的话,那么,它们可以按照点对点的对等配置,直接相互之间进行通信。在小型的办公室和家庭中,无线 LAN 正在变得越来越普及,因为在这样的

环境中,安装以太网太麻烦了。在其他一些地方,比如老式的办公楼、公司的自助餐厅、会议室等,无线 LAN 也在普及。针对无线 LAN 有一个标准,称为 IEEE 802.11,大多数系统都实现了该标准,现在该标准已经非常普及了。我们将在第 4 章讨论该标准。

第三种无线网络用于广域系统中。蜂窝电话所使用的无线网络是一个低带宽无线系统的例子。该系统已经经历了三代革新。第一代是模拟的,只能传送语音;第二代是数字的,也只能传送语音;第三代是数字的,不仅可以传送语音,还可以传送数据。从某种意义上讲,蜂窝无线网络就如同无线 LAN 一样,只不过覆盖的距离更大,位传输速率低一些而已。无线 LAN 的工作速率可以达到 50Mbps 左右,跨越距离可以到几十米。蜂窝系统的速率低于 1Mbps,但是基站与计算机或者电话之间的距离可以用 km 来度量,而不是用米来度量。在第 2 章中我们将进一步讨论这些网络。

除了这些低速网络以外,高带宽广域无线网络也正在迅速发展。最初的关注点是,允许家庭或者商业部门通过无线方式高速接入 Internet,绕过电话系统。这项服务通常称为局部多点分发服务(local multipoint distribution service)。在本书后而我们将学习这项技术。相应的标准也已经开发出来了,称为 IEEE 802.16。我们将在第 4 章中讨论该标准。

几乎所有的无线网络都在某一个点上连接到有线网络中,以便访问文件、数据库和 Internet。实现这种连接有很多种方法,具体方法取决于环境情况。例如,在图 1.12(a)中,我们显示了一架飞机,有一些人正在使用调制解调器和座位上的电话呼叫他们的办公室。这些呼叫都是相互独立的。然而,一种更为有效的方案是如图 1.12(b)所示的飞行中的 LAN。每个座位都配备了一个以太网连接器,乘客可以把它们的计算机插入到该连接器中。在飞机上有一个路由器,它与地面上的某个路由器之间保持一条无线链路,在飞行过程中会变换地面上的路由器。这种配置实际上就是一个传统的 LAN,只不过它与外界的连接是一条无线链路而不是硬件线路而已。

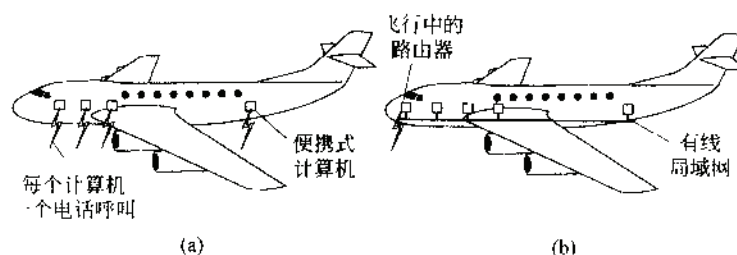


图 1.12 (a) 独立的移动计算机; (b) 一个飞行中的 LAN

许多人相信,无线是未来的潮流(例如 Bt et al., 2000; Leeper, 2001; Varshey and Vetter, 2000),但至少还有一个不同的声音我们可以听得到。以太网的发明者 Bob Metcalfe 这样写道:“移动的无线计算机就好像是移动的无管道卫生间——便携式便壶。它们会出现在交通工具上、建筑工地上、摇滚音乐会上。我的建议是在家里布上线路,并且就待在家里。”(Metcalfe, 1995)。他的话已经载入史册,就如同 1945 年 IBM 的总裁 T.J. Watson 解释为什么 IBM 不会进入计算机商业圈一样:“在 2000 年之前,全世

界只要 4 台或者 5 台计算机就足够了。”

### 1.2.5 家庭网络

家庭网络正在浮出水面。基本的思路是,将来大多数家庭都将会建立起一个网络环境。家庭中的每一个设备都具备与其他设备进行通信的能力,通过 Internet 可以访问所有这些设备。这是一个无人奢求的梦幻般的思想(就好像电视机遥控器或者移动电话一样),但是,一旦它们成为现实,没有人能够想象得出,要是离开了这些东西他们的生活会是什么样。

许多设备都有联网能力。对于有些设备类别,这是显而易见的,下面列出其中一些(也包含了相应的例子):

- (1) 计算机(桌面 PC、笔记本 PC、PDA,以及一些共享的辅助设备)
- (2) 娱乐(电视、DVD、VCR、便携式摄像机、照相机、立体声音响、MP3)
- (3) 无线通信(电话、移动电话、对讲电话、传真)
- (4) 家用电器(微波炉、电冰箱、钟、烤炉、空调、灯)
- (5) 遥测设备(电表、烟雾警报器、防盗自动警铃、自动调温器、婴儿监视器 [babycam])

实际上,家庭计算机网络已经存在了,只不过方式上有些受限制而已。许多家庭已经有一个设备将多台计算机连接到一个快速的 Internet 连接点上。网络化的娱乐设备还没有成为现实,但是越来越多的音乐和电影都可以从 Internet 上下载,所以,将立体声音响和电视机连接到 Internet 上的需求必将会出现。另外,人们还希望与亲朋好友分享自己拍摄的视频内容。无线通信技术已经将一个家庭与外界连接起来了,但是,不久的将来,它将是数字化的,并且与 Internet 连接起来。一个普通的家庭可能会有十多个钟表(包括家用电器中的表),当每年夏时制开始和结束的时候,所有这些钟表需要重新设定两次。如果这些钟表都连接到 Internet 的话,则设定工作可以自动完成。最后,对于家庭以及家庭中财物的远程监控也可能是一种需求。许多家长可能愿意花钱来监视自己的孩子,这样当他们出去吃饭的时候,他们可以在 PDA 上看到正在睡觉的孩子,即使他们已经雇了一个大一点的孩子来家里照看,他们也希望能够直接看到自己的孩子。你可以把每一个这样的应用想象成一个单独的网络,但是,如果能够把所有这些网络集成到一个网络中,则可能会更好。

与其他类型的网络相比,家庭网络有一些本质上不同的属性。首先,网络与设备必须易于安装。作者在过去几年中,已经安装了大量硬件设备,并且在各种计算机上安装了软件,情况千差万别。若给生产商的服务咨询台打求助电话,则无非是这样的结果:(1)请自己阅读手册;(2)重新启动机器;(3)除了该厂商的软硬件之外,去掉其他所有的软硬件,然后重新测试一下;(4)如果一切努力都失败的话,则从厂商的 Web 站点上下载最新的驱动程序;(5)重新格式化硬盘,然后从 CD-ROM 中重装 Windows 系统。让 Internet 电冰箱的买主下载并安装新版本的电冰箱操作系统,这显然会让他们很不愉快。计算机用户比较能够容忍不能正常工作的产品;而汽车、电视机或者电冰箱的用户则往往没有这么好的容忍度。他们期望这些产品从一开始就百分之百地正常工作。

其次,网络与设备必须易于操作。过去的空调设备通常只有一个按钮,该按钮有四个状态:OFF(关)、LOW(低)、MEDIUM(中)和HIGH(高)。现在的空调则有了30页的使用说明手册。一旦空调也连网之后,也许仅仅有关安全性的一章就得30页。这将是绝大多数用户所无法理解的。

第三,低的价格也是成功的关键。人们不会愿意花50美元购买一个Internet自动调温器,因为很少有人认为不在家的时候也能够监视家里的温度是很重要的。然而,如果只需5美元,它也许会卖得很好。

第四,主要的应用可能与多媒体有关,所以,网络需要具备足够的传输能力。如果通过Internet连接的电视只能播放一些抖动不稳的电影,分辨率只有 $320 \times 240$ 像素,并且每秒钟只有10帧图像,那么,这样的电视不会有市场。在大多数办公室中常用的快速以太网对于多媒体来说也是不够的。因此,家庭网络的性能需求比现有的办公网络的性能需求更高,而且,一旦家庭网络真正进入市场的话,它的价格必须低于现有的办公网络的价格。

第五,首先必须能够启动一个或者两个设备,然后逐渐扩充到网络的其他部分。这意味着不需要格式化的过程。如果你先告诉顾客应该购买具有IEEE 1394(FireWire)接口的设备,几年之后又换了一种说法,说USB 2.0才是应该支持的接口,这会让顾客不知所措。网络接口必须在几年之内保持稳定不变,而网络线(如果有的话)则必须保持几十年稳定不变。

第六,安全性和可靠性也是非常重要的。由于接收了病毒邮件而丢失几个文件,是一回事;一个黑客通过他的PDA攻穿你的安全系统,然后把你的家当洗劫一空,是另一回事。

一个有趣的问题是,将来的家庭网络到底是有线的还是无线的?大多数家庭已经安装了6个网络:电、电话、有线电视、水、煤气和下水道。在盖房过程中增加第7个网络并不困难,但是,重新改造现有的居室环境则开销很高。从投资的角度来看,无线网络较为合适;但是,从安全性角度来看,则应该选择有线网络。无线网络的问题在于,它所使用的无线电波可以穿越墙壁或围墙。让你的邻居也使用你的Internet连接,并且阅读你的电子邮件(就好像看到你将这些邮件打印出来一样),你会愿意吗?相信没有一个人会认为这是一件好事。在第8章,我们将学习如何使用加密技术来实现安全性,但是,在家庭网络的环境中,安全性必须非常简单易行,特别对于缺乏经验的用户而言,简单性更为重要。这么说起来很容易,但是做起来要难得多,即使对于有经验的高级用户而言也是如此。

简而言之,家庭网络带来了许多机遇,但是也面临着许多挑战。这些机遇与挑战绝大多数与易于管理、可靠性、安全性的需求有关,特别是面对大量的非技术用户;同时家庭网络也要求能以相对较低的价格获得高的性能。

### 1.2.6 互联网

世界上有许多网络,它们常常使用了不同的硬件和软件。连接到一个网络中的人们常常希望与连接到另一个网络中的人们进行通信。要想做到这一点,就要求那些相互之间不兼容的不同网络连接起来,有时候通过一台称为“网关(gateway)”的机器可以完成这

种连接,并且提供必要的转换,包括硬件层次上的转换和软件层次上的转换。一组相互连接起来的网络称为一个互联网(internetwork 或者 internet)。我们通常用 Internet(首字母大写,中文也称因特网)来表示特定的、世界范围内的互联网,相比之下,互联网这个术语则用来表达更为广泛的含义。

互联网的一种共同的组成形式是,通过一个 WAN 将多个 LAN 组织起来。实际上,如果我们把图 1.9 中的“子网”标注换成“WAN”,图中的其他部分不需要做任何变化,这就成一个互联网了。在这种情况下,子网和 WAN 之间惟一的技术区别在于是否有主机存在。如果在灰色区域中只有路由器,那么它是一个子网;如果它既包含路由器,又包含主机,则是一个 WAN。子网与 WAN 之间的其他区别还涉及到它们的所有权和用途。

子网、网络和互联网往往很容易混淆。子网的概念通常在广域网的环境下才有意义,它指一组路由器和一些通信线路(这些通信线路属于网络运营商)。与此相类似的是,电话系统是由电话交换局组成的,电话交换局相互之间通过高速线路连接起来,再通过低速线路连接到居民家庭和办公楼房。这些线路和设备构成了电话系统的子网,它们是由电话公司拥有并管理的。电话本身(相当于网络中的主机)并不是子网的一部分。子网和主机合起来构成了一个网络。在局域网的情形下,电缆和主机构成了网络,这里不需要子网。

当多个不同的网络相互连接起来的时候,就构成了一个互联网。从我们的观点来看,将一个 LAN 和一个 WAN 连接起来,或者将两个 LAN 连接起来就构成了一个互联网,但是工业界对于此领域中的术语有不同的看法。实践中的法则是,如果不同的组织出资构建了一个网络的不同部分,并且各自维护自己的那一部分,这样我们就得到了一个互联网,而不是单个网络。而且,如果不同的部分使用了不同的底层技术(比如广播传输或者点到点传输)的话,那么,我们可能有了两个网络。

## 1.3 网络软件

最初的计算机网络设计主要考虑的是硬件,其次考虑的才是软件。这种策略现在行不通了。现在的网络软件是高度结构化的。在这一节中,我们从某些细节的角度来讨论软件的结构化技术。这里介绍的方法构成了整本书的基础,以后还将会多次重复出现。

### 1.3.1 协议层次

为了降低网络设计的复杂性,绝大多数网络都组织成一堆相互叠加的层(layer 或者 level),每一层都建立在其下一层的基础之上。不同的网络,其层的数目、各层的名字、内容和功能也不尽相同。每一层的目的都是向上一层提供特定的服务,而把如何实现这些服务的细节对上一层加以屏蔽。从某种意义上讲,每一层都是一种虚拟机,它向上一层提供特定的服务。

这个概念实际上并不陌生,它广泛应用于计算机科学领域中,在有些地方,也称之为信息隐藏、抽象数据类型、数据封装以及面向对象程序设计。基本的思想是,一段(或块)专门的软件(或者硬件)向用户提供一种服务,但是将内部状态和算法的细节隐藏起来。



一台机器上的第  $n$  层与另一台机器上的第  $n$  层进行对话。在对话中用到的规则和约定合起来称为第  $n$  层协议。基本上,所谓协议(protocol)是指通信双方关于如何进行通信的一种约定。打个比方来说,当一位女士被介绍给一位男士的时候,她可能会选择伸出她的手,然后,这位男士可以根据特定的场合是决定握她的手,还是亲吻她的手。比如说,如果在一次商务会议中,她是一位美国律师,那么他应该选择握手;而如果在一个正式的舞会上,她是一位欧洲公主,那么他应该亲吻她的手。如果违反了协议,即使通信还有可能继续进行,但会变得非常困难。

图 1.13 显示了一个 5 层网络。不同机器上包含对应层的实体称为对等体(peer)。这些对等体可能是进程或者硬件设备,甚至可能是人。换句话说,正是这些对等体在使用协议进行通信。

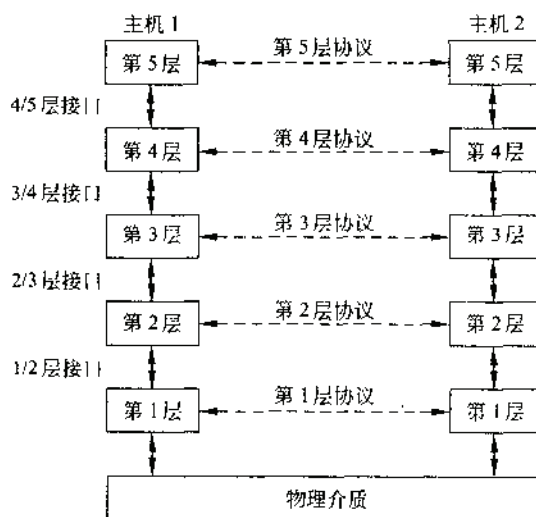


图 1.13 层、协议和接口

实际上,数据并不是从一台机器的第  $n$  层直接传递到另一台机器的第  $n$  层。相反,每一层都将数据和控制信息传递给它的下一层,这样一直传递到最底下的层。第 1 层下面是物理介质(physical medium),通过它进行实际的通信。在图 1.13 中,点线表示虚拟通信,实线表示物理通信。

在每一对相邻层之间是接口(interface)。接口定义了下层向上层提供哪些原语操作和服务。当网络设计者决定在一个网络中应该包含多少层,以及每一层应该提供哪些功能时,其中最重要的一个考虑是定义清楚层与层之间的接口。为了做到这一点,要求每层能完成一组特定的有明确含义的功能。除了尽可能地减少层与层之间必须要传递的信息的数量以外,层之间清晰的接口也会使我们很容易地用某一层的一个实现来代替另一个完全不同的实现(比如说,所有的电话线路都用卫星信道来代替),因为对于新的实现来说,它所需要做的仅仅是向紧邻的上层提供完全相同的一组服务,就如同原来的实现所做的一样。实际上,不同的主机使用不同的实现,这是很常见的。

层和协议的集合称为网络体系结构(network architecture)。网络体系结构的描述必

须包含足够的信息,以便实现者可以为每一层编写程序或者设计硬件,使之遵守有关的协议。实现的细节和接口的规范并不属于网络体系结构的内容,因为它们被隐藏在机器内部,对于外界是不可见的。甚至,一个网络中所有机器上的接口也不必都是一样的,实际上,每台机器只要能够正确地使用所有的协议即可。一个特定的系统所使用的一组协议(每一层一个协议)称为协议栈(protocol stack)。网络体系结构、协议栈以及协议是本书的主要内容。

打一个比方也许有助于更加清晰地表述多层通信的概念。假设有两位哲学家(第3层中的对等进程),其中一个会讲乌尔都语(Urdu)和英语,另一个会讲汉语和法语。由于他们俩人没有共同语言,所以他们每个人都雇佣了一个翻译(第2层中的对等进程),每个翻译又进一步联系了一位秘书(第1层中的对等进程)。哲学家1希望将对兔子(oryctolagus cuniculus)的感情传达给哲学家2。为了做到这一点,他将一条消息(英语)通过2/3层之间的接口传递给他的翻译,这条消息称“I like rabbits”,如图1.14所示。两位翻译有一种双方都能理解的中立语言:荷兰语,所以,这条消息被翻译为“Ik vind konijnen leuk。”。选择语言是第2层协议的事情,所以,使用什么样的语言取决于第2层的对等进程。

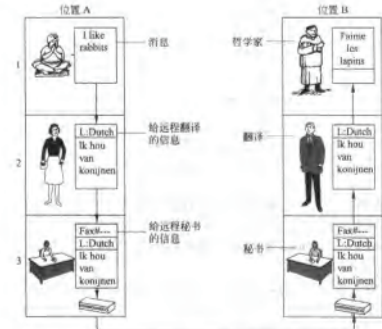


图 1.14 哲学家-翻译-秘书结构

然后,翻译将消息交给秘书,让她传出去,比如说,通过传真机来传送(这是第1层协议)。当消息到达的时候,翻译成法语,并通过2/3层之间的接口传递给哲学家2。注意,每个协议都与其他协议相互独立,只要接口不改变就行。翻译可以不使用荷兰语,比如说,他们可以使用芬兰语,只要他们能够达成一致,并且不改变第1层或者第3层的

接口即可。类似地,秘书可以不选择传真,而是电话,这样也不会影响其他层,甚至不需要通知其他层。每个进程还可以专门针对它的对等体加入一些信息。这部分信息不会被传递到上一层。

现在考虑一个更加技术性的例子:如何向图 1.15 中 5 层网络的最顶层提供通信。假设在第 5 层上运行的一个应用进程产生了一条消息 M,并且将它传递给第 4 层。第 4 层在消息的前面加上一个头(header)以标识该消息,并且把结果传给第 3 层。头部包含了控制信息,例如序列号,这使得即使下面三层没有维护顺序关系,目标机器的第 4 层也仍然可以按照正确的顺序递交消息。在有的层上,头部还可以包含消息大小、时间和其他的控制字段。

在许多网络中,对于第 4 层上传递的消息的大小并没有限制,但是第 3 层协议往往会强加一个限制。因此,第 3 层必须把进入的消息分割成较小的单元(packet,分组或包),并且在每一个分组的前面加上第 3 层的头。在这个例子中,M 被分割成两部分: M1 和 M2。

第 3 层决定使用哪些输出线路,并且把分组传递给第 2 层。第 2 层不仅在每一段信息上加上一个头部信息,还要加上一个尾部信息,并且将结果单元送给第 1 层以便进行物理传输。在接收端的机器上,消息从下往上逐层传递,在传递过程中头信息被逐层剥离。针对下层的头信息不会传递到上层来。

为了理解图 1.15,很重要的两个关系是:虚拟通信和实际通信之间的关系,以及协议和接口之间的关系。例如,从概念上讲,第 4 层中的对等进程可以认为“它们的通信是水平的”,它们使用了第 4 层协议。每一个对等进程可能都有一个类似于 SendToOtherSide 和 GetFromOtherSide 这样的过程,但是,这些过程实际上是与底下的层通过 3/4 层之间的接口进行通信的,并不是直接与另一端进行通信。

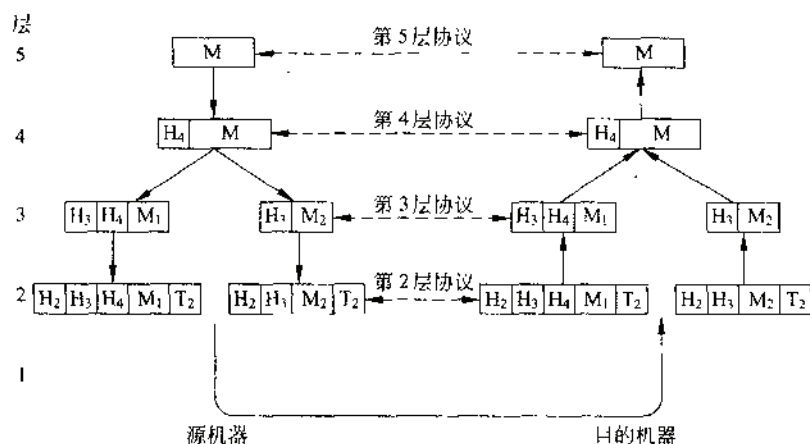


图 1.15 在第 5 层上支持虚拟通信的信息流示意图

关于对等进程的抽象概念对于所有的网络设计都是至关重要的。利用对等进程的思想,在设计完整的网络时,可以把难以管理的任务划分成几个较小的、易于处理的设计问题,也就是分层设计。

尽管本节名为“网络软件”，但是，需要指出的是，协议层次结构中一些较低的层往往是在硬件或者固件中实现的。然而，即使它们被（全部或者部分）嵌入到硬件中，仍然会涉及到复杂的协议算法。

### 1.3.2 各层的设计问题

在计算机网络中，有一些关键的设计问题可能会出现在好几个层中。下面，我们将简要地介绍其中一些比较重要的问题。

在每一层上，都需要有一种机制来标识出发送方和接收方。由于一般的网络中都有许多计算机，而且有些计算机上又有很多进程，所以，这就需要有一种方法使得一台机器上的进程可以指定它要与谁进行通话。由于存在多个目标，所以很自然地就需要一种**编址机制(addressing)**来指定一个特定的目标。

另外一组设计决策与数据传输的规则有关。在有些系统中，数据传输只能在一个方向上进行；而在其他的系统中，数据可以双向传输。协议也必须确定每个连接对应于多少个逻辑信道，并且确定它们的优先级别。许多网络为每个连接提供了至少两个逻辑信道，其中一个用于普通的数据，另一个用于紧急数据。

**错误控制(error control)**是另一个重要的方面，因为物理通信电路并不是完美无缺的。现在已经有了很多种错误检测和错误纠正编码方案，但是，连接两端必须统一使用同一种编码方案，而且，接收方还必须得有办法告知发送方，哪些报文已经正确接收到了，哪些报文还没有接收到。

并不是所有的通信信道都能够保持报文发送时候的先后顺序。为了解决可能出现的顺序错误，协议必须明确地保证接收方能够把所有的报文片断重新正确地组装起来。一种很显然的解决方案是对报文片断进行编号，但是，这种方案还遗留了一个问题没有解决，即当报文片断的到达顺序不正确时该如何进行处理。

在每一层上都会发生的一个问题是，当发送方发送速度很快的时候，如何避免接收方被数据淹没。目前已经提出了许多种方案，以后我们将会讨论到。有些方案涉及到某一种反馈机制，将接收方的当前状况反馈给发送方，可能是直接的反馈，也可能是间接的反馈。其他一些方案则限制发送方以商定的速率进行发送。这个研究主题称为**流控制(flow control)**。

在有些层上另一个必须要解决的问题是，并不是所有的进程都可以接受任意长的消息。这种特性导致了必须要实现这样的机制：拆分报文，传输，然后重组报文。一个相关的议题是，当某一层上的进程坚持要按照单元来传送数据，但是单元又很小以至于单独发送每个单元非常低效，这时该怎么办呢。这里的解决方案是，将几个要发往同一目标的短报文集成成为一个长的报文，然后在接受端再分解成原报文。

为每一对通信进程都建立一个单独的连接，有时候会很不方便，或者非常昂贵，此时下面的层可能会决定为多个不相关的对话使用同一个连接。只要这种**多路复用(multiplexing)**和**多路解复用(demultiplexing)**可以以透明的方式来完成，那么，任何一层都可以使用这种技术。例如，多路复用技术在物理层上是必要的，因为针对所有连接的流量都要在相对少量的物理电路上发送出去。

当源端和目标端之间存在多条路径的时候,必须进行路由选择。有时候,路由选择需要由两层或者多层来决定。例如,为了从伦敦发送数据到罗马,上层可能根据有关法律,决定经过法国还是德国。然后,在低层可能根据当前的流量负载,决定选择其中一条可用的电路。这个话题称为路由(routing)。

### 1.3.3 面向连接与无连接的服务

下层可以向上提供两种不同类型的服务:面向连接的服务,或者无连接的服务。在这一小节中,我们将介绍这两种类型的服务,并且看一下两者的区别。

**面向连接的服务(connection-oriented service)**是基于电话系统模型的。当你打电话的时候,为了与某个人通话,你首先拿起话机,拨对方的号码,然后说话,最后挂机。简单来说,为了使用面向连接的网络服务,用户首先要建立一个连接,然后使用该连接,最后释放连接。有关这种连接最本质的方面在于,它就好像一个管道:发送方把对象(数据位)压入管道的一端,接收方在另一端将它们取出来。在绝大多数情况下,数据位保持原来的顺序,所以,数据位都会按照发送的顺序到达。

在有些情况下,当一个连接建立的时候,发送方、接收方和子网一起协商(negotiation)一组将要使用的参数,比如最大的消息长度、所要求的服务质量,以及其他一些要素。通常情况下,一方应该提出一个建议,然后另一方接受或拒绝该建议,甚至提出相反的建议。

相反,**无连接的服务(connectionless service)**是基于邮政系统模型的。每一条报文(信件)都携带了完整的目标地址,所以,每条报文都可以被系统独立地路由。一般来说,当两条报文被发送给同一个目标的时候,首先被发送的报文将会先到达。然而,因为先发送的报文被延迟而导致后发送的消息先到达的情况也是有可能发生的。

每个服务都可以用一个**服务质量(quality of service)**来表述其特征。有些服务是可靠的,因为它们从来不丢失数据。一个可靠的服务通常是这样来实现的:让接收方向发送方确认收到了每一条消息,因而发送方就可以保证报文已经到达。确认的过程引入了额外的负载和延迟,一般情况下这是值得的,但有时也不尽然。

对于可靠的面向连接服务,一种典型的情形是文件传输。文件的所有者希望保证所有的位都能够正确地到达,而且到达的顺序与发送的顺序相同。如果一种文件传输服务偶尔会出现乱码或者丢失数据位,那么很少有顾客会愿意使用,即使它的传输速度再快也不会受到欢迎。

可靠的面向连接服务有两种变形:报文序列和字节流。前一种形式总是要保持报文的边界。发送两个 1024 字节的报文,收到的仍然是两个独立的 1024 字节的报文,决不可能是 一条 2048 字节的报文。而在后一种形式中,该连接只是一个字节流,没有任何报文边界。当 2048 个字节到达接收方的时候,接收方无法判断在发送的时候它是一个 2048 字节的消息,还是两个 1024 字节的消息,或者是 2048 个 1 字节的消息。如果一本书的每一页面当作单独的一个报文,通过网络发送给一台照排机,此时保持报文的边界可能会非常重要。另一方面,当一个用户登录到一台远程服务器上的时候,可能仅仅是一个从用户计算机到服务器的字节流就足够了。这时报文边界并不重要。



正如上面所提到的,对于有些应用而言,由于确认过程而引入的传输延迟是不可接受的。比如说,数字化的语音传输就是这样一个应用。对于电话用户来说,他们宁可不时地听到线路上有一点噪音,也不愿意等待因确认而造成的延迟。类似地,当传输一个视频会议流量的时候,有少数的像素错误不算什么问题,但是让图像停顿下来以便纠正错误则是很让人不可接受的。

并不是所有的应用都要求连接。例如,随着电子邮件日益普及,电子垃圾也变得日益膨胀起来。电子垃圾邮件的发送方可能并不愿意只是为了发送一条消息就建立连接,然后再拆除连接。百分之百的可靠递交并不那么重要,特别是当它所付出的代价很昂贵的时候更是如此。这里所需要的是一种发送单个电子邮件的方法,它只要确保这些邮件以极高的概率到达,而不需要保证这些消息一定到达。不可靠(意味着没有被确认)的无连接服务通常称为数据报服务(datagram service),它与电报服务非常类似,在电报服务中,它也不会给发送方送回确认消息。

在其他有些情况下,“不需要建立连接就可以发送一条短的消息”这种做法确实非常方便,而且这种方便性也正是我们所期望的,但是可靠性仍然非常必要。对于这些应用来说,有确认的数据报服务(acknowledged datagram service)是非常合适的。这就像在寄送挂号信的时候还要求一个回执一样。当发送方收到回执的时候,他就可以绝对相信这封信已经被送到对方手中了,没有在投递的途中丢失。

还有一种服务是请求-应答服务(request-reply service)。在这种服务中,发送方传输一个数据报,其中只包含一个请求;应答数据报中包含了答案。例如,向本地图书馆询问哪里的人讲维吾尔语就属于这一类服务。请求-应答服务通常用来实现客户-服务器模型中的通信过程:客户发出一个请求,然后服务器作出应答。图 1.16 总结了以上讨论的服务类型。

		服 务	例 子
面向连接		可靠的报文流	页码页序列
		可靠的字节流	远程登录
		不可靠的连接	数字化的语音
无连接		不可靠的数据报	电子垃圾邮件
		有确认的数据报	挂号信
		请求-应答	数据库查询

图 1.16 服务类型及其例子

刚开始的时候,人们对于为什么要使用不可靠连接的想法可能会感到非常迷惑。毕竟,为什么会有人愿意舍弃可靠通信而选择不可靠的通信呢?首先,可靠通信(这里可靠通信的含义是指有确认的)并不总是可以使用的。例如,以太网并没有提供可靠通信。有些分组可能会在传输过程中被损坏。这个问题将由上层的协议来处理。其次,为了提供可靠服务而导致的延迟可能是不可接受的,特别是在实时应用中,比如传输实时多媒体数据。由于这些原因,所以可靠通信和不可靠通信都存在。

### 1.3.4 服务原语

一个服务通常是由一组原语(**primitive**)操作来描述的,用户进程通过这些原语操作可以访问该服务。这些原语告诉该服务执行某个动作,或者将某个对等体所执行的动作报告给客户。如果协议栈位于操作系统中(大多数情况是这样的),则这些服务原语通常是一些系统调用。这些系统调用会进入到内核模式,然后在内核模式中控制该机器,让操作系统发送必要的分组。

到底哪些原语是可以使用的,这取决于所提供的服务。针对面向连接服务的原语与针对无连接服务的原语是不同的。举一个最小的服务原语的例子,在客户-服务器环境中,为了实现一个可靠的字节流,可以考虑的原语如图 1.17 所示。

原 语	含 义
LISTEN	阻塞操作,等待一个进入的连接
CONNECT	与一个正在等待的对等体建立连接
RECEIVE	阻塞操作,等待一个进入的报文
SEND	给对等体发送一个报文
DISCONNECT	终止一个连接

图 1.17 5 个服务原语,用于实现一个简单的面向连接服务

这些原语可能的用法如下所述。首先,服务器执行 LISTEN,表示它已经准备好接收到来的连接。通常,实现 LISTEN 的做法是将它做成一个阻塞的系统调用。在执行了 LISTEN 原语之后,服务器进程被阻塞,直到有连接请求到来为止。

接下来,客户进程执行 CONNECT,以便与服务器建立连接。CONNECT 调用需要指定连接的目标方,即它要跟谁建立连接,所以它可能带一个参数来指定服务器的地址。然后,操作系统通常会向对方发送一个分组,请求建立连接,如图 1.18 中(1)所示。然后客户进程被挂起,直到有应答为止。当该分组到达服务器一方的时候,首先由服务器的操作系统对它进行处理。当系统看到这是一个请求连接的分组时,它检查是否有进程正在监听进入的连接。如果有的话,它做两件事情:对监听进程解除阻塞,并且送回一个确认报文(2)。当客户进程接收到该确认报文后,恢复运行状态。这时候客户和服务器都在运行,它们已经建立了一个连接。需要说明的一点是,确认报文(2)是由协议代码本身产生的,不是对用户层原语的响应。如果一个连接请求到达服务器方时,服务器方并没有监听进程,则结果是未定义的。在有些系统中,该分组可能会被放到队列中等待一段时间,期望在这段时间中会有进程调用 LISTEN。

在现实生活中,与此协议类似的一种情况是:顾客(客户)用电话呼叫一家公司的客户服务主管。当电话铃声响起来的时候,服务主管走到电话旁边。然后客户准备这一次呼叫。当主管拿起电话的时候,连接便建立起来了。

下一步是服务器执行 RECEIVE,以便做好准备接收第一个请求。通常情况下,服务器一旦从 LISTEN 中释放出来之后,马上就会执行 RECEIVE,这时确认消息往往还没有

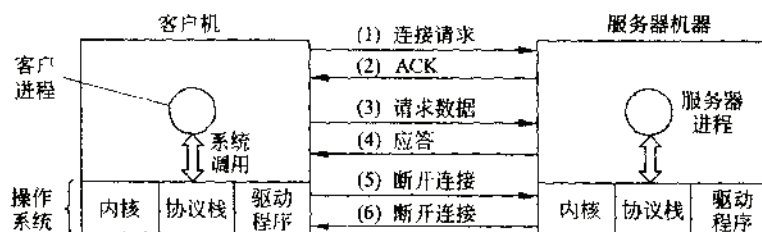


图 1.18 在一个面向连接的网络中,简单的客户-服务器交互过程中发送的分组

到达客户方。RECEIVE 调用会阻塞服务器。

接着,客户执行 SEND,以便将它的请求传出去(3),然后执行 RECEIVE 以便得到应答。

请求分组到达服务器方之后,服务器进程会从阻塞中解放出来,这样它就可以开始处理该请求了。在处理完该请求工作之后,利用 SEND 将请求的答案送回给客户(4)。该分组到达客户方之后,客户方解除阻塞,然后检查服务器送到的答案。如果客户还有其他的请求,它现在可以继续发送这些请求。如果客户的任务已经完成了,则它可以使用 DISCONNECT 终止当前连接。一般情况下,初始的 DISCONNECT 是一个阻塞调用,它会将客户进程挂起,并且给服务器发送一个分组,表明该连接已经不再需要了(5)。当服务器收到这个分组的时候,它也会发出一个 DISCONNECT,作为对客户的响应,并释放该连接。当服务器的分组(6)到达客户机的时候,客户进程恢复运行,该连接被正式断开。总之,这就是面向连接的通信过程。

当然,实际的过程不会这么简单。其中很多地方都有可能出现错误。时间顺序可能会出错(比如 CONNECT 在 LISTEN 之前完成)、分组可能会丢失,等等。我们将在后面的章节中详细地讨论这些问题。而图 1.18 只是简要地说明了客户-服务器之间的通信是如何在一个面向连接的网络上进行的。

我们看到,为了完成这个协议,总共需要 6 个分组,你可能会问:为什么不使用一个无连接的协议呢?这个问题的答案是,在一个理想的环境中,这样做是可以的,这时候只需要两个分组就可以了:一个用于请求,另一个用于应答。然而,无论在哪个方向上出现很大的报文(比如 1MB 大小的文件)的时候,传送错误、丢失分组都有可能发生,所以情况就完全不一样了。如果应答是由几百个分组组成的,则有的分组可能会在传递过程中丢失,那么,客户如何知道是否有数据丢失了呢?客户如何知道最后接收到的分组正好是服务器真正发送的最后一个分组呢?假如客户还要请求第二个文件,那么客户如何区别到底是第二个文件的第一个分组,还是第一个文件迟到的一个分组呢?简而言之,在实践中,在一个不可靠的网络上,简单的请求-应答协议往往是不能胜任的。在第 3 章中,我们将详细地学习各种协议,看看它们如何克服这些问题,以及其他的一些问题。现在,我们只要清楚这一点:在进程之间有一个可靠的、有序的字节流服务有时是非常方便的。

### 1.3.5 服务与协议的关系

服务和协议是截然不同的概念,但是二者却常常被混淆在一起。它们的区别非常重

要,我们有必要在这里再次强调一下。服务是指某一层向它上一层提供的一组原语(操作)。服务定义了该层打算代表其用户执行哪些操作,但是它并不涉及如何实现这些操作。服务也会涉及到两层之间的接口,其中低层是服务提供者,而上层是服务的用户。

与此不同的是,协议是一组规则,用来规定同一层上的对等实体之间所交换的消息或者分组的格式和含义。这些实体利用协议来实现它们的服务定义。它们可以自由地改变协议,但是不能改变服务,因为这些服务对于它们的用户是可见的。按照这种方式,服务和协议是完全分离开的。

换句话说,服务涉及到层之间的接口,如图 1.19 所示。相反,协议涉及到不同机器上对等实体之间发送的分组。请不要混淆这两个概念,这是非常重要的。

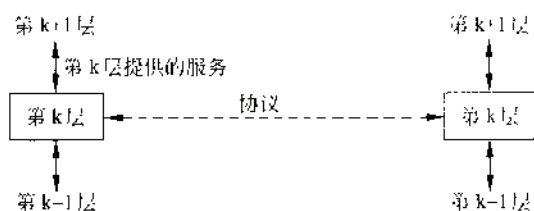


图 1.19 服务和协议之间的关系

我们也可以编程语言来对这两个概念作一个类比。服务就好像是面向对象语言中的抽象数据类型或者对象,它定义了对象上可以执行的操作,但是并没有指定这些操作该如何实现。协议涉及到服务的具体实现,它对于该服务的用户是不可见的。

许多老的协议并没有将服务和协议区分开。在实践中,一个典型的层可能会有一个“SEND PACKET”服务原语,并且在该原语中,由用户提供一个指针,指向一个完全装配好的分组。这样的设计意味着,对于协议的任何一点变化都会暴露给用户。现在,大多数的网络设计者都把这样的设计看作是一种严重的错误。

## 1.4 参考模型

上一节我们抽象地讨论了分层的网络,现在来看几个具体例子。在接下去的两小节中,我们将讨论两种重要的网络体系结构:OSI 参考模型和 TCP/IP 参考模型。尽管与 OSI 模型相关的协议已经很少再使用了,但是,该模型本身是非常通用的,并且仍然有效,在每一层上讨论到的特性也仍然非常重要。TCP/IP 模型有不同的特点:模型本身并不非常有用,但是协议却被广泛使用开了。出于这样的原因,我们对这两个模型都做详细的介绍。而且,有时候,从失败事例中比从成功事例中可以学到更多的知识。

### 1.4.1 OSI 参考模型

OSI 模型如图 1.20 所示(省略了物理介质)。该模型是以国际标准化组织(ISO, International Standards Organization)的一份提案为基础的,它为各层所使用的协议的国际标准化迈出了第一步(Day and Zimmermann, 1983),并且于 1995 年进行了修订(Day,

1995)。该模型称为“ISO OSI(Open Systems Interconnection) Reference Model”,因为它涉及到如何将开放的系统(也就是指那些为了与其他系统相互通信而开放的系统)连接起来。为了简短起见,我们将它称为 OSI 模型。

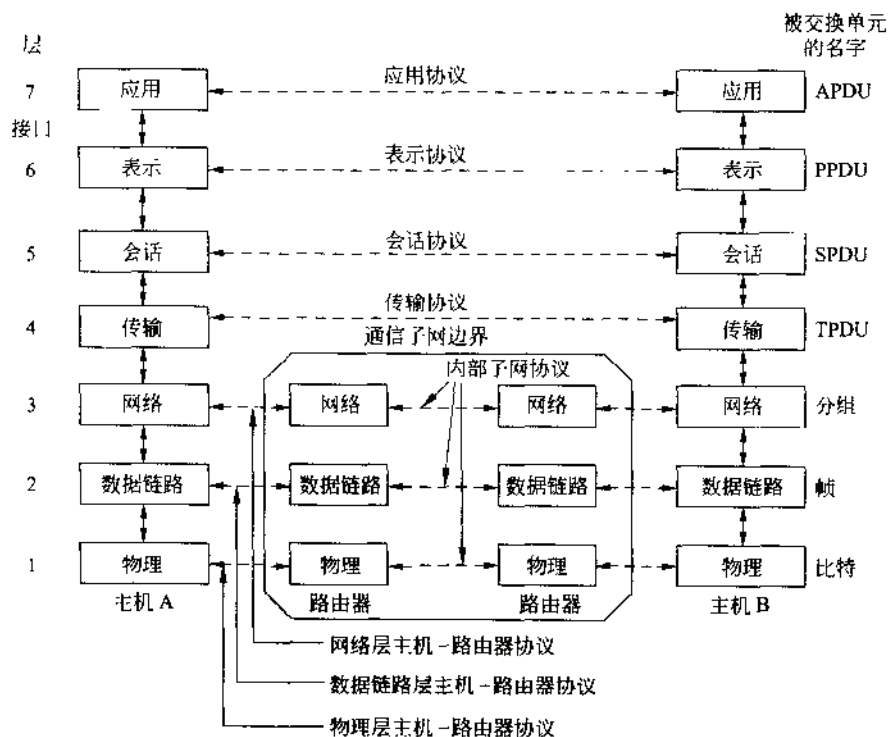


图 1.20 OSI 参考模型

OSI 模型有 7 层。这 7 层的分层原则如下所述:

- (1) 当需要一个不同抽象体的时候,应该创建一层。
- (2) 每一层都应该执行一个明确定义的功能。
- (3) 选择每一层功能的时候,应该考虑到定义国际化的协议。
- (4) 选择层边界的时候,应该使“跨接口所需要的信息流”尽可能最小。

(5) 层数应该足够多,以保证不同的功能不会被混杂在同一层中,同时层数也不能够太多,以避免整个体系结构变得过于庞大。

下面我们将从最底层开始,依次讨论该模型中的每一层。请注意,OSI 参考模型本身并不是一个网络体系结构,因为它并没有定义每一层上所用到的服务和协议。它只是指明了每一层上应该做些什么事情。然而,ISO 也已经为每一层制定了相应的标准,但这些标准并不属于参考模型本身,它们都已作为单独的国际标准发布了。

### 物理层

物理层(physical layer)涉及到在通信信道上传输的原始数据位。在设计的时候必须



要保证,当一方发送了“1”时,在另一方收到的也是“1”,而不是“0”。这里的典型问题是:应该用多少伏的电压来表示“1”;多少伏的电压表示“0”;每一位持续多少纳秒(ns);传输过程是否在两个方向上同时进行;初始连接如何建立;当双方结束之后如何撤销连接;网络连接器的针脚以及每一针的用途是什么。这里的设计问题主要涉及到机械、电子和定时接口(timing interface),以及位于物理层之下的物理传输介质等。

### 数据链路层

**数据链路层(data link layer)**的主要任务是将一个原始的传输设施转变成一条逻辑的传输线路,在这条传输线路上,所有未检测出来的传输错误也会反映到网络层上。数据链路层完成这项任务的做法是:让发送方将输入的数据拆开,分装到**数据帧(data frame)**,通常为几百个或者几千个字节)中,然后顺序地传送这些数据帧。如果是可靠的服务,则接收方必须确认每一帧都已经正确地接收到了,即给发送方送回一个**确认帧(acknowledgement frame)**。

数据链路层上的另一个问题是(大多数高层都有这样的问题),如何避免一个快速的发送方“淹没”掉一个慢速的接收方。所以,往往需要一种流量调节机制,以便让发送方知道接收方当前时刻有多大的缓存空间。通常情况下,这种流调节机制跟错误处理机制集成在一起。

对于广播式网络,在数据链路层上还有另一个问题:如何控制对于共享信道的访问。数据链路层的一个特殊子层,即介质访问控制子层,就是专门针对这个问题的。

### 网络层

**网络层(network layer)**控制子网的运行过程。一个关键的设计问题是确定如何将分组从源端路由到目标端。从源端到目标端的路径可以建立在静态表的基础之上,这些表相当于是网络的“布线”图,而且很少会变化。这些路径也可以在每一次会话开始时就确定下来,例如一次终端会话(比如,登录到一台远程机器上)。另外,这些路径也可以是高度动态的,针对每一个分组都要重新确定路径,以便符合网络当前的负载情况。

如果有太多的分组同时出现在一个子网中,那么这些分组彼此之间会相互妨碍,从而形成传输瓶颈。拥塞控制也属于网络层的范畴。更进一步讲,所提供的服务质量(比如延迟、传输时间、抖动,等等)也是网络层考虑的问题。

当一个分组必须从一个网络传输到另一个网络才能够到达目的地时,可能会发生很多问题。第二个网络所使用的编址方案可能与第一个网络不同;第二个网络可能根本不接受这个分组,因为它太大了;两个网络所使用的协议也可能不一样,等等。网络层应负责解决这些问题,从而允许不同种类的网络可以相互连接起来。

在广播式网络中,路由问题比较简单,所以网络层往往比较薄,甚至根本不存在。

### 传输层

**传输层(transport layer)**的基本功能是接受来自上一层的数据,并且在必要的时候把这些数据分割成小的单元,然后把数据单元传递给网络层,并且确保这些数据片段都能够

正确地到达另一端。而且,所有这些工作都必须高效率地完成,并且必须使上面各层不受底层硬件技术变化的影响。

传输层还决定了将向会话层(实际上最终是向网络的用户)提供哪种类型的服务。其中最为常见的类型是,该传输连接是一个完全无错(error-free)的点到点信道,此信道按照原始发送的顺序来传输报文或者字节数据。然而,其他类型的传输服务也是可能的,例如传输独立的报文(不保证传送的顺序)、将报文广播给多个目标等。服务的类型是在建立连接时就确定下来的(顺便说一下,真正完全无错的信道是不可能实现的,人们使用这个术语的真正含义是指,错误的发生率足够小,以致于在实践中可以忽略这样的错误)。

传输层是一个真正的端到端的层,所有的处理都是按照从源端到目标端来进行的。换句话说,源机器上的一个程序利用报文头与控制信息,与目标机器上的一个类似的程序进行对话。在其下面的各层上,协议存在于每台机器与它的直接邻居之间,而不存在于最终的源机器和目标机器之间,源机器和目标机器可能被许多中间路由器隔离开了。第1层到第3层是被串连起来的,而第4层到第7层是端到端的,它们之间的区别如图1.20所示。

### 会话层

会话层(session layer)允许不同机器上的用户之间建立会话。所谓会话,通常是指各种服务,包括对话控制(dialog control,记录下该由谁来传递数据了)、令牌管理(token management,禁止两方同时执行同一个关键操作),以及同步功能(synchronization,在一个长的传输过程中设置一些检查点,以便在系统崩溃之后还能够崩溃前的点上继续执行)。

### 表示层

在表示层下面的各层中,它们最关注的是如何传递数据位,而表示层(presentation layer)关注的是所传递的信息的语法和语义。不同的计算机可能会使用不同的数据表示法,为了让这些计算机能够进行通信,它们所交换的数据结构必须以一种抽象的方式来定义;同时,表示层还应该定义一种标准的编码方法,用来表达网络线路上所传递的数据。表示层管理这些抽象的数据结构,并允许定义和交换更高层的数据结构(比如银行账户记录)。

### 应用层

应用层(application layer)包含了各种各样的协议,这些协议往往直接针对用户的需要。一个被广泛使用的应用协议是HTTP(HyperText Transfer Protocol,超文本传输协议),它也是WWW(World Wide Web,万维网)的基础。当浏览器需要一个Web页面的时候,它利用HTTP将所要页面的名字发送给服务器,然后服务器将页面送回给浏览器。其他还有一些应用协议用于文件传输、电子邮件以及网络新闻等。

### 1.4.2 TCP/IP 参考模型

现在我们从 OSI 参考模型转换到另一个参考模型,该参考模型不仅被所有广域计算机网络的鼻祖 ARPANET 所使用,也被 ARPANET 的继承者——全球范围内的 Internet 所使用。尽管后面我们还要粗略地讲述 ARPANET 的历史,但是现在先简要地介绍 ARPANET 的一些要点也是有必要的。ARPANET 是由 DoD(美国国防部)资助的一个研究性网络。它通过租用的电话线,将几百所大学和政府部门的计算机设备连接起来。后来卫星和无线电网络也加入进来,原来的协议在与它们互连的时候遇到了问题,所以需要一种新的参考体系结构。因此,能够以无缝的方式将多个网络连接起来,这是从一开始就面临的设计目标之一。在经过了两个基本的协议之后,这个体系结构后来演变成了 **TCP/IP 参考模型**。该体系结构最初是在(Cerf and Kahn, 1974)中定义的。后来在(Leiner et al., 1985)中又展示了一个新的视图。在(Clark, 1988)中讨论了该模型背后的设计思想。

由于美国国防部担心一些贵重的机器、路由器和互连网关可能会在某一时刻突然崩溃,所以,另一个主要的目标是,即使在子网硬件有丢失的情况下,网络必须还能够继续工作,原有的会话不会被打断。换句话说,美国国防部希望:即使源机器和目标机器之间的某一些机器或者传输线路突然不能工作了,只要源机器和目标机器仍然还在工作,那么它们之间的连接还可以继续进行下去。而且,当时已经可以预见,由于不同应用的需求差别很大(从文件传输到实时的语音传输),所以迫切需要一种灵活的网络体系结构。

#### 互联网层

所有这些需求导致最终选择了分组交换网络,它以一个无连接的互连网络层为基础。这一层称为**互联网层(internet layer)**,它是将整个网络体系结构贯穿在一起的关键层。该层的任务是,允许主机将分组发送到任何网络上,并且让这些分组独立地到达目标端(目标端有可能位于不同的网络上)。这些分组到达的顺序可能与它们被发送时候的顺序不同,在这种情况下,如果有必要保证顺序递交的话,则重新排列这些分组的任务由高层来负责。请注意,虽然在 Internet(中文称为因特网)中也包含了互联网层,但是,这里“互联网(internet)”的用法泛指一般含义。

这里我们可以将互联网层与(缓慢的)邮政系统作一个类比。在某一个国家,一个人可以将多封国际信件投递到一个邮箱中,通常情况下,这些信件大多会被投递到目标国家的正确地址,有可能这些信件在沿途会经过一个或者多个国际邮件关卡,但是,这对于用户来说是完全透明的。而且,每个国家有它自己的邮戳,信封大小规格也有所不同,投递的规则也有差异,这些对于用户而言都是不可见的。

互联网层定义了正式的分组格式和协议,该协议称为 **IP(Internet Protocol)**。互联网层的任务是将 IP 分组投递到它们该去的地方。很显然,分组路由和避免拥塞是这里最主要的问题。由于这些原因,所以,我们可以这样说,TCP/IP 的互联网层在功能上类似于 OSI 的网络层。图 1.21 显示了这种对应关系。

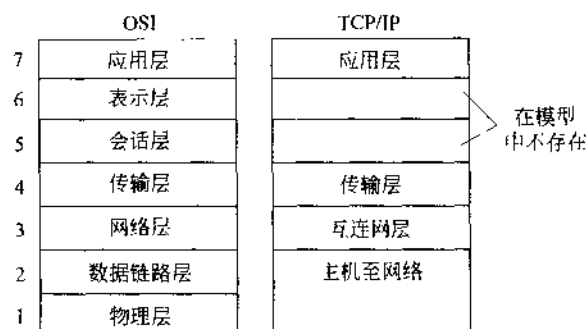


图 1.21 TCP/IP 参考模型

### 传输层

在 TCP/IP 模型中,位于互联网层之上的那一层现在通常称为传输层(transport layer)。它的设计目标是,允许源和目标主机上的对等体之间可以进行对话,就如同 OSI 的传输层中的情形一样。这里已经定义了两个端到端的传输协议。第一个是 TCP (Transport Control Protocol, 传输控制协议),它是一个可靠的、面向连接的协议,允许从一台机器发出的字节流正确无误地递交到互联网上的另一台机器上。它先把输入的字节流分割成单独的小报文,并把这些报文传递给互联网层。在目标方,负责接收数据的 TCP 进程把收到的报文重新装配到输出流中。TCP 还负责处理流控制,以便保证一个快速的发送方不会因为发送太多的报文,超出了慢速接收方的处理能力,而把它淹没掉。

第二个协议是 UDP (User Datagram Protocol, 用户数据报协议),它是一个不可靠的、无连接的协议,主要用于那些“不想要 TCP 的序列化或者流控制功能,而希望自己提供这些功能”的应用程序。UDP 广泛应用于“只需要一次的、客户-服务器类型的请求-应答查询”,以及那些“快速递交比精确递交更加重要”的应用,比如传输语音或者视频。IP、TCP 和 UDP 三者之间的关系如图 1.22 所示。自从这个模型出现以来,许多其他的网络上也都已经实现 IP 了。

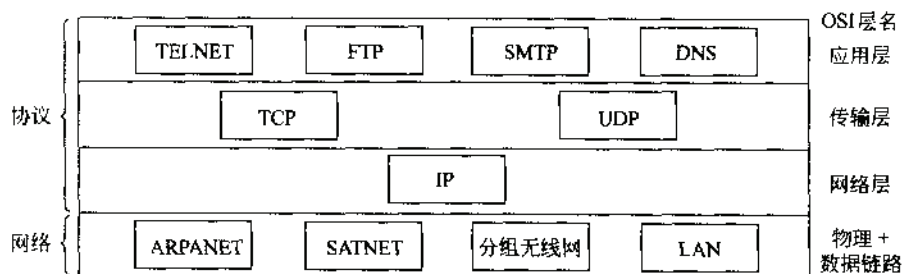


图 1.22 TCP/IP 模型中早期的协议和网络

## 应用层

TCP/IP 模型并没有会话层和表示层。由于当时感觉到并不需要它们,所以没有将它们包含进来。来自 OSI 模型的经验已经证明这种观点是正确的:对于大多数应用来说,这两层并没有用处。

在传输层之上是应用层(application layer),它包含了所有的高层协议。最早的高层协议包括虚拟终端协议(TELNET)、文件传输协议(FTP)和电子邮件协议(SMTP)等,如图 1.22 所示。虚拟终端协议允许一台机器上的用户登录到远程的机器上,并且在远程的机器上进行工作。文件传输协议提供了一种在两台机器之间高效地移动数据的途径。电子邮件协议最初只是一种文件传输的方法,但是后来为此专门开发了一个协议(SMTP)。经过了这么多年的发展以后,许多其他的协议也加入到了应用层上:DNS(Domain Name System,域名系统)将主机名字映射到它们的网络地址;NNTP 用于传递 USENET 的新闻;HTTP 用于获取 WWW 上的页面;等等。

## 主机至网络层

在互联网层下面是一片空白。TCP/IP 参考模型并没有明确规定这里应该有哪些内容,它只是指出,主机必须通过某个协议连接到网络上,以便可以将分组发送到网络上。参考模型没有定义这样的协议,而且不同的主机、不同的网络使用的协议也不尽相同。关于 TCP/IP 模型的书和文章很少讨论这一层上的协议。

### 1.4.3 OSI 参考模型与 TCP/IP 参考模型的比较

OSI 和 TCP/IP 参考模型有很多共同点。两者都以协议栈的概念为基础,并且协议栈中的协议彼此相互独立。而且,两个模型中各个层的功能也大体相似。例如,在两个模型中,传输层以及传输层以上的各层都为希望进行通信的进程提供了一种端到端的、与网络无关的服务。这些层形成了传输提供方。另外,在这两个模型中,传输层之上的各层也都是传输服务的用户,并且是面向应用的用户。

除了这些基本的相似之处以外,两个模型也有许多不同的地方。在这一小节中,我们将注意力集中在两个模型之间的关键差别上。有一点很重要,也需要注意,我们这里比较的是参考模型,而不是对应的协议栈。关于协议本身,我们将在后面讨论。关于完整地比较和对比 TCP/IP 和 OSI 的书,请参考 Piscitello and Chanpin, 1993。

对于 OSI 模型,有三个概念是它的核心:

- (1) 服务;
- (2) 接口;
- (3) 协议。

OSI 模型最大的贡献是使这三个概念的区别变得更加明确了。每一层都为它的上一层执行一些服务。服务的定义指明了该层做些什么,而不是上一层的实体如何访问这一层,或这一层是如何工作的。它定义了这一层的语义。

每一层的接口告诉它上面的进程应该如何访问本层。它规定了有哪些参数,以及结



果是什么。但是它并没有说明本层内部是如何工作的。

最后,每一层上用到的对等协议是本层自己内部的事情。它可以使用任何协议,只要它能够完成任务就行(也就是说提供所承诺的服务)。它也可以随意地改变协议,而不会影响它上面的各层。

这些思想与现代的面向对象的程序设计思想非常吻合。一个对象就如同一个层一样,它有一组方法(或者叫操作),对象之外的过程可以调用这些方法。这些方法的语义规定了该对象所提供的服务集合。方法的参数和结果构成了对象的接口。对象的内部代码是它的协议,对于外部而言是不可见的,也不需要被外界关心。

最初,TCP/IP 模型并没有明确地区分服务、接口和协议三者之间的差异,但是在它成型之后,人们已经努力对它作了改进,以便更加接近于 OSI。例如,互联网层提供的真正服务只有发送 IP 分组(SEND IP PACKET)和接收 IP 分组(RECEIVE IP PACKET)。

因此,OSI 模型中的协议比 TCP/IP 模型中的协议有更好的隐蔽性,当技术发生变化的时候,OSI 模型中的协议相对更加容易被替换为新的协议。最初采用分层协议的主要目的之一就是能够做这样的替换。

OSI 参考模型是在协议发明之前就已经产生的。这种顺序关系意味着 OSI 模型不会偏向于任何某一组特定的协议,因而该模型更加具有通用性。这种做法的缺点是,设计者在这方面没有太多的经验可以参考,因此不知道哪些功能应该放在哪一层上。

例如,数据链路层最初只处理点到点网络。当广播式网络出现以后,必须在模型中嵌入一个新的子层。当人们使用 OSI 模型和已有的协议来建立实际的网络时,才发现这些网络并不能很好地匹配所要求的服务规范(万分的惊讶!),因此不得不在模型中加入一些子层,以便提供足够的空间来弥补这些差异。还有,标准委员会最初期望每一个国家都将有一个由政府来运行的网络并使用 OSI 协议,所以根本不考虑网络互连的问题。总而言之,事情并不是像预期的那样。

而 TCP/IP 却正好相反:协议先出现;TCP/IP 模型只是这些已有协议的一个描述而已。所以,协议一定会符合模型,这肯定没有问题。而且两者确实吻合得很好。惟一的问题在于,TCP/IP 模型并不适合任何其他的协议栈,因此,要想描述其他非 TCP/IP 网络,该模型并不很有用。

现在我们从两个模型的基本思想转到更为具体的方面上来,它们之间一个很显然的区别是层的数目:OSI 模型有 7 层,而 TCP/IP 只有 4 层。它们都有网络层(或者是互联网层)、传输层和应用层,但是其他的层并不相同。

另一个区别在于无连接的和面向连接的通信范围有所不同。OSI 模型的网络层同时支持无连接和面向连接的通信,但是传输层上只支持面向连接的通信,这是由该层的特点所决定的(因为传输服务对于用户是可见的)。TCP/IP 模型的网络层上只有一种模式(即无连接通信),但是在传输层上同时支持两种通信模式,这样可以给用户一个选择的机会。这种选择机会对于简单的请求-应答协议特别重要。

#### 1.4.4 OSI 模型和协议的缺点

不管是 OSI 模型及其协议,还是 TCP/IP 模型及其协议,都不是完美无缺的。对它们

都有不少的批评意见。在这一小节以及下一小节,我们来讨论它们的一些缺点。首先我们从 OSI 模型开始,然后再介绍 TCP/IP 模型的缺点。

在本书的第二版出版的时候(即 1989 年),这个领域中的大多数专家都觉得 OSI 模型及其协议将会统领整个世界,从而把所有其他的技术和标准都排除出局。这种情况并没有发生。为什么?回顾以前的一些教训,可能会给我们一些启示,进而帮助我们做得更好。这些教训可以概括如下:

- (1) 糟糕的时机;
- (2) 糟糕的技术;
- (3) 糟糕的实现;
- (4) 糟糕的政策。

### 糟糕的时机

首先我们来看一看第一个理由:糟糕的时机。一个标准是在什么时候被建立起来的,这对于该标准的成功是绝对非常重要的。M. I. T. 的 David Clark 有一套关于标准的理论,他称之为“两头大象的启示”,如图 1.23 所示。

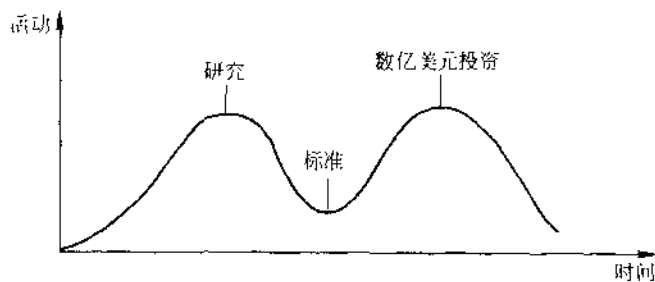


图 1.23 两头大象的启示

该图显示了围绕一个新主题的活动情况。当新主题被首次发现时,会一下子出现大量的研究活动,包括各种各样的形式,比如讨论、论文和会议等。过了一段时间之后,这种活动逐渐平息下来,有些企业发现了该主题,于是数亿美元的投资热潮就开始了。

关键的一点在于,标准的编写工作必须处于两头“大象”的槽中间。如果标准编写得太早,研究工作还未完成,那么人们对于该主题还没有理解透彻;其结果就是标准很差。如果编写标准太晚,则许多公司可能已经通过各种不同的方式投入了大量的资金,因而标准就会被这些公司所忽略。如果两头大象之间的间隔太短(因为大家都急于快速启动)的话,开发标准的人有可能被夹在中间而举步维艰。

现在看起来,标准的 OSI 协议就是这样被夹在中间了。当 OSI 协议出现的时候,与之竞争的 TCP/IP 协议已经被广泛地应用于大学和科研机构了。虽然几十亿美元的投资热潮尚未开始,但是,学院市场本身就足够大,因而许多厂商已经在谨慎地提供 TCP/IP 产品了。当 OSI 出现的时候,这些厂商并不想再支持第二个协议栈,除非他们被迫这样做,因此 OSI 没有得到初始的投入。由于每一家公司都在等待其他的公司先行一步,结果是,没有一家公司先走这一步,所以 OSI 从来没有真正被实现过。

## 糟糕的技术

OSI 一直没有流行起来的第二个理由是,无论是模型还是协议,都有缺陷。之所以选择 7 层,很大程度上是由于策略的原因,而并非技术的因素,其中的两层(会话层和表示层)几乎是空的,而另外两层(数据链路层和网络层)又包含内容太多。

OSI 模型以及相应的服务定义和协议都极其复杂。如果将标准打印出来,堆叠起来的文档可以达到 1 米的高度。它们很难实现,而且操作起来也很低效。写到这里,我脑子出现了 Paul Mockapetris 出的一个谜语,它也在(Rose,1993)中被引用过:

问:当你碰到一个拥有国际标准的霸权主义者时会怎样?

答:他会给你一些你所不能理解的东西。

除了难以理解以外,OSI 的另一个问题是,有些功能,比如编址、流控制和错误控制等,都会在每一层上重复地出现。例如,Saltzer 等(1984)已经指出,要想真正有效,错误控制必须在最高层上完成,所以,在低层上不断地重复这一功能往往是不必要的,也是低效的。

## 糟糕的实现

由于 OSI 模型和协议过于复杂了,因此最初的那些实现不仅庞大,而且很笨拙,也很慢,这一点不足为奇。任何试图使用这些实现的人无不被搞得焦头烂额。没过多久,人们就把“OSI”与“糟糕的质量”联系在一起了。尽管随着时间的推移,这些产品有了改进,但是这种印象留在人们的心中了。

相反,TCP/IP 的早期实现之一是 Berkeley UNIX 的一部分,并且非常好(更不用说它是免费的)。很快地,人们就开始使用它,进而导致形成了一个庞大的用户群,这进一步促进了它的提高和改进,然后又导致更大的用户群体。这是螺旋式上升而不是下降的。

## 糟糕的策略

由于 TCP/IP 最初的实现,很多人,特别是在学术界,都把 TCP/IP 看作是 UNIX 的一部分,而 UNIX 在 20 世纪 80 年代的学术圈中盛极一时,备受宠爱。

相反,OSI 则被认为是欧洲电信部门、欧共体以及后来的美国政府的产物,这种观点部分是正确的,这也说明了,“政府官僚们试图把技术上不足的标准强加给那些正在实际开发计算机网络的可怜的研究人员和程序员”这种做法并不奏效。与 OSI 模型有类似结果的事件有:IBM 在 20 世纪 60 年代的时候宣布,PL/I 将是未来的语言;美国国防部后来做了修正,声称未来的语言是 Ada。

### 1.4.5 TCP/IP 参考模型的缺点

TCP/IP 模型和协议也有问题。首先,该模型并没有清楚地区分服务、接口和协议的概念。好的软件工程师在实践中都要求区分哪些是规范,哪些是实现,这一点 OSI 模型非常谨慎地做到了,但是 TCP/IP 模型并没有做到。因此,在使用新技术来设计新的网络的时候,TCP/IP 模型并不是一个很好的参照。

第二, TCP/IP 模型并不通用, 它不适用于用来描述 TCP/IP 之外的任何其他协议栈。例如, 试图使用 TCP/IP 模型来描述 Bluetooth(蓝牙)是完全不可能的。

第三, 在分层协议环境中, 主机至网络层并不是常规意义上的层的概念。它是一个接口(位于网络层和数据链路层之间)。接口和层的区别是非常重要的, 我们不能对此粗心大意。

第四, TCP/IP 模型并没有区分(甚至没有提及)物理层和数据链路层。它们是完全不同的。物理层必须要考虑铜线、光纤和无线通信的传输特征。而数据链路层的任务则是确定帧的起点和终点的位置, 并且按照期望的可靠程度把这些帧从一端发送到另一端。一个正确的模型应该把它们作为单独的层包含进来。TCP/IP 模型并没有这样做。

最后, 尽管 IP 和 TCP 协议进行了仔细设计, 并且很好地实现了, 但是, 在实际中还有很多其他特定的协议, 通常这些协议是由一群研究生摸索出来的。然后, 这些协议的实现被自由地发布, 从而导致了这些协议被广泛地使用, 确立了其牢固的地位, 因而难以再用其他的协议来取代。现在这样的一些协议反而成了一种障碍。例如, 虚拟终端协议 TELNET 最初被设计用于每秒 10 个字符的机械电传打字终端, 它不能使用图形用户界面和鼠标。然而, 在 25 年之后, 它仍然在广泛使用。

总而言之, 尽管 OSI 模型(去掉会话层和表示层)存在很多问题, 但是, 事实证明它对于讨论计算机网络是非常有用的。但是, OSI 协议并没有流行起来。而 TCP/IP 却正好倒过来, 模型本身实际上并不存在, 但是协议却被广泛使用了。由于计算机科学家也喜欢自己做蛋糕来吃, 所以在本书中, 我们将使用一个修改之后的 OSI 模型, 但是注意力却集中在 TCP/IP 和相关的协议以及更新一些的协议, 如 802、SONET 和蓝牙等。实际上, 我们将使用如图 1.24 所示的混合模型作为本书的框架。

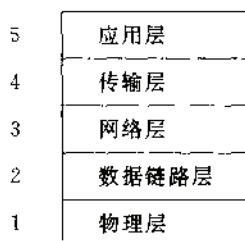


图 1.24 本书所使用的混合参考模型

## 1.5 网络实例

计算机网络的主题覆盖了许多不同种类的网络, 有大的, 有小的, 有知名的, 也有不知名的。它们有不同的目标、规模和技术。在接下去的几小节中, 我们将介绍一些实例, 以便读者对于计算机网络领域中的各种网络有一个认识。

我们将首先从 Internet 开始谈起。Internet 可能是最有名的网络, 我们将讨论它的历史, 它的发展历程, 以及它的技术。然后我们将介绍 ATM, 它常常用于大型(电话)网络的核心区域。从技术角度来看, 它与 Internet 有很大的区别, 两者形成了鲜明的对比。接下来我们再介绍以太网, 即最主要的局域网络。最后, 我们将看一下无线 LAN 的标准

IEEE 802.11。

### 1.5.1 Internet

Internet 并不是单个网络,而是大量不同网络的集合,这些不同的网络使用一组公共的协议,并提供一组公共的服务。Internet 不是一个普通的系统,也不是由任何一个人规划出来的,不受任何人控制。为了更好地理解 Internet,我们从它的发展初期开始谈起,并且看一看它是如何发展起来的,以及为什么发展起来了。如果你要了解 Internet 辉煌的历史,建议你看一看 John Naughton 的书(2000)。这是一本很难得的好书,不仅读起来有趣味性,而且为严谨的网络史学家提供了 20 页的参考引用书目。本节中下面有些材料就是以这本书为基础的。

当然,关于 Internet 及其协议有大量的技术书籍,例如,有关更多的信息,你可以参考 (Maufer, 1999)。

#### ARPANET

故事还得从 20 世纪 50 年代后期开始讲起,在冷战高峰期,美国国防部希望建立一个命令和控制网络,即使在核战争的情况下它也能够保存下来。在当时,所有的军事通信都使用公共电话网络,而电话网络则被认为是非常脆弱的。从图 1.25(a)我们可以看出这种观点的理由。图中黑色的点代表了电话交换局,每个电话交换局都连接了几千部电话。然后,这些交换局又连接到更高层次的交换局(长途局),从而形成了全国性的层次结构,整个结构中只有少量的冗余。这个系统的脆弱性在于,一旦几个关键的长途局遭到破坏,则整个系统有可能被分成多个孤岛。

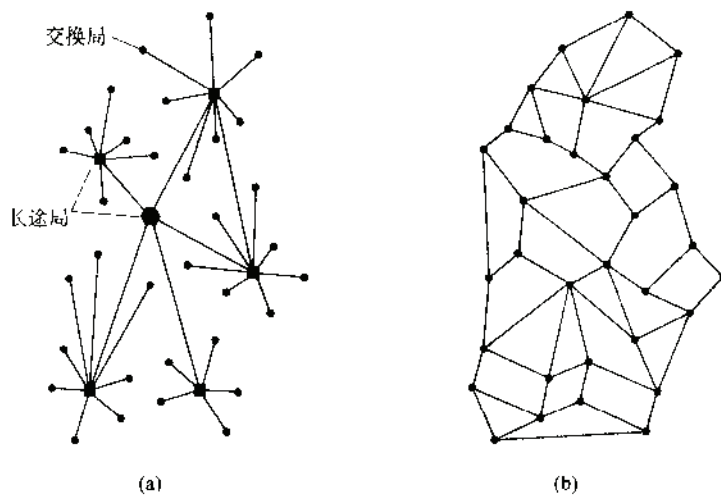


图 1.25

(a) 电话系统的结构; (b) Baran 提出的分布式交换系统

1960 年左右,国防部给了兰德公司一份合同,授权它寻找一个解决方案。兰德公司的一位雇员 Paul Baran 提出了一个高度分布式和容错的设计方案,如图 1.25(b) 所示。



由于任何两个交换局之间的路径都超过了模拟信号不失真传输的最长距离,所以 Baran 考虑在整个系统中使用数字的分组交换技术。Baran 给美国国防部写了几份报告来详细地阐述他的思想。五角大楼的官员接受了这种概念,并且请 AT&T 公司,然后是美国国家电话局来建立一个原型系统。AT&T 公司放弃了 Baran 的想法。这个世界上最大最富有的公司容不得让一个年轻人来告诉自己该如何建立一个电话系统。他们说 Baran 的网络是不可能建立起来的,于是 Baran 的想法被扼杀了。

几年以后,美国国防部还是没有有一个更好的命令和控制系统。为了更好地理解接下来发生的事情,我们必须回到 1957 年的 10 月份,前苏联发射了第一颗人造卫星 (Sputnik),美国跟前苏联在太空领域展开了激烈的竞争。当时艾森豪威尔总统试图发现谁在玩忽职守,他很惊讶地发现,陆军、海军和空军都在为五角大楼的研究经费预算而争吵不停。他的第一反应是建立一个专门的国防研究组织: ARPA, 即 **Advanced Research Projects Agency**。ARPA 没有科学家和实验室,实际上,除了一间办公室和少量的预算(按照五角大楼的标准)以外它什么也没有。它通过发放许可和签约合同的形式,让那些技术思想比较有前景的大学或者公司来完成它的工作。

在刚开始的几年中,ARPA 试图确定它的使命到底是什么,但是在 1967 年,ARPA 的负责人 Larry Roberts 将注意力转到了网络技术上。他联系了不同的专家,以便确定该做些什么。其中一个专家, Wesley Clark, 建议建立一个分组交换的子网,给每个主机都配一个路由器,如图 1.10 所示。

在经过了初始的怀疑和讨论之后,Roberts 接受了这种想法,并且于 1967 年下半年在 ACM SIGOPS 会议上(关于操作系统原理的研讨会,在 Tennessee 的 Gatlinburg 举行)出示了一份多少有点含糊不清的论文。让 Roberts 感到惊讶的是,在这次会议上的另外一篇文章也描述了一个类似的系统,该系统不仅有了设计,而且英国的国家物理实验室 (National Physical Laboratory, NPL) 在 Donald Davies 的指导下已经实现了该系统。NPL 系统不是一个国家级的系统(它只是将 NPL 园区中的几台计算机连接起来了),但是,它证实了分组交换的思想可以正确地工作。而且,它引用了当时已被弃之不用的 Baran 早期的研究工作。Roberts 从 Gatlinburg 回来之后决定建立一个网络,后来该网络称为 ARPANET。

ARPANET 的子网由一些小型机组成,这些小型机称为 IMP (Interface Message Processors, 接口消息处理器),它们通过 56kbps 的传输线连接起来。为了保证高度可靠性,每个 IMP 都将至少连接到两个其他的 IMP 上。该子网是一个数据报子网,所以,如果有一些线路或者 IMP 被毁坏了的话,通过替换的路径仍然可以自动地重新路由消息。

该网络的每个节点都包含一个 IMP 和一台主机,并且位于同一个房间中,通过一条很短的线连接起来。主机给 IMP 发送的消息最长可达 8063 位,然后 IMP 把这些消息分成最多 1008 位的分组,再独立地向目标转发这些分组。每一个分组必须在完整地到达一个节点之后才可以被转发,所以,该子网是第一个电子的、存储-转发类型的分组交换网络。

然后,ARPA 以招标的形式来建立该子网。有 12 家公司参与竞标。在评估了所有的候选公司之后,ARPA 选择了 BBN,这是麻省剑桥的一家咨询公司,然后 ARPA 与

BBN 签署了合同以建立该子网,并编写子网所需要的软件。BBN 选择了经过特殊改进的 Honeywell DDP-316 小型机(含有 12KB 16 位字的核心内存)作为 IMP。该 IMP 没有磁盘,因为移动部件被认为是不可靠的。通过租用电话公司的 56kbps 的线路将这些 IMP 相互连接起来。虽然,56kbps 现在已经成了那些支付不起 ADSL 或者电缆的青少年们才使用的线路,但是在当时这需要很多钱才能买得到。

软件分成两部分:子网和主机。子网软件包括“主机-IMP”连接的 IMP 部分、IMP-IMP 协议,以及一个专门为了提高可靠性而设计的从源 IMP 到目标 IMP 的协议。原始的 ARPANET 设计如图 1.26 所示。

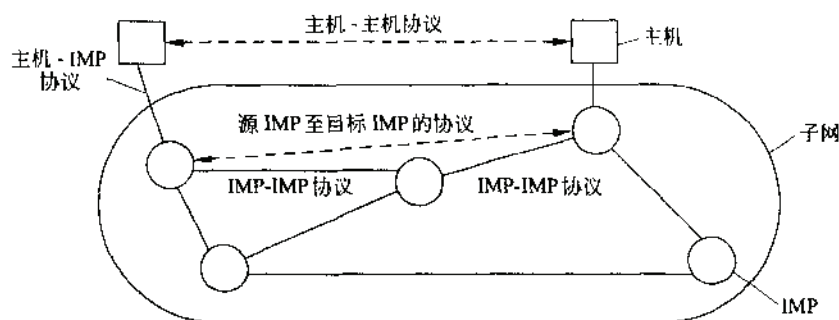


图 1.26 原始的 ARPANET 设计

在子网之外还需要软件,也就是说,主机端的主机-IMP 连接,主机-主机的协议,以及应用软件。很快地,BBN 意识到,当它能够在主机-IMP 的线路上接收到报文,并且在目标端能够把报文放到主机-IMP 线路上的时候,任务就完成了。

Roberts 有一个问题:主机也需要软件。为了解决这个问题,1969 年的夏季,他在犹他州的 Snowbird 召集了一次网络研究人员的会议,其中大多数是研究生。研究生们希望有一个网络专家向他们解释网络的主要设计方案,以及所需要的软件,然后交给每个人编写一部分软件的工作。当他们发现那里没有网络专家,也没有主要设计方案的时候,他们惊呆了。他们必须自己想办法来找到该做的事情。

不管怎么样,在 1969 年 12 月,一个包含有 4 个节点的实验网络可以运行了,这 4 个节点是:UCLA、UCSB、SRI 和犹他大学。之所以选择这 4 个点,是因为这些地方都有 ARPA 的许多合同,而且,它们都有许多不同类型并且完全不兼容的主机(仅仅是为了使这一项工作更加有趣)。随着更多的 IMP 被交付使用,并安装相应的软件,该网络快速增长起来,很快就扩展到了整个美国。图 1.27 显示了 ARPANET 在最初 3 年中是如何快速增长起来的。

除了帮助 ARPANET 成长起来以外,ARPA 还资助了关于卫星网络和移动分组无线网络应用的研究工作。在一次著名的演示中,一辆正在加州行驶的卡车通过分组无线网络给 SRI 发送报文,然后 SRI 将该报文通过 ARPANET 发送到了东海岸,再通过卫星网络发送到伦敦的大学学院。这样,卡车中的研究人员就可以一边在加州行驶,一边使用伦敦的计算机了。

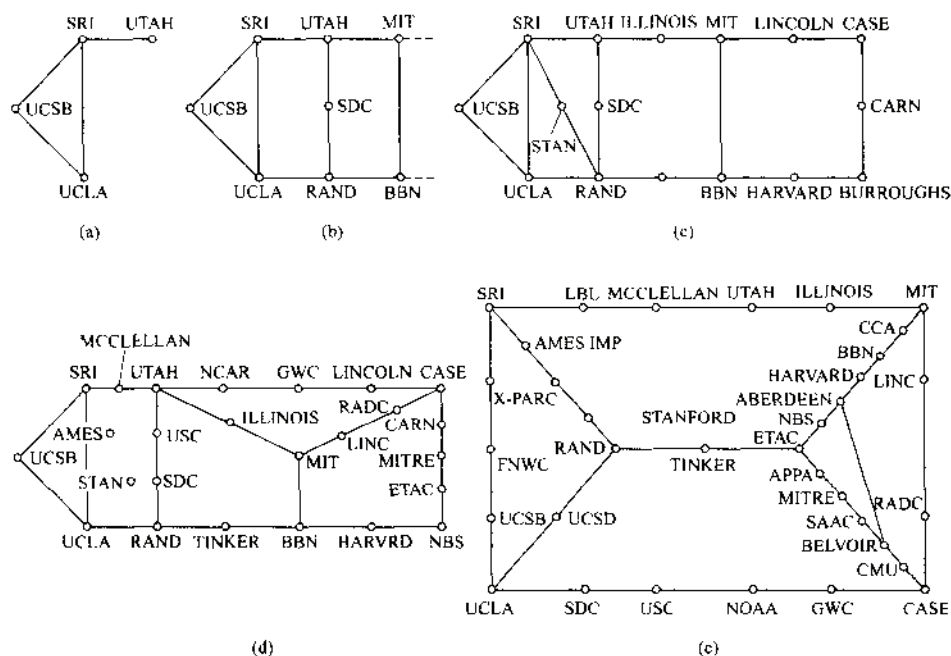


图 1.27 ARPANET 的增长情况

(a) 1969 年 12 月; (b) 1970 年 7 月; (c) 1971 年 3 月; (d) 1972 年 4 月; (e) 1972 年 9 月

这次实验也演示了现有的 ARPANET 不适合于跨越多个网络运行。这导致了更多有关协议的研究工作,最终发明了 TCP/IP 模型和协议(Cerf and Kahn, 1974)。TCP/IP 被专门设计用于处理网络互连的通信,随着越来越多的网络开始挂接到 ARPANET 上,网络互连变得越来越重要了。

为了鼓励采用这些新的协议,ARPA 给了 BBN 和加州大学 Berkeley 分校几个合同,以便将这些新的协议集成到 Berkeley UNIX 中。Berkeley 的研究人员开发了一个很方便的、专门用于连接网络的编程接口(套接字,socket),并编写了许多应用程序、工具以及管理程序,以便使得网络连接更加容易。

真可谓恰逢其时。许多大学刚刚获得了第二台或者第三台 VAX 计算机,这就需要将它们连接成一个 LAN,但是他们却没有这样的联网软件。4.2BSD 是连同 TCP/IP、socket 编程接口以及许多网络工具一起出来的,于是整个包都立即被采用了。而且,通过 TCP/IP,LAN 连接到 ARPANET 是非常容易的,确实许多 LAN 也这样做了。

在 20 世纪 80 年代,其他的一些网络,特别是 LAN 都连接到了 ARPANET 上。随着规模的增长,寻找主机所需要的开销越来越昂贵,所以 DNS(Domain Name System,域名系统)被建立起来,它将机器组织成域,并且将主机名字映射成 IP 地址。从那时开始,DNS 就成为一个通用的分布式数据库系统,它保存了各种各样与名字有关的信息。我们将在第 7 章详细地介绍 DNS。

## NSFNET

到了 20 世纪 70 年代后期,NSF(U. S. National Science Foundation,美国国家科学基金会)看到 ARPANET 对于大学研究工作产生了巨大的影响,ARPANET 使得不同国家的科学家们可以共享数据,可以在研究项目上协同工作。然而,对于任何一个大学,如果它要使用 ARPANET,那么它必须与美国国防部有一个研究合同,而这是许多大学所不具备的。NSF 决定设计一个 ARPANET 的后继网络,并且对所有大学的研究组都开放。为了使这件事情有一个实质性的开头,NSF 决定建立一个骨干网络,将它的 6 个超级计算机中心连接起来,这 6 个计算机中心分别位于 San Diego、Boulder、Champaign、Pittsburgh、Ithaca 和 Princeton。每台超级计算机都配备一台 LSI-11 微型计算机,称为 **fuzzball**(模糊球)的。这些 fuzzball 通过租用来的 56kbps 线路连接起来,形成了子网结构,所以,该网络的硬件技术与 ARPANET 相同。然而,软件技术不同:fuzzball 从一开始就使用 TCP/IP,这使得它成为第一个 TCP/IP 协议的广域网。

NSF 还资助了一些(最终大约 20 个)区域性网络,它们连接到骨干网上,允许数以千计的大学、研究实验室、图书馆和博物馆中的用户访问任何一台超级计算机,并且可以相互通信。该计算机网络,包括骨干网和这些区域网,合起来称为 **NSFNET**。它通过 Carnegie Mellon 机房内一条连接 IMP 和 fuzzball 的链路连接到 ARPANET。最初的 NSFNET 骨干网如图 1.28 所示。

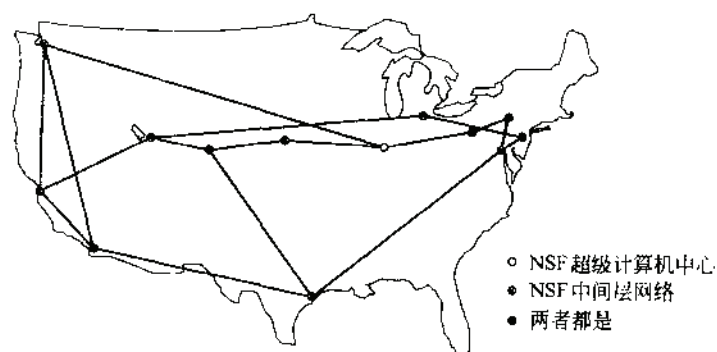


图 1.28 1988 年的 NSFNET 骨干网

NSFNET 很快获得了成功,并且从一开始就超负荷运转。NSF 立即启动它的下一代网络计划,并且给了以 Michigan 为基础的 MERIT 协会一份合同来运行下一代网络。他们从 MCI(已经与 WorldCom 合并)租用了 448kbps 的光纤信道来构建第二代骨干网,并且使用 IBM PC-RT 作为路由器。很快地,网络又超负荷了。到了 1990 年的时候,第二代骨干网升级到了 1.5Mbps。

随着网络的不断增长,NSF 意识到政府不可能永远不停地资助网络发展。而且,商业组织也想要加入网络,但是 NSF 的章程规定:禁止商业组织使用由 NSF 支付的网络。因此,NSF 鼓励 MERIT、MCI 和 IBM 组成一个非营利性的企业:ANS(**Advanced Networks and Services**),以作为向商业化道路上迈出的第一步。1990 年,ANS 接管了

NSFNET,并且将 1.5Mbps 链路升级到 45Mbps,构成了 ANSNET。ANSNET 运行了 5 年,然后被出售给 America Online(美国在线)。此时,许多公司都在提供商业性的 IP 服务了,很明显,政府应该退出网络服务的商业圈。

为了使传输更加容易,并且确保每一个区域网络都可以与其他所有的区域网络进行通信,NSF 又给了 4 个网络运行商一些合同,让他们都来建立网络接入点(Network Access Point,NAP)。这 4 个网络运行商是 PacBell(旧金山)、Ameritech(芝加哥)、MFS(华盛顿特区)和 Sprint(纽约市,为了 NAP,宾夕法尼亚和新泽西也算在了纽约市内)。任何一个网络运行商,如果要想给 NSF 区域网提供骨干网服务,则它必须连接到所有的 NAP。

这种安排意味着,从任何一个区域网发出的分组可以选择骨干网承运商,以便从它所在的 NAP 到达目标 NAP。因此,骨干网承运商不得不为了区域网的业务而在服务和价格上展开竞争,当然,这正是 NSF 的目标。最终的结果是,原来单个默认的骨干网被一个商业驱动的、有竞争性的基础设施所取代。许多人喜欢批评联邦政府总是不能有所革新,但是在网络这个领域中,正是美国国防部和 NSF 建立起了网络基础设施,它形成了 Internet 的基础,然后他们又把该设施交给工业界来运行。

在 20 世纪 90 年代,其他的许多国家和区域也建立起了国家研究网络,通常也采用了 ARPANET 和 NSFNET 的模式。这其中包括欧洲的 EuropaNET 和 EBONE,刚开始的时候它们使用 2Mbps 线路,后来升级到 34Mbps 线路。而且最终欧洲的网络基础设施也交给了工业界来运行。

### Internet 的使用

在 1983 年 1 月 1 日 TCP/IP 成为 ARPANET 上惟一的正式协议之后,连接到 ARPANET 的网络、机器和用户数量飞速地增长。当 NSFNET 和 ARPANET 互连以后,这种增长呈指数的趋势。许多区域网络也加入进来,并且也连接到了加拿大、欧洲和太平洋地区。

在 20 世纪 80 年代中期,人们开始把网络的集合看作一个互联网,后来又看作 Internet(因特网),官方并没有任何正式的引导,只不过在初期的时候为了 fuzzball 而有过一点简单的庆祝。

把 Internet 粘连在一起的是 TCP/IP 参考模型以及 TCP/IP 协议栈。TCP/IP 使得通用的服务成为可能,它作为标准所起的作用可以与 19 世纪的铁路,或者与所有电话公司采用的公共信令协议相媲美。

所谓“在 Internet 上”到底意味着什么呢?我们的定义是,如果一台机器运行了 TCP/IP 协议栈,有一个 IP 地址,并且可以向 Internet 上所有其他的机器发送 IP 分组,那么这台机器就是在 Internet 上。仅仅能够发送和接收电子邮件还不够,因为电子邮件可以通过网关到达 Internet 之外的许多网络。然而,在实践中还有这样的情形:成千上万的个人计算机可以通过调制解调器呼叫一个 ISP(Internet Service Provider,Internet 服务提供商),然后被分配一个临时的 IP 地址,再发送分组到其他的 Internet 主机。所以,我们可以这样认为,只要一台机器被连接到 Internet 服务提供商的路由器上了,那么它也



就上了 Internet。

传统上(指从 1970 到大约 1990 年),Internet 和它的前身网络有 4 种主要的应用:

(1) **电子邮件(E-mail)**。在 ARPANET 的早期,人们就可以编写、发送和接收电子邮件了,现在已经非常普及。许多人每天都会收到大量的消息,并且将它看作与外界交流的主要途径,而且远远超过了电话和缓慢的邮政信件。今天,几乎每一种计算机上都可以使用电子邮件程序。

(2) **新闻组(news)**。新闻组是一些专门的论坛,在这些论坛中,用户们往往有共同的兴趣,他们通过新闻组相互交换消息。现在有上千个新闻组,既有关于技术话题的,也有关于非技术话题的,包括计算机、科学、娱乐和政治。每个新闻组都有它自己的规则、风格和习惯,任何人都不应该违反它们。

(3) **远程登录**。通过 telnet、rlogin 或者 ssh 程序,在 Internet 上任何地方的用户都可以登录到任何一台其他的机器上,只要此用户拥有该机器上的一个账号即可。

(4) **文件传输**。通过 FTP 程序,用户可以将 Internet 上一台机器上的文件复制到另一台机器上。通过这种方式,用户可以访问到大量的文章、数据库和其他的信息。

20 世纪 90 年代早期,Internet 主要流行于学院、政府和工业界的研究人员之间。一个新的应用——万维网(World Wide Web,WWW)改变了这种状况,它将成千上万非学术性的新用户带到了网络上。这种应用形式是由 CERN 的物理学家 Tim Berners-Lee 发明的,它并没有改变任何底层的设施,但却使得网络设施更加容易使用了。再加上由国家超级计算机应用中心(在伊利诺斯州的 Urbana 城)的 Marc Andreessen 编写的 Mosaic 浏览器,万维网使得一个站点有可能安装大量的信息页面,内容可以包括文字、图片、声音,甚至视频片断,页面中还可以嵌入指向其他页面的链接。用户只要在这样的链接上点击一下,就可以快速地转换到由该链接所指向的页面上。例如,许多公司都有一个主页,该主页上包含一些条目分别指向其他的页面,比如产品信息、价目表、销售情况、技术支持、与雇员的通信、股东信息,等等。

在很短的时间内又出现了大量其他种类的页面,包括地图、股市行情表、图书馆卡片目录、电台录音节目,甚至还有指向那些版权已经过期的图书(比如马克·吐温、查里斯·狄更斯等的名著)的完整文本的页面。许多人还有了个人页面(即个人主页)。

在 20 世纪 90 年代这种迅猛增长很大程度上得归功于那些称为 ISP(Internet Service Providers,Internet 服务提供商)的公司。这些公司为家庭用户提供了“呼叫公司某一台计算机并连接到 Internet”的能力,因而这些家庭用户就可以访问电子邮件、万维网以及其他的 Internet 服务。在 90 年代后期,这些公司一年之内就可以注册上千万个新用户,因此完全改变了网络的应用特征:从学院和军事用途变成了公共的工具,非常类似于电话系统的发展情况。现在 Internet 用户数到底有多少并不清楚,但是全球至少有一亿用户,而且有可能很快会达到 10 亿用户以上。

### Internet 的体系结构

在这一部分,我们将简要地描述一下现在的 Internet 概况。由于许多电话公司(telco)和 ISP 兼并到一起了,所以它们的职责就没有分得那么清楚了,我们往往很难说得

清楚在做什么。因此,这部分的描述会比实际情况要简单一些。总体的框架结构如图 1.29 所示。现在我们依次介绍每一部分。

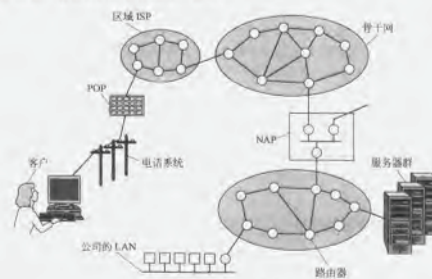


图 1.29 Internet 的概况

我们首先从家庭用户(客户)开始。假设我们的客户通过拨号电话呼叫他(或她)的 ISP,如图 1.29 所示。调制解调器是指 PC 内部的一块卡,它将计算机产生的数字信号转换为模拟信号,然后通过电话系统来传递这些模拟信号。这些信号被传输到 ISP 的 POP (Point of Presence, 汇接点)处,到了这里之后,模拟信号从电话系统中删除,而相应的数字信号被注入到 ISP 的区域网络中。从这时开始,系统将是完全数字化的,并且通过分组交换来转发数据。如果 ISP 是本地的电话公司,那么 POP 可能位于电话交换局所在的地方,这也是客户终端的电话线聚集的地方。如果 ISP 不是本地的电话公司,那么 POP 有可能是下游的一些交换局。

ISP 的区域网络是由一些相互连接的路由器组成的,它们位于该 ISP 提供服务的各个地方。如果一个分组的目的地是由该 ISP 直接服务的一台主机,则此分组将被直接递交给该主机;否则的话,分组将被转送给 ISP 的骨干网运行商。

在该食物链的最顶端是大型骨干网运行商,如 AT&T 和 Sprint 公司。它们经营着大型的骨干网,有上千台路由器,通过高带宽的光纤连接起来。大型公司,以及需要运行服务器群(server farm,指允许每秒钟访问几千个 Web 页面的一群机器)才能提供的服务往往直接连接到骨干网上。骨干网运行商通常鼓励这种直接连接的方式,它们出租一定的空间(称为承运商客栈(carrier hotel),基本上是指与路由器在同一个房间中的设备架),以便使得在服务器群和骨干网之间存在短距离的快速连接。

如果一个传送到骨干网的分组的目的地是一个 ISP,或者是该骨干网所服务的一家公司的话,那么该分组将送到最近的路由器,并且在那里继续路由。然而,由于世界上存在各种不同规模的骨干网,所以,一个分组可能必须要经过另一个竞争的骨干网。为了允许分

组可以跨越不同的骨干网,所有大型的骨干网都要连接到 NAP 上,正如前面已经讨论过了。通常,NAP 是一个充满了路由器的房间,每一个骨干网至少有一个路由器。这个房间中的 LAN 将所有的路由器连接起来,所以,这里的分组可以从任何一个骨干网转发到任何其他骨干网。除了在 NAP 处骨干网相互连接起来以外,大一点的骨干网的路由器之间也会有一些直接连接,这项技术称为私有对等连接技术(private peering)。关于 Internet 有许多看似矛盾的怪事,其中之一是,两家公开竞争顾客的 ISP 往往会私下里合作实行私有对等连接(Metz, 2001)。

现在我们结束有关 Internet 的粗略介绍。在后面的章节中我们还将有大量的篇幅来介绍 Internet 的各个部件,以及这些部件的设计、算法和协议。同时值得一提的是,有些公司已经将内部所有的网络相互连接起来了,通常所用的技术与 Internet 相同。这些企业内部的互联网(intranet)往往只能在公司内部才可以访问,除此以外,它们的工作方式与 Internet 完全相同。

### 1.5.2 面向连接的网络: X.25、帧中继和 ATM

在网络技术发展的初期,曾经有过一场激烈的争论,有的人支持无连接(即数据报)子网,有的人支持面向连接的子网。无连接子网的支持者主要来自子 ARPANET/Internet 社团。前面提到过,美国国防部在资助和构建 ARPANET 的时候,最初的期望是,即使核战争毁掉了很多路由器和传输线,网络也仍然可以继续工作。因此,在优先级列表中,容错性具有很高的优先级,而对客户进行记账则是次要的。这种思路直接导致了无连接的设计方案,每个分组都将独立地进行路由,分组彼此之间完全独立。其结果是,如果有一些路由器在一个会话过程中崩溃了,那么,只要该系统还能够重新动态地配置自己,使得后续的分组的仍然能够找到通向目标的路由路径,且即使它们与以前的分组所走的路径截然不同,该会话也不会受到影响。

面向连接的阵营来自于电话公司。在电话系统中,呼叫方拨对方(被呼叫方)的号码,在通话或者发送数据之前必须等待建立连接。在连接初始化阶段建立起一条经过电话系统的路由路径,在这次呼叫结束之前,这条路径会一直被维护着。所有的语音或者分组都将沿着这同一条路径进行传输。如果该路径上有一条线路或者一个交换机不工作了,那么该呼叫将会中断。这正是美国国防部所不希望发生的。

那么,为什么电话公司会喜欢这种模式呢? 有两个理由:

- (1) 服务质量;
- (2) 记账。

通过预先建立一个连接,子网就可以预留资源,比如缓冲区空间或者路由器的 CPU 能力。如果呼叫方企图建立一次呼叫,但是中间路径上又没有足够的资源可供使用,那么这次呼叫就会被拒绝,呼叫方将听到一种忙信号。按照这种方式,一旦一个连接已经建立起来,那么该连接就会得到很好的服务。对于无连接网络的情形,如果在同一时刻有太多的分组到达同一个路由器,那么该路由器将被阻塞,它可能会丢失分组。发送方最终会注意到这种现象,并重新发送这些分组,但是,服务质量将会很不稳定,这不适合于音频或者视频信息,除非该网络的负载很轻。毋庸多说,提供足够的音频质量是电话公司非常关心

的事情,因此他们倾向于有连接的网络。

电话公司喜欢面向连接的服务的第二个理由是,他们习惯于按照连接时间进行计费。当你进行长途呼叫(或者是非北美地区的本地呼叫)的时候,电话公司对你按每分钟进行收费。在考虑网络模型的时候,他们自然也会倾向于采用一种易于按分钟进行计费的模型。如果你在发送数据之前必须先要建立一个连接,那么这就是计费时钟的开始时间。如果无连接的话,他们就不可能这样收费。

有意思的是,维护计费记录的开销也非常大。如果一家电话公司采用了一种简单的月租计费方式,客户可以无限制地拨打电话,电话公司不用计费,或者不需要保持记录,那么,尽管这种策略会导致更多的电话呼叫,但是该公司可能反而会节省下一大笔钱。然而,政策、规章制度等种种因素都不会倾向于这种做法。有趣的是,在其他行业中,简单的月租服务也存在。例如,有线电视是按每个月进行收费的,而不管你观看了多少节目。它可能已经被设计成“按一次观看收费(pay per view)”作为其基本的概念,但是现在还没有做到这一点,部分原因是因为计费的开销太昂贵(既然大多数电视的质量都已经没有问题了,那么这个令人难堪的因素也就不应该再被打折扣了)。另外,许多主题乐园(游乐园之中)是按天来收费的,允许游客在一天之内无限制地游玩,相比之下,巡回演出的游艺团则是按次数来收费的。

这就是说,电话工业界设计出来的所有网络都具有面向连接的子网,这是不足为奇的。但是令人惊奇的是,Internet 为了对音频和视频信息提供更好的服务质量(这是第 5 章的一个主题),也在这个方向上漂浮不定。现在我们先来看一些面向连接的网络。

#### X.25 和帧中继

第一个面向连接的网络实例是 X.25,它也是第一个公共的数据网络。X.25 是 20 世纪 70 年代部署起来的,当时电话服务在所有的地方都是垄断的,每个国家的电话公司都期望有一个属于自己国家的数据网络。为了使用 X.25,一台计算机首先要与远程的计算机建立一个连接,也就是说,要发出一个电话呼叫。每个连接被分配一个连接号,在数据传输分组中需要使用连接号(使用连接号的原因是,同一时刻可能会打开多个连接)。数据分组非常简单,包含一个 3 字节的头部和最长可达 128 字节的数据部分。头部是由一个 12 位的连接号、分组序列号、一个确认号,以及其他一些位组成的。X.25 网络运行了大约 10 年左右,既有成功的一面,也有失败的一面。

在 20 世纪 80 年代,X.25 网络被一种新的、称为帧中继(frame relay)的网络所取代。帧中继的本质是,它是一个无错误控制的、无流控制的、面向连接的网络。因为它是面向连接的,所以分组会按照发送的顺序被递交(如果它们都被递交的话)。这种按序递交、无错误控制、无流控制的特性使得帧中继非常类似于一个广域的 LAN。它最重要的应用是,能将公司的多个办公区域的 LAN 相互连接起来。帧中继获得了一定程度的成功,至今仍然还有一些地方在使用。

#### 异步传输模式

还有另一种,也是最重要的面向连接的网络是异步传输模式(Asynchronous Transfer

Mode, ATM)。它的名字有点怪异,因为在电话系统中,大多数传输是同步的(严格地依赖于一个时钟),而 ATM 则不是。

ATM 是在 20 世纪 90 年代早期设计的,并且是在广泛的宣传下发展起来的(Ginsburg, 1996; Goralski, 1995; Ibe, 1997; Kim et al., 1994; and Stallings, 2000)。ATM 期望解决所有的网络和通信问题,它的设想是,将语音、数据、有线电视、电报、传真的,由绳子串接起来的铁罐头、手鼓、烟信号等所有的东西都合并到一个集成系统中,该集成系统可以为任何人做任何事情。然而,这种局面并没有发生。在很大程度上,这里的问题与我们前面讨论 OSI 时所描述的类似,即糟糕的时机、糟糕的技术、糟糕的实现以及糟糕的策略。在第一回合刚刚打败了电话公司,Internet 社团中的许多人往往从 Internet 或者电话公司两者对立的角度来看待 ATM,结论自然不言而喻。但实际上并不是这样,在这一回合较量中,即使是最顽固的数据报狂热分子也知道,Internet 的服务质量还是一个有待于进一步改进和解决的问题。长话短说,ATM 比 OSI 要成功得多,它已经在电话系统中被广泛使用了,通常用于传输 IP 分组。因为它现在主要被承运商用于内部传输,所以用户通常感觉不到它的存在,但它肯定是在发挥作用的,并且工作得很好。

#### ATM 虚电路

由于 ATM 网络是面向连接的,所以,在发送数据之前首先要发送一个分组以便建立连接。当这个初始分组经过子网的时候,该路径上所有的路由器都在它们的内部表中建立一个表项,用来标明该连接的存在,并且为它预留必要的资源。这里的连接通常称为虚电路(virtual circuit),类似于电话系统中使用的物理电路。大多数 ATM 网络也支持永久虚电路(permanent virtual circuit),这是指两台(远程)主机之间的永久连接。永久虚电路非常类似于电话领域中租用的线路。每一个连接,无论是临时的,还是永久的,都有一个惟一的连接标识符。虚电路的概念如图 1.30 所示。

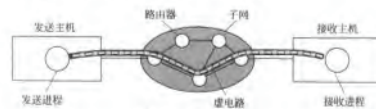


图 1.30 虚电路示意图

一旦建立了一个连接,两边就可以开始传输数据了。ATM 的基本思想是,所有的信息都放在固定长度的小分组中进行传输,这样的分组也称为信元(cell)。每个信元共 53 字节长,其中包含 5 字节的头和 48 字节的有效载荷,如图 1.31 所示。头部包含了连接标识符,所以发送主机和接收主机以及所有的中间路由器都能够分辨出哪个信元属于哪个连接。此信息也使得每个路由器知道该如何转发每个进入的信元。信元路由是由硬件来完成的,所以速度很快。实际上,之所以采用固定长度的信元,最主要的原因是,容易设计



出由硬件路由器来处理短的、固定长度的信元。变长 IP 分组的路由必须要通过软件来完成,所以这是一个相对比较慢的过程。ATM 的另一个优势是,可以设计硬件以便将一个进入的信元复制到多条输出线路上,这是在处理电视节目的时候所必须具备的一种特性,因为电视节目要广播到许多个接收者处。最后,小的信元也不会阻塞线路很长时间,在这种情况下,保证服务质量要容易得多。



图 1.31 ATM 信元示意图

所有的信元都会沿着同样的路由路径到达目标处。发送出去的信元并不一定保证被递交,但是它们的顺序保证不会变化。如果信元 1 和信元 2 按照先后顺序依次被发送出去,那么,如果它们都到达的话,则它们也会按同样的顺序到达,信元 2 决不会在信元 1 的前面到达。但是,它们中的任何一个都有可能在路上丢失,甚至两个都丢失也是有可能的。丢失的信元是否需要恢复,这取决于高层上的协议。请注意,虽然这种保证并不完美,但是它比 Internet 提供的保证要强得多,因为 Internet 中的分组不仅可能会丢失,而且可能会按错序被递交。而 ATM 保证不会按错序递交信元。

ATM 网络的组织形式与传统的 WAN 类似,也有线路和交换机(路由器)。ATM 最常见的速率是 155Mbps 和 622Mbps,但是,更高的速率也是支持的。之所以选择 155Mbps,是因为这是传输高清晰度电视所需要的大致速率。准确的速率是 155.52Mbps,这是为了与 AT&T 的 SONET 传输系统相兼容。我们将在第 2 章学习 SONET。选择 622Mbps 的原因是允许 4 条 155Mbps 的信道在上面传输。

### ATM 参考模型

ATM 有它自己的参考模型,既不同于 OSI 模型,也不同于 TCP/IP 模型。该模型如图 1.32 所示。它包括三层:物理层、ATM 层和 ATM 适配层,另外,用户还可以在顶上放置任何东西。

物理层处理的是物理介质问题,即电压、位时序和各种各样其他的问题。ATM 并没有规定专门的规则集,而是说 ATM 信元可以自己在导线或者光纤上传送,也可以被封装在其他传输系统的净荷中。换句话说,ATM 被设计成与传输介质无关。

ATM 层处理信元和信元传输。它定义了信元的结构,并指出了头部各个域的含义。它也处理虚电路的建立和释放。拥塞控制也在这里解决。

因为大多数应用程序并不愿意直接与信元打交道(尽管有的应用程序可能希望这样),所以在 ATM 层之上又定义了一层,以便允许用户发送大于一个信元的分组。ATM 接口将这些分组分成小的片断,然后独立地传输这些信元,再在另一端将它们重新组装起来。这一层是 AAL(ATM Adaptation Layer, ATM 适配层)。

与以前的二维参考模型不同,ATM 模型是三维的,如图 1.32 所示。用户平面(user plane)处理数据传输、流控制、错误纠正,以及其他的用户功能。而控制平面(control plane)关注的是连接管理。层和平面管理功能与资源管理和层之间的协调有关系。

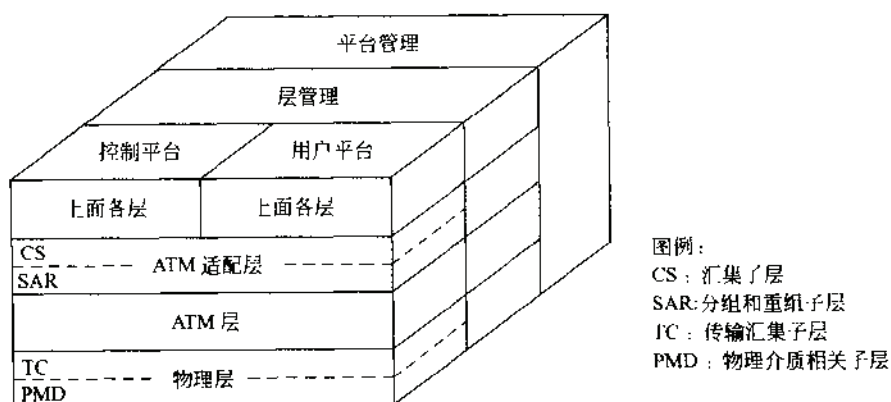


图 1.32 ATM 参考模型

物理层和 AAL 层都划分成两个子层，一个在下面完成实际的工作，另一个将汇聚子层在其上面，向上层提供适当的接口。ATM 模型中的各层和子层的功能如图 1.33 所示。

OSI 层	ATM 层	ATM 子层	功 能
3/4	AAL	CS	提供标准的接口(汇聚)
		SAR	分段和重组
2/3	ATM		流控制 信元头的生成和提取 虚电路/路径管理 信元多路复用/解复用
2	物理	TC	信元速率解耦合 头部校验和的生成和检验 信元生成 信元打包和拆包 生成帧
1		PMD	位时序 物理网络访问

图 1.33 ATM 层和子层及其功能

**PMD(Physical Medium Dependent, 物理介质相关)**子层是与实际电缆的接口。它传送位“0”和位“1”，并处理位时序。对于不同的传输载体和电缆，这一层是不同的。

物理层的另一个子层是 **TC(Transmission Convergence, 传输汇聚)**子层。当信元被传输的时候，TC 层将信元以位串的形式发送给 PMD 层。这样做并不困难。在另一端，TC 子层接收到一个来自 PMD 子层的纯粹的位输入流，它的任务是将这个位流转换成一个信元流，以便提供给 ATM 层。它需要在这个位流中辨别出信元的开始和结束，以及所有相关的事宜。在 ATM 模型中，这个功能是在物理层上实现的。而在 OSI 模型以及很多其他的网络中，这种帧处理工作(也就是说，将一个原始的位流转变成一个帧序列或者信

元序列)是数据链路层的任务。

正如我们前面曾经提到过的,ATM 层管理信元,包括它们的生成和传输。ATM 很多有趣的方面都在此处。它是 OSI 数据链路层和网络层的混合体,它没有再被分成子层。

AAL 层分成 SAR(Segmentation And Reassembly,分段和重组)子层和 CS(Covergence Sublayer,聚集子层)子层。在传输方,SAR 子层把分组分割成信元,然后在目标方又把它们组装回去。上面的子层(即 CS 子层)使得 ATM 系统有可能为不同的应用提供不同种类的服务(比如文件传输和视频点播这两种应用对于错误处理、实时性等有不同需求)。

现在看来,ATM 的前景并不被看好,所以我们在本书中将不再进一步讨论它。然而,由于它已经有了很多实际应用的基础,所以至少在几年之内它仍然存在并发挥作用。有关 ATM 更多的信息,请参看 Dobrowshi and Grise, 2001; and Gadechi and Heckart, 1997。

### 1.5.3 以太网

Internet 和 ATM 都是为广域网络而设计的。然而,许多公司、大学和其他的组织也有大量的计算机需要连接起来。这种需求使得局域网应运而生。在这一小节中,我们将讨论一种最为流行的 LAN: 以太网(Ethernet)。

故事还得从 20 世纪 70 年代初期“纯朴”的夏威夷岛讲起。这里所谓的纯朴,你可以理解为“还没有一个正经可以工作的电话系统”。由于整天不会受到电话的干扰,所以在这里度假会非常愉快,但是,对于夏威夷大学的研究人员 Norman Abramson 和他的同事们来说,这样的生活并不愉快,他们当时正在试图将远处岛屿中的用户连接到檀香山的主计算机上。自己在太平洋底下铺设电缆显然是不现实的,所以他们需要寻找其他的解决方案。

他们找到的一个方案是短距离的无线电波。每个用户终端都配置一个小的无线电接收装置,它有两个频率:上行(去往中心计算机)、下行(来自中心计算机)。当用户希望与主计算机联系的时候,它只要传送一个分组,该分组包含了上行信道中的数据。如果这时候没有其他人在传输数据,那么该分组将会被顺利地发送出去,并且在下行信道上得到确认。如果上行信道发生了冲突,那么终端设备就会察觉到,因为它没有收到确认信息,于是它可以再试着发送。由于在下行信道上只有一个发送者(即中心计算机),所以该信道上永远不会发生碰撞。该系统称为 ALOHANET,它在低流量的情况下工作得很好,但是当上行流量很大的时候,它几乎无法正常工作。

差不多在同一时期,一个名叫 Bob Metcalfe 的学生在 M. I. T. 获得了学士学位以后,到哈佛大学攻读博士学位。在学习期间,他了解到了 Abramson 的工作。他对此非常感兴趣,因此从哈佛毕业之后,他决定先去夏威夷跟 Abramson 工作一个夏天,然后再到 Xerox PARC(Palo Alto Research Center)正式开始工作。当他到了 PARC 的时候,他看到那里的研究人员已经设计并制造出了后来被称为个人计算机的机器。但是,这些机器都是孤立的。他利用从 Abramson 的工作中获得的知识,跟同事 David Boggs 一起设计并实现了第一个局域网(Metcalfe and Boggs, 1976)。

他们将该系统称为以太网(Ethernet),名字来源于光学中的以太(luminiferous ether),过去曾经有这样的看法,电磁辐射经过以太可以传播。(当19世纪英国物理学家James Clerk Maxwell发现电磁辐射可以用一个波动方程来描述的时候,科学家们假设太空中充满了某一种像空气一样的介质,所以电磁辐射才可以传播。直到1887年著名的Michelson-Morley实验之后,物理学家们才发现电磁辐射在真空中也能传播。)

这里的传输介质并不是真空,而是一根粗的同轴电缆(即以太),可以达到2.5km长(每500米需要一个中继器)。一共可以有256台机器连接到局域网系统中,每台机器通过一个收发器连接到电缆上。如果一根电缆上有多台机器并行地接入进来,则该电缆称为多支路电缆(multidrop cable)。系统运行在2.94Mbps的速率上。它的结构框架如图1.34所示。相比ALOHANET,以太网有一个很大的进步是:计算机在传输数据之前,先监听电缆上的信号,看是否有其他的计算机也在传输。如果有的话,则该计算机先等着,直到该电缆上当前的传输工作完成。这样做可以避免干扰现有的传输任务,从而获得比较高的传输效率。ALOHANET并不是这样工作的,因为它来讲,一个岛屿上的终端不可能感知到远处岛屿上的终端是否在传输数据。有了一条简单的电缆之后,这个问题就不存在了。

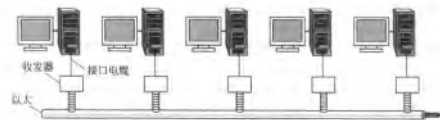


图 1.34 最初的以太网体系结构

尽管计算机在传输之前要先监听电缆上的信号,但是这里仍然存在一个问题:如果两台或者多台计算机都在等待当前的传输任务完成,然后同时开始传输数据,那该怎么办呢?解决的办法是,让每台计算机在它自己的传输过程中也进行监听,如果检测到有干扰,则阻塞电缆以警告所有的发送者。然后退回来等待一段随机时间之后再开始重试。如果第二次冲突又发生了,则随机的等待时间翻一倍,如此进行下去,这样可以使竞争的传输任务分离开,每个任务都有机会先执行。

Xerox的以太网大获成功,所以,1978年DEC、Intel和Xerox拟订了一个针对10Mbps以太网的标准,称为DIX标准。经过两次很小的修改之后,DIX标准于1983年变成IEEE 802.3标准。

Xerox是非常不幸的,它在历史上有过很多开创性的发明(比如个人计算机),但是它未能成功地将这些发明商业化,它的故事称为“Fumbling the Future(探索未来)”(Smith and Alexander, 1988)。Xerox除了将以太网标准化以外,对其他方面的工作并没有多大的兴趣,这时候,Metcalle组建了他自己的公司(3COM),专门销售PC机的以太网适配器。至今3COM已经卖出了上亿块适配器卡了。

以太网在标准化之后继续在发展,而且至今也还在发展。100Mbps,1000Mbps的以太网

以太网版本,甚至更高的速率,也相继出来了。电缆技术有了改进,交换技术和其他的特性也加入进来了。我们将在第4章详细地讨论以太网。

到这里,值得提一下的是,以太网(IEEE 802.3)并不是惟一的 LAN 标准。802 委员会还标准化了令牌总线(802.4)和令牌环(802.5)。为什么需要这三种多少互不兼容的标准呢?这里几乎没有技术的因素,而全部是政策的因素。在标准化的时候,General Motors(通用汽车公司)正在推动的一个 LAN 的拓扑结构与以太网一样(一条直线电缆),但是计算机的传输顺序是通过先发送一个称为令牌(token,在计算机之间传送)的小分组来决定的。只有当一台计算机拥有了令牌的时候,它才可以发送数据,因此避免了传输碰撞。General Motors 宣称,这种方案对于制造汽车是非常关键的,并且不准备放弃这种观点。尽管这样,802.4 基本上还是从人们的视野中逐渐消失了。

类似地,IBM 也有它自己的局域网体系结构:属于它私有的令牌环。令牌将沿着一个环进行传递,拥有令牌的计算机可以传输数据,然后把令牌送回到环上。这种方案已经被标准化为 802.5;它不同于 802.4,至今在 IBM 的一些站点上仍然在使用,但是在 IBM 以外几乎不再使用了。然而,802.5 千兆位版本的标准化工作仍在进行(802.5v),但看起来不太可能赶得上以太网。简单来说,在以太网、令牌总线和令牌环之间曾经有过一场竞争,后来以太网赢了,很大程度上是因为,以太网是第一个 LAN 标准,而且它的两个挑战者并不那么强大。

#### 1.5.4 无线 LAN: 802.11

几乎是在笔记本计算机刚出来的时候,许多人就梦想着一边在办公室里走动,一边还可以让他们的笔记本计算机连接到 Internet 上。因此,有许多研究组很早就开始为这个目标而努力了。最为实际的一条途径是,在办公室和笔记本计算机上安装上短距离的无线发射器和接收器,从而允许它们之间进行通信。这项工作很快导致了无线 LAN 的诞生,并且有一些公司还将产品推广到了市场上。

问题在于,这些无线 LAN 没有两个是相互兼容的。这种标准分叉现象意味着,如果一台计算机安装了一个 X 品牌的无线发射设备,而房间里安装了 Y 品牌的基站,则这两者相互之间不能协同工作。最后,工业界认为,建立无线 LAN 的标准可能是解决这个问题的好办法,所以,原来制订有线 LAN 标准的 IEEE 委员会承担了拟订无线 LAN 标准的任务。该委员会制订出来的标准被命名为 802.11。按照通俗的话来说,它被称为 WiFi。这是一个很重要的标准,值得引起人们的注意,所以我们将使用它的正式名称 802.11。

标准提案中介绍了两种工作模式:

- (1) 有基站的模式;
- (2) 无基站的模式。

在第一种情况下,所有的通信都经过基站,按照 802.11 的术语,基站称为访问点(access point)。在第二种情况下,计算机相互之间直接发送数据,这种模式有时候也称为 ad hoc 网络(ad hoc networking)。一个典型的例子是,两个或者多个人坐在一个房间中,房间里并没有安装无线 LAN,所以他们只好让他们的计算机相互之间直接通信。这两种



模式如图 1.35 所示。



图 1.35  
(a) 带基站的无线网络; (b) Ad-hoc 网络

第一个决定是最容易的：怎么命名这个标准。所有其他的 LAN 标准都有诸如 802.1、802.2、802.3，直到 802.10 这样的名称，所以无线 LAN 的标准很自然地命名为 802.11。剩下的就没那么容易了。

特别是，有些困难是必须要解决的，比如：找到一个合适的工作频段，而且最好该频段是全球范围内可以使用的；必须要考虑到无线电信号只在有限的范围内有效；确保维护用户的隐私；需要考虑电池的寿命是很有限的；担心人的安全（无线电波会致癌吗？）；充分理解计算机移动能力所蕴含的意义；最后，要建立一个有足够带宽的系统，在经济上也必须是切实可行的。

在标准化工作刚开始的时候（20 世纪 90 年代中期），以太网已经在局域网领域中占了支配地位，所以 IEEE 标准化委员会决定让 802.11 与以太网在数据链路层以上保持兼容。特别是，通过无线 LAN 发送 IP 分组的方式，与一台有线计算机通过以太网发送 IP 分组的方式应该是一样的。但是，在物理层和数据链路层，无线 LAN 与以太网有一些本质的区别，该标准必须要处理这些不同之处。

首先，以太网上的计算机在传输数据之前总是在监听着以太（即电缆）的状况。只有当以太空闲的时候，计算机才开始传输数据。对于无线 LAN 的情形，这种思路不会工作得很好。为了看清楚其中的缘由，请参考图 1.36。假设计算机 A 要向计算机 B 传递数据，但是 A 的发射器的无线电范围太短，到达不了计算机 C。如果 C 想要给 B 传递数据，它在开始传递之前先监听以太，但是，它没有听到任何东西，这并不意味着它的传输就会成功。802.11 标准必须要解决这个问题。

第二个必须要解决的问题是，无线信号在碰到固体对象的时候会有反射现象，所以同样的无线信号可能会收到多次（沿着不同的路径）。这种干扰导致了一种称为“多径衰落（multipath fading）”的现象。

第三个问题是，大量的软件并不知道这种移动性。例如，许多字处理软件都有一个打印机列表，用户可以从中选择某一个打印机来打印文件。当该字处理软件运行的时候，如果所在的计算机被移动到了一个新的环境中，那么内置的打印机列表就不再有效了。

第四个问题是，如果一个笔记本电脑原来正在使用一个吊装的基站，然后被移到了另一个基站的范围内，则需要有一种移交的机制。虽然这个问题在蜂窝电话系统中也存在。

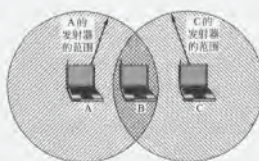


图 1.36 单个无线发射器的范围可能覆盖不了整个系统

在,但是在以太网中并不存在,所以以太网不需要考虑这个问题。特别是,你可以想象一个包含了多个蜂窝单元的网络,每个单元都有它自己的基站,通过基站与以太网连接起来,如图 1.37 所示。从外界来看,整个系统看起来就好像是单个以太网。802.11 系统与外界之间的连接称为一个入口(portal)。



图 1.37 多个蜂窝单元的 802.11 网络

经过一段时间的工作以后,标准化委员会于 1997 年拿出了一个标准,其中就考虑了以上这些问题,以及其他一些问题。它所描述的无线 LAN 运行在 1Mbps 或者 2Mbps 速率上。很快地,人们抱怨速度太慢了,所以委员会又开始制订更高速率的标准。这时在委员会内部又有了分裂,从而导致了两个新的标准(1999 年)。802.11a 标准使用了一个更宽的频段,速率可以达到 54Mbps。802.11b 使用的频段与 802.11 一样,但是使用了不同的调制技术,从而速率达到了 11Mbps。有的人把 802.11b 看得很重要,因为 11Mbps 超过了最初的有线以太网的速率。有可能原来的 1Mbps 802.11 很快就会淘汰掉,但是哪个新标准将会胜出还出现还不明朗。

为了使事情更加复杂化,802 委员会又拿出了另一个变种版本 802.11g,它使用了 802.11a 的调制技术,但是采用 802.11b 的频段。我们将在第 4 章再详细地讨论 802.11。

毋庸置疑,802.11 将会在计算机和 Internet 接入领域引发一场革命。机场,火车站,旅馆,购物商场以及大学都在安装无线 LAN。甚至高档的咖啡店也在安装 802.11,这样顾客在品尝饮料的同时还可以在 Web 上冲浪。可以这样说,802.11 对于 Internet 所做的事情,实际上也相当于笔记本电脑对于计算技术所起到的作用一样:让它移动起来。

## 1.6 网络标准化

现在已经有了许多网络生产商和供应商,他们都有自己的做事思路和方法。如果没有协调的话,事情就会变得一团糟,用户也会无所适从。惟一摆脱这种局面的办法是让大家遵守一些网络标准。

标准不仅使不同的计算机可以相互进行通信,而且也扩大了产品的市场,只要这些产品遵守相应的标准即可。大的市场导致了大的生产、制造业的规模经济、VLSI 实现,以及其他一些好处,比如更低的价格、用户接受程度的提高。在这一节中,我们将快速地看一下这些重要的,但是鲜为人知的国际标准化方面的工作。

所有的标准可以分为两大类:事实标准,以及法定标准。**事实(De facto**, 拉丁语“from the fact(从事实而来的)”)的标准是指那些已经发生了,但是并没有任何正式计划的标准。IBM PC 及其后继产品是小型办公和家庭计算机的事实标准,因为很多生产商都选择了仿制 IBM 的机器。类似地,UNIX 是大学计算机科学系中操作系统的事实标准。

相反,法定(**De jure**, 拉丁语“by law[依据法律]”)的标准是指由某个权威的标准化组织采纳的、正式的、合法的标准。国际性的标准化权威组织通常可以分成两类:国家政府之间通过条约建立起来的标准化组织,以及自愿的、非条约的组织。在计算机网络标准的领域中,有各种类型的一些组织,下面分别进行讨论。

### 1.6.1 电信领域中最有影响的组织

各个国家的电话公司的法律地位有很大的差异。其中一个极端是美国,它有 1500 个独立的、私有的电话公司。AT&T 公司在 1984 年被分解以前,是世界上最大的公司,几乎完全统治了整个电信业。它为全美 80%左右的电话提供服务,遍布美国一半的国土面积,所有其他的公司合起来为剩下的顾客(大部分在农村)提供服务。自从被分解以后,AT&T 继续提供长途电话服务,但现在是和和其他公司竞争。从 AT&T 分裂出来的 7 个区域性的贝尔运行公司以及很多个独立公司提供本地和蜂窝电话服务。由于频繁的兼并以及其他的一些变化,整个行业总是处于动荡变化的状态之中。

在美国,为公众提供通信服务的公司称为**公共承运商(common carrier)**。它们提供的服务以及价格是由一份称为**价目表(tariff)**的文档来规定的。在这份价目表中,其中州际的或者国际的流量部分是由联邦电信委员会批准的,而州内的流量部分则由州公共事业委员会批准。

另一个极端是指这样的国家:所有的通信都由国家政府完全垄断,其中包括邮件、电报、电话,常常还包括电台和电视。世界上大多数的国家都可归到这一类中。在有些情况下,电信权威是一个国家公司;在其他一些情况下,电信权威只是政府的一个分支部门,通常称为**PTT(Post, Telegraph & Telephone administration, 邮电部)**。在全球范围内,总体趋势是朝着自由、竞争的方向发展,而避免政府垄断。大多数欧洲国家已经将他们的 PTT 进行了私有化改造,或者已经部分私有化,但是在其他地方,这个进程仍然非常

缓慢。

由于有了这么多不同的服务供应商,所以,很显然有必要提供全球范围内的兼容性以保证一个国家的个人(和计算机)可以呼叫另一个国家中的个人或者计算机。实际上,这种需求很早以前就存在了。在1865年,欧洲许多政府的代表聚集在一起,形成了一个标准化组织,这就是今天的ITU(International Telecommunication Union,国际电信联盟)的前身。它的任务是对国际电信进行标准化。在当时而言,所谓国际电信是指电报。即使在当时的情况下,有一点也是很明显的:如果一半的国家使用莫尔斯编码,另一半国家使用其他的编码,则问题就来了。当电话也变成一种国际服务的时候,ITU又承担了电话标准化的工作。1947年,ITU成为联合国的一个代理机构。

ITU有三个主要部门:

- (1) 无线通信部门(ITU-R);
- (2) 电信标准化部门(ITU-T);
- (3) 开发部门(ITU-D)。

ITU-R关注全球范围内的无线电频率分配事宜,它将频段分配给有利益竞争的组织。我们将焦点主要集中在ITU-T上,它主要关注电话和数据通信系统。从1953年到1993年,ITU-T也称为CCITT,这是法文中的首字母缩写:Comité Consultatif International Télégraphique et Téléphonique。在1993年3月1日,CCITT进行了重组,以减少官僚主义,并且重新更名以反映出它的新角色。ITU-T和CCITT都在电话和数据通信领域中提出了不少建议。尽管自从1993年以后,原来的CCITT建议都被打上了ITU-T的标记,但是人们常常还会遇到CCITT的建议,比如CCITT X.25。

ITU-T有4类成员:

- (1) 政府部门;
- (2) 部门成员(sector members);
- (3) 合作成员(associate members);
- (4) 管理代理(regulatory agencies)。

ITU-T有将近200个政府成员,几乎包括联合国的每一个成员。由于美国没有PTT,所以它不得不找一个机构加入到ITU-T中以代表美国。这项任务落到了美国国务院的头上,可能是由于在ITU-T中必须要与其他的国家打交道,而这正好是国务院的长处。在ITU-T中大约有500个部门成员,包括电话公司(比如AT&T、Vodafone、WorldCom)、电信设备制造商(比如Cisco、Nokia、Nortel)、计算机厂商(比如Compaq、Sun、Toshiba)、芯片制造商(比如Intel、Motorola、TI)、媒体公司(比如AOL Time Warner、CBS、Sony),以及其他一些感兴趣的公司(比如Boeing、Samsung、Xerox)。各种非营利性的科学组织和工业界社团也是部门成员(比如IFIP和IATA)。合作成员是一些小一点的组织,它们对ITU-T中个别的研究组(Study Group)的工作比较感兴趣。管理代理是一些民间团体,它们非常关注电信业务,比如美国联邦通信委员会。

ITU-T的任务是对电话、电报和数据通信接口提供一些技术性的建议。这些建议通常会变成国际上认可的标准,比如V.24(在美国也称为EIA RS-232),它定义了大多数异步终端和外置调制解调器所使用的连接器中每根针的位置和含义。

应该注意到,ITU-T 的建议仅仅是技术性的建议,政府可以按照自己的意愿采用或者忽略这些建议。在实践中,愿意采用电话标准的国家与不采用标准的国家有很大的区别,虽然这些不采用标准的国家可以自由地建立自己的系统,但是他们付出的代价是将自己孤立起来了。

ITU-T 的实际工作是通过它的 14 个研究组来完成的,通常有 400 人。当前共有 14 个研究组,覆盖了各方面的主题,从电话计费一直到多媒体服务。为了尽可能地完成自己的任务,研究组又分成工作组(Working Party),工作组又进一步分为专家组(Expert Team),专家组再分为特别组(ad hoc group)。

尽管如此,ITU-T 实际还是完成了很多事情。自从它初期成立以来,差不多已经产生了 3000 份建议,合起来大约有 60 000 页纸。而且大多数建议被广泛地应用于实践中。例如,非常流行的 V.90 56kbps 调制解调器标准就是 ITU 的一个建议。

自从 20 世纪 80 年代开始,电信业开始从完全的国家性质转变成全球性的行业,随着这种转变的完成,标准也变得越来越重要,而且,越来越多的组织希望参与到标准制订工作中来。关于 ITU 的更多信息,请参看 Irmer, 1994。

### 1.6.2 国际标准领域中最有影响的组织

国际标准是由 ISO (International Standards Organization, 或 International Organization for Standardization)制定和发布的。ISO 是 1946 年成立的一个自愿的、非条约性质的组织。它的成员是 89 个成员国的国家标准组织。这些成员包括 ANSI(美国)、BSI(英国)、AFNOR(法国)、DIN(德国),以及其他 85 个成员。

ISO 为大量的学科制订标准,从螺钉和螺帽,一直到电话架的外形。目前已经发布了 13 000 多个标准,其中包括 OSI 标准。ISO 有将近 200 个技术委员会(TC, Technical Committee),这些技术委员会按照创建的顺序进行编号,每个技术委员会处理一个专门的主题。TC1 处理螺钉和螺帽(对螺丝钉的螺纹和斜度进行标准化)。TC97 处理计算机和信息处理技术。每个技术委员会会有一些分委员会(SC),分委员会又分成工作组(WG)。

WG 的实际工作大部分是由全球超过 100 000 个志愿者完成的。很多“志愿者”被其雇主指定为 ISO 工作,因为这些雇主们的产品正在进行标准化。其他的“志愿者”是政府官员,他们期望自己国家的一些做事方法变成国际标准。学术领域中的专家在许多 WG 中也很活跃。

在电信标准方面,ISO 和 ITU-T 通常联合起来以避免出现两个正式的但相互不兼容的国际标准(ISO 是 ITU-T 的一个成员)。

美国在 ISO 中的代表是 ANSI(American National Standards Institute, 美国国家标准协会),尽管它有这样一个名字,但实际上它是一个私有的、非政府的、非营利性的组织。它的成员有制造商、公共承运商和其他感兴趣的团体。ANSI 标准常常被 ISO 采纳为国际标准。

ISO 采纳国际标准的程序是经过精心设计的,以便尽可能获得广泛的同意和支持。当某一个国家标准组织感觉到在某一个领域中需要一个国际标准的时候,这个程序就开始了。然后形成一个工作组,由工作组提出一个 CD(Committee Draft, 委员会草案)。然



后该 CD 被传送给所有的成员体,他们有 6 个月的时间来评价这份草案。如果绝大多数成员都同意的话,则再生成一份修订文档,称为 DIS(Draft International Standard,国际标准草案)。然后散发给成员征求意见,并进行投票表决。在这一轮结果的基础上,国际标准的最后文本就可以准备出来,在获得认可之后可以发布。在有较大争议的领域,CD 或者 DIS 可能需要经过几次修订,才能获得足够的票数,整个过程可能要持续几年。

NIST(National Institute of Standards and Technology,国家标准和技术协会)是美国商业部的一个部门,它的前身是美国国家标准局。它颁发美国政府采购的强制性标准。当然,美国国防部除外,它有自己的标准。

在标准领域中另一个很大、也很有影响的组织是 IEEE(Institute of Electrical and Electronics Engineers,电气和电子工程师协会),它是世界上最大的专业组织。除了每年发行大量的杂志和召开几百次会议以外,IEEE 也有一个标准化组,该标准化组专门开发电气工程和计算领域中的标准。IEEE 的 802 委员会已经标准化了很多种类的 LAN。我们在本书后面将要学习其中一些这样的标准。实际的工作是由许多工作组来完成的,如图 1.38 所列。802 各个工作组的成功率并不高,有 802.x 这样的数字抬头并不保证会成功。但是成功之后的影响(特别是 802.3 和 802.11)却是非常巨大的。

序 号	主 题
802.1	LAN 的总体介绍和体系结构
802.2 ▼	逻辑链路控制
802.3 *	以太网
802.4 ▼	令牌总线(在制造业暂时用过一段时间)
802.5	令牌环(IBM 进入 LAN 领域的一项技术)
802.6 ▼	双队列双总线(早期的城域网)
802.7 ▼	关于宽带技术的技术咨询组
802.8 +	关于光纤技术的技术咨询组
802.9 ▼	同步 LAN(针对实时应用)
802.10 ▼	虚拟 LAN 和安全性
802.11 *	无线 LAN
802.12 ▼	需求的优先级(Hewlett Packard 的 AnyLAN)
802.13	不吉利的数字。没有人愿意使用它。
802.14 ▼	有线调制解调器(已废除:一个工业社团首先用过这个标准)
802.15 *	个人区域网络(蓝牙)
802.16 *	宽带无线
802.17	弹性的分组环

图 1.38 802 工作组

---

比较重要的工作组被标记为\*。标记为↓的已经停顿了。标记为†的已经被放弃,或者自行解散了。

### 1.6.3 Internet 标准领域中最有影响的组织

全球性的 Internet 有它自己的标准化机制,与 ITU-T 和 ISO 的标准化机制截然不同。它们之间的区别可以粗略地概括如下:参加 ITU 或者 ISO 标准会议的人总是穿着整齐;而参加 Internet 标准会议的人则穿着牛仔裤(如果在 San Diego 开会的话,则穿着短裤和 T 恤衫)。

ITU-T 和 ISO 的会议是由企业官员和政府公务员参加的,对于他们来说,标准化是他们的工作。他们把标准化看作是一件大好事,并致力于这项工作。相反,Internet 领域中的人则喜欢自由自在,并把这种自由当作一种原则。然而,成千上万的人都在做自己的事情,他们很少有交流。因此,无论多么地令人遗憾,有时候标准还是需要的。

当 ARPANET 刚刚建立起来的时候,美国国防部创建了一个非正式的委员会来监督它。在 1983 年,该委员会更名为 IAB(Internet Activities Board, Internet 活动委员会),并赋予了更多的任务,即,使有关 ARPANET 和 Internet 的研究人员或多或少地朝着同一个方向前进,这有点像是赶着一群猫往前走一样。缩写词“IAB”后来被改为 Internet Architecture Board(Internet 体系结构委员会)。

IAB 大约每 10 个成员牵头从事某一个重要方面的研究工作。IAB 每年开几次会议,以讨论研究的结果,并且给美国国防部和 NSF 提供反馈信息,因为当时美国国防部和 NSF 提供了大部分的经费。当需要一个新的标准(比如,一个新的路由算法)时,IAB 成员就会研究出新的标准,然后宣布新标准带来的变化,于是,研究生作为软件领域的中坚力量,就可以实现该标准。这里的交流过程是通过一系列技术报告(称为 RFC, Request For Comments)来完成的。RFC 被在线存储起来,任何感兴趣的人都可以从 [www.ietf.org/rfc](http://www.ietf.org/rfc) 访问它们。所有的 RFC 都按照创建的时间顺序编号,现在已经超过 3000 个 RFC 了。本书中我们将会引用到很多 RFC。

到了 1989 年的时候,Internet 增长得如此之快,以至于这种极端非正式的风格无法再适应快速的变化了。那时候许多厂商已经提供 TCP/IP 产品了,它们不想仅仅因为 10 个研究人员有了更好的思路就改变这些产品。于是,在 1989 年夏季,IAB 又被再次重组。这些研究人员被移到了 IRTF(Internet Research Task Force, Internet 研究任务组)中,IRTF 连同 IETF(Internet Engineering Task Force, Internet 工程任务组)一起成为 IAB 的附属机构。IAB 又接纳了更多的人参与进来,他们代表了更为广泛的组织,而不仅仅代表研究群体。IAB 是个自身永存的组,其中的成员每次服务两年,新成员由老成员指定。后来,Internet Society(Internet 协会)建立起来了,它由许多对 Internet 感兴趣的人组成。因此,从某种意义上讲,Internet 协会可以与 ACM 或者 IEEE 相提并论。它由选举出来的理事会管理,理事会指定 IAB 成员。

这种组织分离的思路是,让 IRTF 更加专注于长期的研究,而 IETF 处理短期的工程事项。IETF 被分成很多工作组,每个组解决某一个特定的问题。初期的时候,这些工作组的主席集合起来组成指导委员会,以指导整个工程组的工作。工作组的主题包括新的

应用、用户信息、与 OSI 的集成、路由与编址、安全、网络管理以及标准。后来,工作组如此之多(超过了 70 多个),以至于只好再按照领域来划分,每个领域的主席合起来组成指导委员会。

而且,IAB 还按照 ISO 的模式,采纳了一个更加正式的标准化过程。为了将一个基本的思想变成一个标准提案(Proposed Standard),首先要在 RFC 中完整地描述整个思想,并且在 Internet 群体中引起足够的兴趣以保证它的实际意义。为了进一步推进到标准草案(Draft Standard)阶段,必须有一个可正常工作的实现,经过至少两个独立的站点、至少 4 个月的严格测试才可以。如果 IAB 确信这个想法是合理的,并且软件也可以工作,那么它可以声明该 RFC 成为 Internet 标准。有些 Internet 标准已经成为美国国防部标准(MIL-STD),使它们成为美国国防部供应商的强制标准。David Clark 曾经对 Internet 标准发表过这样一句著名评论:Internet 的标准化是由“大概一致的意见”和“可以运行的代码”构成的。

## 1.7 度量单位

为了避免混淆,这里有必要明确地声明:如同一般的计算机科学中的做法一样,本书并不采用传统的英国度量单位(弗隆[forlong]·英石[stong]·双周[fortnight]系统)。主要的度量前缀如图 1.39 所列。这些前缀通常取它们的首字母缩写,再加上单位的首字母缩写(大写)构成复合单位,比如 KB、MB 等。不过,有一个例外(由于历史原因)是,kbps 代表的是 kilobit/sec。因此,1Mbps 通信线路可以传输  $10^6$  bit/s,并且,每  $10^{-12}$  秒有 100psec(或者 100ps)时钟滴答。由于 milli 和 micro 都以字母“m”开头,所以两者必须进行区分。通常情况下,“m”代表 milli,而“ $\mu$ ”(希腊字母)代表 micro。

指数	显式值	前缀	指数	显式值	前缀
$10^{-3}$	0.001	milli	$10^3$	1,000	Kilo
$10^{-6}$	0.000001	micro	$10^6$	1,000,000	Mega
$10^{-9}$	0.000000001	nano	$10^9$	1,000,000,000	Giga
$10^{-12}$	0.000000000001	pico	$10^{12}$	1,000,000,000,000	Tera
$10^{-15}$	0.000000000000001	femto	$10^{15}$	1,000,000,000,000,000	Peta
$10^{-18}$	0.000000000000000001	atto	$10^{18}$	1,000,000,000,000,000,000	Exa
$10^{-21}$	0.0000000000000000000001	zepto	$10^{21}$	1,000,000,000,000,000,000,000	Zetta
$10^{-24}$	0.000000000000000000000001	yocto	$10^{24}$	1,000,000,000,000,000,000,000,000	Yotta

图 1.39 基本度量前缀

另外,值得指出的是,为了计算内存、硬盘、文件和数据库的大小,按照工业界的实践来说,这些度量单位也有细微的区别。在这里,kilo 是指  $2^{10}$ (1024),而不是  $10^3$ (1000),因为内存总是 2 的幂次方。因此,1KB 内存包含 1024 字节,不是 1000 字节。类似地,1MB 内存包含  $2^{20}$ (1 048 576)字节,1GB 内存包含  $2^{30}$ (1 073 741 824)字节,1TB 数据库包含  $2^{40}$ (1 099 511 627 776)字节。然而,1kbps 通信线路每秒传输 1000 位,而 10Mbps 的

LAN 运行在 10 000 000bit/sec 的速度上,因为这些速度并不是 2 的幂次方。不幸的是,许多人倾向于将两种系统混淆起来,特别是磁盘的大小。在本书中,为了避免二义性,我们将使用符号 KB、MB 和 GB 分别代表  $2^{10}$ 、 $2^{20}$  和  $2^{30}$  字节;符号 kbps、Mbps 和 Gbps 分别代表  $10^3$ 、 $10^6$  和  $10^9$  bit/sec。

## 1.8 本书其余部分的概要

本书既讨论了计算机网络的原理,也讨论了计算机网络的实践。差不多每一章都从相关原理的讨论开始,然后用一些例子来说明这些原理。这些例子通常来自于 Internet 和无线网络,因为这两个网络都非常重要,而且差别也非常大。其他的例子也会在适当的地方给出。

本书的结构是按照图 1.24 所示的混合模型来组织的。从第 2 章开始,我们将从协议层次的底层逐渐往上进行讨论。第 2 章介绍了数据通信领域中的一些背景知识。它覆盖了有线的、无线的和卫星的传输系统。这些内容都关注于物理层,但是我们只讨论体系结构,而不是硬件方面的问题。这一章还讨论了物理层的一些例子,比如公共交换电话网、移动电话网络和有线电视网络。

第 3 章通过一系列逐渐复杂的例子讨论了数据链路层及其协议,同时也对这些协议进行了分析。之后,还讨论了一些重要的、实际应用的协议,包括 HDLC(用于低速和中速网络)和 PPP(用于 Internet)。

第 4 章关注于介质访问子层,这也是数据链路层的一部分。它所处理的基本问题是,当网络只有一个共享信道的时候(大多数 LAN 和有些卫星网络就是这样的),如何确定接下去谁该使用网络。这一章给出了很多例子,分别来自于有线 LAN、无线 LAN(特别是以太网)、无线 MAN、蓝牙和卫星网络。本章还讨论了网桥和数据链路交换机,它们用于将 LAN 连接起来。

第 5 章讨论网络层,特别是路由过程,这一章介绍了很多路由算法,有静态的,也有动态的。即使有了很好的路由算法,如果实际流量超过了网络所能承载的流量的话,就将发生拥塞,所以我们讨论了拥塞问题,并讨论了如何防止拥塞现象的发生。仅仅防止拥塞还不够,更好的应该是保证一定的服务质量。我们在这一章也将讨论这个问题。将异构网络连接起来构成互联网,这将会带来大量的问题,这一章也讨论了这些问题。同时,第 5 章还对 Internet 的网络层进行了全面的介绍。

第 6 章讨论了传输层。重点主要在于面向连接的协议,因为许多应用都需要这样的协议。这一章以例子的形式详细地讨论了一个传输服务及其实现。针对这个简单的例子还给出了实际的代码,以便读者可以看清楚它是如何实现的。关于 Internet 传输协议 UDP 和 TCP 也有详细的讨论,同时还涉及到了有关性能方面的一些问题。这一章也介绍了有关无线网络的一些问题。

第 7 章讨论应用层,包括它的协议和应用。第一个话题是 DNS,这相当于 Internet 的电话簿。接下来是电子邮件,包括对电子邮件协议的讨论。然后,我们转移到 Web 上,详细地讨论了静态内容、动态内容、客户端发生的事情、服务器端发生的事情、协议、性能、

无线 Web, 等等。最后, 我们介绍了网络多媒体, 包括流音频、Internet 广播电台, 以及视频点播。

第 8 章讨论了网络安全。这个话题涉及到所有的层, 所以, 在所有的层都介绍完之后再来讨论就会容易得多。这一章从对密码学的基本介绍开始, 然后介绍了如何用密码学的手段来保护通信、电子邮件和 Web。本书最后讨论到的一些主题涉及到安全性如何影响隐私、言论自由、审查制度, 以及其他一些可能会发生冲突的社会问题。

第 9 章包含了一份阅读材料列表, 按照章节的推荐顺序排列。这份参考列表的意图是希望能帮助那些想进一步从事网络研究的读者。这一章还有一份按字母顺序排列的参考文献列表, 本书中引用过的文献都在其中。

作者在 Prentice Hall 的 Web 站点是:

<http://www.prenhall.com/tanenbaum>

该站点上有一个页面包含了许多材料、FAQ、公司、工业界团体、专业组织、标准组织、技术、论文, 等等。

## 1.9 本章小结

计算机网络可以被用来提供大量的服务, 既可以针对公司, 也可以针对个人。对于公司而言, 使用共享服务器的个人计算机网络往往允许内部人员访问公司的信息。通常这种网络会采用客户-服务器模型, 雇员桌面上的客户工作站可以访问放在单独机房里的功能强大的服务器。对于个人来说, 通过网络可以访问各种各样的信息, 以及很多娱乐资源。个人用户往往通过一个调制解调器, 呼叫 ISP 以访问 Internet, 不过, 越来越多的家庭用户也开始具备固定的连接了。另外, 一个很快就会到来的领域是无线网络, 它具有各种新型的应用, 比如移动访问电子邮件和移动商务(m-commerce)。

大致而言, 网络可以分为 LAN、MAN、WAN 和互联网, 每一种网络都有其自己的特征、技术、速率和应用环境。LAN 可以覆盖一个建筑物, 并且可以高速运行。MAN 覆盖一个城市, 比如有线电视系统, 现在有许多人通过这个网络来访问 Internet。WAN 覆盖一个国家或者一个洲。LAN 和 MAN 是不经过交换的(也就是说, 没有路由器); 而 WAN 是需要交换的。无线网络正在受到广泛的欢迎, 特别是无线 LAN。多个网络可以相互连接起来, 从而构成互联网。

网络软件是由协议组成的, 所谓协议, 是指进程之间通信的规则。协议可以是无连接的, 也可以是面向连接的。大多数网络支持协议层次, 在协议层次中, 每一层向它的上层提供服务, 并且使低层所使用的协议的细节与上面的层隔离开。协议栈通常要么基于 OSI 模型, 要么基于 TCP/IP 模型。这两种模型都有网络层、传输层和应用层, 但是其他的层都不相同。设计方面的问题包括多路复用、流控制、错误控制, 等等。本书的大部分内容都在讲述这些协议以及它们的设计。

网络为用户提供服务。这些服务可以是面向连接的, 也可以是无连接的。在有的网络中, 无连接服务是在某一层上提供的, 而面向连接的服务则是由它上面的层提供的。

比较著名的网络包括 Internet、ATM 网络、以太网和 IEEE 802.11 无线 LAN。



Internet 是从 ARPANET 发展而来的,那时候许多其他的网络纷纷加入到 ARPANET 中,从而构成了一个互联网。现在的 Internet 实际上是成千上万个网络的集合,而不是单个网络。它最主要的特征是全部使用了 TCP/IP 协议栈。ATM 被广泛应用于电话系统内部,它的用途是提供长途数据流量服务。以太网是最为流行的 LAN,大多数公司和大学都采用了以太网。最后,无线 LAN 的速率可以很高(达到 54Mbps),现在正在被广泛地部署到各种环境中。

为了让多台计算机相互之间可以通话,这需要大量的标准化工作,不管是硬件方面还是软件方面的。诸如 ITU-T、ISO、IEEE 和 IAB 这样的组织管理了标准化进程中的各个不同部分。

## 习 题

1. 假设你已经将你的狗 Bernie 训练成可以携带一箱 3 盒 8mm 的磁带,而不是一小瓶白兰地。(当你的磁盘满了的时候,你可能会认为这是一次紧急事件。)每盒磁带的容量为 7GB 字节;无论你在哪里,狗跑向你的速度是 18km/h。请问,在什么距离范围内 Bernie 的数据传输速率会超过一条数据速率为 150Mbps(不算额外开销)的传输线?

2. LAN 的一个替代方案是简单的大型分时系统,通过终端为用户提供服务。请说出采用 LAN 的客户-服务器系统的两个好处。

3. 客户-服务器系统的性能会受到两个网络因素的影响:网络的带宽(每秒可以传输多少位数据)和延迟(将第一个数据位从客户端传送到服务器端需要多少秒时间)。请给出一个网络的例子,它具有高的带宽和高的延迟。然后再给出另一个网络的例子,它具有低的带宽和低的延迟。

4. 除了带宽和延迟以外,针对数字化的语音流量,要想让网络提供很好的服务质量,还需要哪个参数?

5. 在存储-转发类型的分组交换系统中,衡量延迟的一个因素是,在通过一个交换机的时候存储和转发一个分组需要多长时间。如果交换的时间为  $10\mu\text{s}$ ,并且客户-服务器系统中的客户在纽约、服务器在加州,那么交换时间是否会成为一个主要因素?假设铜和光纤中的传播速度是真空中光速的  $2/3$ 。

6. 一个客户-服务器系统使用了卫星网络,卫星的高度为 40 000km。在对一个请求进行响应的时候,最佳情形下的延迟是什么?

7. 在未来,当每个人都有有一个家庭终端连接到计算机网络的时候,就有可能对一个重要的未决案件进行即时的公民投票。最终,现在的立法机关都可以撤销了,从而让人民直接表达他们的意愿。这种直接民主的正面影响是非常显然的,请讨论一下某些负面的影响。

8. 有 5 个路由器要连接起来形成一个点到点子网。在每一对路由器之间,设计者可能会放上一条高速链路、一条中速链路,或者一条低速链路,甚至没有链路。如果产生并检查每一个拓扑图需要花掉 100ms 的计算机时间,那么检查所有的拓扑图需要多长

时间?

9. 在一个集中式的二叉树上,有  $2^n - 1$  个路由器相互连接起来;每个树节点上都有一个路由器。路由器  $i$  为了与路由器  $j$  进行通信,它要给树的根发送一条消息。然后树根将消息送下来给  $j$ 。假设所有的路由器对都是等概率出现的,请推导出当  $n$  很大时每条消息的平均跳数的一个近似表达式。

10. 广播式子网的一个缺点是,当多台主机同时企图访问信道的时候会造成信道浪费。作为一个简单的例子,假设时间被分成了离散的时槽,在每个时槽内, $n$  台主机中每台主机企图使用信道的概率为  $p$ 。请问由于碰撞而被浪费的时槽所占的百分比。

11. 请说出使用分层协议的两个理由。

12. Specialty Paint 公司的总裁打算与一个本地的啤酒酿造商合作生产一种啤酒罐。总裁告诉她的法律部门调查此事,后者又请工程部帮忙。结果是,总工程师打电话给啤酒酿造公司的技术负责人讨论该项目的技术问题。然后两位工程师又各自向他们的法律部门作了汇报。然后,法律部门通过电话安排了有关的法律方面的事宜。最后,两位公司总裁讨论了这次合作的经济方面的问题。你认为这是一个 OSI 模型意义上的多层协议的例子吗?

13. 在无连接通信和面向连接的通信两者之间,最主要的区别是什么?

14. 两个网络都可以提供可靠的面向连接的服务。其中一个提供可靠的字节流,另一个提供可靠的报文流。这两者是否相同? 如果你认为相同的话,为什么要有这样的区别? 如果不相同,请给出一个例子说明它们如何不同。

15. 在讨论网络协议的时候,“协商”意味着什么? 请给出一个例子。

16. 图 1.19 显示了一个服务。该图是否还隐含着其他的 service? 如果有的话,在哪里? 如果没有的话,说明为什么没有。

17. 在有些网络中,数据链路层处理传输错误的做法是,请求重传被损坏的帧。如果一帧被损坏的概率为  $p$ ,那么发送一帧所需要的平均传输次数是多少? 假设确认帧永远不会丢失。

18. OSI 模型中的哪一层处理以下的问题:

(a) 把传输的位流分成帧。

(b) 在通过子网的时候决定使用哪条路由路径。

19. 如果在数据链路层上交换的单元称为帧,而在网络层上交换的单元称为分组,那么应该是帧封装分组,还是分组封装帧? 说明你的理由。

20. 一个系统有  $n$  层协议的层次结构。应用程序产生的消息的长度为  $M$  字节。在每一层上需要加上一个  $h$  字节的头。请问,这些头需要占用多少比例的网络带宽。

21. 针对 OSI 参考模型和 TCP/IP 参考模型,请列举出两种相同的处理问题方法,以及两种不相同的处理问题方法。

22. TCP 和 UDP 之间最主要的区别是什么。

23. 图 1.25(b)中的子网被设计用来对抗核战争。需要多少颗炸弹可以将这些节点分成两个互不相连的集合。假设任何一颗炸弹都可以摧毁掉一个节点以及所有与它相连

的链路。

24. Internet 的规模差不多每隔 18 个月翻一倍。虽然没有人能够确切地知道具体的数字,但是,估计 2001 年 Internet 上的主机数目为 1 亿台。请利用这些数据计算出 2010 年 Internet 上将预计会有多少台主机。你相信你的结论吗?请说明你为什么相信或者为什么不相信。

25. 当一个文件在两台计算机之间传输的时候,可能会有两种不同的确认策略。在第一种策略中,该文件被分解成许多个分组,接收方会独立地确认每一个分组,但是文件传输过程作为整体并没有被确认。在第二种策略中,这些分组并没有被单独地确认,但是当整个文件到达的时候,它会被确认。请讨论这两种方案。

26. ATM 为什么使用小的、固定长度的信元?

27. 在原始的 802.3 标准中,一位是多长(按米来计算)?请使用 10Mbps 的传输速率,并且假设同轴电缆的传播速度是真空中光速的 2/3。

28. 一幅图像的分辨率为  $1024 \times 768$  像素,每个像素用 3 字节来表示。假设该图像没有被压缩。请问,通过 56kbps 的调制解调器信道来传输这幅图像需要多长时间?通过 1Mbps 的电缆调制解调器(cable modem)呢?通过 10Mbps 的以太网呢?通过 100Mbps 的以太网呢?

29. 以太网和无线网络既有相同点,也有不同点。以太网的一个特性是,在一个以太网上同一时刻只能传输一帧数据。802.11 也具有这样的特性吗?请讨论你的答案。

30. 无线网络很容易安装,这使得它们并不非常昂贵,因为安装费用通常会占去整个设备费用的很大比例。然而,它们也有一些缺点。请说出两个缺点。

31. 请列举出网络协议国际化的两个优点和缺点。

32. 当一个系统既有永久(固定)部分,又有可移动部分(比如 CD-ROM 驱动器和 CD-ROM)的时候,系统的标准化会显得非常重要,这样,不同的公司既可以生产永久部分,也可以生产可移动部分,并且它们总是能一起工作。请给出计算机工业界之外的三个例子,在每个例子中都存在国际标准。再给出计算机工业界之外的另外三个例子,但是这三个例子中都不存在国际标准。

33. 请列出你每天使用计算机网络的活动情况。如果这些网络突然间不工作了,你的生活将会有什么样的变化?

34. 请说出你的学校或者你工作所在的单位使用哪种网络?请描述一下网络的类型、拓扑结构,以及所使用的交换方法。

35. ping 程序使得你可以给指定的位置发送一个测试分组,并且看一看来回需要多长时间。请试着用一下 ping 程序,看一下从你所在的位置到几个已知的地点需要多长时间。利用这些数据,绘出在 Internet 上的单向传输时间与距离的函数关系。最好使用大学作为目标,因为大学的服务器的位置往往可以精确地知道。例如,berkeley.edu 在加州的 Berkeley;mit.edu 在麻省剑桥;vu.nl 在荷兰的 Amsterdam;www.usyd.edu.au 在澳大利亚的悉尼;www.uct.ac.za 在南非的 Cape Town。

36. 请到 IETF 的 Web 站点 [www.ietf.org](http://www.ietf.org) 去看一看他们在做些什么。选择一个你

---

所喜欢的项目,并对该项目中的问题和所考虑的方案,写一份半页纸的报告。

37. 在网络领域中,标准化是非常重要的。ITU 和 ISO 是最主要的官方标准化组织。请到他们的 Web 站点 [www.itu.org](http://www.itu.org) 和 [www.iso.org](http://www.iso.org),学习和了解一下他们的标准化工作。请写一份简短的报告说明他们已经标准化过的事情的种类。

38. Internet 是由大量的网络构成的。这些网络的组织方式决定了 Internet 的拓扑结构。关于 Internet 的拓扑结构有大量的信息可以在线获得。请使用一个搜索引擎来找到关于 Internet 拓扑结构的信息,并且写一份简短的报告来总结你的发现。

## 第2章 物理层

本章我们将讨论如图 1.24 所示的层次结构中的最低层。它为网络定义了机械的、电气的和时序的接口。我们将从数据传输的理论分析开始,之所以讨论理论分析,其目的只是为了弄明白在一个信道上传输数据时有些什么样的本质限制。

然后我们将讨论三种传输介质:有导向的(铜线和光纤)、无线的(地面上的无线电波)和卫星。这些是在现代网络中使用的关键传输技术的一些背景信息。

本章剩余部分将讲述三个通信系统的例子,它们在实践中被用于广域计算机网络。这三个例子是:(固定的)电话系统、移动电话系统和有线电视系统。所有这三个网络的骨干网都使用了光纤,但是它们的组织方式不同,并且在“最后一英里(last mile)”使用了不同的技术。

### 2.1 数据通信的理论基础

通过某种物理特性(比如电压或者电流)的变化,就可以在线路上传输信息。如果用—个以时间  $t$  为自变量的单值函数  $f(t)$  来表示电压或者电流的值,我们就可以对信号的行为进行建模,并且用数学的手段对信号进行分析。这一节的主题就是对信号的数学分析。

#### 2.1.1 傅立叶分析

在 19 世纪早期,法国数学家傅立叶(Jean-Baptiste Fourier)证明了:任何一个正常的周期为  $T$  的函数  $g(t)$ ,都可以展开成多个(可能无限个)正弦和余弦函数的和:

$$g(t) = \frac{1}{2}C + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft) \quad (2-1)$$

其中,  $f=1/T$  是基频,  $a_n$  和  $b_n$  是  $n$  次谐波(harmonics)的正弦和余弦振幅,  $C$  是常数。这种分解称为傅立叶级数(Fourier series)。利用傅立叶级数,一个函数可以进行重构;也就是说,如果周期  $T$  是已知的,振幅也已经给定了,那么通过等式(2-1)进行求和可以得到原始函数  $g(t)$ 。

一个有限时间的数据信号(所有的数据信号都是这样的)可以想象成它在一遍又一遍地重复整个模式(也就是说,在  $T$  到  $2T$  之间的信号与  $0$  到  $T$  之间的信号完全一样,以此类推)。

对于任何给定的  $g(t)$ ,通过在等式(2-1)两边同时乘以  $\sin(2\pi kft)$ ,然后再从  $0$  到  $T$  求积分,则可以计算出振幅  $a_n$ 。因为:



$$\int_0^T \sin(2\pi kft) \sin(2\pi nft) dt = \begin{cases} 0 & \text{当 } k \neq n \\ T/2 & \text{当 } k = n \end{cases}$$

所以,和式中只留下  $a_n, b_n$  项完全消失。类似地,通过在等式(2-1)两边乘以  $\cos(2\pi kft)$ ,然后再从 0 到  $T$  求积分,则可以计算出  $b_n$ 。另外,只要直接在等式两边求积分,就可以得到  $c$ 。执行这些操作的结果如下:

$$a_n = \frac{2}{T} \int_0^T g(t) \sin(2\pi nft) dt \quad b_n = \frac{2}{T} \int_0^T g(t) \cos(2\pi nft) dt \quad c = \frac{2}{T} \int_0^T g(t) dt$$

### 2.1.2 有限带宽的信号

为了看清楚上述数学原理如何应用到数据通信中,我们考虑一个特殊的例子:传输 ASCII 字符“b”,并且该字符被编码成 8 位的字节。被传输的位模式是 01100010。图 2.1(a)的左半部分显示了计算机传输该字符时的电压输出。对该信号进行傅立叶分析,则可以得到以下的系数:

$$\begin{aligned} a_n &= \frac{1}{\pi n} [\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)] \\ b_n &= \frac{1}{\pi n} [\sin(3\pi n/4) - \sin(37\pi n/4) - \sin(7\pi n/4) + \sin(6\pi n/4)] \\ c &= 3/4 \end{aligned}$$

最初几项的平方根振幅  $\sqrt{a_n^2 + b_n^2}$  如图 2.1(a)右边所示。我们之所以对这些值感兴趣,是因为它们的平方与对应频率处传输的能量成正比。

所有的传输设施在传输过程中都要损失一些能量。如果所有的傅立叶分量都等量地衰减,则结果信号将会在振幅上有所减小,但不会变形(也就是说,它将与图 2.1(a)有同样的方波形状)。不幸的是,所有的传输设施对于不同傅立叶分量的衰减并不相同,因此会导致信号变形。通常情况下,在从 0 到某一个频率  $f_c$  这一段范围内,振幅在传输过程中不会衰减,这里  $f_c$  可以用 Hz(赫兹)来度量,而在此截止频率  $f_c$  之上的频率所对应的振幅都会有不同程度的减弱。传输过程中振幅不会明显减弱的这一段频率范围称为带宽(bandwidth)。在实践中,截止频率并不会那么明显(尖锐),所以,通常引用的带宽是指从 0 到某一个能保留一半能量的频率处。

带宽是传输介质的一种物理特性,通常取决于介质的材料构成、厚度和长度。在有些情况下,电路中可能会引入一个滤波器,以便限制每个顾客可以使用的带宽数量。例如,一条电话线可能具有 1MHz 的带宽(针对短途电话),但是电话公司加了一个滤波器,以限制每个顾客的带宽为 3100Hz。这样的带宽对于正常的语音信号已经足够了,而且,通过限制顾客对资源的使用可以提高整个系统的运行效率。

现在我们来考虑,如果传输设施的带宽很低,以至于只有几个最低的频率才能被传输的话(也就是说只取等式(2-1)的前面几项作为信号函数的近似值),那么图 2.1(a)的信号看起来将会怎么样呢?图 2.1(b)显示了当原始信号经过一个只允许第一个谐波(即基频  $f$ )通过的信道时得到的结果信号。类似地,图 2.1(c)~(e)分别显示了经过高带宽信道时所得到的频谱和合成函数。

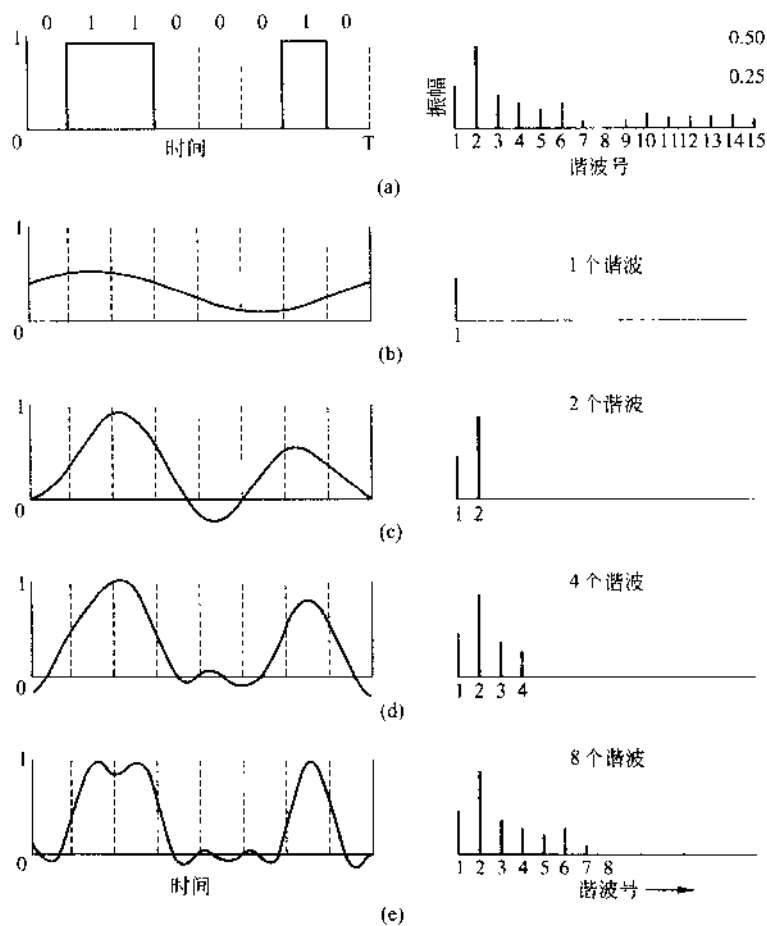


图 2.1

(a) 一个二进制信号和它的平方根傅立叶振幅；(b)~(e) 几个连续的信号，接近于原始的信号

假设位传输率为  $b$  位/秒，发送 8 位所需要的时间为  $8/b$  秒，因此 1 次谐波的频率是  $b/8\text{Hz}$ 。一条普通的电话线通常称为**语音级线路 (voice-grade line)**，它的截止频率(人为设置的)大约在 3000Hz 以上。这个限制意味着，最高可以通过的谐波次数大约是  $3000/(b/8)$  或者  $24\,000/b$ ，该截止频率不是尖锐的。

对于某些数据传输速率，这些数字间的关系如图 2.2 所示。从这些数字中可以看出，如果在语音级电话线上试图以 9600bps 的速率发送数据，则会将图 2.1(a) 所示的信号变换成图 2.1(c) 所示的信号，所以，要想精确地接收到原始的二进制位流，还需要更复杂的信号处理工作。应该可以看出，如果数据传输速率超过了 38.4kbps，即使传输设备没有任何噪声，也不可能传输任何二进制信号。换句话说，限制了带宽，也就限制了数据传输率，即使对于理想的信道也是如此。然而，有一些复杂的编码方案使用了几级电压值，从而可以获得更高的数据传输率。我们将在本章后面讨论这些方案。

bps	T(msec)	第一个谐波(Hz)	发送的谐波数
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0

图 2.2 数据传输率和谐波之间的关系

### 2.1.3 信道的最大数据传输率

早在 1924 年,AT&T 的工程师尼奎斯特(Henry Nyquist)就认识到,即使一条理想的信道,它的传输能力也是有限的。他推导出一个公式,用来表示一个有限带宽、无噪声信道的最大数据传输率。1948 年,香农(Claude Shannon)进一步把尼奎斯特的工作扩展到具有随机(热动力引起的)噪声的信道的情形(Shannon, 1948)。我们在这里简单地总结一下他们的经典结论。

尼奎斯特证明,如果任意一个信号已经通过了一个带宽为  $H$  的低通滤波器,则只要每秒  $2H$ (要求精确)次采样,过滤之后的信号就可以被完全重构出来。采样率超过每秒  $2H$  次是没有意义的,因为通过采样恢复出来的高频成分已经被滤掉了。如果该信号包含了  $V$  个离散级数,则尼奎斯特的定理为:

$$\text{最大数据传输率} = 2H \lg V \text{ (位/秒)}$$

例如,无噪声的 3kHz 信道不可能以超过 6000bps 的速率传输二进制(即只有两级)信号。

到现在为止,我们只考虑了无噪声信道。如果存在随机噪声的话,情况会急剧地恶化。由于系统中分子的运动,随机(热)噪声总是存在的。热噪声的数量可以用信号功率与噪声功率的比值来衡量,该比值称为**信噪比**(**signal-to-noise ratio**)。如果我们将信号功率记作  $S$ ,噪声功率记作  $N$ ,则信噪比为  $S/N$ 。通常情况下,人们并不使用信噪比本身,而是使用  $10 \lg S/N$  的值,该值称为**分贝**(**decibel, dB**)。信噪比  $S/N$  为 10,则称为 10dB;信噪比为 100,则称为 20dB;信噪比为 1000,则称为 30dB;等等。立体声放大器的制造商通常这样来定义带宽(频率范围):在这一段频率范围的每一头给 3dB 的噪声,他们的产品仍然是线性的。这差不多正好是放大因子减半处的点,因为  $\lg 2 \approx 0.5$ 。

香农的主要结论是,一条带宽为  $H$  Hz、信噪比为  $S/N$  的有噪声信道的最大数据传输率为:

$$\text{最大数据传输率(位/秒)} = H \lg(1 + S/N)$$

例如,假设一条信道的带宽为 3000Hz,信噪比为 30dB(这是电话系统中模拟部分的

典型参数),则不管传输的信号级数是多少,也不管采样频率是多少,该信道上最大数据传输率不超过 30 000bps。香农的结论可以从信息论的角度推演得到,并且适用于任何热噪声信道。想要用反例来驳倒这个结论是不可能的。应该注意的是,这只是一个上界,实际系统很少能够达到这个上界。

## 2.2 有导向的传输介质

物理层的目的是将原始的位流从一台机器传输到另一台机器上。有多种物理介质可以用于实际的传输过程。每一种传输介质都有它自己的特性,包括带宽、延迟、造价,以及安装和维护的难易程度。大致上可以将介质分为有导向的介质(比如铜线和光纤)和无导向的介质(比如空气中的无线电波和激光)。在接下来的几节中,我们将讨论这些介质。

### 2.2.1 磁介质

为了将数据从一台机器传输到另一台机器上,最常见的办法之一是将数据写到磁带上,或者其他可擦除介质(比如可刻录的 DVD)上,然后用物理的方法将磁带或者磁盘运送到目标机器上,再将数据读出来。这种方法并不像使用地球同步通信卫星那么复杂,但是它的性价比往往很高,即效率高而成本又低,特别适合于那些要求高带宽或者考虑到每一位传输费用的应用系统。

做一个简单的计算就能使这一点看得更清楚。工业标准的 Ultrium 磁带可以容纳 200GB(字节)信息。一个 60×60×60cm 的箱子可以装下 1000 个这样的磁带,所以总容量为 200TB,或者 1600Tb(即 1.6Pb)。通过联邦快递或者其他的快递公司,在 24 小时之内这一箱磁带可以投递到美国任何一个地方。这样一次传输的有效带宽是 1600 terabits/86,400s,或者 19Gbps。如果离目标只有一小时路程的话,则带宽可以增加超过 400Gbps。现在没有一个计算机网络可以达到这样的传输能力。

对于银行,每天会有大量的数据需要备份到第二台机器上(这样,即使面临大的洪水或者地震灾害的时候,银行还能继续工作),从性能角度来看,其他任何一种传输技术都不可能赶得上磁带的做法。当然,网络正在越来越快,但是磁带的密度也在提高。

如果我们再来看一下成本,也可以发现类似的情形。Ultrium 磁带的批发价大约是 40 美元。一盒磁带至少可以重用 10 次,所以每一箱磁带每一次使用的代价差不多是 4000 美元。再加上运输的费用大约是 1000 美元(可能还会更少一些),我们可以计算出,运送 200TB 的成本大约是 5000 美元。这样算下来,运输 1GB 数据的费用不到 3 美分。没有一个网络能与它竞争。这个例子所隐含的意义是:

不要低估了一辆满载磁带、在高速公路上飞驰的货车的带宽。

### 2.2.2 双绞线

尽管磁带的带宽特性很好,但是其延迟特性却非常差。其传输时间是以分或者小时,而不是以毫秒来计算的。许多应用需要在线的连接。一种最老式,但至今仍很常用的传输介质是双绞线(twisted pair)。双绞线是由两根相互绝缘的铜线组成的,通常这两根铜

线的直径大约为 1mm。这两根铜线以螺旋状的形式绞在一起,就像一个 DNA 分子。之所以要绞在一起,是因为两根平行的线会构成一个很好的天线。当两根线绞在一起后,不同电线产生的干扰波会相互抵消,所以电线的辐射就会显著地减弱。

双绞线最常见的应用是电话系统。几乎所有的电话都是通过双绞线连接到电话公司的交换局的。双绞线可以延伸几公里而不需要放大,但是如果距离再远的话,就需要中继器了。当许多双绞线并行在一起经过一定距离时,比如所有的双绞线都从一个公寓楼连接到电话公司的交换局,那么它们应该捆成一束,再加一层保护套。这些捆起来的绞线如果不拧起来的话,它们会相互干扰。在有的地区,电话线还从电线杆上走,所以你看常常可以看到直径为几厘米厚的电话线束。

双绞线既可以用于传输模拟信号,也可以用于传输数字信号。带宽取决于铜线的厚度(即直径)以及传输的距离;在许多情况下,几公里的传输距离内双绞线可以达到几 Mbps 的带宽。由于双绞线具有足够的传输性能,以及相对较低的成本,所以它的应用很广泛,并且可能还会持续很多年。

双绞线电缆有很多种类。其中两种对于计算机网络非常重要。3 类(category 3)双绞线由两根相互绝缘的铜线轻轻地拧在一起。通常这样四对双绞线被组合在一个塑料套内,一方面可以保护这些铜线,另一方面也将它们组合在一起。大约在 1988 年以前,大多数办公楼都使用 3 类电缆,从每一层的中心配线柜(wiring closet)连接到各个办公室。这种方案使得每个办公室中的 4 部普通电话或者 2 部多路电话可以同时连接到配线柜中的电话公司设备上。

从 1988 年开始引入了更为高级的 5 类(category 5)双绞线。它与 3 类双绞线非常类似,但是每厘米内拧得更紧,这样的好处是串音更少,并且在长距离传输中保持更好的信号质量。这使得它们更加适合于高速的计算机通信。随后的类别是 6 和 7,它们分别可以承载带宽为 250MHz 和 600MHz 的信号(相对而言,3 类和 5 类只能承载 16MHz 和 100MHz 的信号)。

所有这些双绞线类型通常称为 UTP(Unshielded Twisted Pair,无屏蔽双绞线),与此相反的是,IBM 在 20 世纪 80 年代初期引入了体积相对较大、价格昂贵的屏蔽双绞线,但是这种双绞线除了在 IBM 的设备上使用以外,在其他地方并没有流行起来。双绞线电缆如图 2.3 所示。

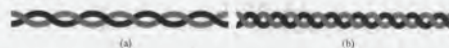


图 2.3  
(a) 3 类 UTP (b) 5 类 UTP

### 2.2.3 同轴电缆

另一种常见的传输介质是同轴电缆(coaxial cable,相对于许多其他的传输介质,常常而称为“coax”,发音为“co-ax”),它比双绞线有更好的屏蔽性,所以它可以在很高的速率



上传输很远的距离。广泛使用的同轴电缆有两种。一种是  $50\Omega$  的电缆,它从一开始就被用于数字传输。另一种是  $75\Omega$  的电缆,常常用于模拟传输和有线电视,但是随着用电缆连接 Internet (Internet over cable) 的发展趋势,这种电缆正变得越来越重要。这种区别是由于历史原因造成的,并非由于技术的因素(例如,早期的偶极天线的阻抗是  $300\Omega$ ,所以,很容易使用现有的  $4:1$  阻抗匹配变压器)。

同轴电缆以硬铜线为芯,外面包上一层绝缘材料,绝缘材料的外面包围上一层密织的网状导体。导体外面又覆盖上一层保护性的塑料外套。部分切开的同轴电缆如图 2.4 所示。



图 2.4 同轴电缆示意图

同轴电缆的这种结构和屏蔽性使得它既有很高的带宽,又有很好的抗噪特性。可能达到的带宽取决于电缆的质量、长度以及数据信号的信噪比。现代的电缆可以达到  $1\text{GHz}$  的带宽。过去,同轴电缆在电话系统中广泛地用于长途线路,但是现在,在长途路径上大部分已经被光纤所取代。然而,同轴电缆仍然广泛地应用于有线电视和城域网中。

#### 2.2.4 光纤

令计算机工业界很多人士最引以为豪的是计算机技术的快速发展。最初(1981 年) IBM PC 运行的时钟速度为  $4.77\text{MHz}$ 。20 年后的今天,PC 可以运行在  $2\text{GHz}$  的速度上,差不多每 10 年增长了 20 倍。

在同样这段时间中,广域数据通信也从  $56\text{kbps}$  (ARPANET) 发展到  $1\text{Gbps}$  (现代的光纤通信),每 10 年增长了 125 倍还多。而与此同时,错误率从每位  $10^{-4}$  下降到几乎为零。

而且,单个 CPU 开始逼近物理的极限,比如光速和热消散问题。相反,利用当前的光纤技术,可以达到的带宽肯定会超过  $50\,000\text{Gbps}$  (即  $50\text{Tbps}$ ),而且很多人还在努力地寻找更好的技术和材料。如今,在实际中,关于  $10\text{Gbps}$  的信号限制问题是由于我们不能够更快地在电子信号和光学信号之间进行转换而造成的,不过,在实验室里,在单根光纤上已经可以达到  $100\text{Gbps}$  的速率了。

在计算能力和通信能力的竞赛过程中,通信赢了。由于这一代计算机科学家和工程师受到尼奎斯特和香农关于铜线带宽限制理论的影响,所以他们并没有意识到实际上带宽几乎是无限的(当然并不是不需要代价)。新的明智的思维方式应该是,所有的计算机都无可救药地慢,而网络应该尽可能地避免计算开销,不管浪费的带宽有多少。在这一小节中,我们将介绍光纤技术,看一看这种传输技术是如何工作的。

一个光纤传输系统有三个关键部件：光源、传输介质和检测器。习惯上，一个光脉冲表示位“1”，没有光则表示位“0”。传输介质是一根极细的玻璃纤维。当有光照到检测器上的时候，它产生一个电脉冲。在光纤的一端放上一个光源，另一端放上一个检测器，我们就有了一个单向的数据传输系统，它接受电信号，转换成光脉冲并传输出去，然后在接收端重新转换成电信号。

这种传输系统会漏光，在实际中并没有用处，它只是说明了一种有趣的物理学原理。当一束光线从一种介质到达另一种介质的时候，譬如从熔融氧化硅到空气中，在氧化硅和空气的边界上，光线会发生折射（弯曲），如图 2.5(a) 所示。这里我们看到一束光入射在界面上的角度是  $\alpha_1$ ，出来的时候角度为  $\beta_1$ 。折射的角度取决于两种介质的特性（特别是它们的折射率）。如果入射角度超过了某一个特定的临界值，则光线就会被反射回到氧化硅中，不会再有光线逃到空气中。因此，入射角度大于等于临界值的光线将被限定在光纤的内部，如图 2.5(b)，它可以传播好几公里而几乎没有损失。

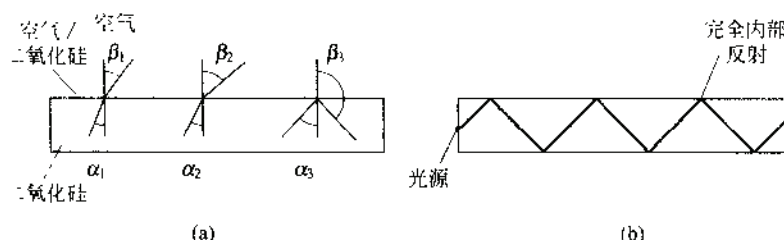


图 2.5

(a) 三个例子：一束光按照不同的角度从氧化硅内部射到空气/氧化硅的界面上；  
(b) 通过完全的内部反射，光线在光纤内部传播

图 2.5(b) 只显示了一束截留光，但是，由于任何入射角度大于临界值的光束都会在内部反射，所以许多不同的光束可以在不同的反射角传播。可以这样说，每一束光都有不同的模式(mode)，所以，一根具有这种特性的光纤称为多模光纤(multimode fiber)。

然而，如果光纤的直径减小到几个光波长大小的时候，则光纤就如同一个波导，光只能按直线传播了，而不会反射，这样的光纤称为单模光纤(single-mode fiber)。单模光纤比较昂贵，但广泛应用于长距离传输。目前可使用的单模光纤可以在 50Gbps 的速率上传输数据 100km 而不用放大。在实验室中，还可以得到更高的数据传输率，但是距离要短一些。

### 通过光纤传输光

光纤是由玻璃制成的，玻璃又是由沙子做成的，而沙子是一种取之不尽用之不竭的原始廉价材料。玻璃的制造可以追溯到古埃及，但是他们的玻璃不能超过 1mm 厚，否则的话，光就不能透过了。足够透明而能用于窗户的玻璃是在文艺复兴时期发明的。现代用于光纤的玻璃非常透明，假设如果海洋中全部是这样的玻璃，而不是海水的话，那么你可以从海面上一直看得到海底，就好像晴天时候从飞机上能够看到地面一样。

光通过玻璃的衰减取决于光的波长(以及玻璃的某些物理特性)。对于光纤中所使用的玻璃而言,这种衰减如图 2.6 所示,图中显示了光纤每公里衰减的分贝。衰减的分贝数可由下面的公式来计算:

$$\text{衰减的分贝数} = 10 \lg(\text{传输的能量}/\text{接收的能量})$$

例如,如果损失了一半的能量,则衰减为  $10 \lg 2 = 3 \text{ dB}$ 。该图显示了光谱中接近红外的部分,也是实践中所用到的部分。可见光的波长相对要短一些,从  $0.4 \sim 0.7 \mu\text{m}$  ( $1 \mu\text{m}$  等于  $10^{-6} \text{ m}$ )。坚持使用米制单位的人喜欢将这些波长写成从  $400 \sim 700 \text{ nm}$ ,但是我们仍然沿用传统的用法。

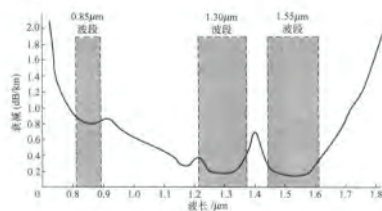


图 2.6 红外区域的光通过光纤时的衰减情况

有三个波段用于光纤通信。它们分别集中在  $0.85$ 、 $1.30$  和  $1.55 \mu\text{m}$  处。后两波段有很好的衰减特性(每公里的损失小于  $5\%$ )。 $0.85 \mu\text{m}$  波段有较高的衰减,但是在这个波段上,我们可以从同一种材料(砷化镓)获得激光和电子。所有这三个波段都有  $25\,000 \sim 30\,000 \text{ GHz}$  的宽度。

光脉冲在光纤中传播的时候会散开,这种现象称为色散(chromatic dispersion)。色散的量取决于波长。保持这些散开的脉冲不会产生交叠的一种办法是加大它们之间的距离,但是只有降低了信号速率才能做到这一点。幸运的是,现在已经发现,通过将脉冲做成一种特殊的形状(该形状与双曲余弦的倒数有关),几乎所有的色散效应都不再存在了,而且,有可能将脉冲发送几千公里而不会有明显的形状失真。这些脉冲称为孤波(soliton)。现在大量的研究工作正在进行,以使孤波尽快走出实验室,进入实用领域。

#### 光纤

光缆和同轴电缆非常相似,只不过光缆没有那一层密织的网。图 2.7(a)显示了一根光纤的内部结构。中间是玻璃芯,光线将在这里传播。在多模光纤中,玻璃芯的直径往往是  $50 \mu\text{m}$ ,相当于一根头发丝的粗细。在单模光纤中,玻璃芯的直径为  $8 \sim 10 \mu\text{m}$ 。

玻璃芯的外面是一个玻璃封套,玻璃封套的折射率比玻璃芯低,这样可以保证所有的光都在玻璃芯内。接下来是一层薄薄的塑料外套,以保护里边的玻璃封套。通常光纤扎

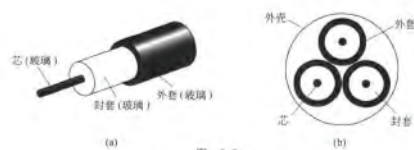


图 2.7 (a) 单根光纤的结构; (b) 三根光纤封装起来的截面

成束,外面再加一层护套。图 2.7(b)显示了一个内含三根光纤的截面。

陆地上的光纤通常被埋在地表 1 米以内,所以,有时候它们也会被农用工具或者地老鼠破坏。在靠近海岸的地方,海洋光纤埋在电缆沟里(利用一种称为海型的工具)。在深水中,它们直接被放在水底,所以很有可能被渔船撞坏,或者被大一点的鱼咬坏。

光纤可以按照三种不同的方式连接起来。第一,它们在连接器上终止,并插入到光纤插口中。连接器损失大约 10%~20% 的光,但是它可以使系统的重新配置工作更加容易。

第二,它们可以通过机械的手段接合起来。机械接合的做法是,将两根小心切割好的光纤头放在一个特殊的套管中,然后将它们适当夹紧。让光线通过结合处,然后适当进行调整以使信号尽可能最大,这样可以提高接合处的质量。对于受过训练的人来说,机械接合大约需要 5 分钟,并且会有 10% 的光损失。

第三,两根光纤可以融合(熔合)在一起,以形成一个非常结实的连接。融合得到的光纤几乎与单根光纤一样好,但尽管如此,少量的衰减仍然存在。

对于所有这三种接合手段,在接合点上可能会发生反射,并且,反射的能量可能会干扰原来的信号。

通常有两种光源可以用作信号源: LED(Light Emitting Diodes, 发光二极管)和半导体激光。它们有不同的特性,如图 2.8 所示。通过在光源和光纤之间插入 Fabry-Perot 干涉计或者 Mach-Zehnder 干涉计,可以对波长进行调节。Fabry-Perot 干涉计是一个简单的由两面平行的镜子构成的共振腔。光线垂直于镜面入射,共振腔的长度会筛选出那些波长为其整数分之一的波。Mach-Zehnder 干涉计将光分为两束,这两束光经过的距离稍有不同。然后它们在终端处重新合起来,于是只有特定波长的波才能同相。

项目	LED	半导体激光
数据传输率	低	高
光纤类型	多模	多模或者单模
距离	短	长
寿命	寿命长	寿命短
温度敏感性	小	敏感
成本	成本低	昂贵

图 2.8 半导体激光和 LED 作为光源的比较

光纤的接收端是一个光敏二极管,当遇到光照时,它会发出一个电脉冲。光敏二极管的典型响应时间为 1ns,所以这就限制了数据传输率为 1Gbps 左右。热噪声也是一个问题,所以光脉冲必须保证具有足够的能量以便能够被检测到。如果脉冲的能量足够强的话,则错误率也可以降低到任意小。

### 光纤网络

光纤可以用于 LAN,也可以用于远程传输,但在远程传输时接入光纤比连接到以太网要复杂得多。解决问题的方法之一是使用环网,因为环网实际上只是一组点到点链路的集合,如图 2.9 所示。每台计算机上的接口都将光脉冲流送给下一段链路,该接口本身也起到了 T 型连接的作用,以便计算机可以发送或者接受消息。

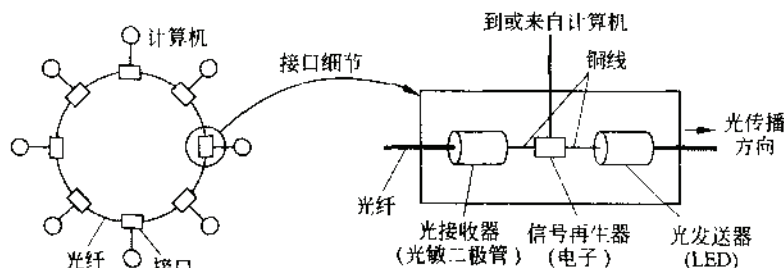


图 2.9 带有有源中继器的光纤环网

有两种接口可以使用。被动接口包含两个接头,分别融合到主光纤上。一个接头的--端有一个 LED 或者激光二极管(用于传输),另一个接头有一个光敏二极管(用于接收)。接头本身完全是被动的,因此也极为可靠,因为即使 LED 或者光敏二极管损坏了,也不会打破整个环,只是一台计算机脱离网络而已。

另一种接口类型是有源中继器(active repeater),如图 2.9 所示。进来的光被转换为一个电信号,如果它已经被减弱的话,则重新产生该信号,使它具有完全的能量,然后再以光的形式传输该信号。与计算机的接口是一根普通的铜线,它还连接到信号再生器上。纯粹的光中继器现在也已经在使用了,这种设备不需要从光信号到电信号,再到光信号的转换过程,所以它们可以以极高的带宽运行。

如果一个有源中继器损坏了,则整个环就被打破了,网络无法继续工作。另一方面,因为在每一个接口上信号都要重新产生,所以单独的从计算机到计算机之间的链路可以达到几公里长,因而整个环的长度几乎没有限制。对于无源接口,在每个连接处都有光损失,所以计算机的数量和环的总长度会大大受限制。

环结构并不是用光纤来构建 LAN 的惟一途径。利用如图 2.10 所示的无源星形(passive star)结构来实现硬件广播也是一种可行的做法。在这个设计方案中,每个接口都有一根光纤,从它的发送器连接到硅柱上,进来的光纤融合到硅柱的一端。类似地,融合到硅柱另一端的光纤也连接到每个接收器上。一旦有一个接口发出一个光脉冲,它就会在无源星形结构中扩散开,从而照亮所有的接收器,这样就实现了信号的广播。实际上,无源星形结构将所有进来的信号组合起来,然后再将合并之后的信号传输到所有的线



路上。由于进来的能量被分散到所有的输出线上了,所以网络中节点的数量受到光敏二极管感光效果的限制。

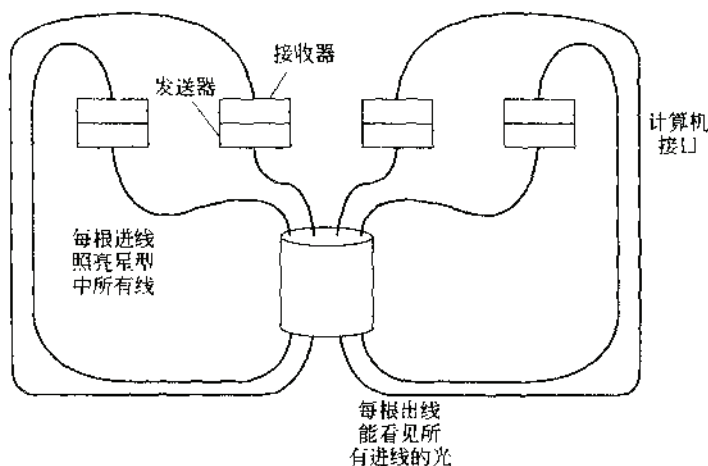


图 2.10 光纤网络中的无源星形连接

### 光纤和铜线的比较

比较一下光纤和铜线是很有意义的。光纤有很多优点。首先,光纤比铜线能够处理更高的带宽,这使得它非常适用于高端网络。由于光纤具有相对较低的衰减,所以在较长的线路上,大约每 50km 才需要中继器,而对于铜线而言,大约每 5km 就需要中继器。实际上,这也节约了网络的成本。另一方面,光纤还具有不受电源震荡、电磁干扰以及电源故障的影响等优点。而且它也不受空气中腐蚀性化学物质侵蚀的影响,这使得它能够应用于恶劣的工业环境。

奇怪的是,电话公司喜欢光纤却是因为另外的理由:光纤细小而且重量较轻。许多现有的电缆管道都已经完全塞满了,因而没有空间再增加新的容量。将所有原来的铜线都清除掉,再换上光纤,这样可以腾出管道的空间,而且,铜线还可以转卖给炼铜商,他们将铜线看作一种高级的矿物原材料。另外,光纤比铜轻得多。一千根 1km 长的双绞线的重量为 8000kg。两根光纤的容量超过了这 1000 根双绞线的容量,但是重量只有 100kg,所以,使用光纤可以极大地降低对于机械支撑系统的需要,建立和维护这样的系统也是非常昂贵的。对于新的路由线路,光纤比铜线更有优势,因为它的安装费用低得多。

最后,光纤不会漏光,而且难以接入进来,这些特性使得光纤很难被搭线窃听,因而具有很高的安全性。

另一方面,光纤也有不足之处。光纤是一项相对陌生的技术,要求较高的操作技能,这并不是每一个工程师都具备的;当光纤被过分弯曲时,容易损坏。由于光传输技术本质上是单向的,所以在双向通信的时候要求使用两根光纤,或者在一根光纤上划分两个频段。最后,光纤接口的成本比电子接口的成本高得多。然而,在将来,凡是超过几米距离的所有固定数据通信显然都应该使用光纤。有关光纤和光纤网络的讨论,请参考

(Hecht, 2001)。

## 2.3 无线传输

我们的时代已经出现了这样的信息“上瘾”者：他们需要随时保持在线。对于这些移动用户，双绞线、同轴电缆和光纤都无济于事。他们需要利用他们的膝上型计算机、笔记本电脑、袖珍电脑、掌上电脑或者手表式计算机来获取数据，而不受陆地上的通信设施的限制。对于这些用户，无线通信是解决问题的办法。在本节中，我们将从一般意义上来介绍无线通信，因为它除了能给那些在海滩上还想冲浪 Web 的用户提供 Internet 连接以外，还有很多其他的重要应用。

很多人相信，将来只有两种通信：光纤的和无线的。所有固定的（即不移动的）计算机、电话、传真等设备将使用光纤，所有移动的设备将使用无线通信。

在有些环境中，即使对于固定的设备，无线通信也有优势。例如，如果由于地形（高山、丛林、沼泽等）的原因而难以将光纤接入到建筑物内，则使用无线通信可能更好一些。值得一提的是，现代的无线数字通信开始于夏威夷岛，太平洋将那里的用户隔开了，而电话系统并不能完全解决他们的通信问题。

### 2.3.1 电磁波谱

当电子运动的时候，它们就会产生电磁波，而电磁波可以在空气中传播（即使在真空中也可以传播）。英国物理学家麦克斯韦（James Clerk Maxwell）在 1865 年就预言了这种波的存在，到 1887 年德国物理学家赫兹（Heinrich Hertz）第一次观测到了这种波。电磁波每秒振动的次数称为它的频率  $f$ ，可以用赫兹（Hz）来度量（以此来纪念物理学家赫兹）。两个相邻的波峰（或者波谷）之间的距离称为波长，通常用希腊字母  $\lambda$ （lambda）来表示。

当一个适当大小的天线连接到一块电路上的时候，电磁波就可以有效地广播出去，在一定距离之外的接收者可以收到电磁波。所有的无线通信都是基于这样的原理来实现的。

在真空中，所有的电磁波按同样的速度传播，跟它们的频率无关。这个速度称为光速  $c$ ，近似为  $3 \times 10^8$  m/s，或者每纳秒大约 1 英寸（30cm）（光速的一个用途是用来重新定义英寸单位：光在真空中 1 纳秒走过的距离）。在铜线或者光纤中，电磁波的速度会慢一些，大约是这个值的 2/3，并且对频率稍有关系。光速是速度的极限，没有物体或者信号可以运动得比光速还快。

$f$ 、 $\lambda$  和  $c$ （真空速度）之间的基本关系是：

$$\lambda f = c \quad (2-2)$$

由于  $c$  是常数，所以，如果我们知道了  $f$ ，就可以算出  $\lambda$ ，反之也一样。一条经验规则是，如果  $\lambda$  的单位是米， $f$  的单位是 MHz，则  $\lambda f \approx 300$ 。例如，100MHz 的波长大约为 3m，1000MHz 的波长大约为 0.3m，0.1m 波长的频率为 3000MHz。

电磁波谱如图 2.11 所示。在波谱中，无线电波、微波、红外光和可见光都可以通过调

节振幅、频率或者波的相位来传输信息。紫外线、X射线和 $\gamma$ 射线可能会更好一些,因为它们的频率更高,但是这种波很难产生和调节,通过建筑物的时候传播情况并不好,而且对生物有害。图 2.11 列出的波段是正式的 ITU 名字,并且也是以波长为根据的,所以 LF 波段为 1~10km(大约 30~300kHz)。术语 LF、MF 和 HF 分别指低频、中频和高频。很显然,当最初分配名字的时候,没有人想到会使用超过 10MHz 的频段,所以,这些高频段后来被命名为甚高频、特高频、超高频、极高频和巨高频段(相应的缩写分别为 VHF、UHF、SHF、EHF 和 THF)。再往上就没有名字了,但是有可能会出“难以置信的”、“惊人的”和“非凡的”高频段(IHF、AHF 和 PHF)。

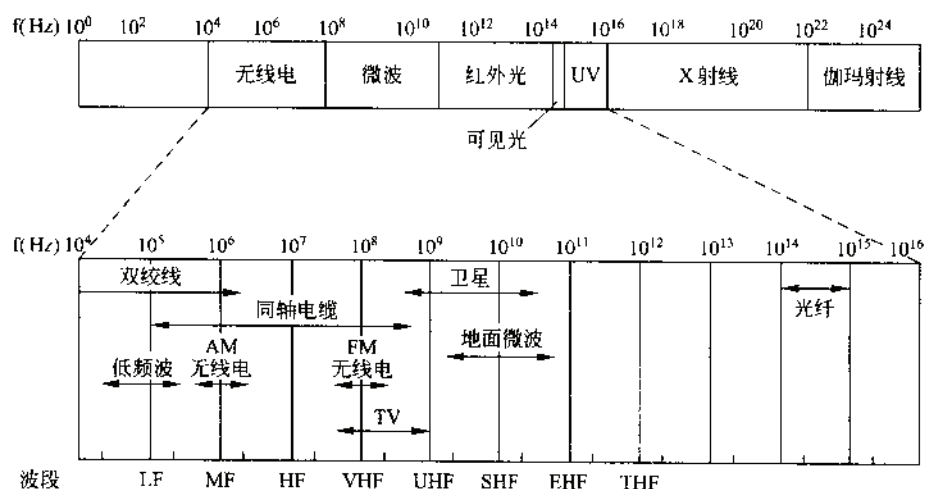


图 2.11 电磁波谱和它在通信中的用途

电磁波可以承载的信息量与它的波长有关。利用当前的技术,在低频处每个赫兹编码少量几位信息是可以做到的,但是在高频处通常可以编码 8 位信息,所以,750MHz 带宽的同轴电缆可以承载几个 Gbps。从图 2.11 明显可以看出,为什么网络界如此看好光纤的前景。

如果我们对等式(2-2)计算出  $f$ ,并且两边对  $\lambda$  求微分,则得到:

$$\frac{df}{d\lambda} = -\frac{c}{\lambda^2}$$

如果我们以有限差分来代替微分,并且只看绝对值,则可以得到:

$$\Delta f = \frac{c\Delta\lambda}{\lambda^2} \quad (2-3)$$

因此,给定了波段的宽度  $\Delta\lambda$ ,我们可以计算出对应的频段  $\Delta f$ ,然后,可以计算出该波段的数据传输率。波段越宽,数据传输率越高。举一个例子,考虑图 2.6 中的  $1.30\mu\text{m}$  波段,其  $\lambda = 1.3 \times 10^{-6}$ ,  $\Delta\lambda = 0.17 \times 10^{-6}$ ,所以  $\Delta f$  大约为 30THz。如果编码率为 8 位/Hz,则数据传输率为 240Tbps。

大多数传输都使用窄的频段(即  $\Delta f/f$  远远小于 1)以获得最佳的接收能力(每很多 W/Hz)。然而,在有些情况下,也会使用宽的频段(有两种变种用法)。在跳频扩频

(frequency hopping spread spectrum)中,发送方每秒几百次地从一种频率跳到另一种频率。这种技术在军事通信中很流行,因为它使得通信过程很难被检测到,因此对方也就不太可能干扰通信。而且,它对于多路衰减(multipath fading)现象也有很好的抵抗性,因为直接的信号总是首先到达接收方。反射信号走的路径长一些,所以到达得晚。那时接收方可能已经改变了频率,根本就不再接收原来频率上的信号了,这样也就消除了原始直接信号与反射信号之间的干扰。最近几年,这项技术已经应用到商业领域中,例如,802.11 和蓝牙都用到了这项技术。

有意思的是,这项技术是由澳大利亚女明星 Hedy Lamarr 参与合作发明的。她的第一任丈夫是一个武器制造商,他告诉她,要想阻止用于控制鱼雷的无线电信号是多么地容易。当她发现他正出售武器给希特勒的时候,她非常害怕,便乔装成一个女仆离开了他,然后到好莱坞继续她的演艺事业。她利用空余的时间,发明了跳频技术来帮助盟军。她的方案用到了 88 个频率,正好是钢琴上键(频率)的数目。由于她和她的朋友(作曲家 George Antheil)的这项发明,他们获得了美国专利 2,292,387。然而,他们未能使美国海军相信他们的发明有任何实际的用途,所以他们没有得到任何专利使用费。在专利过期之后很多年,这项技术才变得流行起来。

另一种扩频的形式是直接序列扩频(direct sequence spread spectrum),它将信号展开在一个很宽的频段上,这项技术在商业领域中也正在受到欢迎。特别是,有些第二代移动电话使用了这项技术,并且它将在第三代移动电话中占主导地位,这要归功于它有很好的光谱效率、抗噪声能力和其他一些特性。有些无线 LAN 也用到了这项技术。我们将在本章后面再讨论扩频。要想了解扩频通信有趣而详细的历史,请参考(Scholtz, 1982)。

现在,我们假设所有的传输都使用一个窄的频段。接下来我们将讨论图 2.11 所显示的电磁谱的各个部分,首先从无线电波开始。

### 2.3.2 无线电传输

无线电波很容易产生,也可以传输很长的距离,并且易于穿透建筑物,所以,无线电波被广泛应用于通信领域,无论是室内的还是室外的。无线电波是全方向的,这意味着它们将从源沿着所有的方向传播开去,所以发射设备和接收设备不必在物理上小心地对齐。

全方向发射有时候是好事,但有时候也是坏事。20 世纪 70 年代,通用汽车公司决定在所有新型号的凯迪拉克汽车上安装计算机控制的防抱死刹车。当驾驶员踏上刹车板的时候,计算机将刹车时收时开,而不是一下将刹车锁死。在一个晴朗的白天,俄亥俄州的一个高速公路巡警开始使用他的新无线电设备呼叫总部,突然他旁边的一辆凯迪拉克像一匹脱了缰的野马似的。当警察把这辆车叫到路边的时候,驾驶员说他什么也没做,汽车就发疯了。

最后,渐渐地发现了规律:凯迪拉克有时候会发疯,但只有在俄亥俄州的主高速公路上,并且只有当高速公路巡警在监视的时候才会发生。有很长一段时间,通用汽车公司都不能理解为什么这款汽车在所有其他的州没有问题,并且在俄亥俄州的非主要公路上也没有问题。在经过了大量的研究之后,他们才发现,相对于俄亥俄州高速公路巡警的新无线电系统所使用的频率而言,凯迪拉克的线路系统正好是一根极好的天线。

无线电波的特性与频率有关。在低频部分,无线电波能够很好地穿透障碍物,但是随着离开源越来越远,能量急剧减少,在空气中大致上是  $1/r^2$ 。在高频部分,无线电波倾向于按直线传播,并且会受到障碍物的阻挡。而且,无线电波也会被雨水吸收。在所有的频率上,无线电波都会受到发动机和其他电子设备的干扰。

由于无线电能够传播很远的距离,所以用户之间的干扰是一个问题。由于这个原因,所有的政府都严格管制无线电发射器的用途,唯独有一个例外,下面将会讲到。

在 VLF、LF 和 MF 波段,无线电波沿着地面传输,如图 2.12(a)所示。在较低频率上,可以在 1000 公里内检测到这些电波,在较高频率上距离范围要小一些。AM 无线电广播使用了 MF 波段,这就是为什么在纽约不太容易得到来自波士顿 AM 电台的地面波。在这些波段中的无线电波可以很容易地穿透建筑物,这也是为什么便携式收音机可以在室内使用的原因。使用这些波段进行数据通信的主要问题是它们的带宽太低(看一看等式(2-3))。

在 HF 和 VHF 波段,地面波会被地球吸收。然而,到达电离层(位于地球 100~500 公里高空处的一层带电粒子)的波可以被电离层折射回来,再返回到地球上,如图 2.12(b)所示。在某些特定的大气条件下,信号可以被反弹多次。业余无线电爱好者可以使用这些波段来进行长距离通话。军队也使用 HF 和 VHF 波段进行通信。



图 2.12  
(a) 在 VLF、LF 和 MF 波段,无线电波沿着地面传播;  
(b) 在 HF 波段,无线电波被电离层折射回来

### 2.3.3 微波传输

在 100MHz 以上的频段内,电波几乎按直线传播,因此它们可以被聚集成窄窄的一束。通过抛物线形状的天线(就像常见的碟形卫星电视天线),可以把所有的能量集中于一小束,从而获得极高的信噪比,但是发射端和接收端的天线必须精确地相互对齐。而且,这种方向性也允许多个排成一行的发射器与多个排成一行的接收器进行通信,只要它们的空间排布有规则,相互之间就不会干扰。在光纤出现之前的几十年时间中,这种微波构成了长途电话传输系统的核心。实际上,在 AT&T 解除管制之后,它的第一批竞争者之一 MCI 公司就使用微波通信技术建立了它的整个系统,它建造了很多微波塔,每个相距几十 km。并且该公司的名字也反映了这一点(MCI 代表了 Microwave Communications, Inc.)。后来 MCI 改成使用光纤,并且跟 WorldCom 公司合并了。



由于微波按照直线传播,所以,如果两个塔相距太远,那么地球本身就会阻挡传输路径(想象一下从旧金山到阿姆斯特丹的路径)。因此,中间每隔一段距离就需要一个中继器。塔越高,则微波能走的距离越远。中继器之间的距离大致上与塔高的平方根成正比。如果塔高为 100m,则中继器之间的距离可以为 80km。

与低频无线电波不同的是,微波并不能够很好地穿透建筑物。而且,即使微波在发射器处已经聚集起来了,但是在空中仍然会有一些发散。有些微波可能会被低处的大气层折射回来,因而比直线传播的微波所走的距离长一些。迟到的微波与直接到达的微波可能会不同相,因而抵消了信号。这种效果称为**多路衰减(multipath fading)**,它往往是一个很严重的问题。它与天气和频率有关。有些运营商保持信道的 10% 是空闲的,以便当多路衰减现象使某些频段临时失效时可以切换过来使用。

对于频段越来越多的需求驱使运营商们开始向高频发展。一直到 10GHz 的频段现在都已经在使用了,但是在 4GHz 左右,一个新的问题出现了:微波被水吸收。微波只有几个厘米的波长,因而会被雨水吸收。如果有人打算建造一个巨大的室外微波炉,用于烘烤飞过的小鸟的话,也许这种吸收效果还不错,但是对于通信来说,这是一个很严重的问题。与多路衰减现象一样,惟一的解决方案是停止使用这些受雨水影响的链路,并且在传输的时候避开这些链路。

总之,微波通信被广泛应用于长途电话通信、移动电话、电视转播,以及其他出现频谱严重短缺的应用领域。它比光纤有几个重要的优点。最主要的优点是不需要路权(right of way),任何人只要每 50km 买一小块地,并且建造一个微波塔,就可以绕过电话系统,直接进行通信了。这就是 MCI 公司如此迅速地崛起为一个新的长途电话公司所采取的做法(Sprint 公司走了另外一条截然不同的道路:它是由南太平洋铁路公司组建的,而该公司已经拥有了大量的路权,所以它只要沿着铁路铺设光纤就可以了)。

而且微波相对比较廉价。建造两个简单的塔(可能只是一根大的柱于再加上四根固定用的绳索)并且在每个塔上架设天线,有可能比在拥挤的都市或者山上铺设 50km 的光纤要便宜得多,也可能比租用电话公司的光纤便宜,特别是当电话公司还没有完全付清新投入的光纤(以代替铜线)开销的时候。

### 电磁频谱的政策

为了避免混乱,关于谁可以使用哪一段频率,有许多国家的和国际的规定。因为每个人都希望更高的数据传输率,所以每个人都想要更多的频段。国家政府为以下用途分配频谱:AM 和 FM 无线电台、电视、移动电话,也为以下的组织和机构分配频谱:电话公司、警察、海军、航空、军队、政府和许多其他的竞争用户。ITU-R 的一个全球性的代理(WARC)试图协调这些分配方案,以便厂商能够制造出可以在多个国家使用的无线电设备。然而,多数国家并不受 ITU-R 建议的约束,例如,为美国执行频谱分配的是 FCC(Federal Communication Commission,联邦通信委员会),它有时候就会拒绝 ITU-R 的建议(通常是因为这些建议要求某个政治上比较强大的组织放弃某一段频谱)。

即使当某一频段已经被分配给某一种用途(比如移动电话),仍然会有另外一些问题涉及到哪个承运商允许使用哪些频率。过去有三个做法被广泛使用过。最早的做法,通

常称为**选美比赛法**(beauty contest),要求每个承运商解释为什么它的提案能够给大众谋取最多的利益。然后由政府官员来决定他们最欣赏谁的故事。让某一个政府官员将价值几十亿美元的财产授予他最喜欢的公司,这通常会导致行贿受贿、腐败、袒护裙带关系,等等。而且,即使对于一个非常认真、诚实的政府官员,如果他认为一家外国公司比国内任何一家公司都能够出色地完成任任务的话,他也需要做很多的解释工作。

这种现象导致了第二个做法:让所有感兴趣的公司抓阄。这种做法的问题在于,对使用频谱没有兴趣的公司也可以参与抓阄活动。比如说,一家快餐连锁店或者鞋帽连锁店抓到了阄,那么它就可以将频谱再转卖给某一个承运商,对于它来说,没有任何风险,但是可以获得巨额利润。

将大笔财产给予并非精心挑选而是有一定随机性的公司,这种做法招来了各界严重的批评,因而导致了第三个做法:将带宽拍卖给出价最高的竞标者。2000年,英国拍卖了第三代移动系统所需要的频率,他们原来期望得到40亿美元,但是,他们实际收到了400亿美元,因为承运商们非常狂热,你死我活地唯恐错失进入移动通信领域的良机。这次事件勾起了其他国家政府的欲望,也激发他们组织类似的拍卖活动。这个做法非常有效,但是也给有些承运商留下了太多的债务,以至于他们面临破产的境地。即使在最好的情况下,也要经营很多年才能补偿这笔巨大的许可费。

另一种完全不同的分配频率的做法是根本不分配频率。允许任何人随意地传输数据,但是对所用的功率进行管制,使得发射台只有很短的距离,因而不会相互干扰。因此,大多数政府都已经留出了一些频段,称为**ISM**(Industrial, Scientific, Medical,工业的、科学的、医药的),用于非授权用途。车库门控制器、无绳电话、无线电控制的玩具、无线鼠标,以及许多其他的无线家用设备都使用ISM频段。为了使这些未经协调的设备之间的干扰尽可能地小,FCC强制所有在ISM频段上工作的设备都必须使用扩频技术。其他一些国家也有类似的规定。

对于不同的国家,ISM频段的位置也有所不同。例如,在美国,功率小于1瓦的设备可以使用图2.13所示的频段而无需FCC的许可。900MHz频段工作得最好,但是太嘈杂了,在全球都不可使用。2.4GHz频段在大多数国家都可以使用,但是会受到微波炉和雷达设备的干扰。蓝牙和有些802.11无线LAN工作在这个频段上。5.7GHz频段目前还比较新,相对而言还没有被开发出来,所以使用这个频段的设备比较贵,但是由于802.11a用到了这个频段,所以,很快它也会变得流行起来。



图 2.13 美国的 ISM 频段

#### 2.3.4 红外线和毫米波

无导向的红外线和毫米波被广泛应用于短距离通信。电视机、录像机和立体声音响上使用的遥控器都用到了红外线通信。相对来说,它们有方向性、便宜、易于制造,但是它们也有一个很大的缺点:它们不能通过固体物质(你可以试着站在遥控器和电视机之间,看一看遥控器是否还能工作)。一般而言,当从长波无线电向着可见光变化的时候,波的行为越来越像可见光,同时也越来越不像无线电波了。

另一方面,红外线不能很好地透过实体墙壁,这也是一个优势。它意味着,一个建筑物的某个房间中的一个红外系统不会干扰相邻房间或者相邻建筑物内的另一个类似的系统,比如,你不可能用你的遥控器去控制邻居家的电视机。而且,红外系统的防窃听安全性比无线电系统好,其原因就在于此。因此,经营红外系统并不需要政府的许可,相反,对于无线电系统,如果它在 ISM 频段以外工作,则必须要获得政府许可。红外通信在桌面环境中也有一定的用途,例如,将笔记本电脑与打印机连接起来,但是在整个通信领域中,它并没有重要的地位。

#### 2.3.5 光波传输

无导向的光信号已经被使用几个世纪了。Paul Revere 在他著名的骑马事件之前,就使用了从老北教堂(Old North Church)发出的二进制光信号。一个现代一点的应用是,将两个建筑物内的 LAN,通过屋顶上安装的激光连接起来。激光的光信号当然是单向的,所以每个建筑物都需要安装自己的激光发生器和光检测器。这种方案提供了非常高的带宽,成本也很低。并且,它也相对容易安装,而且与微波不同,它不要求 FCC 许可。

激光的强度(非常窄的一束光)也是它的弱点。将一束 1mm 宽的激光,瞄准 500 米开外的一个针头大小的目标,这要求像近代(19 世纪)Annie Oakley 那样的枪法。通常,在系统中放置一个透镜可以让激光束轻微地散开。

激光束不能够穿透雨或者浓雾,这是激光的一个缺点。但是,在晴朗的日子里,它们可以工作得很好。然而,作者有一次参加了一个在欧洲的某个现代化宾馆里召开的会议,会议的组织者提供了一个满是终端设备的房间,以便与会者在无聊的报告期间可以阅读自己的电子邮件。由于本地的邮政部门不愿意仅仅为了 3 天的会议而安装大量的电话线,所以组织者在屋顶上放了一个激光设备,并使其对准几公里开外该大学的计算机科学楼。在会议前一天晚上,他们测试了,它能够正常工作。第二天上午九点,天气很晴朗,但是连接完全不通,而且一整天都不能正常工作。那天晚上,组织者再次非常小心地进行了测试,它又一次非常正常地工作了。在随后的两天时间里,这种现象又发生了很多次。

会议结束后,组织者发现了问题所在。白天,太阳的热量使得气流从建筑物的房顶上升,如图 2.14 所示。这些扰乱的气流使得激光束有了偏差,从而总是在检测器的附近晃动。与此相似的大气现象可以使星星闪烁(这也是为什么天文学家喜欢把望远镜放在山顶上的原因——尽可能地在大气上面)。这也是热天时候公路闪闪发光,以及热辐射体看起来波动不稳的原因。

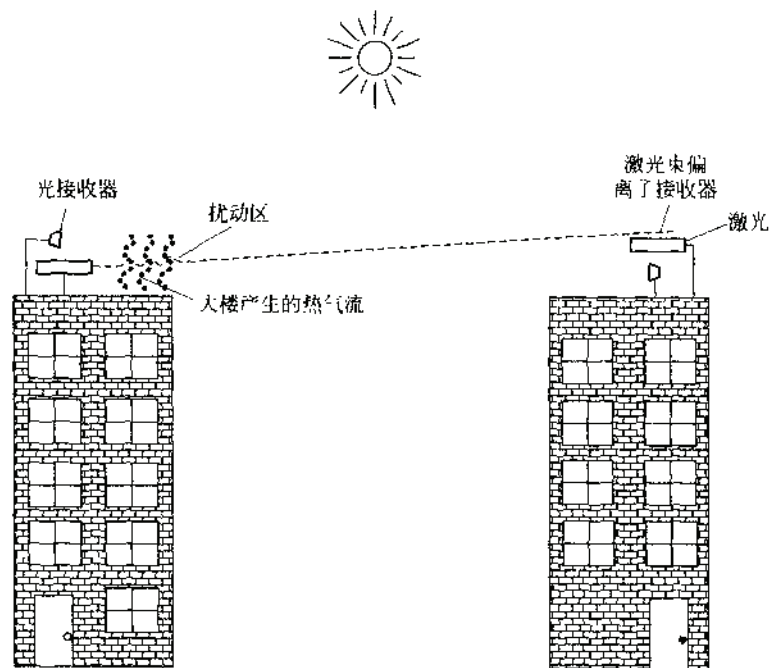


图 2.14 气流可以干扰激光通信系统。这里显示了一个具有两个激光源的双向系统

## 2.4 通信卫星

早在 20 世纪 50 年代和 60 年代初期,人们尝试着利用金属化的气象气球对信号的反射作用来建立通信系统。不幸的是,接收回来的信号太弱,所以没有任何实际用途。后来,美国的海军注意到了天空中一种永久的气象气球——月球,并且通过月球对信号的反射作用建立了一个可实际运行的船-岸通信系统。

一直等到第一颗通信卫星发射上天了,天体通信领域才有了进一步发展。人造卫星和实际卫星之间的区别是,人造卫星在送回信号之前先对它们进行了放大,从而将一种好奇心变成了一个强大的通信系统。

通信卫星有一些令人感兴趣的特性,这些特性对于许多应用很有吸引力。按照最简单的理解方式,你可以将一个通信卫星想象成天空中一个大的微波中继器。它包含几个异频发射应答器(transponder),每一个异频发射应答器都在监听频谱的某一部分,对进来的信号进行放大,然后在另一个频率上将放大后的信号重新发射出去,这样做可以避免与进来的信号相互干扰。向下的波束可以很宽,从而覆盖地球表面相当大一部分;也可以很窄,从而仅覆盖几百公里直径的区域。这种操作模式称为弯曲的管道(bent pipe)。

根据开普勒原理,一颗卫星的轨道周期随着轨道半径之  $3/2$  次幂而变化。卫星越高,则周期越长。在接近地球表面的地方,周期大约为 90 分钟。因此,低轨道的卫星很快就会移出人们的视线,所以,需要很多这样的卫星才能提供连续的覆盖区域。在高度大约为

35 800km 的轨道上,周期为 24 小时。在高度为 384 000km 的轨道上,周期大约是 1 个月,你只要观察一下月亮的运行规律就可以证实这一点。

卫星的周期非常重要,但它并不是确定卫星安放位置的惟一因素。另一个因素是范艾伦带的位置,所谓范艾伦带,是指受地球磁场影响的一些高度带电的粒子层。任何飞进了范艾伦带中的卫星都会很快被毁坏,这是因为受到地球磁场的影响,这些高能带电粒子摧毁了卫星。将这些因素综合起来,可以算出有三个区域对于卫星是安全的。图 2.16 列出了这三个区域以及它们的一些特性。下面我们简要地介绍一下每个区域中的卫星情况。

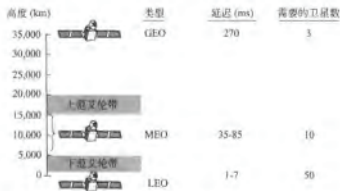


图 2.16 通信卫星和它们的一些特性,包括离地球的高度、上下往返的延迟时间、为覆盖整个地球所需要的卫星数

#### 2.4.1 地球同步卫星

1945 年,科幻小说作家 Arthui C. Clarke 计算出,在赤道轨道上方 35 800 公里高度处的卫星可以保持在天空中静止不动,所以这样的卫星不需要考虑跟踪的问题(Clarke, 1945)。它继续描述了一个完全的通信系统,该系统利用了这些(有人驾驶的)地球同步卫星(geostationary satellite),包括轨道、太阳能电池板、无线电频率以及发射程序。然而不幸的是,他得出的结论是,这样的卫星是不切实际的,因为他认为不可能在轨道中放上耗电的、易碎的真空管放大器,所以他后来不得进一步深入考虑这个想法,但是,他写了一些有关这种卫星的科幻小说。

晶体管的发明改变了这一切。1962 年,第一颗人造通信卫星(Telstar)被发射上天。自那以后,通信卫星变成了一宗几十亿美元的买卖,太空也变得非常有利可图了。这些高空飞行的卫星通常称为 GEO(Geostationary Earth Orbit,地球同步轨道)卫星。

在当前的技术水平下,在 360°的赤道平面内,同步卫星之间的距离小于 2°是不明智的,这样考虑是为了避免干扰。按照 2 度的空间间隔,在太空中同时最多只能有 360/2=180 颗同步卫星。然而,每个异频发射应答器可以使用多个频率和极性以便提高可利用的带宽。

为了避免太空中出现混乱,轨道槽的分配工作是由 ITU 来完成的。这个过程是高度



政治化的,因为任何一个国家(只要它已经度过了石器时代)都希望拥有自己的轨道槽(它可以将轨道槽租赁给出价最高的国家)。然而,没有分配到轨道槽的国家则主张,国家的产权不应该扩展到月球上,对于它的领土上空的轨道槽,没有一个国家拥有合法的权利。另一方面,商业通信并不是同步卫星的惟一应用,电视广播公司、政府和军队都希望从轨道蛋糕中分得一块。

现代卫星可以非常大,重量达到 4000kg,消耗几千瓦的电能,这些电能由太阳能电池板产生。太阳、月亮以及行星重力的影响,都要将卫星从它们预定的轨道槽和方向上移动到其他的位置,这样的影响需要通过卫星上的火箭发动机来消除。这种微调行为称为保持站位(station keeping)。然而,当发动机的燃料用完了的时候(通常 10 年左右),卫星将会漂流,甚至开始翻滚乱动,所以它必须要被关闭。最后,卫星的轨道就会衰减,它又重新进入太空,或者被烧掉,偶尔还会撞击到地球上。

轨道槽并不是争抢的惟一焦点。频率也是争抢的资源,因为下行传输会干扰原有的微波用户。因此,ITU 已经给卫星用户分配了特定的频率。其中主要的频率如图 2.16 所示。C 频段最早被用于商业卫星流量,在这个频段中目前已经分配了两个频率范围,其中较低的频率范围用于下行流量(从卫星发出),较高的频率范围用于上行流量(发向卫星)。为了能够同时在这两个方向上传输流量,要求使用两个信道,每个方向各一个信道。这些频段已经非常拥挤了,因为它们也被许多公共承运商用做地面微波链路。L 和 S 频段则是在 2000 年的时候根据国际协议加入进来的,但是,这两个频段都很窄,也很拥挤。

频段	下行链路	上行链路	带宽	问题
L	1.5 GHz	1.6 GHz	15 MHz	低带宽,拥挤
S	1.9 GHz	2.2 GHz	70 MHz	低带宽,拥挤
C	4.0 GHz	6.0 GHz	500 MHz	地面干扰
Ku	11 GHz	14 GHz	500 MHz	雨水
Ka	20 GHz	30 GHz	3500 MHz	雨水,设备成本

图 2.16 基本的卫星频段

接下去可供商业电信承运商使用的频段是 Ku(K under)频段。该频段目前并不拥挤,在这些频率上,卫星的空间间隔可以近到 1 度。然而,另一个问题出来了:雨水。对于这些短的微波而言,水是最佳的吸收体。幸运的是,大雨通常是局部地区的,所以,使用几个相距范围较远的地面站而不是一个地面站就可以解决这个问题,但是需要增加投入,包括额外的天线、电缆,以及用来在地面站之间快速切换的电子设备。Ka(K above)频段也已经被分配用于商业卫星流量,但是,为了使用该频段而需要配置的设备仍然非常昂贵。除了这些商业频段以外,还有许多政府和军队使用的频段。

一颗现代卫星大约有 40 个异频发射应答器(transponder),每个有 80MHz 带宽。通常,每个异频发射应答器就像一个弯曲管道(bent pipe)一样工作,但是,最新的卫星具备了一定的处理能力,这就使得它可以执行一些复杂的操作。在最早的卫星上,异频发射应答器之间的信道划分是静态的:整个带宽被简单地分成固定的频段。现在,每个异频

发射应答器的波束被分成多个时槽,许多用户可以轮流使用这些时槽。我们将在本章后面详细地学习两种这样的技术(即频分多路复用和时分多路复用)。

第一颗地球同步卫星只有一个空间波束,它大约可以覆盖地球表面的  $1/3$ ,这个范围称为它的**足迹(footprint)**。随着价格、尺寸、微电子功耗的不断下降,同步卫星已经有可能使用更为复杂的广播策略了。每颗同步卫星都装配了多个天线和多个异频发射应答器。每个下行波束可以聚集到很小的地理区域中,所以,多个上行和下行传输可以同时进行。通常情况下,这些所谓的**点波束(spot beam)**是椭圆形状的,它可以小到直径只有几百公里。美国的通信卫星往往用一个宽的波束覆盖 48 个相互连接的州,而为阿拉斯加和夏威夷使用点波束。

通信卫星领域中一个新的发展是低成本的微型站,有时候也称为 **VSAT(Very Small Aperture Terminals, 小孔终端)**(见 Abramson, 2000)。这些微型的终端有一个 1m 或者更小的天线(相比之下,标准的 GEO 天线有 10m),消耗的功率 1W 左右。上行链路可以达到 19.2kbps,质量还非常好,下行链路通常可以超过 512kbps。直播卫星电视可以用这项技术来实现单向传输。

在许多 VSAT 系统中,微型站没有足够的功率来实现相互之间的直接通信(当然是通过卫星)。相反,一种特殊的地面站(称为 **hub(中心站)**),具有很大的高增益天线)可用于转发 VSAT 之间的流量,如图 2.17 所示。在这种操作模式中,不管是发送方,还是接收方,都有大的天线和强大的放大器。这是一种折衷,用较长的延迟来换取廉价的最终用户站。

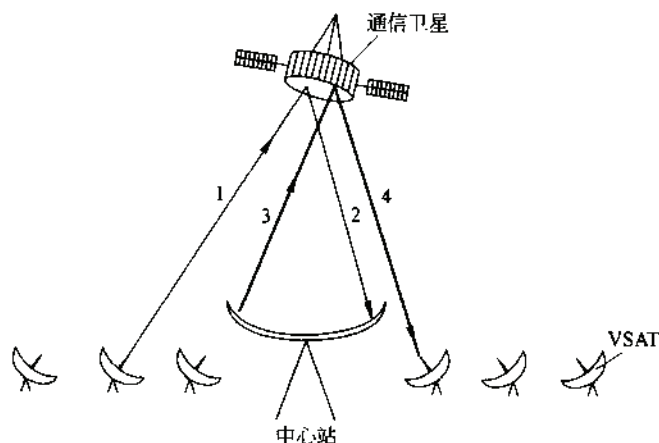


图 2.17 使用中心站的 VSAT

VSAT 在农村地区有较大的应用潜力。它现在并没有受到广泛的欢迎,但是,世界上有一半以上人口居住的地点离最近的电话有 1 小时以上的行走路程。将几千个小村庄用电话线连接起来,所需的费用远远超出了大多数第三世界政府的预算,但是安装 1 米 VSAT 碟形天线,并且用太阳能电池来供电,这往往是可行的。VSAT 提供的技术可以将整个世界连接起来。

通信卫星具备的一些特性与地面上的点到点链路有本质的不同。首先,即使信号从

地面到卫星的往返两程都是以光速(差不多是  $300\,000\text{km/s}$ )来传播的,但是,对于 GEO 卫星来说,较长的距离仍然造成了相当的延迟。根据用户与地面站之间距离的不同,以及卫星的海拔高度,端到端的传输时间大约在  $250\sim 300\text{ms}$  之间。一个典型的值是  $270\text{ms}$  (对于使用了中心站的 VSAT 系统来说,为  $540\text{ms}$ )。

从比较的角度来看,地面微波链路的传输延迟大致上是  $3\mu\text{s/km}$ ;同轴电缆和光纤链路的传输延迟大约是  $5\mu\text{s/km}$ 。后者之所以比前者慢,是因为电磁信号在空气中传输比在固体材料中传输更快。

卫星的另一个重要特性是:它天生就是一种广播介质。它给异频发射应答器的足迹范围内的上千个站发送一条消息,与发送给一个站所需要的成本是一样的。对于有些应用,这种特性非常有用。例如,你可以想象这样的情景:一颗卫星将许多流行的 Web 页面广播给遍布在广阔范围内的大量计算机。尽管这种广播也可以用点到点的线路来模拟,但是,显然卫星广播更加便宜。另一方面,从安全和隐私的角度来看,卫星完全是一个灾难:任何人都可以听到所有的一切。当需要安全性的时候,加密是必需的。

卫星还有另外一种特性:传输一条消息的成本与所经过的距离无关。跨越海洋的会话服务并不比跨越街道的会话更贵。卫星也有极佳的错误率,并且部署起来极为快速,这是用于军事通信中最主要的考虑因素。

#### 2.4.2 中间轨道卫星

在低一点的高度上,即两条范艾伦带之间,我们可以找到 MEO (Medium-Earth Orbit, 中间地球轨道)卫星。如果站在地球上来看,这些卫星慢慢漂移过经线,比如说经过 6 小时绕地球一圈。因此,当它们绕过天空的时候,它们必须被跟踪住轨迹。因为它们比 GEO 低,所以它们在地面上的足迹要小一些,因而功率弱一些的发射器也能够到达这些卫星。现在这些卫星并没有被用于通信领域,所以我们这里不再进一步介绍。作为 MEO 卫星的例子,24 颗 GPS (Global Positioning System, 全球定位系统)卫星的轨道大约在  $18\,000$  公里的高度上。

#### 2.4.3 低轨道卫星

再转移到低一点的高度上,我们就可以发现 LEO (Low-Earth Orbit, 低地球轨道)卫星。由于它们的运动速度极快,所以,一个完整的系统需要大量的 LEO 卫星。另一方面,因为这些卫星与地球相距如此之近,所以地面站并不需要多大的功率,而且,上下往返两程的延迟往往只有几个毫秒。这一小节我们将介绍三个例子:两个针对语音通信,一个针对 Internet 服务。

##### Iridium (铱计划)

正如前面所提到的,在卫星时代的前面 30 年,人们很少使用低轨道卫星,因为这些卫星刚刚进入视野,马上就又离开了。1990 年, Motorola 打破了这种局面,它向 FCC 提交了一份申请,要求许可发射 77 颗低轨道卫星用于“铱计划”(Iridium project, 77 号元素是铱)。该计划后来被修订为只使用 66 颗卫星,所以按理说它应该被更名为 Dysprosium

(铱,66号元素),但是它的发音听起来有点像一种疾病。基本的思路是,一旦一颗卫星移出了视线,另一颗卫星立即就能替代它。这份提议在其他通信公司之中引起了共愤。很快地,每一家公司都想要发射一系列低轨道卫星。

经过为期7年与合作伙伴的修复以及财政准备之后,于1997年Iridium卫星发射上天了。1998年11月开始提供通信服务。不幸的是,自从1990以后移动电话网络发展迅猛,相反,对于大的、重量级的卫星电话的商业需求却被忽略了。因此,Iridium并没有获得利益,并且于1999年8月被迫宣布破产,这是历史上一次很悲壮的法人惨败事件。这些卫星以及其他的财产(价值50亿美元)后来按照外太空旧货的形式,被卖给了一家投资商,实价为25 000 000美元。2001年3月Iridium的服务又重新开始。

Iridium以前和现在的业务都是提供全球性的电信服务,让用户通过手持设备就可以直接与Iridium卫星进行通信。它提供语音、数据、寻呼、传真和导航服务,允许用户在陆地、海洋和空中任何地方使用这些服务。Iridium的客户包括海军、航空和石油开采工业,以及在地球上某些缺少电信基础设施的地区(沙漠、高山、丛林和某些第三世界国家)旅行的人们。

Iridium卫星位于高度为750km处的圆形极地轨道上。它们被排列成6条南北方向的项链,每32纬度一颗卫星。通过6条卫星项链,整个地球被覆盖满了,如图2.18(a)所建议的那样。对化学知识不太了解的人,可以将这种排列想象成一个非常非常大的铱原子,其中地球为原子核,周围的卫星为电子。

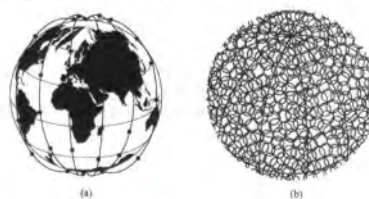


图 2.18

(a) Iridium 的卫星构成了 6 条项链,覆盖了整个地球; (b) 1628 个移动的地面覆盖了地球

每颗卫星最多有 48 个单元格(点波束),在地球表面上总共有 1628 个单元格,如图 2.18(b)所示。每颗卫星具有 3840 个信道的能力,总共有 253 440 个信道。其中有些信道用于寻呼和航海,其他的用于数据和语音。

Iridium 有一个很有趣的特性,那就是,相距较远的客户之间的通信在太空中进行,一颗卫星将数据转发给邻近的另一颗卫星,如图 2.19(a)所示。从这里我们可以看到,北极的一个呼叫者可以直接与一颗卫星联络;通过其他的卫星,他的呼叫请求最终被转发到位于南极的被呼叫者。

### Globalstar

除了 Iridium 的设计,另外一种设计是 Globalstar。Globalstar 系统有 48 颗 LEO 卫星,但是使用了不同于 Iridium 的交换方案。由于 Iridium 系统中需要在卫星之间转发数据,所以在卫星上要有复杂的交换设备,而 Globalstar 则使用了一种传统的弯曲管道(bent-pipe)的设计。如图 2.19(b),在北极发出的呼叫信号被送回到地球上,并且被 Santa 工场的大型地面站捕获到。然后,通过一个地面网络该呼叫被路由到另外一个地面站(位于被呼叫者的附近),并且通过一个弯曲管道连接交给被呼叫者,如图中所示。这种方案的好处是,它把大量的复杂性放到了地面上,这种复杂性在地面上相对比较容易管理。而且,使用大型地面站天线也有好处,它可以发出强的信号、接收弱的信号,这意味着,低功耗的电话也可以使用了。毕竟,电话发出的信号只有几个毫瓦的功率,所以回到地面站的信号非常弱,即使被卫星放大之后仍然会很弱。

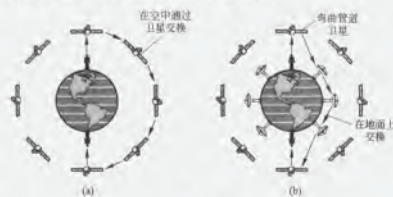


图 2.19  
(a) 在太空中转发信号; (b) 在地面上转发信号

### Teledesic

Iridium 针对的目标是偏远地区的电话用户。下一个例子 Teledesic 的目标则是遍布全球的、对带宽的需求非常大的 Internet 用户。1990 年,移动电话的先驱者 Craig McCaw 和 Microsoft 的创始人 Bill Gates 就有了 Teledesic 的想法。当时, Bill Gates 对于电话公司为计算机用户提供高带宽的迟缓步伐非常不满。Teledesic 系统的目标是,为成千上百万的并发用户提供上行链路可达 100Mbps、下行链路可达 720Mbps 的带宽,用户使用一个小的、固定的、VSAT 类型的天线,从而完全绕过电话系统。对于电话公司来说,这等于是一张空头支票。

最初的设计是, Teledesic 系统包含 288 个小卫星,它们排列成 12 个平面,位于 1350km 高空处,正好在下范艾伦带的下面。后来设计方案又改成 30 颗足速大一点的卫星。数据传输则使用相对不拥挤,但带宽比较高的 Ka 频段。该系统在空中是分组交换的,每颗卫星都有能力将分组路由到相邻的卫星上。当一个用户需要带宽来发送分组



的时候,系统在大约 50ms 以内接受请求,并完成动态分配。如果一切按计划进行的话,该系统将于 2005 年启用。

#### 2.4.4 卫星和光纤

对卫星通信和地面通信作一番比较是非常有意义的。在 20 年前,人们还认为未来的通信将建立在通信卫星的基础上。毕竟,在过去的 100 年中,电话系统的变化非常小,而且当时没有任何迹象表明在接下去的 100 年内将会有大的变化。这种迟缓的发展态势很大程度上是由于环境造成的:人们期望电话公司以合理的价格提供良好的语音服务(电话公司基本上也做到了),然后电话公司可以保证获得投资的收益。对于想传输数据的用户,他们可以使用 1200bps 的调制解调器。这一切都显得非常不错。

1984 年,在美国以及稍后在欧洲引入的竞争急剧地改变了这一切。电话公司开始用光纤代替其长途电话网络,并且引入了诸如 ADSL(Asymmetric Digital Subscriber Line,非对称数字用户线路)这样的高带宽服务。他们还停止了长期以来用高价长途话费贴补本地服务的做法。

突然之间,地面的光纤连接似乎是长期的赢家了。然而,通信卫星又拥有某些光纤所不涉及(有时候是不可达)的市场领域。现在我们来考察一些这样的领域。

首先,虽然在原理上仅仅一根光纤的带宽就有可能比所有已经发射的卫星的带宽还要多,但是,这种带宽对于大多数用户来说是不可用的。现在已安装的光纤基本上都用于电话系统内部,用来同时处理多个长途电话呼叫,而不是为单独的用户提供高带宽服务。而通过卫星,用户完全可以在建筑物的屋顶上立一根天线,绕过电话系统就可以得到很高的带宽。Teledesic 就是以这种想法为基础的。

第二个市场领域是移动通信。现在很多人都希望在跑步、驾车、航行和飞行时能够继续通信。地面光纤链路对于他们没有任何用处,但是卫星链路有这样的潜力。然而,有可能将蜂窝电话和光纤结合起来,这样就可以为绝大多数用户提供足够的通信服务了(但是对于在空中或者海上的人来说可能就不行了)。

第三个领域是本质上的传输模式为广播的场合。卫星发送的消息可以同时被上千个地面站接收到。例如,如果一个组织要将大量的股票、债券、商品价格的信息发送给几千个经销商,则通过卫星系统来发送这些信息可能比在地面上模拟广播更加便宜。

第四个领域是位于恶劣地形或者地面设施建设很差的地区的通信。例如,印度尼西亚有自己的卫星用于国内电话通信。发射一颗卫星的成本比在这个群岛的 13 677 个岛屿之间架设几千条海底电缆要便宜得多。

卫星的第五个市场是指那些很难获得路权来铺设光纤的地区,或者获得路权的代价太高的地区。

第六,当快速部署网络显得非常重要的时候,例如战争期间的军事通信系统,卫星通信显然会胜过光纤地面通信。

总而言之,看起来将来的主流通信是地面光纤与蜂窝无线电通信的结合,但是对于某些特殊的用途,卫星通信更有优势。然而,有一个原则对于所有这些通信方式都适用:经济学。虽然光纤提供了更多的带宽,但是,地面通信和卫星通信在价格上展开激烈的竞

争,这完全是有可能的。如果技术的发展使得部署一颗卫星的成本(比如,将来的某一种航天飞机一次可以放置几十颗卫星)大大下降的话,或者低轨道卫星很好地抓住了市场机遇的话,则光纤未必在所有的市场中都会赢。

## 2.5 公共交换电话网络

如果同一公司或者组织的两台计算机相距很近,现在它们需要进行通信,则最容易的做法是,在它们之间连接一条电缆。LAN 就是这样工作的。然而,当距离很远,或者有很多台计算机,或者这条电缆必须要经过一条公共的道路或者其他公共场所的时候,铺设私有电缆的费用往往是不切实际的。而且,几乎在所有的国家中,在公共地产上架设(或者在地下铺设)传输线路也是非法的。因此,网络设计者必须利用已有的电信设施。

这些设施,特别是公共交换电话网络(Public Switched Telephone Network, PSTN),通常是很多年以前就已经设计好的,它们以一种或多或少可以识别的形式来传输人的语音信号。它们用于计算机之间的通信最多也只是勉强够格而已;但是由于引入了光纤和数字技术,这种情况很快发生了变化。无论如何,电话系统跟计算机网络总是紧密联系在一起的,这值得我们花一些时间来仔细研究电话网络。

为了看清楚这个问题的重要性,我们来粗略地比较两种连接的特性:一是计算机与计算机之间通过一条电缆连接起来;二是通过拨号电话线连接起来。两台计算机之间的电缆可以以  $10^9$  bps 的速率(甚至可能更高)来传输数据。而拨号线路的最大数据传输率为 56 kbps,两者相差了将近 20,000 倍。这种区别就好比一只鸭子慢悠悠地经过一片草地与火箭飞到月亮上的区别一样。如果拨号线路替换成 ADSL 连接,则仍然有 1000~2000 倍的差距。

当然,麻烦在子,计算机系统的用户习惯于与计算机系统打交道,当他们突然面临另一个系统,并且性能差了 3 个或 4 个数量级的时候,他们就会花费大量的时间和精力,试图弄明白如何更有效地使用该系统,这是毫不奇怪的。在本节中,我们将介绍电话系统,并且说明它是如何工作的。要想了解更多关于电话系统内部结构的信息,请参考 Bellamy, 2000。

### 2.5.1 电话系统的结构

当贝尔(Alexander Graham Bell)于 1876 年获得了电话专利(仅比其竞争对手 Elisha Gray 早了几个小时)之后不久,他的新发明就有了大量的需求。最初的市场是电话销售,当时电话还是成对的;然后由顾客自己在一对电话之间拉上一条线。电子穿越地球再返回。如果一个电话所有者想跟其他  $n$  个电话所有者通话,则他必须拉  $n$  根单独的电话线到他们的家里。在一年之内,城市里布满了电话线,这些电话线通过房屋和树木,整个乱七八糟的。很明显,如图 2.20(a)所示,将每部电话与其他每部电话直接全连接的模式,是不可行的。

值得赞扬的是,贝尔看到了这一点,他组建了贝尔电话公司,并且于 1878 年开放了它的第一个交换局(在康涅狄格州(New Haven)城市)。该公司为每个客户的家里或者办

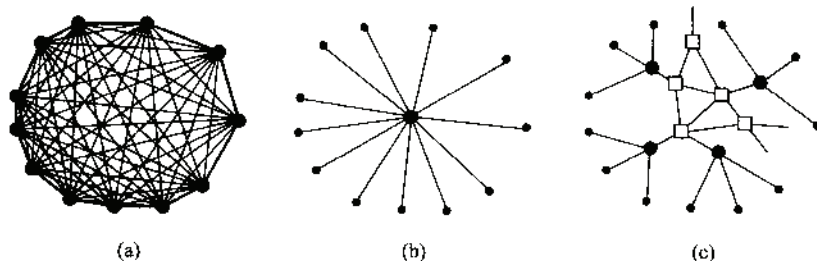


图 2.20

(a) 完全互连的网络; (b) 中心化的交换; (c) 两级层次结构

公室架一条电话线。当客户打电话的时候,首先摇动手柄,以便电话公司办公室里发出铃声,从而引起接线员的注意,然后接线员手工将呼叫方与被呼叫方通过一段跳接用的电缆连接起来。简单交换局的模式如图 2.20(b)所示。

很快地,贝尔系统的交换局遍布到了各地,人们又要求能够在不同城市之间打长途电话,所以贝尔系统开始将交换局连接起来。最初的问题很快又出现了:将每个交换局与其他每个交换局直接通过电话线连接起来,很快整个结构就变得无法管理了,所以就发明了二级交换局来解决这个问题。过了一段时间以后,多个二级交换局出现了,如图 2.20(c)所示。最终,整个层次结构增长到 5 级。

到 1890 年的时候,电话系统的三个主要部分已经具备了:交换局、顾客与交换局之间的线路(现在使用的是平衡型的绝缘双绞线,而原来使用的裸线),交换局之间的长距离连接。虽然从那以后,所有这三个领域都有了改进,但是基本的贝尔系统模型保持了 100 多年没有变化。有关电话系统技术的简单历史,请看 Hawley, 1991。

在 1984 年 AT&T 公司被分解之前,电话系统被组织成一个高度冗余的、多层次的结构。下面的描述是高度简化的,但是说明了其本质的特色所在。每部电话机有两根铜线伸出来,直接连接到电话公司最近的端局(end office,也叫本地中心局,local central office)。这段距离通常是 1~10km,城市比农村要短一些。在美国,大约有 22 000 个端局。每个用户的电话机和端局之间的双线连接在电话行业中称为本地回路(local loop)。如果世界上所有的本地回路都端到端连接起来,则其长度相当于到月球来回 1000 次的长度。

曾经有一段时间,AT&T 公司资产的 80% 是本地回路中的铜。当时,AT&T 公司实际上是世界上最大的铜矿。幸运的是,这样的事实并没有在投资圈里广为人知。如果被别人知道的话,有的市场捣乱者可能会买下 AT&T 公司,停止全美国的所有电话服务,并把所有的电话线收回来,卖给炼铜商,以获得快速的回报。

如果连接到某个端局的用户呼叫另一个也连接到该端局的用户,则局内的交换机制会在这两个本地回路之间建立起一个直接的电连接。在通话过程中,这个连接保持不变。

如果被呼叫的电话连接到另一个端局,则建立连接的过程就不同了。每个端局都有一些外线连接到一个或者多个附近的交换中心,这些交换中心称为长途局(toll office),如果它们在同一个局部地区的话,则称为汇接局(tandem office)。这些线路称为长途连接

干线(toll connection trunk)。如果呼叫方和被呼叫方的端局碰巧都有一条长途干线连接到同一个长途局(如果他们相距很近的话,则这种可能性很大),那么双方的连接就可能在这个长途局内建立起来。图 2.20(c)显示了一个只包含电话(小点表示)、端局(大点表示)和长途局(方块表示)的电话网络。

如果呼叫方和被呼叫方没有共同的长途局,那么他们之间的路径将在更高层次上的某个地方建立起来。初级局(primary office)、区域局(sectional office)和地区局(regional office)形成了一个网络,长途局连接到这一网络中。长途局、初级局、区域局和地区局通过高带宽的内部长途中继线(intertoll trunk,或者 interoffice trunk)相互交换信息。不同种类交换中心的数目以及它们的拓扑结构(两个区域局之间可以有一条直接连接,或者必须经过一个地区局)随着国家的不同而不同,这取决于每个国家的电话的密度。图 2.21 显示了一个中距离连接可能经过的路径。

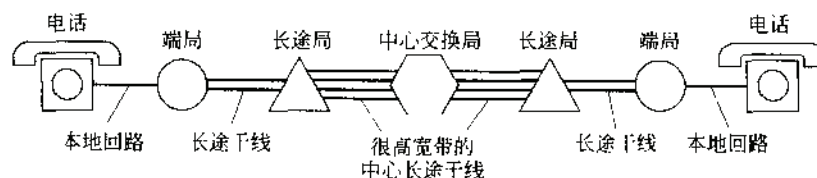


图 2.21 一个中距离电话的典型路径

电话系统中用到了各种传输介质。现在,本地回路由 3 类双绞线构成,而在电话业的早期,电话线杆上两条相距 25cm 的未绝缘的线是很常见的。在交换局之间,同轴电缆、微波,特别是光纤都被广泛使用了。

在过去,整个电话系统中传输的信号都是模拟的,实际的语音信号以电压的形式从源端传输到目标端。随着光纤、数字电路和计算机的出现,现在所有的干线和交换设备都是数字的,只有本地回路仍然是模拟的,它也是整个系统中最后保留使用模拟技术的地方。之所以优先选择数字传输,是因为它不需要像模拟传输那样,当一个长途呼叫的信号经过了许多个放大器之后,还需要重新精确地产生模拟波信号。对于数字传输而言,只需要能够正确地区分 0 和 1 就足够了。这种特性使得数字传输比模拟传输更加可靠。而且,维护工作更加容易,并且维护成本也要便宜得多。

总而言之,电话系统是由三个主要的部件构成的:

- (1) 本地回路(双绞线进入家庭和业务部门,模拟信号);
- (2) 干线(通过光纤将交换局连接起来,数字信号);
- (3) 交换局(电话呼叫从一条干线接入到另一条干线)。

这里稍微离题,介绍一下电话政治学之后,我们将回到这三个主要部件上来,详细地讨论每一个部件。本地回路使得每一个人都可以访问整个电话系统,所以它们是非常重要的。不幸的是,它们也是整个系统中最为脆弱的链路。对于长途干线,主要的问题是如何将多个呼叫收集起来,并且通过同一条光纤将它们发送出去。这项技术称为多路复用,我们将学习三种不同的实现方法。最后,我们将讨论两种基本的但完全不同的交换方法。

### 2.5.2 电话业中的政治学

在1984年之前的几十年中,贝尔系统在全美大多数地区既提供本地电话服务,也提供长途电话服务。在20世纪70年代,美国联邦政府认为这是非法的垄断,并且提起诉讼要将它分解掉。最后政府赢了,1984年1月1日,AT&T公司被分解为AT&T长话公司(AT&T Long Lines)、23个BOC(Bell Operating Company, Bell运行公司)和其他一些部门。23个BOC又被组织成7个区域性BOC(RBOC),以便使它们在经济上可以继续生存下去。美国电信业的整体本质在一夜之间被法官的判决(而不是国会的法案)改变了。

一份名为MFJ(Modified Final Judgement,修订的最终判决书,这个名字本身就是一个矛盾,如果判决书可以被修改的话,它就不是最终的)的最终判决书描述了这次分解案的确切细节。这次事件导致了更为激烈的竞争、更好的服务,以及更低的长途电话价格(无论对个人还是对企业)。然而,本地服务的价格反而上升了,因为来自长途电话服务的补偿费用没有了,所以本地服务必须要自己来承担所有的费用。许多国家现在也在考虑引入类似的竞争办法。

为了明确谁可以提供哪些服务,整个美国被分成164个LATA(Local Access and Transport Areas,本地访问和传输区域)。粗略地说,一个LATA是指可以由一个地区码来覆盖的大区域。在每个LATA的内部,只有一个LEC(Local Exchange Carrier,本地交换承运商),它垄断了该区域内部的传统电话服务。尽管有的LATA包含了一个或者多个电话公司(美国有1500个独立的电话公司以LEC的身份来运行),但是最重要的LEC都是BOC。

LATA之间的所有流量是由另一种不同类型的公司IXC(IntereXchange Carrier,跨区域交换承运商)来处理的。最初,AT&T长话公司是惟一正式的IXC,但是现在WorldCom和Sprint公司也是组建良好的IXC业务竞争者。在分解AT&T公司时非常关注的一个问题是,确保所有的IXC在线路质量、价格,以及顾客为了使用IXC而拨打的电话号码位数等方面,都能够被同等对待。IXC的运行方式如图2.22所示。这里我们可以看到三个LATA的例子,每个LATA有几个端局。LATA2和LATA3还有一个内含汇接局(即LATA内部的长途局)的小层次结构。

任何一个IXC,如果它想处理来自某个LATA的呼叫,则可以在这个LATA中建立一个称为POP(Point of Presence,汇接点)的交换局。LEC负责将每个IXC连接到每个端局,可以是直接的连接,比如像LATA1和LATA3中那样;也可以是间接的连接,比如像LATA2中那样。而且,这种连接关系,无论从技术方面还是经济方面来看,对所有的IXC都必须一视同仁。通过这种方式,LATA1中的电话用户可以选择任何一个IXC以呼叫LATA3中的用户。

作为MFJ的一部分,IXC被禁止提供本地电话服务,而LEC则被禁止提供跨LATA的电话服务,但是两者在其他所有业务范围内都不受限制,比如经营炸鸡餐馆。在1984年,这是一份毫无歧义的声明。不幸的是,技术的发展以一种非常有趣的方式使法律变得过时了。这份协定没有考虑到有线电视和移动电话。当有线电视从一路变成两路,移动



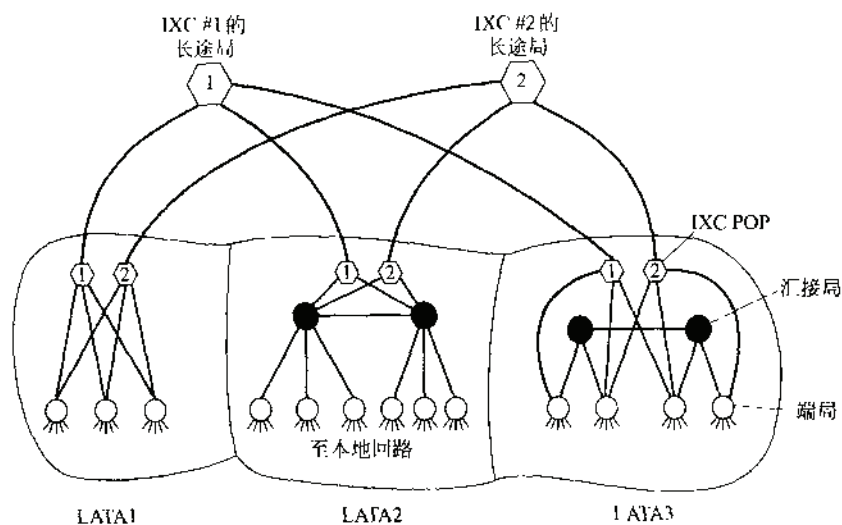


图 2.22 LATA、LEC 和 IXC 之间的关系

所有的圆圈都是 LEC 交换局,每个六边形都属于某一个 IXC,内部的数字标明了该 IXC 的号码

电话在人群中流行起来的时候,LEC 和 IXC 开始购买或者兼并有有线电视和移动电话运营商。

到了 1995 年的时候,美国国会看到,试图维护各种公司之间的业务界限已经不再可行,于是拟订了一份草案,允许有线电视公司、本地电话公司、长途电话承运商和移动电话运营商可以进入彼此的业务领域。其思路是,以后任何公司都可以为它的顾客提供全套的综合服务,包括有线电视、电话和信息服务,并且不同公司在服务和价格上进行竞争。这份草案于 1996 年 2 月正式被批准。最终的结果是,有些 BOC 变成了 IXC,而其他有些公司,比如有线电视运营商,则开始提供本地电话服务,与 LEC 进行竞争。

关于 1996 法案,一个有趣的特点是,它要求 LEC 实现本地电话号码的可移动性。这意味着,一个顾客可以在不改变本地电话号码的情况下,改换一家本地电话公司。这种灵活性消除了很多人的疑虑,使得他们更加倾向于改变 LEC,从而加剧了竞争。其结果是,美国电信业正在进行激烈的重组。而且,其他许多国家也开始仿效。通常其他的国家会等待一段时间,看一看这种实验在美国的效果怎么样。如果效果好的话,他们就会跟着仿照;如果效果不好,他们可以试一试其他的做法。

### 2.5.3 本地回路:调制解调器、ADSL 和无线

现在我们开始详细地讨论电话系统是如何工作的。电话系统的主要部分如图 2.23 所示。这里我们可以看到本地回路、干线,以及长途局和端局,这两种局都包含了用于切换电话呼叫的交换设备。一个端局可以有 10 000 条本地回路(在美国和大多数国家)。实际上,一直到最近以前,区域号码+局号代表了一个端局,所以,(212)601-xxxx 是一个专门的端局,它有 10 000 个电话用户,编号为从 0000~9999。随着本地电话服务竞争的出现,这样的结构不再可行,因为多家公司都希望拥有这一个端局号码。而且,端局号

码也快要用完了,所以有必要引入复杂的映射方案。

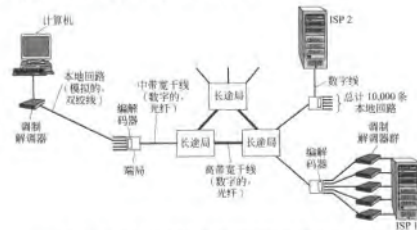


图 2.23 从计算机到计算机的呼叫同时利用了模拟和数字传输，在调制解调器和编解码器之间完成数模转换

我们先从大多数人都很熟悉的部分开始：从电话公司的端局到家庭或者小型业务部门的双线本地回路。本地回路也经常称为“最后一英里(last mile)”，但实际上它的长度可以达到几英里。在过去 100 多年中，它一直传输模拟信号，而且在接下来的几年中可能仍然如此，因为转换为数字信号传输的成本相对较高。然而，即使在这最后的模拟传输堡垒中，目前也正在发生变化。在这一小节中，我们将学习传统的本地回路，以及在本地回路中最新的发展情况，并将重点放在家庭计算机的数据通信上。

当一台计算机希望通过模拟拨号线路发送数字数据的时候，这些数据首先必须转换成模拟的形式，才能通过本地回路进行传输。这个转换过程是通过一种称为调制解调器(modem)的设备来完成的。稍后我们将讨论这种设备。在电话公司的端局中，这些模拟数据又转换成数字形式，以便通过长途干线进行传输。

如果另一端也是一台带调制解调器的计算机，则必须进行相反的过程（从数字到模拟），以便通过目标方的本地回路。这个过程如图 2.23 的右下角所示，图中的 ISP 1 (Internet Service Provider, Internet 服务供应商) 有一群调制解调器，每个调制解调器分别连接到不同的本地回路上。一个 ISP 拥有多少个调制解调器，它就可以处理多少个连接（假设它的服务器或者服务器群有足够的计算能力）。在 56kbps 调制解调器出现之前，这种结构形式非常普遍，但是后来有了变化，稍后再解释其中的原因。

模拟信号是由一个随时间变化的电压构成的，电压的变化代表了一个信息流。如果传输介质非常完美无缺的话，则接收方收到的信号与发送方送出的信号应该完全相同。不幸的是，没有一种介质是完美无缺的，所以接收到的信号不可能与送出的信号完全一致。对于数字数据，信号之间的差异可能会导致错误。

传输线路存在三个主要的问题：衰减(attenuation)、延迟畸变(delay distortion)和噪声(noise)。衰减是指在信号往外传播的过程中能量的损失。这种损失可以表示为每公里多少分贝。能量损失的数量跟频率有关。为了理解这种频率相依赖性的效果，请想象这

样一个信号,它并非是一个简单的波形,而是傅立叶级数中的一些分量。每个分量被衰减的程度不一样,因而导致了在接收方得到一个完全不同的傅立叶频谱。

更糟的是,不同的傅立叶分量在线路上的传播速度也不同。这种速度的差异会导致接收端信号的畸变。

另一个问题是噪声,它是指一些来自于非发射器的多余能量。热噪声是由线路中电子的随机运动而引起的,也是不可避免的。串音则是由于两根相距太近的线路的感应耦合而引起的。有时你在打电话时,可以听到后面还有另一个对话,这就是串音。最后,还有脉冲噪声,这是由于电线上的尖峰干扰或者其他原因引起的。对于数字数据,脉冲噪声可能会消除掉一位或者多位数据。

### 调制解调器

由于上面讨论到的一些问题,特别是,衰减和传播速度都是与频率有关的,所以,在信号中有一个较宽的频率范围是不现实的。不幸的是,在数字信号中用到的方波恰好有一个很宽的频谱,因而它的衰减和延迟畸变很严重。这种影响使得基带(DC)信号传输并不合适,除非是在低速和短距离的情况下使用。

为了绕过与DC信号相关的这些问题,特别是在电话线上使用的时候,我们采用了AC信号传输。首先引入1000~2000Hz范围内的一个连续波,称为正弦载波(sine wave carrier)。它的振幅、频率或者相位被调制后可用来传输信息。在调幅(amplitude modulation)方法中,两种不同的振幅用来分别代表0和1。在调频(frequency modulation,也称为frequency shift keying,移频键控)方法中,使用了两个(或多个)不同频率的连续波(这里术语keying(键控)在工业界也被广泛使用,当作modulation(调制)的同义词)。在形式最为简单的相位调制(phase modulation)中,载波按照统一的间隔,系统地平移0或者180度。一种更好的方案是使用45、135、225或者315相位角度,以便每个时间间隔传输2位信息。而且,这种方案的另一个好处是,在每一个时间间隔结束时总是要求进行一次相移,这使得接收方很容易识别时间间隔的边界。

图2.24显示了三种调制形式。在图2.24(b)中,一种振幅为非0值,另一种振幅为0值。在图2.24(c)中,用到了两种频率。在图2.24(d)中,在每个位边界上,或者出现相移,或者不出现相移。如果一个设备接受一个位序列作为输入,并且产生一个经过以上一种(或者多种)方法调制的载波输出(或者输入输出正好相反),则这样的设备称为调制解调器(modem,代表modulator-demodulator)。调制解调器位于在计算机(数字)和电话系统(模拟)之间。

为了获得更高的速度,仅仅依靠增加采样率是不可能的。尼奎斯特定理表明,即使对于完美的3000Hz传输线路(更何况拨号电话线路还不是完美的),采样率超过6000Hz也是没有意义的。在实践中,大多数调制解调器采样率为2400次/秒,它们的焦点集中在每次采样如何表达更多的位信息。

每秒钟采样的次数可以用波特(baud)来计量。在每一个波特中,发送一个码元(symbol)。因此,一条n波特的线路传输n码元/秒。例如,一条2400波特的线路大约每416.667 $\mu$ s发送一个码元。如果该码元用0伏电压代表逻辑0,1伏电压代表逻辑1,则位

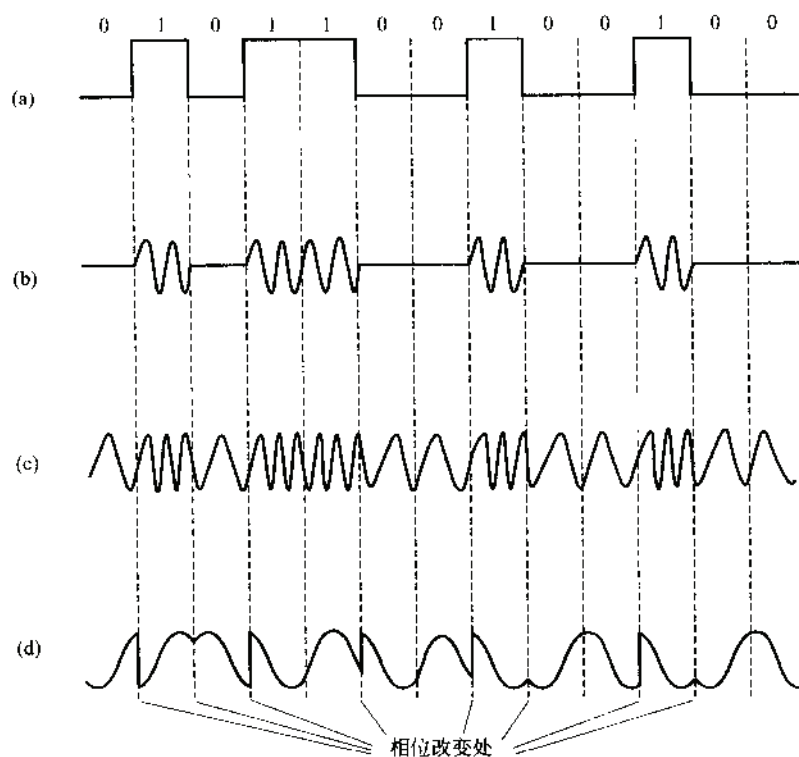


图 2.24

(a) 一个二进制信号; (b) 调幅; (c) 调频; (d) 相位调制

传输率为 2400bps。然而,如果 0、1、2 和 3 伏电压都使用的话,则每个码元可以包含 2 位,因此,2400 波特的线路可以传输 2400 码元/秒,数据传输率为 4800bps。类似地,如果使用四种可能的相移的话,则每个码元也可以表达 2 位,所以同样地,位传输率为波特率的两倍。后一种技术已广泛使用,称为 **QPSK (Quadrature Phase Shift Keying, 正交相移键控)**。

带宽、波特、码元和位传输率的概念通常很容易混淆,所以,这里我们重新声明一次。一种介质的带宽是指在最小衰减的情况下能够通过这种介质的频率范围。它是介质的一种物理特性(通常从 0 到某一个最大的频率),度量单位为 Hz(赫兹)。波特率是指每秒钟的采样次数。每个采样发送一份信息,这份信息称为码元。因此,波特率和码元率是相同的。调制技术(比如 QPSK)决定了每个采样的位数。位传输率是指一条信道上发送的信息的数量,它等于每秒采样数乘以每个采样的位数。

所有高级的调制解调器都组合使用了几种调制技术,以便在每个波特中传输尽可能多的位。通常,多个振幅和多个相移组合起来就可以传输几位/波特。在图 2.25(a),我们可以看到,在 45、135、225 和 315 度处的点,其振幅(即离开原点的距离)为常数。一个点的相移是由从该点到原点的直线与正向的 x 轴所构成的角度来决定的。图 2.25(a)有四种有效的组合,所以每个码元可以传输 2 位。这就是 QPSK。

在图 2.25(b)中,我们可以看到另一种不同的调制方案。这种方案用到了 4 种振幅和 4 种相移,总共有 16 种不同的组合。因此,利用这种调制方案,每个码元可以传输 4 位。它称为 **QAM-16**(**Quadrature Amplitude Modulation**,正交振幅调制)。有时候也使用术语 16-QAM 来代替。例如,在 2400 波特线路上,利用 QAM-16 可以达到传输率 9600bps。

图 2.25(c)是另一种调制方案,它也涉及到振幅和相位。它允许 64 种不同的组合,所以每个码元可以传输 6 位。它称为 **QAM-64**。也可以使用更高阶的 QAM。

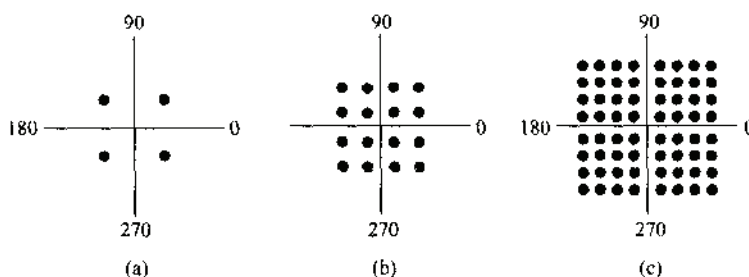


图 2.25

(a) QPSK; (b) QAM-16; (c) QAM-64

图 2.25 中的三个图形分别显示了振幅和相位的几种合法组合,这些图形称为**星座图**(**constellation diagram**)。每个高速调制解调器标准都有它自己的星座模型,并且只能与使用了同样星座模型的调制解调器才可以通话(不过,大多数调制解调器都可以模拟所有低速的调制解调器)。

由于星座模型中有许多点,所以,在检测到的振幅和相位中,即使很小量的噪声也可能导致一个错误,而且这个错误又可能隐含着许多错误的位。为了降低出错的可能性,高速调制解调器的标准都有纠错机制,其做法是在每个采样中增加一些额外的位。这样的方案称为 **TCM**(**Trellis Coded Modulation**,格子架编码调制)。举例来说,V.32 调制解调器标准使用了 32 个星座点,每个码元传输 4 个数据位和 1 个奇偶校验位,在 2400 波特上可以达到 9600bps(带纠错特性)。它的星座模型如图 2.26(a)所示。之所以绕着原点旋转 45 度,是出于工程的原因。旋转的和不旋转的星座模型具有同样的信息传输能力。

在 9600bps 以上,下一步是 14 400bps。它称为 **V.32 bis**。这个速度是这样得来的:在 2400 波特上,每个采样传输 6 个数据位和 1 个奇偶位。它的星座模型有 128 个点,如图 2.26(b)所示,它使用了 QAM-128。传真调制解调器就使用这种速度来传输扫描之后的页面位图(bit map)。QAM-256 并没有用于任何标准的电话调制解调器,但是用于电缆网络上,后面我们将会看到。

在 V.32 bis 之后,下一种电话调制解调器是 **V.34**,它在 2400 波特上,每个采样传输 12 个数据位,因此,数据传输率为 28.8kbps。在这个系列中,最后一种调制解调器是 **V.34 bis**,它在 2400 波特上,每个采样传输 14 个数据位,所以可达到 33.6kbps。

为了进一步提高实际的数据传输率,许多调制解调器在传输数据之前,先对数据进行压缩,这样得到的有效数据传输率可以高于 33.6kbps。另一方面,几乎所有的调制解调



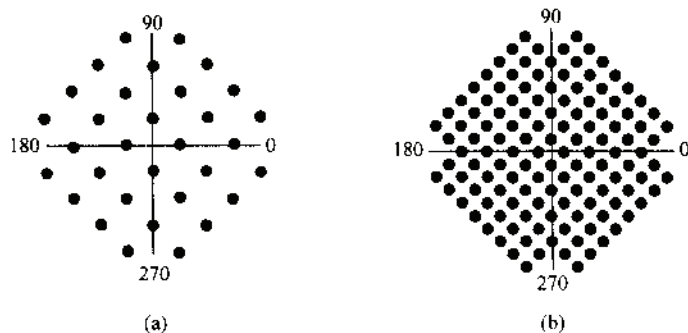


图 2.26

(a) 传输速率为 9600bps 的 V.32; (b) 传输速率为 14400bps 的 V.32 bis

器在开始传输用户数据之前,先对线路进行测试,如果发现质量不好的话,则退回到低一点的速率上(低于标定的最大速率)。因此,用户观察到的有效速率可能会低于、等于,或者高于正式标定的速率。

所有的调制解调器都允许同时两个方向上传输(做法是,在不同的方向上使用不同的频率)。允许在两个方向上同时进行数据传输的连接称为全双工的(full duplex)。一条双车道的公路是全双工的。如果一个连接允许任何一个方向的流量通过,但是同一时刻只有一个方向可以传输数据,则这个连接称为半双工的(half duplex)。单条铁路轨道是半双工的。如果一个连接只允许一个方向的数据通过,则称它为单工的(simplex)。单向的街道是单工的。单工连接的另一个例子是:一根光纤的一头是激光源,另一头是一个光检测器。

标准调制解调器为什么在 33.6kbps 这里停住了呢?其原因是,电话系统的香农限制大约是 35kbps,所以,比 35kbps 更快的速度会违反物理学(热力学部分)的法则。为了弄明白 56kbps 的调制解调器在理论上是否可能,请继续往下读。

但是,为什么会有 35kbps 的理论限制呢?它与本地回路的平均长度,以及这些线路的质量有关系。35kbps 是由本地回路的平均长度决定的。在图 2.23 中,发起于左侧计算机并终止于 ISP1 的呼叫要经过两个本地回路传输模拟信号,一个本地回路在源处,另一个本地回路在目标处。这两个本地回路都在信号中加入了噪声。如果我们可以从中消除掉一个本地回路,则最大速率就可以增加一倍。

ISP2 正是这样的情形。它有一条纯的数字线路,直接到达最近的端局。在干线上使用的数字信号直接被传送给 ISP2,从而消除了它这一端的“编解码器、调制解调器和模拟传输”的过程。因此,当连接的一端是数字信号传输的时候(现在大多数 ISP 都是这样的),则最大数据传输率可以达到 70kbps。如果两个家庭用户通过调制解调器和模拟线路来连接,则最大速率是 33.6kbps。

为什么使用 56kbps 调制解调器,其理由与尼奎斯特理论有关系。电话信道的带宽大约是 4000Hz(包括防护频段)。因此,每秒钟的最大独立采样次数为 8000。在美国,每个采样的位数是 8,其中 1 位被用于控制的用途,因而每秒钟允许传输用户数据 56000 位。

在欧洲,所有这 8 位数据都可以给用户使用,所以 64kbps 的调制解调器也已经在使用了,但是,为了有一个国际性的标准约定,故选择了 56 000。

这种调制解调器的标准称为 **V.90**。它提供了 33.6kbps 的上行信道(从用户到 ISP)和 56kbps 的下行信道(从 ISP 到用户),因为通常情况下,从 ISP 到用户方向上传输的数据比另一个方向上的数据多(比如,请求一个 Web 页面只需少量的字节,而实际的页面可能有几兆字节数据)。从理论上讲,上行信道宽于 33.6kbps 也是可能的,但是对于许多本地回路,即使在 33.6kbps 上也有太多噪声,所以,最后决定给下行信道多分配一点带宽,以便增加下行信道真正在 56kbps 上工作的几率。

V.90 之后的下一步是 **V.92**。对于这样的调制解调器,如果线路允许的话,它的上行信道可以达到 48kbps。老的调制解调器通常在前 30 秒内确定实际使用的速度,而这种调制解调器可以在大约一半的时间(15 秒)内决定将要使用的速度。最后,如果所用的线路具有呼叫等待服务的话,则这种调制解调器允许一个打进来的电话呼叫中断一个 Internet 会话。

#### 数字用户线路(Digital Subscriber Lines)

当电话工业最后达到了 56kbps 的时候,它可以歇下来松一口气了,因为一项任务已经出色地完成了。同时,有线电视工业正在提供高达 10Mbps 的速度(在共享电缆上),卫星公司正在计划提供上行 50Mbps 的速度。Internet 访问已经成为他们的业务中日益重要的一部分,这时,电话公司(LEC)开始意识到,他们需要一种更具竞争力的产品。他们给出的答案是:利用本地回路提供新的数字服务。如果一项服务所提供的带宽比标准的电话服务还要多的话,则有时候这样的服务称为**宽带(broadband)**,不过,这个术语的市场概念实际上超过了特定的技术概念。

开始的时候,有许多相互重叠的服务都使用了 **xDSL(Digital Subscriber Line, 数字用户线路)**这样的一般性名称,只是 x 不同而已。下面我们将讨论这些服务,但是主要焦点集中在一种可能会变得最为流行的服务上:**ADSL(Asymmetric DSL, 非对称数字用户线路)**。由于 ADSL 仍然在开发之中,并不是所有的标准都已经出台,所以,下面给出的有些细节可能将来会有变化,但是基本的框架应该仍然有效。有关 ADSL 的更多信息,请参考(Summers, 1999; and Vetter et al., 2000)。

调制解调器之所以如此慢的原因是,电话的发明是为了承载人类的语音,而且整个电话系统也为了这个目的而被小心地优化过了。数据就好像不是自己亲生的子女,一直没有被认真地对待。每条本地回路都在端局那里被终止,就在这个终止的点上,有一个滤波器把所有 300Hz 以下、3400Hz 以上的频率都减弱了。截止的地方并不尖锐——300Hz 和 3400Hz 都是 3 分贝点——所以,尽管两个 3 分贝点之间的距离是 3100Hz,但是,这一段带宽也常常被当作 4000Hz 来引用。因此,数据也被限制在这一窄窄的频段之中。

使 xDSL 工作的诀窍是,当一名顾客预订了这项服务的时候,进来的线路被连接到另一种不同的交换机上,这种交换机上没有滤波器,因而可以充分发挥本地回路的全部承载能力。于是,限制的因素就变成了本地回路的物理特性,而不再是人工加入的滤波器所建立起来的 3100Hz 带宽限制。

不幸的是,本地回路的能力依赖于几方面的因素,包括它的长度、粗细和综合质量。距离与潜在带宽之间的函数关系如图 2.27 所示。该图假设所有其他的因素都是最佳的(新的线路、捆扎得也不错,等等)。

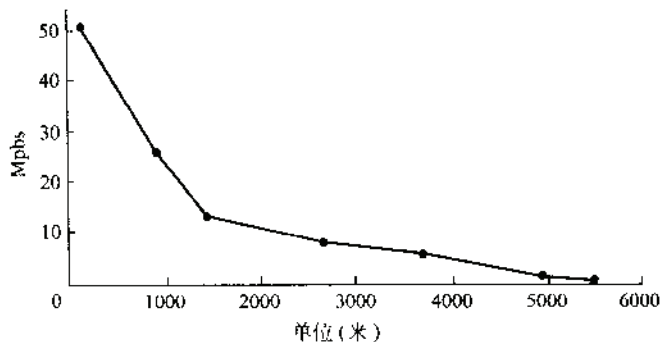


图 2.27 3 类 UTP 用于 DSL 时带宽与距离的关系

该图对于电话公司来说意味着一个问题。当电话公司承诺提供某一个速度的传输率的时候,它同时也划定了一个以端局为中心的圆,超出了这个圆就无法提供这样的服务。这意味着,当远距离的顾客试图申请这项服务的时候,电话公司告诉他们“谢谢您的惠顾,但是,因为您住的地方离最近的端局太远了,超出了我们预定距离 100 多米,所以我们无法提供服务。您考虑搬家吗?”公司选定的速度越低,则圆的半径越大,从而覆盖面越广。但是,速度越低,则这项服务的吸引力就越小,愿意付钱购买这项服务的人就越少。这种情况止所谓是业务碰上了技术的制约(一种可能的解决方案是,在离端局较远的地区建造一些微型的端局。但是,这种方案的成本很高)。

所有的 xDSL 服务都有一些特定的设计目标。第一,这些服务必须在现有的 3 类双绞线本地回路上工作。第二,它们不能影响顾客原来的电话机和传真机。第三,它们必须比 56kbps 还要快。第四,这些服务应该总是可用的,并且按月租方式收费,而不是按每分钟收费。

最初的 ADSL 服务是 AT&T 公司提供的,它的工作原理是,将本地回路上可供使用的频谱(大约是 1.1MHz)分成三个频段: POTS(Plain Old Telephone Service, 传统的简单电话服务)、上行数据流(从用户到端局)和下行数据流(从端局到用户)。使用多个频段范围的技术称为频分多路复用,我们将在本章后面学习这项技术。后来,其他的 ADSL 供应商采用了不同的方法,但是看起来这种频段划分的方法可能会获得成功,所以下面我们将介绍这种方法。

图 2.28 显示了一种称为 DMT(Discrete MultiTone, 离散的多信道调制)的频段划分方法。实际上,它的做法是将本地回路上的 1.1MHz 频谱分成 256 条独立的信道,每条信道 4312.5Hz。信道 0 用于 POTS。信道 1-5 没有被使用,目的是让语音信号与数据信号避免相互干扰。在剩下的 250 条信道中,一条用于上行控制,另一条用于下行控制,其他的信道全部用于用户数据。

从原理上讲,剩下的每条信道都可以被用作全双工数据流,但是谐波、串音和其他的

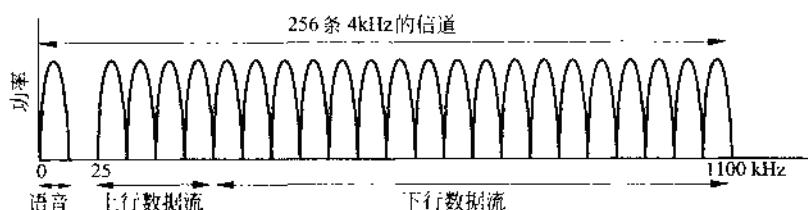


图 2.28 使用 DMT 的 ADSL 频谱划分方案

影响使得实际的系统性能大大低于理论限制。ADSL 供应商可以决定多少信道用于上行数据流,多少信道用于下行数据流。上行和下行各 50%的做法在技术上是可行的,但是大多数供应商倾向于将 80%~90%的带宽分配给下行信道,因为大多数用户的下载数据量超过上传数据量。这种选择正好暗示了 ADSL 中的第一个字母“A”(非对称)。一种常见的分法是,32 条信道用于上行数据流,其余的用于下行数据流。让最高端的少量上行信道成为双向信道,以便增加带宽,这样做也是可能的,但是,这种优化的代价是,要求增加一部分特殊的电路来消除回波。

ADSL 标准(ANSI T1.413 和 ITU G.992.1)允许的速度可以达到 8Mbps 下行速度和 1Mbps 上行速度。然而,很少有供应商提供这样的速度。通常情况下,供应商提供 512kbps 下行速度和 64kbps 上行速度(标准服务),以及 1Mbps 下行速度和 256kbps 上行速度(高级服务)。

每一条信道内部使用的调制方案类似于 V.34,但是采样率为 4000 波特,并非 2400 波特。系统一直监视每条信道中的线路质量,必要的时候会调整数据传输率,所以不同的信道可能会有不同的数据传输率。系统通过 QAM 调制方法发送实际的数据,每个波特可以达到 15 位,所使用的星座图类似于 2.25(b)。譬如,共有 224 条下行信道,每条信道工作在 4000 波特,并且每个波特传输 15 位数据,则下行数据流的带宽为 13.44Mbps。在实践中,信噪比不可能非常理想,所以,这样的速度实际上是达不到的,但是,在较短的高质量回路上,8Mbps 是有可能的(这也是标准中指定下行速度为 8Mbps 的原因)。

图 2.29 显示了一个典型的 ADSL 部署结构。在这种方案中,电话公司的技术人员必须在顾客的家里安装一个 NID(Network Interface Device,网络接口设备)。这个小的塑料盒代表了电话公司的财产到此为止,并且,从这里开始将是顾客的财产。在靠近 NID 的地方是一个分离器(splitter)(有时候,分离器也组合在 NID 中),分离器是一个模拟滤波器,它将 POTS 使用的 0~4000Hz 频段与数据分离开。POTS 信号被路由到原来的电话机或者传真机,而数据信号则被路由到 ADSL 调制解调器中。ADSL 调制解调器实际上是一个数字信号处理器,相当于一组 250 QAM 调制解调器,它们在不同的频率上并行地工作。由于当前大多数的 ADSL 调制解调器都是外置的,所以计算机与它必须通过高速方式连接起来。通常的做法是,在计算机中安装一块以太网卡,然后运行一个只包含两个节点(该计算机和 ADSL 调制解调器)的以太网。偶尔情况下,也可以使用 USB 端口来代替以太网。毫无疑问,在将来,内置的 ADSL 调制解调器也一定会出现。

在线路的另一头,即电话公司的端局,也要安装一个对应的分离器。信号的语音部分

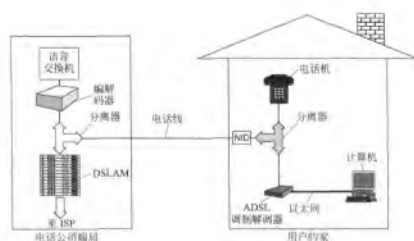


图 2.29 一个典型的 ADSL 设备配置图

在这里被过滤出来,并送到语音交换机中。频率在 26kHz 以上的信号则被路由到一种新的称为 DSLAM(Digital Subscriber Line Access Multiplexer,数字用户线路访问复用器)的设备中,该设备也包含一个数字信号处理器,与 ADSL 调制解调器中的一样。一旦数字信号被恢复成一个位流,就可以重新构造出分组来,然后送给 ISP。

由于语音系统和 ADSL 完全分离了,所以对于电话公司来说,部署 ADSL 相对比较容易。它所需要做的是购买一个 DSLAM 和分离器,并且将 ADSL 用户的线路接到分离器上。其他一些高带宽的服务(比如 ISDN)要求对现有的交换设备作更大的改动。

图 2.29 中的设计方案也有一个缺点,那就是,在顾客的家需要安装一个 NID 和分离器。这些设备只能由电话公司的技术人员来安装,因而就会招致昂贵的“流动服务”(即让技术人员上门提供服务)。因此,另外一种不需要分离器的设备也已经被标准化了。它的非正式名称为 G. lite,而 ITU 的标准号为 G. 992.2。它的部署结构与图 2.29 一样,只不过没有分离器。现有的电话线直接使用。唯一的区别在于,在电话或者 ADSL 调制解调器与电话线的每个插孔中插入了一个微滤波器。电话的微滤波器是一个低通滤波器,它消除了 3400Hz 以上的频率成分;ADSL 调制解调器的微滤波器是一个高通滤波器,它消除了 26kHz 以下的频率成分。然而,这个系统的可靠性不如带分离器的系统,所以 G. lite 用起来至多只能达到 1.5Mbps(相比之下,带分离器的 ADSL 可以达到 8Mbps)。然而,在端局中,G. lite 仍然要求有分离器,但是,这种部署方案并不需要大批的服务人员。

ADSL 只是一个物理层的标准。在它之上运行什么,这要取决于承运商。通常的选择是 ATM,因为 ATM 能够管理服务质量,而且许多电话公司的核心网络上也在运行 ATM。

#### 无线本地回路

自从 1996 年以后,在美国以及随后其他一些国家中,那些希望与传统的本地电话公



司(以前的垄断者)进行竞争的公司也允许开展本地电话服务,这样的公司称为 **ILEC** (**Incumbent LEC**, 准 **LEC**)。其中最有竞争力的公司是长途电话公司(**IXC**)。任何一家 **IXC** 如果想要介入某个城市的本地电话业务,则必须要做以下的事情。第一,购买或者租用一个建筑物作为它在该城市的第一个端局。第二,它必须在这个端局中安装电话交换机和其他的设备,而且,所有这些设备都必须是现货,它可以从各个销售商处提取。第三,它必须在该端局和最近的长途局之间经营一条光纤,以便新的本地顾客将来可以访问它的全国网络。第四,它必须招徕到新的顾客,通常的做法是,宣传自己比其他的 **ILEC** 有更好的服务和更低的价格。

然后,最艰难的工作开始了。假定有一些顾客真正加入进来了。新的本地电话公司(称为 **CLEC**(**Competitive LEC**, 竞争的 **LEC**))如何将顾客的电话机和计算机连接到新开张的端局呢? 购买必要的路权并且铺设电话线或者光纤可能会非常昂贵。许多 **CLEC** 已经发现了一种比传统的双绞线本地回路更加廉价的方案: **WLL**(**Wireless Local Loop**, 无线本地回路)。

从感觉上来看,使用无线本地回路的固定电话有点像一部移动电话,但是这里有三个关键的技术区别。第一,无线本地回路的消费者通常希望具备高速的 Internet 连接能力,其速度至少等于 **ADSL**。第二,新的顾客可能并不介意让 **CLEC** 的工程师在他家的屋顶上安装一个大的有向天线指向 **CLEC** 的端局。第三,用户并不移动,从而消除了因为移动和蜂窝之间移交(本章后面我们将会学习移动电话系统)而带来的所有问题。因此,一个新的工业诞生了: 固定的无线系统(**fixed wireless**, **CLEC** 利用无线本地回路提供的本地电话和 Internet 服务)。

尽管 **WLL** 正式开始运作是 1998 年,不过我们首先得回到 1969 年看一看它的来源。那一年, **FCC** 分配了两个电视频道(位于 2.1GHz, 每个频道各 6MHz)作为教育台。在随后多年中,又加入了 31 个频道,位于 2.5GHz, 总共 198MHz。

教育台一直也没有开播,到了 1998 年, **FCC** 收回了这些频率,并将它们分配给双路广播电台。这些频率很快就被用于无线本地回路了。在这些频率上,微波的波长是 10~12cm 长。它们的传输范围大约是 50km, 穿透植物和雨水的力量还算可以。198MHz 的新频谱立即也用于无线本地回路,所提供的服务名称为 **MMDS**(**Multichannel Multipoint Distribution Service**, 多信道多点分发服务)。 **MMDS** 可以被认为是一个 **MAN** (**Metropolitan Area Network**, 城域网),也可以认为是 **LMDS**(下面将要讨论)的一个孪生兄弟。

这种服务最大的好处是,技术已经非常成熟,并且设备也是现成的。缺点是,总共可用的带宽并不很富裕,所以,在相对较大的地理区域内,许多用户必须共享这一点带宽。

由于 **MMDS** 的带宽比较低,因此毫米波作为一种可考虑的替代技术引起了人们的兴趣。在美国, 28~31GHz(在欧洲是 40GHz)频率上的频段一直没有被分配,其原因是,要建立起工作速度如此快的硅集成电路是非常困难的。随着砷化镓集成电路的发明,这个问题已经被解决了,于是,毫米波用于无线通信的大门也随之被打开了。 **FCC** 对这种需求做出了响应,它将 1.3GHz 分配给一种新的无线本地回路服务,称为 **LMDS**(**Local Multipoint Distribution Service**, 本地多点分发服务)。这次 **FCC** 分配了大量的带宽,足够

任何人使用了。类似地,在欧洲也分配了这样的频段,但是在 40GHz 上。

LMDS 的运行如图 2.30 所示。这里显示的塔包含了多个天线,每个天线指向不同的方向。由于毫米波有很强的方向性,所以,每个天线定义了一个扇区,相互之间独立。在这个频率上,传输范围是 2~5km,这意味着,为了覆盖一个城市,需要很多这样的塔。

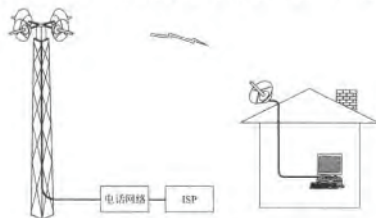


图 2.30 LMDS 系统的结构

如同 ADSL 一样,LMDS 也使用了非对称的带宽分配方案,下行信道的带宽多一些。利用当前的技术,每个扇区可以有 36Mbps 的下行数据流和 1Mbps 的上行数据流,该扇区中的所有用户共享这些带宽。如果每个活动用户每分钟下载 3 个 5KB 大小的页面,则该用户平均占用了整个带宽中的 2000bps,这使得每个扇区中可以容纳最多 18 000 个活动用户。为了使延迟相对比较合理,每个扇区支持的活动用户应该不超过 9000 个。如果像图 2.30 所示的那样有 4 个扇区,则可以支持 36 000 个活动用户。假定在高峰期,3 个顾客中有一个是在线的,则像这样包含 4 个天线的塔可以为方圆 5 公里以内的 10 万个顾客服务。许多潜在的 CLEC 已经计算过这些数字,其中有些得出了这样的结论:只要对毫米波的塔进行适当的投资,它们就可以进入到本地电话和 Internet 业务中,为用户提供可与有线电视相媲美的数据传输率,并且价格更低。

然而,LMDS 有一些问题。首先,毫米波是直线传播的,所以,在屋顶和塔之间必须有一条很清晰的视线。其次,树叶会吸收这些波,所以塔必须足够高,以避免视线中有树遮挡。在 12 月份看起来一条非常清晰的视线,到了 7 月份的时候,由于树上长满了树叶,可能视线就不再清晰了。雨水也会吸收这些波。在某种程度上,由于雨水引入的错误可以通过纠错码得到一定程度的补偿,或者在下雨的时候加大功率。然而,LMDS 服务极有可能首先会在气候干燥的地区使用起来,比如说在亚利桑那州,而不是西雅图。

除非无线本地回路有了标准,这样可以鼓励设备厂商生产有关的产品,并且确保顾客可以在不购买新设备的情况下改变 CLEC,否则的话无线本地回路不可能流行起来。为了提供这样的标准,IEEE 建立了一个名为 802.16 的委员会,由它为 LMDS 拟定一个标准。2002 年 4 月,802.16 已经发布了。IEEE 将 802.16 称为无线 MAN。

IEEE 802.16 设计的应用目标是:数字电话、Internet 访问,连接两个远距离的

LAN、电视和电台广播,以及其他的用途。我们将在第 4 章再详细地讨论 802.16。

#### 2.5.4 干线和多路复用

规模经济在电话系统中起着很重要的作用。在两个交换局之间安装和维护一条高带宽的干线,与安装和维护一条低带宽的干线基本上花费同样多的钱,因为这里的费用主要来自于挖电缆沟所需的开销,而不是铜线或者光纤。因此,电话公司已经开发出了许多精致的技术,用于在一条物理干线上尽可能地并发传输多个会话。这些多路复用方案可以分为两大类: FDM (Frequency Division Multiplexing, 频分多路复用) 和 TDM (Time Division Multiplexing, 时分多路复用)。在 FDM 中,频谱被分成频段,每个用户可以单独拥有某个频段。在 TDM 中,用户轮流(循环法)获得整个带宽,每次仅使用一小段时间。

利用 AM 电台广播可以说明这两种多路复用技术。AM 所分配的频谱大约是 1MHz,大致为 500~1500kHz。不同的频率被分配给不同的逻辑信道(广播台),每个广播台在频谱的某一部分进行工作。信道之间的间隔足够大,以避免相互干扰。这是一个频分多路复用的典型例子。而且(在有些国家),有的广播台有两个逻辑子信道,分别用于音乐和广告。这两个子信道按照时间间隔轮流使用同一个频率,首先是一阵音乐,然后是一阵广告,然后是一阵音乐,等等。这种情形是时分多路复用。

接下来我们将讨论频分多路复用。之后我们将看一看 FDM 如何应用到光纤上(波分多路复用)。然后,我们将转到 TDM 上,最后给出一个使用光纤的高级 TDM 系统(SONET)。

##### 频分多路复用

图 2.31 显示了三条语音级的电话信道通过 FDM 被复用在一起。滤波器将每条语音级信道的有用带宽限制在 3100Hz 左右。当许多信道被复用在一起的时候,每条信道分配 4000Hz,以便它们相互之间隔离开。首先,语音信道的频率被提升上来,每条信道提升的幅度各不相同。确保任何两条信道都不会占用频谱中相同的部分,然后将它们组合起来。请注意,即使在信道之间有了沟(防护频段),相邻信道之间也会有重叠,因为滤波器没有很尖锐(陡)的边界。这种重叠意味着,在一条信道上可以感觉到相邻信道边界上的尖峰脉冲(非热噪声)。

世界上使用的 FDM 方案在某种程度上已经被标准化了。一种广泛使用的标准是将 12 个 4000Hz 的语音信道复用到 60~108kHz 的频段中。这个单位称为一个群(group)。12~60kHz 之间的频段有时用于另一个群。许多承运商给顾客提供的 48~56kbps 的租用线路就是以群为基础的。5 个群(60 条语音信道)可以复用起来形成一个超级群(supergroup)。接下来的单位是主群(mastergroup),一个主群包含 5 个超级群(CCITT 标准)或者 10 个超级群(贝尔系统)。另外也还有多达 230 000 条语音信道的其他标准。

##### 波分多路复用

对于光纤信道,它使用的是频分多路复用的一个变种,称为 WDM (Wavelength Division Multiplexing, 波分多路复用)。光纤上 WDM 的基本原理如图 2.32 所示。这里,

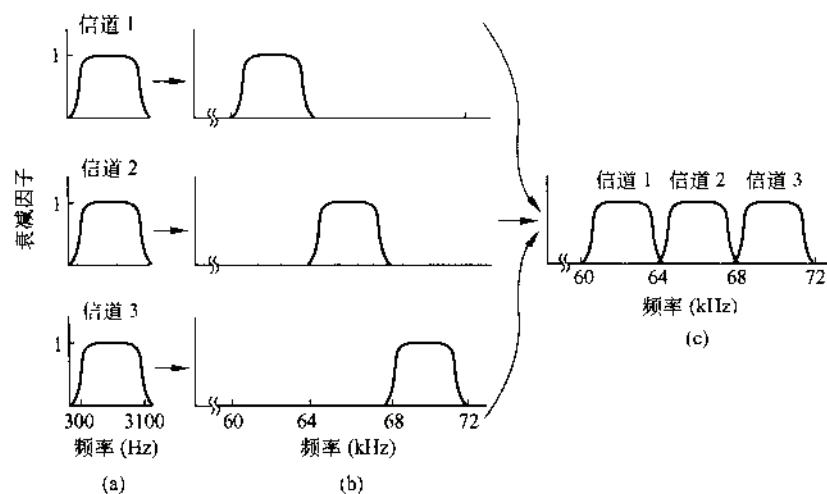


图 2.31 频分多路复用

(a) 原来的带宽; (b) 被升频之后的带宽; (c) 复用之后的信道

4 条光纤汇合到一个组合器 (combiner) 中, 每条光纤的能量位于不同的波长处。四束光被组合到一条共享的光纤上, 然后传输到远处的目标端。在远端, 这束光又被分离到与输入端一样多的 4 条光纤上。每条输出光纤包含一个短的、特殊结构的核, 该核能够过滤出某一种波长, 其他的波长全部被过滤掉。然后, 结果信号可以被路由到它们各自的目的地, 或者以其他不同的方式组合起来以便再次复用传输。

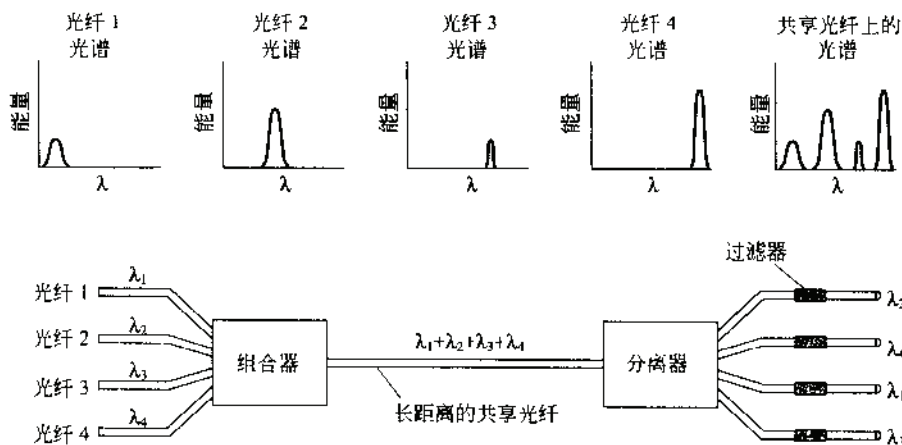


图 2.32 波分多路复用

这里并没有真正的新技术。这只不过是频分多路复用在极高频率上的应用而已。只要每条信道有它自己的频率 (也就是波长) 范围, 并且所有的频率范围都是分开的, 那么, 它们就可以被复用到长距离光纤上。与电子 FDM 之间的惟一区别是, 使用衍射光栅的光纤系统是完全被动的, 因此也极为可靠。

WDM 技术已经发展到一种让计算机技术望尘莫及的速度了。WDM 是在 1990 年

前后发明的。第一个商业系统有 8 条信道,每条信道的带宽为 2.5Gbps。到 1998 年的时候,市场上出现 40 条信道(每条信道的带宽为 2.5Gbps)的系统了。到 2001 年的时候,市场上有了 96 条信道(每条信道的带宽为 10Gbps,总共 960Gbps)的产品。这样的带宽足够每秒钟传输 30 部全长度的电影(MPEG-2 格式)。具有 200 条信道的系统在实验室中已经可以工作了。当信道的数目很大,并且波长的间隔非常接近(比如 0.1nm)的时候,这样的系统通常称为 **DWDM(Dense WDM,密集 WDM)**。

应该指出,WDM 之所以受欢迎的理由是,单根光纤上的能量通常只有几个 GHz 宽,因为目前在电介质和光介质之间不可能转换得更快了。通过在不同的波长上并行地运行多条信道,则聚合起来的带宽就会随着信道的数量而线性增加。由于单根光纤的带宽大约为 25 000GHz(见图 2.6),所以,从理论上算,即使在每赫兹 1 位的情况下(更高的速率也是可能的),仍然有 2500 条 10Gbps 信道的发展空间。

另一个新的发展是全光放大器。以前,每 100 公里就必须分离出所有的信道,并将每条信道转换为电信号,以便单独放大,然后重新转换为光信号,再将所有的信道组合起来。现在,全光放大器可以重新产生整个信号,每隔 1000 公里做一次,不需要再进行多次光-电转换了。

在图 2.32 所示的例子中,我们有一个固定的波长系统。从光纤 1 输入进来的位流输出到光纤 3,从光纤 2 输入进来的位流输出到光纤 1,等等。然而,我们也有可能建立一个具有交换功能的 WDM 系统。在这样的设备中,通过 Fabry-Perot 或者 Mach-Zehnder 干涉计可以对输出滤波器进行调节。有关 WDM 以及将 WDM 应用于 Internet 分组交换的更多信息,请参考(Elmirghani and Mouftah, 2000; Hunter and Andonovic, 2000; and Listani et al., 2001)。

### 时分多路复用

WDM 技术非常不错,但是在电话系统中仍然有大量的铜线存在,所以,我们现在转回头来看看这些铜线该怎么办。尽管 FDM 仍然被用于铜线或者微波信道,但是,它要求模拟电路,并且不适合由计算机来完成。相反,TDM 则可以完全由数字电路来处理,所以最近几年 TDM 已经变得非常广泛了。不幸的是,它只能用于数字数据。由于本地回路产生的是模拟信号,所以,在端局需要执行一个从模拟到数字的转换,而在端局中,所有的本地回路聚集到一起,并组合到少量的输出干线上。

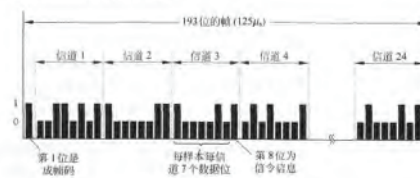
现在我们来查看多个模拟语音信号如何数字化,并组合到一条输出的数字干线上。通过调制解调器发送的计算机数据也是模拟的,所以下面的描述也适用于这些数据信号。模拟信号在端局中被数字化,这项工作是由一个称为**编解码器(codec, coder-decoder)**的设备来完成的,它产生一个 8 位数字序列。编解码器每秒钟采样 8000 次(每次采样 125 $\mu$ s),因为尼奎斯特定理表明,对于 4kHz 电话信道的带宽而言,这样的采样率足以捕获所有的信息。如果采样率再低的话,信息可能会丢失;如果采样率更高的话,也不会获得额外的信息。这项技术称为 **PCM(Pulse Code Modulation,脉冲编码调制)**。PCM 构成了现代电话系统的核心。因此,电话系统中几乎所有的时间间隔都是 125 $\mu$ s 的倍数。

当数字传输作为一种切实可行的技术开始出现的时候,CCITT 却未能为 PCM 达成



一个国际标准。因此,目前世界上不同的国家使用了各种互不兼容的编码方案。

北美和日本使用的方法是 **T1 线路(T1 carrier)**,如图 2.33 所示(从技术上而言,这种格式称为 DS1,而线路称为 T1,但是按照工业界的传统,我们这里不作细微的区分)。T1 线路包含 24 条被复用在一起的语音信道。通常,当结果模拟信号流送给编解码器时,编解码器以轮询的方式对它进行采样,所以,T1 线路并没有使用 24 个独立的编解码器,然后再将每个编解码器的数字输出进行合并。每一条信道按照顺序依次在输出流中插入 8 位,其中 7 位用于数据,1 位用于控制,所以每条信道会产生  $7 \times 8000 = 56\,000\text{bps}$  的数据,以及  $1 \times 8000 = 8000\text{bps}$  的信令信息。



每帧包含  $24 \times 8 = 192$  位,再加上额外一位用于成帧,因而每  $125\mu\text{s}$  产生 193 位数据。这样得到的数据传输率为  $1.544\text{Mbps}$ 。第 193 位用于帧同步。它使用的模式是 0101010101…。通常情况下,接收方必须不断地检查这一位,以确保不会失去同步。如果接收方失去了同步,则可以搜索这样的位模式以便重新获得同步。模拟客户根本不可能产生这样的位模式,因为它对应于  $4000\text{Hz}$  上的一个正弦波,正常情况下已经被过滤掉了。当然,数字客户可以产生这样的位模式,但是当失去帧同步的时候,出现这种位模式的情况很罕见。当一个 T1 系统被全部用于数据的时候,只有 23 条信道真正用于数据。第 24 条信道用于特定的同步模式,从而使得在失去帧同步的时候可以快速地恢复。

当 CCITT 最终达成一致的时候,他们觉得  $8000\text{bps}$  信令信息有点太多了,所以 CCITT 的  $1.544\text{Mbps}$  标准以 8 位数据项而不是 7 位数据项为基础。也就是说,模拟信号被量化成 256 级,而不是 128 级。因此就产生了两种互不兼容的编码调制方案。在公共信道信令(common-channel signaling)方案中,额外的那一位被附在 193 位帧的尾部而不是头部,在奇数帧中它的取值为 10101010…,而在偶数帧中则包含了针对所有信道的信令信息。

在另一个变种关联信道信令(channel-associated signaling)方案中,每条信道都有它自己私有的信令子信道。私有子信道是这样安排的:在每 6 帧中,将第 6 帧中的 8 个用户位之一分配为信令位,所以,在每 6 次采样中,有 5 次是 8 位宽度,余下一次只有 7 位宽度。CCITT 还建议了另一种称为 E1 的 PCM 线路,它的速度为  $2.048\text{Mbps}$ 。这种线路的  $125\mu\text{s}$  基本帧中有 32 个 8 位数据采样。其中 30 条信道用于数据信息,2 条信道用于

信令信息。4 帧为一组，共有 64 个信令位，其中一半用于与信道有关的信令，另一半用于帧同步，或者每个国家保留用于其他的目的。除了北美和日本以外的国家都使用 2.048Mbps 的 E1 线路而不是 T1。

一旦语音信号已经被数字化以后，系统试图用统计技术来减少每条信道所需要的位数。这些技术不仅仅只适用于编码语音信号，也可以用于其他任何模拟信号的数字化处理。所有的压缩方法依据的原理是，信号的变化相对于采样频率而言比较慢，所以 7 位或者 8 位数字级别中的大部分信息是多余的。

一种方法是差分脉冲编码调制(differential pulse code modulation)，它的输出并不是数字化的振幅本身，而是当前值与前一个值之差。因为在 128 的尺度上， $\pm 16$  或者更大的跳跃可能性不大，所以 5 位应该是够了，而不需要 7 位。如果信号偶尔有很大的跳跃，则编码逻辑部分可能需要几个采样周期才能“赶上”它。对于语音信号，这种因为偶尔的跳跃而引入的错误可以忽略。

这种压缩方法的一个变种要求每个采样值与前一个值相差  $\pm 1$  或者  $-1$ 。在这样的条件下，只需传输一位就可以表达新的采样值是在前一个采样值之上还是之下。这种技术称为增量调制(delta modulation)，如图 2.34 所示。像所有那些“假设相邻采样值之间有较小变化”的压缩方法一样，增量编码法对于信号变化太快的情形也难以处理，就如图中所示的那样。当这种情形发生的时候，信息就会丢失。

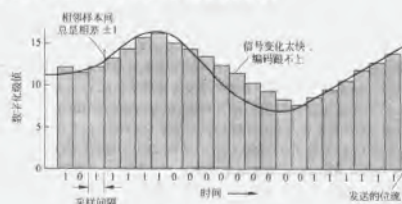


图 2.34 增量调制

对于差分 PCM 的一种改进是，根据前面少量的值来预测下一个值，然后对实际信号值与预测值之间的差值进行编码。当然，发送方和接收方必须使用同样的预测算法。这样的方案称为预测编码(predictive encoding)。这些方案非常有用，因为它们降低了需要编码的数值的范围，因而也降低了需要发送的位数。

时分多路复用允许多个 T1 线路被复用到一个更高级的线路中。图 2.35 显示了一种可能的做法。在左侧，我们看到 4 条 T1 信道被复用到一条 T2 信道中。T2 以及更高级的复用是按位完成的，而不是按字节完成的(但是，24 条语音信道构成一个 T1 帧是按字节完成的)。4 个 1.544Mbps 的 T1 流应该产生 6.176Mbps，但是 T2 实际上是 6.312Mbps。这些多出来的位用于成帧，或者当线路失去同步时用于恢复。T1 和 T3 主

要为顾客所使用,而 T2 和 T4 仅用于电话系统内部,所以它们较少为人所知。

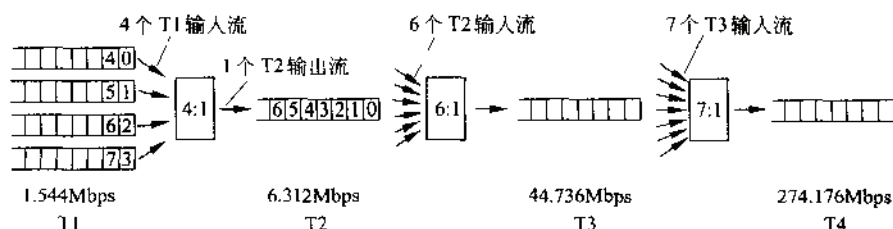


图 2.35 更高阶线路上多路复用的 T1 流

在下一个阶次上,7 个 T2 流按位组合起来,形成一个 T3 流。然后,6 个 T3 流又组合起来,形成一个 T4 流。在每一步组合过程中,都会有少量的额外负载加入进来,用于成帧,或者当发送方和接收方失去同步时用于恢复。

美国和其他国家在基本传输线路(basic carrier)上不一致,与此类似,关于如何通过多路复用得到更高带宽的线路,也同样不一致。美国按照 4、7 和 6 逐步升阶的方案并没有吸引别人也这样做,所以 CCITT 标准要求每一阶都是将 4 个流复用到 1 个流中。而且,美国 and CCITT 标准的成帧和恢复数据也不同。CCITT 针对 32、128、512、2048 和 8192 信道的速度分别是 2.048、8.848、34.304、139.264 和 565.148Mbps。

## SONET/SDH

在光纤使用的早期,每个电话公司都有自己私有的光纤 TDM 系统。当 AT&T 公司于 1984 年被分解以后,本地电话公司必须连接到多家长途电话承运商,而这些承运商各有不同的光纤 TDM 系统,所以,标准化的需求就变得非常迫切了。1985 年,RBOC 的研究机构 Bellcore 开始制定这样的标准,称为 **SONET(Synchronous Optical NETwork, 同步光网络)**。后来,CCITT 也加入进来一起工作,从而于 1989 年产生了一个 SONET 标准,以及一组并行的 CCITT 建议(G. 707、G. 708 和 G. 709)。这些 CCITT 建议称为 **SDH(Synchronous Digital Hierarchy, 同步数字系列)**,它们仅仅在一些很小的方面而不同于 SONET。在美国,几乎所有的长途电话干线的物理层都运行 SONET,其他地方的许多干线也是如此。要想了解更多关于 SONET 的信息,请参考(Bellamy, 2000; Goralski, 2000; and Shepard, 2001)。

SONET 有 4 个主要设计目标。第一个,也是最重要的目标是,SONET 必须使不同的承运商可以协同工作。为了达到这个目标,要求定义一个公共的信令标准,其中涉及到波长、分时、帧结构以及其他的问题。

第二,需要某一种办法来统一美国、欧洲和日本的数字系统,这些系统都基于 64kbps 的 PCM 信道,但是,这些信道被组合起来的方式却各不相同,而且互不兼容。

第三,SONET 必须提供一种办法来复用多条数字信道。在设计 SONET 的时候,美国广泛使用的最高速度的数字线路实际上是 T3,其速度为 44.736Mbps。T4 已经被定义了,但是用得并不多,而且,在 T4 速度以上还没有任何定义。SONET 的部分使命是继续推进这样的速度层次,达到 Gbps,甚至更高。而且,将多条慢速信道复用到一条 SONET

信道中的标准方法也是必要的。

第四, SONEr 必须支持操作 (operation)、管理 (administration) 和维护 (maintenance), 即 OAM。以前的系统在这方面做得并不好。

一个早期的决定是将 SONEr 设计成一个传统的 TDM 系统, 光纤的全部带宽都分配给一条信道, 再将这些信道的时槽划分给不同的子信道。按照这样的做法, SONEr 就是一个同步系统。系统由一个主时钟进行控制, 时钟的精度大约为  $10^{-9}$ 。SONET 线路上的数据位在一个极为精确的时间间隔中被发送出去, 主时钟控制每一个时间间隔。后来, ATM 考虑使用信元交换作为它的基础, 由于它允许信元的到达时间可以是不规律的, 因而它称为异步传输模式 (ATM, Asynchronous Transfer Mode), 这与 SONEr 的同步工作模式形成鲜明的对照。在 SONEr 系统中, 发送方和接收方受限于一个公共的时钟, 而 ATM 则不会。

基本的 SONEr 帧是每  $125\mu s$  间隔中送出 810 字节的数据块。由于 SONEr 是同步的, 所以, 不管是否有实际要发送的数据, 帧都存在。每秒 8000 帧的速率正好符合所有数字电话系统中使用的 PCM 信道的采样率。

对于 810 字节的 SONEr 帧, 最佳的描述方式是一个矩形, 90 列宽, 9 行高, 每个单元对应一个字节。因此, SONEr 系统每秒钟传输 8000 次, 每次  $8 \times 810 = 6480$  位, 因而总的数据传输率为 51.84Mbps。这是基本的 SONEr 信道, 称为 STS-1 (Synchronous Transport Signal-1, 同步传输信号)。所有的 SONEr 干线都是 STS-1 的倍数。

每一帧的前三列保留, 用于系统管理信息, 如图 2.36 所示。其中, 前三行包含段 (section) 的开销, 接下来的 6 行包含线路 (line) 开销。段的开销是在每一段开始和结束时被生成和检查的, 而线路开销则是在每条线路开始和结束时被生成和检查的。

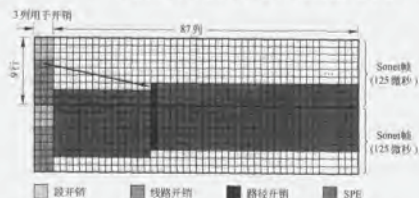


图 2.36 两个背靠背的 SONEr 帧

SONET 发送器发送背靠背的 801 个字节帧, 帧之间没有任何间隙, 即使没有数据的时候也没有间隙 (可以看作这时候在发送空数据)。从接收方的角度来看, 它所看到的是一个连续的数据流, 那么, 它如何知道每一帧从哪里开始呢? 答案在于, 每一帧的前两个字节有一个固定的模式, 接收方搜索这种模式就可以。如果它在大量连续的帧中同样的地方找到了这种模式, 则它假定已经与发送方保持同步了。理论上讲, 一个用户可以按照

非常规律的方式在净荷中插入这种模式,但是在实践中,由于多个用户的数据将被复用到同一帧中,以及其他一些原因,因此,用户这样做是不可能成功的。

剩下的 87 列包含了  $87 \times 9 \times 8 \times 8000 = 50.112\text{Mbps}$  的用户数据。然而,用户数据(称为 **SPE, Synchronous Payload Envelope, 同步净荷包封**)并不总是从第 1 行、第 4 列开始。SPE 可以从帧内的任何一个地方开始。线路开销的第一行包含了一个指针,它指向 SPE 的第一个字节。SPE 的第一列是路径开销(即端到端路径子层协议的头)。

SONET 允许 SPE 可以从帧内任何一个地方开始,甚至可以跨越两帧(如图 2.36 所示),这种能力使得系统非常灵活。例如,当源端正在构造 SONET 空帧的时候,又有一些净荷数据来到了,那么它就可以将这些净荷数据插入到当前帧中,而不用等到下一帧开始。

SONET 的复用层次如图 2.37 所示。从 STS-1 到 STS 192 的数据传输率已经被定义了。对应于 STS-n 的光纤线路称为 OC-n,其复用也是按位进行的,除非由于同步而需要对一个特定的位进行重新排序,其他的位不变。SDH 名称有所不同,它们从 OC-3 开始,因为基于 CCITT 的系统没有接近于 51.84Mbps 的速率。OC-9 线路出现的原因是,它与日本使用的主要高速干线的速度非常接近。在日本也使用 OC-18 和 OC-36。总速率包含了所有的开销。SPE 数据率不包括线路和段的开销。用户数据率不包括所有的开销,只计算 86 列净荷数据。

SONET		SDH	数据传输率(Mbps)		
电子的	光的	光的	总速率	SPE	用户数据
STS-1	OC-1		51.84	50.112	49.536
STS-3	OC-3	STM-1	155.52	150.336	148.608
STS-9	OC-9	STM-2	466.56	451.008	445.824
STS-12	OC-12	STM-4	622.08	601.344	594.432
STS-18	OC-18	STM-6	933.12	902.016	891.648
STS-24	OC-24	STM-8	1244.16	1202.688	118.864
STS-36	OC-36	STM-12	1866.24	1804.032	1783.296
STS-48	OC-48	STM-16	2488.32	2405.376	2377.728
STS-192	OC-192	STM-64	9953.28	9621.504	9510.912

图 2.37 SONET 和 SDH 复用率

顺便提一下,如果一条线路(比如 OC-3)没有被复用,而是仅传输来自一个源的数据,则在线路名称后面加一个字母 c,因此,OC-3c 表示了由三条独立的 OC-1 线路构成的一条 155.52Mbps 线路,而 OC-3c 则表示了来自于单个源的 155.52Mbps 数据流。一条 OC-3c 流内的三个 OC-1 流被按列交替插入,首先是流 1 的第 1 列,然后是流 2 的第 1 列,然后是流 3 的第 1 列,再跟着是流 1 的第 2 列,以此类推,最后形成的帧包括 270 列宽、9 行高。



### 2.5.5 交换

从普通电话工程师的角度来看,电话系统分为两大基本部分:局外部分(本地回路和干线,因为从物理位置来看,它们位于交换局的外部)和局内部分(交换机,它们在交换局的内部)。我们刚才看过了局外部分,现在该看一看局内部分了。

当前,电话系统中用到了两种不同的交换技术:电路交换和分组交换。下面我们首先简要地介绍这两种交换技术。然后我们详细地讨论电路交换,因为它是电话系统的基本工作原理。在后面的章节中,我们将详细地学习分组交换技术。

#### 电路交换

当你或者你的计算机呼叫一个电话的时候,电话系统的交换设备就会寻找一条从你的电话通向接收方电话的物理路径。这项技术称为**电路交换(circuit switching)**,其过程如图 2.38(a)所示。其中 6 个矩形代表了电话运营商的交换局(端局、长途局,等等)。在这个例子中,每个局有 3 条进入的线和 3 条出去的线。当电话呼叫通过一个交换局的时候,在电话进来的线路与某一条出去的线路之间就会建立起一个物理连接(概念上),就如图中的虚线所示。

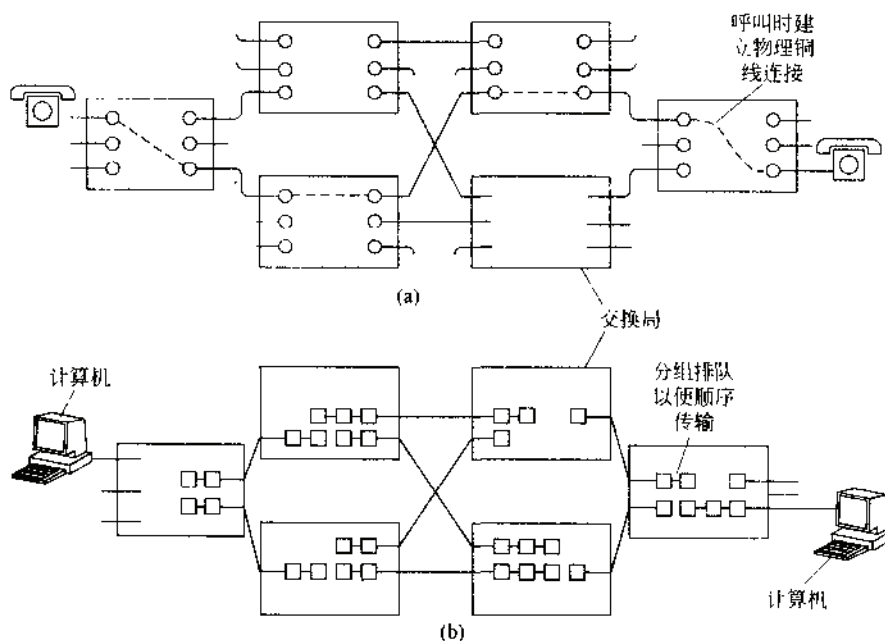


图 2.38

(a) 电路交换; (b) 分组交换

在电话业的早期,建立连接的过程是这样来完成的:接线员在输入插槽和输出插槽中插入一段跳接电缆。实际上,在发明自动电路交换设备的过程中有一个令人惊讶的小故事。自动化的电路交换设备是由 19 世纪密苏里州的一位殡葬工 Almon B.

Strowger 发明的。在电话发明之后不久,当有人死了之后,死者的亲属就会呼叫镇上的接线员,并且说“请接殡葬工”。对于 Strowger 先生来说,不幸的是,这个镇上有两个殡葬工,而电话接线员正好是另一位殡葬工的妻子。他很快明白了,要么他去发明一个自动化的电话交换设备,要么他就得失业。他选择了前一条出路。将近 100 年来,全世界使用的电路交换设备称为 Strowger 装置(历史并没有记载那个失业了的电话接线员是否获得了信息操作员的工作,并且每天在回答诸如“殡葬工的电话号码是多少”之类的问题)。

当然,图 2.39(a)显示的模型是高度简化的,因为实际上,两个电话之间的物理路径中有的部分可能是微波或者光纤,而且在光纤链路上,会有成千上万的电话呼叫被复用在一条专用的路径,并且这条路径会一直持续到该呼叫完成为止。

除了电路交换,另一种交换技术是分组交换,如图 2.38(b)所示。在分组交换技术中,每个分组是独立的,它根据需求被发送出去,事先没有建立专门的路径。每个分组独立地找到通向目标的路径。

电路交换的一个重要特点是,在发送数据之前需要建立一条端到端的路径。从拨完号码到开始响铃,这段时间很有可能达到 10 秒钟,长途电话或者国际电话所需要的时间更长。在这段时间间隔中,电话系统正在寻找路径,如图 2.39(a)所示。请注意,在开始传输数据之前,呼叫请求信号必须一路传向目标,并且沿途要被确认。对于许多计算机应用来说(比如销售点的信用卡验证),长时间的建立过程是不合适的。

在电话呼叫双方之间保留一条路径带来的好处是,一旦连接建立完成,则惟一的数据传输延迟是电磁信号的传输时间,每 1000 公里大约 5ms。建立路径的另一个好处是,没有拥塞的危险,也就是说,一旦电话呼叫已经接通,你永远不会再听到忙信号。当然,在建立连接之前,可能由于交换能力不足,或者干线传输能力不足,你会听到忙信号。

### 报文交换

另一种交换策略是报文交换(message switching),如图 2.39(b)所示。当使用这种交换形式的时候,在发送方和接收方之间事先并没有建立物理路径。相反,当发送方有一块数据要发送的时候,它被存储在第一个交换局(即路由器)中,以后再转发出去,一次一级地中转。每块数据都会被完整地接收下来,并检查错误,然后再重新转发。使用这项技术的网络称为存储-转发网络(store-and-forward network),如第 1 章所提到的那样。

第一个使用报文交换的机电电信系统是电报。在发送局,报文被穿孔在纸带上(离线方式),然后读入系统,并沿着通信线路传输到下一个局,在那里又被穿孔到纸带上。那里的操作员将纸带撕下来,并且通过纸带阅读器读入,每条输出干线上都有一台纸带阅读器。这样的交换局称为撕断纸带式交换局(tape office)。纸带已经消失很久了,并且报文交换也已不再使用,所以本书中我们不再进一步讨论报文交换技术。

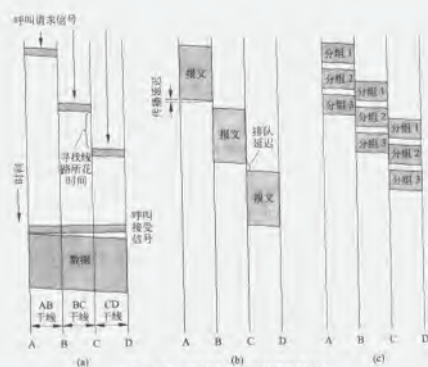


图 2.39 几种交换技术中的事件序列  
(a) 电路交换；(b) 报文交换；(c) 分组交换

#### 分组交换

在报文交换中,对于数据块的大小没有任何限制,这意味着(在一个现代系统中)路由器必须使用硬盘来缓存大块的数据。同时也意味着,一个数据块可能会持续几分钟地占用路由器之间的线路,因而对于交互式通信来说消息交换没有多大用处。为了解决这些问题,因而发明了分组交换,正如第1章所介绍的。分组交换网络对于数据块的大小有一个很严格的上限,这使得这些分组可以被缓存存在路由器的主内存中,而不是磁盘上。由于分组交换网络可以保证用户不会霸占通信线路很长时间(毫秒级),所以它非常适合用来处理交互式通信流量。从图 2.39(b)和(c)可以看出分组交换相比报文交换的另一个好处:如果一个报文包含了多个分组,那么,在后一个分组完全到达之前,前一个分组就可以被转发出去,这样不仅降低了延迟,而且也提高了系统的吞吐量。出于以上这些原因,计算机网络通常是分组交换的,偶尔也会是电路交换的,但绝对不会是报文交换的。

电路交换和分组交换在很多方面有所不同。首先,电路交换要求在通信开始之前先建立一条电路。分组交换不要求预先建立任何连接。只要第一个分组已经准备好了,就可以被发送出去。

在电路交换中,传输之前先建立连接的结果是,在发送方和接收方之间的物理路径上预留了带宽。所有的分组都将沿着这条路径。另一条特性是,让所有的分组沿着同样的

路径也意味着这些分组在到达的时候不会被打乱顺序。在分组交换中,由于没有这样的路径,所以不同的分组可能会沿着不同的路径,取决于分组被发送时候的网络条件。因此,分组在到达的时候可能会错序。

分组交换比电路交换有更强的容错能力。实际上,这正是发明分组交换的真正原因。如果一台交换机不工作了,则所有使用该交换机的电路都终止了,这些电路不可能再传送任何分组。通过分组交换,每个分组在被路由的时候可以绕开这些不工作的交换机。

提前建立一条路径也使得预先保留带宽成为可能。如果已经预留了带宽,那么,当一个分组到达的时候,它可以立即发送出去(占用预留的带宽)。对于分组交换,由于没有预留带宽,所以分组可能需要排队等待,轮到它们的时候才被转发出去。

提前预留带宽也意味着,当一个分组到来的时候不会发生拥塞的现象(除非一下子来了太多的分组,超过了预期的带宽)。但是,在通信之前要首先建立一条电路,这时候可能会由于拥塞而不能成功。因此,对于电路交换和分组交换,拥塞发生的时间不同,前者在建立电路的时候,后者在分组被发送的时候。

如果一条电路已经被预留给某一个特定的用户了,但是并没有流量通过这条电路,则这条电路的带宽就会被浪费,因为它不可能再被用于其他的流量。然而,分组交换并不浪费带宽。所以,从整个系统角度来看,分组交换的效率更高。因此,这里存在一种平衡:要么保证服务,但是浪费资源;要么不保证服务,但是不浪费资源。理解这种平衡对于理解电路交换和分组交换之间的差异非常关键。

分组交换使用存储-转发通信机制。一个分组首先保存到路由器的内存中,然后再发送给下一个路由器。而对于电路交换,所有的数据位只是连续地流过线路。存储-转发技术增加了延迟。

另一个不同之处在于,电路交换是完全透明的。发送方和接收方可以使用任何位速率、格式,或者成帧方法。承运商并不知道这些,也不关心。而在分组交换中,承运商决定了这些基本的参数。打个比方来说,这就好比公路和铁路之间的关系。在公路上,用户可以自由选择交通工具的大小、速度以及其他的特征;而在铁路上,铁道部门决定了这一切。这种透明性使得语音、数据和传真信息共存于电话系统中。

电路交换和分组交换之间的最后一个区别是收费算法。在电路交换中,从历史上沿袭下来的做法是按距离和时间进行收费。对于移动电话,距离通常并不那么重要(除非是国际长途),时间也只是一个并不那么重要的因素(举例来说,一个带 2000 分钟免费通话的收费方案可能比一个带 1000 分钟免费通话的收费方案更加贵,有时候夜里或者周末的话费比平常要便宜)。在分组交换中,连接时间不是一个收费因素,有时流量的大小倒是一个衡量因素。对于家庭用户,ISP 常常按月租费的方式来收取费用,因为这样可以减轻他们的收费工作负担,并且顾客也很容易理解这种收费模型,但是骨干网承运商则按照流量大小收取区域网络的费用。图 2.40 总结了以上这些区别。

电话交换和分组交换是非常重要的,因此,稍后我们将回来继续讨论它们,并且细致地讲述所涉及的一些技术。

项目	电路交换	分组交换
建立呼叫连接	要求	不要求
专门的物理路径	是	否
每个分组沿着同样的路由路径	是	否
分组按序到达	是	否
一台交换机崩溃是否有严重影响	是	否
可用带宽	固定	动态
可能拥塞的时间点	建立呼叫连接的时候	每个分组传送的时候
可能有浪费的带宽	是	否
存储-转发传输机制	否	是
透明性	是	否
收费	每分钟	每个分组

图 2.40 电路交换和分组交换的比较

## 2.6 移动电话系统

传统的电话系统(即使有一天端到端之间有了几个 G 的传输速率的光纤通路)仍然不能满足一群正在不断增长的用户的需求:移动用户。现在,人们希望在飞机上、汽车上、游泳池中,或者在公园漫步的时候也可以打电话。在几年之内,他们还会希望在这些地方可以发送电子邮件或者浏览 Web 网页。因此,无线电话将有极大的利益和市场机会。本节中,我们将细致地学习这个主题。

无线电话有两种基本的流行方式:无绳电话和移动电话(有时候也称为蜂窝电话(cell phone))。无绳电话(cordless phone)是指这样的设备:包含一个基座和一个手持机,两个合起来购买,主要用于家庭。无绳电话从来不用于网络连接,所以我们不再进一步讨论。相反,我们将注意力集中在移动电话系统上,它已经被广泛地用于语音和数据通信。

**移动电话(mobile phone)**已经经过了三代发展,每一代有不同的技术:

- (1) 模拟语音。
- (2) 数字语音。
- (3) 数字语音和数据(Internet、e-mail 等)。

尽管我们的讨论绝大多数是关于这些系统的技术,但是,关注一下政策和细微的市场决策如何产生了巨大的影响,也是非常有意思的。第一个移动系统是在美国由 AT&T 公司设计的,并且得到 FCC 的允许在全国实施。因此,当时全美国只有一个(模拟)系统,在加州购买的电话也可以在纽约使用。相反,当移动电话在欧洲出现的时候,每个国家都自己设计系统,从而导致了一片混乱。

欧洲从它的失败中吸取了教训,所以,当数字系统到来的时候,政府运行的邮电部集



中在一起,共同完成了一个标准化系统(GSM),所以,欧洲的任何一部移动电话都可以在欧洲的任何地方使用。到那时候,美国已经决定,政府不应该卷入到标准化事务中,所以它把数字系统留给了市场自己去运行。其结果是,不同的设备厂商生产了不同种类的移动电话。因此,美国现在有两个主要的数字移动电话系统(两者互不兼容),以及一个小一点的系统。

尽管移动电话最初是由美国领导的,但是现在欧洲的移动电话的拥有量和应用大大超过了美国。全欧洲统一使用同一个系统,这是一部分原因,其他还有一些原因。美国和欧洲第二个不同的地方是电话号码的分区规范。在美国,移动电话号码与常规的固定电话号码混合在一起。因此,呼叫方无法识别一个号码,比如(212)234-5678,是一个固定电话(费用便宜,甚至免费),还是一个移动电话(费用昂贵)。为了避免人们对使用电话神经过敏,电话公司决定,移动电话用户接听电话也必须付费。结果是,许多人害怕仅仅因为接听电话而支付大笔费用,所以对于是否购买移动电话犹豫不决。在欧洲,移动电话有一个特殊的区域码(类似于800和900号码),所以移动电话号码很容易区分。因此,“呼叫方付费”的习惯规则也适用于欧洲的移动电话(不过,国际呼叫的费用另有算法)。

影响用户选择移动电话的第三个因素是,在欧洲广泛使用了预付费的移动电话(在某些地区达到了75%)。这些电话在很多商店都可以买到,其手续并不比买一个收音机更加繁琐。你付了钱就可以走了。这些电话已经预先支付了一定数量的费用(比如20或者25欧元),而且当余额到零的时候你还可以再续费(通过一个保密的PIN码)。因此,欧洲几乎每一个青少年和许多小孩都有了移动电话(通常使用预付费的电话),所以他们的父母总是可以知道他们在哪里,并且不用担心他们会花掉大笔的费用。如果只是在偶尔情况下使用移动电话的话,这种用法也非常合适,因为它不涉及到月租费用,或者接听电话不需要付费。

### 2.6.1 第一代移动电话:模拟语音

关于移动电话的政策和市场情况已经介绍足够多了。现在我们来查看一下移动电话的技术,首先从最早的系统开始。在20世纪最初的几十年中,移动无线电话偶尔用于海军和军事通信。1946年,圣·路易斯建立起了第一个可用于汽车的电话系统。该系统使用了一个大的发射器,放在一个高大的建筑物顶上,并且该系统只有一个信道,同时用于发送和接收。为了通话,用户必须按一下按钮,以打开发送功能,并关闭接收功能。这样的系统称为按钮启动式通话系统(push-to-talk system)。20世纪50年代后期,有几个城市安装了这样的系统。电视节目中的CB无线电、出租车和警车通常使用这项技术。

在20世纪60年代,IMTS(Improved Mobile Telephone System,改进的移动电话系统)开始安装使用。它也使用了一个大功率(200瓦)的发射器,放在一座小山顶上,但是这次有两个频率,一个用于发送,一个用于接收,所以,启动通话的按钮就可以不需要了。由于所有来自移动电话的通信是在不同于往外发送信号的信道上进行的,所以移动用户相互之间不会听到别人的通话(不同于出租车使用的按钮启动式通话系统)。

IMTS支持23个信道,频率范围为150~450MHz。由于信道的数量比较少,所以,用户常常需要等待很长时间才能听到拨通的声音。而且,由于小山顶上发射器的功率很

大,所以邻近的系统必须相距几百公里远才能避免干扰。总而言之,有限的容量使得该系统无法被真正实际使用。

### 高级移动电话系统

AMPS(Advanced Mobile Phone System, 高级移动电话系统)的出现改变了这一切。该系统由贝尔实验室发明,并且于1982年首次在美国安装。随后英国和日本也安装了这样的系统,在英国称它为TACS,在日本称为MCS-1J。尽管该系统现在已经不再先进,但是我们仍然看一看它的细节,因为它的数字版本后继者D-AMPS出于向后兼容性的考虑,直接继承了AMPS的许多基本特性。

在所有的模拟电话系统中,一个地理区域分成许多蜂窝单元(cell),这也就是有时候移动电话称为蜂窝电话(cell phone)的原因。在AMPS中,每个蜂窝单元通常为10~20公里的跨度;在数字系统中,蜂窝单元要小一些。每个蜂窝单元使用一组频率,并且这组频率不会被它的任何一个邻居所使用。蜂窝系统之所以比以前的系统有更大的容量,关键在于它使用了相对较小的蜂窝单元,并且在较近(但不相邻)的蜂窝单元中重复使用相同的传输频率。在IMTS系统中,100公里范围内每个频率只能有一个电话呼叫,而在同样的区域范围内,AMPS系统可以有100个10公里的蜂窝单元,并且在被广泛隔离开的蜂窝单元中,每个频率上可以有10~15个电话呼叫。因此,蜂窝状的设计至少使系统增加了一个数量级的容量,而且蜂窝单元越小,容量增加得越多。此外,蜂窝单元越小,意味着所需要的功率越小,因而发射器和手持机也越小、越便宜。手持电话的功率为0.6瓦;汽车里的发射器的功率为3瓦,这是FCC允许的最大值。

频率重用的思想如图2.41(a)所示。蜂窝单元通常是粗糙的圆形,但是很容易用六角形作为它们的模型。在2.41(a)中,蜂窝单元都是同样大小的,它们可以分成每7个一组。每个字母代表了一组频率。请注意,对于每个频率组,都有一个两蜂窝宽的缓冲带,在缓冲带中该频率组不会被重用,从而保证了好的隔离性和较低的干扰。

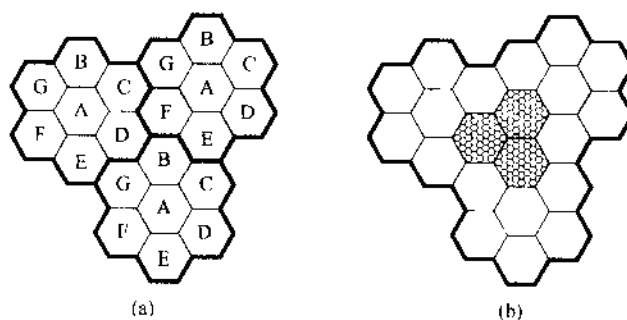


图 2.41

(a) 在相邻的蜂窝单元中,频率不会被重用;(b) 为了增加更多的用户,可以使用更小的单元

找到一个较高的空中位置来放置基站天线是一个大问题。这个问题导致了一些电信承运商与罗马天主教会结成了联盟,因为罗马天主教会在全世界拥有大量高位置的、潜在的天线放置地点,并且在统一的管理之下。

在有些地区,用户数量的增长超出了系统的负载能力,于是,在这些地区,可以把功率降低下来,并且将超过负载的蜂窝单元切分成更小的微蜂窝单元(**microcell**),以便允许更多的频率被重用,如图 2.41(b)所示。有时,当有大量移动用户聚集在一起达几个小时之久的时候,比如运动赛事、摇滚音乐会等,电话公司可以利用可移动的塔(塔上有卫星链路),建立一些临时的微蜂窝单元。蜂窝单元到底应该多大,这是一个复杂的问题,请参考(Hac, 1995)。

在每个蜂窝单元的中心是一个基站,该单元中的所有电话都向它发送信息。基站是由一台计算机和发射器/接收器组成的,并且发射器/接收器连接到一个天线上。在一个小的系统中,所有的基站都连接到一个称为 **MTSO**(**Mobile Telephone Switching Office**,移动电话交换局)或者 **MSC**(**Mobile Switching Center**,移动交换中心)的设备上。大一点的系统可能需要几个 **MTSO**,并且所有的 **MTSO** 都连接到一个二级 **MTSO** 上,以此类推。**MTSO** 就好比是电话系统中的端局一样,而且实际上,它们也确实连接到至少一个电话系统端局。**MTSO** 与基站相互之间进行通信,并且利用一个分组交换网络与 **PSTN** 进行通信。

在任何一个时刻,每个移动电话在逻辑上属于某一个蜂窝单元,并且受该蜂窝单元基站的控制。当一个移动电话在物理上离开一个蜂窝单元的时候,它的基站注意到该电话的信号越来越弱,于是询问周围的基站,它们从该电话上得到多大的功率。然后,该基站将所有权转交给获得最强信号的那个蜂窝单元,也就是该电话当前所在的那个蜂窝单元。然后,该电话就会接到通知它有新的老板了,如果当时正在通话,则该电话会被切换到一个新的信道上(因为老的信道不会被任何一个相邻蜂窝单元重用,所以必须切换到新的信道上)。这个过程称为移交(**handoff**),大约需要 300ms。信道分配是由系统的神经中枢 **MTSO** 来完成的。基站只负责无线电波的中转。

移交过程可以有两种实现方法。在软移交(**soft handoff**)方法中,在原来的基站挂断电话之前,新的基站就获得了电话信号。按照这种方法,通话过程不会失去连贯性。这种做法的缺点是,电话必须具备同时调节到两个频率(老的频率和新的频率)的能力。第一代和第二代移动电话都做不到这一点。

在硬移交(**hard handoff**)方法中,在新的基站获得电话之前,老的基站就挂断了电话。如果新的基站不能获得该电话的话(比如,因为当前没有可用的频率),则这次通话就被迫中断了。用户往往会注意到这种现象,但是在当前的设计中,这种偶然事件是不可避免的。

### 信道

**AMPS** 系统使用了 832 个全双工信道,每个信道由一对单工信道组成。832 个单工发送信道的频率范围为 824~849MHz,832 个单工接收信道的频率范围为 869~894MHz。每个单工信道的频宽为 30kHz。因此,**AMPS** 使用 **FDM** 来分隔信道。

在 800MHz 的频段中,无线电波的波长大约为 40cm,并且按直线传播。它们会被雨水和植物吸收,地面和建筑物对它们有反射作用。这样的情况也有可能发生:移动电话发送的信号通过直接路径到达基站,但是在经过地面或者建筑物反射之后又有一些信

号也到达基站,这样可能会导致回音,或者信号失真(多径衰减)。有时候,甚至还能够听到远处的通话(经过多次反射之后)。

832 个信道分成 4 类:

- (1) 控制(从基站到移动电话),用于管理系统。
- (2) 呼叫(从基站到移动电话),用于提醒移动用户有呼叫到来。
- (3) 访问(双向),用于呼叫建立和信道分配。
- (4) 数据(双向),用于语音、传真和数据。

有 21 条信道被保留用于控制,并且它们固化在每部电话的 PROM 中。由于相同的频率不能在相邻的单元中重复使用,所以,每个单元中实际可用的语音信道的数目远远小于 832,通常为 45 左右。

### 呼叫管理

在 AMPS 中,每部移动电话的 PROM 中有一个 32 位的序列号和一个 10 位数字的电话号码。电话号码的表示方法是这样的:3 位数字的区域码占 10 位,7 位数字的用户号码占 24 位。当电话开机的时候,它对预先设置的 21 条控制信道进行扫描,以便找到最强的信号。

然后,电话广播它的 32 位序列号和 34 位电话号码。尽管语音信道本身是模拟的,但是,如同 AMPS 中所有的控制信息一样,这个广播分组以数字形式被多次发送,并且带有纠错码。

当基站听到了广播信息的时候,它就告诉 MTSO,然后 MTSO 会记录下新顾客的到达情况,同时也通知顾客的主 MTSO 它的当前位置。在正常的操作过程中,移动电话每隔 15 分钟左右重新注册一次。

为了打电话,移动用户打开电话,在键板上输入被呼叫的电话号码,再按下发送(SEND)按钮。然后,该电话将被呼叫的号码以及它自己的标识通过访问信道发送出去。如果发生碰撞的话,它会试着再发送。当基站接到了请求的时候,它通知 MTSO。如果呼叫者是该 MTSO 的公司(或者它的某一个合作伙伴)的顾客,则 MTSO 为这次呼叫寻找一条空闲的信道。如果找到了的话,则通过控制信道将信道号发送回去。然后移动电话自动切换到被选中的语音信道上,并且等待被呼叫方拿起电话。

打进来的电话呼叫的工作方式有所不同。首先,所有空闲的电话都在不断地监听呼叫信道,以便检测是否有发给它们的消息。当呼叫一部移动电话(或者从固定电话上呼叫,或者从另一部移动电话上呼叫)的时候,被呼叫方的主 MTSO 就会收到一个分组,询问被呼叫方现在在哪里。找到之后,一个分组被发送到该电话当前所在蜂窝单元的基站,然后基站在呼叫信道上发送一条广播消息,消息的形式譬如“14 号,你在吗?”,然后被呼叫的电话在访问信道上回答“是的,我在”。基站然后就会这样说“14 号,3 号信道上有人呼叫你”。这时,被呼叫的电话切换到 3 号信道,并且开始响铃(或者播放一段悦耳的音乐,也许这段音乐是电话所有者以前收到的一份生日礼物)。

### 2.6.2 第二代移动电话：数字语音

第一代移动电话是模拟的,第二代移动电话是数字的。正如第一代没有全球范围内的标准一样,在第二代的发展过程中也没有形成统一的标准。现在使用的系统有4种:D-AMPS、GSM、CDMA和PDC。下面我们将讨论前面的三种。PDC只用于日本,它基本上是为了与日本的第一代模拟系统向后兼容而修订的D-AMPS。在市场宣传材料中,有时候用PCS(Personal Communications Services,个人通信服务)来表示第二代(即数字)系统。最初它是指使用1900MHz频段的移动电话,但是现在很少再有这种区分了。

#### D-AMPS—数字的高级移动电话系统

AMPS的第二代是D-AMPS,它是完全数字的。国际标准IS-54及其后续的IS-136描述了D-AMPS。D-AMPS经过了精心细致的设计,以便可以与AMPS共存,因此,在同一个蜂窝单元中,第一代和第二代移动电话可以同时工作。尤其是,D-AMPS与AMPS使用了同样的30kHz信道,并且位于同样的频率处,所以有可能一条信道是模拟的,而相邻的信道是数字的。MTSO根据一个蜂窝单元中的电话混合情况,来决定哪些信道是模拟的,哪些信道是数字的,当单元中电话的混合情况发生变化时,它可以动态地改变信道的类型。

当D-AMPS作为一项服务而引入的时候,一个新的频段也加入进来,以处理可能新增的负载。上行信道在1850~1910MHz范围内,对应的下行信道在1930~1990MHz范围内,同样也是成对的,如同AMPS中一样。在这个频段上,波长为16cm长,所以,一个标准的四分之一波天线只有4cm长,从而导致了更小的电话。然而,许多D-AMPS电话可以同时使用850MHz和1900MHz频段,以获得更宽的可用信道范围。

在一部D-AMPS移动电话上,该电话接收到的语音信号也要被数字化和压缩,所使用的模型比我们前面学习过的增量调制和预测编码方案更加复杂。压缩方案考虑到了人类发声系统的某些细节特点,以使带宽从标准的56kbps PCM编码降低到8kbps甚至更小。该压缩过程是通过一个称为声音编码器(vocoder)的电路来完成的(Bellamy, 2000)。并且,压缩过程是由电话来完成,而不是在基站或者端局完成的,这样可以降低在空中链路上传输的数据量。对于固定电话,在电话一端完成压缩过程不会带来任何好处,因为降低本地回路上的通信流量并不会提高电话系统的容量。

对于移动电话,在手持设备一端完成数字化和压缩过程可以获得巨大的收益,所以,在D-AMPS中,三个用户可以利用时分多路复用技术共享同一对频率。每一对频率支持25帧/秒,每帧40毫秒。每一帧分成6个时槽,每个时槽6.67ms,图2.42(a)显示了最低的频率对。

在每一帧中,三个用户轮流使用上行和下行数据流链路。例如,在图2.42(a)所示的第一个时槽中,用户1可以给基站发送数据,同时用户3从基站接收数据。每个时槽有324位长,其中64位被用于防护时间段、同步和控制,留给用户净荷数据的有260位。在净荷位中,其中101位被用于在空中噪声链路上的纠错用途,所以,最终只有159位被留给压缩之后的语音数据。在每秒50个时槽的情况下,用于压缩后语音数据的有效带宽不



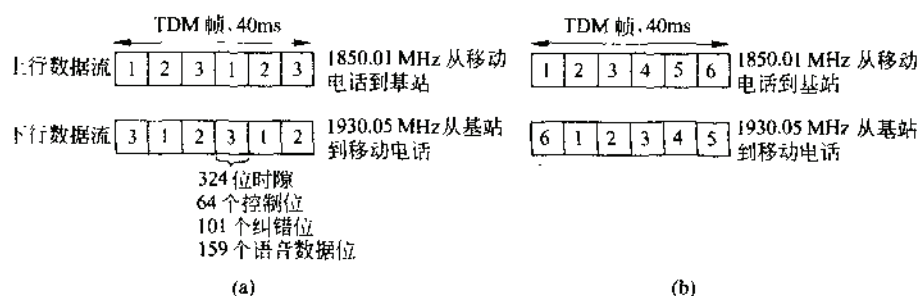


图 2.42

(a) 支持 3 个用户的 D-AMPS 信道; (b) 支持 6 个用户的 D-AMPS 信道

到 8kbps, 大约是标准 PCM 带宽的七分之一。

利用更好的压缩算法, 有可能使语音数据的带宽小于 4kbps, 在这种情况下, 一帧中可以容纳 6 个用户, 如图 2.42(b) 所示。从运营商的角度来看, 能够将 3~6 个 D-AMPS 用户的带宽挤压到一个 AMPS 用户所需要的频段中, 这是一个巨大的胜利, 也说明了 PCS 的一种普及程度。当然, 在 4kbps 带宽下的语音质量与在 56kbps 下所获得的语音质量是不可比的, 但是, 很少有 PCS 运营商宣传他们的高保真语音质量。有一点应该很清楚, 那就是, 对于数据信号, 8kbps 信道甚至还不如早期的 9600bps 调制解调器。

D-AMPS 的控制结构是相当复杂的。简而言之, 16 帧为一组, 构成了一个超级帧, 在每一个超级帧中, 某些特定的控制信息会出现数次(有限次数)。共有 6 个主要的控制信道: 系统配置、实时和非实时控制、呼叫、访问应答和短消息。但是从概念上讲, 它的工作方式与 AMPS 非常相似。当一部移动电话开机的时候, 它与基站联系, 以便宣告自己位于当前的蜂窝单元中, 然后在一个控制信道上监听打进来的电话。当 MTSO 接受一个新的移动电话加入进来的时候, 它通知该用户的主 MTSO, 告诉它现在他在这里, 这样, 给他的电话呼叫就可以被正确地路由过来。

AMPS 和 D-AMPS 之间的一个不同之处是移交的处理方式。在 AMPS 中, MTSO 管理整个移交过程, 不需要移动设备提供任何帮助。从图 2.42 中可以看出, 在 D-AMPS 中, 移动电话有三分之一的时间既不发送数据, 也不接收数据。它使用这些空闲的时槽来测量线路的质量。当它发现信号变得很弱的时候, 它会告诉 MTSO 自己对线路质量不满意, 然后 MTSO 就会断开该连接, 同时移动电话试图转到另一个信号更强的基站上。如同 AMPS 一样, D-AMPS 也需要花 30ms 左右的时间来完成移交过程。这项技术称为 MAHO(Mobile Assisted HandOff, 移动电话辅助移交)。

### GSM—全球移动通信系统

D-AMPS 广泛应用于美国和日本(在日本使用的是一种修订形式)。世界上几乎所有的其他的地区都使用一种称为 GSM(Global System for Mobile communication, 全球移动通信系统)的系统; 在美国现在也开始在小范围内使用 GSM 了。大体上来看, GSM 非常类似于 D-AMPS。它们都是蜂窝状的系统。两个系统都使用了频分多路复用, 每部移动电话在一个频率上发送数据, 在另一个高一点的频率上接收数据(对于 D-AMPS, 接收频

率高出 80MHz;而对于 GSM,接收频率高出 50MHz)。而且,这两个系统都利用时分多路复用技术将每一对频率分成时槽,供多个用户共享。然而,GSM 的信道比 AMPS 的信道更宽(分别为 200kHz 和 30kHz),因而 GSM 能够容纳的用户数量相对少一些(8 比 3),这也使得 GSM 中每个用户的数据传输率比 D-AMPS 高得多。

下面我们将简要地讨论一下 GSM 的一些主要特点。不过,需要说明一点,GSM 标准的篇幅非常长,如果打印出来的话,会超过 5000 页。这份标准材料中大部分内容涉及到 GSM 系统的工程方面,特别是接收器的设计,包括如何处理多路径的信号传播,以及如何处理发射器和接收器之间的同步等。下面的介绍并没有提到这方面的内容。

每个频段的宽度为 200kHz,如图 2.43 所示。一个 GSM 系统有 124 对单工信道,每对单工信道为 200kHz 宽,使用时分多路复用技术支持 8 个独立的连接。每个当前活动的站(这里的站是指用户,不是基站——译者注)都被分配一个时槽(位于某一对信道上)。理论上讲,每个蜂窝单元可以支持 992 条信道,但是,这 992 条信道中有许多是不可用的,目的是为了避免与相邻蜂窝单元发生频率冲突的情况。在图 2.43 中,8 个带阴影的时槽属于同一个连接,每一个方向有 4 个。发送和接收不会同时进行,因为 GSM 无线设备不能同时发送和接收信号,两者之间进行切换需要一定的时间。如果移动站被分配了 890.4/935.4 MHz,并且希望在时槽 2 中给基站发送信号,则系统将使用下面的 4 个阴影时槽(以及它们的后续时槽),在每个时槽中放一些数据,直到所有的数据都被发送出去。

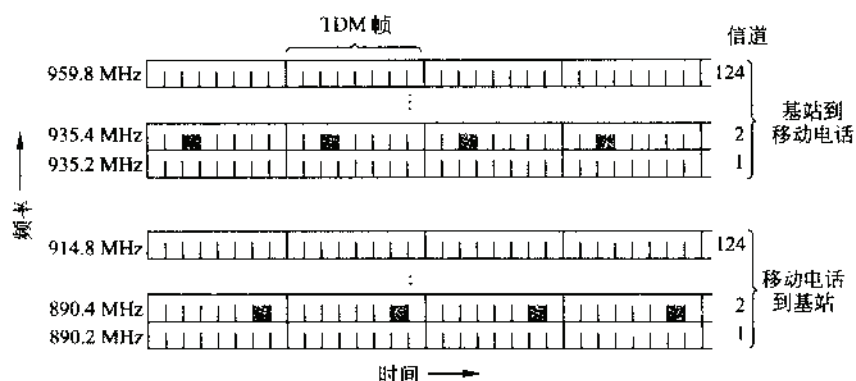


图 2.43 GSM 使用 124 个频道,每个频道使用一个 8 时槽的 TDM 系统

图 2.43 中显示的 TDM 时槽是复杂的成帧层次结构中的一部分。每个 TDM 时槽有一个特定的结构,TDM 时槽组合起来形成了多帧(multiframe)的概念,它也有特定的结构形式。该成帧层次结构的一个简化版本如图 2.44 所示。从这里我们可以看到,每个 TDM 时槽包含一个 148 位的数据帧,它占用信道 577 $\mu$ s(包括每个时槽之后的 30 $\mu$ s 防护时间)。每个数据帧的开始和结束都有 3 个 0 位,用于帧的分界。每个数据帧也包含 2 个 57 位的信息(Information)域,每个信息域都有一个控制位,用来指示随后的信息域是语音还是数据。在两个信息域之间是一个 26 位的同步(Sync)域,接收方利用这个域同步到发送方的帧边界。

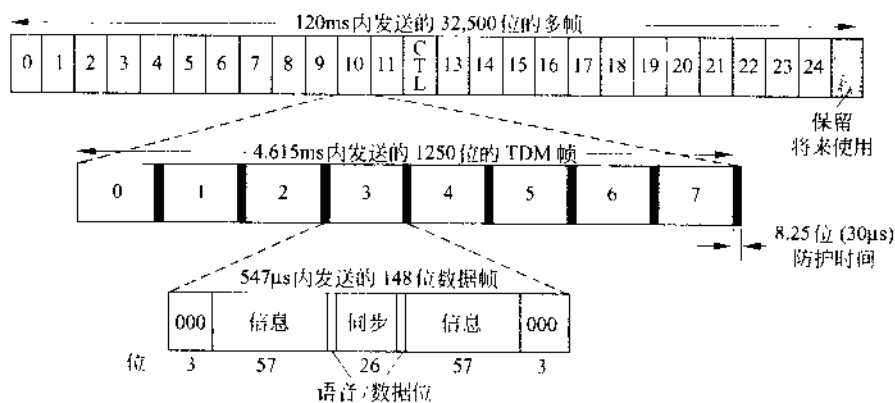


图 2.44 GSM 成帧层次结构的一部分

发送一个数据帧需要  $547\mu\text{s}$ ,但是在每个  $4.615\text{ms}$  以内,一个发射器只允许发送一个数据帧,因为它与其他 7 个站共享同一信道。每条信道的总传输率为  $270.833\text{kbps}$ ,分给 8 个用户使用。这使得每个用户的传输率为  $33.854\text{kbps}$ ,是 D-AMPS(每秒 50 次 324 位,共  $16.2\text{kbps}$ )的两倍多。然而,如同 AMPS 一样,各种额外开销吃掉了相当一部分带宽,最终每个用户只有  $24.7\text{kbps}$  用于纠错前的净荷。在经过纠错之后,有  $13\text{kbps}$  用于语音,这使得语音质量明显好于 D-AMPS(代价是使用了相对较多的带宽)。

从图 2.44 可以看出,8 个数据帧构成了一个 TDM 帧,26 个 TDM 帧构成了一个  $120\text{ms}$  的多帧。在一个多帧的 26 个 TDM 帧中,12 号时槽用于控制,25 号时槽保留为将来使用,所以只有 24 个时槽用于用户流量。

然而,除了图 2.44 中显示的 26 时槽的多帧结构以外,GSM 还使用了 51 时槽的多帧结构(图中没有显示)。这些时槽中有一些用于几条控制信道,GSM 通过这些控制信道完成管理功能。广播控制信道(broadcast control channel)是从基站输出的一个连续流,其中包含了基站的标识和信道的状态。所有的移动站都监视它们的信号强度,以便了解何时进入到一个新的蜂窝单元中。

专用控制信道(dedicated control channel)用于位置更新、注册和建立呼叫。尤其是,每个基站都维护了一个关于当前在它管辖下的移动站的数据库。维护该数据库所需要的信息都是在专用控制信道中发送的。

最后,还有一条公共控制信道(common control channel),它分成三个逻辑子信道。第一个子信道是呼叫信道(paging channel),基站用它来通告有关进入呼叫的情况。每个移动站都要不停地监视该信道,以便发现那些应该由它来回答的电话呼叫。第二个子信道是随机访问信道(random access channel),它允许用户在专用控制信道上请求一个时槽。如果两个请求冲突的话,它们都会被拒绝,必须以后重试。移动站利用专用控制信道中的时槽,可以建立一个电话呼叫。第三个子信道为访问授权信道(access grant channel),用于宣告在专用控制信道中被分配的时槽。

## CDMA—码分多路访问

D-AMPS 和 GSM 都是相对比较传统的系统。它们同时使用了 FDM 和 TDM, 先将频段分成信道, 再将信道分成时槽。然而, 在移动电话技术领域, 还有第三种, 这就是 CDMA (Code Division Multiple Access, 码分多路访问), 它的工作原理完全不同于 D-AMPS 和 GSM。当 CDMA 刚开始被提出来的时候, 工业界对它的反应就如同当哥伦布建议按照相反(错误)的方向航行到达印度时伊莎贝拉女皇的反应一样。然而, 经过一家公司(即 Qualcomm 公司)坚持不懈的努力, CDMA 已经渐趋成熟, 现在不仅已被接受, 而且还被认为是最佳的技术方案, 这也是第三代移动系统的基础。在美国, CDMA 广泛应用于第二代移动系统中, 正与 D-AMPS 进行竞争。例如, Sprint PCS 使用 CDMA, 而 AT&T Wireless 则使用 D-AMPS。国际标准 IS-95 描述了 CDMA, 所以, 有时候也用 IS-95 代指 CDMA。CDMA 的品牌名为 cdmaOne。

CDMA 完全不同于 AMPS、D-AMPS 和 GSM。CDMA 并不将整个可用的频率范围分成几百条窄的信道, 相反, 它允许每个站任何时候都可以在整个频段范围内发送信号。利用编码技术可以将多个并发的传输过程分离开。CDMA 也不再假设冲突的帧被完全丢弃掉。相反, 它假设多个信号可以线性叠加。

在讨论其算法之前, 我们来打一个比方: 在一个机场大厅里, 许多人正在两两交谈。TDM 可以看作是所有人都聚集在大厅的中央, 但是按顺序发表自己的看法。FDM 可以看作是大厅里的人分散在各处, 两两扎一堆, 每一堆的话题同时在进行, 相互之间完全独立。CDMA 可以看作是大厅中央的每个人都在说话, 但是每一对使用不同的语言。讲法语的这一对在谈论关于法国的事情, 并且把所有与法国无关的内容都当作噪声。因此, CDMA 的关键在于, 能够提取出期望的信号, 同时拒绝所有其他的信号, 并把这些信号当作噪声。下面简单地描述一下 CDMA。

在 CDMA 中, 每一位时间又分成  $m$  个时间间隔, 称为时间片(chip)。通常情况下, 每一位分成 64 或者 128 个时间片, 但是在下面给出的例子中, 为了简便起见, 我们将使用 8 个时间片。

每个站被分配一个惟一的  $m$  位的代码, 称为时间片序列(chip sequence)。为了传输位“1”, 一个站将它的时间片序列发送出去; 为了传输位“0”, 它将时间片序列的补码发送出去。除此以外, 不允许再有其他的模式。因此, 对于  $m=8$ , 如果站 A 的时间片序列为 00011011, 则它发送 00011011 就相当于发送了位“1”, 发送 11100100 就相当于发送了位“0”。

按照这种编码方式, 本来每秒发送  $b$  位, 现在变成每秒发送  $mb$  个时间片, 所以, 所需发送的信息量变成了原来的  $m$  倍, 因此, 只有当可用带宽也增加到原来  $m$  倍的情况下, 这种方案才是可行的, 这使得 CDMA 成为一种扩频的通信形式(假设调制方案或者编码技术保持不变)。如果我们有 1MHz 的频段用于 100 个站, 则利用 FDM, 每个站有 10kHz 频段, 它可以以 10kbps(假设每个赫兹 1 位)的速率发送信息。利用 CDMA, 每个站使用完全的 1MHz 频段, 所以, 时间片率为每秒 1M 个时间片。若每位少于 100 个时间片, 则 CDMA 方案中每个站的有效带宽高于 FDM, 而且信道分配问题也随之解决。

从教学角度来考虑,使用双极概念可能会更加方便。假设二进制 0 为 -1,二进制 1 为 +1,并且用圆括号来表示时间片序列,所以针对站 A 的位 1 可以表示为(-1-1-1+1+1-1+1+1)。在图 2.45(a)中,我们给出了 4 个二进制时间片序列,它们已经分配给 4 个示例移动站。图 2.45(b)显示了这 4 个时间片序列的双极表示法。

A: 0 0 0 1 1 0 1 1	A <sub>2</sub> : (-1 -1 -1 +1 +1 -1 +1 +1)
B: 0 0 1 0 1 1 1 0	B <sub>2</sub> : (-1 -1 +1 -1 +1 -1 +1 -1)
C: 0 1 0 1 1 1 0 0	C <sub>2</sub> : (-1 +1 -1 +1 +1 +1 -1 -1)
D: 0 1 0 0 0 0 1 0	D <sub>2</sub> : (-1 +1 -1 -1 -1 -1 +1 -1)
(a)	(b)
6 个示例:	
- - 1 - C	S <sub>1</sub> = (-1 +1 -1 +1 +1 +1 -1 -1)
- 1 1 - B+C	S <sub>2</sub> = (-2 0 0 0 +2 -2 0 -2)
1 0 - - A+B	S <sub>3</sub> = (0 0 -2 +2 0 -2 0 +2)
1 0 1 - A+B+C	S <sub>4</sub> = (-1 +1 -3 -3 +1 -1 -1 +1)
1 1 1 1 A+B+C+D	S <sub>5</sub> = (-4 0 -2 0 -2 0 +2 -2)
1 1 0 1 A+B+C+D	S <sub>6</sub> = (-2 -2 0 -2 0 -2 +4 0)
(c)	
S <sub>1</sub> · C = (1+1+1+1+1+1+1+1)/8 = 1	
S <sub>2</sub> · C = (2+0+0+0+2+2+0+2)/8 = 1	
S <sub>3</sub> · C = (0+0+2+2+0-2+0-2)/8 = 0	
S <sub>4</sub> · C = (1+1+3+3+1-1+1-1)/8 = 1	
S <sub>5</sub> · C = (4+0+2+0+2+0-2+2)/8 = 1	
S <sub>6</sub> · C = (2-2+0-2+0-2+4+0)/8 = -1	
(d)	

图 2.45

(a) 4 个移动站的二进制时间片序列; (b) 时间片序列的双极表示;  
(c) 6 个传输示例; (d) 站 C 的信号恢复过程

每个站都有自己惟一的时间片序列。我们假设符号 S 表示站 S 的 m 位时间片向量, S 表示它的非值(即补码)。所有的时间片序列都是两两正交的,意思是,任何两个不同的时间片序列 S 和 T 的归一化内积(写为 S · T)为 0。我们知道,利用 Walsh 编码方法可以产生这样的正交时间片序列。按照数学语言,时间片序列的正交性可以表示成:

$$S \cdot T = \frac{1}{m} \sum_{i=1}^m S_i T_i = 0 \quad (2-4)$$

其意思是:在 S 和 T 的分量中,对应分量相等的数目与不相等的数目是一样的。这种正交性是非常重要的,后面我们需要用到这种性质。请注意,如果 S · T = 0,则 S ·  $\bar{T}$  也是 0。任何时间片序列与自身的归一化内积一定是 1:

$$S \cdot S = \frac{1}{m} \sum_{i=1}^m S_i S_i = \frac{1}{m} \sum_{i=1}^m S_i^2 = \frac{1}{m} \sum_{i=1}^m (\pm 1)^2 = 1$$

这个结论很显然,因为内积中的每一项都是 1,所以求和的结果是 m。同样也可以求



出  $S \cdot \bar{S} = -1$ 。

在每一位的时间中,一个站为了传输位“1”,它发送自己的时间片序列;为了传输位“0”,它发送自己的时间片序列的非值;或者它可以什么也不发送。现在我们暂时假设所有的站在时间上都是同步的,所以,所有的时间片序列都从同一个时刻开始。

当两个或者多个移动站并发传输的时候,它们的双极信号是线性相加的。例如,如果在一个时间片周期中,三个站输出+1,一个站输出-1,则结果是+2。你可以将这看成是电压值相加:三个站输出+1伏电压,一个站输出-1伏电压,结果得到2伏电压。

在图 2.45(c)中,我们看到 6 个例子:一个或者多个移动站同时传输时间片序列。在第一个例子中,C 传输位“1”,所以我们只得到 C 的时间片序列。在第二个例子中,B 和 C 同时传输位“1”,所以,我们得到它们的时间片序列的和,即:

$$\begin{aligned} & (-1 -1 +1 -1 -1 +1 +1 -1) + (-1 +1 -1 +1 -1 +1 -1 -1) \\ & = (-2 \ 0 \ 0 \ 0 +2 \ +2 \ 0 \ -2) \end{aligned}$$

在第三个例子中,站 A 发送“1”,站 B 发送“0”,其他站都没有任何动静。在第四个例子中,所有 4 个站都发送“1”。在最后一个例子中,A、B 和 D 发送位“1”,而 C 发送位“0”。请注意,图 2.45(c)给出的  $S_1 \sim S_6$  这 6 个序列中,每一个只代表一位的时间。

为了恢复每一个站的位流,接收器必须预先知道这个站的时间片序列。它只要计算出接收到的时间片序列与该站的时间片序列的归一化内积,就可以完成恢复工作。如果接收到的时间片序列为 S,接收器正在监听的站的时间片序列为 C,那么,它只要计算出归一化内积  $S \cdot C$  即可。

为了弄明白其中的道理,请考虑这样的情形:两个站 A 和 C 同时传输位“1”,并且 B 同时传输位“0”。接收器看到了和值  $S = A + \bar{B} + C$ ,然后计算:

$$S \cdot C = (A + \bar{B} + C) \cdot C = A \cdot C + \bar{B} \cdot C + C \cdot C = 0 + 0 + 1 = 1$$

前两项消失了,因为所有的时间片序列是精心挑选的,它们两两正交,满足等式(2-4)。现在你应该很清楚,为什么时间片序列必须具备这样的特性。

考虑这种情形的另外一种方法是,想象这三个时间片序列是各自单独到来的,而不是以总和的形式到达的。然后,接收器将分别与每一个时间片序列计算内积,再将结果加起来。由于正交特性,除了  $C \cdot C$  以外,其他的内积都是 0。实际上,先将它们加起来,再求内积,其结果与先分别求内积再加起来的结果相同。

为了使解码过程更加具体化,我们再次考虑图 2.45(c)给出的 6 个例子,解码过程如图 2.45(d)所示。假设接收器要从 6 个和值  $S_1 \sim S_6$  中提取出站 C 所发送的信息。为了提取站 C 所发送的位信息,接收器先将 S(当前接收到的向量)与图 2.45(b)中的向量 C 逐个分量求乘积,再将结果累加起来,然后取结果的  $1/8$ (因为这里的  $m=8$ )。如图中所示,每次都可以正确地将所发送的位计算出来。这就好像站 C 只喜欢讲法语一样。

在理想情况下,也就是在无噪声的 CDMA 系统中,系统的容量(即移动站的数量)可以为任意大,这是因为,只要为每个采样使用尽可能多的位,则无噪声的尼奎斯特信道的容量就可以任意大。但是,在实践中,一些物理上的限制大大降低了系统的容量。首先,我们已经假设所有的时间片都是同步的。在现实中,这种同步是不可能的。现实中的做法是,让发送方传送一个预定义的时间片序列,该时间片序列必须足够长,以便接收方可

以跟踪到它,然后发送方和接收方就可以保持同步。所有其他的(未同步的)传输都被看成随机噪声。然而,如果噪声不是特别多的话,基本的解码算法仍然可以工作得很好。这里有大量的理论问题涉及到时间片的重叠与噪声级别之间的关系(Pickholtz et al., 1982)。你可能已经想到了,时间片序列越长,则在有噪声情况下正确地检测到该时间片序列的可能性就越大。为了额外的可靠性,位序列可能会使用纠错码。但时间片序列永远不使用纠错码。

在我们的讨论中,一个隐含的假设是,所有移动站的功率值都与接收器所感知的功率值相同。CDMA 通常用于这样的无线系统:基站是固定的,而许多移动站离基站的距离是不一样的,有的距离近,有的距离远。由此,一种可以考虑的做法是,每个移动站给基站发送信号的时候,所使用的功率与它从基站收到的信号的功率成反比。换句话说,如果一个移动站收到的基站信号比较弱,则它使用较强的发送功率,大于那些获得强基站信号的移动站所使用的功率。基站也可以给出显式的命令,指示移动站增加或者降低它们的传输功率。

我们前面也假设了接收器知道发送方是谁。在理论上,接收器只要有足够的计算能力,它就可以并行地针对所有的发送方执行解码算法,因而可以监听所有这些发送方。而在现实中,只是这么说说而已,做起来并非易事。CDMA 还有许多其他复杂的因素,以上给出的简短介绍掩盖了这些因素。然而,CDMA 是一个非常聪明的方案,它迅速地被引入到无线移动通信中。通常 CDMA 运行在一段宽度为 1.25MHz 的频段上(相比之下,IS-AMPS 为 30kHz,GSM 为 200kHz),但是它在这个频段上支持的用户数量比其他两个系统都要多。在实践中,用于每个用户的带宽至少跟 GSM 一样多,而且通常是更多。

期望深入理解 CDMA 的工程师请阅读(Lee and Miller, 1998)。(Crespo et al., 1995)介绍了另一种扩展方案,不过这种扩展方案基于对时间的扩展,并非对频率的扩展。另外,(Sari et al., 2000)也描述了另外一种方案。所有这些参考资料都要求读者具备一点必要的通信工程的背景知识。

### 2.6.3 第三代移动电话:数字语音与数据

移动电话的未来是什么呢?我们现在来快速地浏览一下。有大量的因素正在驱动这个工业向前发展。首先,在固定网络上,数据流量已经超过了语音流量,而且还在呈指数增长,而语音流量的增长比较平缓。工业界的许多专家估计,在移动设备上,数据流量很快就会超过语音流量。其次,电话、娱乐和计算机工业都已经进入了数字化领域,并且正在快速地汇聚到一起。许多人非常期望有一种轻量的便携设备,它可以同时被用作电话、CD 播放器、DVD 播放器、电子邮件终端、Web 接口、游戏机、字处理器,以及所有通过无线方式连接到 Internet(高带宽)的应用。这种设备以及如何连接这种设备正是第三代移动电话的全部内容。有关更多的信息,请参看(Huber et al., 2000; and Sarikaya, 2000)。

回到 1992 年,ITU 试图清楚地描述这个梦想,当时发布了一份关于这个梦想的蓝图,称为 **IMT-2000**,其中 IMT 意为 **International Mobile Telecommunications**。数字 2000 代表了三层意思:(1)希望在 2000 年投入使用;(2)期望运行在 2000MHz 的频率上;

(3)期望这项服务应该具备 2000kHz 的带宽。

这个梦想并没有如期到来。到 2000 年时什么都没有实现。ITU 建议所有的政府保留 2GHz 的频段,以便这种设备可以无缝地从一个国家漫游到其他的国家。中国保留了所要求的带宽,但是其他没有一个国家这样做。最后,人们意识到,对于移动性太强的用户来说,2Mbps 现在有点不太切合实际(因为要想如此快速地完成移交过程非常困难)。比较现实一点的承诺是:对于比较固定的户内用户来说是 2Mbps(直接与 ADSL 进行竞争),对于正在路上行走的人来说是 384kbps,对于汽车内的连接则是 144kbps。然而,整个 3G 的领域是一个很庞大的体系。也许第三代移动电话略微逊色于最初的期望,也未能按原来预期的时间到来,但是,这一切很快会发生。

IMT-2000 网络预期给用户提供的服务包括:

- (1) 高质量的语音传输;
- (2) 消息服务(代替电子邮件、传真、SMS、聊天等);
- (3) 多媒体(播放音乐、观看视频、电影、电视等);
- (4) Internet 访问(Web 浏览,包括带音频和视频的页面)。

其他的服 务还可能包括视频会议、远程出场(telepresence)、群组游戏和移动商务(m-commerce,在商店里用移动电话来支付费用)。而且,所有这些服务都应该是全球性的(当不能找到地面网络的时候,可以通过卫星自动建立连接)、即时可完成的(总是可用的),并且有服务质量保证。

ITU 设想有一项全球性的 ITU-2000 技术,所以制造厂商们就可以生产出能在全 球销售和使用的设备(就好像 CD 播放器和计算机,而不像移动电话和电视机)。单一的技术也使得网络运营商的日子更加好过,并且可以鼓励更多的人使用这些服务。格式之争,比如当录像机刚出来时候 Betamax(Beta 制大尺寸磁带录像系统)与 VHS 之间的争斗,对于商业是不利的。

目前已经有了 一些提案,几经筛选之后,还留下两个主要的提案。第一个是 WCDMA(Wideband CDMA),这是由爱立信公司提出来的。该系统使用了直接序列扩频技术,如前面所描述。它运行在一个 5MHz 的频带上,并且已经设计成可以与 GSM 网络协同工作,但是它并不与 GSM 向后兼容。然而,它具备一种良好的特性,即呼叫者在离开一个 W-CDMA 蜂窝单元并进入另一个 GSM 单元的时候不会丢掉当前的呼叫。该系统由欧盟在艰难地向前推进,欧盟将它称为 UMTS(Universal Mobile Telecommunications System,全球通用的移动通信系统)。

另一个竞争者是 CDMA2000,由 Qualcomm 公司提出。它也是一种直接序列扩频设计方案,它基本上是 IS-95 的一个扩展,并且与 IS-95 向后兼容。它也使用了一段 5MHz 的带宽,但是不能与 GSM 协同工作,而且也不能将一个呼叫移交给一个 GSM 蜂窝单元(或者一个 D-AMPS 蜂窝单元)。与 W-CDMA 的其他技术区别包括使用不同的时间片率、不同的帧时间、不同的频谱,以及使用了不同的时间同步机制。

如果把爱立信和 Qualcomm 公司的工程师关进一个房间,并且告诉他们共同设计一个方案,也许他们会做得到。毕竟,两个系统背后的基本原理都是 5MHz 信道中的 CDMA,谁也不愿意因为坚持自己的时间片率而死掉。麻烦在于,实际的问题并非工程

性的,而是政治因素在起作用(如同以往一样)。欧洲希望有一个能与 GSM 协同工作的系统,而美国则希望要一个能与美国当前已经广泛使用的系统(IS-95)相互兼容的系统。每一方都支持自己本地的公司(爱立信公司的根据地在瑞典,而 Qualcomm 公司在美国加州)。最后,爱立信公司和 Qualcomm 都涉及到大量的有关 CDMA 专利的诉讼案。

1999 年 3 月,当爱立信公司同意购买 Qualcomm 公司的基本方案时,这两家公司终于解决了这些诉讼案。它们也同意遵守同一个 3G 标准,但是是一个有许多不兼容选项的标准,所以需要对技术的差异作出大量的书而说明。尽管有这些争议,3G 设备和服务可能会在接下去的几年时间中陆续出现。

关于 3G 系统已经有很多书面资料了,绝大多数资料都高度赞扬 3G,将它说成是自切片面包以来最伟大的事情。其中有以下这些参考材料:(Collins and Smith, 2001; De Vriendt et al., 2002; Harte et al., 2002; Lu, 2002; and Sarikaya, 2000)。然而,有些反对者认为,现在的工业界被指错了方向(Carber, 2002; and Goodman, 2000)。

在等待 3G 争斗平息下来的同时,有些运营商正在小心翼翼地往 3G 的方向上迈出的步伐。它们的做法是,先进入到一个称为 2.5G 的领域中,其实,更为精确的叫法应该是 2.1G。其中一个系统是 **EDGE(Enhanced Data rates for GSM Evolution)**,它只是一个支持每波特更多数据位的 GSM 系统。麻烦之处在于,每波特更多数据位也意味着每波特更多的错误,所以 EDGE 有几种不同的调制和纠错方案,这些方案对于“使用多少带宽来修正这些因为更高的速度而引入的错误”各有不同的解决办法。

另一个 2.5G 方案是 **GPRS(General Packet Radio Service)**,它是一个位于 D-AMPS 或者 GSM 之上的层叠分组网络。它允许移动站在一个运行语音系统的蜂窝单元中发送或者接收 IP 分组。当 GPRS 在运行的时候,某些频率上的有些时槽被保留用于分组流量。这些时槽的数量和位置可以由基站动态地管理,取决于该蜂窝单元中语音和数据流量的比率。

所有可用的时槽被分成几条逻辑信道,分别用于不同的目的。基站决定哪些逻辑信道被映射到哪些时槽上。一条逻辑信道被用于下载分组,即从基站下载到某个移动站中,每个分组指示了谁是它的目标。为了发送一个分组,移动站先向基站发送一个请求,请求一个或者多个时槽。如果该请求完好无损地到达基站,则基站广播它的应答,其中包括分配给该移动站用于发送分组的频率和时槽。一旦分组已经到达了基站,则它将通过一个有线连接被传输到 Internet 上。由于 GPRS 只是现有语音系统之上的一个层叠系统,所以它最多是 3G 到来之前的一个折衷方案。

尽管 3G 网络还没有完全部署起来,但是,有些研究人员认为 3G 是一个业已完成的研究题目,因此不再对 3G 有任何兴趣了。这些人已经开始研究 4G 系统了(Berezdivin et al., 2002; Guo and Chaskar, 2002; Huang and Zhuang, 2002; Kellerer et al., 2002; and Misra et al., 2002)。关于 4G 系统,一些已经考虑的特性包括高带宽、普遍适用性(任何地方都可以连接)、与有线网络(特别是 IP 网络)的无缝集成、自适应资源和频谱的管理、软件无线接收器,以及多媒体数据的高服务质量。

另一方面,有大量的地方已经安装了 802.11 无线 LAN 访问点,所以,有些人认为 3G 不仅不是一个业已完成的课题,而是一个注定必死的课题。按照这种观点,人们将从一个

802.11 访问点漫游到另一个访问点,并且仍保持连接状态。要说工业界充满了各种潮流,这话一点也不过分。让我们 5 年之后再回头来看一下到底发生了些什么。

## 2.7 有线电视

我们已经相当细致地学习了固定的和无线的电话系统了。很明显,在将来的网络中,它们都将扮演重要的角色。然而,另一种可用于固定网络连接的方法现在也日渐变得重要起来,那就是有线电视网络。许多人已经将他们的电话和 Internet 服务放到有线电视网络上了,而且有线电视运营商也正在积极地推动这项工作,以便提高他们的市场份额。在本节中,我们将从网络系统的角度来详细地介绍有线电视系统,并且将它与我们刚刚学习过的电话系统作一对比。有关有线电视系统更多的信息,请参看(Laubach et al., 2001; Louis 2002; Ovadin, 2001; and Smith, 2002)。

### 2.7.1 共天线电视

关于有线电视的构想在 20 世纪 40 年代后期就已经有了,当时的目的主要是为了给居住在农村和山区的人提供更好的收视效果。系统最初包括以下几个部分:一个大的天线,一般放在山顶上,以便将电视信号从空中接收下来;一个放大器,也称为头端(headend),它可以加强信号;以及一根同轴电缆,系统通过该电缆将电视信号送到用户的家里,如图 2.46 所示。



图 2.46 一个早期的有线电视系统

在早期的时候,有线电视称为共天线电视(Community Antenna Television)。它是一种非常小规模的模式:任何人配备一些电子设备就可以在他的城镇建立这样的服务,然后,用户凑一些钱来支付所有的费用。随着用户数量的增加,在原来电缆的基础上,还需要接入额外的电缆,并且根据需要还要加入放大器。在这样的系统中,信号传输是单向的,从头端传输到用户处。在 1970 年以前,有几千个这样的独立系统存在。

1974 年,时代公司(Time, Inc.)开辟了一个新的频道: Home Box Office,该频道播放一些新的内容(电影),并且只通过电缆广播。随后又出现了一些只在电缆上广播的频道,内容涉及到新闻、体育、烹饪,以及其他许多话题。这种发展引起了工业界的两个变化。第一,大的公司开始购买已有的电视系统,并且铺设新的电缆以便赢得新的用户。第二,出现新的需求:将多个电视系统连接起来。通常有必要将相距较远的城市连接起来



以便新的电视频道能够传播到多个城市。有线电视公司开始在这些城市之间铺设电缆，以将这些城市连接到同一个系统中。这种模式非常类似于电话工业在 80 年前所发生的事情，当时电话公司将孤立的端局连接起来以便能够提供长途电话服务。

### 2.7.2 基于有线电视网络的 Internet

有线电视系统经过多年的发展成熟和壮大，各个城市之间的电缆已经被替换成了高带宽的光纤，这个过程非常类似于电话系统的发展历程。如果一个有线电视系统中长距离路径使用的是光纤，而连接到家庭的是同轴电缆，则这样的系统称为 HFC (Hybrid Fiber Coax) 系统。系统的光学部分和电子部分之间的接口是一些光电转换器，称为光纤节点 (fiber node)。因为光纤的带宽远远超过同轴电缆的带宽，所以，一个光纤节点可以连接多根同轴电缆。图 2.47(a) 显示了一个现代的 HFC 系统的一部分。

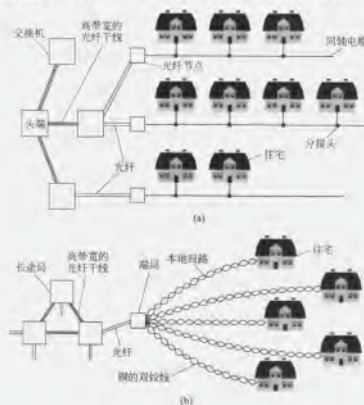


图 2.47  
(a) 有线电视网络；(b) 传统的电话系统

最近几年来，许多有线电视运营商已经决定介入到 Internet 访问业务中，通常也会介入到电话业务中。然而，有线电视设备和电话设备之间的技术差异影响到应该如何来实现这些目标。首先而言，有线电视系统中的单向放大器必须要被转换成双向放大器。

然而，图 2.47(a) 所示的 HFC 系统与图 2.47(b) 所示的电话系统之间还存在一个难

以克服的差异。在有线电视系统的下游邻居中,许多用户共享了一根同轴电缆,而在电话系统中,每个家庭都有自己私有的本地回路。当这根共享电缆被用于电视广播的时候,这种共享特性并不成为问题。所有的电视节目都在这根电缆上广播,无论有 10 个用户还是 1 万个用户都没有问题。当同样这根电缆用于 Internet 访问时,则 10 个用户与 1 万个用户就有很大的不同了。如果一个用户决定下载一个非常大的文件,则他可能会将所有的带宽(包括许多别的用户的带宽)都拿过来使用了。用户数量越多,则带宽的竞争越激烈。电话系统并不存在这样的特点:通过 ADSL 线路下载一个大的文件并不会降低邻居的带宽。另一方面,同轴电缆的带宽远远高于双绞线的带宽。

有线电视工业界解决这个问题的办法是将长的电缆分解开(截短),然后将每一段直接连接到光纤节点上。从末端到每一个光纤节点的带宽几乎是无限的,所以,只要每一段电缆上没有太多的用户,则流量还是可以管理的。现在一根典型的电缆通常有 500 至 2000 户家庭,但是,随着越来越多的用户通过电缆连接到 Internet,负载会变得非常大,因而要求更加细粒度地分解电缆,并引入更多的光纤节点。

### 2.7.3 频谱分配

如果有线电视公司放弃所有的电视频道,将电缆设施完全用于 Internet 访问,则这样做可能会激怒大量的顾客,所以电视公司一直不敢这样做。而且,大多数城市对于电缆上广播的内容都是有严格管制的,所以,即使有线电视运营商真的想这样做,他们也得不到许可。因此,他们需要找到一种办法让电视和 Internet 在同一根电缆上共存。

在北美,有线电视频道通常占用 54~550MHz 范围(除了 88~108MHz 用于 FM 无线电台以外)。这些电视频道都是 6MHz 宽,其中包括防护频段。在欧洲,低端通常是 65MHz,由于 PAL 和 SECAM 所要求的分辨率较高,所以每个频道为 6~8MHz 宽,除此以外,在其他方面分配方案非常类似。频带的较低部分并未使用。现代的电缆在 550MHz 以上也能工作得很好,通常可以达到 750MHz 或者更高。这里选择的一种分配方案是,上行信道在 5~42MHz 频段(在欧洲还可以更高一些),并使用高端的频率作为下行数据流。电缆的频谱如图 2.48 所示。



图 2.48 在一个典型的用于 Internet 访问的有线电视系统中频率的分配情况

请注意,由于电视信号全部是下行的,所以,一种可能的做法是,上行放大器工作在 5~42MHz 区域范围内,下行放大器工作在 54MHz 以上,如图中所示。因此,在这种

方案中,上行和下行带宽是非对称的,因为电视之上的频段范围大于电视之下的频段范围。另一方面,大多数流量可能是下行的,所以有线电视运营商也不会对此感到不满。正如我们以前所看到过的,电话公司即使没有任何技术理由的情况下,通常也会提供一种非对称的 DSL 服务。

在传输数字信号的时候,长的同轴电缆也不比长的本地回路好多少,所以,这里也需要模拟调制。常见的方案是取每个 6MHz 或者 8MHz 的下行信道,然后用 QAM-64 进行调制,或者,如果电缆质量特别好的话,可以使用 QAM 256 进行调制。在 6MHz 信道和 QAM-64 的情况下,我们可以得到 36Mbps。当减去各种开销之后,剩下的有效净荷大约为 27Mbps。如果使用 QAM-256 的话,则有效净荷大约为 39Mbps。欧洲的值大约比这大 1/3。

对于上行信道,即使 QAM 64 也工作得不是很好。来自地面微波、CB(民用)无线电波以及其他信号源有太多的噪声,因此需要使用一种更加保守一点的调制方案——QPSK。利用这种调制方法(如图 2.25),每个波特只产生 2 位数据,而不像下行信道中 QAM 方案每个波特产生 6 位或者 8 位数据。因此,上行带宽与下行带宽之间的非对称性比图 2.48 中显示的非对称性还要严重。

除了更换放大器以外,运营商也必须更换头端,将它从一个哑的放大器替换成一个智能的数字计算机系统,并且通过一个高带宽的光纤接口将它连接到一个 ISP。通常连名字也需要更换,从原来的“头端”变成 CMTS(Cable Modem Termination System,电缆调制解调器终端系统)。在下面的介绍中,我们将坚持使用传统的“头端”,而不使用更换之后的名字 CMTS。

#### 2.7.4 电缆调制解调器

为了通过电视电缆访问 Internet,电缆调制解调器(cable modem)是必要的,它是一个具有两个接口的设备:一个接口连接到计算机,另一个接口连接到有线电视网络。在有线电视连接 Internet 的早期,每个运营商都有自己私有的电缆调制解调器,并且由有线电视公司的技术员来安装。然而,很快局势变得明朗了:有一个开放的标准将会造就一个电缆调制解调器的竞争市场,并且可以把价格压下来,从而鼓励更多的用户使用这项服务。而且,让顾客自己到商店里购买电缆调制解调器并回家安装(就好像使用 V.9x 电话调制解调器一样),这样可以避免可怕的上门服务。

因此,大一点的有线电视运营商与一家称为 CableLabs 的公司合作,以生成一个电缆调制解调器标准,并且测试产品的兼容性。这个标准称为 DOCSIS(Data Over Cable Service Interface Specification),现在已经开始代替原来私有的调制解调器。欧洲的版本称为 EuroDOCSIS。然而,并不是所有的运营商都喜欢这种建立标准的做法,因为他们通过租借调制解调器的方式从顾客手里赚取了不少钱。一个开放的标准允许大量的厂商通过商店销售电缆调制解调器,从而也就结束了这种轻松赚钱的机会。

从调制解调器到计算机的接口是非常简单的,目前通常是 10Mbps 的以太网(或者偶尔也可以是 USB)。将来,整个调制解调器可能是一块很小的卡,可以直接插到计算机内部,就好像 V.9x 内置调制解调器一样。

另一端要复杂一些。该标准的大部分内容涉及到无线电工程,这个主题已经超出了本书的范畴。惟一值得在这里提及的是,电缆调制解调器如同 ADSL 调制解调器一样,总是保持连接的。当电缆调制解调器被打开的时候,它们就会建立连接,而且,只要一直通电,它们就会维持该连接,因为电缆运营商并不按连接时间进行收费。

为了更好地理解电缆调制解调器是如何工作的,我们来看一看当电缆调制解调器被插入到计算机中并且通电以后会发生什么事情。调制解调器扫描下行信道,以寻找一个特殊的分组;该分组是由头端定期发送的,头端通过该分组向刚刚上线的调制解调器提供系统参数。新的调制解调器找到了该分组以后,就在某一个上行信道中宣布它的存在。头端作出响应,并为新的调制解调器分配上行和下行信道。以后,如果头端认为有必要平衡负载的话,这些上行和下行信道的分配方案还可以改变。

然后,调制解调器确定它离头端有多远,其做法是,先发送一个特殊的分组,再看需要多长时间才得到应答。这个过程称为测距(ranging)。对于调制解调器而言,知道它与头端之间的距离是非常重要的,这样它可以调节上行信道的操作方式,并且保证正确地控制好时间。这些上行信道按照时间分成微时槽(minislot)。每个上行分组必须被装配到一个或者多个连续的微时槽中。头端定期地宣布新一轮的微时槽开始了(相当于发令枪),但是,由于沿着电缆传播的时间并不完全相同,所以,所有的调制解调器不可能同时听到发令枪响。由于每个调制解调器知道它离头端有多远,所以它可以计算出第一个微时槽真正开始的确切时间(即多久之前开始的)。微时槽的长度与具体的网络有关。一个典型的净荷域是 8 字节长。

在初始化过程中,头端也为每个调制解调器分配一个微时槽,用于请求上行信道的带宽。通常,多个调制解调器被分配到同一个微时槽中,这样会导致多方竞争。当一台计算机想要发送分组的时候,它先将分组传输给调制解调器,然后调制解调器为该分组请求一定数量的微时槽。如果该请求被头端接受的话,头端通过下行信道发送一个确认,告诉该调制解调器哪些微时槽已经被保留给这个分组了。然后,当所分配的微时槽到来的时候,该分组就可以被发送出去了。如果还有其他的分组,则可以利用头部的一个域来请求发送这些分组。

另一方面,如果请求微时槽时存在竞争的话,则调制解调器不会收到确认,于是,它等待一段随机长的时间,然后再重试这个过程。每次尝试失败以后,这段随机时间的长度就加倍。(熟悉网络的读者可能知道,该算法只不过是采用了二元指数后退方法的分槽 ALOHA。在电视电缆上不能使用以太网,因为用户站不能检测介质的状态。我们将在第 4 章再回到这些问题上。)

下行信道的管理与上行信道不同。首先,下行信道只有一个发送者(头端),所以不会发生竞争,因此不需要微时槽,实际上,下行信道只是一个时分统计多路复用的过程。其次,下行的流量通常远远大于上行流量,所以,下行信道使用了 204 字节的固定分组长度。在 204 字节的分组中,其中包含了 Reed-Solomon 纠错码和其他一些开销,留给用户的净荷只有 184 字节。之所以选择这些数字,是考虑到与 MPEG-2 数字电视保持兼容,所以,电视信道和下行数据信道使用了同样的格式化方法。图 2.49 显示了电缆的总体逻辑结构。

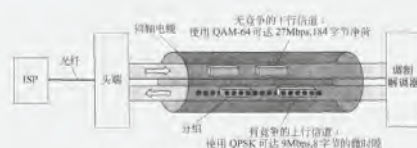


图 2.49 在北美地区,上行和下行信道的细节情况

现在再回到调制解调器的初始化过程,一旦调制解调器已经完成了测距,并且得到了它的上行信道、下行信道和微时槽分配情况,它就可以自由地发送分组了。它发送的第一个分组是给 ISP,请求一个 IP 地址;ISP 接到请求后,利用一个称为 DHCP 的协议动态地分配一个 IP 地址。我们将在第 5 章学习 DHCP 协议。调制解调器也向头端请求并获得精确的时间信息。

接下去的步骤涉及到安全性问题。由于电缆是共享的介质,任何想制造麻烦的人都可以连接上来;并且读取到所有经过他的流量。为了阻止每个人直接窃听到邻居的信息,两个方向上的所有流量都要经过加密。所以,初始化过程的一部分工作是建立起加密用的密钥。你可能会认为,两个陌生的人(头端和调制解调器)要在众目睽睽之下建立起一个秘密的密钥是不可能的。事实并不如你所想的那样,不过,我们要等到第 8 章再来解释这是怎么做到的(简单的答案是:使用 Diffie-Hellman 算法)。

最后,调制解调器必须通过安全信道登录到系统中,并提供它自己的唯一标识符。这时初始化过程才算真正完成了。现在用户可以登录到 ISP,并开始工作了。

关于电缆调制解调器还有很多内容需要解释。一些相关的参考文献是 Adams and Dulchinos, 2001; Donaldson and Jones, 2001; and Dutta-Roy, 2001。

## 2.7.5 ADSL 与有线电视网

ADSL 与有线电视网哪一个更好? 这就好像在问哪一个操作系统更好,或问哪一种语言更好,或问哪一种信仰更好一样。你会得到哪一个答案呢? 这要取决于你问的是谁。下面我们从几个方面来比较一下 ADSL 和有线电视网。在骨干网上,这两者都使用了光纤,但是在边界上它们使用了不同的介质。有线电视网使用了同轴电缆,而 ADSL 则使用了双绞线。从理论上讲,同轴电缆的承载容量超过双绞线几百倍。然而,在有线电视网中,电缆的全部容量并不都被用于数据用户,因为电缆的大部分带宽被浪费在诸如电视节目这样的无用材料上了。

在实践中,有线电视供应商很难统一地说明用户可以获得多大的有效带宽容量。而 ADSL 供应商则可以给出一份关于带宽的声明(即下行流量 1Mbps,上行流量 256kbps),并且一般可以达到 80%。有线电视供应商没有作这样的声明,因为电缆的有效容量取决于当前有多少人连接在用户的电缆段上。有时候它比 ADSL 更好,但有时候可能还不如 ADSL。不过,真正让人觉得讨厌的地方是它的不可预知性。上一分钟有很好的服务并



不能保证下一分钟还有很好的服务,因为说不定正好这时候附近有一个带宽贪婪户刚刚打开了他的计算机。

当 ADSL 系统赢得了越来越多用户的时候,新增加的用户对于原来的用户几乎没有影响,因为每个用户都有自己专用的连接。但是在有线电视网络中,当申请 Internet 服务的用户越来越多的时候,原有用户的性能将会下降。惟一的解决办法是,运营商将繁忙的电缆分解成多段,然后将每一段直接连接到光纤节点上。这样做既耗时,也耗财,所以,出于业务的压力,应该尽量避免这样的事情。

另外,我们已经学习过另一个也像有线电视网一样具有共享信道的系统:移动电话系统。在这里,一群用户也共享了固定数量的带宽,我们将这群用户称为蜂窝单元户 (cellmate)。通常情况下,系统利用 FDM 和 TDM,将当前活动的用户严格地划分到固定的块中,因为语音流量非常平滑。但是,对于数据流量,这种严格的划分非常低效,因为数据用户往往是空闲的,在这种情况下,为他们保留的带宽就会被浪费。不过,从这个角度来看,有线电视用户访问 Internet 更像是移动电话系统而不像固定的系统。

可用性也是 ADSL 和有线电视网非常不一样的地方。每个人都有一部电话,但是并不是所有的人都离端局那么近,以便可以申请到 ADSL。另一方面,并不是所有的人都连接了有线电视,但是,如果你已经有了有线电视,并且有线电视公司确实提供 Internet 访问服务,那么,你就可以使用这项服务。离光纤节点或者头端的距离并不成问题。同时值得注意的是,由于电视网的电缆最初是作为电视传播介质而建立起来的,所以它承载的商业服务非常少。

ADSL 作为一种点到点的介质,本质上要比电视电缆安全得多。任何有线电视用户都可以很容易地读取同一电缆上传输的所有分组。出于这个原因,任何正规的有线电视供应商都会加密两个方向上的所有流量。然而,让你的邻居得到加密之后的消息,当然不如让他什么也得不到更安全。

一般而言,电话系统比有线电视系统更加可靠。例如,它有备份电源,即使停电了它还能继续工作。而在有线电视系统中,如果到任何一个放大器的电源断了,则所有的下游用户立即就会被中断。

最后,绝大多数 ADSL 供应商都允许用户选择 ISP。有时候,甚至有法律要求他们这样做。但是,对于有线电视运营商,事情并不总是这样的。

结论是,ADSL 和有线电视网相似的成分比差异的地方更多。它们提供了可相互类比的服务,而且,随着两者之间竞争的白热化,它们所提供的服务的价格也可能有一定程度的可比性。

## 2.8 本章小结

物理层是所有网络的基础。自然界对所有的信道强加了两个基本的限制,它们决定了信道的带宽。这两个限制是:针对无噪声信道的尼奎斯特限制和针对有噪声信道的香农限制。

传输介质可以是有导向的,也可以是无导向的。基本的有导向介质是双绞线、同轴电

缆和光纤。无导向介质包括空中的无线电波、微波、红外线和激光。一个很快就要到来的传输系统是卫星通信,特别是 LEO 系统。

在大多数广域网络中一个关键的要素是电话系统。电话系统的主要部件是本地回路、干线和交换机。本地回路是模拟的双绞线,为了传输数字数据,本地回路上要求有调制解调器。ADSL 提供的速度可以达到 50Mbps,它的做法是将本地回路分成许多虚拟信道,然后单独调制每一个信道。无线本地回路是另一种新的发展形式,特别是 LMDS。

干线是数字的,可以按照几种不同的方式实现多路复用,包括 FDM、TDM 和 WDM。电路交换和分组交换都是非常重要的。

对于移动应用,固定的电话系统是不适合的。移动电话日前已经被广泛应用于语音通信,很快将被广泛应用到数据通信中。第一代是模拟的,AMPS 占了统治地位。第二代是数字的,其中 D-AMPS、GSM 和 CDMA 是主要的选择方案。第三代将是数字的,并且以宽带 CDMA 为基础。

另外一种访问网络的系统是有线电视系统,它已经从共天线的形式逐步发展到光纤同轴电缆混合的形式。它潜在的优势是可以提供非常高的带宽,但是在实践中,实际可用的带宽主要取决于当前其他活动用户的数量以及他们在做什么事情。

## 习 题

1. 计算函数  $f(t)=t$  ( $0 \leq t \leq 1$ ) 的傅立叶系数。
2. 一条无噪声 4kHz 信道按照每 1ms 一次进行采样,请问最大数据传输率是多少?
3. 电视频道的带宽是 6MHz。如果使用 4 级数字信号,则每秒钟可以发送多少位? 假设电视频道为无噪声信道。
4. 如果在一条 3kHz 的信道上发送一个二进制信号,该信道的信噪比为 20dB,则最大可达到的数据传输率为多少?
5. 在 50kHz 的线路上使用 T1 线路需要多大的信噪比?
6. 在一个光纤网络中,无源星形结构和活动中继器之间的区别是什么?
7. 在  $1\mu\text{m}$  波长上,在  $0.1\mu\text{m}$  的频段中有多少带宽?
8. 现在需要在一条光纤上发送一系列计算机屏幕图像。屏幕的分辨率为  $480 \times 640$  像素,每个像素为 24 位。每秒钟有 60 幅屏幕图像。请问:需要多少带宽? 在  $1.30\mu\text{m}$  波长上,这段带宽需要多少  $\mu\text{m}$  的波长?
9. 尼奎斯特定理对于光纤成立吗? 还是它只适用于铜线?
10. 在图 2.6 中,左边的波段比其他的波段窄。这是为什么?
11. 当无线电天线的直径等于无线电波波长的时候,天线通常工作得最好。常见的天线直径范围 1~5m。请问这将覆盖多大的频率范围?
12. 当两束光以 180 度相位差到达的时候,多径衰减是最大的。对于 50km 长的 1GHz 微波链路,为了使衰减最大,要求多大的路径差?
13. 一束 1mm 宽的激光对准了 100m 开外的建筑物顶上的一个检测器。请问:若要使该激光束偏离检测器,则激光束必须偏离多大的角度?

14. Iridium 计划中的 66 颗低轨道卫星被分成绕着地球的 6 条项链。在它们所使用的高度上,周期是 90 分钟。对于一个固定的发射器,移交的平均间隔是多少?

15. 考虑一颗在同步卫星高度上的卫星,但是它的轨道平面与赤道平面倾斜成  $\varphi$  角。对于地球表面北纬  $\varphi$  度的一个固定用户,该卫星在空中静止不动吗?如果不是的话,请描述它的运动。

16. 在 1984 年以前有多少个端局?当时每个端局是由它的 3 位区域码和前 3 位本地号码来命名的。区域码的第 1 位是 2~9 之间的数字,第 2 位是 0 或者 1,第 3 位可以是任何一个数字。本地号码的前两位数字总是在 2~9 之间,第 3 位可以是任何数字。

17. 仅仅利用正文中给出的数据,请问:如果不改变电话号码编码计划或者不增加额外的设备,美国现有的电话系统最多可以支持多少部电话?这么多的电话数量实际可以达到吗?回答这个问题的时候,假定一台计算机或者传真机也算作一部电话,并且每条用户线路上只有一个设备。

18. 一个简单的电话系统包括两个端局和一个长途局,每个端局通过一条 1MHz 全双工干线连接到长途局。在每 8 小时的工作日中,平均每部电话有 4 次呼叫,每次呼叫平均 6 分钟,10% 的呼叫是长途(即通过长途局)。请问一个端局能够支持最多多少部电话?(假设每条线路为 4kHz)

19. 一个区域电话公司有 10M 个用户。每部电话通过双绞线连接到一个中心局。这些双绞线的平均长度为 10 公里。请问本地回路中的铜价值多少?假设每束线的横截面是一个直径为 1mm 的圆,铜的密度是  $9.0\text{g/cm}^3$ ,并且每 kg 铜可以卖 3\$。

20. 请问石油管道是单工系统,还是半双工系统,或是全双工系统,或者三者都不是?

21. 高速微处理器的价格已经降低到有可能在每个调制解调器中都安装一个这样的微处理器。请问:这对电话线路的错误处理有什么影响?

22. 一个类似于图 2.25 的调制解调器星座图有以下几个坐标点:  $(1, 1)$ 、 $(1, -1)$ 、 $(-1, 1)$  和  $(-1, -1)$ 。请问一个具备这些参数的调制解调器在 1200 波特上可以达到多少 bps?

23. 一个类似于图 2.25 的调制解调器星座图有两个坐标点  $(0, 1)$  和  $(0, 2)$ 。请问该调制解调器使用相位调制还是振幅调制?

24. 在一个星座图中,所有的点都位于一个以圆点为中心的圆上。请问它使用了哪种调制方案?

25. 一个全双工的 QAM-64 调制解调器使用了多少频率?

26. 一个使用 DMT 的 ADSL 系统将 3/4 可用数据信道都分配给下行链路了。在每条信道上它使用 QAM-64 调制方法。请问下行链路的容量是多少?

27. 在图 2.30 的 4 扇区 LMDS 例子中,每个扇区有它自己的 36Mbps 信道。根据排队理论,如果一条信道有 50% 的负载,则排队时间将等于传输时间。在这些条件下,下载一个 5KB 的 Web 页面需要多长时间?通过 1Mbps 的 ADSL 线路,下载这样的页面需要多长时间?通过 56kbps 的调制解调器呢?

28. 有 10 个信号,每个都要求 4000Hz,现在用 FDM 将它们复用在一条信道上。对于被复用的信道,最小要求多少带宽?假设防护频段为 400Hz 宽。

29. 为什么 PCM 采样时间被设置为  $125\mu\text{s}$ ?
30. T1 线路上额外开销的百分比为多少? 也就是说, 1.544Mbps 中有百分之多少没有被递交给最终用户?
31. 比较使用以下方案的 4kHz 无噪声信道的最大数据传输率:
- (a) 每次采样 2 位的模拟编码(比如 QPSK)。
  - (b) T1 PCM 系统。
32. 如果一个 T1 线路系统失去了同步, 它试图使用每一帧的第 1 位来重新获得同步。请问, 平均要检查多少帧才能保证在出错概率为 0.001 的情况下重新获得同步。
33. 调制解调器的解调部分与编解码器的编码部分有没有区别? 如果有的话, 区别是什么? (之所以有此问, 是因为两者都将模拟信号转换成数字信号。)
34. 一个信号在 4kHz 的无噪声信道上以数字方式进行传输, 每  $125\mu\text{s}$  采样一次。请问, 按照以下的编码方法, 每秒钟实际发送多少位?
- (c) CCITT 2.048Mbps 标准。
  - (d) 有 4 位相对信号值的 DPCM。
  - (e) 增量调制。
35. 一个振幅为 A 的纯正弦波, 用增量调制进行编码, 每秒采样 x 次。输出 +1 对应于信号变化  $+A/8$ ; 输出 -1 对应于信号变化  $-A/8$ 。那么, 在没有错误累积的情况下, 可以跟踪的最大频率是多少?
36. SONET 时钟的漂移率大约是  $10^{-9}$ 。请问, 经过多长时间才使漂移等于 1 位的宽度? 该结果有什么含义?
37. 在图 2.37 中, OC-3 的用户数据传输率为 148.608Mbps。请推导一下, 该数值是如何从 SONET OC-3 的参数得来的。
38. 为了调节到低于 STS-1 的数据传输率, SONET 有一个虚拟支流系统(VT, virtual tributary)。一个 VT 是指一部分净荷, 它们可以被插入到一个 STS-1 帧中, 并且与其他的净荷部分组合起来构成数据帧。VT1.5 使用 STS-1 帧的 3 列, VT2 使用 4 列, VT3 使用 6 列, VT6 使用 12 列。请问, 哪个 VT 可以调节到以下的系统:
- (f) DS-1 服务(1.544Mbps)?
  - (g) 欧洲的 CEPT-1 服务(2.048Mbps)?
  - (h) DS-2 服务(6.312Mbps)?
39. 消息交换和分组交换之间的本质区别是什么?
40. 在 OC-12c 连接中, 可用的用户带宽是多少?
41. 三个分组交换网络每个包含 n 个节点。第一个网络是一个星形拓扑结构, 有一个中心交换机; 第二个网络是一个双向环; 第三个网络是一个全连接结构, 从任何一个节点到其他的节点都有一条线路。请问, 从传输路径的跳数来看, 哪个最好? 哪个其次? 哪个最差?
42. 请比较一下在一个电路交换网络中和在一个(负载较轻的)分组交换网络中, 沿着 k 跳的路径发送一个 x 位消息的延迟情况。电路建立的时间为 s 秒, 每一跳的传播延迟为 d 秒, 分组的大小为 p 位, 数据传输率为 b bps。在什么条件下分组网络的延迟比

较短?

43. 假定  $x$  位用户数据将以一系列分组的形式, 在一个分组交换网络中沿着一条共有  $k$ -跳的路径向前传输, 每个分组包含  $p$  位数据和  $h$  位的头, 这里  $x \gg p + h$ 。线路的传输率为  $b$  bps, 传播延迟忽略不计。请问, 什么样的  $p$  值使总延迟最小?

44. 在一个典型的移动电话系统中, 蜂窝单元为六角形, 在相邻的单元内禁止重新使用频段。如果总共有 840 个频率可以使用的话, 则任何一个给定的单元内可以使用多少个频率?

45. 蜂窝单元的实际布局结构很少会像图 2.41 那样规则。即使单个蜂窝单元的形状也往往是不规则的。请给出一个可能的理由说明为什么会这样?

46. 请粗略地估算一下, 为了覆盖旧金山(120 平方公里), 需要多少个 PCS 微蜂窝单元? 每个微蜂窝单元的直径为 100m。

47. 有时候, 当一个移动用户跨越两个蜂窝单元边界的时候, 当前的电话呼叫会被突然中止, 即使所有的发射器和接收器都在正常工作。为什么?

48. D-AMPS 的语音质量明显不如 GSM。这是否是由于 D-AMPS 必须与 AMPS 保持向后兼容, 而 GSM 没有这样的限制造成的? 如果不是的话, 那么真正的原因是什么?

49. 请计算一下 D-AMPS 在同一个蜂窝单元内能够并发支持的最大用户数量。同样也计算 GSM 能够并发支持的最大用户数量。然后解释两者的区别。

50. 假设 A、B 和 C 通过一个 CDMA 系统同时传输位 0, 他们的时间片序列如图 2.45(b)所示。请问结果得到的时间片序列是什么?

51. 在关于 CDMA 时间片序列正交性的讨论中, 提到了: 如果  $S \cdot T = 0$ , 则  $S \cdot \bar{T}$  也是 0。请给出证明。

52. 考虑用另一种方式来看待 CDMA 时间片序列的正交特性。序列对中的每一位要么匹配, 要么不匹配。请从匹配和不匹配的角度来表达正交特性。

53. 一个 CDMA 接收器得到了下面的时间片:  $(-1+1-3+1-1-3+1+1)$ 。假设时间片序列如图 2.45(b)中所定义, 请问哪些移动站传输了数据? 每个站发送了什么位?

54. 在低端, 电话系统是星形的, 邻近范围内的所有本地回路都集中到一个端局。相反, 有线电视网的低端则利用一条长长的电缆, 沿途经过邻近范围内所有的家庭。假定将来的有线电视电缆是 10Gbps 的光纤, 而不再是铜线。请问, 它可以被用来模拟电话模型(每个人都有自己私有的线路连接到端局)吗? 如果可以的话, 请问一根光纤上可以挂接多少户单电话家庭?

55. 一个有线电视系统有 100 个商业频道, 所有这些频道都可以在节目和广告之间进行轮流切换。请问它更像 TDM 还是更像 FDM?

56. 一个有线电视公司决定在一个包含 5000 户家庭的区域内提供 Internet 访问服务。该公司使用一根同轴电缆, 它的频谱分配方案允许每根电缆有 100Mbps 的下行带宽。为了吸引顾客, 该公司决定, 保证每户家庭在任何时候都至少有 2Mbps 的下行带宽。请描述一下该公司需要采取什么措施才能提供这样的保证。

57. 利用图 2.48 中显示的频谱分配方案, 以及正文中给出的信息, 请计算一下, 一个有线电视系统分配给上行信道有多少 Mbps? 给下行信道多少 Mbps?



58. 如果有线电视网络是空闲的话,请问一个用户接收数据可能有多快?

59. 多个 STS-1 数据流的复用过程在 SONET 中扮演了非常重要的角色,这些 STS-1 数据流称为支流(tributary)。一个 3:1 复用器将三个输入 STS-1 支流复用到一个输出 STS-3 流中。复用过程是按字节进行的,也就是说,前三个输出字节分别是支流 1、2 和 3 的第一个字节;接下去的三个字节分别是支流 1、2 和 3 的第二个字节;以此类推。请编写一个程序来模拟这样的 3:1 复用器。你的程序应该包含 5 个进程。主进程创建 4 个进程,其中三个进程分别对应于三个 STS-1 支流,另一个对应于复用器。每个支流进程从一个输入文件中读入 810 字节序列作为一个 STS-1 帧。它们将这些帧(逐个字节)发送给复用器进程。复用器进程接收这些字节,然后(逐个字节地)输出一个 STS-3 帧(将该帧写到标准输出设备上)。进程之间的通信请使用管道。

---

## 第3章 数据链路层

---

本章中,我们将学习网络的第2层——数据链路层的设计原则。我们将主要学习在数据链路层上,两台相邻机器之间实现可靠、有效的通信而涉及到的一些算法。所谓相邻,意思是指两台机器通过一条通信信道连接起来,这里的通信信道在概念上就像一条线(比如同轴电缆、电话线或者点到点的无线信道)。一条信道像一条线,这也暗示了它的一个本质特性,即在一条信道上递交的数据位的顺序与发送的顺序完全相同。

刚开始,你可能认为这个问题非常简单,所以没有什么内容需要学习——机器A把数据位放到线路上,然后机器B将这些数据取下来。不幸的是,通信线路偶尔可能会有错误。而且,它们只有有限的数据传输率,并且,在一位数据的发送时刻和接收时刻之间存在一定的传输延迟。这些限制对于数据传输的效率有非常重要的影响。用于通信过程的协议必须考虑所有这些因素。这些协议正是本章的主题。

在介绍了数据链路层的设计要点之后,我们将通过考察错误的本质、出错的原因,以及如何检测和纠正这些错误来研究数据链路层的协议。然后,我们将学习一系列逐渐复杂的协议,每一个协议都解决了这一层中越来越多的问题。最后,我们将讨论协议的模型和正确性,并给出一些数据链路层协议的例子。

### 3.1 数据链路层设计要点

数据链路层要完成许多特定的功能。这些功能包括:

1. 向网络层提供一个定义良好的服务接口。
2. 处理传输错误。
3. 调节数据流,确保慢速的接收方不会被快速的发送方淹没。

为了实现这些目标,数据链路层从网络层获取到分组,然后将这些分组封装到帧(frame)中以便传输。每一帧包含一个帧头、一个有效载荷域(用于存放分组),以及一个帧尾,如图3.1所示。帧管理构成了数据链路层工作的核心。在本节中,我们将详细地讨论刚才提到的这些问题。

虽然本章只是讨论数据链路层和数据链路协议,但是,我们在本章中将要学习的许多原理,比如错误控制和流控制,同样也适用于传输层和其他的协议。实际上,在许多网络中,这些功能只出现在数据链路层以上的各层中,而没有出现在数据链路层。然而,不管它们出现在哪里,原理是非常一致的,所以,无论我们在哪里学习这些原理都没有关系。在数据链路层中,它们通常表现出最为简单和单纯的形式,这使得数据链路层是细致地学习这些原理的好地方。

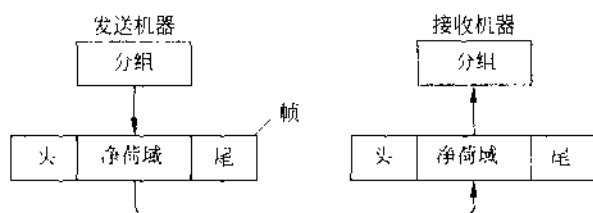


图 3.1 分组和帧之间的关系

### 3.1.1 为网络层提供的服务

数据链路层的功能是为网络层提供服务。最主要的服务是将数据从源机器的网络层传输到目标机器的网络层。在源机器的网络层中有一个实体,称为进程,它将一些数据位交给数据链路层,要求传输到目标机器。数据链路层的任务是将这些位传输给目标机器,然后再将这些数据进一步交给目标机器的网络层,如图 3.2(a)所示。实际的传输过程沿着图 3.2(b)所示的路径,但是我们很容易将这个过程想象成两个数据链路层进程使用一个数据链路协议在进行通信。出于这样的原因,在本章中我们将使用图 3.2(a)的模型。

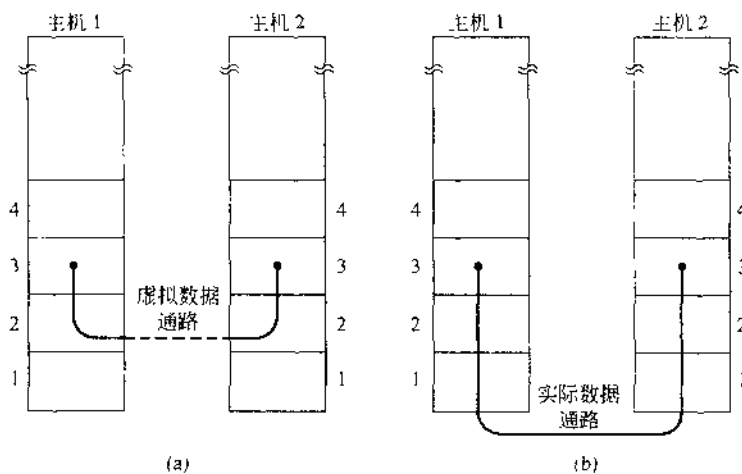


图 3.2

(a) 虚拟通信过程; (b) 实际通信过程

数据链路层的设计目标是提供各种服务。实际提供的服务随系统的不同而不同,但是一般情况下,通常会提供以下三种可能的服务:

- (1) 无确认的无连接服务。
- (2) 有确认的无连接服务。
- (3) 有确认的面向连接服务。

下面我们来依次讨论这些服务。

无确认的无连接服务是指源机器向目标机器发送独立的帧,目标机器并不对这些帧进行确认。事先并不建立逻辑连接,事后也不用释放逻辑连接。若由于线路上有噪声而造成了某一帧丢失,则数据链路层并不会检测这样的丢帧现象,也不会恢复。当错误率很低的时候,这一类服务是非常合适的,这时恢复过程可以留给上面的各层来完成。这类服务对于实时通信也是非常合适的,比如语音传输,因为在这种实时通信中数据迟到比数据损坏更加不好。绝大多数 LAN 在数据链路层上都是使用无确认的无连接服务。

为了提供可靠性,引入了有确认的无连接服务。当提供这种服务时,仍然没有使用逻辑连接,但是,所发送的每一帧都需要单独确认。这样,发送方知道每一帧是否已经正确地到达。如果有一帧在指定的时间间隔内还没有到达,则发送方将再次发送该帧。这类服务尤其适用于不可靠的信道,比如无线系统。

或许值得重点强调的是,在数据链路层上提供确认只是一种优化,永远不应该是一种要求。网络层总是可以发送一个分组,然后等待该分组被确认。如果在定时器超时之前,确认还没有到来,则发送方只要再次发送整个报文即可。这种策略的麻烦在于,帧总是有一个严格的长度限制,这是由硬件决定的,但网络层的分组没有这样的限制。如果一个普通的分组被分装到(比如说)10 帧中,并且 20% 的帧会被丢失,那么为了发送这个分组,可能需要花很长的时间。如果每个帧单独确认和重传,则整个分组很快就会发送过去。在可靠信道上,比如光纤,重量级数据链路协议的额外开销可能是不必要的,但是在无线信道上,由于它们内在的不可靠性,这种开销是非常值得的。

我们再回到有关服务的话题上,数据链路层能够向网络层提供的最复杂的服务是面向连接的服务。利用这种服务,源机器和目标机器在传输数据之前首先建立一个连接。该连接上发送的每一帧都被编号,数据链路层保证每一帧都会真正被接收到。而且,它保证每一帧只被接收一次,并且所有的帧都按照正确的顺序被接收。相反,在无连接服务中,你可以想象得到,如果确认报文丢失了,则一个分组可能会发送多次,因而也会接收多次。与此相反,面向连接的服务相当于为网络层进程提供了一个可靠的位流。

当使用面向连接的服务的时候,数据传输要经过三个不同的阶段。在第一个阶段,建立连接,双方初始化各种变量和计数器,这些变量和计数器记录了哪些帧已经接收到,哪些还没有。在第二个阶段,一个或者多个数据帧被真正传输出去。在第三个也是最后一个阶段中,连接被释放,所有的变量、缓冲区,以及其他用于维护该连接的资源也随之被释放。

考虑一个典型的例子:一个 WAN 子网包含了许多路由器,它们通过租用的点到点电话线连接起来。当某一帧到达一个路由器的时候,硬件首先检查它是否有错误(利用本章后面我们将要学习的技术),然后将该帧传递给数据链路层软件(它有可能被内嵌在网络接口板的一个芯片中)。数据链路层软件检查这一帧,看是否是自己所期望的帧,如果是,则把包含在有效载荷域中的分组交给路由软件。接着,路由软件选择正确的输出线路,并且把分组向下传递给数据链路层软件,然后数据链路层软件将它发送出去。经过两个路由器的数据流情况如图 3.3 所示。

路由代码总是希望所有的工作都能正确地完成,也就是说,在每一条点到点线路上建

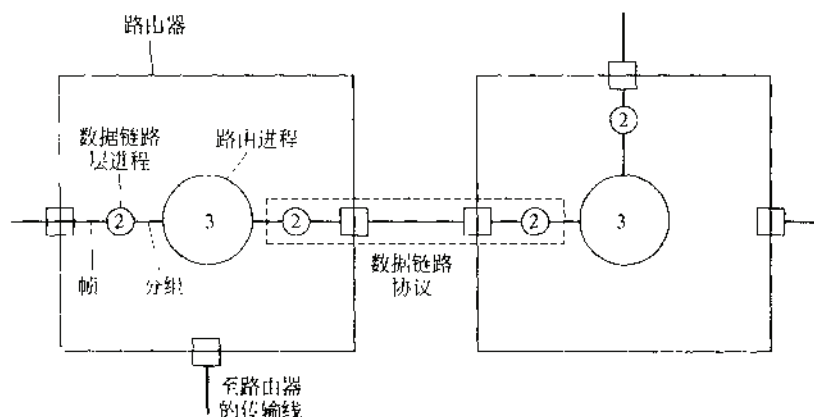


图 3.3 数据链路协议的位置

立起可靠的、有序的连接。它并不希望老是陷入到分组丢失的境地。如图中的虚线框所示,数据链路协议使得不可靠的通信线路看起来更加完美,至少比原来更好。另一方面,尽管我们在每一个路由器中显示了多份数据链路层软件的副本,但实际上,只有一份数据链路层软件,它负责处理所有的线路,每条线路有不同的表和数据结构。

### 3.1.2 成帧

为了向网络层提供服务,数据链路层必须使用物理层提供给它的服务。物理层的任务是接受一个原始的位流,并试图将它递交给目标机器。这个位流并不保证没有错误。接收到的位的数量可能少于、等于或者多于发送的位的数量,而且它们可能有不同的值。检测错误和(有必要的话)纠正错误的工作由数据链路层来完成。

对于数据链路层,一般的做法是将位流分解成离散的帧,并计算每一帧的校验和(本章后面将讨论校验和算法)。当一帧到达目标机器的时候,重新计算校验和。如果新算出来的校验和与该帧中包含的校验和不同,则数据链路层知道传输过程中产生了错误,它就会采取措施来处理错误(比如丢掉坏帧,可能还会送回一个错误报告)。

将原始的位流分解到离散的帧中,这项工作看起来容易,实际做起来要困难得多。一种成帧的办法是在帧之间插入时间间隙(time gap),就好像在普通正文的英文单词之间插入空格一样。然而,网络一般不会对时间的正确性做任何保证,所以,在传输过程中,这些间隙有可能被挤掉,或者其他的间隙被插入进来。

由于依靠时间来标识每一帧的起始和结束位置风险太大,所以有必要设计其他的成帧方法。在这一小节中,我们将讨论4种方法:

- (1) 字符计数法;
- (2) 含字节填充的分界符法;
- (3) 含位填充的分界标志法;
- (4) 物理层编码违例法。

第一种成帧方法利用头部中的一个域来指定该帧中的字符数。当目标端的数据链路



层看到这个字符计数值的时候,它知道后面跟着多少字符,因此也就知道了该帧的结束处在哪里。这项技术如图 3.4(a)所示,其中四帧的大小分别为 5、5、8 和 8 个字符。

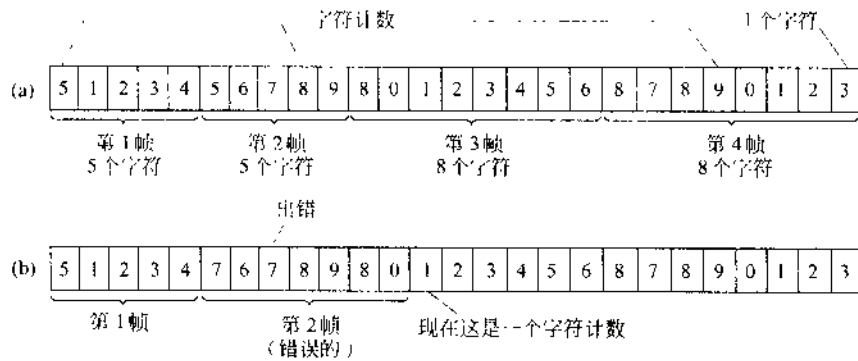


图 3.4 一个字符流  
(a) 无差错; (b) 有一个差错

这个算法的问题在于,计数值有可能因为传输错误而被弄乱。例如,如果第 2 帧中的计数值 5 变成了 7,如图 3.4(b)所示,则目标方就会失去同步,从而不可能再找到下一帧的起始位置。由于校验和是不正确的,所以目标方虽然知道该帧已经被损坏,它仍然无法知道下一帧从哪里开始。在这种情况下,给源方送回一个“请求重传”帧也无济于事,因为目标方并不知道应该跳过多少个字符才能到达重传的开始处。由于这个原因,字符计数方法已经很少使用了。

第二种成帧方法考虑到了错误之后重新同步的问题,它的做法是让每一帧都用一些特殊的字节作为开始和结束。在过去,起始和结束字节是不同的,但是,最近几年中,绝大多数协议倾向于使用相同的字节,称为标志字节(flag byte),作为起始和结束分界符,如图 3.5(a)中的 FLAG 所示。按照这种做法,如果接收方丢失了同步,它只需搜索标志字节就能找到当前帧的结束位置。两个连续的标志字节代表了当前帧的结束和下一帧的开始。

当二进制数据(比如目标程序或者浮点数值)被传输的时候,这种方法有一个严重的问题。当标志字节的位模式出现在数据中时,这个问题就很容易发生。这种位模式往往会干扰帧的分界。解决这个问题的一种方法是,发送方的数据链路层在这种“偶尔”出现的每个标志字节的前面插入一个特殊的转义字节(ESC)。接收端的数据链路层在将数据送给网络层之前删除掉转义字节。这种技术被称为字节填充(byte stuffing)或者字符填充(character stuffing)。因此,成帧用的标志字节与数据中出现的标志字节可以区分开,只要看它前面有没有转义字节即可。

当然,接下来的问题是,如果转义字节也出现在数据中间,那么该怎么办呢?答案是,同样用一个转义字节来填充。因此,任何单个转义字节一定是转义序列的一部分,而两个转义字节则代表了数据中自然出现的一个转义字节。图 3.5(b)显示了一些例子。在所有这些例子中,去掉填充之后被递交给网络层的字节序列与原始的字节序列完全一致。

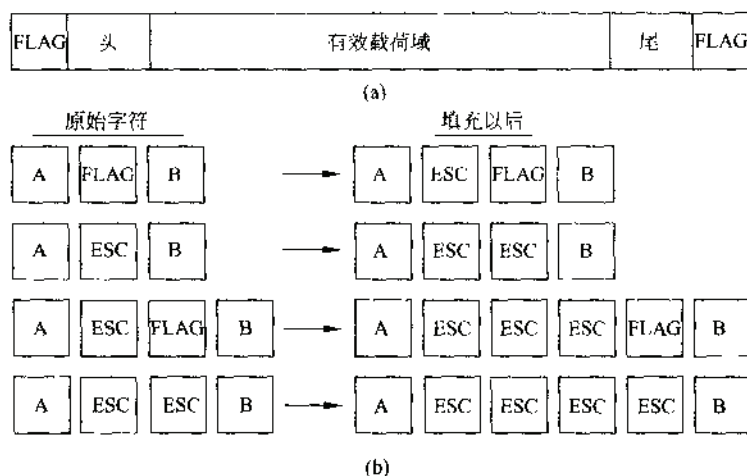


图 3.5

(a) 有标志字节作为分界的帧；(b) 字节填充前后的 4 个字节序列例子

图 3.5 中描述的字节-填充方案是 PPP 协议中使用的填充方案的一个略微简化的形式,而 PPP 协议则是大多数家庭计算机与 Internet 服务供应商进行通信所使用的协议。本章后面我们将讨论 PPP 协议。

使用这种成帧方法的一个主要缺点是,它紧紧地依赖于 8 位字符的模式。事实上,并不是所有的字符码都使用 8 位字符。例如,UNICODE 使用 16 位字符。随着网络的发展,在成帧机制中内含字符码长度的缺点变得越来越明显,所以,有必要开发一种新的技术以便允许任意长度的字符。

新的技术允许数据帧包含任意长度的位,也允许每个字符有任意长度的位。它的工作方式如下所述。每一帧的开始和结束都有一个特殊的位模式: 01111110(实际上就是一个标志字节)。当发送方的数据链路层碰到数据中 5 个连续的位“1”的时候,它自动在输出位流中填充一个位“0”。这种位填充(bit stuffing)机制与字节填充机制非常相似,在字节填充机制中,当发送方看到数据中的标志字节的时候,它就在其前面填充一个转义字节,然后再送到输出字符流中。

当接收方看到 5 个连续的输入位“1”,并且后面是位“0”时,它自动去掉(即删除)该“0”位。就好像字节填充过程对于两方计算机中的网络层完全透明一样,位填充过程也对网络层完全透明。如果用户数据包含了标志模式 01111110,则该标志当作 011111010 来传输,但是存储在接收方内存中的是 01111110。图 3.6 给出了位填充的一个例子。

在位填充机制中,通过标志模式可以明确地识别出两帧之间的边界。因此,如果接收方失去了帧同步,它只需在输入流中扫描标志序列即可,因为标志序列只可能出现在帧边界上,永远不可能出现在数据中。

最后一种成帧方法只适用于那些“物理介质上的编码方法中包含冗余信息”的网络。例如,有些 LAN 用 2 个物理位来编码 1 位数据。通常,“1”位是“高-低”电平对,而“0”位是“低-高”电平对。这种方案意味着每一个数据位都有一个中间电平跃变,这使得接收方

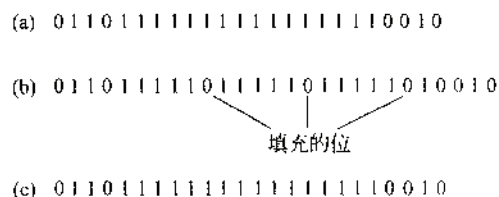


图 3.6 位填充  
(a) 原始的数据; (b) 线路上的数据; (c) 删除填充之后存储在接收方存储器中的数据

很容易定位到位的边界上。“高-高”和“低-低”这两种组合并不用于数据,但是在某些协议中用于帧的分界。

关于成帧机制,最后需要说明的是,许多数据链路协议联合使用字符计数法和其他某一种方法,以保证额外的安全性。当一帧到达时,首先利用计数域定位到该帧的结束处。只有当这个位置上确实出现了正确的分界符,并且帧的校验和也是正确的时候,该帧才被认为是有效的。否则的话,接收方在输入流中扫描下一个分界符。

### 3.1.3 错误控制

解决了“标识每一帧的起始和结束位置”的问题之后,我们现在来看下一个问题:如何确保所有的帧最终都被递交给目标机器上的网络层,并且保持正确的顺序。假定发送方只是不断地往外发送数据帧,而没有考虑到它们是否正确到达。对于无确认的无连接服务,这样可能已经足够了,但是对于可靠的、面向连接的服务,这样做肯定还远远不够。

确保可靠递交的常用方法是向发送方提供一些有关线路另一端状况的反馈信息。通常情况下,协议要求接收方送回一些特殊的控制帧,在这些控制帧中,对于它所接收到的帧进行肯定的或者否定的确认。如果发送方收到了关于某一帧的肯定确认,那么它就知道这一帧已经安全地到达了。另一方面,否定的确认意味着传输过程中产生了错误,所以这一帧必须重传。

更为复杂的是,有时候由于硬件的问题,有的帧完全丢失了(比如噪声突发时)。在这种情况下,接收方根本不会有任何反应,因为它没有理由做出反应。显然,如果在一个协议中,发送方送出了一帧之后就等待肯定的或者否定的确认,那么,若由于硬件故障的原因而丢失了某一帧的话,则发送方就永远等待下去了。

这种可能出现的问题可以通过在数据链路层中引入定时器来解决。当发送方送出一帧时,通常还要启动一个定时器。该定时器的过期时间应该设置得足够长,以保证该帧在正常情况下能够到达目标方,并且在目标方进行处理,然后再将确认送回到发送方。一般情况下,在定时器到期之前,该帧将正确地接收到,并且确认报文也会送回来,这时定时器被取消。

然而,如果原始的帧或者确认报文丢失了的话,则定时器将被触发,从而警告发送方有一个潜在的问题存在。一种简单的解决方案是重新发送这一帧。然而,当有的帧被发送了多次之后,可能会出现这样的危险:接收方将两次或者多次接收同一帧,并且多次将它传递给网络层。为了避免发生这样的情形,一般有必要给送出去的帧分配序列号,这样

接收方能够区别原始帧和重传帧。

不管怎么样,管理好定时器和序列号,以便保证每一帧最终都恰好一次地被传递给目标机器的网络层,这是数据链路层工作的重要组成部分。在本章后面,我们将通过一系列逐渐复杂的例子,来看看如何做好定时器和序列号的管理工作。

### 3.1.4 流控制

在数据链路层(以及更高的各层)中另一个重要的设计问题是,如果发送方发送帧的速度超过了接收方能够接受这些帧的速度,则发送方该如何处理呢?当发送方运行在一台快速(或者负载较轻)的计算机上,而接收方运行在一台慢速(或者负载较重)的计算机上的时候,这种情况很容易发生。发送方持续地以很高的速度往外发送帧,直到接收方完全被淹没。即使传输过程不会出错,但到了某一个点上的时候,接收方也将无法再处理持续到来的帧,从这时开始就要丢弃一些帧了。很显然,必须要采取某种措施来阻止这种情况的发生。

常用的办法有两种。第一种是基于反馈的流控制(feedback-based flow control),接收方给发送方送回信息,允许它发送更多的数据,或者至少也要告诉发送方它的情况怎么样。第二种方法是基于速率的流控制(rate-based flow control)。使用这种方法的协议有一种内置的机制,它限制了发送方传输数据的速率,而无需利用接收方的反馈信息。在本章中,我们将学习基于反馈的流控制方案,因为数据链路层从来不使用基于速率的流控制方案。我们将在第5章中讨论基于速率的方案。

基于反馈的流控制方案有许多种,但是绝大多数使用了同样的基本原理。协议包含了许多定义良好的规则,这些规则规定了发送方什么时候可以发送下一帧,通常在没有得到接收方许可(隐式或者显式许可)之前,禁止发送方往外发帧。例如,当建立一个连接的时候,接收方可能会这样说:“你现在可以发送n帧,但是在发送完n帧之后就别再发送了,等到我下次告诉你发送的时候再继续发送。”稍后我们将讨论这些细节。

## 3.2 错误检测和纠正

正如我们在第2章中所看到的那样,电话系统有三个部分:交换机、局间干线和本地回路。目前在大多数发达国家,前两部分几乎都已经完全数字化。本地回路仍然是模拟的双绞线,由于替换掉这些双绞线需要花费大量的资金,所以在接下去的几年中本地回路还是双绞线。虽然在数字部分很少发生传输错误,但是,在本地回路上错误还很常见。而且,无线通信正在普及,它的错误率比局间的光纤干线高出几个数量级。因此,结论是:在接下去的很多年中,传输错误将一直伴随着我们。我们必须要知道该如何处理传输错误。

由于物理过程而产生的错误,在有些介质(比如无线电波)上常常是突发性的,而不是单个的。突发性的错误与孤立的、单个位的错误相比,既有优点也有缺点。假设数据块的大小为1000位,每一位的错误率是0.001。如果错误是独立的,则大多数数据块将包含一个错误。然而,如果错误是突发性的,发生一次就是连续100位,则平均而言,在100个

数据块中只有 1 个或者 2 个数据块受到影响。突发性错误的缺点是,它比单独的错误更加难以纠正。

### 3.2.1 纠错码

网络设计者已经研究出两种基本的策略用于错误处理过程。一种方法是在每一个被发送的数据块中包含足够的冗余信息,以便接收方可以推断出被发送的数据中肯定有哪些内容。另一种方法也是包含一些冗余信息,但是这些信息只能让接收方推断出发生了错误,但推断不出发生了哪个错误,然后接收方可以请求重传。前一种策略使用了纠错码(error-correcting code),后一种策略使用了检错码(error-detecting code)。使用纠错码的技术通常也称为前向纠错(forward error correction)。

这里的每一项技术都有不同的适用环境。在高度可靠的信道上,比如光纤,比较合理的做法是使用检错码,当偶尔有错误发生时,只需重传整个数据块即可。然而,在错误发生很频繁的信道上,比如无线链路,最好的做法是在每一个数据块中加入足够的冗余信息,以便接收方能够计算出原始的数据块是什么,而不是依靠重传来解决问题,因为重传的数据块本身也可能是错误的。

为了理解错误发生之后是如何被处理的,有必要先来看一下错误到底是什么样的。通常,一帧包含  $m$  个数据位(即报文)和  $r$  个冗余位(校验位)。假设总长度为  $n$ ,即  $n=m+r$ 。包含数据和校验位的  $n$ -位单元通常也称为  $n$  位码字(codeword)。

给定两个码字,比如说 10001001 和 10110001,我们可以确定它们有多少个对应位不相同。在这个例子中,有 3 位不同。为了确定有多少位不同,只要对这两个码字做异或(XOR)运算,然后计算出异或结果中 1 的个数,例如:

```
10001001
10110001
-----
00111000
```

两个码字中不相同的位的个数称为海明距离(Hamming distance)(Hamming, 1950)。它的意义在于,如果两个码字的海明距离为  $d$ ,则需要  $d$  个 1 位错误才能将一个码字转变成另一个码字。

在大多数数据传输应用中,所有  $2^n$  种可能的数据报文都是合法的,但是,根据校验位的计算方法,并非所有  $2^n$  种可能的码字都被用到了。给定了计算校验位的算法以后,我们有可能构造出完整的合法码字列表,并且从这个列表中找到海明距离最小的两个码字。此距离是这整个编码方案的海明距离。

一种编码方案的检错和纠错特性跟它的海明距离有关。为了检测  $d$  个错误,需要一个距离为  $d+1$  的编码方案,因为在这样的编码方案中, $d$  个 1 位错误不可能将一个有效码字改变成另一个有效码字。当接收方看到一个无效码字的时候,它就知道已经发生了传输错误。类似地,为了纠正  $d$  个错误,需要一个距离为  $2d+1$  的编码方案,因为在这样的编码方案中,合法码字之间的距离足够远,因而即使发生了  $d$  位变化,则还是原来的码字离它最近,从而可以惟一确定原来的码字,达到纠错的目的。

下面给出一个简单的检错编码例子。请考虑这样一个编码:在数据后面加上一个奇



偶位 (parity bit)。奇偶位是这样选择的：保证码字中“1”位的数目是偶数 (或者奇数)。例如，当 1011010 以偶数位发送时，后面加上一位变成了 10110100。如果是按奇数位发送，则 1011010 变成 10110101。只加上单个奇偶位的编码方案的距离为 2，因为任何 1 位错误所产生的码字，其奇偶位一定是错误的。因此，它可以用来检测单个错误。

作为纠错编码的简单例子，请考虑下面只有 4 个有效码字的编码：

0000000000, 0000011111, 1111100000, 1111111111

以上编码的距离为 5，这意味着它可以纠正 2 个错误。如果码字 0000000111 到达的话，接收方知道原始的码字一定是 0000011111。然而，如果发生了三个错误，0000000000 变成了 0000000111，则以上编码就不能够正确地纠正错误了。

设想我们要设计一种编码方案，每个码字有  $m$  个报文位和  $r$  个校验位，并且能够纠正所有的单个错误。对于  $2^m$  个合法报文，任一个报文都对应有  $n$  个非法的码字，它们与该报文的距离为 1。这些非法的码字可以这样构成：将该报文对应的合法码字的  $n$  位，逐个取反，可以得到  $n$  个距离为 1 的非法码字。因此，每个合法的报文都要求  $n+1$  个位模式，专门供它使用。由于总共只有  $2^n$  个位模式，所以，我们必须有  $(n+1)2^m \leq 2^n$ 。利用  $n=m+r$ ，这个要求变成了  $(m+r+1) \leq 2^r$ 。在给定  $m$  的情况下，这个条件给出了用于纠正单个错误所需要的校验位数的下界。

实际上，利用海明 (1950 年) 提出的方法，这个理论下界是可以达到的。码字中的每一位连续编号，从最左边位 1 开始，它的右边是位 2，等等。编号为 2 的幂次方的位 (1, 2, 4, 8, 16, ...) 为校验位。剩下的位 (3, 5, 6, 7, 9, ...) 用  $m$  个数据位来填充。每一个校验位都迫使某一组位 (包括它自己) 的奇偶值为偶数 (或奇数)。一个位可能包含在几次奇偶值计算中。为了看清楚位置  $k$  上的数据位对哪些校验位有影响，我们将  $k$  重写成 2 的幂次方的和。例如， $11=1+2+8$ ， $29=1+4+8+16$ 。只有出现在  $k$  的展开式中的这些校验位才校验位置  $k$  上的数据位。

当一个码字到来的时候，接收方将一个计数器初始化为 0。然后，它检查每一个校验位  $k$  ( $k=1, 2, 4, 8, \dots$ )，看它是否有正确的奇偶性。如果没有的话，接收方将  $k$  加到计数器上。如果在所有的校验位都被检查过之后，计数器为 0，即这些校验位都是正确的，则该码字被作为有效码字而接受。如果计数器不为 0，则它包含了不正确位的编号。例如，如果校验位 1、2 和 8 是错误的，则变反的位 (即错误的位) 是 11，因为只有它才被 1、2 和 8 位校验。图 3.7 显示了一些 7 位 ASCII 字符，利用海明码将它们编成了 11 位的码字。注意，数据位出现在位置 3、5、6、7、9、10 和 11 上。

海明码只能纠正单个错误，然而，这里有一个技巧可以用来使海明码也能够纠正突发性的错误。 $k$  个连续的码字被排列成一个矩阵，每行一个码字。通常情况下，传输数据的时候每次一个码字，从左向右。为了纠正突发性的错误，传输数据时每次发送一列，从最左边的列开始。当第 1 列所有的  $k$  位都被发送出去以后，再发送第 2 列，以此类推，如图 3.7 所示。当这一帧到达接收方的时候，接收方重构同样的矩阵，每次 1 列。如果一个突发性错误的长度为  $k$  位，则在  $k$  个码字中，每个码字至多只有 1 位受到影响，但是利用海明码，每个码字可以纠正一个错误，所以整个数据块也可以恢复出来。这种方法利用  $kr$  个校验位，使  $km$  个数据位能够抵抗长度等于或小于  $k$  的单个突发性错误。

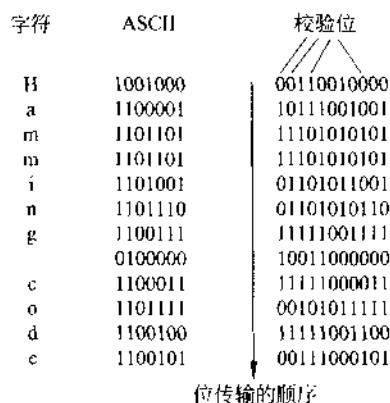


图 3.7 利用海明码来纠正突发性错误

### 3.2.2 检错码

纠错码广泛应用于无线链路,因为无线链路相比铜线或者光纤有更多的噪声,也更容易出错。如果不使用纠错码的话,则几乎任何数据都难以通过。然而,在铜线或者光纤上,错误率非常低,所以,对于这些链路上偶尔出现的错误,利用错误检测和重传机制往往更加有效。

作为一个简单的例子,考虑这样一个信道:错误是孤立的,错误率为每位  $10^{-6}$ 。对于 1000 位的数据块,为了提供纠错功能,需要 10 个校验位;1 兆位的数据将需要 10 000 校验位。如果仅仅为了检测数据块中的单个 1 位错误,则每个数据块 1 个奇偶位就足够了。每 1000 个数据块将需要额外传输一个块(1001 位)。利用“错误检测+重传”的方法,总的开销是每 1 兆位数据只需 2001 位。相比之下,如果利用海明码,则需要 10 000 位。

如果在一块数据中只增加了一个奇偶位,但是在传输这一块数据时发生了一个很长的突发性错误,那么该错误能够被检测到的概率只有 0.5,这是难以接受的。如果每一块数据被当作一个矩阵( $n$  位宽, $k$  位高,如上一小节所述)来发送,则检测到错误的几率会明显增加。为每一列单独计算一个奇偶位,并将它附在矩阵的下边作为最后一行,然后按每次一行发送该矩阵数据。当数据块到达接收方时,接收方检查所有的奇偶位。如果任何一位有错误,则接收方请求重传该数据块。根据需要可以再次请求重传,直到接收到的整个数据块没有任何奇偶错误为止。

这种方法可以检测出长度为  $n$  的单个突发性错误,因为每列只有 1 位被改变了。然而,对于长度为  $n+1$  的突发性错误,如果第 1 位变反,第  $n+1$  位变反,所有其他的  $n-1$  位都是正确的,则这种方法无法检测出这样的错误。(突发性错误并不意味着所有的位都是错误的,它只意味着至少第 1 位和最后一位是错误的。)如果数据块被一个长的突发性错误或者多个短一点的突发性错误影响了的话,则在  $n$  列中,任何一列有正确的奇偶位的概率是 0.5,所以,一个坏块被当前正确数据块接受的概率为  $2^{-n}$ 。

尽管上述方案有时候已经足够了,但是在实践中,广泛使用的是另外一种方法:多项式编码(polynomial code),也称为 CRC(Cyclic Redundancy Check,循环冗余校验码)。多

项式编码的基本思想是：将位串看成是系数为 0 或 1 的多项式。一个  $k$  位的帧看作是一个  $k-1$  次多项式的系数列表，该多项式共有  $k$  项，从  $x^{k-1}$  到  $x^0$ 。这样的多项式认为是  $k-1$  阶多项式。高次（最左边）位是  $x^{k-1}$  项的系数；接下去的位是  $x^{k-2}$  项的系数；依此类推。例如，110001 有 6 位，因此代表了一个共有 6 项的多项式，其系数为 1、1、0、0、0 和 1，即  $x^5 + x^4 + x^0$ 。

多项式的算术运算采用代数域理论的规则，以 2 为模来完成。加法没有进位，减法没有借位。加法和减法都等同于异或。例如：

$$\begin{array}{r} 10011011 \\ + 11001010 \\ \hline 01010001 \end{array} \quad \begin{array}{r} 00110011 \\ + 11001101 \\ \hline 11111110 \end{array} \quad \begin{array}{r} 11110000 \\ - 10100110 \\ \hline 01010110 \end{array} \quad \begin{array}{r} 01010101 \\ - 10101111 \\ \hline 11111010 \end{array}$$

长除法与二进制中的长除运算一样，只不过减法按模 2 进行，具体见上。如果被除数与除数有一样多的位，则称该除数“进入到”被除数中。

当使用多项式编码时，发送方和接收方必须预先商定一个生成多项式(generator polynomial)  $G(x)$ 。生成多项式的最高位和最低位必须是 1。假设一帧有  $m$  位，它对应于多项式  $M(x)$ ，为了计算它的校验和(checksum)，该帧必须比生成器多项式长。基本的思想是在帧的尾部追加一个校验和，使得追加之后的帧所对应的多项式能够被  $G(x)$  除尽。当接收方收到了带校验和的帧之后，它试着用  $G(x)$  去除它。如果有余数的话，则表明传输过程中有错误。

计算校验和的算法如下：

(1) 假设  $G(x)$  的阶为  $r$ 。在帧的低位端加上  $r$  个 0 位，所以该帧现在包含  $m+r$  位，对应多项式为  $x^r M(x)$ 。

(2) 利用模 2 除法，用对应于  $G(x)$  的位串去除对应于  $x^r M(x)$  的位串。

(3) 利用模 2 减法，从对应于  $x^r M(x)$  的位串中减去余数（总是小于等于  $r$  位）。结果就是将被传输的带校验和的帧。它的多项式不妨设为  $T(x)$ 。

图 3.8 显示了当帧为 1101011011，生成器多项式为  $x^4 + x + 1$  时计算校验和的情形。

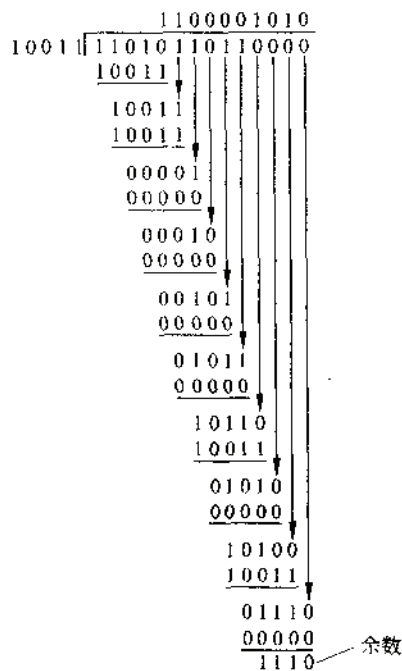
显然， $T(x)$  可以被  $G(x)$  除尽（模 2）。在任何一种除法中，如果你将被除数减掉余数，则剩下的差值一定可以被除数除尽。例如，在十进制中，如果你用 210 278 除以 10 941，则余数为 2399。从 210 278 中减去 2399，得到 207 879，它可以被 10 941 除尽。

现在我们来分析一下这种方法的功能。什么样的错误可以被检测到呢？想象一下在传输过程中发生了一个错误，所以接收方收到的不是  $T(x)$ ，而是  $T(x) + E(x)$ ， $E(x)$  中的每一个“1”位都对应于有一位变反了。如果  $E(x)$  中有  $k$  个“1”位，则表明有  $k$  个 1 位错误发生了。一个突发性错误可以这样来描述：首先是 1，然后是 0 和 1 的混合，最后也是 1，所有其他的位都是 0。

接收方在收到了带校验和的帧之后，用  $G(x)$  来除它，也就是说，接收方计算  $[T(x) + E(x)]/G(x)$ 。 $T(x)/G(x)$  是 0，所以计算的结果是  $E(x)/G(x)$ 。如果错误多项式（即  $E(x)$ ）恰好包含  $G(x)$  作为它的一个因子，则这样的错误将无法检测到；否则其他的错误都能够检测得到。

如果只有一位发生错误，即  $E(x) = x^i$ ，这里  $i$  决定了错误发生在哪一位上。如果

附加4个零后形成的串:11010110110000



传输的帧: 11010110111110

图 3.8 多项式编码校验和的计算过程

如果有两个独立的一位错误,则  $E(x) = x^i + x^j$ , 这里  $i > j$ 。换一种写法,  $E(x)$  可以写成  $E(x) = x^j(x^{i-j} + 1)$ 。如果我们假定  $G(x)$  不能被  $x$  除尽, 则所有的双位错误都能够被检测到的充分条件是, 对于任何小于等于  $i-j$  最大值 (即小于等于最大的帧长度) 的  $k$  值,  $G(x)$  都不能除尽  $x^k + 1$ 。简单的结论是, 低阶的多项式可以保护长的帧。例如, 对于任何  $k < 32,768$ ,  $x^{13} + x^{14} + 1$  都不能除尽  $x^k + 1$ 。

如果有奇数个位发生了错误,则  $E(x)$  包含奇数项(比如  $x^5 + x^2 + 1$ , 但不是  $x^2 + 1$ )。有意思的是,在模 2 的系统中,没有一个奇数项多项式包含  $x+1$  作为因子。因此,以  $x+1$  作为  $G(x)$  的一个因子,我们就可以捕捉到所有包含奇数个位变反的错误情形。

为了解奇数项多项式不可能被  $x+1$  除尽,我们用反证法,假设  $E(x)$  有奇数项,并且可以被  $x+1$  除尽。于是,我们将  $E(x)$  分解成  $(x+1)Q(x)$ 。现在令  $x=-1$ ,则得到  $E(1)=(1+1)Q(1)$ 。由于  $1+1=0 \pmod{2}$ ,所以  $E(1)=0$ 。如果  $E(x)$  有奇数项的话,则用 1 代替所有的  $x$ ,结果总是 1。因此,奇数项多项式不可能被  $x+1$  除尽。

最后,也是最重要的,带  $r$  个校验位的多项式编码可以检测到所有长度小于等于  $r$  的

突发性错误。长度为  $k$  的突发性错误可以用  $x^i(x^{k-1} + \dots + 1)$  来表示,这里  $i$  决定了突发性错误的位置离帧的最右端的距离有多远。如果  $G(x)$  包含一个  $x^i$  项,则它不可能有  $x^i$  作为因子,所以,如果括号内表达式的阶小于  $G(x)$  的阶,则余数永远不可能为 0。

如果突发性错误的长度为  $r+1$ ,则当且仅当错误多项式等于  $G(x)$  的时候,错误多项式除以  $G(x)$  的余数才为 0。根据突发性错误的定义,第一位和最后一位必须为 1,所以它是否与  $G(x)$  匹配要取决于其他  $r-1$  个中间位。如果所有的组合被认为是等概率的,则这样一个不正确的帧被当作有效帧而接收的概率是  $\frac{1}{2^{r+1}}$ 。

同样也可以证明,当一个长度大于  $r+1$  位的突发性错误发生的时候,或者几个短一点突发性错误发生的时候,一个坏帧被当作有效帧而通过检测的概率为  $\frac{1}{2^r}$ ,这里假设所有的位模式都是等概率的。

有一些特殊的多项式已经成为国际标准了,其中在 IEEE 802 中使用的多项式为:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x + 1 \quad | \quad x^7 + x^2 + x + 1$$

以上的多项式有一些很好的特性,其中一个:它能够检测所有长度小于等于 32 的突发性错误,以及所有只影响奇数个位的突发性错误。

虽然计算校验和的算法看起来好像非常复杂,但是 Peterson 和 Brown(1961 年)已经证明了,可以用硬件构造一个简单的移位寄存器电路来计算和验证校验和。在实践中,几乎总是采用这种硬件。几乎所有的 LAN 都使用它,点到点的线路在有些情况下也使用它。

几十年来,人们总是假设:要被校验的帧包含了随机的位。所有对校验和算法的分析也是在这样的假设下进行的。通过对实际数据的观察和分析,已经表明这种假设是非常错误的。其结果是,在某些情况下,未检测出来的错误比原先想象的要多得多 (Partridge et al., 1995)。

### 3.3 基本数据链路协议

为了介绍数据链路协议,我们将首先讨论三个逐渐复杂的协议作为开始。对于有兴趣的读者,通过 Web 网站(参见前言)可以获得这些协议以及后面一些协议的模拟器。在介绍这些协议之前,我们先明确一些有关通信模型的基本假设,这是非常有必要的。首先,我们假设物理层、数据链路层和网络层都是独立的进程,它们通过来回传递报文进行通信。在许多情况下,物理层和数据链路层进程会在一个特殊的网络 I/O 电路中的一个处理器上运行;而网络层代码则在主 CPU 上运行。然而,其他的实现方案也是有可能的(比如,三个进程都在同一个 I/O 电路中运行;或者物理层和数据链路层作为过程,被网络层调用)。无论如何,将这三层作为独立的进程来讨论有助于使概念更加清晰,同时也可以强调每一层的独立性。

另一个关键的假设是,机器 A 希望用一个可靠的、面向连接的服务,向机器 B 发送一个长的数据流。以后我们再考虑 B 也希望向 A 并发地发送数据。假定 A 要发送的数据总是已经准备好了,不必等待这些数据被生成出来。或者说,当 A 的数据链路层请求数



据时,网络层总是能够立即满足数据链路层的要求(这个限制后面也将被去掉)。

我们还假设机器不会崩溃,也就是说,这些协议只处理通信错误,但不处理因为机器崩溃和重新启动而引起的问题。

在涉及到数据链路层的时候,通过接口从网络层传递到数据链路层的分组是纯粹的数据,它的每一位都将被递交到目标机器的网络层。目标机器的网络层可能会将分组的一部分翻译为一个头,这种行为不在数据链路层的考虑范围之内。

当数据链路层接收一个分组时,它就在分组上增加一个数据链路层头和尾,将它封装到一个帧中(见图 3.1)。因此,每个帧包含一个内嵌的分组、一些控制信息(在头中)和一个校验和(在尾部)。然后,帧被传输到另一台机器上的数据链路层。我们假设有一个现成的代码库,其中库过程 `to_physical_layer` 用于发送一帧, `from_physical_layer` 用于接收一帧。负责传输的硬件会计算校验和,并追加在尾部(因而创建了帧尾),所以数据链路层软件不需要担心校验和。例如,本章上节讨论的多项式算法可能会用到。

刚开始时,接收方什么也不做。它只是静静地待着,等待有事情发生。在本章的例子协议中,我们将这样来处理:数据链路层通过过程调用 `wait for event(&event)`,来等待有事情发生。只有当真有事情发生(比如,有一帧到达)时,该过程才返回。过程返回之后,变量 `event` 说明了所发生的事情。对于不同的协议,可能的事件集合也是不同的,每个协议都需要单独定义和描述事件集合。请注意,在一个更加实际的环境中,数据链路层不会像我们所建议的那样,在一个严格的循环中等待事件,而是会接收一个中断;中断将使系统终止当前的工作,转而处理进来的帧。然而,为了简便起见,我们将忽略数据链路层内所有并行活动的细节,假定它在全时地处理我们的一个信道。

当一帧到达接收方时,硬件会计算校验和。如果校验和不正确的话(即有传输错误),则数据链路层会收到通知(`event=cksum_err`)。如果进来的帧没有任何损坏,则数据链路层也会接到通知(`event=frame_arrival`),所以它就可以利用 `from_physical_layer` 获得进来的帧,并进行处理。只要接收方的数据链路层获得了一个完好无损的帧,它就检查头部的控制信息,如果一切都没有问题的话,则将分组部分传递给网络层。帧头部分永远也不会交给网络层。

为什么永远也不会将帧头交给网络层,这里有一个很好的理由:保持网络层和数据链路层完全分离。只要网络层对数据链路协议和帧格式一无所知,那么,当数据链路协议和帧格式有变化时,网络层的软件可以不作任何改变。在网络层和数据链路层之间提供一个严格的接口可以大大地简化软件的设计,因为不同层上的通信协议可以独立地发展。

图 3.9 给出了后面要讨论的许多协议公用的一些声明(C 语言)。其中定义了 5 个数据结构: `boolean`、`seq_nr`、`packet`、`frame_kind` 和 `frame`。 `boolean` 是一个枚举类型,可以取值 `true` 和 `false`。 `seq_nr` 是一个小整数,用来对帧进行编号,以便我们可以区分不同的帧。这些序列号从 0 开始,一直到(含) `MAX_SEQ`,所以,每个需要用到该数的协议都要定义它。 `packet` 是同一台机器上网络层和数据链路层之间,或者网络层对等体之间的信息交换单元。在我们的模型中,它总是包含 `MAX_PKT` 个字节,但是在实际的环境中,它应该是变长的。

```

#define MAX_PKT 1024                /* determines packet size in bytes */
typedef enum {false, true} boolean; /* boolean type */
typedef unsigned int seq_nr;         /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet; /* packet definition */
typedef enum {data, ack, nak} frame_kind; /* frame_kind definition */

typedef struct {                    /* frames are transported in this layer */
    frame_kind kind;                /* what kind of a frame is it? */
    seq_nr seq;                     /* sequence number */
    seq_nr ack;                     /* acknowledgement number */
    packet info;                     /* the network layer packet */
} frame;

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type * event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet * p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet * p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame * r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame * s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0

```

图 3.9 在下面的协议中需要用到的一些定义。这些定义包含在文件 protocol.h 中

frame 由 4 个域组成: kind、seq、ack 和 info,其中前三个包含了控制信息,最后一个可能包含了将要被传输的实际数据。这些控制域合起来称为帧头(frame header)。

kind 域指示了该帧中是否有数据,因为有些协议需要区分“只包含控制信息的帧”和

“也包含数据的帧”。seq 和 ack 分别用于序列号和确认;后面还会详细地描述它们的用法。数据帧的 info 域包含了一个分组;控制帧的 info 域没有用处。一个更加实际的协议实现将会使用一个变长的 info 域,而对于控制帧则完全忽略它。

再提一次,弄清楚分组和帧之间的关系是非常重要的。网络层从传输层获得一个报文,然后在报文上增加一个网络层头,于是创建了一个分组。该分组被传递给数据链路层,然后被放到输出帧的 info 域中。当该帧到达目标机器的时候,数据链路层从帧中提取出分组,然后将分组传递给网络层。按照这种方式,网络层就好像机器一样,可以直接交换分组。

图 3.9 中也列出了许多过程。这些库例程的细节与具体的实现有关,它们的内部工作机理不是我们这里要关心的。前面已经提到过,wait\_for\_event 是一个严格的循环,它等待有事情发生。过程 to\_network\_layer 和 from\_network\_layer 是数据链路层用于向网络层传递分组,或者从网络层接受分组的。注意,from\_physical\_layer 和 to\_physical\_layer 在数据链路层和物理层之间传递帧。另一方面,过程 to\_network\_layer 和 from\_network\_layer 在数据链路层和网络层之间传递分组。换句话说,to\_network\_layer 和 from\_network\_layer 处理 2、3 层之间的接口;而 from\_physical\_layer 和 to\_physical\_layer 处理 1、2 层之间的接口。

在大多数协议中,我们假设信道是不可靠的,并且偶尔会丢失整个帧。为了能够从这种灾难中恢复过来,发送方的数据链路层在发送每一帧的时候,必须启动一个内部定时器或者一个时钟。如果在预设的时间间隔内还没有收到应答的话,则时钟超时,数据链路层会收到一个中断信号。

在我们的协议中,这个过程是这样实现的:让过程 wait\_for\_event 返回 event = timeout。过程 start\_timer 和 stop\_timer 分别打开和关闭定时器。只有当定时器在运行的时候,超时才有可能发生。在定时器运行的同时,允许显式地调用 start\_timer;这样的调用只是重置时钟,等到再经过一个完整的定时器间隔之后才引发下一次超时(除非它再次被重置,或者被关闭)。

过程 start\_ack\_timer 和 stop\_ack\_timer 控制一个辅助定时器,该定时器被用于在特定条件下产生确认。

过程 enable\_network\_layer 和 disable\_network\_layer 用在较为复杂的协议中;在这样的协议中,我们不再假设网络层总是有数据要发送。当数据链路层启动了(enable)网络层的时候,它就允许网络层在有分组要发送的时候中断自己。我们用 event = network\_layer\_ready 来表示这种情况。当网络层被禁用(disable)的时候,它不会引发这样的事件。对于何时允许或者禁止数据链路层的网络层一定要非常谨慎,从而可以避免网络层用大量的分组淹没它(用完所有的缓存空间)。

帧序列号总是在从 0 到 MAX\_SEQ(含)的范围内,这里,在不同的协议中 MAX\_SEQ 也是不同的。通常有必要对序列号按循环加 1(即 MAX\_SEQ 之后是 0)。宏 inc 可以执行这项增一的任务。它之所以定义成宏,是因为在关键路径中它需要内联使用。正如后面我们将会看到,限制网络性能的因素通常在于协议处理过程,所以,把这种简单的操作定义成宏并不会影响代码的可读性,但是确实能够提高性能。而且,由于

MAX\_SEQ 在不同的协议中有不同的值,所以,将它定义成宏,就有可能在同一个二进制模块中包含所有的协议而不会引起冲突。这种能力对于模拟器是非常有用的。

图 3.9 中的声明是下面每个协议的一部分。为了节省空间和便于参考,它们已经被提取出来并且列在一起,但是,从概念上讲,它们应该与协议本身合并在一起。在 C 语言中,合并的方法是,把这些定义放在一个特殊的头文件中,这里是 protocol.h 文件,然后在协议文件中使用 C 预处理器符号 #include 将这些定义包含进来。

### 3.3.1 一个无限制的单工协议

作为第一个例子,我们来考虑一个尽可能简单的协议。在这个协议中,数据只能单向传输。传输方和接收方的网络层总是处于准备就绪的状态。处理的时间可以被忽略。假设缓存空间无穷大。最强的一个条件是,数据链路层之间的通信信道永远不会损坏或者丢失帧。这个完全不现实的协议如图 3.10 所示,我们给这个协议起一个绰号“乌托邦”。

```
* Protocol 1 (utopia) provides for data transmission in one direction only, from
sender to receiver. The communication channel is assumed to be error free
and the receiver is assumed to be able to process all the input infinitely quickly.
Consequently, the sender just sits in a loop pumping data out onto the line as
fast as it can. */
```

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
    frame s;          /* buffer for an outbound frame */
    packet buffer;     /* buffer for an outbound packet */
    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;             /* copy it into s for transmission */
        to_physical_layer(&s);       /* send it on its way */
    } /* Tomorrow, and tomorrow, and tomorrow,
        Creeps in this petty pace from day to day
        To the last syllable of recorded time,
        ~ Macbeth, V, v */
}

void receiver1(void)
{
    frame r;
    event_type event; /* filled in by wait, but not used here */
    while (true) {
        wait_for_event(&event); /* only possibility is frame_arrival */
        from_physical_layer(&r); /* go get the inbound frame */
        to_network_layer(&r, info); /* pass the data to the network layer */
    }
}
```

图 3.10 一个无限制的单工协议

该协议包括两个单独的过程：一个发送过程和一个接收过程。发送过程在源机器的数据链路层上运行；接收过程在目标机器的数据链路层上运行。这里没有用到序列号和确认，所以并不需要 MAX\_SEQ。惟一可能的事件类型是 frame\_arrival（即一个未损坏帧的到来）。

发送过程是一个无限的 while 循环，它尽可能快速地把数据送到线路上。循环体包含三个动作：从（总是就绪的）网络层获取一个分组，利用变量 s 构造一个往外发的帧，然后通过物理层发送该帧。这个协议只用到了帧结构中的 info 域，因为其他的域都跟错误控制或者流控制有关，而该协议并没有错误控制或者流控制方面的限制。

接收过程同样很简单。开始时，它等待有事情发生，这里惟一可能的事件是未损坏帧的到来。最终，帧到达，过程 wait\_for\_event 返回，其中 event 等于 frame\_arrival（这里被忽略）。调用 from\_physical\_layer 将新到达的帧从硬件缓冲区中删除，并且放到变量 r 中，所以数据链路层的代码可以访问该帧。最后，该帧的数据部分被传递给网络层，数据链路层继续回去等待下一帧，实际上它把自己挂起来，直到下一帧到来为止。

### 3.3.2 一个单工的停-等协议

现在我们去掉协议 1 中最不切实际的限制：接收方的网络层能够无限快速地处理进来的数据（或者等价的说法是，在接收方的数据链路层有一个无限数量的缓存空间，用于存储所有进来的帧，以等待它们逐渐被按照顺序进行处理）。然而，我们仍然假设通信信道不会出错，并且数据流量还是单向的。

这里我们必须处理的问题是，如何避免发送方用超过接收方处理能力的大量数据来淹没接收方。其实，如果接收方需要  $\Delta t$  的时间来执行 from\_physical\_layer 和 to\_network\_layer，则发送方发送帧的平均速度必须小于“每  $\Delta t$  时间一帧”的速度。更进一步讲，如果我们假设在接收方的硬件内没有自动缓存和排队机制的话，则发送方必须等到原来的帧被 from\_physical\_layer 取走以后才能发送新的帧，从而避免新的帧覆盖掉原来的帧。

在特定的限制条件下（比如同步传输，以及接收方的数据链路层只处理一条进来的线路），一种可能的做法是，在协议 1 的发送过程中简单地插入一个延迟，以降低发送速度，保证不淹没接收过程。然而，更常见的情形是，每个数据链路层需要处理几条线路，从一帧到达的时刻开始，到它被处理的这一段时间间隔可能会有很大差异。如果网络设计者能够计算出接收方在最差情形下的行为，则他们可以在编写发送方的程序时，把发送速度降低，使得即使每一帧都经历最大的延迟，也不会出现发送方淹没接收方的情形。这种方法的问题是太过于保守。它导致带宽的利用率远远低于最佳值，除非接收方的最好情形和最差情形几乎相同（即数据链路层的反应时间的变化极小）。

针对这个两难问题的一般化解决方案是，让接收方提供反馈信息给发送方。接收方将一个分组传递给网络层之后，它给发送方送回一个小的哑帧，实际上这一帧是给发送方一个许可，允许它发送下一帧。发送方在送出一帧之后，根据协议要求，它要等待一段时间直到小的哑帧（即确认）到达。利用接收方的反馈信息，让发送方知道什么时候可以发送更多的数据，这正是前面提到的流控制的一个例子。



发送方送出一帧,然后先等待一个确认,再继续发送,这样的协议称为**停-等协议**(**stop-and-wait**)。图 3.11 给出了一个单工的停-等协议的例子。

```

* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
sender to receiver. The communication channel is once again assumed to be error
free, as in protocol 1. However, this time, the receiver has only a finite buffer
capacity and a finite processing speed, so the protocol must explicitly prevent
the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
    frame s;          /* buffer for an outbound frame */
    packet buffer;     /* buffer for an outbound packet */
    event_type event;  /* frame_arrival is the only possibility */

    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;             /* copy it into s for transmission */
        to_physical_layer(&s);       /* bye-bye little frame */
        wait_for_event(&event);      /* do not proceed until given the go ahead */
    }
}

void receiver2(void)
{
    frame r, s;        /* buffers for frames */
    event_type event;  /* frame_arrival is the only possibility */
    while (true) {
        wait_for_event(&event);      /* only possibility is frame_arrival */
        from_physical_layer(&r);     /* go get the inbound frame */
        to_network_layer(&r, info);  /* pass the data to the network layer */
        to_physical_layer(&s);       /* send a dummy frame to awaken sender */
    }
}

```

图 3.11 一个单工的停-等协议

虽然这个例子中的数据流量是单工的,只是从发送方到接收方,但是帧的流动却是双向的。因此,两个数据链路层之间的通信信道必须具备双向传输信息的能力。然而,这个协议限定了流量的严格顺序关系:首先发送方发送一帧,然后接收方发送一帧,接着发送方发送另一帧,然后接收方发送另一帧,等等。这里用一个半双工的物理信道足够了。

就像在协议 1 中一样,发送方首先从网络层获取一个分组,用它来构造一帧,然后发

送出去。但现在,与协议 1 不同的是,发送方在回到下一轮循环并从网络层获取下一帧之前,必须等待,直到确认帧到来。发送方的数据链路层甚至根本不检查接收到的帧:对它来讲只有一种可能性,即进来的帧总是一个确认。

在 receiver1 和 receiver2 之间唯一的区别是,receiver2 将一帧递交给网络层之后,它在进入到下一轮 wait 循环之前,先给发送方送回一个确认帧。因为对于发送方来说,惟有确认帧的到来才是重要的,它的内容并不重要,所以,接收方根本不在确认帧中填充任何信息。

### 3.3.3 有噪声信道的单工协议

现在我们来考虑比较常规的情形:一个可能会出错的通信信道。帧可能会被损坏,也可能完全丢失。然而,我们假设,如果一帧在传输过程中被损坏,则接收方的硬件在计算校验和的时候会检测出来。如果一帧被损坏了之后校验和仍然是正确的(这种情况不太可能会出现),那么,这个协议(以及所有其他的协议)将会失败,即,给网络层递交了一个不正确的分组。

粗看起来,好像协议 2 稍作改变就可以工作了:增加一个定时器。发送方送出一帧,接收方只有在正确地接收到了数据之后才送回一个确认帧。如果到达接收方的是一个被损坏的帧,则它将被丢弃。经过一段时间之后发送方将超时,于是它再次发送该帧。这个过程将不断重复,直至该帧最终完好无损地到达接收方。

上面的方案有一个致命的缺陷。在继续往下读之前,请仔细想一想这个问题,并找出哪里可能会出错。

为了看清楚哪里可能会出错,请记住:数据链路层进程的任务是在两个网络层进程之间提供无错误的、透明的通信。机器 A 上的网络层将一系列分组交给它的数据链路层,而它的数据链路层则必须保证,这些分组将丝毫不差地由机器 B 上的数据链路层递交给它的网络层。特别是,B 上的网络层不可能知道一个分组丢失了,或者被复制了多份,所以,数据链路层必须保证任何传输错误都不会导致一个分组被多次递交给网络层。

考虑下面的场景:

(1) 机器 A 的网络层将分组 1 交给它的数据链路层。机器 B 正确地接收到该分组,并且将它传递给机器 B 上的网络层。机器 B 给机器 A 送回一个确认帧。

(2) 确认帧完全丢失了,所以它永远也不会到达机器 A 了。如果信道被破坏之后只丢数据帧,而不丢控制帧的话,则问题就大大简化了,但事实上,信道对这两种帧并不区别对待。

(3) 机器 A 上的数据链路层最终超时了。由于它没有收到确认,所以它(不正确地)认为,它的数据帧丢失了,或者被损坏了,于是它再次发送一个包含分组 1 的帧。

(4) 这个重复的帧也完好无损地到达了机器 B 上的数据链路层,并且无意中又被传递给其中的网络层。如果机器 A 正在给机器 B 发送一个文件,则文件中的一部分内容将会被复制(即,机器 B 所做的文件复制是不正确的,并且错误没有被检测到)。换句话说,该协议失败了。

显然,对于接收方来说,它需要有一种办法能够区分某一帧是第一次接收到的帧,还

是重传的帧。为了做到这一点,一种很显然的做法是,让发送方在它所发送的每一帧的头部放上一个序列号。然后,接收方可以检查它所接收到的每一帧的序列号,看它是新帧还是要丢弃的重复帧。

由于帧头总是越小越好,所以问题便来了:对于序列号,所需要的最小位数是多少?在这个协议中,惟一不明确的地方在于一帧  $m$  和它的直接后续帧  $m+1$  之间。如果帧  $m$  丢失了,或者被损坏了,则接收方将不会对它进行确认,所以,发送方将会不停地发送该帧。一旦这一帧被正确地接收到了,则接收方将会给发送方送回一个确认。这正是潜在的问题所在。根据确认帧是否能够正确地到达发送方,发送方决定发送帧  $m$  或者帧  $m+1$ 。

促使发送方开始发送帧  $m+2$  的事件是帧  $m+1$  的确认帧的到来。但是,这暗示着帧  $m$  已经被正确地接收到了,而且,它的确认帧也已经正确地被发送方接收到了(否则的话,发送方不会开始发送帧  $m+1$ ,更不用说帧  $m+2$  了)。因此,惟一不明确的地方在于任一帧和它的直接前一帧或者后一帧之间,而不在于前一帧和后一帧两者之间。

因此,一位序列号就足够了。在任何一个时刻,接收方期望下一个特定的序列号。如果接收到的帧包含了错误的序列号,则被认为是一个重复帧而加以拒绝。当包含正确序列号的帧到来的时候,它被接受下来,并且传递给网络层。然后,下一个期望的序列号模 2 增 1(即 0 变成 1,1 变成 0)。

图 3.12 显示了这种协议的一个例子。如果在协议中,发送方在准备下一个数据项目之前先等待一个肯定的确认,则这样的协议称为 **PAR(Positive Acknowledgement with Retransmission, 支持重传的肯定确认协议)** 或者 **ARQ(Automatic Repeat reQuest, 自动重复请求协议)**。与协议 2 类似,图 3.12 所示的协议也只有一个方向上传输数据。

协议 3 与以前的协议的不同之处在于,当发送方和接收方的数据链路层在等待状态的时候,两者都有一个变量记录下有关的值。发送方在 `next_frame_to_send` 中记录了下一个要发送的帧的序列号;接收方在 `frame_expected` 中记录了下一个期望的序列号。每个协议在进入无限循环之前都有一个简短的初始化阶段。

发送方在送出了一帧以后启动定时器。如果定时器已经在运行了,则它将被重置,以便等待另一个完整的定时器间隔。在选择定时器间隔的时候,应该保证它足够长,以便该帧到达接收方,并且按照最坏的情形让接收方处理该帧,再允许确认帧传回发送方。只有当这一段时间间隔已经过去之后,发送方才可以假定原先的帧或者它的确认帧已经丢失了,于是再重发原先的帧。如果超时间隔设置得太短的话,则发送方将会发送一些不必要的帧。虽然这些额外的帧不会影响到协议的正确性,但是会损害性能。

发送方在送出一帧并启动了定时器之后,它等待相关的事情发生。只有三种可能的情况:一个确认帧完好无损地到达;一个受损的确认帧蹒跚而至;定时器过期。如果一个有效的确认到来,则发送方从它的网络层获取下一帧,并把它放到缓冲区中,覆盖掉原来的分组。它也会增加序列号。如果一个受损的确认帧到来,或者根本没有确认帧到达,则缓冲区和序列号都不会有任何改变,所以会再次发送原来的帧。

当一个有效帧到达接收方的时候,接收方首先检查它的序列号,看它是否是一个重复分组。如果不是的话,则接受该分组,并传递给网络层,以及生成一个确认。重复的帧和受损的帧都不会传递给网络层。

```

/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1 /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void sender3(void)
{
    seq_nr next_frame_to_send; /* seq number of next outgoing frame */
    frame s; /* scratch variable */
    packet buffer; /* buffer for an outbound packet */
    event_type event;
    next_frame_to_send = 0; /* initialize outbound sequence numbers */
    from_network_layer(&s, buffer); /* fetch first packet */
    while (true) {
        s.info = buffer; /* construct a frame for transmission */
        s.seq = next_frame_to_send; /* insert sequence number in frame */
        to_physical_layer(&s); /* send it on its way */
        start_timer(s, seq); /* if answer takes too long, time out */
        wait_for_event(&event); /* frame_arrival, cksum_err, timeout */
        if (event == frame_arrival) {
            from_physical_layer(&s); /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                stop_timer(s, ack); /* turn the timer off */
                from_network_layer(&s, buffer); /* get the next one to send */
                inc(next_frame_to_send); /* invert next_frame_to_send */
            }
        }
    }
}

void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;
    frame_expected = 0;
    while (true) {
        wait_for_event(&event); /* possibilities: frame_arrival, cksum_err */
        if (event == frame_arrival) { /* a valid frame has arrived. */
            from_physical_layer(&r); /* go get the newly arrived frame */
            if (r.seq == frame_expected) { /* this is what we have been waiting for. */
                to_network_layer(&r, info); /* pass the data to the network layer */
                inc(frame_expected); /* next time expect the other sequence nr */
            }
            s.ack = 1 - frame_expected; /* tell which frame is being acked */
            to_physical_layer(&s); /* send acknowledgement */
        }
    }
}

```

图 3.12 一个支持重传的肯定确认协议

### 3.4 滑动窗口协议

在前面的协议中,数据帧只在一个方向上传输。而在大多数实际环境中,往往需要在两个方向上传输数据。实现全双工数据传输的一种办法是,使用两条独立的通信信道,每条都被用作单工数据信道(两条信道方向不同)。如果这样做的话,则我们拥有两条独立的物理线路,每条线路都有一个“前向”信道(用于数据)和一个“逆向”信道(用于确认)。在这两条线路中,逆向信道的带宽几乎完全被浪费了。实际上,用户付出了两条线路的费用,但是只使用了一条线路的容量。

一种更好的做法是,使用同一条线路来传输两个方向上的数据。毕竟,协议2和协议3已经在两个方向上传输帧了,而且逆向信道与前向信道具有同样的容量。在这种模型中,从机器A到机器B的数据帧可以与从机器A到机器B的确认帧混合在一起。接收方只要检查一下进来帧的头部中的kind域,就可以区别出该帧是数据还是确认。

尽管让数据帧和控制帧在同一条线路上传输是对前面提到的两条独立物理线路方案的一种改进,但是,更进一步的改进仍然是有可能的。当一个数据帧到达的时候,接收方并不是立即发送一个单独的控制帧,而是抑制一下自己并且开始等待,直到网络层传递给它下一个分组。然后,确认信息被附在往外发送的数据帧上(使用帧头中的ack域)。实际上,确认报文搭了下一个外发数据帧的便车。这种“将确认暂时延迟以便可以钩到下一个外发数据帧”的技术称为捎带确认(piggybacking)。

与单独确认帧的方法相比较,使用捎带确认法最主要的好处是更好地利用了信道的带宽。帧头中的ack域只占用几位的开销,而一个单独的帧则需要一个头、确认和校验和。而且,发送的帧越少,也意味着“帧到达”中断也越少,占用接收方的缓冲区可能也越少,这要取决于接收方的软件是如何实现的。在下一个要讨论的协议中,捎带域只占用帧头中的1位,它很少会占用许多位。

然而,捎带确认法也引入了一个在单独确认法中不曾出现过的复杂问题。为了捎带一个确认,数据链路层应该等待下一个分组多长时间?如果数据链路层等待的时间超过了发送方的超时周期,则该帧将会被重传,从而违背了确认机制的本意。如果数据链路层是一个先知者,能够预测将来,那么它就知道下一个网络层分组什么时候会到来,因此可以确定是继续等待下去,还是立即发送一个单独的确认帧,这取决于计划的等待时间是多长。当然,数据链路层不可能预测将来,所以它必须采用某种特殊的方法,比如等待一个固定的毫秒数。如果一个新的分组很快就到来了,则确认就可以被捎带上去。否则的话,如果在这段时间周期之前没有新的分组到来,则数据链路层只是简单地发送一个单独的确认帧。

接下去的三个协议都是双向协议,它们属于同一类,称为滑动窗口协议(sliding window protocol)。这三个协议在效率、复杂性和缓冲区需求等各个方面有所不同,本节后面将会讨论到。如同所有的滑动窗口协议一样,在这三个协议中,每个外发的帧都包含一个序列号,其范围从0到某一个最大值。最大值通常是 $2^n - 1$ ,这样序列号正好可以填入到一个n位的域中。停-等滑动窗口协议使用 $n=1$ ,限制序列号为0和1,但是更加复



杂的版本可以使用任意的  $n$ 。

所有滑动窗口协议的本质是,在任何时刻,发送方总是维持着一组序列号,分别对应于允许它发送的帧。我们称这些帧落在**发送窗口(sending window)**之内。类似地,接收方也维持着一个**接收窗口(receiving window)**,对应于一组允许它接受的帧。发送方的窗口和接收方的窗口不必有同样的上下界,甚至也不必有同样的大小。在有些协议中,这两个窗口有固定的大小,但是在其他一些协议中,它们可以随着帧的发送和接收而增大或者缩小。

尽管这些协议使得数据链路层在发送和接收帧的顺序方面有了更多的自由度,但是我们肯定不能放弃一个基本的需求,即,数据链路层协议将分组递交给网络层的次序必须与发送机器上分组被传递给数据链路层的次序相同。我们同样也不能改变这样的需求:物理通信信道就像“线”一样,也就是说,它必须按照发送的顺序递交所有的帧。

发送方窗口内的序列号代表了那些已经被发送,但是还没有被确认的帧,或者是那些可以被发送的帧。任何时候当有新的分组从网络层到来时,它被赋予下一个最高的序列号,并且窗口的上边界增一。按照这种方法,该窗口持续地维持了一系列未被确认的帧。图 3.13 显示了一个例子。

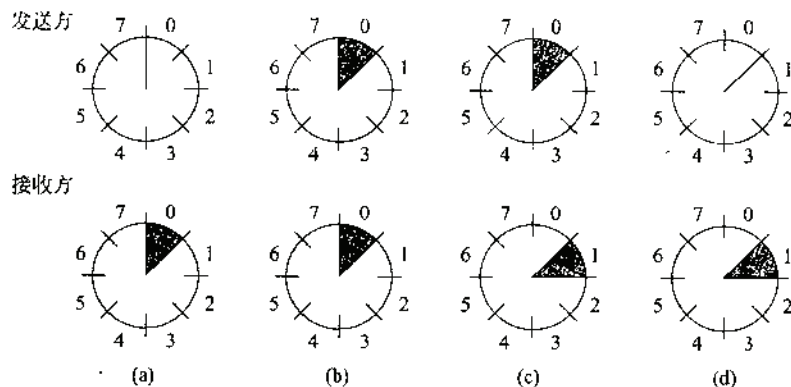


图 3.13 一个大小为 1、有 3 位序列号的滑动窗口

(a) 初始时; (b) 第一帧发送以后; (c) 第一帧接收以后; (d) 第一个确认收到以后

由于当前在发送方窗口内的帧最终有可能在传输过程中丢失或者被损坏,所以,发送方必须在内存中保存好所有这些帧,以便可能进行重传。因此,如果最大的窗口尺寸为  $n$ ,则发送方需要  $n$  个缓冲区来存放未被确认的帧。如果该窗口在某个时候真的达到了它的最大尺寸,则发送方的数据链路层必须强行关闭网络层,直到有一个缓冲区空闲出来为止。

接收方数据链路层的窗口对应于它可以接受的帧。任何落在窗口外面的帧都被丢弃,无需任何提示。当一个新接收到的帧的序列号等于窗口下边界的时候,接收方将它传递给网络层,并生成一个确认,然后将整个窗口向前移动 1 个位置。与发送方的窗口不同的是,接收方的窗口总是保持最初的大小。请注意,窗口的大小为 1 意味着数据链路层只按顺序接受帧,但是对于大一点的窗口,这一条便不再成立。相反,网络层总是按照正

确的顺序接受数据,跟数据链路层的窗口大小没有关系。

图 3.13 显示了一个最大窗口尺寸为 1 的例子。刚开始的时候,没有帧要发送,所以发送方窗口的上下边界相等,但是随着时间的推移,状态的变化如该图所示。

### 3.4.1 1 位滑动窗口协议

在讨论一般的情形以前,我们先来考察一个最大窗口尺寸为 1 的滑动窗口协议。由于发送方在送出一帧以后,在发送下一帧之前要等待前一帧的确认,所以这样的 1 位滑动窗口协议使用了停等的办法。

图 3.14 描述了这样一个协议。跟其他的协议一样,它也是从定义变量开始的。`next_frame_to_send` 指明了发送方正在发送的那一帧。类似地,`frame_expected` 指明了接收方正在等待的那一帧。这两个变量的取值只可能是 0 和 1。

在一般情况下,两个数据链路层中的某一个首先开始,并发送第一帧。换句话说,只有一个数据链路层程序应该在主循环外面包含 `to_physical_layer` 和 `start_timer` 过程调用。如果两个数据链路层同时启动的话,这是一种很罕见的情形,后面再讨论。初始启动的机器从它的网络层获取到第一个分组,然后根据该分组创建一帧,并将它发送出去。当这一帧(或者任何其他帧)到达时,接收方的数据链路层检查该帧,看它是否为重复帧,如同协议 3 一样。如果这一帧正是所期望的,则将它传递给网络层,并且接收方的窗口向前滑动。

确认域包含了刚才最后接收到的、并且没有错误的帧的序列号。如果该序列号与发送方当前正在发送的帧的序列号一致,则发送方知道,存储在 `buffer` 中的帧已经处理完毕,于是,它就可以从网络层获取下一个分组。如果序列号不一致,则它必须继续发送同一帧。无论什么时候只要接收到一帧,则也要送回一帧。

现在我们来看一看协议 4 对于不正常情形的适应能力怎么样。假设计算机 A 试图将它的第 0 帧发送给计算机 B,同时 B 也试图将它的第 0 帧发送给 A。假定 A 发送一帧给 B,但是 A 的超时时间有点太短。因此,A 可能会不停地超时,因而发送一系列相同的帧,并且所有这些帧的 `seq=0`,以及 `ack=1`。

当第一个有效帧到达计算机 B 的时候,它将会被接受,并且 `frame_expected` 设置为 1。所有的后续帧将被拒绝,因为 B 现在正在等待序列号为 1 的帧,而不是序列号为 0 的帧。而且,由于所有的重复帧都有 `ack=1`,并且 B 仍然在等待第 0 帧的确认,所以它不会从网络层获取新的分组。

在每一个被拒绝的重复帧进来之后,B 向 A 发送一帧,其中包含 `seq=0` 和 `ack=0`。最后,这些帧中总会有一帧正确地到达 A,从而引起 A 开始发送下一个分组。丢帧或者提前超时的任何一种组合都不会导致该协议向网络层递交重复的分组,漏掉一个分组,或死锁。

然而,如果双方并发地发送一个初始分组,则会出现一种极为罕见的情形。这种同步的困难程度如图 3.15 所示。(a)部分显示了该协议的正常操作。(b)部分显示了这种罕见的情形。如果 B 在发送自己的帧之前先等待 A 的第一帧,则整个过程如图中(a)所示,每一帧都被接受。然而,如果 A 和 B 同时开始通信,则它们的第一帧有交叉,然后数据链

路层进入到(b)的状态。在(a)中,每一帧到来后都会带给网络层一个新的分组;这里没有任何重复。在(b)中,即使没有传输错误,也会有一半的帧是重复的。类似的情形也发生在提前超时的情况下;有一方明显地提前开始也会发生这样的情形。实际上,如果发生多个提前超时的话,则每一帧有可能要发送三次或者更多。

```

* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1      /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
    seq_nr next_frame_to_send; /* 0 or 1 only */
    seq_nr frame_expected; /* 0 or 1 only */
    frame r, s; /* scratch variables */
    packet buffer; /* current packet being sent */
    event_type event;
    next_frame_to_send = 0; /* next frame on the outbound stream */
    frame_expected = 0; /* frame expected next */
    from_network_layer(&buffer); /* fetch a packet from the network layer */
    s.info = buffer; /* prepare to send the initial frame */
    s.seq = next_frame_to_send; /* insert sequence number into frame */
    s.ack = 1 - frame_expected; /* piggybacked ack */
    to_physical_layer(&s); /* transmit the frame */
    start_timer(s.seq); /* start the timer running */
    while (true) {
        wait_for_event(&event); /* frame_arrival, cksum_err, or timeout */
        if (event == frame_arrival) { /* a frame has arrived undamaged. */
            from_physical_layer(&r); /* go get it */
            if (r.seq == frame_expected) { /* handle inbound frame stream. */
                to_network_layer(&r, info); /* pass packet to network layer */
                inc(frame_expected); /* invert seq number expected next */
            }
            if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
                stop_timer(r.ack); /* turn the timer off */
                from_network_layer(&buffer); /* fetch new pkt from network layer */
                inc(next_frame_to_send); /* invert sender's sequence number */
            }
        }
        s.info = buffer; /* construct outbound frame */
        s.seq = next_frame_to_send; /* insert sequence number into it */
        s.ack = 1 - frame_expected; /* seq number of last received frame */
        to_physical_layer(&s); /* transmit a frame */
        start_timer(s.seq); /* start the timer running */
    }
}

```

图 3.14 1 位滑动窗口协议



送方也会用完它的窗口。这两个因子的乘积基本上说明了这条管道的容量,发送方为了达到尖峰效率,需要利用这条管道来马不停蹄地发送数据。

这项技术称为**管道化技术(pipelining)**。如果信道的容量是  $b$  位/秒,帧长为  $1$  位,往返传输时间为  $R$  秒,则传输一帧所需要的时间为  $1/b$  秒。在一个数据帧的最后一位被发送出去之后,经过  $R/2$  秒的延迟之后该位到达接收方,再经过  $R/2$  秒的延迟之后确认帧回来,总共延迟为  $R$ 。在停-等协议中,这条线路有  $1/b$  秒是忙的,而  $R$  秒是空闲的,所以,线路的利用率  $= 1/(1+bR)$

如果  $1 \ll bR$ ,则效率小于 50%。由于确认帧传输回来总是有一个非 0 的延迟,所以,原则上讲,管道化技术可以使得线路在这段时间中也是忙的。但是,如果这段间隔很小的话,则没有必要将协议搞得这么复杂。

在不可靠通信信道上使用管道化技术来发送帧也会引起一些严重的问题。首先,如果在一个很长的帧流中间有一帧被损坏了,或者丢失了,那该怎么办呢?在发送方发现这一帧有错误之前,有大量的帧已成功到达接收方。当一个被损坏的帧到达接收方时,很显然它应该被丢弃掉,但是,接收方应该如何处理后续到达的正确帧呢?前面提到过,接收方的数据链路层必须按照顺序将分组递交给网络层。在图 3.16 中,我们可以看到管道化技术对于错误恢复的影响。现在我们来细致地考察这个问题。

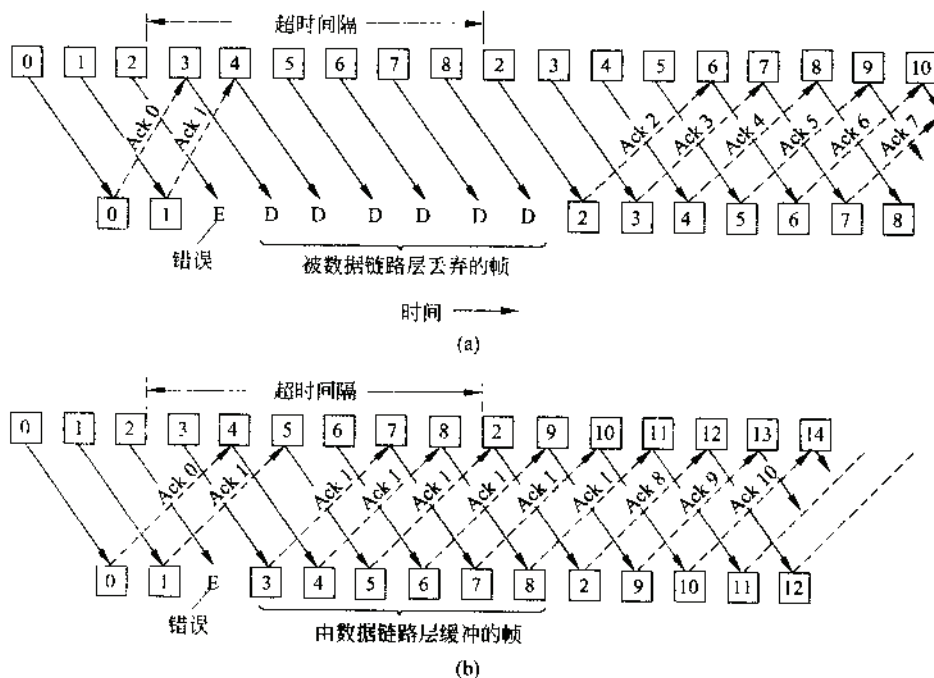


图 3.16 管道化技术与错误恢复

在两种情况下错误的影响:(a)接收方的窗口尺寸为1;(b)接收方的窗口尺寸较大

在使用了管道化技术之后,有两种办法可以用来处理错误。一种办法称为**回退  $n$  帧**



(go back n),它很简单,接收方只要丢弃所有后续的帧,并且不为这些丢弃的帧发送确认即可。这种策略对应于“接收窗口的尺寸为1”的情形。换句话说,除了数据链路层必须要递交给网络层的下一帧以外,它不接受任何帧。如果在定时器超时以前,发送方的窗口已经满了,则管道将空闲。最终,发送方将会超时,并且按照顺序重传所有未被确认的帧,从最初受损或者丢失的那一帧开始。如果错误率比较高的话,则这种方法可能会浪费大量的带宽。

在图 3.16(a)中,我们可以看到回退 n 帧的情形,其中发送方的窗口比较大。第 0 帧和第 1 帧被正确地接收和确认。然而,第 2 帧被损坏或者丢失了。发送方并不知道出现了问题,它继续发送后续的帧,直到第 2 帧的定时器过期为止。然后,它退回第 2 帧,并从这里开始发送 2、3、4 帧,等等。

另一种处理错误的通用策略称为选择性重传(selective repeat)。当使用了这种策略以后,接收到的坏帧被丢弃,但是坏帧后面的好帧继续被缓存起来。当发送方超时以后,它只重传最早的未被确认的那一帧。如果那一帧正确到达接收方的话,则接收方依次将它所缓存的帧递交给网络层。选择性重传策略通常也跟以下的策略结合起来使用:当接收方检测到错误(例如,帧的校验和错误或者序列号不正确)时,它发送一个否定的确认(NAK, negative acknowledgement)。NAK 可以激发重传操作,而不需要等到相应的定时器过期,因此,NAK 可以提高性能。

在图 3.16(b)中,第 0 帧和第 1 帧被正确接收到,并得到确认,而第 2 帧则丢失了。当第 3 帧到达接收方时,那里的数据链路层注意到它已经错过了一帧,所以它针对第 2 帧送回一个 NAK,但是将第 3 帧缓存起来。当第 4 和 5 帧到达之后,它们也被数据链路层缓存起来,而不是被传递给网络层。第 2 帧的 NAK 达到发送方后,发送方立即重新发送第 2 帧。当该帧到达接收方时,数据链路层现在有了第 2、3、4 和 5 帧,于是就可以将它们按照正确的顺序传递给网络层,也可以确认所有这些帧(从第 2 帧到第 5 帧),如图中所示。如果 NAK 也丢失了的话,则发送方的第 2 帧定时器最终会超时,发送方就会重新发送第 2 帧(仅仅这一帧),但是,这可能会非常滞后了。所以,从效果上看,NAK 加快了某一特定帧的重传过程。

选择性重传策略对应于接收方窗口大于 1 的情形。窗口内的任何一帧都能够被接受,并被缓存起来,等到所有此前的帧都到达之后,再传递给网络层。如果窗口很大的话,这种方法需要消耗大量的数据链路层内存。

这两种不同的方法正好是带宽和数据链路层缓存空间的权衡。根据每一种资源的紧缺程度,可以选择使用其中某种方法。图 3.17 显示了一个管道化协议,其中接收方的数据链路层只按序接受进来的帧,发生错误之后的帧都被丢弃。在这个协议中,我们第一次抛掉了“网络层总是有无穷多的分组要发送”的假设。当网络层希望发送一个分组时,它可以引发一个 network\_layer\_ready 事件。然而,为了强制“在任何时候未确认帧的个数都不应该超过 MAX\_SEQ”的流控制规则,数据链路层应该能够阻止网络层给予它过多的工作。库过程 enable\_network\_layer 和 disable\_network\_layer 可以完成这样的功能。

```

/* Protocol 5 (go back n) allows multiple outstanding frames. The sender may transmit up
to MAX_SEQ frames without waiting for an ack. In addition, unlike in the previous
protocols, the network layer is not assumed to have a new packet all the time. Instead,
the network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7          /* should be 2^n - 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
    /* Return true if a <= b < c circularly; false otherwise. */
    if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
    else
        return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
    /* Construct and send a data frame. */
    frame s;          /* scratch variable */

    s.info = buffer[frame_nr];    /* insert packet into frame */
    s.seq = frame_nr;            /* insert sequence number into frame */
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);    /* piggyback ack */
    to_physical_layer(&s);    /* transmit the frame */
    start_timer(frame_nr);    /* start the timer running */
}

void protocol5(void)
{
    seq_nr next_frame_to_send;    /* MAX_SEQ > 1; used for outbound stream */
    seq_nr ack_expected;          /* oldest frame as yet unacknowledged */
    seq_nr frame_expected;    /* next frame expected on inbound stream */
    frame r;          /* scratch variable */
    packet buffer[MAX_SEQ + 1];    /* buffers for the outbound stream */
    seq_nr nbuffered;          /* # output buffers currently in use */
    seq_nr i;          /* used to index into the buffer array */
    event_type event;

    enable_network_layer();    /* allow network_layer_ready events */
    ack_expected = 0;    /* next ack expected inbound */
    next_frame_to_send = 0;    /* next frame going out */

```

图 3.17 一个使用了回退 n 帧技术的滑动窗口协议

```

frame_expected = 0;          /* number of frame expected inbound */
nbuffered = 0;              /* initially no packets are buffered */
while (true) {
    wait_for_event(&event);    /* four possibilities; see event_type above */
    switch(event) {
        case network_layer_ready: /* the network layer has a packet to send */
            /* Accept, save, and transmit a new frame. */
            from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
            nbuffered = nbuffered + 1; /* expand the sender's window */
            send_data(next_frame_to_send, frame_expected, buffer);
                                    /* transmit the frame */
            inc(next_frame_to_send); /* advance sender's upper window edge */
            break;

        case frame_arrival: /* a data or control frame has arrived */
            from_physical_layer(&r); /* get incoming frame from physical layer */

            if (r.seq == frame_expected) {
                /* Frames are accepted only in order. */
                to_network_layer(&r.info); /* pass packet to network layer */
                inc(frame_expected);
                                    /* advance lower edge of receiver's window */
            }

            /* Ack n implies n - 1, n - 2, etc. Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* Handle piggybacked ack. */
                nbuffered = nbuffered - 1; /* one frame fewer buffered */
                stop_timer(ack_expected); /* frame arrived intact; stop timer */
                inc(ack_expected); /* contract sender's window */
            }
            break;

        case cksum_err: break; /* just ignore bad frames */

        case timeout: /* trouble; retransmit all outstanding frames */
            next_frame_to_send = ack_expected; /* start retransmitting here */
            for (i = 1; i <= nbuffered; i++) {
                send_data(next_frame_to_send, frame_expected, buffer);
                                    /* resend 1 frame */
                inc(next_frame_to_send); /* prepare to send the next one */
            }
    }
}

```

图 3.17 (续)

```

    }

    if (nbuffered < MAX_SEQ)
        enable_network_layer();
    else
        disable_network_layer();
}
}

```

图 3.17 (续)

注意, 尽管不同序列号的个数是  $\text{MAX\_SEQ} + 1$ , 分别为  $0, 1, 2, \dots, \text{MAX\_SEQ}$ , 但是, 在任何时候, 未确认帧的最大数量是  $\text{MAX\_SEQ}$ , 而不是  $\text{MAX\_SEQ} + 1$ 。为了看清楚为什么这个限制是必要的, 请考虑下面  $\text{MAX\_SEQ} = 7$  的场景。

- (1) 发送方发送第  $0 \sim 7$  帧。
- (2) 第 7 帧的捎带确认最终被送回到发送方。
- (3) 发送方送出另外 8 帧, 其序列号仍然是  $0 \sim 7$ 。
- (4) 现在第 7 帧的另一个捎带确认也回来了。

问题来了: 属于第二批的 8 帧全部成功到达了吗? 或者这 8 帧全部丢失了(把出错之后丢弃的帧也算作丢失了)? 在这两种情况下, 接收方都会发送第 7 帧的确认。而发送方无从分辨。由于这个原因, 未确认帧的最大数目必须限制为  $\text{MAX\_SEQ}$ 。

虽然协议 5 没有缓存出错后到来的帧, 但是它也没有因此而摆脱缓存的问题。由于发送方可能在将来的某个时刻重传所有未被确认的帧, 所以, 它必须把已经送出的帧保留所有一段时间, 直到它知道接收方已经接受了这些帧。当第  $n$  帧的确认到来时, 第  $n-1$  帧、第  $n-2$  帧等也都自动被确认了。当有些先前捎带确认的帧被丢失或者受损之后, 这个特性显得尤为重要。当任何一个确认到来时, 数据链路层都要检查是否可以释放一些缓冲区。如果释放掉一些缓冲区(即窗口中又有了可以利用的空间), 则原来被阻塞的网络层现在又可以激发更多的 `network_layer_ready` 事件了。

对于这个协议, 我们假设链路上总是有反向的流量可以捎带确认。如果没有这样的流量, 则这些确认报文就不会被送回来。协议 4 并不需要这样的假设, 因为它每次接收到一帧就送回一帧, 即使它刚刚已经送出了这一帧也是如此。在下一个协议中, 我们将用一种非常巧妙的办法来解决这个单向流量的问题。

因为协议 5 有多个未被确认的帧, 所以逻辑上它需要多个定时器, 即每一个未被确认的帧都需要一个定时器。每一帧的超时是独立的, 相互之间没有关系。用软件很容易模拟所有这些定时器: 只需使用一个硬件时钟, 它周期性地引发中断。所有未发生的超时事件构成了一个链表, 链表中的每个节点描述了一个定时器离过期还有多少个时钟滴答、该定时器为哪一帧计时, 以及一个指针指向下一个节点。

为了演示如何实现多个定时器, 请考虑图 3.18(a)中的例子。假设每 100ms 时钟滴答一次。刚开始的时候, 实际的时间为  $10:00:00.0$ ; 有三个超时事件, 分别定在  $10:00:00.5$ 、 $10:00:01.3$  和  $10:00:01.9$ 。每次硬件时钟滴答到来时, 实际的时间被更新, 链表头上的滴答计数器被减 1。当滴答计数器变成 0 的时候, 就引发一个超时事件, 并将

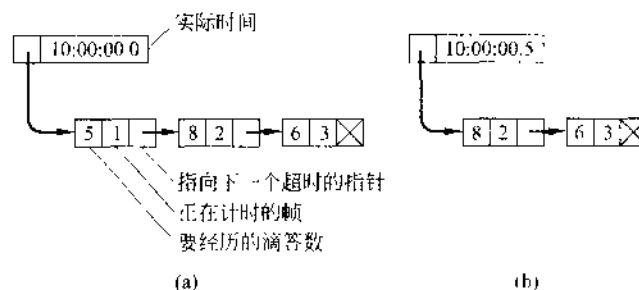


图 3.18 用软件来模拟多个定时器

该节点从链表中移除,如图 3.18(b)所示。虽然这种实现方式也要求在调用 start\_timer 或 stop\_timer 时扫描该链表,但是,在每次滴答中断时它并不要求更多的工作。在协议 5 中, start\_timer 和 stop\_timer 这两个过程都带一个参数,表示正在对哪一帧计时。

### 3.4.3 使用选择性重传的协议

如果错误很少发生的话,则协议 5 可以工作得很好;但如果线路质量很差的话,则在重传的帧上要浪费很多带宽。另一种处理错误的策略是,对于坏帧或者丢失帧后面的帧,接收方也可以接受并缓存起来。这样的协议不会仅仅因为前面的帧被损坏或者丢失,而丢弃后续的帧。

在这个协议中,发送方和接收方都维持了一个窗口,窗口内部包含了那些可以接受的序列号。发送方的窗口大小从 0 开始,以后可以增大到某一个预设的最大值 MAX\_SEQ。而接收方的窗口总是固定大小的,其大小等于 MAX\_SEQ。接收方为其窗口内的每一个序列号保留了一个缓冲区。与每一个缓冲区相关联的还有一位(arrived),用来指明该缓冲区是满的还是空的。任何时候当一帧到达时,接收方通过 between 函数检查它的序列号,看是否落在窗口内。如果确实落在窗口内,并且以前还没有接收到这一帧,则接受该帧,并且保存起来。不管这一帧是否包含了网络层所期望的下一个分组,这个过程是肯定要执行的。当然,该帧一定会被保存在数据链路层中,并且,直到所有序列号比它小的那些帧都已经被按照正确的顺序递交给网络层之后,它才会被传递给网络层。图 3.19 显示了一个使用该算法的协议。

```

/* Protocol 6 (selective repeat) accepts frames out of order but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7 /* should be 2^n - 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_
type;

```

图 3.19 使用选择性重传的滑动窗口协议



```

#include "protocol.h"
boolean no_nak = true; /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1; /* initial value is only for the simulator */
static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
    /* Same as between in protocol5, but shorter and more obscure. */
    return ((a <= b) && (b <= c)) || ((c <= a) && (a <= b)) || ((b < c) && (c < a));
}
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
    /* Construct and send a data, ack, or nak frame. */
    frame s; /* scratch variable */
    s.kind = fk; /* kind == data, ack, or nak */
    if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
    s.seq = frame_nr; /* only meaningful for data frames */
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
    if (fk == nak) no_nak = false; /* one nak per frame, please */
    to_physical_layer(&s); /* transmit the frame */
    if (fk == data) start_timer(frame_nr % NR_BUFS);
    stop_ack_timer(); /* no need for separate ack frame */
}
void protocol6(void)
{
    seq_nr ack_expected; /* lower edge of sender's window */
    seq_nr next_frame_to_send; /* upper edge of sender's window + 1 */
    seq_nr frame_expected; /* lower edge of receiver's window */
    seq_nr too_far; /* upper edge of receiver's window + 1 */
    int i; /* index into buffer pool */
    frame r; /* scratch variable */
    packet out_buf[NR_BUFS]; /* buffers for the outbound stream */
    packet in_buf[NR_BUFS]; /* buffers for the inbound stream */
    boolean arrived[NR_BUFS]; /* inbound bit map */
    seq_nr nbuffered; /* how many output buffers currently used */
    event_type event;

    enable_network_layer(); /* initialize */
    ack_expected = 0; /* next ack expected on the inbound stream */
    next_frame_to_send = 0; /* number of next outgoing frame */
    frame_expected = 0;
    too_far = NR_BUFS;
    nbuffered = 0; /* initially no packets are buffered */
    for (i = 0; i < NR_BUFS; i++) arrived[i] = false;

```

图 3.19 (续)

```

while (true) {
    wait_for_event(&event);    /* five possibilities: see event_type above */
    switch(event) {
        case network_layer_ready: /* accept, save, and transmit a new frame */
            nbuffered = nbuffered + 1;    /* expand the window */
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);
                                                    /* fetch new packet */
            send_frame(data, next_frame_to_send, frame_expected, out_buf);
                                                    /* transmit the frame */
            inc(next_frame_to_send); /* advance upper window edge */
            break;

        case frame_arrival:    /* a data or control frame has arrived */
            from_physical_layer(&r); /* fetch incoming frame from physical layer */
            if (r.kind == data) {
                /* An undamaged frame has arrived. */
                if ((r.seq != frame_expected) && no_nak)
                    send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
                if (between(frame_expected, r.seq, too_far) && (arrived[r.seq % NR_BUFS]
                    == false)) {
                    /* Frames may be accepted in any order. */
                    arrived[r.seq % NR_BUFS] = true;    /* mark buffer as full */
                    in_buf[r.seq % NR_BUFS] = r.info; /* insert data into buffer */
                    while (arrived[frame_expected % NR_BUFS]) {
                        /* Pass frames and advance window. */
                        to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected); /* advance lower edge of receiver's
                                                    window */
                        inc(too_far); /* advance upper edge of receiver's window */
                        start_ack_timer(); /* to see if a separate ack is
                                                    needed */
                    }
                }
            }
            if ((r.kind == nak) && between(ack_expected, (r.ack + 1) % (MAX_SEQ + 1), next
                _frame_to_send))
                send_frame(data, (r.ack + 1) % (MAX_SEQ + 1), frame_expected, out_buf);

            while (between(ack_expected, r.ack, next_frame_to_send)) {

```

图 3.19 (续)

```

        nbuffered = nbuffered - 1; /* handle piggybacked ack */
        stop_timer(ack_expected % NR_BUFS); /* frame arrived intact */
        inc(ack_expected); /* advance lower edge of sender's window */
    }
    break;

case cksum_err:
    if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
    break;

case timeout:
    send_frame(data, oldest_frame, frame_expected, out_buf); /* we timed out */
    break;

case ack_timeout:
    send_frame(ack, 0, frame_expected, out_buf); /* ack timer expired; send ack */
}
if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
}

```

图 3.19 (续)

非顺序接收带来了一些特殊的问题,那些只按顺序接受帧的协议是不用考虑这些问题的。我们用一个例子就可以很容易地演示出其中的麻烦之处。假设我们使用 3 位的序列号,那么,发送方允许连续发送 7 帧,然后开始等待确认。刚开始时,发送方和接收方的窗口如图 3.20(a)所示。现在发送方送出第 0~6 帧。接收方的窗口允许它接受任何序列号在 0~6(含)之间的帧。这 7 帧全部正确地到达了,所以接收方对它们进行确认,并且向前移动它的窗口,允许接收第 7、0、1、2、3、4 或 5 帧,如图 3.20(b)所示。所有这 7 个缓冲区都标记为空。

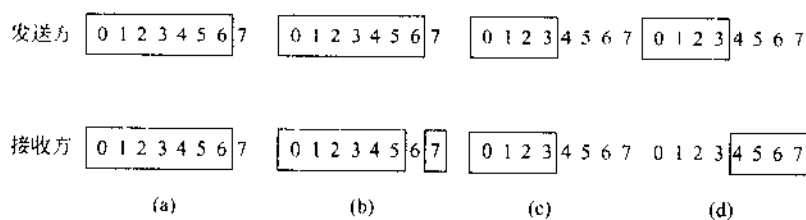


图 3.20

(a) 窗口大小为 7 的初始状态; (b) 7 帧都已送出并接收,但是均未被确认;  
(c) 窗口大小为 4 的初始状态; (d) 4 帧都已送出并接收,但是均未被确认

此时,灾难降临了,闪电击中了电线杆子,所有的确认都被毁掉了。发送方最终超时了,并且重发第 0 帧。当这一帧到达接收方时,接收方检查它的序列号,看是否落在它的窗口中。不幸的是,如图 3.20(b)所示,第 0 帧落在新的窗口中,所以它被接受了。接收

方送回第 6 帧的捎带确认, 因为第 0~6 帧都已经接收到了。

发送方很高兴地得知, 所有它发出去的帧都已经正确地到达了, 所以它向前移动它的窗口, 并立即发送第 7、0、1、2、3、4 和 5 帧。第 7 帧将被接收方接受, 并且它的分组直接传递给网络层。紧接着, 接收方的数据链路层进行检查, 看它是否已经有了一个有效的第 0 帧, 它发现确实已经有了 (即前面重发的第 0 帧), 然后把内嵌的分组传递给网络层。因此, 网络层得到了一个不正确的分组, 从而导致协议失败。

这个问题的本质是, 当接收方向前移动了它的窗口之后, 新的有效序列号范围与老的范围有重叠。因此, 后续的一批帧可能是重复的帧 (如果所有的确认都丢失了), 也可能是新的帧 (如果所有的确认都接收到了)。接收方将无法区分这两种情形。

解决这个难题的办法是, 确保接收方向前移动窗口之后, 新的窗口与老的窗口之间没有重叠。为了保证没有重叠, 最大的窗口尺寸应该不超过序列号范围的一半, 如图 3.20(c) 和 3.20(d) 所做的那样。例如, 如果用 4 位来表达序列号, 则序列号的范围为 0~15。在任何时候, 应该只有 8 个未被确认的帧处于等待状态。按照这种方法, 如果接收方已经接受了 0~7 帧, 并且向前移动了窗口, 以便允许接收第 8~15 帧, 那么, 它可以明确地区分出后续的帧是重传帧 (序列号为 0~7), 还是新帧 (序列号为 8~15)。一般地, 协议 6 的窗口尺寸为  $(MAX\_SEQ+1)/2$ 。因此, 对于 3 位序列号, 窗口尺寸为 4。

一个有意思的问题是: 接收方必须具有多少个缓冲区? 无论如何, 接收方不可能接受序列号低于窗口下边界的帧, 也不可能接受序列号高于上边界的帧。因此, 所需要的缓冲区的数量等于窗口的尺寸, 而不是序列号的范围。在上面的 4 位序列号的例子中, 只需要 8 个缓冲区就够了, 编号为 0~7。当第  $i$  帧到达时, 它被放在  $(i \bmod 8)$  号缓冲区中。请注意, 虽然  $i$  和  $(i+8) \bmod 8$  在“竞争”同一个缓冲区, 但是, 它们永远不会同时在窗口内, 因为, 如果那样的话, 则窗口的尺寸至少为 9。

出于同样的原因, 所需要的定时器的数量也等于缓冲区的数量, 而不是序列号空间的大小。实际上, 每个缓冲区都有一个相关联的定时器。当定时器超时, 缓冲区的内容就要被重传。

在协议 5 中, 有一个隐含的假设是: 信道的负载很重。当一帧到达时, 接收方并不立即发送确认。相反, 它把该帧的确认放在下一个往外发的数据帧中捎带回去。如果反向的流量很轻, 则确认信息将会滞留很长时间。如果在一个方向上有很大的流量, 而另一个方向上根本没有流量, 则只有  $MAX\_SEQ$  个分组被发送出去, 然后协议就阻塞了, 这就是为什么我们必须假设总是有反向流量的原因。

在协议 6 中, 这个问题被修正了。当一个按正常次序发送的数据帧到达之后, 接收方通过 `start_ack_timer` 启动一个辅助的定时器。如果在定时器到期之前, 没有出现反向的流量, 则发送一个单独的确认帧。由于该辅助定时器而导致的中断称为 `ack_timeout` 事件。利用这种方式, 即使单向的数据流也没有问题了, 缺少可以捎带确认的反向数据帧不再是一个障碍了。只需要一个辅助定时器就可以了, 如果该定时器正在运行时 `start_ack_timer` 又被调用了, 则它被重置为一个完整的确认超时时间。

与辅助定时器关联的超时时间应该明显短于与数据帧关联的定时器间隔, 这是非常关键的。这个条件是必要的, 它应该保证一个正确接收的帧能够尽早地被确认, 从而该帧

的重传定时器还没有过期,所以还没有重传该帧。

协议 6 使用了比协议 5 更加有效的策略来处理错误。当接收方有理由怀疑出现了错误时,它就给发送方送回一个否定的确认(NAK)帧。这样的帧实际上是一个重传请求,在 NAK 中指定了要重传的帧。有两种情形接收方应该怀疑:接收到一个受损的帧,或者到达的帧并非是自己所期望的(可能有丢帧错误)。为了避免多次请求重传同一个丢失帧,接收方应该记录下对于某一帧是否已经发送过 NAK。在协议 6 中,如果对于 `frame_expected` 还没有发送过 NAK,则变量 `no_nak` 为 `true`。如果 NAK 被损坏了,或者丢失了,则不会有实质性的伤害,因为发送方最终会超时,无论如何它会重传丢失的帧。如果一个 NAK 被发送出去之后丢了,而接收方又收到一个错误的帧,则 `no_nak` 将为 `true`,并且辅助定时器将被启动。当辅助定时器超时后,一个 ACK 帧将被发送出去,以便将发送方重新同步到接收方的当前状态。

在有些情况下,从一帧被发送出去开始,该帧到达目的地,再在那里被处理,然后它的确认被送回来,这整个过程所需要的时间(几乎)是个常数。在这样的情况下,发送方可以调整它的定时器,让它略微大于正常情况下从发送一帧到接收到它的确认之间的时间间隔。然而,如果这段时间的变化非常大,则发送方必须做出选择,要么将定时器的间隔设置得比较小(其风险是不必要的重传),要么将它设置得比较大(发生错误之后长时间地空闲)。

这两种选择都会浪费带宽。如果反向的流量比较稀少,则确认之前的时间将非常不规则,当有反向流量的时候这段时间非常短,当没有反向流量的时候这段时间非常长。在接收方内部,这种不稳定的处理时间也是一个问题。一般地,当确认间隔的标准偏差与间隔本身相比非常小的时候,定时器可以设置得“紧”一点,这时候 NAK 并不很有用。否则的话,定时器应该设置得“松”一点,从而避免不必要的重传,但是 NAK 可以加快丢失帧或者损坏帧的重传速度。

与超时和 NAK 紧密相关的一个问题是确定哪一帧引发了超时。在协议 5 中,它总是 `ack_expected` 的帧,因为它总是最早的帧。在协议 6 中,没有一种很具体的办法来确定谁引发了超时。假定已经发送了第 0~4 帧,这意味着未确认帧的列表是 01234,按照时间先后顺序排列。现在请想象这样的情形:第 0 帧超时了,第 5 帧(新帧)被发送出去了,第 1 帧超时了,第 2 帧超时了,第 6 帧(又一新帧)也被发送出去了。这时候,未确认帧的列表是 3405126,也是按照发送的时间先后顺序排列。如果所有进来的流量(即,那些包含确认的帧)被丢失一段时间,则这 7 个未确认的帧将会依次超时。

为了避免使该例子过于复杂,我们没有显示定时器的管理过程。我们只是假设在超时时 `oldest_frame` 变量已经设置好了,它指示出哪一帧超时了。

### 3.5 协议验证

实际的协议,以及实现这些协议的程序通常是非常复杂的。因此,人们已经做了大量的研究工作,试图寻找一些形式化的、数学的技术来描述和验证这些协议。在本节中,我们将学习一些模型和技术。虽然我们是在数据链路层的章节中来学习这些内容,但是,这



些内容同样也适用于其他的层。

### 3.5.1 有限状态机模型

在许多协议模型中用到的一个关键概念是有限状态机(finite state machine)。利用这项技术,每个协议机(protocol machine,即发送方或接收方)在任何一个时刻,总是处于一种特定的状态。它的状态是由所有变量的值组成的,其中也包括程序计数器。

在大多数情况下,出于分析的目的,大量的状态可以分成少量的组。例如,考虑协议 3 中的接收方,我们可以从所有可能的状态中抽象出两种最重要的状态:等待 0 号帧,或等待 1 号帧。所有其他的状态都可以看作是瞬时态,只是为到达其中一个主状态的中间步骤而已。通常总是选择协议机在等待下一事件发生的那些时刻(在我们的例子中,即是执行 wait(event)过程调用的时刻)作为协议机的状态。此时,协议机的状态完全由它的变量的状态来确定。于是,状态的数量为  $2^n$ ,这里  $n$  是表达所有这些变量所需要的位数。

整个系统的状态是两个协议机和信道的所有状态的组合。信道的状态是由它的内容来决定的。仍然以协议 3 作为例子,信道有 4 种可能的状态:0 号帧或 1 号帧从发送方往接收方移动,确认帧沿着相反的方向移动,或信道为空。如果我们将发送方和接收方抽象成每一方都有两种状态,则整个系统有 16 种不同的状态。

关于信道的状态有必要澄清一下。所谓一帧“在信道上”,当然是一种抽象的说法。我们实际所指的意思是,一帧可能已经被接收到了,但是在目标方还没有被处理。在协议机执行 FromPhysicalLayer 并对一帧进行处理之前,我们都认为这一帧还“在信道上”。

对于每一种状态,有 0 个或者多个可能的转换(transition),从而到达其他的状态。当有某个事件发生时,状态转换就会发生。对于一个协议机,在以下几种情况下可能会发生转换:一帧被送出、一帧到达、一个定时器到期、产生一个中断,等等。对于信道来说,典型的事件是协议机将一个新帧插入到信道上、一帧被递交给一个协议机,或者由于噪声丢失了一帧。一旦给出了协议机和信道特征的完整描述,我们就有可能画出一个有向图,其中,所有的状态显示为节点,而所有的转换显示为弧。

初始状态(initial state)是一种特殊的状态。它对应于系统开始运行时,或稍后某个方便的起始时刻系统的状况。从初始状态开始,经过一序列的转换可以到达某一些(或者所有)其他的状态。利用图论中一些著名的技术(比如,计算一个图的传递闭包),就有可能确定哪些状态是可达的,哪些状态是不可达的。这样的技术称为可达性分析(reachability analysis)(Lin et al., 1987)。这种分析对于确定一个协议的正确性非常有帮助。

从形式上,一个协议的有限状态机模型可以看作是一个四元组(S, M, I, T),其中:

- S 是指进程和信道可能的状态集合。
- M 是指能在信道上进行交换的帧的集合。
- I 是指进程的初始状态集合。
- T 是状态之间转换的集合。

刚开始时,所有的进程都处于初始状态。然后,各种事件相继发生,比如待传输的帧

已经准备好,或者定时器到期,等等。每一个事件都可能引起一个进程或者信道有所行动,并转换成一种新的状态。只要仔细地列举出每种状态的各种可能的后续状态,就可以构造出可达性图,并且分析该协议。

可达性分析也可以用来检查协议规范中的各种错误。例如,如果某一特定的帧出现在某一特定的状态是有可能的,但是有限状态机并没有说明应该采取何种行动,则该协议规范是错误的(至少是不完整的)。如果存在这样一组状态:从这些状态出发既没有出口,也不能再前进下去(即不能再正确地接收帧),那么我们犯了另一个错误(死锁)。另一个不太严重的错误是,协议规范说明了在某一种状态下如何处理一个事件,而实际上在这种状态下该事件不可能发生(多余的变迁)。其他一些错误也可以被检测出来。

作为有限状态机模型的一个例子,请考虑图 3.21(a)。该图对应于前面介绍的协议 3:每个协议机有 2 种状态,信道上 4 种状态,总共有 16 种状态。然而,从初始状态出发,这些状态并不都是可达的。图中没有画出不可达的状态。为了简化起见,这里省略了校验和错误。

每种状态都由 3 个字符 SRC 来标识,其中 S 为 0 或 1,对应于发送方正在试图发送的那一帧;R 也是 0 或 1,对应于接收方期望接收的那一帧;C 为 0、1、A 或者空(-),对应于信道的状态。在这个例子中,初始状态为(000)。换句话说,发送方发送了 0 号帧,接收方正在等待 0 号帧,并且 0 号帧当前正在信道上。

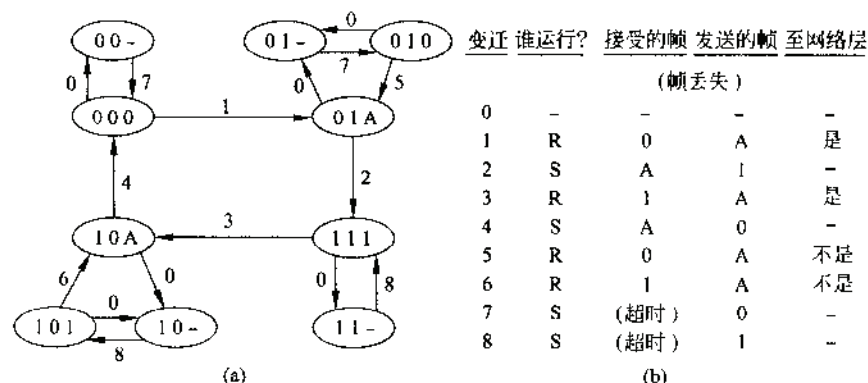


图 3.21

(a) 协议 3 的状态图; (b) 转换

图 3.21 显示了 9 种状态变迁。变迁 0 为信道丢失了其上的内容。变迁 1 为信道正确地将分组 0 递交给接收方,然后接收方将它的状态改变为期望接收 1 号帧,并且发送一个确认。1 号变迁也对应于接收方将分组 0 递交给网络层。其他的变迁如图 3.21(b) 中所列。该图没有显示出当到达帧的校验和发生错误时的情形,因为这种帧的到来并不改变状态(在协议 3 中)。

在正常的操作过程中,变迁 1、2、3 和 4 顺序地不断重复。在每一轮循环中,两个分组被递交,从而将发送方带回到初始状态,即试图发送序列号为 0 的一个新帧。如果信道丢失了 0 号帧,则协议从状态(000)变迁到状态(00-)。最终,发送方超时(变迁 7),系统又回

到(000)。丢失确认帧的情形要复杂一些,它要求两个变迁:7和5,或者8和6,才能修复过来。

采用1位序列号的协议所必须具备的一个特征是:不管事件发生的顺序如何,接收方永远也不会连续递交两个奇数序列号的分组,即中间没有偶数序列号的分组,反之也成立。从图3.21中的状态图可以看出,这样的要求可以用更加形式化的语言来描述:“从初始状态出发,不存在这样的路径:变迁1发生两次,并且在这两次事件中间没有发生变迁3;反之亦然”。从图中可以看出,单就这一点而言,协议是正确的。

一个类似的要求是,不存在这样的路径:它使得发送方改变状态两次(比如,从0到1,再回到0),而接收方的状态仍然不变。假如存在这样的路径的话,那么从对应的事件序列来看,则是两帧彻底地丢失了,而接收方没有察觉。在被递交的分组序列中,将会有一个无法检测的缝隙。

作为一个协议,另一个重要的特征是不允许存在死锁。**死锁(deadlock)**是指这样的情形:不管事件序列如何发生,协议都不可能再前进了(即,不再向网络层递交分组)。从状态图模型的角度来看,死锁的特征可以描述如下:存在这样一个状态子集,从初始状态可以到达该状态子集中,但是该状态子集具有以下两个特性:

- (1) 没有通向子集外部的变迁。
- (2) 在子集中不存在能引起进一步前行的变迁。

一旦一个协议进入到死锁状态,它就永远停在里边了。同样地,很容易从图3.21可以看出,协议3没有死锁。

### 3.5.2 Petri 网模型

有限状态机并不是用形式化手段描述协议的惟一技术。在本小节中,我们将介绍一种完全不同的技术,称为**Petri 网(Petri net)**(Danthine, 1980)。一个 Petri 网有4个基本元素:库所(place)、变迁(transition)、弧(arc)和标记(token)。一个库所(**place**)代表了该系统(或部分系统)可能处的状态。图3.22显示了一个 Petri 网,它有两个库所A和B,图中用圆圈来表示库所。系统当前处于状态A中,所以,在库所A中用一个标记(**token**,粗黑点)可以表示这一点。**转换(transition)**用一个水平的或者垂直的条来表示。每个转换有零个或者多个输入弧,这些输入弧从该转换的输入库所进来,同时它还有零个或者多个输出弧,通过这些输出弧到达它的输出库所。

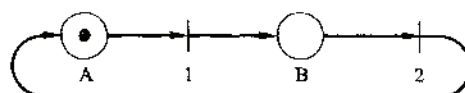


图 3.22 包含 2 个库所和 2 个转换的 Petri 网

如果在一个转换的每个输入库所中至少有一个输入标记,则称该转换是**激活的(enabled)**。任何一个激活的转换随时都可以**激发(fire)**,即,从每个输入库所中删除一个标记,并且在每个输出库所中放上一个标记。如果输入弧的数量与输出弧的数量不相等,则标记就不会保持不变。如果有两个或者多个转换是激活的,则其中任何一个都可以激

发。到底选择激发哪一个转换,这是不确定的,这也正是用 Petri 网来模拟协议之所以非常有用的原因。图 3.22 中的 Petri 网是确定的,它可以被用来模拟任何两阶段的处理过程(比如婴儿的行为:吃、睡、吃、睡,等等)。如同所有的模拟工具一样,不必要的细节都被忽略了。

图 3.23 给出了图 3.12 的 Petri 网模型。与有限状态机模型不同的是,这里没有复合的状态;发送方的状态、信道的状态和接收方的状态都是单独描述的。转换 1 和 2 对应于发送方送出 0 号帧后在正常情况下和超时情况下的行为。转换 3 和转换 4 对应于 1 号帧的情形。转换 5、6 和 7 分别对应于 0 号帧丢失、确认帧丢失和 1 号帧丢失的情形。当到达接收方的数据帧包含错误的序列号的时候,转换 8 和 9 就会发生。转换 10 和 11 代表了下一帧顺序到达接收方以及该帧被递交给网络层的情形。

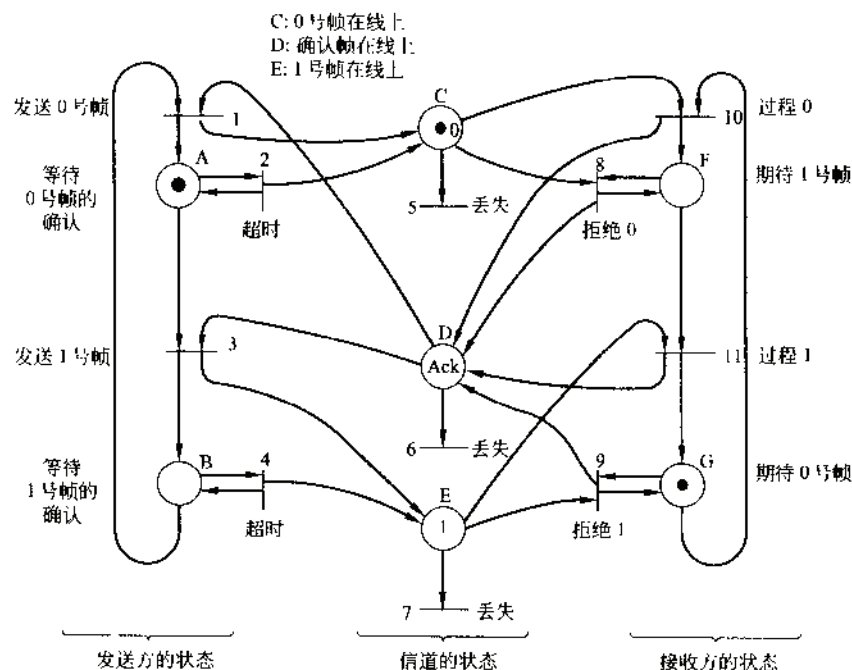


图 3.23 协议 3 的 Petri 网模型

Petri 网也可以用来检测协议中的错误,其方式类似于有限状态机的用法。例如,如果某一个激发序列包含了两次转换 10,但是这两次转换 10 的中间没有转换 11,则协议是不正确的。在 Petri 网中的死锁概念与在有限状态机中的类似。

Petri 网也可以用类似于文法的代数形式来表示。每一个转换对应于文法中的一条规则。每条规则定义了相应转换的输入和输出库所。由于图 3.23 中有 11 个转换,所以它的文法有 11 条规则,我们用 1-11 进行编号,每条规则对应于同一编号的转换。图 3.23 的 Petri 网的文法如下:

1:  $BD \rightarrow AC$

2:  $\Lambda \rightarrow A$

- 3:  $AD \rightarrow BE$
- 4:  $B \rightarrow B$
- 5:  $C \rightarrow$
- 6:  $D \rightarrow$
- 7:  $E \rightarrow$
- 8:  $CF \rightarrow DF$
- 9:  $EG \rightarrow DG$
- 10:  $CG \rightarrow DF$
- 11:  $EF \rightarrow DG$

你可以看出,这确实很有意思,我们把一个复杂的协议简化为 11 条简单的文法规则,而用计算机程序来维护这些文法规则是非常容易的。

Petri 网的当前状态可以被表示为一组无顺序关系的库所集合,每个库所在集合中的重复次数等于它的标记数。任何一条规则,如果其左边的库所都出现了,它就可以被激发,结果是,这些库所从当前状态中删除,并且将该规则的输出库所加入到当前状态中。图 3.23 的状态被标为 ACG,即 A、C 和 G 各有一个标记。因此,规则 2、5 和 10 都是激活的,它们中的任何一条规则都可以被应用,从而会到达一种新的状态(有可能与原来的状态一样)。相反,规则 3( $AD \rightarrow BE$ )不可能被应用,因为 D 没有标记。

### 3.6 数据链路层协议示例

在本节中,我们将介绍几个已经被广泛使用的数据链路协议。第一个是 HDLC,它是一个经典的面向位的协议,该协议的一些变种版本在许多应用中已经被使用几十年了。第二个数据链路协议是 PPP,它被用来将家庭计算机连接到 Internet 上。

#### 3.6.1 HDLC—高级数据链路控制

在本小节中,我们将讨论一组紧密相关的协议,这些协议虽然有点老了,但是仍在广泛使用。它们都演变自最初在 IBM 大型机领域中使用的数据链路协议:SDLC (Synchronous Data Link Control,同步数据链路控制)协议。IBM 在开发了 SDLC 之后,将它提交给 ANSI 和 ISO,希望接受成为美国标准和国际标准。ANSI 对它进行了修改,变成了 ADCCP (Advanced Data Communication Control Procedure,高级数据通信控制规程);ISO 将它修改成为 HDLC (High-level Data Link Control,高级数据链路控制)。然后,CCITT 采纳并修改了 HDLC,作为它的 LAP (Link Access Procedure,链路访问规程),并成为 X.25 网络接口标准的一部分,但是后来又将它修改为 LAPB,使之与 HDLC 的后来版本更加兼容。这么多标准的好处在于,你有很多的标准可供选择。而且,如果你不喜欢这些标准的话,你可以等待下一年的模型。

这些协议都基于相同的原理。所有协议都是面向位的,并且为了确保数据的透明性,它们都使用了位填充。它们相互之间只有一些细微的差别,但是这些差别多少让人觉得不愉快。以下关于面向位的协议的讨论只是一般性的介绍,至于每一种协议的特殊细节,

请参考有关的材料。

所有的面向位的协议都使用了如图 3.24 所示的帧结构。其中 address(地址)域对于有多个终端的线路显得尤为重要,因为在这样的环境中,该域被用于标识一个终端。对于点到点的线路,它有时候被用来区分命令和应答。

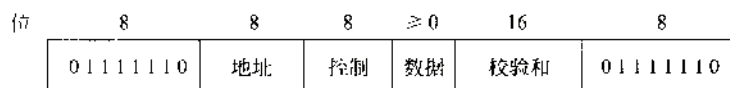


图 3.24 面向位的协议的帧格式

控制(Control)域被用作序列号、确认,以及其他的用途,后面还要进一步讨论。

数据(Data)域可以包含任何信息。它可以任意长,不过,随着帧长度的增加,校验和的效率会降低,因为多个突发性错误的概率会加大。

校验和(Checksum)域是一个循环冗余码,在 3.2.2 节中我们讨论了这种编码的技术。

帧的分界是另一个标志序列(01111110)。在空闲的点到点线路上,它连续不断地传输标志序列。最小的帧包含三个域,总共 32 位(其中不含两头的标志序列)。

共有三种类型的帧:信息帧(Information)、管理帧(Supervisory)和无序号的帧(Unnumbered)。对于这三种帧,控制域的内容如图 3.25 所示。协议使用了一个滑动窗口,其中序列号的长度为 3 位。在任何时候都允许有 7 个未被确认的帧处于等待状态。图 3.25(a)中的 Seq 域是帧的序列号。Next 域是一个捎带的确认。然而,所有的协议都遵从这样的约定:所捎带的并不是最近正确接收到的帧的序列号,而是尚未接收到的第一帧的序列号(即期望接收的下一帧)。是选择最近接收到的帧,还是期望接收的下一帧,这可以是任意的;使用哪一种约定方式并没有关系,只要在协议中保持一致即可。

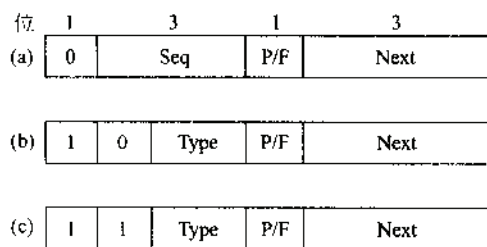


图 3.25 三种帧的 Control 域

(a) 信息帧; (b) 管理帧; (c) 无序号的帧

P/F 位代表 Poll/Final(查询/结束)。当一台计算机(或者集中器)正在询问一组终端时需要用到该域。当用作 P 的时候,该计算机正在请求终端发送数据。终端发送的所有帧,除了最后一帧以外,都将 P/F 位设置为 P,最后一帧的 P/F 位设置为 F。

在有些协议中,P/F 位被用于强迫其他的机器立即发送一个管理帧,而不是等待反向的流量以便捎带窗口信息。P/F 位与无序号帧一起还有一些不很重要的用途。



各种不同的管理帧可通过 Type(类型)域来区分。Type 0 是一个确认帧(正式的名称为 RECEIVE READY,接收就绪),通过这样的确认帧可以指示下一个期望的帧。当没有反向的流量可以用来捎带的时候,就可以使用 Type 0 管理帧。

Type 1 是一个否定的确认帧(正式名称是 REJECT,拒绝)。它用来指示一个传输错误已经被检测到了。Next 域指明了第一个没有被正确接收到的帧的序列号(即期望被重传的帧)。发送方必须重传从 Next 开始的所有未被确认的帧。这种策略与我们的协议 5 比较类似,而不是协议 6。

Type 2 是 RECEIVE NOT READY(接收尚未就绪)。如同 Type 0(RECEIVE READY)一样,它确认直到 Next 的所有帧,但是不包含 Next 帧,但是它告诉发送方不要再发送了。RECEIVE NOT READY 的用意是,通知发送方现在接收方临时有了问题,比如缓冲区短缺,但是,它并不作为滑动窗口流控制的替代办法。当接收方的问题恢复之后,它发送一个 RECEIVE READY、REJECT 或者其他特定的控制帧。

Type 3 是 SELECTIVE REJECT(选择性拒绝)。它只要求重传指定的帧。从这一层意义上,它更像协议 6 而不是协议 5,所以,当发送方的窗口尺寸小于等于序列号空间大小一半的时候,它特别有用。因此,如果接收方希望将乱序的帧缓存起来以备将来可能使用的话,它可以使用 SELECTIVE REJECT 管理帧来强迫重传任何一帧。HDLC 和 ADCCP 都支持这种类型的帧,但是 SDLC 和 LAPB 不允许这样的帧(即,协议中没有 SELECTIVE REJECT),并且没有定义 Type 3 类型的帧。

第三类帧是无序号帧。它有时候被用于控制的用途,但是当要求提供不可靠的无连接服务时,它也可以承载数据。在前面介绍的信息帧和管理帧中,各种面向位的协议基本上是一致的,但是,在无序号帧中,不同的协议有很大的差别。其中有 5 位用于指明帧的类型,但并不是所有的 32 种可能都用到了。

所有的协议都提供了一个命令 DISC(DISConnect),通过该命令,一台机器可以宣布它要宕机了(比如,为了预防性的维护)。它们还有另一个命令,允许一台刚刚连接在线的机器宣布它又回来了,并且强制所有的序列号都回到 0。该命令被称为 SNRM(Set Normal Response Mode,设置正常响应模式)。不幸的是,“正常响应模式”其实并不正常。它是一种不平衡(即非对称)的模式,线路的一头是“主”,另一头是“从”。SNRM 来源于早期时候,当时数据通信意味着一个哑终端与一台大的主机进行通信,这当然是非对称的。为了使协议更加适合于通信双方较为平等的情形,HDLC 和 LAPB 增加了一个命令 SABM(Set Asynchronous Balanced Mode,设置异步的平衡模式),它重置线路,并宣称双方是平等的。这两个协议还有两个命令 SABME 和 SNRME,这两个命令分别等同于 SABM 和 SNRM,但是它们允许使用 7 位序列号的扩展帧格式,而不是 3 位序列号。

所有协议都提供的第三个命令是 FRMR(FraMe Reject,帧拒绝),它被用于指示所到达的帧虽然校验和正确,但是语义不正确。语义错误的例子有:LAPB 中出现了 Type 为 3 的管理帧、短于 32 位的帧、非法的控制帧、序号落在窗口之外的帧的确认帧,等等。FRMR 帧包含一个 24 位的数据域,用于指明该帧有什么错误。数据域中包含了坏帧的控制域、窗口的参数,以及一些用于指示特定错误的位。

如同数据帧一样,控制帧也可能会丢失,或者损坏,所以控制帧也必须被确认。一种

特殊的、称为 UA(Unnumbered Acknowledgement, 无序号的确认)的控制帧正用于这样的目的。由于只有一个控制帧可能处于未确认的状态,所以,“对哪一个帧进行确认”这个问题并不存在歧义。

其他的控制帧被用来处理初始化、查询和状态报告。还有一种控制帧可以包含任意的信息,即 UI(Unnumbered Information: 无序号信息)。这些数据并不被传递给网络层,而是由接收方的数据链路层自己使用。

尽管 HDLC 被广泛使用了,但是它还远远不够完美。有关 HDLC 的各种问题的讨论请查询(Fiorini et al., 1994)。

### 3.6.2 Internet 中的数据链路层

Internet 是由大量的机器(主机和路由器)以及连接这些机器的通信设施构成的。在单个建筑物内,通常使用 LAN 来实现互连;但是绝大多数的广域设施则是通过点到点的租用线路构建起来的。在第 4 章中,我们将讨论 LAN;这里我们将讨论 Internet 中点到点线路上所使用的数据链路协议。

在实践中,点到点通信主要被用于两种情形。第一,数以千计的组织有一个或者多个 LAN,每个 LAN 都有一定数量的主机(个人计算机、用户工作站、服务器等),以及一个路由器(或者网桥,其功能相似)。通常,这些路由器通过一个骨干 LAN 相互连接起来。典型情况下,所有与外界的连接都经过一个或者两个路由器,这个路由器或者这两个路由器通过点到点的租用线路连接到远程的路由器。正是这些路由器和它们的租用线路构成了通信子网,而 Internet 也正是在这些通信子网上建立起来的。

点到点线路在 Internet 上扮演重要角色的第二种情形是:上百万的个人用户要在家利用调制解调器和拨号电话线连接到 Internet。通常的过程是这样的:用户的家庭 PC 机呼叫某一个 Internet 服务供应商的路由器,然后就好像一台全功能的 Internet 主机那样工作。这种操作方法与“在 PC 机和路由器之间通过租用线路进行通信”并无区别,只不过,当用户终止了会话之后,PC 机与路由器之间的连接也随之被终止。图 3.26 演示了家庭 PC 机呼叫 Internet 服务供应商的情形。图中的调制解调器被显示在计算机的外部,这样可以强调它的角色的重要性,现代的计算机大都使用内置的调制解调器。

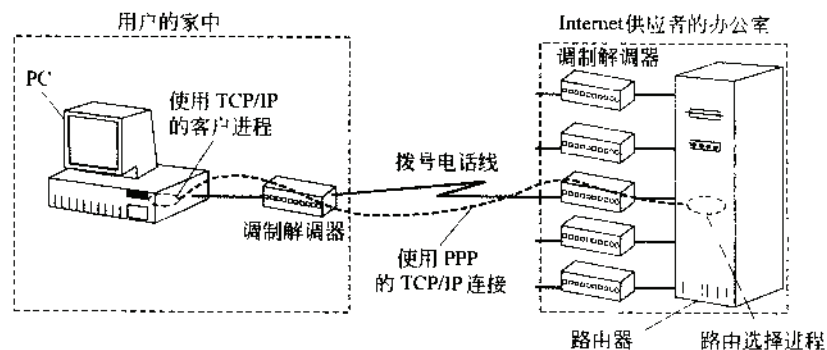


图 3.26 一台家庭个人计算机成为一台 Internet 主机

无论是从路由器到路由器的租用线路的连接,还是从主机到路由器的拨号连接,在线路上都需要某一种点到点的数据链路协议,来完成成帧、错误控制以及本章中我们学习过的其他的数据链路层功能。在 Internet 中使用的数据链路协议称为 PPP,现在我们来讨论该协议。

### PPP 点到点协议(Point-to-Point Protocol)

Internet 需要一个点到点协议,它有多种用途,其中包括传送从路由器到路由器之间的流量,以及从家庭用户到 ISP 之间的流量。该协议是 PPP(Point-to-Point Protocol,点到点协议),RFC1661 定义了该协议,其他几个 RFC(例如 RFC 1662 和 1663)又进一步详细地对它进行了阐述。PPP 处理错误检测、支持多个协议、允许在连接时刻协商 IP 地址、允许身份认证,等等还有其他许多特性。

PPP 提供了 3 类功能:

(1) 一种成帧方法。它可以毫无歧义地分割出一帧的结束和下一帧的开始。并且帧格式支持错误检测。

(2) 一个链路控制协议,可用于启动线路、测试线路、协商参数,以及当线路不再需要的时候可以温和地关闭线路。该协议称为 LCP(Link Control Protocol,链路控制协议)。它支持同步和异步线路,也支持面向字节的和面向位的编码方法。

(3) 一种协商网络层选项的方法,并且协商方法与所使用的网络层协议独立。所选择的方法对于每一种支持的网络层都有一个不同的 NCP(Network Control Protocol,网络控制协议)。

为了看清楚这几部分是如何组合起来的,我们来考虑一种典型的情形:一个家庭用户呼叫一个 Internet 服务供应商,以便让它的家庭 PC 机成为一台临时的 Internet 主机。PC 机首先通过调制解调器呼叫供应商的路由器。当路由器的调制解调器回答了用户的电话呼叫,并建立起一个物理连接之后,PC 机给路由器发送一系列 LCP 分组,它们被包含在一个或者多个 PPP 帧的有效载荷域中。这些分组以及它们的应答信息将选定所使用的 PPP 参数。

一旦双方对 PPP 参数达成一致之后,又会发送一系列 NCP 分组,这些 NCP 分组用于配置网络层。通常情况下,PC 机希望运行一个 TCP/IP 协议栈,所以它需要一个 IP 地址。由于没有足够的 IP 地址可供使用,所以,通常每个 Internet 供应商都会先得到一段 IP 地址范围,然后动态地分配一个地址给每台新近登录的 PC 机,保证它在登录会话过程中使用该地址。如果一个供应商拥有  $n$  个 IP 地址,则它可以允许同时有  $n$  台机器登录进来,但是它的总用户数可以是  $n$  的许多倍。针对 IP 协议的 NCP 负责分配 IP 地址。

这时候,PC 机已经成为一台 Internet 主机,它可以发送和接收 IP 分组,就如同直接硬件连接的 Internet 主机一样。当用户完成了工作以后,NCP 断掉网络层连接,并释放 IP 地址。然后 NCP 停掉数据链路层连接。最后,计算机通知调制解调器挂断电话,释放物理层的连接。

PPP 选择的帧格式与 HDLC 的帧格式非常相似,因为没有理由再重新发明一种新的格式。PPP 和 HDLC 之间最主要的区别是,PPP 是面向字符的,而不是面向位的。特别

是,PPP 在拨号调制解调器线路上使用了字节填充技术,所以,所有的帧都是整数个字节。因此,要想发送一个包含 30.25 个字节的帧是不可能的,而在 HDLC 中则是可能的。PPP 帧不仅可以通过拨号电话线发送出去,也可以通过 SONET 或者真正的面向位的 HDLC 线路(比如从路由器到路由器之间的连接)发送出去。图 3.27 显示了 PPP 帧的格式。

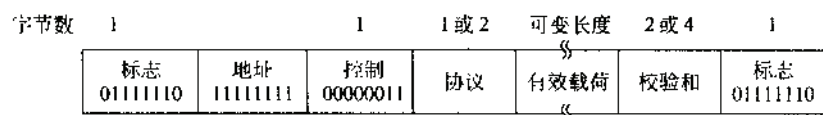


图 3.27 无序号模式操作下的 PPP 完整帧格式

所有的 PPP 帧都以一个标准的 HDLC 标志字节(01111110)作为开始,如果它正好出现在有效载荷域中,则需要字节填充。接下来是地址域,它总是被设置成二进制值 11111111,以表示所有的站都可以接受该帧。使用这样的值可以避免“必须分配数据链路地址”的问题。

地址域之后是控制域,该域的默认值是 00000011。此值表明了这是一个无序号帧。换句话说,在默认方式下,PPP 并没有采用序列号和确认来实现可靠传输。在有噪声的环境下(比如无线网络),可以利用编号模式来实现可靠传输。RFC 1663 定义了确切的细节,但是在实践中它很少被采用。

由于在默认配置下,地址和控制域总是常量,所以 LCP 提供了必要的机制,允许双方协商一个选项,该选项的目的仅仅是省略这两个域,因而每一帧可以节约 2 个字节。

第四个 PPP 域是协议域,它的任务是指明有效载荷域中是哪一种分组。已定义了代码的协议为: LCP、NCP、IP、IPX、AppleTalk 和其他的协议。以 0 位作为开始的协议是网络层协议,比如 IP、IPX、OSI CLNP、XNS。以 1 位作为开始的协议被用于协商其他的协议,这包括 LCP,以及每一个支持的网络层协议都有一个不同的 NCP。协议域的默认大小为 2 字节,但是,通过 LCP 可以将它协商为 1 个字节。

有效载荷域是变长的,最多可达到某一个商定的最大值。如果在线路建立过程中没有通过 LCP 协商该长度,则使用默认长度 1500 字节。如果有需要的话,在有效载荷之后可以加一些填充字节。

在有效载荷域之后是校验和域,通常该域为 2 个字节,但通过协商也可以是 4 个字节。

总而言之,PPP 是一种多协议成帧机制,它适合于在调制解调器、HDLC 位序列线路、SONET 和其他的物理层上使用。它支持错误检测、选项协商、头部压缩,以及(可选)使用 HDLC 类型帧格式的可靠传输。

现在我们从 PPP 帧格式转移到线路的启动和关闭机制上来。图 3.28 是一个简化了的状态图,它显示了一条线路从被启动、使用,一直到被关闭的全过程。该过程不仅适用于调制解调器连接,也适用于从路由器到路由器的连接。

协议从线路处于死(DEAD)状态开始,这时候没有物理层的线路接入进来,也没有物理层连接存在。当物理连接被建立起来之后,线路转移到建立(ESTABLISH)状态。此时,LCP 选项协商开始,如果成功的话,线路状态转移到身份认证(AUTHENTICATE)。

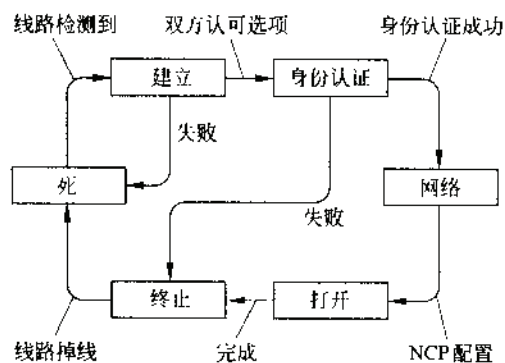


图 3.28 一个简化的状态图：从线路启动到线路关闭

现在，双方可以根据需要检查对方的身份。当进入到网络(NETWORK)状态时，通过调用适当的 NCP 协议可以配置网络层。如果配置成功的话，则到达打开(OPEN)状态，于是可以传输数据了。当数据传输任务完成之后，线路转移到终止(TERMINATE)状态，当线路掉线后，就从终止回到死状态。

在建立状态阶段，LCP 协商数据链路协议的选项。LCP 协议并不真正关心这些选项本身，相反，它关心的是用于协商的机制。它提供了这样一种方法：允许发起进程提出建议，而应答进程接受或者拒绝该建议(可以是全部，也可以是部分)。它同时还提供了一种方法以允许两个进程测试线路质量，看它们是否认为线路足够好到可以建立一个连接。最后，LCP 协议允许当线路不再需要的时候，关闭这些线路。

RFC 1661 中定义了 11 种 LCP 帧。图 3.29 列出了这些帧的含义。4 种 Configure-类型的帧允许发起方(I)提出建议选项值、应答方(R)接受或者拒绝这些建议值。如果应答方拒绝的话，则它可以提出另外的建议值，或者宣布它根本不愿意协商特定的选项。被协商的选项以及它们的建议值都是 LCP 帧的一部分。

名 称	方 向	说 明
Configure-request	I → R	列出建议的选项和值
Configure-ack	I ← R	所有的选项均接受
Configure-nak	I ← R	有些选项没有被接受
Configure-reject	I ← R	有些选项是不可协商的
Terminate-request	I → R	请求关闭线路
Terminate-ack	I ← R	OK, 线路关闭了
Code-reject	I ← R	接收到了未知的请求
Protocol-reject	I ← R	被请求的协议未知
Echo-request	I → R	请送回该帧
Echo-reply	I ← R	这是送回的帧
Discard-request	I → R	只需丢弃该帧(用于测试)

图 3.29 LCP 帧的类型



当一条线路不再需要的时候,通过 Terminate-类型的帧可以将它关闭。Code-reject 和 Protocol-reject 码表示应答方获得了某些无法理解的信息。这种情况有可能意味着发生了一个未被检测到的传输错误,但是更有可能意味着发起方和应答方运行的 LCP 协议的版本不同。Echo-类型的帧可被用于测试线路的质量。最后,Discard-request 类型的帧可帮助调试。如果任何一方在将数据发送到线路上时出现了问题的话,则程序员可以使用这种类型的帧来进行测试。如果该帧到达了目的地,则接收方只要直接丢弃即可,而不要采取一些有可能迷惑测试人员的动作。

可以被协商的选项包括:数据帧的最大有效载荷长度、允许身份认证和选择所用的认证协议、允许在正常操作过程中监视线路质量,以及选择各种头部压缩选项。

关于 NCP 协议,从一般性介绍的角度几乎没有什么可说的了。每一个 NCP 协议都针对某个特定的网络层协议,通过该 NCP 协议,可以配置相应的网络层协议。例如,对于 IP 协议,动态地址分配是最重要的可能选项。

### 3.7 本章小结

数据链路层的任务是将物理层提供的原始位流转换成可供网络层使用的帧流。数据链路层用到了各种成帧的方法,包括字符计数法、字节填充法和位填充法。数据链路协议可以提供错误控制能力,以便重传损坏的或者丢失的帧。为了避免快速的发送方淹没一个慢速的接收方,数据链路协议还要提供流控制功能。滑动窗口机制是一种被广泛使用的技术,它可以方便地将错误控制和流控制结合起来考虑。

滑动窗口协议可以按照发送方的窗口大小和接收方的窗口大小来进行分类。当两个窗口的大小都是 1 的时候,滑动窗口协议变成了停-等协议。当发送方的窗口大于 1(例如,为了避免发送方由于长的传输延迟而阻塞线路)的时候,接收方可以有两种实现办法:除了下一个顺序帧以外其他的帧都丢弃;或者将所有乱序的帧都缓存起来,一直到需要这些帧的时候。

在本章中我们讨论了一系列协议。协议 1 是针对理想的无错误的环境而设计的协议,其中接收方可以处理任何发送给它的流量。协议 2 仍然假设在一个无错误的环境中,但是引入了流控制。协议 3 通过引入序列号,并使用停-等算法来处理错误。协议 4 允许双向通信,并且引入了捎带确认的概念。协议 5 使用了一个滑动窗口,支持回退  $n$  帧。最后,协议 6 使用了选择性重发和否定确认的技术。

有多种技术可以用来为协议建模,通过建模有助于分析协议的正确性(或者有遗漏)。有限状态机和 Petri 网模型通常被用于这样的目的。

许多网络的数据链路层使用了某一种面向位的协议,例如 SDLC、HDLC、ADCCP 或者 LAPB。所有这些协议都使用标志字节作为帧的分界,并且使用位填充技术来避免在数据中出现标志字节。所有这些协议都使用一个滑动窗口来实现流控制。Internet 使用 PPP 作为点到点线路上的基本数据链路协议。



## 习 题

1. 一个上层的分组被切分成 10 帧,每一帧有 80% 的机会可以无损坏地到达。如果数据链路协议没有提供错误控制的话,请问,该报文平均需要发送多少次才能完整地到达接收方?

2. 数据链路协议中使用了下面的字符编码:

A: 01000111; B: 11100011; FLAG: 01111110; ESC: 11100000

为了传输一个包含 4 个字符的帧: A B ESC FLAG, 请给出当使用下面的成帧方法时所对应的位序列(用二进制表达):

(a) 字符计数。

(b) 包含字节填充的标志字节。

(c) 包含位填充的起始和结束标志。

3. 数据片断(A B ESC C ESC FLAG FLAG D)出现在一个数据流的中间,而成帧方法采用的是本章介绍的字节填充算法,请问经过填充之后的输出是什么?

4. 你的一个同学 Scrooge 指出:每一帧的结束处是一个标志字节,而下一帧的开始处又是另一个标志字节,这种做法非常浪费空间。用一个标志字节也可以完成同样的任务,这样就可以节省一个字节。你同意这种观点吗?

5. 位串 011110111110111110 需要在数据链路层上被发送,请问,经过位填充之后实际被发送出去的是什么?

6. 假设使用了位填充成帧方法,请问,因为丢失一位、插入一位,或者篡改一位而引起的错误是否有可能通过校验和检测出来? 如果不能的话,请问为什么不能? 如果能够检测出来的话,请问校验和长度在这里是如何起作用的?

7. 你能够想象得出在什么样的环境下,一个开环协议(比如海明码)有可能更加适合于本章通篇所讨论的反馈类型的协议?

8. 为了提供比单个奇偶位的检错能力更强的可靠性,一种检错编码方案如下:用一个奇偶位来检查所有奇数序号的位,用另一个奇偶位来检查所有偶数序号的位。请问这种编码方案的海明距离是多少?

9. 假设使用海明码来传输 16 位的报文。请问,需要多少个检查位才能确保接收方可以检测并纠正单个位错误? 对于报文 1101001100110101, 请给出所传输的位模式。假设在海明码中使用了偶数位。

10. 假设用偶数位的海明码对一个 8 位字节进行编码,该字节编码前为 10101111, 请问编码之后的二进制值是什么?

11. 接收方收到了一个 12 位的海明码,其 16 进制值为 0xE4F。请问原来的值是多少(用 16 进制表示)? 假设至多只有 1 位发生了错误。

12. 检测错误的一种方法是按  $n$  行、每行  $k$  位来传输数据,并且在每行和每列加上奇偶位。其中右下角是一个检查它所在行和所在列的奇偶位。这种方案能够检测出所有的单个错吗? 2 位错误呢? 3 位错误呢?

13. 对于一个  $n$  行和  $k$  列的数据位块,现在使用水平和垂直的奇偶位来检测错误。假设由于传输错误,其中有 4 位变反了。请推导出该错误未能被检测出来的概率表达式。
14.  $x^7 + x^3 + 1$  被生成器多项式  $x^3 + 1$  除,所得的余数是什么?
15. 利用本章中介绍的标准 CRC 方法来传输位流 10011101。生成器多项式为  $x^3 + 1$ 。请给出实际被传输的位串。假设在传输过程中左边第三位变反了。请证明,这个错误可以在接收端被检测出来。
16. 数据链路协议几乎总是将 CRC 放在尾部,而不是头部,请问这是为什么?
17. 一个信道的位速率为 4kbps,传输延迟为 20ms。请问帧的大小在什么范围内,停-等协议才可以获得至少 50%的效率?
18. 一条 3000 公里长的 T1 骨干线路被用来传输 64 字节的帧,两端使用了协议 5。如果传输速度为  $6\mu\text{s}/\text{公里}$ ,则序列号应该有多少位?
19. 在协议 3 中,当发送方的定时器正在运行的时候,它还有可能启动定时器吗?如果可能的话,请问这种情况是如何发生的?如果不可能的话,请问为什么这是不可能的。
20. 想象这样一个滑动窗口协议,它的序列号有非常多的位,所以序列号几乎永远不会回转。请问 4 个窗口边界和窗口大小之间必须满足什么样的关系?这里窗口的大小是固定不变的,并且发送方和接收方的窗口大小相同。
21. 如果协议 5 中的 between 过程检查的条件是  $a \leq b \leq c$ ,而不是  $a \leq b < c$ ,则对于协议的正确性和效率有影响吗?请解释你的答案。
22. 在协议 6 中,当一个数据帧到达的时候,需要执行一个检查,看它的序列号是否与期望的序列号不同,并且 no\_nak 为真。如果这两个条件都成立,则发送一个 NAK。否则的话,启动辅助定时器。假定 else 子句被省略掉。这种改变会影响协议的正确性吗?
23. 假设在协议 6 中接近尾部的内含三条语句的 while 循环被去掉的话,这样会影响协议的正确性吗?还是仅仅影响协议的性能?请解释你的答案。
24. 假设从协议 6 的 switch 语句中去掉检查校验和错误的那个 case 子句。请问这种变化将如何影响协议的操作?
25. 在协议 6 中,针对 frame\_arrival 的代码中有一部分被用于 NAK。如果收到的帧是一个 NAK,并且另一个条件也满足的话,则这部分代码会被调用到。请给出一个场景,在此场景下这另一个条件是非常关键的。
26. 想象你正在编写一个数据链路层软件,它被用在一条专门给你发送数据的线路上,而不是让你往外发送数据。另一端使用了 HDLC,3 位序列号和一个可容纳 7 帧的窗口。你希望将乱序的帧尽可能多地缓存起来,以提高效率,但是你不允许修改发送方的软件。是否有可能让接收方的窗口大于 1,并且仍然保证该协议不会失败呢?如果可能的话,能够安全地使用的最大窗口是多少?
27. 考虑在一条 1Mbps 的无错误的线路上使用协议 6 的操作。最大的帧长度为 1000 位。每过 1 秒钟产生新的分组。超时间隔为 10ms。如果特殊的确认定时器被去掉的话,则不必要的超时就会发生。平均报文要被传输多少次?
28. 在协议 6 中,  $\text{MAX\_SEQ} = 2^n - 1$ 。这个条件显然是希望尽可能地利用头部的位,

但是,我们无法证明这个条件确实很关键。例如,协议在  $\text{MAX\_SEQ}=4$  的时候也能够正确地工作吗?

29. 利用地球同步卫星在一个 1Mbps 的信道上发送 1000 位的帧,该信道离开地球的传输延迟为 270ms。确认信息总是被捎带在数据帧上。头部非常短,并且使用 3 位序列号。在下面的协议中,最大可获得的信道利用率是多少?

- (a) 停-等协议。
- (b) 协议 5。
- (c) 协议 6。

30. 在一个负载很重的 50kbps 的卫星信道上使用协议 6,数据帧包含 40 位的头和 3960 位的数据,请计算一下浪费在头部和重传的开销占多少比例。假设从地球到卫星的信号传输时间为 270ms。ACK 帧永远不会发生。NAK 帧为 40 位。数据帧的错误率为 1%,NAK 帧的错误率忽略不计。序列号为 8 位。

31. 考虑在一个无错误的 64kbps 卫星信道上单向发送 512 字节的数据帧,有一些非常短的确从另一个方向回来。对于窗口大小为 1、7、15 和 27 的情形,最大的吞吐量分别是多少?从地球到卫星的传输时间为 270ms。

32. 一条 100 公里长的电缆运行在 T1 数据速率上。电缆的传输速度是真空中光速的 2/3。请问电缆中可以容纳多少位?

33. 假设我们用有限状态机模型来模拟协议 4。每个机器需要多少种状态?通信信道上需要多少种状态?整个系统(两台机器和一条信道)需要多少种状态?忽略校验和错误。

34. 对应于图 3.21 中的状态序列 (000)、(01A)、(01-)、(010)、(01A),请给出图 3.23 中的 Petri 网的激发序列。并说明该序列代表了什么。

35. 给定转换规则  $AC \rightarrow B$ 、 $B \rightarrow AC$ 、 $CD \rightarrow E$  和  $E \rightarrow CD$ ,请画出所描述的 Petri 网。利用该 Petri 网,画出从初始状态 ACD 开始的有限状态可达图。这些转换规则说明了哪一个著名的概念?

36. PPP 基本上是以 HDLC 为基础的,HDLC 则使用了位填充技术来防止在有效载荷数据中偶尔出现标志字节,以避免引起混淆。请给出至少一个理由说明为什么 PPP 却使用了字节填充技术?

37. 用 PPP 来发送 IP 分组的最小开销是多少?只计算由于 PPP 本身而引入的开销,不计算 IP 头部的开销。

38. 本实验练习的目标是实现一种错误检测机制,它使用了本章中介绍的标准 CRC 算法。请编写两个程序:生成器 generator 和验证器 verifier。生成器程序从标准输入读入一个  $n$  位的报文,该报文是由 0 和 1 组成的 ASCII 文本字符串,它是第一行。第二行是  $k$  位的多项式,也是 ASCII 文本。该程序输出一行 ASCII 文本(到标准输出),其中包含  $n+k$  个 0 和 1,代表了要被传输的报文。然后,它也输出多项式,就好像它读入进来时一样。验证器程序将生成器程序的输出读入进来,并输出一条报文,说明它是正确的还是错误的。最后,再编写一个程序 alter,根据它的参数(位的顺序号,计算的时候从左向右进行,并且从 1 开始计数)变反第一行中由参数指定的位,两行字符串中其他的内容完全

一致地复制过来。通过输入：

```
generator < file | verifier
```

你应该看到该报文是正确的，但是，当你输入：

```
generator < file | alter arg | verifier
```

你应该得到错误的提示信息。

39. 编写一个程序来模拟一个 Petri 网的行为。该程序应该读入一组转换规则，以及一组状态列表，这些状态对应于网络链路层发出一个新的分组或者接受一个新的分组。初始状态也是要读入的。该程序应该从初始状态开始，随机地选取那些激活的转换，并激发这些转换，检查一下看是否有“一台主机接受了 2 个分组而另一台主机并没有在此期间发出新的分组”这样的情形。

---

## 第4章 介质访问控制子层

---

正如我们在第1章中指出的那样,网络可以分成两大类:使用点到点连接的网络,以及使用广播信道的网络。本章将讨论广播网络和相应的协议。

在任何一个广播式网络中,关键的问题是:当存在多方要竞争使用信道的时候,如何确定谁可以使用信道。为了使这个问题更加清晰,请考虑电话会议的场景:6个人在6部不同的电话机上,这些电话都有连接,所以每个人都可以听到其他的人,也可以对其他人讲话。很可能发生这样的情况:当一个人停止说话的时候,马上有两个或者更多个人开始说话,从而导致混乱。在面对面的会议上,这样的混乱可以通过外部途径来解决,比如,在会上,与会者通过举手的方式请求获得发言权。当只有一条信道可供使用的时候,确定下一个使用者是非常困难的。现在已经有了一些协议专门来解决这个问题,这正是本章的内容。在有些文献中,广播信道有时候也称为多路访问信道(**multiaccess channel**)或者随机访问信道(**random access channel**)。

用于在多路访问信道上确定下一个使用者的协议属于数据链路层的一个子层,称为**MAC(Medium Access Control, 介质访问控制)**子层。在LAN中,MAC子层显得尤为重要,因为许多LAN使用多路访问信道作为它的通信基础。相反,WAN则使用点到点的链路,当然,卫星网络除外。因为多路访问信道和LAN如此紧密相关,所以,在本章中我们将从总体上讨论LAN,其中也包括一些从严格意义上讲不属于MAC子层的内容。

从技术角度而言,MAC子层是数据链路层的底下部分,所以,从逻辑上讲,我们在第3章讨论所有的点到点协议之前应该已经学习过MAC子层了。然而,对于大多数人而言,在很好地理解了两方协议之后,再来理解涉及多方的协议要容易得多。鉴于这样的原因,我们在本书中稍微偏离了自底向上的严格顺序。

### 4.1 信道分配问题

本章的中心主题是如何在多个竞争的用户之间分配单个广播信道。我们首先从一般意义上看一看静态和动态的方案,然后再讨论一些特殊的算法。

#### 4.1.1 LAN和MAN中的静态信道分配方案

在多个竞争用户之间分配单个信道(比如电话干线)的传统做法是频分多路复用(FDM)。如果总共有 $N$ 个用户,则整个带宽分成 $N$ 等份(参见图2.31),每个用户分配一份。由于每个用户都有自己私有的频段,所以用户之间不会有干扰。只有当用户数量比较少而且固定不变,并且每个用户都有繁重的流量负担(比如电话运行商的交换局)的时

候, FDM 才是一种简单有效的分配机制。

然而, 当发送方的数量非常多且经常不断地变化, 或者流量是突发性的, 则 FDM 就会出现一些问题。如果整个频谱被分成  $N$  份, 并且当前只有很少的用户 (比  $N$  少得多) 需要进行通信, 则大量宝贵的频谱将被浪费掉。如果希望进行通信的用户数超过了  $N$  个, 则有些用户将被拒绝, 因为缺少带宽资源; 即使有些已经被分配了频段的用户并不发送或者接收数据, 他们也无法将自己的频段转给其他的用户。

然而, 即使我们假设用户数量能够维持在  $N$  个固定不变, 将单个信道划分成多个静态子信道的做法本质上也是非常低效的。基本的问题在于, 当有些用户停止通信的时候, 他们的带宽实际上就被白白丢掉了。当他们自己不使用这些带宽的时候, 其他的用户也不允许使用。而且, 在大多数计算机系统中, 数据流量往往是突发性的 (通常, 峰值流量与平均流量之比为 1000 : 1)。因此, 大多数信道在大多数时候是空闲的。

静态 FDM 如此之差的性能也可以通过一个简单的排队理论计算看得出来。我们首先假设一个信道的容量为  $C$  bps, 平均延迟时间为  $T$ , 帧到达率为每秒  $\lambda$  帧, 每一帧的长度符合一个指数概率密度函数, 其均值为每帧  $1/\mu$  位。利用这些参数, 帧到达率为每秒  $\lambda$  帧, 信道的服务率为每秒  $\mu C$  帧。根据排队理论, 可以证明, 对于泊松到达和服务时间:

$$T = \frac{1}{\mu C - \lambda}$$

例如, 如果  $C$  为 100Mbps, 平均帧长度  $1/\mu$  为 10 000 位, 帧到达率  $\lambda$  为 5000 帧/秒, 则  $T=200\mu\text{s}$ 。请注意, 如果我们忽略排队延迟, 只是问“在一个 100Mbps 的网络上发送一个 10 000 位的帧需要多长时间”的话, 则我们得到的答案将是  $100\mu\text{s}$  (这是不正确的)。只有当信道上没有竞争的时候, 此结果才成立。

现在我们将单个信道分成  $N$  个独立的子信道, 每个子信道的容量为  $C/N$  bps。现在, 每个子信道的平均输入率变成  $\lambda/N$ 。重新计算  $T$ , 我们可以得到:

$$T_{\text{FDM}} = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = \frac{N}{\mu C - \lambda} = NT \quad (4-1)$$

因此可以得出, 使用 FDM 的平均延迟将是  $T$  的  $N$  倍, 也就是将所有的帧都组织到一个大的中心队列这种情况下的延迟的  $N$  倍。

针对 FDM 的结论同样也适用于时分多路复用 (TDM) 的情形。每个用户被静态地分配到  $N$  分之一的时槽。如果一个用户并不使用分配给他的时槽, 则该时槽就会空闲下来。如果我们从物理上将网络分割开, 则同样的问题也存在。再利用前面的例子, 如果我们用 10 个 10Mbps 的网络来代替一个 100Mbps 的网络, 并且每个用户分配一个网络, 则平均延迟将从  $200\mu\text{s}$  跳跃到  $2\text{ms}$ 。

既然所有传统的静态信道分配方法都不能适应突发性流量, 我们现在就来研究一下动态的方法。

#### 4.1.2 LAN 和 MAN 中的动态信道分配方案

在本章中我们将要讨论许多种信道分配方案, 在讨论第一种方案之前, 有必要仔细地描述一下信道分配问题。这个领域中所有已经做过的工作都是以下面的 5 个关键假设为



基础的:

(1) **站模型(Station Model)**。该模型是由  $N$  个独立的站(比如计算机、电话或者个人通信设备)组成的,每个站都有一个程序或者用户会产生供传输用的帧。这里的站有时候也称为**终端(terminal)**。在长度为  $\Delta t$  的间隔内,产生一帧的概率为  $\lambda \Delta t$ ,这里  $\lambda$  为常数(新帧的到达率)。一旦一帧已被生成,则该站被阻塞(什么也不做了),直到该帧被成功地发送出去为止。

(2) **单信道假设(Single Channel Assumption)**。对于所有的通信都只有一个信道可以使用。所有的站都可以在该信道上传输数据,也可以从该信道接收数据。从硬件的角度来看,所有的站都是平等的,但是,协议软件可能会给各个站分配不同的优先级。

(3) **冲突假设(Collision Assumption)**。如果两帧同时被传输,则它们在时间上就会有重叠,这样得到的信号是混乱的,这种情况称为**冲突(collision)**。所有的站都能够检测冲突事件。冲突的帧必须在以后再次被发送。除了因冲突而产生错误外,不会再有其他的错误。

(4a) **连续时间(Continuous Time)**。在任何时刻都可以开始传输帧,不需要通过一个主时钟将时间分成离散的间隔。

(4b) **分槽时间(Slotted Time)**。时间被分成离散的间隔(即时槽)。帧的传输总是从某一个时槽的起点开始发送。一个时槽可能包含 0、1 或者多个帧,分别对应于空闲的时槽、一次成功的发送,或者一次冲突。

(5a) **载波检测(Carrier Sense)**。一个站在使用信道之前,它可以辨别该信道当前是正在被使用。如果信道被检测出来是忙的,则没有哪个站会企图使用该信道,如此下去一直到信道空闲为止。

(5b) **无载波检测(No Carrier Sense)**。在使用信道之前,站无法检测信道。它们只是盲目地开始传输,以后再判断这次传输是否成功。

下面按顺序对这些假设进行讨论。第一个假设是说,站是独立的,并且以恒定的速率产生帧。它也隐含着假设每个站只有一个程序或者用户,所以,当一个站阻塞的时候,不会有新的帧被生成出来。更加复杂一点的模型允许每个站有多个程序,因此当一个站阻塞的时候,它还可以生成帧,但是,对这些站的分析则要复杂得多。

单信道假设是该模型的核心。除了单个信道外,没有外部的途径可以通信。这些站不可能举起手来请求老师准许发言。

冲突假设也是基本的,但是在有些系统(特别是扩频系统)中,这个假设是不严格的。而且,有些 LAN,比如令牌环网,在站与站之间传递一个特殊的令牌,拥有该令牌的站可以传输一帧。但是在接下来的几节中,我们将坚持单信道竞争和冲突模型。

关于时间有两种可能的假设。既可以是连续时间假设(4a),也可以是分槽时间假设(4b)。有些系统使用前者,而有些系统使用后者,所以我们对两者都要分析和讨论。对于一个给定的系统,它只能支持一种假设。

类似地,一个网络可能具有载波检测功能(5a),也可能没有载波检测功能(5b)。LAN 通常具有载波检测功能。然而,无线网络不可能有效地使用载波检测功能,因为并不是每一个站都在其他任一站的无线电波范围内。在有线的载波检测网络中,如果

个站发现它与另一个传输发生了冲突,则它可以提前终止它的传输过程。出于工程的原因,在无线网络上很少使用冲突检测。请注意,这里的词“载波(carrier)”是指电缆上的电信号,与“快马邮递(Pony Express)”时代的公共承运商(common carrier,比如电话公司)毫无关系。

## 4.2 多路访问协议

关于分配一个多路访问信道的算法已经有很多了。在本节中,我们将学习一些比较有意思的代表性协议,并且给出它们的用法实例。

### 4.2.1 ALOHA

20 世纪 70 年代,夏威夷大学的 Norman Abramson 和他的同事设计出了一种巧妙的新方法来解决信道的分配问题。自从那时开始,许多研究人员又进一步扩展了他们的工作(Abramson, 1985)。Abramson 的工作被称为 ALOHA 系统,尽管它使用了基于地面的无线电广播通信,但是它的基本思想同样也适用于其他的“多个无协调关系的用户竞争单个共享信道使用权”的系统。

我们在这里将要讨论两个版本的 ALOHA: 纯 ALOHA 和分槽 ALOHA。它们的区别在于是否将时间分成离散的时槽以便所有的帧都必须同步到时槽中。纯 ALOHA 不要求全局的时间同步,而分槽 ALOHA 则需要。

#### 纯 ALOHA

ALOHA 系统的基本思想非常简单: 当用户有数据要发送的时候就让它们传输。当然,这样做可能会有冲突,冲突的帧将被损坏。然而,由于广播的反馈特性,发送方只要通过监听信道,总是可以知道它的帧是否被毁坏,其他的用户也可以做到这一点。在一个 LAN 中,发送方立即就可以得到反馈;而在卫星信道上,需要延迟 270ms 之后发送方才知知道它的传输是否成功。如果由于某种原因无法在传输的时候进行监听,则确认就很有必要。如果送出去的帧被毁坏了,则发送方只要等待一段随机的时间,然后再次发送该帧。等待的时间必须是随机的,否则的话同样的帧会不停地冲突,因为重发的节奏完全一致。如果系统中多个用户共享同一个信道的方法会导致冲突,则这样的系统往往称为竞争(contention)系统。

图 4.1 给出了一个 ALOHA 系统中生成帧的框架结构。我们假设所有的帧都是同样长度的,因为对于 ALOHA 系统,采用统一长度的帧比允许可变长度的帧更能达到最大的吞吐量。

任何时候当两帧试图同时占用信道时,冲突就会发生,并且两帧都会被破坏。如果一个新帧的第一位与前一帧的最后一位正好发生了重叠,则这两帧都将被完全毁坏,稍后都要重传。校验和不可能(也不应该)区分出是完全丢失,还是局部差错。对于校验和技术而言,坏了就是坏了。

一个有趣的问题是: ALOHA 信道的效率怎么样? 换句话说,在这样混乱的情况下,

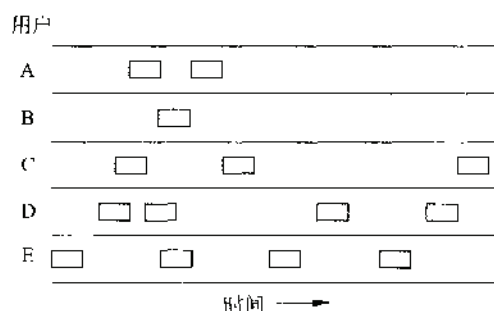


图 4.1 在纯 ALOHA 中, 帧的传输完全是在任意时间进行的

能够侥幸逃脱冲突而被传输出去的帧占多大比例呢? 我们首先考虑这样的情形: 有无穷多个交互用户坐在他们的计算机(站)前面。每个用户总是处于两种状态中的一种: 敲键或等待。刚开始时, 所有的用户都处于敲键的状态。当敲入一行之后, 用户停止敲键, 开始等待应答。然后计算机将这一行字符作为一帧传输出去, 并且检查信道看是否传输成功。如果成功, 则用户就会看到回复, 并且继续敲入字符。如果不成功, 则用户继续等待, 该帧被不停地重传, 直到被成功发送出去为止。

我们用“帧时(frame time)”来表示传输一个标准的、固定长度的帧所需要的时间(即, 帧的长度除以位速率)。现在我们假定无穷多个用户按照泊松分布来生成新的帧, 平均每个“帧时间”产生  $N$  帧。这里假设存在无穷多个用户是必要的, 因为这样可以确保  $N$  不会随着用户变成阻塞状态而下降。如果  $N > 1$ , 则这群用户生成帧的速度大于信道的处理速度, 因此, 几乎每一帧都要经受冲突。对于合理的吞吐量, 我们应该期望  $0 < N < 1$ 。

除了新生成的帧以外, 每个站也会由于有的帧遭受冲突而重传这些帧。我们进一步假设在每个“帧时间”中, 老帧和新帧合起来共有  $k$  次传输的概率也符合泊松分布, 其均值为  $G$ 。显然,  $G \geq N$ 。在载荷较低的情况下(即  $N \approx 0$ ), 冲突很少发生, 因此重传也很少, 于是  $G \approx N$ 。在载荷较高的时候, 将会有很多冲突, 所以  $G > N$ 。在所有这些载荷的情况下, 吞吐量  $S$  是载荷  $G$  乘以每一次传输成功的概率  $P_0$ , 也即  $S = GP_0$ 。其中  $P_0$  是一帧没有遭到冲突的概率。

如果从一帧被发送出去开始, 在一个“帧时间”内没有其他的帧被发送, 则这一帧不会遭到冲突, 如图 4.2 所示。在什么样的条件下, 图中的阴影帧将会毫无损坏地到达呢? 假设发送一帧所需要的时间为  $t$ , 如果其他的用户在  $t_0 \sim t_0 + t$  之间生成了一帧, 则该帧的结束部分将与阴影帧的开始部分发生冲突。实际上, 阴影帧的命运在它的第一位被送出去之前就已经注定了, 但是, 由于在纯 ALOHA 中, 一个站在发送帧之前并不会监听信道, 所以, 它无法知道其他的帧是否已经在信道上了。类似地, 在  $t_0 + t \sim t_0 + 2t$  之间开始发送的任何其他帧也会冲突到阴影帧的结束部分。

在给定的一个帧时间中共生成  $k$  帧的概率服从泊松分布:

$$\Pr[k] = \frac{G^k e^{-G}}{k!} \quad (4-2)$$

所以, 生成零帧的概率为  $e^{-G}$ 。在两个帧时间长的间隔之中, 所生成的帧的平均数是

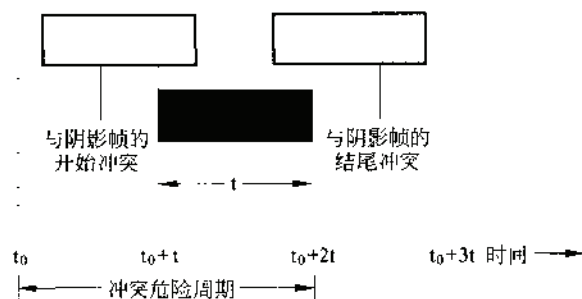


图 4.2 阴影帧的冲突危险周期

$2G$ 。因此,在整个冲突危险期中,不存在其他流量的概率是  $P_0 = e^{-2G}$ 。利用  $S = GP_0$ ,则可以得到:

$$S = Ge^{-2G}$$

所出现的帧流量与吞吐量之间的关系如图 4.3 所示。当  $G = 0.5$  的时候,吞吐量最大,为  $S = 1/2e$ ,大约等于 0.184。换句话说,我们最希望的信道利用率为 18%。这个结果并不理想,但是,对于这种任何人都可以随意发送的传输方式,要想达到 100% 的成功率几乎是不可能的。

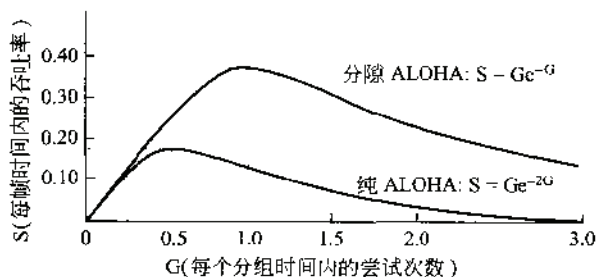


图 4.3 ALOHA 系统中吞吐量与帧流量之间的关系

### 分槽 ALOHA

1972 年,Roberts 发表了一种能将 ALOHA 系统的容量增加一倍的方法 (Roberts, 1972)。他的建议是将时间分成离散的间隔,每个间隔对应于一帧。这种方法要求用户遵守统一的时槽边界。一种同步时间的办法是,由一个专门的站在每个间隔起始的时候发出一个脉冲信号,就好像一个时钟一样。

Roberts 的方法也称为分槽 ALOHA (slotted ALOHA); 与纯 ALOHA 不同的是,在分槽 ALOHA 中,计算机并不是在敲入回车键的时候就可以立即发送帧了,相反,它必须要等到下一个时槽的开始时刻。因此,连续纯 ALOHA 变成了离散的 ALOHA。由于冲突危险周期现在被减小了一半,所以,对于我们的测试帧而言,在同一个时槽中没有其他流量的概率是  $e^{-G}$ ,于是可以得到:

$$S = G e^{-G} \quad (4-3)$$

正如你可以从图 4.3 中看到的那样,分槽 ALOHA 的尖峰在  $G=1$  处,此时吞吐量为  $S=1/e$ ,大约等于 0.368,是纯 ALOHA 的两倍。如果系统运行在  $G=1$  处,则空时槽的概率为 0.368(从等式 4.2 可以得出)。使用分槽 ALOHA,我们期望的最好结果是:37% 为空时槽,37% 为成功,剩下 26% 为冲突。如果在更高的  $G$  值上运行,则空时槽数会降低,但是冲突数会呈指数增长。为了看出冲突数是如何随着  $G$  的变化而快速增长的,请考虑一个测试帧的传输过程。该测试帧能够避免一次冲突的概率是  $e^{-G}$ ,即所有其他用户也在该时槽中的概率。于是,冲突的概率为  $1 - e^{-G}$ 。要求  $k$  次尝试才能成功传输的概率(即  $k-1$  次冲突之后才有一次成功的概率)为:

$$P_k = e^{-G} (1 - e^{-G})^{k-1}$$

于是,每帧传输次数期望为:

$$E = \sum_{k=1}^{\infty} k P_k = \sum_{k=1}^{\infty} k e^{-G} (1 - e^{-G})^{k-1} = e^G$$

所以,结果是, $E$  随  $G$  呈指数增长,信道载荷微小的增长也会极大地降低信道的性能。

分槽 ALOHA 是非常重要的,但是,在早期的时候,它的重要性并没有明显地表现出来。它是在 20 世纪 70 年代设计出来,并用于一些实验系统中,之后差不多就被遗忘了。当“通过有线电视的电缆来访问 Internet”的技术被发明时,立即就出现了一个问题,即如何在多个竞争的用户之间分配一条共享的信道,于是分槽 ALOHA 又被从遗忘的角落中找出来,来解决当前面临的问题。这样的情况常常会发生:一些非常完善有效的协议由于政策上的原因而被弃置不用(比如某个大公司希望每个人都按照它的方式来行事),但是多年以后,有些聪明的人就会发现,某一个长期被弃置不用的协议恰好可以解决他们当前的问题。出于这样的原因,本章中我们将学习一些非常优秀的协议,尽管它们目前并没有被广泛地使用,但是只要有足够的网络设计师了解它们,则在将来的应用中它们就有可能被用到。当然,我们也会学习许多当前正在使用之中的协议。

#### 4.2.2 载波检测多路访问协议

利用分槽 ALOHA,可以达到的最佳信道利用率是  $1/e$ 。这并不令人惊奇,由于每个站都可以随意地发送数据,它并不关心其他的站是否也在发送数据,所以,频繁地发生冲突是难免的。然而,在局域网中,每个站可以检测其他的站当前正在做什么,然后再根据情况调整自己的行为,这在局域网中是有可能的。这些网络可以获得比  $1/e$  好得多的利用率。在这一小节中,我们将讨论一些提高性能的协议。

如果在一个协议中,每个站都监听是否存在载波(即是否有传输),并采取相应的动作,则这样的协议称为**载波检测协议(carrier sense protocol)**。现在已经有了许多这种类型的协议。Kleinrock 和 Tobagi(1975)已经详细地分析了几个这样的协议,下面我们将介绍几个载波检测协议。

##### 持续的和非持续的 CSMA

我们将要学习的第一个载波检测协议称为**1-持续 CSMA(Carrier Sense Multiple Access,载波检测多路访问)**。当一个站有数据要发送的时候,它首先监听信道,看当时是



否有其他的站正在传输数据。如果信道忙的话,该站会等待直至信道空闲。当该站检测到信道空闲的时候,它就发送一帧数据。如果有冲突发生的话,该站等待一段随机的时间,然后再次检测和发送。这样的协议称为 1-持续的,因为当一个站发现信道空闲的时候,它传输数据成功的概率为 1。

传播延迟对于协议的性能有重要的影响。当一个站刚刚开始发送数据之后,另一个站也做好发送数据的准备并且开始检测信道,这样的几率虽然小,但也是有的。如果第一个站的信号还没有到达第二个站,则第二个站将会检测到信道是空闲的,于是也将开始发送数据,从而导致冲突。传播延迟越长,则这种影响也会变得越发重要,同时,协议的性能也越差。

即使传播延迟为 0,仍然可能会发生冲突。如果在一个站的传输过程中有两个站都准备好数据了,则这两个站都会等待第一个站的传输结束,然后,它们恰好同时开始传输数据,因而也会导致冲突。如果这两个站不是那么着急的话,则冲突会少得多。即使如此,此协议也比纯的 ALOHA 要好得多,因为这两个站都非常礼貌,不会去打扰第三个站的帧。直观来看,这种做法将比纯 ALOHA 有更好的性能;同样地,也会高于分槽 ALOHA 的性能。

第二个载波检测协议是非持续的 CSMA(nonpersistent CSMA)。在这个协议中,每个站在企图传送数据之前要理智得多,不像前一个协议那样贪婪。一个站在发送数据之前,先要检测信道。如果没有人在发送数据,则该站自己开始发送数据。然而,如果信道当前正在被使用之中,则该站并不持续地对信道进行监听,以便一旦前一次传输结束它好立即抓住机会发送数据。相反,它会等待一段随机的时间,然后再重复同样的算法。因此,此算法将会导致更好的信道利用率,但是比起 1-持续 CSMA 来,也导致了更长的延迟。

最后一个协议是 p-持续 CSMA(p-persistent CSMA)。它应用于分槽的信道,其工作方式如下所述。当一个站准备好要发送数据的时候,它会检测信道。如果信道是空闲的,则它按照概率  $p$  的可能性发送数据。在概率  $q=1-p$  的情况下,它会将传送数据的任务延迟到下一个时槽。如果下一个时槽也是空闲的,则它或者传送数据,或者再次延迟,其概率分别为  $p$  和  $q$ 。这个过程会一直继续,直到该帧被发送出去,或者另一个站也开始传送数据。在后者的情况下,该站的处理方式如同发生了冲突一样,即等待一段随机的时间,然后再重新开始。如果该站刚开始的时候就检测到信道忙的话,它会等待到下一个时槽,然后再应用上面的算法。图 4.4 显示了这三个协议,以及纯 ALOHA 和分槽 ALOHA 的吞吐量和流量之间的关系。

#### 带冲突检测的 CSMA

持续的和非持续的 CSMA 协议是对 ALOHA 的改进,因为这些协议都保证了当检测到信道忙时,所有的站都不再传送数据。另一个改进是,对于每一个站而言,一旦它检测到有冲突,它就放弃它当前的传送任务。换句话说,如果两个站都检测到信道是空闲的,并且同时开始传送数据,则它们几乎立刻就会检测到有冲突发生。它们不应该再继续传送它们的帧,因为这样只会产生垃圾而已;相反,一旦检测到冲突之后,它们应该立即停止传送数据。快速地终止被损坏的帧可以节省时间和带宽。该协议称为 CSMA/CD



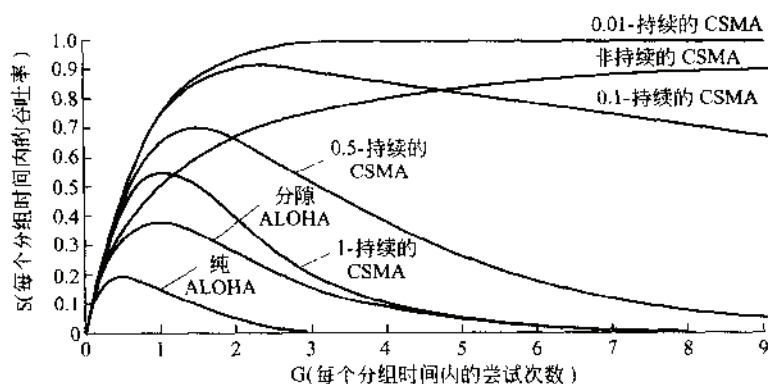


图 4.4 各种随机访问协议的信道利用率与载荷之间的比较

(CSMA with Collision Detection, 带冲突检测的 CSMA), 广泛应用于 LAN 中的 MAC 子层。特别是, 它是当前最为流行的以太网 LAN 的基础, 所以, 我们有必要专门花一点时间来详细地介绍它。

如同许多其他的 LAN 协议一样, CSMA/CD 也使用了图 4.5 所示的概念模型。在  $t_0$  点上, 一个站已经完成了帧的传送, 其他需要发送帧的站现在可以试图发送了。如果有两个或者多个站同时进行传送的话, 则冲突就会发生。通过对接收到的信号功率或者脉冲宽度进行检查, 并将它与原始发送的信号进行比较, 就可以检测到是否有冲突发生。

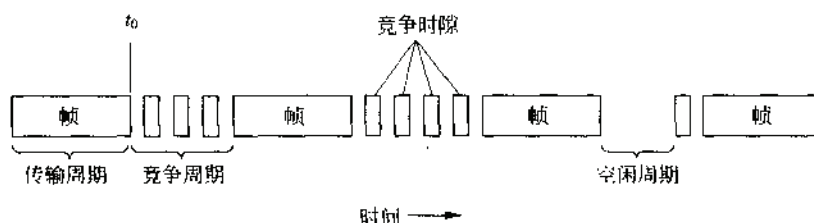


图 4.5 CSMA/CD 可能处于三种状态之一：竞争、传输和空闲

当一个站检测到冲突之后, 它立即放弃它的传送任务, 并等待一段随机的时间, 然后再重新尝试传送, 假定此时没有其他的站也开始传送。因此, 我们的 CSMA/CD 模型将由三部分组成: 交替出现的竞争和传输周期, 以及当所有的站都静止 (比如没有传输任务) 时候的空闲周期。

现在我们来查看竞争算法的细节。假定两个站同时在  $t_0$  时刻开始传送数据。它们需要多长时间才能意识到已经发生了冲突呢? 该问题的答案对于确定竞争周期的长度非常关键, 进而也会影响到延迟和吞吐量。不难理解, 检测冲突的最小时间是将信号从一个站传送到另一个站所需要的时间。

基于这样的推理, 你可能会认为, 如果一个站在开始传送数据之后的“一段完整的电缆传播时间”之内没有监听到冲突的话, 则它就可以确定自己已经“抓住”电缆了。这里“抓住”的意思是指: 所有其他的站都知道它正在传送数据, 所以就不会干扰它了。这个

结论是不正确的。请考虑下面最坏的情形。假设两个相距最远的站之间传播信号所需要的时间为  $\tau$ 。在  $t_0$  时刻,一个站开始传送数据。在  $\tau - \epsilon$  时刻,也就是在信号到达最远的站之前的那一刻,该站也开始传送数据了。当然,它几乎立刻就检测到了冲突,便停止传输了。由这次冲突引起的微小的噪声尖峰要到  $t_0 + \tau - \epsilon$  时刻才能回到原来那个站。换句话说,在最差的情况下,只有当一个站传输了  $t_0 + 2\tau - \epsilon$  之后还没有监听到冲突,这才可以确保它已经抓住了信道。由于这样的原因,我们将竞争间隔建模成一个分槽 ALOHA 系统,其时槽宽度为  $2\tau$ 。在一条 1 公里长的同轴电缆中,  $\tau \approx 5\mu\text{s}$ 。为了简化起见,我们将假定每个时槽只包含 1 位。一旦信道已经被一个站抓住了,则该站就可以在任何它期望的速率上传输数据了,当然不局限于每  $2\tau$  秒 1 位这样的速率。

很重要的一点是,冲突检测是一个模拟(analog)的过程。当一个站传输数据的时候,它的硬件必须监听电缆,如果它读回来的信息与发送出去的信息不一致的话,它就知道已经发生冲突了。这意味着,信号编码方案必须允许检测冲突(例如,两个 0 伏的信号即使冲突了也可能检测不到)。由于这个原因,通常需要使用特殊的编码方案。

同时值得注意的是,一个正在发送数据的站必须不停地监视信道,监听那些有可能代表冲突的噪声尖峰。出于这样的原因,单信道的 CSMA/CD 本质上是一个半双工系统。一个站要同时发送和接收数据是不可能的,因为在每次传送过程中,接收逻辑被用于监听冲突了。

为了避免误解,值得说明的一点是,MAC 子层协议不保证可靠递交。即使在没有发生冲突的情况下,接收方也可能会由于种种原因而没有正确地复制到帧数据(比如因为缓冲区空间不够,或者中断被漏掉了)。

### 4.2.3 无冲突的协议

一旦一个站已经确定无疑地抓住了信道,则冲突就不会发生了,尽管如此,在竞争周期过程中冲突仍有可能发生。这些冲突严重地影响了系统的性能,特别是当电缆很长(即  $\tau$  很大)而帧的长度又很短的时候。CSMA/CD 并不是普遍可适用的。在本小节中,我们将介绍另外一些协议,它们解决了信道竞争问题,冲突根本不会发生,即使在竞争周期过程中也不会发生冲突。大多数这样的协议并没有被用在当前主流的系统;但是,在一个快速变化的领域中,有一些具有优异特性的协议可被用于将来的系统,这通常是一件好事。

在接下来将要描述的协议中,我们都假定共有  $N$  个站接入到系统中,每个站都有惟一的地址,从 0 到  $N-1$ 。有些站在部分时间中可能是不活动的,这无关紧要。我们也假定传播延迟是可以忽略的。但是,基本的问题仍然存在:在一次成功的传输之后哪个站将会获得信道?我们继续使用图 4.5 中含有离散竞争时槽的模型。

#### 位图协议

我们第一个要介绍的无冲突协议采用了基本的位图方法(basic bit-map method),在该协议中,每个竞争周期正好包含  $N$  个时槽。如果 0 号站有一帧数据要发送,则它在第 0 个时槽中传送一个“1”位。在这个时槽中,其他的站不允许发送数据。不管 0 号站怎么

做,1号站有机会在1号时槽中传送一个“1”,但是只有当它有一帧在排队等待的时候才这样做。一般地,第j号站通过在j号时槽中插入一个“1”来声明自己有一帧要发送。当所有N个时槽都通过之后,每个站都知道了哪些站希望传送数据。这时候,它们便按照数字顺序开始传送数据了,如图4.6所示。

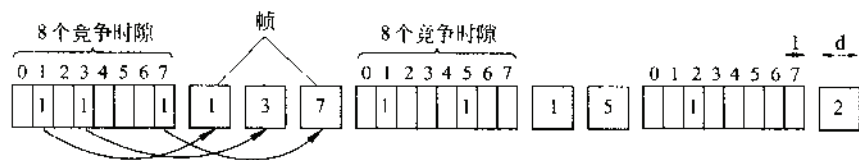


图 4.6 基本的位图协议

由于大家都对将要发生的事情非常清楚,并且遵守统一的规定,所以永远也不会发生冲突。当最后一个排队的站传送完成了它的帧之后,每个站都可以很容易地检测到这个事件,于是,另一个N位的竞争周期又开始了。如果当一个站的位时槽刚刚过去的时候,它才作好传送的准备,那么,它就非常不幸了,只有保持沉默,直到每个站都有机会发送出数据,然后新的位图再次到来。像这样的协议,在实际传送数据之前先广播自己有数据要发送的愿望,称为**预留协议(reservation protocol)**。

现在我们简要地分析一下这个协议的性能。为了简便起见,我们将用竞争位时槽作为单位来计量时间,假定数据帧是由d个时间单位构成的。在载荷很低的情况下,数据帧会非常少,主要是不断重复出现的位图。

我们从序号较低的站,比如0号站或者1号站的角度来考虑总体的情形。典型情况下,当它已经作好发送数据的准备时,“当前”的时槽将位于位图中间的某个地方。平均而言,该站必须等待 $N/2$ 个时槽以完成当前的扫描,再等待另外N个时槽以完成下一次扫描,然后才可以开始传输数据。

从高序号的站来看则情形会好很多。一般地,它只需等待半个扫描周期( $N/2$ 个位时槽)就可以开始传输数据了。高序号的站往往不必等待到下一次扫描。由于低序号的站必须等待平均 $1.5N$ 个时槽,而高序号的站必须等待平均 $0.5N$ 个时槽,对于所有的站而言,平均数是N个时槽。在低载荷情况下的信道效率很容易计算。每一帧的额外开销是N位,数据长度为d位,于是信道利用率为 $d/(N+d)$ 。

在高载荷的情况下,若所有的站在任何时候都有数据要发送,则N位竞争周期被分摊到N帧上,因此,每一帧的额外开销只有1位,所以,信道利用率为 $d/(d+1)$ 。一帧的平均延迟等于它在站内的排队时间,加上它到达队列头部之后另外的 $N(d+1)/2$ 时间。

### 二进制倒计数协议

基本位图协议的一个问题是,每个站的开销是1位,所以它不可能很好地扩展到包含上千个站的网络中。通过使用二进制的站地址,我们可以做得更好。如果一个站想要使用信道,则它以二进制位串的形式广播它的地址,并且从高序的位开始。假定所有的地址都有同样的长度。来自不同站的每个地址中的位被布尔或(OR)在一起。我们将这样的

协议称为**二进制倒计数(binary countdown)**协议。它被用于 Datakit 中(Fraser, 1987)。它隐式地假定传输延迟是可以忽略的,所以,所有的站都同时看到地址宣告位。

为了避免冲突,必须使用一条仲裁规则:对于一个站而言,如果它看到,在它的地址位中,一个值为 0 的高序位被改写成 1 了,则它就放弃。例如,如果站 0010、0100、1001 和 1010 都试图要获得信道,在第一个位时间中,这些站分别传送 0、0、1 和 1。它们被 OR 在一起,得到 1。站 0010 和 0100 看到了 1,它们就知道有高序的站也在竞争信道,所以它们放弃这一轮的竞争。而站 1001 和 1010 则继续。

接下来的位为 0,于是两者继续竞争。再接下来的位为 1,所以站 1001 放弃。最后的胜者是 1010,因为它有最高的地址。在赢得了竞争之后,它现在可以传输一帧,之后又开始新一轮竞争。图 4.7 演示了该协议。该协议有这样一种特性:高序站的优先级比低序站的优先级高,这可能是好事,也可能是坏事,取决于特定的环境。

这种方法的信道利用率为  $d/(d + \log_2 N)$ 。然而,如果精心地选择帧格式,使得发送方的地址正好是帧内的第一个域,那么,即使这样,  $\log_2 N$  位也不会浪费,所以,信道利用率为 100%。

Mok 和 Ward(1979)描述了二进制倒计数方法的一个变种,它使用了一个并行接口而不是串行接口。他们还建议使用虚拟的站序号,从 0 开始,一直到包含所有的站;每次传输之后,成功的站会被改变为最小的编号,其他的站也相应地循环改变编号,这样做的目的是为了给那些长时间沉默的站更高的优先级。例如,如果站 C、H、D、A、G、B、E、F 的优先级分别为 7、6、5、4、3、2、1,则 D 成功传输之后,它就被放到了列表的尾部,于是按优先级的顺序为 C、H、A、G、B、E、F、D。因此,C 的虚拟站号仍为 7,但是 A 则从 4 移到了 5,而 D 从 5 降到了 0。这时,惟有其他站都不参与竞争,D 才能够继续获得信道。

二进制倒计数协议是一个简单的、精致的、高效的协议,它有待于被重新启用。希望有一天它又会找到一个新的家。

#### 4.2.4 有限竞争协议

对于在一个电缆网络中如何获取信道的问题,我们已经考虑了两种基本的策略:一种是竞争的方法,就如同在 CSMA 中的做法那样;另一种是无竞争的方法。每一种都可以用下面的两个重要性能指标来衡量:在低载荷情况下的延迟,以及在高载荷情况下的信道利用率。在载荷较轻的条件下,竞争的方法(即纯或者分槽 ALOHA)更为理想,因为它的延迟很短。随着载荷的增加,竞争方法变得越来越没有优势,因为信道仲裁所需要的开销变得越来越大。对于无冲突的协议,则结论刚好相反。在低载荷情况下,它们有很长的延迟,但是随着载荷的增加,信道的效率反而提高,而不像竞争协议那样变得更差。

显然,如果我们能够把竞争协议和无冲突协议的优势结合起来,那是最好了。这样得

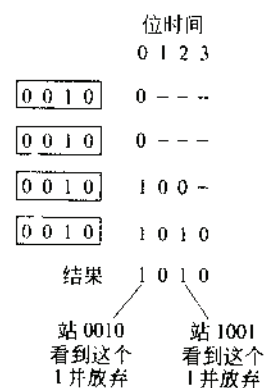


图 4.7 二进制倒计数协议:  
虚线表示不再参与竞争

到的新协议在低载荷的情况下使用竞争的做法,因而延迟较短,但是在载荷较高的情况下使用无冲突的技术,从而获得很好的信道效率。我们把这样的协议称为有限竞争协议(limited-contention protocol),实际上,这样的协议确实存在,我们正好用它来结束关于载波检测网络的学习。

到现在为止我们学习过的竞争协议都是对称的,也就是说,每个站企图获得信道的概率值为  $p$ ,并且所有的站都使用同样的  $p$  值。很有意思的是,如果在协议中,为不同的站分配不同的概率值,有时候系统的整体性能会有所提高。

在讨论非对称的协议之前,我们来快速地计算一下对称协议的性能情况。假设共有  $k$  个站在竞争信道的使用权。每个站在每个时槽中的传送概率为  $p$ 。那么,在一个给定的时槽中,某一个站能够成功地获得信道的概率为  $kp(1-p)^{k-1}$ 。为了找到  $p$  的最优值,我们按照  $p$  对它求微分,再将结果设置为 0,解出  $p$  值。通过这样做之后,我们发现,  $p$  的最佳值为  $1/k$ 。将  $p=1/k$  代入,则得到:

$$\text{Pr}[p \text{ 为最优值时的成功率}] = \left[ \frac{k-1}{k} \right]^{k-1} \quad (4-4)$$

这个概率的曲线如图 4.8 所示。对于站数量较少的情形,成功的几率很高,但是,一旦站的数量达到 5 个以后,概率值很快下降,接近于  $1/e$  值。

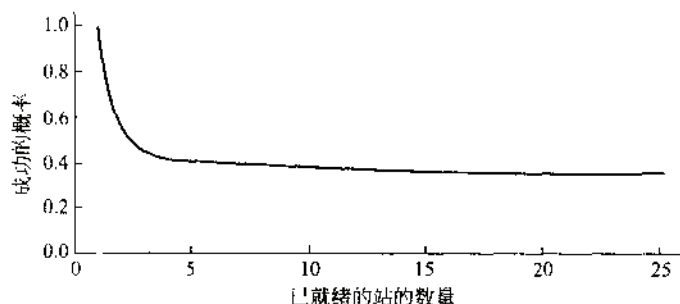


图 4.8 对称竞争信道的成功概率

从图 4.8 可以看出的很明显的一点是,只要减少参与竞争的站的数量,则某一个站获得信道的概率就会增加。有限制的竞争协议正是这样做的。它们首先将所有的站划分成组,这些组不必是两两不相交的。只有 0 号组的成员才允许竞争 0 号时槽。如果有一个成员竞争成功了,则它获得信道,便可以传送它的帧。如果该时槽是空闲的,或者发生了冲突,则 1 号组的成员竞争 1 号时槽,以此类推。通过适当的分组办法,每个时槽中的竞争数量可以大大减少,从而使得每个时槽中的行为尽可能地靠近图 4.8 的左侧。

协议的诀窍在于如何将站分配到各个时槽中。在讨论一般的情形以前,我们先来考虑一些特殊的情形。在一种极端的情况下,每个组只包含一个站。这样的分配方案可以保证永远不会发生冲突,因为对于任何给定的时槽,至多只有一个站参与竞争。前面我们已经看到过这样的协议(比如二进制倒数计数协议)。另一种特殊的情形是,每个组分配两个站。在一个时槽中,两个站都要传送数据的概率是  $p^2$ ,对于小的  $p$ ,这是可以忽略的。随着在同一个时槽中分配的站数越来越多,冲突的概率也随之增加,但是,给大家竞争



机会的位图扫描的长度却缩减了。我们需要的是一种可以动态地将所有的站分配到各个时槽的方法：当载荷很低的时候，每个时槽中有许多站参与竞争；而当载荷很高的时候，每个时槽中只有很少的站，甚至只有一个站。

### 自适应树搜索协议

一种特殊的、简单的分配方案是，采用第二次世界大战中美国军方为了测试士兵是否感染梅毒而设计的算法(Dorfman, 1943)。简短来说，军方从  $N$  个士兵身上提取血样。然后从每份血样中各取一部分倒入同一个试管中，再对这份混合的血样进行抗体测试。如果没有发现抗体的话，则所有的士兵都是健康的。如果出现了抗体的话，则再准备两份新的混合血样，一份由  $1 \sim N/2$  号士兵的血样混合而成，另一份由剩余的士兵的血样混合而成。递归重复此过程，直到找出所有被感染的士兵。

考虑该算法的计算机化的版本(Capetanakis, 1979)，一种很自然的想法是，把网络中的站看作是二叉树的叶子，如图 4.9 所示。在一次成功传送之后的第一个竞争时槽，即 0 号时槽中，所有的站都允许尝试获取信道。如果发生了冲突，则在 1 号时槽中，只有该树中 2 号节点之下的那些站才可以竞争。如果其中之一获得了信道，则这一帧之后的那个时槽被保留给节点 3 下面的那些站。另一方面，如果节点 2 下面的两个或者多个站希望传送数据，则 1 号时槽中就会发生冲突，在这种情况下，2 号时槽就由节点 4 下面的站来竞争。

从本质上来看，如果在 0 号时槽中发生了冲突，则整棵树都会被遍历到(深度优先)，以便找到所有已准备要传送的站。每一个位时槽都跟树中某一特定的节点相关联。如果在一个位时槽中发生了冲突，则在该位时槽所对应的节点的左和右子节点上继续递归地进行搜索。如果一个位时槽是空闲的，或者在位时槽中只有一个站要传送数据，则对相应节点的搜索便停止，因为这表明已经找到了该节点下面所有已准备要传送的站了(如果有多个站已准备好的话，就会发生冲突)。

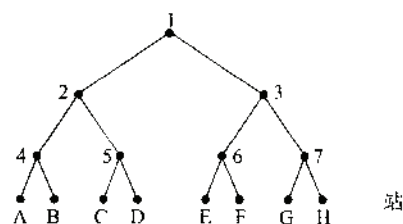


图 4.9 包含 8 个站的树

当系统的载荷较重的时候，将 0 号时槽专门用于节点 1 往往是不值得的，因为只有当恰好一个站要发送帧的时候，这才是有意义的，而当载荷较重的时候这种事件的可能性非常小。类似地，有人可能会说，基于同样的理由，节点 2 和节点 3 也可以跳过去。考虑更为一般的情形，到底应该从树的哪一级上开始搜索呢？很明显，系统的载荷越重，则越是从树的下而节点开始搜索。我们假设每个站根据最近的流量，估计出已经准备好要发送数据的站有  $q$  个。

现在我们对树的级数从上往下进行编号，在图 4.9 中，节点 1 位于第 0 级，节点 2 和 3 位于第 1 级，以此类推。请注意，第  $i$  级上的每个节点下面包含了总站数的  $2^i$ 。如果  $q$  个已经准备就绪的站是均匀分布的，则第  $i$  级上，某一个特定的节点下面的站中应该有  $2^{i+q}$  个是准备就绪的。从直观来看，我们总是期望从一个最优的级数上开始搜索这棵



树,并且,在这个最优的级数上,每个时槽中参与竞争的站的平均数为 1,也就是说,在这个级数上, $2^{-i}q=1$ 。解这个方程,我们可以得到  $i=\log_2 q$ 。

这个基本的算法已经有了大量的改进算法,Bertsekas 和 Gallager(1992)已经对这些算法作了细致的讨论。例如,考虑只有站 G 和 H 想要传送数据的情形。在节点 1 上,冲突发生了,所以往下落到节点 2 上,却发现它是空闲的。再探查节点 3 是毫无意义的,因为它肯定会有冲突(我们已经知道了在节点 1 下面有两个或者多个站要发送数据,并且这些站都不在节点 2 的下面,所以,它们肯定都在节点 3 的下面)。对节点 3 的探查可以跳过去,从而直接尝试节点 6。当发现这次探查仍然是空的时候,节点 7 可以跳过去,接下来就可以尝试节点 G 了。

#### 4.2.5 波分多路访问协议

另一种完全不同的信道分配方法是利用 FDM、TDM 或者两者结合起来,将信道分成多个子信道,然后动态地根据需要分配这些子信道。像这样的方案通常被用于光纤 LAN 上,从而允许在同一时刻,不同的会话使用不同的波长(即频率)。在这一小节中,我们将讨论这样的协议(Humblet et al., 1992)。

构建全光学 LAN 的一种简单方法是使用一个无源的星型连接器(如图 2.10)。实际上,从每个站出来的两根光纤都被熔接到一个玻璃圆柱体上。一根光纤是输出到圆柱体,另一根是从圆柱体输入到站内。从任何一个站输出的光都会照亮圆柱体,所有其他的站也都可以检测到这束光。无源星型结构可以处理数百个站。

为了允许在同一时刻有多个站进行传输,波谱被分成多个信道(波段, wavelength band),如图 2.31 所示。在这个协议中,即 WDMA(Wavelength Division Multiple Access, 波分多路访问),每个站被分配两个信道。一个窄的信道用作控制信道,通过它可以向每个站发送通知信息;一个宽的信道用作数据信道,通过它每个站都可以输出数据帧。

每个信道被分成时槽组,如图 4.10 所示。我们假定在控制信道中时槽的数目为  $m$ ,在数据信道中时槽的数目为  $n+1$ ,这里  $n$  个时槽被用于数据,最后一个时槽被该站用来报告它的状态(最主要的是,这两个信道中哪些时槽是空的)。在这两个信道中,时槽序列无限重复,其中 0 号时槽需要用特殊的方法来标记,这样后来者才能够检测到它。一个全局的时钟被用来同步所有的信道。

该协议支持三种类型的通信流量:(1)恒定速率的、面向连接的通信流量,比如未压缩的视频流;(2)可变速率的、面向连接的通信流量,比如文件传输;(3)数据报流量,比如 UDP 分组。对于两种面向连接的协议,基本的思想是,如果 A 要与 B 进行通信,则 A 必须首先在 B 的控制信道的一个空时槽中插入一个 CONNECTION REQUEST 帧。如果 B 接受的话,则通信过程可以在 A 的数据信道上进行。

每一个站都有两个发送器和两个接收器,如下:

- 一个固定波长的接收器,用于监听它自己的控制信道。
- 一个可调节波长的发送器,用于在其他站的控制信道上发送信息。
- 一个固定波长的发送器,用于发送数据帧。
- 一个可调节波长的接收器,用于选择监听一个数据发送器。

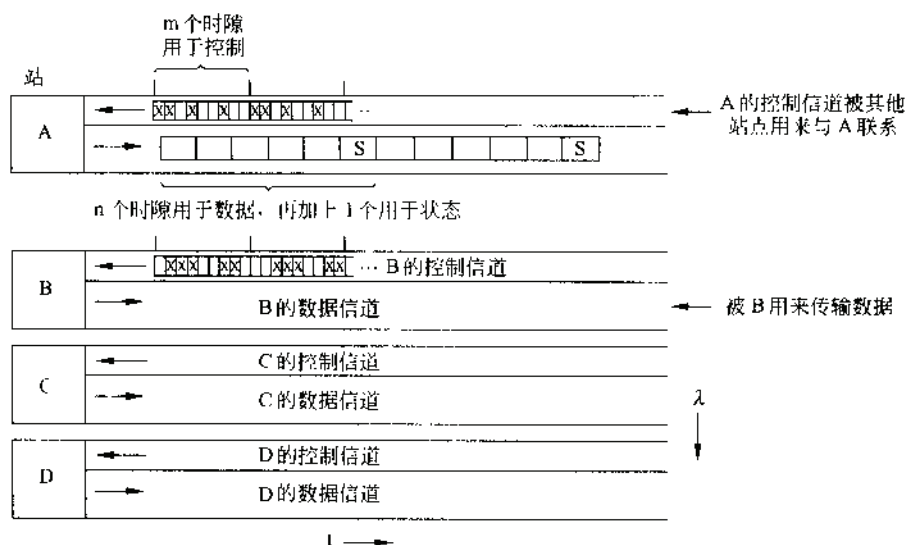


图 4.10 波分多路访问

换句话说,每个站监听自己的控制信道,用来接收进入的请求,但是它必须调节发送器的波长以便获取数据。波长调节可以通过 Fabry-Perot 或者 Mach Zehnder 干涉计来完成,这两种干涉计可以保留期望的波长范围,同时过滤掉所有其他的波长。

现在我们来考虑站 A 与站 B 如何建立起一个第 2 类的通信信道,比如用于文件传输的信道。首先,A 将它的数据接收器调节到 B 的数据信道上,并且等待状态时槽的到来。该时槽将会提供以下信息:当前分配了哪些控制时槽,以及哪些是空闲的。例如,在图 4.10 中,我们可以看到,在 B 的 8 个控制时槽中,0、4 和 5 是空闲的。其他的控制时槽都已经被占用了(用交叉号来表示)。

A 从这些空闲的控制时槽中挑选出一个,比如说 4 号时槽,并且在该时槽中插入它的 CONNECTION REQUEST 消息。由于 B 一直在监视它自己的控制信道,所以它会看到 A 的请求,并且将 4 号时槽分配给 A,以接受 A 的请求。B 通过它的数据信道的状态时槽,来宣布它的分配结果。当 A 看到了状态时槽中的分配结果时,它知道自己有了一个单向的连接。如果 A 请求双向连接的话,则 B 现在必须对 A 重复同样的算法。

当 A 正在试图获取 B 的 4 号控制时槽的时候,有可能 C 也在做同样的事情。A 和 C 都将得不到该控制时槽,它们只要监视 B 的数据信道中的状态时槽,就可以注意到自己失败了。现在它们各自等待一段随机的时间,然后再次重试。

此时,每一方都有一种无冲突的方法来向另一方发送短的控制消息。为了执行文件传输,A 现在向 B 发送一条控制消息,比如,“请注意观察我的下一个 3 号数据输出时槽,那里有送给你一个数据帧”。当 B 获得了这条控制消息的时候,它将它的接收器调节到 A 的输出信道,以便读取数据帧。根据高层协议的规定,B 如果愿意的话,可以使用同样的机制送回一个确认。

请注意,如果 A 和 C 都跟 B 有了连接,并且两者都突然地告诉 B 去检查 3 号时槽的

数据,则问题就来了。B将随机地从中选择一个请求,于是,另一份传送的数据便丢失了。

对于恒定速率的通信流量,可以使用该协议的一个变种。当A请求建立连接的时候,它同时也可以表达这样的信息:如果我在每一次出现3号时槽的时候都发送一帧给你,是否可以呢?如果B能够接受(即,以前还没有将3号时槽答应给别人),则一个有带宽保障的连接就建立起来了。如果B不能接受,则A可以再试一试其他的建议方案,这要取决于它有多少空闲的输出时槽了。

第3类(数据报)流量仍然使用另一个变种。它并不是在刚刚找到的控制时槽(4)中写入一条CONNECTION REQUEST消息,而是一条“DATA FOR YOU IN SLOT 3”(3号时槽中有给你的数据)消息。如果在下一个3号数据时槽中,B是空闲的,则这次传送就会成功。否则的话,数据帧就会丢失。按照这种方式,根本不需要建立任何连接。

该协议的其他一些变种也是有可能的。例如,并不是每个站都有自己独立的控制信道,而是所有的站共享同一个控制信道;在每个组中,每个站被分配到一块时槽,这样可以有效地将多个虚拟信道复用到一条物理信道上。

另外一种可能的做法是,将每个站的信道分成 $m$ 个控制时槽,随后再加上 $n+1$ 个数据时槽,这样每个站只要处理一个可调节的发送器和一个可调节的接收器。这种做法的缺点是,发送方必须等待更长的时间才能抓住一个控制时槽,而且连续的数据帧离得比较远,因为中间有一些控制信息。

人们还提出并实现了很多其他的WDMA协议,这些协议在许多细节方面各有不同。有的协议只有一个控制信道,其他的有多个控制信道。有的协议考虑了传播延迟,而其他的则没有考虑。有的协议在模型中显式地考虑了调节波长所需要的时间,而其他的协议则忽略了这一段时间。另外,这些协议在处理过程复杂性、吞吐量和伸缩性方面各有所不同。当系统中用到了大量频率的时候,这样的系统有时候称为DWDM(Dense Wavelength Division Multiplexing,密集波分多路复用)。有关更多的信息请参见(Bogineni et al., 1993; Chen, 1994; Goralski, 2001; Kartalopoulos, 1999; and Levine and Akyildiz, 1995)。

#### 4.2.6 无线LAN协议

随着移动计算和通信设备的数量的增长,将它们与外部世界连接起来的需求也越来越强烈。即使是非常早期的移动电话,它也具备了与其他电话连接的能力。早期的便携式计算机没有这样的能力,但是,没过多久,调制解调器在笔记本计算机上便很普及了。要想连接到网络上,这些计算机必须与墙上的电话线插口连接起来。要求使用有线的方式与固定的网络连接起来,这意味着这些计算机是可便携的,但不是可移动的。

为了实现真正的移动性,笔记本计算机需要使用无线(或者红外)信号进行通信。按照这种方式,用户在徒步行走或者驾船游玩时仍然可以阅读或者发送电子邮件。如果一个系统中的笔记本计算机通过无线电波进行通信,则该系统可以被认为是一个无线LAN,我们在1.5.4节中已经讨论过了。这些LAN与传统的LAN稍微有一些不同的特性,它们要求特殊的MAC子层协议。在这一小节中,我们将介绍一些这样的协议。有关无线LAN的更多信息,请参见Geier, 2002; O'Hara and Petrick, 1999。

无线 LAN 的一种常见的配置方案是,在一个办公楼内静态地放置一些基站(也称为访问点)。所有的基站通过铜线或者光纤连接起来。如果基站和笔记本计算机的传输功率被调节到大约 3 或 4 米的距离范围,那么,每个房间变成了一个蜂窝单元,整个大楼变成了一个大的蜂窝系统,就如同我们在第 2 章中学习过的传统蜂窝电话系统中的情形一样。与蜂窝电话系统不同的是,每个单元只有一个信道,该信道覆盖了整个可用的带宽,并且也覆盖了该单元中所有的站。典型情况下,它的带宽是 11~54Mbps。

在我们下面的讨论中,我们将简单地假设所有的无线电发射器都有某个固定的范围。当一个接收器同时位于两个活动的发射器的范围内时,结果信号通常是混乱的,也是无效的,换句话说,我们在本小节的讨论中将不再进一步考虑 CDMA 类型的系统了。有一点很重要,也必须认识到,在有些无线 LAN 中,并不是所有的站都在另一个站的范围之内,这将会导致一系列的复杂性。而且,对于室内的无线 LAN,站与站之间的墙壁对于每个站的有效范围也可能会有很大的影响。

一种简单而又直接的使用无线 LAN 的方法可能是试一试 CSMA: 每个站监听是否有其他的通信流量,只有当没有其他站在传送数据的时候它才可以传送。这种做法的麻烦在于,此协议并非真的很合适,因为这里的冲突发生在接收方,而并非是发送方。为了看清楚问题的实质,请考虑图 4.11 中 4 个无线站的情形。对于我们的问题而言,这些无线站到底是基站还是笔记本计算机无关紧要。无线电波的范围是这样的: A 和 B 相互之间都在对方的范围之内,所以它们相互之间可能会干扰对方;C 也可能会同时干扰到 B 和 D,但是不会干扰 A。

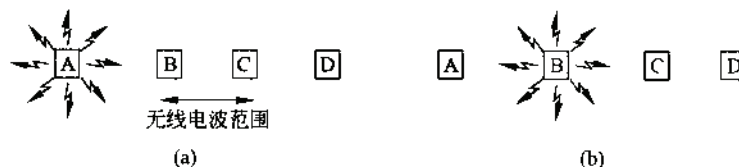


图 4.11 一个无线 LAN

(a) A 发送的情形; (b) B 发送的情形

首先考虑当 A 向 B 传送数据时的情形,如图 4.11(a)所示。如果 C 正在检测介质的话,那么它将不会听到 A,因为 A 在它的距离范围之外,因此它会错误地得出结论:它可以向 B 传送数据。如果 C 真的开始传送数据了,则在 B 处将会产生干扰,从而扰乱了 A 送出的帧。由于竞争者离得太远而导致一个站无法检测到潜在的介质竞争对手,这个问题称为**隐藏站问题(hidden station problem)**。

现在考虑相反的情形: B 向 A 传送数据,如图 4.11(b)所示。如果 C 正在检测介质,则它将会听到有一个传输正在进行,从而也会错误地得出结论:它不能向 D 发送数据,而实际上,它所听到的传输过程只会影响到 B 和 C 之间的区域中的接收过程,但是,不会影响到目标接收方(D)所在的区域。这个问题称为**暴露站问题(exposed station problem)**。

所以,问题是,在开始一个传输之前,一个站真正希望知道的是在接收方的周围是否有活动情况。CSMA 只是告诉它,在检测载波的站周围是否有活动发生。对于有线的情形,所有的信号传播到所有的站,所以,在同一时刻,系统中任何地方都只能有一个传输过

程在进行。对于一个基于短距离无线电的系统而言,多个传输过程可以同时发生,只要它们有不同的目标方,并且这些目标方相互之间都在对方的范围之外。

考虑这个问题的另一种方法是想象一个办公楼,其中每个雇员都有一台无线的笔记本电脑。假设 Linda 希望给 Milton 发送一条消息,Linda 的计算机在发送之前,检测一下局部的环境,发现当前并没有活动的传输过程,于是便开始发送数据。然而,在 Milton 的办公室中仍然有可能会发生冲突,因为另外一个第三方当前也在给他发送数据,并且此第三方所在的位置离 Linda 很远,以至于 Linda 的计算机根本就检测不到。

#### MACA 和 MACAW

早前为无线 LAN 设计的一个协议是 MACA (Multiple Access with Collision Avoidance,避免冲突的多路访问)(Karn, 1990)。MACA 背后的基本思想是,发送方刺激一下接收方,让它输出一个短帧,因此,接收方附近的站可以检测到该帧,从而在接下去的数据帧(较大)传输过程中它们不再发送数据了。图 4.12 演示了 MACA 协议的过程。

现在我们来考虑 A 如何向 B 发送一帧。A 首先给 B 发送一个 RTS (Request To Send) 帧,如图 4.12(a)所示。这个短帧(30 字节)包含了随后将要发送的数据帧的长度。然后,B 用一个 CTS (Clear to Send) 作为应答,如图 4.12(b)所示。此 CTS 帧包含了数据帧长度(从 RTS 帧中复制过来)。A 在收到了 CTS 帧之后便开始传输。

现在我们来看看,如果其他有的站也听到了这些帧,它们会如何反应。如果一个站听到了 RTS 帧,那么它一定离 A 很近,它必须继续保持沉默,至少等待足够长的时间以便在无冲突情况下 CTS 被送回给 A。如果一个站听到了 CTS,则它一定离 B 很近,在接下来的数据传输过程中它必须一直保持沉默,只要检查一下 CTS 帧,它就可以知道数据帧的长度(即数据传输要持续多久)。

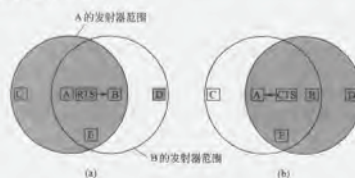


图 4.12 MACA 协议  
(a) A 发送一个 RTS 给 B; (b) B 给 A 返回一个 CTS

在图 4.12 中,C 落在 A 的范围内,但不在 B 的范围内。因此,它听到了 A 送出的 RTS,但是没有听到 B 送出的 CTS。只要它没有干扰 CTS,那么在数据帧被传输的过程中,它可以自由地发送任何信息。相反,D 落在 B 的范围内,但不在 A 的范围内。它听不到 RTS 帧,但是听到了 CTS 帧。听到了 CTS 帧,这意味着它与一个将要接收数据帧的



站离得很近,所以,它会等到那一帧的传送完成以后再发送任何信息。站 E 听到了这两条控制消息,所以,与 D 一样,在数据帧完成之前它必须保持安静。

尽管有了这些防范措施,冲突仍有可能发生。例如,B 和 C 可能会同时给 A 发送 RTS 帧。这些帧将发生冲突,因而会丢失。在发生了冲突的情况下,一个失败的发送方(即,在期望的时间间隔内没有听到 CTS)将等待一段随机的时间,以后再重试。其中所用的算法是二元指数后退算法,当我们讨论以太网的时候将要学习该算法。

在 MACA 模拟研究的基础上,Bharghavan 等(1994 年)进一步改进了 MACA,提高了它的性能,并且将它们的新协议命名为 MACAW(MACA for Wireless,无线的 MACA)。开始时,他们发现,如果没有数据链路层的确认,则在传输层注意到丢帧之前,这些丢失的帧不会被重传,而到那时候就太晚了。他们解决该问题的做法是,在每个成功的数据帧之后引入一个 ACK 帧。他们还注意到了 CSMA 有一些用处,也就是,当附近的一个站正在给某一个目标站发送 RTS 的时候,它可以避免当前的站也给同一个目标站发送 RTS 帧,所以载波检测被加入进来。而且,他们决定为每一个单独的数据流(即源-目标对),而不是为每一个站运行后退算法。这项改进提高了协议的公平性。最后,他们还增加了一种供各个站之间交换有关拥塞信息的机制,以及一种使后退算法面对临时问题时反应不很强烈的方法,从而提高了系统的性能。

## 4.3 以太网

我们刚才已经结束了对于信道分配协议的一般性讨论,现在我们应该来看一看这些原理是如何应用到实际系统中的,特别是 LAN。正如 1.5.3 节所讨论的那样,IEEE 已经标准化了许多种局域网和城域网,这些标准都在 IEEE 802 的名字下面。有一些网络幸存下来了,但其他很多网络并没有存活下来,如图 1.38 所示。有些相信轮回转世的人可能会认为达尔文又回来了,他作为 IEEE 标准化协会的一名成员来清除掉那些不再适合于生存的协议。在幸存者当中,最重要的是 802.3(以太网)和 802.11(无线 LAN)。对于 802.15(蓝牙)和 802.16(无线 MAN),现在还很难说今后会怎么样。要想知道答案,请参考本书未来出版的第 5 版。802.3 和 802.11 有不同的物理层,也有不同的 MAC 子层,但是,它们有共同的逻辑链路控制子层(定义在 802.2 中),所以,在这两种网络中,与网络层的接口是相同的。

我们在 1.5.3 小节中已经介绍了以太网,这里将不再重复同样的内容。相反,我们将焦点集中在以太网的技术细节、协议,以及最近在高速(千兆)以太网领域中的发展情况。对于以太网和 IEEE 802.3,除了下面将要讨论到的两个微小区别以外,它们是完全相同的,所以,许多人常常不加区分地使用术语“以太网”和“IEEE 802.3”,我们后而也将这样做。有关以太网的更多信息,请参见(Breyer and Riley, 1999; Seifert, 1998; 以及 Spurgeon, 2000)。

### 4.3.1 以太网电缆

既然名字“以太网”是指电缆(即以太),那么,我们首先从这里开始讨论。通常使用的



电缆有 4 种,如图 4.13 所示。

名称	电缆	最大的段长度	每段节点数	优点
10Base5	粗同轴电缆	500m	100	早期的电缆,现在已经废弃了
10Base2	细同轴电缆	185m	30	不需要集线器
10Base-T	双绞线	100m	1024	最便宜的系统
10Base-F	光纤	2000m	1024	最适合于在楼与楼之间使用

图 4.13 最常见的几种以太网电缆

从历史顺序来看,最先出现的是 **10Base5** 电缆,俗称**粗以太网**。它就好像是一根黄色的庭院水管,每隔 2.5 米标记了分接头的插入处(实际上,802.3 标准并不要求电缆必须是黄色的,但是它确实建议使用黄色)。要连接到这样的电缆上,通常需要使用**插入式分接头(vampire tap)**,在分接头中,有一根针被非常小心地插入到同轴电缆的内芯中。术语 10Base5 的含义是,它运行在 10Mbps 的速率上,使用基带信令,并且,所支持的分段长度可以达到 500 米。第一个数字是以 Mbps 为单位的速率值,然后紧跟着单词“Base”(有时候是“BASE”),这标明了它使用基带传输。过去还有一种宽带传输,称为 10Broad36,但是,它一直没有在市场上流行起来,后来就消失了。最后,如果介质是同轴电缆,则它的长度被附在“Base”之后,以 100m 为单位(四舍五入)。

在历史上第二种电缆类型是 **10Base2**,或者**细以太网**。与庭院软管式的粗以太网电缆不同的是,它很容易弯曲。要连接到细以太网上,不再是使用插入式分接头,而是使用工业标准的 BNC 连接器来构成 T 型接头。BNC 连接器更容易使用,并且也更可靠。细以太网比较便宜,也易于安装,但是,它的每一段最长只能是 185m,而且每一段只能容纳 30 台机器。

对于这两种介质,检测电缆断裂、电缆超长、分接头坏掉,或者 BNC 连接器松动成了大问题。由于这样的原因,人们已经研制了许多技术来捕捉故障。基本的原理是,向电缆中注入一个已知形状的脉冲。如果该脉冲碰到了一个故障,或者到达了电缆的末端,则会产生一个回波送回来。只要仔细地记录下从发送脉冲到接收到回波这一段时间间隔,就有可能确定回波的来源。这项技术称为**时间域反射计(time domain reflectometry)**。

由于找到电缆断裂处非常不方便,因而该问题导致了另一种完全不同的连线模式,在这种连线模式中,所有的站都有一条电缆连接到一个中心**集线器(hub)**上,通过中心集线器,所有的站被连接到一起,就好像它们都被焊接到一起一样。通常,这些连线就是电话公司的双绞线,因为大多数的办公楼已经布好了这些线,而且往往有足够的空余线可以使用。这种方案称为 **10Base-T**。集线器并不会缓存任何进来的流量。我们将在本章后面讨论这种思想的一个改进版本(交换机),它能缓存进来的流量。

这三种连线的方案如图 4.14 所示。对于 10Base5,一个**收发器(transceiver)**紧紧地夹住电缆,以便它的分接头可以接触到电缆的内芯。收发器中也包含了用于载波检测和冲突检测的电路,当检测到冲突的时候,收发器也会发送一个特殊的无效信号到电缆上,以确保所有其他的收发器也知道发生了冲突。

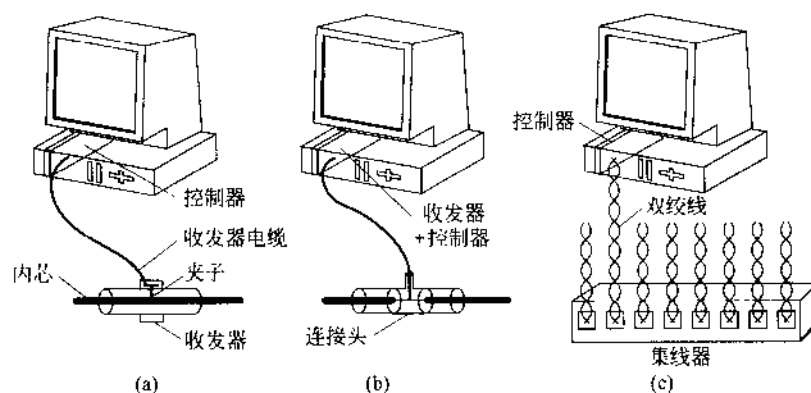


图 4.14 3 种以太网电缆  
(a) 10Base5; (b) 10Base2; (c) 10Base-T

对于 10Base5,通过一根收发器电缆(transceiver cable)或者下路电缆(drop cable)将收发器连接到计算机的接口卡上。收发器电缆可以达到 50m 长,它包含 5 对独立的屏蔽双绞线。其中两对分别用于数据输入和数据输出。还有两对用于控制信息的输入和输出。第五对并不总是会用到,计算机通过这对线可以给收发器电路供电。有些收发器最多允许 8 台邻近的计算机连上来,这样可以减少所需收发器的数目。

收发器电缆终止于计算机内部的接口卡。该接口卡包含一个控制器芯片,用来向收发器传送帧,或者从收发器接收帧。控制器负责将数据装配成正确的帧格式,以及为送出去的帧计算校验和,并且对进来的帧检验其校验和。有些控制器芯片也为进来的帧管理了一个缓冲区池;为要传送出去的帧管理了一个缓冲区队列;并且可以与主计算机进行直接的内存传输,以及其他的网络管理功能。

对于 10Base2,与电缆的连接处只是一个无源的 BNC T 型连接器。收发器电路位于控制器卡上,每个站总是有自己独立的收发器。

对于 10Base-T,根本就没有共享的电缆,每个站通过一根专用的(即非共享的)电缆连接到集线器(一个充满了电路的箱子)上。在这种配置中,增加和去掉一个站都非常简便,而且,电缆断裂也很容易检测。10Base-T 的缺点是,从集线器出来的电缆的最大长度只有 100 米,如果使用高质量的 5 类双绞线的话,也许能够达到 200 米。然而,由于它在现有的网络中的广泛应用,以及它的易维护性,所以,它很快占据了统治地位。本章后面还将讨论 10Base-T 的一个更快的版本(100Base-T)。

第四种以太网电缆是 10Base-F,它采用了光纤。这种连接方式由于连接器和终结器的成本开销而非常昂贵,但是,它有极好的抗噪声能力,适用于楼与楼之间的连接,或者用于远距离隔开的集线器之间的连接。线的长度即使到上千米也是允许的。它还提供了良好的安全性,因为在光纤上搭线窃听比在铜线上要困难得多。

图 4.15 显示了在一个建筑物内进行布线的几种方案。在图 4.15(a)中,用一根电缆蛇形地穿越各个房间,每个站都在离自己最近的点上接入电缆。在图 4.15(b)中,一根垂直的主干线从地下室通到房顶上,在每一层上用水平的电缆通过特殊的放大器(中继器)

连接到主干线上。在有些建筑物内,水平的电缆是细缆,而骨干电缆是粗缆。最普遍的拓扑结构是树形,如图 4.15(c),因为在一个网络中如果某些站对之间存在两条路径的话,则两个信号之间会产生干扰。

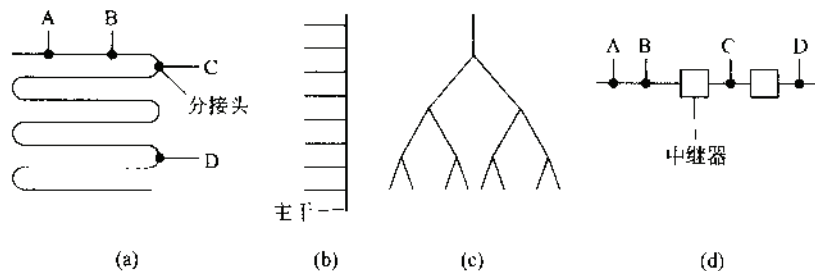


图 4.15 电缆拓扑结构  
(a)线形; (b)主干; (c)树型; (d)分段

在每一种版本的以太网中,每段电缆的长度都有最大值限制。为了构建更大的网络,多根电缆可以通过**中继器(repeater)**连接起来,如图 4.15(d)所示。中继器是一个物理层设备。它在两个方向上接收、放大(重新生成)和重传信号。从软件角度来看,通过中继器连接起来的一系列电缆段与单根电缆没有任何区别(除了中继器带来了一点延迟以外)。一个系统可以包含多根电缆段和多个中继器,但是两个收发器之间不能超过 2.5 公里远,并且任何两个收发器之间的路径上不得跨越多于 4 个中继器。

### 4.3.2 曼彻斯特编码

没有一种以太网版本使用直接的二进制编码,即 0 伏表示“0”位,5 伏表示“1”位,因为这样会导致歧义。如果一个站发送了位串 0001000,则其他的站有可能会错误地将它解释为 10000000 或者 01000000,因为它们无法区分到底是发送方空闲(0 伏)呢,还是一个“0”位(也是 0 伏)。这个问题可以这样来解决:用+1 伏表示 1,用-1 伏表示 0。但是,如果接收方使用的采样频率与发送方生成信号时所使用的频率稍微有点差别的话,则仍然会有问题。不同的时钟速度也会让接收方和发送方无法同步到统一的位边界上,特别是经过了一长串连续的 0 或者一长串连续的 1 之后。

这里所需要的是一种“让接收方在没有外部时钟参考的情况下,可以毫无歧义地确定每一位的起始、结束或者中间位置”的方法。有两种这样的方法,分别称为**曼彻斯特编码(Manchester encoding)**和**差分曼彻斯特编码(differential Manchester encoding)**。利用曼彻斯特编码,每一位的周期分成两个相等的间隔。二进制“1”位在发送时,在第一个间隔中为高电压,在第二个间隔中为低电压。二进制“0”正好相反:首先是低电压,然后是高电压。这种方案可以保证每一个位周期中都有一个中间电压变化,这使得接收方很容易与发送方同步起来。曼彻斯特编码的一个缺点是,它所要求的带宽是直接二进制编码的两倍,因为脉冲是位宽度的一半。例如,为了在 10Mbps 速率上发送数据,则每秒钟信号必须要改变 20M 次。图 4.16(b)显示了曼彻斯特编码。

差分曼彻斯特编码是基本曼彻斯特编码的一个变种,如图 4.16(c)所示。在这种编

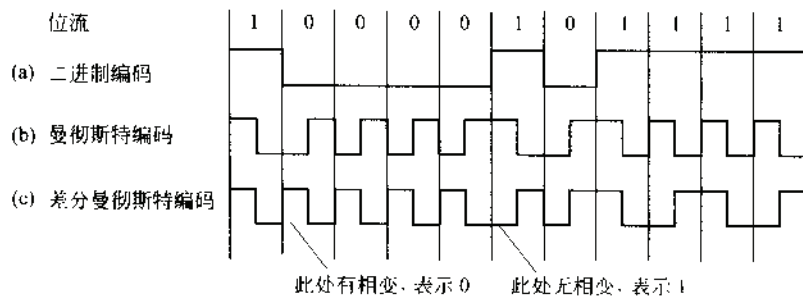


图 4.16

码中,如果在间隔的起始处没有相变,则表示位“1”;如果在间隔的起始处出现了相变,则表示位“0”。在这两种情况下,位周期的中间也会有一个相变。差分方案需要更加复杂的设备,但是提供了更好的抗噪声能力。由于曼彻斯特编码方案比较简单,所以,所有的以太网系统都使用了这种编码方案。高信号为+0.85 伏,低信号为-0.85 伏,直流电压值为 0 伏。以太网并没有使用差分曼彻斯特编码方案,但是其他的 LAN(比如 802.5 令牌环网)使用了这种编码方案。

### 4.3.3 以太网 MAC 子层协议

早期的 DIX(DEC、Intel、Xerox)帧结构如图 4.17(a)所示。每一帧都从一个 8 字节的前导域(Preamble)开始,该前导域包含了位模式 10101010。这个位模式经过曼彻斯特编码之后,将会产生一个 10-MHz 的方波(6.4 $\mu$ s),因而使得接收方的时钟与发送方的时钟很方便地同步到一起。当然,对于一帧的剩余部分,它们也必须保持同步,它们利用曼彻斯特编码可以识别位的边界。

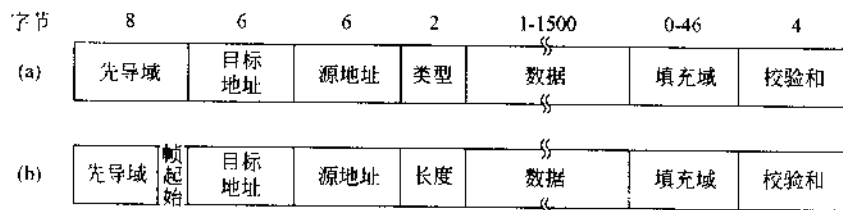


图 4.17 帧格式

(a) DIX 以太网; (b) IEEE 802.3

帧结构中包含两个地址:一个是目标地址,另一个是源地址。尽管标准中允许使用 2 字节或者 6 字节的地址,但是,针对 10Mbps 基带标准定义的参数只使用 6 字节的地址。目标地址的最高位若是 0,则表示普通地址;若是 1,则表示组地址。利用组地址,使得多个站可以监听同一个地址。当一帧被发送给一个组地址的时候,该组中的所有站都会接收到该帧。向一组站发送数据称为**多播**(multicast,也称为**多点传送**)。由全部的“1”位构成的地址被保留用于**广播**(broadcast)。如果一帧的目标地址域中包含全部的“1”,则网络

上所有的站都会接受该帧。多播和广播之间的差异是非常重要的,这里有必要再次重复一下。一个多播帧被发送给以太网上选择出来的一组站;而广播帧则被发送给以太网上的所有站。多播是有选择性的,但是要涉及到组的管理。广播是粗粒度的,并不要求任何组管理的支持。

有关编址的另一个有趣的特征是使用第 46 位(紧邻最高位)来区分局部地址或全局地址。局部地址是由每个网络管理员分配的,在局部网络之外并无意义。相反,全局地址则是由 IEEE 统一分配的,它可以保证世界上任何两个站都不会有同样的全局地址。全局地址的可用位数为  $48-2=46$  位,大约等于  $7 \times 10^{13}$  大小的空间。基本的思路是,任何一个站只要给出正确的 48 位数值就可以惟一标识另外一个站。我们要到网络层部分再介绍如何确定目标站。

接下来是类型(Type)域,它告知接收方应该如何处理这一帧。在同一台机器上,有可能同时有多个网络层协议在使用,所以,当一个以太网帧到达的时候,内核必须知道应该将此帧交给哪一个网络层协议。类型域指定了应该将此帧交给哪个进程。

接下来是数据域部分,最多可以达到 1500 个字节。在制定 DIX 标准的时候,这个值的选择有一定的随意性,最主要的依据是,收发器需要足够的内存(RAM)来存放一个完整的帧,而 RAM 在 1978 年的时候还很昂贵。这个上界值越大,则意味着需要更多的 RAM,因而也意味着收发器的造价更高。

除了有最大的帧长度限制外,这里还有最小的帧长度限制。虽然有时候 0 字节的数据域也是有用的,但是,它会带来一个问题。当一个收发器检测到冲突的时候,它会截断当前的帧,这意味着冲突帧中已经送出的位将会出现在电缆上。为了更加容易地区分有效帧和垃圾数据,以太网要求有效帧必须至少 64 字节长,从目标地址算起一直到校验和,也包括这两个域本身。如果一帧的数据部分少于 46 字节的话,则使用填充(Pad)域来填充该帧,以便达到最小的长度。

限制最小帧长的另一个(也是更重要的)理由是避免出现这样的情况:当一个短帧还没有到达电缆的远端的时候,发送站已经完成了该短帧的传送,而在电缆的远端处,该帧可能与另一帧发生冲突。这个问题如图 4.18 所示。在 0 时刻,位于电缆一端的站 A 送出一帧。我们假设该帧到达另一端的传播时间为  $\tau$ 。正好在该帧到达另一端之前的某一时刻(即,在  $\tau-\epsilon$  时刻),位于最远处的一个站, B, 也开始传送数据。当 B 检测到它所接收到的信号比它发送的信号更强的时候,它知道已经发生了冲突,所以它放弃了自己的传送任务,并且产生一个 48 位的突发噪声以警告所有其他的站。换句话说,它阻塞了以太,以便保证发送方不会漏掉这次冲突。大约在  $2\tau$  的时候,发送方看到了突发噪声,并且也放弃了它的传送任务。然后它等待一段随机的时间,以后再重试。

如果一个站试图传送一个非常短的帧,则可以想象:虽然冲突发生了,但是在突发噪声回到发送方( $2\tau$ )之前,传送任务已经完成了。然后,发送方将会不正确地得出结论:这一帧已经被成功地发送出去了。为了避免发生这样的情况,所有的帧至少需要  $2\tau$  时间才能完成发送,因此,当突发噪声回到发送方时,传送过程仍在进行。对于一个最大长度为 2500 米、具有 4 个中继器的 10Mbps LAN(符合 802.3 规范)来说,在最差情况下,往返一



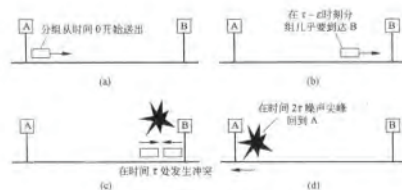


图 4.18 冲突检测需要  $2\tau$  时间

周的时间大约是  $50\mu s$ ，其中包括了通过 4 个中继器所需要的时间，毫无疑问，这段时间也肯定不会等于 0。因此，最小的帧至少也需要这样长的时间来传输。在  $10\text{Mbps}$  的情况下，一位需要  $100\text{ns}$ ，所以 500 位是保证可以工作的最小帧。考虑到加上一安全余量，该数字被增加到 512 位，或者 64 字节。对于小于 64 字节的帧，则通过填充域填充到 64 字节长。

随着网络速度的提高，最小帧长也必须成比例地增加，或者最大电缆长度成比例地缩短。对于一个工作在  $1\text{Gbps}$  速率上的 2500 米的 LAN 来说，最小帧长将必须达到 6400 字节。或者换一种做法，最小帧长可以是 640 字节，但是任何两个站之间的最大距离为 250 米。当我们转向几千兆 bps 网络的时候，这些限制真是变得越来越痛苦了。

以太帧的最后一个域是校验和 (Checksum)。它实际上是数据域的一个 32 位散列码。如果有一些数据位接收错误的话 (由于电缆上噪声的原因)，则校验和几乎肯定是错误的，因此错误可以被检测出来。这里的校验和算法是第 3 章中讨论的循环冗余校验码的一种。它只提供检错功能，并不提供纠错功能。

当 IEEE 标准化以太网时，委员会对 DIX 帧格式做了两个改动，如图 4.17(b) 所示。第一个改动是将前导域降低到 7 个字节，并且将空出来的一个字节用作帧起始 (Start of Frame) 分界符，这样做的目的是为了与 802.4 和 802.5 兼容。第二个改动是将类型域变成了长度 (Length) 域。当然，现在接收方无法确定应该对接收到的帧做何处理，但是，这个问题可以这样来解决：在数据部分增加一个小的头信息，由它来提供帧类型的信息。当我们在本章后面讨论到逻辑链路控制的时候，我们将讨论数据部分的格式。

不幸的是，当 802.3 被公布时，已经有大量的针对 DIX 以太网的硬件和软件在使用了，所以，很少再有厂商和用户热衷于将类型域转变成长度域。1997 年，IEEE 只好认输了，并且声明两种做法都是很好的。幸运的是，在 1997 年之前使用的所有类型域都大于 1500。因此，任何小于或者等于 1500 的数值可以被解释为长度域；而任何大于 1500 的数值则可以被解释为类型域。现在，IEEE 维持了人人使用其标准的局面，同时，其他每一个人可以继续保持自己原来的做法，而不用有任何心虚的感觉。



#### 4.3.4 二元指数后退算法

现在我们来了解一下当冲突发生时,如何来确定随机等待的时间。该随机模型如图 4.5 所示。在一次冲突之后,时间被分成离散的时槽,其长度等于最差情况下在以太介质上往返传播所需要的时间。为了达到以太网所允许的最长路径,时槽的长度被设置为 512 位时间,或  $51.2\mu\text{s}$ ,就像前面提到的那样。

在第一次冲突发生以后,每个站等待 0 个或者 1 个时槽之后再重试发送。如果两个站冲突之后又选择了同一个随机数,则它们将再次冲突。在第二次冲突之后,每个站随机选择 0、1、2 或者 3,然后等待这么多个时槽。如果第三次冲突又发生了(发生的概率为 0.25),则从 0 到  $2^3-1$  之间随机选择一个数,并等待如此多个时槽。

一般地,在第  $i$  次冲突之后,在  $0 \sim 2^i-1$  之间随机选择一个数,然后等待这么多个时槽。然而,到达 10 次冲突之后,随机数的区间固定在最大值 1023 上,以后不再增加了。在 16 次冲突之后,控制器放弃努力,并且给计算机送回一个失败报告。进一步的失败恢复工作取决于高层协议。

这个算法称为二元指数后退(binary exponential backoff),它可以动态地适应发送站的数量。如果针对所有冲突的随机数最大值都是 1023,则两个站第二次发生冲突的几率几乎可以忽略,但是,在一次冲突之后的平均等待时间将是数百个时槽,这样会引入明显的延迟。另一方面,如果每个站总是等待 0 个或者 1 个时槽,那么,如果 100 个站都要发送数据的话,则只有当其中 99 个站选择 1 而剩下一个站选择 0 时,它们才不再发生冲突。这可能需要几年的时间。上面介绍的算法的做法是,随着越来越多连续冲突的发生,随机等待的间隔也呈指数增加。这种做法的好处是,如果只有少量的站发生冲突,则它可以确保较低的延迟;但是,当许多站发生冲突时,它也可以保证在一个相对合理的时间间隔内解决冲突问题。将后退的步子终止在 1023 上,这样可以避免增长得太大。

到目前为止的介绍中,CSMA/CD 并没有提供确认。很少发生冲突并不保证所发送的数据位不会被电缆上的尖噪声所破坏,因此,对于可靠的通信过程,目标方必须对校验和进行检验,如果校验和正确,则给源送回一个确认帧。通常,对于协议而言,此确认帧只不过是另一帧而已,它也要像数据帧那样竞争信道。然而,对竞争算法进行简单的修改,就可以加速回帧确认过程(Tokoro and Tamaru, 1997)。所需要做的工作是,将成功传输之后的第一个竞争时槽保留给目标站。遗憾的是,标准并不允许这样做。

#### 4.3.5 以太网的性能

现在,我们来简要地讨论一下在重载荷和恒定载荷(即,总是有  $k$  个站要传送数据)条件下以太网的性能情况。关于二元指数后退算法的严格分析是非常复杂的。相反,我们将采用 Metcalfe 和 Boggs(1976 年)的方法,并假定每个时槽中重传的概率是不变的。如果每个站在一个竞争时槽中传送帧的概率为  $p$ ,那么,在这个时槽中,某一个站获得信道的概率  $A$  为:

$$A = kp(1-p)^{k-1} \quad (4-5)$$

当  $p=1/k$  的时候,  $A$  最大;并且当  $k \rightarrow \infty$  的时候,  $A \rightarrow 1/e$ 。竞争间隔正好等于  $j$  个时

槽的概率为  $A(1-A)^{j-1}$ , 所以, 每一次竞争的平均时槽数为:

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}$$

由于每个时槽的时间间隔为  $2\tau$ , 所以, 平均竞争间隔  $w$  为  $2\tau/A$ 。假设最优的  $p$ , 并且竞争时槽的平均数永远不超过  $e$ , 于是,  $w$  至多为  $2\tau e \approx 5.4\tau$ 。

如果传送每一帧平均需要  $P$  秒, 那么, 当许多站都要传送帧的时候,

$$\text{信道效率} = \frac{P}{P + 2\tau/A} \quad (4-6)$$

这里我们可以看出, 任何两个站之间的最大电缆距离也会影响性能, 因而导致产生了一些不同于图 4.15(a) 的拓扑结构。电缆越长, 则竞争间隔也越长。这正是之所以以太网标准规定最大电缆长度的原因。

对于每帧  $e$  个竞争时槽的最优情形, 我们从帧的长度  $F$ 、网络带宽  $B$ 、电缆长度  $L$  以及信号的传播速度  $c$  来看一下等式 (4-6) 是有意义的。利用  $P = F/B$ , 等式 (4-6) 变成:

$$\text{信道效率} = \frac{1}{1 + 2BLE/cF} \quad (4-7)$$

当分母中的第二项变大时, 网络的效率将会变低。特别是, 在帧长度给定的情况下, 增加网络的带宽或者距离 (即  $BL$  的乘积) 将会降低网络的效率。不幸的是, 许多关于网络硬件的研究工作都要增大此乘积值。人们总是希望在长的距离上有高的带宽 (例如, 光纤城域网 [MAN]), 但是, 上面的公式说明了, 用这种方式实现的以太网可能并不是适合这些应用的最佳系统。当我们在本章后面学习交换式以太网的时候, 我们将会看到其他一些实现以太网的方法。

利用等式 (4-7), 在  $2\tau = 51.2\mu s$  以及 10Mbps 数据速率的情况下, 信道的效率与发送站数量之间的关系如图 4.19 所示。对于 64 字节的时槽时间, 64 字节的帧并不是最有效的, 这并不奇怪。另一方面, 当帧长度为 1024 字节, 以及每个竞争间隔趋近于  $e$  个 64 字节时槽的时候, 竞争周期为 174 字节长, 效率为 0.85。

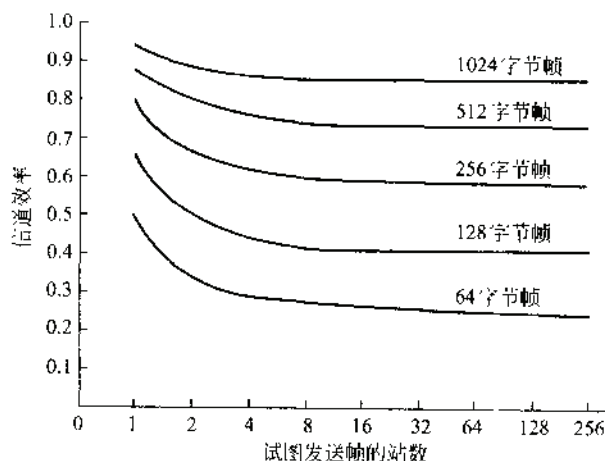


图 4.19 速率为 10Mbps、长为 512 位的时槽的以太网效率

为了确定在高载荷条件下准备要传送的站的平均数,我们可以使用下面的粗略计算方法。每一帧占用信道的时间为一个竞争周期和一个帧传输时间,总共为  $P+w$  秒。因此,每秒钟传送的帧数为  $1/(P+w)$ 。如果每个站生成帧的平均速率为  $\lambda$  帧/秒,那么,当系统在状态  $k$  的时候,所有未阻塞的站合起来的总输入率为  $k\lambda$  帧/秒。由于在平衡状态下,输入率和输出率应该是相等的,所以,我们可以将这两个表达式用等号连接起来,然后求出  $k$ (请注意, $w$  是  $k$  的一个函数)。关于该问题一个更复杂的分析请参考(Bertsekas and Gallager, 1992)。

可能值得提一下的是,关于以太网(和其他的网络)有大量理论性的性能分析。几乎所有这些工作都假设网络流量是泊松分布的。当研究人员开始检查实际数据的时候,他们发现网络流量很少呈泊松分布,而是自相似的(Paxson and Floyd, 1994; and Willinger et al., 1995)。这就意味着即使在一段较长的时间上进行平均也不能使流量变得更平滑。一小时内每分钟的帧平均数的变化情况与一分钟内每秒钟的帧平均数的变化是一致的。这一发现导致的结果是,有关网络流量的大多数模型都不能应用于现实世界,而应该有所保留地使用(甚至要相当保留才行)。

#### 4.3.6 交换式以太网

随着以太网中的站越来越多,流量也急剧上升。最终,LAN 将会达到饱和。一种办法是提高速度,比如将 10Mbps 换成 100Mbps。但是,随着多媒体数据的快速增长,即使 100Mbps 或者 1Gbps 的以太网也会变得饱和。

幸运的是,还有另一种办法可以处理不断增长的载荷:交换式以太网,如图 4.20 所示。这种系统的核心是一个交换机(switch),它包含一块高速的底板,以及一个往往可以容纳 4~32 块插线卡的空间,每块插线卡包含 1~8 个连接器。绝大多数情况下,每个连接器有一个 10Base-T 双绞线接口,可以连接到一台主计算机上。

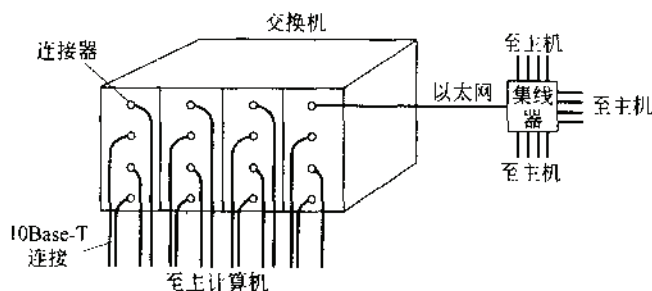


图 4.20 交换式以太网的一个简单示例

当一个站希望传送一个以太网帧的时候,它向交换机送出一个标准帧。获得该帧的插卡可能会检查这一帧的内容,看它的目标站是否也连接在同一块卡上。如果是的话,该帧将被复制到那里。如果不是,则通过交换机的高速底板,该帧被送到目标站的插卡中。通常底板会使用私有的协议,它的运行速度可以达到好几个 Gbps。

那么,如果两台机器连接到同一块插卡,并且它们同时传送数据,则会怎么样呢? 这

要取决于这块插卡的构造方式。一种可能是,卡上的所有端口都用线连在一起,从而构成了一个卡上局域网。对于这种发生在卡上 LAN 中的冲突,检测和处理办法与 CSMA/CD 网络中的其他冲突是一样的:利用二元指数后退算法进行重传。对于这种类型的插卡,任何时刻每块卡只能有一个传输任务,但是所有的插卡可以并行传输。在这种设计中,每块插卡构成了它自己的冲突域(collision domain),与其他的插卡完全独立。如果每个冲突域只有一个站的话,则冲突不可能发生,因而可以提高性能。

在另一种类型的插卡中,每个输入端口支持缓存功能,所以,进来的帧在到达之后被保存在插卡的 RAM 中。这种设计方案允许所有的输入端口同时接收(和传送)帧,它们可以并行地、全双工地工作,这是单信道的 CSMA/CD 所不可能做到的。一旦一帧已经被完全接收了,于是插卡对它进行检查,看它的目标站是同一插卡上的另一个端口,还是一个远程端口(位于其他的插卡上)。在前一种情况下,该帧可以直接被传送到目的地。而在后一种情况下,必须要通过底板才能将这一帧传送给正确的插卡。在这种设计中,每一个端口是一个独立的冲突域,所以冲突不会发生。总的系统吞吐量通常可以比 10Base-5 提高一个数量级(在 10Base-5 中,整个系统就是一个冲突域)。

由于交换机只要求每个输入端口接收标准的以太网帧,所以,我们有可能将某些端口用作集线器。在图 4.20 中,右上角的端口并不是被连接到一台单独的计算机上,而是一个 12 口的集线器上。当帧到达集线器的时候,它们将会按照通常的方法竞争信道,包括冲突和二元指数后退过程。成功的帧将会到达交换机,在交换机上它们会像其他进来的帧一样被处理:通过高速的底板被交换到正确的输出线路上。集线器比交换机要便宜得多,但是,由于交换机的价格在不断下降,所以,集线器很快将被淘汰。然而,有一些遗留下来的集线器仍然在使用。

#### 4.3.7 快速以太网

早期的时候,10Mbps 简直像是天堂一样,就好像对于早期使用 300bps 声学调制解调器的用户而言,1200bps 也像天堂一样。作为帕金森定律(“做工作的时候总是要把所有可用的时间都用上才能完成”)的一种应用,看起来数据也没有个尽头,总是要把所有可用的传输带宽都用掉才罢休。为了发掘出速度的潜力,各种工业界组织提出了两种新的基于环的光纤 LAN。一种称为 FDDI(Fiber Distributed Data Interface,光纤分布式数据接口),另一种称为光纤信道(Fibre Channel<sup>①</sup>)。我们长话短说,这两种光纤 LAN 都可以用作主干网络,但是它们都不能突破到桌面上。在这两种网络中,站的管理太复杂了,从而也导致了复杂的电路和昂贵的价格。应该从中吸取的教训是 KISS(Keep It Simple, Stupid: 傻瓜,应该尽可能地简单)。

无论如何,光纤 LAN 的失败也为大于 10Mbps 速度的普通以太网留下了发展的空间。许多安装场所需要更多的带宽,因此将大量的 10Mbps LAN 通过各种中继器、网桥、路由器和网关迂回曲折地连接起来,但是,对于网络管理员来说,它们就好像是被泡泡糖和铁丝网粘连在一起了。

<sup>①</sup> 之所以称为“fibre channel”,而不是“fiber channel”,是因为该文档的编辑是一个英国人。

在这种环境下,IEEE 于 1992 年重新召集起 802.3 委员会,要求赶快提出一个快速的 LAN。其中一个建议是,仍然保持 802.3 原来的面貌不变,但是让它运行得更快而已。另一个建议是,完全重做 802.3,给予它更多的特性,比如支持实时的流量和数字化的语音,但是仍保留原来的名称(出于市场考虑的原因)。经过多次争论之后,802.3 委员会决定仍然保留 802.3 原来的工作方式,但是让它运行得更快。按照计算机工业界的习惯做法,在这样的情况下,那些提议失败的人不会甘心,他们组成了自己的委员会,并且对他们提议的 LAN 进行了标准化(最终为 802.12)。不幸的是,他们最终还是失败了。

802.3 之所以决定直接提升原来以太网的速度,是因为以下 3 个主要理由:

- (1) 需要向后兼容现有的以太网 LAN。
- (2) 害怕一个新的协议可能存在一些不可预料的问题。
- (3) 期望在技术发生变化之前就完成这项工作。

这项工作很快就完成了(按照标准委员会的规范),其结果是 **802.3u**,最终于 1995 年 6 月份被正式批准。从技术上讲,802.3u 并不是一个新的标准,而是原有的 802.3 标准的一份补充(因而也加强了它的向后兼容性)。但是在实践中,大家都称它为**快速以太网(fast Ethernet)**,而不是 802.3u,所以,我们也将使用这样的叫法。

快速以太网背后的基本思想非常简单:保留原来的帧格式、接口和过程规则,只是将位时间从 100ns 降低到 10ns。只要将电缆的最大长度降低到十分之一,则照搬 10Base-5 或 10Base-2,仍然可以及时地检测到冲突,从技术来讲这是可能的。然而,由于 10Base-T 的连线方式具有压倒性的优势,所以,快速以太网也完全采纳了这种设计。因此,所有的快速以太网系统也使用集线器和交换机;而不允许使用带插入式分接头或者 BNC 连接器的多支路电缆。

然而,还有其他一些选择仍然要确定下来,其中最重要的是支持什么样的电缆类型。一种观点是 3 类双绞线,其理由是,在实践中,西方国家的每个办公室都有至少 4 组 3 类(或更好的)双绞线,从办公室连接到 100m 以内的电话接线柜。有时候会有两条这样的电缆。因此,若采用 3 类双绞线的话,则无需重新布线就有可能在桌面计算机上使用快速以太网,这对于许多组织来说具有极大的好处。

3 类双绞线的主要缺点是,它不能够在 100m 长的电缆上承载 200M 位带宽的信号(采用曼彻斯特编码的 100Mbps),而针对 10Base-T 规定的从计算机到集线器的最大距离是 100m(参见图 4.13)。相反,5 类双绞线可以很容易地传输 100m,光纤可以走得更远。最后折衷的选择是允许所有这三种可能的电缆,如图 4.21 所示,但是,对于 3 类双绞线的方案,需要增加专门的激励机制以便使它具有额外的承载能力。

名称	线缆	最大的段距离	优点
100Base-T4	双绞线	100m	使用 3 类 UTP
100Base-TX	双绞线	100m	100Mbps 的速率,全双工(5 类 UTP)
100Base-FX	光纤	2000m	100Mbps 的速率,全双工;长距离

图 4.21 最初的快速以太网电缆



3类UTP方案被称为100Base-T4,它使用了25MHz的信令速度,比标准以太网的20MHz仅仅快了25%(记住,图4.16所显示的曼彻斯特编码中,对于每秒10兆位中的每一位都要求两个时钟周期)。然而,为了达到所要求的带宽,100Base-T4要求4对双绞线。由于过去几十年来标准的电话线每根电缆都有4对双绞线,所以,大多数办公室都能够满足这一要求。当然,这也意味着要放弃你的办公室电话,但是,相对于为快速的电子邮件所付出的代价而言,这也不算什么了。

在4对双绞线之中,一对总是给集线器发送信号,一对总是接收集线器的信号,另外两对可以切换到当前的传输方向上。为了达到所要求的带宽,这里并没有使用曼彻斯特编码,在现代时钟和如此短距离的情况下,这种编码已经没有必要了。此外,发送的信号有三态,所以,在单个时钟周期过程中,线上可以包含一个0、1或者2。利用三对双绞线在一个方向上传输,并且用到了三态信号,因而它可以传输27种可能的符号,这使得它发送4位数据还有一定的冗余。在每秒钟的25M个时钟周期中,每个周期都可以传输4位,因而可以达到所要求的100Mbps速率。而且,利用剩下的一对双绞线,反向信道上总是有33.3Mbps。这种方案被称为8B/6T(8个二进制位映射到6个三进制数上),它不太可能因为这种进制转换的设计思想而赢得人们的赞誉,但是在现有的布线环境下,它可以工作得很好。

对于5类双绞线,相应的设计方案100Base-TX要简单得多,因为5类双绞线可以处理125MHz的时钟速率。每个站只用到两对双绞线,一对用于发送信号到集线器,另一对用于从集线器接收信号。这里没有使用直接的二进制编码,相反,它使用了一种被称为4B/5B的编码方案。这种编码方案来自于FDDI,并且与它兼容。5个时钟周期分为一组,其中每一个周期包含了两个信号值之中的一个,这样每一组就有32种组合。在这32个组合中,其中16个被用来传输4位组:0000、0001、0010、...、1111。剩下的16个组合中,有一些被用于控制,比如标记帧的边界。这些用到的组合必须谨慎选取,以便提供足够的电压变化来维护时钟的同步情况。100Base-TX系统是全双工的,接入的站可以在100Mbps速率上发送数据,同时也可以可以在100Mbps上接收数据。通常100Base-TX和100Base-T4合起来称为100Base-T。

最后一种选择方案是100Base-FX,它使用两根多模光纤,每个方向用一根,所以它也是全双工的,每个方向上都是100Mbps的速率。而且,站和集线器之间的距离可以达到2公里。

1997年,作为对于大众化需求的响应,802委员会增加了一种新的电缆类型:100Base-T2,它允许快速以太网可以在两对现有的3类双绞线上运行。然而,它需要一个复杂的数字信号处理器来处理所要求的编码方案,这使得这种选择方案非常昂贵。迄今为止,由于它的复杂性、造价,以及许多办公楼已经用5类UTP重新布线了,所以这种方案很少被实际使用。

对于100Base-T,两种互连设备可能会被用到:集线器和交换机,如图4.20所示。在一个集线器中,所有的输入线(或者起码来说,到达某一块插卡的所有线路)在逻辑上都是连在一起的,从而构成了一个冲突域。所有的标准规则,包括二元指数后退算法,都是适用的,所以,这样的系统仍然像老式的以太网那样工作。特别是,同一时刻只有一个站可



以传输数据。换句话说,集线器要求半双工的通信方式。

在一个交换机中,每一个输入帧都被缓存在一块插入线卡上,然后,如果有必要的话,这些帧被通过一块高速底板从源卡传递到目标卡上。底板没有被标准化,它也不需要标准化,因为它被完全隐藏在交换机的里边。如果过去的经验有任何指导意义的话,交换机厂商将会在商业竞争中生产出更快的底板,从而提高系统的吞吐量。因为对于正常的以太网冲突算法来说,100Base FX 光缆太长了,所以,它们必须要连接到交换机上,从而每一根光缆构成自己的冲突域。因此,不允许使用集线器来连接 100Base-FX。

最后要说明的一点是,几乎所有的交换机都可以处理 10Mbps 和 100Mbps 站的混合组成结构,这样使得网络更新换代变得更为容易。当一个站点获得了越来越多的 100Mbps 工作站的时候,它就有必要购买一定数量的线卡,并将这些线卡插入到交换机中。实际上,标准本身也提供了有关的算法,允许两个站自动协商最优的速度(10 或 100Mbps),以及双工工作模式(半双工或者全双工)。大多数快速以太网产品都利用这种特性来进行自动配置。

#### 4.3.8 千兆以太网

当快速以太网标准的墨迹尚未干透的时候,802 委员会开始着手建立一个更快的以太网(1995)。它很快被赋予千兆以太网(gigabit Ethernet)的称号,1998 年正式被 IEEE 批准,并命名为 802.3z。该名字也意味着,除非有人很快地发明出一个位于 z 之后的新字母,否则千兆以太网就是一个终点了。下面我们将讨论千兆以太网的一些关键特性,关于更多的信息,请参考(Seifert, 1998)。

802.3z 委员会的目标与 802.3u 委员会的目标本质上是相同的:使以太网快上 10 倍,并且仍然与现有的所有以太网标准保持向后兼容。特别是,千兆以太网必须提供无确认的数据报服务,并且支持单播(unicast)和多播(multicast)两种模式;千兆以太网必须使用已有的 48 位编址方案,并且维护同样的帧格式,包括帧的最小和最大长度。最终的标准实现了所有这些目标。

千兆以太网的所有配置都是点到点的,而不像最初的 10Mbps 标准那样是多路分支的,这种多路分支的方式现在已经被称为经典的以太网。在最简单的千兆以太网配置中,如图 4.22(a)所示,两台计算机直接连接起来。然而,更常见的情形是,让一台交换机或者一个集线器连接到多台计算机上,可能还会连接到其他的交换机或者集线器上,如图 4.22(b)所示。在这两种配置中,每根以太网电缆都只有两个设备,不会更多,也不会更少。

千兆以太网支持两种不同的操作模式:全双工模式和半双工模式。“正常”的模式是全双工模式,它允许两个方向上的流量可以同时进行。当有一台中心交换机将周围的计算机(或者其他的交换机)连接起来的时候,就会使用这种全双工的模式。在这种配置下,所有的线路都具有缓存能力,所以每台计算机或者交换机在任何时候都可以自由地发送帧。发送方在发送之前不必检测信道以确定别人是否正在使用信道,因为竞争不可能发生。在一台计算机与交换机之间的连线上,通过这条线路到达交换机惟一可能的发送方是该计算机;即使交换机当前正在给计算机发送数据,计算机的传输操作也会成功(因为



图 4.22

(a) 一个只有两个节点的以太网；(b) 一个包含多个节点的以太网

线路是全双工的)。由于这里不会发生冲突,所以不需要使用 CSMA/CD 协议,因此,电缆的最大长度是由信号强度来决定的,而不是由突发性噪声在最差情况下传回到发送方所需的时间来决定。交换机可以自由地混合和匹配各种速度。就如同快速以太网一样,千兆以太网也支持自动配置特性。

另一种操作模式是半双工模式,当计算机被连接到一个集线器而不是交换机的时候,就会用到这种模式。集线器不会将进来的帧缓存起来,相反,它在内部用电子的方式将所有这些线路连接起来,模拟在经典以太网中所使用的多路分支电缆。在这种模式下,冲突是有可能的,所以要求使用标准的 CSMA/CD 协议。因为一个最小的帧(即 64 字节)现在可以以经典以太网中 100 倍的速度进行传输,所以,最大的距离将减小到 1%,即 25m。这样才能保持以太网的本质特性,也就是,即使在最差的情况下,当突发性噪声回来的时候,发送方仍然在传输数据。对于一根 2500m 长的电缆,发送方在 1Gbps 速率上发送一个 64 字节的帧的时候,在该帧到达另一端的十分之一路程之前它就已经完成了发送任务,更不用提到达另一端的往返时间了。

802.3x 委员会考虑到 25 米长的半径范围是不可接受的,因此在标准中加入了两个特性以使加大此半径范围。第一个特性称为**载荷扩充(carrier extension)**,它的本质是让硬件在普通的帧后面增加一些填充数据,以便将帧的长度扩充到 512 字节。由于这些填充数据是由发送方的硬件加进来,并且由接收方的硬件删除掉,所以,软件对此并不知情,这意味着现有的软件无需任何改变。当然,对于用户数据只有 46 字节(即 64 字节帧的净荷域)的情形来说,使用 512 字节之后,线路的效率只有 9%。

第二个特性称为**帧串(frame bursting)**,它允许发送方将多个帧连接在一起,把它们串起来一起传输出去。如果整个串小于 512 个字节,则硬件会再次对它进行填充。如果有足够的帧在等待传输,则这种方案将会非常高效,应该优先于载荷扩充。由于采用了这些新的特性,所以网络的半径可以扩展到 200 米,对于大多数办公室来说,这样的距离可能足够了。

客观地讲,你很难想象一个组织会特意地购买和安装千兆以太网卡,以求达到高的性能,但是却又将计算机连接到一个集线器上,来模拟全冲突的经典以太网。集线器比交换机要便宜得多,但是,千兆以太网的接口卡相对来说仍然非常昂贵。然而,通过购买廉价

的集线器来降低开销,并因此而大幅地降低新系统的性能,这显然是非常愚蠢的。诚然,在计算机领域中,向后兼容性是至关重要的,所以 802.3z 委员会也要求做到这一点。

千兆以太网既支持铜线,也支持光纤,如图 4.23 所列。在光纤上以 1Gbps 或者接近于 1Gbps 的速率传输信号,这意味着光源必须在 1ns 以内被打开或者关闭。LED 不可能达到这样快的操作速度,所以需要使用激光。有两个波长允许使用:  $0.85\mu\text{m}$ (短)和  $1.3\mu\text{m}$ (长)。 $0.85\mu\text{m}$  的激光非常便宜,但是不能在单模光纤上工作。

名称	线缆	最大的段距离	优点
1000Base-SX	光纤	550m	多模光纤( $50, 62.5\mu\text{m}$ )
1000Base-LX	光纤	5000m	单模( $10\mu\text{m}$ )或者多模( $50, 62.5\mu\text{m}$ )
1000Base-CX	2 对 STP	25m	屏蔽的双绞线
1000Base-T	4 对 UTP	100m	标准的 5 类 UTP

图 4.23 千兆以太网的电缆类型

有三种光纤直径是允许的:  $10, 50$  和  $62.5\mu\text{m}$ 。第一种针对单模光纤,后两种针对多模光纤。然而,并不是所有六种组合都是允许的,并且最大的段距离取决于到底使用了哪一种组合。图 4.23 中给出的数值是最佳情形下的值。特别是,只有当  $1.3\mu\text{m}$  的激光作用在  $10\mu\text{m}$  的单模光纤上的时候才能够达到 5000m 的距离,尽管这种方式是最为昂贵的选择,但是对于校园主干网而言,这是最佳的选择,期望它能够流行起来。

1000Base-CX 选择方案使用了短的屏蔽铜线电缆。它的问题在于,它既要与其上而的高性能光纤竞争,又要与其下面的廉价的 UTP 竞争。不管怎么样,它不太可能会广泛应用。

最后一种选择方案是 4 对 5 类 UTP 联合起来工作。因为这种连线方式在很多地方都已经安装好了,所以,它可能是“穷人”的千兆以太网。

千兆以太网对于光纤使用了新的编码规则。在 1Gbps 速率上的曼彻斯特编码将要求 2G 频带的信号,这太难达到了,而且也太浪费带宽了。相反,千兆以太网选择了一种新的基于光纤信道的编码方案,称为 **8B/10B**。每个 8 位字节在光纤上被编码为 10 位,因此称为 8B/10B。因为对于每一个输入字节共有 1024 种可能的输出码字,所以,在选取可用码字的时候有一定的余地。在选择码字的时候用到了下面两条规则:

- (1) 不允许一个码字中有超过 4 个连续相等的位。
- (2) 不允许一个码字中 0 的个数或者 1 的个数超过 6 个。

之所以这样选择,是为了在数据流中保持足够多的位变化,从而确保接收方与发送方保持同步,同时也保持光纤上 0 的个数和 1 的个数尽可能地相等。而且,许多输入字节有两种可能的码字被分配给它们。当编码器在选择码字的时候,它的选择依据是,尽可能使到目前为止 1 的个数和 0 的个数相等。这种平衡 0 和 1 的个数的做法是有必要的,这样可以保持信号的 DC 分量尽可能低一些,以便它在通过变换器的时候不会有改变。虽然计算机科学家并不喜欢让变换器的特性来指导他们的编码方案,但有时候生活就是这样的。

使用 1000Base-T 的以太网使用了另外一种不同的编码方案,因为要将铜线上的数据计时到 1ns 以内实在太困难了。这种方案使用 4 对 5 类双绞线,以便允许 4 个符号可以并行地被传出去。在编码每个符号的时候用到了 5 级电压值。这种编码方案允许一个符号被编码成 00、01、10、11,或者一个专门用于控制的特殊值。因此,每对双绞线有 2 个数据位,或者每个时钟周期有 8 个数据位。时钟运行在 125MHz 的频率下,所以这种方案允许 1Gbps 的操作速度。至于为什么使用 5 级电压值而不是 4 级电压值,这是为了留一些组合给成帧和控制的用途。

1Gbps 的速度是非常快的,例如,如果一个接收方在 1ms 的时间内忙于其他的某一项任务,因而没有清空某条线路上的输入缓冲区,那么,在这 1ms 的间隔内,最多可以有 1953 帧被累积起来。而且,当千兆以太网上的一台计算机沿着下行线路给另一台位于经典以太网网络中的计算机发送数据的时候,缓冲区溢出极有可能发生。作为这两种情形的一个必然结果是,千兆以太网支持流控制(如同快速以太网一样,但是它们的做法并不相同)。

流控制是这样实现的:一方给另一方发送一个特殊的控制帧,告诉它暂停一段时间。控制帧也是正常的以太网帧,它的类型为 0x8808。数据域的前两个字节给出了命令;如果有参数的话,则后续的字节为参数信息。在流控制过程中用到了 PAUSE 帧,其参数指明了暂停多长时间,所用单位为最小帧时间。对于千兆以太网,该时间单位为 512ns,允许暂停时间长达 33.6ms。

当千兆以太网刚刚被标准化出来以后,802 委员会便又得回去工作。IEEE 告诉他们要启动一个 10G 速率的以太网。由于难以找到位于 z 之后的字母,所以他们放弃了这种做法,改成使用两个字母的后缀。他们继续工作,并且在 2002 年的时候,该标准被 IEEE 批准为 802.3ae。100G 的以太网出现还会有多远呢?

#### 4.3.9 IEEE 802.2: 逻辑链路控制

现在我们也许应该回来比较一下本章中学习的内容以及上一章中学习的内容。在第 3 章中,我们看到了,利用各种数据链路协议,两台机器可以在不可靠的线路上进行可靠的通信。这些协议提供了错误控制(使用确认)和流控制(使用一个滑动窗口)的能力。

相反,在本章中,我们一直没有提及可靠通信。以太网以及其他的 802 协议所提供的是一种尽力投递的数据报服务。有时候,这种服务已经足够了。例如,对于传输 IP 分组而言,这种可靠性保证是不要求的,甚至也是不期望的。一个 IP 分组只是被插入到一个 802 净荷域中,然后发送出去。如果它丢失了,也就弃之不管了。

然而,在其他有些系统中,要求使用一个具有错误控制和流控制特性的数据链路协议。IEEE 定义了这样一个协议,它可以运行在以太网和其他的 802 协议之上。该协议被称为 LLC(Logical Link Control,逻辑链路控制),而且,它通过提供一种统一的格式,以及向网络层提供一个接口,从而隐藏了各种 802 网络之间的差异。此格式、接口和协议基本上都以我们在第 3 章中学习过的 HDLC 协议为基础。LLC 构成了数据链路层的上半层,MAC 子层在它的下边,如图 4.24 所示。

LLC 的典型用法如下所述。发送方机器上的网络层利用 LLC 的访问原语,把一个

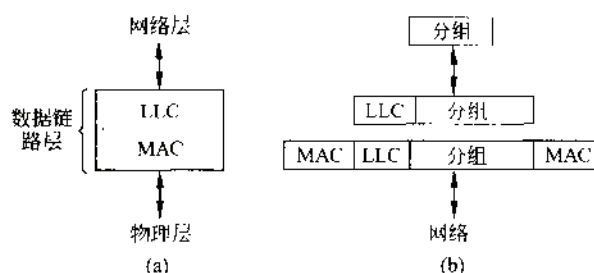


图 4.24

(a) LLC 的位置; (b) 协议格式

分组传递给 LLC。LLC 子层增加一个 LLC 头,其中包含了序列号和确认号。然后,结果得到的结构被插入到 802 帧的净荷域中,并发送出去。在接收方一端,执行相反的过程。

LLC 提供了三种服务选择:不可靠的数据报服务、有确认的数据报服务,以及面向连接的可靠服务。LLC 头包含三个域:一个目标访问点、一个源访问点,以及一个控制域。这两个访问点指明了该帧从哪个进程来,以及要被递交给哪个进程,相当于取代了 DIX 的 type 域。控制域包含了序列号和确认号,与 HDLC 的风格(见图 3.24 所示)非常接近,但是并不相同。当在数据链路层上需要一个可靠的连接的时候,这些域就会被用到;在这种情况下,所使用的协议与第 3 章中讨论的那些协议非常类似。对于 Internet,只要尽最大努力投递 IP 分组就可以了,所以在 LLC 层上并不要求确认。

#### 4.3.10 关于以太网的回顾

以太网已经发展 20 多年了,并且在发展过程中还没有出现真正有实力的竞争者,所以,它可能还会持续发展很多年。对于 CPU 结构、操作系统或者编程语言,很少有哪一个在统治了 20 年之后还能够进入到第 3 个 10 年中。很明显,以太网已经走上了这样的一条正确道路。为什么?

以太网之所以具有如此强大的生命力,最主要的理由可能是它的简单性和灵活性。在实践中,简单性带来了可靠、廉价,以及易于维护等特性。当插入式分接头被 BNC 连接器取代之后,就很少再出现失败了。对于一个长期以来工作得非常出色的事物,人们总是不愿意将它替换掉,尤其是当他们得知了在计算机工业界有许多非常糟糕的事情之后。许多被冠以“升级或者更新”的事物比原来的还要更差一些。

简单性也可以理解为造价低廉。细的以太网和双绞线相对来说是非常便宜的。接口卡的造价也很低。只有当引入了集线器和交换机之后,才会真正要求有一些投资,但是,等到真的需要集线器和交换机的时候,以太网往往已经很好地建立起来了。

以太网非常易于维护。不需要安装特殊的软件(驱动程序除外),也不需要管理配置表。而且,增加新的主机也非常简单,只要将它们接入进来就可以了。

另一点是,利用 TCP/IP,以太网很容易互联,而且, TCP/IP 已经在实践中占了主导地位。IP 是一个无连接的协议,所以它非常适合于以太网,因为以太网也是无连接的。IP 不太适合于 ATM,因为 ATM 是面向连接的。这种不一致性无疑削弱了 ATM 的发



展机会。

最后,以太网已经在多个关键的方面取得了显著的进展。从速率来看,以太网已经提升了几个数量级,从结构来看,集线器和交换机被引入进来,但是,这些变化并不要求软件也跟着发生变化。如果网络销售人员在一个大客户那里展示的时候这样说,“我为你们带来了一种新奇的网络。你们所需要做的事情是,丢弃所有的硬件,并且重写所有的软件”,那么,他就有问题了。FDDI、光纤信道和 ATM 在刚刚引入的时候,其速度都超过了以太网,但是,这些网络与以太网不兼容,并且更加复杂,而且难以管理。最终,以太网赶上了它们的速度,所以,它们不再有任何优势,除了 ATM 已经渗透到电话系统的核心以外,其他几种网络技术都悄然退出舞台。

## 4.4 无线 LAN

尽管以太网已经非常普及了,但是,它仍然会面临竞争。无线 LAN(无线局域网)正在日渐普及,越来越多的办公楼、机场和其他的公共场合配备了无线 LAN。我们在图 1.35 中已经看到,无线 LAN 可以有两种配置方案:有一个基站,或者没有基站。因此,802.11 LAN 标准考虑到了这一点,对这两种方案都提供了支持,稍后我们将会看到。

我们在 1.5.4 小节中给出了有关 802.11 的一些背景信息。现在我们来仔细地看一看这些技术。在本节中,我们将介绍协议栈、物理层无线电传送技术、MAC 子层协议、帧结构,以及服务。有关 802.11 的更多信息,请参考(Crow et al., 1997; Geier, 2002; Heegard et al., 2001; Kapp, 2002; O'Hara and Petrick, 1999; and Severance, 1999)。要想获得最为确凿的技术信息,请参考 802.11 标准本身。

### 4.4.1 802.11 协议栈

包括以太网在内的所有 802 标准,它们所使用的协议在结构上有一个共性。图 4.25 给出了 802.11 协议栈的一个部分视图。其中物理层与 OSI 的物理层对应得非常好,但是,在所有的 802 协议中,数据链路层都被分成了两个或者更多个子层。在 802.11 中,MAC(Medium Access Control,介质访问控制)子层确定了信道的分配方式,也就是说,它决定了下一个该由谁传输数据。在 MAC 子层的上面是 LLC(Logical Link Control,逻辑链路控制)子层,它的任务是隐藏 802 各个标准之间的差异,使得它们对于网络层而言都是一致的。在本章前面讨论以太网的时候我们曾经学习过 LLC,因此这里不再重复这一部分内容。

1997 年,802.11 标准规定了在物理层上允许三种传输技术。红外线方法使用了与电视遥控器相同的技术。其他两种方法使用短距离的无线电波,所用到的技术分别称为 FHSS 和 DSSS。这两种技术都用到了一部分不要求许可的频段(2.4GHz ISM 频段)。无线电控制的垃圾门自动开关也使用了这部分频谱,所以你的笔记本电脑可能会发现它在与你的垃圾门竞争频段。无绳电话和微波炉也使用了该频段。所有这些技术都工作在 1Mbps 或者 2Mbps 的速率上,并且功率非常低,因此一般不太会有严重的冲突。1999 年,两种新的技术被引入进来,以便达到更高的带宽。这两种技术称为 OFDM 和



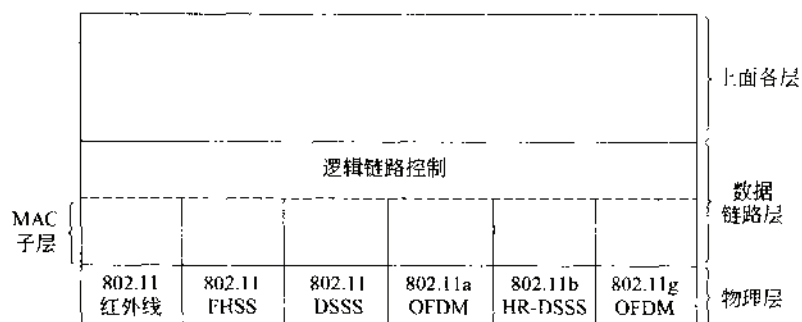


图 4.25 802.11 协议栈的部分视图

HR-DSSS。它们的工作速率分别可以达到 54Mbps 和 11Mbps。2001 年,第二种 OFDM 调制技术又被引入进来,它与第一种 OFDM 调制技术工作在不同的频段上。现在我们将简短地讨论每一种技术。从技术来看,它们都属于物理层,所以应该在第 2 章中讨论,但是由于它们与一般的 LAN 和 802.11 MAC 子层更加接近,所以我们在这里讨论这些技术。

#### 4.4.2 802.11 物理层

在前面提到的 5 种传输技术中,每一种都能够将一个 MAC 帧从一个站发送到另一个站。然而,它们所使用的技术,以及所能够达到的速度却各不相同。有关这些技术的详细讨论超出了本书的范围,但是,下面对每一种技术的简短描述以及所提供的一些关键字,可以让感兴趣的读者在 Internet 或者其他地方搜索到更多的信息。

红外线技术使用了 0.85 或者 0.95 微米波段上的漫射传输。它允许两种速率:1Mbps 和 2Mbps。在 1Mbps 上,它所用的编码方案是这样的:每 4 位成一组,每个组被编码成一个 16 位的码字,其中包含 15 个 0 和一个 1,这种编码称为灰色编码(gray code)。它具有这样的特性:在时间同步中的一个小错误只会导致输出中的一位错误。在 2Mbps 上,所用的编码方案为:取出 2 位,生成一个 4 位的码字,4 位之中也只有一个 1,即 0001、0010、0100、1000 四个码字之一。红外线信号不能够穿透墙壁,所以不同房间中的信元是相互隔离的。然而,由于带宽较低(以及太阳光对红外信号的干扰),这不是一种很通用的选择方案。

FHSS(Frequency Hopping Spread Spectrum,跳频扩频)使用了 79 个信道,每个信道的宽度为 1MHz,从 2.4GHz ISM 频段的低端开始往上。它使用一个伪随机数发生器,来产生跳频序列。只要所有的站中的随机数发生器都使用同样的种子,并且这些站在时间上保持同步,那么,它们将会同时跳到同样的频率上。在每个频率上所花的时间长度称为停延时间(dwell time),这是一个可调整的参数,但是必须小于 400ms。FHSS 的随机性提供了一种很公平的方式来分配无许可限制的 ISM 频段中的频谱。它同时也提供了一定程度的安全性,因为如果入侵者不知道跳频序列或者停延时间的话,他就不可能窃听所传输的信号。在较长的距离上,多径衰减(multipath fading)现象可能是一个问题。FHSS 提供了很好的抵抗能力。并且它对于无线电干扰也相对不敏感,这使得它非常适合用于

建筑物之间的链路上。它的主要缺点是带宽较低。

第三种调制方法为 **DSSS(Direct Sequence Spread Spectrum, 直接序列扩频)**, 它也被限制在 1 或 2Mbps 的速率上。所用的编码方案与 2.6.2 节介绍的 CDMA 系统有一些相似之处, 也有不同之处。它使用了“巴克序列(Barker sequence)”, 每一位在传输的时候需要 11 个时间片。它使用 1M 波特的相移调制, 当在 1Mbps 上工作的时候, 每个波特传输 1 位, 当在 2Mbps 速率上工作的时候每个波特传输 2 位。过去几年来, 在美国, FCC(联邦通信委员会)要求所有的无线通信设备都在 ISM 频段上工作, 并使用扩频技术, 但是在 2002 年 5 月, 由于新技术的出现, 这条规则被废除了。

第一个高速无线 LAN, 即 802.11a, 使用了 **OFDM(Orthogonal Frequency Division Multiplexing, 正交频分多路复用)**, 在更宽的 5GHz ISM 频段中它可以达到 54Mbps。正如其名字中的 FDM 所隐含的, 它用到了不同的频率, 在 52 个频率中, 48 个用于数据, 4 个用于同步, 这与 ADSL 没有什么不同。由于多个传输过程会在多个不同的频率上同时进行, 所以这项技术也可以被看作是一种扩频形式, 但是与 CDMA 和 FHSS 有所不同。将信号分割成许多个窄的频段, 这比直接使用一个宽的频段更有优势, 其中包括: 可以更好地避免窄带干扰, 以及有可能使用非邻接的频段。这种技术用到了一种复杂的编码系统, 基于相移调制, 速度可以达到 18Mbps; 基于 QAM, 速度更高。在 54Mbps 上, 216 个数据位被编码到 288 位的码元中。OFDM 的部分动机是为了与欧洲的 HiperLAN/2 系统(Doufexi et al., 2002)兼容。从每个赫兹的位传输率而言, 这项技术有非常好的频谱效率, 并且, 它对于多径衰减也有很好的抵抗能力。

接下来, 我们看一看 **HR-DSSS(High Rate Direct Sequence Spread Spectrum, 高速率的直接序列扩频)**。这是另一种扩频技术, 它使用每秒 11 兆时间片, 从而在 2.4GHz 频段内达到了 11Mbps。它被称为 802.11b, 但并不是 802.11a 的继续。实际上, 它的标准是首先被通过的, 而且它也是首先进入市场的。802.11b 支持的数据率为 1、2、5.5 和 11Mbps。其中前两种慢速率使用了相移调制方案(为了与 DSSS 兼容), 运行在 1M 波特率上, 每个波特分别为 1 位和 2 位。后两种快速率方案使用了 Walsh/Hadamard 编码, 运行在 1.375M 波特率上, 每个波特分别为 4 位和 8 位。在运行过程中, 数据率有可能会动态地调整, 以便达到当前载荷和噪声条件下的最优可能速度。在实践中, 802.11b 的运行速度几乎总是 11Mbps。尽管 802.11b 比 802.11a 更慢一些, 但是, 它的范围却是后者的 7 倍左右, 这在许多情况下是非常重要的。

802.11g 是 802.11b 的一个增强版本, 它是在解决了专利使用问题之后于 2001 年 11 月由 IEEE 批准的。它使用了 802.11a 的 OFDM 调制方法, 但是运行在 2.4GHz ISM 频段内, 这一点与 802.11b 一样。在理论上, 它的运行速度可以达到 54Mbps。目前还不清楚这个速度在实践中是否可行。它的真正含义是, 802.11 委员会已经产生了三种不同的高速无线 LAN: 802.11a、802.11b 和 802.11g(不算三种低速的无线 LAN)。有人可能会问, 对于一个标准委员会而言, 这是否是一件好事。也许 3 是他们的幸运数字。

#### 4.4.3 802.11 MAC 子层协议

现在我们从电气工程领域转移到计算机科学领域中来。802.11 MAC 子层协议与以

太网的 MAC 子层不同,因为与有线环境相比,无线环境具有一些内在的复杂性。在以太网中,一个站只要等到以太空闲下来,就可以开始传输了。如果在前 64 个字节以内没有收到返回来的噪声尖峰的话,则几乎可以肯定该帧已经被正确地递交了。在无线环境中,这样的条件并不成立。

首先,前面提到的隐藏站问题不可避免,如图 4.26(a)所示。由于并不是所有的站都在其他站的无线电范围以内,所以,在一个单元的某一部分中正在进行的传输也可能不会被同一个单元中的其他地方接收到。在这个例子中,站 C 正在给站 B 发送数据。如果 A 首先监听信道,则它什么也不会听到,从而错误地得出结论:现在可以向 B 发送数据了。

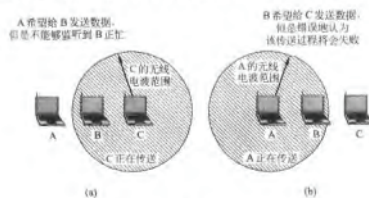


图 4.26  
(a) 隐藏站问题; (b) 暴露站问题

而且,相反的问题也存在,即暴露站的问题,如图 4.26(b)所示。这里 B 希望发送数据给 C,所以它监听信道。当它听到了一次传输的时候,它错误地得出结论:现在不能向 C 发送数据,而实际上, A 可能在向 D(图中没有显示)传输数据。而且,大多数无线电设备都是半双工的,这意味着它们不能够同时在一个频率上既传输数据,又监听噪声尖峰。由于这些问题,因此,802.11 并没有像以太网那样采用 CSMA/CD。

为了解决这些问题,802.11 支持两种操作模式。第一种模式称为 DCF (Distributed Coordination Function, 分布式协调功能),它并没有用到任何中心控制手段(在这一点上,与以太网类似)。另一种模式称为 PCF (Point Coordination Function, 点协调功能),它使用基站来控制单元内的所有活动。所有的 802.11 实现都必须都支持 DCF,而 PCF 则是可选的。现在我们依次来讨论这两种模式。

当使用 DCF 的时候,802.11 使用了一个称为 CSMA/CA (CSMA with Collision Avoidance, 避免冲突的 CSMA) 的协议。在这个协议中,物理信道的监听手段与虚拟信道的监听手段都用到了。CSMA/CA 支持两种操作方法。在第一种方法中,当一个站想要发送数据的时候,它首先监听信道。如果信道空闲,它就开始发送。在发送过程中它并不监听信道,而是直接送出整个帧;对于这种情况,在接收方,有可能由于干扰而使该帧数据被破坏。如果信道正忙,则发送方推迟到信道空闲,然后再开始发送。如果冲突发生的话,则冲突的站等待一段随机的时间,它们使用以太网的二元指数后退算法来计算这段时

间,以后再试着重新传送。

CSMA/CA 操作的另一种模式以 MACAW 为基础,它用到了虚拟信道监听方法,如图 4.27 所示。在这个例子中,A 希望向 B 发送数据。站 C 位于 A 的无线范围内(有可能也在 B 的范围内,但这无关紧要)。站 D 在 B 的无线范围内,但是不在 A 的范围内。

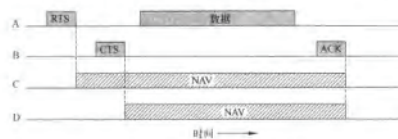


图 4.27 在 CSMA/CA 中,虚拟信道监听用法

当 A 决定要向 B 发送数据的时候,协议开始工作。A 首先向 B 发送一个 RTS 帧,请求一个给 B 发送帧的许可。当 B 收到该请求的时候,它可能会决定给予许可,在这种情况下,它送回一个 CTS 帧。A 收到了 CTS 帧之后,便发送出它的帧,并启动一个 ACK 定时器。B 在正确地收到了该数据帧之后,用一个 ACK 帧作为应答,从而终止协议交换过程。如果在 ACK 帧回到 A 之前,A 的定时器超时了,则整个协议再重新运行。

现在我们从 C 和 D 的角度来考虑这个交换过程。C 在 A 的无线范围以内,所以它可能会收到 RTS 帧。如果它确实收到了,则它意识到,很快有人要发送数据了,所以,为了全局着想,它不再传送任何信息,直到该协议的交换完成为止。根据 RTS 请求中所提供的信息,它可以估算出该序列将需要多长时间,其中包括最终的 ACK,所以它为自己声明了一种虚拟信道,并且该信道正忙,在图 4.27 中用 NAV(Network Allocation Vector, 网络分配向量)标示出来。D 并没有听到 RTS,但是它听到了 CTS,所以,它也声明了 NAV 信号。请注意,NAV 信号并不被传送出去,它们只是一种内部的提醒信号,用来保持一定时间的安静。

与有线网络相反,无线网络是有噪声的,也是不可靠的,譬如说微波炉也使用了不需要许可的 ISM 频段,所以会干扰无线网络。因此,一帧被成功地传出去的概率随着帧长的增加而减小。如果任何一位发生错误的概率为  $p$ ,则  $n$  位长度的帧被完整地正确接收的概率是  $(1-p)^n$ 。例如,对于  $p=10^{-4}$  而言,正确地接收到一个完整的以太网帧(12,144 位)的概率小于 30%。如果  $p=10^{-3}$ ,则 9 帧中大约有一帧将会被损坏。即使  $p=10^{-5}$ ,则超过 1% 的帧将被损坏,算起来,每秒钟将会达到近 10 帧,而且,如果所传输的帧比最大长度短的话,损坏的帧会更多。总而言之,如果一帧太长的话,它要完好无损地传输过去的几率会非常小,这样就可能不得不重传。

为了解决噪声信道的问题,802.11 允许所传输的帧被分成小的碎片,每个分片有自己的校验和。在使用停-等协议进行传输的时候,这些分片被单独编号和确认,也就是说,发送方只有在接收到了第  $k$  个分片的确认之后才传输第  $k+1$  个分片。一旦一个站利用 RTS 和 CTS 获得了信道,则多个分片可以成一排被发送出去,如图 4.28 所示。分片序

列称为一个分片串(fragment burst)。

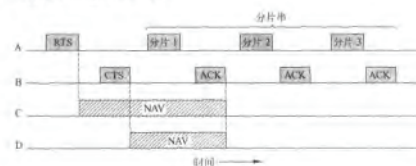


图 4.28 一个分片串

由于分片机制使得只需要重传损坏的分片，而无需重传整个帧，因此，它提高了整体吞吐量。标准并没有规定分片的大小，这是每个单元内部的一个参数，所以，通过基站可以调整该参数。NAV 机制仅仅用于在下一个确认到来之前使其他的站保持安静，然而，下面将要介绍的另一种机制可以允许一个完整的分片串被发送出去而不会受到干扰。

上面所有的讨论都适用于 802.11 DCF 模式。在这种模式中，没有中心控制，所有的站都在竞争时间，就好像以太网中的站所做的那样。另一种模式是 PCF，在这种模式中，基站对其他的站进行表决，问它们是否要发送帧。由于在 PCF 模式中，传输顺序完全是由基站控制的，所以不会发生冲突。标准中规定了表决机制，但是没有规定表决频率，表决顺序，也没有规定是否所有的站都需要获得同等的服务。

基本的机制是，让基站周期性地广播一个信标帧(beacon frame)，差不多每秒 10~100 次。信标帧包含了系统参数，比如调频和停驻时间(针对 FHSS)，时钟同步等。它同时也邀请新的站申请表决服务。一旦一个站已经申请到了特定速率的表决服务，则它实际上获得了一定的带宽保证，从而就有可能具备服务质量保证。

对于移动的无线设备而言，电池的寿命往往是一个问题，所以 802.11 也特别注意到了电源管理的问题。特别是，基站可以指示一个移动站进入睡眠状态，直到由基站或者用户显式地唤醒为止。然而，基站让一个移动站进入睡眠，这意味着在移动站睡眠期间，基站必须负责把所有发送给该移动站的帧全部缓存起来。以后再把它们送给移动站。

在一个单元内，PCF 和 DCF 可以共存。乍看之下，一套中心控制机制和一套分布式控制机制同时运行，看起来似乎是不可能的，但是，802.11 提供了一种办法可以做到这一点。它是通过小心地定义帧间的时间间隔来做到的。当一帧被发送出去以后，对于任何一个站，必须等待一段特定长度的死时间(dead time)之后，它才可以发送帧数据。总共有 4 种不同的时间间隔定义，每一种都有特殊的用途。图 4.29 显示了这 4 种间隔。

最短的时间间隔是 SIFS(Short InterFrame Spacing, 短帧间间隔)。它的用途是，允许一个对话中的各个部分有机会首先被发送。这包括：让接收方发送一个 CTS 来响应一个 RTS；让接收方发送一个 ACK 作为对一个分片的确认，或者对一个完整数据帧的确认；让一个分片串的发送方传输下一个分片，而无需再发送一个 RTS。

在一个 SIFS 间隔之后，总是恰好只有一个站会得到发送应答的授权。如果它未能



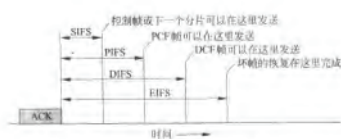


图 4.29 802.11 中的帧间间隔

够利用它的机会,则经过一段时间会到达 PIFS(PCF InterFrame Spacing, PCF 帧间间隔)时间点,于是基站可能会发送一个信标帧,或者一个表决帧。这种机制使得一个正在发送数据帧或者分片序列的站可以在无人妨碍的情况下完成该帧的发送任务,但是,当前面的发送方已经完成了发送任务的时候,基站无需跟机渴的用户竞争就有机会获取信道。

如果基站不想有任何动作,则会到达 DIFS(DCF InterFrame Spacing, DCF 帧间间隔)时间点,任何一个站都可能试图获得信道以便发送一个新的帧。常用的竞争规则在这里仍然适用,如果冲突发生的话,则可能需要用到二元指数后退算法。

最后一个时间间隔是 EIFS(Extended InterFrame Spacing, 扩展帧间间隔),它被用来报告坏帧,只有刚刚接收到坏帧或者未知帧的站才会使用这个间隔。为这种事件赋予最低优先级的想法是,由于接收方可能不知道接下去该怎么办,所以它应该实实在在地等待一段时间,以避免干扰两个站之间的一个正在进行的对话。

#### 4.4.4 802.11 帧结构

802.11 标准定义了三种不同类型的帧用于通信:数据帧、控制帧和管理帧。每一种帧都有一个头,该头部包含了各种用于 MAC 子层的域。除此以外,还有一些头是由物理层使用的,但这些头绝大多数被用来处理所涉及到的调制技术,所以这里我们不讨论这些头。

数据帧的格式如图 4.30 所示。首先是帧控制(Frame Control)域。它本身有 11 个子域,其中第一个子域是协议版本(Protocol Version),由于有了这个版本域,因此在同一个单元内,协议的两个不同版本可以同时工作。接下来是类型域(Type,如数据帧、控制帧或者管理帧)和子类型域(Subtype,如 RTS 或者 CTS)。To DS 和 From DS 域表明了该帧是发送到或者来自于跨单元的分布系统(比如以太网)。MF 域意味着后面跟着还有更多的分片。Retry 域表明了这是以前发送的某一帧的重传。电源管理域(Power management, Pwr)是由基站使用的,基站利用这个域使接收方进入睡眠状态,或者从睡眠状态中唤醒过来。More 域表明发送方还有更多的帧要发送给该接收方。W 位指定了该帧的帧体已经用 WEP(Wired Equivalent Privacy)算法加密过了。最后,0 位告诉接收方,凡是该位已被设置(为 1)的帧序列,必须严格按照顺序来处理。

数据帧的第二个域为持续时间(Duration)域,它提供了这样的信息:该帧和它的确认帧将会占用信道多长时间。该帧也会出现在控制帧中,其他站通过该域来实现 NAV

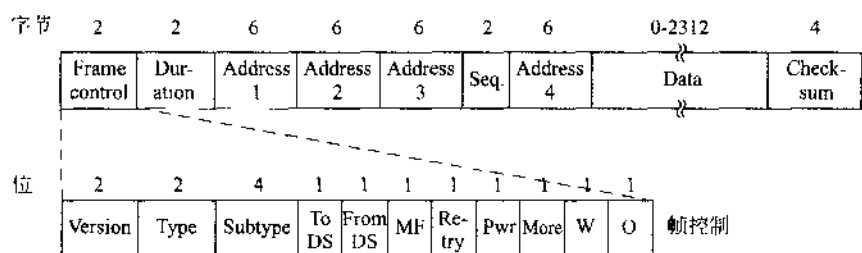


图 4.30 802.11 数据帧

机制。帧头中包含了四个地址,这些地址都是标准的 IEEE 802 格式。很显然,源和目标地址是必不可少的,那么其他两个地址是做什么用的呢?以前曾经提到过,有的帧可能会通过一个基站进入或者离开一个单元。对于跨单元的通信流量,另外两个地址被用于源和目标基站。

序列号(Sequence)域使得分片可以被编号。在 Sequence 域的 16 位中,12 位标识该帧,4 位标识该分片。数据(Data)域包含了净荷,其长度可以达到 2312 个字节,后面跟着常用的校验和(Checksum)域。

管理帧的格式与数据帧的格式非常类似,惟一不同的是,管理帧少了一个基站地址,因为管理帧被严格限定在一个单元中。控制帧也要短一些,它只有一个或者两个地址,没有 Data 域,也没有 Sequence 域。对于控制帧,关键的信息在于 Subtype 域,通常为 RTS、CTS 或者 ACK。

#### 4.4.5 服务

802.11 标准声明了每个符合标准的无线 LAN 必须提供 9 种服务。这些服务可以分成两类:5 种分发服务和 4 种站服务。分发服务涉及到对单元的成员关系的管理,并且会影响到单元之外的站。与之相反,站服务则只与一个单元内部的活动有关系。

5 种分发服务是由基站提供的,它们处理站的移动性:当移动站进入单元的时候,通过这些服务与基站关联起来;当移动站离开单元的时候,通过这些服务与基站断开联系。这 5 种分发服务如下。

(1) **关联(Association)**。移动站利用该服务连接到基站上。典型情况下,当一个移动站进入到一个基站的无线电距离范围之内的时候,这种服务就会被用到。当一个站到来的时候,它就会宣告它的身份和能力。所谓能力,包括所支持的数据率、对 PCF 服务的需要(即表决)和电源管理需求。基站可能会接受,也可能会拒绝该移动站。如果该移动站被基站接受的话,那么它必须证明它自己的身份。

(2) **分离(Disassociation)**。不管是移动站,还是基站,都有可能解除关联关系。一个站在离开或者关闭之前,应该先使用这项服务。但是,基站在停下来进行维护之前也可能用到该服务。

(3) **重新关联(Reassociation)**。利用这项服务,一个站可以改变它的首选基站。这项服务对于那些从一个单元移动到另一个单元中的移动站来说,是非常有用的。如果这项

服务使用正确的话,则在移交过程中不会有数据丢失(但是,802.11 如同以太网一样,也只是一种尽力传递型服务(best-efforts service))。

(4) **分发(Distribution)**。这项服务决定了如何路由那些发送给基站的帧。如果帧的目标对于基站来说是本地的,则该帧将被直接发送到空中。否则的话,它们必须被通过有线网络来转发。

(5) **融合(Integration)**。如果一帧需要通过一个非 802.11 的网络来发送,并且该网络使用了不同的编址方案或者不同的帧格式,则通过这项服务可以将 802.11 格式的帧翻译成目标网络所要求的帧格式。

余下的 4 种服务都是在单元内部进行的(也就是说,只跟一个单元内部的活动有关)。当关联过程完成之后,这些服务才可能会用到。这 4 种服务如下。

(1) **认证(Authentication)**。因为未授权的站很容易就可以发送或者接收无线通信流量,所以,任何一个站必须首先证明了它自己的身份之后才允许发送数据。当基站接受了一个移动站的关联请求(即同意进入它的单元)之后,基站将给它发送一个特殊的质询帧,以确定该移动站是否知道原先分配给它的密钥(口令)。移动站只要加密质询帧,并送回给基站,就可以证明它是知道密钥的。如果结果正确的话,则移动站就被完全接纳到单元中。在初始的标准中,基站不必向移动站证明它自己的身份,但是目前正在修补标准中的这个缺陷。

(2) **解除认证(Deauthentication)**。如果一个原先已经通过认证的移动站要离开网络,则它需要解除认证。在解除认证之后,该移动站可能就不再使用网络了。

(3) **私密性(Privacy)**。如果在无线 LAN 上发送的信息需要保密的话,则它必须被加密。这项服务管理加密和解密。指定的加密算法是 RC4,其发明者是 MIT 的 Ronald Rivest。

(4) **数据投递(Data delivery)**。最后,真正的目的是为了传输数据,所以,802.11 必须提供一种传送和接收数据的方法。由于 802.11 参考了以太网的模型,而以太网的传输过程并不保证 100%可靠,所以,802.11 的传输过程也不保证可靠性。上面的层必须处理检错和纠错工作。

802.11 单元中有一些参数是可以被检测的,在有些情况下还可以被调整。这些参数涉及到加密、超时间隔、数据速率、信标帧周期,等等。

目前,世界各地的办公楼、机场、宾馆、餐馆以及校园等都开始配备了基于 802.11 的无线 LAN。而且,可望还会有快速的增长。要了解 CMU 在广泛部署 802.11 方面的经验,请参考(Hills, 2001)。

## 4.5 宽带无线网络

我们在室内停留的时间太长了。现在让我们走出去,看看外面是否有感兴趣的网络。事实上,外面也有相当多的事情在发生,其中有一些事情跟所谓的“最后一英里(last mile)”有关系。由于在许多国家,电话系统已经被解除管制了,所以,与传统电话公司进行竞争的公司现在通常也允许提供本地的语音和高速 Internet 服务。实际上,这种需求

是大量存在的,问题在于,将光纤、同轴电缆,或者 5 类双绞线铺设到数百万的家庭和商业场所,其代价将会高得惊人。那么,竞争公司该怎么办呢?

答案是宽带无线网。只要在城外的小山上竖立一根大的天线,然后在客户们的屋顶上也安装上天线,并将这些天线指向小山上的天线,这比挖沟铺电缆线要容易得多,也便宜得多。因此,与电信竞争的公司可以提供几兆位速率的无线通信服务,比如语音、Internet、点播电影等;而且在提供这些服务的同时,这些公司也可以获得很好的利益。正如我们在图 2.30 中所看到的,LMDS 正是因为这个目的而发明的。然而,直至最近,各个承运商才设计出他们自己的系统。由于缺乏统一的标准,所以硬件和软件难以大批量生产,从而导致价格很高,用户的接受程度却并不高。

工业界中的许多人意识到,开发一个宽带无线标准是当务之急,所以,IEEE 在这种形势下成立了一个委员会来制定这样的标准,其中的成员来自于一些重要的公司和学术界。在 802 的编号空间中,下一个可以使用的编号为 **802.16**,所以该标准就以此为代号。这项工作是在 1999 年 7 月份启动的,最终的标准于 2002 年 4 月才批准。该标准的官方叫法是“固定宽带无线访问系统的空中接口(Air Interface for Fixed Broadband Wireless Access Systems)”。然而,有些人更喜欢把它称为无线 MAN (Metropolitan Area Network,城域网)或者无线本地回路(wireless local loop)。我们把所有这些术语都看作是等同的,因此可以相互交换使用这些术语。

如同其他某些 802 标准一样,802.16 在多个方面受到了 OSI 模型的严重影响,包括层和子层、术语、服务原语,等等。不幸的是,如同 OSI 一样,它也非常复杂。在本节中,我们将简短地描述一下 802.16 中比较突出的一些方面,请注意,这份介绍并不完整,它忽略了很多细节。关于宽带无线网络的更多一般性信息,请参看 Bolcskei et al., 2001; and Webb, 2001。关于 802.16 的特殊信息,请参考 Eklund et al., 2002。

#### 4.5.1 802.11 和 802.16 的比较

这会儿,你可能会这样想:为什么要设计一个新的标准呢?为什么不直接使用 802.11 呢?有许多很好的理由可以说明为什么不使用 802.11,最主要的是因为 802.11 和 802.16 各自解决的是不同的问题。在讨论 802.16 的技术之前,可能有必要先花一点篇幅来说明一下为什么必须要制定一个新的标准。

802.11 和 802.16 所运行的环境在某些方面是非常类似的,最主要的是,它们的设计目标都是提供高带宽的无线通信。但是,在有些大的方面,它们也各有差异。首先,802.16 针对建筑物提供服务,而建筑物是不会移动的。它们不会经常性地从一个单元迁移到另一个单元中。802.11 中的大部分内容都在处理移动性问题,但是在 802.16 中,这些都变得不再重要了。其次,建筑物中往往会有多台计算机,因而会带来复杂性;如果终端站只是一台笔记本电脑的话,则这种复杂性便不会发生了。由于在通常情况下,建筑物的业主愿意在通信设施上投入的资金,比笔记本电脑的主人所投入的钱要多得多,因而建筑物可以获得更好的无线电信号。这种差异也意味着 802.16 可以使用全双工通信模式,而 802.11 则极力避免这种模式以降低无线电的开销。

因为 802.16 往往会覆盖一个城市的一部分,所以,它允许的距离可能会达到几公里,

这意味着基站的感知功率将有很大的变化范围,来自不同站的信号的功率差异很大。这种差异也会影响到信噪比率,因而需要多种调制方案。而且,由于这是覆盖城市范围的开放通信,因此安全性和私密性将非常重要,必须强制执行。

而且,每个单元中的用户数量可能比一般的 802.11 单元中的用户数量多得多,并且,这些用户比 802.11 中的用户期望使用更多的带宽。毕竟,一个公司不太可能会把 50 个员工集中到一个房间中,然后让他们用笔记本电脑同时观看 50 部不同的电影,以此来检查 802.11 无线网络是否会达到饱和状态。出于这样的理由,802.16 需要更多的带宽,超过了 ISM 频段的宽度,这迫使 802.16 运行在更高的 10~66GHz 的频率范围上,这是尚未使用的频段中惟一仍然还可以使用的范围了。

但是,这些毫米波的波长比 ISM 频段中的波要短一些,并且它们具有不同的物理特性。毫米波的一个特性是可以被水吸收,而且很严重(可能是雨水,但是也可以扩展到雪、冰雹,甚至不幸的话,也可以被浓雾吸收)。因此,与室内环境相比,错误处理显得尤为重要。毫米波可以被聚焦到定向波束中(802.11 是全方向的),所以,802.11 中涉及到多径传播的选择在这里是毫无意义的。

另外一个问题是服务质量,虽然 802.11 针对实时流量提供了某种程度的支持(使用 PCF 模式),但是,它并不是真正为电话和重型的多媒体应用而设计的。相反,802.16 的目标是完全支持所有这些应用,因为这正是个人应用和商业应用的目标。

简而言之,802.11 的设计目标是使之成为移动的以太网,而 802.16 的设计目标则是无线的、但是固定位置的有线电视。这些差异如此之大,以至于制定出来的标准也大相径庭,因为它们分别要为不同的情况进行优化。

802.16 与蜂窝电话系统的比较也值得在这里简短提一下。在介绍移动电话的时候,我们提到了窄带的、面向语音的、低功率的移动站,它们使用中等长度的微波进行通信。迄今为止还没有人在 GSM 移动电话上观看高分辨率的、长达 2 小时的电影。即使 UMTS 也不太有希望改变这种情况。简短而言,无线 MAN 领域中的需求比移动电话领域中的需求要多得多,所以,它需要一个完全不同的系统。在将来,802.16 是否可以被用于移动设备将是一个有趣的问题。目前它并没有为移动设备做优化,但是,这种可能性是存在的。现在,它的焦点集中在固定位置的无线设备上。

#### 4.5.2 802.16 协议栈

802.16 的协议栈如图 4.31 所示。它的总体结构与其他 802 网络非常类似,但是,它的子层更多一些。最底下的子层用于处理传输。对于传统的窄带无线电,往往采用常规的调制方案。在物理传输层之上是一个会聚子层(convergence sublayer),它将不同的技术与数据链路层隔离开。实际上,802.11 也有类似的子层,只不过 802 委员会并没有正式为它使用一个 OSI 类型的名字。

上面的图中没有显示所有的物理层协议,实际上,现在正在增加两种新的物理层协议。802.16a 标准将支持 OFDM,频率范围为 2~11GHz。802.16b 标准将运行在 5GHz 的 ISM 频段中。两者都试图尽可能地靠近 802.11。

数据链路层包含三个子层。底下的子层处理私密性和安全性,这对于公共的户外网



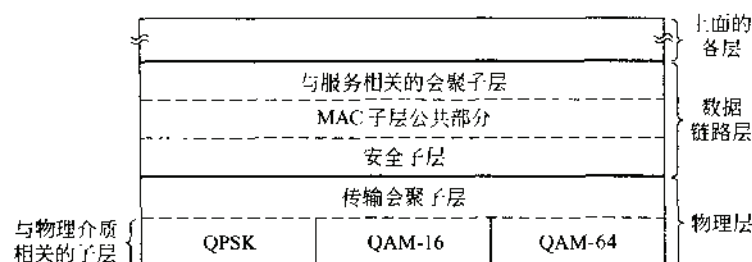


图 4.31 802.16 协议栈

络比私有的室内网络来说要重要得多。这一子层处理加密、解密和密钥管理。

接下来是 MAC 子层公共部分。这是主要协议(比如信道管理)所在之处。802.16 的模型是由基站控制整个系统。它可以调度下行(即从基站到用户)信道,并且在管理上行(即从用户到基站)信道的过程中扮演重要的角色。MAC 子层中一个不同寻常的特性是,为了针对电话和多媒体通信提供服务质量(QoS)保证,MAC 子层完全是面向连接的,这一点与其他的 802 网络截然不同。

与服务相关的会聚子层替代了其他 802 协议中的逻辑链路子层。它的功能是与网络层建立一个接口。这里的复杂之处在于,802.16 的一个设计目标是与数据报协议(比如 PPP、IP 和以太网)和 ATM 进行无缝集成。问题在于,分组协议是无连接的,而 ATM 是面向连接的。这意味着每一个 ATM 连接必须被映射到一个 802.16 连接上,从原理上说这是一件非常直接的事情。但是,一个进来的 IP 分组应该被映射到哪一个 802.16 连接上呢?这个问题正是由这个子层来处理的。

### 4.5.3 802.16 物理层

正如上面所提到的,宽带无线网络需要大段的频谱,惟一可以找得到的位置是在 10~66GHz 的范围内。这些毫米波有一个很有趣的特性,也就是:它们按直线传播;这个特性使它们不像声波,而是类似于光线。同时,这是比它们更长的微波所不具有的特性。此特性也导致了这样的结果:基站可以有多个天线,每个天线指向周边地区的不同扇形区域,如图 4.32 所示。每个扇区有它自己的用户,与相邻的扇区保持相对独立,这是蜂窝单元无线电波所不具备的特性,因为蜂窝无线电波是全方向的。

因为毫米波段信号的强度会随着与基站的距离的增加急剧地衰减,所以信噪比也会随着与基站的距离的增加而下降。由于这样的理由,802.16 采用了三种不同的调制方案,到底使用哪一种取决于用户站离开基站的距离。对于距离比较近的用户,使用 QAM-64,这样可以达到 6 位/波特;对于中等距离的用户,使用 QAM-16,可以达到 4 位/波特;对于距离较远的用户,使用 QPSK,可以达到 2 位/波特。例如,对于典型的 25MHz 的频谱宽度,QAM-64 可以达到 150Mbps,QAM-16 可以达到 100Mbps,QPSK 可以达到 50Mbps。换句话说,用户离开基站越远,数据率越低(这非常类似于我们在图 2.27 中看到的 ADSL 的情形)。这三种调制技术的星座图如图 2.25 所示。

在明确了一个宽带系统的产生目标,以及上述物理限制的影响之后,802.16 的设计

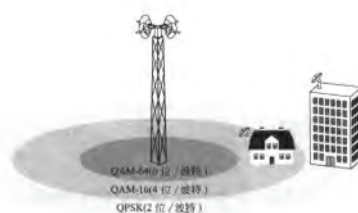


图 4.32 802.16 传输环境

者们便开始努力工作,尽可能有效地使用可用的频谱。他们并不喜欢 GSM 和 DAMPS 的工作方式。在这两个系统中,上行流量和下行流量所使用的频段并不相同,但是宽度相等。对于语音来说,在大多数情况下上行和下行流量可能是对称的,但是对于 Internet 访问而言,通常情况下,下行流量比上行流量要多得多。因此,802.16 提供了一种更为灵活的办法来分配带宽。它使用了两种方案: FDD(Frequency Division Duplexing,频分双工制)和 TDD(Time Division Duplexing,时分双工制),图 4.33 显示了 TDD 的情形。这里,基站周期性地往外发送帧。每一帧包含多个时槽。前面的时槽用于下行流量,然后是一段防护时间,各站利用这段时间切换方向。最后,有一些时槽用于上行流量。每个方向上所用的时槽的数量可以动态改变,以便符合每个方向上流量的带宽情况。

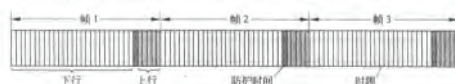


图 4.33 时分双工制中的帧和时槽

基站将下行流量映射到时槽上。因而基站完全控制了方向。上行流量要复杂得多,取决于所要求的服务质量。我们在后面讨论 MAC 子层的时候再来讨论时槽的分配方案。

物理层上另一个有趣的特性是,它能够多个相邻的 MAC 帧包装到一次物理传送过程中。此特性提高了频谱的利用率,因为它降低了前面同步用的信号数量,以及物理层头部的数量。

同时值得一提的是,在物理层中,802.16 使用海明码来完成前向的错误纠正。几乎所有其他的网络都只是简单地利用校验和来检测错误,并且当接收帧发生错误的时候都要求重传。但是,在广域宽带环境中,传输错误可能会频繁发生,因此在物理层上进行错误纠正,而在上面的层上进行校验和检查。采用错误纠正的实际效果是,使得信道看起来

比它本来的状况更好(采用同样的方法,CD-ROM 表现得非常可靠,这仅仅是因为在物理层上有一半以上的信息位被用于错误纠正)。

#### 4.5.4 802.16 MAC 子层协议

数据链路层被分成三个子层,我们在图 4.31 中已经看到过了。由于我们要到第 8 章才学习密码学,所以,现在很难表达清楚安全子层是如何工作的。现在我们只需知道,通过安全子层,所有被传输的数据都经过了加密,从而是保密的。在一个帧中,只有净荷部分被加密,头部并没有加密。这意味着,如果有人监听的话,他可以看到谁跟谁在通话,但是不知道他们相互之间在说什么。

如果你已经对密码学有所了解了,那么,这里有一段关于安全子层的描述。如果你对密码学并不了解,那么你可能会发现下一段内容多少有一点难懂(但是你可能会考虑在学完了第 8 章之后再重读这一段)。

当一个用户连接到基站的时候,用户和基站利用 X.509 证书以及 RSA 公钥密码算法完成双向的认证过程。净荷部分是利用一个对称密码系统来加密的,比如块链接(CBC)的 DES 或者两个密钥的三重 DES。AES(Rijndael)可能很快也会被加入进来。完整性检查可以使用 SHA-1 算法。这还算容易理解,是不是?

现在我们来看一看 MAC 子层公共部分。MAC 帧占用整数倍物理层时槽。每一帧由一些子帧构成,其中前面两个子帧是下行流和上行流的映射图。这些映射图指明了哪个时槽中是什么内容,以及哪些时槽是空闲的。下行流映射图也包含了各种系统参数,以通知那些刚刚上线的新站。

下行信道的分配非常简单、直接。基站简单地确定哪一个子帧中放什么样的内容。上行信道要复杂一些,因为多个用户在没有协调的情况下都要使用上行信道,因而会发生竞争。上行信道的分配方案与服务质量问题有紧密的关系。有 4 类服务,定义如下:

- (1) 位速率为常数的服务;
- (2) 位速率可变的实时服务;
- (3) 位速率可变的非实时服务;
- (4) 尽力投递服务。

802.16 中的所有服务都是面向连接的,每个连接属于上述 4 类服务之一,这是在连接被建立的时候确定的。这种设计与 802.11 或者以太网截然不同,在 802.11 或者以太网中,MAC 子层上不存在连接。

对于位速率为常数的服务,其目标是传输未经压缩的语音信号,比如在 T1 信道上传输语音。这种服务需要在预定的时间间隔中发送预定数量的数据。通过将特定的时槽分配给这种类型的每个连接,系统就可以提供这种服务。一旦带宽已经被分配了,则时槽就可以自动使用了,用户站无需再请求每个时槽。

对于位速率可变的实时服务,其目标是传输压缩的多媒体,或者用于其他一些软的实时应用,在这样的应用中,每个时刻需要的带宽可能会发生变化。802.16 提供这种类型服务的做法是,基站每隔一段固定的时间就选出一个用户,并问它这次需要多少带宽。

对于位速率可变的非实时服务,其目标是非实时的、但是繁重的传输任务,比如大的文件传输。对于这种服务,基站经常性地选出用户,而不是在严格预定的时间间隔中选出用户。一个常数位速率的用户可以在它的帧中设置某一位,来请求基站选中自己,以便发送附加的(位速率可变的)流量。

如果一个站连续  $k$  次不响应基站的选择,则基站将它放到一个多播组中,剥夺它的表决权。相反,当多播组被选中的时候,该组中的任何一个站都可以响应,来竞争服务。通过这种方式,那些流量很少的站不会浪费宝贵的选票。

最后,尽力投递服务可以用于所有其他的应用。这里没有表决的过程,用户必须与其他的尽力投递型用户竞争带宽。对带宽的请求是在这样的时槽中完成的:在上行流映射图中标记为可用于竞争的时槽。如果一个请求成功的话,则该成功信息将被标注在下一个下行流映射图中。如果请求不成功,则不成功的用户必须以后重新请求。为了最低限度地避免冲突,系统也使用了以太网的二元指数后退算法。

802.16 标准定义了两种带宽分配形式:按站分配和按连接分配。在第一种情形中,每个站将建筑物内所有用户的需求集中起来,然后为他们集体进行请求。当它获得了带宽时,它将这份带宽分发给它认为合适的用户。在第二种情形中,基站直接管理每一个连接。

#### 4.5.5 802.16 帧结构

所有的 MAC 帧都以一个通用的头部作为开始。该头部之后跟着一个可选的净荷域和一个可选的校验和(CRC),如图 4.34 所示。在控制帧中净荷域是不需要的,例如,那些请求信道时槽的控制帧不需要净荷域。(令人惊讶的是,)校验和也是可选的,这是因为纠错是在物理层上进行的,并且在实践中,实时帧也不需要考虑重传。如果并没有打算考虑重传的话,那为什么还要烦劳校验和呢?

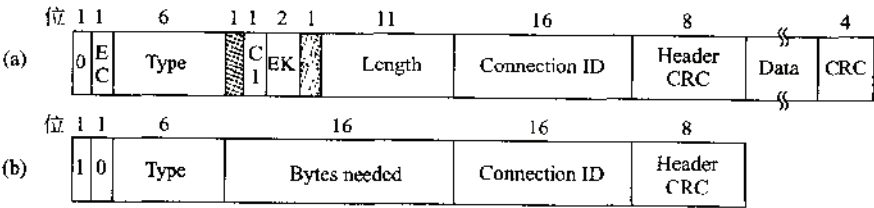


图 4.34

(a) 一个通用的帧结构; (b) 一个带宽请求帧

下面简要地解释一下图 4.34(a)中的头域。EC 位指明了净荷域是否经过了加密。Type(类型)域标识了该帧的类型,通常指明了是否存在包装和分段。CI 域表明最后的校验和是否存在。EK 域指明了所使用的是哪一个加密密钥(如果用到了加密的话)。Length(长度)域给出了该帧的总长度,包括头部。Connection ID 域指明了该帧属于哪一个连接。最后,Header CRC 域是一个仅仅针对头部的校验和,所用的多项式为  $x^8 + x^2 + x + 1$ 。

图 4.34(b)显示了另一种头类型,它被用于那些请求带宽的帧。它的开始处是一个

“1”位,而不是“0”位。它的第2和第3个字节构成了一个16位的数字,该数字指明了需要多少带宽来承载规定数量的字节,除了这部分信息以外,其他域与通用头非常相似。带宽请求帧并不包含净荷,也不包含针对整帧的CRC。

关于802.16还有很多内容要介绍,但这里并非是介绍这些内容所在之处。更多的信息请参考802.16标准本身。

## 4.6 蓝牙技术

1994年,L. M. Ericsson公司对于“在无电缆情况下将它的移动电话和其他设备(比如PDA)连接起来”产生了浓厚的兴趣。它与其他4家公司(IBM、Intel、Nokia和Toshiba)一起组成了一个SIG(Special Interest Group,特别兴趣集团)联盟来开发一个无线标准,用于将计算和通信设备或附加部件通过短距离的、低功耗的、低成本的无线电波相互连接起来。这个项目被命名为**蓝牙(Bluetooth)**,该名字来源于北欧的一个海盗王Harald Blaatand(Bluetooth)II(940~981),他统一(即征服)了丹麦和挪威。

虽然最初的想法只是要去掉设备之间的电缆,但是,很快它便扩大了范畴,开始侵入到无线LAN的领地中去了。尽管这样的转变使得该标准更加有应用价值,但是这也造成了与802.11进行竞争的局面。更加糟糕的是,这两个系统还会相互电场干扰。同时值得注意的是,Hewlett-Packard曾经在几年以前引入了一个红外网络,以便将计算机的外围设备通过无线的方式连接起来,但遗憾的是,这种网络并没有真正大规模地流行起来。

尽管如此,到1999年7月,蓝牙SIG勇敢地发布了一份1500页的规范(1.0版本)。此后不久,正在考虑无线个人域网络(PAN)的IEEE标准组802.15采纳了蓝牙的文档作为它的基础,并开始对它进行修订。这件事情虽然看起来非常怪异,因为802.15在对一项已经有了细致规范,但是尚无兼容实现(还需要进一步调整)的技术进行标准化,然而,历史表明,通过一个像IEEE这样的中立机构来管理一个开放的标准往往有助于一项技术的推广和应用。更为精确地说,我们应该注意到,蓝牙规范是针对整个系统的,从物理层到应用层,面面俱到。IEEE 802.15委员会仅仅对物理层和数据链路层进行了标准化,协议栈中的其他部分并没有纳入到它的规范之中。

尽管IEEE于2002年批准了第一个PAN标准802.15.1,然而,蓝牙SIG仍然在积极地改进它的方案。蓝牙SIG和IEEE的版本并不完全一样,期望在不久的将来它们会汇聚到同一个标准上。

### 4.6.1 蓝牙的体系结构

作为学习蓝牙系统的开始,我们首先快速地浏览一下蓝牙系统所包含的内容以及它所针对的目标。蓝牙系统的基本单元是一个**微微网(piconet)**,微微网包含一个主节点,以及10米距离之内至多7个活动的从节点。在同一个大的房间中可以同时存在多个微网络,它们甚至可以通过一个桥节点连接起来,如图4.35所示。一组相互连接的微微网称为一个**分散网(scatternet)**。



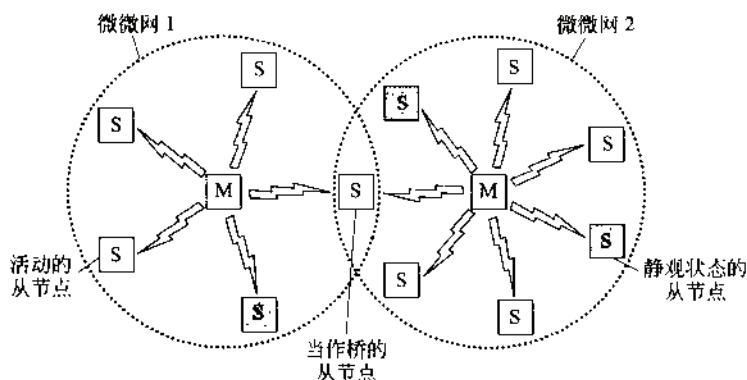


图 4.35 两个微微网连接起来构成一个分散网

在一个微微网中,除了 7 个活动的从节点以外,还可以有多达 255 个静观节点 (parked node)。所谓静观节点是指这样的设备,主节点已经将它们切换到一种低功耗状态,以便降低它们的电源消耗。一个处于静观状态的设备,除了响应主节点的激活或者指示信号以外,不做其他任何事情。在活动状态和静观状态之间,还有两种中间电源状态:限制(hold)和监听(sniff)状态,但是这两种状态不是我们这里所关心的。

这种主/从模式的设计理由是,设计者期望将一个完整的蓝牙芯片的实现代价降低到 5 美元以下。其直接后果是,从设备基本上都是一些哑设备,只能完成一些主节点告诉它们该做的事情。实际上,微微网是一个中心化的 TDM 系统,主节点控制了时钟,它决定了每个时槽中哪个设备可以进行通信。所有的通信都是在主节点和从节点之间进行的,从节点与从节点之间的直接通信是不可能的。

#### 4.6.2 蓝牙应用

大多数网络协议只是为通信个体提供信道,至于用这些协议来做哪些事情,这是应用设计者们应该考虑的问题。例如,802.11 并没有规定用户应该使用他们的笔记本电脑来收发邮件、浏览 Web 页面,或者别的事情。与此相反,蓝牙 V1.1 规范列出了 13 种需要支持的专门应用,并且为每一种应用提供了不同的协议栈。不幸的是,这种做法导致了极大的复杂性,这里我们不对这些协议栈进行讨论,仅仅在图 4.36 中列出这 13 种应用。这里的应用也被称为应用轮廓(profile)。通过粗略地看一看这些轮廓,我们会更加清楚地了解到蓝牙 SIG 努力要达到的目标。

第一个应用轮廓是一般的访问,它实际上并不是一个应用,而是其他实际应用赖以存在的基础。它的主要任务是为主从节点之间建立和维护安全的链路(信道)提供一种方法。另一个相对比较通用的应用是服务发现轮廓,蓝牙设备利用该服务可以发现其他设备提供哪些服务。所有的蓝牙设备都应该实现这两个轮廓。其余的轮廓都是可选的。

串行端口轮廓是一个传输协议,其他大多数轮廓会用到该协议。串行端口轮廓模拟一条串行线,对于那些老的使用串行线的应用程序,该轮廓特别有用。

英文名称	中文名称	说明
Generic access	一般访问	针对链路管理的过程
Service discovery	服务发现	用于发现所提供的服务的协议
Serial port	串行端口	用于代替串行端口电缆
Generic object exchange	一般的对象交换	为对象移动过程定义“客户-服务器”关系
LAN access	LAN 访问	在移动计算机和固定 LAN 之间的协议
Dial-up networking	拨号联网	允许一台笔记本电脑通过移动电话进行呼叫
Fax	传真	允许一台移动的传真机与一部移动电话进行通话
Cordless telephony	无绳电话	将一个手持话机与本地的基站连接起来
Intercom	内部通信联络系统	数字步话机
Headset	头戴通话	允许免提的话音通信
Object push	对象推送	提供一种交换简单对象的方法
File transfer	文件传送	提供一种更为通用的文件传送设施
Synchronization	同步	允许一个 PDA 与另一台计算机进行数据同步

图 4.36 蓝牙应用轮廓

一般的对象交换轮廓为数据移动操作定义了客户-服务器关系。客户发起这样的数据移动操作,然而,从设备既可以是客户,也可以是服务器。如同串行端口轮廓一样,它也是其他一些轮廓的基础(积木块)。

接下来的三个轮廓都是针对网络连接的。LAN 访问轮廓使得一个蓝牙设备可以连接到一个固定网络中,它直接与 802.11 竞争。拨号联网轮廓是蓝牙整个工程的最初动机。它使得一台笔记本电脑可以连接到一部移动电话上(该电话包含一个内置的无线调制解调器)。传真轮廓与拨号联网轮廓类似,只不过,它允许无线传真机利用移动电话来发送和接收传真,两者之间无需连线。

接下来的三个轮廓都是针对电话的。无绳电话轮廓提供了一种办法将无绳电话的手持机与基站连接起来。目前,大多数无绳电话不能当作移动电话来使用,但是在将来,无绳电话和移动电话可能会合并起来。内部通信联络系统轮廓使得两部电话就像步话机那样连接起来。最后,头戴通话轮廓为头戴设备和基站之间提供了免提话音通信的能力,例如,在驾驶汽车的时候就是一部免提电话。

剩下的三个轮廓都被用于在两个无线设备之间实际交换对象。这些对象可以是名片、图片或者数据文件。尤其,同步轮廓的主要用途是,当一个 PDA 或者笔记本电脑离开家的时候,通过同步轮廓可以将数据装到 PDA 或者笔记本中,而当它回到家里的时候,又可以将数据收集回来。

真的有必要分清楚所有这些应用的细节,并且为每一种应用提供不同的协议栈吗?也许并没有这个必要,但是,由于存在许多个不同的工作组,他们分别在设计该标准的不同部分,因此,每个工作组都关注它自己特定的问题,从而形成了它自己的轮廓。你可以

把这看成是 Conway 法则在起作用(在 1968 年 4 月份的 Datamation 杂志上, Melvin Conway 评述说, 如果你安排  $n$  个人编写一个编译器, 那么, 你将会得到一个  $n$  步的编译器, 或者更一般地, 软件结构反映出了编写该软件的小组的结构)。或许蓝牙标准根本不用 13 个协议栈, 两个就可以了, 一个用于文件传输, 另一个用于流式的实时通信。

### 4.6.3 蓝牙协议栈

蓝牙标准有许多协议, 它们按照松散的方式被组织到各个层中。层的结构并不遵从 OSI 模型, 或 TCP/IP 模型, 或 802 模型, 或其他任何已知的模型。然而, IEEE 正在修订蓝牙标准, 以便强行将它纳入到 802 模型中。经过 802 委员会修改之后的蓝牙基本协议结构如图 4.37 所示。

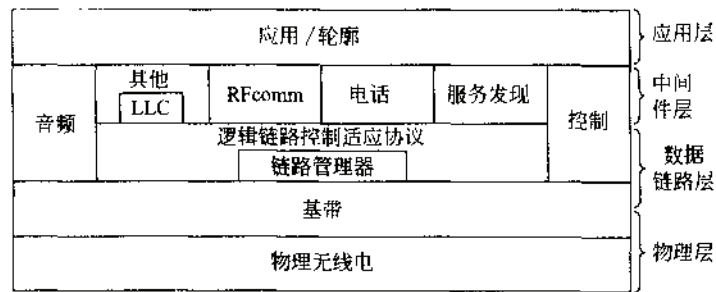


图 4.37 802.15 版本的蓝牙协议结构

最底层是物理无线电层, 它很好地对应了 OSI 和 802 模型中的物理层。该层处理与无线电传送和调制有关的问题。在这一层上的许多考虑都涉及到如何使系统的造价更加低廉, 以便形成巨大的市场。

基带层有点类似于 MAC 子层, 但是也包含了物理层的要素。它涉及到主节点如何控制时槽, 以及这些时槽如何组织成帧。

接下来一层是一组多少有些相关的协议。链路管理器负责在设备之间建立逻辑信道, 包括电源管理、认证和服务质量。逻辑链路控制适应协议(logical link control adaptation protocol, 通常称为 L2CAP)为上面各层屏蔽了传输细节。它类似于标准的 802 LLC 子层, 但从技术上讲有所不同。对于音频协议和控制协议, 正如其名字所表示的, 它们分别处理与音频和控制相关的事宜。上层应用可以直接操纵这两个协议, 而不必通过 L2CAP 协议。

再往上一层是中间件层, 它是由许多不同的协议混合而成的。为了与 802 的其他网络保持兼容, IEEE 将 802 LLC 插在这里。RFcomm、电话和服务发现协议都是这里的专门协议。RFcomm(Radio Frequency communication, 无线电频率通信, 或者射频通信)是指模拟 PC 上用于连接键盘、鼠标、调制解调器, 及其他设备的标准串口通信。它的设计意图是允许传统的设备更加容易地使用它。电话协议是一个实时协议, 用于三种面向话音的轮廓。它也管理呼叫的建立和终止。最后, 服务发现协议可用来找到网络内的服务。

最上层是应用和轮廓所在的地方。它们利用低层上的协议来完成它们的任务。每个

应用都有它自己的专用协议子集。特定的设备,比如头戴设备,通常只包含其应用所需要的那些协议,而不会包含其他的协议。

在下面的小节中,我们将介绍蓝牙协议栈中最低的三个层,因为这三层大致上对应于物理层和 MAC 子层。

#### 4.6.4 蓝牙无线电层

无线电层将位信息从主节点移动到从节点,或者反过来。它是一个低功率的系统,距离范围为 10 米,运行在 2.4GHz ISM 频段上。该频段被分成 79 个信道,每个 1MHz。采用频移调制方案,每 Hz 1 位,所以总数据率为 1Mbps,但是,这段频谱中有相当一部分被消耗在各种开销上。为了公平地分配信道,蓝牙使用了跳频扩频技术,每秒 1600 跳,停顿时间为  $625\mu\text{s}$ 。在一个微微网中的所有节点同步跳频,主节点规定了跳频的序列。

因为 802.11 和蓝牙都运行在 2.4GHz ISM 频段上,而且在同样的 79 个信道上,所以它们会相互干扰。由于蓝牙跳频的速度比 802.11 快,所以,蓝牙设备破坏 802.11 的传输过程的可能性更大一些。由于 802.11 和 802.15 都是 IEEE 的标准,所以,针对这个问题,IEEE 正在寻找解决方案,但是,由于这两个系统都考虑到 ISM 频段不需要许可,因而都使用了 ISM 频段,所以很难找到好的解决办法。802.11a 标准使用了另外的 ISM 频段(5GHz),但是它的距离范围比 802.11b 要短很多(由于无线电波的物理特性),所以,使用 802.11a 标准并不是针对所有情形都适合的完美解决方案。有些公司解决这个问题的做法是禁止蓝牙设备。一个符合市场规则的方案是,力量较强(这里的力量不是指功率,而是指政治和经济)的一方,要求力量较弱的一方修改它的标准以避免与前者干扰。关于这个问题的一些想法,请参考(Landford et al., 2001)。

#### 4.6.5 蓝牙基带层

基带层是蓝牙标准中最接近 MAC 子层的地方。它将原始的位流转变成帧,并且定义了一些关键的格式。在最简单的形式中,每个微微网的主节点定义一个时槽序列,每个时槽的间隔为  $625\mu\text{s}$ ,主节点的传输过程从偶数时槽开始,从节点的传输过程从奇数时槽开始。这是传统的时分多路复用的做法,主节点拥有一半时槽,而所有的从节点共享另一半时槽。帧的长度可以为 1、3 或者 5 个时槽。

在跳频分时机制中,每一跳有一个大约  $250\sim 260\mu\text{s}$  的停顿时间,这样才能使无线电路变得稳定。停顿时间再短一些也是有可能的,但是需要更高的造价。对于一个单时槽的帧来说,在停顿之后,625 位中的 366 位被留下来了。在这 366 位之中,其中 126 位是一个访问码和头部,余下 240 位才是数据。当 5 个时槽被串到一起的时候,只需要一个停顿周期就够了,而且所用的停顿周期可以稍短一些,所以,在 5 个时槽中,共有  $5\times 625=3125$  位,其中 2781 位可用于基带层。因此,越长的帧比单时槽的帧,利用率越高。

每一帧都是在一个逻辑信道上进行传输的,该逻辑信道位于主节点与某一个从节点之间,称为链路(link)。蓝牙标准中共有两种链路。第一种是 ACL 链路(Asynchronous Connection-Less,异步无连接链路),它用于那些无时间规律的分组交换数据。在发送方,这些数据来自于 L2CAP 层;在接收方,这些数据被递交给 L2CAP 层。ACL 流量的投递

模型为尽力投递(best-effort)型,它的投递没有任何保证。帧可能会丢失,也可能需要重传。对于一个从节点,它与主节点之间只可以有一条 ACL 链路。

另一种链路是 **SCO 链路(Synchronous Connection Oriented,面向连接的同步链路)**,它主要用于实时数据,比如电话连接。这种信道是在每个方向中的固定时槽中分配的。由于 SCO 链路的实时性本质,在这种链路上发送的帧永远不会被重传。相反,通过前向纠错机制可以提供高的可靠性。一个从节点与它的主节点之间可以有多达 3 条 SCO 链路。每条 SCO 链路可以传送一个 64 000bps 的 PCM 音频信道。

#### 4.6.6 蓝牙 L2CAP 层

L2CAP 层有三个主要的功能。第一,它接受来自上面各层的分组,分组可以达到 64KB 大小,并且 L2CAP 层将这些分组打碎到帧中,以便于传输。在远端,这些被打碎的帧又被重组到分组中。

第二,L2CAP 层处理多个分组源的多路复用和解复用。当一个分组已经被重组起来的时候,L2CAP 层决定由哪一个上层协议来处理它,例如,由 RFcomm 或者电话协议来处理该分组。

第三,L2CAP 处理与服务质量有关的需求,其中包括在建立链路时的需求,也包括在常规操作过程中的服务质量需求。而且,在建立链路时,还需要协商最大可允许的净荷长度,这样可以避免一个大分组的设备淹没一个小分组的设备。这个特性是必要的,因为并不是所有的设备都能够处理 64KB 的最大分组。

#### 4.6.7 蓝牙的帧结构

帧格式有几种,其中最重要的格式如图 4.38 所示。帧的开头是一个访问码,它通常标识了主节点,所以,当从节点位于两个主节点的无线电覆盖范围内的时候,它可以利用这个访问码(access code)来区分这一帧属于哪一个主节点。接下来是一个 54 位的头(header),其中包含了典型的 MAC 子层的域。然后是数据域,最多可以达到 2744 位(对于一个 5 时槽的传输而言)。对于一个单时槽的帧,除了数据域为 240 位以外,格式的其他方面都一样。

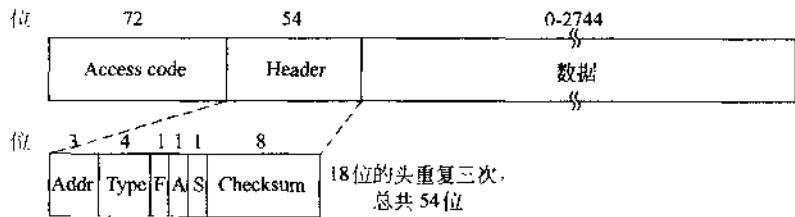


图 4.38 一个典型的蓝牙数据帧

现在我们来快速地看一下头部中的内容。Addr(地址)域标识了该帧的接收目标是 8 个活动设备中的哪一个。Type(类型)域标识了帧的类型(ACL、SCO、表决或者空)、数据域中所使用的纠错类型,以及该帧有多少个时槽长度。F(Flow,流)位是由从节点使用



的,当它的缓冲区满了因而不能再接收任何数据的时候,从节点会利用 F 位来声明这个事实。这是一种基本的流控制形式。A(Acknowledgement,确认)位被用来在一帧中捎带一个 ACK。S(Sequence,序列)位被用于帧的编号,以便检测重传帧。由于使用了停等协议,所以 1 位就够了。然后是 8 位的头部校验和 Checksum。这 18 位的头部重复三次,构成 54 位的头,如图 4.38 所示。在接收方,通过一个简单的电路可以检查每一位的三份副本。如果三份副本都相同,则该位可被接受。如果不相同,则少数服从多数,看哪种选择多就赢。因此,54 位的传输能力被用于发送 10 位的头。这种做法的原因在于,为了在一个噪声环境中利用廉价的、计算能力较弱的低功耗(2.5mW)设备可靠地发送数据,大量的冗余是有必要的。

对于 ACL 帧,数据域中用到了多种格式。然而 SCO 帧要简单得多:数据域总是 240 位。共有 3 个变种,分别允许 80、160 或者 240 位实际净荷,余下的位用于纠错。在最可靠的版本(80 位净荷)中,内容部分被重复三次,与头部的处理方法相同。

由于从节点可能只使用偶数的时槽,所以它每秒可以得到 800 个时槽,就如同主节点一样。若每个时槽有 80 位的净荷,则从节点的信道能力是 64 000bps,主节点的信道能力也是 64 000bps,这对于单个全双工的 PCM 话音信道来说已经足够了(这也正是为什么选择每秒 1600 跳变化率的原因)。这些数字也意味着,尽管微微网有 1Mbps 的原始带宽,但是一个使用最可靠格式的全双工话音信道(每个方向 64 000bps)会使得微微网彻底饱和。对于最不可靠的格式(每个时槽 240 位,没有任何冗余),可以同时支持三个全双工话音信道,这也是为什么每个从节点最多只允许 3 个 SCO 链路的原因。

关于蓝牙技术还有很多内容值得介绍,但是限于篇幅只能到此为止了。有关更多的信息,请参考(Bhagwat, 2001; Bisdikian, 2001; Bray and Sturman, 2002; Haartsen, 2000; Johansson et al., 2001; Miller and Bisdikian, 2001; and Sairam et al., 2002)。

## 4.7 数据链路层交换

许多组织有多个 LAN,它们往往希望将这些 LAN 连接起来。通过一种称为网桥(bridge)的设备,多个 LAN 可以连接起来。网桥运行在数据链路层上,它们通过查看数据链路层的地址来完成帧转发的任务。由于它们不应该检查所转发的帧的净荷域,所以它们可以传输 IPv4(现在的 Internet 中使用的分组)、IPv6(将来的 Internet 中的分组)、AppleTalk、ATM、OSI 或者任何其他类型的分组。相反,路由器要检查分组中的地址,并根据此地址进行路由。虽然看起来好像网桥和路由器之间有这样明显的区别,但是有些现代的技术发展,比如交换式以太网的出现,又弄混了这潭清水,后面我们将会看到这一点。在本节中,我们将会讨论网桥和交换机,特别是它们用于连接不同的 802 LAN 的情形。要想全面地了解网桥、交换机以及相关的话题,请参考(Perlman, 2000)。

在讨论网桥的技术之前,有必要先看一些适合使用网桥的常见情形。我们将列举出六个理由来说明为什么一个组织不应该再维持多个 LAN 的局面。

第一,许多大学和公司的部门都有自己的 LAN,这些 LAN 主要将部门内部的个人计算机、工作站和服务器连接起来。由于各个部门的目标不同,所以,不同的部门选择不

同的 LAN,往往不会顾及其他部门所做的事情。但迟早部门之间要相互沟通,所以就会需要用到网桥。在这个例子中,多个 LAN 之所以存在的原因是各个部门需要自己管理内部的网络。

第二,一个组织可能分布在几个大楼,这些楼之间有一定的地理距离。在每一个楼内有一个独立的 LAN,然后通过网桥和光纤链路将这些 LAN 连接起来,这种做法比起在整个地理范围内运行一个网络(用一条电缆或者光缆连接起来)要经济实惠得多。

第三,有时候可能有必要将一个逻辑上的单个 LAN 分成多个独立的 LAN 以便适应网络的载荷。例如,在许多大学中,需要几千台工作站供学生和教师使用。通常文件被保存在文件服务器上,然后在需要的时候被下载到用户的机器上。这个系统的规模很大,因而不适合把所有的 workstation 都放在一个 LAN 中——所需要的总带宽太高了。相反,可以通过网桥将多个 LAN 连接起来,如图 4.39 所示。每个 LAN 包含一组工作站,并且有它自己的文件服务器,所以,绝大多数流量被限制在单个 LAN 之中,从而不会给骨干网增加载荷。

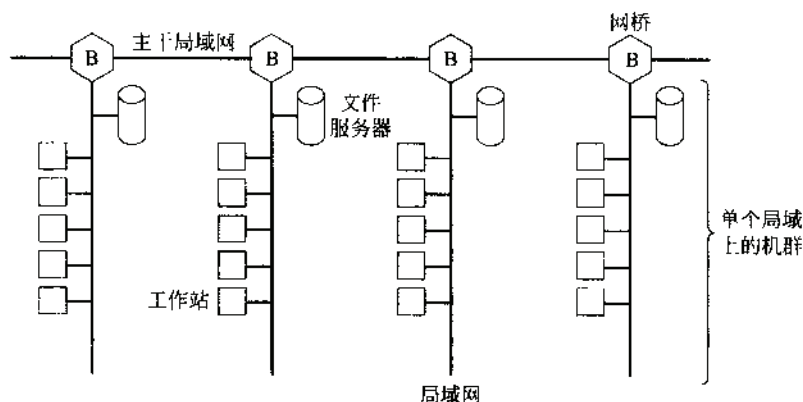


图 4.39 通过一个骨干网将多个 LAN 连接起来,可以处理比单个 LAN 的容量高得多的载荷

值得提及的是,虽然我们在画 LAN 的时候,通常用多分支的电缆来表示,如图 4.39 中那样(典型的画法),但是,现在我们往往使用集线器(hub)或者交换机来实现 LAN。然而,将多台机器直接接在一根长长的多分支电缆上,这种做法与将机器连接到一个集线器上的做法在功能上是等价的。在这两种做法中,所有的机器都属于同一个冲突域,它们都要使用 CSMA/CD 协议来发送帧。然而,交换式的 LAN 有所不同,以前我们曾经看到过,稍后还会看到这种 LAN。

第四,在有些情况下,从载荷的角度而言,一个 LAN 足够了,但是,相距最远的机器之间的物理距离太大了(比如说超过了以太网的限制 2.5 公里)。即使铺设电缆非常容易,这样的网络也无法正常工作,因为往返时延太长。惟一的解决方案是将 LAN 分成多个段,段之间用网桥连接起来。因此,使用了网桥之后,网络所能覆盖的总物理距离可以成倍增加。

第五,还存在可靠性问题。在一个单独的 LAN 上,如果一个有缺陷的节点持续不断地往外发送垃圾数据流,则它可能会明显地削弱 LAN 的能力。我们可以在一些关键的

地方插入一些网桥,就好像大楼中的防火门一样,这样可以避免一个失控的节点降低整个系统的性能。网桥与中继器(repeater)不一样,中继器的功能是复制所有它所看到的数据,而网桥呢,我们可以对它编程,使它能区别对待那些该转发的和不该转发的数据。

第六,也是最后一点,网桥可以提升一个组织的安全性。绝大多数的 LAN 接口都有一种混杂模式(promiscuous mode),在这种模式下,所有的帧都被送给计算机,而不是只将目标地址指向该计算机的帧才送给它。间谍和爱管闲事的人非常喜欢这个特性。系统管理员可以在各个地方插入网桥,并且小心地对网桥进行配置,让它不要转发那些敏感的数据,这样做可以隔离网络的各个部分,从而每部分的流量不会逃离出去并落入坏人之手。

理想情况下,网桥应该是完全透明的,这意味着我们可以将一台机器从电缆的一段移动到另一段上,而无需改变任何硬件、软件和配置表。而且,任何一段上的机器也应该可以与另一段上的机器进行通信,并且无需关心这两段所使用的 LAN 的类型是什么,也不用关心这两段中间的那些段上的 LAN 的类型。这个目标有时候可以达到,但并不总能达到。

#### 4.7.1 从 802.x 到 802.y 的网桥

我们已经明白了为什么需要网桥,现在来看一看网桥是如何工作的。图 4.40 演示了一个简单的两端口网桥的操作过程。主机 A 位于一个无线 LAN(802.11)之中,它有一个分组要发送给主机 B,而主机 B 位于一个固定的以太网(802.3)中,无线 LAN 通过网桥连接到该以太网。待发送的分组被传递到 LLC 子层中,并获得一个 LLC 头(在图中用黑色表示)。然后它被传递到 MAC 子层中,前面又附加上一个 802.11 头(同时还应该有一个尾部,图中没有显示出来)。整个分组单元被通过无线电波发送出去,基站将它接收下来,发现它需要被转送给固定的以太网。当它到达连接 802.11 网络和 802.3 网络的网桥的时候,它又从物理层开始往上传送。在网桥的 MAC 子层中,802.11 头被剥掉,剥掉 802.11 头之后的分组(仍然带有 LLC 头)然后被递交给网桥的 LLC 子层。在这个例子中,分组的目的地是一个 802.3 LAN,所以在网桥的 802.3 一侧,它又往下传递,最终到达以太网。请注意,连接 k 个不同 LAN 的网桥将需要 k 个不同的 MAC 子层和 k 个不同的物理层,每一种 LAN 各需要一个。

现在看来,将一帧从一个 LAN 传送到另一个 LAN 是非常容易的事情。实际上并不是这样。在这一小节中,我们将看到,为不同的 802 LAN(和 MAN)建立网桥的时候将会碰到许多困难。我们把焦点集中在 802.3、802.11 和 802.16 上,但是,其他的网络之间的连接也会有一些问题,可以说,每一种网络都有它自己独特的困难。

首先,每一种 LAN 使用了不同的帧格式(见图 4.41)。以太网、令牌总线和令牌环网之间的差异是由于历史和大公司利益的原因而造成的,但是与此不同的是,这里的差异在某种程度上是合法的。例如,802.11 中的 Duration(持续时间)域(参见 4.4.4 节)是由于 MACAW 协议的原因,它在以太网中是没有意义的。因此,在不同 LAN 之间的复制操作需要重新填充格式,这将会占用 CPU 时间,并且还需要计算新的校验和,而且,有可能由于网桥的内存中的数据位错误而导致在转发过程中引入无法检测的错误。

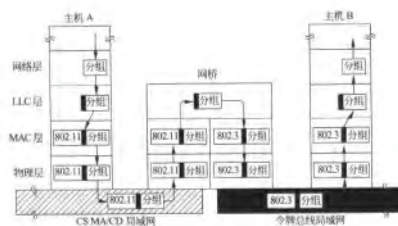


图 4.40 从 802.11 到 802.3 的 LAN 网桥的操作过程

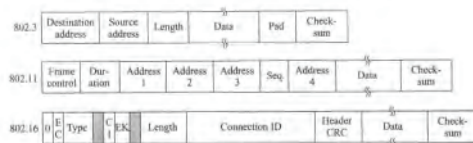


图 4.41 IEEE 802 帧格式(未按比例绘制)

第二个问题是,相互连接的 LAN 之间并不一定工作在相同的数据速率上。当一长串帧持续地从一个快速的 LAN 被传送到一个慢速的 LAN 中的时候,网桥不可能按照接收帧的速度来处理这些帧。例如,如果一个千兆以太网正在以最快的速度往一个 11Mbps 802.11b LAN 发送数据,那么网桥必须将这些帧缓存起来,还得尽快发送出去以防耗尽内存。如果一个网桥连接了三个或者更多个 LAN,那么即使所有的 LAN 都运行在同样的速度上,如果多个 LAN 正在试图往同一个 LAN 输出数据,则同样的问题也存在。

第三个问题可能也是最严重的问题,那就是,不同的 802 LAN 有不同的最大帧长度限制。如果一个较长的帧必须被转发到某一个 LAN,但是该 LAN 又无法接受这么长的帧,那么,这时显然就会出现。将长帧分割成多片,这并不是这一层上该解决的问题,这一层上所有的协议都假定要么帧到达,要么帧没有到达,而没有提供将小的分片重组为帧的功能。这并不是说这样的协议无法设计。设计这样的协议是没有问题的,而且事实上也已经存在这样的协议了。但是,没有一个数据链路层协议提供这样的功能,所以,网桥也不能插手到帧的净荷中。基本来说,这个问题没有解决办法。如果要转发的帧太大,则它必须被丢弃。这样可以保持透明性。

另一个问题是安全性。802.11 和 802.16 都支持数据链路层的加密功能。以太网不支持。这意味着,当网络流量经过了一个以太网以后,无线网络所采用的各种加密服务都不再有效。更糟的是,如果一个无线站使用了数据链路层加密特性,那么,当它的流量经过一个以太网到达目的地的时候,就无法对数据进行解密了。如果无线站不使用加密特性,那么它的流量将会暴露在空气中。无论哪种情况都是一个问题。

针对安全问题的一个解决方案是,在更高的层上执行加密过程。但是,随之而来的是,一个 802.11 站必须知道它是否在跟 802.11 网络中的另一个站进行通信(如果是的话,则使用数据链路层加密特性;如果不是的话,则不使用数据链路层加密特性)。让终端站做这样的选择将会破坏透明性。

最后一点是服务质量。802.11 和 802.16 都提供了服务质量特性,但是形式不同:802.11 使用 PCF 模式,而 802.16 使用常数速率的连接。以太网没有服务质量的观念,所以,当数据流量经过了以太网之后,无论从哪里来,还是到哪里去,服务质量的特性都会丢失。

#### 4.7.2 本地的网络互连

上一小节讨论了通过一个网桥来连接两个不同的 IEEE 802 LAN 的时候所碰到的一些问题。然而,在规模较大的组织中,它往往有多个 LAN,即使这些 LAN 都是以太网,要将它们全部互连起来也会引发各种问题。理想情况下应该可以这样做:出去购买专门针对 IEEE 标准而设计的网桥,然后将连接器插到网桥的插口上,于是一切都应该可以正常工作了。应该不要求有任何硬件上的变化,不要求软件上的变化,不需要设置地址交换,也不用下载路由表或者参数,什么都不用。只要插上电缆,然后就万事大吉了。而且,原有的 LAN 操作也根本不会受到网桥的影响。换句话说,网桥应该是完全透明的(对于所有的硬件和软件都是不可见的)。令人惊奇的是,这实际上是有可能的。下面我们看一看这种魔术般的事情是如何实现的。

在最简单的情形下,一个透明网桥工作在混杂模式下,它接受所有跟它相连的 LAN 上传送的帧。例如,考虑图 4.42 中的配置。网桥 B1 被连接到 LAN 1 和 LAN 2,网桥 B2 被连接到 LAN 2、LAN 3 和 LAN 4。如果在 LAN 1 上有一个目标地址为 A 的帧到达网桥 B1,则该帧被立即丢弃,因为它已经在正确的 LAN 上了;但是,如果在 LAN 1 上有一个目标地址为 C 或者 F 的帧到达 B1 的话,则它将会被转发出去。

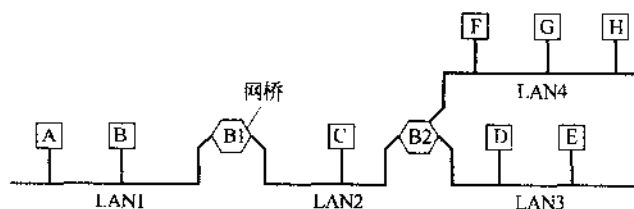


图 4.42 包含 4 个 LAN 和两个网桥的配置

当一帧到达的时候,网桥必须决定将该帧丢弃还是转发,如果是转发的话,还必须决



定将它转发到哪个 LAN 上。网桥通过在其内部的一张大的(散列)表中查寻一帧的目标地址来做出这样的决定。该散列表列出了每一个可能的目标地址,并且指明了它属于哪一条输出线路(LAN)。例如,B2 的表列出了 A 属于 LAN 2,因为 B2 只需知道该把发送给 A 的帧放到哪个 LAN 上就行了。事实上,它并不关心以后该帧是否还需要进一步的转发处理。

当最初网桥被插入进来的时候,所有的散列表都是空的。所有的网桥都不知道哪个目标地址该往哪里去,所以,网桥使用了一种扩散算法(flooding algorithm):对于每一个发向未知目标地址的进入帧,网桥将它输出到所有的 LAN 中(除了它到来的那个 LAN 以外)。随着时间的推移,这些网桥将会学习到每个目标地址都往哪里去,其过程将在下面描述。一旦知道了一个目标地址以后,以后发送给该地址的帧将只被放到正确的 LAN 上,而不再被扩散。

透明网桥所用的算法是逆向学习法(backward learning)。正如上面所提到的,网桥工作在混杂模式下,所以,它们可以查看到每个 LAN 上发送的所有的帧。通过检查这些帧的源地址,网桥就可以识别出通过哪个 LAN 可以访问到哪台机器。例如,如果在图 4.42 中,网桥 B1 看到 LAN 2 上的一帧来自于 C,那么它就知道通过 LAN 2 可以到达 C,所以,它就在散列表中构造一个表项,注明发送给 C 的帧应该使用 LAN 2。以后所有来自于 LAN 1 并且目标地址为 C 的帧都将被转发,而所有来自于 LAN 2 并且目标地址为 C 的帧将被丢弃。

当机器和网桥被加电或停电,或者从一个地方移动到另一个地方的时候,网络拓扑结构会发生变化。为了处理这种动态的拓扑结构,在构造一个散列表项的时候,该帧的到达时间也被记录到表项中。当一帧到达的时候,如果它的源地址已经在散列表中,那么,对应表项中的时间值被更新为当前时间。因此,与每个表项相关联的时间值反映了最后看到该机器上发送出一帧的时间。

在网桥中有一个进程定期地扫描散列表,并且将那些时间值在几分钟以前的表项都清除掉。按照这种方法,如果将一台计算机从一个 LAN 上拔下来,然后搬到同一个楼内的另一个地方,再将它重新接入到网络中,那么,几分钟之内该计算机就可以回到正常的操作状态,而无需任何人工干预。这个算法也意味着,如果一台计算机静止了几分钟时间(即几分钟之内不发送数据),那么任何发送给它的流量又要被扩散,直到它下一次发送出一帧为止。

对于一个进入的帧,它在网桥中的路由过程取决于它在哪个 LAN 上到达(源 LAN),以及它的目标地址在哪个 LAN 上(目标 LAN)。过程如下:

- (1) 如果目标 LAN 和源 LAN 相同,则丢弃该帧。
- (2) 如果目标 LAN 和源 LAN 不同,则转发该帧。
- (3) 如果目标 LAN 未知,则使用扩散法。

每当一帧到达时,这个算法必须被执行一遍。有一些专用的 VLSI 芯片可以在几个微秒内完成检查和更新表项的动作。

### 4.7.3 生成树网桥

为了提高可靠性,有些站点在 LAN 对之间并行地使用了两个或者多个网桥,如图 4.43 所示。然而,这种做法也引入了一些新的问题,因为它导致拓扑结构中产生了回路。

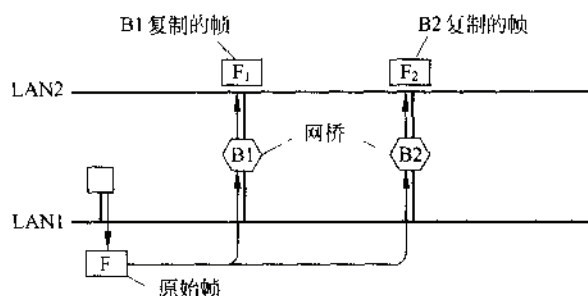


图 4.43 两个并行的透明网桥

通过观察在图 4.43 中一个带有未知目标地址的帧(F)的处理过程,我们可以看清楚这些问题的一个简单示例。按照处理未知目标帧的正常规则,每个网桥都使用扩散法,在这个例子中,这意味着每个网桥都要将 F 复制到 LAN 2 上。很快地,网桥 B1 看到了 F<sub>2</sub> 帧,F<sub>2</sub>也带有未知目标地址,所以网桥 B1 将它复制到 LAN 1 上,于是生成 F<sub>3</sub>(图中没有显示)。同样地,网桥 B2 将 F<sub>1</sub> 复制到 LAN 1 上,生成 F<sub>4</sub>(图中也没有显示)。现在网桥 B1 转发 F<sub>4</sub>,网桥 B2 转发 F<sub>3</sub>。这个循环将会无限进行下去。

这个难题的解决途径是,让这些网桥相互之间进行通信,然后用一棵可以到达所有 LAN 的生成树覆盖实际的拓扑网络。实际上,为了构造一个理想中的无环拓扑结构,LAN 之间一些潜在的连接被忽略掉了。例如,在图 4.44(a)中,我们看到 9 个 LAN 通过 10 个网桥相互连接起来。我们可以将这种网络配置抽象成一个图,每个 LAN 为一个节点。如果两个 LAN 之间有一个网桥连接,则将它们用一条弧连接起来。如图 4.44(b)所示,通过去掉虚线弧,整个图可以被简化为一棵生成树。利用这棵生成树,从每个 LAN 到任何其他 LAN,总是恰好只有一条路径。一旦所有的网桥统一使用同一棵树,则 LAN 之间的所有转发都将沿着这棵树来进行。由于从每个源到每个目标都只有惟一的路径,所以环是不可能产生的。

为了建立起生成树,首先这些网桥要选择其中之一作为树的根。它们的选择方法是,让每个网桥广播它自己的序列号。每个网桥都有自己的序列号,这是由生产商指定的,可以保证全球惟一。最低序列号的网桥将变成生成树的根。接下来,建立起一棵从根到每个网桥和 LAN 的最短路径树。这棵树就是生成树。如果一个网桥或者 LAN 失效了,则再计算一棵新的树。

这个算法的结果是,从每一个 LAN 到根建立了一条惟一的路径,因而从每一个 LAN 到其他任何一个 LAN 也建立了一条惟一的路径。虽然这棵树覆盖了所有的 LAN,但是,在树中并不是所有的网桥都是必须要出现的(为了避免环)。即使在生成树已经被建立起来之后,为了自动检测拓扑结构的变化并且更新生成树,在正常操作过程中这个算

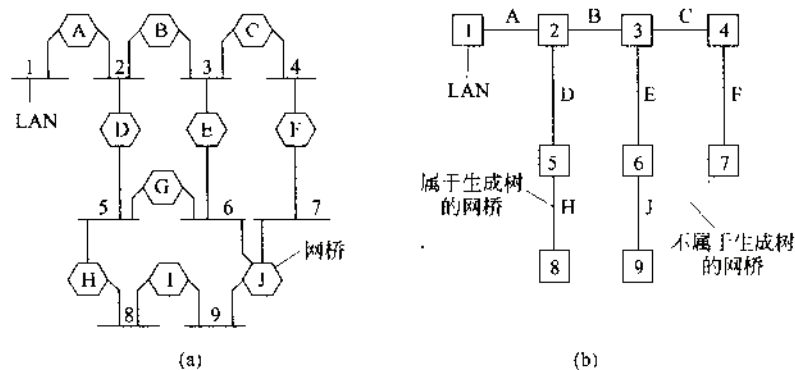


图 4.44

(a) 相互连接的 LAN; (b) 覆盖所有 LAN 的生成树, 虚线不属于生成树

法仍然继续运行。构建生成树的分布式算法是由 Radia Perlman 发明的, 详细的描述参见 (Perlman, 2000)。该算法也在 IEEE 802.1D 中被标准化了。

#### 4.7.4 远程网桥

网桥的一种常见用法是连接两个(或者多个)远距离的 LAN。例如, 一个公司可能在几个城市都有工厂, 每个工厂有独立的 LAN。理想情况下, 所有的 LAN 都应该相互连接起来, 从而整个系统就好像一个大的 LAN 一样运行。

这个目标可以这样来实现: 在每个 LAN 上安放一个网桥, 并且用点到点的线路(比如从电话公司租用的线路)将每一对网桥连接起来。图 4.45 演示了一个具有 3 个 LAN 的简单系统。这里应用了常规的路由算法。为了看清楚这一点, 最简单的办法是把 3 条点到点线路想象成无线 LAN, 然后它就变成了一个具有 6 个 LAN、用 4 个网桥连接起来的系统。请注意, 到现在为止我们所提到的 LAN 必须有主机连接在其上面。

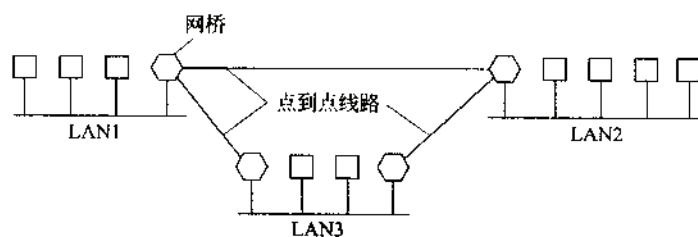


图 4.45 远程网桥可以用来连接远距离的 LAN

在点到点线路上可以使用各种协议。一种可能的做法是选择某一个标准的点到点数据链路协议, 比如 PPP, 由该协议把完整的 MAC 帧放到净荷域中。如果所有的 LAN 都是同类型的, 那么这种策略可以工作得很好, 惟一需要解决的问题是将帧转发到正确的 LAN 中。另一种做法是, 在源网桥一端将 MAC 头和尾剥掉, 把剩下的部分放在点到点协议的净荷域中。然后, 在目标网桥一端, 生成一个新的 MAC 头和尾。这种做法的缺点

是,到达目标主机上的校验和并不是源主机计算得到的校验和,所以,网桥内存中的环数据位所引起的错误可能无法被检测出来。

#### 4.7.5 中继器、集线器、网桥、交换机、路由器和网关

到现在为止,我们在本书中已经看到了很多种方法可将帧或者分组从一段电缆转移到另一段电缆上。我们已经提到了中继器(repeater,也称为中继器)、网桥、交换机、集线器、路由器和网关。所有这些设备都有实际的应用,但是,它们的工作方式或多或少有一些细微的差别。由于有如此多的设备种类,所以,有必要看一看它们工作方式的相似之处和不同之处。

首先,这些设备运行在不同的层上,如图 4.46(a)所示。之所以存在不同层的问题,是因为不同的设备使用不同的信息来决定如何交换。在典型的场景中,用户生成某些数据,然后将这些数据发送给一台远程机器。这些数据先被传递给传输层,传输层会加上一个头,比如 TCP 头,然后将结果得到的数据往下传递给网络层。网络层也会加上一个头,形成一个网络层分组,比如,形成一个 IP 分组。在图 4.46(b)中,我们看到,灰色阴影中的是 IP 分组。然后,该分组到达数据链路层,数据链路层加上它自己的头和校验和(CRC),并将结果得到的帧交给物理层以便传出去,比如说,通过一个 LAN 传出去。

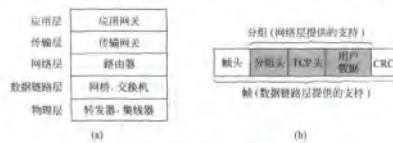


图 4.46  
(a) 每一层上的设备; (b) 帧、分组和头

现在我们来看一看交换设备,了解一下它们与分组和帧的关系。在最底层,即物理层中,我们可以看到中继器。中继器是模拟设备,用于连接两根电缆段。在一段上出现的信号被放大之后放到另一段上。中继器并不理解帧、分组和头的概念。它们只理解电压值。例如,在经典的以太网中,为了将电缆的最大长度从 500m 扩展到 2500m,以太网允许 4 个中继器。

接下来是集线器。集线器有许多条输入线路,它将这些输入线路连接起来。在任何一条线路上到达的帧都将被发送到所有其他的线路上。如果两帧同时到达,则它们将会冲突,就好像它们在同一根同轴电缆上一样。换句话说,整个集线器构成了一个冲突域。连接到同一个集线器上的所有线路必须运行在同样的速度上。集线器与中继器不同,它们通常不会放大进入的信号,并且可以容纳多块网卡。每块卡上有多个输入,但是,集线器与中继器的差别并不大。与中继器类似的是,集线器不检查 802 地址,也不以任何方式使用 802 地址。图 4.47(a)中显示了一个集线器。

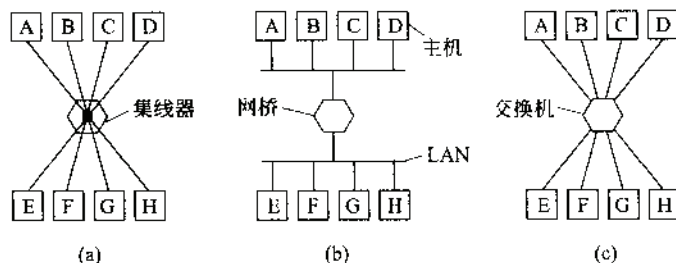


图 4.47

(a) 集线器; (b) 网桥; (c) 交换机

现在我们向上移动到数据链路层,在这一层上我们可以找到网桥和交换机。本书前面我们只是详细地讨论过网桥。一个网桥连接两个或者多个 LAN,如图 4.47(b)所示。当一帧到达的时候,网桥内部的软件从帧头中提取出目标地址,然后在一张表中查找该地址,以确定该把这一帧发送到哪里去。对于以太网,该地址是 48 位的目标地址,如图 4.17 中所示。如同集线器一样,现代网桥也有线卡,每块线卡通常支持某种特定类型的 4 条或者 8 条输入线路。例如,针对以太网的线卡不能够处理令牌环的帧,因为它不知道从帧头中的哪个地方找到目标地址。然而,一个网桥可以有多个线卡,每块线卡针对不同的网络类型和不同的速度。在一个网桥中,每条线路有它自己的冲突域,这与集线器不同。

交换机与网桥类似,它们都基于帧地址进行路由。实际上,许多人常常把这两个术语等同起来使用。主要的区别在于,交换机常常被用来连接独立的计算机,如图 4.47(c)中所示。因此,当 4.47(b)中的主机 A 想要给主机 B 发送帧的时候,网桥拿到了帧之后,只是将它丢弃。相反,在图 4.47(c)中,交换机必须主动地将帧从 A 转发给 B,因为没有其他的办法可以完成从 A 到 B 的帧转发工作。由于交换机的每个端口通常连接到一台计算机上,所以,交换机必须有足够的空间,以便容纳比网桥更多的线卡数量,毕竟网桥的设计目标是连接 LAN。每一块线卡都提供了缓冲区空间,以便将在它的端口上到达的帧缓存起来。由于每个端口都有它自己的冲突域,所以交换机永远不会由于冲突而丢失帧。然而,如果帧到达的速度超过了这些帧被重新传送出去的速度,那么,交换机可能会用完缓冲区空间,从而不得不开始丢帧。

为了能够减轻这个问题,现代的交换机这样来处理:一旦目标头域(即头部中的目标域)已经进来,尽管帧的其余部分还没有到达,则只要输出线路可以使用,交换机就开始转发该帧。这些交换机并没有使用“存储-转发”交换方式。有时候它们被称为直通型交换机(cut-through switch)。通常,直通转发过程完全是由硬件来完成的,而传统上,网桥往往包含一个实际的 CPU,存储-转发的交换过程由软件来实现。但是,由于所有现代的网桥和交换机都包含了特殊的、用于交换的集成电路部分,所以,交换机和网桥之间的差别更是一个市场因素,而并非技术因素。

到现在为止,我们已经看到了中继器和集线器,以及网桥和交换机,其中,中继器和集线器非常类似,网桥和交换机也非常类似。现在我们往上转移到路由器,它不同于所有前面提到的设备。当一个分组进入到一个路由器中的时候,帧头和帧尾被剥掉,位于帧的净



荷域中(如图 4.46 中的阴影部分)的分组被传递给路由软件。路由软件利用分组的头信息来选择一条输出线路。对于一个 IP 分组,分组头将包含一个 32 位(IPv4)或者 128 位(IPv6)的地址,而不是 48 位的 802 地址。该路由软件并没有看到帧地址,甚至也不知道该分组来自于一个 LAN,还是一条点到点的线路。我们将在第 5 章学习路由器和路由算法。

再往上一层我们可以找到传输网关。它们将两台使用了不同的面向连接传输协议的计算机连接起来。例如,假设一台计算机使用了面向连接的 TCP/IP 协议,另一台计算机使用了面向连接的 ATM 传输协议,现在它们需要通话。于是,传输网关可以将分组从一个连接复制到另一个连接中,并且根据需要对分组重新格式化。

最后,应用网关理解数据的格式和内容,并且将消息从一种格式转译为另一种格式。例如,电子邮件网关可以将 Internet 消息转译为移动电话的 SMS 消息。

#### 4.7.6 虚拟 LAN

局域网连接的早期时候,在许多办公楼中,沿着电缆管道铺设了黄色的粗电缆。这些电缆沿途的每一台计算机都被接入进来。通常会有许多电缆,这些电缆被连接到一个中心骨干上(如图 4.39 所示),或者连接到一个中心集线器上。至于哪一台计算机属于哪一个 LAN,这很难弄得很清楚。相邻办公室中的所有人都被放在同一个 LAN 上,也不管他们是否应该属于同一个 LAN。地理位置超越了逻辑结构。

随着 20 世纪 90 年代 10Base-T 和集线器的出现,一切都发生了变化。办公楼被重新布线(尽管相当昂贵),去掉了所有的黄色庭院软管,从每个办公室到中心接线柜之间安装了双绞线,通常中心接线柜位于每个走廊的尽头,或者在一个中心机房内,布线结构如图 4.48 所示。如果负责布线工作的副总裁有远见的话,他会选择安装 5 类双绞线;如果他是一个善于控制费用的专家的话,则会使用现有的(3 类)电话线(只有当几年以后出现快速以太网的时候,才来替换这些电话线)。

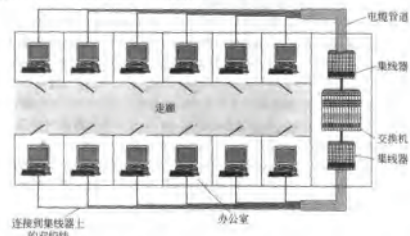


图 4.48 一个办公楼:使用了多个集线器和一台交换机的中心布线方案

对于集线器式(后面还要讨论到交换式的)以太网,通常有可能从逻辑上而不是从物理上配置 LAN。如果一家公司想要  $k$  个 LAN,那么它只要购买  $k$  个集线器。通过谨慎地选择哪一个连接器插入到哪一个集线器上,该公司可以按照特定的组织意义来构建每一个 LAN,而无需顾及地理位置。当然,如果同一个部门中的两个人在不同的楼内上班,那么,他们有可能连接到不同的集线器上,从而属于不同的 LAN。然而,即使是这样的情况,比起完全根据地理位置来决定 LAN 成员关系也要好多了。

谁在哪个 LAN 上真的很重要吗? 毕竟,在几乎所有的组织中,所有的 LAN 都是相互连接在一起的。简单的答案是: 是的,谁在哪个 LAN 上真的很重要。网络管理员希望将 LAN 上的用户分成适当的组,以反映出组织的结构,而不是大楼的物理布局结构,他们有很多个理由这样做。一个理由是安全性。任何一个网络接口都可以被设置到混杂模式下,从而将所有经过该处的流量复制下来。在许多部门中,比如研发部、专利部和财务部,有许多信息是不希望被流传到部门之外的。在这样的情况下,把一个部门中所有的人放在一个 LAN 中,并且不让任何流量传送到 LAN 以外是很有意义的。从管理层来讲,他们不喜欢听到这样的话: 除非每个部门中的所有人员都在相邻的办公室内并且没有任何中间干扰,否则,要把每个部门中的人放在单独的 LAN 中是不可能的。

第二个理由是载荷。有些 LAN 比别的 LAN 有更重的载荷,所以有时候期望将它们隔离开。例如,研究人员在运行各种试验的时候有可能会大量消耗网络资源,从而使他们的 LAN 饱和,这时候,财务部门的人员可能并不热心于要贡献出他们的网络资源来帮助研究部门。

第三个理由是广播。大多数 LAN 支持广播,而且,许多上层协议也大量使用这种特性。例如,当一个用户希望将一个分组发送给一个 IP 地址  $x$  的时候,它怎样知道应该在帧中放上哪一个 MAC 地址呢? 我们将在第 5 章讨论这个问题,但是现在先简单介绍一下答案,那就是,它广播一帧,该帧中包含这样一个问题: 谁拥有 IP 地址  $x$ ? 然后它等待答案。关于广播的用途还有许多例子。随着越来越多的 LAN 相互连接起来之后,经过每台机器的广播数量将随着机器的数量而线性增加。

与广播有关的一个问题是,偶然情况下网络接口崩溃之后将会产生无休止的广播帧流量。这种广播风暴(broadcast storm)的后果是: (1)整个 LAN 的通信容量将被这些帧占用; (2)所有这些互连的 LAN 上的所有机器将忙于处理和丢弃这些广播帧,从而使得网络和机器的能力被削弱。

初看起来,利用网桥或者交换机将多个 LAN 隔离开,这样就可以将广播风暴限制在一定的范围内,但是,如果要想达到透明效果的话(即,一台计算机跨越网桥移动到另一个不同的 LAN 中而不引起他人的注意),那么网桥必须要转发广播帧。

看清楚了为什么有些公司希望采用多个 LAN,并且每个 LAN 有独立的限定范围之后,我们回到原来的问题: 将逻辑拓扑结构从物理拓扑结构上脱离开。假定一个用户在公司内部从一个部门被调动到另一个部门,但是没有换办公室;或者他没有变换部门,但是换了办公室。在集线器布线方式下,为了将用户移动到正确的 LAN 中,这意味着需要让网络管理员到接线柜那里把分配给用户计算机的连接器从一个集线器上拔下来,再插到另一个集线器上。

在许多公司中,组织结构总是在不停地变化,这意味着系统管理员要花大量的时间用于拔下连接器再插回到另外的地方。而且,在某些情况下,这种改变甚至是根本不可能的,因为用户机器的双绞线离正确的集线器太远了(比如在其他的大楼中)。

为了更加灵活地响应用户的请求,网络供应商开始考虑另一种工作方式,即用纯软件的方式重新对大楼进行布线。结果得到的概念称为 VLAN(虚拟 LAN),而且 802 委员会已经将 VLAN 标准化了。现在许多组织已经部署了 VLAN。接下来我们将讨论 VLAN。有关 VLAN 更多的信息,请参见(Breyer and Riley, 1999; and Seifert, 2000)。

VLAN 建立在一种特殊设计的、支持 VLAN 的交换机的基础之上,但是,VLAN 的外围也可以有一些集线器,如同图 4.48 中一样。为了建立一个基于 VLAN 的网络,网络管理员首先要确定共有多少个 VLAN,哪些计算机将位于哪个 VLAN 上,每个 VLAN 叫什么名称。通常 VLAN 用颜色来命名(非正式),因为这样做之后就有可能打印出一张颜色图来显示机器的物理布局情况,其中红色 LAN 中的成员用红色来表示,绿色 LAN 中的成员用绿色来表示,以此类推。按照这种方式,在同一个视图中可以同时表示出物理布局和逻辑布局。

例如,考虑图 4.49(a)中的 4 个 LAN,其中 8 台机器属于 G(灰色)VLAN,7 台机器属于 W(白色)VLAN。在图中,两个网桥 B1 和 B2 将 4 个物理 LAN 连接起来。如果采用中心双绞线的话,那么可能也会有 4 个集线器(图中没有画出),但是,从逻辑上而言,一条多分支的电缆与集线器是相同的。之所以用这种方式来画图,是为了避免使该图显得散乱。而且,术语“网桥”通常被用于每个端口上有多台机器的情形,如图中所示,但是在别的方面,“网桥”和“交换机”本质上是相互交换使用的术语。图 4.49(b)显示了同样的机器和 VLAN,但是用到了交换机,并且每个端口上只有一台计算机。

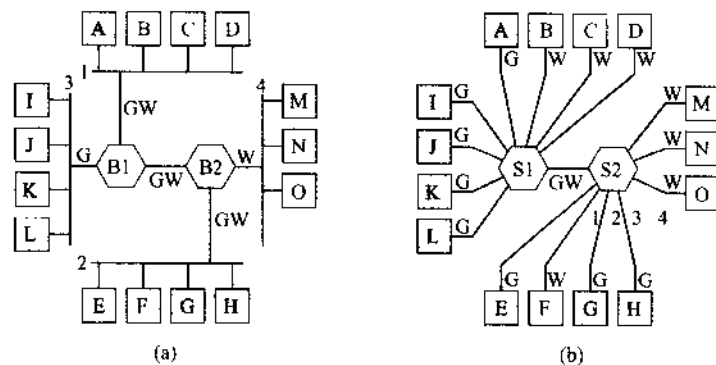


图 4.49

- (a) 通过 2 个网桥将 4 个物理 LAN 组织到 2 个 VLAN 中: 灰色和白色 VLAN;  
(b) 通过交换机将同样的 15 台机器组织到 2 个 VLAN 中

为了使 VLAN 正常地运行,在网桥或者交换机中必须建立起配置表。这些配置表指明了通过哪些端口(线路)可以访问到哪些 VLAN。当一帧到来的时候,比如说,该帧来自于灰色 VLAN,那么,这一帧必须要被转发到所有标记为 G 的端口。这一条规则对于普通的(即单播)流量以及多播和广播流量都是成立的。

请注意,一个端口也可以被标记上多种 VLAN 颜色。在图 4.49(a)中,我们可以清

楚地看到这一点。假设机器 A 广播了一帧。网桥 B1 接收到这一帧,并且发现它来自于灰色 VLAN 上的一台机器,所以,它将该帧转发到所有标记为 G 的端口上(当然除了该帧进来的端口以外)。由于 B1 只有另外两个端口,并且这两个端口都被标记为 G,所以,该帧被转发到这两个端口上。

在 B2 上,情形有所不同。这里,网桥 B2 知道 LAN 4 中没有标记为 G 的灰色机器,所以,这一帧在这里不会被转发。它只被转发到 LAN 2 中。如果 LAN 4 中有一个用户应该调换部门,并且被移动到灰色 VLAN 中,那么 B2 内部的配置表必须被更新,连接到 LAN 4 的端口不再被标记为 W,而是更新为 GW。如果机器 F 变成灰色,那么,连接到 LAN 2 的端口应该变成 G,而不再是 GW。

现在,让我们想象一下这样的情形:LAN 2 和 LAN 4 中的所有机器都变成灰色。那么,不仅 B2 中到达 LAN 2 和 LAN 4 的端口都要被标记成 G,而且,B1 中到达 B2 的端口也要从 GW 变成 G,因为来自于 LAN 1 和 LAN 3 的白色帧不用再转发给 B2 了。在图 4.49(b)中,同样的条件也成立;在这里,因为每个端口只连接一台机器,所以它的外面只有一个 VLAN,因此每个端口只被标记为一种颜色。

到现在为止,我们总是假定网桥和交换机通过某种途径知道进来的帧是什么颜色。那么,它们如何知道帧的颜色呢?目前有三种方法正在使用,如下:

- (1) 每个端口都被分配一种 VLAN 颜色。
- (2) 每个 MAC 地址都被分配一种 VLAN 颜色。
- (3) 每个 3 层协议或者 IP 地址被分配一种 VLAN 颜色。

在第一种方法中,每个端口都用 VLAN 颜色来标记。然而,对于这种方法,只有当一个端口上的所有机器都属于同一个 VLAN 的时候,它才可以正常工作。在图 4.49(a)中,对于 B1,连接 LAN 3 的端口符合这个条件,而连接 LAN 1 的端口不符合这个条件。

在第二种方法中,网桥或者交换机有一张表,其中列出了与其相连接的每台机器的 18 位 MAC 地址,以及该机器所属的 VLAN。在这样的条件下,有可能在一个物理 LAN 中混合多个 VLAN,如图 4.49(a)中的 LAN 1 所示。当一帧到达的时候,网桥或者交换机必须要做的事情是,从帧中提取出 MAC 地址,然后在表中查寻,看该帧来自于哪一个 VLAN。

第三种方法是,网桥或者交换机检查帧的净荷域,比如说,将所有的 IP 机器划分或一类,将所有的 AppleTalk 机器划分成另一类。对于前者,IP 地址也可以被用来标识一台机器。当许多机器都是笔记本电脑的时候,由于这些计算机可能会被搬动到任何一个地方,所以这种策略会非常有用。因为每一台移动的计算机都有它自己的 MAC 地址,所以,仅仅知道所使用的是哪一台计算机,这并不能说明该计算机位于哪一个 VLAN 中。

这种方法惟一的一个问题是,它违反了网络的最基本规则:层的独立性。本来净荷域跟数据链路层的事情毫无关系。因此,网桥或者交换机不应该检查净荷域,更不应该基于它的内容来作出决定。使用这种方法的一个后果是,一旦 3 层协议有了变化(比如说,从 IPv4 升级到 IPv6),则交换机立刻就失败。不幸的是,市场上存在这种工作方式的交换机。

当然,根据 IP 地址进行路由并没有错误——第 5 章中几乎所有的内容都在讨论 IP

路由——但是,将多个层混合起来显然是在找麻烦。交换机厂商有可能对这种观点嗤之以鼻,他们会解释说,他们的交换机同时能理解 IPv4 和 IPv6,所以,一切都不用担心。但是,当 IPv7 出来的时候又会怎么样呢?他们可能又会这样说:购买新的交换机,真的有那么糟糕吗?

### IEEE 802.1Q 标准

再深入地想一想 VLAN 这个题目,你就会发现,真正重要的是帧本身的 VLAN,而不是发送方机器的 VLAN。如果存在某一种办法可以在帧头中标识出 VLAN,那就根本不需要检查净荷数据了。对于一个新的 LAN 标准,比如 802.11 或者 802.16,在头部增加一个 VLAN 域是非常容易的事情。实际上,802.16 中的 Connection ID(连接标识符)域在本质上非常类似于 VLAN 标识符。但是,对于最为主流的 LAN——以太网,情况又怎么样?以太网的帧中并没有任何多余的域可以存放 VLAN 标识符。

1995 年,IEEE 802 委员会终于将这个问题提到日程上。经过多次讨论之后,它非常不可思议地改变了以太网的帧头。新的格式于 1998 年被发表在 IEEE 标准 802.1Q 中。这份新的格式包含了一个 VLAN 标签(tag),稍后我们将会讨论该标签。很自然地,当以太网头部被重构以后,一定还需要做一些其他方面的改变。很快可以想到的一些问题有:

- (1) 我们需要抛弃现有的成千上百万的以太网卡吗?
- (2) 如果不抛弃这些网卡的话,谁来生成新的域?
- (3) 对于那些已经达到了最大长度的帧又该怎么办呢?

当然,802 委员会知道这些问题(这是残酷的现实),而且也必须拿出可行的解决方案。

解决方案的关键在于,我们必须意识到,其实真正使用 VLAN 域的只有网桥和交换机,用户机器并不使用 VLAN 域。因此,在图 4.49 中,VLAN 域出现在网桥或者交换机之间的线路上是非常必要的,但是在到达终端站的线路上它们并不是必要的。因此,为了使用 VLAN,网桥或者交换机必须理解 VLAN,但这已经是一个基本的要求了。现在我们只是引入额外的要求,即要求网桥或者交换机必须理解 802.1Q,一些新的设备已经做到这一点了。

至于是否需要抛弃所有原来的以太网卡,答案是不。记住,802.3 委员会甚至未能使人们接受“将 Type 域改变为 Length 域”的做法。你可以想象一下,要是发表一份声明称“所有现有的以太网卡必须被抛弃”将会有什么样的反应。然而,当新的以太网卡进入到市场上的时候,期望这些网卡将会与 802.1Q 兼容,它们将正确地填充 VLAN 域。

那么,如果最初的发送方并没有生成 VLAN 域的话,则该由谁来填充呢?答案是,在转发该帧过程中第一个支持 VLAN 的网桥或者交换机加入 VLAN 域,下行路径中最后一个支持 VLAN 的网桥或者交换机将这些 VLAN 域去掉。但是它们如何知道哪一帧属于哪一个 VLAN 呢?好,第一个网桥或者交换机可以给一个端口分配 VLAN 号,可以查看 MAC 地址或者检查净荷域(应该是禁止的)。现在还做不到所有的以太网卡都是 802.1Q 兼容的,但是我们也只能从现在做起了。在这里,一个比较现实的期望是,所有的千兆以太网卡从一开始就是 802.1Q 兼容的,当人们升级到千兆以太网的时候,802.1Q



将自动被引入进来。至于帧长超过 1518 字节的问题,802.1Q 只是将限制值提高到 1522 字节。

在传递过程中,许多网络环境中有一些遗留的机器(往往是传统的以太网或者快速以太网),它们不能理解 VLAN,而其他的机器(往往是千兆以太网)则可以理解 VLAN。这种情况如图 4.50 所示,其中阴影符号能够理解 VLAN,而空心符号则无法理解 VLAN。为了简单起见,我们假定所有的交换机都能够理解 VLAN。如果不是这样的情形的话,那么,第一个可理解 VLAN 的交换机将会根据 MAC 或者 IP 地址加上相应的标签。

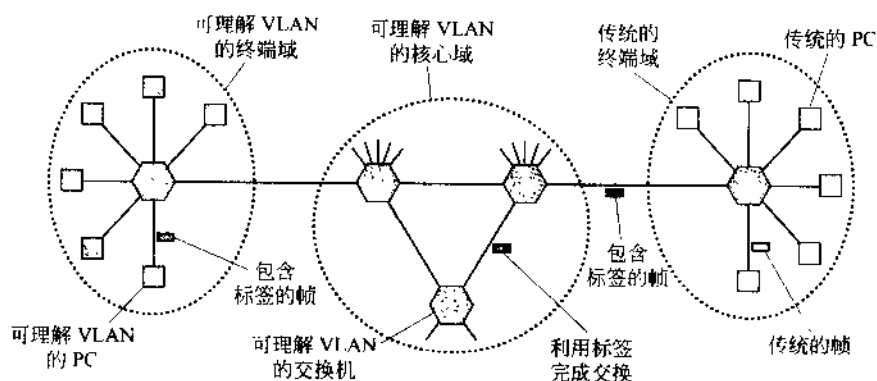


图 4.50 从传统的以太网到可理解 VLAN 的以太网之间的传递过程  
其中阴影符号能够理解 VLAN,空心符号不能理解 VLAN

在这个图中,可理解 VLAN 的以太网卡直接生成了包含标签(即 802.1Q)的帧,并且在进一步交换的时候用到了这些标签。为了完成交换工作,交换机必须要知道在每个端口上哪些 VLAN 是可达的,就如同以前一样。只有当交换机知道了哪些端口连接到灰色 VLAN 上的机器之后,知道一帧属于灰色 VLAN 才是有意义的。因此,交换机需要一张由 VLAN 来索引的表,它可以指明应该使用哪些端口,以及它们是遗留下来的,还是可以理解 VLAN 的。

当一台传统的 PC 将一帧发送到一个可理解 VLAN 的交换机的时候,该交换机根据它对于发送方 VLAN 的认识(可以使用端口、MAC 地址或者 IP 地址)建立一个新的包含标签的帧。从这个点开始,“发送方是一台传统的机器”这个事实与这一帧已经没有关系了。类似地,如果一个交换机需要将一个包含标签的帧递交给一台传统的机器,那么在递交之前,它必须将这一帧重新格式化成传统的格式。

现在我们来查看 802.1Q 帧的格式。如图 4.51 所示。惟一的变化是加入了一对 2 个字节的域。第一个 2 字节域是 VLAN 协议 ID。它的值总是 0x8100。由于这个数值大于 1500,所以,所有的以太网卡都会将它翻译成一种类型(type),而不是一个长度(length)。传统的网卡处理这样的帧是没有意义的,因为按理说这样的帧不会被发送给传统的网卡。

第二个 2 字节域包含三个子域。最主要的是 VLAN identifier(VLAN 标识符)域,它占用低 12 位。这正是一切的关键所在——这一帧属于哪一个 VLAN? 3 位长度的

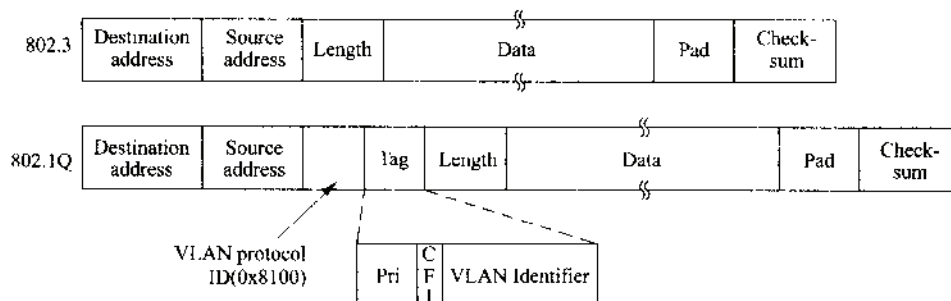


图 4.51 802.3(传统)帧格式和 802.1Q 以太网帧格式

Priority(优先级)域与 VLAN 根本一点关系也没有,但是,由于修改以太网头部是一种 10 年才会发生一次的事件,而且这次事件花了三年时间,涉及到一百人,因此,为什么在做这件事情的时候不再引入一些好的特性呢?这个域使得交换设备有可能区分硬的实时流量与软的实时流量,以及对于时间不敏感的流量,这样做的目的是为了在以太网上提供更好的服务质量。如果要在以太网上传递话音信息的话,那么这是很需要的(不过,公平地来讲,IP 中有一个类似的域已经存在四分之一世纪了,但是几乎没有人用它)。

最后一位 CFI(Canonical Format Indicator, 规范的格式指示器)应该称为 CEI(Corporate Ego Indicator, “团体自我指示器”)。它最初的意图是,指明 MAC 地址是 little-endian 还是 big-endian 字节序,但是,在其他一些争论过程中,这种用法渐渐被丢掉了。现在,这一位置“1”标志着:净荷中包含一个冻结了的 802.5 帧,它希望在目标处能找到另一个 802.5 LAN,而中间则是由以太网来承载的。当然,这整个安排跟 VLAN 一点关系也没有。但是,标准委员会的政治学与常规的政治学也没有什么不同:如果你赞成我提出的位,那么我也会给你的位投赞成票。

正如我们上面所提到的,当一个包含了标签的帧到达一个可理解 VLAN 的交换机的时候,该交换机利用 VLAN ID 作为索引,在一张表中查找哪些端口需要发送该帧。但是,这张表又从哪里来呢?如果手工来构造的话,那么我们又回到了问题的起点:需要手工对网桥进行配置。透明网桥的优美之处就在于,它是即插即用的,并不要求任何手工配置工作。丢失这样好的一个特性简直是一种耻辱。幸运的是,可理解 VLAN 的网桥也可以根据它们所看到的标签自动进行配置。如果一个来自端口 3 的帧包含了 VLAN 4 的标签,那么,很显然,端口 3 上有一台机器在 VLAN 4 中。802.1Q 标准说明了如何动态地建立这张表,其中主要引用了在 802.1D 中被标准化的 Perlman 算法。

在结束关于 VLAN 路由的讨论之前,有一种现象值得最后提一下。Internet 和以太网领域中的许多人极其推崇无连接的网络技术,而极力反对任何在数据链路层或者网络层上建立连接的做法。然而,令人惊奇的是,VLAN 实际上引入了一种类似于连接的机制。为了正确地使用 VLAN,每一帧携带一个新的、特殊的标识符,该标识符可以被当作索引以便在交换机的内部表中查找这一帧的发送目标。这个过程与面向连接的网络中的路由机制正好完全一致。在无连接的网络中,真正被用来进行路由的是目标地址,而不是某种形式的连接标识符。我们将在第 5 章更加详细地讨论这种爬行方式的连接机制。

## 4.8 本章小结

有些网络只有一个信道用于所有的通信。在这些网络中,关键的设计问题是,如何在所有期望使用这惟一信道的竞争站之间分配该信道。目前已经有许多种信道分配算法。图 4.52 概括了一些比较重要的信道分配方法。

方法	说明
FDM	为每个站专门分配一段频率
WDM	针对光纤的动态 FDM 方案
TDM	为每个站分配一个时槽
纯的 ALOHA	任何时候都不进行同步传输
分槽的 ALOHA	在明确定义的时槽内随机传输
1-持续 CSMA	标准的载波检测多路访问
非持续的 CSMA	当信道被检测到忙的时候随机延迟
p-持续 CSMA	CSMA,但是持续的概率为 p
CSMA/CD	CSMA,但是在检测到冲突之后就放弃
位图	用一个位图进行轮循
二进制倒计数	在所有准备就绪的站中,下一个是编号最大的站
树径协议 (tree walk)	通过选择机制来减少竞争
MACA, MACAW	无线 LAN 协议
以太网	支持二进制指数后退算法的 CSMA/CD
FHSS	跳频扩频
DSSS	直接序列扩频
CSMA/CA	避免冲突的 CSMA

图 4.52 公用信道的系统和信道分配方法

最简单的信道分配方案是 FDM 和 TDM。当站的数量比较小而且固定不变,并且流量连续的时候,这两种方案比较有效。它们被广泛地应用于这样的环境中,比如用于电话主干上的带宽分割。

当站的数量比较大而且可变,或者流量具有突发性变化的时候,FDM 和 TDM 就非常不合适了。另一种可以选择的方案是 ALOHA 协议,包括分时槽的和不分槽的 ALOHA 协议。ALOHA 以及它的许多变种和衍生方案都已经有了广泛的讨论和分析,并且已经应用于许多实际的系统中。

如果信道的状态可以被检测到,那么,当一个站正在传输的时候,其他的站就可以避免同时也启动一个传输过程。这项技术被称为载波检测,它已经产生了许多种协议,分别

应用于 LAN 和 MAN 中。

有一类非常知名的协议可以完全消除竞争,或者至少可以极大地减少竞争。二进制倒数计数法可以完全消除竞争。树径协议动态地将站划分成两个不相交的组,一个组允许传输,另一个组不允许传输,从而减少了竞争。它划分站的做法是:循环地划分下去,直到只有一个站允许发送帧。

无线 LAN(WLAN)有它们自己的问题和解决方案。最大的问题是由隐藏的站引起的,所以 CSMA 无法正常工作。其中一类以 MACA 和 MACAW 为代表的解决方案是,试图激发出目标站周围的传输过程,以使 CSMA 工作得更好。跳频扩频和直接序列扩频也可以使用。IEEE 802.11 将 CSMA 和 MACAW 结合起来产生了 CSMA/CA。

以太网是最为主流的局域网连接方式。它使用 CSMA/CD 来分配信道。老式的局域网使用一根电缆将机器串接起来。但是,现在最为普遍的是采用双绞线连接到集线器和交换机的方式。速度从 10Mbps 到 1Gbps 不等,而且现在还在继续提高。

无线 LAN 已经变得很普遍了,在这个领域中 802.11 占了统治地位。它的物理层允许 5 种不同的传输模式,包括红外、各种扩频方案和一个多信道的 FDM 系统。在每个单元中存在一个基站的情况下,802.11 可以工作;如果没有基站,它也可以工作。所用的协议是 MACAW 的一个变种,采用了虚拟载波检测机制。

无线 MAN 正在兴起。它们是宽带系统,使用无线电波来代替电话连接中的最后一英里。它们使用了传统的窄带调制技术。服务质量很重要,802.16 标准定义了 4 类:常数速率服务、两种可变的位速率服务,以及一种尽力投递服务。

蓝牙系统也是无线的,但是其目标更多地瞄准了桌面系统,它用无线的方式将头戴设备和其他的外设连接到计算机。它也可用来将外设(比如传真机)连接到移动电话上。如同 802.11 一样,它使用了 ISM 频段,采用跳频扩频技术。由于考虑到许多环境中的噪声水平,以及实时交互的需求,在它的各种协议中内置了精细的前向纠错机制。

既然有了这么多不同的 LAN,自然就需要一种将它们相互连接起来的途径。网桥和交换机正好可以担当此任。利用生成树算法可以建立即插即用的网桥。在 LAN 互连的领域中,一个新的发展是 VLAN,它可以将多个 LAN 的逻辑拓扑结构与它们的物理拓扑结构分离开来。为此而引入了一种新的以太网帧格式(802.1Q),通过此格式可以使现有的组织更加方便地引入 VLAN。

## 习 题

1. 在这个练习中,请使用本章中的一种规则(方案),但是在计算之前请先声明这种规则。在一个 100Mbps 的信道上,待传输的帧随机地到达。如果当一帧到达的时候该信道正忙,那么它必须排队等待。帧的长度呈指数分布,均值为每帧 10 000 位。对于下列每一种帧到达率,请给出平均一帧的延迟,包括排队时间和传输时间。

- (a) 90 帧/秒
- (b) 900 帧/秒
- (c) 9000 帧/秒

2.  $N$  个站共享一个 56-kbps 的纯 ALOHA 信道。每个站平均每 100 秒输出一个 1000 位的帧,即使前面的帧还没有被送出,它也这样进行(比如这些站可以将送出的帧缓存起来)。请问  $N$  的最大值是多少?

3. 考虑在低载荷情况下纯的 ALOHA 和分槽的 ALOHA 的延迟。哪一个延迟更小? 请说明你的理由。

4. 10 000 个航线预定站正在竞争使用一个分槽的 ALOHA 信道。这些站平均每小时发出 18 次请求。时槽为  $125\mu\text{s}$ 。总的信道载荷大约是多少?

5. 一大群 ALOHA 用户每秒钟产生 50 个请求,包括原始的请求和重传的请求。时槽单位为 40ms。

(a) 首次发送成功的几率是多少?

(b) 恰好  $k$  次冲突之后成功的概率是多少?

(c) 所需传送次数的期望值是多少?

6. 对一个无限用户的分槽 ALOHA 信道的测试表明,10%的时槽是空闲的。

(a) 信道载荷  $G$  是多少?

(b) 吞吐量是多少?

(c) 该信道是载荷不足,还是过载了?

7. 在一个无限用户的分槽 ALOHA 系统中,一个站在冲突之后到重传之间的平均时槽数目为 4。请画出该系统的延迟与吞吐量之间的曲线关系。

8. 如果一个 LAN 使用了下列协议,请问在最差情况下,一个站  $s$  在开始传送帧之前必须要等待多长时间?

(a) 基本的位图协议。

(b) 改变虚拟站序列号的 Mok-Ward 协议。

9. 一个 LAN 使用了 Mok 和 Ward 版本的二进制倒数计数法。在一个特定的时刻,10 个站的虚拟序列号为 8、3、4、5、1、7、3、6、9 和 0。接下来要发送帧的站依次为 4、3 和 9。当这三个站完成了它们的传输任务之后,新的虚拟站序列号是什么?

10. 16 个站的编号从 1 到 16,它们正在竞争一个使用了可适应树径协议的共享信道。如果地址编号为素数的所有站突然间全部要发送帧,请问需要多少位时槽才能解决竞争?

11.  $2^n$  个站使用可适应树径协议来共享访问一根电缆。在一个特定的时刻,两个站准备要发送帧。请问,如果  $2^n \gg 1$ ,则沿着树经过的最小、最大和平均时槽数分别是多少?

12. 我们在本章中讨论过的无线 LAN 使用了诸如 MACA 这样的协议,而没有使用 CSMA/CD。请问在什么样的条件下有可能使用 CSMA/CD?

13. WDMA 和 GSM 信道访问协议有哪些共同的特性? 关于 GSM 请参考第 2 章。

14. 6 个站的编号从 A 到 F,它们使用 MACA 协议进行通信。请问有可能同时发生两个传输操作吗? 说明你的理由。

15. 一个办公楼有 7 层,每一层有 15 个相邻的办公室。每个办公室的前面墙上包含一个终端插口,所以,在垂直面上,这些插口构成了一个矩形网格,在水平方向和垂直方向上插口之间均有 4 米距离。假定在任何一对插口之间,无论是水平的、垂直的,或是对角



的,拉一根直接的电缆都是可行的,那么,请问,若使用下面的结构,需要多少米电缆可将所有的插口连接起来:

(a) 星型结构,中间有一台路由器。

(b) 802.3 LAN。

16. 标准的 10Mbps 以太网的波特率是多少?

17. 画出位流 0001110101 的曼彻斯特编码。

18. 画出上一个问题中的位流的差分曼彻斯特编码。假设线路的初始状态为低电压。

19. 一个 1km 长、10Mbps 的 CSMA/CD LAN (不是 802.3), 其传播速度为  $200\text{m}/\mu\text{s}$ 。在这个系统中不允许使用中继器。数据帧的长度为 256 位,其中包括 32 位的头部、校验和以及其他的开销。在一次成功的传输之后,第一个位时槽将被预留给接收方,以便它抓住信道并发送一个 32 位的确认帧。假定没有冲突,请问有效数据率为多少 (不包括各种开销)?

20. 两个 CSMA/CD 站都企图传送大的文件 (即多帧文件)。在每一帧被送出之后,它们就使用二进制指数后退算法竞争信道。在第  $k$  轮结束竞争的概率是多少? 每个竞争周期的平均轮数是多少?

21. 考虑在一条 1km 长的电缆 (无中继器) 上建立一个 1Gbps 速率的 CSMA/CD 网络。信号在电缆中的速度为  $200\,000\text{km/s}$ 。请问最小的帧长度为多少?

22. 一个通过以太网传送的 IP 分组有 60 字节长,其中包括所有的头部。如果没有使用 LLC 的话,则以以太网帧中需要填补字节吗? 如果需要的话,请问需要填补多少字节?

23. 以太网帧必须至少 64 字节长,这样做的理由是,当电缆的另一端发生冲突的时候,传送方仍然还在发送过程中。快速以太网也有同样的 64 字节最小帧长度限制,但是,它可以以快 10 倍的速度发送数据。请问它如何有可能维持同样的最小帧长度限制?

24. 有些书将以以太网帧的最大长度说成是 1518 字节,而不是 1500 字节。这些书错了吗? 请说明你的理由。

25. 尽管千兆以太网假设的递交速率为 1Gbps,但是,1000Base-SX 规范声明了时钟的运行速度应该为 1250MHz。这里超高的时钟速度是为了提供额外的安全余量吗? 如果不是,那会是什么?

26. 千兆以太网每秒钟能够处理多少帧? 请仔细想一想,并考虑所有有关的情形。提示:请考虑千兆位以太网的实质。

27. 请说出两个网络,它们允许将多个连续的帧包装在一起。为什么这个特性值得专门提出来?

28. 图 4.27 显示了 4 个站 A、B、C 和 D。你认为后两个站中哪一个最接近 A? 为什么?

29. 假设一个 11Mbps 的 802.11b LAN 正在通过无线电信道传送一批连续的 64 字节的帧,位错误率为  $10^{-7}$ 。请问平均每秒钟将有多少帧被损坏?

30. 一个 802.16 网络有一个 20MHz 宽度的信道。请问每秒钟可以给一个客户站发送多少位?

31. IEEE 802.16 支持 4 类服务。请问哪一类服务最适合用来发送非压缩的视频数据?

32. 为什么有些网络会使用纠错码,而不使用检错+重传的机制?请给出两个理由。

33. 从图 4.35 中,我们可以看到,一个蓝牙设备可能同时位于两个微微网中。是否有理由说明一个设备不可能同时是这两个微微网中的主节点?

34. 图 4.25 显示了几种物理层协议。这些协议中哪一个最接近蓝牙物理层协议?它们之间的最大差异是什么?

35. 在蓝牙网络的主节点和从节点之间,蓝牙支持两种类型的链路。请问这两种链路是什么?每一种分别有什么样的用途?

36. 在 802.11 的跳频扩频调制方法中,信标帧包含了停延时间。你认为蓝牙中类似的信标帧中也包含了停延时间吗?请讨论你的答案。

37. 考虑图 4.44 中相互连接的 LAN。假设主机 a 和 b 在 LAN 1 上,主机 c 在 LAN 2 上,主机 d 在 LAN 8 上。刚开始的时候,所有的网桥内部的散列表都是空的,并且使用了图 4.44(b)中所示的生成树。在下面给出的每一个事件发生以后(首先(a),然后(b),然后(c),以此类推),不同的网桥中的散列表将如何变化。

(a) a 向 d 发送帧。

(b) c 向 a 发送帧。

(c) d 向 c 发送帧。

(d) d 移动到 LAN 6 上。

(e) d 向 a 发送帧。

38. 在一个扩展的 LAN 中使用生成树来转发帧的一个结果是,有的网桥可能根本不参与帧的转发过程。请在图 4.44 中标出三个这样的网桥。既然这些网桥没有被用于转发帧,那么是否有理由要保留这些网桥呢?

39. 假设一台交换机上的线卡有 4 条输入线路。下面的情形发生得很频繁:在一个线卡的某一条线路上到达的帧必须在同一线卡的另一条线路上送出。作为这种情况的一个结果,交换机的设计者要面对什么样的选择?

40. 一个专门为了用于快速以太网而设计的交换机有一块可以传送 10Gbps 的底板。请问在最差情况下,它每秒钟可以处理多少帧?

41. 考虑图 4.49(a)中的网络。如果机器 J 突然变成白色的话,图中的标记需要做相应的变化吗?如果需要的话,该怎么变?

42. 简略地描述一下存储-转发型交换机和直通型交换机之间的区别。

43. 从损坏帧的角度而言,存储-转发型交换机比起直通型交换机更有优势。请说明这种优势是什么。

44. 为了使得 VLAN 可以工作,在交换机和网桥内部需要有相应的配置表。如果图 4.49(a)中的 VLAN 使用集线器而不是多分支的电缆,那会怎么样呢?集线器也需要配置表吗?为什么需要,或者为什么不需要?

45. 图 4.50 中,在右侧的传统终端域中的交换机是一个可理解 VLAN 的交换机。在那里有可能使用传统的交换机吗?如果可能,请问那将会如何工作?如果不可能,请问

---

为什么不可能？

46. 请编写一个程序来模拟以太网上 CSMA/CD 协议的行为：当一帧被传送的时候，有  $N$  个站都准备要发送。你的程序应该报告出每一个站成功地开始发送帧的时间。假设每一个时槽只有一次时钟滴答 (51.2ms)，并且冲突检测和发送干扰序列只需要一个时槽。所有的帧都是最大可允许的长度。

## 第5章 网络层

网络层关注的是如何将分组从源端沿着网络路径送达目标端。为了将分组送到目标端,有可能沿路要经过许多跳(hop)中间路由器。这种功能显然与数据链路层的功能不同,数据链路层的目标只是将帧从线的一端传送到另一端。因此,网络层是处理端到端数据传输的最低层。

为了实现这个目标,网络层必须知道通信子网(即所有路由器构成的集合)的拓扑结构,并且在拓扑结构中选择适当的路径。同时,网络层还必须仔细地选择路由器,以避免发生某些通信线路和路由器负载过重,而其他线路和路由器空闲的情形。最后,当源和目标位于不同网络中的时候,就会产生新的问题。这需要由网络层来解决这些问题。在本章中,我们将讨论所有这些议题,并且利用 Internet 及其网络层协议(IP 协议)来进一步阐明这些议题,同时本章也会讨论无线网络中相应的问题。

### 5.1 网络层设计要点

在本节中,我们将介绍一下网络层设计人员必须要抓住的一些要点,其中包括需要为传输层提供的服务以及子网的内部设计。

#### 5.1.1 存储-转发分组交换

在介绍网络层的细节之前,可能有必要再次说明一下网络层协议的运行环境。图 5.1 大致勾画出了这样的环境。系统中最主要的部件是网络承运商的设备(通过传输线路连接起来的路由器)和客户的设备,在图 5.1 中,网络承运商的设备位于阴影椭圆内,而客户的设备位于椭圆之外。主机 H1 通过一条租用的线路,直接连接到承运商的路由器 A 上;相反,H2 则位于某一家客户所拥有并运行的 LAN 上,该 LAN 中有一台路由器 F。路由器 F 也通过一条租用的线路连接到网络承运商的设备上。我们之所以将 F 画在椭圆的外面,是因为它并不属于承运商,但是从网络结构、软件和协议的角度来看,它与承运商的路由器可能并没有区别。路由器 F 是否属于承运商的子网仍是有争议的,但是对于本章的目标而言,客户网络周边上的路由器被认为是子网的一部分,因为它们运行的算法与承运商的路由器上运行的算法相同(本章我们关注的是算法)。

这种网络配置的用法如下所述。如果有一台主机要发送一个分组,那么它将分组传送给最近的路由器,该路由器或者在它自己的 LAN 上,或者在一条通向承运商的点到点链路上。该分组将被存储在路由器上,一直到它完全到达路由器为止,所以路由器可以验证它的校验和。然后它被沿路转发到下一台路由器,直到到达目标主机为止,最后在目标

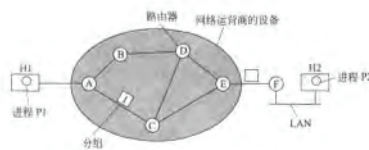


图 5.1 网络层协议的环境

主机上它被递交给相应的进程。这种机制即为存储-转发分组交换机制，在前一章我们已经看到过了。

### 5.1.2 向传输层提供的服务

在网络层/传输层的接口面上，网络层向传输层提供服务。一个重要的问题是，网络层应该向传输层提供哪种类型的服务。在设计网络层服务的时候，下面一些目标需要考虑：

- (1) 所提供的服务应该独立于路由器技术。
- (2) 路由器的数量、类型和拓扑关系对于传输层来说应该是不可见的。
- (3) 传输层可以使用的网络地址应该有一种统一的编址方案，甚至可以跨越多个 LAN 和 WAN。

给定了这些目标之后，网络层设计者有非常大的自由度来编写这些提供给传输层的服务的详细规范。这种自由度通常会演变为两个竞争派别之间的激烈争斗。最终讨论的焦点集中在网络层应该提供面向连接的服务还是提供无连接的服务。

一个阵营(以 Internet 社团为代表)认为，路由器的任务仅仅是传送分组，不用再做别的事情。按照他们的观点(根据 30 年来从一个实际可运行的计算机网络中获得的实践经验)，不管子网是如何设计的，从本质上讲它总是不可靠的。因此，主机应该接受“网络是不可靠的”这样的事实，并且自己来完成错误控制(即错误检测和纠正)和流控制任务。

这种观点很快地导致了这样的结论：网络服务应该是无连接的，基本的原语是 SEND PACKET 和 RECEIVE PACKET，以及少量其他的原语。特别是，分组的排序和流控制不应该在这里完成，因为主机将会完成这些工作，做两遍同样的工作通常不会带来额外的好处。而且，每个分组必须携带完整的目标地址，因为每个被发送的分组在传输过程中独立于它前面的那些分组(如果此前还有分组的话)。

另一大阵营(以电话公司为代表)认为，子网应该提供可靠的、面向连接的服务。他们声称，全球电话系统 100 年来的成功经验是一个很好的范例。在他们的观点中，服务质量是最主要的因素，如果在子网中没有连接，那么要实现服务质量是很难的，特别对于诸如语音和视频这样的实时流量。

这两大阵营都有最好的例证，即 Internet 和 ATM 来说明他们的观点。Internet 提供



了无连接的网络层服务；ATM 网络提供了面向连接的网络层服务。然而，有趣的是，随着服务质量变得越来越重要，Internet 也在不断进化。特别是，它现在正在努力获得一些通常跟面向连接服务关联在一起的特性，后面我们将会看到这一点。实际上，我们在第 4 章学习 VLAN 的时候，已经大略地看到了这种进化。

### 5.1.3 无连接服务的实现

在看过网络层所提供的两类服务之后，我们现在来看一看网络层的内部是如何工作的。根据所提供的服务类型，可能有两种不同的组织。如果提供的是无连接的服务，那么，所有的分组都被独立地传送到子网中，并且独立于路由，不需要提前建立任何辅助设施。在这样的上下文环境中，分组通常称为数据报 (datagram)，类似于电报 telegram)，且子网称为数据报子网 (datagram subnet)。如果使用了面向连接的服务的话，那么，在发送数据分组之前，必须首先建立起一条从源路由器到目标路由器之间的路径。这个连接称为一个 VC (virtual circuit, 虚电路)，类似于电话系统中建立的物理电路，且子网称为虚电路子网 (virtual-circuit subnet)。在本小节中，我们将讨论数据报子网；在下一小节中，我们将讨论虚电路子网。

现在我们来看一看数据报子网是如何工作的。假设图 5.2 中的进程 P1 有一个很长的消息要发送给 P2。它将消息递交给传输层，告诉传输层将该消息递交给主机 H2 上的进程 P2。传输层代码运行在 H1 上，通常在操作系统内部。它在消息的前面加上一个传输头，然后将结果交给网络层，这里的网络层可能只不过是操作系统内部的另一个过程。

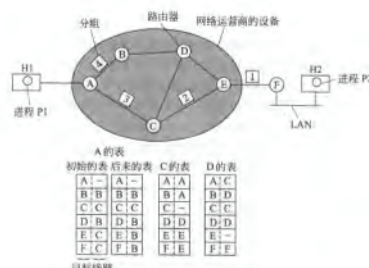


图 5.2 数据报子网内的路由

我们假设消息的长度是最大分组长度的 4 倍，所以，网络层必须将消息分割成 4 个分组：1、2、3 和 4，然后利用某一种点到点协议（比如 PPP）将这些分组依次发送给路由器

A。到这里,网络承运商将传输任务接管过来。每一台路由器都有一个内部表,它指明了:针对每一个可能的目标地址应该将分组送到哪里去。每一个表项包含两个元素:一个是目标地址,另一个是针对该目标地址所使用的输出线路。当然,路由器只允许使用直接连接的线路。例如,在图 5.2 中,A 只有两条输出线路(分别通向 B 和 C),所以,每一个进来的分组必须被发送给这两台路由器之一,即使它的目标地址是其他某一台路由器也是如此。A 的初始路由表如图中的“初始表”所示。

当分组 1、2 和 3 到达 A 的时候,它们被暂时保存起来(以便检验它们的校验和)。然后,根据 A 的路由表,每一个分组被转发给 C。然后分组 1 被转发给 E,进一步被转发给 F。当它到达 F 的时候,它被封装到一个数据链路层的帧中,通过 LAN 被发送给 H2。分组 2 和 3 也经过了同样的路径。

然而,分组 4 的情形有所不同。当它到达 A 之后,尽管它的目标也是指向 F,但是它被发送给路由器 B。出于某种原因,A 决定采用不同于前三者的路径来发送分组 4。有可能它了解到在 ACE 路径上发生了流量拥塞,因而更新了路由表,如图中的“后来的表”所示。管理这些路由表并作出路由选择的算法称为**路由算法(routing algorithm)**。这是我们本章中将要学习的一个主题。

#### 5.1.4 面向连接服务的实现

对于面向连接的服务,我们需要一个虚电路子网。现在我们来讨论它是如何工作的。隐藏在虚电路背后的思想是,不需要像图 5.2 那样每次为一个分组选择一条新的路径。相反,当一个连接被建立起来的时候,从源机器到目标机器之间的一条路径被选择作为连接的一部分,并且保存在中间这些路由器的内部表中。对于所有在这个连接上通过的流量,都使用这条路径,这与电话系统的工作方式完全一致。当连接被释放之后,虚电路也随之终止。在面向连接的服务中,每个分组都包含一个标识符,指明了它属于哪一个虚电路。

作为一个例子,请考虑图 5.3 的情形。在这里,主机 H1 已经建立了与主机 H2 之间的连接 1。在每一个路由表中,该连接都被记录在第一个表项中。A 的路由表的第一行说明了:如果一个分组包含了连接标识符 1,并且来自于 H1,那么它将被发送到路由器 C,并且赋予连接标识符 1。类似地,C 中的第一个表项将该分组路由到 E,也赋予连接标识符 1。

现在我们来考虑一下,如果 H3 也希望与 H2 建立连接则情形会怎么样。H3 选择连接标识符 1(因为它是在发起连接,而且这是它惟一的连接),并且告诉子网要建立虚电路。这导致了路由表中的第二行。请注意,这里有了一个冲突,因为,尽管 A 很容易就能够区分来自于 H1 的连接 1 分组和来自于 H3 的连接 1 分组,但是,C 无法区分这两种分组。由于这个原因,A 给第二个连接的输出流量分配一个不同的连接标识符。这种避免冲突的做法正说明了为什么路由器需要具备“在输出分组中替换连接标识符”的能力。在有些上下文环境中,这被称为**标签交换(label switching)**。

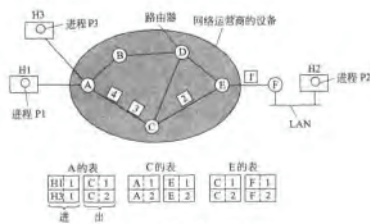


图 5.3 虚电路子网内的路由

#### 5.1.5 虚电路子网和数据报子网的比较

虚电路和数据报都有各自的支持者和反对者。这里试图从多个角度对两者作一下总结。图 5.4 列出了最主要的比较项目，当然，完美主义者总是有可能找到图中各项的反例。

比较项目	数据报子网	虚电路子网
建立电路	不需要	要求
地址信息	每个分组包含完整的数据地址和目标地址	每个分组包含一个简短的 VC 号
状态信息	路由器不保留任何有关连接的状态信息	每个 VC 都要求路由器为每个连接建立表项
路由	每个分组被独立地路由	当 VC 建立的时候选择路径，所有的分组都沿着这条路径
路由器失效的影响	没有，除非在崩溃过程中分组丢了	所有经过此失效路由器的 VC 都将终止
服务质量	很难实现	如果有足够的资源可以提前分配给每一个 VC，则很容易实现
拥塞控制	很难实现	如果有足够的资源可以提前分配给每一个 VC，则很容易实现

图 5.4 数据报和虚电路子网的比较

在子网内部，虚电路和数据报之间存在几个折衷。一个是路由器的内存空间和带宽之间的平衡。虚电路机制使得分组只要包含电路号即可，而不用包含完整的目标地址。如果分组都很短的话，那么，每个分组中的完整目标地址可能需要消耗相当多的开销，因而会浪费带宽。因为使用虚电路而付出的代价是路由器内部的表空间。根据通信电路和路由器之间的相对价格情况，可能虚电路更合算，也可能数据报更合算。

另一个平衡点是建立虚电路所需要的时间和地址解析的时间。在使用虚电路的时

候,它要求有一个建立阶段,这个阶段既要花费时间,也要消耗资源。然而,在一个虚电路子网中处理一个数据分组的方法却非常简单:路由器只要使用电路号作为索引,在内部表中找到该分组的目标去向即可。在一个数据报子网中,路由器需要执行一个相对复杂的查找过程,以便定位到该目标的表项。

另一个问题是在路由器内存中所要求的表空间的数量。在数据报子网中,针对每一个可能的目标地址都要求有一个表项,而在虚电路子网中,只要为每一个虚电路提供一个表项即可。然而,这种优势并非绝对,因为建立连接的分组也需要被路由,它们也使用目标地址,如同数据报子网的做法一样。

从保证服务质量,以及在子网内避免拥塞的角度来讲,虚电路有一些优势,因为当建立连接的时候,虚电路子网可以提前预留资源(比如缓冲区空间、带宽和 CPU 周期)。当分组开始陆续到来之后,所需要的带宽和路由器 CPU 资源都已经准备就绪了。而对于数据报子网,要想避免拥塞是非常困难的。

对于事务处理系统(比如商场通过电话来验证信用卡购物的有效性),用于建立和清除虚电路所需要的开销有可能会妨碍虚电路的使用。如果系统中大量的流量都是这种类型的话,那么在子网内部使用虚电路就毫无意义了。另一方面,永久虚电路,即手工建立起来并且会持续几个月或者几年的虚电路,有可能在这里会很有意义。

虚电路也还有脆弱性的问题。如果一台路由器崩溃了,或者内存中的数据丢失了,那么,即使它一秒钟之后又重新启动了,所有从它这里经过的虚电路都将不得不被中断。相反,如果一台数据报路由器停止了,则只有当时还有分组尚留在路由器队列中的用户会受到影响,甚至这些用户也不全部受到影响,这取决于这些分组是否已经被确认了。一条通信线路的失败对于使用了该线路的虚电路来说是致命的,但如果使用了数据报的话,则这种失败很容易得到补偿。对于数据报子网,路由器可以平衡通信流量,因为在传输一个很长的分组序列过程中,路由器可以在半中间改变传输路径。

## 5.2 路由算法

网络层的主要功能是将分组从源机器路由到目标机器中。在大多数子网中,分组需要经过多跳(hop)才能到达目的地。惟一个值得指出的例外是广播式网络,但即使在广播式网络中,如果源机器和目标机器不在同一个网络中,则路由仍然是一个问题。选择路径的算法以及这些算法所使用的数据结构是网络层设计最主要的内容。

**路由算法(routing algorithm)**是网络层软件的一部分,它负责确定一个进来的分组应该被传送到哪一条输出线路上。如果子网内部使用了数据报,那么路由器必须针对每一个到达的数据分组重新选择路径,因为从上一次选择了路径之后,最佳的路径可能已经改变了。如果子网内部使用了虚电路,那么只有当一个新的虚电路被建立起来的时候,才需要确定路由路径。因此,数据分组只要沿着已经建立的路径向前传递就行了。后一种情形有时候也称为会话路由(**session routing**),因为在一个完整的用户会话(比如一个终端上的登录会话或者一个文件的传输)过程中,传输路径必须保持有效。

对路由(即确定该使用哪一条路径)和转发(即当一个分组到达的时候所采取的动作)

进行区分有时候是非常有用的。你可以想象路由器的内部有两个进程。其中一个进程在每个分组到达的时候对它进行处理,它在路由表中查找该分组所对应的输出线路。这个进程即为**转发(forwarding)**。另一个进程负责填充和更新路由表,这正是路由算法起作用的地方。

无论是针对每个分组独立地选择路由路径,还是只有建立新连接的时候才选择路由路径,在一个路由算法中有些特性总是期望能够具备:正确性、简单性、健壮性、稳定性、公平性和最优性。正确性和简单性无需多加解释,但是乍看起来对健壮性的要求则没有那么显然了。一旦一个重要的网络投入运行了,它有可能需要连续运行数年而不能有全局性的失败。在此期间,将会有各种各样的硬件和软件失败。主机、路由器和线路可能会经常性地失败,网络拓扑结构也可能会多次发生变化。路由算法应该能够处理拓扑结构和流量方面的各种变化,而且不要求所有主机都停止所有的工作,并且每当某台路由器崩溃的时候也不要求网络重新启动。

稳定性对于路由算法来说也是一个重要的目标。有一些路由算法无论运行多长时间,它们都不会达到平衡。一个稳定的算法会达到平衡,并且保持平衡状态不变。公平性和最优性听起来非常显然——肯定不会有人反对这两种特性,但是,事实上,它们通常是两个相互矛盾的目标。举一个简单的例子来说明这种冲突,请看图 5.5。假设在 A 和 A' 之间、B 和 B' 之间以及 C 和 C' 之间有足够流量使得水平的链路达到饱和。为了使总流量达到最大,X 和 X' 之间的流量应该完全被切断。不幸的是,X 和 X' 可能看不到这一点。很显然,在全局效率和单个连接的公平性之间必须有一种折衷的处理办法。

我们在试图找到公平性和最优性之间的平衡办法之前,首先必须确定要优化的到底是什么。使分组的平均延迟最小,这是一种很显然的选择,但是,使网络总的吞吐量最大化也是一种选择。而且,这两个目标也是相互冲突的,因为对于任何队列系统,在接近容量的情况下进行操作意味着会有很长的排队延迟。作为一种折衷,许多网络企图使一个分组必须经过的跳数最小化,因为降低了跳数,实际上相当于减小了延迟,也减少了所消耗的带宽数量,从而也提高了吞吐量。

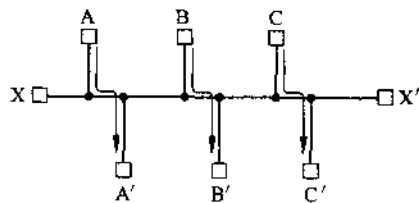


图 5.5 公平性和最优性之间的冲突

路由算法可以分成两大类:非自适应的和自适应的。**非自适应的算法(nonadaptive algorithm)**不会根据当前测量或者估计的流量和拓扑结构,来调整它们的路由决策。相反,从 I 到 J(对于所有的 I 和 J)所使用的路由选择是预先在离线情况下计算好的,在网络启动的时候被下载到路由器中。这个过程有时候也称为**静态路由(static routing)**。

与此相反,**自适应算法(adaptive algorithm)**则会改变它们的路由决策,以反映出拓扑结构的变化,通常也会反映出流量的变化情况。不同的自适应算法在多个方面有所不同:获取信息的来源不同(例如,来自于本地、来自相邻的路由器,或者来自于所有的路由器)、改变路径的时间策略不同(比如,每隔  $\Delta t$  秒、当负载变化的时候,或者当拓扑结构改变的时候)、用于优化的度量不同(比如,距离、跳数或者估计的传输时间)。在本节中,我们将



讨论各种各样的路由算法,包括静态的和动态的。

### 5.2.1 优化原则

在讨论具体的算法之前,我们可以在不考虑网络拓扑结构和流量的情况下,先对最优路径给出一条一般性的论述。这条论述也称为**最优优化原则(optimality principle)**。它声明如下:如果路由器J是在从路由器I到路由器K的最优路径上,那么,从J到K的最优路径也必定沿着同样的路由路径。为了更清楚理解这一点,我们将从I到J的路径部分记作 $r_1$ ,余下的路径部分记作 $r_2$ 。如果从J到K还存在一条路由路径比 $r_2$ 更好的话,那么,它可以与 $r_1$ 串接起来,从而得到一条更好的从I到K的路由路径,这与 $r_1 r_2$ 是最优路径的假设相违背。

最优优化原则的一个直接结果是,从所有的源到一个指定目标的最优路径的集合构成了一棵以目标节点为根的树。这样的树称为**汇集树(sink tree)**,如图 5.6 所示,图中的距离度量是跳数。请注意,汇集树不必是惟一的,可能存在其他的树也具有同样的路径长度。路由算法的目标是为所有的路由器找到并使用汇集树。

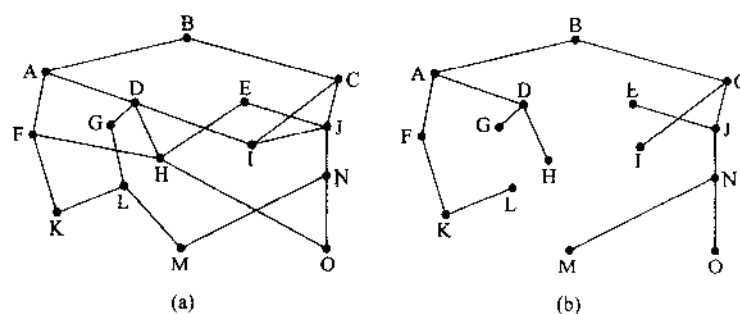


图 5.6

(a) 一个子网, (b) 路由器 B 的汇集树

由于汇集树确实是一棵树,它不会包含任何环,所以每个分组将在有限跳数之内被递交给目标主机。然而,实际的情形并非如此简单。在运行过程中,链路和路由器可能会断开或停止工作,然后又恢复工作,所以,不同的路由器对于当前的拓扑结构可能会有不同的看法。而且,我们也必须做出设计决定:每台路由器是否独立地获取那些用于计算汇集树的信息,或是通过其他的方法来收集这些信息。稍后我们将回到这个问题上来。不管怎么样,最优优化原则和汇集树为各种路由算法提供了一个衡量基准。

### 5.2.2 最短路径路由

现在我们开始讨论各种切实可行的路由算法,首先介绍一项有多种应用形式并被广泛使用的技术,之所以先从这项技术开始,是因为它非常简单而且易于理解。它的基本思路是,建立一个子网图,图中的每个节点代表一台路由器,每条弧代表一条通信线路(通常称为一条链路)。为了在一对给定的路由器之间选择一条路由路径,路由算法只需在图中找到这对节点之间的最短路径即可。

最短路径的概念值得做一些解释。一种衡量路径长度的方法是跳数。若采用这种度量方法,则图 5.7 中的 ABC 和 ABE 是等长的。另一种度量是以 km(千米或公里)为单位的地理距离,在这种情况下,ABC 明显比 ABE 长很多(假定该图是按照比例来画的)。

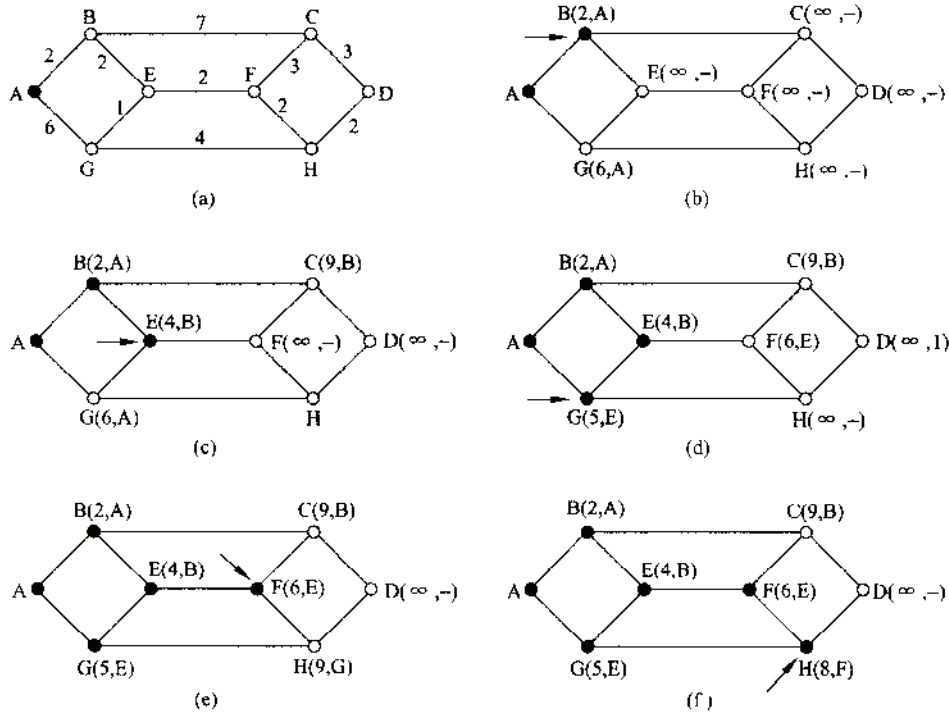


图 5.7 计算从 A 到 D 的最短距离的前 5 步,图中的箭头表明的是工作节点

然而,除跳数和物理距离之外,其他许多度量也是可能的。例如,我们可以用一个标准的测试分组进行几个小时的测试,然后用平均的排队和传输延迟来标记每一条弧。如果采用了这种标记方法,则最短路径就变成了最快的路径,而不是弧数最少或者距离最短的路径。

一般而言,弧段上面的标记可以是距离、带宽、平均流量、通信开销、平均队列长度、测量得到的延迟以及其他因素的一个函数,得通过计算出来。通过改变这个加权函数,路由算法就可以根据任何一种准则或者多种准则的组合来计算“最短”路径。

现在已经有了一些计算图中两个节点之间最短路径的算法。这应该归功于 Dijkstra (1959)。每一个节点都有一个标记(在圆括号内),标出了从源节点开始沿着当前已知的最佳路径到达该节点的距离。初始时,所有的路径都不知道,所以,所有的节点都被标记为无限远。随着算法的不断进行,陆续有一些路径被找到,于是节点的标记可能会发生变化,以反映出更好的路径。每个标记可能是暂时的,也可能是永久的。初始的时候,所有的标记都是暂时的。当已经发现一个标记代表了从源节点到该节点的最短可能路径的时候,该标记变成永久的,以后不再改变。

为了说明标记算法的工作过程,请看一下图 5.7(a)中的带权无向图,这里的权值代

表了一种度量,比如说距离。我们现在想找到从 A 到 D 的最短路径。开始时,我们将节点 A 标记为永久的,在图中用一个实心的圆来表示。然后我们依次检查每一个与 A(即工作节点,working node)相邻的节点,并且用它们与 A 之间的距离重新对它们进行标记。每当一个节点被重新标记的时候,我们也要标记出这次探查动作是从哪个节点发起的,因而以后我们可以重构出最终的路径。在检查了每一个与 A 相邻的节点之后,我们再检查整个图中所有暂时性标记的节点,并且使得其中具有最小标记的那个节点成为永久性的,如图 5.7(b)所示。然后,这个节点变成新的工作节点。

现在我们从 B 开始,检查所有与 B 相邻的节点。对于每一个与 B 相邻的节点,如果节点 B 上的标记加上从 B 到该节点的距离小于该节点原来的标记,那么我们找到了一条更短的路径,所以该节点需要重新标记。

在检查了所有与工作节点相邻的节点,并且在可能的情况下修改了暂时性的标记之后,算法需要对整个图进行搜索,以便找到具有最小标记值的暂时性节点。这个节点将变成永久性节点,并且成为下一轮的工作节点。图 5.7 显示了算法的前 5 个步骤。

为了看清楚该算法的工作原理,请看图 5.7(c)。在这个点上,我们刚刚把 E 变成永久性节点。假设存在一条比 ABE 还要短的路径,比如说 AXYZE,则有两种可能性:节点 Z 已经是永久性的,或者它还没有变成永久性节点。如果 Z 是永久性的,则 E 已经被探查过了(当 Z 变成永久性节点之后的下一轮会探查 Z 的相邻节点),所以路径 AXYZE 不会逃脱我们的搜索范围,因而它不可能是一条最短路径。

现在考虑 Z 仍然是暂时性标记的情形。如果节点 Z 上的标记大于或者等于 E 上的标记,则 AXYZE 不可能是一条比 ABE 更短的路径;如果 Z 上的标记小于 E 上的标记,则应该首先是 Z 而不是 E 变成永久性节点,然而再从 Z 探查 E。

图 5.8 给出了这个算法。全局变量 `n` 和 `dist` 描述了节点图。在 `shortest_path` 被调用之前这两个变量已经被初始化了。这段程序与前面描述的算法之间的惟一区别是,在图 5.8 所示的程序中,我们从终端节点 `t` 开始计算最短路径,而不是从源节点 `s` 开始。由于在一个无向图中,从 `t` 到 `s` 的最短路径与从 `s` 到 `t` 的最短路径是相同的,所以,无论从哪一个节点开始其实并没有多大关系(除非有几条最短路径,在这种情况下,逆向搜索有可能会发现不同的最短路径)。之所以选择后向搜索的原因是,每个节点都被标记了在它之前搜索的节点,而不是在它之后搜索的节点。因此,当最终的路径被复制到输出变量 `path` 中的时候,该路径也要被翻转过来。通过翻转搜索结果,两种效果相互抵消,因此,答案按正确的顺序产生。

### 5.2.3 扩散法

另一种静态算法是扩散法(flooding)。在扩散法中,每一个进来的分组将被发送到除了它进来的那条路线之外的每一条输出线路上。很显然,扩散法会产生大量的重复分组,实际上,除非采取某一种办法来抑制扩散过程,否则扩散法将会产生无限多的分组。一种办法是,在每个分组的头中包含一个跳计数器,经过每一跳之后该计数器减一,当计数器到达 0 的时候该分组被丢弃。理想情况下,跳计数器的初始值应该等于从源到目标之间路径的长度。如果发送方并不知道该路径有多长的话,它可以将计数器的初始值设置为

最坏情形下的长度,即子网的直径。

```
#define MAX_NODES 1024          /* maximum number of nodes */
#define INFINITY 1000000000     /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES]; /* dist[i][j] is the distance from i to j */
void shortest_path(int s, int t, int path[])
{ struct state {                /* the path being worked on */
    int predecessor;            /* previous node */
    int length;                 /* length from source to this node */
    enum {permanent, tentative} label; /* label state */
} state[MAX_NODES];
int i, k, min;
struct state * p;
for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
    p->predecessor = -1;
    p->length = INFINITY;
    p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;          /* k is the initial working node */
do {
    /* Is there a better path from k? */
    for (i = 0; i < n; i++) /* this graph has n nodes */
        if (dist[k][i] != 0 && state[i].label == tentative) {
            if (state[k].length + dist[k][i] < state[i].length) {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }

    /* Find the tentatively labeled node with the smallest label. */
    k = 0; min = INFINITY;
    for (i = 0; i < n; i++)
        if (state[i].label == tentative && state[i].length < min) {
            min = state[i].length;
            k = i;
        }
    state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0; k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

图 5.8 从子网节点图计算最短路径的 Dijkstra 算法

抑制扩散的另一种技术是记录下哪些分组已经被扩散过了,从而避免再次发送这些分组。为了实现这个目标,一种做法是让源路由器在它接收到的来自于其主机的每个分组中放上一个序列号。然后,在每个路由器上,针对每个源路由器都需要一个列表,其中列出了那些已经见到过的、来自于该源路由器的序列号。如果一个进来的分组已经位于该列表中,那么它不用再扩散了。

为了避免该列表无限地膨胀,每个列表应该使用一个计数器  $k$  作为参数,它表示一直到  $k$  的所有序列号都已经看到过了。当一个分组进来的时候,很容易就可以检查该分组是否为一个重复分组;如果是的话,则丢弃该分组。而且, $k$  以下的整个列表就不再需要了,因为  $k$  本身已经有效地对这部分列表作了概括。

一个稍微实际一点的扩散法变种是选择性扩散(selective flooding)。在这个算法中,路由器并不是将每个进来的分组都输出到每一条线路上,而是只输出到那些大概方向正确的线路上。将一个应该向西传送的分组发送到一条向东的线路上通常是没有意义的,除非网络拓扑结构极为怪异并且路由器也知道这样的事实。

在许多应用中扩散法并不很实用,但它还是有一些用处的。例如,在军事应用中,大量的路由器有可能在一瞬间被炸得粉碎,所以扩散法的高度健壮性正是所期望的效果。在分布式数据库应用中,有时候需要同时更新所有的数据库,在这种情况下扩散法可能会非常有用。在无线网络中,一个站送出的所有消息可以被位于该站的无线距离范围内的所有站接收到,这实际上正是扩散法,有些算法利用了这种特性。扩散法的第四种可能的用途是把它当作一种度量标准,用来比较其他的路由算法。扩散法总是选择最短的路径,因为它并行地选择每一条可能的路径。因此,没有其他的算法能够产生更短的延迟(如果我们忽略因扩散进程本身而产生的开销的话)。

#### 5.2.4 距离矢量路由

现代的计算机网络通常使用动态的路由算法,而不是上面介绍的静态路由算法,因为静态算法不会考虑网络的当前负载情况。其中有两个动态算法最为常见,即距离矢量路由算法和链路状态路由算法。在这一小节中,我们看一看距离矢量路由算法;在下一小节中,我们将讨论链路状态路由算法。

距离矢量路由(distance vector routing)算法是这样工作的:每个路由器维护一张表(即一个矢量),表中列出了当前已知的到每个目标的最佳距离,以及所使用的线路。通过在邻居之间相互交换信息,路由器不断地更新它们内部的表。

距离矢量路由算法有时候也被叫作其他的名称,最为常见是分布式 Bellman-Ford 路由算法和 Ford-Fulkerson 算法,这两种叫法都是根据其设计者的名字来命名的(Bellman, 1957; Ford and Fulkerson, 1962)。距离矢量路由算法最初是 ARPANET 使用的路由算法,也被用于 Internet 的 RIP 协议。

在距离矢量路由算法中,每个路由器维护了一张路由表,它以子网中的每个路由器为索引,并且每个路由器对应一个表项。该表项包含两部分:为了到达该目标路由器而首选使用的输出线路,以及到达该目标路由器的时间估计值或者距离估计值。所使用的度量可能是跳数,或者是以毫秒计算的时间延迟,或者是沿着该路径排队的分组数目,或者



其他类似的值。

假定路由器知道它到每一个邻居的“距离”。如果所用的度量是跳数的话,那么该距离只是 1 跳。如果所用的度量为队列长度的话,那么路由器只需检查每一个队列即可。如果度量值为延迟的话,则路由器的测量方法很简单,它只要直接发送一个特殊的 ECHO 分组,而后接收方加上时间戳,并且尽可能快地送回来即可。

举例来说,假设使用延迟作为度量标准,并且路由器知道它到每一个邻居路由器的延迟。每隔  $T$  毫秒(ms),每个路由器向它的每一个邻居发送一个列表,其中包含了它到每一个目标的延迟估计值;同时,它也从每一个邻居路由器接收到一个类似的列表。假设一个路由器接收到来自邻居  $X$  的一个列表,其中  $X_i$  表示  $X$  估计的到达路由器  $i$  所需要的时间。如果该路由器知道它到  $X$  的延迟为  $m$  毫秒,那么它也知道在  $X_i + m$  毫秒之内经过  $X$  可以到达路由器  $i$ 。一个路由器针对每一个邻居都执行这样的计算,就可以发现最佳的估计值,然后新的路由表中使用这个最佳的估计值以及对应的输出线路。请注意,在计算过程中不使用老的路由表。

这个更新过程如图 5.9 所示。其中(a)部分显示了一个子网。(b)部分的前 4 列显示了  $J$  从邻居路由器接收到的延迟矢量。 $A$  声称它到  $B$  有 12ms 的延迟,到  $C$  有 25ms 的延迟,到  $D$  有 40ms 的延迟,等等。假定  $J$  已经测量和估计了它到邻居  $A$ 、 $I$ 、 $H$  和  $K$  的延迟分别为 8、10、12 和 6 ms。

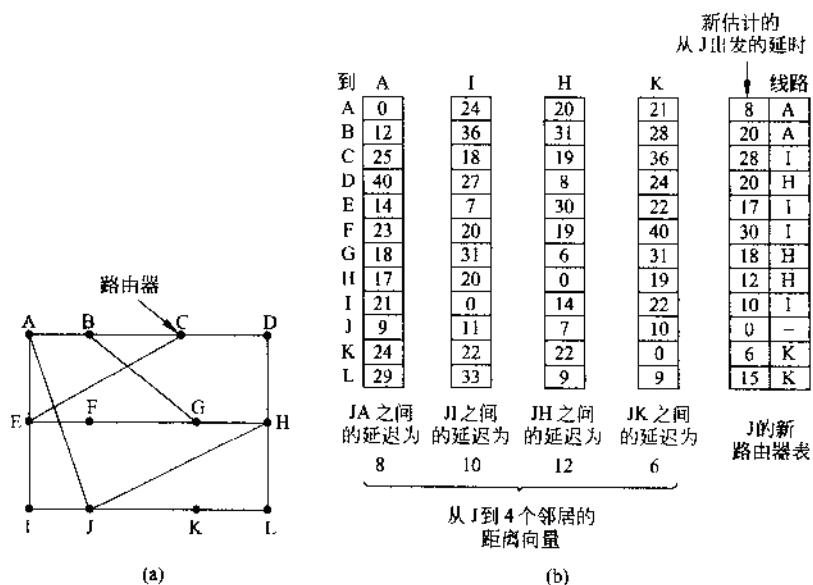


图 5.9

(a) 一个子网; (b) 从 A、I、H、K 的输入, 以及 J 的新路由表

现在考虑  $J$  如何计算它到路由器  $G$  的新路径。它知道在 8ms 之内可以到达  $A$ , 并且  $A$  声称可以在 18ms 内到达  $G$ , 所以  $J$  知道, 如果它将这些目标地址为  $G$  的分组转发给  $A$  的话, 那么它到  $G$  的延迟为 26 ms。类似地, 它计算出经过  $I$ 、 $H$  和  $K$  到达  $G$  的延迟分别

为  $41(31+10)$ 、 $18(6+12)$  和  $37(31+6)$  ms。在这些值之中,最好的结果是 18,所以在 J 的路由表中,对应于 G 的表项中的延迟值为 18 ms,所用的路径经过 H。对于所有其他的目标也执行同样的计算过程,最后得到的新路由表如图中最后一列所示。

### 无穷计算问题

距离矢量路由算法在理论上可以工作,但是在实践中它有一个严重的缺陷:虽然它总是能够得到正确的答案,但是它收敛到正确答案的速度可能非常慢。尤其是,它对于好消息的反应非常快,但是对于坏消息的反应非常迟缓。请考虑这样一个路由器,它到目标 X 的最佳路径非常大。如果在下一次交换信息的时候,邻居 A 突然报告说它到 X 有一个非常短的延迟,那么,该路由器只需将发送给 X 的流量切换到通向 A 的线路上。经过一次矢量交换,好消息就起作用了。

为了看清楚好消息的传播有多快,请考虑图 5.10 中的 5 节点(直线型)子网,这里的延迟度量为跳数。假定 A 最初处于停机状态,所有其他的路由器都知道这一点。换句话说,它们都将到 A 的延迟记录为无穷大。

A	B	C	D	E	
•	•	•	•	•	
	$\infty$	$\infty$	$\infty$	$\infty$	初始时
	1	$\infty$	$\infty$	$\infty$	第 1 次交换后
	1	2	$\infty$	$\infty$	第 2 次交换后
	1	2	3	$\infty$	第 3 次交换后
	1	2	3	4	第 4 次交换后

(a)

A	B	C	D	E	
•	•	•	•	•	
	1	2	3	4	初始时
	3	2	3	4	第 1 次交换后
	3	4	3	4	第 2 次交换后
	5	4	5	4	第 3 次交换后
	5	6	5	6	第 4 次交换后
	7	6	7	6	第 5 次交换后
	7	8	7	8	第 6 次交换后
	•	•	•	•	...

(b)

图 5.10 无穷计算问题

当 A 启动时,其他的路由器通过矢量交换知道了这一点。为了简化起见,我们假定有一个全局的定时器,它定期地激发所有的路由器同时进行矢量交换。在第一次交换的时候,B 知道它左边的邻居到 A 的延迟为 0。于是 B 在它的路由表中建立一个表项,说明 A 在它左边一跳的位置上。所有其他的路由器仍然认为 A 是停机的。这时候,针对 A 的路由表项如图 5.10(a)中的第二行所示。在接下去的交换中,C 知道 B 到 A 的路径长度为 1,所以它更新自己的路由表,指明它到 A 的路径长度为 2,但是 D 和 E 要到以后才能听到这个好消息。很显然,好消息扩散的速度是每交换一次往远处走一跳。如果一个子网中最长的路径是 N 跳的话,那么经过 N 次交换之后,每个路由器都将知道新恢复的线路和路由器。

现在我们考虑图 5.10(b)的情形,在这里所有的线路和路由器最初都是打开的。路由器 B、C、D 和 E 到 A 的距离分别为 1、2、3 和 4。突然间 A 停机了,或者 A 和 B 之间的线路断了,实际上,从 B 的角度来看,这两者是相同的。

在第一次分组交换的时候,B 没有听到来自 A 的任何信息。幸运的是,C 说:别担心,我有一条通向 A 的长度为 2 的路径。B 并不知道 C 的路径是经过 B 自身的。B 所知

道的是,C可能有10条线路,每条都独立地通向A,并且长度为2。因此,B认为它可以通过C到达A,路径长度为3。在第一次交换之后,D和E并不更新它们的A表项。

在第二次交换的时候,C注意到,它的每一个邻居都声称有一条通向A的长度为3的路径。它随机地挑选出一条,并且将它到A的距离更新为4,如图5.10(b)中的第三行所示。通过后续的交换,可以得到图5.10(b)中余下的记录历史。

从这个图中,我们应该很清楚地看出为什么坏消息传播得很慢:没有一个路由器知道一个比它的所有邻居的最小值大超过1的值。逐渐地,所有的路由器都会趋向无穷大,但是所需交换的次数依赖于代表无穷大的数值。由于这样的原因,明智的做法是将无穷大设置为最长的路径加1。如果使用时间延迟作为度量标准的话,那就没有明确定义的上限值了,所以,需要一个比较大的值来代表无穷大,这样可以避免将一条很长延迟的路径当成断路来处理了。因此,这个问题称为**无穷计算(count-to-infinity)**。现在有一些工作试图要解决这个问题(比如RFC 1058中的毒性逆转的水平分裂法),但是总体而言,没有哪一种办法真正解决得很好。这个问题的核心在于,当X告诉Y它有一条路径的时候,Y无从知道它自己是否就在这条路径上。

### 5.2.5 链路状态路由

在1979年以前,ARPANET一直使用距离矢量路由算法,而在此之后则被替换为链路状态路由算法。之所以发生这种更换主要是由于两个问题。第一,由于延迟度量是队列长度,所以在选择路径的时候并没有考虑线路带宽。刚开始的时候,所有的线路都是56kbps,所以,线路带宽并不是一个因素,但是当有些线路被升级到230kbps,另外一些线路被升级到1.544Mbps之后,带宽因素变成了一个重要的问题。当然,将延迟度量做一下修改,把线路带宽作为一个因素考虑进去也是有可能的。但是,第二个问题仍然存在,即距离矢量路由算法需要很长时间才能收敛到稳定状态(无穷计算的问题)。由于这些原因,距离矢量路由算法被一个全新的算法所替代,该算法称为**链路状态路由算法(link state routing)**。现在,链路状态路由算法的许多变种算法得到了广泛的应用。

链路状态路由算法背后的思想非常简单,可以用5个部分加以描述。每一个路由器必须完成以下的工作:

- (1) 发现它的邻居节点,并知道其网络地址。
- (2) 测量到各邻居节点的延迟或者开销。
- (3) 构造一个分组,分组中包含所有它刚刚知道的信息。
- (4) 将这个分组发送给所有其他的路由器。
- (5) 计算出到每一个其他路由器的最短路径。

实际上,完整的拓扑结构和所有的延迟信息都使用实验的方法来测量,并分发给每一个路由器。然后每个路由器运行Dijkstra算法就可以找出从它到每一个其他路由器的最短路径。下面我们详细地考虑上述的每一个步骤。

#### 发现邻居节点

当一个路由器启动的时候,它的第一个任务是找出哪些路由器是它的邻居。为了实

现这个目标,它只需在每一条点到点线路上发送一个特殊的 HELLO 分组即可。线路另一端的路由器应该送回一个应答来说明它是谁。这些名字必须是全局惟一的,因为当一个远程路由器以后听到有三个路由器都连接到 F 上的时候,它必须能够确定这三个路由器所提到的 F 是否是同一个路由器 F。

当两个或者多个路由器通过一个 LAN 连接起来的时候,情形会稍微复杂一些。图 5.11(a)显示了一个 LAN 直接连接到三个路由器 A、C 和 F 上的情形。如图中所示,每个路由器都连接到一个或者多个其他的路由器上。

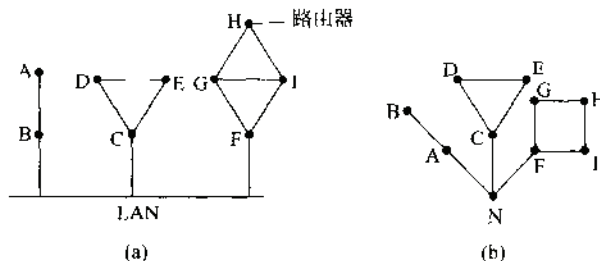


图 5.11

(a) 9 个路由器和一个 LAN; (b) (a) 的一个图形模型

一种对 LAN 建模的方法是将 LAN 本身当作一个节点来考虑,如图 5.11(b)所示。这里我们引入了一个新的人工节点 N,它与 A、C 和 F 连接起来。在 LAN 上从 A 到 C 是可能的,所以在这里可以表示为路径 ANC。

### 测量线路开销

链路状态路由算法要求每一个路由器知道它到各个邻居节点之间的延迟,或者至少有一个合理的估计值。为了确定这份延迟信息,最直接的办法是在这条线路上发送一个特殊的 ECHO 分组,另一端必须立即送回一个应答。通过计算出往返时间,再除以 2,则发送方路由器就可以得到一个合理的延迟估计值。如果要想得到更好的结果,则可以多次执行这样的测试过程,然后取它们的平均值。当然,这种方法隐含了一个假设,即两个方向上的延迟是对称的,而在实践中并不总是这样的。

一个有趣的话题是,在测量延迟的时候是否将负载考虑进去。如果需要考虑负载的因素,则用于计算往返时间的定时器必须在 ECHO 分组进入队列的时候开始启动;如果忽略负载的因素,则定时器应该在 ECHO 分组到达队列头部的时候开始启动。

这两种方法都可能会有争议。因流量因素而导致的延迟也被包含在测量结果之中,这意味着当一个路由器面对两条具有相同带宽的线路的时候,由于一条线路总是有很重的负载,而另一条线路总是负载较轻,所以,路由器在选择的时候总是将负载较轻的线路作为较短的路径来考虑。这样的选择将会导致良好的性能。

不幸的是,在计算延迟的时候将负载考虑进去也有争议。请考虑图 5.12 中的子网,该子网被分成东西两部分,这两部分通过两条线路 CF 和 EI 连接起来。

假设东部和西部之间的流量大多数使用了线路 CF,因此,这条线路的负载比较重,从

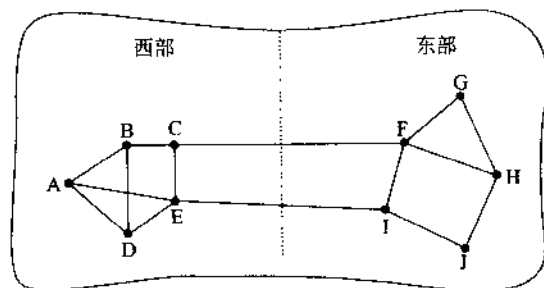


图 5.12 通过两条线路将东西两部分连接起来的子网

而导致较长的延迟。如果在计算最短路径的时候将排队的延迟也算进去的话，则 EI 变得更有吸引力。当新的路由表被建立起来之后，东西部之间的大多数流量都从 EI 上经过，从而使这条线路负载超重。因此，在下一次更新路由表的时候，CF 将成为最短路径。这样的结果是，路由表可能变得震荡不定，从而导致一些古怪的路由问题和许多潜在的其他问题。如果忽略负载的因素而只考虑带宽的话，这个问题就不会发生。另外一种选择是，将负载分散在这两条线路上，但是这种方案不能充分利用最佳的路径。然而，为了避免在选择最佳路径过程中的震荡现象，一种有可能算是比较明智的做法是，利用先验知识将负载合理地分布在多条线路上。

### 创建链路状态分组

一旦所需要的交换信息已经收集到了，每个路由器的下一步工作是建立一个包含所有这些数据的分组。该分组的内容首先是发送方的标识，接着是一个序号(Seq)和年龄(Age, 后而再介绍)，以及一个邻居列表。对于每个邻居，同时也要给出到这个邻居的延迟。图 5.13(a)显示了一个例子子网，每条线路上标出了延迟信息。这六个路由器所对应的链路状态分组如图 5.13(b)所示。

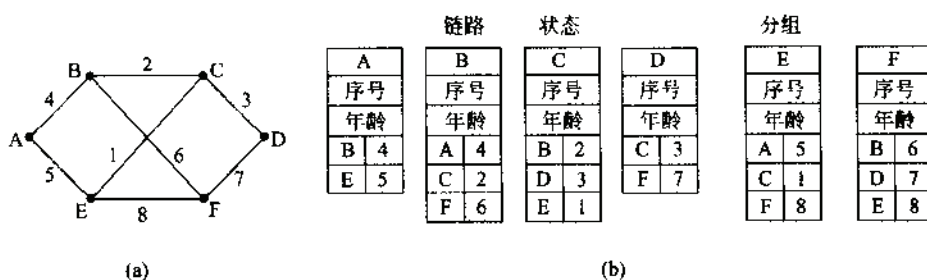


图 5.13

(a) 一个子网；(b) 该子网的链路状态分组

创建链路状态分组很容易。难的是确定什么时候创建分组。一种可能的做法是定期地创建分组，也就是说，过了一定的间隔之后创建链路状态分组。另一种做法是，当某些重要的事情发生的时候才创建分组，比如当一条线路断掉或者一个邻居节点停机时，或当



它们重新恢复运行时,或当它们的特性发生了适当变化时。

### 发布链路状态分组

链路状态路由算法最技巧的部分是如何可靠地发布链路状态分组。当分组被发布出去并且被安装以后,首先获得分组的路由器将改变它们的路由路径。因此,不同的路由器有可能使用不同版本的拓扑结构,从而导致了不一致性、环、不可达的机器,或者其他的问题。

首先我们将描述最基本的发布算法,然后再对它进行提炼和改进。基本的思路是,使用扩散法来发布链路状态分组。为了控制扩散过程,每一个分组都包含一个序列号,序列号随着每一个新的分组而递增。每个路由器记录下它所看到的所有(源路由器、序列号)对。当一个新的链路状态分组进来的时候,路由器在已经看到的分组列表中检查这个新进来的分组。如果它是新的,那么除了它到来的那条线路之外,在其他的线路上全部转发该分组。如果它是一个重复分组,则将它丢弃。如果一个分组的序列号小于当前所看到过的来自该源路由器的最大序列号,则它将被当作过时分组而拒绝,因为该路由器已经有了更新的数据。

这个算法还有一些问题,不过,这些问题是可以克服的。第一,如果序列号回转(即达到了最大的整数值之后再加 1 又回到最小值)了,则可能会产生混淆。这里的解决方案是使用一个 32 位的序列号。即使每秒钟产生一个链路状态分组,那也需要 137 年才可能发生回转,所以,这种可能性可以忽略不计。

第二,如果一个路由器崩溃了,那么它将丢失所有的序列号记录。如果它再从 0 开始,那么,下一个分组将被作为重复分组而拒绝。

第三,如果一个序列号被破坏了,比如发送方发送的序列号为 4,但是由于产生了 1 位错误,所以接收方看到的序列号是 65 540,那么,序列号从 5 到 65 540 的分组都将被当作过时分组而拒绝,因为接收方认为当前的序列号是 65 540。

所有这些问题的解决方案是在每个分组的序列号之后包含年龄信息(age),并且每秒钟将年龄减 1。当年龄到 0 的时候,来自该路由器的信息被丢弃。通常情况下,每隔一段时间,比如说 10 秒钟,一个新的分组就会到来,所以,只有当一个路由器停机的时候(或者 6 个连续的分组被丢失,这种情形发生的可能性不大),路由器信息才会超时。在初始扩散过程中,每个路由器也要递减 age 域,这样可以确保没有分组会丢失,也不会生存不定长的时间(如果一个分组的 age 为 0,则它要被丢弃)。

对这个算法做一些精炼可以使它更加健壮。当一个链路状态分组被扩散到一个路由器中的时候,它并没有立即被加入到队列中等待传输。相反,它首先被放到一个保留区中等待一段时间。如果在这个分组被送出去之前,另一个来自于同一个源路由器的链路状态分组也到来了,那么它们的序列号就需要作比较。如果它们相等,则重复分组被丢弃。如果它们不相等,则老的分组被丢掉。为了防止在路由器至路由器之间的线路上产生错误,所有的链路状态分组都要被确认。当一条线路空闲的时候,路由器就对保留区进行循环扫描,以便选择一个分组或者确认并将其发送出去。

在图 5.13(a)所示的子网中,路由器 B 所使用的数据结构如图 5.14 所示。这里的每

一行对应于一个刚刚到达的,但是还没有完全处理完毕的链路状态分组。该表记录了分组的来源、序列号和年龄,以及数据。而且,从 B 到 A、C 和 F 的三条线路中,每条记录针对每一条线路都有发送和确认标志。发送标志表明该分组必须在所指示的线路上被发送出去。确认标志表明它必须在这条线路上被确认。

源	序列号	年龄	发送标志			确认标志			数据
			A	C	F	A	C	F	
A	21	60	0	1	1		10	0	
F	21	60	1	1	0	0	0	0	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

图 5.14 图 5.13 中路由器 B 的分组缓冲区

在图 5.14 中,来自 A 的链路状态分组可以直接到达,所以 B 必须将它发送给 C 和 F,但是向 A 送回确认,图中的标志位说明了这一点。类似地,来自 F 的分组必须被转发给 A 和 C,并且向 F 送回确认。

然而,第三个分组,即来自 E 的分组有所不同。它到达两次,一次经过 EAB,另一次经过 EFB。因此,它只需被发送给 C,但是要向 A 和 F 确认,正如它的标志位所示。

如果一个重复分组到来的时候原来的分组仍然在缓冲区中,那么相应的标志位必须要作相应的改变。例如,如果在表中的第四个表项被转发出去之前,C 的状态的有一份副本从 F 到达,那么,这六位将被改变为 100011,以表明该分组必须也要向 F 确认,但是不用转发给 F 了。

### 计算新的路由路径

一旦一个路由器已经获得了全部的链路状态分组之后,它就可以构造出完整的子网图了,因为每条链路都已经被表示出来了。实际上,每条链路被表示了两次,每个方向各表示一次。这两个值可以取平均,也可以分开使用。

现在可以在路由器本地运行 Dijkstra 算法,以便构建出到所有可能目标的最短路径。路由器可以将该算法的结果安装在路由表中,然后恢复正常的操作。

对于一个具有  $n$  个路由器的子网,每个路由器有  $k$  个邻居,那么,用于存储输入数据所要求的内存与  $kn$  成正比。对于规模比较大的子网,这可能是一个问题。而且,计算时间可能也是一个考虑因素。不过,在许多实践环境中,链路状态路由算法工作得很好。

然而,硬件或者软件问题也可能会破坏这个算法(对于其他算法也一样)。例如,如果一个路由器声称它有一条线路,但是实际上并没有这条线路,或者它有一条线路但是忘记了这条线路,那么,算法所得到的子网图将是不正确的。如果一个路由器不能转发分组,或者在转发分组的时候破坏了这些分组,那么同样也会引起麻烦。最后,如果路由器耗尽了内存,或者在路由计算的时候产生了错误,则一样会出现问题。当子网的规模达到几万或者几十万个节点的时候,某一台路由器偶尔失败的可能性就变得不可忽视了。解决这些问题的技巧在于,当不可避免的事故有的时候,尽可能将这种损害限制在最小的范围

内。Perlman(1988)详细地讨论了这些问题以及相应的解决办法。

链路状态路由算法被广泛地应用于实际的网络中,所以现在提一下某些使用该算法的例子协议。在 Internet 中被广泛使用的 OSPF 协议就用到了链路状态算法。我们将在 5.6.4 小节中介绍 OSPF。

另一个链路状态协议是 IS-IS(Intermediate System-Intermediate System,中间系统对中间系统),它是专门为了 DECnet 而设计的,后来被 ISO 采纳,用于无连接网络层协议 CLNP。自此以后,它也被作了修改以便能够处理其他的协议,例如最著名的 IP 协议。IS-IS 也被用在一些 Internet 骨干网上(包括早先的 NSFNET 骨干网),以及一些数字蜂窝系统中,比如 CDPD。Novell Netware 使用了 IS-IS 的一个小的变种(NLSP)来路由 IPX 分组。

基本上,IS-IS 分发了一个路由器拓扑结构图,路由器从这个图中能够计算出最短路径。每个路由器在它的链路状态信息中宣称,哪些网络层地址是它可以直接到达的。这些地址可以是 IP、IPX、AppleTalk 或者任何其他地址。IS-IS 甚至可以同时支持多种网络层协议。

OSPF 采纳了 IS-IS 设计中的许多创意(OSPF 是在 IS-IS 之后几年才设计出来的)。这些创意包括:一种扩散链路状态更新信息的自稳定方法、在 LAN 上指派路由器(designated router)的概念、计算路径分裂和支持多种度量标准的方法。因此,IS-IS 和 OSPF 之间的差异非常小。最重要的差别是,在 IS-IS 的编码方法中,同时携带多个网络层协议的信息非常容易,也非常自然,这是 OSPF 不具备的特性。IS-IS 的这种优势对于大型的多协议环境特别有价值。

### 5.2.6 分级路由

随着网络规模的增长,路由器的路由表也成比例地增长。不断增长的路由表不仅消耗路由器的内存,而且还需要更多的 CPU 时间来扫描路由表,需要更多的带宽来发送有关的状态报告。当网络增长的时候,可能会达到这样一个点:在这个点上每个路由器不太可能再为其他每一个路由器维护一个表项,所以,路由选择过程必须分级进行,就好像电话网络中的做法那样。

在采用了分级路由之后,路由器被划分成区域(region),每个路由器知道如何将分组路由到自己所在区域内的目标地址,但是对于其他区域的内部结构毫不知情。当不同的网络被相互连接起来的时候,很自然地就会将每个网络当作一个独立的区域,这样做的好处是,一个网络中的路由器不必知道其他网络的拓扑结构。

对于大型的网络,一个两级的分层结构可能还不够;可能有必要将区域组织成群(cluster),将群组织成区(zone),将区组织成组(group),等等,直到将所有的集合名词用完为止。举一个多级分层结构的例子,请考虑如何将一个分组从加利福尼亚州的伯克利路由到肯尼亚的马琳迪。伯克利的路由器知道加利福尼亚州的详细拓扑结构,但是可能将所有州际的流量发送给洛杉矶(加州的一个城市)的路由器。洛杉矶的路由器能够将流量路由给美国其他的路由器,但是,它会将国际流量发送到纽约。纽约的路由器可以直接将所有的流量发送至目标国家中负责处理国际流量的路由器,比如肯尼亚的内罗毕。最

后,该分组将沿着肯尼亚国家中的树往下传送,一直到达马琳迪。

图 5.15 给出了一个两级层次的定量分析例子,该例子中包含了 5 个区域。路由器 1A 的完整路由表有 17 个表项,如图 5.15(b)所示。如果采用分级路由,则路由表如图 5.15(c)所示,所有针对本区域内的路由器的表项都跟原先一样,但是,所有其他的区域都被压缩到单个路由器中,所以,所有到区域 2 的流量都要经过 1B-2A 线路,其余的远程流量都经过 1C-3B 线路。分级路由使得路由器 1A 的路由表从 17 项降低为 7 项。随着区域数与每个区域中路由器数量之比值的增加,节省下来的表空间也随之增加。

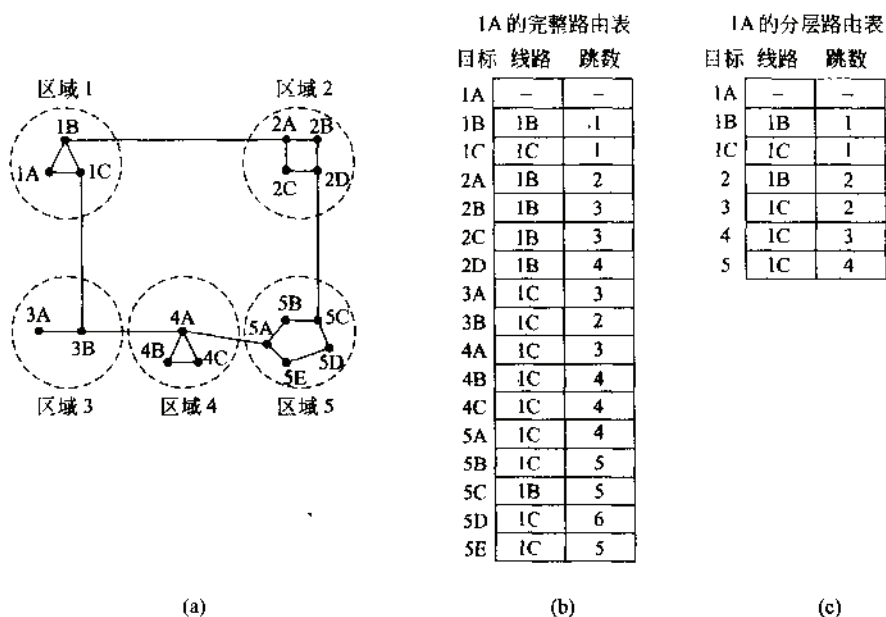


图 5.15 分级路由

不幸的是,这种空间的节省不是免费得来的。它需要付出代价,其代价的形式是增加了路径长度。例如,从 1A 到 5C 的最佳路径是经过区域 2,但采用了分级路由之后,所有到区域 5 的流量都要经过区域 3,因为对于区域 5 中的大多数目标来说这是更好的选择。

当一个单独的网络变得非常大的时候,一个有趣的问题是:应该分多少级才比较合适?例如,考虑一个具有 720 个路由器的子网。如果没有分级的话,那么每个路由器需要 720 个路由表项。如果子网被分成 24 个区域,每个区域 30 个路由器,那么每个路由器只需要 30 个本地表项,加上 23 个远程表项,总共 53 个表项。如果采用三级层次结构,总共 8 个群,每个群包含 9 个区域,每个区域 10 个路由器,那么,每个路由器需要 10 个表项用于记录本地路由器,8 个表项用于到同一群内其他区域的路由,7 个表项用于远程的群,总共 25 个表项。Kamoun 和 Kleinrock(1979)发现,对于一个包含  $N$  个路由器的子网,最优的级数是  $\ln N$ ,并且每个路由器要求  $e \ln N$  个表项。他们还证明了,由于分级路由而导致的有效平均路径长度的增长是非常小的,通常是可以接受的。

### 5.2.7 广播路由

在有些应用中,主机需要给其他多个或者所有主机发送消息。例如,用于发布天气预报的服务、股市行情最新报告或者现场直播节目等,它们的最佳工作方式是,将消息广播给所有的机器,然后让那些感兴趣的机器读取数据。同时给所有的目标发送一个分组,这称为**广播(broadcasting)**;为了实现广播,人们已经提出了各种各样的方法。

一种不要求子网具有任何特殊特性的广播方法是,让源机器简单地给每一个目标单独发送一个分组。这种方法不仅浪费带宽,而且也要求源机器拥有所有目标机器的完整地址列表。在实践中,这也许是惟一可能的做法,但它也是最不期望的做法。

扩散法是另一种很显然的候选方法。尽管扩散法并不适合于普通的点到点通信,但是对于广播而言,它可能很值得考虑,尤其当下面所介绍的方法都不适用的时候。用扩散法来实现广播所涉及到的问题,与用扩散法作为点到点路由算法的问题是一样的:扩散过程产生了太多的分组,从而消耗太多的带宽。

第三种算法是**多目标路由(multidestination routing)**。如果使用了这种方法的话,每个分组或者包含一组目标,或者包含一个位图,由该位图来指定所期望的目标。当一个分组到达一个路由器的时候,路由器检查所有的目标,以确定哪些输出线路是必要的(只要一条输出线路是到达至少一个目标的最佳路径,那么它就是必要的)。路由器为每一条需要用到的输出线路生成一份新的副本,在这份副本中只包含那些使用这条线路的目标。实际上,原来的目标集合被分散到这些输出线路上。在经过足够的跳数之后,每个分组只包含一个目标,因此可以被当作普通的分组来对待。多目标路由方法就如同单个地址的分组一样,只不过当多个分组必须沿着同样的路径被转发的时候,其中一个分组承担全部的费用,而其他的分组都是免费搭乘的。

第四种广播算法显式地使用了以发起广播的路由器为根的**汇集树(sink tree)**,或者也可以使用任何其他适当的**生成树(spanning tree)**。生成树是子网的一个子集,它包含所有的路由器,但是不包含任何环。如果每个路由器都知道它的哪些线路属于一棵生成树,那么,它就可以将一个进来的广播分组复制到除了该分组到来的那条线路之外的所有生成树线路上。这种方法可以最佳地使用带宽,并且所生成的分组也绝对是完成这项任务所需要的最少数量的分组。惟一的问题是,每个路由器必须都知道某一棵生成树才可以使用这种方法。有时候这样的信息是可以得到的(比如采用了链路状态路由算法的时候),但是有时候无法获得这样的信息(比如采用了距离矢量路由算法的时候)。

最后一种广播算法是,即使当路由器根本不知道任何关于生成树的信息的时候,也要试图达到前一种方法的近似效果。一旦挑明之后,你就会发现它的原理非常简单,其基本的想法称为**逆向路径转发(reverse path forwarding)**。当一个广播分组到达一个路由器的时候,该路由器对它进行检查,看它到来的那条线路是否是通常用来给广播源发送分组的那条线路。如果是的话,那就很有可能该广播分组是沿着最佳路径被转发过来的,因而是到达当前路由器的第一份副本。如果是这种情况,则路由器将该分组转发到除了到来的那条线路之外的所有其他线路上。然而,如果广播分组是从其他任何一条并非首选的到达广播源的线路上到来的话,该分组被当作一个可能的重复分组而丢弃。



逆向路径转发算法的一个例子如图 5.16 所示。(a)部分显示了一个子网,(b)部分显示了该子网中路由器 I 的一棵汇集树,(c)部分显示了逆向路径算法是如何工作的。在第一跳的时候,I 发送分组给 F、H、J 和 N,如汇集树中的第二行所示。这些分组中的每一个都是在通向 I 的首选路径(假定首选的路径都沿着汇集树)上到来的,在图 5.16(c)中字母外面加一个圆圈就表示这样的分组。在第二跳的时候,共产生了 8 个分组,其中,在第一跳接收到分组的路由器各产生 2 个分组。结果是,所有这 8 个分组都到达了以前没有访问过的路由器,其中 5 个是沿着首选线路到来的。在第三跳所产生的 6 个分组中,只有 3 个是在首选路径(在 C、E 和 K)上到来的,其他的都是重复分组。在经过第五跳和 24 个分组以后,广播过程终止了。相比之下,如果完全沿着汇集树的话,只需要 4 跳和 14 个分组。

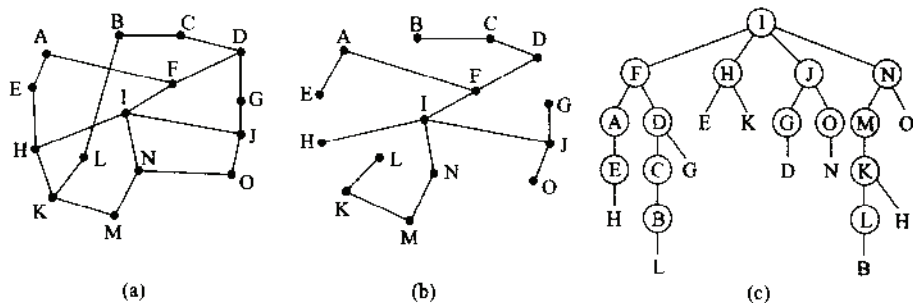


图 5.16 逆向路径转发

(a) 一个子网; (b) 汇集树; (c) 由逆向路径转发算法建立起来的树

逆向路径转发的主要优点是,它的效率相对合理,而且也易于实现。它并不要求路由器都知道生成树,也不像多目标路由方法那样要求每个分组中包含目标列表或者位图的开销。而且,它也不像扩散法那样要求特殊的机制来终止广播或扩散过程(在扩散法中,一种终止扩散的机制是,要求在每个分组中包含一个跳计数器,并且还需要提前知道子网的直径信息;另一种做法是,每个路由器为每个源维护一个已经看到过的分组的列表)。

### 5.2.8 多播路由

有些应用会有这样的要求:一些分布在各处的进程需要以组的方式协同工作,比如,一组进程实现了一个分布式数据库系统。在这种情形下,组中的进程常常需要给其他所有的成员发送消息。如果组的规模比较小,那么它只要以点到点的方式给每一个其他的成员发送消息即可。如果组的规模比较大,那么这种策略的代价就会非常昂贵。有时候也可以使用广播机制来实现这一点,但是,在一个有上百万个节点的网络中使用广播机制来通知 1000 台机器会显得非常低效,因为大多数接收者对于这样的消息并没有兴趣(甚至更糟的是,他们虽然有兴趣,但实际上他们不应该看到这些消息)。因此,我们需要有一种办法能够给一些有明确定义的组发送消息,这些组的成员数量虽然很多,但是与整个网络规模相比却很小。

给这样一个组发送消息称为多播(multicasting),它的路由算法称为多播路由



(multicast routing)。在这一小节中,我们将介绍一种实现多播路由的方法。有关多播路由的更多信息,请参见 Chu et al., 2000; Costa et al., 2001; Kasera et al., 2000; Madruga and Garcia-Luna-Aceves, 2001; Zhang and Ryu, 2001。

多播传输要求对组进行管理。首先需要有一种办法来创建和销毁组,并且允许进程加入或者离开组。路由算法并不关心如何实现这些任务。它关心的是,当一个进程加入组的时候,它需要把这个事实告诉它的主机。对于路由器来说,它们的哪些主机属于哪些组是非常重要的。当主机与组之间的从属关系发生变化的时候,主机必须将这些变化告诉路由器,或者由路由器定期地询问它们的主机。无论采用哪一种方式,路由器必须知道它们的哪些主机属于哪些组。路由器再告诉它们的邻居,所以,主机与组的从属信息在整个子网中传播开来。

为了实现多播路由,每个路由器计算一棵生成树,该生成树覆盖所有其他的路由器。例如,在图 5.17(a)中,我们有两个组:1 和 2。有的路由器上连接了一些属于一个或者两个组的主机,如图中所标出的那样。针对最左边路由器的一棵生成树如图 5.17(b)所示。

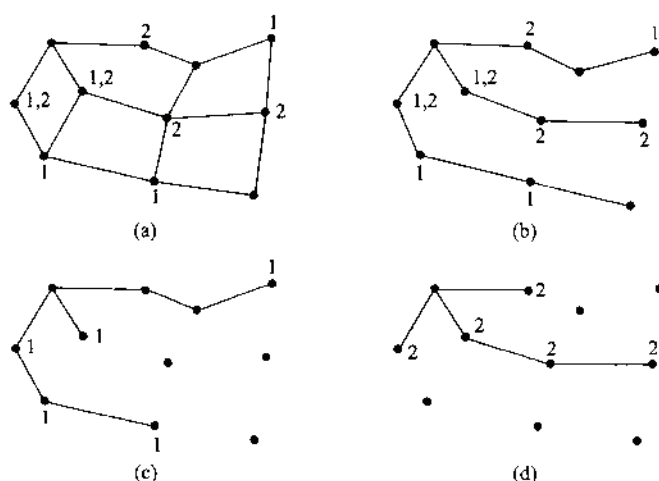


图 5.17

(a) 一个网络; (b) 最左边路由器的生成树;  
 (c) 针对组 1 的多播树; (d) 针对组 2 的多播树

当一个进程给一个组发送多播分组的时候,第一个路由器检查它的生成树,并对该树进行修剪,去掉那些并不通向该组成员主机的所有线路。在我们的例子中,图 5.17(c)显示了针对组 1 剪枝之后的生成树。类似地,图 5.17(d)显示了针对组 2 剪枝之后的生成树。多播分组只沿着正确的生成树被转发。

修剪生成树有很多种可能的方法。最简单的修剪方法用于那些采用了链路状态路由算法的网络,这时候每个路由器都知道完整的拓扑结构,包括哪些主机属于哪些组。然后,修剪工作从每条路径的末端开始,逐步向根路由器前进,同时去掉所有不属于相应组的路由器。

当使用了距离矢量路由算法的时候,修剪策略可能完全不同。基本的算法是逆向路径转发。然而,如果一个路由器接收到某一个特定组的多播消息,但是它的主机对这个组并不感兴趣,而且它也没有连接到其他的路由器,那么,它以一条 PRUNE 消息作为应答,告诉发送方以后不要再给它发送这个组的任何多播消息了。如果一个路由器自己的主机中没有组成员,并且它在所有的线路上都接收到这样的 PRUNE 消息,那么,它也以一条 PRUNE 消息作为应答。按照这种递归的方式,子网被修剪成一棵恰当的生成树。

这个算法一个潜在的缺点是,它很难扩展到大型的网络。假设一个网络有  $n$  个组,每个组平均有  $m$  个成员。对于每个组,有  $m$  棵被修剪之后的生成树需要保存,所以总共有  $mn$  棵树。如果有许多大规模的组,则存储所有这些树的开销将会非常可观。

另一种设计方法是使用基于核心的树(core-based tree,简称 CBT,参见 Ballardie et al., 1993)。在这种方法中,针对每个组只计算一棵生成树,其中的根(即核心)在接近组中间的位置上。如果一台主机需要发送多播消息,它只要将多播消息发送给核心路由器,然后核心路由器将会沿着生成树完成多播转发。尽管这棵树不可能对于所有的源都是最优的,但是,将每个组的存储开销从  $m$  棵树降低到一棵树,这是一个很大的节省。

### 5.2.9 移动主机路由

如今,成千上万的人拥有便携式计算机。通常,他们希望在任何一个地方都能够阅读最新的电子邮件,以及访问他们日常使用的文件系统。这些移动主机引入了一种新的复杂性:为了将一个分组路由到一台移动主机上,网络首先得找到这台主机。如何将移动主机融入到一个网络中,这是一个非常新的主题;在这一小节中我们将概略地介绍一些有关的问题,并且给出一种可能的解决方案。

网络设计者通常使用的世界模型如图 5.18 所示。图中有一个 WAN,它包含了一些路由器和主机。与 WAN 相连的是一些 LAN、MAN,以及我们在第 2 章中学习过的无线蜂窝单元。

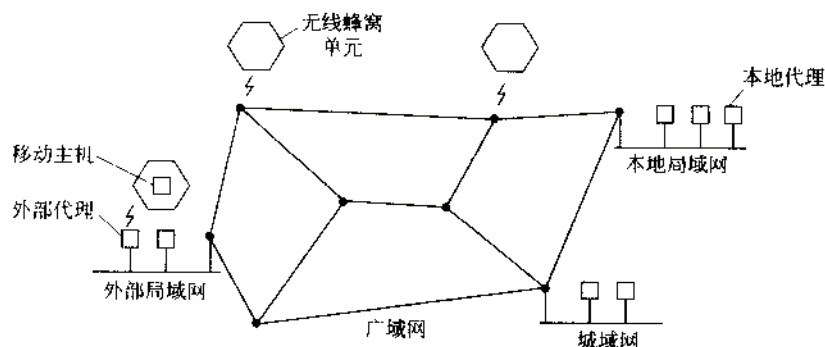


图 5.18 一个 WAN 连接了一些 LAN、MAN 和无线单元

永远不会移动的主机称为固定主机(stationary host)。它们通过铜线或者光纤连接到网络中。我们还可以区分另外两种其他类型的主机。一种是迁移主机(migratory host),它们基本上也是固定的,但是经常会从一个固定的站点移动到另一个固定的站点,

并且只有当它们物理上连接到网络的时候才使用网络。漫游主机(roaming host)实际上是在移动过程中执行计算,它们希望在移动的时候还能够保持与网络的连接。我们将使用术语**移动主机(mobile host)**来代表后两类主机,也就是说,移动主机是指那些离开了原始站点(home)还想继续连接网络的主机。

假设所有的主机都有一个永远不变的永久性**主场所(home location)**。主机也有一个永久性的主地址,用于确定它们的主场所,就好像用电话号码 1-21-5551212 来表示美国(国家代码 1)和曼哈顿(Manhattan,地区号 212)一样。在包含移动主机的系统中,路由算法的目标是,能够做到用移动主机的主地址给它们发送分组,而且不管这些主机在哪里,这些分组都能够被递交给它们。当然,这里的关键是如何找到这些主机。

在图 5.18 的模型中,整个世界(按地理位置)被分成许多小的单位,我们将这些单位称为区域,通常一个区域是一个 LAN 或者无线蜂窝单元。每个区域有一个或者多个**外部代理(foreign agent)**,所谓外部代理是指这样的进程:它们记录下所有当前正在访问该区域的移动主机。而且,每个区域有一个**本地代理(home agent)**,所谓本地代理是指这样的进程:它们记录下那些“主场在这个区域,但是当前正在访问其他区域”的主机。

当一台新的主机进入一个区域的时候,不管它是直接通过电缆连接到网络中(比如插入到 LAN 中),还是通过漫游方式进入到无线蜂窝单元中,该主机必须在外部代理那里注册自己。注册过程通常如下所述:

- (1) 每个外部代理周期性地广播一个分组,宣布它的存在以及地址。一个新到达的移动主机可能会等待这样的消息,但是,如果这样的消息不能够很快到来的话,那么移动主机可以广播一个询问分组:这里有外部代理吗?

- (2) 移动主机向外部代理请求注册,它提供自己的主地址、当前的数据链路层地址,以及一些安全信息。

- (3) 外部代理与移动主机的本地代理进行联系,告诉它:你的一个主机在我这里。从外部代理发送到本地代理的消息中包含了外部代理的网络地址。该消息也包含了相应的安全信息,以便让本地代理确信该移动主机确实在这个外部代理处。

- (4) 本地代理对安全信息进行检查,在安全信息中包含了一个时间戳,通过这个时间戳可以证明该消息是刚刚(几秒钟内)产生的。如果安全检查通过的话,则本地代理告诉外部代理可以继续进行。

- (5) 当外部代理得到了来自本地代理的确认之后,它在本地表中加入一个表项,并通知移动主机注册已经完成。

理想情况下,当一台主机离开一个区域的时候,它也应该宣布自己的离开请求,以便让外部代理注销自己,但是许多用户在完成了工作之后总是突然地关掉计算机。

当一个分组被发送给移动主机的时候,它将被路由到该主机的本地 LAN 中,因为那里才是该主机的地址所指的网路位置,如图 5.19 中的第 1 步所示。在图中,发送方位于 Seattle(西雅图)的西北城,它希望跨越美国大陆给纽约的一台主机发送一个分组。发送给移动主机的分组在纽约的本地 LAN 上被本地代理截获,然后本地代理查找该移动主机的新(临时)的场所,并且找到管理该移动主机的外部代理(在洛杉矶)的地址。

然后本地代理要做两件事情。第一,它将该分组封装在另一个外送分组的净荷域中。

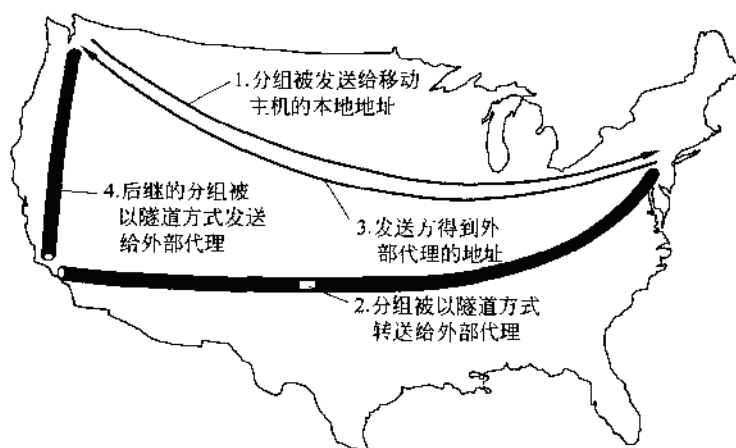


图 5.19 发送给移动主机的分组路由过程

并将此外送分组发送给外部代理,如图 5.19 中的第 2 步所示。这种机制称为隧道;后面我们还将更详细地讨论隧道。外部代理得到了这个被封装的分组之后,它从净荷域中提取出原来的分组,并且将它当作数据链路帧发送给移动主机。

第二件事是,本地代理告诉发送方,以后给移动主机发送分组的时候不要直接将这些分组发送到移动主机的主地址,而是将它们封装到新分组的净荷域中,这些新分组显式地使用外部代理的地址(第 3 步)。所以,后续的分组现在可以通过外部代理被直接路由给移动主机(第 4 步),完全绕过了移动主机的主场所。

现有的各种方案在几个方面有所不同。第一,存在这样的问题:该协议中有多少工作该由路由器来完成、有多少工作该由主机来完成,而对于主机而言,又该由哪一层来完成。第二,在少数方案中,沿途的路由器记录下映射的地址,所以在分组到达主场所之前,它们可以截获和重定向这些分组。第三,在有些方案中,每个进来访问的主机都可以获得一个惟一的临时地址;而在其他的方案中,此临时地址指向一个代理,该代理负责处理所有进来访问的主机的流量。

第四,对于那些目标地址与真正被递交的地址不相同的分组,各种方案有不同的处理方法。一种选择是改变分组的目标地址,然后重新传送修改之后的分组即可。另一种办法是,整个分组,连同主地址和所有其他的信息,都被封装到另一个分组的净荷域中,然后将该分组发送到临时的地址。最后,在安全性方面各种方案也有所不同。一般来说,当一台主机或者路由器接收到像“现在开始吧,请将 Stephany 的所有邮件发送给我”这种形式的消息的时候,它可能会有一连串的疑问,比如,谁在跟它通话,这样做是否合适,等等。下面的文献讨论并比较了几种移动主机协议: Hac and Guo, 2000; Perkins, 1998a; Snoeren and Balakrishnan, 2000; Solomon, 1998; Wang and Chen, 2001。

#### 5.2.10 Ad Hoc 网络中的路由

我们现在已经看到了当主机是移动的而路由器是固定的时候,路由工作是如何进行的。另一种更极端的情形是,路由器本身也是移动的。下面是一些可能的情形:

- (1) 在没有网络基础设施的战场上使用的军事设备。
- (2) 在大海上航行的船只。
- (3) 在地震中基础设施被破坏了以后,那些负责急救工作的人所使用的设备。
- (4) 一群配备了笔记本电脑的人聚集在一个没有 802.11 网络的地区。

在所有这样或那样的情形中,每个节点包含了一个路由器和一个主机,而且往往是在同一台机器上。如果一个网络中的节点正好是两两相邻的,那么该网络称为 **Ad Hoc 网络**,或者 **MANET(Mobile Ad hoc NETWORKS,移动 Ad hoc 网络)**。现在我们对 Ad Hoc 网络进行简略的讨论,更详细的信息请参见(Perkins, 2001)。

Ad Hoc 网络之所以区别于有线网络的原因在于,那些关于“固定拓扑结构、已知的固定邻居以及 IP 地址和场所之间的固定关系”等的常用规则都不再可用了。路由器可以来回走动,到了一个地点之后又走了,然后又立即出现在一个新的地点。对于一个有线网络,如果一台路由器有一条有效的路径可以通向某一个目标,那么该路径将会一直有效(除非系统中某个地方出现了故障)。而对于一个 Ad Hoc 网络,拓扑结构可能一直在发生变化,所以任何一条期望的有效路径可以在毫无警告的情况下发生变化。毋庸多说,这些情况使得 Ad Hoc 网络中的路由方法显著地不同于有线网络中的路由方法。

针对 Ad Hoc 网络目前已经提出了许多种路由算法。其中一个值得感兴趣的路由算法是 **AODV(Ad hoc On-demand Distance Vector)** 路由算法(Perkins and Royer, 1999)。它是 Bellman-Ford 距离矢量算法的远亲,但是可以在移动环境中工作,并且它考虑到了在这种环境下带宽有限和电源寿命相对较短的特殊限制。另一个比较不寻常的特性是,它是一个按需算法,也就是说,只有当有人想给某一个目标发送分组的时候,它才会计算到该目标的路由路径。现在我们来看一看这个算法。

### 路径发现

在任何时刻,Ad Hoc 网络都可以用一个节点(路由器+主机)图来描述。如果两个节点在它们的半径范围内可以直接通信的话,则将这两个节点连接起来(也就是说,在图中,它们之间有一条弧)。因为在两个节点之中,其中一个节点的发射器的功率比另一个大,所以有可能出现这样的情况:A 连接到 B,但是 B 并没有连接到 A。然而,为了简化起见,我们假设所有的连接都是对称的。同时还应该说明的是,两个节点都在对方的无线电距离范围内,仅仅这一条并不意味着它们就可以连接起来。它们之间可能有建筑物、山坡或者其他妨碍通信的障碍物。

为了描述 AODV 算法,请考虑图 5.20 中的 Ad Hoc 网络,在这个网络中,节点 A 上的一个进程想要给节点 I 发送分组。AODV 算法在每个节点上维护了一张表,该表格以目标节点作为关键字,每个表项给出了有关一个目标的信息,包括将分组发送给哪个邻居才可以到达这个目标。假设 A 检查自己的表,并没有发现针对 I 的表项。因此,它现在必须要找出一条通向 I 的路由路径。这种只有当需要的时候才发现路径的特性使得这个算法是一个“按需(on-demand)”算法。

为了定位到 I, A 构造一个特殊的 ROUTE REQUEST 分组,并且将它广播出去。该分组首先到达 B 和 D,如图 5.20(a)中所示。实际上,在图中,B 和 D 之所以连接到 A 的



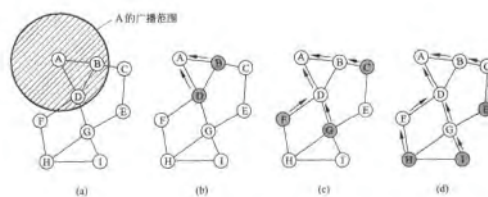


图 5.20

(a) A 的广播范围; (b) 在 B 和 D 接收到 A 的广播信息之后; (c) 在 C、F 和 G 接收到 A 的广播信息之后; (d) 在 E、H 和 I 接收到 A 的广播信息之后阴影节点是新的接收节点; 箭头显示了可能的逆向路由路径

原因是因为它们可以接收到来自 A 的通信流量。例如, 在图中, F 与 A 之间没有弧, 这是因为它不能够接收到 A 的无线电信号, 因此, F 与 A 没有直接连接。

ROUTE REQUEST 分组的格式如图 5.21 所示。它包含了源和目标地址, 通常是 IP 地址, 这两个地址标识出了谁在找谁。该分组中也包含了一个 RequestID, 这是由每个节点单独维护的一个本地计数器, 每当当一个节点广播 ROUTE REQUEST 消息的时候该计数器增一。源地址和 RequestID 两个域合起来惟一标识了一个 ROUTE REQUEST 分组, 从而当一个节点接收到 ROUTE REQUEST 分组的时候可以丢弃重复的分组。

源地址	请求 ID	目标地址	源序列号	目标序列号	跳计数
-----	-------	------	------	-------	-----

图 5.21 ROUTE REQUEST 分组的格式

每个节点除了维护请求 ID 计数器之外, 还维护了另一个序列计数器, 任何时候当它发送出一个 ROUTE REQUEST 分组(或者应其他人的 ROUTE REQUEST 分组)的时候, 该序列计数器增一。它的功能有点像时钟, 用来区别新的和老的路由路径。图 5.21 中的第四个域是 A 的序列计数器; 第五个域是 A 所看到过的、最近的 I 的序列号值(如果它从来没有看到过 I 的序列号则该值为 0)。通过下面的介绍, 你就会很清楚这些域的用法。最后一个域是跳计数(hop count), 它记录了该分组已经经过了多少跳。它的初始值为 0。

当 ROUTE REQUEST 分组到达一个节点(在本例中为 B 和 D)的时候, 该节点将按照下面的步骤进行处理。

(1) 在本地的历史表中查找(源地址, 请求 ID)对, 以确定是否曾经看到过并处理过该请求。如果这是一个重复分组, 那么丢弃该分组, 处理过程结束。如果它不是一个重复分组, 那么将这一对信息(源地址, 请求 ID)写到历史表中, 以便将来的重复分组可以被识别出来并加以拒绝; 然后处理过程继续进行。

(2) 接收方在它的路由表中查找该分组中的目标地址。如果查到一条较新的通向该目标的路径的话,则它给源节点送回一个 ROUTE REPLY 分组。这里所谓较新的路径是指存储在路由表中的目标序列号大于或者等于 ROUTE REQUEST 分组中的目标序列号。如果小于的话,那么当前节点存储的路由路径比该源节点以前发送给同一目标的路径还要老,于是执行第 3 步。

(3) 由于接收方并不知道通向该目标的较新的路由路径,所以,它增加跳计数域,并且重新广播 ROUTE REQUEST 分组。同时它也从分组中提取出数据,并且将它作为一个新的表项保存在逆向路由表中。这份信息将被用来构造逆向路由路径,因而以后应答分组可以回到源节点。图 5.20 中的箭头显示了逆向路由路径。针对这条新建立的逆向路由表项,同时也启动一个定时器。如果定时器到期的话,该表项即被删除。

B 和 D 都不知道 I 在哪里,所以,它们各自创建一个指向 A 的逆向路由表项,如图 5.20 中的箭头所示,同时它们也将分组广播出去,其中跳计数被设置为 1。B 发出的广播分组到达 C 和 D。C 在它的逆向路由表中加入一个表项,并且将分组重新广播出去。与此相反,D 拒绝该分组,因为这是一个重复分组。类似地,D 的广播分组也被 B 拒绝。然而,F 和 G 接受了 D 的广播分组,并且将它保存起来,如图 5.20(c)所示。当 E、H 和 I 接收到了广播分组之后,ROUTE REQUEST 最终到达了一个知道 I 所在地的目标节点,即 I 节点本身,如图 5.20(d)所示。请注意,虽然这里我们用三个分立的阶段来演示广播的过程,但实际上,不同节点的广播动作是单独进行的,没有任何协调处理。

I 接收到 ROUTE REQUEST 请求分组之后,它建立一个 ROUTE REPLY 分组作为应答,分组结构如图 5.22 所示。其中源地址和目标地址都直接复制自接收到的请求分组,但是目标序列号则取自当前内存中的计数器。跳计数域被设置为 0。Lifetime 域控制该路径在多长时间内有有效。I 节点用单播(unicast)方式将该分组传送给 G,因为它从 G 接收到 ROUTE REQUEST 分组。然后该分组沿着逆向路径到达 D,最后到达 A。在每个节点上,跳计数域都被增一,所以,每个节点都可以看到自己离目标(I)有多远。

源地址	目标地址	目标序列号	跳计数	Lifetime
-----	------	-------	-----	----------

图 5.22 ROUTE REPLY 分组的格式

回程中的每个中间节点都要检查该分组。如果下面 3 个条件中的 1 个或者多个满足的话,它们就会把这个分组的信息放到本地路由表中,作为到 I 的一条路由路径:

- (1) 目前还没有通向 I 的路由路径。
- (2) 在 ROUTE REPLY 分组中 I 的序列号大于路由表中的值。
- (3) 序列号相等,但是新的路径更短。

通过这种方式,逆向路径上的所有节点都免费得到了通向 I 的路由路径,这算是 A 的路径发现过程中的一种副产品。而对于那些“虽然接收到了前面的 ROUTE REQUEST 分组,但是并不位于逆向路径上”的节点(在本例中包括 B、C、E、F 和 H),当相关联的定时器到期的时候,逆向路由表中相应的表项自然被丢弃。

在一个大型的网络中,AODV 算法会产生许多广播分组,即使对于比较近的目标也是如此。用下面的办法可以减少广播分组的数量。发送方将 IP 分组的 TTL 域(Time to

live)初始化为期望的网络直径,在每一跳上该域被减1。如果该域减到0,则分组被丢弃,因而不被广播了。

因此,发现过程可以作如下修改。为了定位到一个目标,发送方广播一个 ROUTE REQUEST 分组,其中 TTL 域设置为 1。如果在一段合理的时间内没有应答分组回来,则再发送一个 ROUTE REQUEST 分组,这次将 TTL 域设置为 2。以后的每一次尝试分别使用 3、4、5,等等。按照这种方法,搜索过程首先在本地进行,然后扩大到越来越宽的环形范围内。

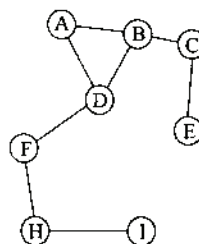
### 路径维护

因为节点可以移动或者被关闭,所以,网络的拓扑结构自然就会发生变化。例如,在图 5.20 中,如果 G 关闭了,那么 A 并不会立即意识到它刚刚用过的通向 I 的路径(ADGI)已经不再有效了。路由算法必须能够处理这样的情况。每个节点定期地广播一个 HELLO 消息,并且期望它的邻居会作出应答。如果没有应答到来的话,发送广播消息的节点就知道它的邻居已经离开了无线电接收范围,不再跟自己有连接了。类似地,如果它试图给一个邻居发送分组的时候没有得到应答,那么它也知道该邻居已经不再有效了。

这些信息可以被用来清除掉那些不再有效的路由路径。对于每一个可能的目标,每个节点(比如说 N)记录下满足如下条件的邻居:在最近的  $\Delta T$  秒内曾经给它发送过到达该目标的分组。这些邻居也称为针对该目标的活动邻居(active neighbor)。N 的做法是,建立一个以目标节点为关键字的路由表,其中的每一个表项包含了到达该目标的输出节点(outgoing node)、到该目标的跳计数、最近的目标序列号以及针对该目标的活动邻居列表。在我们的例子拓扑结构中,节点 D 的一个可能的路由表如图 5.23(a)所示。

目标	下一跳	距离	活动邻居	其他的域
A	A	1	F,G	
B	B	1	F,G	
C	B	2	F	
E	G	2		
F	F	1	A,B	
G	G	1	A,B	
H	F	2	A,B	
I	G	2	A,B	

(a)



(b)

图 5.23

(a) 在 G 停机之前 D 的路由表; (b) G 停机之后的图

当 N 的任何一个邻居变得不可达的时候,它就检查它的路由表,看看哪些目标的路由路径用到了该不可达邻居。对于每一条这样的路径,它都要通知对应的活动邻居,告诉它们经过 N 的路径现在已不再有效了,必须从它们的路由表中清除掉。然后,活动邻居再告诉它们的活动邻居,以此类推,递归进行下去,直到所有依赖于该不可达节点的路径

全部从路由表中清除掉为止。

作为路径维护的一个例子,请考虑前面的例子,但是现在 G 突然停机了。改变之后的拓扑结构如图 5.23(b)所示。当 D 发现 G 已经死掉的时候,它检查自己的路由表,发现到达 E、G 和 I 的路径都要用到 G,针对这些目标的活动邻居集合是{A, B}。换句话说,A 和 B 的某些路由路径依赖于 G,所以 D 必须通知它们这些路径已经不再工作了。D 给它们发送一些分组以告诉它们路径有了变化,它们收到这些分组之后就会相应地更新它们的路由表。D 也会从它的路由表中清除掉 E、G 和 I 的表项。

从以上的描述中可能并不能清楚地看出 AODV 和 Bellman-Ford 之间的关键差别,实际上关键的差别在于,在 AODV 算法中,路由器节点并没有周期性地发送包含整个路由表的广播信息。此差别不仅节约了带宽,也节省了电源的消耗。

AODV 算法也能够完成广播和多播路由。有关细节的信息,请参考 Perkins and Poyer, 2001。Ad Hoc 路由是一个非常热门的研究领域,目前已经发表了大量的研究工作。其中一些论文包括(Chen et al., 2002; Hu and Johnson, 2001; Li et al., 2001; Raju and Garcia-Luna-Aceves, 2001; Ramanathan and Redi, 2002; Royer and Toh, 1999; Spohn and Garcia-Luna-Aceves, 2001; Tseng et al., 2001; Zadeh et al., 2002)。

### 5.2.11 对等网络中的节点查询

在网络领域中,一个相对比较新的现象是对等网络(peer-to-peer network)。在对等网络中,大量的人相互联系以便共享资源,通常这些人通过永久的有线连接接入到 Internet 上。有趣的是,对等技术的第一个大范围应用是一宗大众犯罪案件:5 千万 Napster 用户在没有得到版权许可的情况下交换那些受版权保护的歌曲,直到上了法庭并经过激烈的辩论之后 Napster 才被强行关闭。然而,实际上,对等技术有很多很有意思的合法用途。它与路由问题也有一点类似,但是跟我们前面已经学过的路由问题又不完全一样。不管怎么样,它值得我们稍作探讨。

对等网络之所以如此有趣的一个原因是,它是完全分布式的。所有的节点都是对称的,网络中没有中心控制,也没有层次组织。在一个典型的对等系统中,每个用户总会包含一些令其他用户感兴趣的信息。这些信息可能是自由软件、(不受版权限制的)音乐、图片,等等。如果用户数量非常大的话,那么,他们相互之间可能并不认识,而且他们也不知道该到哪里去寻找他们想要的信息。一种解决方案是建立一个巨大的中心数据库,但是,出于某些原因(比如没有人愿意维护这样的数据库),这种方案可能并不可行。因此,问题就演变成:在缺少中心数据库或者中心索引的情况下,一个用户如何找到一个包含了 he 想要的信息的节点。

我们假设每个用户都有一个或者多个数据项目,比如歌曲、图片、程序、文件,等等,其他的用户可能会对这些数据项目感兴趣。每个项目都有一个名字,形式为 ASCII 字符串。一个潜在的用户只知道该 ASCII 字符串名字,他想知道是否有人拥有这份内容的副本,如果有的话,他们的 IP 地址是什么。

举一个例子,考虑一个分布式的家谱数据库,每个家谱专家都有一些其祖先或者亲属的记录,他(或者她)把这些记录放到网上,记录形式可以是人物的图片、音频,甚至是视频

片断。不同的人可能会有相同的祖先,所以,一个祖先可能会有多份记录分布在多个不同的节点上。记录的名字采用人物的名字,命名方式是全局统一的。有一天,一个家谱专家在一份档案中发现了他的曾祖父的遗嘱,在这份遗嘱中曾祖父把他的金表袋留给了他的侄子。现在,家谱专家知道了曾祖父侄子的名字,他希望能够找到是否其他人有他的记录。因此,问题变成了,如果没有一个中心数据库,我们该如何找到谁有这样的记录呢?

研究人员已经提出了多种不同的算法来解决这个问题。我们将要介绍的一种算法的名字为 Chord(Dabek et al., 2001a; and Stoica et al., 2001)。下面简要地描述一下 Chord 的工作机理。Chord 系统是由  $n$  个参与对等网络的用户组成的,每个用户可能保存了一些记录,而且每个用户也保存了一些索引信息以供其他的用户使用。每个用户节点都有一个 IP 地址,通过一个散列函数 hash,该 IP 地址将被映射为一个  $m$  位的数值。Chord 使用 SHA-1 算法作为散列函数。SHA-1 是一个密码学算法,我们将在第 8 章讨论该算法。现在我们只需知道,它是这样一个函数:接受一个变长的字节串作为实参,然后生成一个高度随机的 160 位数值。因此,我们可以将任何一个 IP 地址转换成一个 160 位的数值,该数值称为节点标识符(node identifier)。

从概念上来看,所有的  $2^{160}$  个节点标识符以升序的方式被组织成一个大的圆环。有些节点标识符对应于参与网络的节点,但是大多数并没有对应的节点。在图 5.24(a)中,我们显示了当  $m=5$  时候的节点标识符环(只不过忽略了中间的那些弧段)。在这个例子中,标识符为 1、4、7、12、15、20 和 27 的那些节点对应于实际的网络节点,在图中用阴影圆来表示;其余的节点并不对应于实际的网络节点。

现在我们来定义函数  $\text{successor}(k)$ ,该函数的返回值是:在圆上,从节点  $k$  开始沿着顺时针方向遇到的第一个实际节点的节点标识符。例如,  $\text{successor}(6)=7$ ,  $\text{successor}(8)=12$ ,  $\text{successor}(22)=27$ 。

记录的名字(比如歌曲名、祖先的名字等等)也用 hash(即 SHA-1)函数进行散列处理,得到一个 160 位的数值,该数值称为键(key)。因此,为了将 name(一条记录的 ASCII 名字)转换成一个键,我们使用  $\text{key}=\text{hash}(\text{name})$ ,这是对 hash 函数的一次本地过程调用。如果一个人拥有一条家谱记录,其名字为 name,而且他希望其他人也可以使用该记录,那么,他首先建立一对关联(name, my-IP-address),然后请  $\text{successor}(\text{hash}(\text{name}))$  来保存这一对关联信息。如果针对这个名字有多条记录(位于不同的节点上),那么它们的关联信息将保存在同一个节点上。按照这种方法,数据库的索引信息被随机地分布在节点上。考虑到容错的因素,我们可以使用  $p$  个不同的散列函数,因此每一对关联信息将被保存在  $p$  个节点上,但是为了简化起见我们这里不考虑这种做法。

如果后来某个用户想要查找 name,那么,他首先对 name 进行散列运算以得到 key,然后使用  $\text{successor}(\text{key})$  就可以找到那个保存其索引关联信息的节点的 IP 地址。第一步非常容易,而第二步并不容易。在给定一个键的情况下,为了能够找到对应于该键的节点的 IP 地址,每个节点必须维护一些管理性的数据结构。其中之一是在节点标识环上它的后继节点的 IP 地址。例如,在图 5.24 中,节点 4 的后继节点是 7,而节点 7 的后继节点是 12。

现在查找过程如下所述。发请求的节点向它的后继节点发送一个分组,该分组中包



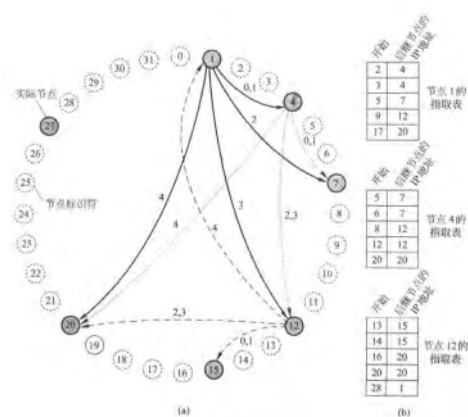


图 5.24

(a) 分布在图上的 32 个节点标识符, 阴影节点对应于实际的机器, 虚线显示了节点 1、4 和 12 的虚拟邻居。虚线上的标记是表中的索引。(b) 指取表的三个例子

含了它的 IP 地址, 以及待查找的记录的关键。该分组沿着环向前传播, 直到到达被查询的节点标识符的后继节点。该节点对分组进行检查, 看自己是否有合适的信息与该键匹配。如果有的话, 直接将这信息返回给最初发出请求的节点, 其 IP 地址就在分组中。

第一个优化措施是, 每个节点可以同时保留其后继节点和前继节点的 IP 地址, 所以查询请求既可以沿着顺时针方向传递, 也可以沿着逆时针方向传递, 取决于哪条路径更短。例如, 图 5.24 中的节点 7 沿着顺时针方向可以找到节点标识符 10, 而沿着逆时针方向可以找到节点标识符 3。

尽管有了两个方向可供选择, 但是, 对于一个大型的对等网络系统, 以线性方式搜索所有的节点仍然是非常低效的, 因为每次搜索涉及到的节点的平均数量为  $n/2$ 。为了加快搜索过程, 每个节点也要维护一张表, 在 Chord 中该表称为指取表 (finger table)。指取表中有  $m$  个表项, 索引值从 0 到  $m-1$ , 每个表项指向一个不同的实际节点。每个表项有两个域:  $start$  和  $successor(start)$  的 IP 地址, 图 5.24(b) 显示了三个例子节点的指取表。在节点  $k$  上第  $i$  个表项的两个域分别为:

$start = k + 2^i \pmod{2^n}$

successor(start[i]) 的 IP 地址

请注意,每个节点保存了相对少量的节点的 IP 地址,但是从节点标识符的角度来看,这些节点中的绝大多数都与当前节点较为接近。

有了指取表之后,在节点  $k$  上查找  $key$  的过程如下所述。如果  $key$  落在  $k$  和  $successor(k)$  之间,那么,  $successor(k)$  节点拥有关于  $key$  的信息,于是搜索过程终止。否则的话,节点  $k$  搜索指取表,找到这样的表项:其  $start$  域在  $key$  的前面但是最接近于  $key$ 。然后将请求直接发送给该表项中的 IP 地址,请它继续完成搜索过程。由于它更加接近  $key$  但仍然在  $key$  的前面,所以它的机会比原来的节点更好,它只需少量的查询就能够返回所要的答案。实际上,由于每次查询都使得到达目标节点的距离减小了一半,所以,可以证明,平均的查找次数为  $\log_2 n$ 。

我们来看第一个例子,请考虑在节点 1 上查找  $key=3$ 。由于节点 1 知道 3 位于 1 和它的后继节点 4 之间,所以,期望的节点是 4,搜索过程终止,并且返回节点 4 的 IP 地址。

现在看第二个例子,请考虑在节点 1 上查找  $key=14$ 。由于 14 并不位于 1 和 4 之间,所以需要检查指取表。在指取表中到达 14 最接近的节点是 9,所以该请求被转发给 9 表项中的 IP 地址,即节点 12 的 IP 地址。节点 12 看到 14 落在它和它的后继节点(15)之间,所以它返回节点 15 的 IP 地址。

下面看第三个例子,请考虑在节点 1 上查找  $key=16$ 。同样地,节点 1 将查询请求发送给节点 12,但是这次节点 12 无法知道答案了,所以,它在指取表中查找  $start$  域最接近于 16 但是在 16 之前的表项。它找到了  $start$  域为 14 的表项,并从该表项中找到了节点 15 的 IP 地址。然后给节点 15 发送一个查询请求。节点 15 看到 16 位于它和后继节点(20)之间,所以它将节点 20 的 IP 地址返回给最初的请求者,即节点 1。

由于任何时候都有节点加入或者节点离开,所以 Chord 需要一种办法来处理这些操作。我们假定,当系统开始运行的时候,它的规模足够小,因此系统中的节点可以直接相互交换信息,从而建立起最初的圆环和指取表。之后就需要一个自动的过程,下面描述这个过程。当一个新的节点( $r$ )想要加入的时候,它必须与某一个已有的节点进行联络,并请它为自己查找  $successor(r)$  的 IP 地址。然后新节点请  $successor(r)$  查找它的前继节点。然后新节点请这两个节点将自己插在它们之间,当然也在圆上。例如,如果图 5.24 中的 24 想要加入的话,它请任何一个节点帮它查找  $successor(24)$ ,结果得到 27,然后它请 27 查找它的前继节点(20)。当 24 告诉这两个节点它自己要加入系统之后,20 用 24 作为它的后继节点,27 用 24 作为它的前继节点。另外,节点 27 也要将 20-24 范围内的键移交给节点 24。到这时候,24 已经完全插入了。

然而,许多指取表现在还是错误的。为了修正这些指取表,每个节点运行一个后台进程,该进程定期地重新计算每一个指取项,做法很简单,只要调用  $successor$  即可。当这些查询(即  $successor$  调用)中的某一个查询碰到新节点的时候,对应的表项就更新过来了。

当一个节点正常离开的时候,它将它的键移交给它的后继节点,并通知前继节点它要离开了,所以,前继节点可以链接到待离开节点的后继节点。当一个节点崩溃的时候,由于它的前继节点不再指向有效的后继节点,所以问题随之而来了。为了缓解这个问题,每

个节点不仅要与它的直接后继节点保持联系,而且也要与  $s$  步以内的直接后继节点保持联系,这样即使  $s-1$  个连续节点同时失败,它也可以跳过去,将环重新连接起来。

Chord 已经被用来构建一个分布式的文件系统(Dabek et al., 2001b),也被用于其他一些应用,而且 Chord 的研究工作仍然还在进行。在(Rowstron and Druschel, 2001a; and Rowstron and Druschel, 2001b)中描述了另一个不同的对等系统 Pastry 以及它的应用。第三个对等系统被称为 Freenet,请参见(Clarke et al., 2002)。关于第四个对等系统请参见(Ratnasamy et al., 2001)。

### 5.3 拥塞控制算法

当一个子网或者子网的一部分中出现太多分组的时候,网络的性能开始下降。这种情况称为**拥塞(congestion)**。图 5.25 描述了这种症状。当主机传送到子网中的分组数量在子网的容量范围内的时候,所有这些分组都可以被递交(除了有些受到传输错误的影响以外),被递交的分组数量与发送的分组数量成正比。然而,随着流量的快速递增,路由器不再能够处理所有这些分组,于是它们开始丢失分组。而且这将会使事情更加糟糕。在流量很高的时候,性能将严重恶化,几乎没有分组能够被递交。

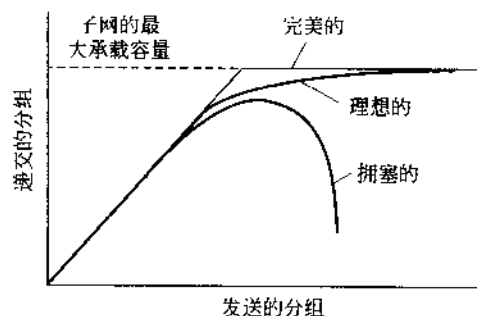


图 5.25 当流量太大时,拥塞就会发生,性能急剧降低

拥塞之所以发生有多个因素。如果突然之间在三条或者四条输入线路上开始有分组流到达,而且这些分组流都需要同样的输出线路,那么,路由器将建立一个队列。如果路由器没有足够的内存来存放所有这些分组,那么有的分组就会丢失。多增加一些内存可能有助于将路由器的性能提升到一定的点上,但是,Nagle(1987)发现,如果路由器有无限数量的内存,那么,拥塞现象会变得更差,而不是更好,因为当分组到达队列前端的时候,这些分组已经超时了(反复地),重复分组也已经被发送出来了。所有这些分组都将被尽职地转发给下一台路由器,从而增加了到达目标机器的整条路径的负载。

慢速的处理器也可能引起拥塞。如果路由器的 CPU 非常慢,以至于难以完成必要的维护工作(如缓冲区排队、更新路由表,等等),那么,即使有多余的线路容量,分组也需要进入到队列之中。仅仅对线路进行升级而不改变处理器,或者反过来,仅仅升级处理器而不改变线路,通常不会有很明显的效果,这样的做法往往只是转移了瓶颈而已。只升级系统的一部分而不是全部,通常只是将瓶颈转移到系统中其他的地方。这个问题会一直

存在,直到所有的部件相互平衡之后才会解决。

值得提出来的一点是拥塞控制和流控制之间的差异,因为它们之间的关系非常微妙。拥塞控制的任务是确保子网能够承载所到达的流量。这是一个全局性的问题,涉及到各方面的行为,包括所有的主机、所有的路由器、路由器内部的存储转发处理过程,以及所有可能会削弱子网承载容量的其他因素。

与此相反,流控制只与特定的发送方和特定的接收方之间的点到点流量有关。它的任务是,确保一个快速的发送方不会持续地以超过接收方吸收能力的速率传输数据。流控制通常涉及到的做法是,接收方向发送方提供某种直接的反馈,以便告诉发送方另一端的情形到底怎么样。

为了看清楚这两个概念之间的差异,请考虑一个光纤网络,它的传输容量为1000Gbps,在这个网络上,一台超级计算机试图以1Gbps的速率给一台个人计算机传送一个文件。尽管这里并没有拥塞(网络本身没有任何问题),但是流控制却是需要的,以便强迫超级计算机经常停下来,给个人计算机以喘息的机会。

作为另一个极端的情形,请考虑这样一个存储-转发型网络:它的线路为1Mbps,它有1000台大型的计算机,其中一半的机器正在试图给另一半机器以100kbps的速率传送文件。这里的问题并不是快速的发送方会淹没慢速的接收方,而是总流量超过了网络的处理能力。

拥塞控制和流控制之所以常常被混淆的原因是,有些拥塞控制算法在网络出现问题的时候,通过往各个源端发送消息来告诉它们减慢发送数据的速度。因此,一台主机既可能因为接收方不能处理负载的原因,也可能因为网络不能处理负载的原因而接收到“减慢速度”的消息。后面我们还将回到这个话题上。

在本节中,我们将首先讨论拥塞控制的一个通用模型,然后我们看一看从一开始就预防拥塞的各种方法。之后,我们来学习一旦拥塞发生以后的各种动态处理算法。

### 5.3.1 拥塞控制的通用原则

在像计算机网络这样的复杂系统中,许多问题都可以从控制论的角度来看待。这种方法导致所有的解决方案可以分成两类:开环的(open loop)和闭环的(close loop)。开环的解决方案试图用良好的设计来解决问题,它的本质是从一开始就保证问题不会发生。一旦系统启动并运行起来了,就不需要中途做修正。

完成开环控制的手段有:确定何时接受新的流量、确定何时丢弃分组及丢弃哪些分组,以及在网络的不同点上执行调度决策。所有这些手段的共同之处是,它们在做出决定的时候不考虑网络的当前状态。

相反,闭环方案则建立在反馈环路的概念基础之上,当这种方法用于拥塞控制的时候,它有三个部分:

- (1) 监视系统,检测到何时何地发生了拥塞。
- (2) 将该信息传递到能够采取行动的地方。
- (3) 调整系统的运行,以改正问题。

在监视子网的拥塞情况时可以使用多种度量标准,其中最主要的度量包括:由于缺

少缓冲区空间而丢弃的分组所占的百分比、平均队列长度、超时和重传分组的数量、平均分组延迟,以及分组延迟的标准方差。对于所有这些度量值,数值越大,表示拥塞的可能性也越大。

反馈环路中的第二步是将从检测点上收集来的有关拥塞的信息传递到能够采取行动的点上。一种很显然的办法是,检测到拥塞的路由器给流量源(可能有多源)发送一个分组,告诉它们发生了问题。当然,这些额外的分组恰好在于网发生拥塞的时候(即很难处理更多负载的时候)又增加了子网的负载。

然而,其他的办法也是有可能的。例如,在每个分组中可以保留一位或者一个域,当拥塞度量超过某一个阈值的时候,由路由器来填充该位或者域。当一台路由器检测到拥塞状况的时候,它在所有的输出分组中填充该域,以警告它的邻居。

还有另外一种办法是,让主机或者路由器周期性地向外发送探测分组,显式地询问有关拥塞的情况。然后,在有问题的区域中,可以利用这些信息来路由流量。有些无线电台利用直升飞机在城市上空飞行,来报告关于道路的拥塞状况,因而有可能让他们的移动听众开车的时候(发送分组的时候)绕过拥塞的路段。

在所有的反馈机制中,期望的效果是,有关拥塞的知识将会导致主机采取适当的行动来减轻拥塞。为了使一种方案能够正确地工作,必须要谨慎地调整时间尺度。如果每次两个分组连续到达的时候,路由器就会叫“停止(STOP)”,而每次当路由器空闲了  $20\mu\text{s}$  的时候,它就会叫“前进(GO)”,那么,系统将会剧烈震荡,永远不会稳定下来。另一方面,如果它要等待 30 分钟才能有所指示的话,那么拥塞控制机制的反应将会太迟缓,因而不会有实际的用途。要想让拥塞机制正常地工作,就需要取一个合理的平均值,但是,取一个正确的时间常数值并不是一件很简单的事情。

目前已经出现了很多拥塞控制算法。为了能够用一种合理的方法将这些算法组织起来,Yang 和 Reddy(1995)提出了一种针对拥塞控制算法的分类方法。他们首先将所有的算法分成开环算法和闭环算法,如前所述。然后,他们又进一步将开环算法分成两类:一类在源端采取动作,另一类在目标端采取动作。闭环算法也分成两个子类:显式反馈和隐式反馈。在显式反馈算法中,从拥塞点向源端发送分组以警告源端。在隐式反馈算法中,源端利用本地观察到的现象,比如确认分组送回来所需要的时间,来推断出是否存在拥塞。

一旦出现了拥塞,这意味着当前的负载(临时性地)超过了(系统中的一部分)资源的处理能力。于是马上可以想到两种解决方案:增加资源,或者降低负载。例如,子网可以利用拨号电话线路临时性地增加某些点之间的带宽。在卫星系统上,增加发射功率常常可以获得更高的带宽。把流量分散到多条路由路径上而并不总是使用最佳的路由路径也可以有效地增加带宽。最后,把那些空闲的备用路由器(这些路由器可以使系统更能容错)启动起来,也可以在发生严重拥塞的时候增加系统的处理能力。

然而,有时候增加系统的容量是不可能的,或者系统已经达到了极限。在这种情况下,解决拥塞问题的惟一办法是降低系统的负载。有几种办法可以降低负载,包括:拒绝为某些用户提供服务、给某些用户或者全部用户降低服务等级,以及让用户以一种更有预见性的方式来安排他们的需求。



上面提到的这些方法中,有些方法(稍后我们将学习这些方法)更加适合于虚电路(virtual circuit)。对于那些内部使用虚电路的子网来说,这些方法可以在网络层上使用。对于数据报子网来说,这些方法有时候可以用在传输层连接上。在本章中,我们将集中讨论它们在网络层中的应用。在下一章中,我们将会看到在传输层上如何实现拥塞控制。

### 5.3.2 拥塞预防策略

作为开始,首先我们来看一看开环系统中的拥塞控制方法。这些系统的设计思想是从一开始就将发生拥塞的可能性降低到最小,而不是任其发生并且在既成事实之后再作出反应。它们试图在各个层次上使用适当的策略来达到这个目标。在图 5.26 中,我们可以看到各种可能会影响到拥塞的数据链路层、网络层和传输层策略(Jain, 1990)。

层	策略
传输层	<ul style="list-style-type: none"> <li>• 重传策略</li> <li>• 乱序缓存策略</li> <li>• 确认策略</li> <li>• 流控制策略</li> <li>• 确定超时的策略</li> </ul>
网络层	<ul style="list-style-type: none"> <li>• 子网内部的虚电路与数据报策略</li> <li>• 分组排队和服务策略</li> <li>• 分组丢弃策略</li> <li>• 路由算法</li> <li>• 分组生存期管理</li> </ul>
数据链路层	<ul style="list-style-type: none"> <li>• 重传策略</li> <li>• 乱序缓存策略</li> <li>• 确认策略</li> <li>• 流控制策略</li> </ul>

图 5.26 影响拥塞的策略

我们首先从数据链路层开始,然后逐步向上进行讨论。重传策略牵涉到一个发送方发送分组之后多久会超时,以及超时之后重传什么。一个激进的发送方在超时之后用回退  $n$  步的技术来重传所有未确认的分组,它给系统带来的负载要比一个使用选择性重传策略的保守发送方多得多。与此紧密相关的是缓存策略。如果接收方只是机械地丢弃所有的乱序分组,那么,这些分组将不得被重传,从而带来了额外的负载。从拥塞控制的角度而言,选择性重传比回退  $n$  步的策略明显要好得多。

确认策略也会影响到拥塞。如果每个分组都立刻被确认的话,那么确认分组将会带来额外的流量。然而,如果确认消息被保存起来以便在反向流量中捎带回去,那么有可能导致额外的超时和重传。一个紧凑的流控制方案(比如一个小的窗口)将降低数据传输率,因此有助于缓解拥塞。

在网络层上,选择使用虚电路还是使用数据报,这也会影响到拥塞的情况,因为许多拥塞控制算法只能与虚电路子网一起工作。分组排队和服务策略关系到路由器是否为每条输入线路使用一个队列,是否为每条输出线路使用一个队列,或者为两者各使用一个队

列。它也关系到分组的处理顺序(比如轮循法或者基于优先级的处理方法)。丢弃策略是指当没有空间的时候,指明该丢弃哪个分组的规则。一个好的策略有助于缓解拥塞的局面,而一个较差的策略则使得局面更加糟糕。

好的路由算法有助于避免拥塞,它可以将流量分散到所有的线路上,而一个不好的路由算法则会将大量的流量发送到本来已经拥塞的线路上。最后,分组生存期管理策略负责处理一个分组在被丢弃之前应该生存多长时间。如果生存期太长的话,则丢弃的分组可能会长时间地妨碍工作,但是,如果生存期太短的时候,则分组在到达目的地之前就有可能超时,从而导致重传。

在传输层中,如同数据链路层的情形一样,同样的问题也会发生,但是除此以外,在传输层上确定超时间隔更加困难,因为跨越一个网络的传输时间,比两台路由器之间线路上的传输时间更加难以预测。如果超时间隔太短的话,则会发送不必要的额外分组。如果太长的话,则拥塞将会缓解,但是一旦分组丢失,则响应时间将会增加。

### 5.3.3 虚电路子网中的拥塞控制

上面介绍的拥塞控制方法基本上都是开环的:它们都试图从一开始就避免出现拥塞,而不是在拥塞发生之后再行控制。在这一小节中,我们将介绍几种用于虚电路子网的动态拥塞控制方法。在随后的两小节中,我们将讨论一些可用于任何一个子网的拥塞控制技术。

一种广泛应用的、防止已经拥塞的子网进一步恶化的技术是准入控制(admission control)。其思想很简单:一旦出现拥塞的信号,则不再创建任何虚电路,直到问题排除为止。因此,在拥塞的情况下,企图建立新的传输层连接将会失败。这时候让更多的人进来只会使情况更糟。虽然这种方法有点粗野,但是它非常简单,而且也易于实现。在电话系统中,当一台交换机超过负载的时候,它也会采用准入控制的方法,不再送出拨号音。

另一种方法是允许建立新的虚电路,但是谨慎地选择路由,使所有新的虚电路都绕开有问题的区域。例如,在图 5.27(a)的子网中,两台路由器已经拥塞了。

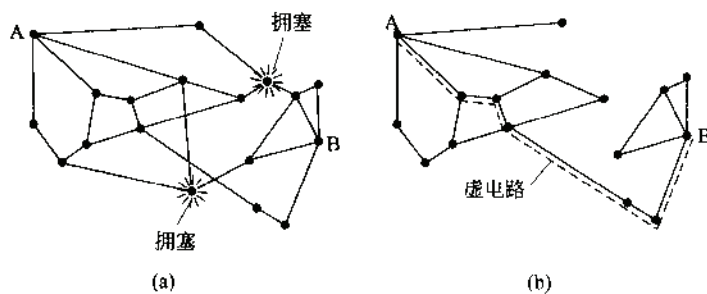


图 5.27

(a) 一个拥塞子网; (b) 消除了拥塞之后的子网,同时还显示了从 A 到 B 的一条虚电路

假设一台连接到路由器 A 的主机想要与另一台连接到路由器 B 的主机建立一个连接。通常情况下,这个连接将会经过其中一台拥塞的路由器。为了避免这种情形,我们可

以重画子网,如图 5.27(b)所示,省略掉拥塞的路由器和它们所有的线路。图中的虚线显示了一条可能的虚电路路径,它避开了拥塞的路由器。

与虚电路有关的另一种策略是,当建立虚电路的时候,在主机和子网之间进行协商以达成一致的约定。这份约定通常规定了流量的容量和形状、所要求的服务质量和其他的参数。为了保证这份约定的有效性,在建立电路的时候子网往往会在沿途预留资源。这些资源可以是路由器中的表空间和缓冲区空间,也可以是线路上的带宽。通过这种方式,在新的虚电路上不可能发生拥塞,因为所有需要的资源保证能够获得。

这种资源预留的做法既可以被作为一个标准的操作过程而在任何时候都使用,也可以仅仅当子网发生拥塞的时候才使用。任何时候都使用这种做法(即预留资源)的一个缺点是,它会导致资源的浪费。如果有 6 条虚电路可能会使用 1Mbps 的带宽,现在它们要通过同一段 6Mbps 的物理线路,那么,该线路将被标记为已满,即使这 6 条虚电路不太可能同时满负载地传输数据,它们也不会腾出带宽作为他用。因此,在通常情况下,这种拥塞控制的代价是有未被使用(即浪费)的带宽。

#### 5.3.4 数据报子网中的拥塞控制

现在我们转到一些既可用于数据报子网也可用于虚电路子网的拥塞控制方法。每台路由器很容易就可以监视到它的输出线路和其他资源的使用情况。例如,它可以给每条线路关联一个实变量  $u$ ,其取值在 0.0 和 1.0 之间,它代表了这条线路最近的利用率。为了使  $u$  的值尽可能地反映实际的情况,路由器需要定期地对线路的瞬时利用率  $f$  (0 或者 1)进行采样,然后根据下面的公式更新  $u$ :

$$u_{\text{新}} = a u_{\text{旧}} + (1-a)f$$

这里常数  $a$  代表了路由器忘记最近的历史情况有多快。

每当  $u$  超过了特定的阈值的时候,该输出线路进入到一种“警告”状态。路由器对每个新到来的分组都要进行检查,看它的输出线路是否处于警告状态。如果是的话,就需要采取某种措施。可以采取的措施有多种,下面分别进行讨论。

##### 警告位

旧的 DECNET 体系的做法是,在分组的头中设置一个特殊的位来指示警告状态;在帧中继(frame relay)网络中也是如此。当分组到达它的目标端的时候,传输实体将这一位复制到下一个确认分组中,所以,这一位也被送回到源主机。然后,源主机就可以削减流量。

只要路由器处于警告状态之中,它就会不停地设置警告位,这意味着源主机也会不停地得到这样的设置了警告位的确认分组。源主机可以监视这种设置了警告位的确认分组的比例,然后相应地调整它的传输速率。只要警告位仍然在源源不断地回来,则源主机就不停地减慢它的传输速率。当警告位确认分组减到极少量的时候,源主机增加它的传输速率。请注意,由于沿途的每台路由器都可能设置警告位,所以,只有当所有的路由器都排除了问题之后,流量才可以增加上去。

### 抑制分组(choke packet)

上面介绍的拥塞控制算法有点太过于精细了。它使用一种非常曲折的办法来告诉源主机要放慢速度。为什么不直接告诉它呢?在这种方法中,路由器给源主机送回一个抑制分组(choke packet),并且在抑制分组中指明原分组的目标地址。同时,原来的分组被加上一个标记(设置头部中的一位),因而它在前行的路径上不会再产生更多的抑制分组。除此以外,分组的转发过程如同平常一样。

当源主机收到了抑制分组的时候,按照要求,它发送给指定目标的流量必须减少 X 百分比。由于发送给同一目标的其他分组可能已经在路上了,并且它们也将会产生更多的抑制分组,所以,在一段固定的时间间隔以内,该主机应该忽略掉所有指向此目标的抑制分组。过了这一段间隔以后,该主机继续监听下一段时间间隔内的抑制分组。如果又收到抑制分组的话,则线路仍然拥塞,所以,该主机进一步减少流量,并再次忽略掉一段时间内的抑制分组。如果在监听周期内没有收到抑制分组的话,该主机可能会再次增加流量。该协议中暗含的反馈信息有助于防止拥塞同时又不影响正常的分组流(除非真有问题发生)。

主机通过调整策略参数(比如窗口大小)可以减慢流量。通常情况下,第一个抑制分组使得数据率减少到原来的 0.50,第二个抑制分组使数据率减少到 0.25,以此类推。增加数据率的幅度要小得多,这样可以避免很快又发生拥塞。

研究人员在这个拥塞控制算法的基础上又提出了几个变种算法。在其中一个变种算法中,路由器维护了几个阈值。根据当前流量已经达到了哪一个阈值,路由器在抑制分组中可以包含一个轻微的警告,或者一个严厉的警告,或者最后的通牒。

另一个变种算法是使用队列长度或者缓冲区利用率,而不是线路利用率作为触发抑制分组的标准。当然,如同针对 u 的做法一样,对这些度量标准也可以使用指数加权。

### 逐跳(hop-by-hop)抑制分组

当网络速度很高或者路由器离源主机的距离很远的时候,给源主机发送抑制分组并不能很好地起作用,因为反应太慢。比如,请考虑这样的情形:San Francisco(旧金山)的一台主机(图 5.28 中的路由器 A)正在给 New York(纽约)的一台主机(图 5.28 中的路由器 D)发送数据,速度为 155 Mbps。如果 New York 的主机现在用完了缓冲区空间,那么抑制分组回到 San Francisco 需要 30ms 时间,然后 San Francisco 的主机才能慢下来。抑制分组的传播过程如图 5.28(a)中的第 2、3、4 步所示。在这 30ms 期间,A 又给 D 发送了 4.6Mb 数据。即使 San Francisco 的主机立刻停机了,沿途路径上的 4.6Mb 数据也会继续到来,并且必须要对它们进行处理。只有在图 5.28(a)的第 7 个图中,New York 的路由器才会注意到分组流减慢了。

另一种办法是让抑制分组影响到沿途的每一跳,如图 5.28(b)中的序列所示。在这里,只要抑制分组到达了 F,则 F 必须按照要求减慢向 D 的分组流。为了做到这一点,要求 F 为 D 的分组流分配更多的缓冲区,因为源主机仍然在全速发送数据,但是 F 这么做却让 D 立刻缓解了,就好像电视广告中的头痛疗法一样。在下一步中,抑制分组到达 E,

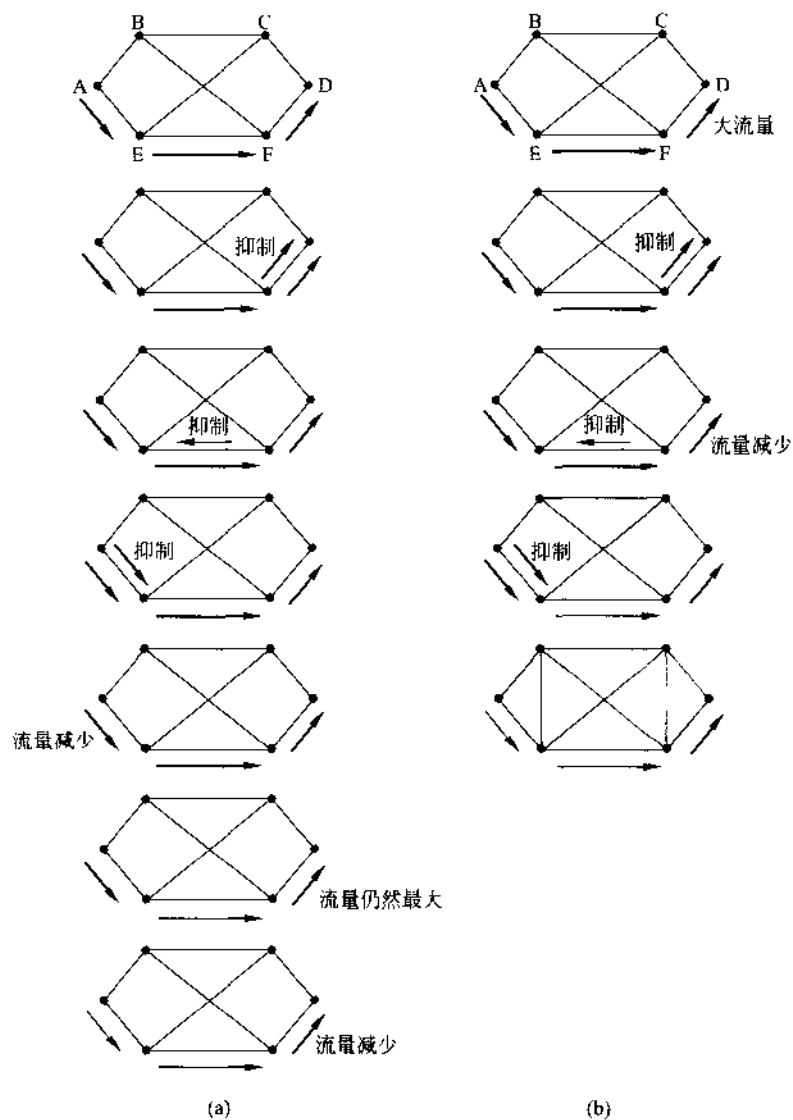


图 5.28

(a) 一个只影响源主机的抑制分组；(b) 一个影响到沿途每一跳的抑制分组

它告诉 E 要减慢向 F 的分组流。这一行动增加了 E 的缓冲区需求,但是却让 F 立即缓解了。最后,抑制分组到达 A,分组流才真的减慢下来。

这种逐跳方案的实际效果是,在拥塞点上拥塞现象很快得到了缓解,但是其代价是上游路径上需要消耗更多的缓冲区空间。使用这种方法可以将拥塞消灭在萌芽状态,同时又不会丢失任何分组。有关该算法思想的详细讨论和模拟结果请参考 (Mishra and Kanakia, 1992)。



### 5.3.5 负载丢弃

当以上任何一种方法都不能消除拥塞的时候,路由器可以亮出它的重型炮弹了,即**负载脱落(load shedding)**。负载脱落是一种富有想象力的说法,它是指当路由器因为来不及处理分组而被淹没的时候,只要将这些分组丢弃即可。此术语来源于电力发电领域,它是指“在炎热的夏日里,当电力需求超过了供电能力时,为了避免电力系统崩溃而有意切断某些特定区域的电力供给”的做法。

一台正在被分组淹没的路由器可以随机地丢掉一部分分组,但是通常它还可以做得更好一些。到底应该丢掉哪些分组,这应该取决于所运行的应用的类型。对于文件传输而言,老的分组比新的分组更有价值,因为丢掉分组 6 而保留分组 7~10 将使得在接收方出现一个空隙,有可能导致重传分组 6~10(如果接收方的习惯做法是丢弃乱序分组的话)。如果一个文件总共只有 12 个分组,那么,丢弃分组 6 可能意味着要重传分组 7~12,而丢弃分组 10 可能只要求重传 10~12。相反,对于流媒体应用,一个新的分组比老的分组更加重要。前一种策略(即旧的分组比新的分组更好)通常称为**葡萄酒(wine)**策略,而后一种策略(即新的分组比旧的分组更好)通常称为**牛奶(milk)**策略。

上述方法要想真正有效地实施,必须有发送方的紧密配合。对于许多应用,有的分组比其他的分组更为重要。例如,某些视频压缩算法会定期地传输一个完整的帧,然后以帧差(与完整帧之间的差异)的形式传输后续的帧。在这种情况下,丢弃帧差中的分组显然比丢弃完整帧中的分组要好得多。又如另一个例子,考虑传输一个既包含 ASCII 文本又包含图片的文档。很显然,丢弃某个图像中的一行像素比丢掉一行可读的文本带来的损失要小得多。

为了实现一种智能的丢弃策略,应用程序必须在它们的分组中标明优先级,以指明这些分组的重要性。如果它们做到了这一点,那么当路由器必须要丢掉分组的时候,它们可以首先丢掉最低优先级的分组,然后丢掉次低级的分组,以此类推。当然,除非有重要的动机促使人们把有些分组标记为除了 VERY IMPORTANT(非常重要)——NEVER, EVER DISCARD(永不丢弃)之外的其他标记,否则没有人会这样做的。

这种动机有可能以费用的形式来体现,低优先级的分组比高优先级的分组更加便宜。或者,发送方有可能只允许在负载较轻的情况下才可以发送高优先级的分组,然而,随着负载的增加,这些分组也会被丢弃,因此鼓励用户不要再发送这样的分组了。

另一种选择是,允许用户超越在建立虚电路时候协商好的限制值(比如,使用比允许值还要高的带宽),但是也要服从这样的条件:所有超出部分的流量必须被标记为低优先级的。这样的策略其实并不坏,因为它充分利用了空闲的资源,只要其他没有人在使用这些资源,则任何一台主机都可以使用它们,但是当总体资源紧张的时候,它将得不到这些资源的使用权。

#### 随机的早期检测

无疑,当一开始检测到拥塞的时候就采取措施,比等到拥塞严重影响了工作之后再采取措施显然要有效得多。这导致了另一种思想:在实际耗尽所有的缓冲区空间之前就开

始丢弃分组。采用这种思想的一种流行算法为 RED(Random Early Detection, 随机的早期检测)(Floyd and Jacobson, 1993)。在有些传输协议(包括 TCP)中,针对分组丢失的响应措施是源主机减慢传输速度。这种做法背后的依据是,TCP 是针对有线网络而设计的,而有线网络是非常可靠的,所以,分组丢失的主要原因是缓冲区满了,而不是传输错误。这样可以减少帮助减少拥塞。

通过让路由器在情况变得恶化之前(即名字中“早期”的意思)就开始丢弃分组,RED 的思想是让路由器在还有机会的时候就采取行动,要不然就太迟了。为了确定该什么时候开始丢弃分组,路由器维护了其队列最近的平均长度值。当某一条线路上的队列平均长度超过一定阈值的时候,该线路被认定是拥塞的,从而对它采取措施。

由于路由器可能无法判断哪一个源是引起拥塞的罪魁祸首,所以,它能够做到的也就是随机地从拥塞的队列中选取分组。

那么,路由器用什么方法来告知源主机有关的问题呢?一种办法是给源发送一个抑制分组,如前面所述。这种做法的一个问题是,它在已经拥塞的网络上引入了更多的负载。另一种不同的策略是,只是将选取出来的分组丢弃,而不向源主机报告。源主机最终会感觉到缺少了确认,并采取相应的措施。由于它知道分组的丢失往往是因为拥塞和丢弃分组而造成的,所以它用减慢发送速度作为对策,而不是更加费力地发送更多的分组。这种隐式的反馈机制只有当源主机使用减慢传输速率作为分组丢失的响应对策时才起作用。在无线网络中,由于大多数的分组丢失是因为空中链路上的噪声而引起的,所以,这种方法并不适用。

### 5.3.6 抖动控制

对于像音频流和视频流这样的应用,只要传输时间是恒定的,那么它就不会介意到底需要 20ms 还是 30ms 才将分组递交给目标主机。分组到达时间的变化量(即标准偏差)被称为抖动(jitter)。比如,在高抖动的情况下,有的分组需要 20ms 到达,而其他的分组需要 30ms 到达,这将使得声音或者电影的质量很不稳定。图 5.29 演示了抖动的情况。与此相反,如果 99% 的分组都在 24.5ms 至 25.5ms 的延迟范围内被递交给目标主机的话,则当然是可以接受的。

当然,所选择的范围必须是切合实际的。在选择这一段范围的时候,必须要考虑到光速的传输时间和经过路由器的最小延迟,可能还要适当保留一点余量用于某些不可避免的延迟。

通过计算出沿途每一跳的期望传输时间,就可以对抖动加以控制。当一个分组到达一台路由器的时候,该路由器对它进行检查,看这个分组比预定的时间来早了,还是来晚了。这些信息被保存在分组之中,每一跳都相应地更新。如果一个分组提前到达的话,那么它尽可能多停留一段时间,以便回到预定的时间点上。如果一个分组比预定的时间到达得晚的话,则路由器尽可能快地将它转发出去。

对于多个正在竞争同一条输出线路的分组,路由器接下去该转发哪一个分组呢?实际上,它所使用的算法总是可以选择偏离预定时间最远的那个分组。按照这种方法,比预定时间提前到达的分组将会减慢下来,而落后于预定时间的分组则会逐渐加速,无论哪一

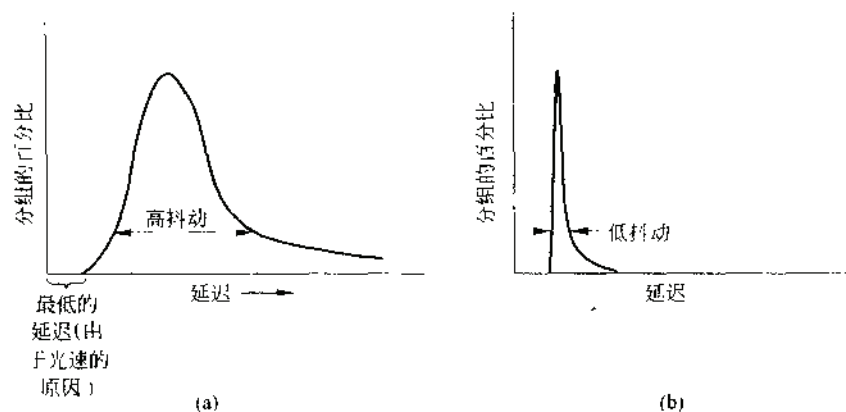


图 5.29

(a) 高抖动的情形; (b) 低抖动的情形

种情形,都可以减缓抖动的程度。

在有些应用中,比如视频点播(video on demand),也可以通过这样的做法来消除抖动:接收方将分组缓存起来,然后从缓冲区中获取数据并输出到显示器上,而不是实时地从网络上获取数据。然而,对于其他的应用,特别是那些要求人与人之间进行实时交互的应用,比如 Internet 电话和视频会议,因缓存而带来的延迟是不可接受的。

拥塞控制是一个非常活跃的研究课题。(Gevros et al., 2001)概述了有关拥塞控制的最新研究状况。

## 5.4 服务质量

我们在上一节中讨论到的技术的目标是减少拥塞和提高网络性能。然而,随着多媒体网络连接需求的增长,通常仅仅依靠这些特殊的手段还不够。除此以外还需要做更多的工作,以便通过网络和协议的设计来保证服务质量。在本节中,我们将继续研究网络的性能,但是,现在将焦点放在如何提供与应用的需求相匹配的服务质量上。然而,首先必须声明,在本节介绍的想法中,有许多还没有定型,将来仍会有变化。

### 5.4.1 需求

从一个源到一个目标的分组流(stream)称为一个流(flow)。在面向连接的网络中,属于同一个流的所有分组将会走同样的路由路径;而在无连接的网络中,它们可能会走不同的路径。我们可以用 4 个基本的参数来描述每个流的需求特征:可靠性、延迟、抖动和带宽。这 4 个特征合起来决定了一个流所要求的**服务质量 (Quality of Service, QoS)**。图 5.30 列出了一些常见的应用,以及它们的需求的严格程度。

前 4 种应用对于可靠性有很严格的要求。任何一位都不允许被错误递交。为了实现这个目标,通常的做法是,发送方计算每一个分组的校验和,接收方验证此校验和。如果一个分组在传输过程中被损坏了,那么它不会被确认,最终发送方重传该分组。这种策略

可以带来很高的可靠性。后面 4 种(音频/视频)应用可以容忍错误,所以,它们不需要计算或者验证校验和。

应用	可靠性	延迟	抖动	带宽
电子邮件	高	低	低	低
文件传输	高	低	低	中
Web 访问	高	中	低	中
远程登录	高	中	中	低
音频点播	低	低	高	中
视频点播	低	低	高	高
电话	低	高	高	低
视频会议	低	高	高	高

图 5.30 服务质量需求的严格程度

文件传输应用,包括电子邮件和视频,它们对于延迟并不敏感。如果所有的分组都统一延迟了几秒钟,则不会有任何伤害。然而,一些交互应用,比如 Web 浏览和远程登录,则对于延迟比较敏感。另外,一些实时的应用,比如电话和视频会议,它们有非常严格的延迟要求。如果在一次电话通话过程中,每一句话中的每一个字都延迟了正好 2 秒钟,那么用户将认为这样的连接是不可接受的。另一方面,从一台服务器上播放音频或者视频文件并不要求很低的延迟。

现在考虑分组到达接收方的时间点的规律性,前三种应用对于这种规律性并没有强烈的要求。远程登录应用对此比较敏感,因为如果远程连接抖动得厉害的话,屏幕上的字符也会表现得快一阵慢一阵。视频和音频,特别是音频,对于抖动极为敏感。如果当一个用户正在通过网络观看一段视频的时候,所有的帧恰好都延迟了 2.000 秒,那么不会有任何影响。但是如果每一帧的传输时间在 1 秒和 2 秒钟之间随机地变化的话,则结果将会非常可怕。对于音频数据流,即使几个毫秒的抖动也能够很明显地感觉到。

最后,不同的应用对于带宽的需求也是不同的,电子邮件和远程登录应用并不要求很高的带宽,但是各种形式的视频应用总是需要较高的带宽。

ATM 网络把数据流分成 4 大类,分类的依据是它们对于 QoS 的要求,如下所示:

- (1) 位速率为常数的应用(比如电话)。
- (2) 位速率可变的实时应用(比如压缩的视频会议)。
- (3) 位速率可变的非实时应用(比如通过 Internet 观看电影)。
- (4) 最大可用位速率的应用(比如文件传输)。

这样的分类也可以用于其他的用途,或者用于其他的网络。常数位速率是指试图模拟一条物理的线路,提供恒定的带宽,并保持恒定的延迟。当视频信号被压缩的时候,由于有些帧的压缩比率大于其他的帧,所以会发生位速率可变的情况。因此,如果发送的帧中包含了很多细节,那么它就会要求发送许多位数据,而如果一帧中只包含一面白色的墙,那么它可能会被压缩成极少的位。最大可用位速率这种类别是针对像电子邮件这样

的应用,它们对于延迟和抖动并不敏感。

#### 5.4.2 获得好的服务质量所使用的技术

现在我们已经了解了有关 QoS 需求方面的情况,那么,如何实现这样的需求呢?好,首先说明,这里没有神奇的秘诀。没有一项技术能够以最优的方式来提供高效的、可靠的 QoS。相反,研究人员已经提出了许多技术,而且一些实际的解决方案通常将多种技术结合起来使用。我们现在来讨论其中这样一些技术,系统设计人员可以用这些技术来实现 QoS。

##### 过度提供资源

一种很容易想到的解决方案是提供足够的路由器容量、缓冲区空间和带宽,以保证分组能够顺利地通过。这种方案的问题在于,它的代价太昂贵。随着时间的推移,设计者们对于“到底多少资源才算足够”逐渐会有所认识,因而这项技术有可能变得实用起来。从某种程度而言,电话系统就属于过度提供资源的系统。拿起一个电话而没有立刻听到拨号音的情形非常少见。由于电话系统中总是有足够可用的容量,所以,用户的需求总是可以得到满足。

##### 缓冲能力

在接收方,数据流在被递交之前可以先缓存起来。将数据流缓存起来并不会影响到可靠性和带宽,但会增加延迟,然而,这样可以消除抖动。对于音频和视频点播,抖动是主要的问题,所以这项技术非常有用。

在图 5.29 中,我们已经看到了高抖动和低抖动之间的差异。在图 5.31 中,我们看到一个分组流在刚被递交的时候有明显的抖动。分组 1 是在  $t=0$  秒时从服务器发送出来的,并且在  $t=1$  秒时到达客户机器;分组 2 在途中的延迟更长一些,它花了 2 秒钟才到达。当这两个分组到达之后,它们被缓存在客户机器上。

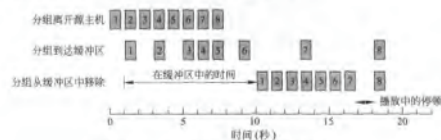


图 5.31 通过缓存分组的办法来平滑输出

在  $t=10$  秒时,客户开始播放。这时候,分组 1~6 都已经被缓存起来了,所以,它们可以同时从缓冲区中转移出来,由播放器平滑地将它们播放出来。不幸的是,分组 8 延迟得太久,等到该播放它的时候,它还没有到来,因此,播放过程必须停止下来,等到它到来之后才能继续进行,这样导致在音乐或者电影中出现烦人的停顿现象。我们可以将起始



时间再多延迟一会儿,从而缓解这个问题,但是这样做的代价是要求一个更大的缓冲区。包含流式音频或者视频的商业 Web 站点所使用的播放器在开始播放之前都要先缓存 10 秒钟左右的流数据。

### 流量整形

在上面的例子中,源端按照统一的间隔逐个发送分组,但是在其他的情形下,源端有可能不规律地发送分组,这些分组可能会在网络中造成拥塞。如果服务器同时处理多个流的话,那么这种不均匀的输出模式是很常见的;服务器其他的动作也可能造成不均匀输出,比如快速地转发和重绕、用户认证等等。而且,我们前面所使用的方法(缓存)并不总是可行的,比如,对于视频会议就无法使用。然而,如果能够采取措施来保证服务器(或者更一般的情形,普通的主机)以均衡的速率发送数据的话,那么服务质量将会更好。现在我们将介绍一种称为**流量整形**(traffic shaping)的技术,它在服务器端而不是在客户端对流量进行平滑处理。

流量整形是指调节数据传输的平均速率(以及突发性)。与此不同的是,我们以前学过的滑动窗口协议则是限制同一时刻正在传输的数据量,而不是这些数据被发送的速率。当一个连接被建立起来的时候,用户和子网(即顾客和网络承运商)对于他们之间的电路上应该遵循什么样的流量模式(即流量的形状)达成了一致的协议。有时候称之为**服务等级协定**(service level agreement)。只要顾客履行了协定中他的义务,并且只根据约定好的模式来发送分组,那么,承运商就必须按照承诺,适时地将分组递交到目的地。**流量整形**技术可以减少拥塞,因此也有助于承运商兑现它的承诺。这样的协定对于文件传输之类的应用并不那么重要,但是对于实时数据的传输非常重要,比如音频和视频连接等,它们有严格的服务质量要求。

实际上,采用了流量整形技术之后,顾客相当于这样告诉承运商:我的传输模式将会如此这般,你能够处理吗?如果承运商答应了,那么问题就变成:承运商如何辨别该顾客是否遵循原来的协定,以及如果顾客不遵循协定的话承运商该怎么办。对一个业务流进行监视,这种行为称为**流量监管**(traffic policing)。对流量的形状达成一致的协定,并且随后对它进行监管,这在虚电路子网上比在数据报子网上容易实现得多。然而,在数据报子网中,我们可以将同样的思想应用到传输层的连接上。

### 漏桶算法

请想象这样一个桶,它的底部有一个小洞,如图 5.32(a)所示。不管该桶进水的速率是多少,当桶中还有水的时候,水流出去的速率是一个常数  $\rho$ ;当桶中没有水的时候,出水速率当然为 0。而且,一旦该桶满了,则再往里流的水将会沿着桶沿溢出,自然流失了(也就是说,不会到达底部洞口的输出水流中)。

同样的思想也可以应用到分组上,如图 5.32(b)所示。从概念上讲,在每个主机连接到网络的接口中都包含一个漏桶,也即一个有限长度的内部队列。如果当该队列满的时候,又有一个分组到来,那么该分组将被丢弃。换句话说,如果在一台主机上,队列中的分组数目已经达到了最大值,这时又有一个或者多个进程要发送分组,那么,新发送的分组

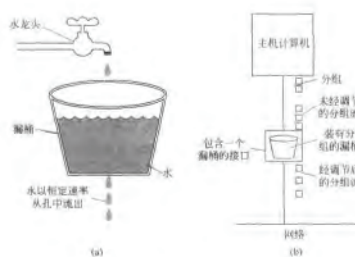


图 5.32  
(a) 一个盛水的漏桶；(b) 一个装分组的漏桶

将被毫不客气地丢弃。这种机制既可以被内置到硬件接口中，也可以由主机的操作系统来模拟实现。此算法称为漏桶算法 (leaky bucket algorithm)，最早是由 Turner (1986) 提出来的。实际上，这个算法非常简单，它只不过是一个常数服务时间的单服务器排队系统而已。

假设主机每过一个时钟滴答 (tick) 才允许把一个分组放到网络上。当然，这项限制既可以由接口卡来实现，也可以由操作系统来实现。实际上，这种机制可以将主机内用户进程发送出来的一个不均匀分组流变成了网络上的一个均匀分组流，它把突发的分组流变得很平滑，从而极大地降低了拥塞的几率。

当分组的大小全部相同的时候 (比如 ATM 信元)，这个算法可以按照上面的描述来使用。然而，当分组的大小可变的时候，通常更好的做法是允许每个时钟滴答通过固定数量的字节，而不只是一个分组。因此，如果制定的规则是允许每个滴答通过 1024 个字节，那么，每个滴答允许发送 1 个 1024 字节的分组，或者 2 个 512 字节的分组，或者 4 个 256 字节的分组，依此类推。如果剩余的字节数量太少的话，则下一个分组必须等到下一个滴答。

要实现原始的漏桶算法非常容易。漏桶是由一个有限队列构成的。当一个分组到达的时候，如果队列中还有空间的话，则它被添加到队列的末尾；否则，该分组被丢弃。在每一个时钟滴答周期到来的时候，主机发送一个分组 (除非队列为空)。

支持字节计数的漏桶算法几乎也可以用同样的方法来实现。在每一个滴答周期，首先将一个计数器初始化为  $n$ 。如果队列中第一个分组的字节数少于计数器的当前值，则该分组被发送出去，并且计数器减去该分组的字节数。其余的分组也可能被发送出去，只要计数器的值比它们的字节数大就行。当计数器的值小于队列中下一个分组的长度的时候，传输过程终止，直到下一个滴答再开始。到达下一个滴答的时候，字节计数器被重置。

于是分组流又继续发送。

举一个漏桶的例子,假设一台计算机可以按照每秒钟 25 兆字节(200Mbps)的速率产生数据,并且网络也可以运行在同样的速度上。然而,路由器只能在较短的时间间隔内(基本上而言,能坚持到它们的缓冲区被填满)以这样的速率接受数据。在长时间情况下,它们的最佳工作速率不超过每秒 2 兆字节。现在,假定数据以 1 兆字节大小的突发方式到来,每秒钟来一次 40ms 的突发数据。为了将平均速率降低到 2Mbps,我们可以采用这样一个漏桶: $\rho=2\text{Mbps}$ ,容量  $C$  为 1MB。这意味着即使是高达 1MB 的突发数据,都可以被毫无丢失地处理完。不管这些突发数据进来的速度有多快,它们都被分布到 500ms 的时间段上。

在图 5.33(a)中,我们看到进入到漏桶的数据速率为 25Mbps,持续时间为 40ms。在图 5.33(b)中,我们看到排出去的数据流以均匀的速率 2Mbps 持续了 500ms。

### 令牌桶

漏桶算法强迫输出模式保持严格的均匀速率,而不管通信流量的突发性程度如何。对于许多应用,更好的做法是,当大量的突发数据到来的时候,应该允许输出流适当地加快,因此我们需要一个更加灵活,并且最好不会丢失数据的算法。令牌桶算法(token bucket algorithm)正是这样一个算法。在这个算法中,漏桶中保存的是令牌,这些令牌是由时钟产生的,每隔  $\Delta T$  秒产生一个。在图 5.34(a)中,我们看到一个桶中有三个令牌,同时有 5 个分组正在等待传送。要使一个分组被传送出去,它必须要抓住并销毁一个令牌。在图 5.34(b)中,我们看到这 5 个分组中有 3 个已经被传送出去了,但是另外 2 个分组还在等待更多的令牌被生成出来。

令牌桶算法提供了另一种不同于漏桶算法的流量整形方法。漏桶算法不允许空闲的主机将许可权保存起来以便以后发送大的突发数据。而令牌桶算法则允许将许可权保存起来,直至到达桶的最大尺寸  $n$ 。这种特性意味着即使多达  $n$  个分组的突发数据也可以被一次发送出去,从而使得在输出流中也存在突发现象,并且对于突然之间到来的突发性输入可以更快地响应。

两个算法的另一个不同之处是,当令牌桶满了之后,令牌桶算法会丢弃令牌(相当于传输许可,或者传输容量),但是不丢弃分组。相反地,在漏桶算法中,当漏桶满了的时候,算法会丢弃分组。

在这里做一点小小的变化也是有可能的,那就是,每个令牌代表的是发送  $k$  个字节的权利,而不是发送一个分组的权利。只有当当前可用的令牌数量足够覆盖一个分组的字节长度的时候,该分组才会被传送出去。余下零碎的令牌被保留起来以供将来使用。

漏桶算法和令牌桶算法也可以用来平滑两台路由器之间的流量,还可以用来调节主机的输出流量,正如在我们的例子中那样。然而,一个很显然的区别是,对于一个用于调节主机输出流量的令牌桶,当规则有指示的时候,它可以使主机停止发送工作。但是,如果让一台路由器停止发送而它的输入分组仍然在源源不断地进来,那么这样可能会导致丢失数据。

基本令牌桶算法的实现非常简单,它只不过是一个记录令牌个数的变量而已。每隔

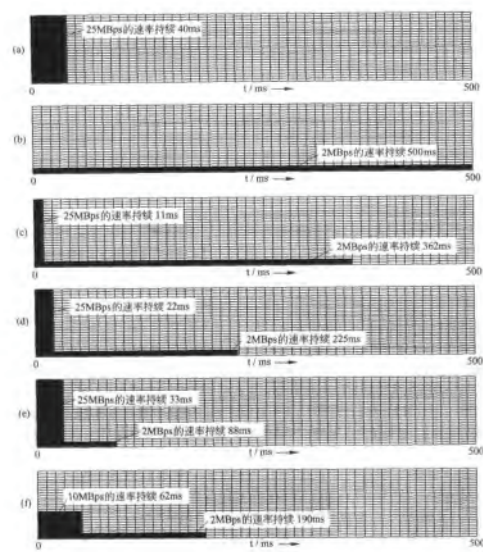


图 5.33

(a) 进入到编组的输入数据；(b) 编组的输出；(c)~(e) 当变量分别为 250KB、500KB 和 750KB 时编组的输出；(f) 一个 500KB 的令牌桶和一个 10Mbps 编组的输出

$\Delta T$  秒，该计数器增一；当一个分组被发送出去的时候，该计数器减一。当计数器减到 0 的时候，就不能够再发送分组。在按字节计数的变种算法中，每隔  $\Delta T$  秒，计数器增加  $k$ ；当发送分组的时候，减去每一个发送的分组长度。

从本质上讲，令牌桶所做的是：允许突发流量，但是不得超过一个预定的最大长度。请看图 5.33(c) 所示的例子。这里我们有一个容量为 250KB 的令牌桶。令牌到达的速率允许输出速率为每秒 2MB。假定当 1MB 突发数据到达的时候，令牌桶满了，那么，该令牌桶可以以 25Mbps 的速率发送大约 11ms 时间。然后，它必须回到 2Mbps 速率。

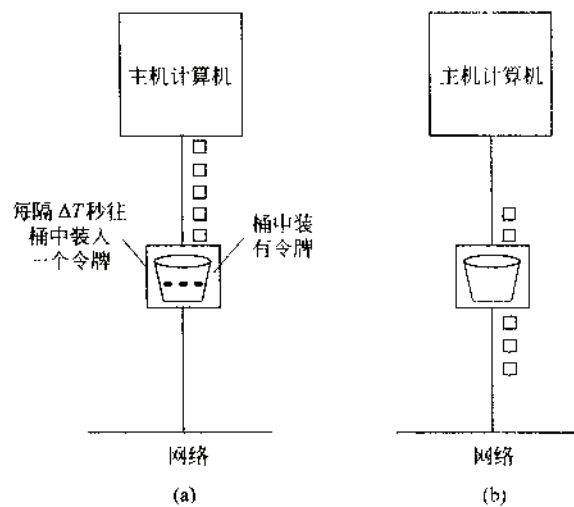


图 5.34 令牌桶算法  
(a) 之前; (b) 之后

直至所有的突发数据全部发送出去为止。

计算“以最大速率发送突发数据的持续时间”需要用一点技巧。它并不是简单地用 1MB 除以 25Mbps 就可以了,因为当突发数据正在被发送的时候,又会有新的令牌到来。如果我们记突发时间长度为  $S$  秒,令牌桶的容量为  $C$  字节,令牌的到达速率为  $\rho$  字节/秒,最大的输出速率为  $M$  字节/秒,那么,我们可以看到,一次突发性的输出将包含  $C + \rho S$  字节。我们也知道,在长度为  $S$  秒的最大速度突发过程中,字节数量为  $MS$ 。因此,我们有:

$$C + \rho S = MS$$

解这个方程式,我们可以得到  $S = C / (M - \rho)$ 。在我们的例子中,  $C = 250\text{KB}$ ,  $M = 25\text{Mbps}$ ,  $\rho = 2\text{Mbps}$ ,所以,我们得到突发时间长度大约为 11ms。图 5.33(d) 和图 5.33(e) 分别显示了容量为 500KB 和 750KB 的令牌桶的情形。

关于令牌桶算法的一个潜在问题是,虽然通过小心地选择  $\rho$  和  $M$ ,最大的突发时间间隔可以被调节到适当的值,但是令牌桶算法送出的数据可能又是大量的突发性数据。通常人们希望能够降低尖峰速率,但不是回到原先漏桶算法的低速率值。

一种使流量更加平滑的办法是,在令牌桶之后再插入一个漏桶。漏桶的速率应该高于令牌桶的  $\rho$  值,但是低于网络的最大速率。图 5.33(f) 显示了在一个 500KB 的令牌桶后面接一个 10Mbps 漏桶之后产生的输出。

要完全用好所有这些方案可能并不容易。从本质上讲,网络必须要适应算法,保证被发送的分组和字节不会超过许可的数量。不管怎么样,这些工具都提供了有效的办法来将网络流量调整成更加可管理的形式,以便能够满足服务质量要求。

### 资源预留

能够调节网络流量的形状,这是保证服务质量的一个良好开端。然而,要想有效地使



用这样的信息,这隐含着一个流的所有分组必须沿着同样的路径被转发。如果这些分组被随机地分散在路由器上,那就不可能提供任何质量保证了。因此,有必要在源端和目标端之间建立起像虚电路这样的途径,属于这个流的所有分组必须沿着这条路径被转发。

一旦一个流有了特定的路径之后,我们就有可能在这条路径上预留资源,以确保可以达到预定的传输容量。有三种潜在的资源可以被保留:

- (1) 带宽;
- (2) 缓冲区空间;
- (3) CPU 周期。

第一种资源“带宽”是最为显然的。如果一个流的要求是 1Mbps,而输出线路的容量为 2Mbps,那么,试图在这条线路上直接发送三个这样的流将不可能正常工作。因此,预留带宽意味着对任何一条线路都不能超额预订。

第二种常常短缺的资源是缓冲区空间。当一个分组到来的时候,它通常被滞留在网络接口卡上,这是由硬件本身完成的。然后,路由器软件将该分组复制到一个 RAM 缓冲区中,并且对该缓冲区进行排队,以备在选择的输出线路上将分组发送出去。如果没有缓冲区空间的话,那么该分组必须被丢弃,因为没有地方可以存放该分组。对于比较好的服务质量,路由器可以为一个流保留一定的缓冲区空间,从而该分组流不必跟其他的流争用缓冲区空间。因此,当该流需要使用缓冲区空间的时候,它总是可以得到缓冲区,一直达到最大的限额为止。

最后,CPU 周期也是一种稀有资源。为了处理每一个分组,需要占用路由器的 CPU 时间,所以,一台路由器每秒钟只能处理有限多个分组。为了保证每个分组都能够得到及时的处理,必须要确保 CPU 不会过载。

初看起来,好像可以这样来计算:如果处理一个分组需要  $1\mu s$ ,那么,路由器每秒钟可以处理 1 百万个分组。这样的结论是不正确的,原因是,由于负载中的总体波动情况,路由器 CPU 总是有一些空闲的周期。如果 CPU 需要用到每一个周期才能完成它的任务的话,那么,即使由于偶尔的空闲而丢失少量的 CPU 周期,也会使它的工作积压起来而导致难以按时完成工作。

然而,即使实际负载略微轻于理论的处理能力,我们也要建立队列,而且延迟仍然会发生。请考虑这样的情形:分组随机地到达,平均到达率为每秒  $\lambda$  个分组。每个分组所要求的 CPU 时间也是随机的,CPU 的平均处理能力为每秒钟  $\mu$  个分组。假定分组的到达和处理均服从泊松(Poisson)分布,则利用排队理论可以证明,一个分组所经历的平均延迟  $T$  为:

$$T = \frac{1}{\mu} \times \frac{1}{1 - \lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1 - \rho}$$

这里  $\rho = \lambda/\mu$  是指 CPU 的利用情况。第一个因子  $1/\mu$  是指在没有竞争的情况下分组的处理时间。第二个因子是指由于跟其他流竞争而导致的减慢因素。例如,如果  $\lambda = 950\,000$  分组/秒,  $\mu = 1\,000\,000$  分组/秒,那么,  $\rho = 0.95$ ,每个分组经历的平均延迟将是  $20\mu s$ ,而不是  $1\mu s$ 。这个时间值包含了排队时间和处理时间;当负载很低的时候(即  $\lambda/\mu \approx 0$ ),也可以看出这一点。假如说在流的路径上有 30 台路由器的话,那么仅仅排队延迟就会达

到  $600\mu\text{s}$ 。

### 准入控制

现在,我们所看到的来自某个分组流的流量已经有了很好的形状,而且所有的分组可能都沿着同样的路径,沿途的每台路由器可能都已经提前预留了资源。当这样的—个分组流被转发给—台路由器的时候,该路由器必须根据它的能力和它为其他流所作出的承诺,决定允许该分组流或者拒绝它。

要决定接受或者拒绝一个分组流,这并不是简单地比较—下该流所要求的(带宽,缓冲区,CPU 周期)与路由器当前剩余的这三种资源的数量。路由器所做的事情要复杂得多。首先,尽管有的应用可能知道它们的带宽需求,但是很少有应用会知道它们对于缓冲区和 CPU 周期的需求,所以,至少需要用—种适当的方式来描述这些流。其次,有的应用比其他的应用更能够容忍偶然错误的底线。最后,有的应用可能希望对流的参数进行协商,而其他的应用可能不希望这样。例如,电影播放器通常运行在 30 帧/秒的速率上,但是,如果没有足够的带宽来支持 30 帧/秒的话,用户可能希望降低到 25 帧/秒的速率。类似地,每帧的像素数、音频带宽和其他一些属性可能也允许相应地做一些调整。

因为流协商过程中会涉及到许多方(包括发送方、接收方,以及发送方和接收方之间沿途的所有路由器),所以,在描述流的时候,必须要精确地指出那些可以被协商的参数。这样的一组参数被称为—个流规范(flow specification)。通常,发送方(比如视频服务器)生成—个流规范,在此流规范中指出它所希望使用的参数。当这份流规范沿着路径传播的时候,每台路由器对它进行检查,并根据需要修改相应的参数。对参数的修改只可以降低该流的质量,而不能提高它的质量(比如说,修改为更低的数据速率,而不是更高的数据速率)。当流规范到达另一端的时候,其中的参数就建立起来了。

举—个关于流规范的例子,如图 5.35 所示,它建立在 RFC 2210 和 2211 的基础上。例子中的流规范有 5 个参数,第一个参数是令牌桶速率,指每秒钟放到令牌桶中的字节数。这是发送方可能的最大持续传输速率(在—段较长时间间隔中的平均值)。

第二个参数是令牌桶的大小,以字节为单位。譬如,如果令牌桶速率为 1Mbps,令牌桶大小为 500KB,那么,持续填充 4 秒钟之后,令牌桶就被填满了(如果没有任何输出的话)。之后发送的任何令牌都将丢失。

第三个参数是尖峰数据率,这是指最大可容忍的传输速率,即使在较短时间内的传输率也要受它约束。发送方不得超过这个速率。

最后两个参数指定了分组的最大和最小长度,包括传输层和网络层的头(比如 TCP 和 IP)。

最小长度很重要,因为,不管—个分组有多短,在对它进行处理的时候有些 CPU 开销是固定的。—台路由器可能每秒钟能够处理 10 000 个 1KB 长度的分组,但是可能无法处理 100 000 个 50 字节长度的分组,尽管后者的数据率比前者更低—些。最大分组长度也

参数	单位
令牌桶速率	字节/秒
令牌桶大小	字节
尖峰数据率	字节/秒
最小分组大小	字节
最大分组大小	字节

图 5.35 流规范例子

很重要,因为可能会存在一些无法超越的内部网络限制。例如,如果路由路径上有一段是以太网的话,那么,不管其他网络如何处理分组,分组的最大长度将被限制于不得超过 1500 字节。

一个有趣的问题是,路由器如何将一个流规范转变成一组特定的资源预留设置呢?这种映射关系并没有被标准化,它与特定的实现相关。假设一台路由器每秒钟可以处理 100 000 个分组。如果转发给它的流的速率为 1MBps,最小和最大分组长度为 512 字节,那么,该路由器可以计算出来,它每秒钟可能从该流中获得 2048 个分组。在这种情况下,它必须为该流保留 2% 的 CPU 容量,或者也可以保留更多的 CPU 容量以避免长久的排队延迟。如果一台路由器的策略是“所有分配的 CPU 容量永远不要超过 50%”(50%意味着 2 倍的延迟),并且它已经分配了 49%,那么,该流只能被拒绝了。对于其他的资源也需要类似的计算方法。

流规范越是严密,则它对路由器越是有用。如果一个流规范声明了令牌桶速率为 5MB/s,但是分组的大小可以在 50~1500 字节之间变化,那么,分组速率将可以在 3500~105 000(分组/秒)之间变化。路由器可能对于后者这么大的数值感到惊慌不安,因而拒绝这样的流,然而,如果最小分组长度为 1000 字节的话,那么 5MBps 的流可能已经被接受了。

### 比例路由

绝大多数路由算法都试图“首先发现到达每个目标的最佳路径,然后沿着最佳路径将所有的流量发送到该目标节点”。另外有一种不同的方法,它的设计初衷是提供更高的服务质量,其做法是将到达每个目标节点的流量分散到多条路径上。由于路由器通常不可能完全了解全网络范围内的流量情况,所以,将流量分散到多条路径上的惟一可行做法是使用本地可用的局部信息。一种简单的方法是将流量平均分配到输出链路上,或者根据这些链路的容量按比例进行分配。然而,更为复杂的算法也是可行的,参见(Nelakuditi and Zhang, 2002)。

### 分组调度

如果一台路由器正在处理多个流,那么,这样的情形是非常危险的:一个流占用了太多的系统资源,而其他的流得不到资源。如果按照分组到达的顺序来处理分组,则意味着一个激进活跃的发送方将会占用沿途路由器上的大多数资源,同时降低了其他发送方的服务质量。为了避免这样的情形,研究人员已经设计了各种分组调度算法(Bhatti and Crowcroft, 2000)。

第一个分组调度算法是公平排队(fair queueing)算法(Nagle, 1987)。该算法的本质是,路由器为每一条输出线路使用一组单独的队列,每个流一个队列。当一条线路空闲的时候,路由器轮流扫描这些队列,从下一个队列中取出第一个分组。按照这种方法,如果有  $n$  台主机在争用一条指定的输出线路,那么,在每发送出的  $n$  个分组中,每台主机各有一个分组。主机发送的分组再多也不可能提高这个比例。

尽管这个算法是一个很好的开端,但是它有一个问题:那些使用大分组的主机得到

的带宽,比那些使用小分组的主机要更多一些。Demers 等(1990)提出了一种改进算法,该算法中的轮流循环扫描是逐个字节进行的,而不是逐个分组进行的。实际上,它反复地扫描队列,以字节为单位,直到找到每个分组结束时刻的那个滴答。然后,该算法按照每个分组结束时刻的顺序对它们进行排列,并且按照同样的顺序发送这些分组。图 5.36 演示了该算法。

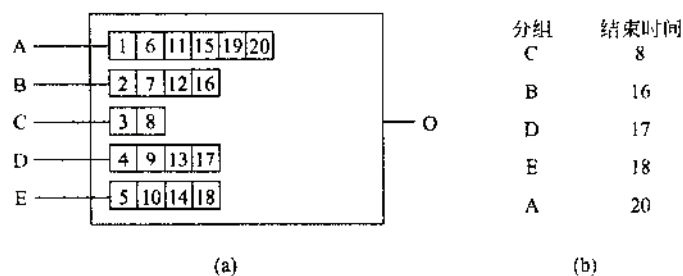


图 5.36

(a) 在路由器的线路 O 上有 5 个分组在排队; (b) 5 个分组的结束时刻

在图 5.36(a)中,我们看到,5 个分组的长度从 2 到 6 个字节不等。在(虚拟的)时钟滴答 1 时刻,线路 A 上的分组的第一个字节被发送出去,然后是线路 B 上的分组的第一个字节,依此类推。经过 8 个时钟滴答之后,第一个结束的分组是 C。这些分组被保存的顺序如图 5.36(b)所示。在没有新分组到来的情况下,这些分组将按照图中列出的顺序被发送出去,即从 C 到 A。

这个算法的一个问题是,它赋予所有的主机同等的优先级。在许多情况下,视频服务器期望获得比普通的文件服务器更多的带宽,因而它们希望每个滴答得到两个或者更多的字节。这样修改之后的算法称为加权的公平排队(weighted fair queuing)算法,它已经被广泛使用了。有时候,算法中的权值等于从一台机器输出的流的数目,所以,每个进程得到相等的带宽。(Shreedhar and Varghese, 1995)中讨论了该算法的一个高效实现。现在的趋势是,经过一台路由器或者交换机的分组的实际转发工作往往由硬件来完成了(Elhanany et al., 2001)。

### 5.4.3 综合服务

从 1995 到 1997 年之间,IETF 做了很大的努力来设计流式多媒体的体系结构。这项工作最后产生了 20 多个 RFC,从 RFC 2205~2210 开始。此项工作的一般叫法为基于流的算法(flow-based algorithm)或者综合服务(integrated service)。这些算法同时针对单播和多播应用。作为单播的一个例子是,用户从一个新闻站点抓取一段视频片段:作为多播的一个例子是,多个数字电视台将它们的节目以 IP 分组流的方式广播到各地的接收端。接下来我们将注意力集中在多播上,因为单播可以看作是单播的一个特例。

在许多多播应用中,组中的成员可能会动态地变化,例如,用户进入到一个视频会议中,很快就厌烦了,然后切换到一个肥皂剧或者橄榄球频道。在这样的条件下,让发送方提前预留带宽的做法并不能很好地工作,因为它将要求每个发送方必须跟踪听众们的进

来和离开情况。对于一个拥有上百万个用户的电视传输系统而言,这样的设计显然不会工作得很好。

### 资源预留协议

在综合服务体系结构中,主要的 IETF 协议是资源预留协议 (Resource reSerVation Protocol, RSVP)。它是在 RFC 2205 和其他一些 RFC 文档中描述的。利用该协议可以完成资源预留工作;若要发送数据则还需要使用其他的协议。RSVP 允许多个发送方给多个接收组传送数据,也允许接收方单独自由地切换信道,并且在优化使用带宽的同时消除拥塞的发生。

在 RSVP 协议最简单的形式中,它通过生成树来实现多播路由,关于用生成树来实现多播路由前面已经讨论过了。每个组都分配一个组地址。为了给一个组发送分组,发送方把该组的地址放到这些分组中。然后,标准的多播路由算法建立起一棵覆盖所有组成员的生成树。此路由算法并不是 RSVP 的一部分。与常规的多播传输惟一不同的地方在于,这里的多播传输需要一些额外的信息,这些信息被周期性地多播传输给生成树中的路由器,告诉它们在内存中维护特定的数据结构。

举例来说,请考虑图 5.37(a)中的网络,主机 1 和主机 2 是多播发送方,主机 3、4 和 5 是多播接收方。在这个例子中,发送方和接收方是分离的,但是一般情况下,这两个集合是重叠的。针对主机 1 和主机 2 的多播树分别如图 5.37(b)和图 5.37(c)所示。

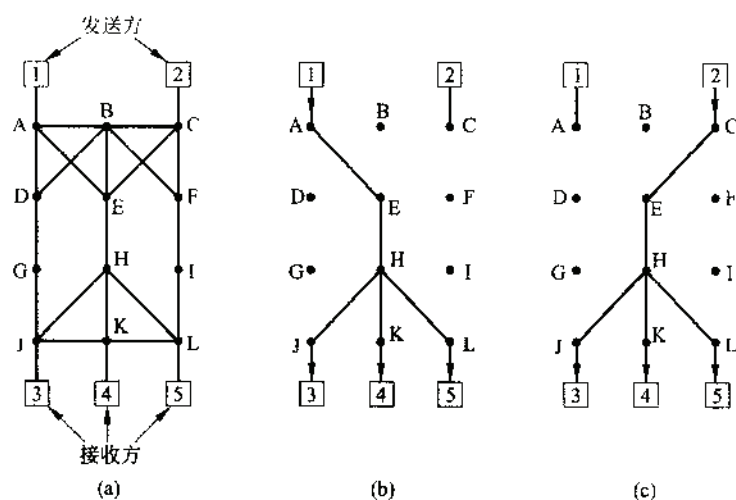


图 5.37

(a) 一个网络; (b) 主机 1 的多播生成树; (c) 主机 2 的多播生成树

为了获得更好的接收效果并且消除拥塞,一个组中的任何一个接收方都可以发送一条预留消息,沿着树上溯到发送方。利用前面讨论过的逆向路径转发算法,该消息被传播到发送方。在沿途每一跳,路由器会看到此预留要求,并预留必要的带宽。如果带宽不够用的话,它往回报告失败。当这条预留消息到达多播源的时候,从发送方到接收方一路上



沿着生成树已经预留了带宽。

图 5.38(a)显示了这种资源预留的一个例子。在这里,主机 3 请求一条通向主机 1 的信道。一旦该信道已经被建立起来了,则从主机 1 到主机 3 的分组流将不会再出现拥塞。现在请考虑一下,如果接下来主机 3 为了要同时观看两套电视节目,所以要预留一条通向另一个发送方(即主机 2)的信道,那会怎么样呢? 第二条路径将会被预留,如图 5.38(b)所示。请注意,从主机 3 到路由器 E 之间需要两条独立的信道,因为有两个独立的流需要通过这条路径。

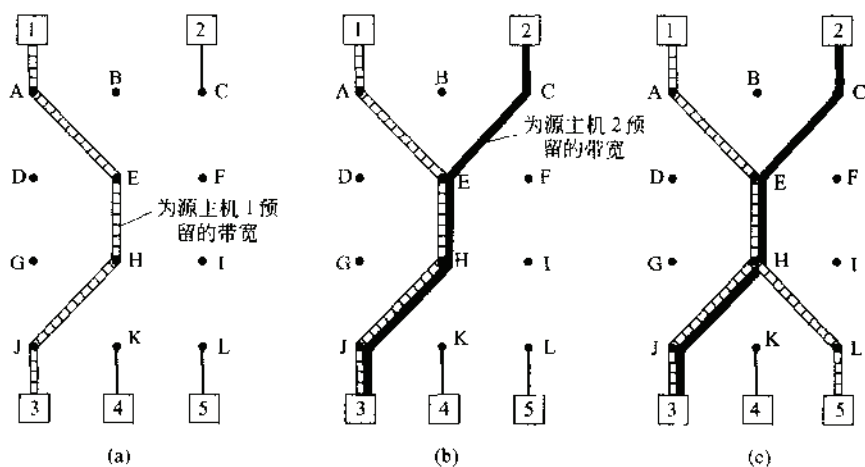


图 5.38

- (a) 主机 3 请求一条通向主机 1 的信道; (b) 然后主机 3 请求另一条通向主机 2 的信道;  
(c) 主机 5 请求一条通向主机 1 的信道

最后,在图 5.38(c)中,主机 5 决定观看主机 1 传送的节目,所以它也请求预留资源。首先,从主机 5 到路由器 H 之间,专门的带宽被预留出来。然而,路由器 H 看到,它已经有了一份来自主机 1 的流,所以,如果必要的带宽已经被预留了的话,它就没有必要再次预留了。注意,主机 3 和 5 请求的带宽数量可能会不相同(比如,主机 3 有一台黑白电视机,所以它不需要彩色信息),所以,被预留的带宽数量必须足够大,以便满足最大需求的接收方。

在进行资源预留的时候,接收方可以(有选择地)指定一个或者多个期望的数据源。它也可以指定“在预留期间这些选择是否固定不变”,或者指定“它是否希望以后还可以改变数据源”。路由器利用这些信息来优化带宽的使用计划。尤其是,如果两个接收方都同意以后不再改变数据源的话,那么它们只需共享一条路径即可。

在完全动态的情形下,采用这种策略的理由是:将被预留的带宽与数据源的选择分离开。一旦接收方已经预留了带宽,那么,它可以切换到另一个数据源,并且保留现有的路径上对于新数据源仍然有效的那一部分。例如,如果主机 2 正在传输几个视频流,那么,主机 3 可以随意地切换这几个流,而根本不用改变它所做的预留;路由器并不关心接收方正在观看什么节目。

#### 5.4.4 区分服务

基于流的算法有能力为一个或者多个流提供非常好的服务质量,因为它们的路由路径上预留了必要的资源。然而,这些算法也有一个缺点,要求提前一步建立每一个流。当应用中存在几千几万个流的时候,这种做法的扩展性非常差。而且,它们需要在路由器中为每个流维护一份内部状态,这使得它们易受“路由器崩溃”的攻击。最后,这些算法还要求路由器代码作实质性的改变,而且为了建立数据流,这些算法也牵涉到路由器之间复杂的消息交换。因此,RSVP 几乎没有具体的实现,甚至类似于 RSVP 的实现也很少。

由于这些原因,IETF 设计了另一个更加简单的服务质量方法,该方法不要求提前建立流,它主要由每台路由器在局部范围内实现,而不牵涉到整条路径。这种方法称为**基于类别(class-based)**的服务质量(相对于基于流的服务质量)。IETF 也已经对这种方法的体系结构进行了标准化,称为**区分服务(differentiated service)**。RFC 2474、2475 和其他一些 RFC 文档对区分服务进行了描述。下面我们来介绍区分服务。

区分服务(DS)可以由一组路由器来提供,这些路由器构成了一个管理域(比如一个 ISP 或者一家电话公司)。管理规范中定义了一组服务类别,每个服务类别与相应的转发规则相对应。如果一个客户已经申请了区分服务,那么,进入到该管理域的客户分组中可能携带一个服务类型(Type of Service)域,在该管理域中,有些类别(比如高级服务)的分组比其他的类别可以获得更好的服务。属于同一个类别的通信流量可能要先经过处理以便符合特定的形状特征,比如通过一个具有特定输出速率的漏桶。如果运行商对于商机比较敏感的话,它可能对每一个高级服务类别的分组收取额外的费用,或者每个月收取固定的额外费用并允许最多 N 个高级类别的分组。请注意,这种方案并不要求提前建立通道,也没有资源预留,也不需要花时间为每个流进行端到端的协商,这是不同于综合服务的地方。这也使得 DS 相对容易实现。

实际上,在其他的工业领域中也有基于类别的服务。例如,包裹托运公司通常提供昼夜服务、两天内送达和三天内送达服务。航空公司提供头等舱、商务舱和经济舱服务。长距离的火车通常也有多个服务等级。甚至巴黎的地铁还有两种服务类别。对于分组而言,类别之间的差异可以体现在延迟、抖动、在拥塞情况下被丢弃的概率,以及其他的可能性(如果不是以太网帧的话,还有其他一些可能)。

为了更清楚地理解基于流的服务质量与基于类别的服务质量之间的差异,请考虑 Internet 电话的例子。如果使用基于流的方案,则每个电话呼叫都有专用的资源,从而服务质量可以保证。如果使用基于类别的方案,则所有的电话呼叫合起来共享同一份资源,即预留给电话类别的那份资源。其他类别的分组,比如文件传输中的分组,不可能夺走电话类别的资源,但是,任一个电话呼叫都没有单独预留给它的私有资源。

#### 快速型转发

如何选择服务类别,这要取决于每个运行商,但是,由于通常情况下大多数分组需要在不同运行商的子网之间进行转发,所以,IETF 正在定义与网络无关的服务类别。最简单的服务类别是**快速型转发(expedited forwarding)**,所以,我们先从这种类别开始讨论。

RFC 3246 描述了有关快速型转发的细节。

在“快速型转发”背后的思想非常简单。我们可以把服务类别分为两种：常规的和快速的。大多数通信流量应该属于常规流量，但是，有一小部分分组属于快速类别。快速类别的分组应该可以直接通过子网，就好像不存在其他任何分组一样。这种“双管道”系统的一个符号化表示如图 5.39 所示。请注意，这里只有一条物理线路。图中的两根逻辑导管表示了一种预留带宽的方法，而不是第二根物理线路。

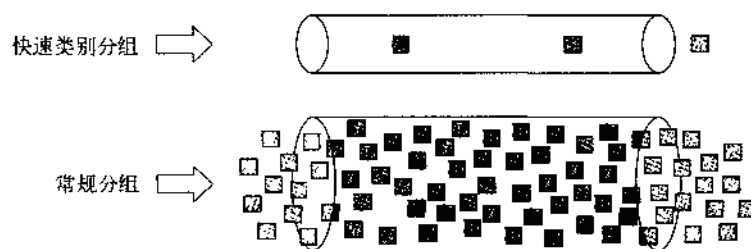


图 5.39 快速类别的分组经过一个网络

实现这种策略的一种做法是，在编写路由器程序的时候，每条输出线路有两个输出队列，一个用于快速类别的分组，另一个用于常规分组。当一个分组到来的时候，根据它的类别排入相应的队列。路由器应该使用类似于“加权的公平排队”这样的分组调度机制。例如，如果 10% 的通信流量是快速类别的，而 90% 的流量是常规的，那么，可以把 20% 的带宽专门分配给快速类别的分组，剩下的带宽分配给常规分组。这样做实际上相当于为快速类别的流量分配了两倍于它们所需要的带宽，从而使它们有相对较短的延迟。这种分配方案也可以这样来实现：每隔 4 个常规分组传输一个快速类别的分组（假定这两种类别的分组长度分布是类似的）。按照这种做法，期望达到的目标是，即使当一个子网的实际负载很重的时候，快速类别的分组看到的仍然是一个没有超载的子网。

### 确保型转发

一种更加精细的管理服务类别的方案称为**确保型转发** (assured forwarding)。它是在 RFC 2597 中描述的。它规定了 4 种优先级别，每种级别都有它自己的资源。而且，对于正在经历拥塞条件的分组，它定义了三种丢弃概率：低、中、高。将两者结合起来，共有 12 种服务类别。

图 5.40 显示了在确保型转发环境下一处理分组的办法。第 1 步是将分组分类，每个分组被归入 4 种优先级别之一。这一步有可能是在发送主机上完成的（如图中所示），也可能是在入口（第一个）路由器上完成的。在发送主机上实现分类的优点是，它可以利用更多的信息来确定哪些分组属于哪些流。

第 2 步是根据分组的类别对它们进行标记。为了做到这一点，需要使用分组头部的一个域。幸运的是，在 IP 分组中有一个 8 位的服务类型 (type of service) 域可以使用，稍后我们将会看到 IP 分组的各个域。RFC 2597 规定了这 8 位中的 6 位被用于指定服务类别，从而为过去的服务类别和将来的服务类别保留了必要的编码空间。

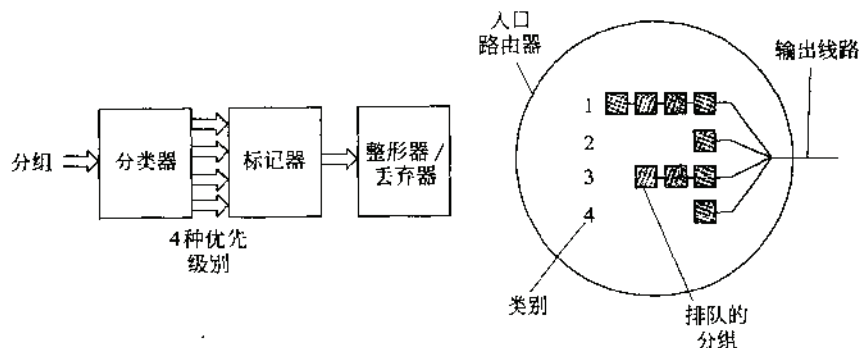


图 5.40 确保型转发数据流的一种可能的实现

第 3 步是将分组通过一个整形器/丢弃器,它们可能会延迟或者丢弃一部分分组,以便将 4 个分组流调整到可以接受的形状,比如,通过漏桶或者令牌桶做到这一点。如果分组太多的话,那么,根据分组的丢弃类别,有些分组可能会在这个地方被丢弃。在这里,也可能使用一些涉及到度量或者反馈机制的细致方案。

在这个例子中,前面三个步骤都是在发送主机上执行的,所以,这三个步骤之后的输出流现在被送入入口路由器。值得提及的是,这些步骤可以由特殊的网络软件来完成,甚至也可以由操作系统来完成,这样做的好处是可以避免改变现有的应用系统。

#### 5.4.5 标签交换和 MPLS

当 IETF 完成了综合服务和区分服务的时候,有些路由器厂商还在努力提出更好的转发方法。其工作思路是,在每个分组的前端增加一个标签,然后根据这个标签而不是根据目标地址进行路由。将这个标签做成内部表中的一个索引值,就可以把“找到正确的输出线路”这件事情变成简单的表格查询。利用这样的技术,路由过程可以很快地完成,而且沿途预留必要的资源也很容易做到。

显然,标记分组流的做法非常接近于虚电路的思想。X.25、ATM、帧中继,以及所有其他包含虚电路子网的网络也会在每个分组中放上一个标签(即虚电路标识符),也会在内部表中查询,并且根据查到的表项进行路由。虽然实际的情况是,Internet 社团中的许多人非常不喜欢面向连接的网络,但是,这种思想看起来又倒回去了,而这一次却提供了快速的路由和服务质量。然而,Internet 构造路由路径的处理方法与面向连接的网络构造路由路径的方法有本质的不同,所以,这里的标签技术肯定不是传统的电路交换。

这种“新的”交换思想有几个不同的(私有的)名字,其中包括 **label switching** 和 **tag switching**(中文都翻译成标签交换)。最终 IETF 开始对这种思想进行标准化,它所用的名字是 **MPLS(MultiProtocol Label Switching, 多协议标签交换)**。我们以后也称它为 MPLS, RFC 3031 和其他许多 RFC 文档描述了 MPLS。

顺便提一下,有的人会这样明确地区分路由(routing)和交换(switching)。他们认为,路由是指这样一个过程:在一张表中查找一个目标地址,以便确定该往哪里发送;相反,交换过程则是利用分组中的一个标签作为一张转发表中的一个索引。然而,请注意,

这些定义并不是完全通用的。

第一个问题是该把标签放在哪里。由于 IP 分组并不是针对虚电路而设计的,所以在 IP 头部并没有可以存放虚电路号的空间。由于这个原因,必须要在 IP 分组的前端加上一个新的 MPLS 头。如果从一台路由器到另一台路由器之间的线路使用 PPP 作为帧协议的话,那么,帧格式中包含了 PPP、MPLS、IP 和 TCP 头,如图 5.41 所示。因此,从这种意义上讲,MPLS 位于 2.5 层上。

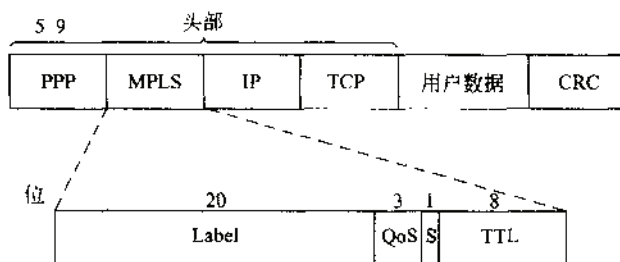


图 5.41 利用 IP、MPLS 和 PPP 传送一个 TCP 段

一般的 MPLS 头有 4 个域,其中最重要的是 Label 域,它存放的是索引。QoS 域指明了服务的类别。S 域涉及到在层次网络中叠加多个标签的做法(下面会进一步讨论)。如果 S 为 0,则该分组被丢弃,这个特性可以防止在路由不稳定的情况下无限循环的问题。

因为 MPLS 头既不属于网络层分组,也不属于数据链路层的帧,所以,MPLS 是一个独立于这两层的扩展。以外,这种属性也意味着我们有可能建立起 MPLS 交换,它既能够转发 IP 分组,也可以转发 ATM 信元,取决于它所看到的是什么样的数据。这个特性也正是 MPLS 名字中“多协议”的来源。

当一个经过 MPLS 增强的分组(或信元)到达一台支持 MPLS 的路由器的时候,该标签被当作内部表中的一个索引,以确定该使用哪一条输出线路,同时也确定该为它使用哪一个新的标签。这里交换标签的过程是所有的虚电路子网都要使用的,因为标签只有局部的含义,两台不同的路由器可能会将不相关的分组(但是它们具有相同的标签)转送给同一条输出线路上的另一台路由器。为了在另一端能够区分这些分组,所以,在每一跳上必须重新映射标签。我们在图 5.3 中已经看到过这种机制了。MPLS 使用了同样的技术。

MPLS 与传统虚电路的一个区别是聚集的级别。在通过子网的时候,每个分组流有它自己的标签集合肯定是可以做得到的,然而,对于路由器来说,更常见的做法是将多个目标端相同(即送达同一个特殊路由器或者 LAN)的流组合起来,并为这些流使用同一个标签。这些被组合起来共享同一个标签的分组流称为属于同一个 FEC (Forwarding Equivalence Class, 转发等价类)。该类别不仅规定了这些分组的去向,而且也隐含了它们的服务类别(从区分服务的意义上),因为从转发的角度而言,所有的分组都被同样对待。

若使用传统的虚电路路由方法,要想把几条具有不同端点(end point)的独立路径组



合到同一个虚电路标识符上是不可能的,因为在最终目标端无法将它们区分开。如果使用 MPLS,那么,因为分组中除了标签以外,仍然包含它们的最终目标地址,所以,在标签路径的末端,标签头可以被去掉,然后利用网络层的目标地址,按照常规的方法继续向前转发。

MPLS 方案和传统的 VC 设计之间一个主要的区别是如何建立转发表。在传统的虚电路网络中,当一个用户想要建立一个连接的时候,首先发送一个专门的分组(setup packet)到网络中,以便将路径创建起来,并建立起转发表的表项。MPLS 并没有使用这种工作方式,因为它并没有这样的提前为每个连接建立路径的步骤(如果那样的话,将会打破现有的 Internet 软件的工作方式)。

相反,在 MPLS 中,有两种创建转发表项的方法。第一种是**数据驱动(data-driven)**的方法,当一个分组到来的时候,第一台接收该分组的路由器与该分组下一跳的下游路由器联系,请它为这个分组流生成一个标签。这种方法被递归地应用到沿途的路由器上。实际上,这是一种按需创建虚电路的做法。

完成这项扩散工作的协议必须非常小心地处理每一个环节,以避免产生环。它们通常会使用一种称为**有色绳索(colored thread)**的技术。FEC 的后向传播就好比是一条惟一颜色的绳索又被拉回到子网中。如果一台路由器看到了一种它已经见过的颜色,那么它知道这是一个环,于是就可以采取补救措施。这种数据驱动的方法主要被用于那些底层传输技术采用 ATM 的网络(比如许多电话系统)。

另一种在非 ATM 网络中使用的方法是**控制驱动(control-driven)**的方法。它有一些变种,其中一种方法如下面所描述。当一台路由器启动的时候,它检查在哪些路由路径中自己是最终的目标(比如在它的 LAN 中有哪些主机)。然后,它为这些路由路径创建一个或者多个 FEC,并为每一个 FEC 分配一个标签,再将这些标签传递给它的邻居路由器。然后,邻居路由器将这些标签放到它们的转发表中,并发送新的标签给它们的邻居,直到所有的路由器都得到了路径。在路径建立过程中,路由器也可以预留资源,以便保证适当的服务质量。

MPLS 也可以同时运行在多个层次上。在最高层次上,每个网络承运商可以被看作一种元路由器(metarouter),在源和目标之间有一条路径通过多个元路由器。这条路径可以使用 MPLS。然而,在每个承运商的网络中也可以使用 MPLS,这导致了第二层次的标签。实际上,一个分组可以携带一个完整的标签栈。图 5.41 中的 S 位允许一台正在删除标签的路由器知道该分组是否还有其他的标签。对于最低层次的标签,该位被设置为 1;对于所有其他的标签,该位被设置为 0。在实践中,这项设施主要被用来实现虚拟私有网络和递归隧道。

尽管 MPLS 背后的基本思想非常直截了当,但是它的细节却异常复杂,并且它有许多变种和优化,所以,我们不再继续讨论这个话题了。有关更多的信息,请参考(Davie and Rekhter, 2000; Lin et al., 2002; Peppinjak and Guichard, 2001; and Wang, 2001)。

## 5.5 网络互联

到现在为止,我们一直(隐含着)假设所讨论的网络是一个同质的网络,即每台机器在每一层上使用同样的协议。不幸的是,这样的假设太过于乐观了。实际上,有许多不同的网络存在,包括 LAN、MAN 和 WAN。而且,在每一层上,有大量的协议在广泛使用。在本节中,我们将详细地讨论当两个或者多个网络被连接起来构成一个互联网(internet)的时候所涉及的一些问题。

长期以来,人们对两种观点一直争论不休,一种观点认为,今天存在这么多种网络类型只是暂时现象,随着人们都找到了自己最喜爱的网络之后,这种现象就会消失;另一种观点则认为,这种现象是不可避免的,并且会永远存在下去,这是由客观世界的本质特征决定的。既然有这么多种不同的网络,自然也就意味着有不同的协议。

我们相信,大量不同类型的网络(以及协议)将长期共存,原因如下所述。首先,有些网络已经有了非常大的安装基数。几乎所有的个人计算机都在运行 TCP/IP。许多大型商业机构中的大型主机在运行 IBM 的 SNA。相当一部分电话公司在运行 ATM 网络。有些 PC 局域网仍然在使用 Novell NCP/IPX 或者 AppleTalk。最后,无线网络是一个正在兴起的潮流,它也会伴随着许多不同的协议。由于遗留的问题以及新技术的产生,这种多网络共存的局面还将持续很多年,而且并不是所有的供应商都认为“让他们的顾客能够很容易地迁移到另一个供应商的系统是符合其利益的”。

第二,随着计算机和网络变得越来越便宜,购买设备的决策权也在组织中逐渐向下转移。许多公司有类似这样的策略:购买超过一百万美元的设备必须要经过最高管理层批准,购买超过十万美元的设备必须经过中级管理层批准,而购买十万美元以下的设备只需部门主管批准即可,无需更高层的批准。这种策略很容易导致这样的局面:工程部门安装的是 UNIX 工作站,并且运行 TCP/IP;而市场部门安装的是 Mac,并且运行 AppleTalk。

第三,不同的网络(比如 ATM 和无线网络)使用完全不同的技术,因此一个必然的结果是,随着新的硬件技术的不断发展,也需要新的软件来配合这些新的硬件。例如,如今的普通家庭就好像 10 年前的大多数办公室一样:家里有很多计算机,它们相互之间不能通信。在将来,家里的电话、电视机和其他一些电器可能都是网络化的,所以用户可以远程对它们进行控制。这样的新技术毫无疑问会带来新的网络和新的协议。

为了看清楚不同的网络有可能用什么样的方式连接在一起,请考虑图 5.42 中的例子。这里我们看到一家公司的网络分散在多个场所,这些场所通过一个广域 ATM 网络连在一起。其中一个场所配备了 FDDI 光纤骨干网,它将一个以太网、一个 802.11 无线 LAN 和公司数据中心的 SNA 网络连接在一起。

将所有这些网络连接起来的目的是,允许任何一个网络中的用户可以与其他网络中的用户进行通信,也允许任何一个网络中的用户可以访问其他网络中的数据。要想实现这个目标,就意味着要将分组从一个网络发送到另一个网络中。由于不同的网络在很多重要的环节上有不同的处理方法,所以,要将一个网络中的分组转发到另一个网络中并不

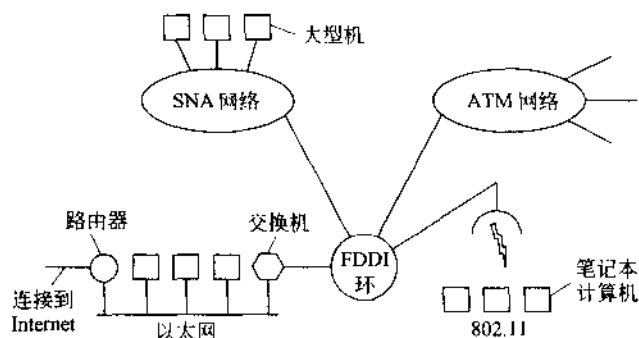


图 5.42 一组相互连接的网络

总是那么容易的,下面我们将会看到这一点。

### 5.5.1 网络的不同之处

不同的网络在很多方面都有所不同。有些不同之处位于物理层或者数据链路层,比如不同的调制技术或者不同的帧格式。我们这里并不关心这些不同之处。相反,我们在图 5.43 中列出了在网络层上可能出现的一些差异,正是这些差异使得网络互联比单个网络中的操作更加困难。

项目	一些可能性
所提供的服务	面向连接的服务,或者无连接的服务
协议	IP、IPX、SNA、ATM、MPLS、AppleTalk 等
编址方式	平面的(802)或者层次的(IP)
多播	支持或者不支持(以及广播)
分组大小	每个网络都有它自己的最大值限制
服务质量	支持,或者不支持;许多不同的种类
错误处理	可靠的、有序的,以及无序的递交
流控制	滑动窗口、速率控制,或者其他控制手段,或者无控制
拥塞控制	漏桶、令牌桶、RED、抑制分组等
安全性	隐私规则、加密等
参数	不同的超时值、流规范等
记账	按连接时间、按分组、按字节,或者根本不记账

图 5.43 网络的某些不同之处

当一个网络上的某个源发送出的分组必须要经过一个或者多个外部网络才能到达目标网络(目标网络的类型可能与源网络不同)的时候,网络之间的接口处可能会产生许多问题。首先,当分组从一个面向连接的网络传送到另一个无连接的网络的时候,这些分组

可能要被重新排序,这是发送方不期望的行为,而且接收方也不打算处理顺序问题。在不同网络之间传送分组常常需要进行协议转换,如果所要求的功能在新的协议中难以表达的话,那么这种协议转换就会非常困难。有时候还必须要进行地址转换,这可能会涉及到某种类型的目录系统。多播分组在通过一个不支持多播的网络的时候,可能需要针对每个目标生成单独的分组。

不同网络所使用的最大分组长度也是不同的,这可能是一件非常让人头痛的事情。请问,当一个 8000 字节的分组通过一个最大分组长度为 1500 字节的网络的时候,你该怎么办呢? 当一个要求实时递交的分组通过一个并未提供实时性保证的网络的时候,服务质量上的差异也是一个问题。

对于不同的网络,错误控制、流控制和拥塞控制通常也是不一样的。如果源和目标都期望所有的分组都能够毫无错误地被按序递交,但是存在一个中间网络一碰到拥塞就会丢弃分组,那么许多应用将无法工作。而且,如果分组可能会漫无目标地游荡一段时间,然后突然冒出来并且被递交给目标的话,那么,除非应用程序专门针对这种行为进行处理,否则问题会接踵而至。不同的安全机制、参数设置和记账规则,甚至国家的隐私法律也可能会引起问题。

### 5.5.2 网络如何连接起来

网络可以通过不同的设备相互连接起来,在第 4 章中我们已经看到过许多种这样的设备。现在我们简短地回顾一下。在物理层上,通过中继器或者集线器可以将网络连接起来,它们通常只是简单地将数据位从一个网络搬移到另一个同类型的网络中。中继器和集线器大多数是模拟设备,它们并不理解有关数字协议方面的内容(它们只不过重新生成模拟信号)。

再往上一层我们可以使用网桥和交换机,它们运行在数据链路层上。它们可以接受帧,检查 MAC 地址,并且将这些帧转发到另一个不同的网络中。在这个过程中,它们可以做一点小小的协议转换,比如,从以太网转换到 FDDI 或者转换到 802.11。

在网络层上,我们可以用路由器将两个网络连接起来。如果两个网络的网络层截然不同,那么路由器有可能会转换分组格式,不过,这样的分组转换现在越来越少了。能够处理多种协议的路由器称为**多协议路由器(multiprotocol router)**。

在传输层上,我们可以使用传输网关(transport gateway)。所谓传输网关,是指两个传输层连接之间的接口。例如,通过传输网关可以让一个分组流在 TCP 网络和 SNA 网络之间传递,由于这两个网络有不同的传输协议,所以,从本质上讲,这里的传输网关把一个 TCP 连接和一个 SNA 连接粘连起来了。

最后,在应用层上,应用网关可以翻译消息的语义。举例来说,Internet 电子邮件(RFC 822)和 X.400 电子邮件之间的网关必须解析电子邮件消息,并且改变各个头域。

在本章中,我们将注意力集中在网络层的互联方法上。为了看清楚网络层的互联如何不同于数据链路层的交换过程,请参考图 5.44。在图 5.44(a)中,源机器 S 希望给目标机器 D 发送一个分组。这两台机器位于不同的以太网网络中,并且这两个网络通过一台交换机连接起来。S 将分组封装到一帧中,并将它发送出去。当该帧到达交换机的时候,交

交换机检查它的 MAC 地址,知道该帧的目的地在 LAN 2。然后交换机将该帧从 LAN 1 中删除,并将它放到 LAN 2 上。

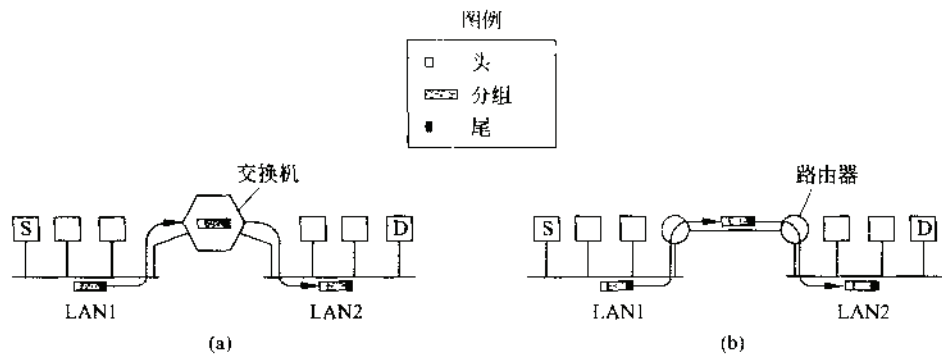


图 5.44

(a) 通过交换机连接起来的两个以太网; (b) 通过路由器连接起来的两个以太网

现在我们来考虑同样的网络结构,但是两个以太网通过一对路由器连接起来,而不是通过交换机连接起来。这两台路由器通过一条点到点的线路连起来,它可能是一条上千公里长的租用线路。现在,路由器接收到该帧之后,它从该帧的数据域中提取出对应的分组,然后检查分组中的地址(例如,这是一个 IP 地址),并且在它的路由表中查找该地址。路由器根据这个地址,决定将该分组发送给远程路由器,有可能它要将该分组封装到另一种帧类型中,取决于点到点线路所用的协议。在另一端,该分组又被放到一个以太网帧的数据域中,然后被发送到 LAN 2 上。

交换式(或者网桥式)网络与路由网络之间的一个本质区别是,通过交换机(或者网桥),整个帧是以 MAC 地址为基础进行传输的;而对于路由网络,路由器从帧中提取出分组,然后利用分组中的地址来决定它的目标去向。交换机不必理解分组中所使用的网络层协议,而路由器必须要理解分组中的网络层协议。

### 5.5.3 级联虚电路

有两种可能的网络互连方式:一种是面向连接的虚电路子网级联;另一种是数据报互连方式。现在我们按照顺序介绍这两种风格,但是,首先我需要提醒一下。在过去,大多数(公共的)网络是面向连接的(帧中继、SNA、802.16 和 ATM 仍然是面向连接的)。后来,随着 Internet 的普及,数据报网络变得非常流行。然而,如果你认为数据报网络将会永恒的话,那将会是一个错误。在网络领域中,惟一永恒的事情就是变化。随着多媒体网络应用变得越来越重要,有可能面向连接的网络风格又会以这样或那样的形式重新回来,因为有了连接以后,保证服务质量比无连接情况下要容易得多。因此,我们下面花一点篇幅介绍一下面向连接的网络互连。

在级联虚电路模型中,如图 5.45 所示,建立一个到达远程网络中主机的连接所使用的方式,与建立常规连接的方式非常类似。子网看到了连接的目标端在远处(本子网以外),于是建立一条虚电路通向本子网中离目标网络最近的路由器。然后,它再建立一条



从该路由器到达外部网关(多协议路由器)的虚电路。此外部网关在它的内部表中记录下这条虚电路,并且继续前行,建立另一条虚电路,到达下一个子网中的路由器。这个过程持续下去,直至到达目标主机为止。

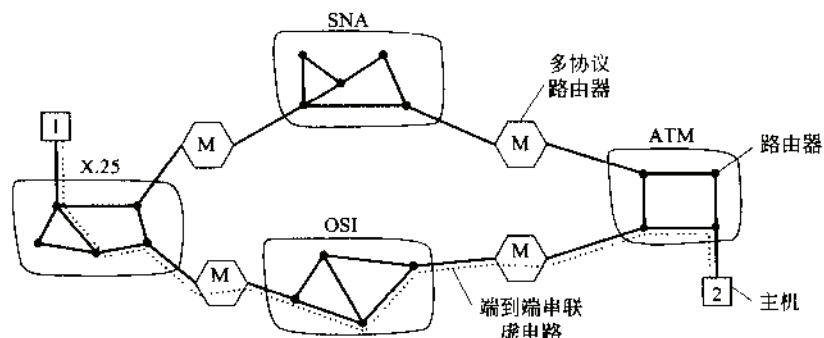


图 5.45 使用级联虚电路实现网络互连

一旦数据分组开始沿着这条路径被传送,每个网关负责将进来的分组转发出去,必要的时候它可以转换分组的格式和虚电路号。很显然,所有的数据分组必须要经过同样的网关序列。因此,一个流中的分组永远不会因为网络而造成错序。

这种方法的本质特征是,从源主机到目标主机之间沿途经过一个或者多个网关,在这条路径上建立起一个虚电路序列。每个网关维护一些内部表,记录下哪些虚电路经过了该网关、这些虚电路分别通向哪里,以及新的虚电路号是什么等。

如果所有的网络都有大致相同的特性,那么,这种方案可以工作得很好。例如,如果所有的网络都可以保证可靠地递交网络层分组,那么,在不考虑沿途路由器崩溃的情况下,从源到目标的分组流也是可靠的。类似地,如果所有的网络都不保证可靠递交的话,那么,虚电路串联起来之后也是不可靠的。另一方面,如果源机器所在的网络能够保证可靠递交,但是中间网络可能会丢失分组,那么,虚电路串联之后将会完全改变服务的本质。

在传输层上,串联虚电路也很常见。特别是,它有可能使用 SNA(该 SNA 终止于某一个网关)建立一条数据位管道(bit pipe),然后从该网关到下一个网关使用 TCP 连接。按照这种方式,我们可以建立起跨越不同网络和协议的端到端虚电路。

#### 5.5.4 无连接的网络互连

另一种互连网络模型是数据报模型,如图 5.46 所示。在这种模型中,网络层为传输层提供的惟一服务是,将数据报送入子网,然后就不管了。在这里,网络层上根本没有虚电路的概念,更别提虚电路串联了。这种模型并不要求属于同一个连接的所有分组必须经过同样的网关序列。在图 5.46 中,从主机 1 到主机 2 的数据报在经过互连网络的时候并没有走同样的路由路径。路由器针对每一个分组做出路由决定,它可能会根据当前时刻的流量情况来转发每一个分组。这种策略可以使用多条路由路径,从而有可能获得比串联虚电路模型更高的带宽。另一方面,在目标主机上,假定所有的分组都能够到达,那么,它们的到达顺序是不保证的。

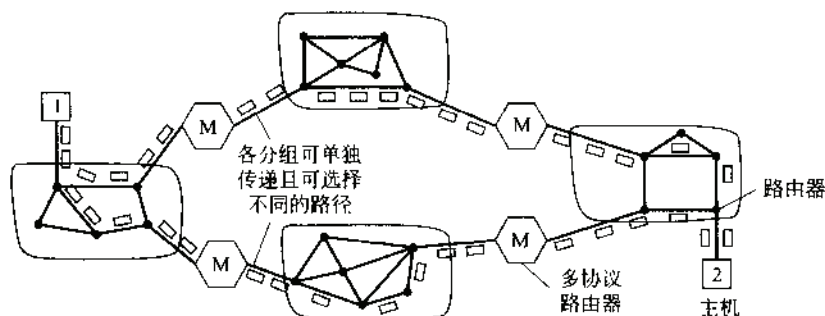


图 5.46 一个无连接的网络互连

图 5.46 中的模型并不像看起来的那么简单。首先,如果所有的网络都有它自己的网络层协议,那么,对于一个分组而言,要从一个网络传输到另一个网络中是不可能的。你可以把多协议路由器的功能想象成,它实际上是把分组从一种格式转换成另一种格式,但是,除非这两种格式对于同样的信息域有非常相近的表示,否则,这种转换总是不完全的,最终注定要失败。由于这个原因,通常很少使用转换。

第二,更严重的是编址问题。请想象这样一种简单的情形:Internet 上的一台主机试图向邻近的 SNA 网络中的一台主机发送一个 IP 分组。IP 地址和 SNA 地址是不同的。首先需要在 IP 和 SNA 地址之间有一个双向映射表。而且,可编址的概念也是不同的。在 IP 中,主机(实际上是接口卡)有地址,而在 SNA 中,除了主机之外的实体(比如硬件设备)也可以有地址。最好的话,我们必须维护一个完全的数据库,它尽可能地将所有的映射关系都建立起来,但是,维护这样的数据库常常是许多问题的根源。

另一种思路是设计一个全局通用的“互联网”分组,并且所有的路由器都能够识别这种分组。实际上,这正是 IP 网络的做法,也就是说,IP 分组在传输过程中可以通过很多个网络。当然,事实可能是这样的,IPv4(当前的 Internet 协议)将所有其他的格式都逐出了市场,而 IPv6(将来的 Internet 协议)还没有流行起来,其他新的格式还没有发明出来,但是,历史可能会选择其他方式。当商业公司意识到有一种自己控制的私有格式可以促进自己的商业利益的时候,要想让所有的公司都遵循同一种格式将是非常困难的。

现在我们再次简短地描述一下两种网络互连的方法。串联虚电路模型本质上与在单个子网内使用虚电路有同样的优点:可以提前预留缓冲区、分组顺序可以得到保证、可以使用较短的头部,并且也可以避免那些由于迟到的重复分组而引起的麻烦。

它也有一些缺点:要求路由器为每个打开的连接使用一定的表格空间、无法使用其他的路由路径来避免拥塞的区域,容易受到沿途路由器失败的影响。而且,如果沿途涉及到的网络中有一个不可靠的数据报网络,那么,串联虚电路模型还有另一个缺点,即难以实现(但不是不可能)串联虚电路。

用数据报模型实现网络互连的特性与数据报子网的特性非常一致:有可能会拥塞,但是,也更能适应拥塞;而对路由器失败的时候有更好的健壮性;需要更长的头部。各种自适应路由算法也可能被用于互连网络中,就如同在单个数据报网络中一样。

用数据报模型实现网络互连最大的优点是,它可以包含那些内部未使用虚电路的子

网。许多 LAN、移动网络(例如飞机和海军舰队)、甚至有些 WAN 都属于这一类网络。当一个互连网络包含了至少一个这类网络的时候,如果网络互连策略是建立在虚电路基础上的话,则会出现严重的问题。

### 5.5.5 隧道技术

为了将两个不同的网络相互连接起来,要提出一种通用的处理办法是非常困难的。然而,在实践中却存在一种常见的、易于处理的特殊情形。这种情形是,源和目标主机位于相同类型的网络中,但是,它们中间却存在不同类型的网络。举例来说,请考虑一家跨国银行,它在巴黎有一个基于 TCP/IP 的以太网,在伦敦也有一个基于 TCP/IP 的以太网,但是在巴黎和伦敦之间有一个非 IP 的广域网(比如 ATM),图 5.47 显示了这样的情形。

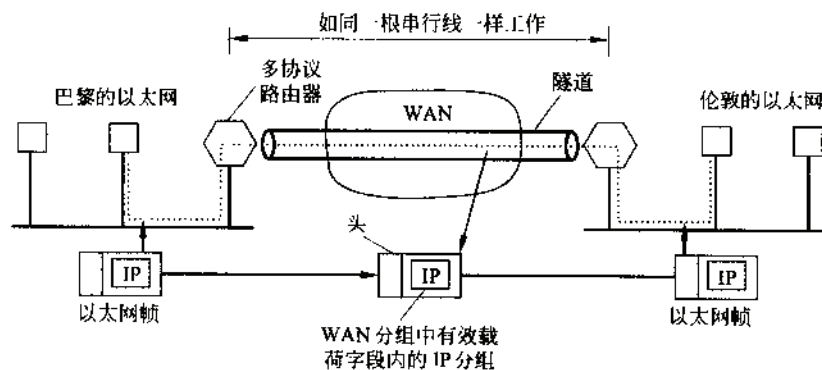


图 5.47 从巴黎到伦敦以隧道方式传递一个分组

这个问题的解决方案是一种称为隧道(tunneling)的技术。为了给主机 2 发送一个 IP 分组,主机 1 在构造分组的时候将主机 2 的 IP 地址包含进去,然后将它插入到一个以太网帧中,该以太网帧的地址指向巴黎的多协议路由器,最后主机 1 把该帧送到以太网上。当多协议路由器得到了该帧的时候,它提取出 IP 分组,并将它插入到 WAN 网络层分组的净荷域中,并且将 WAN 网络层分组的目标地址指向伦敦的多协议路由器的 WAN 地址。当这个 WAN 网络层分组到达伦敦的多协议路由器的时候,伦敦的多协议路由器将 IP 分组提取出来,然后将它放在一个以太网帧中并发送给主机 2。

这里的 WAN 可以看成是一个大的隧道,它从一个多协议路由器延伸到另一个多协议路由器。IP 分组只是从隧道的一端被传送到另一端,中间没有任何阻隔。它根本不用关心中间还要涉及到 WAN。两端以太网上的主机也不需要关心这些事情。只有多协议路由器必须要理解 IP 和 WAN 分组。实际上,从一台多协议路由器到另一台多协议路由器之间的整段距离就好像是一条串行线一样。

用一个类比可能会使隧道的概念更加清晰。请考虑有一个人驾着一辆汽车从巴黎出发要去伦敦。在法国境内,该汽车可以依靠自己的马力向行驶,但是当它到了英吉利海峡的时候,它被装入到高速列车中,经过海底铁路隧道到达英国(汽车不允许直接在隧道

中行驶)。实际上,这里的汽车被当作货物一样运到另一端,如图 5.48 所示。到了隧道另一端,汽车被运到英国的公路上,它又可以依靠自己的马力向前行驶了。分组在通过外部网络时所使用的隧道技术也是以同样的方式工作的。



图 5.48 利用隧道将一辆汽车从法国运到英国

#### 5.5.6 互联网路由

经过一个互联网的路由过程非常类似于在单个子网内部的路由,但是,前者比后者更加复杂一些。例如,请考虑图 5.49(a)中的互联网,其中 5 个网络通过 6 台路由器(可能是多协议路由器)连接在一起。在这种情况下,用图(graph)的模型来表示将会非常复杂,因为每台路由器都可以直接访问(即可以发送分组给)另一台跟它一样也连接到同一网络的路由器。例如,图 5.49(a)中的 B 通过网络 2 可以直接访问 A 和 C,通过网络 3 也可以直接访问 D。这样就导致了图 5.49(b)中的图表示。

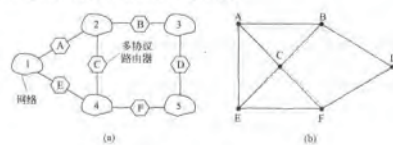


图 5.49  
(a) 一个互联网；(b) 互联网的图表示

一旦图已经被构造出来了,我们就可以在这一组多协议路由器上使用已知的路由算法,比如距离向量和链路状态算法。这实际上提供了两级路由算法:在每个网络内部使用一个内部网关协议(interior gateway protocol),但是在网络之间则使用一个外部网关协议(exterior gateway protocol)(这里的术语“网关”是“路由器”的老式叫法)。事实上,由于每个网络都是独立的,所以,它们可以使用完全不同的路由算法。因为一个互联网内部的每个网络都独立于所有其他的网络,所以每个网络通常称作一个自治系统(AS, Autonomous System)。

一个典型的互联网分组从它所在的 LAN 出发,被传送给本地的多协议路由器(地址在 MAC 层头部)。当它到了多协议路由器之后,网络层代码利用其自己的路由表,决定

将该分组转发到哪一个多协议路由器。如果利用该分组本身的网络协议就可以到达此路由器的话,那么,该分组被直接转发出去。否则的话,该分组被封装到中间网络所要求的协议中,然后经过隧道被传送出去。这个过程不断重复,直到该分组到达目标网络。

互联网路由与网络内部路由之间的一个区别是,互联网路由可能要求跨越国际边界。于是各种各样的法律就会介入进来,比如,瑞典严格的隐私法限制从瑞典输出有关其公民的个人数据。另一个例子是,加拿大的法律规定,凡是源端和目标端都在加拿大的数据流量不允许离开加拿大。这条法律意味着,从安大略湖的温索尔(Windsor, Ontario)发送到范库弗峰(Vancouver)的流量不允许经过附近的底特律(Detroit),尽管经过底特律的路径更短,也更便宜。

内部路由和外部路由之间的另一个区别是代价。在单个网络内部,通常只使用一种收费算法。然而,不同的网络可能属于不同的管理部门,而且,有可能一条路由路径比另一条路由路径更加便宜。类似地,不同网络提供的服务质量可能有所不同,这也可能是选择路由路径的因素之一。

### 5.5.7 分段

每个网络都会限制其分组的最大长度。这种限制有多方面的原因,其中包括:

- (1) 硬件(比如以太网帧的长度限制)。
- (2) 操作系统(比如所有的缓冲区都是 512 字节)。
- (3) 协议(比如,分组长度域中的位数)。
- (4) 遵从某一个国家(或国际)标准。
- (5) 期望将“因错误而引入的重传次数”减少到某种程度。
- (6) 期望防止分组占用信道时间太长。

所有这些因素导致的结果是,网络设计者们无法自由地选择他们所期望的最大分组长度。最大净荷长度的范围从 48 字节(ATM 信元)到 65 535 字节(IP 分组)不等,但是,网络层之上其他层的净荷长度通常可以更大一些。

很显然,当一个较大的分组想要通过一个最大分组长度较小的网络的时候,问题就随之而来了。一种方案是从一开始就保证不会发生这样的问题。换句话说,互联网所使用的路由算法应该可以避免将分组转发给那些不能对它们进行处理的网络。然而,这种方案根本不是一种解决方案。如果原始的源分组太大了,以至于目标网络无法处理这么大的分组,那该怎么办呢?路由算法总不可能绕过目标网络吧。

从根本上讲,这个问题的惟一解决方案是让网关把分组分割成段(fragment),然后发送每一个段,就好像发送独立的互联网分组一样。然而,正如每个孩童的父母们所清楚的那样,把一个大的物体变成小的碎片要比其逆过程容易得多(物理学家将这种效应称为热力学第二定律)。分组交换网络也有同样的“把小的分段复原为大的分组”的问题。

为了将分段重新组成原始的分组,可以考虑两种相互对立的策略。第一种策略是,由“小分组”网络引起的分段过程对于沿途后续的网络都变成透明,也就是说,从该网络一直到最终的目标途中的每个网络都感觉不到曾经发生过分段,如图 5.50(a)所示。在这种方法中,小分组网络有一些通向其他网络的接口网关(很可能是一些特殊的路由器)。当



一个超大的分组到达某一个网关的时候,该网关将它分割成多个分段,每个分段的地址都指向同样的出口网关,在出口网关处这些分段又被重新组合起来。按照这种方法,任何一个分组通过这样的小分组网络都是透明的。后续的网络根本感觉不到曾经发生过分段。例如,ATM 网络有专门的硬件来提供透明的分段过程,它可以将分组分割成信元,然后将信元重组成分组。在 ATM 领域中,这个分段过程(fragmentation)被称为分割(segmentation,或者在中文中也称为分段);其实概念是相同的,只不过有些细节不同而已。

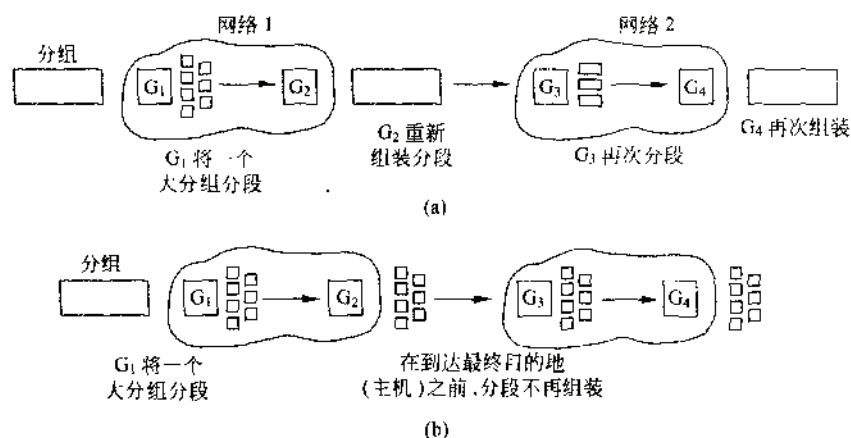


图 5.50

(a) 透明的分段过程; (b) 不透明的分段过程

透明的分段过程非常直接、简单,但是也有一些问题。首先,出口网关必须知道什么时候它已经接收到所有的分片了,所以,每个分段中必须提供一个计数域或者一个“分组末尾”位。其次,所有的分组必须经过同一个网关发送出去。由于不允许有些分段沿着一条路径到达最终目标,而另一些分段沿着另一条路径到达最终目标,所以,可能会损失一点性能。最后一个问题是,在通过一系列小分组网络的时候,由于不断地对大的分组进行分段和重组而带来的额外开销会很大。ATM 要求使用透明的分段策略。

另一种分段策略是,在任何一个中间网关上都不用执行分段重组的操作。一旦一个分组已经被分段了,则每个分段都被当作原始的分组一样来对待。所有的分段都通过出口网关(或者多个出口网关),如图 5.50(b)所示。重组过程只在目标主机上进行。IP 网络采用了这种分段策略。

不透明的分段策略也有一些问题。例如,它要求每台主机都具备重组分段的能力。另一个问题是,当大的分组被分段以后,总的开销相应地增加了,因为每个分段必须有一个头。在第一种方法中,当一个分组退出小分组网络之后,这种额外的开销就消失了;而在这种方法中,这样的开销一直持续到走完后面的旅程为止。然而,不透明分组策略的一个优点是,现在可以使用多个出口网关,从而可能获得更高的性能。当然,如果使用了串联的虚电路模型的话,那么这种优点也就不复存在了。

当一个分组被分段的时候,我们必须对这些分段进行编号,以便可以重构原始的数据

流。一种编号的方法是使用一棵树。如果分组 0 必须要被分段的话,那么,每一个分段称为 0.0、0.1、0.2,等等。如果后来,这些分段本身又要被分段,那么,新的分段编号为 0.0.0、0.0.1、0.0.2、...、0.1.0、0.1.1、0.1.2,等等。只要在头部中为最差的情形保留了足够多的域,并且其他地方也没有生成重复的编号,那么,这种方案是可以保证,不管分片的到达顺序如何,在目标端总能够将所有的分片正确地重组起来。

然而,如果有一个网络丢失或者丢弃了部分分组,则端到端的重传是必要的,这对于以上的编号系统来说是非常不幸的。假设一个 1024 位的分组初始的时候被分成 4 个大小相等的分段:0.0、0.1、0.2 和 0.3。其中分段 0.1 在途中丢失了,但是其他三个分段都到达了目标主机。最终,源主机超时,于是重传原来的分组。只不过这一次墨菲法则 (Murphy's law) 没有起作用,路由路径通过一个长度限制为 512 位的网络,所以,这次生成两个分段。当新的分段 0.1 到达目标端的时候,接收方认为 4 个分段都到了,于是不正确地重构分组。

另一个完全不同(但是更好)的编号系统是,让互联网协议定义一个足够小的基本分段长度值,以便基本的分段能够通过每一个网络。当一个分组被分段的时候,除了最后一个分段以外其他所有的分段都等于基本分段长度,而最后一个分段只会更短一些。出于效率的原因,一个互联网分组可能包含多个分段。互联网头部必须提供原始的分组号,以及该分组中包含的(第一个)基本分段的编号。按照惯例,在互联网头部还必须有一位来指明“该互联网分组中包含的最后一个基本分段是否为原始分组的最后一个分段”。

这种方法要求在互联网头部有两个序列号域:原始的分组号和分段号。很显然,在基本分段长度和分段号的位数之间需要进行权衡。因为基本分段长度假定能够被每一个网络所接受,所以,包含多个分段的互联网分组的后续分段过程不会出现问题。这里最终的限制是,基本分段的长度为 1 位或者 1 个字节,而分段号则是指当前分段在原始分组中的位偏移或者字节偏移,如图 5.51 所示。

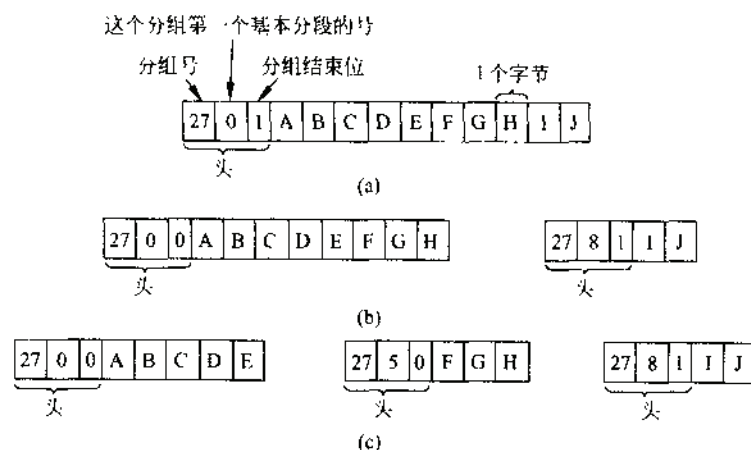


图 5.51 当基本数据长度为 1 个字节时的分段情况

(a) 原始分组,包含 10 个数据字节;(b) 当通过一个最大分组长度为“8 个净荷字节+头部”的网络之后的分段;

(c) 当通过一个最大分组长度为 5 的网关之后的分段

有些互联网协议进一步扩展了这种方法,它们将虚电路上的整个传输数据流当作一个巨大的分组,所以,每个分段包含了该分段中第一个字节的绝对字节编号。

## 5.6 Internet 上的网络层

在讨论 Internet 网络层的特殊性之前,我们有必要先看一看那些“最初驱动了 Internet 的设计、并且使得 Internet 今日如此之成功”的原则。如今,人们好像已经忘记了这些原则。RFC 1958 中列出了这些原则,并且对它们进行了讨论。这份文档值得好好读一读(对于所有的协议设计者这应该是必修的环节,而且最后应该要通过考试)。这份 RFC 文档着重描述了在(Clark, 1988; and Saltzer et al., 1984)中提到的一些思想。现在概要地列出我们这里所认为的前 10 条原则(按照重要性程度从最重要到不重要的顺序排列)。

(1) **保证它能够工作。**直到有了多个原型系统可以成功地相互通信之后,才可以最终确定设计或者确定标准。现在的设计者往往先编写出一份 1000 页的标准,并申请批准之后,才发现标准中有严重的缺陷,而且它根本不能工作。然后他们再编写 1.1 版本的标准。这不是正确的工作方式。

(2) **尽可能使它简单。**毫无疑问,任何时候都应该使用最简单的方案。奥卡姆的威廉(William of Occam)在 14 世纪的时候就已经提出这条原则了(称为奥卡姆的剃刀,“Occam's razor”)。换成现代的术语是:决斗特性(fight feature)。如果一项特性并非绝对本质的特性,那么就不考虑该特性。尤其是,如果通过组合其他的特性也能够获得同样效果的情况下。

(3) **作出明确的选择。**如果有几种方法可以完成同样的事情,则选择其中一种方法。用两种或者多种方法来做同样的事情简直是在自找麻烦。通常标准会有多个选项,或者有多种模式,或者有一些参数,因为多个实力强大的参与方坚持认为他们的方法是最好的。设计者应该坚决抵制这种倾向,要学会说“不”。

(4) **尽可能做到模块化。**这条原则直接导致了协议栈的思想,每一层上的协议独立于所有其他的协议。按照这种方法,如果实际环境中要求改变一个模块或者一层,则其他的模块或者层不会受到影响。

(5) **期望具备异构性。**在任何一个大型的网络中,不同类型的硬件、传输设施和应用都有可能存在。为了能够对它们进行处理,网络的设计必须简单、通用和灵活。

(6) **避免使用固定不变的选择和参数。**如果不可避免要使用参数的话(比如最大的分组长度),那么,最好的办法是让发送方和接收方协商一个值,而不是定义固定的参数值。

(7) **寻找一个好的设计,它不必是最完美的。**通常设计者有一个好的设计,但是它不能够处理一些怪异的特例。设计者不应该对该设计进行大幅修改,而应该坚持这个好的设计,并且将支持怪异特例的负担转移到那些对此有特殊需求的人身上。

(8) **对于发送操作一定要严格,而对于接收操作要有一定的容忍度。**换句话说,只发送那些严格符合标准的分组,但是,容许接收到的分组可能是不完全符合标准的,并且要

试图对它们进行处理。

(9) **要考虑伸缩性。**如果一个系统需要有效地处理上百万台主机和几十亿用户,那么,存在任何一个中心化的数据库都是难以容忍的,也就是说,不能使用中心化的数据库;同时,必须将负载尽可能均匀地分布到所有可供利用的资源上。

(10) **要考虑性能和代价。**如果一个网络的性能很差,或者代价特别高,那么没有人会使用这样的网络。

现在我们放下这些一般原则,开始讨论 Internet 网络层的细节。在网络层上,我们可以将整个 Internet 看作是一组相互连接的子网络(subnetwork)或者自治系统(ASes, Autonomous Systems)的集合。Internet 的网络层并没有实际的结构,只有一些大的骨干网络。这些骨干网络是由高带宽的线路和快速路由器构成的,连接在骨干网络上的是—些区域(中等规模)网络,连接在这些区域网络上的是许多大学、公司和 Internet 服务供应商的 LAN。图 5.52 给出了半层次结构的组织示意图。

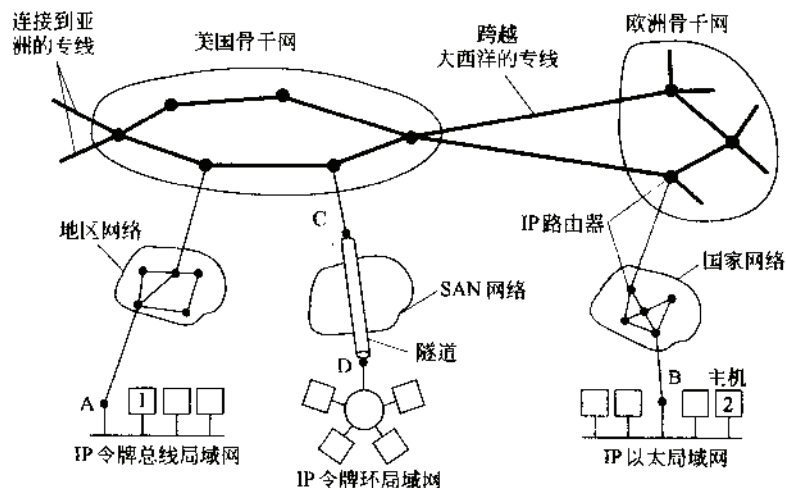


图 5.52 Internet 是由许多网络相互连接之后构成的集合

将整个 Internet 粘合在一起的正是网络层协议: IP(Internet Protocol)。与大多数老式的网络层协议不同的是,IP 协议从设计之初就考虑到了网络互联的需求。一种比较好的看待网络层的做法就应该像这样。IP 的任务是提供一种尽力投递的(best-efforts,即不提供任何保证)方法将数据报从源端传输到目标端,它并不关心源机器和目标机器是否在同样的网络中,也不关心它们之间是否还有其他的网络。

Internet 中的通信过程如下所述。传输层接受数据流,并且将数据流分装到数据报中。理论上,每个数据报最多可容纳 64KB,但是实际上,数据报通常不超过 1500 个字节(因而它们正好可被放到一个以太网帧中)。每个数据报被传输到 Internet 上,在途中它们有可能被分成更小的单元。当所有这些分片最终到达目标机器的时候,它们又被网络层重新组装起来,恢复成原来的数据报。然后,该数据报被递交给传输层,传输层将它插入到接收进程的输入流中。正如从图 5.52 中可以看出,主机 1 发送的一个分组必须要经

过 6 个网络才能到达主机 2。在实践中,通常沿途经过的网络数不止 6 个。

### 5.6.1 IP 协议

学习 Internet 网络层最恰当的开始之处是 IP 数据报本身的格式。每个 IP 数据报包含一个头部和一个正文部分。头部有一个 20 字节的定长部分和一个可选的变长部分。图 5.53 显示了 IP 数据报的头部格式。IP 数据报头部的传输采用了 big-endian 字节序:从左到右,Version 域的高序字节最先被传送出去。(SPARC 是 big-endian 字节序的;而 Pentium 则是 little-endian 字节序的)。在 little-endian 字节序的机器上,无论是发送还是接收都需要进行软件转换。

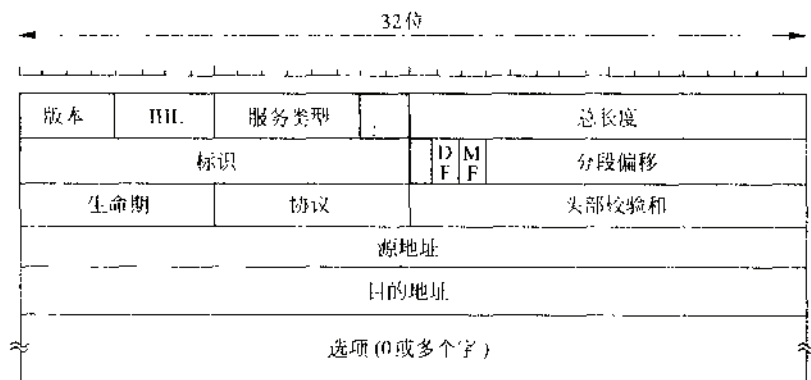


图 5.53 IPv4(Internet 协议)头部

版本(Version)域记录了数据报属于哪一个版本的协议。由于每个数据报中包含了版本信息,所以版本之间的迁移过程就有可能持续很多年,在此过程中,有的机器运行老的版本,而其他的机器运行新的版本。目前正在进行从 IPv4 到 IPv6 的版本迁移过程,虽然已经进行多年了,但是还没有结束的迹象(Durand, 2001; Wiljakka, 2002; and Waddington and Chang, 2002)。甚至有的人认为这个迁移过程永远也不会结束(Weiser, 2001)。关于版本编号的问题,这里也顺便提一下,IPv5 是一个试验性的实时流协议,它一直没有被广泛应用。

由于头部的长度不是固定的,所以头部的 IHL 域指明了该头部有多长(以 32 位字的长度为单位)。IHL 的最小值为 5,这表明头部没有可选项。此 4 位域的最大值为 15,这限制了头部的长度最大为 60 字节,因此选项(即 Options 域)最多为 40 字节。对于某些选项,比如记录一个分组沿途路径的选项,40 字节往往太小了,这使得这样的选项其实没有什么用处。

服务类型(Type of service)域是少数几个“在过去几年中其含义略有改变”的域之一。无论是改变之前还是改变之后,它的用途是区分不同的服务种类。可靠性和速度的各种组合都是可能的。对于数字化的语音数据,速度比精确性更为重要;对于文件传输,正确性比速度更加重要。

最初的时候,此 6 位域(从左至右)包含了一个 3 位的 Precedence 域和三个标志 D、T



和 R。Precedence 域是一个从 0(普通级别)至 7(网络控制分组)的优先级。通过三个标志位,源主机可以指定它最关心的是什么: {延迟(Delay)、吞吐量(Throughput)、可靠性(Reliability)}。理论上,这些域使得路由器可以在多种选择之中作出路由决定,比如,路由器利用这些域选择一条高吞吐量、高延迟的卫星线路,或者选择一条低吞吐量、低延迟的租用线路。而在实践中,当前的路由器通常完全忽略服务类型域。

最后,IETF 承认了失败,并且将该域作了轻微的改变,以适应区分服务的需要。这 6 位合起来表示每个分组属于以前讨论过的服务类别中的哪一类。这些类别包括 4 个排队优先级、3 种丢弃可能性和一些历史类别。

总长度(Total Length)域包含了该数据报中的所有内容,即头和数据。最大的长度是 65 535 字节。目前情况下,这样的上界还是可以容忍的,但是,将来有了千兆网络之后,可能会需要更大的数据报。

标识(Identification)域的用途是让目标主机确定一个新到达的分段属于哪一个数据报。同一个数据报的所有分段包含同样的 Identification 值。

接下来是一个未使用的位,然后是两个 1 位域。DF 代表“不分段(Don't Fragment)”。这是针对路由器的一条命令,它让路由器不要分割该数据报,因为目标主机无法将分片重组回原来的数据报。例如,当一台计算机启动的时候,它的 ROM 可能向网络请求给它发送一个包含内存映像的单个数据报。发送方在数据报中标记了 DF 位之后,它就知道该数据报将作为一个整体到达接收方,不过这意味着该数据报必须避开最优路径上的小分组网络,而不得走次优的路径。所有的机器都要求能接受 576 字节或者更少的分段。

MF 代表“更多的分段(More Fragments)”。除了最后一个分段以外其他所有的分段必须设置这一位。它的用途是,接收方可以知道什么时候一个数据报的所有分段都已经到达了。

分段偏移(Fragment offset)域指明了该分段在当前数据报中的什么位置上。除了一个数据报的最后一个分段以外,其他所有的分段必须是 8 字节的倍数,这里 8 字节是基本分段单位。由于该域有 13 位,所以,每个数据报最多有 8192 个分段。因此,最大的数据报长度为 65,536 字节,比 Total Length 域还要大 1。

TTL(Time to live)域是一个用于限制分组生存期的计数器。这里的计数时间单位为秒,因此最大的生存期为 255 秒。在每一跳上该计数器必须被递减,而且,当数据报在一台路由器上排队时间较长时,该计数器必须被多倍递减。在实践中,它只是跳计数器,当它递减到 0 的时候,分组被丢弃,路由器给源主机送回一个警告分组。此项特性可以避免数据报长时间地逗留在网络中,有时候当路由表被破坏之后,这种事情是有可能发生的。

当网络层组装完成一个完整的数据报之后,它需要知道该如何对它进行处理。协议(Protocol)域指明了该将它交给哪个传输进程。TCP 是一种可能,但是 UDP 或者其他的协议也是可能的。协议的编号是整个 Internet 全球统一的。RFC 1700 中列出了以前的协议和其他分配的编号,但是现在,协议的编号位于一个在线数据库中(从 [www.iana.org](http://www.iana.org) 中可以查到)。

头部校验和(Header checksum)域只校验头部。这样的校验和对于检测“因路由器中的坏内存而产生的错误”非常有用。其算法是这样的:当数据到达时,所有的 16 位(半字)累加起来,然后再取结果的补码。该算法的意图是,当数据到达之后,Header checksum 的计算结果应该为 0。该算法比常规的加法更加稳定。请注意,在每一跳上,Header checksum 域必须重新计算,因为至少有一个域总是要改变的(即 Time to live 域),但是,通过一些技巧可以加速计算。

源地址(Source address)域和目标地址(Destination address)域表示网络号和主机号。我们将在下一小节介绍 Internet 地址。选项(Options)域的设计意图是:提供一种途径允许后续版本的协议包含一些原来的设计中没有出现的信息;允许实验人员试验新的想法;避免为那些不常使用的信息分配头部域。选项是变长的,每个选项的第一个字节是一个标识码,它标明了该选项。有的选项后面跟着一个 1 字节的选项长度域,然后是一个或多个数据字节。Options 域被补齐到 4 字节的倍数。最初的时候定义了 5 个选项,如图 5.54 所列,但是,后来又加入了一些新的选项。现在,完整的选项列表可以从这里查到: [www.iana.org/assignments/ip-parameters](http://www.iana.org/assignments/ip-parameters)。

选 项	说 明
Security	规定了数据报的秘密程度
Strict source routing	给出了必须要遵循的完整路径
Loose source routing	给出了一组路由器,路由过程中不能漏掉这些路由器
Record route	让每台路由器都附上它的 IP 地址
Timestamp	让每台路由器都附上它的地址和时间戳

图 5.54 一些 IP 选项

安全(Security)选项指明了信息的秘密程度。理论上,军用路由器可能使用这个域来指定在路由的时候不允许通过某些在军事上被认为是“敌对分子”的国家。而在实践中,所有的路由器都忽略该选项,所以,它惟一实际的用途是帮助间谍们更加容易找到好的材料。

严格的源路由(Strict source routing)选项给出了从源到目标的完整路径,其形式是一系列 IP 地址。数据报必须严格地沿着这条路径向前传输。对于系统管理员,这是非常有用的,他们可以在路由表被破坏的时候发送紧急分组,或者用这个选项来测量时间。

宽松的源路由(Loose source routing)选项要求该分组穿越所指定的路由器列表,并且要求按照列表中的顺序前进,但是,在途中也允许经过其他的路由器。通常情况下,该选项往往只提供少数路由器,以强迫走一条特殊的路径。例如,为了强迫一个从伦敦到悉尼的分组必须先向西而不是向东,该选项可以指定纽约(New York)、洛杉矶(Los Angeles)和檀香山(Honolulu)的路由器。当出于政治或者经济的考虑而要求经过或者避免某些国家的时候,这个选项特别有用。

记录路径(Record route)选项告诉沿途的路由器,将它们的 IP 地址附到该选项域中。这使得系统管理员可以跟踪路由算法中的错误(比如,“为什么从休斯顿[Houston]到达

拉斯[Dallas]的分组要首先经过东京[Tokyo]?”)。当 ARPANET 刚开始建立起来的时候,没有一个分组会经过 9 台以上的路由器,所以,40 字节的选项在当时是足够的。正如前面所提到的,现在 40 字节显得太小了。

最后,时间戳(Timestamp)选项与记录路由选项类似,只不过每台路由器除了记录 32 位 IP 地址以外,还要记录一个 32 位时间戳。这个选项主要也被用于调试路由算法。

### 5.6.2 IP 地址

Internet 上的每台主机和路由器都有一个 IP 地址,IP 地址包含网络号和主机号。并且,这种组合是惟一的:原则上,Internet 上的任何两台机器不会有相同的 IP 地址。所有的 IP 地址都是 32 位长,被用于 IP 分组的 Source address 和 Destination address 域。很重要的一点是,实际上 IP 地址引用的并不是一台主机,相反,它真正引用的是一个网络接口,所以,如果一台主机同时位于两个网络上,那么它必须拥有两个 IP 地址。然而,在实践中,大多数主机都在一个网络上,所以只有一个 IP 地址。

过去几十年来,IP 地址被分成了 5 大类,如图 5.55 所示。这种分配方案称为分类的编址方案(classful addressing)。现在这种方案已经不再使用了,但是,在文献中仍然经常引用这种方案。稍后我们将讨论另一种替代方案。

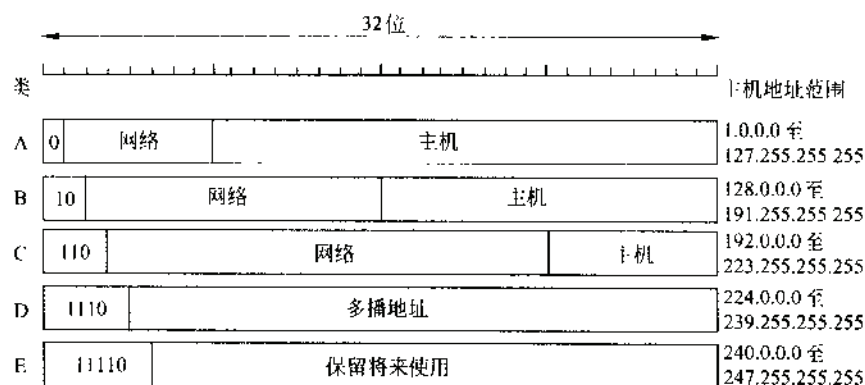


图 5.55 IP 地址的格式

A、B、C 和 D 类地址的格式分别允许多达 128 个网络,每个网络 1600 万台主机;或者 16 384 个网络,每个网络 6 万 4 千台主机;或者 2 百万个网络(比如 LAN),每个网络多达 256 台主机(不过有些地址是特殊的)。IP 地址分类中也支持多播,即数据报被直接发送给多台主机。以 1111 开头的地址是保留地址,以备将来使用。现在连接到 Internet 的网络超过了 500 000 多个,而且这个数字还在逐年递增。网络号是由一个非营利性的机构来管理的,以避免冲突,该机构的名称为 ICANN(Internet Corporation for Assigned Names and Numbers)。同时,ICANN 把部分地址空间委托给各种区域性的权威机构,然后这些权威机构又将 IP 地址分配给 ISP 和其他的公司。

32 位数值的网络地址通常写作点分十进制标记法(dotted decimal notation)。在这种格式中,4 个字节中的每个字节用十进制表示,从 0 到 255。例如,32 位的十六进制地址

C0290614 写成 192.41.6.20。最低的 IP 地址是 0.0.0.0,最高的 IP 地址是 255.255.255.255。

值 0 和 -1(即全 1)有特殊的含义,如图 5.56 所示。值 0 表示当前网络,或者当前主机。值 -1 被用作广播地址,其含义是所指网络中的所有主机。

0 0		本机				
0 0	...	0 0	主机		当前网络上的主机	
1 1					本地网络上的广播	
网络		1 1 1 1		...	1 1 1 1	远程网络上的广播
127		任意值				回环

图 5.56 一些特殊的 IP 地址

当主机刚刚启动的时候,它使用 IP 地址 0.0.0.0。用 0 作为网络号的 IP 地址表示当前的网络。这些地址使得网络内的机器在不知道网络号的情况下就可以引用自己所在的网络(但是,它们必须知道网络的类型以便知道该包含多少个 0)。由全 1 构成的地址允许在本地网络(通常是一个 LAN)上进行广播。如果 IP 地址中的网络号部分指向一个适当的网络,而主机域部分全部为 1,那么,通过这样的地址可以向 Internet 上的任何远程网络发送广播分组(不过,许多网络管理员禁止这种特性)。最后,所有形如 127. xx. yy. zz 的地址都被保留用作回环测试。发送至这类地址的分组都不会被输出到线路上;这些分组直接在本地被处理,并且被看作进入的分组。这使得在发送方不知道本地网络号的情况下就可以发送分组给本地网络。

## 子网

正如我们已经看到的,一个网络中的所有主机必须有相同的网络号。当网络不断增长的时候,IP 编址方案的这种特性可能会有问题。例如,考虑一个大学的网络,刚开始的时候,该大学的计算机科学系使用一个 B 类地址建立起它的以太网。一年以后,电子工程系也想连接到 Internet 上,所以他们购买了一个中继器,将计算机系的以太网络扩展到他们的办公楼。随着时间的推移,其他许多系也购买了计算机,并且想连接到 Internet 上,所以,每个以太网只允许 4 个中继器的限制很快就达到了。为了解决这样的问题,需要使用另一种组织结构。

许多大学已有的主机数量往往超过了 60 000 台,然而,由于网络地址的缺乏,要想获得第二个网络地址是非常困难的。问题的根源在于这一条规则:单个 A、B 或 C 类地址只能引用一个网络,而不是一组 LAN。随着越来越多的组织面临这样的窘境,IP 编址系统有必要稍作修改以便能够适应这种局面。

问题的解决方案是,允许将一个网络分成多个部分供内部使用,但是对于外部世界仍然像单个网络一样。如今,一个典型的校园网络可能如图 5.57 所示,一台主路由器连接到一个 ISP 或者一个区域网络,而大量的以太网遍布在学校的各个系。每个以太网都

有它自己的路由器,该路由器连接到学校的主路由器上(可能通过一个骨干 LAN 连接起来,但是,这里并不关心路由器之间是如何连接的)。

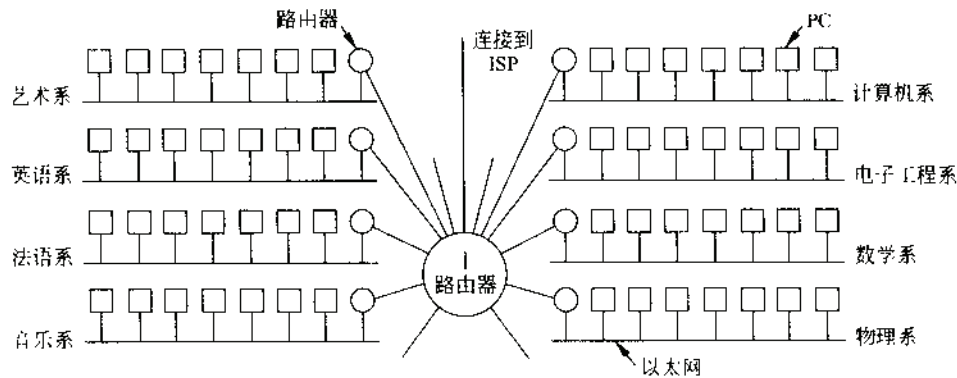


图 5.57 一个典型的校园网络,各个系有自己的 LAN

在有关 Internet 的文献中,把网络的各个部分(在本例中,指各个以太网)称为子网(subnet)。我们在第 1 章中提到的子网是指一个网络中所有路由器和通信线路的集合,所以,这与现在的用法发生了冲突。所幸的是,从上下文中应该很容易区分出“子网”的真正含义。在这一部分和下一部分中,我们将使用新的定义。

当一个分组到达主路由器中的时候,主路由器怎么知道该把它交给哪一个子网(以太网)呢?一种办法是,在主路由器中建立一张表,表中共有 65 536 个表项,它指明了校园内每台主机使用哪一台路由器。这种想法是可行的,但是,它要求在主路由器中建立一张非常大的表,而且,当主机被加入进来,或者移动,或者清理出去的时候,这种做法需要大量的手工维护工作。

相反,为了解决这个问题,人们发明了另一种不同的方案。它的基本思想是,一个 B 类地址不再是 14 位网络号和 16 位主机号,而是从主机号中拿出一些位构成一个子网号。例如,如果该大学有 35 个系,那么,它可以使用 6 位子网号和 10 位主机号,从而最多可以支持 64 个以太网,每个以太网最多可以容纳 1022 台主机(0 和 -1 是保留的,前面已经提到过)。如果事实证明这种分割办法不正确的话,以后还可以再改变。

为了实现对子网的支持,主路由器需要一个子网掩码(subnet mask),它代表了“网络+子网号”与主机号之间的分割方案,如图 5.58 所示。子网掩码也可以用点十进制标记法来表示,外加一个“/”,并跟上“网络+子网”部分的位数。对于图 5.58 中的例子,子网掩码可以写成 255.255.252.0。另一种标记是用“/22”来表示子网掩码有 22 位长。

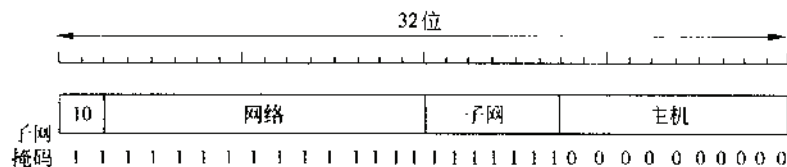


图 5.58 一个 B 类网络被分成 64 个子网



在网络的外部,子网是不可见的,所以,在分配新的子网的时候,不需要跟 ICANN 联系,也不用改变任何外部的数据库。在本例中,第一个子网的 IP 地址可能从 130.50.4.1 开始,第二个子网的 IP 地址可能从 130.50.8.1 开始,第三个子网可能从 130.50.12.1 开始,依此类推。为了看清楚为什么这些子网是按 4 递进的,请注意,对应的二进制地址如下所示:

子网 1: 10000010 00110010 000001|00 00000001

子网 2: 10000010 00110010 000010|00 00000001

子网 3: 10000010 00110010 000011|00 00000001

这里的竖线(|)代表了子网号与主机号之间的边界。竖线的左边是 6 位子网号,右边是 10 位主机号。

为了看清楚这些子网是如何工作的,这里有必要解释一下路由器是如何处理 IP 分组的。每台路由器都有一张表,其中列出了一些形如(网络,0)的 IP 地址和一些形如(当前网络,主机)的 IP 地址。第一种 IP 地址说明了如何到达远程网络;第二种 IP 地址说明了如何到达本地主机。与每个表项相关联的是能够到达该目标的网络接口,以及其他一些信息。

当一个 IP 分组到来的时候,路由器在它的路由表中查找该分组的目标地址。如果该分组的目标在一个远程网络上,那么,它被转发到表中指定的接口上的下一台路由器。如果它的目标是一台本地主机(比如,在路由器的 LAN 上),那么它直接发送给目标主机。如果在路由表中找不到分组的目标网络的话,路由器将此分组转发给一台有更多扩展表的默认路由器。这个算法意味着,每台路由器只需要记录下其他的网络和本地主机就可以了,而不是所有的(网络,主机)对,从而大大减少了路由表的表项。

在引入子网划分以后,路由表也变了,它加入了形如(当前网络,子网,0)和(当前网络,当前子网,主机)的表项。因此,一台位于子网 k 上的路由器知道如何到达所有其他的子网,也知道如何到达子网 k 上的所有主机。它不必知道关于其他子网上的主机的详细情况。实际上,所需要做的改变是让每台路由器做一个布尔与(AND)操作,将分组的目标地址与网络的子网掩码进行“与”操作可以消除主机号,然后在路由表中查找此结果地址(在确定该分组的目标属于哪一类网络之后)。例如,如果分组的目标地址是 130.50.15.6,它到达主路由器的时候,130.50.15.6 与子网掩码 255.255.252.0/22 进行与操作,得到地址 130.50.12.0。主路由器在路由表中查找该地址,以确定通过哪条输出线路可以到达子网 3 的路由器。因此,划分子网的技术实际上建立了一个由网络、子网和主机构成的三级层次结构,从而降低了路由器的表空间。

#### CIDR-Classless InterDomain Routing(无类别域间路由)

IP 已经广泛使用几十年了,它一直工作得非常好,Internet 的指数级增长也正说明了这一点。然而,不幸的是,它自己却正在变成这种过度流行的牺牲品:它的地址快要耗尽了。这一迫在眉睫的危机引起了 Internet 社团中大量的讨论和争辩,到底该怎么办呢?在这一部分中,我们将讨论一下这个问题,并且介绍几种已经被提出来的方案。

回到 1987 年的时候,少数幻想家曾经预言,有一天 Internet 可能会增长到 100 000 个

网络。大多数专家对此不屑一顾,认为即使发生的话也是几十年后的事情了。然而,1996年第100 000个网络连接到了Internet上。正如上面所提到的,问题是Internet很快就要用完IP地址了。原则上,总地址数量超过了20亿个,但实际上,由于地址空间被分成了类(见图5.55),所以有数百万个地址被浪费了。尤其是B类网络,对于大多数组织而言,一个A类网络有16M个地址太大了,而一个C类网络只有256个地址又太小了。所以,一个B类网络有65 536个地址还比较合适。在Internet民间传说中,这种情况称为三只熊问题(就如“Goldilocks and the Three bears”中一样)。

在现实中,一个B类地址对于大多数组织来说又太大了。研究表明,在所有的B类网络中,有一多半网络中的主机不超过50台。其实一个C类地址就够他们用了,但是,毫无疑问,每个申请B类地址的组织都认为有一天他们的主机数量会超过8位域空间。现在回头来看,可能当初应该让C类网络使用10位而不是8位作为主机号,那样每个网络就可以有1022台主机了。如果是这样的设计方案,那么大多数组织可能就会选择C类网络,这样就会有50多万个C类网络(相比之下,只有16 384个B类网络)。

不过,这也很难责怪Internet的设计者没有提供更多(更小)的B类地址。在决定要创建这三类地址的时候,Internet还只是一个研究性的网络,它将美国的主要研究性大学连接起来(加上少量的公司和从事网络研究工作的军方站点)。当时没有人意识到Internet会变成一个具有大量市场的、敢于跟电话网络抗衡的通信系统。那时候,肯定有人会这样说“美国大约有2000所学院和大学,即使它们全部连接到Internet上,甚至其他国家的许多大学也连接上来,总数也不会达到16 000,因为全世界没有这么多大学。而且,让主机号的位数正好是字节长度的倍数也可以加速分组的处理过程。”

然而,如果在分类方案中分配20位给B类网络号的话,则另一个问题又出现了:路由表的膨胀。从路由器的角度来看,IP地址空间是一个两级层次结构,分别是网络号和主机号。路由器并不知道关于所有主机的情况,但是它们必须知道关于所有网络的情况。如果50多万个C类网络都在使用的话,那么,整个Internet上的所有路由器都需要维护一张包含50多万个表项的路由表,每个网络对应一个表项,此表项指明了通过哪条线路可以到达该网络,同时路由表中还提供其他一些信息。

50多万个表项的实际物理存储也许是可行的,但是,对于关键的路由器,将这些表保存在I/O板的静态RAM中则是非常昂贵的。一个更加严重的问题是,与表格的管理有关的各种算法的复杂性将超过线性的增长。更糟的是,大多数已有的路由器软件和固件是在Internet上只有1000个互连的网络的时候设计的,在当时,10 000个网络看起来还很遥远。当时做出的设计决策对于现在而言,远不是最优的情形了。

而且,各种路由算法要求每台路由器定期地传送它的路由表(比如,距离矢量协议)。表格越大,在传送过程中丢失部分表格的可能性越大,从而导致另一端的数据不完整,并可能进一步导致路由不稳定。

通过增加IP地址的分级层次,路由表问题可望得到解决。例如,让每个IP地址包含国家、州/省、城市、网络 and 主机域,然后,每台路由器只需要知道如何到达每个国家、如何到达它自己国家中的各个州或者省、如何到达它所在州或者省的各个城市,以及如何到达

它所在城市的其他网络即可。不幸的是,这种方案要求比 32 位多得多的 IP 地址,而且使用地址的效率也不高(比如,Liechtenstein(列支敦士登,中欧的一个国家)将拥有与美国一样多的地址位数)。

简而言之,有些方案解决了一个问题,但是又引入了新的问题。一种目前正在使用的、并且给予 Internet 额外喘息空间的方案是 **CIDR(Classless InterDomain Routing,无类别域间路由)**。CIDR 背后的基本思想是,将剩余的 IP 地址以可变大小块的方式进行分配,而不管它们所属的类别。RFC 1519 描述了这种思想。比如说,如果一个站点需要 2000 个地址,那么,它可以获得一个以 2048 作为字节边界的地址块,其中包含 2048 个地址。

放弃了分类规则之后,转发过程将变得更加复杂。在老式的分类系统中,转发过程如下所述。当一个分组到达一台路由器的时候,该分组的目标 IP 地址做一份副本,之后将地址副本右移 28 位,以产生一个 4 位的类别号。然后,通过一个 16 路的选择器,以便将分组送到 A、B、C 和 D(如果支持的话)四个不同的分支,其中有 8 路选择 A 类、4 路选择 B 类、2 路选择 C 类,D 类和 E 类各一路。然后针对每一类的情形用掩码得到 8 位、16 位或者 24 位网络号,再向右对齐到一个 32 位字中。然后分别在 A、B 或者 C 表中查询该网络号,通常对 A 和 B 类网络直接使用索引技术,而对 C 类网络则使用散列技术。一旦找到了对应的表项之后,则可以确定输出线路,然后将分组转发出去。

使用了 CIDR 之后,这个简单的算法就不再有效了。相反,每个路由表项必须进行扩展,除了原来的信息以外,它还需要一个 32 位的掩码。因此,现在只需要一张路由表,它是由所有网络的三元组(IP 地址,子网掩码,输出线路)构成的。当一个分组到来的时候,路由器首先将它的目标 IP 地址提取出来。然后(从概念上)对路由表逐项扫描,用掩码去掩(即“与”)目标地址,再将它与表项中的网络号进行比较,看两者是否匹配。在这个过程中有可能找到多个匹配的表项(这些表项的子网掩码的长度不同),如果出现这样的情况,则使用掩码长度最长的那个表项。因此,如果同时匹配到一个/20 的掩码和一个/24 的掩码,则使用/24 的表项。

针对 CIDR 的路由过程,已经提出了一些复杂的算法以加速地址匹配过程(Ruiz-Sanchez et al., 2001)。商业的路由器使用定制的 VLSI 芯片,将这些算法嵌在硬件中。

为了使转发算法更加易于理解,我们来考虑一个例子。假设从 194.24.0.0 开始的数百万个地址都是可以使用的。剑桥大学需要 2048 个地址,它分配到的地址范围从 194.24.0.0 到 194.24.7.255,掩码为 255.255.248.0。接下来,牛津大学申请 4096 个地址。由于 4096 个地址的块必须位于 4096 的字节边界上,所以,牛津大学申请的地址不可能从 194.24.8.0 开始。相反,他们获得了从 194.24.16.0 至 194.24.31.255 的地址块,掩码为 255.255.240.0。现在,爱丁堡大学申请 1024 个地址,它获得了从 194.24.8.0 至 194.24.11.255 的地址块,掩码为 255.255.252.0。图 5.59 概括了这些地址块的分配情况。

现在全世界所有的路由表都已经更新了这三个大学网络的表项。每个表项包含一个基地址和一个子网掩码。这三个表项(用二进制表示)如下:

大学	首地址	末地址	地址个数	记作
剑桥(Cambridge)	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
爱丁堡(Edinburgh)	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(可分配)	194.24.12.0	194.24.15.255	1024	194.24.12/22
牛津(Oxford)	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

图 5.59 IP 地址的分配示例

地址	掩码
C: 11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000
E: 11000010 00011000 00001000 00000000	11111111 11111111 11111100 00000000
O: 11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000

现在请考虑,当一个目标地址为 194.24.17.4 的分组到达一台路由器的時候会怎么样呢?该目标地址用二进制表示如下所示:

11000010 00011000 00010001 00000100

首先,它与剑桥大学的掩码进行布尔与操作,得到:

11000010 00011000 00010000 00000000

该值与剑桥大学的基地址并不匹配,所以,接下来将原来的目标地址与爱丁堡大学的掩码进行布尔与操作,得到:

11000010 00011000 00010000 00000000

该值与爱丁堡大学的基地址并不匹配,所以,接下来试验牛津大学的掩码,得到:

11000010 00011000 00010000 00000000

这次与牛津大学的基地址完全匹配。如果接下来没有再找到匹配的表项的话,那么,路由器将使用牛津大学的表项,该分组沿着此表项中指定的输出线路被发送出去。

现在,我们从内布拉斯加州奥马哈城(Omaha, Nebraska)的一台路由器的角度来看待这三个大学的网络,该路由器只有 4 条输出线路:明尼阿波利斯(Minneapolis)、纽约(New York)、达拉斯(Dallas)和丹佛(Denver)。当这台路由器上的软件得到了前面三个新的表项之后,它注意到,这三个表项可以合并到同一个**聚集表项(aggregate entry)**中,其二进制地址和子网掩码如下所示:

11000010 00000000 00000000 00000000      11111111 11111111 11100000 00000000

这个表项指示路由器,将所有指向这三个大学网络的分组发送到纽约的输出线路上。通过将三个表项聚集起来,奥马哈路由器的路由表中减少了两个表项。

如果纽约有一条通向伦敦的线路用于传输所有到英国的流量,那么,它也可以使用一个聚集表项。然而,如果它有单独的线路通向伦敦和爱丁堡,那么,它必须使用三个独立的表项。在 Internet 上,聚集表项是一项被广泛使用的技术,它可以减少路由器表的空间。

关于这个例子最后需要说明的是,奥马哈路由器中的聚集表项也会将“目标地址位于一段尚未分配的地址范围”的分组发送到纽约。只要这一段地址还没有被真正分配,这种情况就不会发生,因为这些地址还没有被分配,所以就不会存在这样的分组。然而,如果

后来这一段地址被分配给加利福尼亚州的一家公司,那么,就需要增加一个表项 194.24.12.0/22 来专门处理这样的分组。

### NAT—网络地址转换

IP 地址已经非常缺乏了。一个 ISP 可能有一个/16(即以前的 B 类)的地址空间,因此它可以有 65 534 个主机号。如果它的用户数量超过了这个数,那么它就有问题了。对于那些通过拨号连接 Internet 的家庭用户,解决这个问题的一种办法是动态分配 IP 地址,也就是说,当计算机拨号并登录进来的时候给它分配一个 IP 地址,当会话结束的时候,再把 IP 地址收回。按照这种方法,一个/16 的地址空间可以处理多达 65 534 个活动用户,这对于一个拥有几十万用户的 ISP 来说可能已经非常不错了。当一个会话终止的时候,它的 IP 地址又可以被重新分配给另一个呼叫者。对于一个家庭用户数量不算太多的 ISP 来说,这种策略可以工作得很好,但是,对于那些主要为商业客户提供服务的 ISP 来说,这种策略就无法正常工作了。

问题在于,商业客户总是期望在工作时间保持持续在线的状态。无论是小型的商业客户(比如三个人组成的旅行社),还是大型的集团公司,他们总是有多台计算机,并且通过一个 LAN 将这些计算机连接起来。有些计算机是雇员的 PC 机,其他的计算机可能是 Web 服务器。一般来说,在 LAN 上有一台路由器,该路由器通过一条租用的线路连接到 ISP,以便提供持续的 Internet 连接。这种连接方式意味着每台计算机必须有它自己的 IP 地址,并且全天使用该 IP 地址。实际上,对于一个 ISP 来说,所有商业客户拥有的计算机的总数不能够超过该 ISP 所拥有的 IP 地址的数量。如果 ISP 有一个/16 的地址空间,那么,计算机的总数不得超过 65 534。如果 ISP 有几万个商业用户的话,那么这个限制很快就会被突破。

更加糟糕的是,越来越多的家庭用户开始包租 ADSL 或者通过电视网络连接 Internet。这些服务有两个特点:(1)用户得到一个永久的 IP 地址;(2)不存在连接费用(只有月租费),所以许多 ADSL 用户或者有线电视用户通常长久地登录在网络上。这些技术的发展加剧了 IP 地址短缺的局面。像针对拨号用户那样使用动态分配 IP 地址的做法现在行不通了,因为在任何一个时刻,正在使用的 IP 地址的数量可能是该 ISP 所拥有的地址数量的许多倍。

现在我们进一步考虑更加复杂的情形,许多 ADSL 和有线电视用户家里有两台或者更多台计算机,通常每个家庭成员有一台计算机,他们都希望利用 ISP 分配给他们的唯一的 IP 地址,全天候地保持在线状态。这里的解决方案是,将所有的 PC 通过一个 LAN 连接起来,而且需要在 LAN 中配置一台路由器。从 ISP 的角度来看,这样的家庭用户跟那些具有多台计算机的小型商业用户是完全一样的。欢迎来到 Jones 家庭公司。

IP 地址短缺的问题并不是一个只有在将来某个时候才可能发生的理论问题。现在,此时此地,这个问题就来了。对于整个 Internet 而言,根本的解决方案是迁移到 IPv6,它有 128 位地址。这个迁移过程正在进行,但是进展非常缓慢,可能需要很多年才能完成。因此,有些人认为需要一个短时期内有效的快速修补方案。这种快速修补方案是以 NAT (Network Address Translation,网络地址转换)的形式出现的,RFC 3022 描述了 NAT,下



面我们将简要地介绍一下 NAT。有关更多的信息,请参考(Dutcher, 2001)。

NAT 背后的基本思想是,为每个公司分配一个 IP 地址(或者,最多分配少量的 IP 地址),用于传输 Internet 流量。在公司内部,每台计算机有惟一的 IP 地址,它使用该地址来传输内部流量。然而,当一个分组离开公司的网络,发向 ISP 的时候,它需要执行一个地址转换。为了保证这种方案的可行性,有三段 IP 地址范围已经被声明为私有地址。任何一家公司可以在他们的内部随意地使用这些地址。惟一的规则是,包含这些地址的分组不应该出现在 Internet 上。这三段保留的地址范围为:

10.0.0.0	—10.255.255.255/8	(16 777 216 个主机地址)
172.16.0.0	—172.31.255.255/12	(1 048 576 个主机地址)
192.168.0.0	—192.168.255.255/16	(65 536 个主机地址)

第一段范围提供了 16 777 216 个地址(按照惯例,除掉 0 和 -1),大多数公司即使并不需要这么多地址,通常也会选择这一段地址。

NAT 的操作过程如图 5.60 所示。在公司内部,每台机器都有一个形如 10.x.y.z 的地址。然而,当一个分组离开公司的时候,它首先要通过一个 NAT 盒(NAT box)。此 NAT 盒将内部的 IP 源地址(在图中为 10.0.0.1)转换成该公司所拥有的真实 IP 地址,在本例中为 198.60.42.12。NAT 盒通常与防火墙组合在一起,防火墙提供一层安全性,它控制哪些流量可以进入公司的网络,哪些流量可以离开公司的网络。我们将在第 8 章学习防火墙。另外,将 NAT 盒集成到公司的路由器中也是有可能的。

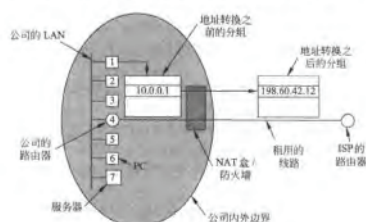


图 5.60 NAT 盒的位置和操作过程

到现在为止,我们一直没有提及一个非常细微的细节:当应答分组回来(比如从一个 Web 服务器回来的应答)的时候,本质上它的目标地址是 198.60.42.12,那么,NAT 盒如何知道该用哪一个地址来替代呢? NAT 解决的问题正在于此。如果在 IP 头部有一个备用的域,那么,NAT 盒就可以用该域来记录下真正的发送方到底是谁,但是,仅仅只有 1 位还是没有用。从原理上来说,我们可以创建一个新的 IP 头选项来存放真实的源地址,但是,这么做将要求改变整个 Internet 上所有机器的 IP 代码,以便这些机器都能够处理新的选项。这显然不是一种有效的快速修补方案。

NAT 的实际做法如下所述。NAT 设计者们注意到,大多数 IP 分组携带的要么是 TCP 净荷,要么是 UDP 净荷。当我们在第 6 章学习 TCP 和 UDP 的时候,我们将会看到,这两者的头部都包含了一个源端口和一个目标端口。下面我们只讨论 TCP 端口,但是,所讨论的内容同样也适用于 UDP 端口。这里的端口是 16 位的整数,它指示了 TCP 连接从哪里开始,以及从哪里结束。正是这些端口域,才使得 NAT 能够工作。

当一个进程希望与另一个远程进程建立 TCP 连接的时候,它绑定到一个本地机器尚未使用的 TCP 端口上。该端口称为源端口(source port),它告诉 TCP 代码,凡是属于该连接的进来分组,都应该发送给这个进程。这个进程也要提供一个目标端口(destination port),以指明分组被送到远程机器上之后应该交给谁。从 0 至 1023 之间的端口都是保留端口,用于一些知名的服务。例如,80 端口被用于 Web 服务器,所以,远程客户可以找到 Web 服务。每个向外发送的 TCP 消息都包含一个源端口和一个目标端口。这两个端口合起来标识出了客户端和服务端正在使用该连接的进程。

打个比方也许能够更清楚地说明端口的用途。假设一家公司只有一个主电话号码。当人们拨打该号码的时候,他们到达接线员那里,接线员问他们要哪个分机,然后为他们接通该分机。这里的主号码就好比是公司的 IP 地址,两端的分机号就好比是端口。端口是编址系统中另一个 16 位的值,它标识了哪个进程接受哪个进来的分组。

利用 source port(源端口)域,我们可以解决前面的映射问题。任何时候当一个向外发送的分组进入到 NAT 盒的时候,源地址 10. x. y. z 被公司的真实 IP 地址所取代,而且, TCP 的 source port 域被一个索引值取代,该索引值指向 NAT 盒的地址转换表中 65 536 个表项之一。该表项包含了原来的 IP 地址和原来的源端口。最后, NAT 盒重新计算 IP 头和 TCP 头的校验和,并将校验和插入到分组中。这里之所以要替换 source port 域,是因为从机器 10. 0. 0. 1 和 10. 0. 0. 2 出发的连接可能碰巧使用了同一个端口,比如都使用了 5000,所以,仅仅使用 source port 还不足以惟一标识发送进程。

当一个分组从 ISP 到达 NAT 盒的时候, NAT 盒从 TCP 头中提取出源端口,用它作为索引值从 NAT 盒的映射表中找到对应的表项,然后从该表项中提取出内部 IP 地址和原来的 TCP source port,并将它们插入到分组中。然后重新计算 IP 和 TCP 校验和,并插入到分组中。最后将该分组传递给公司内部的路由器,它使用 10. x. y. z 地址进行正常的路由。

使用 NAT 也可以缓解 ADSL 和有线电视用户的 IP 地址短缺问题。当 ISP 给每个用户分配地址的时候,它使用 10. x. y. z 地址。当用户机器发送的分组离开 ISP 进入主 Internet 的时候,它们也要通过一个 NAT 盒,由 NAT 盒将这些分组中的源地址转换为 ISP 的主 Internet 地址。在反向的路径上,所有的分组都要经过逆向的地址映射。从这个角度来看,对于外部 Internet 而言, ISP 和它的 ADSL 或有线电视家庭用户看起来就好像是一个大的公司一样。

虽然这种方案从某种程度上解决了问题,但是, IP 社团中的许多人认为它有点匪夷所思。简单地总结一下,这里可以列举出它的一些缺陷。首先, NAT 违反了 IP 的结构模型。IP 的结构模型声明了每一个 IP 地址均惟一标识了一台机器。Internet 的软件结构也是建立在这样的事实基础之上的。然而,采用了 NAT 之后,成千上万台机器可能会使

用地址 10.0.0.1(事实上也确实如此)。

第二,NAT 将 Internet 从一个无连接的网络改变成一个面向连接的网络。问题是,NAT 盒必须为每一个从它这里经过的连接维护必要的信息(映射关系)。让网络来维护连接的状态,这是面向连接的网络的一种特性,而不是无连接网络的特性。如果 NAT 盒崩溃,则它的映射表就丢失了,于是,它的所有 TCP 连接都被破坏。在不使用 NAT 的情况下,路由器崩溃不会影响到 TCP。发送进程只是在几秒钟之内发生超时,然后重传所有未被确认的分组。使用了 NAT 之后,Internet 就如同电路交换的网络一样,变得非常脆弱。

第三,NAT 违反了最基本的协议分层规则:第  $k$  层不应该对“第  $k+1$  层将在第  $k$  层的净荷域中放置什么样的内容”作任何假设。这条基本原则可以保证层与层之间的独立性。如果 TCP 后来又升级到 TCP-2,它的头结构有所不同(比如使用 32 位的端口),那么 NAT 将不再工作。分层协议的总体思想是,保证某一层上的变化不会要求另一层也跟着改变。NAT 破坏了这种独立性。

第四,Internet 上的进程并不一定总是使用 TCP 或者 UDP。如果机器 A 上的一个用户决定使用一种新的传输协议与机器 B 上的用户进行通话(比如,一个多媒体应用),那么,由于 NAT 的介入,这样的应用将无法工作,因为 NAT 盒将无法正确地定位到 TCP 的源端口(Source port)。

第五,有些应用会在正文内容中插入 IP 地址。然后接收方从正文中提取出这些地址,并使用它们。由于 NAT 对这些地址一无所知,它不可能替换这些地址,所以,远端系统若企图使用这样的地址就会失败。标准的文件传输协议 **FTP(File Transfer Protocol)** 就是以这种方式来工作的,除非采取特殊的防范措施,否则在通过 NAT 的情况下它是不能工作的。类似地,H.323 Internet 电话协议(我们将在第 7 章讨论该协议)也有这样的特性,所以,在有 NAT 的情况下它不能工作。对 NAT 打上补丁以便可以与 H.323 一起工作,这是有可能的,但是,每次有新应用出现的时候都要为 NAT 盒打补丁显然不是一个好主意。

第六,由于 TCP Source port 域是 16 位的,所以,至多只有 65 536 台机器可以被映射到同一个 IP 地址上。实际上,这个数值还要略小一些,因为前 4096 个端口被保留作特殊的用途。然而,如果可以使用多个 IP 地址的话,那么,每个地址可以处理多达 61 440 台机器。

RFC 2993 讨论了 NAT 的这些问题以及其他一些问题。通常,NAT 的反对者们说,通过用一种临时的劣等手段来修补 IP 地址不足的问题,从而缓减实现真正方案(即迁移到 IPv6)的压力,这实在不是一件好事情。

### 5.6.3 Internet 控制协议

在 Internet 的网络层上,除了 IP 用于数据传输以外,其他还有一些协议,包括 ICMP、ARP、RARP、BOOTP 和 DHCP。在本节中,我们将依次讨论这些协议。

#### Internet 控制消息协议

路由器可以严密地监视 Internet 的操作。当意外事情发生时,通过 **ICMP(Internet**

**Control Message Protocol, Internet 控制消息协议**)可以报告有关的事件,ICMP 也可以用于测试 Internet。已经定义的 ICMP 消息类型大约有 10 多种,图 5.61 列出了最重要的一些消息类型。每一种 ICMP 消息类型都被封装在一个 IP 分组中。

消息类型	说明
DESTINATION UNREACHABLE(目标不可达)	不能递交分组
TIME EXCEEDED(超时)	TTL 域到达 0
PARAMETER PROBLEM(参数问题)	无效的头域
SOURCE QUENCH(源端抑制)	抑制分组
REDIRECT(重定向)	告诉路由器有关地理信息
ECHO(ECHO 请求)	问一台机器是否还活着
ECHO REPLY(ECHO 应答)	是的,我还活着
TIMESTAMP REQUEST(时间戳请求)	同 ECHO 请求一样,但是加上时间戳
TIMESTAMP REPLY(时间戳应答)	同 ECHO 应答一样,但是加上时间戳

图 5.61 主要的 ICMP 消息类型

当子网或者路由器不能定位到一个分组的目标,或者当一个设置了 DF 位的分组由于途中经过一个“小分组”网络而不能被递交的时候,则路由器可以使用 DESTINATION UNREACHABLE 消息来报告情况。

当一个分组由于它的计数器到达 0 而被丢弃的时候,路由器发送 TIME EXCEEDED 消息。这种事件也是以下情况的一种征兆:分组进入了路由循环,或者有大量的拥塞,或者定时器的值设置得太小。

PARAMETER PROBLEM 消息表示,在头域中检测到一个非法的值。这个问题说明了发送主机的 IP 软件中存在错误,或者也可能是中途路由器的软件中存在错误。

SOURCE QUENCH 消息以前被用来抑制那些发送太多分组的主机。当一台主机接收到这条消息的时候,它应该将发送速度减慢下来。这种消息现在很少使用了,因为当拥塞发生的时候,这些分组无疑是火上浇油。现在,Internet 中的拥塞控制任务主要是在传输层上完成的;我们将在第 6 章中学习有关的细节。

当路由器注意到一个分组看起来被错误地转发过来的时候,它使用 REDIRECT 消息,将可能的错误信息告诉发送方主机。

ECHO 和 ECHO REPLY 消息可以用来判断一个指定的目标是否可达,以及是否还活着。目标主机接收到 ECHO 消息之后,它应该送回一个 ECHO REPLY 消息。TIMESTAMP REQUEST 和 TIMESTAMP REPLY 消息的用途类似,只不过在应答消息中包含了请求消息的到达时间和应答消息的发出时间。此项设施可以用来测量网络的性能。

除了这些消息以外,ICMP 协议还定义了一些消息。通过以下页面可以查看到在线列表 [www.iana.org/assignments/icmp-parameters](http://www.iana.org/assignments/icmp-parameters)。

### ARP—地址解析协议(Address Resolution Protocol)

尽管 Internet 上的每台机器都有一个(或多个)IP 地址,但是,真正在发送分组时候使用的并不是 IP 地址,因为数据链路层硬件并不理解 Internet 地址。如今,公司和大学里的绝大多数主机都通过一块接口卡连接到 LAN 上,该接口卡只能理解 LAN 地址。例如,每一块以太网卡在出厂的时候都配置了一个 48 位的以太网地址。以太网卡的厂家从一个中心权威机构申请一块地址,这样可以保证任何两块网卡都不会有相同的地址(以避免两块网卡出现在同一个 LAN 上的时候发生冲突)。网卡根据其 48 位以太网地址来发送和接收数据帧。它们对于 32 位 IP 地址完全一无所知。

现在问题就来了:如何将 IP 地址映射到数据链路层的地址,比如以太网地址呢?为了解释这一工作过程,我们来看一下图 5.62 中的例子,这个例子演示了一个规模较小的大学,它只有几个 C 类(现在称为 /24)网络。在这里,我们有两个以太网,一个在计算机系,它的 IP 地址是 192.31.65.0,另一个在电子工程系,它的 IP 地址为 192.31.63.0。这两个网络都连接到一个校园骨干环(比如 FDDI)上,骨干环网的 IP 地址是 192.31.60.0。以太网上的每台机器都有一个惟一的以太网地址,我们将它们标记为 E1 至 E6;FDDI 环上的每台机器都有一个 FDDI 地址,我们将它们标记为 F1 至 F3。

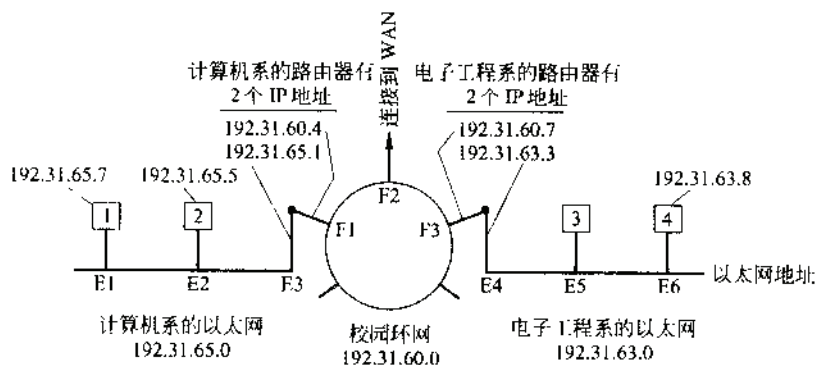


图 5.62 三个相互连接的 /24 网络:两个以太网和一个 FDDI 环

我们首先看一下主机 1 上的用户如何给主机 2 上的用户发送分组。假定发送方知道了目标接收方的名字,可能是像 mary@eagle.cs.uni.edu 这样的名字。第一步是找到主机 2 的 IP 地址,比如说要找到 eagle.cs.uni.edu 的地址。这个查找过程可以通过域名系统(Domain Name System,简称 DNS)来完成,关于域名系统我们将在第 7 章中学习。现在,我们假设 DNS 返回主机 2 的 IP 地址(192.31.65.5)。

主机 1 的上层软件现在构建一个分组,其 Destination address(目标地址)域为 192.31.65.5,然后它将该分组交给 IP 软件来发送。IP 软件看到该地址后,检查一下看这个目标地址是否也在它自己的网络上,但是,它需要某一种办法来找到目标主机的以太网地址。一种解决方案是,在系统中使用一个配置文件,该配置文件包含了从 IP 地址到以太网地址的映射关系。虽然这种方案肯定是可行的,但是,对于拥有几千台机器的组织来



说,保持所有主机上的配置文件总是能够及时得到更新,则是一件既容易出错,又费时的任务。

一个更好的方案是,主机 1 发送一个广播分组到以太网络上,问“谁拥有 IP 地址 192.31.65.5?”该广播分组将会到达以太网 192.31.65.0 上的每一台主机,并且每台主机都会检查它自己的 IP 地址。只有主机 2 会用它自己的以太网地址(E2)作为应答。通过这种方式,主机 1 得知,IP 地址 192.31.65.5 所在的主机的以太网地址为 E2。在这里,问这个问题和得到应答这两个过程所使用的协议称为 ARP(Address Resolution Protocol,地址解析协议)。Internet 上几乎每一台机器都在运行这个协议。RFC 826 定义了 ARP。

与使用配置文件相比,使用 ARP 的优点是简单。系统管理员只要给每台机器分配一个 IP 地址,并且确定好子网掩码,其他不用做任何事情。ARP 会负责处理好所有其他的事情。

这时候,主机 1 上的 IP 软件构建一个以太网帧,其目标地址为 E2,并且把 IP 分组(目标指向 192.31.65.5)放到以太网帧的净荷域中,然后将它发送到以太网上。主机 2 的以太网卡检测到这一帧,识别出这是给自己的帧,于是将它接收进来,并产生一个中断。以太网驱动程序从净荷域中提取出 IP 分组,并将它传递给 IP 软件,IP 软件看到它的目标地址正是指向自己的,于是对它进行处理。

为了使 ARP 的效率更高,我们可以进行各种优化处理。首先,一旦一台机器已经运行了 ARP,那么,它可以将结果缓存起来,以便稍后它还要与同一台机器进行通信。下一次通信的时候,它就可以在缓存中找到地址映射关系,从而不必再进行第二次广播。在许多情况下,主机 2 将需要送回一个应答,这迫使它也要运行 ARP 以确定发送方的以太网地址。此 ARP 广播是可以避免的,因为主机 1 可以将它的“IP-以太网”地址映射关系包含在它的 ARP 分组中。当 ARP 广播分组到达主机 2 的时候,(192.31.65.7, E1)映射对也进入到主机 2 的 ARP 缓存中,以便将来使用。实际上,该以太网上的所有机器都可以将这对映射关系放到它们的 ARP 缓存中。

另一种优化方法是,让每台机器在启动的时候广播它的地址映射关系。此次广播的形式通常是这样的:主机发送 ARP 请求查找它自己的 IP 地址。按理来说,网络上应该不会有应答。这个广播分组的副作用是,其他每台主机在 ARP 缓存中加入一个映射表项。如果意外地收到了一个应答,那么,一定是两台机器被分配了相同的 IP 地址。新的机器应该通知系统管理员,并且停止启动。

为了使地址映射关系可以改变(例如,当一块网卡出现故障并且换上新网卡的时候,以太网地址改变了),ARP 缓存中的映射表项应该在几分钟之后超时。

现在我们再来看图 5.62,只不过这一次主机 1 想要给主机 4(192.31.63.8)发送分组。使用 ARP 将会失败,因为主机 4 不会看到广播消息(路由器不会转发以太网层次上的广播消息)。这里可以有两种解决方案。第一,可以对计算机系的路由器进行配置,让它也响应对于网络 192.31.63.0(可能还有其他的本地网络)的 ARP 请求。在这种情况下,主机 1 将在 ARP 缓存中增加一项(192.31.63.8, E3),并且很愉快地将所有送给主机 4 的流量发送给本地路由器。这种方案被称为代理 ARP(proxy ARP)。第二种方案是,让主机 1 立即看到目标主机在另外一个远程网络上,并且将所有这样的流量都发送给一个

默认的以太网地址,由它负责处理所有的远程流量,在本例中即是 E3。这种方案并不要求让计算机系的路由器知道它要为哪些远程网络提供服务。

无论采用哪一种方法,主机 1 都要将 IP 分组包装到一个目标地址为 E3 的以太网帧的净荷域中。当计算机系的路由器接收到此以太网帧的时候,它从净荷域中提取出 IP 分组,然后在它的路由表中查找该分组的 IP 地址。它发现,发给网络 192.31.63.0 的分组应该首先到达路由器 192.31.60.7。如果它还不知道 192.31.60.7 的 FDDI 地址,那么,它广播一个 ARP 分组到环网上,得到 192.31.60.7 的 FDDI 地址为 F3。然后,它将该分组插入到一个目标地址为 F3 的 FDDI 帧的净荷域中,并将该帧发送到环网上。

在电子工程系的路由器中,FDDI 驱动程序从净荷域中提取出分组,并且将它交给 IP 软件,IP 软件发现它需要将该分组发送给 192.31.63.8。如果此 IP 地址没有出现在它的 ARP 缓存中,那么,它在电子工程系的以太网上广播一个 ARP 请求,得知目标地址是 E6,所以它建立一个目标地址为 E6 的以太网帧,并把分组放到它的净荷域中,然后将它发送到以太网上。当该以太网帧到达主机 4 的时候,主机 4 将分组从帧中提取出来,并且交给 IP 软件以便进一步处理。

从主机 1 经过一个 WAN 到达另一个远程网络的工作方式本质上也是完全相同的,只不过,这一次计算机系路由器中的路由表指明了要使用 WAN 路由器,它的 FDDI 地址为 F2。

### RARP、BOOTP 和 DHCP

ARP 解决的问题是,给定一个 IP 地址,如何找到对应的以太网地址。有时候,反向的问题也必须要解决,即:给定一个以太网地址,它对应的 IP 地址是什么呢?特别是,当一台无盘工作站启动的时候,这个问题就会发生。无盘工作站通常要从远程文件服务器获得其操作系统的二进制映像。但是它如何得到它的 IP 地址呢?

第一个设计的解决方案是使用 **RARP**(Reverse Address Resolution Protocol,反向地址解析协议)(在 RFC 903 中定义)。该协议允许一台新启动的工作站广播它的以太网地址,并说:“我的 48 位以太网地址是 14.04.05.18.01.25,这里有人知道我的 IP 地址吗?”RARP 服务器看到这个请求之后,在它的配置文件中查找该以太网地址,然后将对应的 IP 地址送回给工作站。

使用 RARP 比直接将 IP 地址嵌在内存映像中要好得多,因为这样做带来的好处是,同一份映像可被用于所有的机器。如果 IP 地址被嵌在映像的内部,那么,每台工作站都需要它自己专用的映像。

RARP 的一个缺点是,它使用全 1 的目标地址(受限的广播)来跟 RARP 服务器进行通信。然而,路由器并不会转发这样的广播消息,所以,每个网络上都需要配一台 RARP 服务器。为了克服这个问题,人们又发明了另一个称为 **BOOTP** 的启动协议。与 RARP 不同的是,BOOTP 使用 UDP 消息,而 UDP 消息可以被路由器转发。它同时也为无盘工作站提供了额外的信息,包括存放内存映像的文件服务器的 IP 地址、默认路由器的 IP 地址,以及即将使用的子网掩码。RFC 951、1048 和 1084 描述了 BOOTP 协议。

BOOTP 的一个严重问题是,它要求手工配置一张“将 IP 地址映射到以太网地址”的

表。当一台新的主机加入到 LAN 中的时候,首先必须由管理员为它分配一个 IP 地址,并且将它的(以太网地址,IP 地址)映射关系手工加入到 BOOTP 配置表中,然后该主机才可以使用 BOOTP。为了避免这一极易出错的手工步骤,人们又对 BOOTP 进行了扩展,并且给它一个新的名字: **DHCP(Dynamic Host Configuration Protocol,动态主机配置协议)**。DHCP 既允许手工分配 IP 地址,也允许自动分配 IP 地址。RFC 2131 和 2132 描述了 DHCP 协议。在大多数系统中,它已经相当程度上取代了 RARP 和 BOOTP。

如同 RARP 和 BOOTP 一样,DHCP 的基本思想也是用一台专门的服务器来为每一台发出请求的主机分配 IP 地址。这台服务器与发出请求的主机可以不在同一个 LAN 上。由于通过广播消息可能无法到达 DHCP 服务器,所以,每个 LAN 上需要一个 **DHCP 中继代理(DHCP relay agent)**,如图 5.63 所示。

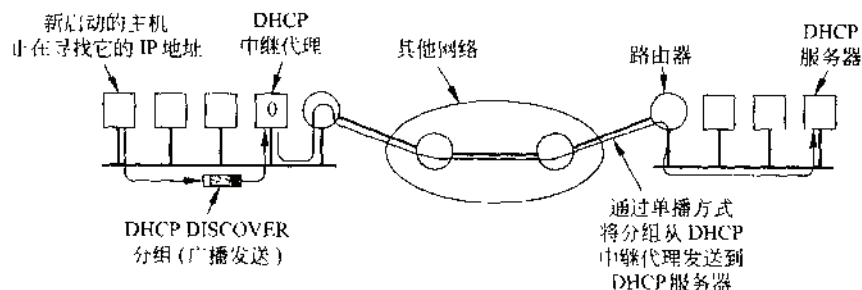


图 5.63 DHCP 的操作过程

对于一台新启动的机器,为了找到它的 IP 地址,它首先广播一个 DHCP DISCOVER 分组。它所在 LAN 上的 DHCP 中继代理截获到所有的 DHCP 广播消息。当它发现了 DHCP DISCOVER 分组的时候,它用单播方式将该分组发送给 DHCP 服务器,而 DHCP 服务器可能位于一个远程网络上。中继代理需要的惟一信息是 DHCP 服务器的 IP 地址。

“从一个 IP 地址池中自动为主机分配地址”带来的一个问题是,所分配的 IP 地址可以使用多长时间? 如果一台主机离开了网络,并且它没有将地址交回给 DHCP 服务器,那么,该地址将永久丢失了。经过一段时间运行之后,可能会丢失许多地址。为了避免发生这种情况,IP 地址分配的有效周期只能是一段固定长的时间,这项技术称为 **租用(leasing)**。在到达租用期之前,主机必须向 DHCP 服务器申请 **续租(renewal)**。如果一台主机未能发出这样的请求,或者请求被拒绝的话,它也许就不能再使用以前得到的 IP 地址了。

#### 5.6.4 OSPF—内部网关路由协议

我们已经学完了 Internet 控制协议,现在该转移到下一个话题上: Internet 上的路由过程。正如我们前面所提到的,Internet 是由大量的自治系统(AS)构成的,每个 AS 由不同的组织来运行,其内部可以使用自己的路由算法。例如,如果公司 X、Y 和 Z 的内部网络都在 Internet 上的话,那么,这三个网络通常可以被看作三个 AS,它们的内部可能使用

不同的路由算法。然而,有一套统一的标准,不仅对于内部路由有用,同时也可以简化 AS 之间边界上的路由实现,而且还允许代码重用。在这一小节中,我们将学习 AS 内部的路由算法;在下一小节中,我们将学习 AS 之间的路由算法。AS 内部的路由算法称为**内部网关协议 (interior gateway protocol)**; AS 之间的路由算法称为**外部网关协议 (exterior gateway protocol)**。

最初的 Internet 内部网关协议是一个基于 Bellman-Ford 算法的距离矢量协议 (RIP),它是从 ARPANET 中继承过来的。它在小系统中工作得很好,但是,随着 AS 变得越来越大,该协议就不适合了。同时它还存在“计数到无穷”的问题,而且一般情况下收敛速度非常慢,所以,1979 年 5 月,它被一个链路状态协议所取代。1988 年,IETF (Internet Engineer Task Force, Internet 工程任务组织)开始开发其后继的协议。此后继协议称为 **OSPF (Open Shortest Path First, 开放的最短路径优先)**,它于 1990 年成为标准。现在许多路由器厂商都支持 OSPF,并且它已经成为最主要的内部网关协议。接下来我们将概要地介绍一下 OSPF 的工作过程,有关详细的信息,请参考 RFC 2328。

由于有了其他路由协议的长期工作经验,负责设计新协议的工作组列出了一长串必须要满足的需求。首先,该算法必须被发表在公开的文献中,这便是 OSPF 中“(O)” (开放的)的含义。由某一家公司拥有的私有方案是无法做到这一点的。第二,新的协议必须支持多种距离度量,包括物理距离、延迟,等等。第三,它必须是一个动态算法,是一个能够自动而且快速地适应网络拓扑变化的路由算法。

第四,对于 OSPF 来说这是新的需求,它必须支持建立在服务类型基础上的路由选择。新的协议必须能够区分实时流量和其他的流量,并使用不同的路由方法。IP 协议有一个服务类型 (type of service) 域,但是原来的路由协议都没有使用这个域。OSPF 中包含了该域,但是仍然没有人使用它,最终它又被去掉了。

第五条与上面一条也有关系,也就是说,新的协议必须要实现负载均衡,即把负载分散到多条线路上。大多数以前的协议都将所有的分组通过最优路径发送出去,而次优的路径根本不使用。在许多情况下,将负载分散到多条线路上可以获得更好的性能。

第六,需要支持分层次的系统。到 1988 年的时候,Internet 已经增长到相当大的规模,以至于任何一台路由器都不可能知道完整的 Internet 拓扑结构。新的路由协议必须考虑到这样的情况,不要求路由器知道完整的拓扑结构。

第七,要求提供适度的安全性,以防止恶作剧的学生向路由器发送虚假的路由信息来欺骗路由器。最后,对于那些通过隧道连接到 Internet 的路由器,新协议也必须能够对它们进行处理。以前的协议并不能很好地解决这样的问题。

OSPF 支持三种连接和网络:

- (1) 两台路由器之间的点到点线路;
- (2) 支持广播传送的多路访问网络 (比如大多数 LAN);
- (3) 不支持广播传送的多路访问网络 (比如大多数分组交换的 WAN)。

所谓多路访问 (multiaccess) 网络是指这样的网络,它有多台路由器,每台路由器可以直接与其他所有的路由器进行通信。所有的 LAN 和 WAN 都有这个特性。图 5.64(a) 显示了一个包含所有这三种网络的 AS。请注意,主机通常并不参与到 OSPF 中。





在一个区域的内部,每台路由器都有同样的链路状态数据库,并运行同样的最短路径算法。它的主要任务是计算出从它这里到同一区域中任何其他一台路由器之间的最短路径,其中也包括那台与骨干区域相连接的路由器;在一个区域中,至少有一台路由器连接到骨干上。如果一台路由器同时连接到两个区域,那么针对这两个区域它需要维护单独的数据库,而且必须为每个区域单独运行最短路径算法。

在正常操作过程中,可能需要三种路由路径:区域内的路径、区域之间的路径和 AS 之间的路径。区域内的路径是最容易的,因为源路由器已经知道了它与目标路由器之间的最短路径。区域之间的路径总是按三步来进行:从源路由器到达骨干区域;跨过骨干区域到达目标区域;再到达目标路由器。此算法强迫在 OSPF 上建立起一个星形配置结构,其中骨干区域是中心点(集线器),其他的区域是向外的辐射点。所有的分组“照原样”从源端被路由到目标端,它们没有被封装,也没有通过隧道传输,除非途中有一个区域到骨干区域之间的连接是一条隧道。图 5.65 显示了由一些 AS 和区域构成的部分 Internet。

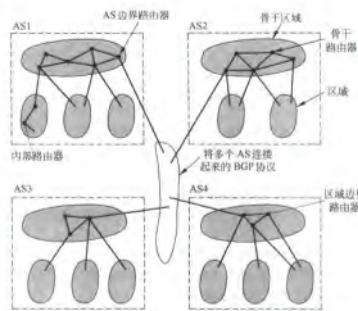


图 5.65 在 OSPF 中,AS、骨干和区域之间的关系

OSPF 区分 4 种路由器:

- (1) 内部路由器,完全在一个区域的内部;
- (2) 区域边界路由器,连接两个或多个区域;
- (3) 骨干路由器,位于骨干区域上;
- (4) AS 边界路由器,与其他 AS 中的路由器进行通信。

这 4 种路由器允许有重叠。例如,所有的边界路由器自然就是骨干的一部分。而且,如果一台路由器虽然在骨干上,但是并不属于任何其他的区域,那么它也是一台内部路由

器。图 5.65 演示了所有这 4 类路由器的例子。

当一台路由器启动的时候,它在所有的点到点线路上发送 HELLO 消息,并且在 LAN 上将 HELLO 消息多播到一个包含所有其他路由器的组。在 WAN 上,它需要一些配置信息,以便知道该与谁通信。每台路由器从应答消息中得知谁是它的邻居。同一个 LAN 上的路由器都是邻居。

OSPF 协议需要在邻接的路由器之间相互交换信息才能工作,邻接(adjacent)的路由器与邻居路由器不是同样的概念。尤其是,让一个 LAN 中的每台路由器都跟该 LAN 中的其他每台路由器进行通话显然是非常低效的。为了避免这样的情形,OSPF 要求从一个 LAN 中选举出一台路由器作为指派路由器(designated router)。指派路由器与该 LAN 上所有其他的路由器都是邻接的(adjacent),并且与它们交换信息。如果两台邻居路由器不是邻接的,则它们相互之间并不交换信息。有一台备份的指派路由器总是保持最新的状态数据,以便当主指派路由器崩溃的时候,可以很容易地切换过来,立即取代原来的主指派路由器。

在正常操作过程中,每台路由器周期性地扩散 LINK STATE UPDATE 消息到它的每台邻接路由器。这条消息给出了它的状态信息,并且提供了在拓扑数据库中用到的权值。这些扩散消息也需要被确认,以保证它们的可靠性。每条消息都有一个序列号,所以,路由器收到一条 LINK STATE UPDATE 消息后,它可以判断出这条消息中的信息比它当前拥有的信息更老,还是更新。当一条线路刚刚启用,或者停止使用,或者权值改变的时候,路由器也要发送 LINK STATE UPDATE 消息。

通过使用 LINK STATE REQUEST 消息,每一对邻接路由器中的任一台路由器都可以向另一台路由器请求链路状态信息。这个算法的结果是,每一对邻接路由器都可以检查谁有最新的数据;新的信息就是通过这种方式被传播到整个区域中的。所有这些消息都是以原始的 IP 分组被发送出去的。图 5.66 概括了这 5 种消息。

消息类型	说明
HELLO	用于发现谁是自己的邻居
LINK STATE UPDATE	向邻居们提供发送方的开销(权值)
LINK STATE ACK	对链路状态更新消息的确认
DATABASE DESCRIPTION	宣布发送方所拥有的状态信息的更新情况
LINK STATE REQUEST	向邻接路由器请求链路状态信息

图 5.66 5 种 OSPF 消息

最后,我们把所有的片断整理到一起。通过扩散法,每台路由器把它的邻居信息和开销(权值)告诉给它的区域中所有其他的路由器。这些信息使得每台路由器都可以构建出它的区域的拓扑图,并且计算出最短路径。骨干区域也是这样运行的。而且,骨干路由器也接受来自区域边界路由器的信息,这样可以计算出从每台骨干路由器到其他每台路由器的最优路径。这些信息又被传回到区域边界路由器,区域边界路由器再将这些信息广播到它们的区域中。利用这些信息,一台路由器在发送跨区域的分组时,它可以选择最优

的出口路由器到达骨干区域。

### 5.6.5 BGP—外部网关路由协议

在一个 AS 内部,推荐使用的路由协议是 OSPF(尽管它肯定不是惟一一个正在使用的内部网关路由协议)。在 AS 之间,则可以使用另一个协议: **BGP (Border Gateway Protocol, 边界网关协议)**。之所以在 AS 之间需要一个完全不同的协议,是因为内部网关协议和外部网关协议的目标并不相同。一个内部网关协议所需要做的事情是,尽可能有效地将分组从源端传送到目标端。它不必考虑政治方面的因素。

然而,外部网关协议路由器则必须要考虑大量有关政治的因素(Metz, 2001)。例如,一个公司的 AS 可能希望能够给所有的 Internet 站点发送分组,同时也能够接收来自任何一个 Internet 站点的分组。然而,它可能不希望承载那些“源端在一个外部 AS 中,而目标端在另一个外部 AS 中”的分组的中转任务,尽管它自己的 AS 正好位于这两个外部 AS 之间的最短路径上(“那是他们的问题,不是我们的问题”)。另一方面,它可能愿意转送其邻居们的流量,或者愿意为那些已经付费的 AS 提供流量中转服务。例如,电话公司可能很愿意为他们的顾客提供传输中介的服务,但是不愿意为别人也提供这样的服务。无论是一般意义上的外部网关协议,还是特殊的 BGP 协议,它们在设计的时候都提供了许多种路由策略,这些策略可被强制用于那些跨越 AS 的流量。

典型的路由策略可能会涉及到政治、安全或者经济方面的考虑因素。下面是一些路由限制的例子:

- (1) 对于那些经过了某些特定 AS 的流量,不提供传输服务;
- (2) 永远不会把伊拉克放在一条起始于五角大楼的路径上;
- (3) 从不列颠哥伦比亚(British Columbia)到安大略(Ontario)不经过美国;
- (4) 只有当没有其他的选择可以到达目标的时候,才经过阿尔巴尼亚(Albania);
- (5) 在 IBM 起始或者终止的流量不应该经过 Microsoft。

通常,这些策略是以手工的方式被配置到每台 BGP 路由器中的(或者,也可能使用某种形式的脚本),它们本身并不是协议的一部分。

从一台 BGP 路由器的角度来看,这个世界是由 AS 和连接 AS 的线路构成的。如果一条线路的两头分别与两个 AS 中各一台边界路由器相连接,那么称这两个 AS 是相连的。根据 BGP 对于中转流量的用途,我们可以将网络分成三大类。第一类是**末端网络(stub network)**,它与 BGP 图之间只有一个连接。中转流量不可能使用这些网络,因为另一端没有路由器接应。然后是**多连接网络(multiconnected network)**。除非这些网络拒绝中转流量,否则就可以利用它们来传输中转流量。最后一种网络是**穿越网络(transit network)**,比如骨干网,它们都愿意处理第三方的分组,不过可能会有些限制条件,而且通常需要付费。

BGP 路由器对之间通过建立 TCP 连接来相互通信。使用这种方法既提供了可靠的通信,又隐藏了有关中途网络的所有细节情况。

BGP 基本上是一个距离矢量协议,但是它与大多数其他的距离矢量协议(比如 RIP)有显著的不同。每一台 BGP 路由器并不仅仅维护它到每个目标的开销值,而且还记录下

所使用的路径。类似地,每一台 BGP 路由器并不是定期地告诉它的邻居们关于到每个可能目标的估计开销值,而是将它所用的确切路径告诉它的邻居们。

举一个例子,请考虑图 5.67(a)中所示的 BGP 路由器。尤其请注意 F 的路由表。假设它使用 FGCD 作为到达 D 的路径。当它的邻居们告诉它路由信息的时候,它们都提供了完整的路径,如图 5.67(b)所示(为了简化起见,这里只显示了目标 D)。

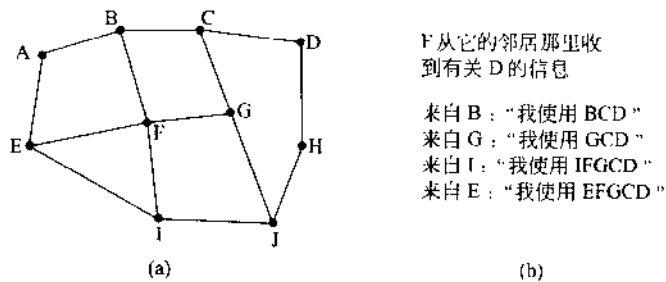


图 5.67

(a) 一组 BGP 路由器; (b) 发送给 F 的路由信息

邻居提供的所有路径都到达之后, F 对这些路径进行检查,看哪一条路径是最优的。它很快地丢弃掉来自 I 和 E 的路径,因为这些路径都经过 F 本身。然后需要在 B 和 G 的路径之间做出选择。每一台 BGP 路由器都包含这样一个模块:它检查所有至给定目标的路径,并给它们评分,然后为每条路径返回一个数,代表它到目标的“距离”。如果一条路径违反了某一项策略限制,则它的分值自动为无穷大。然后路由器采用距离最短的那条路径。这里的评分函数并不是 BGP 协议的一部分,它可以是系统管理员期望的任何一个函数。

BGP 很容易地解决了一直困扰着其他距离矢量路由算法的“计数到无穷”的问题。例如,假设 G 崩溃了,或者线路 FG 停止使用了。然后 F 接收到来自其余三个邻居的路径信息。在这些路径信息中,到 D 的路径分别是 BCD、IFGCD 和 EFGCD。它立即就可以看出,后两条路径是没有意义的,因为它们都经过了 F 本身,所以它选择 FBCD 作为新的路径。其他的距离矢量算法通常会做出错误的选择,因为它们无法辨别哪些邻居有通向目标的独立路径,哪些邻居没有独立的路径。有关 BGP 的定义,请参考 RFC 1771-1774。

### 5.6.6 Internet 多播

普通的 IP 通信是在一个发送方和一个接收方之间进行的。然而,对于有些应用,它们需要一个进程能够同时向大量接收方发送数据。这样的例子应用有:更新备份数据、分布式数据库、将股票价格传送给多个经纪人,以及处理数字会议(即多方会议)中的电话呼叫。

IP 通过 D 类地址来支持多播。每个 D 类地址标识了一组主机。总共有 28 位可用于标识多播组,所以可以有 250M 多个组同时并存。当一个进程给一个 D 类地址发送分组的时候,网络会尽力将它投递给指定的组中的所有成员,但是并不保证一定投递成功。有

些成员可能收不到该分组。

IP 支持两种组地址：永久地址和临时地址。永久组地址总是可以使用，无需事先建立组。每个永久组都有一个永久组地址。下面是一些永久组地址的例子：

- 224.0.0.1      一个 LAN 上的所有系统
- 224.0.0.2      一个 LAN 上的所有路由器
- 224.0.0.5      一个 LAN 上的所有 OSPF 路由器
- 224.0.0.6      一个 LAN 上的所有 OSPF 指派路由器

临时组则有所不同，只有在创建了临时组之后才可以使用它们。一个进程可以要求它的主机加入到某一个指定的组中。以后，它也可以要求它的主机离开该组。当一台主机上最后一个进程离开一个组的时候，此主机不再属于这个组。每台主机都记录了它的进程当前属于哪些组。

IP 多播是通过一些特殊的多播路由器来实现的，这些多播路由器可能与标准的路由器在同一台机器上，也可能不是。每台多播路由器差不多每分钟一次向它所在 LAN 上的主机(地址为 224.0.0.1)发送一条硬件(即数据链路层)多播消息，请它们报告一下它们的进程当前属于哪些组。每台主机收到消息之后，送回应答，并且在应答消息中告诉多播路由器自己对哪些 D 类地址感兴趣。

这些查询和应答分组使用了一个称为 **IGMP (Internet Group Management Protocol, Internet 组管理协议)** 的协议，该协议大致上与 ICMP 类似。它有两种分组：查询和请求，每一种分组各有一个简单、固定的格式，其中在净荷域的第一个字中包含了一些控制信息，在第二个字中包含了一个 D 类地址。RFC 1112 描述了 IGMP 协议。

多播路由是通过生成树来完成的。为了让每一台多播路由器都能够为每个组建立起一棵覆盖所有组成员的生成树，每一台多播路由器都使用一种被修改过的距离矢量协议，与它的邻居路由器交换信息。多播路由协议使用了各种优化措施来对这棵树进行修剪，以消除掉那些对特定的组并不感兴趣的路由器和网络。多播路由协议大量使用了隧道技术，以避免干扰那些不在生成树上的节点。

### 5.6.7 移动 IP

许多 Internet 用户都有便携式计算机，当他们走访一个外地的 Internet 站点，甚至在旅途中的时候，他们也希望能够与 Internet 保持连接的状态。不幸的是，IP 的编址系统使得像这样的异地工作说起来容易做起来很难。在这一小节中，我们将讨论这个问题及其解决方案。有关更细节的描述请参考(Perkins, 1998a)。

真正的罪魁是 IP 编址方案本身。每个 IP 地址包含一个网络号和一个主机号。例如，考虑一台 IP 地址为 160.80.40.20/16 的机器。其中 160.80 给出了网络号(十进制写法为 8272)；40.20 是主机号(十进制写法为 10260)。全世界所有路由器的路由表中都指明了通过哪一条线路可以到达网络 160.80。当一个进来的分组中的 IP 地址为 160.80.xxx.yyy 的时候，该分组就被输出到这条线路上。

如果突然之间，使用该地址的那台机器被搬到了某一个远方的 Internet 站点中，那么，发送给该地址的分组仍然被路由到它原来的 LAN(或路由器)中。机器的使用者将收



不到电子邮件,也无法接收到其他的信息。你当然可以为这台机器分配一个可在新的网络中使用的新 IP 地址,但是这种做法并不值得称道,因为做了这样的变化之后,将需要通知很多人、很多程序和数据库。

另一种办法是,让路由器在路由的时候使用完整的 IP 地址,而不仅仅使用网络号信息。然而,这种策略将要求每一台路由器维护成千上百万的表项,这样的开销对于 Internet 来说简直是一个天文数字。

当人们开始期望在任何地点都能够将他们的笔记本电脑连接到 Internet 的时候, IETF 建立了一个工作组来寻求解决方案。该工作组很快整理出了任何一个方案都应该达到的一系列目标,其中最主要的有以下几点:

- (1) 每台移动主机必须能够在任何地方使用它的主 IP 地址(home IP address)。
- (2) 对于固定的主机,不允许要求改动它们的软件。
- (3) 也不允许要求改动路由器软件和路由表。
- (4) 发送给移动主机的大多数分组不应该绕道而行。
- (5) 当移动主机在家里(即它的主场所)的时候,不应该有任何额外的开销。

实际上,5.2.9 节中已经介绍了所选中的解决方案。现在简短地回顾一下,如果一个站点希望它的用户能够漫游,那么它必须创建一个本地代理(home agent)。如果一个站点允许其他的访问者进到它的网络中来,那么它必须创建一个外部代理(foreign agent)。当移动主机在一个外地站点中启动的时候,它与当地的外部代理联系,并且进行注册。然后,外部代理与该用户的本地代理进行联系,并且交给它一个移交地址(care-of address),通常这是外部代理自身的 IP 地址。

当一个分组到达用户的本地 LAN 中的时候,它被转送给某一台与 LAN 相连接的路由器。然后,该路由器试图按照常规的方法来寻找目标主机,也就是说,它发送一个 ARP 广播分组,问:160.80.40.20 的以太网地址是什么?本地代理用它自己的以太网地址来响应此查询请求。然后,路由器将目标地址为 160.80.40.20 的分组全部发送给本地代理。本地代理又通过隧道的方式将这些分组转送到移交地址,其做法是,将这些分组封装到一些新 IP 分组的净荷域中,并将新分组发送给外部代理。然后外部代理将原来的分组解出来,并传递给移动主机的数据链路层地址。此外,本地代理也将移交地址告诉给发送方,所以,将来的分组可以直接以隧道方式发送给外部代理。这种方案满足了上面列举的所有要求。

有一个小的细节也许值得提一下。当移动主机移动的时候,路由器有可能将它(快要失效)的以太网地址缓存起来了。通过一种称为主动 ARP(*gratuitous ARP*)的技巧,就可以用本地代理的以太网地址来取代原来缓存的以太网地址。这是一种特殊的、主动发送给路由器的 ARP 消息,其目的是为了替换掉缓存中一个特定的映射表项,此时,要替换掉的是正要离开的移动主机的映射表项。当移动主机以后回来的时候,它可以使用同样的技巧再一次更新路由器缓存中的映射表项。

这种方案也不限制一台移动主机变成它自己的外部代理(也就是说,外部代理就是移动主机自身),但是,只有当移动主机在当前站点中逻辑上也连接到 Internet 上(即有能力作为外部代理)的时候,这样做才是可行的。而且,移动主机还必须能够获得一个(临时

的)移交地址。该移交 IP 地址必须属于它当前连接的 LAN。

IETF 的移动主机方案也解决了其他许多尚未提及的问题。例如,如何发现这些代理?一种方案是,每个代理定期地广播它的地址和它愿意提供的服务类型(比如,本地代理、外部代理,或者两种服务都能提供)。当一台移动主机到达某一个地方的时候,它只要监听这些广播分组就可以了,这些广播分组也称为广告分组(advertisement)。另一种方案是,移动主机可以主动广播一个分组,以宣告自己的到来,并且期望当地的外部代理做出回应。

另一个必须要解决的问题是,不礼貌的移动主机如果没有说再见就离开了,那该怎么办?解决的方案是,每一次注册之后只保持在一段固定的时间内才有效。如果它没有定期来刷新的话,那么它将超时,所以,外地主机可以清除掉有关它的状态表信息。

还有另一个问题是安全性。如果本地代理接收到一条消息,请求它将所有给 Roberta 的分组统统转发给某一个 IP 地址,那么,除非它能够确认这个请求真的是 Roberta 发送的,并且没有人在模仿她,否则最好不要冒然就答应这个请求。基于密码学的认证协议可以用来实现这样的身份认证任务,我们将在第 8 章中学习这样的协议。

最后一点是,IETF 工作组也考虑到了移动性的级别。请想象飞机上的一个以太网,它主要用于导航和将各种航空计算机设备连接起来。在这个以太网上,一台标准路由器通过无线链路与地面上的有线 Internet 进行通话。有一天,一个聪明的市场主管想到了可以在飞机的所有座位上安装上以太网连接器,这样,那些携带移动计算机的旅客们也可以通过这些插头上网了。

现在我们有了一级移动性:飞机自身的计算机对于以太网来说是固定的,而旅客们的计算机对于以太网来说是移动的。另外,飞机上的路由器相对于地面上的路由器而言也是移动的。如果系统 A 是移动的,另一个系统 B 相对于系统 A 也是移动的,那么,这种嵌套的移动性可以利用递归的隧道技术来处理。

#### 5.6.8 IPv6

虽然 CIDR 和 NAT 可能赢得了许多年的时间,但是,每一个人都意识到,当前的 IP 版本(IPv4)的日子已经为期不远了。除了这些技术问题以外,还有另一个隐藏在背后的问题。在 Internet 的早期,它主要被用于大学、高科技工业和美国政府(特别是美国国防部[Dept. of Defense])。20 世纪 90 年代中期开始,人们对于 Internet 的兴趣不断膨胀,Internet 开始为各种各样的人所使用,尤其是有着不同需求的人们。首先,大量携带无线移动计算设备的人通过 Internet 与他们的家庭或者企业保持联系。其次,随着计算机工业、通信业和娱乐业的不断交融,有可能在不久的将来,世界上的每一部电话和电视都将变成 Internet 节点,从而几十亿台机器将会使用音频和视频点播。在这样的形势下,很显然,IP 必须也要进一步发展,并且要变得更加灵活。

1990 年,IETF 看到了这些问题已经在即,于是开始启动 IP 新版本的设计工作,新版本的 IP 将有用不完的地址,而且还将解决许多其他的问题,同时也更加灵活和高效。它的主要目标是:

- (1) 即使地址空间的分配效率不高,也需要支持几十亿台主机;

- (2) 降低路由表的大小;
- (3) 对协议进行简化,以便路由器更加快速的处理分组;
- (4) 提供比当前的 IP 更好的安全性(认证和隐私);
- (5) 更加关注于服务的类型,特别是针对实时数据的服务类型;
- (6) 允许通过指定范围来支持多播传输;
- (7) 允许主机在不改变地址的情况下就可以漫游;
- (8) 允许协议未来还可以发展;
- (9) 允许新老协议共存多年。

为了开发出一个满足所有这些需求的协议,IETF 在 RFC 1550 中发表了一个寻求提案和讨论的声明,共收到 21 份回应材料。但是,这些回应材料并不全都是完整的提案。到了 1992 年 12 月,有 7 个提案被拿到了桌面上。这些提案相差甚大,涉及范围非常广泛,从“对 IP 作微小的修改”,到“完全抛掉 IP 而用一个全然不同的协议来替代”。

一种提案是在 CLNP 之上运行 TCP,CLNP 有 160 位地址,所以它提供了足够的地址空间,而且也将统一两个主要的网络层协议。然而,许多人认为,这样做好像是承认了 OSI 领域中所做的事情实际上是正确的,而在 Internet 圈子中却存在策略性的错误。CLNP 的模式与 IP 非常相近,所以,这两种协议并没有实质性的不同。实际上,最终选中的协议与 IP 之间的差异,远远超过了 CLNP 与 IP 之间的差异。反对 CLNP 的另一个理由是它对服务类型的支持太差,在有效地传输多媒体数据的时候这是非常必要的。

IEEE Network 发表了三种比较好的提案(Deering, 1993; Francis, 1993; and Katz and Ford, 1993)。在经过多次讨论、修订和定位之后,Deering 和 Francis 两份提案被组合起来又作了修改,然后得到一个现在称为 SIPP(Simple Internet Protocol Plus,增强的简单 Internet 协议)的协议,最终,它被选中,并且称之为 IPv6。

IPv6 很好地满足了以上列出的设计目标。它保持了 IP 的优良特性,丢弃或者削弱了 IP 中不好的特性,并且在必要的地方增加了新的特性。一般而言,IPv6 并不与 IPv4 兼容,但是,它与其他一些辅助性的 Internet 协议则是兼容的,包括 TCP、UDP、ICMP、IGMP、OSPF、BGP 和 DNS,有时候也要求做一点小小的改动(大多数改动是为了处理更长的地址)。下面将讨论 IPv6 的主要特性,有关更多的信息可以在 RFC 2460-2466 中找到。

首先也是最重要的,IPv6 有比 IPv4 更长的地址。IPv6 的地址有 16 字节长,这解决了 IPv6 一开始就想要解决的问题:使用一个能有效地提供几乎无限 Internet 地址的空间。稍后我们还要更多地谈论到 IPv6 的地址。

IPv6 第二个主要的改进是对头部进行了简化。它只包含 7 个域(相比之下 IPv4 有 13 个域)。这一变化使得路由器可以更快地处理分组,从而提高了路由器的吞吐量,并缩短了延迟。同样地,我们后面还要讨论 IPv6 的头结构。

第三个主要改进是更好地支持选项。这一变化对子新的头部来说是本质的,因为以前那些必需的域现在变成了可选的,而且,选项的表达方式也有所不同,这使得路由器可以非常简单地跳过那些与它无关的选项。此特性也加快了分组的处理速度。

第四个改进代表了 IPv6 的重大进步,即在安全性方面的改进。IETF 已经听腻了关

于早熟的 12 岁少年用他们的个人计算机闯入 Internet 上的银行和军事堡垒的新闻故事。在 IPv6 的设计过程中,一种强烈的感觉是要增强安全性。在新的 IP 中,认证和隐私是关键的特征。然而,后来这些特征也被引入到 IPv4 中,所以,IPv6 和 IPv4 在安全性方面的差异已经没有那么大了。

最后,更加值得关注的是服务质量。过去,人们在这方面已经作了大量的努力,现在,随着 Internet 上多媒体的增长,服务质量的需求也更加紧迫了。

### IPv6 主头部

IPv6 的头部如图 5.68 所示。对于 IPv6,版本(Version)域总是 6(对于 IPv4,该域总是 4)。在从 IPv4 到 IPv6 的迁移过程(这个过程可能要持续十年以上)中,路由器通过检查该域来确定分组的类型。顺便提一下,做这样的测试在关键路径中需要浪费少量的指令,所以,许多实现软件可能会试图避免此测试过程,它们的做法是,利用数据链路层中的某个域来区分 IPv4 分组和 IPv6 分组。按照这种方法,所有的分组可以被直接传递给相应的网络层处理器。然而,让数据链路层了解网络分组类型,这种做法完全违背了基本的设计原则,即每一层不应该了解上层给它传递过来的数据位的含义。在“做得正确(Do it right)”与“做得快速(Make it fast)”两大阵营之间的讨论无疑将会既漫长又激烈。

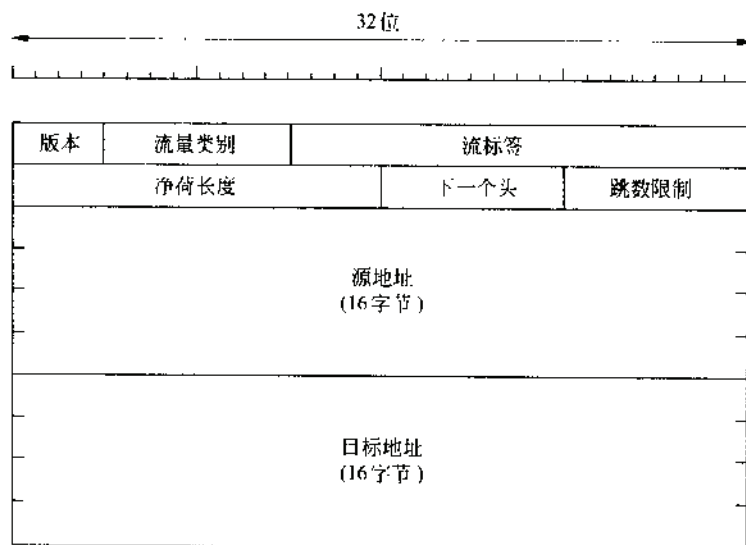


图 5.68 IPv6 固定的头(必需的)

流量类别(Traffic class)域的用途是,按照各种不同的实时递交需求将分组区分开。在 IP 中专门为此目的而设置一个域是一开始就有的想法,但实际上只有少数路由器实现了这个域。现在已经在做一些实验来确定如何更好地将这个域用于多媒体数据的递交过程。

流标签(Flow label)域也是试验性的,但它将来仍然有用途,通过该域,源端和目标端

可以建立一个具有特殊属性和需求的伪连接。例如,从某台特定主机上的一个进程到另一台主机上的一个进程之间的分组流可能有严格的延迟要求,因此需要预留带宽。这时可以提前建立一个流(flow),并分配一个标识符。当一个 Flow label 域非 0 的分组出现的时候,所有的路由器都在自己的内部表中查找该 Flow label 值,看它要求哪一种特殊的待遇。实际上,这样的流是两种传输模型相结合的一种尝试:数据报子网的灵活性和虚电路子网的质量保证。

每个流是通过源地址、目标地址和流编号来指定的,所以,在给定的一对 IP 地址之间,可以同时有许多个活动的流。而且,按照这种方法,来自不同主机的两个流即使有相同的流标签,当它们通过同一台路由器的时候,路由器也能够利用源地址和目标地址将它们区分开。流标签的选取最好是随机的,而不是从 1 开始顺序分配,因此,路由器最好对它们进行散列处理。

净荷长度(Payload length)域指明了紧跟在图 5.68 所示的 40 字节头之后还有多少字节数。在 IPv4 中该域的名字为总长度(Total length),之所以改成现在的名字是因为其含义略有不同:40 字节的头部不再像以前那样作为长度中的一部分。

下一个头(Next header)域正显示了 IPv6 的关键之处。IPv6 头部之所以能够得以简化的原因是,它还可以有附加(可选)的扩展头。该域指明了如果当前头之后还有扩展头的话,该扩展头是哪一种扩展头(当前已经定义了 6 种扩展头)。如果当前的头是最后一个 IP 头,那么,Next header 域指定了该分组将被传递给哪一个传输协议处理器(比如 TCP、UDP)。

跳数限制(Hop limit)域被用来避免分组永远留在网络中。在实践中,它与 IPv4 中的 TTL(Time to live,生存期)域是一样的,也就是说,在每一跳上该域中的值都要被递减。IPv4 中的 TTL 域理论上是一个以秒为单位的时间值,但是所有的路由器都不按照时间值来使用该域,所以,在 IPv6 中将名字改过来,以便反映出它的实际用法。

接下来是源地址(Source address)和目标地址(Destination address)域。在 Deering 的原始提案(SIP)中,他使用了 8 字节地址,但是在提案审阅过程中,许多人认为,如果用 8 个字节作为 IPv6 中的地址,那么在几十年之内地址空间将再次被用完,而如果使用 16 字节的地址,则永远也用不完。其他有人争辩说,16 字节太多了,没有必要;还有人建议使用 20 字节的地址以便与 OSI 数据报协议兼容。甚至还有人建议使用可变长度的地址。在经过了激烈的争论之后,最终作出决定,固定长度的 16 字节地址是最好的折衷方案。

为了书写 16 字节的地址,IETF 也设计了一种新的标记法。16 个字节被分成 8 组来书写,每一组 4 个十六进制数字,组之间用冒号隔开,如下所示:

8000:0000:0000:0000:0123:4567:89AB:CDEF

由于许多地址的内部可能有很多个 0,所以,有三种优化方法也可以使用。第一种,在一个组内,前导的 0 可以省略,比如,0123 可以写成 123。第二,16 个“0”位构成的一个或多个组可以用一对冒号来代替,因此,上面的地址现在可以写成:

8000::123:4567:89AB:CDEF

第三,IPv4 地址现在可以写成一对冒号再加上老式的点分十进制数,例如:



.,192.31.20.46

弄清楚 IPv6 有多少个地址可能并没有必要,但是,16 字节的地址确实非常多。具体来说的话,应该有  $2^{128}$  个 16 字节地址,近似等于  $3 \times 10^{38}$  个地址。如果整个地球,包括陆地和水面,都覆盖上计算机的话,那么,IPv6 将保证每平方米有  $7 \times 10^{25}$  个地址。化学系的学生将会注意到,这个数值超出了阿伏伽德罗常数(Avogadro's number)。IPv6 的目标并不是要为地球表面上的每一个分子都分配单独的 IP 地址,毕竟我们还不会走那么远。

在实践中,地址空间的使用效率不会非常高,就好像电话号码空间一样(曼哈顿(Manhattan)的区号 212 几乎已经用满了,而怀俄明州(Wyoming)的区号 307 几乎还是空的)。在 RFC 3194 中,Durand 和 Huitema 对此作了计算,他们利用电话号码分配方案作为参照,经计算得到的结果是,即使在最不利的情形下,地球表面(包括陆地和水面)上每平方米仍将有远远超过 1000 个 IP 地址。在任何可能的情形下,每平方米将有几万亿个 IP 地址。总而言之,在可预见的将来,我们不太可能用得完这些地址。

比较一下 IPv4(如图 5.53)与 IPv6 的头(如图 5.68),看看在 IPv6 中省掉了什么,这是非常有意义的。IHL 域不再出现了,因为 IPv6 头有固定的长度。协议(Protocol)域也被拿掉了,因为 Next header 域指明了最后的 IP 头后面跟的是什么(比如 UDP 或者 TCP 分段)。

所有与分段有关的域都被去掉了,因为 IPv6 采用另一种方法来实现分段的功能。首先,所有遵从 IPv6 的主机都应该能够动态地确定将要使用的数据报长度。由于有了这条规则,所以分段就变得不再有必要。而且,数据报长度的最小值也从 576 字节增加到 1280 字节,以便允许 1024 字节的数据和许多头信息。此外,当主机发送了一个非常大的 IPv6 分组时,如果路由器不能转发这么大的分组的话,它并不是对该分组进行分段,而是送回一条错误消息。路由器通过此消息告诉主机,所有将来发送给这一目标的分组都要分解得更小一些。从根本上来讲,让主机从一开始就发送合适大小的分组,比让沿途的路由器动态地对分组进行分段要有效得多。

最后,校验和(Checksum)域也被去掉了,因为计算校验和会极大地降低性能。现在往往使用的是可靠网络,而且数据链路层和传输层通常有它们自己的校验和,所以,在网络层上再使用校验和,相比它所付出的性能代价而言是不值得的。去掉了所有这些特性之后得到的是一个精简的网络层协议。因此,这份设计方案已经满足了 IPv6 的目标,即一个快速,但仍然灵活,并且具有足够大地址空间的协议。

#### 扩展头部

有些省略掉的 IPv4 域偶尔还会有用,所以,IPv6 引入了(可选的)扩展头(extension header)的概念。这些扩展头可以用来提供一些额外的信息,但是,它们的编码方式更加高效。现在已经定义了 6 种扩展头,如图 5.69 中所列。每一种扩展头都是可选的,但是,如果有多个扩展头出现的话,那么它们必须直接跟在固定头部的后面,而且最好使用表中列出的顺序。

有些扩展头有固定的格式,其他扩展头包含可变数目的可变长度域。对于所有这些可变项,每一项都被编码成一个(类型,长度,值)三元组。类型(Type)是一个单字节域,

它指明了这是哪个选项。Type 的值有特殊的选取方法,它的前 2 位告诉路由器应该如何处理此选项。选择方案有:跳过此选项;丢弃该分组;丢弃该分组并送回一个 ICMP 分组;与前一种选择相同,但是对于多播地址不送回 ICMP 分组(这样可以避免一个坏的多播分组产生大量的 ICMP 报告。)

扩展头	说明
逐跳选项	针对路由器的各种信息
目标选项	针对目标端的各种附加信息
路由	要访问的路由器列表(宽松的)
分段	数据报分段的管理
认证	验证发送方的身份
加密的安全净荷	有关加密内容的信息

图 5.69 IPv6 扩展头

长度(Length)也是一个单字节域,它说明了值域有多长(从 0 到 255 个字节)。值(Value)域是任何必要的信息,可以长达 255 个字节。

逐跳头(hop-by-hop header)中是一些沿途所有路由器必须要检查的信息。到现在为止,已经定义了一个选项:对超过 6 万 4 千的数据报的支持。该头的格式如图 5.70 所示。当使用这种扩展头的时候,固定头中的净荷长度(Payload length)域被设置为 0。

Next header (下一个头)	0	194	4
Jumbo payload length(超大的净荷长度)			

图 5.70 对于超大数据报(jumbogram)的逐跳扩展头

与所有的扩展头一样,逐跳扩展头的起始字节也指定了接下去是哪一种头。该字节之后的字节指示了当前逐跳扩展头有多长,其中不包括起始的 8 个字节,因为这 8 个字节是必须要有的。所有的扩展头都是以这种方式开始的。

接下去的两个字节表明了“该选项定义的是数据报的长度(代码 194)”,以及“长度值是一个 4 字节的数值”。最后 4 个字节给出了数据报的长度。小于 65 535 的长度值是不允许的,第一台路由器将会丢弃这种短于 65 535 的分组,并且送回一个 ICMP 错误消息。使用这种扩展头的数据报称为超大数据报(jumbogram),对于那些必须要通过 Internet 来传输千兆字节数据的超级计算机应用来说,超大数据报有非常重要的用途。

目标选项扩展头(destination options header)的用途是针对那些只需要在目标主机上被翻译的域。在 IPv6 的初始版本中,惟一定义的选项是空选项(null option),利用空选项可以将一个头拉长到 8 字节的倍数,所以,最初时候它将不会被使用。当初之所以将它包括进来,是为了确保新的路由软件和主机软件可以对它进行处理,也许有一天有人会想到一种新的目标选项。

路由扩展头列出了在通向目标的途中必须要访问的一台或者多台路由器。它非常类似于 IPv4 的宽松源路由机制,在宽松源路由机制中,凡是列出来的地址,必须要严格按顺序被访问到,但是,这些地址中间也可以经过一些没有列出来的其他路由器。路由头的格式如图 5.71 所示。

Next header (下一个头)	Header extension length (头扩展长度)	Routing type (路由类型)	Segments left (剩余段数)
与类型有关的数据			

图 5.71 路由扩展头

路由扩展头的前 4 个字节包含了 4 个单字节整数。下一个头(Next header)和头扩展长度(Header extension length)域如上面所述。路由类型(Routing type)域给出了该扩展头剩余部分的格式。类型为 0 则表示在第 1 个字后面是一个保留的 32 位字,然后是一定数量的 IPv6 地址。将来根据需要还可以发明其他的类型。最后,剩余段数(Segments left)域记录了在地址列表中还有多少个地址尚未被访问到。每次当一个地址被访问到的时候,该域中的数值减一。当它被减到 0 的时候,该分组就完全自由了,它不需要再遵循任何路由路径了。通常到这个时候它离目标已经非常接近,所以最佳路径也非常显然了。

分段扩展头(fragment header)涉及到与分段有关的事项,其处理方法与 IPv4 中的做法非常类似。该扩展头保存了数据报的标识符、分段号,以及有一位指明了后面是否还有更多的分段。然而,与 IPv4 中不同的是,在 IPv6 中,只有源主机才可以将一个分组进行分段。沿途的路由器可能不会进行分段。尽管这一变化是一个与过去截然不同的重要思路转折,但是,它简化了路由器的工作,使得路由过程更加高效。正如上面所提到的,如果路由器面临一个太大的分组,那么它可以丢弃该分组并且向源主机送回一个 ICMP 分组。这一信息将使源主机把该分组分割成小的片段(在这些小的片段中将需要使用分段扩展头),然后再试着重新发送。

认证扩展头(authentication header)提供了一种“让分组的接收方确定分组发送方身份”的机制。加密的安全净荷扩展头使得有可能加密一个分组的内容,所以,只有真正的接收方才可以读取分组中的内容。这两个扩展头使用密码学技术来完成它们的任务。

### 一些争议

由于 IPv6 的整个设计过程非常开放,并且许多参与的人都有自己强烈的建议,所以,IPv6 的许多选择都是非常有争议的,至少说起来是这样的,事实上这也毫不奇怪。下面我们简短地对这些争议做一番总结。要想了解全面的细节,请参考有关的 RFC 文档。

我们前面已经提到了有关地址长度的争论,最终的结果是一个折衷的方案: 16 字节固定长度的地址。

另一个争论发生在跳数限制(Hop limit)域的长度上。有一方强烈地认为,将最大跳数限制在 255 以内(也就是说,用一个 8 位的域)是一个很显然的错误。毕竟,32 跳的路径现在是非常普遍的,10 年以后比这更长的路径也会变得十分普遍。这些人争辩说,使

用大的地址长度是很有远见的,但是,使用小的跳计数器则非常短视。在他们眼里,计算机科学家能够犯的最大错误就是在有些地方提供太少的数据位。

他们得到的回应是,这样的意见一旦贯彻就可能要增加每一个域,从而导致一个非常臃肿的头部。而且,跳数限制域的功能是避免分组长时间地逗留在网络中,但 65 535 跳实在是太长了。而且,随着 Internet 的增长,越来越多的长距离链路将会建立起来,这将使得从一个国家到达另一个国家可能只要经过几跳就可以了。如果从源端和目标端分别到达它们相应的国际网关需要超过 125 跳的话,那么,它们的国家主干网一定有问题了。所以,最终 8 位支持者们赢得了胜利。

另一个热点问题是最大分组长度。超级计算机团体希望分组可以超过 64KB。当一台超级计算机开始传输数据的时候,它的业务就开始了,它当然不希望每 64KB 被中断一次。反对大分组的观点是,如果一个 1MB 的分组被送到一条 1.5Mbps 的 T1 线路上,那么,该分组将会占用线路 5 秒钟,对于共享同一线路的交互用户而言,这是一个明显感觉得到的延迟。在这一点上,最终达成一致的意见是:正常的分组被限制在 64KB 以内,但是允许使用逐跳扩展头来传送巨大数据报。

第三个热点话题是去掉 IPv4 校验和。有些人把这种做法比作从一辆汽车上拆除了刹车。拆除了刹车以后,汽车轻多了,所以,它可以跑得更快,但如果发生意外事件的话,你就有问题了。

校验和反对派的论点是,任何真正关心数据完整性的应用,不管用什么方法,一定要有一个传输层校验和,所以,在 IP 中使用另一个校验和(而且还有一个数据链路层的校验和)纯属多余。更进一步,经验表明,计算 IP 校验和是 IPv4 中一个主要的开销。反对校验和的阵营赢得了胜利,所以,IPv6 没有保留校验和。

移动主机也是一个争论的焦点。如果一台便携式计算机正在世界各地旅行,那么,它是继续使用同样的 IPv6 地址与目标进行通信,还是使用本地代理和外部代理的方案呢?移动主机也在路由系统中引入了非对称性。很可能有这样的情形,少量移动计算机可以很容易地听到一台大型固定路由器发出的强信号,但是,固定路由器听不清移动主机发出的微弱信号。因此,有人希望在 IPv6 中显式地对移动主机提供支持。由于在任何一个提案中都找不到大家一致的意见,所以最终这一努力以失败而告终。

可能最大的争论在于安全性。每个人都承认安全性非常重要,但是争议在于:在哪里和如何实现安全性。首先是在哪里实现安全性。支持在网络层上实现安全性的观点是,安全性变成一种标准的服务,于是所有的应用都可以使用安全服务,而无需任何提前计划。反对派则认为,真正的安全应用一般只需要端到端的加密,其中源应用完成加密过程,目标应用完成解密过程。最起码而言,网络层的实现可能会有错误,用户对此没有任何控制能力却要受到它的约束。对这种观点的回应是,这样的应用可以不使用 IP 的安全特性,改而自己实现端到端的安全性。对方又反驳道,那些不信任网络层正确性的人也不想为这一慢速而又庞大的 IP 实现付出额外的费用,尽管它的网络层安全功能已经被禁止了。

与“在哪里实现安全性”相关的另一个话题是,许多(并非全部)国家都有针对密码系统的严格出口法规。有些国家,特别是法国和伊拉克,也限制密码系统在国内的使用,如

此对于公安部门来说,老百姓就没有任何秘密了。因此,如果一个 IP 实现用到了一定强度的密码系统,那么,它就不可能从美国(和许多其他的国家)出口到全球的客户手中。为此,软件厂商不得不维护两套软件,一套给国内客户使用,另一套用于出口,这是许多计算机厂商强烈反对的事情。

有一点倒是大家都没有任何异议,那就是,没有人期望在某一个星期天早晨的时候 IPv4 Internet 被关闭,然后在星期一早晨 IPv6 Internet 又回来了。相反,刚开始的时候,一些 IPv6“孤岛”首先转变过来,它们通过隧道方式进行通信。随着 IPv6 岛的增加,它们将合并成更大的岛,最终所有的岛都合并到一起,于是 Internet 将被完全转变过来。由于在当前部署的 IPv4 路由器上有着巨大的投资,所以,这个转变过程可能需要 10 年的时间。由于这个原因,人们已经做了大量的努力来保证这个迁移过程尽可能地平稳。关于 IPv6 更多的信息,请参考(Loshin, 1999)。

## 5.7 本章小结

网络层向传输层提供服务,它既可以建立在虚电路基础之上,也可以建立在数据报基础之上。在这两种情形下,它的主要任务是将分组从源端传送到目标端。在虚电路子网中,路由决策是在建立虚电路的时候作出的;而在数据报子网中,路由决策是针对每一个分组而作出的。

在计算机网络中用到了许多路由算法。静态的算法包括最短路径路由算法和扩散算法。动态算法包括距离矢量路由算法和链路状态路由算法。大多数实际的网络使用其中某一个算法。有关路由的其他重要话题包括分级路由、移动主机的路由、广播路由、多播路由和对等网络中的路由等。

子网很容易变得拥塞起来,从而增加了分组的延迟,降低了分组吞吐量。网络设计者企图通过正确的设计来避免拥塞,他们用到的技术包括重传策略、缓存、流控制,等等。如果拥塞真的发生了,那么它必须要被处理,既可以采取送回抑制分组的方法,也可以采用脱落负载的方法,或者使用其他的方法。

拥塞控制再往前走一步,就应该考虑如何获得所承诺的服务质量。用于实现服务质量的方法包括客户端的缓冲、流量整形、资源预留,以及准入控制。专门被设计用于提供好的服务质量的几种途径包括综合服务(包括 RSVP)、区分服务和 MPLS。

不同的网络在很多方面有所不同,所以,当多个网络相互连接起来的时候,问题就来了。有时候,通过隧道方式让分组穿越一个异类网络,问题就可以得到解决,但是,如果源网络和目标网络属于不同类型的话,这种方法就会失败。当不同的网络具有不同的最大分组长度的时候,有可能需要用到分段机制。

Internet 有各种各样的与网络层相关的协议。其中既包括数据传输协议 IP,也包括控制协议 ICMP、ARP 和 RARP,以及路由协议 OSPF 和 BGP。Internet 很快就要用完 IP 地址了,所以,IETF 已经开发出 IP 的一个新版本协议 IPv6。



## 习 题

1. 请列举出两个适合于使用面向连接服务的计算机应用,再列举出两个最好使用无连接服务的计算机应用例子。

2. 请问有没有可能发生这样的情形:面向连接的服务也会(或者至少应该)以乱序的方式递交分组?请解释原因。

3. 数据报子网将每个分组当作独立的单位进行路由,所以每个分组的路由过程独立于其他所有的分组。虚电路子网不必采用这种方式,因为每个数据分组都沿着一条预先确定的路径向前传送。这是否意味着虚电路子网并不需要具备将独立的分组从任意的源端路由到任意目标端的能力呢?请解释你的答案。

4. 请给出三个在建立连接时有可能需要协商的协议参数的例子。

5. 请考虑以下涉及到实现虚电路服务的设计问题。如果在子网内部使用虚电路,那么,每个数据分组必须有一个3字节的头,每台路由器必须提供8字节的存储空间用于电路标识。如果子网内部使用数据报,那么,每个数据分组需要一个15字节的头,但是不要路由器的表空间。假设每一跳每 $10^6$ 字节的传输开销为1美分。快速路由器内存的价格是每字节1美分,2年以后就贬值了,这里假设每周的工作时间为40小时。平均每个会话的持续时间为1000秒,在这段时间中平均传输200个分组。平均每个分组要求4跳。请问哪种实现方法更加便宜,便宜多少?

6. 假设所有的路由器和主机都正常工作,并且它们的软件也都没有错误,请问一个分组被递交到错误目的地的可能性有没有(无论可能性有多小)?

7. 请考虑图5.7中的网络,但是忽略线路上的权值。假设它使用扩散法作为路由算法。如果一个从A发向D的分组的最大跳计数值为3,请列出它将要走的所有路径。同时也说明它需要消耗多少跳带宽。

8. 给出一个简单的试探算法,要求:在一个网络中,在指定的源和指定的目标之间找出两条路径,要求这两条路径在失去任何一条通信线路的情况下还能够幸存下来(假设存在这样的两条路径)。可以认为网络中的路由器足够可靠,所以不必担心路由器崩溃的可能性。

9. 考虑图5.13(a)中的子网。该子网使用了距离矢量路由算法,下面的矢量刚刚到达路由器C:来自B的矢量为(5,0,8,12,6,2);来自D的矢量为(16,12,6,0,9,10);来自E的矢量为(7,6,3,9,0,4)。经测量,到B、D和E的延迟分别为6、3和5。请问C的新路由表将会怎么样?请给出将使用的输出线路以及期望的延迟。

10. 假设在一个50台路由器的网络中,用8位数值记录延迟信息,并且每秒钟交换延迟矢量两次,请问,分布式路由算法需要在每条(全双工)线路上消耗多少带宽?假设每台路由器有三条线路连接到其他的路由器。

11. 在图5.14中,每一行上两组ACF位的布尔或(OR)是111。请问这仅仅是一种偶然情况,还是在所有情况下对子所有的子网都成立?

12. 对于4800台路由器的三层次分级路由,请问应该选择多大的区域和群才可以将



22. 一个数据报子网允许路由器在必要的时候丢弃分组。一台路由器丢弃一个分组的概率为  $p$ 。请考虑这样的情形：源主机连接到源路由器，源路由器连接到目标路由器，然后目标路由器连接到目标主机。如果任一台路由器丢掉了一个分组，则源主机最终会超时，然后再重试发送。如果主机至路由器以及路由器至路由器之间的线路都计为一跳，那么：

- (a) 一个分组每次传输中的平均跳数是多少？
- (b) 一个分组的平均传输次数是多少？
- (c) 每个接收到的分组平均要求多少跳？

23. 描述一下警告位方法和 RED 方法的两个主要区别。

24. 请说明为什么漏桶算法应该每个时钟滴答只允许一个分组，而不管分组的大小？

25. 在一个特殊的系统中用到了漏桶算法的字节计数变种算法。其规则是，每个时钟滴答可以发送一个 1024 字节的分组，或者两个 512 字节的分组，以此类推。请就这个系统说出一个本章正文中没有提到的严重限制。

26. 一个 ATM 网络使用令牌桶方案进行流量整形。每隔  $5\mu\text{s}$  一个新的令牌被放到桶中。每个令牌正好对应一个信元，每个信元包含 48 字节数据。请问最大可承受的数据速率为多少？

27. 在一个 6Mbps 的网络上，有一台主机通过一个令牌桶进行流量调整。令牌桶的填充速率为 1Mbps。初始时候它被填充到 8Mb 的容量。请问该计算机以 6Mbps 的全速率可以传输多长时间？

28. 想象这样一个流规范：最大分组长度为 1000 字节，令牌桶速率为每秒 10MB，令牌桶的大小为 1M 字节，最大传输速率为每秒 50MB。请问以最大速度传输的突发数据会持续多长时间？

29. 图 5.37 中的网络使用 RSVP 预留资源，主机 1 和主机 2 的多播树如图中所示。假设主机 3 请求一条带宽为 2MB/s 的信道用于接收主机 1 的流，以及一条带宽为 1MB/s 的信道用于接收主机 2 的流。同时，主机 4 请求一条带宽为 2MB/s 的信道用于接收主机 1 的流；主机 5 请求一条带宽为 1MB/s 的信道用于接收主机 2 的流。请问，在路由器 A、B、C、E、H、J、K 和 L 上，总共需要为这些请求预留多少带宽？

30. 一台路由器上的 CPU 每秒钟可以处理 2 百万个分组。提供给路由器的负载为每秒钟 1.5 百万个分组。如果从源端到目标端的路径上包含 10 台路由器，请问 CPU 花在排队和服务上的时间为多少？

31. 请考虑使用快速型转发方法的区分服务用户。是否可以保证快速型分组比常规的分组会经历更短的延迟？为什么是，或者为什么不是？

32. 在串联虚电路的互联网络中，需要分段机制吗？还是说只有在数据报系统中才需要分段机制？

33. 通过一个串联虚电路子网的隧道过程非常简单、直接：一端的多协议路由器只要建立一条通向另一端的虚电路，然后将分组通过虚电路传送过去即可。在数据报子网中也可以使用隧道技术吗？如果可以的话，请问该怎么做？

34. 假设主机 A 被连接到一台路由器 R1 上，R1 又连接到另一台路由器 R2 上，R2

被连接到主机 B。假定一条 TCP 消息包含 900 字节的数据和 20 字节的 TCP 头,现在该消息被传递给主机 A 的 IP 代码,请它递交给主机 B。请写出在三条链路上传输的每个分组中 IP 头部的 Total length、Identification、DF、MF 和 Fragment offset 域。假定链路 A-R1 可以支持的最大帧长度为 1024 字节,其中包括 14 字节的帧头;链路 R1-R2 可以支持的最大帧长度为 512 字节,其中包括 8 字节的帧头;链路 R2-B 可以支持的最大帧长度为 512 字节,其中包括 12 字节的帧头。

35. 一台路由器往外发送大量的总长度(数据+头)为 1024 字节的 IP 分组。假定这些分组生存 10 秒钟时间长,请问,路由器运行的最大线速度为多少才不至于发生 IP 数据报的 ID 编号空间重绕的危险?

36. 一个 IP 数据报使用了严格的源路由选项,现在它被分段了。你认为该选项应该被复制到每个分段中呢,还是只需放到第一个分段中就足够了? 请解释你的答案。

37. 假定最初的时候 B 类地址的网络部分并不是 16 位,而是 20 位。请问那将会有多少个 B 类网络?

38. 如果一个 IP 地址的十六进制表示为 C22F1582,请将它转换成点分十进制标记。

39. Internet 上一个网络的子网掩码为 255.255.240.0。请问它最多能够处理多少台主机?

40. 假定从 198.16.0.0 开始有大量连续的 IP 地址可以使用。现在 4 个组织 A、B、C 和 D 按照顺序依次申请 4000、2000、4000 和 8000 个地址。对于每一个申请,请利用 w. x. y. z/s 的形式写出所分配的第一个 IP 地址、最后一个 IP 地址,以及掩码。

41. 一台路由器刚刚接收到以下新的 IP 地址: 57.6.96.0/21、57.6.104.0/21、57.6.112.0/21 和 57.6.120.0/21。如果所有这些地址都使用同一条输出线路,那么,它们可以被聚集起来吗? 如果可以的话,它们被聚集到哪个地址上? 如果不可以的话,请问为什么?

42. 从 29.18.0.0 到 29.18.128.255 之间的 IP 地址集合已经被聚集到 29.18.0.0/17。然而,这里有一段空隙,即从 29.18.60.0 到 29.18.63.255 之间的 1024 个地址还没有被分配。现在这段空隙地址突然要被分配给一台使用不同输出线路的主机。请问是否有必要将聚集地址分割成几块,然后把新的地址块加入到路由表中,再看一看是否可以重新聚集? 如果没有必要这样做的话,那该怎么办呢?

43. 一台路由器的路由表中有以下的(CIDR)表项:

地址/掩码	下一跳
135.46.56.0/22	接口 0
135.46.60.0/22	接口 1
192.53.40.0/23	路由器 1
默认	路由器 2

对于下面的每一个 IP 地址,请问,如果一个到达分组的目标地址为该 IP 地址,那么路由器该怎么办?

- (a) 135.46.63.10
- (b) 135.46.57.14
- (c) 135.46.52.2
- (d) 192.53.40.7
- (e) 192.53.56.7

44. 许多公司有这样的策略：通过两台或者多台路由器将公司连接到 Internet 上，以提供一定的冗余，因而当其中一台路由器停机的时候系统还能工作。这种策略有可能与 NAT 一起工作吗？请解释你的答案。

45. 你刚刚向一个朋友解释了 ARP 协议。当你解释完之后，他说：“我明白了，ARP 给网络层提供了一项服务，所以它是数据链路层的一部分。”你该如何向他解释呢？

46. ARP 和 RARP 都将地址从一个空间映射到另一个空间。从这个角度而言，它们是相似的。然而，它们的实现却完全不同。它们最主要的差别在哪里？

47. 请描述一种在目标主机上重组 IP 分段的方法。

48. 大多数的 IP 数据报重组算法都有一个定时器，以避免因为一个丢失的分段而永久地牵制住多个重组缓冲区。假定一个数据报被分成了 4 个分段。前三个分段都到达了，但是，最后一个分段延迟了。最终，定时器超时了，接收方内存中的三个分段被丢掉。过了一会儿，最后一个分段突然来了。请问接收方该如何处理这个分段？

49. 在 IP 和 ATM 中，校验和仅仅作用在头部，而没有包括数据部分。你认为选择这种设计方案的理由是什么？

50. 有一个人生活在 Boston (波士顿)，现在她带着自己的便携式计算机去 Minneapolis (明尼阿波利斯) 旅游。让她惊讶的是，在 Minneapolis 目的地的 LAN 是一个无线 IP LAN，所以她根本不需要插网线。请问，是否仍然需要通过本地代理和外部代理这一整套过程才能正确地接收电子邮件或者其他的流量？

51. IPv6 使用 16 字节的地址。如果每隔  $1\text{ps}(10^{-12}\text{秒})$  就分配掉 1 兆个地址，那么，请问整个地址空间可以持续分配多久？

52. IPv4 头部中使用的 Protocol 域并没有出现在 IPv6 的固定头中。请问为什么？

53. 当 IPv6 协议被引入进来的时候，ARP 协议需要作改变吗？如果需要的话，这种改变是概念性的还是技术性的？

54. 编写一个程序来模拟扩散路由算法。每个分组应该包含一个计数器，在每一跳上该计数器减一。当计数器到达 0 的时候，该分组被丢弃。时间是离散的，每条线路在每个间隔中只处理一个分组。请完成该程序的三个版本：所有的线路都被扩散、除了进来的线路以外其他所有的线路都被扩散、只有最佳的  $k$  条线路（静态选择）才被扩散。请从延迟和所用带宽的角度，对扩散算法和确定性路由算法 ( $k=1$ ) 做一下比较。

55. 编写一个程序来模拟一个使用离散时间的计算机网络。每个时间间隔中，每个路由器队列中的第一个分组走一跳。每台路由器只有有限数量的缓冲区。如果一个分组到来的时候路由器没有缓冲区空间了，那么它将被丢弃，并且不再重传。同时，另有一个端到端协议，它完全支持超时和确认分组，在该协议中，最终源路由器会重新生成丢弃的分组。请打印出网络的吞吐量与端到端超时间隔之间的函数关系，这里错误率是一个



参数。

56. 编写一个函数来完成 IP 路由器中的转发过程。该函数有一个 IP 地址参数。它也要访问一张全局表,全局表由许多三元组构成。每个三元组包含三个整数:一个 IP 地址、一个子网掩码和所用的输出线路。该函数利用 CIDR 在表中查找由参数指定的 IP 地址,然后返回对应的输出线路值。

57. 使用 `tracert`(UNIX 环境)或者 `tracert`(Windows 环境)程序跟踪一下从你的计算机到其他各洲的大学的路由路径。你将可以发现一些跨越大洋的链路。以下有些站点你可以试一试。

`www.berkeley.edu` (California, 美国的加利福尼亚州)

`www.mit.edu` (Massachusetts, 美国的马萨诸塞州)

`www.vu.nl` (Amsterdam, 荷兰的阿姆斯特丹)

`www.ucl.ac.uk` (London, 英国的伦敦)

`www.usyd.edu.au` (Sydney, 澳大利亚的悉尼)

`www.u-tokyo.ac.jp` (Tokyo, 日本的东京)

`www.uct.ac.za` (Cape Town, 南非的开普敦)

## 第6章 传 输 层

传输层不仅仅是另外一层,它是整个协议层次的核心所在。它的任务是在源机器和目标机器之间提供可靠的、性价比合理的数据传输功能,并且与当前所使用的物理网络完全独立。如果没有传输层,那么分层协议的整个概念将变得没有意义。在本章中,我们将详细地学习传输层,包括传输服务、设计、协议和性能。

### 6.1 传输服务

在本节中我们将介绍传输服务。我们首先看一下向应用层提供的是什么类型的服务。为了使传输服务的问题更加具体化,我们将讨论两个传输层原语集合。第一个比较简单(假想的),它说明了最基本的思想;第二个是在 Internet 中常用的接口。

#### 6.1.1 向上层提供的服务

传输层的最终目标是向它的用户(通常是应用层中的进程)提供高效的、可靠的和性价比合理的服务。为了实现这个目标,传输层需要充分利用网络层提供给它的服务。在传输层内部,完成这项工作的硬件和/或软件称为**传输实体(transport entity)**。传输实体可能位于操作系统的内核,或者在一个独立的用户进程中,或者以一个链接库的形式被绑定到网络应用中,或者位于网络接口卡上。网络层、传输层和应用层之间的逻辑关系如图 6.1 所示。

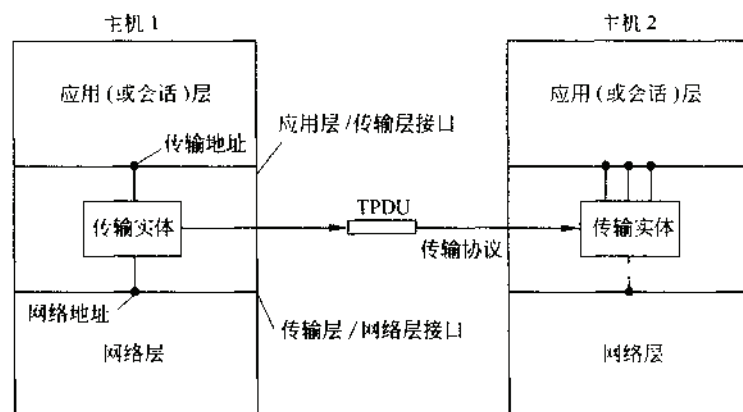


图 6.1 网络层、传输层和应用层

如同网络层有两种不同类型的服务(面向连接的和无连接的)一样,传输服务的类型也有两种。面向连接的传输服务在许多方面与面向连接的网络服务类似。在这两者之中,连接有三个阶段:建立连接、数据传输和释放连接。在这两个层上,编址和寻址以及流控制也是类似的。另外,无连接的传输服务与无连接的网络服务也非常相似。

于是,一个很显然的问题是:既然传输层服务与网络层服务如此相似,那为什么还要两个独立的层呢?为什么用一层还不够呢?问题的答案有点微妙,但非常关键,为了看清楚要点所在,我们回到图 1.9。传输层的代码完全运行在用户的机器上,但是网络层主要运行在由承运商控制的路由器上(至少对于广域网是如此)。如果网络层提供的服务不够用,那会怎么样呢?如果它频繁地丢失分组该怎么办?如果路由器时常崩溃又该怎么办呢?

问题发生了,这就是答案所在。用户在网络层上并没有真正的控制权,所以它们不可能用最好的路由器或者在数据链路层上用更好的错误处理机制来解决服务太差的问题。惟一的可能是在网络层之上的另一层中提高服务质量。如果在一个面向连接的子网中,一个传输实体在执行一个长时间的传输过程时,突然接到通知说它的网络层连接已经被意外终止了,而且也不知道当前正在传输的那些数据到底怎么样,那么,该传输实体可以与远程的传输实体建立起新的网络层连接。利用新建立的连接,它可以向对方发送一个查询请求,询问哪些数据已经到达,哪些数据还没有到达,然后从中断的地方开始继续向对方发送数据。

从本质来讲,由于传输层的存在,这使得传输服务有可能比网络服务更加可靠。丢失的分组和损坏的数据可以在传输层上检测出来,并且由传输层来补偿。而且,传输服务原语可以通过调用库过程(函数)来实现,从而使得这些原语独立于网络服务原语。在不同的网络之间,网络服务原语可能有很大的差别(比如,无连接的 LAN 服务可能完全不同于面向连接的 WAN 服务)。将网络服务隐藏在一组传输服务原语的背后,其好处是,一旦改变了网络服务,只要求替换一组库过程即可,新的库过程使用了不同的底层网络服务,但是实现了同样的传输服务原语。

感谢传输层,由于有了传输层,应用开发人员可以根据一组标准的原语来编写代码,而且他们的程序有可能运行在各种各样的网络上,他们不用处理不同的子网接口,也不用担心不可靠的传输过程。如果所有实际的网络都是完美无缺的,并且具有相同的服务原语,也保证不会发生变化,那么传输层可能就不再需要了。然而,在现实世界中,传输层承担了将子网的技术、设计和各种缺陷与上层隔离的关键作用。

由于这个原因,许多人习惯于将网络的层分成两部分:第 1 层至第 4 层为一部分,第 4 层之上为另一部分。底下的 4 层可以被看作**传输服务提供者**(**transport service provider**),而上面的层则是**传输服务用户**(**transport service user**)。这种“提供者-用户”的区分对于层的设计有重要的影响,同时也把传输层放到了一个关键的位置上,因为它构成了可靠数据传输服务的提供者和用户两者之间的主要边界。

### 6.1.2 传输服务原语

为了让用户访问传输服务,传输层必须为应用程序提供一些操作,也就是说,提供一

个传输服务接口。每个传输服务都有它自己的接口。在这一小节中,我们首先介绍一个简单的(假想的)传输服务以及它的接口,以此来看清楚传输服务的本质所在。在下一小节我们将介绍一个实际的例子。

传输服务类似于网络服务,但是两者之间也有一些重要的区别。最主要的区别是,网络服务企图如实地按照实际网络所提供的服务来建立模型。由于实际网络可能会丢失分组,所以网络服务一般来说是不可靠的。

与此相反,(面向连接的)传输服务是可靠的。当然,实际的网络并非没有错误,但是,这正好是传输层的目标:在不可靠的网络上提供可靠的服务。

举一个例子,请考虑 UNIX 中通过管道连接起来的两个进程。它们假设两者之间的连接是完美的。它们并不想知道有关确认、分组丢失、拥塞以及诸如此类的事情。它们想要的是一个百分之百可靠的连接。进程 A 把数据放进管道的一端,进程 B 可以从另一端将数据取出来。这就是面向连接的传输服务的真正含义所在——将网络服务的各种缺陷隐藏起来,因而用户进程只要假设存在一个无错误的位流就可以了。

另一方面,传输层也可以提供不可靠(数据报)服务。然而,相对来说关于这种服务并没有太多内容可说,所以在本章中我们将注意力主要集中在面向连接的传输服务上。不过,有一些应用,比如客户-服务器模型的计算和流式多媒体应用,很大程度上得益于无连接的传输服务,所以,后面我们还是会介绍到这种服务。

网络服务和传输服务之间的第二个区别是它们针对的目标(即它们的用户)不同。网络服务仅仅被传输实体使用。通常用户不会编写自己的传输实体,因此,很少有用户或者程序会看到完全裸露的网络服务。相反,许多程序(和程序员)可以看到传输原语。因此,传输服务必须用起来非常方便、容易。

为了初步了解一下传输服务的基本面貌,请考虑图 6.2 中列出的 5 个原语。这个传输接口是非常精简的,但是它给出了一个面向连接的传输接口应该要完成的一些本质工作。它使得应用程序可以建立并使用连接,用完之后再释放连接,对于许多应用来说这已经足够了。

原语	发送的分组	含义
LISTEN	(无)	阻塞,直到有某个进程试图与它建立连接
CONNECT	CONNECTION REQ	主动地尝试建立一个连接
SEND	DATA	发送信息
RECEIVE	(无)	阻塞,直到一个 DATA 分组到来
DISCONNECT	DISCONNECTION REQ	这一端希望释放一个已经建立的连接

图 6.2 一个简单传输服务的原语

为了看清楚这些原语的可能用法,请考虑这样一个应用:它支持一个服务器和多个远程客户。首先,服务器执行 LISTEN 原语,一般的做法是调用一个库过程,由它执行一个系统调用,并且阻塞该服务器,直到有客户来连接。当一个客户希望与该服务器进行通话的时候,它执行 CONNECT 原语。在 CONNECT 原语中,传输实体阻塞调用方,并且

给服务器发送一个分组。封装在该分组净荷中的是一条发送给服务器传输实体的传输层消息。

现在我们快速地说明一下关于术语的用法。由于缺乏更好的术语,所以,我们不得不使用比较难看的缩略词 **TPDU**(**Transport Protocol Data Unit**, 传输协议数据单元)来代表从一个传输实体发送至另一个传输实体的消息。因此,TPDU(传输层之间交换的数据单元)被包含在分组(网络层之间交换的数据单元)的内部。而分组则被包含在帧(数据链路层之间交换的数据单元)的内部。当一帧到达的时候,数据链路层对帧头进行处理,然后把帧的净荷域中的内容传递给网络实体。网络实体对分组头进行处理,然后把分组的净荷域向上传递给传输实体。这种嵌套关系如图 6.3 所示。

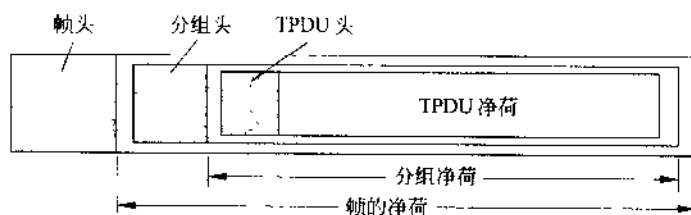


图 6.3 TPDU、分组和帧的嵌套关系

现在回到我们的客户-服务器例子上来,客户的 **CONNECT** 调用将使得传输实体发送一个 **CONNECTION REQUEST TPDU** 给服务器。当这个 TPDU 到达服务器端的时候,服务实体执行检查,看服务器是否正被阻塞在 **LISTEN** 调用中(即对于进来的连接请求很有兴趣)。然后,它解除服务器的阻塞,并且给客户送回一个 **CONNECTION ACCEPTED TPDU**。当这个 TPDU 到达客户端的时候,客户也被解除阻塞,于是连接建立起来了。

现在双方就可以通过 **SEND** 和 **RECEIVE** 原语交换数据了。在最简单的形式中,任何一方都可以执行(阻塞的)**RECEIVE** 原语,以等待另一方执行 **SEND** 原语。当 TPDU 到来的时候,接收方被解除阻塞。然后它可以对这个 TPDU 进行处理,并发送一个应答。只要双方对于发送数据的顺序有统一的认识,则这种方案可以工作得很好。

请注意,在传输层上,即使一个非常简单的单向数据交换过程也会比网络层上的交换过程复杂得多。发送的每一个数据分组(最终)都要被确认。携带控制 TPDU 的分组也要被确认,无论是隐式方式还是显式方式。这些确认是由传输实体使用网络层协议来管理的,它们对于传输用户是不可见的。类似地,传输实体也需要关心定时器和重传的问题。这些机制对于传输用户全部不可见。对于传输用户而言,一个连接就是一个可靠的位管道:一个用户在一端将位数据塞进去,然后这些位数据就会神奇地出现在另一端。正是这种隐藏复杂性的能力才使得分层协议成为如此功能强大的工具。

当不再需要一个连接的时候,传输用户必须将它释放,以便使两个传输实体内部的表空间有机会被重新使用。释放连接有两种方式:非对称的和对称的。在非对称方式中,任何一方都可以执行 **DISCONNECT** 原语,从而导致它的传输实体将一个 **DISCONNECT TPDU** 发送给远程的传输实体。另一端收到该 TPDU 之后,连接就被释



放了。

在对称方式中,连接的两个方向彼此独立,每个方向需要单独被关闭。当一方执行 DISCONNECT 的时候,这意味着它不再需要发送数据了,但是它仍然希望接受对方发送过来的数据。在这种模型中,只有当双方都执行了 DISCONNECT 的时候,一个连接才真正被释放。

图 6.4 给出了一个用这些简单的原语来建立和释放连接的状态图。每一个迁移都是由某一种事件触发的:或者是本地的传输用户执行了一个原语,或者是接收到了一个分组。为了简单起见,这里假定每一个 TPDU 都是单独确认的。我们也假定采用了对称的连接释放模型,并且由客户先释放连接。请注意,这种模型比较简单,后面我们将会讨论一些更加实际的模型。

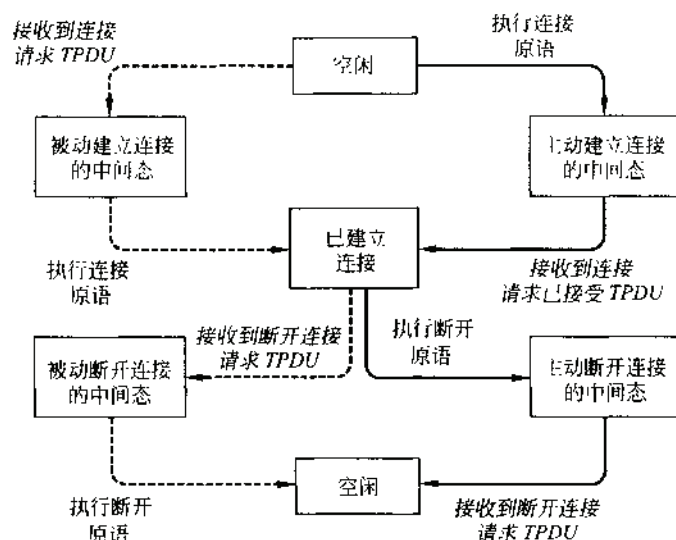


图 6.4 一个简单的连接管理方案  
因分组到达而引起的迁移用斜体字标注  
实线显示了客户的状态序列;虚线显示了服务器的状态序列

### 6.1.3 Berkeley Socket(伯克利套接字)

现在我们简短地看一看另一组传输原语,即 Berkeley UNIX 中使用的 TCP socket (套接字)原语。这些原语被广泛应用于 Internet 程序设计中。图 6.5 列出了这些原语。粗略地讲,它们符合我们前面的第一个例子中的模型,但是提供了更多的特性和灵活性。这里我们不再介绍对应的 TPDU。关于这些 TPDU,要等到本章后面我们学习 TCP 的时候再作讨论。

表中列出的前 4 个原语由服务器按照顺序执行。SOCKET 原语创建一个新的端点(end point),并且在传输实体中为它分配相应的表空间。此调用的参数规定了以后将会用到的地址格式、期望的服务类型(比如可靠的字节流),以及所用的协议。SOCKET 调

用成功以后,它返回一个普通的文件描述符,以便在后续的调用中使用,所以,SOCKET 调用与 OPEN 调用的工作方式一样。

原语	含义
SOCKET	创建一个新的通信端点
BIND	将一个本地地址关联到一个套接字上
LISTEN	宣布愿意接受连接;给出队列大小
ACCEPT	阻塞调用方,直到有人企图连接上来
CONNECT	主动地尝试建立一个连接
SEND	在指定的连接上发送数据
RECV	从指定的连接中接收数据
CLOSE	释放指定的连接

图 6.5 TCP 的套接字原语

新创建的套接字并没有网络地址。通过 BIND 原语可以为套接字分配地址。一旦服务器已经为一个套接字绑定了一个地址,则远程客户就能够与它建立连接。之所以不让 SOCKET 调用直接创建一个地址是因为有些进程对于它们的地址比较在意(比如,它们多年来一直使用同样的地址,所以每个人都知道它们的地址),而其他的进程并不在意。

接下来是 LISTEN 调用,它分配一定的空间以便对进来的连接请求进行排队,因此多个客户可以同时发起连接请求。与我们第一个例子中的 LISTEN 不同的是,套接字模型中的 LISTEN 并不是一个阻塞调用。

为了阻塞等待一个进来的连接,服务器执行 ACCEPT 原语。当一个请求连接的 TPDU 到来的时候,传输实体创建一个新的套接字并返回一个与其关联的文件描述符,这个新的套接字与原来的套接字具有同样的属性。然后服务器可以复制一个进程或者线程来处理这个新套接字上的连接,而服务器自身又回去继续等待原套接字上的下一个连接。ACCEPT 返回一个普通的文件描述符,所以,服务器可以按照标准的方式,好像访问文件一样对它进行读或者写操作。

现在我们来看一看客户端的情形。同样地,这里首先也使用 SOCKET 原语来创建一个套接字,但是由于服务器并不关心它所用的地址,所以客户端不必调用 BIND 原语。CONNECT 原语阻塞调用方,并且主动发起连接过程。当 CONNECT 调用完成(即接收到服务器发送过来的正确的 TPDU)的时候,客户进程被解除阻塞,于是连接就建立起来了。现在双方都可以使用 SEND 或者 RECV,在新建立起来的全双工连接上发送或者接收数据。如果 SEND 和 RECV 调用不要求特殊选项的话,服务器或者客户也可以使用标准的 UNIX 系统调用 READ 和 WRITE。

在套接字模型中,连接的释放是对称的。当双方都执行了 CLOSE 原语之后,连接就被释放了。

#### 6.1.4 套接字程序设计的例子：一个 Internet 文件服务器

本小节我们介绍一个例子来说明套接字各个函数的用法,请考虑图 6.6 中所示的客户和服务代码。这里我们有一个非常简单的 Internet 文件服务器和一个与该服务器配合使用的例子客户。这份代码有许多限制(后面将会提到),但原则上,服务器代码可以被编译成二进制代码,并且在任何连接到 Internet 的 UNIX 系统上运行。然后,客户代码也可以被编译,并且可在 Internet 上任何地方的其他 UNIX 机器上运行。客户代码在执行的时候需要正确的参数,以便获取到服务器机器上的任何文件,前提条件是服务器本身必须能够访问这些文件。客户获取到的文件被送到标准输出,当然,你可以将标准输出重定向到某一个文件或者一个管道。

我们首先来看服务器的代码。在开始处它包含一些标准的头文件,其中后三个头文件包含了一些主要的、与 Internet 有关的定义和数据结构。接下来是一个宏定义,将 SERVER\_PORT 定义成 12345。这个数值是任意选取的。1024 至 65 535 之间的任何一个数值都可以正确地工作,只要其他的进程没有用到它即可。当然,客户和服务必须使用同样的端口。如果该服务器提供的服务变成了全球范围内引人注目的亮点服务(不过不太可能,这个服务器太简单了),那么它将会被分配一个小于 1024 的永久端口号,并且出现在 [www.iana.org](http://www.iana.org) 上。

服务器代码中接下来两行定义了两个用到的常数。第一个常数决定了在文件传输过程中数据块的大小。第二个常数决定了允许多少个连接请求在排队等待处理,一旦连接请求到达这个数目以后,再进来的连接请求将被丢弃。

在声明了局部变量之后,服务器代码开始执行。首先它对一个用来存放服务器 IP 地址的数据结构进行初始化。该数据结构很快将被绑定到服务器的套接字中。这里的 `memset` 调用将这个数据结构设置为全 0。紧随其后的三条赋值语句分别填充该数据结构的三个域。其中第三个域包含了服务器的端口。函数 `htonl` 和 `htons` 是必要的,它们将参数中的值转换成一种标准格式,所以,该服务器的代码既可以运行在 big-endian 字节序的机器(比如 SPARC)上,也可以运行在 little-endian 字节序的机器(比如 Pentium)上。它们的确切语义我们这里并不关心。

接下来服务器创建一个套接字,并且检查是否出错(通过比较 `s<0` 来确定)。在产品版本的服务器代码中,错误消息可能包含更多的说明信息。`setsockopt` 调用是必要的,它允许这个端口可以被重复使用,所以服务器能够永久地运行,处理一个又一个请求。现在,IP 地址被绑定到套接字中,然后检查 `bind` 调用是否成功。初始化过程中的最后一步是调用 `listen`,这样就宣告了本服务器愿意接受进来的调用,并告诉系统,当服务器在处理请求时如果有新的请求到来的话,请系统保留多达 `QUEUE_SIZE` 个请求。如果队列满了之后又有新的请求到来,则直接将后来的请求丢弃即可。

到这个点上,服务器进入它的主循环,这是一个永不退出的循环。终止服务器的惟一做法是从外部将服务器杀死(kill)。`accept` 调用阻塞服务器,直到某个客户试图与它建立连接为止。如果 `accept` 调用成功,则它返回一个文件描述符,利用该文件描述符可以进行读写操作,就好像利用文件描述符从管道读写数据一样。然而,管道是单向的,与此不

同的是,套接字是双向的,所以,sa(socket address,套接字地址)既可以被用来从连接中读取数据,也可以被用来向连接写数据。

连接被建立起来之后,服务器从连接中读取文件名。如果文件名还没有被送过来,则服务器阻塞住,以等待文件名的到来。服务器获得了文件名之后,它打开该文件,然后进入一个循环,在循环体中它交替地从文件中读取数据块并且将数据块写到套接字中,这个过程一直持续到整个文件被复制完成为止。然后,服务器关闭文件和连接,并等待下一个连接的到来。它无限地重复这个循环。

现在我们来看一看客户代码。为了理解客户代码的工作过程,首先有必要理解客户程序被调用的方式。假设客户程序被称为 client,那么,一个典型的调用如下:

```
client flits.cs.vu.nl /usr/tom/filename >f
```

只有当服务器程序已经在 flits.cs.vu.nl 机器上运行,并且该机器上确实存在 /usr/tom/filename 文件,而且服务器程序对该文件有读访问权限的时候,上面这个 client 调用才能生效。如果这个调用成功的话,那么,服务器上的文件将通过 Internet 被传递过来,并且写到客户本地的 f 文件中,然后客户程序退出。由于在一次传输之后,服务器仍然在运行,所以,客户可以再次启动,以获取其他的文件。

客户代码首先是一些包含语句和声明。在开始执行的地方,它先判断自己在被用户调用的时候是否有正确的参数个数(这里 argc=3 代表了程序名加上两个参数)。请注意,argv[1]包含了服务器的名字(比如 flits.cs.vu.nl),通过 gethostbyname 它被转换为一个 IP 地址。这里的函数 gethostbyname 使用 DNS 来查询机器名。我们将在第 7 章学习 DNS。

接下来客户创建一个套接字并执行一些初始化工作。之后,客户使用 connect 函数,企图与服务器建立一个 TCP 连接。如果在指定名字的机器上,服务器程序已经启动和运行了,并且它被绑定到 SERVER\_PORT 端口上,而且服务器当前是空闲的,或者在 listen 队列中还有空间,那么,连接(最终)将被建立起来。客户利用该连接可以将文件的名字发送过去,做法很简单,只要在套接字上执行写操作即可。发送的字节数是名字长度加 1,因为名字尾部的 0 字节也必须被发送过去,以便告诉服务器文件名在哪里结束。

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345 /* arbitrary, but client & server must
    agree */
#define BUF_SIZE 4096 /* block transfer size */
```

图 6.6 使用套接字的客户代码和服务端代码

```

int main(int argc, char * * argv)
{
    int c, s, bytes;
    char buf[BUF_SIZE];           /* buffer for incoming file */
    struct hostent * h;           /* info about server */
    struct sockaddr_in channel;   /* holds IP address */

    if (argc != 3) fatal("Usage: client server-name file name");
    h = gethostbyname(argv[1]);   /* look up host's IP address */
    if (! h) fatal("gethostbyname failed");

    s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s < 0) fatal("socket");
    memset(&channel, 0, sizeof(channel));
    channel.sin_family = AF_INET;
    memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
    channel.sin_port = htons(SERVER_PORT);

    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");

    /* Connection is now established. Send file name including 0 byte at end. */
    write(s, argv[2], strlen(argv[2]) + 1);

    /* Go get the file and write it to standard output. */
    while (1) {
        bytes = read(s, buf, BUF_SIZE); /* read from socket */
        if (bytes <= 0) exit(0);         /* check for end of file */
        write(1, buf, bytes);           /* write to standard output */
    }

    fatal(char * string)
    {
        printf("%s\n", string);
        exit(1);
    }

    #include <sys/types.h>           /* This is the server code */
    #include <sys/fcntl.h>
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <netdb.h>

```

图 6.6 (续)



```

#define SERVER_PORT 12345                /* arbitrary, but client & server must
    agree */
#define BUF_SIZE 4096                    /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char * argv[])
{
    int s, b, l, fd, sa, bytes, on = 1;
    char buf[BUF_SIZE];                  /* buffer for outgoing file */
    struct sockaddr_in channel;           /* holds IP address */

    /* Build address structure to bind to socket. */
    memset(&channel, 0, sizeof(channel)); /* zerochannel */
    channel.sin_family = AF_INET;
    channel.sin_addr.s_addr = htonl(INADDR_ANY);
    channel.sin_port = htons(SERVER_PORT);

    /* Passive open. Wait for connection. */
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* createsocket */
    if (s < 0) fatal("socket failed");
    setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

    b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
    if (b < 0) fatal("bind failed");

    l = listen(s, QUEUE_SIZE);            /* specify queue size */
    if (l < 0) fatal("listen failed");
    /* Socket is now set up and bound. Wait for connection and process it. */
    while (1) {
        sa = accept(s, 0, 0);             /* block for connection request */
        if (sa < 0) fatal("accept failed");

        read(sa, buf, BUF_SIZE);          /* read file name from socket */

        /* Get and return the file. */
        fd = open(buf, O_RDONLY);          /* open the file to be sent back */
        if (fd < 0) fatal("open failed");

        while (1) {
            bytes = read(fd, buf, BUF_SIZE); /* read from file */
            if (bytes <= 0) break;           /* check for end of file */
            write(sa, buf, bytes);          /* write bytes to socket */
        }
        close(fd);                        /* closefile */
    }
}

```

图 6.6 (续)

```

        close(sa);                                /* close connection */
    }
}

```

图 6.6 (续)

现在客户进入了一个循环,它逐个数据块地将文件从套接字中读出来,再复制到标准输出上。完成这个过程之后,它就退出了。

过程 fatal 打印出一条错误消息,然后退出程序。服务器代码也需要同样的过程,但是考虑到页面空间紧张,所以它被省略掉了。由于客户和服务器的代码是单独编译的,而且通常运行在不同的计算机上,所以它们并不共享同一份 fatal 代码。

从本书的 Web 站点上读者可以获取到这两个程序(以及与本书有关的其他一些材料),做法是,在下面的页面上:

<http://www.prenhall.com/tanenbaum>

单击封面旁边的 Web 站点链接。将这两个程序下载下来之后,你可以在任何一个 UNIX 系统(比如 Solaris、BSD、Linux)上进行编译,做法是:

```

cc -o client client.c -lsocket -lnsl
cc -o server server.c -lsocket -lnsl

```

服务器的启动方法很简单,只要输入:

```
server
```

即可。客户需要两个参数,正如前面所讨论的。在 Web 站点上也有一个 Windows 的版本可供使用。

说明一下,作为一个服务器,本小节中的服务器程序并不完善。它的错误检查还很欠缺,错误报告也很一般。很显然,它还没有涉足到安全性,而且,由于它使用了 UNIX 的系统调用,所以它还不是平台无关的。同时它也作了某些技术上不合理的假设,比如假设文件名一定适合缓冲区(即不会超出缓冲区的大小),并且可以以原子方式传送出去。由于它按照严格的顺序关系来处理所有的请求(因为它只有一个线程),所以它的性能很差。尽管存在这些缺点,但它确实是一个完整的、可以工作的 Internet 文件服务器。在本章的练习中,希望读者对它进行改进。有关套接字程序设计更多的信息,请参考(Stevens, 1997)。

## 6.2 传输协议的要素

传输服务是通过两个传输实体之间所使用的一个传输协议来实现的。传输协议在有些方面非常类似于我们在第 3 章中学习过的数据链路协议。这两种协议都要处理错误控制、顺序管理和流控制,以及其他一些问题。

然而,两者之间也存在一些重要的差别。这些差别是由于这两种协议运行的环境有很大的差异而引起的,如图 6.7 所示。在数据链路层上,两台路由器直接通过一条物理信

道进行通信,而在传输层上,此物理信道被整个子网所替代。这种环境差异对于协议而言有许多重要的影响,本章后面我们将会看到。

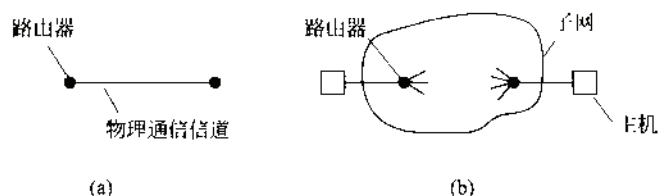


图 6.7

(a) 数据链路层的环境; (b) 传输层的环境

首先,在数据链路层中,路由器不必指定它要与哪一台路由器进行通话——每条输出线路惟一指定了一台专门的路由器。而在传输层中,要求显式地指定目标的地址。

其次,在图 6.7(a)中的线上建立一个连接的过程非常简单:另一端总是在那里(除非它崩溃了才不在那里)。两头都不需要做很多事情。而在传输层中,初始的连接建立过程非常复杂,后面我们将会看到。

数据链路层和传输层之间另一个非常恼人的差别是,子网中可能有一定的存储容量。当路由器发送一帧时,该帧可能到达,也可能丢失,但是它不可能先蹦跶一会儿,再躲到远处一个角落中,然后 30 秒之后在一个不合适的时刻又突然冒出来。如果子网使用了数据报,并且内部采用了自适应路由算法,则一个分组先被保存几秒钟,然后再被递交出去的情形不仅是可能的,而且是很常见的。子网的这种保存分组的能力有时候会导致灾难性的后果,因而要求采用某些特殊的协议。

数据链路层和传输层之间最后一个差别是数量的差别,而并非质的差别。这两种协议都要求缓冲和流控制,但是,由于传输层上往往会出现大量动态变化的连接,所以,传输层需要使用一种不同于数据链路层中使用的方法。在第 3 章中,有些协议为每条线路分配了固定数量的缓冲区,所以,当一帧到达的时候,总是有缓冲区可用。在传输层中,由于必须要管理大量的连接,因此,为每条线路分配很多专用缓冲区的思路不再有吸引力。在本节中,我们将讨论所有这些以及其他的重要问题。

### 6.2.1 编址

当一个应用(比如一个用户)进程希望与另一个远程的应用进程建立连接的时候,它必须指定要连接到哪个应用进程上(无连接的传输过程也有同样的问题:每个消息被发送给谁呢?)。通常使用的方法是为那些能够监听连接请求的进程定义相应的传输地址。在 Internet 上,这些端点称为端口(port)。在 ATM 网络中,它们称为 AAL-SAP。我们将使用一般化的术语 TSAP(Transport Service Access Point, 传输服务访问点)。同样地,网络层上类似的端点(即网络层地址)称为 NSAP(Network Service Access Point, 网络服务访问点)。IP 地址是 NSAP 的特例。

图 6.8 显示了 NSAP、TSAP 和传输连接之间的关系。应用进程(包括客户和服务

器)可以将自己关联到一个 TSAP 上,以便与远程的 TSAP 建立连接。如图所示,这些连接需要途径每台主机上的 NSAP。采用 TSAP 的目的是,在有些网络中,每台计算机只有一个 NSAP,但是可能有多个传输端点共享此 NSAP,所以它需要某一种方法来区分这些传输端点。

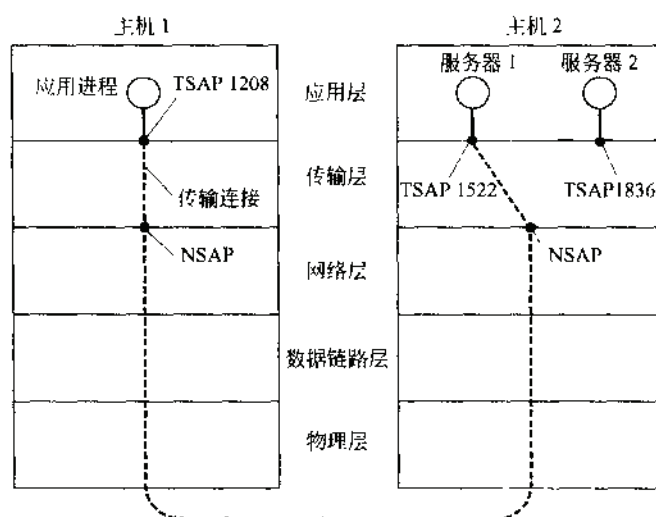


图 6.8 TSAP、NSAP 和传输连接

针对传输连接,一种可能的场景如下所述:

(1) 主机 2 上的时间服务器进程将自己关联到 TSAP 1522 上,以等待进来的连接请求。一个进程如何将自己关联到一个 TSAP 上,这不属于网络模型的范畴,完全取决于本地的操作系统。例如,用我们的 LISTEN 调用就可以做到这一点。

(2) 主机 1 上的应用进程希望知道当前的时间,所以它发出一个 CONNECT 请求,同时指定 TSAP 1208 作为源,TSAP 1522 作为目标。这个动作最终导致了在主机 1 的应用进程和主机 2 的服务器 1 之间建立一个传输连接。

(3) 然后应用进程发送一个请求,希望知道当前的时间。

(4) 时间服务器进程用当前的时间作为响应。

(5) 然后,传输连接被释放。

请注意,在主机 2 上很可能还有其他的服务器被关联到其他的 TSAP 上,它们也在等待经过同一个 NSAP 进入的连接请求。

上面描绘的场景非常美好,但是我们忽略了一个细微的问题:主机 1 上的用户进程如何知道时间服务器被关联到 TSAP 1522 上?一种可能是,很多年以来时间服务器一直被关联在 TSAP 1522 上,渐渐地所有的网络用户都知道了这个信息。在这个模型中,具有固定 TSAP 地址的服务被罗列在一些知名的文件中,比如 UNIX 系统上的 /etc/services 文件,该文件列出了哪些服务器被永久地关联到哪些端口上。

然而,固定的 TSAP 地址仅仅对于少数永不改变的关键服务(比如 Web 服务器)才适合,一般而言,用户进程通常需要跟另一个只存在较短时间的用户进程通话,而且后者往

往没有提前预知的 TSAP 地址。更进一步,如果一台机器上有大量潜在的服务器进程,但是大多数服务器进程又很少会被使用,那么,让每个服务器进程都主动地、全天候地监听一个 TSAP 地址将是非常浪费资源的。简而言之,这里需要一种更好的方案。

这样一种方案的简化形式如图 6.9 所示。它被称为初始连接协议(initial connection protocol)。它的做法并不是让每一个可对外提供服务的服务器都在一个知名的 TSAP 上监听,相反,每一台愿意为远程用户提供多个服务的机器都使用一个特殊的进程服务器(process server),此进程服务器为那些较少被使用的服务器提供代理功能。它同时监听一组端口,以等待外来的连接请求。每个服务的潜在用户总是从一个 CONNECT 请求开始,在 CONNECT 请求中指定了这些用户想要的服务的 TSAP 地址。如果没有专门的服务器在等待他们,则他们就会得到一个与进程服务器之间的连接,如图 6.9(a)所示。

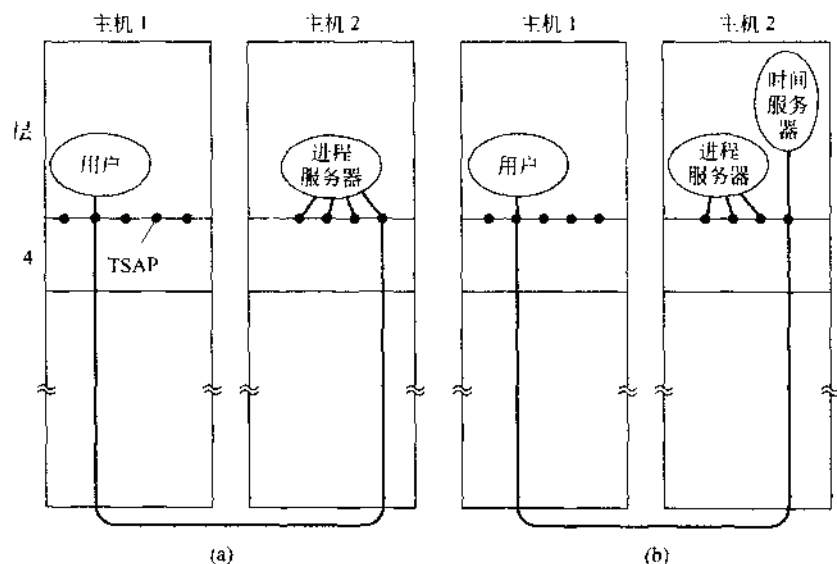


图 6.9 主机 1 的用户进程如何与进程 2 的时间服务器建立一个连接

进程服务器接到了进来的请求之后,它启动与该请求对应的服务器,并允许它继承自己与用户之间已有的连接。然后新的服务器执行用户请求的工作,而进程服务器则回去继续监听新的请求,如图 6.9(b)所示。

虽然初始连接协议对于那些可以根据需要而创建的服务器来说工作得很好,但是,在许多情形下还是有一些服务必须独立于进程服务器而存在。例如,一个文件服务器需要在特殊的硬件(一台带有硬盘的机器)上运行,不能够当有用户要与它通话的时候才被临时创建。

为了处理这些情形,通常考虑使用另一种方案。在这个模型中有一个称为名字服务器(name server)的特殊进程,有时候也称为目录服务器(directory server)。为了找到一个与给定的服务名字(比如“时间(time of day)”)相对应的 TSAP 地址,用户需要与名字服务器(它总是在监听一个知名的 TSAP 地址)建立一个连接。然后,用户发送一条消息指定它想要的服务名字,名字服务器送回相应的 TSAP 地址。之后,用户将它与名字服务



器之间的连接释放掉,再与期望的服务建立一个新的连接。

在这个模型中,当一个新的服务被创建的时候,它必须向名字服务器注册,并把它的服务名字(通常是一个 ASCII 字符串)和 TSAP 告诉名字服务器。名字服务器将这份信息记录到它的内部数据库中,所以,以后当用户查询的时候,它就知道答案了。

名字服务器的功能非常类似于电话系统中查号服务台的接线员——她提供的是从名字到电话号码之间的映射关系。如同在电话系统中一样,很重要的一点是,名字服务器(或者初始连接协议中的进程服务器)使用的知名 TSAP 地址一定是真正众所周知的。如果你不知道查号服务接线员的电话号码,那么你根本不可能拨打她的号码来查询她自己的号码。如果你认为自己所拨打的信息服务号码是非常显然的,那么当你到国外有机会的时候也可以试一试。

### 6.2.2 建立连接

建立一个连接听起来很容易,但是它实际上却出奇的琐碎。初看起来,好像一个传输实体只要给目标端发送一个 CONNECTION REQUEST TPDU,然后等待 CONNECTION ACCEPTED 应答就足够了。当网络可能丢失、存储或者重复分组的时候,问题就会发生。这些行为导致了极为严重的复杂性。

想象一个子网非常拥塞,以至于几乎所有的确认分组都无法及时地回到发送方,因此,每个分组都超时,因而被迫传输两次或者三次。假设该子网内部使用数据报,并且每个分组可以走不同的路径。有些分组可能受到子网内流量拥塞的影响,从而需要很长时间才能到达,也就是说,它们被存储在子网内部,过了一段时间以后才被送出来。

最可能的梦魇随之而来了。一个用户与一家银行建立了一个连接,并发送消息告诉银行将一大笔钱转到一个并不完全信任的人的账户下,然后释放该连接。不幸的是,此场景中的每个分组都被复制并保存在子网中。当连接被释放以后,所有的分组又从子网中冒出来,并且按序到达目标端:请银行建立一个新的连接、转一笔钱(第二次!),然后释放连接。银行无法辨别这些是重复的分组。它必须假定这是第二笔独立的交易,所以再转账一次。这一小节余下的部分我们将学习如何处理这些延迟的重复分组,同时将重点放在“以可靠的方法来建立连接”的算法上,利用这些算法可以避免发生像上面提到的这种梦魇。

问题的关键是由于网络中存在延迟的重复分组。对付这一问题的方法多种多样,但是没有一种方法让人满意。一种方法是使用一次性的传输地址(用完之后即扔掉)。在这种方法中,每当需要一个传输地址的时候,总是会生成一个新的传输地址。当一个连接被释放的时候,该地址被丢弃,永不再使用。这种策略使得图 6.9 中的进程服务器模型无法工作。

另一种可能的做法是,发起方为每个连接分配一个连接标识符(即一个序列号,每当建立一个连接的时候,该序列号增一),并且每个 TPDU(也包括请求建立连接的那个 TPDU)都包含该连接标识符。当每个连接被释放以后,每个传输实体可以更新一张内部表,该表中以“(对等传输实体,连接标识符)”的形式列出了所有过期的连接。当一个连接请求到来的时候,传输实体可以在表中进行检查,看它是否属于某一个以前已被释放掉

的连接。

不幸的是,这种方案有一个最为基本的缺陷:它要求每个传输实体无限期地维护一定数量的历史信息。如果一台机器崩溃了,它的内存全部丢失,那么,它就不可能知道哪些连接标识符已经被用过了。

因此,我们需要采用不同的方法。我们不再允许分组在子网中可以生存无限长时间,而是设计一种机制来消灭掉那些已经过时但仍然停留在子网中的分组。如果我们能确保没有一个分组的生存期会超过某一个已知的时间阈值,那么,问题就会变得好控制多了。

利用下面的一种(或多种)技术,分组的生存期可以被限定在一个已知的最大值之内:

- (1) 受限制的子网设计。
- (2) 在每个分组中放置一个跳计数器。
- (3) 为每个分组打时间戳。

第一种方法包括任何一种可避免分组进入循环的方法,并结合某一种“限定(已知的)最长可能路径上拥塞延迟的上界”的做法。第二种方法是,将跳计数器初始化到某个适当的值,然后每次分组被转发的时候跳计数器减一。网络协议简单地丢弃掉那些跳计数器变成零的分组。第三种方法要求每个分组携带它的创建时间,路由器负责丢弃掉那些年龄超过某个预设值的分组。后一种方法要求同步所有路由器的时钟,而这本身并不是一件很容易完成的任务,除非通过网络之外的手段能够实现同步,比如,通过使用 GPS 或者某一个无线站定期地广播精确的时间。

在实践中,我们不仅要保证一个分组最终会死亡,而且要保证它的所有确认最终也要死亡,所以我们现在引入一个  $T$  值,它是分组的实际最大生存期的某个不太大的倍数。此倍数与具体的协议有关,它只影响  $T$  的长短。如果在一个分组被发送出去之后,我们等待了  $T$  秒时间,那么,我们可以确信该分组的所有痕迹现在都没有了,它和它的确认将来不会突然冒出来而使问题复杂化。

限定了分组生存期的上界之后,我们就有可能设计一种简单的方法来安全地建立连接。下面描述的方法来源于 Tomlinson(1975),该方法解决了前面提到的问题但是引入了一些特殊的问题。Sunshine 和 Dalal(1978)又进一步精炼了这种方法。它的一些变种被广泛应用于实践中,包括 TCP 中。

为了解决一台机器崩溃之后丢失所有内存的问题,Tomlinson 建议让每台主机都配备一个报告时间的时钟。不同主机上的时钟不需要同步。假定每个时钟均采用了二进制计数器的形式,并且在每个统一的时间间隔内计数器递增自己。而且,计数器的位数必须等于或者超过序列号的位数。最后,也是最重要的是,假定时钟持续不停地运行,即使主机停机它也不停。

基本的思路是确保两个编号相等的 TPDU 永远不会同时有效。当建立一个连接的时候,时钟的低  $k$  位被用于初始序列号(也是  $k$  位)。因此,与第 3 章中的协议不同的是,每个连接都从一个完全不同的初始序列号开始对它的 TPDU 进行编号。序列号空间应该足够大,以便当序列号回绕的时候,原来那些具有相同序列号的 TPDU 都早已经消失了。在时间和初始序列号之间的这种线性关系如图 6.10 所示。

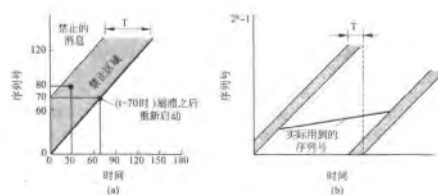


图 6.10

(a) TPDU 不能够进入禁止区域; (b) 重新同步的问题

一旦双方的传输实体已经统一了初始序列号,则可以用任何一个滑动窗口协议来控制数据流。事实上,初始序列号曲线(图中的粗线)并不是线性的,而是一条梯状线,因为时钟是按离散的步伐向前走的。为了简化起见,我们忽略这一细节。

在主机崩溃时有一个问题会发生。当主机又启动的时候,它的传输实体并不知道它崩溃时处在序列号空间的什么地方。一种方案是要求传输实体在恢复运行之后空等  $T$  秒钟,以便让所有老的 TPDU 都死掉。然而,在一个复杂的互联网环境中,  $T$  可能非常大,所以,这种策略显然并不切实可行。

为了避免在崩溃之后要求  $T$  秒的死等时间,有必要对序列号的用法引入一项新的限制。通过一个例子我们就能清楚地看出这项限制的必要性。假设分组的最大生存期  $T$  为 60 秒,时钟滴答为每秒钟一次。正如图 6.10(a)中的粗线所示,在  $x$  时刻打开的连接 5 的初始序列号为  $x$ 。请想象在  $t=30$  秒时,在(稍早之前被打断的)连接 5 上发送的一个普通数据 TPDU 被赋予序列号 80。我们称它为 TPDU X。主机在送出了 TPDU X 之后立即就崩溃了,然后又重新启动。在  $t=60$  秒时,它开始重新打开连接 0 至连接 4。在  $t=70$  秒时,它重新打开连接 5,根据要求它使用初始序列号 70。在接下来的 15 秒内,它发送 TPDU 70 至 TPDU 80。因此,在  $t=85$  秒时,一个新的 TPDU(序列号为 80,并且属于连接 5)被送到子网中。不幸的是,TPDU X 仍然存在。如果它在新的 TPDU 80 之前到达接收方的话,那么,TPDU X 将被接受,而真正的 TPDU 80 则被当作重复分组而拒绝。

为了防止这样的问题,我们必须保证:对于任何一个序列号,在它有可能被用作初始序列号之前的  $T$  秒时间内,禁止使用该序列号(即分配给新的 TPDU)。时间和序列号的非法组合区域如图 6.10(a)中的禁止区域所示。在任何连接上发送任何 TPDU 之前,传输实体必须读取时钟的信息,并检查看这个 TPDU 是否落在禁止区域中。

该协议在两种情况下可能会陷入困境。如果一台主机在一个新打开的连接上发送了大量的数据,并且速度太快,则实际序列号与时间之间的曲线有可能比初始序列号与时间之间的曲线还要陡。这意味着任何连接上的最大数据速率为每个时钟滴答一个 TPDU。同时这也意味着,在崩溃并重新启动之后,传输实体在打开一个新的连接之前必须等待一定

数量的时钟滴答,以免同样的编号被使用两次。这两点都要求使用短的时钟滴答(几个微秒 $[\mu\text{s}]$ ,甚至更短)。

不幸的是,由于发送速度太快而造成由下往上进入禁止区域并不是陷入困境的惟一情形。从图 6.10(b)我们可以看出,即使在低于时钟速率的任何数据率上,实际使用的序列号与时间之间的曲线最终也会从左边进入到禁止区域中。实际序列号曲线的斜度越大,则这种事件越晚发生。正如上面所述,在发送每一个 TPDU 之前,传输实体必须对它进行检查,以确定是否会进入禁止区域,如果是的话,则要么延迟  $T$  秒再发送该 TPDU,要么重新同步序列号。

基于时钟的方法解决了重复的数据 TPDU 延迟到达的问题,但是,为了真正有效地使用这种方法,首先必须要建立一个连接。由于控制 TPDU 也可能被延迟,所以,要让双方对初始序列号达成一致会有潜在的问题。例如,假设连接是这样建立起来的:主机 1 给远程对等主机 2 发送一个 CONNECTION REQUEST TPDU,其中包含了建议的初始序列号和远程端口号。然后,接收方,即主机 2 送回一个 CONNECTION ACCEPTED TPDU 作为对这个请求的确认。如果 CONNECTION REQUEST TPDU 丢失了,但是是一个延迟的重复 CONNECTION REQUEST TPDU 突然出现在主机 2 上,那么,双方将会建立一个不正确的连接。

为了解决这个问题,Tomlinson(1975)引入了三步握手(three-way handshake)的过程。这个建立连接的协议并不要求双方以同样的序列号开始发送数据,所以它可以与一些不要求全局时钟的同步方法一起使用。图 6.11(a)显示了当主机 1 发起连接请求时的正常建立过程。主机 1 选择一个序列号  $x$ ,并且发送一个包含  $x$  的 CONNECTION REQUEST TPDU 给主机 2。主机 2 回应一个 ACK TPDU 作为对  $x$  的确认,并且在 ACK TPDU 中宣告它自己的初始序列号  $y$ 。最后,主机 1 在它发送的第一个数据 TPDU 中,对主机 2 选择的初始序列号进行确认。

现在我们来查看当延迟的、重复的控制 TPDU 出现的时候三步握手协议是如何工作的。在图 6.11(b)中,第一个 TPDU 是一个来自于老的连接中并且被延迟了的重复 CONNECTION REQUEST。该 TPDU 到达主机 2,而主机 1 对此并不知情。主机 2 对这个 TPDU 的回应是给主机 1 发送一个 ACK TPDU,其效果相当于验证一下主机 1 是否真的请求建立一个新的连接。当主机 1 拒绝了主机 2 的连接建立请求之后,主机 2 意识到这是延迟的重复 TPDU 所开的一个玩笑,于是放弃了连接。因此,一个延迟的重复 TPDU 并没有任何伤害。

最糟糕的是当延迟的 CONNECTION REQUEST 和 ACK 同时浮现在子网中的时候。这种情形如图 6.11(c)所示。如同在前一个例子中一样,主机 2 得到一个延迟的 CONNECTION REQUEST,并回应了它。这里的关键点是,你必须意识到,主机 2 已经建议使用  $y$  作为从主机 2 到主机 1 之间流量的初始序列号,同时也要知道,现在已经没有包含序列号为  $y$  的 TPDU 或者对  $y$  的确认了。当第二个延迟的 TPDU 到达主机 2 的时候,主机 2 看到  $z$  已被确认而不是  $y$  已被确认,这个事实让主机 2 知道了这也是一个老的重复分组。在这里,必须要意识到的要点是,所有老的 TPDU 的组合都不能够让协议失败,也不能导致在无人期望的情况下偶然建立一个连接。

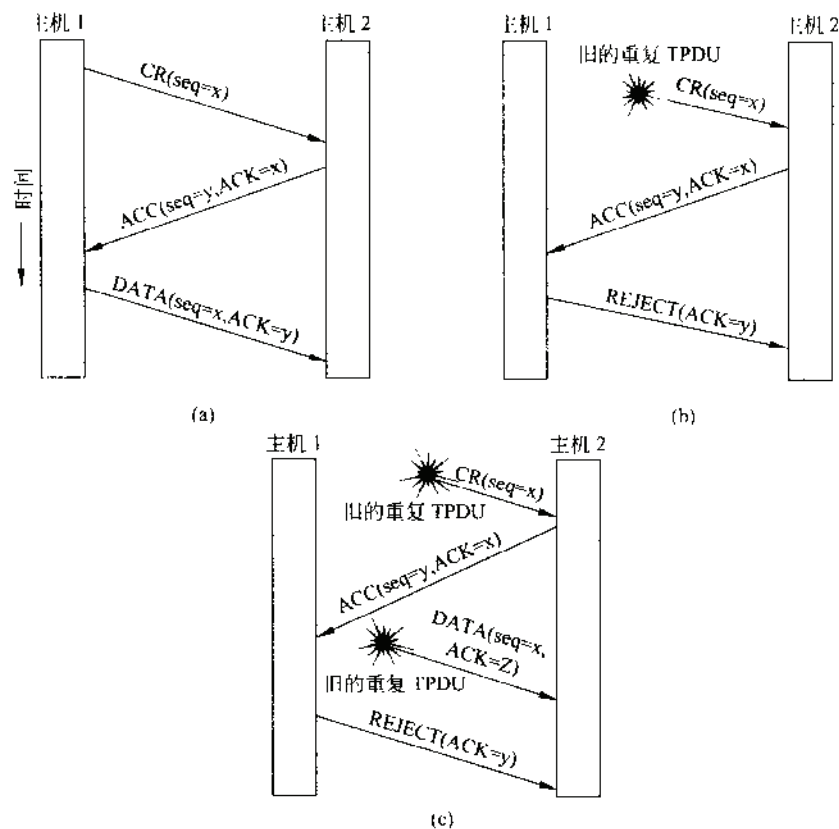


图 6.11 使用三步握手法建立连接的三个协议场景。这里 CR 代表 CONNECT REQUEST

(a) 正常的操作；(b) 老的 CONNECTION REQUEST 重复 TPDU 出现了；

(c) CONNECTION REQUEST 重复 TPDU 和 ACK 重复 TPDU 出现了

### 6.2.3 释放连接

释放一个连接要比建立一个连接容易得多。然而，这其中也存在许多让人意想不到的陷阱。正如我们前面提到过的，有两种终止连接的风格：非对称释放和对称释放。非对称释放连接是电话系统的工作方式：当一方挂机的时候，连接就中断了。对称释放连接的方法是把连接看成两个独立的单向连接，并要求单独释放每一个单向连接。

非对称释放方法较为粗暴，可能会导致数据丢失。请考虑图 6.12 中的场景。当连接被建立起来之后，主机 1 发送一个 TPDU，它正确地到达了主机 2。然后，主机 1 发送另一个 TPDU。不幸的是，主机 2 在第二个 TPDU 到达之前发出了 DISCONNECT TPDU。结果是，该连接被释放了，而数据却丢失了。

很显然，我们需要一个更加复杂的释放协议以避免数据丢失。一种方法是使用对称释放方式。在对称释放过程中，每个方向被单独释放，两个方向互不影响。在这种方法中，即使当主机发送了 DISCONNECT TPDU 以后，它仍然可以接收数据。



```

sequenceDiagram
    participant H1 as 主机1
    participant H2 as 主机2
    H1->>H2: CR
    H1->>H2: ACC
    H1->>H2: DATA
    H2->>H1: DATA
    Note over H1: 在一个循环请求后  
后不再提交数据
  
```

图 6.12 粗恶的断开连接方式, 有的数据丢失了

图 8.13 西军问题

假设1号蓝军的指挥官发送这样一条消息：“我建议我们在3月29日的黎明时分发起进攻。怎么样？”现在假设该消息到达了2号蓝军，指挥官同意这一建议，并且他的同僚安全地回到了1号蓝军。进攻会如期进行吗？可能不会，因为2号蓝军的指挥官不知道他的同僚是否能赶到。如果同僚没有过去的话，1号蓝军将不会发动进攻，所以对他来说，贸然发动进攻将是十分愚蠢的。

现在我们对协议进行改进,将它变成一个三步握手协议。原始建议的发起方必须对应答消息进行确认。假设中间没有消息丢失,那么,2号蓝军将得到确认,但是,1号蓝军的指挥官现在犹了。毕竟,他不知道他的确认信是否送过去了,如果确认信没有送到2号蓝军的话,他知道2号蓝军就不会发动进攻。我们当然可以设计一个四步握手协议,但

是,这并不能帮助我们解决问题。

实际上,可以证明,能够正确地完成这一任务的协议并不存在。我们用反证法,假设存在某一个协议可以正确地工作。该协议的最后一条消息可能是至关重要的,也可能不是。如果不是的话,则从协议中去掉这条消息(以及其他所有无关紧要的消息),这样我们得到的协议中每条消息都是至关重要的。如果最后一条消息没有被送过去的话则情形会怎么样呢?我们刚才说过了它是至关重要的,所以,如果它丢失了的话,则进攻就不会如期进行。由于最后一条消息的发送方永远也无法确定它是否正确到达了,所以,他不会冒险发动进攻。更糟的是,另一支蓝军也明白这一点,所以,它也不会发动进攻。

为了看清楚两军队问题与释放连接问题之间的相关性,只要用“断开连接”来代替“发动进攻”就可以。如果任何一方一定要在确定另一方已经作好了断开连接的准备之后才准备断开连接的话,那么,断开连接的操作将永远也不可能发生。

在实践中,当释放连接的时候,通常有一方愿意冒比进攻白军更大的风险,所以,实际情况还不至于如此绝望。图 6.14 显示了使用三步握手法释放连接的四个场景。虽然这个协议并非完全没有错误,但是通常情况下已经足够了。

在图 6.14(a)中,我们可以看到正常的情形,其中一个用户发送一个 DR (DISCONNECTION REQUEST) TPDU,以发起释放连接的过程。当它到达的时候,接收方也送回一个 DR TPDU,并启动一个定时器,该定时器的目的是为了防止它的 DR 丢失。当这个 DR 到达的时候,最初的发送方送回一个 ACK TPDU,并且释放连接。最后,当 ACK TPDU 到达的时候,接收方也释放连接。释放一个连接意味着传输实体将有关该连接的信息从它的内部表(该表记录了所有当前已打开的连接)中删除,并且通过某种方式通知该连接的所有者(传输用户)。这个动作与传输用户发出一个 DISCONNECT 原语有所不同。

如果最后的 ACK TPDU 丢失的话,则如图 6.14(b)所示,这种情形可以通过一个定时器来补救。当定时器超时的时候,不管怎么样,连接都被释放。

现在考虑第二个 DR 被丢失的情形。发起释放连接操作的用户将接收不到期望的应答,所以它将超时,于是再次尝试释放连接。在图 6.14(c)中,我们可以看到这个工作过程,图中假定第二次握手过程中没有 TPDU 丢失,所有的 TPDU 都被及时地递交给正确的接收方。

在图 6.14(d)所示的最后一个场景中,假设由于丢失 TPDU 的原因,因而所有重传 DR 的尝试也失败;除此以外,其他的情况与图 6.14(c)中相同。经过 N 次重试之后,发送方放弃了,并且释放连接。同时,接收方超时,于是退出。

虽然这个协议在通常情况下已经足够了,但是在理论上,如果初始的 DR 和 N 次重传全部丢失的话,则协议可能失败。发送方将放弃并且强行释放连接,而另一方对于所有的释放连接尝试一无所知,它仍然处于活跃状态。这种情况会导致一个半开的连接。

如果我们不允许发送方在经过 N 次重试之后即放弃,而是让它不停地重试,直至得到对方的应答,那么,这个问题就可以避免。然而,如果另一方也允许超时的话,那么发送方将真的要无限尝试下去了,因为没有应答会送过来。如果我们不允许接收方超时的话,那么,在图 6.14(d)中,该协议将被悬挂起来了。

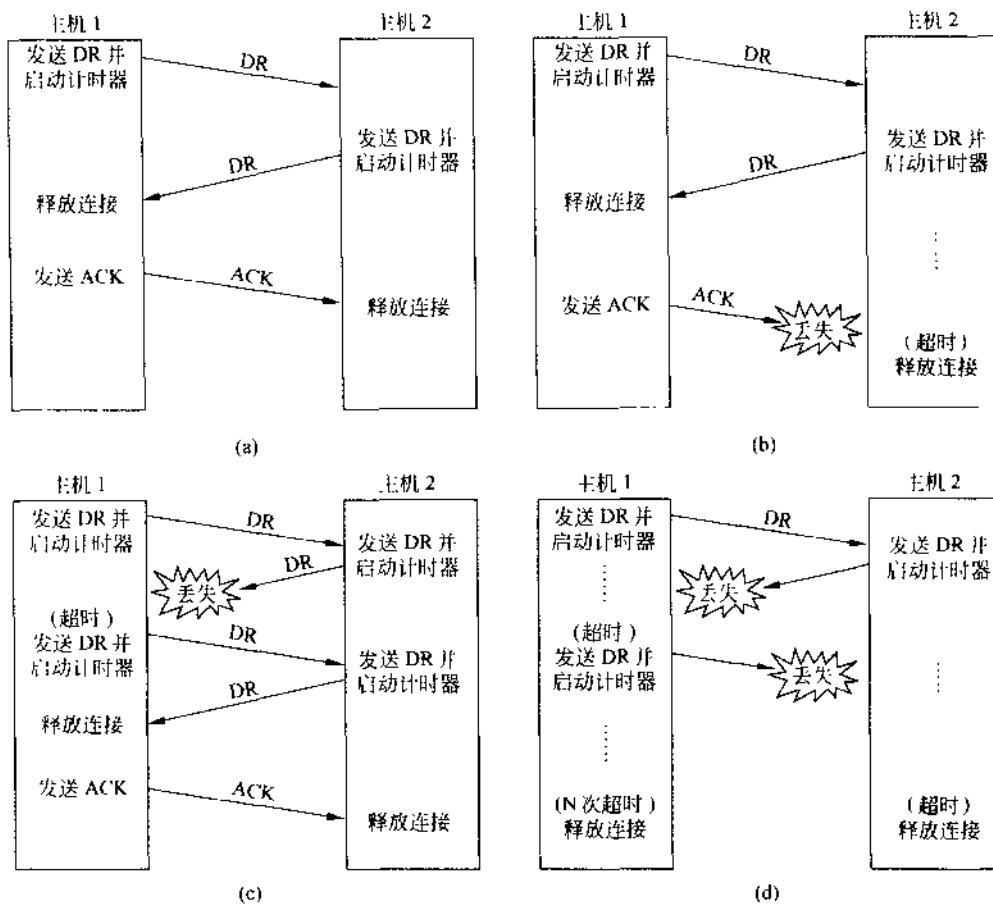


图 6.14 释放连接的 4 种情况

(a) 一般情况下的三步握手过程；(b) 最后的 ACK 丢失了；(c) 应答丢失了；(d) 应答和后续的 DR 都丢失了

杀死半开连接的一种办法是采用这样一条规则：如果在规定的一段时间以内没有 TPDU 到来的话，该连接将被自动断开。那样的话，如果一方断开连接了，则另一方将检测到该连接已经没有动作了，于是也会将其释放。当然，如果引入这条规则的话，那么，每当传输实体发送一个 TPDU 的时候，它必须要让一个定时器先停止，然后再重新启动。如果该定时器超时了，则传送一个哑的 TPDU，其目的仅仅是为了避免另一方断开此连接。另一方面，在采用了自动断开连接的规则之后，如果一个空闲连接上多个连续的哑 TPDU 全部丢失的话，则首先有一方将自动断开连接，然后另一方也将自动断开连接。

我们不再进一步讨论有关释放连接的问题了，但是现在你应该很清楚，释放一个可能有数据丢失的连接并不像初看起来的那么简单。

#### 6.2.4 流控制和缓冲

在详细地讨论了连接的建立和释放过程以后，现在我们来看一看在使用连接的过程

中如何对它们进行管理。其中一个关键问题我们前面已经看到过了：流控制。传输层中的流控制问题在有些方面与数据链路层中的情形相似，但是在另一些方面又有所不同。基本的相似之处是，在这两层上，都需要在每个连接上使用滑动窗口或者其他的方案以避免一个快速的发送方淹没掉一个慢速的接收方。主要的不同之处在于，一台路由器通常只有相对少量的线路，而一台主机可能有大量的连接。这一差异决定了在传输层上实现数据链路层的缓冲策略将是不切实际的。

在第3章的数据链路层协议中，所有的帧在发送路由器和接收路由器两端都被缓冲起来。例如，在第3章的协议6中，发送方和接收方都要求专门为每条线路使用  $\text{MAX\_SEQ}+1$  个缓冲区，其中一半用于输入，一半用于输出。如果一台主机允许最多 64 个连接，并且使用 4 位序列号，那么，这个协议将要求 1024 个缓冲区。

在数据链路层中，发送方必须将往外发的帧缓冲起来，因为以后可能要重传这些帧。如果子网提供的是数据报服务，那么发送端传输实体必须也要缓冲 TPDU，理由相同。如果接收方知道发送方缓冲了所有的 TPDU 一直到它们被确认为止，那么，接收方可能会，也可能不会为特定的连接分配特定数量的缓冲区，这要看它怎么觉得合适。例如，接收方可能只维护一个缓冲区池，让所有的连接共享这同一个池。当一个 TPDU 进来的时候，接收方企图为其动态地分配一个新的缓冲区。如果分配成功了，则接受该 TPDU；否则将它丢弃。由于发送方已经准备好要重传那些被子网丢失的 TPDU，所以，让接收方丢弃 TPDU 并无损害，只不过浪费一些资源而已。发送方会一直尝试发送 TPDU，直至接收到确认为止。

简而言之，如果网络服务不是可靠的，那么，发送方必须将所有发送的 TPDU 缓冲起来，就好像数据链路层中的做法一样。然而，对于可靠的网络服务，其他的平衡策略也是有可能的。尤其是，如果发送方知道接收方总是有缓冲区空间的话，它就不需要保留自己发送出去的 TPDU 的副本。然而，如果接收方不能够保证每一个进来的 TPDU 都被接受的话，则发送方无论如何一定要提供缓冲能力。在后一种情况下，发送方不能信任网络层的确认，因为网络层的确认仅仅意味着该 TPDU 到达了，而不见得它已经被接受。后面我们还会讨论到这个重要的话题。

即使接收方已经同意要缓冲 TPDU，接下来还有缓冲区大小的问题。如果大多数 TPDU 的长度都差不多，则很自然的做法是将缓冲区组织成一个相等大小缓冲区的池，每个缓冲区容纳一个 TPDU，如图 6.15(a) 所示。然而，如果 TPDU 长度的偏差范围很大，既可能是终端上输入的少量字符，也可能是文件传输过程中的上千个字符，那么，用固定长度的缓冲区池就有问题了。如果将缓冲区的大小设置成最大可能的 TPDU 长度，那么，当短的 TPDU 到来的时候就会浪费内存空间。如果将缓冲区的大小设置成比 TPDU 的最大长度还小的话，则长的 TPDU 就需要多个缓冲区，从而带来额外的复杂性。

缓冲区大小问题的另一方法是使用可变大小的缓冲区，如图 6.15(b) 所示。这里的优点是更好的内存利用率，付出的代价是使缓冲区的管理更加复杂。第三种可能的方案是为每个连接使用一个大的循环缓冲区，如图 6.15(c) 所示。如果所有连接的负载都很重的话，则此系统也能够很好地利用内存；但如果有些连接的负载较轻，则系统的内存使用情况很差。

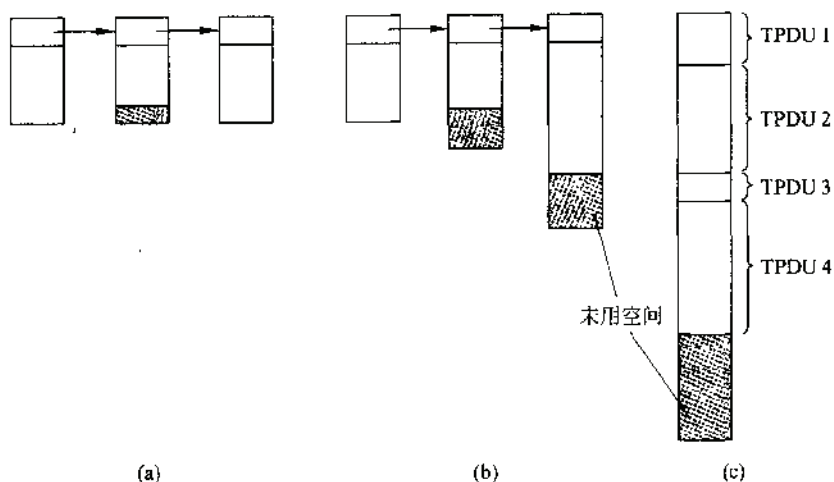


图 6.15

(a) 链式的固定大小的缓冲区; (b) 链式的可变大小的缓冲区; (c) 每个连接使用一个大的循环缓冲区。

在源端缓冲和目标端缓冲之间的最佳折衷取决于该连接所承载的流量的类型。对于低带宽的突发性流量,比如交互终端所产生的流量,最好两端都不用给它分配任何专用的缓冲区,而是根据需要动态地分配缓冲区。由于发送方无法确定接收方是否能够获得一个缓冲区,所以发送方必须保留 TPDU 的一份副本,直至它被确认为止。另一方面,对于文件传输和其他的高带宽流量,最好的做法是,接收方分配一批可容纳整个窗口 TPDU 的缓冲区,从而允许最大速度的数据流。因此,对于低带宽的突发流量,最好在发送方缓冲数据,而对于高带宽的平滑流量,最好在接收方缓冲数据。

随着连接被打开和关闭,以及流量模式的变化,发送方和接收方需要动态地调整它们的缓冲区分配策略。因此,传输协议应该允许发送主机请求另一端的缓冲区空间。缓冲区可以被分配给每个连接,或者集体分配给两台主机之间当前正在运行的所有连接。或者换一种做法,由于接收方知道它的缓冲区情况(但是不知道流量的情况),所以它可以告诉发送方“我已经为你预留了 X 个缓冲区。”如果打开的连接的数量应该增加的话,则为每个连接分配的缓冲区数有必要相应地减少,因此,协议应该提供这种协商能力。

为了管理动态的缓冲区分配过程,较为通用和合理的做法是将缓冲过程与确认机制分离开,从而不同于第 3 章中的滑动窗口协议。动态的缓冲区管理实际上意味着一个可变大小的窗口。刚开始的时候,发送方根据它所了解到的需求情况,请求一定数量的缓冲区。然后,接收方根据它的能力分配尽可能多的缓冲区。每次当发送方传输一个 TPDU 的时候,它必须减小缓冲区分配数,当分配数到达 0 的时候停止发送。然后,接收方在反向流量中独立地捎带确认和缓冲区分配数。

图 6.16 显示了一个在 4 位序列号的数据报子网中动态窗口管理的一种可能做法。假设如图中所示,缓冲区分配信息被放在单独的 TPDU 中,而不是捎带在反向流量中被传送到另一端。初始时, A 想要 8 个缓冲区,但是, B 只分配了 4 个。然后 A 发送 3 个 TPDU,其中第 3 个 TPDU 丢失。在第 6 行, B 确认已经接收到直至(含)序列号 1 的所有



TPDU,从而允许 A 释放这些缓冲区;同时进一步通知 A,允许发送 3 个从序列号 1 之后开始的 TPDU(即 TPDU 2、3 和 4)。A 知道它已经发送了 TPDU 2,所以它认为现在可以发送 TPDU 3 和 4,于是它接下去就这么做了。这时候它被阻塞住,它必须等待分配更多的缓冲区。然而,在阻塞过程中,因超时而导致的重传仍然可以进行(第 9 行),因为它们使用的缓冲区已经被分配了。在第 10 行,B 确认已经接收到直至(含)序列号 4 的所有 TPDU,但是拒绝让 A 继续发送。这样的情形对于第 3 章中的固定窗口协议是不可能发生的。下一个从 B 到 A 的 TPDU 表明 B 已分配了另一个缓冲区,从而允许 A 继续发送。

A	消息	B	说明
1	→	<请求 8 个缓冲区>	→ A 想要 8 个缓冲区
2	←	<ack=15, buf=4>	← B 只准许消息 0~3
3	→	<seq=0, data=m0>	→ A 现在还有 3 个缓冲区
4	→	<seq=1, data=m1>	→ A 现在还有 2 个缓冲区
5	→	<seq=2, data=m2>	→ 消息丢失了,但是 A 认为它还有一个缓冲区
6	←	<ack=1, buf=3>	← B 确认消息 0 和 1,并允许发送消息 2~4
7	→	<seq=3, data=m3>	→ A 还有 1 个缓冲区
8	→	<seq=4, data=m4>	→ A 已经没有缓冲区了,必须停止
9	→	<seq=2, data=m2>	→ A 超时,并重传
10	←	<ack=4, buf=0>	← 所有的消息都确认了,但是 A 仍然被阻塞
11	→	<ack=4, buf=1>	→ A 现在可以发送消息 5
12	←	<ack=4, buf=2>	← B 又从某个地方找到了一个新的缓冲区
13	→	<seq=5, data=m5>	→ A 有 1 个缓冲区
14	→	<seq=6, data=m6>	→ A 现在又被阻塞了
15	←	<ack=6, buf=0>	← A 仍然被阻塞
16	...	<ack=6, buf=4>	← 潜在的死锁

图 6.16 动态缓冲区分配过程

箭头显示了传输的方向,省略号(...)代表一个丢失的 TPDU

在数据报网络中,如果控制 TPDU 也可能丢失的话,则这种缓冲区分配方案有可能引发一些潜在的问题。请看第 16 行,B 现在已经为 A 分配了更多的缓冲区,但是,这个指示缓冲区分配的 TPDU 丢失了。由于控制 TPDU 并没有序列号,也不超时,所以 A 现在死锁了。为了避免这种情况,每台主机应该定期地在每个连接上发送一些包含确认和缓冲区状态的控制 TPDU。采用这种方法,死锁迟早会被打破。

到现在为止,我们一直暗含着假定发送方数据率的惟一限制因素是接收方可供使用的缓冲区空间的数量。随着内存价格的不断下跌,我们有可能为主机配备足够的内存,从而使得缓冲区短缺不再成为一个问题,即使可能发生,也极为少见。

当缓冲区空间不再限制最大数据流的时候,另一个瓶颈将会出现:子网的承载能力。如果邻接的路由器之间每秒钟至多交换  $x$  个分组,并且在一对主机之间有  $k$  条互不相交的路径,那么,不管每台主机有多少可用的缓冲区空间,它们每秒钟交换的 TPDU 不可能超过  $kx$  个。如果发送方费力地往前推进(即每秒钟发送超过  $kx$  个 TPDU)的话,子网将阻塞,因为它不可能以超过 TPDU 进来的速度来递交这些 TPDU。

这里需要的是一种基于子网的承载容量而不是接收方缓冲容量的动态调节机制。很显然,发送方必须使用流控制机制,以避免发送方同一时刻有太多未被确认而悬着的 TPDU。Belsnes(1975)建议使用一种滑动窗口流控制方案,在他的方案中,发送方动态地调整窗口的大小,以便与网络的承载容量相匹配。如果网络每秒钟能够处理  $c$  个 TPDU,并且处理周期时间(包括发送、传播、排队、接收方的处理过程以及确认的返回)为  $r$ ,那么,发送方的窗口应该是  $cr$ 。如果发送方使用这样大小的窗口,那么,它通常可以按流水线方式全速地运行。网络性能只要减小一点点就有可能导致发送方阻塞。

为了定期地调整窗口的大小,发送方可以监视这两个参数( $c$  和  $r$ ),然后重新计算期望的窗口大小。承载容量很容易确定,发送方只要计算在一段时间中被确认的 TPDU 的数量,然后除以这段时间的长度即可。在测量过程中,发送方应该尽可能快地发送 TPDU,以确保限制确认速率的因素是网络的承载容量,而不是较低的输入率。一个 TPDU 从发送出去到被确认所需要的时间可以精确地测量出来,它可以是运行过程中所维护的一个平均数。由于可用于任何一个流的网络容量在不停地随时间而变化,所以,发送方也应该频繁地调整窗口的大小,以便跟上其承载容量的变化。后面我们将会看到,Internet 使用了一种类似的方案。

### 6.2.5 多路复用

将几个会话复用到少数的连接、虚电路和物理链路上,这种做法在网络体系结构的几个层上均扮演了重要的角色。在传输层上,对多路复用的需求来自于多个方面。例如,如果一台主机上只有一个网络地址可以使用,那么,这台机器上所有的传输连接必须使用这同一个地址。当一个 TPDU 进来的时候,传输实体需要某一种方法来指明应该将它交给哪一个进程。这种情形被称为向上多路复用(upward multiplexing),如图 6.17(a)所示。在这个图中,4 个不同的传输连接全都使用了同样的网络连接(即 IP 地址)与远程主机进行通信。

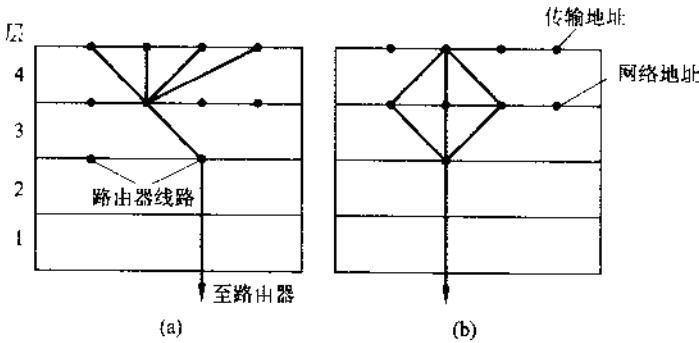


图 6.17

(a) 向上多路复用;(b) 向下多路复用

之所以多路复用机制在传输层上非常有用还有另外一个原因。例如,假设一个子网的内部使用了虚电路,并且每条虚电路上有最大数据速率限制。如果一个用户所需要的

带宽超过了一条虚电路所能提供的带宽,那么,一种办法是打开多个网络连接,并且采用轮循方法将流量分布到这些网络连接上,如图 6.17(b)所示。这种操作方式被称为**向下多路复用(downward multiplexing)**。如果打开  $k$  个网络连接,则实际的带宽增加至  $k$  倍。一个有关向下多路复用的常见例子是使用 ISDN 线路的家庭用户。这条线路提供了两个独立的连接,每个 64kbps。使用这两个连接来呼叫一个 Internet 供应商,并且将流量分布在两条链路上就有可能达到 128kbps 的有效带宽。

### 6.2.6 崩溃恢复

如果主机和路由器都有可能崩溃,那么,如何从这些崩溃事件中恢复运行就成为一个问题。如果传输实体完全在主机内部,则从网络和路由器的崩溃中恢复的方案是非常简单的。如果网络层提供的是数据报服务,那么,传输实体总是在期望那些丢失了的 TPDU,并且它们知道该如何处理这些 TPDU 被丢失的情形。如果网络层提供的是面向连接的服务,那么,由于虚电路丢失了,所以处理的办法是建立一条新的虚电路,然后询问远程的传输实体:哪些 TPDU 已经接收到了,哪些还没有接收到。之后再重传那些尚未接收到的 TPDU。

一个更加麻烦的问题是如何从主机的崩溃中恢复过来。尤其是,当服务器崩溃并且快速启动之后,客户可能期望还能够原来的基础上继续工作。为了说明其中的困难程度,我们假设一台主机(客户)正在使用一个简单的停-等协议,来给另一台主机(文件服务器)发送一个大文件。服务器上的传输层只是简单地将接收进来的 TPDU 逐个传递给传输用户。在传输到一半的时候,服务器崩溃了。当服务器恢复运行以后,它的内部表又被重新初始化,所以,它不知道原来的传输过程已经进行到哪儿了。

为了试图从原先的状态中恢复过来,服务器可能给所有其他的主机发送一个广播 TPDU,宣告自己刚才崩溃了,请它的客户们告诉它关于所有已打开连接的状态信息。每个客户可能处于以下两种状态之一:有一个未完成的 TPDU,  $S1$ ;或者,没有未完成的 TPDU,  $S0$ 。根据这一状态信息,客户必须确定是否重传最近的 TPDU。

初看起来似乎很明显:当客户知道服务器崩溃的时候,只有当客户有一个未确认的 TPDU 正在等待处理(即处于状态  $S1$ )时它才应该重传。然而,进一步探究这种做法便会发现其中的困难所在。例如,请考虑这样的情形:服务器的传输实体首先发送一个确认,然后当确认被送出以后,向应用进程执行一个写数据操作。将一个 TPDU 写到输出流中,以及发送一个确认,这是两件不可能同时完成的独立事件。如果在确认被发送出去之后,但是在写操作完成之前服务器崩溃了,那么,客户将会接收到确认,所以,当崩溃恢复之后广播消息到来的时候,它处于状态  $S0$  中。因此客户不会重传,从而不正确地认为 TPDU 已经到达服务器的应用进程中了。客户的这一决定将导致丢失一个 TPDU。

这时候你可能会这样想:“这个问题很容易解决,只要重新修改一下程序,让它先执行写操作再发送确认即可”。你可以再试一试。请想象一下,如果写操作已经完成了,但是在发送确认之前服务器崩溃了,那么,客户将处于状态  $S1$  中,因此它会重传,从而导致在服务器应用进程的数据流中出现一个未被检测到的重复 TPDU。

不管如何正确地编写客户和服务器的代码,总是存在使协议无法正确地恢复运

行的情形。服务器程序的实现方法可以有两种：先发送确认，或者先写数据。而客户程序可以有四种不同的实现方法：总是重传最后的 TPDU、永远不重传最后的 TPDU、仅当处于状态 S0 中时才重传，或者仅当处于状态 S1 中时才重传。两者结合起来有 8 种组合，但是正如我们将会看到的，对于每一种组合都存在一些使协议失败的事件。

在服务器端可能有三种事件：发送一个确认(A)、将数据写到输出进程(W)和崩溃(C)。这三种事件可以有 6 种不同的发生顺序：AC(W)、AWC、C(AW)、C(WA)、WAC 和 WC(A)，这里的括号表示 A 或者 W 不可能跟在 C 的后面(即一旦崩溃之后，它就崩溃了，不可能再发生其他的事件)。图 6.18 显示了客户和服务策略的 8 种组合，以及每一种组合的有效事件序列。请注意，对于每一种策略，总是存在某一个使协议失败的事件序列。例如，如果客户总是重传的话，则 AWC 事件将生成一个未检测到的重复 TPDU，但是其他两个事件可以正确地工作。

由发送主机使用的策略	由接收主机使用的策略					
	先确认，后写			先写，后确认		
	AC(W)	AWC	C(AW)	C(WA)	WAC	WC(A)
总是重传	OK	DUP	OK	OK	DUP	DUP
从不重传	LOST	OK	LOST	LOST	OK	OK
在 S0 中重传	OK	DUP	LOST	LOST	DUP	OK
在 S1 中重传	LOST	OK	OK	OK	OK	DUP

OK = 协议运行正确  
 DUP = 协议产生了一条重复信息  
 LOST = 协议丢失了一条信息

图 6.18 客户和服务策略的不同组合

进一步精心地修改协议也没有用。即使在服务器试图写 TPDU 之前，客户与服务器再多交换几个 TPDU，从而客户可以确切地知道接下来将要发生什么事情，但客户也没有办法知道崩溃动作恰好发生在写操作之前还是之后。结论是不可避免的：在没有同步事件的条件下，主机的崩溃和恢复对于上面的层来说不可能是透明的。

按照更加一般的术语表达，这个结论可以重新叙述为：从 N 层的崩溃中恢复的过程只能由 N+1 层来完成，并且仅当 N+1 层保留了足够的状态信息时才有可能恢复。正如前面所提到的，只要连接的每一端记录了它当前的传输状态信息，传输层就能够从网络层的失败中恢复过来。

这个问题使我们进入到另一层面的思考：所谓端到端的确认到底意味着什么。原则上，传输协议是端到端的，而不像下面的层那样是链式的。现在请考虑这样的情形：一个用户向一个远程数据库发出一些事务请求。假设远程传输实体的实现方式是先将 TPDU 传递给上一层，然后再确认。即使在这个例子中，用户机器上接收到一个确认并不一定意味着远程主机已经有足够的运行时间来真正更新数据库了。一个真正的端到端确认是指：一旦接收到这个确认就表明一项工作确实已经完成了，而没有接收到确认则表明这项工作尚未完成。在实践中，这样的端到端确认可能是无法实现的。Saltzer 等

(1984)更加详细地讨论了这一点。

## 6.3 一个简单的传输协议

为了使前面讨论过的概念和方法更加具体化,在这一节中我们将详细地学习一个传输层例子。这里我们将使用图 6.2 中列出的面向连接的服务原语。之所以选择这些面向连接的原语,是为了使这个例子更加类似于流行的 TCP 协议(但是比 TCP 简单)。

### 6.3.1 服务原语示例

我们的第一个问题是如何具体地表达这些传输原语。CONNECT 是很容易的:我们只要提供一个库过程 connect 即可,利用适当的参数来调用该函数就可以建立一个连接。这里的参数是本地和远程的 TSAP。在调用过程中,调用方被阻塞(即挂起),而传输实体试图建立一个连接。如果建立连接成功了,则调用方被解除阻塞,于是它可以开始传输数据了。

如果一个进程希望能够接受进来的连接请求,则它可以调用 listen,指定一个特定的 TSAP 作为监听的对象。然后,该进程阻塞,直到某个远程进程企图与这个 TSAP 建立连接。

请注意,这个模型是高度非对称的。一方是被动的,它执行 listen 调用,并等待有事情发生。另一方是主动的,它发起建立连接的过程。一个有趣的问题是,如果主动方首先开始的话,那该怎么办呢?一种策略是,如果远程 TSAP 上没有进程在监听的话,则建立连接失败。另一种策略是,让发起方阻塞(可能永远)下去,直到有监听进程出现为止。

我们的例子使用了折衷的办法,也就是说,在接收方,让连接请求保留一定的时间。如果在定时器过期之前,该主机上的一个进程调用了 listen 的话,则连接可以建立起来;否则,连接请求被拒绝,然后,调用方解除阻塞,并得到一个错误返回值。

为了释放一个连接,我们将使用库过程 disconnect。当双方都断开连接的时候,连接被释放。换句话说,我们使用了对称的断开连接模型。

关于数据传输,其问题与建立连接过程是一样的:发送方是主动的,而接收方是被动的。我们将采用与建立连接时同样的方案来处理数据传输过程:send 是一个主动调用,它的任务是将数据传出去;receive 是一个被动调用,它阻塞调用进程直至一个 TPDU 到达。

因此,我们的具体服务定义是由 5 个原语构成的:CONNECT、LISTEN、DISCONNECT、SEND 和 RECEIVE。每个原语正好对应于一个库过程,由该库过程执行原语的功能。服务原语和库过程的参数如下所示:

```
connum = LISTEN(local)
connum = CONNECT(local, remote)
status = SEND(connum, buffer, bytes)
status = RECEIVE(connum, buffer, bytes)
status = DISCONNECT(connum)
```



LISTEN 原语宣告了调用方希望在指定的 TSAP 上接受外来的连接请求。该原语的用户将被阻塞,直到有一台远程主机企图与它建立连接。该原语没有超时。

CONNECT 原语带有两个参数:一个本地 TSAP(即传输地址)local 和一个远程 TSAP remote,此原语试图在两者之间建立一个传输连接。如果它成功了,则在 connum 中返回一个非负的整数,它可被用来在后续的调用中标识该连接。如果它失败了,则失败的原因以一个负数的形式被放在 connum 中。在我们的简单模型中,每个 TSAP 只可以参与到一个传输连接中,所以,一种可能的失败原因是:其中一个传输地址(指 local 或 remote)当前正在使用。其他一些可能的原因包括:远程主机停机了、非法的本地地址和非法的远程地址。

SEND 原语将缓冲区中的内容以一条消息的形式通过指定的传输连接发送出去,如果有必要的话这些内容被分成几个数据单元。在 status 中返回的可能错误包括:无此连接、非法的缓冲区地址,或者负的计数值。

RECEIVE 原语表示调用者期望接受数据。进来消息的长度将被放在 bytes 中。如果远程进程已经释放了该连接,或者缓冲区地址是非法的(即位于用户程序之外),则 status 将被设置成一个可指明问题本质的错误代码。

DISCONNECT 原语终止一个传输连接。参数 connum 指明了哪一个连接。可能的错误有:connum 属于另一个进程,或者 connum 不是一个有效的连接标识符。错误代码被放在 status 中返回,0 表示成功。

### 6.3.2 传输实体示例

在查看传输实体例子的实际代码之前,请一定要明白,这个例子与第 3 章中展示的例子非常相似:它更多的是一个教学示范协议,而不是一个正式的协议提案。出于简化的目的,许多在产品系统中必需的技术细节(比如全面的错误检查)在这里被忽略掉了。

传输层利用网络服务原语来发送和接收 TPDU。在这个例子中,我们需要选择一组网络服务原语。其中一种选择方案是不可靠的数据报服务。为了使这个例子尽可能地简单,我们不做这样的选择。若采用不可靠的数据报服务,则传输代码将非常庞大和复杂,而且大部分代码在处理丢失的和延迟的分组。此外,有关处理这些问题的大多数想法已经在第 3 章中详细地讨论过了。

与此相反,我们选择使用一个面向连接的、可靠的网络服务。通过这种做法,我们可以把焦点集中在那些不会出现在低层的传输问题上。这包括建立连接、释放连接和信用管理(credit management),等等。一个建立在 ATM 网络基础上的简单传输服务可能与此类似。

一般而言,传输实体可能是主机操作系统的一部分,也可能是运行在用户地址空间中的一个库函数包。为了简化起见,我们的例子假设它是一个库函数包,但是,只需作很小的变化就可以将它变成操作系统的一部分(主要的变化是由于用户缓冲区的访问方式有所不同)。

然而,值得提一下的是,在这个例子中,“传输实体”并不是一个真正独立的实体,而是用户进程的一部分。尤其是,当用户执行一个阻塞的原语(比如 LISTEN)时,整个传输实

体也被阻塞了。虽然这样的设计对于一台只有单个用户进程的主机来说是非常合适的,但是,在一台有多个用户的主机上,最自然的做法是让传输实体成为一个单独的进程,从而与所有的用户进程隔离开。

与网络层的接口是两个过程 `to_net` 和 `from_net`(没有列出来)。每个过程有 6 个参数。第一个是连接标识符,它被一对一地映射到网络虚电路上。接下来是 Q 和 M 位,Q 位设置为 1 表示这是一条控制消息,而 M 位设置为 1 则表示这条消息后面还有更多的数据在下一个分组中。之后是分组的类型,图 6.19 列出了可供选择的 6 种分组类型。最后是一个指向数据本身的指针,以及一个给出了数据字节数的整数值。

网络分组	含义
CALL REQUEST	请求建立一个连接
CALL ACCEPTED	对 CALL REQUEST 的应答
CLEAR REQUEST	请求释放一个连接
CLEAR CONFIRMATION	对 CLEAR REQUEST 的应答
DATA	用于传输数据
CREDIT	管理窗口使用的控制分组

图 6.19 本例子中用到的网络层分组

传输实体在调用 `to_net` 时,填充所有的参数供网络层读取;而在 `from_net` 调用中,网络层将一个进来的分组进行解析,并将传输层的内容传递给传输实体。传输层以过程参数的形式向网络层传递信息,而不是以进来的或者送出去的实际分组的形式来传递信息,因而有效地将网络层协议的细节隐藏起来。如果在底层虚电路的滑动窗口已满时,传输实体试图发送一个分组,则它将在 `to_net` 中被挂起,直至滑动窗口中有空间为止。这种机制对于传输实体而言是完全透明的,网络层可以使用类似于第 3 章的协议中 `enable_transport_layer` 和 `disable_transport_layer` 这样的命令来控制该机制。分组层窗口的管理也是由网络层来完成的。

除了这种透明的挂起机制以外,传输实体也可以调用显式的 `sleep` 和 `wakeup` 过程(没有列出来)。当传输实体逻辑上被阻塞住以等待一个外部事件(通常是等待一个分组的到来)发生的时候,它可以调用 `sleep` 过程。在 `sleep` 被调用之后,传输实体(当然也包括用户进程)停止执行。

传输实体的代码如图 6.20 所示,每个连接总是处于以下 7 种状态之一:

- (1) IDLE—连接还没有建立;
- (2) WAITING—CONNECT 已经被执行,CALL REQUEST 已送出;
- (3) QUEUED—CALL REQUEST 已经到达;还没有 LISTEN;
- (4) ESTABLISHED—连接已经建立;
- (5) SENDING—用户正在等待发送分组的许可;
- (6) RECEIVING—RECEIVE 已经完成;
- (7) DISCONNECTING—在本地,DISCONNECT 已经完成。

```

#define MAX_CONN 32 /* max number of simultaneous connections */
#define MAX_MSG_SIZE 8192 /* largest message in bytes */
#define MAX_PKT_SIZE 512 /* largest packet in bytes */
#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED 3
#define LOW_ERR 3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC,CLEAR_REQ,CLEAR_CONF,DATA_PKT,CREDIT}
pkt_type;
typedef enum {IDLE, WAITING, QUEUED, ESTABLISHED, SENDING, RECEIVING,
DISCONN} cstate;

/* Global variables. */
transport_address listen_address; /* local address being listened to */
int listen_conn; /* connection identifier for listen */
unsigned char data[MAX_PKT_SIZE]; /* scratch area for packet data */

struct conn {
    transport_address local_address, remote_address;
    cstate state; /* state of this connection */
    unsigned char * user_buf_addr; /* pointer to receive buffer */
    int byte_count; /* send/receive count */
    int clr_req_received; /* set when CLEAR_REQ packet received */
    int timer; /* used to time out CALL_REQ packets */
    int credits; /* number of messages that may be sent */
}; conn[MAX_CONN + 1]; /* slot 0 is not used */

void sleep(void); /* prototypes */
void wakeup(void);
void to_net(int cid, int q, int m, pkt_type pt, unsigned char * p, int bytes);
void from_net(int * cid, int * q, int * m, pkt_type * pt, unsigned char * p, int * bytes);

int listen(transport_address t)
{ /* User wants to listen for a connection. See if CALL_REQ has already arrived. */
    int i, found = 0;

    for (i = 1; i <= MAX_CONN; i++) /* search the table for CALL_REQ */

```

图 6.20 传输实体示例代码

```

        if (conn[i].state == QUEUED && conn[i].local_address == t) {
            found = i;
            break;
        }

    if (found == 0) {
        /* No CALL_REQ is waiting. Go to sleep until arrival or timeout. */
        listen_address = t; sleep(); i = listen_conn;
    }

    conn[i].state = ESTABLISHED;          /* connection is ESTABLISHED */
    conn[i].timer = 0;                    /* timer is not used */
    listen_conn = 0;                      /* 0 is assumed to be an invalid address */
    to_net(i, 0, 0, CALL_ACC, data, 0);   /* tell net to accept connection */
    return(i);                            /* return connection identifier */
}

int connect(transport_address l, transport_address r)
{ /* User wants to connect to a remote process; send CALL_REQ packet. */
    int i;
    struct conn *cptr;

    data[0] = r; data[1] = l;             /* CALL_REQ packet needs these */
    i = MAX_CONN;                         /* search table backward */
    while (conn[i].state != IDLE && i > 1) i = i - 1;
    if (conn[i].state == IDLE) {
        /* Make a table entry that CALL_REQ has been sent. */
        cptr = &conn[i];
        cptr->local_address = l; cptr->remote_address = r;
        cptr->state = WAITING; cptr->clr_req_received = 0;
        cptr->credits = 0; cptr->timer = 0;
        to_net(i, 0, 0, CALL_REQ, data, 2);
        sleep(); /* wait for CALL_ACC or CLEAR_REQ */
        if (cptr->state == ESTABLISHED) return(i);
        if (cptr->clr_req_received) {
            /* Other side refused call. */
            cptr->state = IDLE; /* back to IDLE state */
            to_net(i, 0, 0, CLEAR_CONF, data, 0);
            return(ERR_REJECT);
        }
    }
    else return(ERR_FULL);                /* reject CONNECT; no table space */
}

int send(int cid, unsigned char bufptr[], int bytes)
{ /* User wants to send a message. */

```

图 6.20 (续)

```

int i, count, m;
struct conn * cptr = &conn[cid];

/* Enter SENDING state. */
cptr->state = SENDING;
cptr->byte_count = 0; /* # bytes sent so far this message */
if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
if (cptr->clr_req_received == 0) {
    /* Credit available; split message into packets if need be. */
    do {
        if (bytes - cptr->byte_count > MAX_PKT_SIZE) { /* multipacket
            message */
            count = MAX_PKT_SIZE; m = 1; /* more packets later */
        } else { /* single packet message */
            count = bytes - cptr->byte_count; m = 0; /* last pkt of this message */
        }
        for (i = 0; i < count; i++) data[i] = bufptr[cptr->byte_count + i];
        to_net(cid, 0, m, DATA_PKT, data, count); /* send 1 packet */
        cptr->byte_count = cptr->byte_count + count; /* increment bytes sent
            so far */
    } while (cptr->byte_count < bytes); /* loop until whole message sent */
    cptr->credits--; /* each message uses up one credit */
    cptr->state = ESTABLISHED;
    return(OK);
} else {
    cptr->state = ESTABLISHED;
    return(ERR_CLOSED); /* send failed; peer wants to disconnect */
}
}

int receive(int cid, unsigned char bufptr[], int * bytes)
/* User is prepared to receive a message. */
{
    struct conn * cptr = &conn[cid];

    if (cptr->clr_req_received == 0) {
        /* Connection still established; try to receive. */
        cptr->state = RECEIVING;
        cptr->user_buf_addr = bufptr;
        cptr->byte_count = 0;
        data[0] = CRED;
        data[1] = 1;
        to_net(cid, 1, 0, CREDIT, data, 2); /* send credit */
        sleep(); /* block awaiting data */
        * bytes = cptr->byte_count;
    }
}

```

图 6.20 (续)



---

```

    }
    cptr->state = ESTABLISHED;
    return(cptr->clr_req_received ? ERR_CLOSED : OK);
}

int disconnect(int cid)
{ /* User wants to release a connection. */
    struct conn * cptr = &conn[cid];

    if (cptr->clr_req_received) { /* other side initiated termination */
        cptr->state = IDLE; /* connection is now released */
        to_net(cid, 0, 0, CLEAR_CONF, data, 0);
    } else { /* we initiated termination */
        cptr->state = DISCONN; /* not released until other side agrees */
        to_net(cid, 0, 0, CLEAR_REQ, data, 0);
    }
    return(OK);
}

void packet_arrival(void)
{ /* A packet has arrived, get and process it. */
    int cid; /* connection on which packet arrived */
    int count, i, q, m;
    pkt_type ptype;
    /* CALL_REQ, CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT, CREDIT */
    unsigned char data[MAX_PKT_SIZE]; /* data portion of the incoming packet */
    struct conn * cptr;

    from_net(&cid, &q, &m, &ptype, data, &count); /* go get it */
    cptr = &conn[cid];

    switch (ptype) {
        case CALL_REQ: /* remote user wants to establish connection */
            cptr->local_address = data[0]; cptr->remote_address = data[1];
            if (cptr->local_address == listen_address) {
                listen_conn = cid; cptr->state = ESTABLISHED; wakeup();
            } else {
                cptr->state = QUEUED; cptr->timer = TIMEOUT;
            }
            cptr->clr_req_received = 0; cptr->credits = 0;
            break;

        case CALL_ACC: /* remote user has accepted our CALL_REQ */
            cptr->state = ESTABLISHED;

```

图 6.20 (续)

```

        wakeup();
        break;

    case CLEAR_REQ:                /* remote user wants to disconnect or reject call */
        cptr->clr_req_received = 1;
        if (cptr->state == DISCONN) cptr->state = IDLE; /* clear collision */
        if (cptr->state == WAITING || cptr->state == RECEIVING || cptr->state ==
            SENDING) wakeup();
        break;

    case CLEAR_CONF:                /* remote user agrees to disconnect */
        cptr->state = IDLE;
        break;

    case CREDIT:                    /* remote user is waiting for data */
        cptr->credits += data[1];
        if (cptr->state == SENDING) wakeup();
        break;

    case DATA_PKT:                 /* remote user has sent data */
        for (i = 0; i < count; i++) cptr->user_buf_addr[cptr->byte_count + i] = data[i];
        cptr->byte_count += count;
        if (m == 0) wakeup();
    }
}

void clock(void)
{ /* The clock has ticked, check for timeouts of queued connect requests. */
    int i;
    struct conn *cptr;
    for (i = 1; i <= MAX_CONN; i++) {
        cptr = &conn[i];
        if (cptr->timer > 0) { /* timer was running */
            cptr->timer--;
            if (cptr->timer == 0) { /* timer has now expired */
                cptr->state = IDLE;
                to_net(i, 0, 0, CLEAR_REQ, data, 0);
            }
        }
    }
}

```

图 6.20 (续)

当以下任何一种事件发生的时候,状态之间可能发生迁移:执行一个原语、一个分组到达或者定时器过期。

图 6.20 中显示的过程有两种类型。大多数可以被用户程序直接调用。然而, packet\_arrival 和 clock 有所不同。它们本质上是由外部事件来触发的, 分别为一个分组到达和时钟滴答中断。实际上, 它们是中断程序。我们假定当一个传输实体过程正在运行的时候, 这两个中断过程永远不会被调用。只有当用户进程在睡眠或者在传输实体的外部执行的时候, 它们才可能被调用。这个特性对于例子代码的功能正确性是非常重要的。

由于在分组头中存在 Q(Qualifier, 质量)位, 所以, 我们可以避免传输协议头的开销。普通的数据消息是被作为  $Q=0$  的数据分组来发送的; 而传输协议的控制消息则是被作为  $Q=1$  的数据分组来发送的, 在我们的例子中, 控制消息只有一种(CREDIT)。接收端的传输实体可以检测到这些控制消息, 并对它们进行处理。

传输实体使用的主要数据结构是数组 conn, 每个潜在的连接都有一条相应的记录。该记录维护了连接的状态, 包括两端的传输地址、在这个连接上发送和接收的消息数、当前的状态、用户缓冲区的指针、到现在为止发送或者接收的消息的字节数、一个标志位说明远程用户是否已经发出了 DISCONNECT、一个定时器, 以及一个计数器记录了当前可允许发送的消息数。在我们的简单例子中, 并不是所有这些域都用到了, 但是一个完全的传输实体将需要所有这些消息, 甚至可能还要更多的信息。假定每个 conn 记录都被初始化到 IDLE 状态。

当用户调用 CONNECT 的时候, 网络层接到指令, 要给远程机器发送一个 CALL REQUEST 分组, 而用户则被置入睡眠状态。当这个 CALL REQUEST 分组到达另一端的时候, 系统产生一个中断, 然后传输实体的 packet\_arrival 被调用, 它检查本地用户是否正在监听指定的地址。如果是的话, 则送回一个 CALL ACCEPTED 分组, 于是远程用户被唤醒; 如果不是, 则 CALL REQUEST 分组被排入队列, 并等待 TIMEOUT 个时钟滴答。如果在这段时间内有某个传输用户调用 LISTEN 的话, 则连接可以建立起来; 否则的话, 此连接请求超时, 该请求被拒绝, 并且主机返回一个 CLEAR REQUEST 分组, 以免它永远阻塞下去。

尽管我们去掉了传输协议的头部, 但是, 我们仍然需要一种办法来记录哪个分组属于哪个传输连接, 因为多个连接可能同时并存。最简单的做法是使用网络层虚电路号作为传输连接号。而且, 虚电路号也可以被用作 conn 数组中的索引。当网络层 k 号虚电路上有一个分组到来的时候, 它属于传输连接 k, 该连接的状态被记录在 conn[k] 中。对于一台主机上主动发起的连接而言, 连接号是由发起连接的传输实体选择的; 而对于进来的连接请求, 连接号则是由网络层选择的, 网络层可以选择任何一个尚未使用的虚电路号。

为了避免在传输实体内部必须要提供和管理缓冲区, 这里我们使用了一种不同于常规滑动窗口的流控制机制。当一个用户调用 RECEIVE 的时候, 传输实体发送一个特殊的信用消息(credit message)给对方机器上的传输实体, 并由对方的传输实体将信用值记录在其 conn 数组中。当 SEND 被调用的时候, 传输实体查看在指定的连接上是否已经到达了一个信用消息。如果确实到达了的话, 则发送此消息(若有必要, 放到多个分组中), 并且递减信用值。如果没有的话, 传输实体自身进入到睡眠状态, 直至信用消息到来。这种机制保证了: 除非另一方已经完成了一个 RECEIVE 原语, 否则就不发送任何消息。其结果是, 任何时候当一个消息到达时, 总是保证有一个缓冲区可以使用, 从而可

把新到达的消息放到这个缓冲区中。我们很容易就可以将这种方案进一步泛化,从而允许接收方提供多个缓冲区和请求多个消息。

你应该记住,图 6.20 中的代码只是一个简单的传输实体实现。一个真正的传输实体通常需要检查所有由用户提供的参数的有效性、要处理网络层崩溃的恢复问题、处理请求的碰撞问题,还要支持一个更加一般化的传输服务,包括诸如中断、数据报、SEND 和 RECEIVE 原语的非阻塞版本之类的设施。

### 6.3.3 传输实体作为一个有限状态机的示例

编写传输实体是一项困难而又要求严格的任务,尤其是针对更加实用的协议。为了减少出错的几率,用有限状态机来表达协议的状态往往是非常有用的。

前面已经看到了,在我们的例子协议中,每个连接有 7 种状态。我们也可能分离出 12 种事件,通过这些事件将一个连接从一种状态转移到另一种状态。其中 5 种事件正好是 5 个服务原语。另外 6 种事件是 6 种合法分组的到达。最后一个是定时器超时。图 6.21 以矩阵的形式显示了主要的协议动作。这里的列代表了状态,行代表了 12 种事件。

图 6.21 的矩阵(即有限状态机)中的每个项目至多有三个域:一个断言、一个动作和一个新状态。断言指明了它的动作将在什么条件下发生。例如,在左上角的项目中,如果 LISTEN 被执行但是没有空余的表空间(断言 P1)的话,则 LISTEN 失败,状态不改变。另一方面,如果一个 CALL REQUEST 分组到达一个正在被监听的传输地址(断言 P2),则连接立即被建立起来。另一种可能性是,P2 是假的,即没有 CALL REQUEST 到来,在这种情况下,连接仍然处于 IDLE 状态,它继续等待一个 CALL REQUEST 分组。

值得指出的是,在矩阵中选择的这些状态并非完全是由协议本身决定的。在这个例子中,并没有 LISTENING 状态,也许有人认为在 LISTEN 之后增加一种 LISTENING 状态是非常合理的。这里之所以没有 LISTENING 状态,是因为每一种状态都与一个连接记录项关联在一起,但是 LISTEN 并没有创建连接记录。为什么不创建呢?因为我们已经决定使用网络层的虚电路号作为连接的标识符,而对于一个 LISTEN,虚电路号最终是在 CALL REQUEST 分组到来的时候由网络层选择的。

动作 A1 至 A12 是主要的动作,比如发送分组、启动定时器等。这里并没有列出所有次要的动作,比如初始化一个连接记录的域。如果一个动作涉及到唤醒一个正在睡眠的进程,则唤醒之后的动作也被考虑在前一动作之内。例如,如果一个 CALL REQUEST 分组进来,而此时一个进程正在睡眠中等待该分组,那么,紧跟在唤醒之后的动作,即发送 CALL ACCEPT 分组,也被当作 CALL REQUEST 动作的一部分。在每个动作完成之后,连接可能会转移到一种新的状态,如图 6.21 所示。

用矩阵来表达协议有三方面的优点。第一,在这种形式中,程序员很容易系统地检查状态和事件的每一种组合,以确定一个动作是否有必要。在产品化的协议实现中,有些组合可能被用于错误处理。在图 6.21 中,我们没有区分不可能的情形和非法的情形。例如,如果一个连接处于 WAITING(等待)状态之中,则 DISCONNECT 事件是不可能的,因为用户被阻塞住,它根本不可能再执行任何原语。另一方面,在 SENDING(发送)状态中,它不希望看到数据分组,因为信用消息还没有发出来。如果到达一个数据分组,则认

		状态						
		空闲	等待	排队	建立	发送	接收	断开
原语	LISTEN	P1:~/空闲 P2:A1/建立 P2:A2/空闲		~建立				
	CONNECT	P1:~/空闲 P1:A3/等待						
	DISCONNECT				P4:A5/空闲 P4:A6/断开			
	SEND				P5:A5/建立 P5:A6/发送			
	RECEIVE				A9/接收			
进来的分组	Call_req	P3:A1/建立 P3:A4/排队						
	Call_acc		~/建立					
	Clear_req		~/空闲		A10/建立	A10/建立	A10/建立	~/空闲
	Clear_conf							~/空闲
	DataPkt						A12/建立	
时钟	Credit				A11/建立	A7/建立		
	超时			~/空闲				

#### 断言

P1: 连接表满  
P2: Call\_req 正在进行 (未完成)  
P3: LISTEN 正在进行 (未完成)  
P4: Clear\_req 正在进行 (未完成)  
P5: 信用可以使用

#### 动作

A1: 发送 Call\_acc  
A2: 等待 Call\_req  
A3: 发送 Call\_req  
A4: 启动定时器  
A5: 发送 Clear\_conf  
A6: 发送 Clear\_req  
A7: 发送消息  
A8: 等待信用  
A9: 发送信用  
A10: 设置 Clear\_req\_received 标志  
A11: 记录信用  
A12: 接受消息

图 6.21 有限状态机形式的例子协议

每个项目都有一个可选的断言(predicate)、一个可选的动作和新的状态; 符号~表示没有大的动作发生;

断言上面有一条横线表示该断言的否定; 空白项目对应于不可能或者无效的事件。

为是一个协议错误。

用矩阵来表达协议的第二个优点体现在实现协议的时候。你可以想象一个二维数组, 它的元素  $a[i][j]$  是一个指向某一个过程的指针或者索引, 该过程专门处理当处于状态  $j$  时发生事件  $i$  的情形。一种可能的实现方法是传输实体编写成一个短的循环, 在循环的最上边等待一个事件。当事件发生的时候, 首先定位到相关的连接上, 然后将它的状态提取出来。知道了事件和状态之后, 传输实体只要索引到数组  $a$  中, 再调用相应的过程



即可。这种实现方法给出了一种比我们的传输实体更加规范和系统化的设计思路。

有限状态机方法的第三个优点体现在协议描述上。在有些标准的文档中,协议是以图 6.21 中的有限状态机的形式来给出的。如果传输实体是由有限状态机来驱动的,而程序中的有限状态机又是建立在标准中的有限状态机基础之上的,那么,从这种描述出发,实现一个可以工作的传输实体将会非常容易。

有限状态机方法的主要缺点是,它可能比我们前面例子中采用的直接编程方法更加难以理解。然而,我们可以将有限状态机画成一个图,如图 6.22 所示,从而在某种程度上解决这个问题。

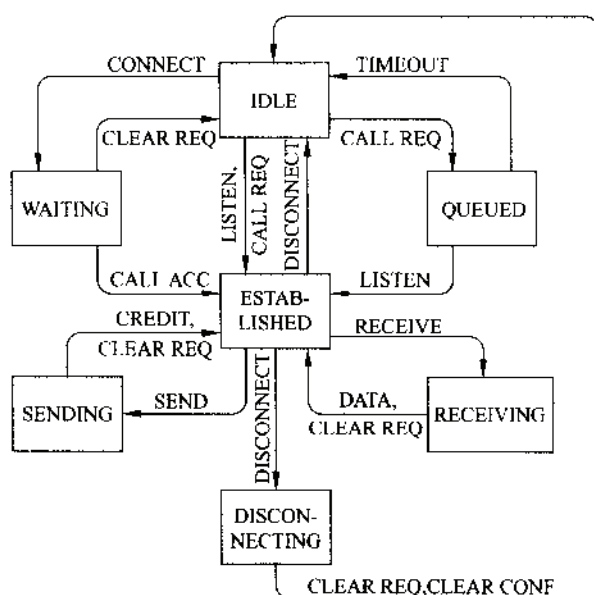


图 6.22 协议示例的图形表示(为了简化起见,那些保持状态不变的迁移被省略掉了)

## 6.4 Internet 传输协议—UDP

Internet 的传输层上有两个主要协议,一个无连接的协议和一个面向连接的协议。在接下去的两节中,我们将学习这两个协议。无连接的协议是 UDP;面向连接的协议是 TCP。因为 UDP 基本上只不过是 IP 再加一个短的头,所以,我们先从 UDP 开始。同时我们也将讨论 UDP 的两个应用。

### 6.4.1 UDP 介绍

Internet 协议族支持一个无连接的传输协议:UDP(User Datagram Protocol,用户数据报协议)。UDP 为应用程序提供了一种方法来发送经过封装的 IP 数据报,而且不必建立连接就可以发送这些 IP 数据报。RFC 768 描述了 UDP。

UDP 传输的数据段(segment)是由 8 字节的头和净荷域构成的。图 6.23 描述了头

信息。两个端口分别被用来标识出源机器和目标机器内部的端点。当一个 UDP 分组到来的时候,它的净荷部分被递交给与目标端口相关联的那个进程。这种关联关系是在调用了 BIND 原语或者其他某一种类似的做法之后建立起来的,我们在图 6.6 中介绍 TCP 的时候已经看到过了(UDP 的绑定过程也是一样的)。实际上,采用 UDP 而不是原始的 IP,其最主要的价值是增加了源端口和目标端口。如果没有端口域,则传输层将不知道该如何处理分组;而有了端口域之后,它就可以正确地递交数据段了。

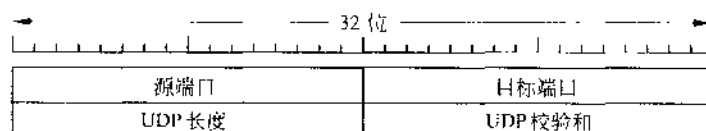


图 6.23 UDP 头

当目标端必须将一个应答送回给源端的时候,源端口是必需的。发送应答的进程只要将进来的数据段中的 source port(源端口)域复制到输出的数据段中的 destination port(目标端口)域,就可以指定在发送方机器上由哪个进程来接收应答。

UDP length(UDP 长度)域包含了 8 字节的头和数据部分。UDP checksum(UDP 校验和)是可选的,如果不计算的话,则在该域中存放 0(如果真正的计算结果是 0 的话,则该域中存放的是全 1)。除非数据的质量并不要紧(比如数字化的语音),否则就不应该将校验和功能关闭。

另外值得明确地提出来的可能是 UDP 没有做到的一些事情。UDP 并不考虑流控制、错误控制,在收到一个坏的数据段之后它也不重传。所有这些工作都留给用户进程。UDP 所做的事情是提供一个接口,并且在接口中增加解复用(demultiplexing)的特性。它利用端口的概念将数据段解复用到多个进程中。这就是它所做的全部工作。对于那些需要精确地控制分组流,或者需要错误控制,或者需要时间控制的应用来说,UDP 提供的仅仅是“医嘱”而已。

UDP 尤其适用的一个领域是在客户-服务器的情形下。通常,客户给服务器发送一个短的请求,并且期望一个短的应答回来。如果这里的请求或者应答丢失的话,客户就会超时,于是它只要重试即可。不仅代码编写起来简单,而且只需要很少的消息(每个方向一条消息),比起那些要求先建立连接的协议来说简单多了。

用这种方式来使用 UDP 的一个应用是 DNS(Domain Name System,域名系统),我们将在第 7 章中学习 DNS。简单来说,如果一个程序需要根据某一个主机名(比如 www.cs.berkeley.edu)来查找它的 IP 地址,那么,它可以给 DNS 服务器发送一个包含该主机名的 UDP 分组。服务器用一个包含该主机 IP 地址的 UDP 分组作为应答。事先不需要建立连接,事后也不需要释放连接。在网络上只要两条消息就够了。

#### 6.4.2 远过程调用

从某种意义上,向一台远程主机发送一个消息并获得一个应答,就如同在编程语言中执行一个函数调用一样。在这两种情形下,你都要提供一个或者多个参数,然后获得一个

结果。这种现象导致人们试图将网络上的请求-应答交互过程,做成像过程调用那样可以进行类型匹配和转换。这样的结构使得网络应用更加易于编程,而且人们对这种处理方式也更加熟悉。例如,请想象一个名为 `get_IP_address(host_name)` 的过程,它的工作方式为:向 DNS 服务器发送一个 UDP 分组,然后等待应答,如果在规定时间内没有接收到应答的话,则超时并重试。通过这种方式,网络的所有细节对于程序员而言全部隐藏起来了。

这个领域中的关键工作是由 Birrell 和 Nelson(1984)完成的。简单来说,Birrell 和 Nelson 的建议是,允许本地的程序调用远程主机上的过程。当机器 1 上的进程调用机器 2 上的一个过程的时候,机器 1 上的调用进程被挂起,而机器 2 上被调用的过程则开始执行。参数信息从调用方传输到被调用方,而过程的执行结果则从反方向传递回来。对于程序员而言,所有的消息传递都是不可见的。这项技术称为 **RPC(Remote Procedure Call, 远过程调用)**,目前已经成为许多网络应用的基础。按照传统的概念,调用过程称为客户,被调用的过程称为服务器,本节我们也将使用这些名字。

RPC 背后的思想是,尽可能地使一个远过程调用看起来像本地过程调用一样。在最简单的形式中,为了调用一个远过程,客户程序必须绑定(链接)一个小的库过程,这个小的库过程称为**客户存根(client stub)**,它位于客户地址空间中,但是代表了服务器过程。类似地,服务器需要绑定一个称为**服务器存根(server stub)**的过程。正是这些过程,隐藏了从客户到服务器的过程调用的远程特性。

在执行 RPC 过程中,实际的步骤如图 6.24 所示。步骤 1 是客户调用客户存根。这是一个本地过程调用,其参数被按照常规的方式压入到栈中。在步骤 2 中,客户存根将参数包装到一个消息中,然后通过一个系统调用来发送该消息。包装参数的过程被称为**列集(marshaling)**。在步骤 3 中,内核将消息从客户机器发送到服务器机器上。在步骤 4 中,内核将进来的分组传递给服务器存根。最后,在步骤 5 中,服务器存根利用散集(unmarshaling)得到的参数调用服务器过程。调用的结果沿着相反的方向按同样的路径传递。

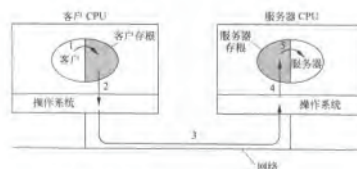


图 6.24 执行远过程调用的步骤(阴影部分为存根)

这里需要说明的关键一点是,用户编写的客户过程只是按照普通过程调用的方式来调用客户存根,而且客户存根与服务器过程有同样的名字。由于客户过程和客户存根在

同样的地址空间中,所以,参数的传递也是按通常的方式进行的。类似地,服务器过程被同一地址空间中的一个过程所调用,并且它接收到的也正好是它所期望的参数。按照这种方式,网络通信的输入和输出并不是在套接字上完成的,而是通过仿造一个普通的过程调用来完成。

尽管 RPC 有非常精巧的概念模型,但是,这其中也隐藏着一些陷阱。最大的陷阱是指针参数的用法。在正常情况下,将指针传递给一个过程并不是一个问题。被调用的过程可以像调用方那样使用这个指针,因为双方生存在同样的虚拟地址空间中。而对于 RPC,传递指针是不可能的,因为客户和服务器位于不同的地址空间中。

在有些情况下,可以使用一些技巧来实现指针的传递。假设第一个参数是一个指向某个整数  $k$  的指针。客户存根可以对  $k$  进行列集,并发送给服务器。然后,服务器存根创建一个指向  $k$  的指针,并且将该指针传递给服务器过程,而这正是服务器过程所期望的。当服务器过程将控制权返回给服务器存根的时候,后者将  $k$  送回给客户;在客户端,新的  $k$  值被复制到老的  $k$  中,就好像服务器对它做了改变一样。实际上,这个过程相当于用“复制-恢复(copy-restore)”机制代替了按引用调用(call-by-reference)的标准调用序列。不幸的是,这种技巧并不总是可以正确地工作,例如,若指针指向一个图形或者其他复杂的数据结构。由于这个原因,对于可被远程调用的过程,其参数必须强加一些限制。

第二个问题是,在一些弱类型的语言,比如 C 中,编写一个计算两个矢量(即数组,但不指定矢量的大小)内积的过程是完全合法的。每个矢量都有一个专门的值作为终止标记,而且该终止值只有调用过程和被调用过程才知道。在这样的条件下,客户存根要想对参数执行列集操作从本质上讲是不可能的:它不知道该如何确定参数的长度。

第三个问题是,在实践中,要想推断出参数的类型并不总是可能的,即使你从正式的规范或者代码本身也不见得能做得好。一个例子是 printf,它可能有任意多个参数(至少一个),而且这些参数可以是整数、短整数、长整数、字符、字符串、可变长度的浮点数和其他类型的任意组合。试图以远过程的方式来调用 printf 是不太现实的,因为 C 语言对类型的要求特别宽松。然而,有一条规则这样说:只要你不像在 C(或者 C++)中那样编写程序,那你就可以使用 RPC,其实这种说法也并不能被广泛接受。

第四个问题涉及到全局变量的使用。正常情况下,调用过程和被调用过程除了通过参数进行通信以外,也可以使用全局变量作为通信的手段。如果被调用的过程现在转移到了远程机器上,那么,这样的代码就会失败,因为全局变量不再为双方所共享了。

这些问题并不意味着 RPC 就没有希望了。事实上,它的应用仍然十分广泛,但是在实践中,需要有一些限制才能保证它工作得很好。

当然,RPC 不一定非得使用 UDP 分组,但是,RPC 和 UDP 是一对很好的搭档,而且,UDP 常常被用于 RPC。然而,当参数或者结果值可能超过最大的 UDP 分组的时候,或者当所请求的操作并不幂等(即不能安全地重复多次执行,比如计数器递增的操作)的时候,可能有必要建立一个 TCP 连接,然后利用该连接来发送请求,而不是使用 UDP 来完成远程调用。

### 6.4.3 实时传输协议

客户-服务器 RPC 是 UDP 被广泛应用的一个领域,另一个领域是实时多媒体应用。尤其是,随着 Internet 广播电台、Internet 电话、音乐点播、视频会议、视频点播和其他的多媒体应用越来越普及,人们发现每一种应用都在重复设计几乎相同的实时传输协议。逐渐地人们意识到,为多媒体应用制定一个通用的实时传输协议是一种很好的想法,因此就诞生了 RTP(Real-time Transport Protocol,实时传输协议)。RFC 1889 描述了 RTP 协议,目前该协议已经得到了广泛的应用。

RTP 在协议栈中的位置有点古怪。最终 RTP 被放在用户空间中,并且(通常)运行在 UDP 之上。它的操作方式如下所述。多媒体应用通常包含多个音频、视频、文字流,可能还包括其他的流。这些流被送入到 RTP 库中,而 RTP 库位于多媒体应用的用户空间中。然后,RTP 库将这些流复用用到 RTP 分组中,同时也对它们进行编码,然后,这些 RTP 分组被填充到一个套接字中。在套接字的另一端(即套接字的下面接口部分,位于操作系统的内核中)生成 UDP 分组,然后这些 UDP 分组被嵌入到 IP 分组中。如果当前计算机是在以太网上,则这些 IP 分组被放到以太网帧中以便传输出去。图 5.25(a)显示了这种情况下的协议栈。分组的嵌套情况如图 5.25(b)所示。

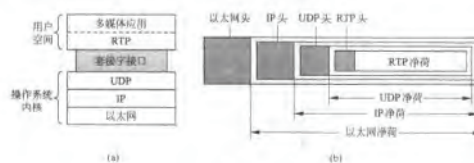


图 5.25

(a) RTP 在协议栈中的位置; (b) 分组嵌套情况

这种设计的结果是,你很难说得清 RTP 位于哪一层上。由于它运行在用户空间,并且被链接到应用程序中,所以,它无疑看起来像是一个应用协议。另一方面,它又是一个通用的、与具体应用无关的协议,它仅仅提供了一些传输设施,所以,它看起来也像一个传输协议。可能最恰当的描述是:它是一个在应用层上实现的传输协议。

RTP 的基本功能是将几个实时数据流复用到一个 UDP 分组流中。这个 UDP 流可以被发送给一台目标主机(单播传输模式),也可以被发送给多台目标主机(多播传输模式)。因为 RTP 仅仅使用了常规的 UDP,所以路由器并不会对它的分组有任何特殊的对待,除非路由器支持一些常规的 IP 服务质量特性。特别是,对于递交的可靠性、抖动等特性并没有特别的保证。

在 RTP 流中发送的每个分组被赋予一个编号,而且每个分组的编号比它前面的分组

大1。这种编号机制使得目标端能够确定是否有分组丢失了。如果一个分组被丢失了,则目标端能够采取的最佳动作是利用插值法近似地估计出已被丢失的中间值。重传并不是一种切合实际的选择方式,因为重传的分组可能会到达得很晚,那时已经不再有用了。因此,RTP没有流控制,没有错误控制,没有确认,也没有请求重传的机制。

每个RTP净荷可能包含多个样本,它们可以按照应用系统期望的任何一种方式进行编码。为了允许网络互连,RTP定义了几种配置轮廓(比如单音频流),而且,对于每一种轮廓,可以允许多种编码格式。例如,一个单音频流的编码方式有:8kHz的8位PCM采样、增量编码、预测编码、GSM编码、MP3,等等。RTP提供了一个头域可以让源端用来指定编码方法,但除此以外,RTP不再涉及到编码的工作方式。

除了编码以外,许多实时应用还需要另一种设施,即时间戳机制。RTP的思路是,允许源端将一个时间戳与每个分组中的第一个样本关联起来。这里的时间戳是相对于整个流的起始时间,因此,仅仅时间戳之间的差值才是重要的,时间戳的绝对值并没有意义。这种机制使得目标端可以做少量的缓冲工作,然后在整个流开始之后正确的毫秒数时间点上播放每一个样本,因此,每个样本所在分组的到达时间与样本的播放时间之间保持一定的独立性。时间戳机制不仅减小了抖动的影响,而且也允许多个流相互之间可以同步到一起。例如,一个数字电视程序可能有一个视频流和两个音频流。两个音频流中的内容可能是立体声广播,也可能是电影节目中的两个语言声道(一个声道是原始的语种,另一个声道是本地语言的配音),从而让观众有选择语言的机会。每个流分别来自于不同的物理设备,但是,如果它们的时间戳来自于同一个计数器的话,那么,即使这些流的传输过程稍微有一点不太规律,它们仍然可以非常同步地播放出来。

图6.26显示了RTP头的结构。它包含3个32位的字,并可能有一些扩展域。第一个字包含了Version(版本)域,现在版本号已经到达2了。我们希望这个版本非常接近于最终的版本,因为这里只剩下一个值(即3)的编号空间了(不过,我们也可以将3定义成特殊的含义,它代表实际的版本号在一个扩展字中)。

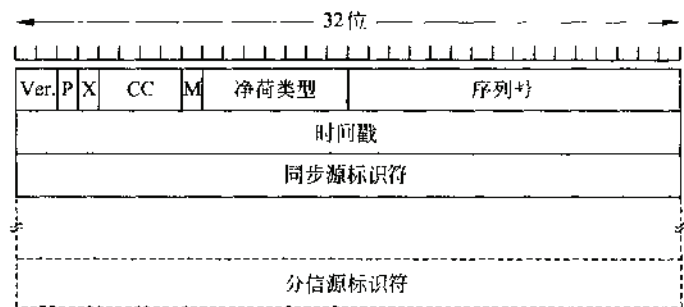


图 6.26 RTP 头

P位表示该分组已经被填充到4字节的倍数。最后的填充字节指明了有多少个字节被填充进来。X位表示出现了一个扩展头。扩展头的格式和含义没有定义。惟一已定义的事情是:扩展头的第一个字指定了扩展头的长度。这是一种针对任何不可预见之需求的安全措施。



CC 域指明了后面共有多少个分信源(contributing source),可以从 0 到 15(见下面进一步的介绍)。M 位是一个与应用相关的标记位,它可以被用来标记一个视频帧的开始、音频信道中一个字的开始,或者其他由应用来理解的某一件事情。Payload type(净荷类型)指出该分组使用了哪一种编码算法(比如未压缩的 8 位音频、MP3,等等)。由于每个分组都带有这个域,所以,在传输过程中编码方法可以改变。Sequence number(序列号)只是一个计数器,在每个被发送出去的 RTP 分组中该计数器都要递增。该域可被用来检测丢失的分组。

时间戳是由 RTP 流的源产生的,它注明了该分组中的第一个样本是什么时候产生的。这个值可以有助于减缓接收方的抖动,其做法是:将播放的时间点与分组的到达时间点分离开。Synchronization source identifier(同步源标识符)指明了该分组属于哪一个流。正是通过这个域,才可以将多个数据流复用到一个 UDP 分组流中,或者从一个 UDP 分组流中解复用出多个数据流。最后,如果在媒体制作过程中使用了混合器的话,则可以使用 Contributing source identifier(分信源标识符)。在这种情况下,混合器是同步源,被混合的流列在这里。

RTP 有一个姊妹协议,称为 RTCP(Realtime Transport Control Protocol,实时传输控制协议)。它处理反馈、同步和用户界面等,但是不传输任何数据。它的第一个功能是可以用来向源端提供有关延迟、抖动、带宽、拥塞和其他网络特性的反馈信息。编码进程可以充分利用这些信息,因此,当网络状况比较好的时候,它可以提高数据速率(从而达到更好的质量),而当网络状况不好的时候,它可以减低数据速率。通过提供连续的反馈信息,编码算法可以持续地作相应的调整,从而在当前条件下提供尽可能最佳的质量。例如,如果在传输过程中带宽有时候增加,有时候减少,那么,编码算法可以根据要求从 MP3 切换到 8-位 PCM 或者切换到增量编码。利用 Payload type 域可以告诉目标端当前分组使用的是哪一种编码算法,因而有可能根据需要动态地改变算法。

RTCP 也处理流之间的同步。问题是,不同的流有可能使用不同的时钟,并且有不同的粒度和不同的漂移速率。利用 RTCP 可使它们保持同步。

最后,RTCP 也提供了一种方法来命名各个源(比如使用 ASCII 字符)。这些信息可以显示在接收方的屏幕上,从而提醒用户当前正在跟谁通话。

有关 RTP 的更多信息,请参考(Perkins, 2002)。

## 6.5 Internet 传输协议—TCP

UDP 是一个简单的协议,它有一些非常合适的用途,比如客户-服务器交互过程和多媒体应用,但是对于大多数 Internet 应用来说,它们更需要可靠的、按序递交的特性。UDP 不能提供这样的功能,所以 Internet 还需要另一个协议,这就是 TCP,目前它是 Internet 上承担任务最为繁重的一个协议。现在我们来详细地学习这个协议。

### 6.5.1 TCP 介绍

TCP(Transmission Control Protocol,传输控制协议)是专门为了在不可靠的互联网络上提供一个可靠的端到端字节流而设计的。互联网络与单个网络不同,因为互联网络的

不同部分可能有截然不同的拓扑、带宽、延迟、分组大小和其他的参数。TCP 的设计目标是能够动态地适应互联网络的这些特性,而且当面对多种失败的时候仍然足够健壮。

TCP 的原始正式定义位于 RFC 793 中。随着时间的推移,各种错误和不一致性逐渐被检测出来,而且在某些领域中需求也发生了变化。RFC 1122 详细地阐述了这些内容以及一些错误修补方案。RFC 1323 又进一步作了扩展。

每台支持 TCP 的机器都有一个 TCP 传输实体,它或者是一个库过程,或者是一个用户进程,或者是内核的一部分。在所有这些情形下,它管理 TCP 流,以及与 IP 层之间的接口。TCP 传输实体接受本地进程的用户数据流,并且将它们分割成不超过 64KB(在实践中,考虑到在每个帧中都希望有 IP 和 TCP 头,所以通常不超过 1460 数据字节)的分片,然后以单独的 IP 数据报的形式发送每一个分片。当包含 TCP 数据的数据报到达一台机器的时候,它们被递交给 TCP 传输实体,然后 TCP 传输实体再重构出原始的字节流。为了简化起见,我们有时候仅仅用“TCP”来代表 TCP 传输实体(软件片断)或者 TCP 协议(一组规则)。通过上下文你应该可以很清楚地推断出其实际的含义。例如,在“用户将数据交给了 TCP”这句话中,很显然这里指的是 TCP 传输实体。

IP 层并不保证数据报一定被正确地递交到目标端,所以,TCP 需要判断超时的情况,并且根据需要重传数据报。即使被正确递交的数据报,也可能存在错序的问题,所以,这也是 TCP 的责任,它必须把接收到的数据报按照正确的顺序重新装配成用户的消息。简而言之,TCP 必须提供可靠性,这也正是大多数用户所期望的而 IP 又没有提供的功能。

### 6.5.2 TCP 服务模型

正如我们在 6.1.3 节中所讨论的,要想获得 TCP 服务,发送方和接收方必须创建一种被称为套接字的端点。每个套接字有一个套接字号(地址),它是由主机的 IP 地址以及本地主机局部的一个 16 位数值组成的,此 16 位数值被称为端口(port)。端口是一个 TSAP 的 TCP 名字。为了获得 TCP 服务,首先必须要显式地在发送机器的套接字和接收机器的套接字之间建立一个连接,有关套接字的调用如图 6.5 中所列。

一个套接字有可能同时被用于多个连接。换句话说,两个或者多个连接可能终止于同一个套接字。每个连接可以用两端的套接字标识符来标识,即(socket1, socket2)。TCP 不使用虚电路号或者其他的标识符。

1024 以下的端口号被称为知名端口(well-known port),它们被保留用于一些标准的服务。例如,如果一个进程希望与某一台主机建立一个连接以便利用 FTP 传输一个文件,那么,它可以连接到目标主机的 21 号端口,从而与它的 FTP 守护进程(daemon)建立联系。你可以在 [www.iana.org](http://www.iana.org) 上找到所有知名端口的列表。目前已经分配了 300 多个。图 6.27 中列出了一些尤为知名的端口。

我们当然可以让 FTP 守护进程在系统启动的时候就关联到 21 号端口,让 telnet 守护进程关联到 23 号端口,等等。然而,这样做将会使内存散乱在这些守护进程中,而且大多数时间这些进程都是空闲的。因此,一般的做法是让一个守护进程(在 UNIX 中称为 inetd,代表“Internet daemon”的意思)同时关联到多个端口上,然后等待第一个进来的连

端口	协议	用途
21	FTP	文件传输
23	Telnet	远程登录
25	SMTP	电子邮件
69	TFTP	简单文件传输协议(Trivial FTP)
79	Finger	查询有关一个用户的信息
80	HTTP	万维网(World Wide Web 或者 WWW)
110	POP-3	远程电子邮件访问
119	NNTP	USENET 新闻

图 6.27 一些已被分配的端口

接。当第一个连接进来的时候,inetd 分叉出(fork)一个新的进程,并且在这个进程中执行适当的守护程序,然后由这个守护程序来处理该连接请求。按照这种做法,对于除了 inetd 以外的守护程序,只有当确实有工作需要它们来完成的时候,它们才被激活。Inetd 是通过一个配置文件才知道对于哪个端口应该使用哪个守护程序。因此,系统管理员可以这样来配置系统:对于比较忙的端口(比如 80 端口)使用永久的守护程序,而其他的端口则让 inetd 来处理。

所有的 TCP 连接都是全双工的,并且是点到点的。所谓全双工,意味着同时可在两个方向上传输数据;而点到点则意味着每个连接恰好有两个端点。TCP 并不支持多播或者广播传输模式。

一个 TCP 连接就是一个字节流,而不是消息流。端到端之间并不保留消息的边界。例如,如果发送进程将 4 个 512 字节的数据块写到一个 TCP 流中,那么,在接收进程中,这些数据有可能按 4 个 512 字节块的方式被递交,也可能是 2 个 1024 字节的数据块,或者是 1 个 2048 字节的数据块(见图 6.28),或者其他的方式。接收方无法获知这些数据被写入字节流时候的单元大小。

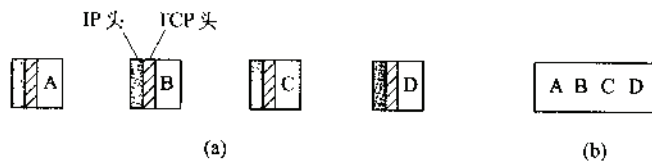


图 6.28

(a) 以单独 IP 数据报的形式发送 4 个 512 字节的数据段;

(b) 在一个 READ 调用中,这 2048 字节的数据被一次递交给应用程序

UNIX 中的文件也具有这样的特性。读文件的程序无法判断该文件是一次被写入一个块,还是一次写入一个字节,或者整个文件被一次性写入的。如同 UNIX 文件的情形一样,TCP 软件不理解 TCP 字节流的含义,而且也无意于弄清其含义。一个字节只是一

个字节而已。

当一个应用将数据传递给 TCP 的时候, TCP 可能立即将数据发送出去, 也可能将它缓冲起来(为了收集更多的数据从而一次发送出去), 这完全由 TCP 软件自己来决定。然而, 有时候, 应用程序确实希望自己的数据立即被发送出去, 例如, 假设一个用户已经登录到一台远程机器上, 用户每输入一行命令就会敲入回车键, 这时候该命令行应该被立即发送至远程主机, 而不应该被缓冲起来等待下一行命令。为了强迫将数据发送出去, 应用程序可以使用 PUSH 标志, 它相当于告诉 TCP 不要延迟传输过程。

有些早期的应用把 PUSH 标志当作一种划分消息边界的标记符来使用。虽然这种做法有时候可以正常工作, 但有时候它会失败, 因为并不是所有的 TCP 实现都把 PUSH 标志传送给接收端的应用程序。而且, 如果在第一个 PUSH 标志被发送出去之前又有一些 PUSH 标志到来的话(比如因为输出线路忙而造成这样的情形), 那么, TCP 可能会将所有已被 PUSH 的数据收集起来放到一个 IP 数据报中, 因而这些分片之间不再存在隔离标志了。

有关 TCP 服务最后一个值得在这里提出来的特性是紧急数据(urgent data)。当一个交互用户通过敲入 DEL 或者 CTRL-C 键来打断一个已经开始运行的远程计算过程的时候, 发送方应用把一些控制信息放在数据流中, 然后将它连同 URGENT 标志一起交给 TCP。这一事件将使得 TCP 停止继续积累数据, 而是将该连接上已有的所有数据立即传输出去。

当目标端接收到紧急数据的时候, 接收方应用被中断(比如, 按 UNIX 的术语来说得到了一个信号), 所以它停止当前正在做的工作, 并且读入数据流以找到紧急数据。紧急数据的尾部应该被标记出来, 因而应用程序能够知道紧急数据的结束位置。紧急数据的起始处并没有被标记出来, 所以, 如何发现紧急数据要取决于具体的应用程序。这种方案基本上只是提供了一种原始的信号机制, 其余的工作全部留给应用程序自己来处理。

### 6.5.3 TCP 协议

在这一小节中, 我们将概述性地介绍一下 TCP 协议。然后在下一小节中我们将逐个讨论 TCP 协议的头部的每个域。

TCP 的一个关键特征, 也是主导了整个协议设计的特征是, TCP 连接上的每个字节都有它自己独有的 32 位序列号。在 Internet 最初的时候, 路由器之间的线路绝大多数是 56-kbps 的租用线路, 所以, 一台满负荷全速运行的主机差不多一周可以遍历所有的序列号。在现代的网络速度上, 序列号的消耗速度是惊人的, 后面我们将会看到这一点。确认和窗口机制分别使用了不同的 32 位序列号, 下面我们将会讨论到。

发送端和接收端的 TCP 实体以数据段的形式交换数据。TCP 数据段(TCP segment)是由一个固定的 20 字节的头(加上可选的部分)以及随后的 0 个或者多个数据字节构成的。TCP 软件决定数据段的大小。它可以多次写操作中的数据累积起来, 放到一个数据段中, 或者将一次写操作中的数据分割到多个数据段中。有两个因素限制了段的长度。第一, 每个数据段, 包括 TCP 头在内, 必须适合 IP 的 65 515 字节净荷大小; 其次, 每个网络都有一个最大传输单元(maximum transfer unit)或者 MTU, 每个数据段必须适合于

MTU。在实践中,MTU 通常是 1500 字节(以太网的净荷大小),因此它规定了数据段长度的上界。

TCP 实体使用的基本协议是滑动窗口协议。当发送方传送一个数据段的时候,它也会启动一个定时器。当该数据段到达目标端的时候,接收方的 TCP 实体返回一个携带了确认号的数据段(如果有数据要发送的话,则包含数据,否则就不包含数据),其中确认号的数值等于接收方期望接收的下一个序列号。如果发送方的定时器在确认数据段到达之前过期的话,则发送方再次发送原来的数据段。

尽管这个协议听起来非常简单,但是它涉及到许多非常微妙的细节,后面我们将会介绍这些细节。数据段到达的顺序可能是错误的,所以可能会发生这样的情形:字节 3072~4095 已经到达了,但是它不能被确认,因为字节 2048~3071 还没有到达。数据段在传输过程中可能会延迟很长时间,因而发送方超时并重传数据段。重传的数据段有可能包含不同的字节范围,因此,这要求接收方必须谨慎地记录下:到目前为止哪些字节已经正确地接收到了。然而,由于数据流中的每个字节都有它自己惟一的偏移值,所以,接收方能够完成细致的管理工作。

TCP 必须做好处理这些问题的准备,并且采用有效的方法来解决这些问题。尽管面临各种各样的网络问题,研究人员还是做了大量的努力来优化 TCP 流的性能。下面将要讨论的一些算法被许多 TCP 实现所采用。

#### 6.5.4 TCP 数据段的头

图 6.29 显示了 TCP 数据段的布局结构。每个数据段的起始部分是一个固定格式的 20 字节的头。固定的头部之后可能跟着头选项。在选项之后,如果该数据段有数据部分的话,则后面跟着最多可达  $65535 - 20 - 20 = 65495$  个数据字节,此式子中的第一个 20 是指 IP 头,第二个 20 指 TCP 头。无任何数据的 TCP 段也是合法的,通常被用于确认和控制消息。

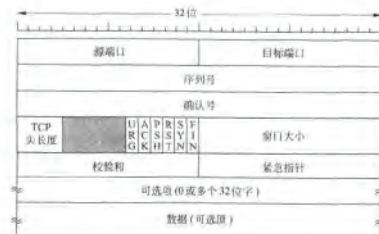


图 6.29 TCP 头



现在我们来逐个地分析 TCP 的头部。Source port(源端口)和 Destination port(目标端口)域标明了-一个连接的两个端点。知名的端口被定义在 [www.iana.org](http://www.iana.org) 上,但是,每台主机可以自由地分配其他的端口。一个端口加上其主机的 IP 地址构成了一个 48 位的惟一端点。源端点和目标端点合起来标识了一个连接。

Sequence number(序列号)和 Acknowledgement number(确认号)域完成它们的常规功能。请注意,后者指定的是下一个期望的字节,而不是已经正确接收到的最后一个字节。这两个域都是 32 位长,因为一个 TCP 流中的每一个数据字节都已经被编号了。

TCP header length(TCP 头长度)域指明了在 TCP 头部包含多少个 32 位的字。此信息是必需的,因为 Options(选项)域是可变长度的,所以整个头部也是变长的。从技术上讲,这个域实际上指明了数据部分在段内部的起始位置(以 32 位字作为单位进行计量),但因为这个数值正好是按字为单位的头部长度,所以,两者的效果是等同的。

接下来是未使用的 6 位域。这个域已经保留了超过四分之一世纪的时间而仍然原封未动,这样的事实正好也说明了 TCP 的设计者们考虑得是多么的周到。几乎没有协议需要利用这个域来修正原始设计中的错误。

然后是 6 个 1 位标志。如果 Urgent pointer(紧急指针)被使用了的话,则 URG 被设置为 1。Urgent pointer 被用来指示出紧急数据在当前数据段中的位置,它是一个相对于当前序列号的字节偏移值。这项设施可以代替中断消息。正如前面我们所提到的,这项设施是一种“允许发送方给接收方发信号,但是又不让 TCP 本身涉及到中断原因”的原始方法。

ACK 位被设置为 1 表示 Acknowledgement number 是有效的。如果 ACK 为 0,则该数据段不包含确认信息,所以,Acknowledgement number 域应该被忽略。

PSH 位表示这是带有 PUSH 标志的数据。因此,接收方在收到数据后应立即请求将数据递交给应用程序,而不是将它缓冲起来直到整个缓冲区接收满为止(这样做的目的可能是为了效率的原因)。

RST 位被用于重置一个已经混乱的连接,之所以会混乱,可能是由于主机崩溃,或者其他的原因。该位也可以被用来拒绝一个无效的数据段,或者拒绝一个连接请求。一般而言,如果你得到的数据段被设置了 RST 位,那说明你这一端有了问题。

SYN 位被用于建立连接的过程。在连接请求中,SYN=1 和 ACK=0 表示该数据段没有使用捎带的确认域。连接应答捎带了一个确认,所以有 SYN=1 和 ACK=1。本质上,SYN 位被用来表示 CONNECTION REQUEST 和 CONNECTION ACCEPTED,然后进一步用 ACK 位来区分这两种可能的情况。

FIN 位被用于释放一个连接。它表示发送方已经没有数据要传输了。然而,在关闭一个连接之后,关闭进程可能会在一段不确定的时间内继续接收到数据。SYN 和 FIN 数据段都有序列号,从而保证了这两种数据段被按照正确的顺序进行处理。

TCP 中的流控制是通过一个可变大小的滑动窗口来完成的。Window size(窗口大小)域指定了从被确认的字节算起可以发送多少个字节。Window size 域为 0 也是合法的,这相当于说,到现在为止多达 Acknowledgement number -1 个字节已经接收到了,但是接收方现在状态不佳,需要休息一下,等一会儿再继续接收更多的数据,谢谢。以后,接



收方可以通过发送一个具有同样 Acknowledgement number 但是非零 Window size 域的数据段,告诉发送方继续发送数据段。

在第 3 章的协议中,“对已经接收到的帧的确认”和“允许发送新帧的授权机制”捆绑在一起。这是每个协议采用固定窗口大小的必然结果。在 TCP 中,确认机制和发送新数据的许可机制被完全分离开。实际上,接收方可以这样说:我已经接收到了序列号 k 之前的字节,但是现在我不想再要更多的数据了。这种分离机制(实际上是可变大小的窗口)带来了格外的灵活性。后面我们将详细地学习这种机制。

Checksum(校验和)也提供了额外的可靠性。它校验的范围包括头部、数据,以及图 6.30 所示的概念性的伪头部。在计算校验和的时候,TCP 的 Checksum 域被设置为 0,如果数据域的长度为奇数的话,则数据域被填补一个额外的 0 字节。校验和算法很简单,它只是将所有的 16 位字按 1 的补码形式累加起来,然后取累加结果的补码。因此,当接收方在整个数据段(包括 Checksum 域)上执行同样的计算过程的时候,结果应该是 0。

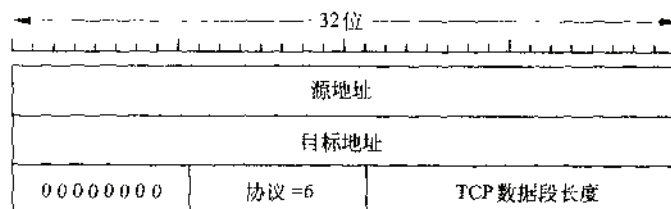


图 6.30 在 TCP 校验和计算中的伪头部

伪头部包含源机器和目标机器的 32 位 IP 地址、TCP 的协议号(6),以及 TCP 数据段(包括 TCP 头)的字节计数。在 TCP 校验和计算中包含伪头部有助于检测到错误递交的分组,但是,这样做也违反了协议层次原则,因为伪头部中的 IP 地址属于 IP 层,不属于 TCP 层。UDP 在计算校验和的时候也使用了同样的伪头。

Options(选项)域提供了一种增加额外设施的方法,在普通的头中不需要这些额外的设施。最重要的选项是“允许每台主机指定它愿意接受的最大 TCP 净荷长度”。采用大的数据段通常比小的数据段更有效率,因为这 20 字节的头部可以被分摊到更多的数据上,但是低档的主机可能处理不了大的数据段。在连接建立过程中,每一方可以宣布它的最大净荷长度,并且查看对方给出的最大值。如果一台主机没有使用这个选项,那么它默认可以接受 536 字节的净荷。所有的 Internet 主机都要求能够接受  $536 + 20 = 556$  字节的 TCP 数据段。两个方向上的最大数据段长度可以不相同。

对于高带宽、高延迟,或者两者兼备的线路,64-KB 的窗口通常是一个问题。在一条 T3 线路(44.736Mbps)上,它只需 12ms 就输出一个完整 64-KB 窗口。如果往返传播延迟是 50ms(对于越洋光缆,这是很典型的),则发送方将有 3/4 的空闲时间在等待确认。在卫星连接上,情形可能会更糟。越是大的窗口,越是允许发送方不停地送出数据,但是使用 16 位的 Window size 域,却无法表达这样的大小值。RFC 1323 提出了一个 Window scale(窗口尺度)选项,从而使得发送方和接收方可以协商一个窗口尺度因子。这个因子值也允许双方将 Window size 域向左移动至多 14 位,因此允许窗口可达  $2^{30}$  个字节。现

在,大多数 TCP 实现都支持这个选项。

RFC 1106 中考虑的另一个选项现在也被广泛实现了,即使用选择性重传协议,而不是回退  $n$  段的协议。如果接收方收到了一个坏的数据段,然后又收到了大量好的数据段,则常规的 TCP 协议最终将超时,并重传所有未被确认的数据段,包括那些已被正确接收的数据段(即回退  $n$  段的协议)。RFC 1106 引入了 NAK,从而允许接收方请求一个特定的数据段(或者多个数据段)。当它得到了这些数据段之后,它可以确认所有已被缓冲的数据,因而减少了重传的数据量。

### 6.5.5 TCP 连接的建立

TCP 使用了在 6.2.2 节中讨论过的三步握手法来建立连接。为了建立一个连接,某一方,比如说服务器,通过执行 LISTEN 和 ACCEPT 原语(既可以指定一个特定的源,也可以不指定)被动地等待一个进来的连接请求。

另一端,比如说客户,执行一个 CONNECT 原语,同时指定以下参数:它希望连接的 IP 地址和端口、它愿意接受的最大 TCP 分段长度,以及一些可选的用户数据(比如口令)。CONNECT 原语发送一个  $\text{SYN}=1$  和  $\text{ACK}=0$  的 TCP 数据段,然后等待应答。

当这个数据段到达目标端的时候,那里的 TCP 实体查看一下是否有一个进程已经在 Destination port 域中指定的端口上执行了 LISTEN。如果没有的话,它送回一个设置了 RST 位的应答,以拒绝客户的连接请求。

如果某个进程正在监听该端口,那么, TCP 实体将进来的 TCP 数据段交给该进程。然后该进程可以接受或者拒绝这个连接请求。如果它接受的话,则送回一个确认数据段。在正常情况发送的 TCP 数据段顺序如图 6.31(a)中所示。请注意, SYN 数据段只消耗 1 字节的序列号空间,所以它的确认是非常明确的,毫无二义性。

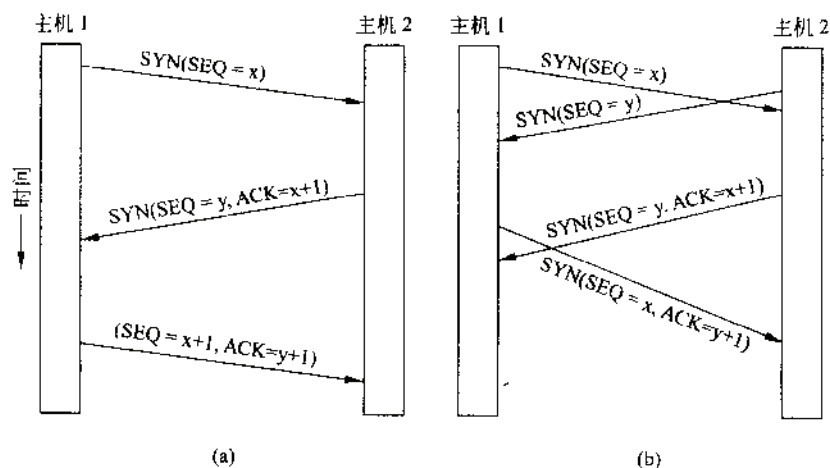


图 6.31

(a) 正常情况下 TCP 连接的建立过程; (b) 连接请求碰撞

如果两台主机同时企图在同样的两个套接字之间建立一个连接,则事件序列如图 6.31 (b) 中所示。这些事件的结果是,只有一个连接被建立起来,而不是两个,因为所有的连接都是由它们的端点来标识的。如果第一个请求导致建立了一个由(x, y)标识的连接,而第二个请求也建立了这样一个连接,那么,在 TCP 实体内部只有一个表项,即(x, y)。

一个连接上的初始序列号不是 0,其原因前面已经讨论过了。TCP 采用了基于时钟的方案,每隔  $4\mu\text{s}$  一个时钟滴答。为了更加安全起见,当一台主机崩溃的时候,它需要等待一段时间(即最大的分组生存期)之后才可以重新启动,这样可以确保以前的连接中的分组不可能仍然在 Internet 的某个地方游荡。

### 6.5.6 TCP 连接的释放

虽然 TCP 连接是全双工的,但是,为了理解 TCP 连接的释放过程,最好将 TCP 连接看成一对单工连接。每个单工连接被单独释放,两个单工连接相互之间独立。为了释放一个连接,任何一方都可以发送一个设置了 FIN 位的 TCP 数据段,这表示它已经没有数据要发送了。当 FIN 数据段被确认的时候,这个方向上就停止传送新数据。然而,另一个方向上可能还在继续无限制地传送数据。当两个方向都停止的时候,连接才被释放。通常情况下,为了释放一个连接,需要 4 个 TCP 数据段:每个方向一个 FIN 和一个 ACK。然而,第一个 ACK 和第二个 FIN 有可能被包含在同一个数据段中,从而将总数降低到 3 个。

正如在电话通话过程中,双方说完再见之后就同时挂断电话一样,一个 TCP 连接的两端可能会同时发送 FIN 数据段。这两个数据段按常规的方法被确认,然后连接被停止。实际上,两台主机按顺序先后释放连接,或者同时释放连接,这两者之间并没有本质的区别。

为了避免两军队问题,这里需要使用定时器。如果在两倍最大分组生存期内 FIN 的应答没有到来的话,FIN 的发送方直接释放连接。另一方最终会注意到,好像对方已经不再监听该连接了,因而也会超时。虽然这种方案并不是完美无缺的,但由于理论上不存在完美的解决方案,所以,TCP 也只好这样做了。在实践中,这种做法很少发生问题。

### 6.5.7 TCP 连接的管理模型

建立连接和释放连接所要求的步骤可以用一个有限状态机来表达,该状态机的 11 种状态如图 6.32 所列。在每一种状态中,都存在一些合法的事件。当合法事件发生的时候,可能需要采取某个动作。当其他事件发生的时候,则报告一个错误。

每个连接都从 CLOSED 状态开始。当它执行了一个被动的打开操作(LISTEN),或者一个主动的打开操作(CONNECT)的时候,它就离开 CLOSED 状态。如果另一端执行了相对应的操作,则连接被建立起来,当前状态变成 ESTABLISHED。连接的释放过程可以由任何一方发起。当释放完成的时候,状态又回到 CLOSED。

状态	说明
CLOSED	没有活动的连接,或者未完成的连接
LISTEN	服务器正在等待进来的连接请求
SYN RCVD	一个连接请求已经到达;等待 ACK
SYN SENT	应用程序已经开始打开连接
ESTABLISHED	正常的数据传输状态
FIN WAIT 1	应用程序说它已经结束连接了
FIN WAIT 2	另一方已经同意释放连接
TIMED WAIT	等待所有的分组逐渐消失
CLOSING	双方试图同时关闭连接
CLOSE WAIT	另一方已经发起了释放连接的过程
LAST ACK	等待所有的分组逐渐消失

图 6.32 在 TCP 连接管理有限状态机中用到的状态

有限状态机如图 6.33 所示。图中用粗线画出了一种常见的情形：一个客户主动地连接到一个被动的服务器上,其中客户部分用实线,服务器部分用虚线。细线是不正常的事件序列。图 6.33 中的每条线都被标记了一对“事件/动作”。这里的事件既可以是用户发起的系统调用(CONNECT、LISTEN、SEND 或者 CLOSE),也可以是一个数据段到达(SYN、FIN、ACK 或者 RST),或者也可能是另外一种情形,即两倍最大分组生存期的超时事件;而动作可能是发送一个控制数据段(SYN、FIN 或者 RST),或者什么也不做(在图中标记为一)。括号中显示的是说明。

为了更好地理解这个图,你可以首先沿着客户的路径(粗实线),然后沿着服务器的路径(粗虚线)来查看。当客户机器上的一个应用程序发出 CONNECT 请求的时候,本地的 TCP 实体创建一条连接记录,并将它标记为 SYN SENT 状态,然后发送一个 SYN 数据段。请注意,在一台机器上可能同时有许多个连接处于打开(或者被动打开)的状态,它们可能代表了多个应用程序,所以,状态是针对每一个连接的,并且每个连接的状态被记录在相应的连接记录中。当 SYN+ACK 到达的时候,TCP 发送出三步握手过程的最后一个 ACK 数据段,并且切换到 ESTABLISHED 状态。现在就可以发送和接收数据了。

当一个应用结束的时候,它执行 CLOSE 原语,从而使本地的 TCP 实体发送一个 FIN 数据段,并等待对应的 ACK(虚线框标记了主动的关闭过程)。当 ACK 到达的时候,发生一次状态迁移,切换到 FIN WAIT 2,而且,连接的一个方向现在被关闭。当另一方也关闭的时候,一个 FIN 数据段会到来,然后它被确认。现在,双方都已经关闭了,但是,TCP 要等待一段最大分组生存期的时间,以确保该连接的所有分组都已经消失了,以防万一发生确认被丢失的情形。当定时器到期之后,TCP 删除该连接记录。

现在我们从服务器的角度来看一下连接管理的情况。服务器执行 LISTEN,并等待有人连接上来。当一个 SYN 进来的时候,它被确认,并且服务器进入到 SYN RCVD 状

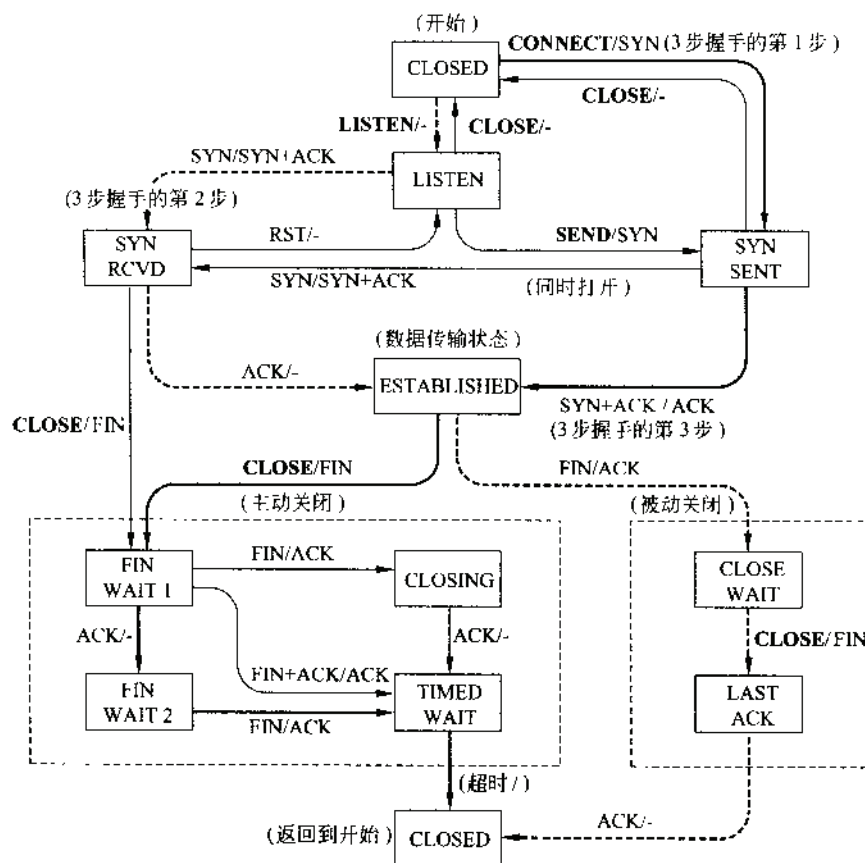


图 6.33 TCP 连接管理有限状态机

粗实线是客户的正常路径,粗虚线是服务器的正常路径;细线是不正常的事件  
每个迁移都被标记了引起该迁移的事件,以及由此而产生的动作,中间用斜线分开

态。当服务器的 SYN 本身被确认的时候,三步握手过程结束,服务器进入到 ESTABLISHED 状态。从现在开始双方可以传输数据了。

当客户完成的时候,它执行 CLOSE,从而导致发送一个 FIN 到达服务器(虚线框标记了被动的关闭过程)。然后,服务器接到信号。当它也执行 CLOSE 的时候,TCP 实体给客户发送一个 FIN 数据段。当客户的确认回来的时候,服务器释放该连接,并且删除相应的连接记录。

### 6.5.8 TCP 传输策略

正如前面所提到的,TCP 中的窗口管理并不像大多数数据链路协议那样,直接依赖于确认机制。例如,假设接收方有一个 4096 字节的缓冲区,如图 6.34 所示。如果发送方传送了一个 2048 字节的数据段,并且该数据段已被正确地接收到,那么,接收方将确认该数据段。然而,由于它现在只剩下 2048 字节的缓冲区空间(在应用程序从缓冲区中取走

数据之前),所以,它将宣告自己有一个 2048 字节大小的窗口,并且该窗口从下一个期望的字节开始。

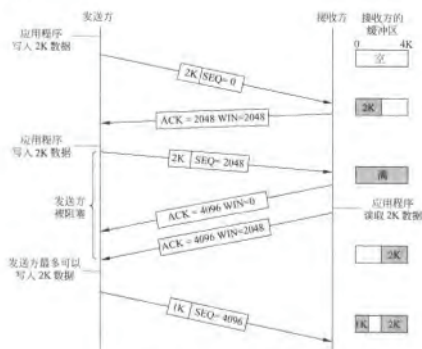


图 6.34 TCP 中的窗口管理

现在发送方又传输了 2048 字节,它被确认了,但是接收方宣告的窗口变成了 0 大小。发送方必须停止下来,等待接收主机上的应用进程从缓冲区中取走一些数据,到那时候, TCP 实体可以向发送方宣告一个更大的窗口。

当窗口为 0 的时候,发送方不能够再正常地发送数据段了,但这里有两种意外情形。第一,紧急数据仍可发送,比如,让用户杀掉远程机器上运行的某一个进程。第二,发送方可以发送一个 1 字节的数据段,以便让接收方重新宣告下一个期望的字节和窗口大小。TCP 标准明确地提供了这个选项,以避免当一个宣告窗口信息的数据段丢失之后发生死锁的情形。

发送方并不要求一接到应用传递过来的数据就必须马上将数据传出去。接收方也不要尽可能快地发送确认数据段。例如,在图 6.34 中,当第一块 2KB 数据到来的时候, TCP 知道它有 4KB 的窗口可以使用,所以它完全可以仅仅将这些数据缓冲起来,直到另一 2KB 数据到来,因而可以传递一个包含 4KB 净荷的数据段。TCP 实体可以充分利用这种自由度来提高性能。

考虑一个指向某个交互式编辑器的 telnet 连接,该编辑器对用户的每次击键动作都做出响应。在最差的情况下,当一个字符到达发送端的 TCP 实体的时候, TCP 创建一个 21 字节的 TCP 数据段,并将它交给 IP 作为一个 41 字节的 IP 数据报发送出去。在接收



端, TCP 立即发送一个 40 字节的确认(20 字节的 TCP 头加上 20 字节的 IP 头)。以后, 当编辑器读取了这个字节之后, TCP 发送一个窗口更新数据段, 它将窗口向前移动 1 个字节。这个分组也是 40 字节。最后, 当编辑器处理了该字符以后, 它发送一个 41 字节的分组作为该字符的回显。总共累计起来, 对于每个敲入的字符, 需要使用 162 字节的带宽, 发送 4 个数据段。对于带宽紧缺的场合, 这种处理方法显然并不合适。

针对这样的情况, 许多 TCP 实现采用的一种优化办法是, 将确认和窗口更新数据段延迟 500ms, 希望能够获得一些数据, 从而免费搭载过去。假设编辑器在 500ms 内回显的话, 则现在只需要给远程用户送回一个 41 字节的分组即可, 从而将分组数和所用带宽减少了一半。

尽管这条规则减少了接收方放在网络上的负载, 但是发送方的工作方式仍然非常低效, 它发送的 41 字节的分组只包含 1 个字节的数据。避免这种用法的一种办法是采用 **Nagle 算法**(Nagle, 1984)。Nagle 的建议非常简单: 当数据以每次一个字节的方式进入到发送方的时候, 发送方只是发送第一个字节, 然后将其余的字节缓冲起来, 直到送出去的那个字节被确认为止。然后将所有缓冲的字节放在一个 TCP 数据段中发送出去, 并且继续开始缓冲字节, 直到前面送出去的字节全部被确认。如果用户敲键很快, 而网络又很慢, 则每个数据段能包含相当数量的字符, 从而大大减少所用的带宽。另外, 如果传递进来的数据足够多, 多到可以填充一半的窗口或者填满一个最大数据段的话, 则该算法也允许发送一个新的分组。

Nagle 算法被广泛应用于各种 TCP 实现, 但是, 也有很多时候最好禁止使用这个算法。尤其是, 当一个 X Window 应用在 Internet 上运行的时候, 鼠标的移动事件必须被发送给远程计算机。(X Window 系统是大多数 UNIX 平台上使用的窗口系统。) 把这些移动事件收集起来一批一批地发送出去将使得鼠标的移动极为不连贯, 从而让用户感到不快。

另一个可能使 TCP 性能退化的问题是 **愚笨窗口综合症(silly window syndrome)**(Clark, 1982)。当数据以大块的形式被传递给发送端 TCP 实体, 但是接收端的交互式应用每次仅读取一个字节数据的时候, 这个问题就会发生。为了看清此问题, 请参考图 6.35。刚开始的时候, 接收端的 TCP 缓冲区是满的, 发送方也知道这一点(即有一个大小为 0 的窗口)。然后, 交互式应用从 TCP 流中读取一个字符, 这个动作让接收端的 TCP 非常高兴, 所以它发送一个窗口更新数据段给发送方, 告诉它现在你可以发送 1 个字节过来。发送方被迫发送 1 个字节, 现在缓冲区又满了, 所以, 接收方对这 1 字节的数据段进行确认, 并设置窗口大小为 0。这个过程可能会永久地持续下去。

Clark 的解决方案是禁止接收方发送只有 1 个字节的窗口更新数据段。相反, 它必须等待一段时间, 直到有了一定数量的可用空间之后再告诉对方。特别是, 只有当接收方能够处理它在建立连接时宣告的最大数据段大小, 或者它的缓冲区一半已空(相当于两者之中取较小的值)的时候, 它才应该发送窗口更新数据段。

而且, 发送方若不发送太小的数据段也会有所帮助。相反, 它应该试着等待一段时间, 直到积累足够的窗口空间以便发送一个满的数据段, 或者至少包含接收方缓冲区的一半大小(发送方必须根据过去接收到的窗口更新数据段的模式来估计接收方缓冲区的

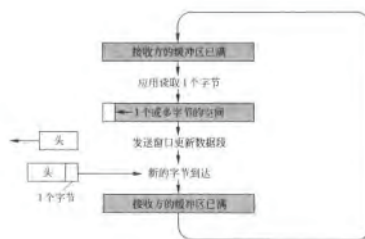


图 6.35 阻塞窗口症状

大小)。

Nagle 的算法和 Clark 针对阻塞窗口综合症的解决方案是相互补充的。Nagle 试图解决由于发送端应用每次向 TCP 传递一个字节而引起的问题;Clark 则试图解决由于接收端应用每次从 TCP 流中读取一个字节而引起的问题。这两种方案都是有效的,而且可以一起工作。它们要达到的目标是,发送方不要发送太小的数据段,接收方也不要请求太小的数据段。

接收端的 TCP 实体除了向发送方宣告较大的窗口以外,还可以进一步提高性能。如同发送端的 TCP 一样,接收端的 TCP 也可以缓冲数据,所以它可以阻塞上层应用的 READ 请求,直至它有大块的数据可以提供。这样做可以减少调用 TCP 的次数,从而减少了额外的开销。当然,这样做也增加了响应的时间,但是,对于像文件传输这样的非交互式应用来说,效率可能比单个请求的响应时间更加重要。

接收方的另一个问题是如何处理乱序的数据段。这些数据段可以被保留,或者被丢弃,这完全由接收方来决定。当然,从确认的角度来说,只有当被确认的字节之前所有的数据都已经接收到了以后,接收方才可以发送确认数据段。如果接收方得到了数据段 0、1、2、4、5、6 和 7,那么,它可以确认数据段 2 中最后一个字节之前(含该字节)的所有数据。当发送方超时的时候,它就会重传数据段 3。如果接收方已经将数据段 4 至 7 缓冲起来了,那么,在接收到数据段 3 之后,它可以确认数据段 7 末尾之前的所有字节。

#### 6.5.9 TCP 拥塞控制

当一个网络面对的负载超过了它的处理能力时,拥塞就会发生。Internet 也不例外。在这一小节中,我们将讨论在过去四分之一世纪以来人们开发出来的各种处理拥塞的算法。尽管网络层也试图管理拥塞,但是,绝大多数繁重的任务是由 TCP 来完成的,因为针对拥塞的真正解决方案是减慢数据率。

理论上,通过使用一条从物理学中套用过来的法则:分守恒法则,拥塞现象就可以

得到控制。它的基本思想是,只有当一个老的分组离开(即被递交)之后才允许向网络往入一个新的分组。TCP 企图通过动态地维护窗口的大小来实现这个目标。

管理拥塞的第一步是检测拥塞。在过去,检测拥塞是非常困难的。由于丢失分组而引起的超时可能有两种情形:(1)传输线路上有噪声;或者(2)在一台拥塞的路由器上分组被丢弃。要区分这两种情形是很困难的。

现在,由于传输错误而导致的分组丢失情况相对较少发生,因为大多数长距离干线是光纤(不过无线网络是另外一回事了)。因此,Internet 上的大多数传输超时都是由于拥塞而引起的。所有的 Internet TCP 算法都假定超时是由于拥塞引起的,并且通过监视超时的情况来判断是否出现问题,就好像矿工通过观察他们的报警器来了解自己的处境一样。

在讨论 TCP 如何应对拥塞之前,我们先来描述一下若要从一开始就避免发生拥塞该做些什么事情。当一个连接被建立起来的时候,首先双方必须要选择一个合适的窗口大小。接收方可以根据其缓冲区的大小来指定窗口的大小。如果发送方遵守此窗口大小的限制,则接收端不会发生缓冲区溢出的问题,但是,有可能由于网络内部的拥塞而导致发生问题。

在图 6.36 中,我们用水压来说明这个问题。在图 6.36(a)中,我们看到一根粗的水管通向一个小容量的接收容器。只要水龙头(发送方)送出的水不超出水桶的容量,则水就不会遗失。在图 6.36(b)中,限制因素并不是水桶的容量,而是输水管(网络)的内部承载能力。如果大量的水以极快的速度到来的话,则这些水将倒流回去,从而有些水将会流失(在这种情况下,水将会溢出漏斗)。

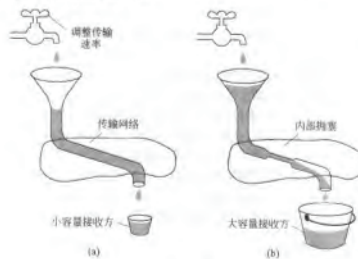


图 6.36  
(a) 快速的网络向小容量的接收方传输数据; (b) 快速的网络向大容量的接收方传输数据

Internet 的方案是,首先意识到这里存在两个潜在的问题,网络容量和接收方的容量,然后单独地处理每一个问题。为此,每个发送方维护两个窗口:第一个是接收方允许

的窗口,第二个是拥塞窗口(congestion window)。每个窗口反映了发送方可以传送的字节数量。最终允许发送的字节数量是两个窗口的最小值。因此,有效窗口是“发送方认为没有问题的窗口”与“接收方认为没有问题的窗口”中较小的那个窗口。如果接收方说“可以发送 8KB”,但是发送方知道,超过 4KB 的突发数据就会堵塞网络,那么,它就发送 4KB。另一方面,如果接收方说“可以发送 8KB”,而发送方知道,即使多达 32KB 的突发数据也可以很容易地通过网络,那么,它按照对方的要求发送完整的 8KB 数据。

当一个连接被建立起来的时候,发送方将拥塞窗口初始化为该连接上当前使用的最大数据段长度。然后,它发送一个最大的数据段。如果该数据段在定时器过期之前被确认,则它将拥塞窗口增加一个数据段的字节数,从而使拥塞窗口变成两倍的最大数据段长度,然后发送两个数据段。如果这两个数据段中的每一个都被确认了,则拥塞窗口再增加两个最大数据段长度。当拥塞窗口达到  $n$  个数据段的时候,如果所有  $n$  个数据段都被及时确认的话,则拥塞窗口增加这  $n$  个数据段所对应的字节数。实际上,每一批被确认的突发数据段都会使拥塞窗口加倍。

拥塞窗口一直呈指数增长,直至发生超时,或者达到接收方窗口的大小。这里的思想是,如果一定大小的突发数据,比如说 1024、2048 和 4096 字节,都被正常地传送过去,但是,8192 字节的突发数据却发生了超时,则拥塞窗口应该被设置为 4096 以避免拥塞。只要拥塞窗口保持在 4096 上,则无论接收方准许的窗口空间有多大,发送方都不会发送超过 4096 字节的突发数据。这个算法被称为慢启动算法(slow start),但是它实际上一点也不慢(Jacobson, 1988)。它是指数量级的,所有的 TCP 实现都要求支持该算法。

现在我们来分析 Internet 的拥塞控制算法。除了接收方窗口和拥塞窗口以外,它还使用了第三个参数:一个阈值(threshold),初始时该参数为 64KB。当一次超时发生的时候,阈值被设置为当前拥塞窗口的一半,而拥塞窗口被重置为一个最大数据段。然后使用慢启动算法来决定网络的处理能力,不过当增长到阈值的时候便停止。从这个点开始,每一次成功的传输都会使拥塞窗口线性地增长(即每次突发数据仅增长一个最大数据段),而不是成倍地增长。实际上,这个算法是在猜测,将拥塞窗口减小一半可能是可以接受的,然后再从这个点开始慢慢地往上增长。

图 6.37 显示了拥塞算法是如何工作的。这里的最大数据段长度是 1024 字节。初始时,拥塞窗口是 64KB,但是,发生了一次超时之后,阈值被设置到 32KB,而拥塞窗口被设置为 1KB(对应于 0 号传输)。然后,拥塞窗口呈指数增长,一直到达阈值(32KB)。然后从这个点开始,它按线性增长。

13 号传输非常不幸(它应该是有思想准备的),这时发生了超时。于是阈值被设置为当前窗口的一半(现在的窗口为 40KB,所以一半是 20KB),并且又重新开始慢启动过程。从 14 号传输开始,确认数据段又能够及时到达,前四次每次都使拥塞窗口加倍,但是,之后又变成线性增长了。

如果不再发生超时的话,则拥塞窗口将继续增长,直至到达接收方窗口的大小。在这个点上,它将停止增长,而且,只要不再发生超时并且接收方的窗口不改变大小,则拥塞窗口保持不变。顺便提一下,如果一个 ICMP SOURCE QUENCH 分组到来并且被传递给 TCP,则这个事件将被当作超时一样来对待。RFC 3168 描述了另一种更加新颖的拥塞控

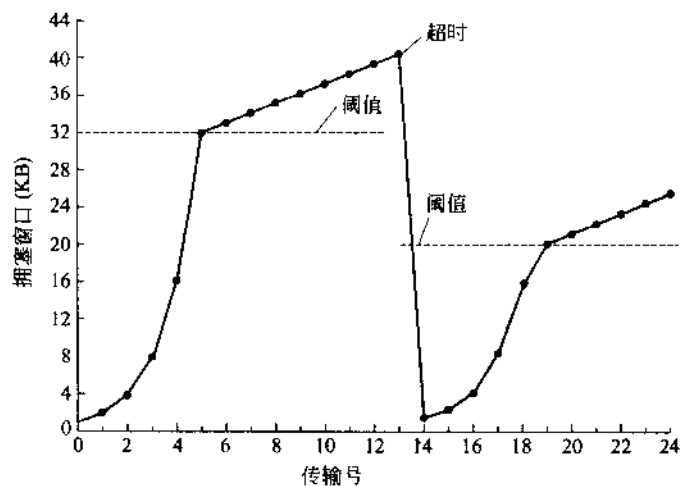


图 6.37 Internet 拥塞算法的一个例子

制算法。

#### 6.5.10 TCP 定时器管理

TCP 使用多个定时器(至少从概念上来讲是定时器)来完成它的工作。其中最重要的是重传定时器(retransmission timer)。当 TCP 实体发送一个数据段的时候,它同时也启动一个重传定时器。如果在定时器过期之前该数据段被确认的话,则定时器被停止。另一方面,如果在确认到来之前定时器先到期,则数据段被重传(并且该定时器又被重新启动)。于是问题就来了:超时间隔应该为多长?

这个问题在 Internet 的传输层上比在第 3 章介绍的一般数据链路协议中更加困难。在数据链路协议中,期望的延迟是高度可预测的(即方差很小),所以,定时器可以被设置到期望的确认到达时间之后再稍微过一点即可,如图 6.38(a)所示。由于在数据链路层中确认极少被延迟(因为不存在拥塞),所以,如果在预期的时间点上确认没有到来,则往往意味着原始帧或者确认帧已经丢失了。

TCP 面临截然不同的环境。TCP 确认数据段回到发送方所需时间的概率密度函数看起来更像 6.38(b),而不是图 6.38(a)。要想确定到达目标端的往返时间是非常复杂的,即使知道了这一段时间,要确定超时间隔也是非常困难的。如果超时间隔被设置得太短,比如说图 6.38(b)中的  $T_1$ ,则会发生大量不必要的重传,从而许多无用分组反而堵塞 Internet。如果超时间隔被设置得太长(比如  $T_2$ ),则一旦分组丢失之后,由于太长的重传延迟,所以性能会受到影响。而且,确认到达时间分布的均值和方差也会随着拥塞的发生或者解决而在几秒钟时间内迅速地改变。

解决的方案是使用一个高度动态的算法,它根据网络性能的连续测量情况,不断地调整超时间隔。TCP 通常采用的算法是 Jacobson(1988)算法,其工作原理如下所述。对于每一个连接,TCP 维护一个变量 RTT,它代表了到达连接目标端的往返时间的当前最佳

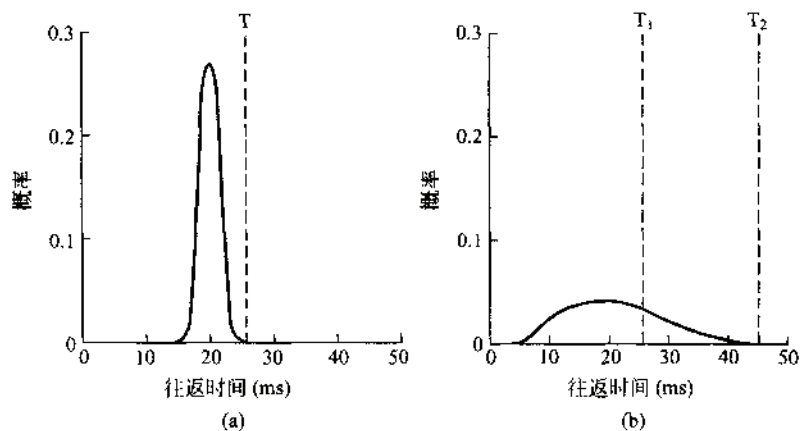


图 6.38

(a) 在数据链路层中确认到达时间的概率密度；(b) 在 TCP 中确认到达时间的概率密度

估计值。当一个数据段被发送出去的时候, TCP 启动一个定时器, 该定时器有两个作用, 一是看需要多长时间该数据段被确认; 二是若时间太长的话, 则触发重传动作。如果在定时器过期之前确认数据段回来的话, 则 TCP 测量一下这次确认所花的时间, 比如说  $M$ 。然后它根据下面的公式更新 RTT:

$$RTT = (RTT + (1 - \alpha))M$$

这里  $\alpha$  是一个平滑因子, 它决定了老的 RTT 值所占的权重。典型情况下  $\alpha = 7/8$ 。

即使有了一个好的 RTT 值, 要选择一个合适的重传超时时间间隔仍然不是一件很容易的事情。正常情况下, TCP 使用  $\beta RTT$  作为重传超时时间间隔, 但诀窍在于如何选择  $\beta$ 。在最初的实现中,  $\beta$  总是 2, 但经验表明常数值不够灵活, 因为当发生变化时它不能够很好地做出反应。

1988 年, Jacobson 提出来让  $\beta$  大致上与确认到达时间概率密度函数的标准偏差成正比, 因而大的变化意味着大的  $\beta$ , 反之亦然。尤其是, 他建议使用平均偏差作为标准偏差的粗略估计。他的算法要求维护另一个被平滑的偏差变量  $D$ 。当一个确认到达的时候, 该算法计算出期望值与观察值之差  $|RTT - M|$ 。然后利用以下公式计算该差值的一个平滑值, 并保存在  $D$  中:

$$D = \alpha D + (1 - \alpha)|RTT - M|$$

这里的  $\alpha$  可能与平滑 RTT 时所用的值相同, 也可能不同。虽然  $D$  并不完全等同于标准偏差, 但是它已经足够好了, 而且 Jacobson 证明了只需使用整数加法、减法和移位就可以计算  $D$  值——这是一个很大的优势。现在, 大多数 TCP 实现都使用这个算法, 并且将超时时间间隔设置为:

$$\text{Timeout} = RTT + 4 \times D$$

这里选择因子 4 多少有点随意, 但是它有两个好处。第一, 用一个移位操作就可以实现乘 4 的计算; 第二, 它可以将不必要的超时和重传降低到最小, 因为在 4 倍标准偏差以后再进来的分组不足总数的百分之一(实际上, Jacobson 最初说使用 2, 但是后来的工作



证明了使用 4 可以达到更好的性能)。

伴随着 RTT 的动态估计而引发的一个问题是,当一个数据段超时并重新发送以后该怎么办?当确认进来的时候,你是无法判断该确认是针对第一次传输呢,还是针对后来的重传。若猜测错误,则会严重影响 RTT 的估计。Phil Karn 发现了这个问题的一种解决办法。他是一名业余无线电爱好者,对于利用无线电(一种人所皆知的不可靠介质,即使在天气很好的时候,也有一半的分组能够传输成功)来传输 TCP/IP 分组有浓厚的兴趣。他提出的建议很简单:对于已被重传的数据段,并不更新 RTT。相反,每次失败以后,超时间隔被加倍,直到有数据段能一次通过为止。这个修正算法被称为 **Karn 算法**,大多数 TCP 实现使用了此算法。

重传定时器并不是 TCP 使用的惟一定时器。第二种定时器是**持续定时器(persistence timer)**。它的设计意图是为了避免以下所述的死锁情况。接收方发送一个窗口大小为 0 的确认,让发送方等一等。后来,接收方更新了窗口,但是,携带更新消息的分组丢失了。现在,发送方和接收方都在等待对方的进一步动作。当持续定时器过期的时候,发送方给接收方发送一个探测消息。接收方对探测消息的响应是将窗口大小告诉发送方。如果它仍然为 0,则持续定时器被再次设置,并开始新一轮循环。如果它非 0,则现在就可以发送数据了。

有些 TCP 实现使用了第三个定时器:**保活定时器(keepalive timer)**。当一个连接空闲了较长一段时间以后,保活定时器可能到期,从而促使某一方查看另一方是否仍然还在。如果另一方无法应答的话,则连接被终止。这个特性是有争议的,因为它增加了额外的开销,而且有可能由于暂时的网络分区而终止掉一个本来很正常的连接。

每个 TCP 连接使用的最后一个定时器是在关闭时该连接处于 TIMED WAIT 状态中所使用的定时器。它运行两倍的分组生存期时间,以确保当连接被关闭以后,该连接上创建的所有分组都完全消失。

#### 6.5.11 无线 TCP 和 UDP

理论上,传输协议应该独立于下面网络层所使用的技术。尤其是,TCP 不应该关心 IP 到底是运行在光纤上,还是通过无线电波来传输。在实践中,这确实是个问题,因为大多数 TCP 实现都已经小心地作了优化,而优化的基础是一些假设条件,这些假设条件对于有线网络是成立的,但对于无线网络却并不成立。忽略无线传输的特性将会导致一个逻辑上正确但是性能奇差的 TCP 实现。

基本的问题是拥塞控制算法。现在,几乎所有的 TCP 实现都假设超时是由于拥塞引起的,而不是由于丢失的分组而引起的。因此,当定时器到期的时候,TCP 减慢速度,发送少量的数据(比如 Jacobson 的慢启动算法)。这种做法背后的思想是减少网络的负载,从而缓解拥塞。

不幸的是,无线传输链路是高度不可靠的。它们总是丢失分组。处理丢失分组的正确办法是再次发送这些分组,而且要尽可能快速地重发。减慢速度只会使事情更糟。比如说,如果 20% 的分组丢失的话,那么,当发送方每秒钟传输 100 个分组的时候,总吞吐量是每秒钟 80 个分组。如果发送方减慢到每秒钟 50 个分组的话,则吞吐量下降到每秒

钟 40 个分组。

事实上,当有线网络上丢失了一个分组以后,发送方应该减慢速度;而当无线网络上丢失了一个分组以后,发送方应该更加努力地重试发送。当发送方不知道底层网络的类型时,它很难做出正确的决定。

通常情况下,从发送方到接收方的路径是异质的。前 1000km 可能经过一个有线网络,但是,最后 1km 可能是无线的。现在,要想在一次超时的基础上做出正确的决定就更加困难了,因为这关系到问题发生的地点。Bakne 和 Badrinath(1995)考虑的解决方案(被称为间接 TCP[indirect TCP])是将 TCP 连接分成两个单独的连接,如图 6.39 所示。第一个连接是指从发送方到基站之间的路径;第二个连接是指从基站到接收方之间的路径。基站只是简单地在两个方向上为两个连接复制分组。

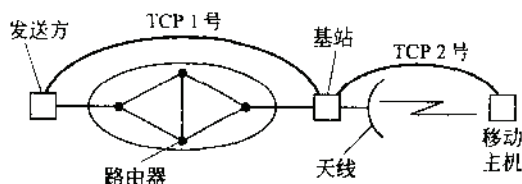


图 6.39 将一个 TCP 连接分成两个连接

这种方案的优点是,分开之后的两个连接是同质的。第一个连接上的超时可能会导致发送方减慢发送速度,而第二个连接上的超时则会导致加快发送速度。这两个连接上的其他参数也可以被分别进行调整。这种方案的缺点是,它违反了 TCP 的语义。由于原连接的每个部分都是一个完全的 TCP 连接,所以,基站按照通常的方式来确认每一个 TCP 数据段。但现在,发送方接收到一个确认并不意味着接收方已经得到了相应的数据段,而仅仅意味着基站已经接收到该数据段。

Balakrishnan 等(1995)提出了另一种不破坏 TCP 语义的方案,该方案需要对基站中的网络层代码作少许的修改才可以工作。其中一个修改是增加一个监视代理(snooping agent),它负责监视从基站发送给移动主机的 TCP 数据段以及从移动主机返回来的确认,并且将它们缓存起来。如果监视代理看到有一个 TCP 数据段被传送给一台移动主机,但是在它的定时器(相对较短)到期之前并没有看到相应的确认回来,那么,它直接重传该数据段,而不告知源发送方自己已经这么做了。当它看到了从移动主机发出的重复确认时,无疑这意味着移动主机一定漏掉了某些数据,因而它也会重传数据。重复的确认在此处被丢弃,以免发送方错误地将它们理解为拥塞。

然而,这种透明性的一个缺点是,如果无线链路的分组损坏率很高,则源端可能在等待一个确认的时候发生超时,并且调用拥塞控制算法。而如果采用间接 TCP 方案,则拥塞控制算法永远也不会被启动,除非在网络的有线部分真的发生了拥塞。

Balakrishnan 等的论文也针对“移动主机发出的数据段的丢失问题”提出了解决方案。当基站注意到在进来的数据段序列号中有了缝隙的时候,它利用 TCP 的选项,生成一个要求有选择地重传遗漏字节的请求。

利用这些修补方案,无线链路的两个传输方向将变得更加可靠,而源端无需知道这

切,而且,这样做也不会改变 TCP 的语义。

虽然 UDP 并不像 TCP 那样受到同样这些问题的影响,但是,无线通信也给 UDP 带来了不少困难。主要的麻烦在于,使用 UDP 的程序总是期望它是非常可靠的。尽管大家都知道这种可靠性是没有保证的,但是,人们仍然期望 UDP 通信是近乎完美的。在一个无线环境中,UDP 离完美实在差得太远了。对于那些“能够以非常高的代价从 UDP 消息丢失的失败中恢复过来”的程序,突然之间将它从一个“理论上消息可能会丢失但实际上却很少丢失”的环境中转换到一个频繁地丢失消息的环境中,可能会导致性能上的灾难。

无线通信也会影响到除了性能以外的其他领域。例如,一台移动主机如何发现并连接一台本地打印机,而不是其主场所的打印机呢?跟这个问题有些相似的另一个问题是如何获得本地蜂窝单元的 WWW 页面,甚至可能连它的名字都不知道。而且,WWW 页面的设计者总是倾向于假设客户和服务器的带宽可以使用。在每个页面上都放置一个大的标志图案(logo)有可能反而达不到预期的效果,因为用户通过一条慢速的无线链路每次访问页面时需要等待 10 秒钟,这可能会使用户无比厌烦。

随着无线网络的日益普及,在无线网络中运行 TCP 的问题也变得更加尖锐。有关这个领域的更多工作可以参考(Barakat et al., 2000; Ghani and Dixit, 1999; Huston, 2001; and Xylomenos et al., 2001)。

#### 6.5.12 事务型 TCP

在本章前面部分我们看到了远过程调用可以作为一种实现客户-服务器系统的方法。如果请求和应答消息都非常小,因而它们可以被放到单个分组中,并且对应的操作是幂等的,那么,我们只需直接使用 UDP 即可。然而,如果这些条件不满足的话,则 UDP 就没有任何吸引力了。例如,如果应答消息非常大,那么,该消息的分片必须被按照顺序组织起来,而且还要设计一种机制来重传丢失的分片。实际上,应用系统需要重新发明一套 TCP。

很显然,这种做法是不恰当的,但是,使用 TCP 本身也是不恰当的。问题在于效率的因素。利用 TCP 来完成一个 RPC 调用的正常分组序列如图 6.40(a)所示。在最佳情形下需要 9 个分组。

这 9 个分组如下:

- (1) 客户发送一个 SYN 分组,请求建立一个连接。
- (2) 服务器发送一个 ACK 分组,以便对客户的 SYN 分组进行确认。
- (3) 客户完成三步握手过程。
- (4) 客户发送实际的请求。
- (5) 客户发送一个 FIN 分组,表示它已经完成了所有的数据发送工作。
- (6) 服务器确认第 4 步发送的请求,以及第 5 步发送的 FIN。
- (7) 服务器将应答数据送回给客户。
- (8) 服务器发送一个 FIN 分组,表示它也完成所有的工作了。
- (9) 客户对服务器的 FIN 进行确认。

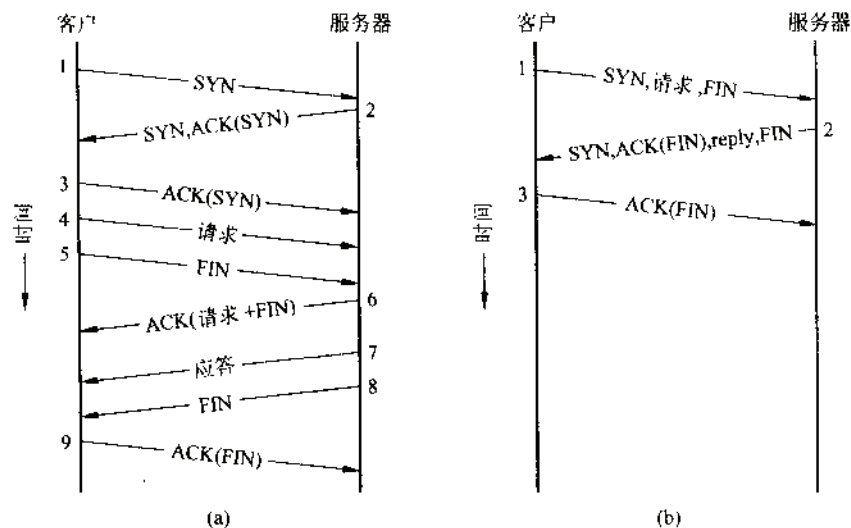


图 6.40

(a) 使用普通 TCP 的 RPC 过程; (b) 使用 T/TCP 的 RPC

请注意,这是最佳的情形。在最差的情况下,客户的请求和 FIN 将被单独确认,同样地,服务器的应答和 FIN 也要被单独确认。

很快地你就会想到这样一个问题:是否有一种办法可以把“用 UDP 来实现 RPC 的高效率(只要两条消息)”与 TCP 的可靠性结合起来呢?答案是:差不多是可以的。有一个试验性的 TCP 变种可以做到这一点,它被称为 **T/TCP (Transactional TCP, 事务型 TCP)**,RFC 1379 和 1644 描述了 T/TCP。

这里的中心思想是,对建立连接的标准序列稍作修改,从而允许在建立连接过程中传输数据。图 6.40(b)演示了 T/TCP 协议。客户的第一个分组包含了 SYN 位、请求本身和 FIN 位。实际上,它相当于这样说:我希望建立一个连接;这里是数据;我的任务完成了。

当服务器接到请求时,它执行查询或者计算工作以完成客户的请求,并且生成应答的内容,然后选择如何做出响应。如果应答的内容可以被放到一个分组中,那么它给出如图 6.40(b)中的应答,相当于这样说:我确认你的 FIN;这里是答案;我的任务也完成了。然后,客户确认服务器的 FIN,通过三条消息协议结束了。

然而,如果服务器执行的结果大于一个分组,那么,服务器也可以不打开分组的 FIN 位,这样它就可以发送多个分组,然后再关闭这个方向的传输过程。

有可能值得提一下的是,T/TCP 并不是惟一的一种改进提案。另一种提案是 **SCTP (Stream Control Transmission Protocol, 流控制传输协议)**。它的特性包括消息边界保留、多种递交模式(比如无序递交)、多目标(备用目标),以及有选择的确认(Steward and Metz, 2001)。然而,如果某一项技术(或者其他的事物)已经很好地工作了很长时间,那么,一旦有人提出改进方案,则在“用户需要更多的特性”和“如果原来的方案没有被打破,

则不要对它修改”两大阵营之间总是会有一番激烈的争斗。

## 6.6 性能问题

在计算机网络中,性能问题是非常重要的。当成百上千台计算机相互连接起来的时候,无法预知结果的复杂交互过程是很常见的。这种复杂性常常会导致很差的性能,而且无人知道其中的缘由。在本节中,我们将讨论许多与网络性能有关的事项,以便了解可能存在哪些问题以及如何处理这些问题。

不幸的是,理解网络的性能更像是一门艺术,而不是一门科学。这里很少有可在实践中应用的基础理论。我们最好的做法是给出一些来自于实践的经验规则,并且展示一些真实世界中的例子。我们有意将这部分讨论推迟到学习了 TCP 的传输层之后,目的就是能够在讲解过程中可以用 TCP 作为例子。

传输层并不是出现性能问题的惟一地方。我们在上一章介绍网络层的时候已经看到过一些性能问题了。然而,网络层主要关注的是路由和拥塞控制。一些更加广泛的、面向系统的问题往往会涉及到传输层,所以,在这一章中讨论这些问题是非常合适的。

在接下来的 5 小节中,我们将看一看网络性能的 5 个方面:

- (1) 性能问题。
- (2) 测量网络性能。
- (3) 具有更好性能的系统设计。
- (4) 快速的 TPDU 处理。
- (5) 未来高性能网络的协议。

顺便提一下,我们需要为传输实体之间交换的数据单元起一个更加一般化的名字。TCP 的术语是数据段,它很容易混淆,而且在 TCP 领域之外从来没有人把它用作这样的含义。ATM 的术语有 CS-PDU、SAR-PDU 和 CPCS-PDU,但是它们都特别针对于 ATM。分组很显然被用于网络层,而消息(message)则属于应用层。由于缺少一个标准的术语,我们只好再倒回去,将传输实体之间交换的数据单元称为 TPDU。当我们同时指 TPDU 和分组的时候,我们将使用分组作为集合性术语,比如在“CPU 必须足够快速以便能够实时地处理进来的分组”中,我们既指网络层的分组,也指封装在网络层分组中的 TPDU。

### 6.6.1 计算机网络中的性能问题

有些性能问题,比如拥塞,是由于临时资源过载引起的。如果到达一台路由器的流量突然之间超过了它的处理能力,那么,拥塞就会发生,从而导致性能问题。在上一章中我们已经详细地学习过拥塞控制了。

当网络中存在结构性的资源不平衡时,性能也会退化。例如,如果将一条千兆位的通信线路连接到一台低档的 PC 上,则性能较差的 CPU 将无法快速地处理进来的分组,因此有些分组将丢失。这些分组最终会被重传,因而既增加了延迟,又浪费了带宽,而且往往降低了性能。



过载也可能被同步触发。例如,如果一个 TPDU 包含一个坏的参数(比如,它的目标端口),那么,在许多情形下,接收方会尽职地送回一个错误通知。现在请考虑,如果将一个坏的 TPDU 广播给 10 000 台机器则会发生什么样的情况:每台机器都可能送回一个错误消息。由此引发的广播风暴(broadcast storm)可能会严重地削弱网络的性能。以前,UDP 协议对于发送给广播地址的 UDP TPDU 中的错误也是响应一条错误消息,所以 UDP 也有同样的广播风暴问题,后来协议作了修改,使主机不再响应这样的错误以避免广播风暴。

同步触发过载的另一个例子是在电源失败之后发生的情形。当电源恢复的时候,所有的机器同时跳转到它们的 ROM 中,以便启动系统。一个典型的启动序列可能如下:首先与某一台(DHCP)服务器联系,以便获得一个真实的身份;然后与某一台文件服务器联系,以便获得操作系统的一份副本。如果成千上百台机器同时做这些事情,则服务器可能会不堪重负而崩溃。

即使不存在同步过载,并且资源也足够,则同样也可能由于缺少系统总体协调而发生性能退化的现象。例如,如果一台机器有足够的 CPU 能力和内存,但是并没有为缓冲区空间分配足够的内存,则缓冲区溢出现象仍然会发生,并且有些 TPDU 将丢失。类似地,如果系统的调度算法没有赋予足够高的优先级给处理进来 TPDU 的进程,那么,有些 TPDU 就可能会丢失。

另一个系统协调问题是正确地设置超时间隔。当一个 TPDU 被发送出去时,TCP 通常会设置一个定时器,以便当该 TPDU 丢失时采取适当的措施。如果超时间隔设置得太短,则将会发生不必要的重传,从而堵塞线路。如果超时间隔设置得太长,那么当 TPDU 被丢失之后将导致不必要的延迟。其他可以调整的参数包括:等待可捎带确认的反向数据多长时间之后再发送单独的确认;经过多少次重传尝试之后才放弃。

千兆位网络又引入了新的性能问题。例如,请考虑从 San Diego(圣地亚哥)向 Boston(波士顿)发送一块 64KB 的数据,以便填满接收方的 64KB 缓冲区。假设该链路是 1Gbps,并且光纤中光速的单向延迟为 20ms。初始时,在  $t=0$  时刻,管道是空的,如图 6.41(a)所示。仅仅  $500\mu\text{s}$  以后,所有的 TPDU 都被输出到光纤上,如图 6.41(b)所示。领头的 TPDU 现在位于 Brawley(布劳利)附近的某个地方(仍然在南加利福尼亚州)。然而,发送方在得到一个新的窗口更新通知之前它必须停止发送数据。

在 20ms 以后,领头的 TPDU 到达波士顿,如图 6.41(c)中所示,并且该 TPDU 被确认。最后,从开始发送数据算起经过 40ms 以后,第一个确认回到发送方,第二批数据现在可以被传输了。由于在 40ms 之中这条传输线路只被使用了 0.5ms,所以,它的有效利用率大约为 1.25%。对于运行在千兆位线路上的老式协议来说,这种情况是非常典型的。

在分析网络性能的时候需要记住的一个很有用的数值是带宽-延迟乘积(bandwidth-delay product)。它是由带宽(单位为 bps,即位/秒)与往返延迟时间(单位为秒)相乘而得到的。此乘积值是从发送方至接收方之间的管道的往返容量(单位为位)。

对于图 6.41 中的例子,带宽-延迟乘积是 40 兆位。换句话说,发送方在第一个确认回来之前必须发送 40 兆位数据才能使管道保持全速状态。它需要用这么多位数据来填



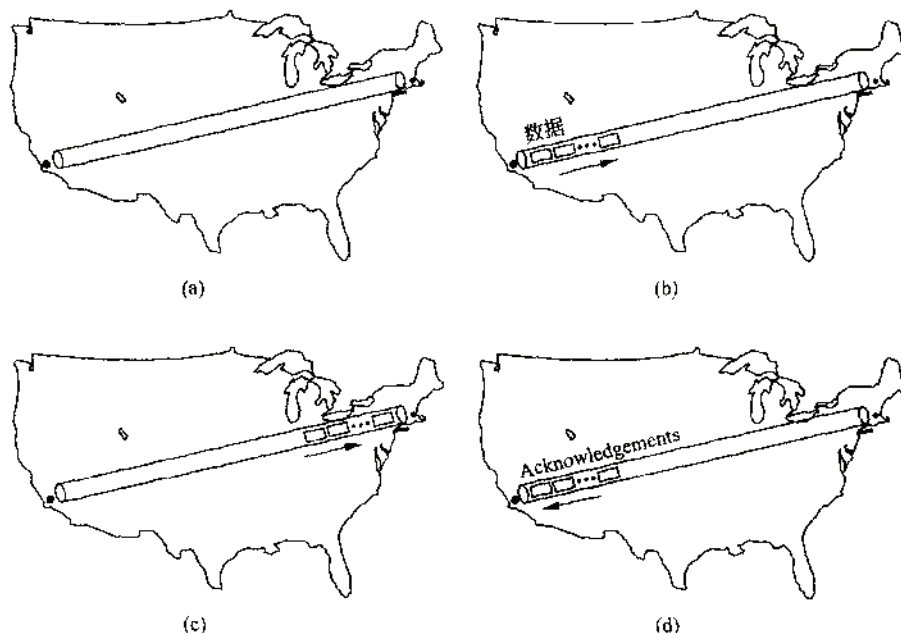


图 6.41 从圣地亚哥向波士顿传送 1 兆位数据的状态：  
(a)  $t=0$  时刻；(b)  $500\mu\text{s}$  以后；(c)  $20\text{ms}$  以后；(d)  $40\text{ms}$  以后

满该管道(包括两个方向)。这就是为什么传输半兆位数据只能获得 1.25% 的有效利用率：它只是管道容量的 1.25%。

现在可以得出结论：为了更好的性能，接收方的窗口必须至少与带宽-延迟之乘积一样大，最好比该乘积值还要大一点，因为接收方可能并不立刻就做出响应。对于越洋千兆位线路来说，至少需要 5 兆字节的窗口。

如果发送一兆位数据的时候效率非常低，那么，想象一下发送一个只有几百字节的短的请求时又会怎么样呢？除非当第一个客户在等待应答时这条线路还有其他的用途，否则千兆位的线路并不比一兆位的线路更好，只是费用更贵一些而已。

对延迟比较敏感的应用，比如音频和视频应用，它们的另一个性能问题是抖动。仅仅有较短的平均传输时间还不够。这些应用还要求较小的标准偏差。要想同时获得较短的平均传输时间和较小的标准偏差，这需要在工程上经过严格的质量把关和其他的努力。

## 6.6.2 网络性能的测量

当一个网络的运行效果很差的时候，它的用户通常会向网络运行商抱怨并要求提高网络的质量。为了改善网络的性能，网络操作人员首先必须确定发生了什么问题。为了找出真正的问题所在，操作人员必须进行测量工作。在这一小节中，我们来看一看网络性能的测量问题。下面的讨论以 Mogul(1993)的工作为基础。

用来改善网络性能的基本循环过程包括以下步骤：

- (1) 测量有关的网络参数和性能。

(2) 试图理解当前的网络状况。

(3) 改变一个参数。

这些步骤不断重复,直到网络的性能已经足够好,或者改善性能的全部空间都已经被发掘出来了。

测量工作可以有许多做法,也可以在许多地点或场所进行(既指物理位置,也指协议栈中的位置)。最基本的一种测量手段是:在开始某一个动作的时候启动一个定时器,然后确定该动作需要多长时间。例如,知道一个 TPDU 需要多长时间才被确认是一个很关键的测量指标。其他有一些测量指标可以通过计数器来完成,即记录某种事件发生的次数,比如丢失的 TPDU 的数量。最后,人们通常对于某些事物的数量比较感兴趣,比如在特定的时间间隔内所处理的字节数。

测量网络的性能和参数有许多潜在的陷阱。以下我们列出其中一部分。任何一种系统化的网络性能测量手段都应该小心地避免这些陷阱。

#### **确保样本空间足够大**

不要测量发送一个 TPDU 的时间,而是重复地测量,比如说测量 1 百万次,然后再取平均。采用大量的样本将可以减小所测量的均值和标准方差中的不确定性。这种不确定性可以利用标准的统计公式来计算。

#### **确保样本具有代表性**

理想情况下,这 1 百万次测量的完整序列应该在一天或者一周的不同时刻进行重复,从而可以看到不同的系统负载对于所测量指标的影响。例如,对于拥塞的测量,如果仅仅在没有拥塞的那一时刻来测量拥塞,则这样的测量和结果并没有用。有时候测量结果初看起来可能不符合直觉,比如在 10、11、1 和 2 点钟网络严重拥塞,但是中午时候没有拥塞(所有的用户都去吃午饭去了)。

#### **当使用粗粒度时钟的时候一定要谨慎**

计算机时钟的工作原理是,每隔固定的时间间隔就递增某一个计数器。例如,一个毫秒定时器每隔 1ms 就让一个计数器加 1。使用这样的定时器来测量一个持续时间小于 1ms 的事件是有可能的,但要非常小心。(当然,有些计算机还有更加精确的时钟。)

例如,为了测量出发送一个 TPDU 所需要的时间,当进入传输层代码时以及离开传输层代码时,应该将系统时钟(比如说以毫秒为单位)读出来。如果 TPDU 真正的发送时间是  $300\mu\text{s}$ ,则两次读取的时间之差要么是 0,要么是 1,这两个结果都是错误的。然而,如果重复测量 1 百万次,则所有测量的总和类加起来,再除以 1 百万,则平均时间比  $1\mu\text{s}$  还要精确得多。

#### **确保在测试过程中不会发生不可预知的事情**

在一个大学的网络系统中进行测量有可能发生这样的情况:有一天当一个大型的实验项目正在运行的时候你测量的结果跟第二天测量出来的结果可能会有所不同。同样

地,如果有的研究人员决定在你们的网络上运行一个视频会议,而这时候你正好在测量,那么你得到的结果可能会有偏差。你最好在一个空闲的系统上运行测试过程,并且根据需要自己来创建所有的工作负载。不过这种做法也有缺陷。你可能认为在凌晨3点钟的时候不会有人使用网络,但是,当自动备份程序这时候开始将所有的磁盘数据复制到磁带上的时候,你的想法就不再正确了。而且,此时其他时区的用户可能会访问你精美的WWW页面,从而也导致繁重的流量。

### 缓存机制可能会破坏测量的正确性

为了测量文件传输时间,最显然的方法是打开一个大的文件并读取文件中所有的数据,再关闭文件,然后看这个过程花了多长时间。然后,多次重复这样的测量过程以便获得一个好的平均值。然而,麻烦在于,系统可能会将文件缓存起来,所以,仅仅第一次测量才真正涉及到网络传输,其他的测量只不过从本地的缓存中读取数据而已。因此,这样的测量结果本质上是毫无价值的(除非你的目标是为了测量缓存机制的性能)。

通常你只要简单地采用溢出缓存的方法就可以避免缓存带来的问题。例如,如果缓存空间的大小为10MB,那么,测试循环可以轮流地打开、读取和关闭两个10-MB文件,这样做的目的是强迫缓存的命中率为0。不过,除非你绝对确定自己理解了缓存算法,否则仍然要非常小心。

缓冲机制也有类似的影响。一个很流行的TCP/IP性能测试工具曾经报告UDP可以获得比物理线路允许的能力还要好得多的性能。这是怎么发生的呢?在调用UDP的时候,通常当内核接受了消息之后,控制权马上就返回给应用程序了,而消息则被加入到传输队列中。如果主机上有足够的缓冲区空间的话,则执行1000次UDP调用并不意味着所有的数据都已经被发送出去了,大多数数据仍然在内核中,但是性能测试工具认为它们都已经被传送出去了。

### 理解你所测量的指标

当你要测量读取一个远程文件所需要的时间时,你的测量结果取决于以下诸多因素:网络、客户和服务端端的操作系统、所使用的硬件接口卡、接口卡的驱动程序,等等。如果你谨慎地执行了测量过程的话,那么,你最终得到的结果是在你所使用的配置环境中的文件传输时间。如果你的目标是要调整这一特殊的配置环境,那么这些测量结果将是非常有用的。

然而,如果你在三个不同的网络系统上进行类似的测量以便决定应该选购哪一块接口卡,那么你的结果可能完全不具备参考价值,因为其中一个网络驱动程序非常糟糕,它只能发挥接口卡的10%性能。

### 在往外推广结果时要谨慎

假设你利用模拟的网络负载来测量某一性能指标,比如网络的负载从0(空闲)到0.4(40%的容量),如图6.42中的数据点和实线所示。看到这样的结果,你很容易想到按线性方式将结果推广到40%以上的区域,如图中的点线所示。然而,许多与队列机制有关

的结果都会牵涉到一个因子  $1/(1-\rho)$ , 这里  $\rho$  是负载, 所以, 真正的结果可能看起来更像虚线所示, 它的上升速度比线性要快得多。

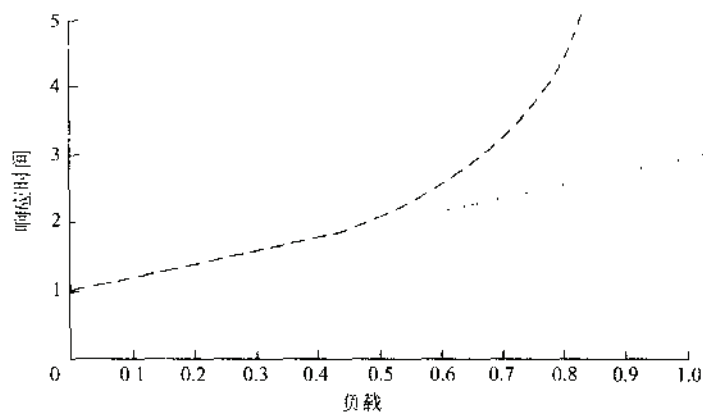


图 6.42 响应时间与负载之间的函数关系

### 6.6.3 具有更好性能的系统设计

测量网络的性能并且有针对性地进行修补通常可以显著地提高网络的性能,但是,它们不可能代替一个良好的设计。一个设计很差的网络的改进余地毕竟有限,除了改进的办法以外,最好的做法是从头开始重新设计。

在这一小节中,我们将根据许多网络的切实经验来展示一些经验规则。这些规则涉及到了系统设计,而不仅仅是网络设计,因为软件和操作系统通常比路由器和接口卡更加重要。其中大多数思想对于熟练的网络设计者们来说是常识,是经过一代又一代口授相传得来的。这些规则最初由 Mogul(1993)明确地整理出来,我们这里的介绍主要来源于此。另外一份参考材料是(Metcalfe, 1993)。

#### 规则 1: CPU 速度比网络速度更加重要

长久以来的经验表明,几乎在所有的网络中,操作系统和协议的开销占据了实际数据传输时间的大部分。例如,在理论上,以太网上的最小 RPC 时间是  $102\mu\text{s}$ ,它对应于最小的(64 字节)请求和紧随其后的最小(64 字节)应答。而在实践中,克服软件的开销以便使 RPC 时间尽可能接近理论值,这是一项非常重要的成绩。

类似地,在 1Gbps 上运行的最大问题是如何快速地将数据位从用户的缓冲区中转移到光纤上,以及让接收方的 CPU 尽可能快速的处理这些到来的数据位。简而言之,如果你的 CPU 速度加快了一倍,那么,你通常可以获得接近两倍的吞吐量。仅仅将网络的容量加倍通常没有效果,因为瓶颈一般在主机上。

#### 规则 2: 减少分组的数量可以减少软件开销

在处理 TPDU 的过程中,每个 TPDU 都有一定量的固定开销(比如,头部的处理)。

每个字节也有一定量的处理开销(比如计算或验证校验和)。当发送 1 兆字节的时候,不管 TPDU 的长度为多少,每字节的开销是一样的。然而,若使用 128 字节的 TPDU,则意味着它的每 TPDU 开销是使用 4 KB TPDU 方案的 32 倍。因此,这份开销快速地增加上来了。

除了 TPDU 开销以外,在下面层中的一些开销也值得考虑。每个到来的分组都会引发一个中断。在一个现代的流水线方式的处理器上,每个中断都会打断 CPU 的流水线、干扰缓存的工作、要求改变内存管理环境,并且强迫保存相当数量的 CPU 寄存器。因此,在发送数据时将 TPDU 的数量减少  $n$  倍,则中断数和分组开销也相应地减少  $n$  倍。

这个结论可以进一步推广:收集足够数量的数据之后再传送出去,从而减少另一端的中断数。Nagle 算法和 Clark 针对愚笨窗口综合症解决方案正是希望做到这一点。

### 规则 3: 使环境切换的次数减到最少

环境切换(比如从内核模式切换到用户模式)对于性能有严重的影响。如同中断一样,它们也具有同样的坏特性,最糟的是导致原来大量的缓存无法被命中。通过以下办法可以减少环境切换:让发送数据的库过程在内部建立一套缓冲机制,直到有了相当数量的数据之后才真正调用发送过程。类似地,在接收端,进来的小 TPDU 应该被收集起来,然后再成批地而不是单独地传递给用户,这样可以使环境切换的次数尽可能地减到最少。

在最好的情形下,一个进来的分组会导致从用户模式切换到内核模式,然后再切换到接收进程,并将新到达的数据交给它。不幸的是,在许多操作系统中,除此以外还需要额外的环境切换。例如,如果网络管理器以特殊进程的形式运行在用户空间中,那么,当一个分组到来的时候,处理的过程很可能是这样的:首先引发一次从当前用户到内核的环境切换,然后再从内核切换到网络管理器,之后再切换回到内核,最后从内核切换到接收进程。这个切换过程如图 6.43 所示。在每个分组上的所有这些环境切换非常浪费 CPU 时间,同时也严重地影响了网络的性能。

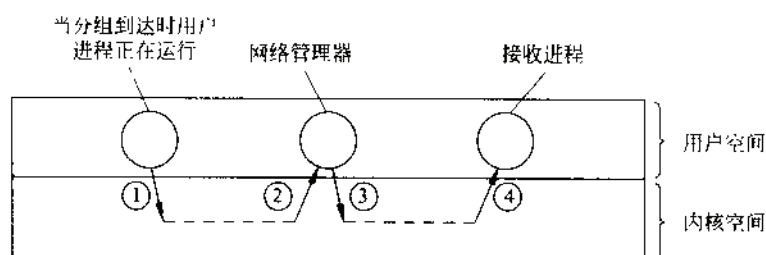


图 6.43 在用户空间运行网络管理器时,处理一个分组需要 4 次环境切换

### 规则 4: 使复制操作的次数减到最少

比多次环境切换更糟糕的是多次复制操作。对于一个新到达的分组,在提取出它的 TPDU 之前对该分组执行 3 次或者 4 次复制操作,这样的情形并不罕见。当网络接口卡将一个分组接收到专门的卡上硬件缓冲区中以后,该分组通常要被复制到内核缓冲区中。

在内核中,它又被复制到网络层缓冲区中,然后被复制到传输层缓冲区中,最后被复制到接收应用进程中。

一个精巧的操作系统每次会复制一个字,但是在常见的做法中,每个字要求大约 5 条指令(加载、保存、索引寄存器递增、测试数据末尾和一个条件分支)。假设每个分组执行 3 次复制,每个 32 位字的复制操作需要 5 条指令,则每个字节的复制需要  $15/4$  或者大约 4 条指令。在一个 500MIPS 的 CPU 上,一条指令需要 2ns,所以,每个字节需要 8ns 处理时间,或者每一位大约 1ns,因此最大的处理数据速率大约为 1Gbps。当考虑了其他各种开销,包括头部的处理、中断处理和环境切换以后,或许还能达到 500Mbps,不过请注意,我们这里还没有考虑实际的数据处理过程。很明显,要想处理全速运行的 10Gbps 以太网目前还不可能。

实际上,在全速状态下,可能连 500Mbps 的线路都还无法处理。在上面的计算中,我们假定一台 500MIPS 的机器每秒钟可以执行任意 500 兆条指令。而事实上,只有当该机器没有引用内存的时候它才可以运行在这样的速度上。内存操作通常比寄存器与寄存器之间的指令要慢 10 倍左右(即每条指令大约 20ns)。如果 20% 的指令要引用内存(即缓存没有命中),这在处理进来的分组时是很有可能,那么,平均的指令执行时间为  $5.6\text{ns}$  ( $0.8 \times 2 + 0.2 \times 20$ )。由于每个字节需要执行 4 条指令,所以每个字节要花费 22.4ns 时间,或者 2.8ns/位,这样得到的结果大约为 357Mbps。考虑了 50% 的开销之后我们可以得到 178Mbps。请注意,这里使用硬件也无济于事。问题的根源在于操作系统执行了太多的复制操作。

#### **规则 5: 你可以购买更多的带宽,但无法要求更低的延迟**

接下来的三条规则涉及到通信,而不再是协议处理。第一条规则是:如果你想要更多的带宽,那么你直接购买带宽即可。在第一条光纤旁边再铺设一条光纤就可以将带宽翻一番,但这样做并不能降低延迟。如果想要缩短延迟,则必须改进协议软件、操作系统或者网络接口。即使所有这些你都做到了,如果瓶颈是传输时间的话,则延迟还是无法降低下来。

#### **规则 6: 应该想办法避免拥塞,而不是从拥塞中恢复**

“一分预防胜过十二分治疗”这句古老的格言无疑特别适用于网络拥塞。当一个网络拥塞的时候,分组会丢失,带宽被浪费,传输过程中引入无用的延迟,等等。从拥塞中恢复过来并不容易,这需要时间和耐心。最好的做法是从一开始就不让拥塞发生。避免发生拥塞就好像接种 DTP 疫苗(白喉、破伤风及百日咳的三合一疫苗)一样:当你接种的时候它会让你疼一下,但是它让你避免将来更多的疼痛。

#### **规则 7: 避免超时**

定时器在网络中是必要的,但是它们应该尽量少用,而且超时应该尽量少发生。当一个定时器到期的时候,通常需要重复执行某一个动作。如果确实需要重复执行这个动作,则执行该动作;否则,不必要的重复就是一种浪费了。



避免额外工作的办法是小心地将定时器设置为稍稍超过保守的时间边界一点点。如果定时器的超时间隔设置得太长的话,则当 TPDU 丢失(不太可能)的时候,它所在的连接上将会增加少量的额外延迟。如果一个定时器在本不该到期的时候却超时了,则它 would 用掉宝贵的 CPU 时间,浪费掉带宽,而且毫无理由地给(可能)很多路由器增加了额外的负担。

#### 6.6.4 快速的 TPDU 处理

上一小节的基本思想是,妨碍网络提高速度的主要障碍是协议软件。在这一小节中,我们将讨论一些加速协议软件的方法,要想了解更多的信息,请参考(Clark et al., 1989; and Chase et al., 2001)。

TPDU 的处理开销分为两部分:每 TPDU 的开销(overhead per TPDU)和每字节的开销(overhead per byte)。这两部分都有改进的余地。快速 TPDU 处理的关键是,分离出正常的发送操作(单向数据传输)并对它们作特殊处理。尽管 TCP 需要一个特殊的 TPDU 序列才能使一个连接进入到 ESTABLISHED 状态,但是,一旦进入该状态之后,TPDU 的处理就非常简单了,并且一直持续到有一方主动关闭该连接为止。

我们首先来讨论位于 ESTABLISHED 状态中的发送端,看一看它有数据要发送时的情形。为了清楚起见,这里假定传输实体运行在内核中,不过,如果传输实体是一个用户空间的进程,或者是发送进程内部的一个库,则同样的想法也适用。在图 6.44 中,发送进程进入(trap)到内核中并执行 SEND。传输实体所做的第一件事情是测试一下,看是否为正常的发送操作:状态为 ESTABLISHED,双方均不打算关闭当前连接,被发送的是一个普通的(即不是带外的[out-of-band])完整的 TPDU,而且接收方有足够的窗口空间可供使用。如果所有的条件都满足,则无需进一步测试就可以使用发送端传输实体的快速路径。在通常情况下,大部分时间都通过该快速路径来传送数据。

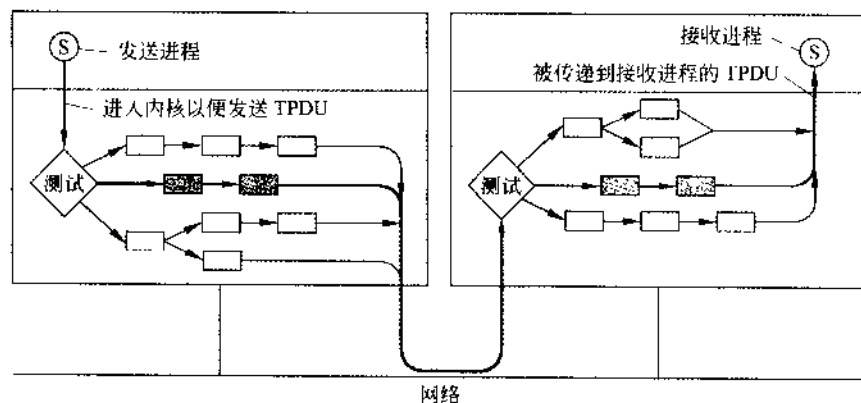


图 6.44 粗线显示了从发送方至接收方的快速路径。该路径上的处理步骤用阴影来表示

通常情况下,连续的数据 TPDU 的头几乎是相同的。为了充分利用这样的特点,传输实体可以在内部保存一个原型头。然后,在快速路径的起始处,该原型头被尽可能快速

地逐字复制到一个空闲的缓冲区中。对于那些随着不同的 TPDU 而有所不同的域,缓冲区中的这些域随后被改写。这些域往往可以很容易地从状态变量中获得,比如“下一个序列号”。然后传输实体将一个指向完整 TPDU 头的指针和一个指向用户数据的指针传递给网络层。网络层可以遵循同样的策略(图 6.44 中没有显示)。最后,网络层将结果所得到的分组交给数据链路层,由它传输出去。

现在我们考虑 TCP/IP 作为以上原则在实践中如何工作的一个例子。图 6.45(a)显示了 TCP 头。有的域在单向数据流的连续 TPDU 之中总是相同的,图中用阴影来表示这些域。发送端传输实体所需要做的事情是:将 5 个字从原型头复制到输出缓冲区中,填充下一个序列号(从内存中的一个字复制过来),计算校验和,递增内存中的序列号。然后将该头部和数据交给一个特殊的 IP 过程,由它来发送一个常规的、最大的 TPDU。然后 IP 将它的 5 字原型头[如图 6.45(b)]复制到缓冲区中,并且填充 Identification(标识)域,然后计算它的校验和。现在这个分组已经作好往外传送的准备了。

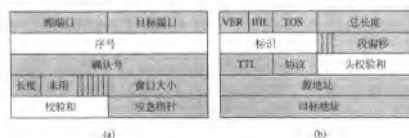


图 6.45

(a) TCP 头; (b) IP 头

在这两个头中,阴影域可直接从原型头中复制过来,无需任何改变

现在我们来一看图 6.44 中接收端的快速路径处理过程。第 1 步是找到与进来的 TPDU 相对应的连接记录。对于 TCP,连接记录可能被保存在一张散列表中,该散列表的键是两个 IP 地址和两个端口的某一个简单函数。一旦定位到该表中的连接记录之后,则比较两个地址和两个端口,以验证是否找到了正确的连接记录。

为了进一步加速连接记录查找过程,通常可以使用一种优化措施,它的做法是维护一个指针,让它指向最近使用过的连接记录,并且在开始查找时首先对它进行测试。Clark 等(1989)试用了这种方法,发现它的命中率超过 90%。(McKenney and Dove, 1992)中描述了其他一些探查方法。

找到了连接记录之后,然后对进来的 TPDU 进行检查,看它是否是一个正常的 TPDU。连接的状态是 ESTABLISHED,双方均不打算关闭该连接,此 TPDU 是一个完整的 TPDU,它没有设置特殊的标志,并且它的序列号正好是接收方期望的序列号。这些测试只是一些指令而已,如果所有的条件都满足,则调用一个特殊的快速路径 TCP 过程。

快速路径过程更新该连接记录,并且将数据复制给用户。在复制的时候,它也计算校验和,从而避免了两对数据进行遍历操作。如果校验和正确的话,则更新连接记录,并

送回一个确认。这种“首先快速地检查一下头部看是否是自己所期望的 TPDU,然后让一个特殊的过程来处理这种 TPDU”的一般方案称为头部预测法(header prediction)。许多 TCP 实现使用了这种方法。当这种优化方案与本章中讨论的其他优化方案一起使用的时候,就有可能使 TCP 的运行速度达到本地的内存至内存复制速度的 90%(这里假设网络本身的速度足够快)。

有可能获得较大性能增益的另两个领域是缓冲区管理和定时器管理。缓冲区管理中的主要问题是避免不必要的复制操作,上一小节已经介绍过了。定时器管理是非常重要的,因为几乎所有的定时器设置都不应该过期。之所以设置定时器是为了应对 TPDU 丢失的情形,但是,大多数 TPDU 都会正确地到达,而且它们的确认也会正确地到达。因此,在定时器很少出现超时的情况下,对定时器的管理进行优化是非常重要的。

一种常见的方案是,利用链表结构将定时器事件保存起来,并且按过期时间进行排序。链表中的每个记录包含一个计数器,其中表头记录中的计数器指明了离过期时间还有多少个滴答。每个后续记录中包含的计数器指明了在前一个记录过期之后还有多少个滴答。因此,如果 3 个定时器分别在 3、10 和 12 个滴答之后过期的话,则三个计数器分别为 3、7 和 2。

在每一个时钟滴答到来时,表头记录中的计数器被减 1。当它减到 0 的时候,它的事件被处理,并且链表中的下一个记录变成链表的头。新表头记录的计数器值不用改变。在这种方案中,插入和删除定时器是高开销的操作,其执行时间正比于链表的长度。

如果最大的定时器间隔是有界的,并且可以预先知道的话,则我们可以使用一种更加高效的方法。这里需要用到一个数组,被称为定时轮(timing wheel),如图 6.46 所示。每个时间槽对应于一个时钟滴答。图中显示的当前时间为  $T=4$ 。三个定时器分别被安排在从现在开始 3、10 和 12 个滴答之后过期。如果一个新的定时器突然被设置成在 7 个滴答之后过期,那么,只需在第 11 槽中加入一个记录即可。类似地,如果要取消一个在  $T+10$  之后过期的定时器,则只要搜索第 14 槽中起头的链表,并将所请求的记录移除即可。请注意,图 6.46 中的数组不可能容纳超过  $T+15$  的定时器。

当时钟滴答到来的时候,当前时间指针向前移动一个时间槽(循环移动,即到达最底下之后又回到最上边)。如果现在指向的记录不为 0,则所有的定时器都要被处理。在 (Varghese and Lauch, 1987) 中讨论了定时轮思想的许多变种。

### 6.6.5 针对千兆网络的协议

九十年代初,千兆网络开始出现。人们最初的反应是在千兆网络上使用老的协议,但是各种各样的问题很快就来了。在这一小节中我们将讨论其中的一些问题,同时也指出,当我们转移到更加快速的网络时,我们可以使用新的协议来解决这些问题。

第一个问题是,许多协议使用了 32 位序列号。当 Internet 开始的时候,路由器之间的线路主要是 56kbps 的租用线路,所以,一台全速发送数据的主机需要一星期的时间才用完一轮序列号(即导致序列号回绕)。对于 TCP 的设计者们来说, $2^{32}$  可能在当时是一个近似于无穷大的数,因为一周以前发送的老分组仍然停留在网络中的危险性几乎是不存在的。对于 10Mbps 以太网,回绕时间变成了 57 分钟,尽管时间更短了,但仍然可以管

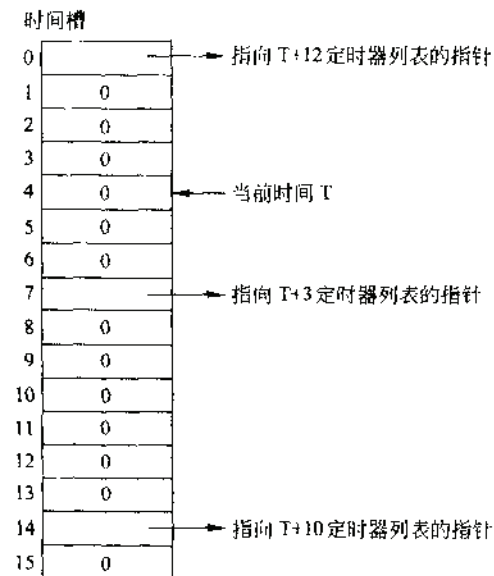


图 6.46 定时轮

理。对于 1Gbps 以太网,如果它全速发送数据到 Internet 上,则回绕时间大约是 34 秒钟,低于 Internet 上的最大分组生存期,即 120 秒。突然间, $2^{32}$  不再是一个很好的无穷大近似数了,而且,当老的分组仍然存在的时候,发送方就开始循环使用序列号空间了。不过, RFC 1323 提供了一种躲避这个问题的办法。

问题在于,许多协议设计者总是简单地,并且不加声明地假定:用完整个序列号空间所需要的时间大大超过最大分组生存期。因此,他们根本不用担心“当序列号发生回绕的时候,老的重复分组仍然停留在网络中”的问题。在千兆位速度上,这条未声明的假设不再成立。

第二个问题是,通信速度的提高比计算速度的提高要快得多。(给计算机工程师的建议:去打败那些通信工程师!我们的将来全靠你们了。)在上个世纪七十年代, ARPANET 的运行速度为 56kbps,它上面的计算机运行在大约 1MIPS 的速度上。分组为 1008 位,所以,ARPANET 每秒钟能够递交大约 56 个分组。每个分组几乎可以使用 18ms,主机可以花 18 000 条指令来处理一个分组。当然,这样做将会耗掉所有的 CPU 资源,但是它可以为每个分组花 9000 条指令,并且将剩下的一半 CPU 能力用于完成实际的工作。

我们将这些数字与“1000MIPS 的计算机通过一条千兆位的线路来交换 1500 字节的分组”的情形作一下比较。现在,分组流的速率为每秒钟超过 80 000 个,所以,如果我们希望保留一半 CPU 能力用于应用系统的话,则主机的系统必须在  $6.25\mu\text{s}$  以内完成分组的处理过程。在  $6.25\mu\text{s}$  中,1000MIPS 的计算机可以执行 6250 条指令,仅相当于 ARPANET 主机所用指令数的 1/3。而且,现代的 RISC 指令系统中的每条指令比老式的 CISC 指令系统中的每条指令所做的工作更少,所以,这个问题比表面上看到的更加糟

糕。结论如下：随着技术的发展，用于协议处理的时间比过去更少了，所以，协议必须更加简单才行。

第三个问题是，如果一条线路的“带宽-延迟之乘积”非常大，则在这条线路上使用回退  $n$  步协议<sup>①</sup>的性能将非常差。例如，请考虑一条 4000km 长的 1Gbps 线路。往返传输时间是 40ms，在这段时间中发送方可以传输 5 兆字节数据。如果一个错误被接收方检测出来，那么，发送方接到错误通知的时候，已经过去 40ms 了。如果使用回退  $n$  步协议，则发送方不仅要重传坏的分组，而且也要重传该分组后面的 5 兆字节数据。很明显，这是资源的极大浪费。

第四個问题是，千兆位线路本质上不同于 1 兆位线路，因为长的千兆位线路的主要制约因素是延迟，而不是带宽。在图 6.47 中，我们看到以各种不同的传输速度将一个 1 兆位文件传输 4000 公里所需要的时间。在速度增加到 1Mbps 以前，传输时间主要受到发送速率的制约；到 1Gbps 的时候，40ms 的往返延迟时间远远超过了将数据发送到光纤上所需要的 1ms 时间。进一步增加带宽也不会有任何帮助。

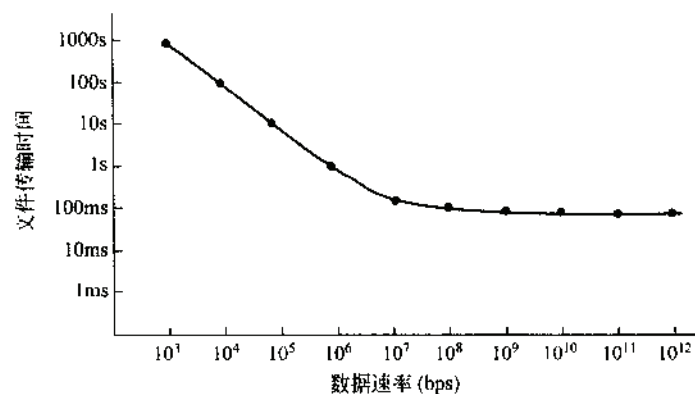


图 6.47 在 4000 公里的线路上传输并确认一个 1 兆位文件所需要的时间

对于网络协议来说，图 6.47 有一些不样的启示。从图中可以看出，比如像 RPC 这样的停-等协议有一个固有的性能上界。此限制是由于光速造成的，光学技术再怎么进步也无法改善这种状况（不过，新的物理学法则可能会有所帮助）。

值得提及的第 5 个问题与其他的问题不同，它不是技术性的，也不涉及到协议，而是新的应用导致的结果。简单来说，问题是这样的：对于许多千兆位应用，比如多媒体应用，分组到达时间的偏差与平均延迟本身一样重要。“慢速但一致”的递交速率通常更加优于“快速但跳跃”的速率。

以上介绍的都是问题，现在我们转到这些问题的处理办法上来。首先我们给出一些一般性的论述，然后讨论协议机制、分组的布局结构和协议软件。

所有的千兆网络设计者应该铭记于心的基本原则是：

<sup>①</sup> 译注：这里的回退  $n$  步协议 (go back  $n$  protocol) 是泛指。在传输层上，我们曾经提到过回退  $n$  段协议；而在数据链路层上，我们在第 3 章看到过回退  $n$  帧协议。

为了速度而设计,不是为了带宽优化而设计。

老的协议的设计目标通常是使线路上传输的数据尽可能地降低到最小,一般的做法是使用小的域,并且将这些小域组合起来包装到字节或者字中。现在,带宽已经足够了,而协议处理成了问题,所以,在设计协议的时候应该考虑将协议处理的负担降低到最小。显然,IPv6 的设计者们很清楚这条原则。

一种很吸引人的加速办法是用硬件来构建快速的网络接口。这种策略的困难在于,除非协议极其简单,否则,所谓硬件就意味着是一块带有第二个 CPU 和自己独立程序的插卡。为了确保网络协处理器比主 CPU 便宜,它往往是一块相对慢速的芯片。这种设计的直接结果是,主(快速的)CPU 的许多时间是空闲的,它要等待第二个(慢速的)CPU 来完成关键性的工作。认为主 CPU 在等待期间还有其他的工作要做,这是不切实际的。而且,当两个通用 CPU 相互通信的时候,可能会发生竞争条件,所以,两个处理器之间的通信协议需要精心设计,以保证两者能正确地同步。通常,最好的做法是使协议尽可能地简单,并且让主 CPU 来完成协议处理工作。

现在我们来考察一下在高速协议中反馈的问题。由于(相对)较长时间的延迟回路,在协议中应该尽可能地避免反馈:接收方给发送方送回信号需要很长时间。第一个反馈例子是利用滑动窗口协议来控制传输速率。为了避免由于接收方给发送方发送窗口更新信息而造成固有的(长时间)延迟,最好的做法是使用基于速率的协议。在这样的协议中,发送方可以发送所有它希望传送的内容,只要发送的速率不超过发送方和接收方预先商量好的速率即可。

第二个反馈例子是 Jacobson 的慢启动算法。这个算法执行多次探查以确定网络的处理能力。对于高速网络,执行几次小的探查来确定网络的响应能力将会浪费大量带宽。一种更加有效的方案是,让发送方、接收方和网络三者在建立连接的时候都预留必要的资源。提前预留资源还有另一个好处,即更加容易减少抖动。简而言之,越是向高速发展,则越是无情地将设计方案推向面向连接的操作,或者非常接近于面向连接的操作。当然,如果将来带宽变得非常充足,以至于人们不再关心是否会浪费大量带宽,那么,设计规则将变得很不一样。

分组的布局结构也是千兆网络中一个很重要的考虑因素。头部应该尽可能地少包含一些域,以便减少处理时间;而且头部中的域应该足够大以便能胜任它的职责,另外,这些域应该是按字对齐的,以方便处理。这里提到的“头部中的域应该足够大”是指像“序列号回绕时老的分组仍然存在”或者“由于窗口域太小而导致接收方不能宣告足够大的窗口空间”等诸如此类的问题不再发生。

头部和数据部分应该被单独校验,这有两方面的原因。第一,这使得有可能只检验头部,不校验数据。第二,在把数据复制到用户空间之前可以先验证头部是正确的。只有在将数据复制到用户空间的时候,传输实体才有必要对数据部分进行校验和检查;如果头部不正确的话,则有可能把数据复制到错误的进程中。为了避免这种不正确的复制动作,但又允许在复制过程中对数据进行校验和检查,因而有必要将两个校验和分开。

最大的数据长度应该非常大,从而即使在延迟时间很长的情况下仍能够有效地工作。而且,数据块越大,头部所占总带宽的比例就越小。1500 字节显得太小了。



另一个有价值的特性是在连接请求过程中顺带发送正常数量的数据的能力。按照这种方法,可以节省一次往返时间。

最后谈一下关于协议软件的问题。一个关键的思想是将注意力集中在成功的情形上。许多老的协议总是倾向于将重点放在发生错误(比如分组丢失了)时该怎么办的问题上。为了使协议运行得更加快速,设计者应该努力的目标是,当一切都很正常的时候将处理时间降低到最小。当发生错误时使处理时间降低到最小,这是相对次要的。

第二个软件问题是复制时间的最小化。正如我们前面提到过的,复制数据往往是主要的开销来源。理想情况下,应该由硬件将每个进来的分组当作一个连续的数据块转储到内存中。然后,通过一个块复制操作,由软件将分组复制到用户缓冲区中。根据缓存机制的工作方式,甚至有可能希望避免使用复制循环。换句话说,为了复制 1024 个字,最快速的方法可能是 1024 条紧密相连的 MOVE 指令(或者 1024 对“加载-保存”指令)。复制过程非常关键,所以最好的做法是小心地用汇编语言来手工编写复制代码,除非有一种办法可以让编译器精确地产生最优的代码。

## 6.7 本章小结

传输层是理解分层协议的关键。它提供了各种服务,其中最重要的服务是一个从发送方至接收方之间端到端的、可靠的、面向连接的字节流。通过一组服务原语可以访问此服务(允许建立、使用和释放连接)。Berkeley socket(套接字)提供了一个通用的传输层接口。

传输协议必须能在不可靠的网络上完成连接管理工作。由于有些延迟的重复分组可能会在不恰当的时刻出现,因而导致连接的建立过程变得非常复杂。为了处理这样的情形,需要使用三步握手法来建立连接。释放连接比建立连接要容易得多,但由于存在两军队问题,故释放连接也并非轻而易举。

即使在网络层完全可靠的情况下,传输层也有大量的工作要做。它必须处理所有的服务原语,管理连接和定时器,以及分配和使用信用。

Internet 有两个主要的传输协议:UDP 和 TCP。UDP 是一个无连接的协议,它主要是对 IP 分组进行了包装,但同时也引入了额外的特性,即:让多个进程复用同一个 IP 地址,并且在接收分组的时候解复用到多个进程中。UDP 可以被用于客户-服务器之间的交互过程,例如,使用 RPC 来实现客户和服务器之间的交互。UDP 也可以被用来建立实时协议,比如 RTP。

主要的 Internet 传输协议是 TCP。它提供了一个可靠的双向字节流。TCP 在所有的数据段上使用了一个 20 字节的头。在 Internet 中,数据段可能会被路由器分成更小的段,所以,主机必须具备分段重组的能力。目前已经有大量的工作深入到 TCP 性能优化的领域中,它们主要使用了 Nagle、Clark、Jacobson、Karn 和其他的算法。无线链路为 TCP 增加了各种各样的复杂性。事务型 TCP 是对 TCP 的一个扩展,它使用较少数量的分组来处理客户-服务器之间的交互。

网络性能通常是由协议和 TPDU 的处理开销来决定的,在速度很高的时候这种状况

变得更加糟糕。在设计协议时应该使以下几个量尽可能地最小：TPDU 的数量、环境切换次数，以及每个 TPDU 被复制的次数。对于千兆网络，要求使用尽可能简单的协议。

## 习 题

1. 在图 6.2 的传输原语例子中，LISTEN 是一个阻塞调用。这是必要的吗？如果不是，请解释如何有可能使用一个非阻塞的原语。与正文中描述的方案相比，你的方案有什么优点？

2. 在图 6.4 所示的模型中，它的假设条件是网络层的分组有可能会丢失，因此，分组必须被单独确认。假如网络层是百分之百可靠的，它永远不会丢失分组，那么图 6.4 需要做什么样的变化（如果需要的话）？

3. 在图 6.6 的两部分中，有一条注释说明了 SERVER\_PORT 在客户和服务端中必须相同。为什么这一条如此重要？

4. 假设采用时钟驱动方案来生成初始序列号，该方案用到了一个 15 位宽度的时钟计数器。并且，每隔 100ms 时钟滴答一次，最大分组生存期为 60s。请问，每隔多久需要重新同步一次？

(a) 在最差情况下？

(b) 当数据每分钟用掉 240 个序列号的时候？

5. 为什么最大分组生存期 T 必须足够大以便确保不仅分组本身消失而且它的确认也消失，然后协议才有效？

6. 想象用两步握手过程而不是三步握手过程来建立连接。换句话说，第三个消息不再需要了。现在有可能死锁吗？请给出一个例子，或者证明死锁不存在。

7. 想象一个泛化的 n-军队问题，在这个问题中，任何两支蓝军达成一致的意见之后就足以取得胜利。是否存在一个能保证蓝军必赢的协议？

8. 请考虑主机的崩溃恢复问题（即图 6.18）。如果写操作和发送确认之间的间隔可以相对非常小的话，那么，为了使协议失败的几率最小，两种最佳的发送方-接收方策略是什么？

9. 对于正文中描述的传输实体（见图 6.20），死锁有可能吗？

10. 图 6.20 所示的传输实体的实现者出于好奇心，因而决定在 sleep 过程中放上一些计数器，以便收集有关 conn 数组的统计信息。其中包括这样的信息：处于 7 种可能状态之一的连接数目， $n_i (i = 1, \dots, 7)$ 。他编写了一个很大的 FORTRAN 程序用于分析数据，在写完程序之后，他发现  $\sum n_i = \text{MAX\_CONN}$  这个关系总是成立的。请问，是否还有其他仅仅涉及到这 7 个变量的恒等关系？

11. 对于图 6.20 中给出的传输实体，当它的用户发送一个 0 长度的消息时会发生什么情况？请讨论你的答案的意义所在。

12. 在图 6.20 的传输实体中，针对可能发生的每一个事件，请说明当用户在发送状态（sending）睡觉的时候，该事件是否为合法的？

13. 请讨论一下信用协议和滑动窗口协议的优点和缺点。

14. UDP 为什么有必要存在? 难道只让用户进程发送原始的 IP 分组还不够吗?
15. 请考虑一个建立在 UDP 基础上的简单应用层协议,它允许客户从一个远程服务器获取文件,而且该服务器位于一个知名的地址上。客户首先发送一个请求,该请求中包含了文件名,然后服务器以一个数据分组序列作为响应,这些数据分组包含了客户所请求的文件的的不同部分。为了确保可靠性和顺序递交,客户和服务器使用了停一等协议。忽略显然存在的性能问题,你还看得到这个协议的另一个问题吗? 请仔细想一想进程崩溃的可能性。
16. 一个客户向 100km 以外的服务器发送一个 128 字节的请求,两者之间通过一条 1Gbps 的光纤进行通信。在远过程调用中这条线路的效率是多少?
17. 继续考虑上一个问题的情形。对于给定的 1Gbps 线路和 1Mbps 线路,请计算一下最小的可能响应时间。由此你可以得出什么结论?
18. UDP 和 TCP 在递交消息的时候,都使用端口号来标识目标实体。请给出两个理由说明为什么这两个协议要发明一个新的抽象 ID(端口号),而不是使用进程 ID(在设计这两个协议的时候,进程 ID 早已经存在了)。
19. 最小的 TCP MTU 的总长度(包括 TCP 和 IP 的开销但是不包括数据链路层的开销)是多少?
20. 数据报的分段和重组机制是由 IP 来处理的,对于 TCP 是不可见的。这是否意味着 TCP 不用担心数据错序到达的问题呢?
21. RTP 被用来传输 CD 品质的音频,这样的音频信号包含一对 16 位的采样值,每个采样值对应于一个立体声声道,采样的频率为每秒钟 44 100 次。请问 RTP 每秒钟必须传输多少个分组?
22. 请问有可能将 RTP 代码放到操作系统内核中,与 UDP 代码放在一起吗? 解释你的答案。
23. 主机 1 上的一个进程已经被分配了端口 p,主机 2 上的一个进程已经被分配了端口 q,请问这两个端口之间有可能同时存在两个或者多个 TCP 连接吗?
24. 在图 6.29 中我们看到,除了 32 位 Acknowledgement number(确认号)域以外,在第 4 个字中还有一个 ACK 位。这是否真正加入了有用的信息呢? 为什么是或为什么不是?
25. TCP 数据段的最大净荷是 65 495 字节。为什么选择了如此奇怪的一个数呢?
26. 请描述一下进入到图 6.33 的 SYN RCVD 状态的两种方法。
27. 请指出当 Nagle 算法被用在一个极其拥塞的网络上时它的一个潜在缺点。
28. 请考虑在一条往返时间为 10ms 的无拥塞线路上使用慢启动算法的效果。接收窗口为 24KB,最大数据段长度为 2KB。请问需要多长时间才发送满窗口的数据?
29. 假设 TCP 的拥塞窗口被设置为 18KB,并且出现了一个超时。如果接下来的 4 次传输全部成功的话,则接收窗口将是多大? 假设最大数据段长度为 1KB。
30. 如果 TCP 往返时间 RTT 当前是 30ms,接下来的确认分别在 26、32 和 24ms 之后到达,那么,若使用 Jacobson 算法,新的 RTT 估计值为多少? 请使用  $\alpha = 0.9$ 。
31. 一台 TCP 机器正在通过一条 1Gbps 的信道发送 65 535 字节的满窗口数据,该

信道的单向延迟为 10ms。请问,可以达到的最大吞吐量是多少?线路的效率是多少?

32. 一台主机在一条线路上发送 1500 字节的 TCP 净荷,其中最大分组生存期为 120s,要想不让序列号回绕,该线路的最快速度为多少?将 TCP、IP 和以太网的开销都考虑进去,假设以太网帧可以被连续发送。

33. 在一个网络中,最大的 TPDU 长度为 128 字节,最大的 TPDU 生存期为 30s,序列号为 8 位,请问每个连接的最大数据率是多少?

34. 假设你正在测量接收一个 TPDU 所需要的时间。当一个中断发生的时候,你读出系统时钟的值(以毫秒为单位)。当 TPDU 被完全处理之后,你再次读出时钟的值。你测量的结果是:270 000 次为 0ms,730 000 次为 1ms。请问,接收一个 TPDU 需要多长时间?

35. 一个 CPU 执行指令的速率为 1000MIPS。数据的复制可以按每次 64 位来进行,每个字的复制动作需要花费 10 条指令。如果一个进来的分组要被复制 4 次,那么,这个系统能处理一条 1Gbps 的线路吗?为了简单起见,假设所有的指令,包括读或者写内存的指令,都在 1000MIPS 的全速率上运行。

36. 为了处理当序列号回绕时老的分组仍然存在的问题,你可以使用 64 位序列号。然而,从理论上讲,光纤的运行速度可以达到 75Tbps。要求最大分组生存期为多少才可以确保将来的 75Tbps 网络不会发生回绕问题?假设使用 64 位序列号,并且像 TCP 那样每个字节都有自己的序列号。

37. 说出一个“在 UDP 上运行 RPC”比事务型 TCP 好的优势。再说出一个 T/TCP 比 RPC 好的优势。

38. 在图 6.40(a)中,我们看到,完成 RPC 需要 9 个分组。是否存在正好需要 10 个分组的情形呢?

39. 在 6.6.5 节中,我们曾经计算过,一条千兆位线路每秒钟可以向主机转储 80 000 个分组,但是主机只有 6250 条指令用于处理每个分组,剩下一半 CPU 时间留给应用系统。这个计算过程假设每个分组的大小为 1500 字节。请针对 ARPANET 的分组长度(128 字节)重新计算一遍。在这两次计算中,都假设所给出的分组长度包含了所有的开销。

40. 对于一个运行在 4000km 距离上的 1Gbps 网络,限制的因素是延迟,而并非带宽。请考虑这样一个 MAN:源端和目标端之间的平均距离为 20km。请问在什么数据率上,由于光速导致的往返时间等于 1KB 分组的传输延迟?

41. 请为下面的网络计算带宽-延迟之乘积:(1)T1(1.5Mbps);(2)以太网(10Mbps);(3)T3(45Mbps)和(4)STS-3(155Mbps)。假设 RTT 为 100ms。请回忆一下,本章曾经提到过,TCP 头有 16 位保留用于窗口大小(Window Size)。根据你的计算,能想出有什么隐含的意义吗?

42. 对于地球同步卫星上的一条 50Mbps 的信道,它的带宽-延迟之乘积是多少?如果所有的分组都是 1500 字节(包括开销),那么,窗口应该为多大(按分组为单位)?

43. 图 6.6 所示的文件服务器离完美差远了,它在许多方面都还可以改进。请作以下修改:

(c) 给客户增加第三个参数,它指定了一个字节范围。

(d) 给客户增加一个标志 `w`,通过该标志可以将文件写到服务器上。

44. 修改图 6.20 中的程序,使它具有错误恢复功能。增加一种新的分组类型: `reset`, 当一个连接被双方打开以后,在任何一方都没有关闭该连接的情况下这种分组可能会到达。这种事件在连接的两端同时发生,这意味着原先被送出去的任何一个分组要么已被递交,要么已被破坏,总之它不会还在子网中。

45. 写一个程序来模拟传输实体中的缓冲区管理,请使用滑动窗口来进行流控制,而不是使用图 6.20 中的信用系统。让高层的进程随机地打开连接、发送数据,然后关闭连接。为了使这个程序更加简单,让所有的数据都从机器 A 发向机器 B,而在反方向上没有数据。在 B 端用不同的缓冲区分配策略进行试验,比如为特定的连接使用专门的缓冲区,或者所有的连接使用一个公共的缓冲区池;针对每一种策略,测量一下所达到的总吞吐量。

46. 设计和实现一个聊天系统,它允许多组用户同时进行聊天活动。有一个聊天协调器位于某一个知名的网络地址上,它使用 UDP 与聊天客户进行通信,并且为每个聊天会话建立聊天服务器,而且还维护了一个聊天会话目录。每个聊天会话有一个聊天服务器。聊天服务器使用 TCP 与客户进行通信。在聊天客户中,用户可以启动、加入或者离开一个聊天会话。请设计和实现协调器、服务器和客户的代码。

---

## 第7章 应用层

---

在介绍了所有的预备知识以后,我们现在来讨论应用层,在这里你可以找到所有的网络应用。应用层下面的各层提供了可靠的传输,但它们并不真正为用户做事情。在本章中,我们将学习一些实际的网络应用。

然而,即使在应用层上,仍然有必要支持多种协议,以便使各种应用能够工作。因此,在开始介绍这些应用之前,我们将先介绍其中的一个协议,这就是 DNS,它在 Internet 上处理命名问题。之后,我们将介绍三个实际的应用:电子邮件、万维网(World Wide Web),最后是多媒体。

### 7.1 DNS—域名系统

虽然从理论上来说,通过网络(例如 IP)地址,程序可以访问主机、邮箱和其他资源,但是人们很难记住这些地址。而且,若发送电子邮件到 tana@128.111.24.41,则意味着,如果 Tana 所属的 ISP 或组织将邮件服务器迁移到了另一台使用不同 IP 地址的机器上,那么她的电子邮件地址必须也要跟着改变。因此,人们引入了 ASCII 名字以便将机器名与机器地址分离开。这样,Tana 的地址可能是 tana@art. ucsb. edu。但是,网络本身只能理解数字形式的地址,所以需要引入某种机制以便将 ASCII 字符串转换成网络地址。在本节中,我们将学习如何在 Internet 上完成这种映射。

我们首先回到早期的 ARPANET 中,在这里只有一个简单的 hosts.txt 文件,它列出了所有的主机和它们的 IP 地址。每天晚上,所有的主机都到一个维护此文件的站点上将它取回来。对于一个拥有几百台大型分时机器的网络而言,这种方法工作得非常好。

但是,当数千台小型机和 PC 机被连接到网络中以后,大家都意识到这种方法将不再继续奏效了。首先,这个文件将变得非常庞大。然而更重要的是,如果不采用集中模式管理机器名的话,则主机名冲突的现象将会频繁发生,而在一个巨大的国际性网络中,考虑到负载和延迟,要想实现这种集中管理将是难以想象的。为了解决这些问题,人们发明了 DNS(Domain Name System,域名系统)。

DNS 的本质是,它发明了一种分层次的、基于域的命名方案,并且用一个分布式数据库系统来实现此命名方案。DNS 的主要用途是,将主机名和电子邮件目标地址映射成 IP 地址,但它还有其他的一些用途。RFC1034 和 1035 定义了 DNS。

简要地说,DNS 的使用方法如下所述。为了将一个名字映射成 IP 地址,应用程序调用一个名为解析器(resolver)的库过程,并将该名字作为参数传递给此过程。在图 6.6 中我们看到的 gethostbyname 就是一个解析器例子。然后,解析器向本地的 DNS 服务器



发送一个 UDP 分组,之后,本地 DNS 服务器查找该名字,并且将找到的 IP 地址返回给解析器,解析器再将 IP 地址返回给调用方。有了 IP 地址以后,应用程序就可以与目标机器建立一个 TCP 连接,或者给它发送 UDP 分组。

### 7.1.1 DNS 名字空间

管理一个大型的、并且经常变化的名字集不是一件容易的事情。在邮政系统中,名字管理是这样来完成的:要求所有的信件必须(隐式地或显式地)指定收件人的国家、州或省、城市、街道地址。通过这种分层次的地址,纽约州 White Plains 市的 Main 大街上的 Marvin Anderson 与德克萨斯州奥斯汀市的 Main 大街上的 Marvin Anderson 之间就不会出现名字冲突了。DNS 采用了同样的工作方式。

从概念上来说,Internet 被分成 200 多个顶级域(domain),每个域包含许多主机。每个域又被分成若干个子域,子域又被进一步划分,以此类推。所有这些域可以用一棵树来表示,如图 7.1 所示。树的叶节点表示不包含子域的域(当然,它们仍包含主机)。一个叶节点域可以只包含一台主机;它也可以代表一个公司,从而包含几千台主机。

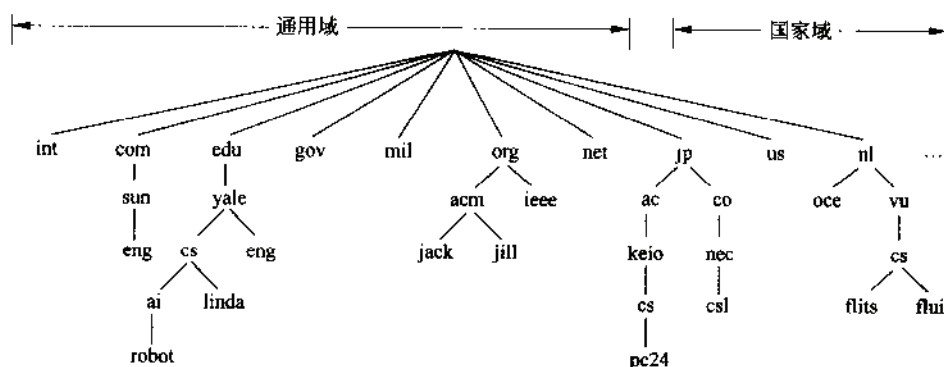


图 7.1 Internet 域名空间的一部分

顶级域有 2 种:通用域和国家域。最初的通用域是:com(商业的)、edu(教育性机构)、gov(美国联邦政府)、int(一些国际性组织)、mil(美国军队)、net(网络供应商)、org(非赢利性组织)。每个国家有一个国家域,其定义位于 ISO 3166 中。

2000 年 11 月,ICANN 批准了 4 个新的通用的顶级域,即: biz(商贸, business)、info(信息)、name(人们的名字)和 pro(职业,比如医生和律师)。另外,在一些特殊行业的要求下,ICANN 又引入了 3 个更为特殊的顶级域,它们是: aero(航空业)、coop(合作社)和 museum(博物馆)。将来还会增加其他的顶级域。

顺带提一下,随着 Internet 的日益商业化,它也越来越受争议。以 pro 为例,它的意图是专门针对经过资质认定的专业人员。但谁是专业人员?又由谁来认定呢?医生和律师显然是专业人员。但自由摄影师、钢琴教师、魔术师、水管工人、理发师、灭虫工人、文身画家、雇佣兵又如何呢?这些职业算是专业人员并且适合于 pro 域吗?如果是,那么由谁来认定每个从业者的资格呢?

一般来说,要获得一个像 name-of-company.com 这样的二级域名是非常简单的。申请者只需到对应的顶级域(在这里是 com)的登记员处,去查一查所请求的名字是不是可以使用,并且是否是其他人的商标。如果没有问题,则申请者支付少量的年费以后就可以得到此名字了。到目前为止,在 com 域中,几乎每一个常见的(英语)单词都已经被注册了。你可以试试常用的物品、动物、植物、身体各部位等单词,几乎所有的单词都已经被注册了。

每个域的名字是从它向上到(未命名的)根节点的路径,各个部分之间用句点(读作“dot”)分开。这样,Sun Microsystems 公司的工程部门可能是 eng.sun.com.,而不是像/com/sun/eng 这样的 UNIX 风格的名字。请注意,这种分层次的命名机制意味着 eng.sun.com. 中的 eng 不会与 eng.yale.edu. 中的 eng 发生冲突,而耶鲁大学的英语系可能会使用后者作为它的域名。

域名可以是绝对的,也可以是相对的。绝对域名总是以句点作为结束(例如 eng.sun.com.),而相对域名则不然,它必须在一定的上下文环境中被解释出来才有意义,从而惟一地确定其真实的含义。绝对域名和相对域名都引用了域名树中一个特定的节点,以及它下面的所有节点。

域名不区分大小写,因此,edu、Edu 和 EDU 的含义都一样。各组成部分的名字最多可以有 63 个字符长,整个路径的名字必须不超过 255 个字符。

原则上,可以用两种不同的方法将一个域名插入到域名树中。例如,cs.yale.edu 完全可以被列在 us 国家域的下面,从而变成 cs.yale.ct.us。然而,在实践中,美国的大多数组织都位于某一个通用域的下面,而美国之外的大多数组织则位于其国家域的下面。一个组织同时在两个顶级域下注册域名也是允许的,但除了跨国公司(例如 sony.com 和 sony.nl)以外,很少有组织这样做。

每个域自己控制如何分配它下面的域。例如,日本的 ac.jp 和 co.jp 域分别对应于 edu 和 com;但荷兰不作这样的区分,它把所有的组织直接放在 nl 下。因此,以下三个都是大学的计算机科学系:

- (1) cs.yale.edu (美国耶鲁大学,Yale University)
- (2) cs.vu.nl (荷兰阿姆斯特丹自由大学,Vrije University)
- (3) cs.keio.ac.jp (日本庆应义塾大学,Keio University)

为了创建一个新的域,创建者需要得到该新域的上级域的许可。例如,如果耶鲁大学建立了一个 VLSI(超大规模集成电路)组,并且希望将它称为 vlsi.cs.yale.edu,那么这个组必须获得 cs.yale.edu 的管理员的许可。同样地,如果创立了一所新的大学,比如说是 University of Northern South Dakota,那么它必须请求 edu 域的管理员将 unsd.edu 分配给它。按照这种方式就可以避免名字冲突,并且每个域都可以知道它所有的子域。一旦一个新的域已被创建并注册,则无需域名树中任何上层域的许可,它就可以创建子域,比如 cs.unsd.edu。

命名机制遵循的是组织的边界,而不是物理网络的边界。例如,即使计算机科学系和电子工程系在同一幢楼里,并使用同一个 LAN,但它们仍然可以有完全不同的域。同样地,即使计算机科学系分散在 Babbage Hall 和 Turing Hall 这两幢楼里,但这两幢楼里的

主机通常仍属于同一个域。

### 7.1.2 资源记录

无论是单主机域还是顶级域,每个域都可以有一组与它相关联的资源记录。对于一台主机来说,最常见的资源记录就是它的 IP 地址,但除此以外还存在许多其他种类的资源记录。当解析器把一个域名传递给 DNS 时,DNS 所返回的是与该域名相关联的资源记录。因此,DNS 的基本功能是将域名映射到资源记录上。

每条资源记录是一个五元组。尽管资源记录被编码成二进制的形式以保证效率,但是在大多数的叙述材料中,我们还是用 ASCII 文本来表示资源记录,每一条占一行。在接下来的介绍中,我们将使用如下的格式:

Domain\_name(域名) Time\_to\_live(生存期) Class(类别) Type(类型) Value(值)

Domain\_name(域名)指出了这条记录适用于哪个域。通常每个域有许多条记录,而数据库的每一份副本也保存了有关多个域的信息。因此 Domain\_name 是匹配查询条件的主要搜索关键字。数据库中资源记录的顺序是无关紧要的。

Time\_to\_live(生存期)域用于指示该记录的稳定程度。极为稳定的信息会被分配一个很大的值,比如 86400(1 天时间内的秒数);而非常不稳定的信息则会被分配一个较小的值,比如 60(1 分钟)。稍后当我们讨论到缓存机制时,我们将再回到这个问题上。

每条资源记录的第三个域是 Class(类别)。对于 Internet 信息,它总是 IN。对于非 Internet 信息,则可以使用其他的代码,但是在实践中很少见。

Type(类型)域指出了这是什么类型的记录。图 7.2 列出了最重要的一些类型。

类型	含义	值
SOA	授权的开始	本区域的参数
A	一台主机的 IP 地址	32 位整数
MX	邮件交换	优先级,希望接受该域电子邮件的机器
NS	名字服务器	本域的服务器的名称
CNAME	规范名	域名
PTR	指针	一个 IP 地址的别名
HINFO	主机的描述	用 ASCII 表示的 CPU 和操作系统
TXT	文本	未解释的 ASCII 文本

图 7.2 IPv4 中最主要的 DNS 资源记录类型

SOA 记录给出了有关该名字服务器区域(后面将会介绍)的主要信息来源的名称、名字服务器管理员的电子邮件地址、一个惟一的序列号,以及各种标志和超时值。

最重要的记录类型是 A(Address,地址)记录,它包含了某一台主机的 32 位 IP 地址。每台 Internet 主机必须至少有一个 IP 地址,以便其他机器能与它进行通信。某些主机有两个或多个网络连接,在这种情况下,每个网络连接(因此也就是每个 IP 地址)会有一条

A 类型的资源记录。我们也可以将 DNS 配置成在这些记录中循环,即为第一个请求返回第一条记录,为第二个请求返回第二条记录,等等。

其次最重要的记录类型是 MX 记录,它也指定一台主机的名字,该主机将为这个特定的域接受电子邮件。之所以使用 MX 记录,是因为并非每台机器都做好了接受电子邮件的准备。例如,如果有人打算发送电子邮件给某个人,比如 bill@microsoft.com,则发送方主机就需要在 microsoft.com 中找到一台愿意接受电子邮件的邮件服务器。MX 记录正可以提供这样的信息。

NS 记录指定名字服务器。例如,一般情况下,在每个 DNS 数据库中,针对每个顶级域都会有一条 NS 记录,这样一来,举例来说,电子邮件就可以被发送到域名树中远处的部分。稍后我们将回到这个问题上。

CNAME 记录允许创建别名。例如,如果一个人很熟悉 Internet 的常规命名规则,他打算给 MIT 计算机科学系的一个人发送一个消息,而且他只知道此人的登录名为 paul,那么,他可能会猜测此人的邮件地址是 paul@cs.mit.edu。事实上这个地址不正确,因为 MIT 计算机科学系的域是 lcs.mit.edu。但是,MIT 可以创建一条 CNAME 记录,以便为那些不知情的人和程序指引到正确的方向上,这也算是为他们提供一项服务吧。类似下面这样的一条记录就可以完成此任务:

```
cs.mit.edu 86400 IN CNAME lcs.mit.edu
```

如同 CNAME 一样,PTR 也指向另一个名字。但是 CNAME 只是一个宏定义,而 PTR 与 CNAME 不同,它是一种正规的 DNS 数据类型,它的确切含义要取决于它所在的上下文。在实践中,PTR 几乎总是被用来将一个名字与一个 IP 地址关联起来,以便能够查找 IP 地址并返回对应机器的名字,这种功能被称为反向查找(reverse lookups)。

HINFO 记录允许人们找到一个域对应于哪种机器和操作系统。最后,利用 TXT 记录,每个域可以按照任意的方式来标识自己。这两种记录类型都是为了用户方便使用,它们都不是必需的,因此任何一个程序不能期望总是能得到它们(即使得到了这两个记录的信息,程序可能还无法处理这些信息)。

最后,我们还有 Value 域,它的值可以是数字、域名或者 ASCII 字符串,其语义取决于记录的类型。在图 7.2 中给出了每种主要记录类型的 Value 域的简短描述。

图 7.3 显示了在一个域的 DNS 数据库中人们有可能找到的信息种类的例子。它描述了图 7.1 中 cs.vu.nl 域的(半假设的)数据库的一部分。该数据库包含七种类型的资源记录。

图 7.3 中第一个非注释行给出了该域的一些基本信息,我们以后将不再关心这些信息。接下来的两行给出了该域所在位置的文本信息。再下面的两行给出了两个投递电子邮件的地点,因此,发送给 person@cs.vu.nl 的电子邮件将被设法投递到这两个地点。首先应该尝试的是 zephyr(一台特定的机器),如果它失败了,下一个选择应该尝试 top。

接下来是一个空白行,其目的是为了增加可读性,再下面的几行指出了 flits 是一台

```

: Authoritative data for cs.vu.nl
cs.vu.nl.      86400  IN  SOA      star boss (9527,7200,7200,241920,86400)
cs.vu.nl.      86400  IN  TXT      "Divisie Wiskunde en Informatica."
cs.vu.nl.      86400  IN  TXT      "Vrije Universiteit Amsterdam "
cs.vu.nl.      86400  IN  MX       1 zephyr.cs.vu.nl.
cs.vu.nl.      86400  IN  MX       2 top.cs.vu.nl.

flits.cs.vu.nl. 86400  IN  HINFO    Sun Unix
flits.cs.vu.nl. 86400  IN  A         130.37.16.112
flits.cs.vu.nl. 86400  IN  A         192.31.231.165
flits.cs.vu.nl. 86400  IN  MX       1 flits.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX       2 zephyr.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX       3 top.cs.vu.nl.
www.cs.vu.nl.   86400  IN  CNAME    star.cs.vu.nl
ftp.cs.vu.nl.   86400  IN  CNAME    zephyr.cs.vu.nl

rowboat         IN  A         130.37.56.201
                IN  MX       1 rowboat
                IN  MX       2 zephyr
                IN  HINFO    Sun Unix

little-sister   IN  A         130.37.62.23
                IN  HINFO    Mac MacOS

laserjet        IN  A         192.31.231.216
                IN  HINFO    "HP Laserjet III Si" Proprietary

```

图 7.3 一个针对 cs.vu.nl 域的 DNS 数据库的可能部分

运行 UNIX 的 Sun 工作站,并给出了它的两个 IP 地址。然后又给出了用于处理发往 flits.cs.vu.nl 的电子邮件的三个选择,第一个选择自然是 flits 本身,但如果它停机了,则 zephyr 和 top 将是第二和第三选择。再下面是一个别名: www.cs.vu.nl,这样不用指定一台特定的机器就可以使用这个地址了。创建这个别名的好处是,使得 cs.vu.nl 在改变它的 WWW 服务器的同时,人们原来所使用的地址继续有效。类似地,ftp.cs.vu.nl 也是一个别名。

接下来的四行包含了一台工作站 rowboat.cs.vu.nl 的典型条目。它提供的信息包括 IP 地址、主邮件存放处和次邮件存放处,以及关于这台机器的信息。再接下来的条目针对一个自己不能接收邮件的非 UNIX 系统,最后的两个条目针对一台连接到 Internet 的激光打印机。

这里没有显示(并且也不在这个文件中)的是那些被用来查找顶级域的 IP 地址。为了查找远方的机器,这些 IP 地址是必需的,但由于它们不是 cs.vu.nl 域的一部分,所以它们不在这个文件中。这些 IP 地址是由根服务器提供的,而根服务器的 IP 地址则位于一个系统配置文件中;当 DNS 服务器启动时,该配置文件被加载到 DNS 缓存中。全球共分布了 10 多台根服务器,每一台都知道所有顶级域服务器的 IP 地址。因此,对于任何一台机器,它只要知道了至少一台根服务器的 IP 地址,那么,它就可以查找任何 DNS 名字了。

### 7.1.3 名字服务器

至少从理论上来说,只要一台名字服务器就可以包含整个 DNS 数据库,并且可以响应所有的 DNS 查询。但是在实践中,这台服务器会因为负载过重而变得毫无用处。而且,一旦它停机了,则整个 Internet 将会瘫痪。

为了避免由于单个信息源而带来的各种问题,DNS 名字空间被划分为一些不重叠的区域(zones)。针对图 7.1 中的名字空间,一种可能的划分方法如图 7.4 所示。每个区域包含域名树的一部分,同时也包含了存放该区域信息的名子服务器。一般情况下,一个区域有一台主名字服务器以及一台或多台次名字服务器。主名字服务器从自己硬盘上的一个文件中读取信息,次名字服务器则从主名字服务器中获得信息。为了提高可靠性,一个区域的某些服务器可以被放置在该区域之外。

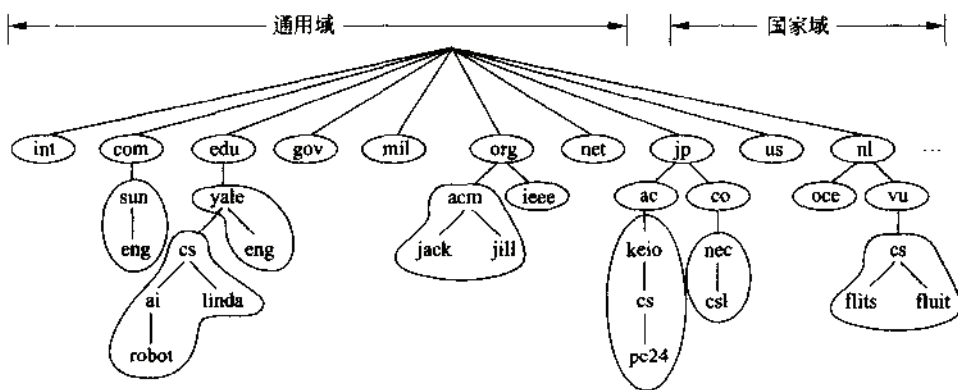


图 7.4 显示区域划分的部分 DNS 名字空间

区域边界应该放置在区域中的什么位置处,这是由该区域的管理员来决定的。这个决定在很大程度上取决于期望使用多少名字服务器以及将它们放在何处。例如,在图 7.4 中,耶鲁大学的 yale.edu 有一台服务器,它能处理 eng.yale.edu 但不能处理 cs.yale.edu,后者是一个独立的区域,它有自己的名字服务器。当有些系(比如英语系)不希望运行自己的名字服务器,而另外一些系(比如计算机科学系)却希望运行自己的名字服务器时,就有可能作出这样的决定。因此,cs.yale.edu 是一个独立的区域,而 eng.yale.edu 却不是。

当解析器接收到一个域名查询时,它将该查询传递给本地的一个名字服务器。如果被查询的域名落在该名字服务器的管辖范围内,例如 ai.cs.yale.edu 在 cs.yale.edu 的管辖范围内,那么该服务器返回权威的记录。权威记录(authoritative record)是指这样的记录:它来自于管理该记录的权威机构,因此总是正确的。权威记录与缓存的记录大不相同,后者可能是过期的。

然而,如果被请求的域是远程的,并且本地没有关于它的信息,那么本地名字服务器向顶级名字服务器发送一条查询此域的消息。为了让这个过程更加清晰,请考虑图 7.5



中的例子。在这里,机器 flits.cs.vu.nl 上的解析器想知道主机 linda.cs.yale.edu 的 IP 地址,在第 1 步中,它给本地名字服务器 cs.vu.nl 发送一条查询消息。该查询包含了被查找的域名、类型(A)和类别(IN)。

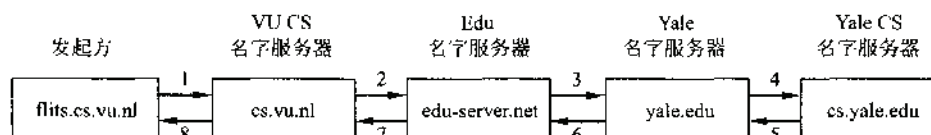


图 7.5 解析器如何通过 8 个步骤来查找一个远程名字

假设本地名字服务器以前从未收到过关于这个域的查询,并且对该域一无所知。那么它可以询问附近其他的一些机器,但如果它们都不知道,则它就给它的数据库中指定的 edu 服务器 edu-server.net(见图 7.5)发送一个 UDP 分组。这台服务器不太可能知道 linda.cs.yale.edu 的地址,可能也不知道 cs.yale.edu,但是它一定知道自己所有的子域,因此它将请求转发给 yale.edu 的名字服务器(第 3 步),后者再将请求转发给必定拥有权威资源记录的 cs.yale.edu(第 4 步)。由于每个请求都是从客户端传递到服务器端,因此,在第 5 步到第 8 步中,所请求的资源记录又被反向地传递回来。

一旦这些记录被取回到名字服务器 cs.vu.nl 中,则它们将进入到该服务器的缓存中,以备将来之需。但是,这些信息不是权威的,因为在 cs.yale.edu 上所做的修改不会被传播到全球所有可能知道该域信息的缓存中。正因为如此,缓存中的内容不应该被保存太久。这正是在每条资源记录中包含生存期(Time\_to\_live)域的原因。此信息告诉远程名字服务器,该记录可被缓存多久。如果某台机器使用同一个 IP 地址已经有几年时间了,那么将它的信息缓存 1 天可能是安全的。但对于不太稳定的信息,则在几秒钟或者一分钟后就清除掉相应的记录可能更加安全一些。

值得提及的是,这里描述的查询方法被称为递归查询(recursive query)。这是因为每台不包含被请求信息的服务器都转到别的地方去继续查找,然后它再往回报告结果。除此以外,还存在另一种可供选择的方法。在这种方法中,当一个查询在本地不能被满足时,查询失败,但是,尝试路线中下一台服务器的名字被返回来。有些服务器没有实现递归查询,它们总是返回下一台要尝试的服务器的名字。

还需要指出的是,如果一个 DNS 客户在它的定时器过期之前未能得到回应,则下一次它通常会尝试另一台服务器。这里的假设是,服务器可能是正常关机的,而并非是请求或应答在途中丢失了。

虽然 DNS 对于 Internet 的正常运行非常重要,但是它实际上所做的事情是将机器的符号名映射到它们的 IP 地址上。它不能帮助定位到人员、资源、服务或者一般的对象。为了定位这些对象,人们定义了另一个被称为 LDAP(Lightweight Directory Access Protocol,轻量级目录访问协议)的目录服务。它是 OSI X.500 目录服务的一个简化版本,RFC 2251 描述了 LDAP。它将信息组织成一棵树,允许按不同的部件进行搜索。我们可以把 LDAP 看作是“黄页”电话本。在本书中我们不再对此作进一步的讨论,有关更

多的信息请参见(Weltman and Dahbura, 2000)。

## 7.2 电子邮件

电子邮件,或者 e-mail,正如它的众多爱好者所知道的那样,已经存在 20 多年了。在 1990 年以前,它主要被用于学术界。在整个 20 世纪 90 年代,它一下子变得普及起来,并且呈指数地增长,以至于现在每天发送的电子邮件数量远远超过了传统邮件(snail mail, 纸质邮件)信函的数量。

与大多数其他的通信方式一样,电子邮件有它自己的约定和风格。特别是它非常不拘小节,而且用户的门槛很低。那些从来没有梦想过给某个大人物打一个电话或者只是写一封信的人,也可以毫不犹豫地给他发一封随意书写的电子邮件了。

电子邮件中充满了诸如 BTW(By The Way, 顺便)、ROTFL(Rolling On The Floor Laughing, 笑得满地打滚)和 IMHO(In My Humble Opinion, 恕我直言)等行话。很多人还在他们的电子邮件中使用一些被称为微笑图标(smiley)或情绪图标(emoticon)的小巧的 ASCII 符号。图 7.6 中复制了一些更为有趣的符号。对于大多数符号,把书顺时针旋转 90 度将使它们更加清楚。关于一本提供了 650 多个微笑图标的小册子,请参见 Sanderson and Dougherty, 1993。

微笑图标	含 义	微笑图标	含 义	微笑图标	含 义
: )	我很高兴	=  : -)	Abe Lincoln(林肯)	: +)	大鼻子
: -(	我很沮丧/愤怒	=) : -)	Uncle Sam(山姆大叔)	: -))	双下巴
: -	我没兴趣(表示冷漠)	* < : -)	Santa Claus(圣诞老人)	: -( )	小胡子
;-)	我在眨眼	< : -(	笨蛋	# : -)	乱蓬蓬的头发
: -( ( )	我在大叫	(- :	澳洲土著	8-)	戴眼镜
: ( *	我在呕吐	: -)X	带蝶形领结的男人	C : -)	大脑袋

图 7.6 一些微笑图标(它们不会出现在期末考试中 : ))

第一个电子邮件系统只是简单地由一些文件传输协议组成,同时也遵从约定: 每个消息(即文件)的第一行包含了收信人的地址。随着时间的推移,这种方法的局限性日益明显。以下是其中一些不足:

(1) 要想给一组人发送一个消息极为不方便。经理们常常需要用这种功能给他们所有的下属发送备忘录。

(2) 消息没有内部结构,这使得计算机处理起来非常困难。例如,如果一个被转发的消息包含在另一个消息中,则难以从收到的消息中提取出被转发的部分。

(3) 发信方(寄信人)永远不知道消息是否到达。

(4) 如果某个人计划因公外出几周,同时希望他的秘书处理所有进来的电子邮件,则这种工作方式很不容易安排。

(5) 用户界面与传输系统集成得不好,它要求用户先编辑一个文件,然后退出编辑器

并调用文件传输程序。

(6) 不可能创建和发送混合了文本、图片、传真和语音的消息。

随着经验的累积,人们提出了一些设计更为精巧的电子邮件系统。1982年,ARPANET的电子邮件提案被作为RFC 821(传输协议)和RFC 822(消息格式)发表。在此基础上作了较小修订的RFC 2821和RFC 2822已经成为Internet标准,但大家仍然把Internet电子邮件称为RFC 822。

1984年CCITT起草了它的X.400建议书。经过20年的竞争,基于RFC 822的电子邮件系统得到了广泛的使用,而那些基于X.400的系统却已经消失了。由一小撮计算机科学系的研究生发起并推动的系统,如何打败了由全球所有PTT(邮电部)、众多政府以及计算机业界主要厂商大力推动的官方国际标准,这令我们想起了圣经中David和Goliath的故事。

RFC 822成功的原因并不是因为它有多么好,而是由于X.400设计得如此糟糕和复杂,以至于没有人能很好地实现它。譬如一个组织需要做出一个选择,要么选择一个简单的但能运转的基于RFC 822的电子邮件系统,要么选择一个想象中非常完美但无法工作的基于X.400的电子邮件系统,在这种情况下大多数组织选择了前者。也许这其中隐藏了一定的教训。因此,我们对电子邮件的讨论将集中在Internet电子邮件系统上。

### 7.2.1 结构与服务

在这一小节中,我们将概括地介绍电子邮件系统的功能及其组织方式。电子邮件系统通常由两个子系统组成:用户代理(user agent)和消息传输代理(message transfer agent);用户代理让用户能阅读和发送电子邮件,而消息传输代理则将消息从源端传送到目标端。用户代理是本地程序,它们提供了命令行方式、菜单方式或图形化方式以便与电子邮件系统进行交互。消息传输代理一般是系统守护进程(daemon),也就是在后台运行的进程,它们的任务是在系统中传递电子邮件。

一般来说,电子邮件系统支持5项基本功能。我们现在来看一看这些功能。

**撰写(composition)**是指创建消息和回信的过程。虽然可以用任何文本编辑器来编辑消息体,但是系统本身能够帮助填写地址,并且将众多的头域附加到每个消息中。例如,在回复消息时,电子邮件系统可以从收到的电子邮件中提取出发信人的地址,并自动将它插入到回信中适当的位置上。

**传输(transfer)**指的是把消息从发信人处传递到收信人处。在大多数情况下,这要求与目标端或者某台中间机器建立一个连接,再输出消息,然后释放连接。电子邮件系统应该自动地完成传输任务,而不会打扰用户。

**报告(reporting)**必须告诉发信人该消息怎么样了。它被投递了吗?它被拒收了吗?它被丢失了吗?在许多应用中,对投递的确认非常重要,甚至可能还具有法律意义(“法官大人,我的电子邮件系统不是十分可靠,因此我猜测电子传票可能在某个地方弄丢了”)。

**显示(displaying)**收到的消息是必需的,这样人们才可以阅读他们的电子邮件。有时还需要进行转换或者调用一个特殊的阅读器,例如,当消息是一个PostScript文件或者数字语音的时候。有时候最好还能够作一些简单的转换和格式编排。

**处理(disposition)**是最后一步,它关心的是收信人在收到消息后如何处理消息,可能的处理方式包括:不阅读而直接丢弃消息、阅读后再丢弃、保存消息,等等。也可能是取出并重读已经保存的消息,转发消息,或者对消息做其他的处理。

除了这些基本的服务,有些电子邮件系统,特别是公司的内部邮件系统,还提供了各种各样的高级特性。我们现在来简要地介绍其中一些高级特性。当人们搬家或者离开一段时间的时候,他们可能希望他们的电子邮件能被转发到一个新的地方,因此系统应当能够自动地完成这项任务。

大多数系统允许用户创建**邮箱(mailbox)**以保存收到的邮件。因此需要有一些命令来创建和删除邮箱、查看邮箱内容、在邮箱中插入和删除消息等。

公司经理常常需要把一条消息发送给他们的每一个下属、客户或供应商。这就引出了**邮件列表(mailing list)**的想法,邮件列表是一个电子邮件地址的列表。当一个消息被发送给邮件列表时,相同的消息副本会被投递给列表中的每一个人。

其他的高级特性包括抄送、密件抄送、高优先级电子邮件、秘密(即加密的)电子邮件、当主收信人当前不在时可转送给候选的收信人,以及秘书能够阅读和回复其老板的电子邮件的能力。

在工业界,电子邮件现在被广泛用于公司的内部通信。它允许相距遥远的雇员们合作完成一些复杂的项目,甚至他们跨越了多个时区。在电子邮件的讨论中,由于消除了大多数与级别、年龄和性别相关的线索,因此,这有助于将讨论集中在想法上而不受到在公司中的地位的影响。通过使用电子邮件,一个暑期实习生的好想法可能比一个执行副總裁的平庸想法更有影响。

电子邮件系统中的一个关键思想是将**信封(envelope)**与它的内容区分开。信封封装了消息,它包含了传输消息所需要的所有信息,例如目标地址、优先级和安全等级,所有这些都有别于消息本身。消息传输代理利用信封来进行路由,这就好像邮局的做法一样。

信封内部的消息由两部分组成:**消息头(header)**和**消息体(body)**。消息头包含用户代理所需的控制信息。消息体则完全提供给收信人。图 7.7 中显示了信封和消息。

### 7.2.2 用户代理

正如我们在前面看到的那样,电子邮件系统包括两个基本部分:用户代理和消息传输代理。在这一小节中,我们将介绍用户代理。用户代理通常是一个程序(有时被称为邮件阅读器),它能够接受各种命令,用户通过这些命令可以撰写、接收和回复消息,也可以维护邮箱。有些用户代理具有菜单或者图标驱动的界面,用户通过鼠标可以方便地执行操作,而其他一些用户代理则从键盘接收 1 个字符的命令。从功能上来说,它们都是相同的。有些系统是菜单或图标驱动的,但也有键盘快捷键。

#### 发送电子邮件

为了发送一个电子邮件消息,用户必须提供消息、目标地址,可能还有其他一些参数。创建消息比较容易,用户可以使用独立的文本编辑器、字处理程序,或者可能是内置在用户代理中的特殊的文本编辑器。目标地址必须采用用户代理可以处理的格式。许多用户

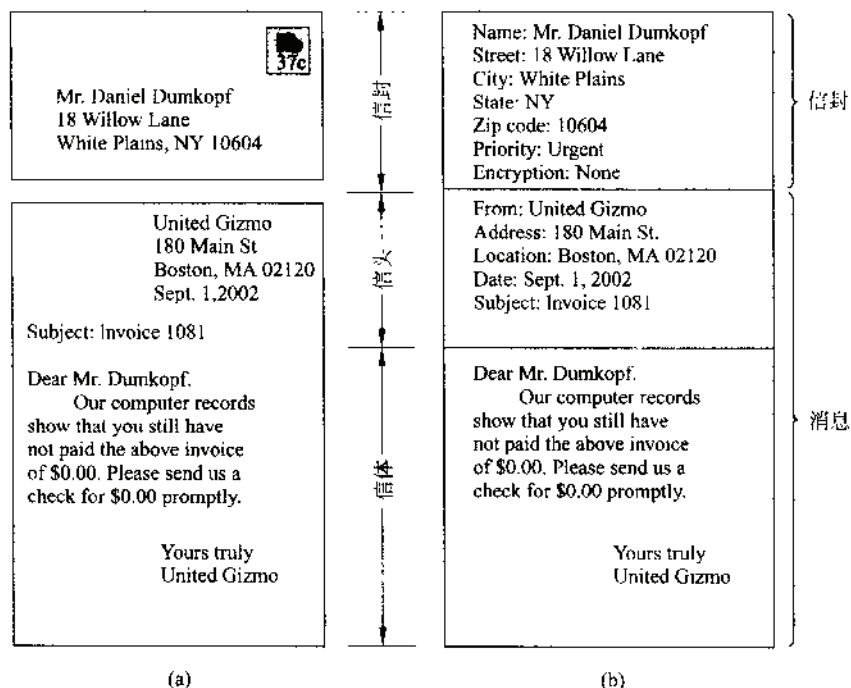


图 7.7 信封和消息

(a) 纸质邮件；(b) 电子邮件

代理要求地址的格式为 `user@dns-address`。在本章的前面我们已经学习过 DNS, 这里就不再重复这部分内容了。

然而, 需要说明的是, 还存在其他格式的地址。尤其是, X. 400 的地址与 DNS 地址看上去完全不同。X. 400 地址由一些“attribute=value”(“属性=值”)对组成, 它们相互之间用斜线分开, 例如:

```
/C = US/ST = MASSACHUSETTS/L = CAMBRIDGE/PA = 360 MEMORIAL DR./CN = KEN SMITH/
```

这个地址指定了国家、州、地区、个人地址和一个普通名(Ken Smith)。除此以外, 还可能有许多其他的属性, 因此, 如果你知道足够多的其他属性(例如, 公司名和职位名称), 你就可以给某个你不知道其确切邮件地址的人发送电子邮件。尽管 X. 400 的名字与 DNS 名字比较起来相当不方便, 但大多数电子邮件系统都有别名(有时候被称为昵称), 通过这些别名, 用户只需输入或者选择一个人的名字就可以得到正确的电子邮件地址。因此, 即使采用了 X. 400 地址, 一般也不必真正键入这些奇怪的字符串。

大多数电子邮件系统支持邮件列表, 因而用户只要用一个命令就可以把相同的消息发送给一群人。如果邮件列表是在本地维护的, 那么用户代理只是给每一个目标收信人发送一条独立的消息。然而, 如果邮件列表是在远程维护的, 那么消息将在远程被展开。例如, 如果一群鸟类观察家有一个名为 birders 的邮件列表, 该列表被安装在



meadowlark.arizona.edu 上,那么任何发送到 birders@meadowlark.arizona.edu 的消息都首先被路由到亚利桑那大学(University of Arizona),然后在那里被展开成发送给所有邮件列表成员的独立消息,而无论他们在世界上的哪个地方。这个邮件列表的使用者并不知道它是否是邮件列表,它也可能只是 Gabriel O. Birders 教授的个人邮箱。

### 阅读电子邮件

一般来说,当用户代理被启动的时候,在屏幕上显示任何东西之前,用户代理首先检查用户邮箱,并且将进来的电子邮件收取下来。然后它显示邮箱中邮件的数量,或者为每封邮件显示一行摘要,并等待用户命令。

作为用户代理工作方式的一个演示例子,我们来看一个典型的邮件场景。在启动了用户代理之后,用户请求获取关于他的电子邮件的一份摘要。于是,类似于图 7.8 所示的信息就会出现在屏幕上。每一行指向一条消息。在这个例子中,邮箱包含了 8 条消息。

#	标志	字节数	发送方	标题
1	K	1030	asw	Changes to MINIX
2	KA	6348	trudy	Not all Trudys are nasty
3	KF	4519	Amy N. Wong	Request for information
4		1236	bal	Bioinformatics
5		104110	baashoek	Material on peer-to-peer
6		1223	Frank	Re: Will you review a grant proposal
7		3110	guido	our paper has been accepted
8		1204	dmr	Re: My student's visit

图 7.8 显示一个邮箱的内容的例子

这里的每一行包含了从对应消息的信封或消息头中提取出来的几个域。在一个简单的电子邮件系统中,到底应该显示哪些域,这是内置在程序中的。而在复杂一点的系统中,用户可以通过提供一个**用户轮廓(user profile)**,来指定要显示哪些域,所谓用户轮廓是一个描述显示格式的配置文件。在这个简单的例子中,第一个域是消息号。第二个域是标志(Flags),它可以包含 K,表示该消息不是新的,而是以前阅读过并保存在邮箱中的消息;也可以是 A,表示该邮件已经被回复过;还可以/或者是 F,表示该消息已经被转发给其他人。其他的标志也是有可能的。

第三个域指出了该消息的长度,第四个域指出谁发送了该消息。由于这个域只是简单地从消息中提取出来的,因此它可能包含了发信人的名字、全名、姓名首字母、登录名,或者发信人选择放在那里的任何其他内容。最后,标题(Subject)域给出了消息内容的一个简短摘要。那些没有填写标题域的人,常常发现他们的邮件往往得不到最高的回复优先级。

当消息头被显示出来以后,用户可以执行诸如显示消息、删除消息等任何操作。较早的系统是基于文本的,一般使用单字符命令来完成这些任务,比如 T(type message,撰写



消息)、A(answer message, 回复消息)、D(delete message, 删除消息)以及 F(forward message, 转发消息), 同时用一个参数来指定待处理的消息。比较新一点的系统采用图形界面, 通常用户用鼠标来选择一条消息, 然后单击一个图标来撰写、回复、删除或者转发该消息。

电子邮件从只是一个文件传输器开始, 已经走过了很长的一段路。复杂的用户代理使得人们有可能管理大量的电子邮件。对于那些每年接收和发送成千上万条消息的人来说, 这样的工具的价值是无法估量的。

### 7.2.3 消息格式

现在, 让我们从用户界面转向电子邮件消息本身的格式。首先, 我们将介绍采用了 RFC 822 的基本 ASCII 电子邮件。之后, 我们再介绍 RFC 822 的多媒体扩展。

#### RFC 822

消息是由一个基本的信封(在 RFC 821 中描述)、一些头域、一个空行和消息体组成的。每个头域(逻辑上)由一行 ASCII 文本组成, 其中包括域名、一个冒号, 对于大多数头域来说还包括一个值。RFC 822 是几十年前被设计出来的, 它没有明确地区分信封域和消息头域, 虽然在 RFC 2822 中已经对 RFC 822 做了修订, 但由于 RFC 822 已经被广泛使用开了, 因此要想完全重新设计是不可能的。在一般的用法中, 用户代理创建一条消息并把它传递给消息传输代理, 然后消息传输代理利用某些头域来构造出实际的信封, 而这个信封则是有点老式的消息与信封的混合体。

图 7.9 中列出了与消息传输相关的主要头域。To: 域给出了主收信人的 DNS 地址, 有多个收信人也是允许的。Cc: 域给出了所有次收信人的地址。从投递的角度来说, 主收件人和次收件人没有区别, 这完全是心理上的差别, 这种差别也许对于相关的人而言很重要, 但是对邮件系统来说并不重要。Cc: (Carbon copy, 抄送)这个术语有点过时, 因为计算机不使用复写纸, 但是它已经被约定成俗了。Bcc: (Blind carbon copy, 密件抄送)域与 Cc: 域类似, 只是在发送给主收信人和次收信人的所有副本中, 这一行被删除了, 这个特性允许人们在主收信人和次收信人都不知道的情况下, 向第三方发送邮件副本。

头域	含义
To:	主收信人的电子邮件地址
Cc:	次收信人的电子邮件地址
Bcc:	密件抄送的电子邮件地址
From:	创建消息的人(或人们)
Sender:	实际发送者的电子邮件地址
Received	在传递路径上每个传输代理增加的行
Return-Path	可用来确定返回至发送者的路径

图 7.9 与消息传输相关的 RFC 822 头域

接下来的 From: 和 Sender: 两个域分别指出是谁写的消息以及是谁发送的消息, 这二者不必相同。例如, 一个商务经理可以撰写一条消息, 但她秘书可能是实际传送消息的人。在这里, 经理应该被列在 From: 域中, 而秘书应该被列在 Sender: 域中。From: 域是必需的, 但是, 如果 Sender: 域与 From: 域相同, 那么 Sender: 域可以被省略。一旦邮件无法投递并且必须被退回发送方时, 这些域就是必需的了。

在投递路径上, 每个消息传输代理都会给邮件加上一个包含 Received: 域的行, 这一行包含了该代理的标识符、接收到此消息的日期和时间, 以及其他一些可被用来查找路由系统中错误的信息。

Return-Path: 域由最后一个消息传输代理加入进来, 用来指出如何返回至发送者。从理论上来说, 从所有的 Received: 头 (除了发送者邮箱的名字以外) 中可以收集到 Return-Path: 域的信息, 但实际上很少用这种方式来填充该域, 它一般只包含发送者的地址。

除了图 7.9 中列出的域以外, RFC 822 消息还可以包含各种供用户代理或者收信人使用的头域。图 7.10 中列出了最常见的一些头域, 它们大部分都是不言自明的, 因此我们不对所有的头域都做详细的介绍。

头域	含义
Date:	发送消息的日期和时间
Reply-To:	回复消息应该被发送到这里的电子邮件地址
Message-Id:	以后引用这条消息时的一个惟一号
In-Reply-To	此回复消息所对应的消息的 Message-Id
References:	其他相关的 Message-Id
Keywords:	用户选择的关键词
Subject:	用于显示在一行上的简短消息摘要

图 7.10 RFC 822 消息头中使用的一些域

当邮件的撰写者或者邮件的发送者都不希望看到回复时, 有时可以使用 Reply-To: 域。例如, 市场经理撰写了一个电子邮件消息, 来向客户介绍一个新产品。秘书发送了这个消息, 但 Reply-To: 域列出的是销售部门的负责人, 他可以回答问题并处理订单。当发送者有两个电子邮件账号, 并希望消息被回复到另一个账号时, 这个域也会很有用。

RFC 822 文档明确地指出, 用户可以发明新的消息头以供自己私人使用, 只要这些消息头以字符串 X- 开头即可。RFC 822 保证将来的消息头不会使用以 X- 作为开头的名字, 以避免官方的消息头与私用的消息头之间发生冲突。有时, 一些聪明的大学生拼凑出像 X-Fruit-of-the-Day: 或 X-Disease-of-the-Week: 这样的域, 尽管它们不一定有说明性的含义, 但它们是合法的。

在消息头之后是消息体。用户可以在这里放他们想放的任何内容。有些人用精心设计的签名来结束他们的消息, 这些签名包括简单的 ASCII 卡通画、对某些权威著作的引

用、政治声明以及各种不承担责任的声明(例如,XYZ 公司不对我的观点负责;实际上,它甚至不理解我的观点)。

**MIME—多用途 Internet 邮件扩展**

在 ARPANET 的早期,电子邮件中的消息只能用英文来书写并且以 ASCII 码的形式来表示消息文本。对于这样的环境,RFC 822 完全能够胜任:它指定了消息头,但是把内容完全留给用户。现在,在全球范围的 Internet 上,这种方法就不再够用了。问题包括发送和接收两个方面:

- (1) 用带有重音符的语言(例如,法语和德语)来撰写的消息。
- (2) 用非拉丁字母(例如,希伯来语和俄语)来撰写的消息。
- (3) 用不带字母的语言(例如,汉语和日语)来撰写的消息。
- (4) 完全不包含文本的消息(例如,音频或图像)。

在 RFC 1341 中提出了一种解决方案,并且在 RFC 2045-2049 中对它做了修改。这种被称为 **MIME(Multipurpose Internet Mail Extensions,多用途 Internet 邮件扩展)** 的解决方案现在已被广泛使用。我们下面将对它作介绍,有关 MIME 的更多信息,请参见 RFC 文档。

MIME 的基本思想是继续使用 RFC 822 的格式,但在消息体中增加了结构,并且为非 ASCII 码的消息定义了编码规则。由于它没有偏离 RFC 822,所以人们可以用已有的邮件程序和协议来发送 MIME 消息。所有必须要改变的是发送和接收的程序,而这些可以由用户自己来完成。

MIME 定义了五种新的消息头,如图 7.11 所示。第一个消息头只是告诉接收消息的用户代理,它正在处理一条 MIME 消息,以及该消息使用的是哪个版本的 MIME。所有不包含 MIME-Version: 消息头的消息,都被假定认为是英语的明文消息,并按照这种特定的方式来进行处理。

头域	含义
MIME-Version:	标识了 MIME 版本
Content-Description:	供人阅读的字符串,描述了消息的内容
Content-Id	惟一标识符
Content Transfer-Encoding:	为了传输,如何包装消息体
Content-Type:	内容的类型和格式

图 7.11 MIME 增加的 RFC822 消息头

Content-Description: 消息头是一个 ASCII 字符串,它指出了消息中有些什么内容。这个消息头是必需的,这样收信人就知道是否值得解码和阅读该消息。如果这个字符串说的是“圣巴巴拉仓鼠的照片”,而收到该消息的人并不是一个狂热的仓鼠迷,那么这条消息就可能被丢弃,而不会被解码成一幅高清晰的彩色照片。

Content-Id: 消息头标识了消息的内容,它与标准的 Message-Id: 头使用相同的格式。

Content-Transfer-Encoding: 消息头指出了如何包装消息体,以便通过一个可能只接受字母、数字和标点符号的网络进行传输。编码方案有五种(加上一个扩充新方案的选项)。最简单的方案就是 ASCII 文本。每个 ASCII 字符占 7 位,如果每行都不超过 1000 个字符,那么电子邮件协议就可以直接接受。

其次最简单的方案与上一方案相同,但使用的是 8 位字符,也就是说,从 0 到 255 并且包括 255 在内的所有值。这种编码方案违反了(最初的)Internet 电子邮件协议,但是 Internet 上有些对最初协议进行了一些扩展的电子邮件实现却使用该方案。虽然声明这种编码方案并不能使其合法化,但显式地使用它至少可以在发生错误时能有所解释。使用 8 位编码的消息仍然必须遵循标准的最大行长度。

更糟糕的是那些使用二进制编码的消息。它们是任意的二进制文件,不仅使用全部的 8 位,甚至也不遵循每行 1000 个字符的限制。可执行程序就属于这一类。虽然无法保证二进制编码的消息总是能够正确地到达,但有的人仍然尝试这样做。

二进制消息的正确编码方式是使用 **base64 编码(base64 encoding)**,它有时也被称为 **ASCII 盔甲(ASCII armor)**。在这种方案中,每 24 位是一个组,每组被分成 4 个 6 位的单元,每个单元被当作一个合法的 ASCII 码字符来发送。在 base64 编码中,“A”代表 0,“B”代表 1,以此类推,接着是 26 个小写字母、10 个数字,最后是“+”和“/”分别代表 62 和 63。“==”和“=”分别表示最后一个组只含有 8 位或者 16 位。回车和换行被忽略,所以它们可被任意插入以保证每一行足够短。使用这种方案后,任意的二进制文本都可以被安全地发送出去。

对于那些几乎全是 ASCII 字符,但有少量非 ASCII 字符的消息,base64 编码的效率有些低。为了代替 base64 编码,它们可以使用一种称为可打印的引用编码(**quoted-printable encoding**)的编码。它也是 7 位的 ASCII 编码,但所有超过 127 的字符都被编码成一个等号后面跟着 2 个用 16 进制数字来表示的字符值。

总之,二进制数据应该以 base64(或者可打印的引用编码)的形式编码之后再发送。如果有正当的理由不采用这些方案,则可以在 Content-Transfer-Encoding: 头中指定一个用户自定义的编码方案。

图 7.11 中显示的最后一个消息头是最有趣的一个。它指定了消息体的本质特性。在 RFC 2045 中定义了 7 种类型,每一种都有一个或多个子类型。类型和子类型之间用斜线隔开,譬如:

```
Content Type: video/mpeg
```

在消息头中必须显式地给出子类型,没有默认值可供使用。图 7.12 给出了 RFC 2045 中指定的类型和子类型的初始列表。在那以后又加入了许多新的类型和子类型,同时,随着需求的增长,新增加的条目也被不断地加入进来。

现在我们来简单地浏览一下类型列表。text 类型用于直接的 ASCII 文本。text/plain 组合用于普通的消息,接收以后无需解码和进一步处理就可以被显示出来。通过该选项,普通的消息只需附加少些额外的消息头就能以 MIME 格式进行传输。

text/enriched 子类型允许在文本中包含一种简单的标记语言。该标记语言提供了

一种与系统无关的方法来表示粗体、斜体、较小和较大的磅尺寸、缩进、对齐、下标和上标，以及简单的页面布局。这种标记语言建立在 SGML (Standard Generalized Markup Language, 标准通用标记语言) 的基础上，其中 SGML 也被用作 WWW 的 HTML 的基础。例如，消息：

The `<bold>` time `</bold>` has come the `<italic>` walrus `</italic>` said ...

将被显示成：

The **time** has come the walrus said ...

类型	子类型	说明
Text	Plain	未格式化的文本
	Enriched	包含简单的格式化命令的文本
Image	Gif	GIF 格式的静止图像
	Jpeg	JPEG 格式的静止图像
Audio	Basic	可听见的声音
Video	Mpeg	MPEG 格式的电影
Application	Octet-stream	未解释的字节序列
	Postscript	PostScript 格式的可打印文档
Message	Rfc822	MIME RFC 822 消息
	Partial	为了传输而被分割的消息
	External-body	消息本身必须通过网络来获取
Multipart	Mixed	按照特定顺序的独立部分
	Alternative	同样的消息,但是不同的格式
	Parallel	必须同时查看的部分
	Digest	每一部分都是一个完整的 RFC 822 消息

图 7.12 RFC 2045 中定义的 MIME 类型和子类型

这种标记语言需要由接收系统来选择适当的解释办法。如果粗体和斜体是可用的，则直接使用即可；否则，可以使用颜色、闪烁、下划线、反相显示等加以强调。不同的系统可以有不同的选择。

当 Web 开始流行时，一个新的子类型 `text/html` 又被加入进来(在 RFC 2854 中)，从而可以在 RFC 822 的电子邮件中发送 Web 页面。RFC 3023 又为可扩展标记语言 (extensible markup language) 定义了一个子类型 `text/xml`。在本章的后面我们将学习 HTML 和 XML。

下一个 MIME 类型是 `image`，它被用于传输静止的图片。现在有许多种格式被广泛用于存储和传输图像，既可以是压缩的，也可以是未压缩的。其中两种格式 GIF 和 JPEG

几乎被内置在所有的浏览器中,但除此以外也还有很多其他的图像格式,它们也已经被加入到最初的列表中。

audio 和 video 类型分别用于声音和运动图像。请注意,video 类型只包含可视信息而不包含声道。如果要传输一部有声音的电影,那么根据所使用的编码系统,视频和音频部分可能必须要被分开来传输。第一个定义的视频格式是由运动图像专家组(MPEG, Moving Picture Experts Group,这个名字非常恰切)设计的格式,但此后又增加了其他一些格式。除了 audio/basic 以外,在 RFC 3003 中增加了一种新的音频类型: audio/mpeg,从而使人们可以通过电子邮件来发送 MP3 音频文件。

application 类型是对那些需要进行外部处理而且未被其他类型覆盖的格式的统称。octet-stream 只是一个未解释的字节序列。当用户代理接收到这样一个流时,它的处理方式可能是:建议用户将它复制到一个文件中,并提示用户输入文件名。然后,接下来的操作由用户自己来决定。

另一个定义的子类型是 postscript,它代表了由 Adobe 公司定义并被广泛用来描述打印页面的 PostScript 语言。许多打印机都有内置的 PostScript 解释器。尽管用户代理可以直接调用一个外部的 PostScript 解释器来显示接收到的 PostScript 文件,但这样做并非没有危险。PostScript 是一种全功能的编程语言。只要有足够的时间,一个十足的受虐狂可以用 PostScript 编写一个 C 语言编译器或者一个数据库管理系统。为了显示一个收到的 PostScript 消息,解释器将会执行包含在消息内部的 PostScript 程序,该程序除了显示一些文本以外,还可以读取、修改或删除用户的文件,而且还有其它一些严重的副作用。

message 类型使得一个消息可以被完全封装在另一个消息中。这种方案对于转发电子邮件很有用,例如,当一个完整的 RFC 822 消息被封装在另一个外部消息中时,则应该使用 rfc822 子类型。

partial 子类型使得有可能将一个被封装的消息分成多个部分并分别传输它们(例如,如果被封装的消息太长)。通过该子类型的一些参数,目标端能够将所有的部分按照正确的顺序重新组装起来。

最后,external-body 子类型可被用于非常长的消息(例如,视频电影)。它不是将 MPEG 文件包含在消息中,而是给出一个 FTP 地址,因此接收方的用户代理可以在需要的时候通过网络获取该文件。当向一个邮件列表中的人发送一部电影时,如果预期这些人当中只有一部分人想看该电影,那么这种功能非常有用(试想一下包含广告视频的电子邮件吧)。

最后一个类型是 multipart,它允许一个消息包含多个部分,并且每个部分的开始和结束被明确地分出界限。mixed 子类型允许每个部分都不相同,而且无需强加任何附加结构。许多电子邮件程序允许用户在一个文本消息中提供一个或多个附件,这些附件可以通过 multipart 类型来发送。

与 mixed 相反的是,alternative 子类型允许同一个消息被包含多次,但是被表示成两种或多种不同的媒体。例如,一个消息可以被按照三种形式来发送,分别是:直接的 ASCII 文本、复文本和 PostScript 格式。一个设计合理的用户代理在接收到这样的消息



时,如果有可能的话,则用 PostScript 来显示它。第二选择应该是复文本格式。如果这两者都不可能,则显示简单的 ASCII 文本。这些部分应该按从简单到复杂的顺序排列,以帮助那些使用非 MIME 用户代理的收信人也能在一定程度上理解消息的含义(例如,即使非 MIME 的用户也可以阅读简单的 ASCII 文本)。

alternative 子类型也可以被用于多语言并存的情形。从这种意义上来说,埃及的罗塞塔石碑(Rosetta Stone)可以被看成是一条早先的 multipart/alternative 消息。

图 7.13 中给出了一个多媒体的例子。在这里,生日问候同时被当作文本和歌曲进行传送。如果接收者有播放音频的功能,则他的用户代理就会获取到声音文件 birthday.snd,并将它播放出来。如果接收者不能播放音频,那么歌词就会悄无声息地显示在屏幕上。各个部分之间的界限由两个连字符来划分,连字符后面跟着一个(由软件生成的)字符串,字符串的值由 boundary 参数来指定。

```
From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched

Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
    access-type "anon-ftp";
    site="bicycle.abcd.com";
    directory="pub";
    name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

图 7.13 一个包含复文本(enriched)和音频(audio)两种可选子类型的 multipart 消息

请注意,在这个例子中,Content-Type 消息头出现在三个位置上。在顶层,它指出该消息有多个部分。在每个部分中,它给出了该部分的类型和子类型。最后,在第二个部分中,它必须要告诉用户代理应该获取什么类型的外部文件。为了显示这种用法的轻微不同之处,我们在这里使用了小写字母(请注意,所有的消息头是不区分大小写的)。同样地,对于不是按 7 位 ASCII 编码的外部消息体,则需要有 content transfer-encoding 头。

让我们回到多部分消息的子类型上,除了刚才介绍的子类型以外,还存在另外两种可能。当必须同时“查看”所有的部分时,应该使用 parallel 子类型。例如,电影通常有一个

音频通道和一个视频通道,如果这两个通道同时被播放出来,而不是顺次播放出来,则电影的播放效果会更好。

最后,当许多消息被打包成一个复合消息时,需要使用 digest 子类型。例如,Internet 上的一些讨论组从订阅者那里收集消息,然后把它们作为单个 multipart/digest 消息发送给整个组。

#### 7.2.4 消息传输

消息传输系统关注于将消息从发信方转发给收信方。为了做到这一点,最简单的方法是建立一个从源机器到目标机器间的传输连接,然后传输消息。在讨论了这个过程通常是如何完成的以后,我们将会介绍在某些情况下这种方法不能工作,以及这时候该怎么办。

##### SMTP 简单邮件传输协议

在 Internet 中,电子邮件是这样来传输的:源机器与目标机器的 25 号端口之间建立一个 TCP 连接。监听这个端口的程序是一个实现了 SMTP(Simple Mail Transfer Protocol,简单邮件传输协议)的电子邮件守护程序。这个守护程序接受进来的连接请求,并且将消息复制到正确的邮箱中。如果一个消息不能够被投递的话,则向消息的发送方返回一个包含该消息第一部分的错误报告。

SMTP 是一个简单的 ASCII 协议。在与目标机器的 25 号端口建立起 TCP 连接以后,发送方机器(作为客户)等待接收方机器(作为服务器)首先开始通话。服务器首先发送一行文本,在这行文本中它给出了自己的标识,并且告诉客户它是否已经准备好接收邮件。如果服务器还没有准备好,则客户将释放连接,以后再重试。

如果服务器愿意接受电子邮件,则客户声明这封电子邮件来自于谁以及将要交给谁。如果目标端确实存在这样的收件人,则服务器指示客户发送此消息。然后客户发送消息,服务器确认消息。因为 TCP 提供了可靠的字节流传输,所以这里不需要校验和。如果还有更多的电子邮件,则现在可以继续发送。当两个方向上所有的电子邮件都交换完成以后,连接被释放。为了发送图 7.13 中的消息,一个简单的会话过程如图 7.14 所示,图中也包含了 SMTP 所使用的数字代码。客户端发送的行用 C:来标识,而服务器发送的行则用 S:来标识。

对图 7.14 做一些说明也许会有所帮助。来自客户的第一条命令确实是 HELO。在 HELLO 的各种四字符缩写中,这种缩写相比于其竞争者而言有诸多优点。随着时间的流逝,为什么所有的命令必须是四个字符的原因现在已经不得而知了。

在图 7.14 中,消息只被发送给一个收件人,因此这里只使用一条 RCPT 命令。利用 RCPT 命令可以将一条消息发送给多个接收者。每条消息被单独确认或拒绝。即使有些收件人被拒绝(由于目标端并不存在这样的收件人),该消息也可以被发送给其他的收件人。

最后,尽管客户的四字符命令的语法被严格地限定,但是回复的语法就不是那么严格了。实际上,只有数字代码才真正很重要。每一种实现都可以在数字代码后面加上它所

```

S: 220 xyz.com SMTP service ready
C: HELO abcd.com
S: 250 xyz.com says hello to abcd.com
C: MAIL FROM: <elinor@abcd.com>
S: 250 sender ok
C: RCPT TO: <carolyn@xyz.com>
S: 250 recipient ok
C: DATA
S: 354 Send mail; end with "." on a line by itself
C: From: elinor@abcd.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abcd.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have a nice day.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/enriched
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:     access-type="anon-ftp";
C:     site="bicycle.abcd.com";
C:     directory="pub";
C:     name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
S: 250 message accepted
C: QUIT
S: 221 xyz.com closing connection

```

图 7.14 从 elinor@abcd.com 传输一条消息到 carolyn@xyz.com

喜欢的任何字符串。

为了更好地理解 SMTP 协议和本章中描述的其他协议是如何工作的,最好试一试这些协议。在所有的情况下,首先来到一台已接入 Internet 的计算机前面。在 Unix 系统上,在一个 shell 中键入:

```
telnet mail.isp.com 25
```

请用你的 ISP 的邮件服务器的 DNS 名字来替换 mail.isp.com。在 Window 系统中,首先单击“开始(Start)”,然后单击“运行(Run)”,并且在对话框中键入上述命令。这条命令将建立一个 telnet(也就是 TCP)连接,连接到那台机器的 25 号端口上。25 号端口是

SMTP 端口(图 6.27 列出了部分常用的端口)。你可能会得到像这样的回应:

```
Trying 192.30.200.66...
Connected to mail.isp.com
Escape character is '^['.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2002 13:26 +0200
```

前三行来自于 telnet 程序,告诉你它在做什么。最后一行来自于远程机器上的 SMTP 服务器,表明它愿意与你通话,也愿意接受电子邮件。为了找出该 SMTP 服务器可以接受哪些命令,请键入:

```
HELP
```

从现在开始,像图 7.14 中所示的、以客户端的 HELO 命令开始的命令序列就有可能出现。

值得指出的是,使用 ASCII 文本行作为命令并不是一个意外。大多数 Internet 协议都是以这种方式来工作的。采用 ASCII 文本使得协议更加易于测试和调试。正如我们在上面所看到的那样,通过手工发送命令的方式可以测试这些协议,而且转储出来的消息也非常易于阅读。

尽管 SMTP 协议已经被很好地定义了,但还是会出现一些问题。一个问题与消息的长度有关。一些较老的实现不能够处理超过 64KB 的消息。另外一个问题与超时有关。如果客户和服务器的超时间隔不同,则有可能它们中的一个已经放弃了而另外一个仍然在工作,从而意外地终止连接。最后,在很少的情况下,可能会触发巨大的邮件风暴。例如,如果主机 1 有一个邮件列表 A 而主机 2 有一个邮件列表 B,每个邮件列表都将对方当作一个成员包含进去,那么,发送给任意一个邮件列表的消息都会产生无限数量的电子邮件流量,除非有人检查过这种情况。

为了避免这些问题,RFC2821 中定义了扩展的 SMTP(ESMTP,extended SMTP)。想要使用 ESMTP 的客户在开始时必须发送 EHLO 消息而不是 HELO。如果此消息被拒绝,则说明该服务器是一个常规的 SMTP 服务器,客户应该按通常的方式来处理。如果 EHLO 消息被接受,则允许使用新的命令和参数。

### 7.2.5 最后的投递

到现在为止,我们一直假定所有的用户都在能够发送和接收电子邮件的机器上工作。正如我们所看到的,电子邮件的投递方式是,首先在发送方和接收方之间建立起 TCP 连接,然后在此连接上传递电子邮件。由于 APRANET(以及后来的 Internet)的所有主机事实上总是保持在线状态,它们随时可以接受 TCP 连接,所以,这种模型很好地工作了几十年。

然而,随着越来越多的人利用调制解调器通过 ISP 来访问 Internet,这种局面被打破了。问题在于:当 Elinor 想给 Carolyn 发送电子邮件而 Carolyn 不在线时,那会怎么样呢? Elinor 不能够与 Carolyn 建立 TCP 连接,因此不能够运行 SMTP 协议。

一种解决方案是在 ISP 的某一台机器上运行一个消息传输代理,由它为客户接收电

子邮件并将邮件存储在 ISP 的机器上的客户邮箱中。由于这个代理可以随时保持在线，所以一天 24 小时给它发送电子邮件都没有问题。

### POP3

不幸的是，这种解决方案引发了另一个问题：用户如何从 ISP 的消息传输代理那里得到电子邮件呢？这个问题的解决方案是建立另一个协议，用户传输代理（在客户的 PC 上）通过这个协议与消息传输代理（在 ISP 的机器上）联系，并且将电子邮件从 ISP 复制到用户机器上。其中一个这样的协议是 POP3（Post Office Protocol Version 3，邮局协议第 3 版），RFC1939 中描述了此协议。

上一小节介绍的情形（发送方和接收方都有永久的 Internet 连接）如图 7.15(a) 中所示。图 7.15(b) 显示了当发送方（当时）在线而接收方不在线时的情形。

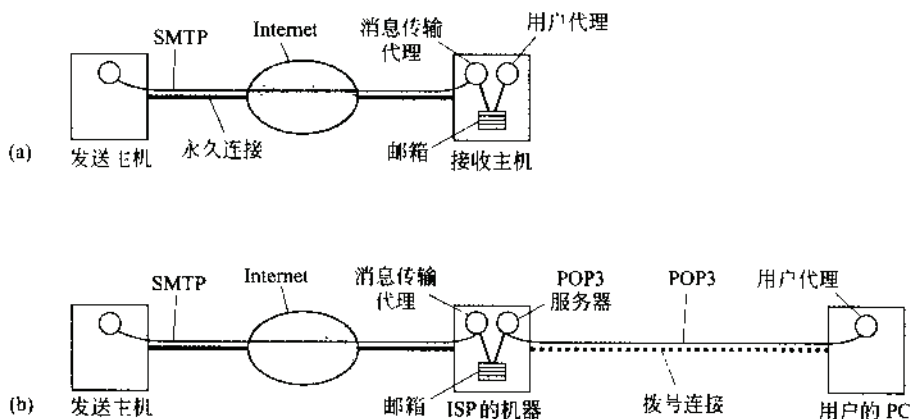


图 7.15

(a) 当接收方有永久的 Internet 连接，并且用户代理和传输代理运行在同一台机器上时发送和读取电子邮件的情形；(b) 当接收方通过拨号连接到 ISP 时读取电子邮件的情形

当用户启动邮件阅读器时 POP3 开始工作。邮件阅读器呼叫 ISP（除非已经有有了一个连接）并且在 110 端口上与消息传输代理建立一个 TCP 连接。一旦连接已被建立，则 POP3 协议按照顺序经历以下三种状态：

- (1) 授权
- (2) 事务
- (3) 更新

授权状态处理的是用户登录过程。事务状态处理的是用户收取电子邮件并将它们标记为从邮箱中删除。在更新状态中才真正地将电子邮件删除。

通过输入以下类似的命令，你可以观察到这些行为：

```
telnet mail.isp.com 110
```

这里 mail.isp.com 代表了你的 ISP 的邮件服务器的 DNS 名字。Telnet 建立一个到

POP3 服务器监听的 110 端口的 TCP 连接。一旦服务器接受了 TCP 连接,它就发送一条 ASCII 消息,表明它已就绪。通常这条消息以 +OK 开始,后面跟着一些注释。图 7.16 举例说明了 TCP 连接建立以后发生的情形。如同前面一样,标记为 C:的行来自于客户(用户),而标记为 S:的行来自于服务器(ISP 的机器上的消息传输代理)。

```

S: +OK POP3 server ready
C: USER carolyn
S: +OK
C: PASS vegetables
S: +OK login successful
C: LIST
S: 1 2505
S: 2 14302
S: 3 8122
S: .
C: RETR 1
S: (sends message 1)
C: DELE 1
C: RETR 2
S: (sends message 2)
C: DELE 2
C: RETR 3
S: (sends message 3)
C: DELE 3
C: QUIT
S: +OK POP3 server disconnecting
```

图 7.16 使用 POP3 协议获取三个消息

在授权状态过程中,客户首先发送它的用户名然后发送口令。在成功登录以后,客户可以发送 LIST 命令,这条命令让服务器列出邮箱中的内容,每行一个消息,并给出消息的长度。此列表以句号作为结束。

然后客户可以通过 RETR 命令来获取消息,并利用 DELE 命令把它们标记为删除。当获取了所有的消息(可能也将它们标记为删除)以后,客户发出 QUIT 命令以终止事务状态,并进入更新状态。当服务器删除了所有的消息以后,它发送一个应答并中断 TCP 连接。

尽管 POP3 协议具有下载一个特定的消息或一组消息并把它们保留在服务器上的能力,但是,大多数电子邮件程序只是下载所有的消息并清空邮箱。这种行为意味着,在实践中惟一的副本被保存在用户的硬盘上。如果硬盘毁坏了,则所有的电子邮件可能会永久地丢失。

我们现在来简单地总结一下电子邮件系统是如何为 ISP 的客户工作的。Elinor 使用某个电子邮件程序(即用户代理)来创建一个给 Carolyn 的消息,然后单击一个图标以便将它发送出去。电子邮件程序把消息交给 Elinor 主机上的消息传输代理。消息传输代理看到该消息是发送给 carolyn@xyz.com 的,所以它利用 DNS 来查找 xyz.com 的 MX 记录(这里 xyz.com 是 Carolyn 的 ISP)。此查询的结果返回了 xyz.com 的邮件服务器的 DNS 名字。现在,消息传输代理再次利用 DNS 来查找这台机器的 IP 地址,例如,通过



gethostbyname。然后它与这台机器的 25 号端口上的 SMTP 服务器建立一个 TCP 连接。通过一个类似于图 7.14 中的 SMTP 命令序列,它把消息传输到 Carolyn 的邮箱中,并中断 TCP 连接。

在适当的时候,Carolyn 打开她的 PC,连接到她的 ISP 上,并启动她的电子邮件程序。电子邮件程序与 ISP 邮件服务器的 110 端口上的 POP3 服务器建立一个 TCP 连接。这台机器的 DNS 名字或者 IP 地址通常是在安装电子邮件程序时或在订购 ISP 的服务时被配置的。当 TCP 连接被建立起来以后,Carolyn 的电子邮件程序运行 POP3 协议,使用类似于图 7.16 中的命令将邮箱的内容获取到她的硬盘上。一旦所有的电子邮件都已经被传回来了,则 TCP 连接被释放。实际上,与 ISP 的连接此时也可以中断了,因为所有的电子邮件都已经在 Carolyn 的硬盘上了。当然,若要发送回复的话,则还需要与 ISP 之间的连接,所以通常在获取电子邮件后并不马上中断连接。

## IMAP

如果一个用户在 ISP 上有一个电子邮件账户,并经常通过同一台 PC 来访问该账户,则对他来说,POP3 工作得非常好,而且由于 POP3 的简单性和健壮性,它已得到了广泛的使用。然而,计算机工业界的一个事实是,一旦当某一样事物工作得很好时,有人就会要求更多的特性(也会得到更多的缺陷)。这种情况也发生在电子邮件上。例如,许多人在公司或学校有一个电子邮件账户,他们希望在各个地方都可以访问该账户,比如在办公室的机器上、在家里的 PC 上,或者出差时可通过便携机来访问该账户,休假时在电脑咖啡馆也可以访问该账户。虽然 POP3 协议允许这么做,但是,由于在每次交互时它通常要下载所有的消息,因此,其结果是,用户的电子邮件很快分布到许多机器上,数量或多或少,甚至一部分不是用户的机器。

这个缺点导致了另一个最后投递协议的诞生:IMAP(Internet Message Access Protocol,Internet 消息访问协议),它被定义在 RFC 2060 中。我们在前面看到,POP3 基本上假设用户在每次交互后将会清除邮箱然后脱机工作,而与此不同的是,IMAP 假设所有的电子邮件都会永久地保存在服务器上的多个邮箱中。IMAP 提供了阅读消息或者阅读部分消息的扩展机制,当使用一个很慢的调制解调器来读取一个包含很大音频和视频附件的多部分(multipart)消息的文本部分时,这是一个很有用的特性。由于在 IMAP 的工作模型中,假设消息不会被传输到用户的计算机上作永久的保存,所以,IMAP 提供了创建、销毁和维护服务器上多个邮箱的机制。通过这种方式,一个用户可以为每个联系人维护一个邮箱,在阅读了消息之后将消息从收件箱移到该联系人的邮箱中。

IMAP 有许多特性,比如不需要像图 7.8 中所示的那样通过到达编号来定位邮件,而可以利用属性(例如,将 Bobbie 寄送过来的第一条消息给我)来定位邮件。与 POP3 不同,IMAP 不仅可以投递进来的电子邮件,也可以接受往外发送邮件至目的地的任务。

IMAP 协议的一般风格与图 7.16 所示的 POP3 协议的风格相似,不过它有更多的命令。IMAP 服务器监听 143 端口。图 7.17 给出了 POP3 和 IMAP 的一个比较。然而,需要指出的是,并不是每个 ISP 同时支持这两种协议,也并不是每个电子邮件程序都支持这两种协议。因此,在选择电子邮件程序时,搞清楚它支持哪些协议是非常重要的,并且也

要确定 ISP 至少支持其中一种。

特性	POP3	IMAP
协议定义的位置	RFC 1939	RFC 2060
所用的 TCP 端口	110	143
电子邮件存放的位置	用户 PC 上	服务器上
阅读电子邮件的位置	脱机	联机
连接时间的要求	少	多
服务器资源的使用	最小的	大量的
多个邮箱	否	是
准备份邮箱	用户	ISP
方便移动用户	否	是
用户对下载的控制	弱	强
是否可以部分下载消息	否	是
磁盘限额是否成问题	否	有时候可能是问题
便于实现	是	否
广泛支持	是	正在增长

图 7.17 POP3 和 IMAP 的比较

### 投递特性

不论使用 POP3 或者 IMAP,许多系统提供了钩子(hook)的功能以便对进来的邮件进行额外的处理。对于许多电子邮件用户来说,一个特别有价值的特性是设置过滤器(filter)的能力。所谓过滤器,实际上是一些当电子邮件进来时或用户代理启动时被检查的规则而已。每条规则指明了一个条件和一个动作。例如,可以指定这样的规则:来自于老板的任何消息进入 1 号邮箱,来自于一组指定的朋友的消息进入 2 号邮箱,在标题行中包含某些令人讨厌的特殊单词的所有消息都被直接丢弃。

有些 ISP 提供了一个过滤器来自动地对进来的电子邮件进行分类,分成重要的或广告(垃圾邮件)邮件,并把每个消息存放到相应的邮箱中。这种过滤器的典型工作方式是:首先检查邮件源是不是一个已知的广告发送者;然后往往会检查标题行。如果数百个用户接收到一个具有相同标题行的消息,那么它可能是广告。其他一些技术也被用于广告邮件的检测。

另外一个常见的投递特性是将进来的电子邮件(临时性地)转发给另一个不同的地址。这个地址甚至可能是一台由某个商业寻呼服务机构所操纵的计算机,然后它通过无线电或者卫星寻呼该用户,并且在用户的寻呼机上显示 Subject:行。

关于最后投递的另一个常见特性是安装一个假期守护程序(vacation daemon)的能力。它是这样一个程序:对每个进来的消息进行检查,并且给发送者发送一条类似如下

的简单回复：

Hi. I'm on vacation. I'll be back on the 24th of August. Have a nice summer.  
(你好,我在休假。我将于8月24号回来。祝你夏日快乐。)

这种回复也可以指明在此期间如何处理紧急事务,或者针对特定的问题应该找谁联系,等等。大多数假期守护程序会记录下它们给哪些人已经发送过预制的消息,以免向同一个人再次发送相同的回复。好的守护程序还会检查进来的消息是否是发送给一个邮件列表的,如果是,就不会发送预制的消息。(在夏天,给大的邮件列表发送消息的人可能不希望得到数百封详述每个人假期计划的回复邮件)。

当作者给一个声称每天收到600条消息的人发送电子邮件时,作者曾经碰到过一种极端的投递处理的形式。他的身份不便在这里公开,以免本书的部分读者也给他发送电子邮件。让我们称呼他为John吧。

John安装了一个电子邮件机器人,由它检查每封进来的电子邮件,看是否来自于一个新的通信者。如果是,它送回一条预制的消息,解释说John不能够亲自读取他所有的电子邮件。相反,他建立了一个个人的FAQ(常见问题列表)文档来回答很多经常被问到的问题。通常,新闻组有FAQ,而个人一般都没有。

John的FAQ给出了他的地址、传真和电话号码,并且告知如何与他的公司联系。FAQ也解释了如何邀请他演讲,描述了在哪里可以得到他的论文和其他文档。FAQ还提供了一些链接来指向他写的软件、他运作的一个会议,以及他作为编辑之一的一个标准,等等。这种方法可能是必要的,但也许个人FAQ是一种极端情形的象征。

### Webmail

最后一个值得提及的话题是Webmail。有些Web站点,例如Hotmail和Yahoo,给任何有需求的人提供了电子邮件服务。它们的工作方式如下所述。它们有正常的消息传输代理来监听25号端口以接受进来的SMTP连接。比如说你要联系Hotmail,那么你必须获得它的DNS MX记录,例如,在一个Unix系统上键入以下命令:

```
host-a-v hotmail.com
```

假设邮件服务器是mx10.hotmail.com,那么再键入:

```
telnet mx10.hotmail.com 25
```

于是,你建立了一个TCP连接,通过此连接,你可以用正常的方式来发送SMTP命令。到现在为止,一切都很正常,只不过这些服务器通常非常繁忙,所以你可能需要尝试很多次才能建立一个TCP连接。

这里有趣的部分是,电子邮件是如何被投递给用户的。基本上,当用户进入到一个电子邮件Web页面中时,一个请求输入用户登录名和口令的表单就会出现。当用户单击了登录(Sign In)按钮时,登录名和口令就被发送到服务器,然后服务器对此进行验证。如果登录成功,则服务器找到用户的邮箱,并且建立一个类似于图7.8中的列表,只不过它被格式化成HTML语言的Web页面。然后,这个Web页面被送回给浏览器以便显示出

来。该页面上的很多项目都是可以单击的,所以,你可以阅读消息、删除消息,等等。

## 7.3 万 维 网

万维网(World Wide Web)是一个结构性的框架,其目的是访问遍布在整个 Internet 上数百万台机器中的相互链接的文档。在过去的 10 年中,它从一种分发高能物理数据的方法,演变成数百万人所认为的“Internet”应用。它的迅速普及和流行来源于这样的事实:它有易于为初学者所使用的丰富多彩的图形界面,以及它提供了巨大的信息财富,覆盖了从 aardvarks(土豚)到 Zulus(祖鲁族人)几乎每一个可以想到的主题。

1989 年 Web(也被称为 WWW)诞生于欧洲原子能研究中心 CERN。CERN 有几台加速器,来自各个欧洲参与国的大型的科学家研究组利用这些加速器进行粒子物理研究。这些研究组的成员常常来自于 6 个或者更多的国家。大多数试验非常复杂,需要几年时间的预先计划和设备建造。Web 源于这样一个需求:让这些分散在各个国家的大型研究组通过使用经常变化的报告、计划、绘图、照片和其他文档来进行合作。

1989 年 3 月,CERN 的物理学家 Tim Berners-Lee 提出了最初的链接文档网的建议。18 个月后,第一个(基于文本的)原型投入运行。1991 年 12 月,在德克萨斯州的圣安东尼奥(San Antonio, Texas)举行的 Hypertext '91 会议上,进行了一次公开演示。

这次演示和随后的宣传引起了其他研究人员的注意,这使得伊利诺伊大学(University of Illinois)的 Marc Andreessen 开始开发第一个图形化的浏览器 Mosaic,并且于 1993 年 2 月发布。Mosaic 是如此地受欢迎,以至于一年后 Andreessen 离开学校创办了一家公司: Netscape 公司(Netscape Communications Corp.),公司的目标是开发客户端、服务器以及其他 Web 软件。当 1995 年 Netscape 公开发售股票时,投资者很显然地认为这是下一个微软,他们花了 15 亿美元来购买股票。这项记录极其令人吃惊,因为这家公司只有一个产品,而且正在负债累累地经营,并且它在招股说明书中宣称它不期望在可预见的将来就能盈利。在接下来的 3 年时间里,Netscape Navigator 与 Microsoft 的 Internet Explorer 卷入了一场“浏览器大战”,每一方都疯狂地增加比对手更多的功能(也因此而增加了更多的错误)。1998 年美国在线(America Online)以 42 亿美元收购了 Netscape 公司,从而结束了 Netscape 作为独立公司的短暂生涯。

1994 年,CERN 和 MIT 签署了建立万维网联盟(World Wide Web Consortium)(有时缩写为 W3C)的协议,该组织致力于进一步开发 Web、对协议进行标准化,并鼓励站点之间的互操作性。Berners Lee 担任联盟的主管。从那时起,已经有几百所大学和公司加入了联盟。尽管现在关于 Web 的书籍多得数不清,但获取关于 Web 最新信息的最佳之处(很自然地)还是在 Web 本身上。W3C 联盟的主页是 [www.w3.org](http://www.w3.org),感兴趣的读者可以从那里找到涵盖该联盟所有文档和活动的页面链接。

### 7.3.1 结构概述

从用户的观点来看,Web 是由一个巨大的全球范围的文档或 Web 页面(Web page)集合组成的,Web 页面通常也被简称为页面。每个页面可以包含指向全球任何其他

页面的链接(link)。通过单击一个链接,用户可以跟随这个链接,来到它所指向的页面。这个过程可以无限地重复下去。让一个页面指向另一个页面的想法现在被称为超文本(hypertext),这种想法是早在发明 Internet 之前的 1945 年由一个梦想家,MIT 的电子工程系教授 Vannevar Bush 发明的。

用一个被称为浏览器(browser)的程序可以浏览页面,Internet Explorer 和 Netscape Navigator 是最流行的两种浏览器。浏览器取回所请求的页面,对它上面的文本和格式命令进行解释,并在屏幕上按正确的格式显示出来。图 7.18(a)给出了一个例子。与许多 Web 页面一样,该页面也从标题开始,中间包含一些信息,最后以页面维护者的电子邮件地址作为结束。链接到其他页面的文本字符串被称为超链接(hyperlink),浏览器一般通过增加下划线、用特殊颜色显示,或者综合这两种方法来突出显示它们。为了跟随一个链接,用户将鼠标光标放在突出显示的区域上,这会使光标发生变化,然后用户单击链接。尽管存在像 lynx 这样的非图形化的浏览器,但它们远没有图形化的浏览器那么流行,所以我们将把注意力放在后者上。基于语音的浏览器也正在开发之中。

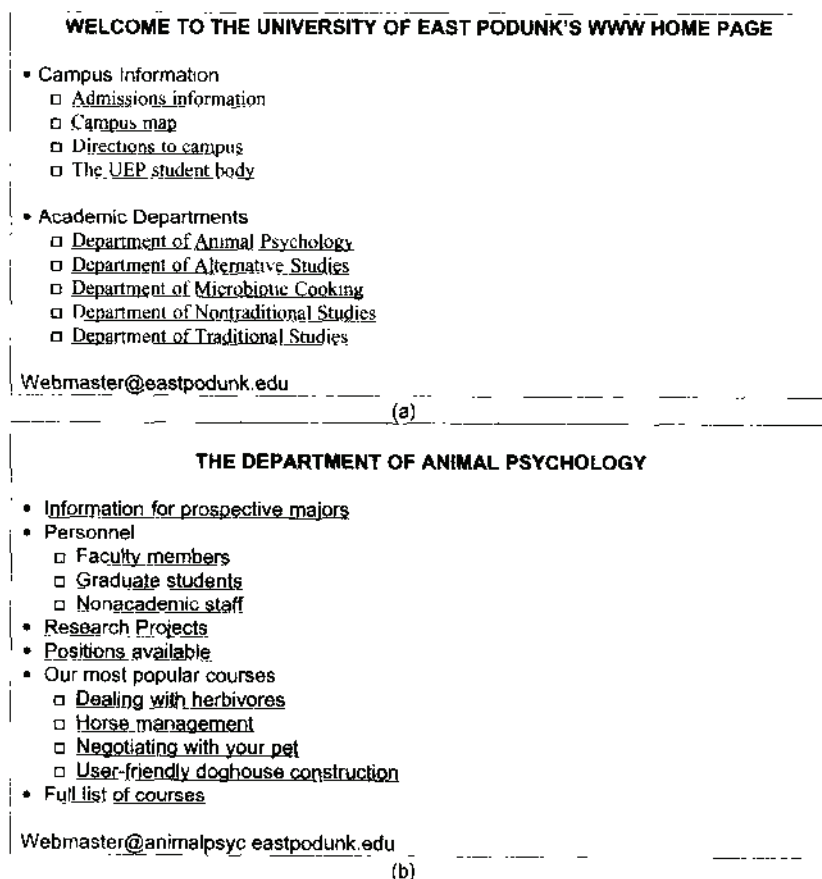


图 7.18

(a) 一个 Web 页面;(b) 单击 Department of Animal Psychology(动物心理学系)后到达的页面

对动物心理学系有好奇心的用户,通过单击该系的(带下划线的)名字,可以了解到更多的信息。然后浏览器取回该名字所链接的页面并显示出来,如图 7.18(b)所示。在这里还可以单击其他带下划线的项以取回其他的页面,如此等等。新的页面可以与第一个页面在同一台机器上,也可以在相距半个地球的另一台机器上。用户对此无法判断。取回页面的过程由浏览器来完成,不需要用户的任何帮助。如果用户返回到主页,那么,已经浏览过的链接在显示时可能带点划线(也可能是不同的颜色),以便与还没有浏览过的链接相区别。请注意,单击主页上的校园信息(Campus Information)行则什么也不做,它没有下划线,这表明它只是文本,并没有被链接到另一个页面上。

Web 的基本工作模型如图 7.19 所示。在这里,浏览器正在客户机器上显示一个 Web 页面。当用户单击了一行指向 abcd.com 服务器上页面的文本时,浏览器为了跟随超链接,它发送一条消息给 abcd.com 服务器,以请求所要的页面。页面到达之后即被显示出来。如果该页面包含一个指向 xyz.com 服务器上页面的超链接,并且用户也单击了该链接,那么浏览器向那台机器发送一个页面请求,如此可以无限地继续下去。

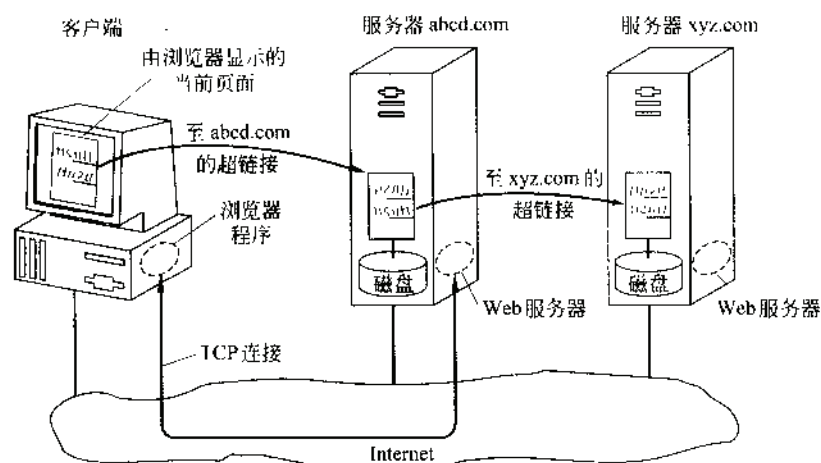


图 7.19 Web 模型的一部分

## 客户端

现在我们来更详细地看一看图 7.19 中的客户端。从本质上来说,浏览器是一个程序,它能够显示 Web 页面,也能够捕捉住对于已显示页面上各个项目的鼠标单击事件。当一个项目被选中时,浏览器跟随此超链接,并将所选择的页面取回来。因此,内嵌的超链接需要有一种方法来命名 Web 上的任何其他页面。Web 页面是用 URL (Uniform Resource Locator, 统一资源定位符) 来命名的。一个典型的 URL 如下所示:

`http://www.abcd.com/products.html`

我们将在本章的后面解释 URL。现在,我们只需知道 URL 有 3 个部分:协议名 (http)、页面所在机器的 DNS 名 (www.abcd.com),以及(通常是)包含页面的文件的名



字(products.html)。

当用户单击一个超链接时,浏览器执行一系列步骤以取回该超链接所指向的页面。假设一个用户正在浏览 Web,他在 Internet 电话簿上发现了一个指向 ITU 主页 <http://www.itu.org/home/index.html> 的链接。我们来看一看当这个链接被选中时所发生的步骤。

- (1) 浏览器确定 URL(只需检查一下被选中的是什么)。
- (2) 浏览器向 DNS 询问 [www.itu.org](http://www.itu.org) 的 IP 地址。
- (3) DNS 的回复是 156.106.192.163。
- (4) 浏览器与 156.106.192.163 上的 80 端口建立一个 TCP 连接。
- (5) 然后它发送一个请求,要求获取文件/home/index.html。
- (6) [www.itu.org](http://www.itu.org) 服务器发送文件/home/index.html。
- (7) TCP 连接被释放。
- (8) 浏览器显示/home/index.html 中的所有文本。
- (9) 浏览器取回并显示该文件中的所有图片。

许多浏览器在屏幕下方的状态栏中显示它们现在正在执行哪一步。通过这种方式,当性能不佳时,用户就可以看到这是因为 DNS 没有响应,还是因为服务器没有响应,或者只是因为页面传输期间发生了网络拥塞。

为了能够显示新的页面(或者任何页面),浏览器必须要理解页面的格式。为了让所有的浏览器都能理解所有的 Web 页面,Web 页面是用一种被称为 HTML 的标准语言来编写的。这种 HTML 语言专门被用于描述 Web 页面。在本章的后面我们将详细地讨论 HTML 语言。

虽然浏览器基本上是一个 HTML 解释器,但大多数浏览器都有许多按钮和特性,这使得用户在 Web 上的航行更加容易。大多数浏览器都有一个按钮用于回退到上一个页面、有一个按钮用于前进到下一个页面(这只有当用户已经回退到以前的页面时才有效),还有一个按钮用于直接跳到用户自己的起始页。大多数浏览器都有一个按钮或菜单项以便在一个给定的页面上设置书签,还有一个按钮或菜单项可显示书签列表,这样,只需单击几次鼠标就可以再次访问这些页面中的任何一个。页面也可以被保存到磁盘上,或者被打印出来。一般有非常多的选项可被用来控制屏幕的布局以及设置各种用户参数。

Web 页面除了有普通文本(不带下划线)和超文本(带下划线)以外,还可以包含图标、线条画、地图和照片。这些元素的每一种都可以被(任意地)链接到另一个页面上。用户单击这样的一个元素,就可以让浏览器取回被链接的页面并且在屏幕上显示该页面,这个过程就如同用户在超链接文本上进行单击一样。对于像照片和地图这样的图片,下一步取回哪一个页面取决于用户单击了图片的哪个部分。

并非所有的页面都包含 HTML。一个页面可以由 PDF 格式的文档、GIF 格式的图标、JPEG 格式的照片、MP3 格式的歌曲、MPEG 格式的视频,或者其他数百种文件类型的任何一种组成。由于标准的 HTML 页面可以链接到这些文件类型中的任何一种,因此,当浏览器遇到一个它无法解释的页面时就会出现问题。

大多数浏览器并没有为日益增多的文件类型内置相应的解释器,这会使浏览器越来

越大,相反,它们采用了一种更加通用的解决方案。当服务器返回一个页面时,它也返回一些有关该页面的附加信息。这些信息包含了该页面的 MIME 类型(见图 7.12)。text/html 类型的页面可被直接显示,其他一些内置类型的页面也这样处理。如果一个页面的 MIME 类型不是一种内置的类型,则浏览器参照它的 MIME 类型表,以决定如何显示该页面。这张表将每一种 MIME 类型与一个查看器(viewer)关联在一起。

有两种可能的扩展浏览器的方式:插件和辅助应用程序(helper application)。插件(plug-in)是一个代码模块,浏览器从磁盘上的一个特定目录中将插件取出来,并安装成自己的一个扩展模块,如图 7.20(a)所示。由于插件运行在浏览器的内部,因此它们可以访问当前的页面,也可以修改页面的外观。当插件完成了它的工作以后(通常是当用户移到了另一个不同的 Web 页面以后),它就被从浏览器的内存中移除掉。

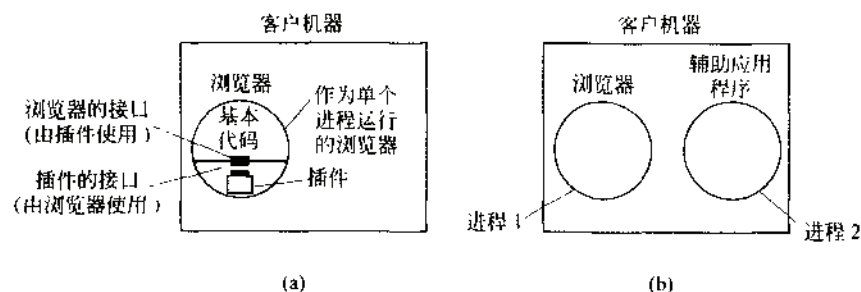


图 7.20  
(a) 浏览器插件; (b) 辅助应用程序

每一种浏览器都定义了一组过程,所有的插件必须实现这些过程,这样浏览器才可以调用插件。例如,通常浏览器的基本代码调用一个专门的过程以便将待显示的数据传递给插件。这组过程是插件的接口,它与特定的浏览器相关。

另外,浏览器也为插件提供了一组它自己的过程,以便向插进提供服务。在浏览器的接口中,较为典型的过程有申请和释放内存、在浏览器的状态栏上显示一条消息,以及向浏览器查询有关的参数。

在一个插件可被使用以前,首先必须安装该插件。一般的安装过程是,首先用户到插件的 Web 站点上下载一个安装文件。在 Windows 平台上,这通常是一个带 .exe 扩展名的自解压 zip 文件。当用户双击此 zip 文件时,一个附着在 zip 文件前端的小程序开始执行,该程序将插件解压缩出来,并把它复制到浏览器的插件目录中。然后,该程序执行适当的调用以便注册该插件的 MIME 类型,并将插件与该类型关联起来。在 UNIX 平台上,安装程序通常是一段负责复制和注册任务的 shell 脚本。

另一种扩展浏览器的方式是使用一个辅助应用程序(helper application)。这是一个完整的程序,它作为独立的进程来运行,如图 7.20(b)所示。由于辅助应用程序是一个单独的程序,因此它没有为浏览器提供接口,也不使用浏览器的服务。相反地,它一般只是接受一个临时文件的名字(待显示的内容被保存在这个文件中),然后打开该文件并显示其内容。通常情况下,辅助应用程序是一些独立于浏览器而存在的大型程序,比如 Adobe

公司用于显示 PDF 文件的 Acrobat Reader 或者 Microsoft Word。有些程序(例如 Acrobat)带一个插件,并且由这个插件来调用辅助程序本身。

许多辅助应用程序使用 MIME 的 application 类型。目前已经定义了相当多的子类型,例如,PDF 文件的 application/pdf 以及 Word 文件的 application/msword。通过这种方式,URL 就可以直接指向一个 PDF 文件或者 Word 文件,当用户单击它时,浏览器自动启动 Acrobat 或 Word,并将含有待显示内容的临时文件的名字传递给它。因此,用户可以将浏览器配置成能处理几乎无限多种文档的类型,而且无需改变浏览器。现代的 Web 服务器常常被配置了数百种“类型/子类型”的组合,每当安装一个新程序的时候,新的类型/子类型组合也随之被加入进来。

辅助应用程序并不局限于只能使用 MIME 的 application 类型。例如,Adobe Photoshop 使用 image/x-photoshop,而 RealOne Player 能够处理 audio/mp3。

在 Windows 平台上,当一个程序被安装到计算机上时,它把自己希望处理的 MIME 类型注册到系统中。当多个查看器可用于同一种于类型,比如 video/mpg 时,这种机制将会导致冲突。其结果是,最后注册的程序覆盖掉原有的(MIME 类型,辅助应用程序)关联关系,从而将这种类型占为己有。因此,安装一个新程序可能会改变浏览器处理已有类型的方式。

在 UNIX 平台上,注册过程一般不是自动的。用户必须手工更新某些特定的配置文件。这种做法需要做更多的工作,但减小了发生意外的可能性。

浏览器也能够打开本地的文件,而不一定非得从远程的 Web 服务器上取回文件。由于本地文件没有 MIME 类型,因此,对于除了像 text/html 和 image/jpeg 这样的内置类型以外的其他类型,浏览器需要某一种方法来决定该使用哪一个插件或辅助应用程序。为了处理本地的文件,辅助应用程序可以与一个文件扩展名关联起来,就好像与一个 MIME 类型关联起来一样。通过标准的配置,当浏览器打开 foo.pdf 时,它会用 Acrobat 来打开此文件,而当打开 bar.doc 时,它将会用 Word 来打开此文件。有些浏览器使用 MIME 类型、文件扩展名,甚至从文件本身提取的信息来猜测 MIME 类型。特别是 Internet Explorer,在有可能的情况下,它对文件扩展名的依赖甚至超过了对 MIME 类型的依赖。

同样地,由于许多程序都希望(实际上是渴望)处理某些类型的文件,比如说.mpg,所以这里也可能发生冲突。在安装过程中,供专业人员使用的程序常常会针对他们可能会涉及到的 MIME 类型和扩展名,显示一些复选框,从而允许用户来选择合适的程序,这样就不会无意中覆盖掉已有的关联关系。面向消费者市场的程序则假设用户根本不知道什么是 MIME 类型,它们只是尽可能地夺取到所有的类型,而根本不关心以前安装的程序所做过的事情。

能够对浏览器扩展大量新的类型是很方便的,但也会导致麻烦。当 Internet Explorer 取回了一个扩展名为 exe 的文件时,它知道该文件是一个可执行程序,因而没有辅助应用程序。它应该采取的动作很显然是运行这个程序,然而,这可能是一个巨大的安全漏洞。一个恶意的 Web 站点所需做的事情是,生成一个包含许多图片(比如电影明星或体育明星)的 Web 页面,并且将所有的图片都链接到一个病毒上。于是,用户单击一幅

图片就会导致取回一个未知的、可能恶意的可执行程序,并且在用户的机器上运行。为了预防像这样的有害来客,用户可以将 Internet Explorer 配置成有选择地自动运行未知的程序,但是,并不是所有的用户都懂得如何管理它的配置。

在 UNIX 平台上,shell 脚本也存在类似的问题,但这要求用户有意识地将 shell 安装成辅助应用程序。所幸的是,这种安装十分复杂,没有人能够在无意中完成这种安装(甚至很少有人会有意地这么做)。

### 服务器端

关于客户端的介绍到此为止,现在我们来看一看服务器端。正如我们在上文中所看到的,当用户键入一个 URL 或者单击一行超文本时,浏览器就会解析 URL,并将 http:// 和下一个斜线之间的部分解释成待查找的 DNS 名字。有了服务器的 IP 地址以后,浏览器与该服务器上的 80 端口建立一个 TCP 连接,然后它发送一条命令,其中包含了 URL 的剩余部分,即该服务器上的一个文件的名字。然后服务器返回文件以供浏览器显示。

乍看起来,Web 服务器与图 6.6 中的服务器非常类似。该图中的服务器像实际的 Web 服务器一样,得到一个待查找的文件的名字并返回结果。在这两种情况下,服务器在它的主循环中执行的步骤如下:

- (1) 接受来自客户(浏览器)的 TCP 连接。
- (2) 获取所需文件的名字。
- (3) 获取文件(从磁盘上)。
- (4) 将文件返回给客户。
- (5) 释放该 TCP 连接。

现代的 Web 服务器有更多的特性,但从本质上来说,这就是 Web 服务器所做的工作。

这种设计的一个问题是,每次请求都要求进行一次磁盘访问以获取文件。其结果是,Web 服务器每秒钟能够处理的请求数,不超过它能够访问磁盘的次数。高端的 SCSI 磁盘的平均访问时间是 5ms 左右,这将限制服务器每秒钟最多可处理 200 个请求,如果经常需要读取大的文件,那么能处理的请求次数会更少。对于一个大的 Web 站点来说,这个数字太小了。

一种显而易见的改进方法(所有的 Web 服务器都采用了这种方法)是在内存中维护一个缓存,其中保存着  $n$  个最近使用过的文件。服务器在到磁盘上取文件之前,首先检查缓存,如果缓存中已经有该文件,则直接从内存中取出文件,从而避免了磁盘访问。虽然真正有效的缓存需要大量的主内存,以及一定的额外处理时间来检查缓存和管理其内容,但是,节省下来的时间几乎总是抵得上这些开销和费用。

构建快速 Web 服务器的下一个步骤是让服务器变成多线程模式。其中一种设计方案如图 7.21 所示,服务器由一个前端模块(front-end module)和  $k$  个处理模块(processing module)组成,前端模块接受所有的进来请求。 $k+1$  个线程全部属于同一个进程,这样所有的处理模块都可以访问当前进程地址空间中的缓存。当一个请求进来时,前端模块接受它,并建立一条描述该请求的简短记录,然后将该记录递交给其中一个处理

模块。在另一种可能的方案中,它取消了前端模块,因此,每个处理模块试图获得它自己的请求,但为了防止冲突,这时需要一个锁协议(locking protocol)。

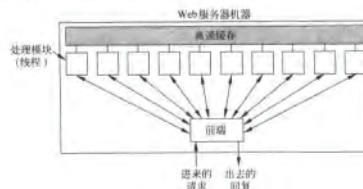


图 7.21 带有前端和处理模块的多线程 Web 服务器

处理模块首先检查缓存,看其中是否有所需的文件。如果缓存中有该文件,则处理模块修改记录,在记录中增加一个指向该文件的指针。如果缓存中没有该文件,则处理模块执行一次磁盘操作,将文件读入缓存(可能要丢弃其他一些缓存的文件,以便为它腾出空间)。从磁盘上读入文件以后,文件被放在缓存中,同时也被送回给客户。

这种方案的优点是,当一个或多个处理模块因为等待磁盘操作的完成而被阻塞(因此它们不占用 CPU 时间)时,其他的模块可以继续处理其他的请求。当然,为了比单线程模型有更加实质的改进,这里有必要使用多个磁盘,因而多个磁盘可以同时工作。使用  $k$  个处理模块和  $k$  个磁盘后,其吞吐量可以达到在单线程服务器和单个磁盘情形下的  $k$  倍。

从理论上来说,一个单线程服务器和  $k$  个磁盘也能获得  $k$  倍的增益,但是其代码和管理工作将会非常复杂,因为它不能使用普通的阻塞版本的 READ 系统调用来访问磁盘。若采用多线程的服务器,则仍然可以使用阻塞的 READ 调用,因为 READ 只阻塞发出调用的那个线程,而不是整个进程。

现代的 Web 服务器所做的不只是接受文件名和返回文件。事实上,对每个请求的实际处理过程可能非常复杂。由于这个原因,在许多服务器中,每个处理模块要执行一系列步骤。前端将每个进来的请求传递给第一个空闲的模块,然后该模块根据这个特定的请求,执行下列步骤中的某一个子集。

- (1) 确定被请求的 Web 页面的名字。
- (2) 鉴别客户的身份。
- (3) 针对客户的身份,执行访问控制。
- (4) 针对 Web 页面,执行访问控制。
- (5) 检查缓存。
- (6) 从磁盘上取回被请求的页面。
- (7) 确定在回复中应包含的 MIME 类型。



- (8) 处理各种零碎的事项。
- (9) 将回复返回给客户。
- (10) 在服务器的日志中增加一个条目。

第 1 步是必需的,因为进来的请求中可能没有包含文件实际名字的字符串。例如,在 URL `http://www.cs.vu.nl` 中没有文件名。在这种情况下,有必要将它扩展到某一个默认的文件名。同样地,现代的浏览器可以指定用户的默认语言(例如意大利语或英语),这样,如果存在该语言的 Web 页面,则服务器就可以选择该页面。一般来说,由于有各种各样的文件命名习惯,所以,文件名的扩展已经不像它刚出现时那样微不足道了。

第 2 步包括验证客户的身份。对于那些并非对全体公众都开放的页面来说,这一步是必需的。我们将在本章的后面讨论一种认证方法。

第 3 步针对的是,在给定了客户身份和位置的情况下,看是否存在某些限制会影响到这个请求。第 4 步检查是否存在与页面自身相关联的访问限制。如果被请求的页面所在的目录中存在一个特殊的文件(比如, `htaccess`),那么它可能限定了只有特殊的域才能访问该页面,比如,只能是来自公司内部的用户才可以访问。

第 5 步和第 6 步涉及到取回页面。第 6 步需要具备同时读取多个磁盘的能力。

第 7 步根据文件的扩展名、文件开头的几个字、配置文件以及其他可能的来源,来确定 MIME 类型。第 8 步处理各种杂项,比如建立一个用户轮廓,或者收集特定的统计数据。

第 9 步送回结果,第 10 步在系统日志中增加一个条目以便于管理。将来可以从这些日志中挖掘出有关用户行为的有价值的信息,比如说,人们访问页面的顺序。

如果服务器在每秒钟时间里收到了过多的请求,那么,无论并行使用多少个磁盘,CPU 都将无法执行所有的处理工作。解决的办法是增加更多的节点(计算机),可能还要使用复制的磁盘,以免这些磁盘成为下一个瓶颈。这就导致产生了图 7.22 中的**服务器场(server farm)**模型。前端仍然接受进来的请求,但是会将它们分散到多个 CPU 而不是多个线程上,以减小每台计算机上的负载。各台机器本身可以跟以前一样使用多线程和流水线模型。

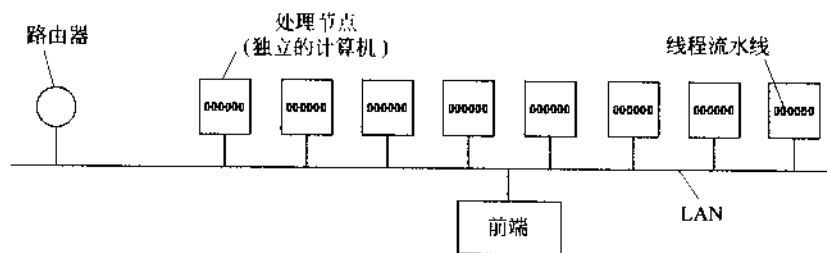


图 7.22 服务器场

服务器场方案也存在一个问题,也就是,它不再有共享的缓存,因为除非使用昂贵的共享内存的多处理器结构,否则,每个处理节点只有它自己的内存。克服这种性能损失的一种办法是让前端记录下每个请求被转发到哪个处理节点中,并且把对于同一个页面的



后续请求都转发给同一个节点。这种做法使得每个节点专门处理某些页面,从而不会因为每个缓存中都保留每个文件而浪费缓存空间。

服务器场的另一个问题是,客户的 TCP 连接终止于前端,所以回复也必须经过前端。图 7.23(a)显示了这种情形,进来的请求(1)和出去的回复(4)都通过了前端。有时候,使用一种被称为 TCP 移交(TCP handoff)的技巧可以避免这个问题。通过这种技巧,TCP 端点可以被传递到处理节点上,因此,处理节点可以直接回复给客户,如图 7.23(b)中的(3)所示。对于客户来说,这种移交过程可以做到是透明的。

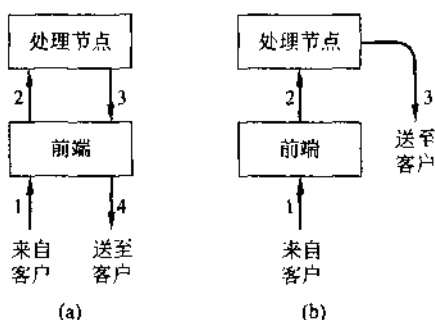


图 7.23

(a) 正常的请求-回复消息序列; (b) 使用 TCP 移交时的序列

## URL—统一资源定位符

我们已经反复说过,Web 页面中可以包含指向其他 Web 页面的指针。现在我们来详细地看一看这些指针是如何实现的。当最初 Web 被建立起来时,为了让一个页面指向另一个 Web 页面,很显然需要有一些命名和定位页面的机制。尤其是,在显示一个被选中的页面之前,首先必须回答三个问题:

- (1) 这个页面叫什么?
- (2) 这个页面在哪里?
- (3) 如何才能访问这个页面?

如果每个页面都以某种方式被分配了一个惟一的名称,那么在标识页面的时候就不会存在任何歧义。但是,问题还没有解决。请考虑一下人与页面之间的类比关系吧。在美国,几乎每个人都有一个社会保险号;由于任何两个人都不应该有相同的社会保险号,因此它是一个惟一的标识符。然而,如果你仅仅知道一个社会保险号,那么你就无法找到该保险号的所有者的地址,当然你也无法知道到底应该用英语、西班牙语还是汉语来给他写信。Web 基本上也有同样的问题。

Web 选择的解决方案是,用一种能同时解决这三个问题的方法来标识页面。每个页面被分配一个 URL(统一资源定位符,Uniform Resource Locator),此 URL 能有效地充当该页面在全球范围内的名字。URL 包括三个部分:协议(也被称为方案[scheme])、页面所在机器的 DNS 名字,以及惟一地指示特定页面的本地名字(通常只是页面所在机器上的一个文件名)。例如,本书作者所在系的 Web 站点上包含了一些关于阿姆斯特丹

(Amsterdam)大学和城市的视频。该视频页面的 URL 是:

`http://www.cs.vu.nl/video/index-en.html`

这个 URL 由三个部分组成: 协议(`http`)、主机的 DNS 名(`www.cs.vu.nl`)以及文件名(`video/index-en.html`), 各个部分之间用特定的标点符号分隔开。这里的文件名是在 `cs.vu.nl` 上相对于默认 Web 目录的一个路径。

许多站点为某些文件名提供了内置的快捷方式。在许多站点中, 一个空的文件名被默认为该组织的主页。通常情况下, 当所命名的文件是一个目录时, 这暗示着是指一个名为 `index.html` 的文件。最后, `~user/` 可能被映射到 `user` 的 WWW 目录, 然后被映射到该目录中的 `index.html` 文件。因此, 尽管本书作者的主页的实际文件名是某个默认目录中的 `index.html`, 但是你仍然可以用下面的 URL 来访问:

`http://www.cs.vu.nl/~ast/`

现在我们可以看一看超文本是如何工作的。为了让一小块文本可以被单击, 页面的编写者必须提供两部分信息: 被显示在页面上的可单击文本, 以及选择了文本后所转向的页面的 URL。在本章的后面我们将解释命令的语法。

当文本被选中时, 浏览器通过 DNS 来查找主机名。一旦知道了主机的 IP 地址, 浏览器就与该主机建立一个 TCP 连接; 然后通过这个连接, 用指定的协议来发送文件名。瞧, 页面就回来了。

这种 URL 方案直接让浏览器用多种协议来获得各种不同种类的资源, 从这层意义上来说, 它是非常开放的, 有很强的扩展性。事实上, 针对其他各种常见协议的 URL 也都被定义了。图 7.24 列出了更多常见 URL 的简化形式。

名称	用途	例子
http	超文本(HTML)	<code>http://www.cs.vu.nl/~ast</code>
ftp	FTP	<code>ftp://ftp.cs.vu.nl/pub/minix/README</code>
file	本地文件	<code>file:///usr/suzanne/prog.c</code>
news	新闻组	<code>news:comp.os.minix</code>
news	新闻文章	<code>news:AA0134223112@cs.utah.edu</code>
gopher	Gopher	<code>gopher://gopher.tc.umn.edu/11/Libraries</code>
mailto	发送电子邮件	<code>mailto:JohnUser@acm.org</code>
telnet	远程登录	<code>telnet://www.w3.org:80</code>

图 7.24 一些常见的 URL

我们现在简单地看一下这个列表。`http` 协议是 Web 本身的语言, 也是 Web 服务器所使用的语言。HTTP 代表了超文本传输协议(HyperText Transfer Protocol), 我们将在本章后面更详细地介绍 HTTP。

Internet 的文件传输协议 FTP(File Transfer Protocol)利用 `ftp` 协议来访问文件。FTP 有二十多年的历史了, 它已经树立了牢固的根基。由于全球有大量的 FTP 服务器,

所以 Internet 上任何地方的人都可以登录并下载所有已被放到 FTP 服务器上的文件。Web 并没有改变这一点;它只是使得用 FTP 来获取文件更加容易,因为 FTP 的界面用起来不够方便(但它比 HTTP 功能更强大,例如,它允许机器 A 上的用户将一个文件从机器 B 传输到机器 C)。

对于一个本地文件,通过使用 file 协议或者更简单地,仅仅使用它的名字,就可以把它当作一个 Web 页面来访问。这种方法与使用 FTP 非常类似,但它不要求有一个服务器。当然,它只能用于本地文件而不能用于远程文件。

在 Internet 出现之前很早就有了 USENET 新闻系统,它包括大约 30 000 个新闻组,在这些新闻组里,数百万人通过张贴和阅读与新闻组的话题相关的文章,来讨论各种各样的话题。用户可以用 news 协议来阅读一篇新闻文章,就好像它是 Web 页面一样。这意味着一个 Web 浏览器同时也是一个新闻阅读器。实际上,许多浏览器都包含一些按钮或菜单项,以使得阅读 USENET 的新闻甚至比使用标准的新闻阅读器还要容易。

news 协议支持两种格式。第一种格式指定一个新闻组,用户利用这种格式可以从预先配置的新闻站点上获得一个文章列表。第二种格式要求给出一篇特定新闻文章的标识符,比如说 AA0134223112@cs.utah.edu。然后,浏览器用 NNTP (网络新闻传输协议, Network News Transfer Protocol)从它预先配置的新闻站点上取回指定的文章。我们在本书中不学习 NNTP,但是它大致上基于 SMTP 协议,并且也有类似的风格。

Gopher 系统使用 gopher 协议,该系统是在明尼苏达大学(University of Minnesota)设计的,并以该校的运动队“金色地鼠”(也是一句俚语,表示“去找”,或“去取”)来命名。在时间上,Gopher 比 Internet 还要早几年。Gopher 是一种信息检索方案,在概念上与 Web 本身很类似,但它只支持文本而不支持图像。现在它基本上已经过时了,人们很少再使用它。

最后两种协议并非真正用于取回 Web 页面,但不管怎么样它们还是有用的。mailto 协议允许用户从 Web 浏览器中发送电子邮件,其做法是:单击 OPEN 按钮,并指定一个由 mailto:开头的 URL,在 mailto:后面跟着收件人的电子邮件地址。大多数浏览器的响应是:启动一个电子邮件程序,并且已经填好了地址和一些消息头域。

telnet 协议被用来与远程机器建立一个在线的连接,它的使用方法与 telnet 程序相同,这并不奇怪,因为大多数浏览器只是调用 telnet 程序(它也被用作一个辅助应用程序)。

简而言之,URL 已经被设计成不仅允许用户在 Web 中航行,而且还处理 FTP、news、Gopher、电子邮件以及 telnet,这使得所有提供这些非 HTTP 服务的特殊用户界面程序变得不再有必要了,并且因此也把几乎所有的 Internet 访问全部集成到一个单一的程序,即 Web 浏览器中。要不是事实上这个主意是由一位物理研究员想出来的,否则,它很容易被当作是某个软件公司广告部门的杰作了。

尽管 Web 拥有所有这些好的特性,但是,其日益增长的用途却暴露出了 URL 方案中一个固有的弱点。每个 URL 指向一台特定的主机。对于那些被频繁访问的页面,最好能有多个相距甚远的复本,以减少网络流量。问题是,URL 并没有提供任何一种“不指定页面所在位置就能够引用页面”的方法。用户不可能这样说:“我想要页面 xyz,但我不关心你从哪里获取该页面”。为了解决这个问题,并使复制页面成为可能,IETF 正致力于

URN(通用资源名, Universal Resource Name)系统的研究。URN 可以被看作是一种泛化的 URL, 尽管在 RFC 2141 中已经给出了一个建议的语法, 但这个话题仍然是一个研究性的课题。

### 无状态特性与 Cookie

正如我们反复看到过的, Web 基本上是无状态的, 它没有登录会话的概念。浏览器向服务器发送一个请求并取回一个文件, 然后服务器就忘记了它曾经见到过这个特殊的客户。

首先, 当 Web 仅仅被用来取回公开可用的文档时, 这个模型是完全胜任的。但当 Web 开始提供其他功能时, 就出现问题了。例如, 一些 Web 站点要求客户注册后(而且可能要付费)才能使用这些站点的服务。这带来的问题是, 服务器如何才能区分来自注册用户的请求和来自其他人的请求? 第二个例子来自电子商务。如果一个用户在电子商店里闲逛, 并且不时地将商品投入她的购物车中, 服务器如何才能跟踪购物车的内容? 第三个例子是像 Yahoo 这样的可定制的 Web 门户网站。用户可以建立一个细致的初始页面, 其中只包括他们想要的信息(例如, 他们的股票和他们喜欢的运动队), 但是, 如果服务器不知道用户是谁, 那么它如何才能显示正确的页面呢?

乍一看, 你可能想到了服务器可以通过观察用户的 IP 地址来跟踪他们。但是这个想法并不奏效。首先, 许多用户在共享的计算机上工作, 特别是在公司里。在这种情况下, IP 地址只标识计算机而不标识用户。其次, 更糟糕的是, 许多 ISP 使用 NAT, 因而所有用户发出的分组具有相同的 IP 地址。从服务器的角度来看, ISP 所有的用户(可能有数千个之多)都使用相同的 IP 地址。

为了解决这个问题, Netscape 公司发明了一项被称为 **cookie** 的备受批评的技术。这个名字来自于很早以前的程序员行话, 即程序调用一个过程并取回一些信息, 稍后可能需要使用这些信息来完成某些工作。从这种意义上来说, UNIX 文件描述符或 Windows 对象句柄都可以被看作 cookie。后来在 RFC 2109 中对 cookie 作了正式定义。

当客户请求一个 Web 页面时, 服务器除了提供所请求的页面以外, 还可以提供一些附加的信息。这些信息中可能包括一个 cookie, cookie 是一个小的(最多 4KB)文件(或字符串)。如果用户没有禁用 cookie, 则浏览器把接收到的 cookie 存储在客户机器硬盘上的 cookie 目录中。cookie 只是一些文件或字符串, 而不是可执行程序。原则上, cookie 可能包含病毒, 但由于 cookie 仅被当作数据来对待, 所以即使是病毒, 它也没有正式的运行途径, 因而不可能造成破坏。然而, 有些黑客总有可能利用浏览器的错误(bug)来激活病毒。

一个 cookie 可以包含至多 5 个域, 如图 7.25 所示。Domain(域名)指出一个 cookie 来自什么地方。浏览器总是应该执行检查以确保服务器没有谎报它们的域名。每个域在每个客户端可以存储不超过 20 个 cookie。Path(路径)是服务器目录结构中的一个路径, 它标识了服务器文件树的哪些部分可能会使用该 cookie。路径通常是/, 这意味着整棵树。

域名	路径	内容	过期时间	安全
toms-casion.com	/	CutomerID=497793521	15-10-02 17:00	是
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	否
aportal.com	/	Prefs=Stk;SUNW+ORCL;Spt;Jets	31-12-10 23:59	否
sneaky.com	/	UserID=3627239101	31-12-12 23:59	否

图 7.25 一些 cookie 的例子

内容 (Content)域采用“name=value”(名字=值)的形式。这里 name 和 value 都可以是服务器期望的任意值。这个域正是存放 cookie 内容的地方。

过期时间 (Expires)域指定了 cookie 在什么时候过期。如果这个域不存在,则浏览器在退出时将 cookie 丢弃,这样的 cookie 称为非持久的 cookie(nonpersistent cookie)。如果提供了时间和日期,则这样的 cookie 称为持久的 cookie(persistent cookie),它被一直保存在客户端,直至过期为止。过期时间采用格林威治标准时间。为了从客户的硬盘上删除一个 cookie,服务器只需再将它发送一次,但是指定一个已经过去的过期时间即可。

最后,通过设置安全 (Secure)域,可以指示浏览器只向安全的服务器才返回该 cookie。这个特性可用于电子商务、银行和其他的安全应用。

我们现在已经看到了如何获得 cookie,但怎样使用 cookie 呢?就在浏览器向某个 Web 站点发送一个页面请求之前,浏览器检查它的 cookie 目录,以确定这个请求的目标域是否在当前客户端放置了 cookie。如果确实放置了,则该域放置的所有 cookie 都被包含到请求消息中。服务器得到了这些 cookie 以后,就可以按它所期望的方式来解释这些 cookie。

我们现在来看一看 cookie 的一些可能用法。在图 7.25 中,第一个 cookie 是 toms-casino.com 设置的,用来标识一个顾客。当同一个客户下个星期登录进去花掉更多的钱时,浏览器将 cookie 发送过去,这样服务器就知道他是谁了。有了顾客 ID,服务器就可以在数据库中查找顾客记录,并且利用这些记录信息来创建一个合适的 Web 页面以供显示。如果已知该顾客有赌博的习惯,则这个页面可能由扑克游戏、今日赛马比赛名单,或老虎机组成。

第二个 cookie 来自 joes-store.com。这里的场景是客户在商店里闲逛,同时寻找和购买好的东西。当她发现一件便宜货并单击它时,服务器创建一个包含商品数量和产品代码的 cookie,并把它送回给客户。当客户继续在商店里闲逛时,该 cookie 也跟随每个新的页面请求被返回到服务器端。随着顾客购买的物品越来越多,服务器将它们加入到 cookie 中。在图中,购物车包含三种商品,其中第三种商品需要两件。最后,当顾客单击“前往结账 (PROCEED TO CHECKOUT)”时,这个 cookie 现在包含了所购买商品的完整列表,它和当前的请求一起被发送出去。通过这种方式,服务器就能够精确地知道顾客已经购买了哪些东西。

第三个 cookie 被用于 Web 门户网站。当用户单击一个指向该门户的链接时,浏览



器将 cookie 发送过去。它告诉该门户站点,应该创建这样一个页面:其中包含 Sun 公司和 Oracle 公司的股票价格,以及纽约喷气机队(New York Jets)橄榄球赛的结果。由于 cookie 最多可以达到 4KB,所以它有足够的空间来存放关于报纸头条新闻、本地天气、特价信息等更多详细的参数设置。

cookie 也可以使服务器自身获益。例如,假设一个服务器想知道它有多少个不重复的访问者,以及每个人在离开该站点前看过多少页面。当第一个请求到来时,它不会随带 cookie,因此服务器送回一个包含“Counter=1”的 cookie。如果用户后来再次单击这个站点,则客户将把这个 cookie 送回给服务器。服务器每次增加计数器并送回给客户。服务器通过检查这个计数器,就可以知道有多少人在看了第一个页面后就离开了,有多少人看了第二个页面,等等。

cookie 也可能被滥用。从理论上来说,cookie 只应该被送回到源站点,但黑客们能利用浏览器中的各种错误,来截获那些并非发给他们的 cookie。因为有些电子商务站点将信用卡号码放在 cookie 中,所以 cookie 被滥用的潜在可能性是显而易见的。

关于 cookie 的一个有争议的用途是秘密地收集有关用户的 Web 浏览习惯的信息。其工作过程如下所述。一家广告公司,比如说 Sneaky 广告公司,它联系了一些主要的 Web 站点,并且在这些站点的页面上,放置一些标语广告来为它的企业客户宣传他们的产品,它当然向这些站点的所有者支付一定的费用。广告公司并不是向这些站点提供一个 GIF 或 JPEG 文件以便放在每个页面上,而是给这些站点提供一个 URL,让它们将这个 URL 加入到每一个页面中。广告公司分发的每个 URL 的文件部分包含一个惟一的数字,例如:

<http://www.sneaky.com/382674902342.gif>

当用户第一次访问一个包含如此广告的面 P 时,浏览器取回 HTML 文件。然后浏览器检查该 HTML 文件,并看到了指向 [www.sneaky.com](http://www.sneaky.com) 上图像文件的链接,因此它向 [www.sneaky.com](http://www.sneaky.com) 发送一个对该图像的请求。一个包含广告的古F文件,连同—个包含了惟一用户 ID(即图 7.25 中 3627239101)的 cookie,一起被返回给客户。Sneaky 公司记下了这样的事实:此 ID 的用户曾经访问过页面 P。这很容易做到,因为被请求的文件(382674902342.gif)只在页面 P 上被引用。当然,实际的广告可能出现在成千上万的页面上,但每次都有一个不同的文件名。Sneaky 公司也许在每次发布广告时从产品制造商那里获得几美分的收益。

随后,当这个用户访问另一个包含 Sneaky 广告的古B页面时,浏览器从服务器取回 HTML 文件以后,它看到了一个指向 <http://www.sneaky.com/493654919923.gif> 的链接,于是请求该文件。由于浏览器已经有一个来自 sneaky.com 域的 cookie,于是它将这个含有用户 ID 的 cookie 也包含在请求中。Sneaky 现在知道该用户已经访问了第二个页面。

在—段时间以后,即使用户从未单击任何广告,Sneaky 也能够建立起有关该用户浏览习惯的完整轮廓。当然,它还没有用户的名字(不过它有用户的 IP 地址,也许再根据其



他的数据库信息,就足以推断出用户的名字)。然而,如果该用户曾经将他的名字提供给任何一个与 Sneaky 合作的站点,那么现在 Sneaky 就可以将带有用户名字的完整轮廓信息卖给任何一个有意购买此类信息的人了。出售这些信息可能会给 Sneaky 带来足够的利润,使它能在更多的 Web 站点上放置更多的广告,从而收集更多的信息。这整桩交易中最阴险的部分是,大多数用户完全没有意识到这种信息收集方式,他们甚至可能认为自己是安全的,因为他们没有单击任何广告。

如果 Sneaky 想要更加卑鄙(supersneaky),则页面上的广告就不必是标准的标语广告。一个背景颜色的单像素“广告”(因此是不可见的)与标语广告有完全相同的效果:它要求浏览器取回这个  $1 \times 1$  像素的 gif 图像,并将该像素所属域的所有 cookie 发送给服务器。

为了维护一定程度的隐私,有些用户将他们的浏览器配置成拒绝所有的 cookie。然而,这会给那些合法使用 cookie 的 Web 站点带来困难。为了解决这个问题,用户有时候可以安装反 cookie 软件,它们是一些特殊的程序,当每个进来的 cookie 到达时对它进行检查,并根据用户指定的选择(例如,哪些 Web 站点是可以信任的)来决定是接受还是丢弃。这使用户可以细粒度地控制哪些 cookie 可以被接受,而哪些应该被拒绝。现代的浏览器,例如 Mozilla ([www.mozilla.org](http://www.mozilla.org)),已经内置了一些精心制作的、由用户来控制 cookie 的机制。

### 7.3.2 静态 Web 文档

Web 的基础是将 Web 页面从服务器传输到客户端。在最简单的形式中,Web 页面是静态的,也就是说,它们存在于服务器上并等待被取走。在这种意义上,即使一段视频也是一个静态的 Web 页面,因为它也是一个文件。在这一小节中我们会详细地讨论静态 Web 页面。在下一小节中,我们将讨论动态内容。

#### HTML—超文本标记语言

现在,Web 页面是用一种被称为 HTML(HyperText Markup Language,超文本标记语言)的语言来编写的。HTML 允许用户在 Web 页面中包含文本、图形和指向其他 Web 页面的指针。HTML 是一种标记语言,一种描述了如何格式化文档的语言。“标记”这个术语的来源是,在过去,那时审稿人真的通过对文档进行标记以告诉印刷工应该使用什么字体等。因此,标记语言包含了用于格式化的显式命令。例如,在 HTML 中,<b>表示粗体字的开始,</b>表示粗体字的结束。相对于其他无显式标记的语言而言,标记语言的优点是:为它编写浏览器非常简单;浏览器只需理解标记命令即可。TeX 和 troff 是另外两种著名的标记语言。

由于所有的标记命令都被嵌在每个 HTML 文件中,并且标记命令已经被标准化,所以,任何一个 Web 浏览器都可以读取任何 Web 页面,并且对页面重新格式化。能够在获得 Web 页面以后重新对它们进行格式化是非常重要的,因为一个页面有可能是在  $1600 \times 1200$  大小的 24 位颜色的窗口中被设计出来的,但它必须被显示在一个  $640 \times 320$  大小的

8 位颜色的窗口中。

下面我们将对 HTML 作简单的介绍,这里的介绍只是让你对它有一个大致的概念。尽管你可以用任何一个标准的编辑器来编写 HTML 文档,而且也确实有许多人这样做,但是,你也可以使用专用的 HTML 编辑器或文字处理器,它们能够为你完成大部分工作(但是,你对最终结果的所有细节的控制能力也相应地减弱了)。

一个 Web 页面由一个头部和一个主体组成,它们都被夹在<html>和</html>标签(格式化命令)之间,不过,如果缺少这两个标签的话,大多数浏览器也不会出错。正如图 7.26(a)所示,头部由<head>和</head>标签括起来,而主体部分则由<body>和</body>标签括起来。标签中的字符串被称为指示符(directive)。大多数 HTML 标签都采用这种格式,即<something>表示某一个标注 something 的开始,</something>表示该标注的结束。大多数浏览器都有一个 VIEW SOURCE(查看源文件)的菜单项或类似的命令。当用户选择了这个菜单项以后,浏览器就会显示当前页面的 HTML 源文件,而不是格式化之后的输出。

标签既可以是小写也可以是大写。因此<head>和<HEAD>含意相同,但是新版本的 HTML 标准要求只能使用小写。HTML 文档的实际布局结构是无关紧要的。HTML 解析器忽略额外的空格和回车,因为它们必须重新对文本进行格式化,以使得这些文本适合当前的显示区域。因此,编写者可以任意添加空格,以使得 HTML 文档更具可读性,而这也是 HTML 文档所极其需要的。另外一个后果是,因为空行在显示时被忽略,所以它们不能被用来分割段落。因此就需要一个显式的标签。

有些标签带有(命名的)参数,这些参数被称为属性(attribute)。例如,

```

```

是一个<img>标签,它带有参数 src 和 alt,并且 src 被设置为 abc,alt 被设置为 foobar。对于每一个标签,HTML 标准给出了它允许带的参数的列表,以及这些参数的含义。由于每个参数已被命名,所以这些参数被给出的顺序并不重要。

从技术上讲,HTML 文档是用 ISO 8859-1 Latin-1 字符集编写的,但是,如果用户的键盘只支持 ASCII 字符,则需要用转义序列来代表特殊的字符,比如 è。标准中给出了特殊字符的列表。它们都以“&”符号作为开头,以“;”作为结束。例如,&nbsp;表示空格,&grave;表示 è,&acute;表示 é。由于<,>和 & 有特殊的含意,所以它们也只能用转义序列来表示,分别是 &lt;,&gt;和 &amp;。

头部中的主要内容是标题(title),它由<title>和</title>来分割,另外一些特定的元信息(meta-information)也可以在这里出现。标题本身不显示在页面上。有些浏览器用它作为页面窗口的标题。

现在我们来看一看图 7.26 中列出的其他一些特性。图 7.26 中使用的所有标签和其他一些标签被列出在图 7.27 中。标题(heading)由<h>标签生成,其中 n 是一个 1 到 6 之间的数字。<h1>是最重要的标题;<h6>是最不重要的标题。浏览器负责在屏幕上恰当地显示它们。通常较小数字的标题会采用较大和较重的字体来显示。浏

```

<html>
<head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
 <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. <p>
<br>
<h2> Product information </h2>
<ul>
<li> <a href="http://widget.com/products/big"> Big widgets </a>
<li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers </h2>
<ul>
<li> By telephone: 1-800-WIDGETS
<li> By fax: 1-415-765-4321
</ul>
</body>
</html>

```

(a)



(b)

图 7.26

(a) 示例性 Web 页面的 HTML 源文件；(b) 格式化后的页面

浏览器也可能为每种级别的标题选择使用不同的颜色。通常<h1>标题的字体较大较粗，并且在标题的上方和下方至少有一个空行。相反，<h2>标题较小，且上方和下方只有较少的空白。

标签	说明
<html>...</html>	声明该 Web 页面以 HTML 来编写
<head>...</head>	定义页面的头部
<title>...</title>	定义标题(不显示在页面上)
<body>...</body>	定义页面的主体
<h n>...</h n>	定义一个级别为 n 的标题
<b>...</b>	设置...为粗体
<i>...</i>	设置...为斜体
<center>...</center>	在页面上水平居中
<ul>...</ul>	将一个未排序(项目符号)列表括起来
<ol>...</ol>	将一个编号列表括起来
<li>...</li>	将未排序或编号列表中的一个表项括起来
 	强制换行
<p>	段落开始
<hr>	插入水平线
	在此显示图像
<a href="...">...</a>	定义超链接

图 7.27 一些常见的 HTML 标签,有些带有附加的参数

标签<b>和<i>分别用于进入粗体和斜体模式。如果浏览器不能够显示粗体和斜体,那么它必须使用其他某种方法来表现它们,例如,为每一种模式使用不同的颜色或者反色显示。

HTML 提供了许多机制来构造列表,包括嵌套的列表。列表以<ul>或者<ol>开头,在这两种情况下,表项都以<li>开头。<ul>标签代表一个未排序列表的开始。在源文件中以<li>标记的每个表项,在显示时前面会加上一个项目符号(•)。这种机制的一个变种是<ol>,它用于排序的列表。当使用这个标签时,浏览器用数字对<li>表项进行编号。除了使用不同的开始和结束标签以外,<ul>或<ol>具有相同的语法和类似的结果。

<br>、<p>和<hr>标签都被用于指示文本分区之间的边界。精确的格式是由与页面相关联的样式表(见后)来决定的。<br>标签仅仅是强制分行。通常,浏览器不会在<br>后面插入一个空行。相反,<p>开始一个段落,例如,它可能插入一个空行,还可能有一些缩进。(理论上,</p>用于标记段落的结束,但是它很少被用到;大多数 HTML 作者甚至根本不知道它的存在。)最后,<hr>强制产生一个分断,并画一条跨越屏幕的水平线。

HTML 允许将图像内联到一个 Web 页面上。<img>标签指定了在页面的当前位置上显示一个图像。它可以有一些参数。src 参数给出了图像的 URL。HTML 标准没

有规定哪些图像格式是允许的。在实践中,所有的浏览器都支持 GIF 和 JPEG 文件。浏览器可以自由地支持其他的格式,但是这种扩展是双刃剑。例如,如果一个用户习惯于使用支持 BMP 文件的浏览器,那么,他可能在他的 Web 页面中包含 BMP 文件,可是以后他会惊奇地发现,其他的浏览器忽略了他所有的伟大艺术作品。

<img> 的其他参数还包括: align,它控制图像相对于文本基线的对齐方式(top[上对齐],middle[居中],bottom[下对齐]);alt,当用户禁用图像时,alt 参数中指定的文字可被显示出来以代替图像;ismap,它表示该图像是否是一个活动地图(即可单击的图片)。

最后,我们讨论超链接,它使用<a> (anchor,锚) 和</a> 标签。与<img>类似,<a> 也有一些参数,包括 href(超链接的 URL)和 name(超链接的名字)。<a> 和</a> 之间的文本被显示出来。如果这段文本被选中,则浏览器通过超链接转到一个新的页面上。在<a> 和</a> 之间也可以放一个<img>图像,在这种情况下,单击图像也可以激活超链接。

作为一个例子,请考虑下面的 HTML 片断:

```
<a href="http://www.nasa.gov"> NASA's home page </a>
```

当含有此片断的页面被显示时,出现在屏幕上的是:

NASA's home page

如果用户随后单击了此文本,则浏览器马上取回 URL 地址为 http://www.nasa.gov 的页面并显示出来。

作为第二个例子,现在来考虑:

```
<a href="http://www.nasa.gov">  </a>
```

当页面被显示时,它显示一幅图片(比如航天飞机的图片)。单击这幅图片就可以切换到 NASA 的主页,就如同在上一个例子中单击带下划线的文本一样。如果用户禁止了自动图像显示功能,则浏览器在图片的位置上显示文本 NASA。

<a> 标签也可以带一个 name 参数以便生成一个超链接,从而使得超链接可以指向一个页面的中间部分。例如,一些 Web 页面以一个可单击的内容表作为开头。通过单击内容表上的条目,用户可以跳转到页面中对应的部分。

HTML 一直在发展。HTML 1.0 和 HTML 2.0 不支持表格,但是在 HTML 3.0 中增加了表格。一个 HTML 表格包含一行或多行,每行由一个或多个单元格组成。单元格中可以包含各种各样的材料,包括文本、图像、图标、照片,甚至其他的表格。单元格可以被合并到一起,例如,一个标题可以横跨多列。页面的作者对于布局有有限的控制力,包括对齐、边界风格和单元格边距,但是浏览器在显示表格时有最终的决定权。

图 7.28(a) 列出了一个 HTML 表格的定义,图 7.28(b)给出了该表格一种可能的显示形式。这个例子只显示了 HTML 表格的一些基本特性。表格以<table>标签作为开始。另外一些信息被用来描述该表格的一般特性。

```

<html>
<head> <title> A sample page with a table </title> </head>
<body>
<table border=1 rules=all>
<caption> Some Differences between HTML Versions </caption>
<col align=left>
<col align=center>
<col align=center>
<col align=center>
<tr> <th>Item <th>HTML 1.0 <th>HTML 2.0 <th>HTML 3.0 <th>HTML 4.0 </tr>
<tr> <th> Hyperlinks <td> x <td> x <td> x <td> x </tr>
<tr> <th> Images <td> x <td> x <td> x <td> x </tr>
<tr> <th> Lists <td> x <td> x <td> x <td> x </tr>
<tr> <th> Active Maps and Images <td> &nbsp; <td> x <td> x <td> x </tr>
<tr> <th> Forms <td> &nbsp; <td> x <td> x <td> x </tr>
<tr> <th> Equations <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Toolbars <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Tables <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Accessibility features <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
<tr> <th> Object embedding <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
<tr> <th> Scripting <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
</table>
</body>
</html>

```

(a)

HTML 各版本之间的一些区别

项目	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0
超链接 (Hyperlink)	x	x	x	x
图像 (Image)	x	x	x	x
列表 (List)	x	x	x	x
活动地图和图像 (Active Map and Image)		x	x	x
表单 (Form)		x	x	x
等式 (Equation)			x	x
工具栏 (Toobar)			x	x
表格 (Table)			x	x
可达性 (Accessibility feature)				x
对象嵌入 (Object embedding)				x
脚本 (Scripting)				x

(b)

图 7.28

(a) 一个 HTML 表格; (b) 这个表格的一种可能的显示

<caption> 标签可被用来提供一个图表标题。表格中的每一行以一个<tr> (表格行) 标签作为开始。单独的单元格被标记为<th> (表格头) 或者<td> (表格数据)。之



所以作这样的区分,是为了让浏览器为表格头和表格数据使用不同的表现方式,就好像我们在例子中所做的那样。

在表格中还可以出现很多属性。它们包括指定单元格水平和垂直的对齐方式、单元格内部的对齐方式、边界、单元格的组合、单位以及其他更多的属性。

在 HTML 4.0 中,更多的新特性被加入进来。这些新特性包括为残疾用户设计的访问特性、对象嵌入(<img>标签的泛化,所以其他的对象也可以被嵌入到页面中)、支持脚本语言(因而允许动态的内容),等等。

当一个 Web 站点较为复杂,它的诸多页面由同一公司的多个作者分别编写时,人们往往希望可通过某一种方法来避免不同的页面有不同的外观。这个问题可以通过使用样式表(style sheet)来解决。当使用样式表时,每个单独的页面不再使用诸如粗体和斜体之类的物理样式。相反,页面的作者使用逻辑样式,例如<dn>(定义)、<em>(弱重点)、<strong>(强重点)和<var>(程序变量)。逻辑样式被定义在样式表中,每个页面的开始处引用样式表。通过这种方式,所有的页面有相同的样式,而且,如果 Web 站点管理员决定将<strong>从蓝色的 14 磅斜体字改变为鲜粉红色的 18 磅粗体字,那么,他只需改变一个定义就可以将整个 Web 站点转换过来。样式表与 C 程序中的#include 文件具有可比性:在头文件中改变一个宏定义,则所有包含此头文件的程序文件中的宏引用也随之而改变。

## 表单

HTML 1.0 基本上是单向的。用户可以向信息提供者请求页面,但是很难把信息发送回去。随着越来越多的商业组织开始使用 Web,双向通信的需求越来越大。例如,许多公司希望能够通过他们的 Web 页面来接受产品定单;而软件厂商则希望通过 Web 来分发软件,并且让客户以电子的方式来填写注册卡;提供 Web 搜索的公司则希望他们的客户能够输入搜索关键字。

这些需求导致了从 HTML 2.0 开始加入了表单(form)的功能。表单包含各种框或者按钮,它们允许用户输入信息或者做出选择,然后把信息送回给页面的所有者。<input>标签正是被用于这个目的。它使用很多参数来决定所显示的框的大小、性质和用法。最常见的形式是,可接受用户文本的空白域、可被选中的框、活动地图和提交(submit)按钮。图 7.29 中的例子展示了其中一些形式。

我们通过讲解这个例子来开始关于表单的讨论。与所有其他的表单一样,这个表单被包围在<form>和</form>标签之间。未被标签包围的文本仅仅被显示出来而已。所有常用的标签(例如<b>)都允许在表单中使用。在这个表单中使用了三种输入框。

第一种输入框出现在文本“Name”之后。这个框有 46 个字符宽并期待用户输入一个字符串,然后这个字符串被存储在 customer 变量中以便后而进一步处理。<p>标签指示浏览器在下一行上显示后面的文本和框,无论当前行上是否还有空间。通过使用<p>和其他的布局标签,页面的作者可以控制表单在屏幕上的外观。

表单的下一行要求输入用户的街道地址,它有 40 列宽,也是独自行。接着下面一行询问城市、州和国家。在这些域之间没有使用<p>标签,因此,如果放得下的话,浏览

```

<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/widgetorder" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street Address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size=4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>

```

(a)

(b)

图 7.29

(a) 一个定单的 HTML 描述；(b) 格式化后的页面

器会在一行上显示这些域。就浏览器而言,这个段落只是包含六个项:三个字符串与三个框交替出现。浏览器从左至右按照直线依次显示这六个项,如果当前行不能容纳下一项,则转到新的一行继续显示。因此,你可以想象在一个 1600×1200 的屏幕上,这三个字符串和它们对应的框会显示在同一行上,但是在一个 1024×768 的屏幕上它们可能会被分割到两行上。在最坏的情形下,“Country”这个词被显示在一行的末尾而它对应的框却在下一行的开始。

接下来一行要求输入信用卡号和失效日期。只有当采取了足够的安全手段以后,才可以在 Internet 上传输信用卡号。我们将在第 8 章中讨论一些相关的安全话题。

在失效日期之后我们遇到了一个新的特性:单选按钮(radio button)。当用户必须在两个或者更多的选项中选择其中之一时,则需要用到单选按钮。这里的选择模型有点类似于带有六个选台按钮的汽车收音机。浏览器在表单中显示这些框,用户通过单击它

们(或者使用键盘)来选择或者不选择它们。单击其中一个按钮的同时也会关掉同组中其他的选择。按钮的外观效果由浏览器决定。“Widget Size”也使用了两个单选按钮。这两个组是由它们的 name 域来区分的,而不是通过像 `< radiobutton > ... </radiobutton>` 这样划定的静态范围来分开。

value 参数用来指示哪个单选按钮被按下了。根据用户选择的是哪一个信用卡按钮,变量 cc 将被设置为字符串“mastercard”或者“visacard”。

在两组单选按钮之后是发货选项,它用复选框(checkbox)来表示。它既可以被选中,也可以未被选中。单选按钮必须恰好有一个被选中,而与此不同的是,每个 checkbox 类型的框都可以被选中,也可以未被选中,相互之间完全独立。例如,当通过 Electropizza 的 Web 页面来订购比萨饼时,用户可以选择沙丁鱼和洋葱以及菠萝(如果她能够忍受的话),但是她不能够为同一个比萨饼同时选择小号、中号和大号。比萨饼的辅料可通过三个单独的复选框来表示,而比萨饼的尺寸则通过一组单选框来表示。

顺带提一下,若要从一长串选项中作出一个选择,则单选框有点不方便。因此,HTML 提供了 `<select>` 和 `</select>` 标签,以便将可选项列表括起来,这种做法与单选按钮具有同样的语义(除非给出 multiple 参数,在这种情况下它的语义与复选框一样)。一些浏览器把 `<select>` 和 `</select>` 之间的选项显示为下拉菜单。

我们现在已经看到了 `<input>` 标签的两种内置类型: radio 和 checkbox。实际上,我们也已经看到了第三种: text。因为这种类型是默认类型,所以,我们没有必要加上参数 `type=text`,但是我们可以这样做。另外两种类型是 password(口令)和 textarea(文本区域)。password 框与 text 框是一样的,只不过当用户输入字符时,所输入的字符不被显示出来。textarea 框也与 text 框一样,只不过它可以容纳多行。

回到图 7.29 的例子中,下面我们介绍 submit(提交)按钮。当这个按钮被单击时,表单上的用户信息被送回到提供此表单的机器上。与所有其他的类型一样,submit 是一个保留字,浏览器能够理解它。这里的 value 字符串是被显示在按钮上的标注。所有的框都可以有 value 参数;但是只有在这里我们才真正需要这个属性。对于 text 框,value 域的内容会随着表单一起被显示出来,但是用户可以编辑和删除它。checkbox 框和 radio 框也可以被初始化,但是要通过一个名为 checked 的域(因为 value 域只能给出文本,但不能表示一个优先选择项)。

当用户单击 submit 按钮时,浏览器将收集到的信息打包到一个长长的单行中,然后送回给服务器进行处理。“&”被用来分割各个域,“+”被用来表示空格。就我们的表单例子而言,这一行的内容看上去如图 7.30 所示(由于页面不够宽因此分成了三行)。

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&
state=NY&country=USA&cardno=1234567890&expires=6/98&cc=mastercard&
product=cheap&express=on
```

图 7.30 从浏览器到服务器的一种可能应答(其中携带了用户填写的信息)

这个字符串必须以一行的形式,而不是三行的形式被送回给服务器。如果一个 checkbox 框没有被选中,则它在字符串中被省略。字符串的含义由服务器来解释。我们

在本章的后面将讨论服务器如何完成这项工作。

## XML 和 XSL

HTML, 无论是否包含表单, 都不能给 Web 页面提供任何结构。它把内容和格式信息混合在一起。随着电子商务和其他应用越来越普及, 对 Web 页面进行结构化以及把内容从格式中分离出来的需求也在不断增长。例如, 如果一个程序想要通过搜索 Web 的方式来得到某些书或者 CD 的最低价格, 那么它需要分析很多 Web 页面, 查看每一项的标题和价格。对于 HTML 格式的 Web 页面, 一个程序很难知道标题在哪儿、价格在哪儿。

由于这个原因, W3C 对 HTML 进行了增强, 以使得 Web 页面可被结构化, 因而便于自动处理。有两种新的语言已经被开发出来用于这个目的。首先, XML (eXtensible Markup Language, 可扩展标记语言) 用一种结构化的方式来描述 Web 内容; 其次 XSL (eXtensible Style Language, 可扩展样式语言) 以一种独立于内容的方式来描述格式。这两者都是既庞大而又复杂的话题, 因此我们下面的简短介绍只是粗浅和表面的, 但是通过这些介绍, 你应该可以了解它们是如何工作的。

考虑图 7.31 中的 XML 文档的例子。它定义了一个称为 book\_list 的结构, 这代表了书的列表。每本书有三个域: title(标题)、author(作者)和 year(出版年份)。这些结构特别简单。在结构中也允许出现重复的域(比如多个作者)、可选域(比如, 附带 CD-ROM 的标题)和选择性的域(比如, 如果书还在销售中则是书店的 URL, 或者如果书已绝版则是一个拍卖站点的 URL)。

在这个例子中, 每一个域本身是一个完整的实体, 但是, 再进一步细分这些域也是可以的。例如, author 域可以被分割成下面的形式, 以便为搜索和格式化提供更细粒度的控制。

```
<author>
  <first_name> Andrew </first_name>
  <last_name> Tanenbaum </last_name>
</author>
```

每个域可以被划分成子域, 子域再被划分成子子域, 如此可以划分至任意深度。

图 7.31 中的文件所做的事情仅仅是定义了一个包含三本书的图书列表。它没有描述如何在屏幕上显示该 Web 页面。为了提供格式信息, 我们需要另外一个包含 XSL 定义的文件 book\_list.xsl。这个文件是一个样式表, 它描述了如何显示这个页面。(除了使用样式表以外也还有其他的替代办法, 比如将 XML 转换成 HTML, 但是这些替代办法超出了本书的讨论范围)。

如图 7.32 所示的 XSL 示例文件可被用来格式化图 7.31 中的 XML 文件。首先, 该文件包含一些必要的声明, 这些声明指定了 XSL 标准的 URL, 然后它也以 <html> 和 <body> 标签作为开始。这两个标签像往常一样定义了此 Web 页面的开始。接下来是一个表格定义, 其中包含三个列的标题。请注意, 除了 <th> 标签以外, 这里还有 </th> 标签, 关于这两个标签我们到目前为止还没有讨论过。XML 和 XSL 规范比 HTML 规范

```

<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="book_list.xsl"?>

<book_list>
  <book>
    <title> Computer Networks, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2003 </year>
  </book>
  <book>
    <title> Modern Operating Systems, 2/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2001 </year>
  </book>
  <book>
    <title> Structured Computer Organization, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 1999 </year>
  </book>
</book_list>

```

图 7.31 一个用 XML 来表示的简单 Web 页面

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">

<html>
<body>

<table border="2">
  <tr>
    <th> Title</th>
    <th> Author</th>
    <th> Year </th>
  </tr>

  <xsl:for-each select="book_list/book">
    <tr>
      <td> <xsl:value-of select="title"/> </td>
      <td> <xsl:value-of select="author"/> </td>
      <td> <xsl:value-of select="year"/> </td>
    </tr>
  </xsl:for-each>
</table>

</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

图 7.32 XSL 格式的样式表

要严格得多。它们声明,当浏览器面对语法不正确的文件时,即使它能够判断出 Web 设计者要表达的含义,它也必须拒绝这样的文件。如果浏览器接受一个语法错误的 XML 或 XSL 文件,并且自己修补了其中的错误,那么该浏览器违反了标准中的规定,在一致性测试中它将被拒绝。然而,浏览器可以精确地指出错误的位置。为了对付当前大量粗糙的 Web 页面,这种较为严格的措施是非常有必要的。

语句:

```
<xsl:for each select="book_list/book">
```

与 C 语言中的 for 语句非常类似。它使浏览器重复执行循环体(以</xsl:for-each>作为结束),每本书执行一次。每次循环输出五行:<tr>,标题,作者和年份,然后是</tr>。在循环之后,输出结束标签</body>和</html>。浏览器解释这个样式表所得到的结果与内含此表格的 Web 页面是完全相同的。然而,在这种格式中,程序可以分析 XML 文件并且很容易就可以找到 2000 年以后出版的书。值得强调的是,虽然我们的 XSL 文件中包含了一种循环,但是,用 XML 和 XSL 编写的 Web 页面仍然是静态的,因为它们仅仅包含了一些指示浏览器如何显示页面的指令,就如同 HTML 页面一样。当然,为了使用 XML 和 XSL,浏览器必须能够解释 XML 和 XSL,但是现在的大多数浏览器都有这种能力。至于 XSL 是否会取代传统的样式表,目前还不清楚。

我们还没有介绍如何来做到这一点,但是 XML 允许 Web 站点设计者充分使用一些定义文件。这些定义文件已经预先定义了一些结构。通过包含这些定义文件,就有可能构造出非常复杂的 Web 页面。关于其他更多这方面的信息以及 XML 与 XSL 的其他特性,可以参考有关此类主题的书,其中有两本书是(Livingston, 2002; and Williamson, 2001)。

在结束关于 XML 和 XSL 的讨论之前,值得评论一下 WWW 联盟和 Web 设计者团体之间的一场意识形态方面的论战。HTML 的最初目标是指定文档的结构,而不是它的外观。例如:

```
<h1> Deborah's Photos </h1>
```

指示浏览器强调这个标题,但是并没有说明任何关于字体、磅尺寸或颜色等方面的信息。这些因素留给浏览器来决定,因为浏览器知道显示器的特性(例如,它有多少像素)。然而,许多 Web 页面设计者希望对他们设计的页面的显示外观有绝对的控制,所以一些新的标签被加入到 HTML 中以便控制相应的显示外观,例如:

```
<font face="helvetica" size="24" color="red"> Deborah's Photos </font>
```

而且,HTML 中也加入了一些方法来精确地控制在屏幕上的显示位置。这种做法的麻烦之处在于它不是可移植的。尽管一个页面在创建者的浏览器中有可能显示得非常完美,但是在另一个浏览器,或者同一个浏览器的另一个版本,或者不同的屏幕分辨率中,它可能是一片混乱。从某种程度上来讲,XML 试图向原始目标回归,即只指定文档的结构而不指定它的外观。然而,通过使用 XSL 仍然可以管理外观的显示。但是这两种格式可能会被误用。你需要留意这一点。



除了用于描述 Web 页面以外,XML 也可以被用于其他的用途。一个正在日益显著的用途是,XML 被当作应用程序之间的通信语言。特别地,SOAP(Simple Object Access Protocol,简单对象访问协议)是一种在应用之间执行 RPC(远过程调用)的方法,并且与具体的语言和系统无关。客户按照 XML 消息的格式来构造一个请求,并利用 HTTP 协议(在后面描述)将它发送给服务器。服务器同样以 XML 格式的消息送回一个回复。通过这种方式,异构平台上的应用就可以相互通信了。

### XHTML—扩展的超文本标记语言

HTML 不断地演进以满足新的需求。工业界的许多人认为,在将来,大多数支持 Web 访问的设备不是 PC 机,而是无线的、手持的 PDA 类型的设备。相对于那些能够采用启发式方法来处理网页语法错误的大型浏览器而言,这些设备的内存非常有限。因此,HTML 4 的下一步是一种十分挑剔的语言,它被称为 XHTML(eXtended HyperText Markup Language,扩展的超文本标记语言),而不是 HTML 5,因为它本质上是重新用 XML 来表示的 HTML 4。这时,我们认为像<h1>这样的标签没有任何内在的含义。但为了达到 HTML 4 的效果,在 XSL 文件中需要有一个定义。XHTML 是新的 Web 标准,它应该被用于所有新的 Web 页面,以便达到最大程度的跨平台和跨浏览器的可移植性。

XHTML 和 HTML 4 有六个主要的差别,以及许多小的差异。我们现在来看一看这些主要的差别。首先,XHTML 页面和浏览器必须严格地遵守标准。从而再也没有劣质的 Web 页面了。这种特性是从 XML 继承而来的。

其次,所有的标签和属性必须是小写的。像<HTML>这样的标签在 XHTML 中是无效的。而使用像<html>这样的标签则是强制性的。类似地,<img SRC="pic001.jpg">这样的写法也是被禁止的,因为它含有大写字母的属性。

第三,结束标签是必需的,即使</p>也不例外。对于那些天生没有结束标签的标签,例如<br>、<hr>和<img>,则在最后的">"之前必须有一个斜线,例如:

```

```

第四,属性必须被包含在引号之中。例如:

```
<img SRC="pic001.jpg" height=500 />
```

不再是允许的。尽管 500 只是一个数,但也必须被放在引号之中,就像 JPEG 文件的名字一样。

第五,标签的嵌套必须是正确的。在过去,只要最后到达的状态是正确的,则嵌套的正确与否并没有要求。例如:

```
<center> <b> Vacation Pictures </center> </b>
```

过去是合法的,而在 XHTML 中就不再合法了。标签的关闭顺序必须与打开的顺序相反。

第六,每个文档必须指明它的文档类型,正如我们在图 7.32 中所看到的。关于所有

这些变化的讨论,无论是主要的还是次要的,请参见 [www.w3.org](http://www.w3.org)。

### 7.3.3 动态 Web 文档

到目前为止,我们使用的还是图 6.6 所示的模型:客户向服务器发送一个文件名,然后服务器返回这个文件。在 Web 发展的早期,所有的内容实际上都像这样是静态的(仅仅是文件而已)。然而,近年来,越来越多的内容变成动态的,也就是说,它们不是被存储在磁盘上,而是根据需要再生成。内容的生成既可以发生在服务器端,也可以发生在客户端。下面我们将依次讨论这两种情况。

#### 服务器端动态 Web 页面的生成

为了看清楚为什么需要在服务器端生成内容,请考虑上一小节所述的表单的用法。当一个用户填完一个表单然后单击 submit 按钮时,一条消息被发送给服务器,该消息中包含了表单的内容和用户填写的域信息。该消息不同于以前我们看过的指定一个文件名的消息。服务器所需要做的是,把消息交给一个程序或者脚本来处理。在处理过程中,通常会涉及到:利用用户提供的信息在服务器磁盘上的数据库中查找一条记录,然后生成一个定制的 HTML 页面返回给客户。例如,在电子商务应用中,当用户单击了“PROCEED TO CHECKOUT(前往结账)”以后,浏览器将包含购物车内容的 cookie 返回给服务器,然后,在服务器上,必须要调用某个程序或脚本来处理这个 cookie,并生成一个 HTML 页面作为响应。这个 HTML 页面可能会显示一个表单,其中包含了购物车中项目的列表、用户最近提供的送货地址,同时还要求用户核实这些信息并选择付款方式。处理 HTML 表单信息所要求的步骤如图 7.33 所示。



图 7.33 处理 HTML 表单信息的步骤

处理表单和其他交互式 Web 页面的传统方法是一种称为 CGI (Common Gateway Interface, 公共网关接口) 的系统。它是一个标准化的接口,允许 Web 服务器与后端程序及脚本进行通信,这些后端程序和脚本能够接受输入信息(例如,来自表单),并生成 HTML 页面作为响应。通常,这些后端程序是用 Perl 脚本语言编写的,因为比起一般的程序来说,Perl 脚本写起来既容易又快速(不过,起码你要懂得如何用 Perl 语言来编程)。习惯上,这些脚本被放在一个称为 cgi-bin 的目录中,用户在 URL 中可以看到这个目录。有时候也使用另一种脚本语言 Python 来代替 Perl。

我们来看一看 CGI 通常是如何工作的,作为一个例子,请考虑这样的情况: Truly Great Products 公司的一件产品没有质量保证登记卡。相反,顾客被告知:到 [www.tgpc.com](http://www.tgpc.com) 进行在线注册。在这个页面上有一个超级链接是这样显示的:

Click here to register your product (单击这儿注册你的产品)

此链接指向一个 Perl 脚本, 比如说 `www.tgpc.com/cgi-bin/reg.perl`。如果这个脚本被调用时无任何参数, 则它送回的 HTML 页面包含了注册表单。当用户填写了这个表单并单击 submit 按钮时, 一条包含了用户所填写的值、并且如图 7.30 所示风格的消息被送回给这个脚本。然后此 Perl 脚本对参数进行解释, 并且为这个新顾客在数据库中建立一条记录, 再送回一个包含注册号和服务电话号码的 HTML 页面。这不是处理表单的惟一方法, 但它是一种很常见的方法。有关编写 CGI 脚本和 Perl 编程的书籍非常多, 包括 (Hanegan, 2001; Lash, 2002; and Meltzer and Michalski, 2001)。

CGI 脚本并不是在服务器端生成动态内容的惟一方法。另外一种常见的方法是在 HTML 页面中嵌入少量的脚本, 然后让服务器来执行这些脚本以便生成最终发送给客户的页面。一种流行的编写这些脚本的语言是 PHP (PHP: Hypertext Preprocessor, 超文本预处理器)。为了使用 PHP, 服务器必须能够理解 PHP (就好像浏览器必须能够理解 XML, 才可以解释用 XML 编写的 Web 页面一样)。通常, 服务器要求包含 PHP 的 Web 页面的文件扩展名是 php, 而不是 html 或者 htm。

图 7.34 展示了一小段 PHP 脚本。在任何一台安装了 PHP 的服务器上, 它都应该可以工作。除了在 `<? php ... ? >` 标签内部的 PHP 脚本以外, 它包含的其他内容都是正常的 HTML。它所做的工作是生成一个 Web 页面, 并且在此 Web 页面中描述了它所了解到的关于对方浏览器的信息。浏览器通常在发送请求 (和任何适当的 cookie) 的同时也会发送其他某些信息, 这些信息被放在变量 `HTTP_USER_AGENT` 中。当图 7.34 中的代码被放到 ABCD 公司的 WWW 目录中的文件 `test.php` 中时, 用户一旦输入 URL `www.abcd.com/test.php`, 则他看到的 Web 页面就会显示出有关他所用的浏览器、语言和操作系统之类的信息。

```
<html>
<body>

<h2> This is what I know about you </h2>
<?php echo $HTTP_USER_AGENT ?>

</body>
</html>
```

图 7.34 一个内嵌 PHP 的示例 HTML 页面

PHP 特别擅长于处理表单, 而且用起来比 CGI 简单得多。图 7.35(a) 显示了它如何处理表单的一个例子。此图中包含了一个内有表单的普通 HTML 页面。它惟一不寻常的地方是它的第一行, 该行指明了当用户完成输入并提交表单后应该调用 `action.php` 文件来处理参数。该页面显示两个文本框, 一个要求输入名字, 另一个要求输入年龄。当用户填写了这两个框并且表单被提交后, 服务器对客户送回来的图 7.30 类型的字符串进行解析, 并且把名字放到 `name` 变量中, 年龄放到 `age` 变量中。然后, 作为回应, 服务器开始处理 `action.php` 文件, 如图 7.35(b) 所示。在处理这个文件的过程中, PHP 命令被执行。如果用户在框中输入的是 “Barbara” 和 “24”, 那么被送回来的 HTML 文件将如图 7.35(c) 所

示。因此,使用 PHP 以后,处理表单变得非常简单。

```
<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(a)

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```

(b)

```
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 25
</body>
</html>
```

(c)

图 7.35

(a) 一个包含表单的 Web 页面; (b) 一段处理表单输出的 PHP 脚本;  
(c) 当输入分别是“Barbara”和 24 时 PHP 脚本的输出

尽管 PHP 非常易于使用,但它实际上是一种面向 Web 和服务器数据库之间的交互过程的、功能强大的程序设计语言。它具有变量、字符串、数组,以及在 C 语言中可以找得到的绝大多数控制结构,而且它也有强大的 I/O 功能(远远超出了 printf)。PHP 是开放源码的,可以免费获取。PHP 被经过了特殊的设计,以便与 Apache 服务器更好地协同工作,而 Apache 也是开放源码的,并且是世界上使用最为广泛的 Web 服务器。要了解 PHP 的更多信息,请参看(Valade, 2002)。

我们现在已经看到了两种不同的生成动态 HTML 页面的方法: CGI 脚本和内嵌的 PHP。此外,还有第三种技术,称为 JSP(JavaServer Pages, Java 服务器页面),它与 PHP 非常相似,只不过页面中的动态部分是用 Java 编程语言而不是 PHP 来编写的。使用这种技术的页面文件的扩展名为 jsp。第四种技术,ASP(Active Server Page, 活动的服务器页面)是 Microsoft 版本的 PHP 和 JSP。它使用 Microsoft 私有的脚本语言 Visual Basic Script 来生成动态内容。使用这种技术的页面文件的扩展名为 asp。到底选择 PHP、JSP 还是 ASP 通常更多地与策略(开放源码? Sun 公司? Microsoft 公司?)相关而并非取决

于技术,因为这三种语言基本上差不多。

所有的生成动态内容的技术合起来有时候被称为动态 HTML(dynamic HTML)。

#### 客户端动态网页的生成

CGI、PHP、JSP 和 ASP 脚本解决了处理表单以及与服务器上的数据库进行交互的问题。它们都可以接受来自表单的信息,在一个或多个数据库中查找信息,然后利用查找的结果生成 HTML 页面。它们所不能做的是响应鼠标移动事件,或者直接与用户交互。为了达到这个目的,有必要在 HTML 页面中嵌入脚本,而且这些脚本在客户机上被执行而不是在服务器上。从 HTML 4.0 开始,可以通过<script>标签来使用这样的脚本。最流行的客户端脚本语言是 JavaScript,所以我们现在简短地看一看 JavaScript。

JavaScript 是一种脚本语言,粗略地说,它受到了 Java 程序设计语言中一些思想的启发。但它绝对不是 Java。与其他的脚本语言类似,它是一种非常高级的语言。例如,仅仅一行 JavaScript 代码就可以弹出一个对话框、等待文本输入,然后将结果字符串存储到一个变量中。像这种高级的特性使得 JavaScript 非常适合于设计交互式的 Web 页面。另一方面,由于它没有被标准化,而且它的变异比用 X 光机捕捉到的果蝇还要快,所以,要想编写出在所有平台上都能工作的 JavaScript 程序非常困难,但也许有一天它会稳定下来。

作为 JavaScript 程序的一个例子,请看图 7.36 中的程序。与图 7.35 类似,它显示了一个表单要求输入名字和年龄,并且预测这个人明年的年龄。主体部分基本上与 PHP 例子相同,主要的区别在于 submit 按钮的声明,以及其中的赋值语句。赋值语句告诉浏览器,当按钮被单击时请调用 response 脚本,并且将表单作为一个参数传递给它。

这里全新的是在 HTML 文件头部的 JavaScript 函数 response 的声明,HTML 文件的头部通常是标题、背景颜色等保留的。这个函数从表单的 name 域中提取出相应的值,并将它当作一个字符串存放在 person 变量中。它也提取出 age 域的值,并且使用 eval 函数将该值转化为一个整数,然后加 1 并把结果存放在 years 变量中。接着,它打开一个用于输出的文档,并且利用 writeln 方法向这个文档做四次写操作,然后关闭文档。从该文档中的各种 HTML 标签可以看出,它是一个 HTML 文件。然后浏览器在屏幕上显示这个文档。

图 7.35 和图 7.36 看上去相似但是它们的处理方式却完全不同,理解这一点非常重要。在图 7.35 中,当用户单击了 submit 按钮以后,浏览器将表单中的信息收集到一个如图 7.30 所示的风格的长字符串中,然后将它送回给对应的服务器,即发送此页面的那个服务器。服务器看到这个 PHP 文件的名称,然后执行该文件。PHP 脚本产生一个新的 HTML 页面,然后该页面被送回给浏览器以便显示出来。在图 7.36 中,当 submit 按钮被单击时,浏览器解释执行该页面上包含的 JavaScript 函数。所有的工作都在本地的浏览器内部完成。浏览器并没有与服务器联系。因此,JavaScript 函数的结果基本上立刻被显示出来,而如果使用 PHP 的话,则在结果 HTML 到达客户端以前可能有几秒钟的延迟。服务器端脚本和客户端脚本之间的区别及相应的执行步骤如图 7.37 中所示。在两种情况下,列出的步骤都从表单被显示出来之后开始编号。步骤 1 为接受用户输入。

```

<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
    var person = test_form.name.value;
    var years = eval(test_form.age.value) + 1;
    document.open();
    document.writeln("<html> <body>");
    document.writeln("Hello " + person + " <br>");
    document.writeln("Prediction: next year you will be " + years + " years old.");
    document.writeln("</body> </html>");
    document.close();
}
</script>
</head>
<body>
<form>
Please enter your name: <input type="text" name="name">
<p>
Please enter your age: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)">
</form>
</body>
</html>

```

图 7.36 使用 JavaScript 来处理表单

接下来是对用户输入的处理,这一步骤在两种情况下并不相同。

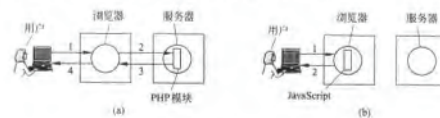


图 7.37

(a) 使用 PHP 的服务器端脚本; (b) 使用 JavaScript 的客户端脚本

这种区别并不意味着 JavaScript 比 PHP 更好。它们的用途完全不同。PHP(同时也暗示着 JSP 和 ASP)是在需要与远程的数据库进行交互时才被使用;而 JavaScript 则是当需要在客户计算机上与用户交互时才被使用。HTML 页面有可能(也通常)既使用 PHP 也使用 JavaScript,不过,PHP 和 JavaScript 不可能完成同样的工作,或者拥有同一个按钮。

JavaScript 是一门全功能的程序设计语言,具有 C 或者 Java 的所有能力。它有变量、字符串、数组、对象、函数和所有常用的控制结构。它还包含大量的专为针对 Web 页面的设施,包括管理窗口和框架的能力,设置和获取 cookie、处理表单、处理超链接等。图 7.38 给出了一个使用递归函数的 JavaScript 程序的例子。



```

<html>
<head>
<script language="javascript" type="text/javascript">

function response(test_form) {
    function factorial(n) {if (n == 0) return 1; else return n * factorial(n - 1);}
    var r = eval(test_form.number.value);    // r = typed in argument
    document.myform.mytext.value += "Here are the results.\n";
    for (var i = 1; i <= r; i++)              // print one line from 1 to r
        document.myform.mytext.value += (i + "! = " + factorial(i) + "\n");
    }
}
</script>
</head>

<body>
<form name="myform">
Please enter a number: <input type="text" name="number">
<input type="button" value="compute table of factorials" onclick="response(this.form)">
<p>
<textarea name="mytext" rows=25 cols=50> </textarea>
</form>
</body>
</html>

```

图 7.38 一个计算和打印阶乘的 JavaScript 程序

JavaScript 也可以跟踪鼠标在屏幕中对象上的运动。许多含有 JavaScript 的 Web 页面具有这样的特性：当鼠标光标移动到一些文字或图像上时可以触发一些动作。通常是改变图像或者突然出现一个菜单。这种行为在 JavaScript 中很容易编写，从而实现生动的 Web 页面。图 7.39 给出了一个例子。

```

<html>
<head>
<script language="javascript" type="text/javascript">
if (!document.myurl) document.myurl = new Array();
document.myurl[0] = "http://www.cs.vu.nl/~ast/im/kitten.jpg";
document.myurl[1] = "http://www.cs.vu.nl/~ast/im/puppy.jpg";
document.myurl[2] = "http://www.cs.vu.nl/~ast/im/bunny.jpg";
function pop(m) {
    var urx = "http://www.cs.vu.nl/~ast/im/cat.jpg";
    popupwin = window.open(document.myurl[m], "mywind", "width=250,height=250");
}
</script>
</head>

<body>
<p> <a href="#" onMouseover="pop(0); return false;" > Kitten </a> </p>
<p> <a href="#" onMouseover="pop(1); return false;" > Puppy </a> </p>
<p> <a href="#" onMouseover="pop(2); return false;" > Bunny </a> </p>
</body>
</html>

```

图 7.39 响应鼠标移动的交互式 Web 页面

JavaScript 不是使 Web 页面具有高度交互性的惟一方法。另外一种流行的方法是使用 **applet**。所谓 **applet**，是指已经被编译成 **JVM(Java Virtual Machine, Java 虚拟机)** 机器

指令的 Java 小程序。Applet 可以被嵌入到 HTML 页面中(在<applet>和</applet>之间),并被具有 JVM 能力的浏览器解释执行。因为 Java applet 被解释执行而不是被直接执行,所以,Java 解释器可以防止它们做坏事。至少在理论上是这样的。然而在实践中,编写 applet 的人已经发现了,在 Java I/O 库中有无穷无尽的错误(bug)可被发掘和利用。

Microsoft 对于 Sun 的 Java applet 的回应是,允许 Web 页面包含 **ActiveX 控件**(ActiveX control),所谓 ActiveX 控件,是指一种已经被编译成 Pentium 机器指令的程序,它们可以直接在硬件上执行。这种特性使得 ActiveX 控件比解释执行的 Java applet 更快更灵活,因为它可以完成任何普通程序能够做到的事情。当 Internet Explorer 浏览器在 Web 页面中发现一个 ActiveX 控件时,它会下载这个控件,检验它的身份然后执行该控件。然而,下载和运行外部程序将会导致安全问题,我们将在第 8 章中讨论这些安全问题。

由于几乎所有的浏览器都可以解释 Java 程序和 JavaScript,所以如果设计者希望制作出具有高度交互性的 Web 页面,则他至少有两种技术可供选择,另外,如果多平台的移植性不是一个问题,则也可以考虑使用 ActiveX 技术。作为一般性的规则,通常的情况是,JavaScript 比较容易编写,Java applet 执行得更快一些,而 ActiveX 控件是运行速度最快的。而且,由于所有的浏览器实现了完全相同的 JVM,但是没有两个浏览器实现了相同版本的 JavaScript,所以,Java applet 比 JavaScript 程序具有更好的移植性。要想了解关于 JavaScript 的更多信息,有很多书可以参考,每一本都有很多页(通常超过 1000 页),例如(Easttom, 2001; Harris, 2001; and McFedries, 2001)。

在结束动态 Web 内容这个主题之前,让我们简单地总结一下到现在为止所讨论过的内容。完整的 Web 页面可以通过服务器上的各种脚本来动态地生成。一旦这些 Web 页面已经被浏览器接收下来,则它们将被当作普通的 HTML 页面来对待并显示出来。如图 7.40 所示,服务器端的脚本可以用 Perl、PHP、JSP 或者 ASP 来编写。

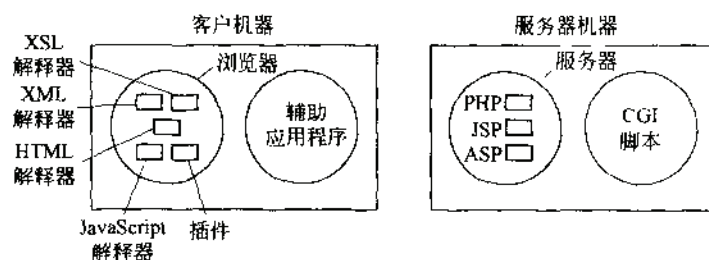


图 7.40 生成和显示内容的各种不同方法

在客户端也可以实现动态内容生成。首先用 XML 来编写 Web 页面,然后浏览器根据 XSL 文件将页面转换为 HTML。JavaScript 程序可以完成任意的计算任务。最后,插件和辅助应用程序可以被用来显示各种格式的内容。

#### 7.3.4 HTTP—超文本传输协议

在整个 WWW 上使用的传输协议是 HTTP (HyperText Transfer Protocol, 超文本传输协议)。它指定了客户可以向服务器发送什么样的消息, 并且得到什么样的回应消息。每次交互由一个 ASCII 请求组成, 随后是一个类似于 RFC 822 MIME 的回应。所有的客户和服务器都必须遵守这个协议, 它被定义在 RFC 2616 中。在这一小节中, 我们将讨论它的一些重要特性。

##### 连接

对于浏览器来说, 为了与服务器联系, 最常用的方法是与服务器机器上的 80 端口建立一个 TCP 连接, 不过, 这个过程并不是正式要求的。使用 TCP 的意义在于, 浏览器和服务器都不需要担心消息丢失、重复消息、长消息以及消息确认等问题。所有这些问题都由 TCP 协议来处理。

在 HTTP 1.0 中, 当连接被建立起来以后, 浏览器发送一个请求, 之后一个回应消息被送回来。然后 TCP 连接被释放。在早期, Web 页面通常只包含 HTML 文本, 在这种情况下, 这样的做法已经足够了。但是, 最近几年来, 通常 Web 页面还含有大量的图标、图像以及其他很养眼的内容, 所以建立一个 TCP 连接来仅仅传输一个图标, 则显得代价太昂贵了。

这种现象导致产生了 HTTP 1.1, 它支持持续连接 (persistent connection)。通过这种连接, 就有可能在建立一个 TCP 连接以后, 发送请求并得到回应, 然后发送更多的请求并得到更多的回应。通过把建立和释放 TCP 连接的开销分摊到多个请求上, 则对于每个请求而言, 由于 TCP 而造成的相对开销被大大地降低了。而且, 还可以发送流水线请求, 也就是说在请求 1 的回应到来之前就发送请求 2。

##### 方法

尽管 HTTP 是为了用于 Web 而设计的, 但为了着眼于未来的面向对象应用, 它被有意识地设计得比 Web 所需要的更加通用。由于这个原因, HTTP 并不仅仅只支持请求一个 Web 页面, 而是支持更加通用的操作, 也被称为方法 (method)。也正是这种一般性才使得 SOAP 得以存在。每个请求包含一行或多行 ASCII 文本, 其中第一行的第一个词是被请求的方法的名字。图 7.41 中列出了内置的方法。要想访问一般的对象, 也可以使用与对象相关的其他方法。方法名区分大小写, 所有 GET 是合法的方法而 get 不是。

GET 方法请求服务器发送页面 (按照最为一般化的情形, 我们是指对象, 但是在实践中, 通常只是一个文件)。该页面以 MIME 适当地编码。大部分发送给 Web 服务器的请求都是 GET 方法。GET 的常见形式是:

```
GET filename HTTP/1.1
```

这里的 filename 是要被取回来的资源 (文件) 的名字, 1.1 是所用的协议的版本号。

方法	说明
GET	请求读取一个 Web 页面
HEAD	请求读取一个 Web 页面的头部
PUT	请求存储一个 Web 页面
POST	附加一个命名的资源(例如 Web 页面)
DELETE	移除 Web 页面
TRACE	送回收到的请求
CONNECT	保留作将来使用
OPTIONS	查询特定选项

图 7.41 内置的 HTTP 请求方法

HEAD 方法只是请求消息头,而不是真正的页面。利用这个方法可以得到一个页面的最后修改时间,也可以收集各种为建立索引面需要的信息,或者只是测试一下 URL 的有效性。

PUT 方法与 GET 方法相反:它不是读取页面,而是写入页面。通过这个方法可以在远程服务器上建立起一组 Web 页面。这个请求的主体包含了页面。页面有可能利用 MIME 来编码,在这种情况下,跟在 PUT 后面的行可能包含 Content-Type 和认证头,通过认证头可以证明呼叫者确实有权执行所请求的操作。

POST 方法与 PUT 方法有点类似。它也携带一个 URL,但不是替换掉原来的数据,而是(按照泛化意义上的说法)将新的数据“附加”到原来的数据后面,比如说,向新闻组张贴一条消息,或者向公告牌系统添加一个文件。在实践中,PUT 和 POST 方法都用得不多。

DELETE 方法的用途可能与你想象的一样:移除页面。与 PUT 方法类似,认证和许可机制在这里起到了很重要的作用。DELETE 方法并不保证一定会成功,因为即使远程的 HTTP 服务器愿意删除这个页面,但是,该页面文件的访问模式可能禁止 HTTP 服务器修改或者删除它。

TRACE 方法用于调试。它指示服务器送回收到的请求。当请求没有被正确地处理而客户希望知道服务器实际得到的是什么样的请求时,这个方法将非常有用。

CONNECT 方法现在没有使用,它被保留作将来使用。

OPTIONS 方法提供了一种办法来让客户向服务器查询它的属性,或者查询某个特定文件的属性。

每个请求得到一个回应,在回应消息中包括一个状态行,可能还有附加的信息(例如全部或者部分 Web 页面)。状态行包括一个 3 位数字的状态码,该状态码指明了这个请求是否被满足,如果没有满足的话则原因是什么。如图 7.42 所示,第一个数字被用于把回应分成 5 个大组。1xx 码很少被用于实践中。2xx 码意味着这个请求被成功地处理,相应的内容(如果有的话)被返回。3xx 码告诉客户应该检查其他的位置:或者使用另一个不同的 URL,或者在它自己的缓存中查找(后面讨论)。4xx 码意味着由于客户的错误

而请求失败,比如无效请求或者不存在的页面。最后,5xx 码意味着服务器自身有问题,有可能是服务器代码中有错误,也可能是临时负载过重。

状态码	含义	例子
1xx	信息	100=服务器同意处理客户的请求
2xx	成功	200=请求成功;204=没有内容存在
3xx	重定向	301=页面移动了;304=缓存的页面仍然有效
4xx	客户错误	403=禁止的页面;404=页面未找到
5xx	服务器错误	500=服务器内部错误;503=以后再试

图 7.42 回应消息中状态码的分组表示

### 消息头

请求行(例如 GET 方法的行)后面可能跟着附加的行,其中包含更多的信息。它们被称为请求头(request header)。这些信息可以与一个过程调用的参数相类比。回应也可能有回应头(response header)。有些头可以在两个方向上使用。图 7.43 中列出的是一些最重要的消息头。

头	类型	内容
User-Agent	请求	关于浏览器和它的平台的信息
Accept	请求	客户能处理的页面的类型
Accept-Charset	请求	客户可以接受的字符集
Accept-Encoding	请求	客户能处理的页面编码方法
Accept-Language	请求	客户能处理的自然语言
Host	请求	服务器的 DNS 名字
Authorization	请求	客户的信任凭据的列表
Cookie	请求	将一个以前设置的 cookie 送回给服务器
Date	双向	消息被发送时的日期和时间
Upgrade	双向	发送方希望切换到的协议
Server	回应	关于服务器的信息
Content-Encoding	回应	内容是如何被编码的(例如 gzip)
Content-Language	回应	页面所使用的自然语音
Content-Length	回应	以字节计算的页面长度
Content-Type	回应	页面的 MIME 类型
Last-Modified	回应	页面最后被修改的时间和日期
Location	回应	指示客户将请求发送到别处的命令
Accept-Ranges	回应	服务器将接受指定了字节范围的请求
Set-Cookie	回应	服务器希望客户保存一个 cookie

图 7.43 一些 HTTP 消息头

User-Agent 头允许客户将它的浏览器、操作系统和其他属性信息告知服务器。在图 7.34 中,我们看到了服务器魔术般地拥有了这些信息,并且能够在 PIIP 脚本中根据需要产生这样的信息。客户利用这个头向服务器提供此类信息。

四个 Accept 头用于这样的情形:当客户对于可接受的信息有限制时,它通过这些头告诉服务器自己希望接受哪些信息。第一个头指定了哪些 MIME 类型是可以接受的(例如, text/html)。第二个头给出了字符集(比如 ISO 8859 1 或者 Unicode-1-1)。第三个头处理压缩方法(比如 gzip)。第四个头指明了一种自然语言(例如西班牙语)。如果服务器有多个页面可供选择的话,那么它可以利用这些信息来向客户提供它所需要的页面。如果客户的请求不能满足的话,则返回一个错误码并且请求失败。

Host 头是服务器的名字。它是从 URL 中提取出来的。这个头是必需的。之所以需要使用它的原因是,有些 IP 地址可能对应了多个 DNS 名字,所以服务器需要某种方法来分辨出该把这个请求传递给哪一个主机。

对于那些被保护的页面, Authorization 头是有用的。在这种情况下,客户可能需要证明自己有权来查看被请求的页面。这个头就可用于这种情况。

尽管 cookie 是在 RFC 2019 中而不是 RFC 2616 中被阐述的,但它们也有两个头。客户利用 Cookie 头可以向服务器返回一个以前由服务器所在域中某台机器发送的 cookie。

Date 头能够被双向使用,它包含消息被发送时的时间和日期。Upgrade 头的用途是,使得过渡到将来(可能不兼容的)版本的 HTTP 协议更加容易。它允许客户声明它能够支持什么版本,也允许服务器声明它正在使用什么版本。

下面我们讨论仅仅被服务器用在回应消息中的头。第一个, Server, 允许服务器说明它是谁,以及如果它愿意的话,还可以提供其他某些属性。

接下来四个头都以 Content- 开头,这些头允许服务器描述它所发送的页面的属性。

Last-Modified 头说明了该页面最后被修改的时间和日期。这个头在页面缓存机制中起到了重要的作用。

Location 头的用途是,服务器可以用它来通知客户,它应该尝试另外一个不同的 URL。如果页面被移动了,或者允许多个 URL 指向同一个页面(可能在不同的服务器上),则可以使用这个头。它也可被用于另外一种情况,即公司的主 Web 页面在 com 域中,但是该页面根据客户的 IP 地址或者首选的语言,将客户的请求重定向到一个国家的或者地区的页面中。

如果一个页面非常大,则小的客户可能不想一次获取所有的内容。有些服务器可以接受对于一定字节范围的请求,也就是说浏览器通过多个小单位的请求将页面取回来。Accept-Ranges 头声明了服务器愿意处理这种部分页面请求。

第二个与 cookie 相关的头, Set-Cookie, 是服务器向客户发送 cookie 的途径。客户应该将 cookie 保存下来,并且在以后向服务器发送请求时返回这个 cookie。

## HTTP 使用举例

因为 HTTP 是一个基于 ASCII 的协议,所以,对于一个坐在终端前面的人(相对于浏



览器)来说,他很容易与 Web 服务器直接通话。他所需要的是一个指向服务器 80 端口的 TCP 连接。鼓励读者亲自尝试一下这样的场景(最好用一个 UNIX 系统,因为其他一些系统不返回连接状态)。下面的命令序列可以做到这一点:

```
telnet www.ietf.org 80 >log
GET /rfc.html HTTP/1.1
Host: www.ietf.org

close
```

这个命令序列启动一个 telnet 连接(即 TCP 连接),该连接指向 IETF 的 Web 服务器 www.ietf.org 的 80 端口。会话的结果被重定向到文件 log 中,以便于以后查看。接下来是 GET 命令,它指明了文件名和协议。下一行的 Host 头是必需的。再接下来的空白行也是必需的。它告诉服务器没有更多的请求头了。Close 命令指示 telnet 程序中断此连接。

你可以用任何一个编辑器来检查 log 文件。该文件必定与图 7.44 所示的列表类似,除非 IETF 最近改变了它的页面。

```
Trying 4.17.168.6...
Connected to www.ietf.org.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: avoid browser bug

<html>
<head>
<title>IETF RFC Page</title>

<script language="javascript">
function url() {
var x = document.form1.number.value
if (x.length == 1) {x = "000" + x }
if (x.length == 2) {x = "00" + x }
if (x.length == 3) {x = "0" + x }
document.form1.action = "/rfc/rfc" + x + ".txt"
document.form1.submit
}
</script>

</head>
```

图 7.44 www.ietf.org/rfc.html 的输出开始部分

前三行是 telnet 程序的输出,而不是远程站点的输出。以 HTTP/1.1 开头的行是 IETF 的回应,表明它愿意以 HTTP/1.1 与你通话。接下来是一些头信息,然后是内容。

我们已经看到过除了 ETag 和 X-Pad 以外的所有头信息,这里 ETag 是与缓存有关的页面惟一标识符,X-Pad 是非标准的,可能是针对某些有错误的浏览器的一种回避措施。

### 7.3.5 性能增强

Web 的极度流行差点也毁了它自己。服务器、路由器和线路经常超载运行。许多人开始把 WWW 称作 World Wide Wait(世界范围的等待)。如此无限延迟的结果是,研究人员开发出了各种各样的技术来提高性能。现在我们来讨论其中的三种技术:缓存、服务器复制和内容分发网络(content delivery network)。

#### 缓存

一种相当简单的提高性能的方法是,将已经被请求过的页面保存起来以防它们被再次请求。这项技术对于那些被大量访问的页面特别有效,例如 www.yahoo.com 和 www.cnn.com。将页面储存起来以备后续使用的做法被称为**缓存(caching)**。通常的过程是,由某个进程来维护缓存的内容,这个进程被称为**代理(proxy)**。为了使用缓存技术,浏览器必须也要做相应的配置,以便所有的页面请求都被发送给代理,而不是真正的服务器。如果代理进程有这个页面,则马上返回该页面。如果没有,则它从服务器处取回页面,并把该页面加入到缓存中以备将来使用,然后将它返回给请求该页面的客户。

两个与缓存相关的重要问题是:

- (1) 应该由谁来实施缓存?
- (2) 页面应该被缓存多久?

对于第一个问题有多个答案。个人 PC 通常使用代理,所以它们能够快速找到以前访问过的页面。在一个公司的 LAN 上,代理通常是一台机器,该 LAN 上所有其他的机器共享同一个代理,所以,如果一个用户访问了某个页面,然后同一 LAN 上的另一个用户也想要访问这个页面,则后者可以从代理的缓存中获取该页面。许多 ISP 也运行了代理,从而为它们的用户提高访问速度。通常所有这些缓存同时运行,所以,客户的请求首先到达本地的代理。如果本地代理失败了,则本地代理向 LAN 代理发出请求。如果 LAN 代理也不成功,则它会请求 ISP 的代理。这个请求必须成功,它要么从自己的缓存中,要么从更高层的代理,或者从服务器本身获得页面。一个涉及多级顺序代理的方案被称为**分级缓存**。图 7.45 给出了一种可能的实现方式。

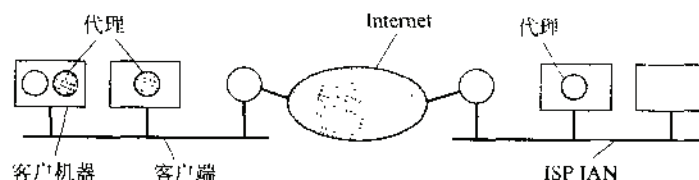


图 7.45 包含三个代理的分级缓存

页面应该被缓存多久,这是一个比较复杂的问题。有些页面根本不应该被缓存。例如,一个包含 50 支最活跃股票的价格的页面每秒钟都在变化。如果它被缓存了,那么,从

缓存中获取页面的用户将得到已失效的(即过时的)数据。另一方面,一旦当天的股票交易结束以后,则这个页面会持续有效数个小时或数天,直到下一个交易日开始。因此,一个页面的可缓存性可能会随着时间的变化而有所不同。

决定何时将一个页面从缓存中移除的关键在于,用户愿意忍受的页面过时程度(因为缓存的页面被保存在磁盘上,所以,所消耗的存储空间一般不是一个问题)。如果一个代理很快地抛弃页面,那么它很少会返回一个过时的页面,但是它的效率可能不是很高(例如,命中率非常低)。如果一个代理将页面保存很长时间,则命中率可能会很高,但是相应的代价是经常返回过时的页面。

可以有两种方法来处理这个问题。第一种采用启发式方法来猜测每个页面应该保存多久。常用的做法是基于 Last-Modified 头(参见图 7.43)来决定保存时间。如果一个页面在一个小时以前被修改过,那么它在缓存中被保存一个小时。如果它是在一年以前被修改的,那么显然它是一个非常稳定的页面(例如,希腊和罗马神话故事中神的列表),所以有理由认为它在一年中不会改变,因而可以被缓存一年。尽管这种启发式方法在实践中工作得很好,但是它仍然时不时地返回过时的页面。

另一种方法的代价更高,但通过使用 RFC 2616 中有关缓存管理的特性,可以消除返回过时页面的可能性。这些特性中最有用的一个是 If-Modified-Since 请求头,代理可以向服务器发送此请求头。它指定了代理想要的页面,以及缓存的页面上次被修改的时间(从 Last-Modified 头中获得)。如果从那时以来这个页面还没有被修改过,则服务器送回一个简短的 Not Modified(未被修改)消息(图 7.42 中的状态码 304),它指示代理使用缓存的页面。如果该页面在那以后被修改过,则服务器返回新的页面。尽管这种方法总是需要一条请求消息和一条应答消息,但是当缓存的页项仍然有效时应答消息非常短。

这两种方法可以很容易地结合起来。在取回了页面之后的第一个  $\Delta T$  时间内,代理直接向请求该页面的客户返回缓存的页面。当页面已经被缓存了一段时间以后,代理使用 If-Modified-Since 消息来检查它的新旧程度。在选择  $\Delta T$  时不可避免地需要某一种启发式方法,即根据该页面最后一次修改是在多久以前来推断  $\Delta T$ 。

包含动态内容(例如,由 PHP 脚本生成)的 Web 页面永远不应该被缓存起来,因为下一次的参数可能不会相同。为了处理这种及其他类似的情况,应该有一种通用的机制可以让服务器指示沿途到客户的路径上所有的代理都不要不检查当前页面的时效性就再次使用它。这种机制也可以被用于任何可能变化很快的页面。RFC 2616 还定义了一些缓存控制机制。

还有一种提高性能的方法是抢先缓存。当代理从服务器上取回一个页面时,它可以检查该页面,看该页面上是否含有超链接。如果有,则代理可以请求相关的服务器,预先将这些链接所指的页面下载并缓存起来,就好像客户需要这些页面一样。这项技术可以减少后续请求的访问时间,但是它也可能提升了那些从不真正被访问的页面,甚至这些流量可能会影响到通信线路。

显然,Web 缓存具有重要的价值。关于它还有很多可以讨论的内容。实际上,关于 Web 缓存已经有了几本专著,例如 Rabinovich and Spatscheck, 2002; and Wessels, 2001;但是我们现在该转移到下一个话题了。

## 服务器复制

缓存技术是一种提高性能的客户端技术,但是还存在服务器端的技术。服务器最常用的提高性能方法是把它们的内容复制到多个分布广泛的位置上。这种技术有时候称为**镜像(mirroring)**。

镜像技术的一种典型用途是,一家公司的主页包含一些图像,这些图像的链接分别指向该公司在东部、西部、北部和南部的区域性 Web 站点。然后,用户单击最近的一个区域站点图像就可以访问相应的服务器。从这时开始,所有的请求都指向被选中的服务器。

被镜像的站点通常是完全静态的。公司决定在哪里放置这些镜像站点,并且为每个区域安置一台服务器,每个位置上所放的内容可多可少(可能在迈阿密[Miami]站点省略吹雪机,而在安克雷奇[Anchorage]站点省略沙滩毯)。对站点的选择通常几个月或者几年保存稳定不变。

不幸的是,在 Web 上存在一种称为**瞬间拥挤(flash crowd)**的现象,当这种现象发生时,一个从前不知名的、无人访问的、如同死水般的站点突然成为宇宙的中心。例如,在 2000 年 11 月 6 日之前,佛罗里达州秘书处的 Web 站点, [www.dos.state.fl.us](http://www.dos.state.fl.us), 一直非常平静,它只是提供了佛罗里达州内阁会议的记录,以及关于如何在佛罗里达州成为一个公证人的指导材料。但是在 2000 年 11 月 7 日,当美国总统的选举结果突然间与佛罗里达州一些县的数千张具有争议的选票联系到一起时,这个站点变成了世界上访问量最大的五个 Web 站点之一。无需多说,它不能处理如此负载,几乎完全瘫痪。

这里需要的是有一种办法能够让突然觉察到流量大量上升的 Web 站点,自动地把自己克隆到足够数量的位置上,并且保持这些站点能够正常运行直至流量风暴过去,到那个时候它可以关闭一部分或者所有的克隆站点。要想具有这种能力,一个站点需要提前与某一家拥有大量宿主站点的公司达成协议,在协议中声明,该站点可以在需要的时候创建一些复制站点,并且根据实际使用能力来支付费用。

一种更加灵活的策略是,以每个页面为基础,并且根据页面的流量源来创建动态的复制页面。关于这个题目的一些研究工作请参考(Pierre et al., 2001; and Pierre et al., 2002)。

## 内容分发网络

资本主义的闪光点是,有人已经想到了如何利用 World Wide Wait(世界范围的等待)现象来赚钱。它的工作方式是这样的。一些称为 **CDN(Content Delivery Network, 内容分发网络)** 的公司去找内容提供商(音乐站点、报纸,以及其他一些希望它们的内容能够被简单而又快速地访问的组织)会谈,表示愿意将它们的内容有效地分发给最终用户,但是收取一定的费用。当合同签署以后,内容所有者把它的 Web 站点的内容交给 CDN,以便先做预处理(稍后将讨论),然后再分发。

然后,CDN 与大量的 ISP 进行会谈,表示愿意支付合理的费用,以便在 ISP 中放置一台可以远程管理的服务器,同时可以充实 ISP 的 LAN 上的有价值内容。对于这些 ISP 来说,这不仅是一个收入来源,同时也使它们的顾客在访问 CDN 的内容时有更好的响应

时间,因此这些 ISP 比起那些没有接受 CDN 提议的其他 ISP 来说,有更强的竞争优势。在这样的条件下,对于这些 ISP 来说,与一个 CDN 签约是无需更多考虑的。结果是,那些最大的 CDN 公司已经在全球部署了超过 10,000 台服务器。

将内容复制到世界范围内的成千上万个站点上,很明显有极大的性能提高潜力。然而,要使得 CDN 能够工作,必须有一种方法能够将客户的请求重定向到最近的 CDN 服务器上,最好该 CDN 服务器与客户在同一个 ISP 中。而且,这种重定向必须在不修改 DNS(或者 Internet 标准设施的任何其他部分)的情况下完成。下面简短地介绍一下最大的 CDN, Akamai,是如何做到的。

整个过程从内容提供商向 CDN 递交它的 Web 站点开始。然后 CDN 将每个页面通过一个预处理器,该预处理器用修改过的 URI 替换掉原来所有的 URL。这种策略背后的工作模型是,内容提供商的 Web 站点是由许多小的页面(只有 HTML 文本)构成的,但是这些页面通常链接至很大的文件,比如图像、音频和视频文件。修改后的 HTML 页面被保存在内容提供商的服务器上,并且客户可以用通常的方式来获取这些页面;而图像、音频和视频则位于 CDN 的服务器上。

为了看清楚整个方案是如何实际工作的,请考虑图 7.46(a)中 Furry Video 的 Web 页面。在预处理以后,它被转换成图 7.46(b),并作为 `www.furryvideo.com/index.html` 被放到 Furry Video 的服务器上。

```
<html>
<head> <title> Furry Video </title> </head>
<body>
<h1> Furry Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="bears.mpg"> Bears Today </a> <br>
<a href="bunnies.mpg"> Funny Bunnies </a> <br>
<a href="mice.mpg"> Nice Mice </a> <br>
</body>
</html>
```

(a)

```
<html>
<head> <title> Furry Video </title> </head>
<body>
<h1> Furry Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="http://cdn-server.com/furryvideo/bears.mpg"> Bears Today </a> <br>
<a href="http://cdn-server.com/furryvideo/bunnies.mpg"> Funny Bunnies </a> <br>
<a href="http://cdn-server.com/furryvideo/mice.mpg"> Nice Mice </a> <br>
</body>
</html>
```

(b)

图 7.46

(a) 原来的网页; (b) 转化之后的同一页面

当一个用户在浏览器中输入 URL `www.furryvideo.com` 时, DNS 返回 Furry Video 自己的 Web 站点的 IP 地址,从而允许主(HTML)页面以通常的方式被客户获取。当用

户单击任何一个超链接时,浏览器请求 DNS 查找 `cdn-server.com`,DNS 也如实返回 CDN 服务器的 IP 地址。然后浏览器向该 IP 地址发送一个 HTTP 请求,期望返回一个 MPEG 文件。

由于 `cdn-server.com` 并没有存储任何内容,所以这种情况并不会发生。相反,它只是 CDN 的伪 HTTP 服务器。它检查文件名和服务器名,以确定所请求的是哪个内容提供商的哪个页面。它也检查这个请求的 IP 地址,并且在它的数据库中查找,以确定该用户可能在哪里。利用这些信息,它确定 CDN 的哪一台内容服务器能够为该用户提供最好的服务。这种判断是很困难的,因为地理上最近的服务器不一定是网络拓扑意义上最近的服务器,而且,网络拓扑意义上最近的服务器可能当时正很忙。在做出决定以后,`cdn-server.com` 送回一个状态码为 301 的回应,并且包含一个 Location 头,其中给出了离客户最近的 CDN 内容服务器上的文件 URL。对于这个例子,我们假设这个 URL 是 `www.CDN-0420.com/furryvideo/bears.mpg`。然后浏览器以通常的方式来处理这个 URL,以得到实际的 MPEG 文件。

图 7.47 中显示了所涉及到的步骤。第一个步骤是查找 `www.furryvideo.com` 以得到它的 IP 地址。之后,浏览器以通常的方式获取并显示 HTML 页面。该页面包含三个指向 `cdn-sever` 的超链接[见图 7.46(b)]。比如说,当第一个超链接被选中时,浏览器向 DNS 查找它的地址(步骤 5),并且该地址被成功返回(步骤 6)。当请求 `bears.mpg` 的消息被发送给 `cdn-server`(步骤 7)时,客户被告知转向 `CDN-0420.com`(步骤 8)。当浏览器按照指示继续进行(步骤 9),它从代理的缓存中获得了 MPEG 文件(步骤 10)。真正使整个机制工作起来的特性是步骤 8,也就是说,伪 HTTP 服务器把客户的请求重定向到离用户最近的 CDN 代理上。

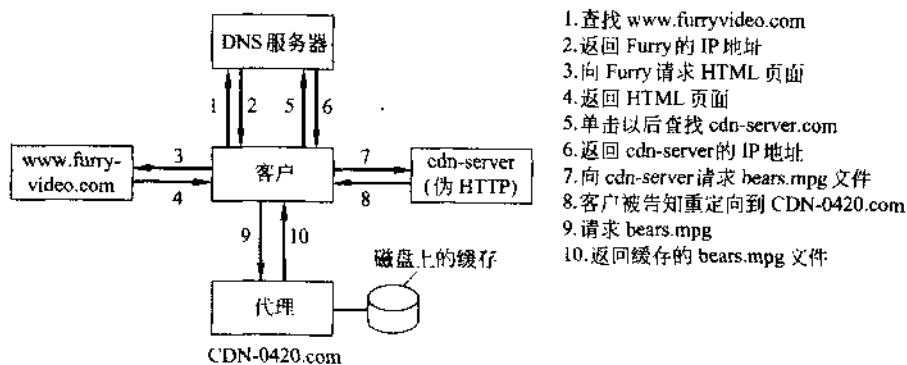


图 7.47 当使用了 CDN 时查找一个 URL 所涉及到的步骤

客户被重定向的那台 CDN 服务器通常是一个具有极大缓存空间的代理,它往往预先加载了最重要的内容。然而,如果某个人请求的文件不在缓存中,那么,该 CDN 服务器从真正的服务器上取回文件,然后放到缓存中以备将来使用。通过把内容服务器做成一个代理而不是一个完整的复制服务器,CDN 就能够做到在磁盘大小、预加载时间和各种性能参数之间达到平衡。



关于内容分发网络的更多内容,请参见(Hull, 2002; and Rabinovich and Spatscheck, 2002)。

### 7.3.6 无线 Web

人们对能够通过无线链路来访问 Web 的小型便携式设备有相当大的兴趣。事实上,人们在这个方向上已经采取了一些试探性的措施。毫无疑问,在接下来的几年里,这个领域将会发生许多变化,但是当前一些与无线 Web 相关的思想仍然值得我们去研究,这样我们就可以知道目前的状况以及可能的发展。在本小节中,我们将迎合当前的市场情况,重点讨论两个广域无线 Web 系统: WAP 和 i-mode。

#### WAP—无线应用协议

一旦 Internet 和移动电话变得普及起来,有些人很快就会想到要把它们组合到一部移动电话中,并且使用一个内置的屏幕来访问电子邮件和 Web。这里的“有些人”是一个最初由 Nokia、Ericsson、Motorola 和 phone.com(原来的 Unwired Planet)领导的联盟,现在这个联盟自称拥有数百个成员。他们建立的系统被称为 WAP(Wireless Application Protocol,无线应用协议)。

一个 WAP 设备可能是一部增强了功能的移动电话; PDA,或者无任何语音功能的笔记本电脑。WAP 规范允许所有这些设备,甚至更多。基本的思路是使用现有的数字无线设施。用户完全可以通过无线链路来呼叫一个 WAP 网关,并且向它发送 Web 页面请求。然后 WAP 网关检查它的缓存中是否存在这个被请求的页面。如果存在,则发送该页面;如果不存在,则通过有线 Internet 取回该页面。本质上,这意味着 WAP 1.0 是一个每分钟连接负荷相当高的电路交换系统。长话短说,人们不喜欢通过一个很小的屏幕来访问 Internet 并且还要按每分钟付费,所以 WAP 是失败的(尽管它还存在其他的问题)。然而,WAP 与它的竞争者 i-mode(将在下面讨论)似乎会汇聚到一项类似的技术上,所以 WAP 2.0 也许会很成功。由于 WAP 1.0 是对无线 Internet 的第一次尝试,它至少值得我们简要地描述一番。

在本质上,WAP 是一个专门针对 Web 访问的协议栈,但是由于无线设备往往有慢速的 CPU、较少的内存和很小的屏幕,所以 WAP 为这样的设备和低带宽的连接做了特殊的优化。这些需求显然不同于标准的桌面 PC 的情形,从而导致了一些协议差异。图 7.48 显示了 WAP 的协议层。

无线应用环境(WAE)
无线会话协议(WSP)
无线传输协议(WTP)
无线传输层安全(WTLS)
无线数据包协议(WDP)
承载层(GSM、CDMA、D-AMPS、GPRS 等)

图 7.48 WAP 协议栈

除了费用因素以外,影响 WAP 被广泛接受的另一个因素无疑是由于它没有使用 HTML。相反,WAP 层使用了一种称为 WML(Wireless Markup Language,无线标记语言)的语言,它是 XML 的一个应用。因此,在原理上,一个 WAP 设备只能访问那些已经被转换为 WML 的页面。然而,这极大地限制了 WAP 的价值,因为这种体系结构要求有一个很好的过滤器将 HTML 动态地转换为 WML,以增加可被 WAP 设备访问的页面数量。图 7.49 展示了这种体系结构。

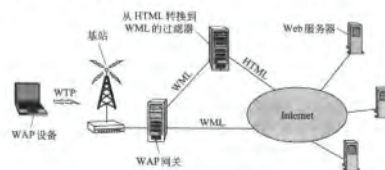


图 7.49 WAP 体系结构

公正地说，WAP可能有点超前了。当WAP刚开始被启动的时候，在W3C以外几乎无人知道XML，所以，最初有关WAP的报道都称“WAP不使用HTML”。一个更加准确的标题可能应该是：“WAP已经使用新的HTML标准了”。但是一旦这种损害已经造成，则很难再修复，而且WAP 1.0从没有抓住过机会。接下来，我们先讨论它的主要竞争对手，之后我们还会再次讨论WAP。

### I-Mode

当电信运营商和计算机公司的多产业联盟忙于采用 HTML 的最先进版本来打造一个开放标准的时候,在日本,另外一项研发工作也正在进行的之中。一个日本人, Mari Matsunaga,发明了一种不同寻常的访问 Web 的方式,称为 i-mode (information-mode, 信息模式)。她让以前日本电信垄断运营商的无线子公司相信她的方法是正确的,1999 年 2 月, NTT DoCoMo 公司宣布它是:你作用户日本电话电报公司 (在日本启动了无线服务。3 年内它便拥有了超过 3 亿 5 千万日本用户,他们可以访问 40 000 多个独特的 i-mode Web

站点。由于它成功地赚到了钱,所以令世界上大多数电信公司都垂涎三尺,尤其是在 WAP 似乎无法进行下去的情况下。下面我们来看一看 i-mode 是什么以及它是如何工作的。

i-mode 系统有三个主要部件:新的传输系统、新的手持机,以及新的 Web 页面设计语言。传输系统是由两个分离的网络组成的:现有的电路交换的移动电话网络(与 D-AMPS 类似),以及一个专门为 i-mode 服务而构建的新的分组交换网络。语音模式使用电路交换网络,并且按连接时间的分钟数进行计费。i-mode 使用分组交换网络,并且总是在线(类似于 ADSL 或者通过电缆联网),所以并非按连接时间来计费。相反,每个发送的分组都要收费。目前还不可能同时使用这两个网络。

手持机看上去像是附加了小屏幕的移动电话。NTT DoCoMo 花大力气来宣传 i-mode 设备是更好的移动电话,而不是无线 Web 终端,尽管它们的确是无线 Web 终端。实际上,可能大多数用户根本不知道自己已经在 Internet 上。他们认为自己的 i-mode 设备是具有增强功能的移动电话。为了与 i-mode 的这种模式保持一致,这些手持机不是用户可编程的,尽管它们相当于内含了一台 1995 年时候的 PC 机,也许还可以运行 Windows 95 或者 UNIX。

当 i-mode 手持机被打开的时候,呈现给用户的是一个经官方批准的服务分类列表。共 1000 多种服务被分成大约 20 个类别。每个服务实际上是一个小的 i-mode Web 站点,由一家独立的公司来运行。在官方菜单上的主要类别包括电子邮件、新闻、天气、体育、游戏、购物、地图、星象预测、娱乐、旅游、地区指南、铃声、医药处方、赌博、家庭银行和股票价格。这些服务瞄准的是十多岁的青少年和二十多岁的年轻人,他们比较喜欢电子设备,特别是具有时尚色彩的电子设备。目前有超过 40 家公司在出售铃声,仅仅这一事实就足以说明一些问题。最流行的应用是电子邮件,它允许长达 500 字节的消息,因此它可被看作是对 SMS(短消息服务)的极大改进,因为短消息最多只有 160 字节。另外,游戏也很流行。

i-mode Web 站点超过 40 000 个,但是用户必须通过输入它们的 URL 来访问它们,而不是从菜单中选择站点。在某种意义上,官方菜单列表就好像是一个 Internet 门户,它允许其他的 Web 站点可以通过单击的方式来访问,而无需再输入 URL。

NTT DoCoMo 紧紧地控制着官方服务。一个服务若想要进入到这个列表中,则必须符合各种已发布的规定。例如,该服务不能造成坏的社会影响,日-英词典必须有足够的词汇量,铃声服务必须经常增加新的铃声,而且该站点不允许激怒流行的行为或者对 NTT DoCoMo 有坏的影响(Frengle, 2002)。这 40 000 个 Internet 站点可以做它们想做的一切。

i-mode 的商业模式与传统 Internet 相差甚远,所以值得在这里做一番解释。基本的 i-mode 订阅费是每个月几美元。由于每个收到的分组都有费用,所以基本的订阅模式仅包含少量的分组。用户也可以选择另外一种含更多免费分组的订阅模式,并且每个分组的费用也随着用户的使用量从每个月 1MB 到每个月 10MB 而迅速降下来。如果在一个月的半中间,免费分组已经用完,则用户可以通过在线方式购买更多的分组。

为了使用一项服务,你必须订阅该服务,做法很简单,你只需单击该服务,然后输入你

的 PIN 码即可。大多数官方服务每个月收取 1 到 2 美元。NTT DoCoMo 把这些费用加到电话账单中,其中 91% 转给服务供应商,自己保留 9%。如果一个非官方的服务有 1 百万个用户,则它每个月必须送出 1 百万份(大约)1 美元的账单。如果该服务成为 NTT DoCoMo 的官方服务,则 NTT DoCoMo 会处理记账事宜,并且每个月把 910 000 美元汇入该服务供应商的银行账户中。无需处理记账事宜是服务供应商希望成为官方服务的最主要动机,这也为 NTT DoCoMo 带来了更多的收入。另外,当一项服务成为官方服务以后,它就会出现在初始菜单上,从而使该服务的站点更容易被找到。用户的电话账单包括电话呼叫的费用、i-mode 订阅费、服务订阅费和其他额外分组的费用。

尽管 i-mode 在日本取得了巨大的成功,但是目前还不清楚是否在美国和欧洲也能流行起来。在某些方面,日本的环境与西方不同。首先,西方大多数潜在用户(例如十多岁的青少年、大学生和商务人员)的家里已经有了大屏幕的 PC 机,而且几乎肯定有 Internet 连接,其速率至少 56Kbps。而在日本,很少人的家里有与 Internet 连接的 PC 机,部分原因是因为缺少足够的空间,同时也因为 NTT 过于昂贵的本地电话服务费(安装一条线大约要 700 美元,本地电话呼叫每小时 1.5 美元)。对于大多数用户,i-mode 是他们惟一的 Internet 连接。

第二,在西方,人们不习惯每月花费 1 美元仅为了访问 CNN 的 Web 站点,而为了访问 Yahoo 的 Web 站点还要花费 1 美元,为了访问 Google 的 Web 站点再花费 1 美元,如此等等,更不用说每下载 1MB 还要花费数美元。为了响应用户的要求,西方大多数 Internet 供应商现在都是每月收取固定的费用,而与实际的使用量无关。

第三,对于许多日本人来说,使用 i-mode 的主要时间是当他们上下班或者上学放学时在火车或者地铁中的那段时间。在欧洲,乘坐火车上下班的人比在日本少得多,在美国几乎没有人这么做。在家里,你在一台配有 17 寸显示器、1Mbps 的 ADSL 连接,以及所有流量均免费的计算机旁边再来使用 i-mode,显然毫无意义。然而,正如当初根本没有人预见到移动电话的广泛普及一样,所以,i-mode 也可能会在西方找到自己的一席之地。

正如我们上面提到的,i-mode 手持机对于语音信号使用现有的电路交换网络,对于数据信号使用新的分组交换网络。数据网络基于 CDMA,以 9600bps 的速率来传输 128 字节的分组。图 7.50 中给出了网络的结构图。手持机采用 LTP(Lightweight Transport Protocol,轻量传输协议)并通过空中链路来跟一个协议转换网关进行通话。该网关通过一条宽带光纤连接到 i-mode 服务器,i-mode 服务器又连接至所有的服务。当用户从官方菜单上选择一项服务时,该请求被发送至 i-mode 服务器,它缓存了绝大多数页面以提高性能。对那些不出现在官方菜单上的站点的请求则绕过 i-mode 服务器,直接通过 Internet 进行连接。

现在的手持机的配置如下:CPU 的速率大约为 100MHz、几兆字节的闪存(flash ROM),可能有 1MB 的 RAM,以及一个小的内置屏幕。i-mode 要求手持机屏幕至少是 72×94 像素,但有些高端的设备具有 120×160 像素。屏幕通常有 8 位颜色,因而允许 256 种颜色。这对于显示照片是不够的,但是对于线条画和简单的卡通来说则足够了。由于没有鼠标,屏幕上的导航操作需要通过箭头键来完成。

软件结构如图 7.51 所示。最底层是一个简单的实时操作系统,用来控制硬件。然后

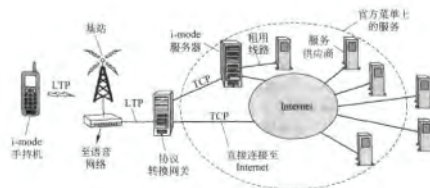


图 7.50 i-mode 数据网络的结构,图中也显示了传输协议

是网络通信模块,它使用 NTT DoCoMo 私有的 LTP 协议。在此之上是一个简单的窗口管理器,它能够处理文本和简单的图像(GIF 文件)。由于屏幕最多只有 120×160 像素,所以这里也没有太多需要管理的東西。

第四层包含了 Web 页面解释器(即浏览器)。i-mode 不使用完整的 HTML,而只是它的一个子集,称为 cHTML(compact HTML,紧凑的 HTML)。cHTML 基本上以 HTML 1.0 为基础。这一层也允许辅助应用程序和插件,就好像 PC 浏览器的做法一样。其中一个标准的辅助应用程序是 JVM 略微修改版本的解释器。

用户交互模块		
插件	cHTML 解释器	Java
简单的窗口管理器		
网络通信		
实时操作系统		

图 7.51 i-mode 的软件结构

最上面是用户交互模块,它负责管理与用户的通信。

我们现在来更仔细地看一看 cHTML。正如刚才所提到的,它近似于 HTML 1.0,但是为了在移动手持机上使用,它也作了一些省略和扩展。它被提交给 W3C 做标准化,但是 W3C 对此并不感兴趣,所以它很可能仍然是一个私有产品。

在 cHTML 中,绝大多数基本的 HTML 标签都是允许使用的,包括<html>、<head>、<title>、<body>、<h1>、<center>、<ul>、<ol>、<menu>、<li>、<br>、<p>、<hr>、<img>、<form>和<input>。但是,它不允许使用标签<b>与<i>。

标签<a>也是允许的,以便链接至其他的页面,但是 cHTML 又引入了一种新的 tel 模式,用于拨打电话号码。在某种意义上,tel 类似于 mailto。当一个使用 mailto 模式的超链接被选中时,浏览器弹出一个表单,以便向该链接中指定的目标方发送电子邮件。当



一个使用 tel 模式的超链接被选中时,浏览器拨打电话号码。例如,在一个地址簿中有许多人的简单图片。当用户选择其中之一时,手持机将会呼叫他或者她。RFC 2806 讨论了电话 URL。

cHTML 浏览器在其他方面有所限制。它不支持 JavaScript、框架、样式表、背景颜色或者背景图像。它也不支持 JPEG 图像,因为解压缩操作需要花费太多时间。Java applet 则是允许的,但是(目前)限制在 10KB 以内,因为在空中链路上传输速度很慢。

尽管 NTT DoCoMo 去掉了一些 HTML 标签,但同时它也增加了一些新的标签。<blink> 标签使文字闪烁。虽然禁止使用<b>(因为 Web 站点不应该处理外观显示)与加入<blink>(它仅仅涉及到外观显示)这两者看起来很不一致,但是 NTT DoCoMo 确实是这么做的。另外一个新的标签是<marquee>,它在屏幕上以证券报价器的方式滚动显示其内容。

一个新的特性是<br>标签的 align 属性。之所以需要这个属性,是因为在一个典型的 6 行高度、每行 16 个字符的屏幕上,单词可能会被从中间截断,如图 7.52(a)所示。Align 属性有助于减少这种问题,使得有可能得到如图 7.52(b)类似的效果。有趣的是,日本人并不因为单词被截断成两行而感到痛苦。对于日文汉字(Kanji),根据所支持的字体,屏幕被分成 9×10 像素或者 12×12 像素的矩形单元格。每个单元格正好容纳一个日文汉字,一个日文汉字与英语中的一个单词等价。在日语中,词之间的换行总是允许的。

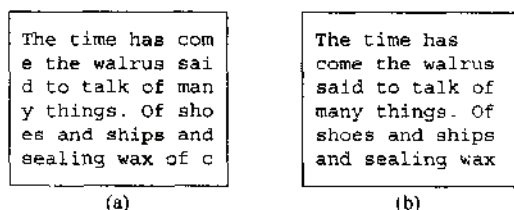


图 7.52 适合 16×6 屏幕的 Lewis Carroll 著作名段

尽管日语中有几万个日文汉字,但 NTT DoCoMo 还发明了 166 个被称为 emoji 的特殊新汉字,大多数都非常可爱——实际上,它们是一些类似于图 7.6 中微笑图标那样的象形文字。它们包括了代表以下事物的符号:星座黄道十二宫、啤酒、汉堡包、游乐园、生日、移动电话、狗、猫、圣诞节、破碎的心、吻、心情、睡觉,当然还有一个代表机灵和可爱的符号。

另一个新的特性是允许用户使用键盘来选择超链接的能力,很显然,这对于无鼠标的计算机来说是一个重要的特性。图 7.53 的 cHTML 文件中举例说明了这种特性的用法。

尽管客户端在某种程度上受到了限制,但 i-mode 服务器却是一台功能完全的计算机,它具有所有常见的配置。它支持 CGI、Perl、PHP、JSP、ASP,以及一般 Web 服务器通常支持的所有其他功能。

图 7.54 给出了在第一代系统中真正实现的 WAP 和 i-mode 的一份简短比较。尽管有些差别可能看起来非常小,但通常这些差别是非常重要的。例如,15 岁的人没有信用卡,所以,利用电子商务来购买东西,以及通过电话账单来付账,这两种能力将使得他们对



```

<html>
<body>
<h1> Select an option </h1>
<a href="messages.shtml" accesskey="1"> Check voicemail </a> <br>
<a href="mail.shtml" accesskey="2"> Check e-mail </a> <br>
<a href="games.shtml" accesskey="3"> Play a game </a>
</body>
</html>

```

图 7.53 cHTML 文件例子

于系统的兴趣有显著的差异。

关于 i-mode 的更多信息, 请参看(Frengle, 2002; and Vacca, 2002)。

## 第二代无线 Web

WAP 1.0 基于一些被公认的国际标准, 人们曾经期望它是未来移动商务中的重要工具, 但是它失败了。I-mode 只是日本年轻人的电子玩具, 一切都是私有的。但是它获得了巨大的成功。接下来将会怎么样呢? 每一方都从第一代无线 Web 中学到了一些东西。WAP 联盟学到了内容方面的教训。没有足够多的 Web 站点使用你的标记语言, 这将是致命的。NTT DoCoMo 则学到了: 一个与小型手持设备和日本文化紧密相关联的私有封闭系统不是一个好的出口产品。双方得出的结论是: 要让大量的 Web 站点采用你的格式来表达它们的内容, 那么, 一个开放的、稳定的、可被普遍接受的标记语言是非常必要的。格式争战对商业竞争没有任何好处。

特性	WAP	i-mode
它是什么	协议栈	服务
设备	手持机, PDA, 笔记本电脑	手持机
接入	拨号	总是在线
底层网络	电路交换	两种: 电路交换 + 分组交换
数据速率	9600bps	9600bps
屏幕	单色	彩色
标记语言	WML(XML 应用)	cHTML
脚本语言	WML 脚本	无
计费方式	按分钟数	每个分组
购物付款方式	信用卡	电话账单
象形文字	无	有
标准化	WAP 论坛开放标准	NTT DoCoMo 私有
使用范围	欧洲、日本	日本
典型用户	商人	年轻女人

图 7.54 第一代 WAP 与 i-mode 的比较

这两种服务都将进入到第二代无线 Web 技术领域中。WAP 2.0 首先出现,所以我们将拿它作为例子来讲述。WAP 1.0 做的有些事情是正确的,这些还将继续下去。首先,WAP 可以被承载在不同类型的网络上。第一代使用了电路交换网络,但是分组交换网络是一种选择,即使现在还是这样。第二代系统有可能使用分组交换,例如,GPRS。另外,WAP 最初的目标是支持各种各样的设备,从移动电话到功能强大的笔记本电脑,现在仍然坚持这个目标。

WAP 2.0 也有一些新的特性,最重要的一些是:

- (1) 推(push)模型与拉(pull)模型。
- (2) 允许将电话集成到应用中。
- (3) 多媒体消息。
- (4) 包含 264 个象形文字。
- (5) 与存储设备之间的接口。
- (6) 在浏览器中支持插件。

大家都知道拉模型:客户请求一个页面并且获得该页面。推模型是指无需请求就可以获得数据,例如,连续地获得股票价格或者交通路况报警信息。

语音和数据开始融合,WAP 2.0 有多种方式来支持它们的融合。我们在前面的介绍中已经看到过一个例子,即 i-mode 能够把一个屏幕图标或者文本链接到一个要呼叫的电话号码上。除了电子邮件和电话以外,WAP 2.0 还支持多媒体消息。

i-mode 中 emoji 的广泛流行刺激 WAP 联盟也发明了 264 个它自己的 emoji。其种类包括动物、器具、衣服、情绪、食物、人体、性别、地图、音乐、植物、体育、时间、工具、汽车、武器和天气。非常有趣的是,该标准只是命名了每个象形文字,而并没有给出实际的位图,可能是担心某一种文化中表达“睡觉”或者“拥抱”的图案冒犯了其他的文化。i-mode 不存在这个问题,因为它只是为一个国家而设计的。

提供一个存储接口并不意味着每部 WAP 2.0 电话都配备一个大硬盘。闪存(Flash ROM)也是一种存储设备。一个支持 WAP 功能的无线相机可以先将图像临时存储到闪存中,然后再将最好的图像传输到 Internet 上。

最后,插件能够扩展浏览器的功能。并且 WAP 2.0 还提供了一种脚本语言。

在 WAP 2.0 中也体现了各方面的技术差异。两个最大的技术差异是协议栈和标记语言。WAP 2.0 继续支持图 7.48 所示的老的协议栈,但是也支持标准的、包含 TCP 和 HTTP/1.1 在内的 Internet 协议栈。然而,它对 TCP 作了四个小的(但是兼容的)修改(以便简化代码):(1)使用固定的 64KB 窗口,(2)没有慢启动,(3)最大 MTU 为 1500 字节,(4)一种略微有所不同的重传算法。TLS 是由 IETF 标准化的传输层安全协议,我们将在第 8 章中讨论该协议。最初的许多设备可能会同时包含这两个协议栈,如图 7.55 所示。

与 WAP 1.0 的另一个技术差异是标记语言。WAP 2.0 支持 XHTML Basic,它是专门为小型无线设备而设计的。由于 NTT DoCoMo 也同意支持这个子集,所以,Web 站点设计者可以使用这种格式,并且知道他们的页面既可以在固定位置的 Internet 上正常工作,也可以在所有的无线设备上正常工作。这些决定最终将结束多年来妨碍无线 Web 工

业成长的标记语言格式之争。

XHTML	
WSP	HTTP
WTP	TLS
WTLS	TCP
WDP	IP
承载层	承载层
WAP 1.0 协议栈	WAP 2.0 协议栈

图 7.55 WAP 2.0 支持两个种协议栈

关于 XHTML Basic,也许值得再提几句。它是为移动电话、电视、PDA、售货机、寻呼机、汽车、游戏机,甚至手表而设计的。由于这个原因,它不支持样式表、脚本或者框架,但是支持大多数标准标签。这些标签被分成 11 个模块。有些是必需的;有些是可选的。所有这些都被定义成 XML。图 7.56 中列出了这些模块和一些示例标签。我们不再讨论所有这些示例标签,有关更多的信息,你可以在 [www.w3c.org](http://www.w3c.org) 上找到。

模块	是否必需	功能	示例标签
Structure(结构)	是	文档结构	Body,head,html,title
Text(文本)	是	信息	br,code,dfn,em,hn,kbd,p,strong
Hypertext(超文本)	是	超链接	a
List(列表)	是	逐项列举的列表	dl,dt,dd,ol,ul,li
Form(表单)	否	待填表单	form,input,label,option,textarea
Table(表格)	否	矩形表格	caption,table,td,th,tr
Image(图像)	否	图片	img
Object(对象)	否	Applet,地图等	object,param
Meta-information(元信息)	否	额外信息	meta
Link(链接)	否	类似于<a>	link
Base(基信息)	否	URL 起点	base

图 7.56 XHTML Basic 的模块和标签

尽管 WAP 和 i-mode 一致同意使用 XHTML Basic,但是对于它们来说,在空中还潜伏着另外一种威胁: 802.11。第二代无线 Web 假设运行在 384kbps 的速率上,远好于第一代的 9600bps,但是比起 802.11 所提供的 11Mbps 或 54Mbps 来说则差远了。当然,802.11 并不是到处都存在,但是越来越多的饭店、宾馆、商店、公司、机场、汽车站、博物馆、大学、医院和其他组织决定为他们的雇员和顾客安装基站,因此,在市区中可能会有足够的覆盖面,以至于人们愿意步行几个街区到有 802.11 服务的快餐店,坐下来享受一杯

咖啡,同时写上一封电子邮件。商家可能会不失时机地把 802.11 的标志放置在他们可接受的信用卡的标志旁边,这也是因为同样的原因:吸引顾客。城市地图(当然是可下载的)可能会用绿色来表示覆盖了 802.11 的区域,而用红色来表示未被覆盖的区域,所以人们可以从一个基站漫步到另一个基站,就好像游牧部落的牧民们在沙漠中从一个绿洲跋涉到另一个绿洲。

尽管快餐店可能很快就会安装上 802.11 基站,但是农场可能还做不到,再加上 802.11 覆盖范围的限制(最好的可以达到几百米),所以 802.11 的覆盖情况将会很不一致,实际的覆盖区域往往被限制在城市的商业区。这可能会导致双模式的无线设备,也就是说,当能够接收到信号时使用 802.11,当不能够接收到信号时则回退到 WAP。

## 7.4 多媒体

无线 Web 是一件新发展起来的激动人心的事物,但它不是惟一的一件。对于许多人来说,多媒体是网络的圣杯。每当提到多媒体这个词的时候,无论是技术的倡导者,还是市场的开拓者,都不约而同地开始垂涎三尺。前者看到了在为每个家庭提供(交互式的)视频点播中的巨大技术挑战,后者则看到了其中同样巨大的利润。由于多媒体要求非常高的带宽,因此要让它通过固定的连接来工作非常困难。在无线系统中,即使传输 VHS 质量的视频也是几年以后的事情了,所以我们的讨论将集中在有线系统上。

从字面来说,多媒体就是两种或多种媒体。如果本书的出版商想要加入到当前关于多媒体的广告宣传行列中,那么,它可以宣称这本书使用了多媒体技术。毕竟,本书确实包含了两种媒体:文字和图形(插图)。然而,当大多数人提到多媒体的时候,他们一般是指两种或者多种连续媒体(continuous media)的组合,所谓连续媒体,是指必须在一段明确定义的时间间隔内被播放出来的媒体,通常伴有某些用户交互动作。在实践中,这两种媒体通常是音频和视频,也就是声音加上移动图像。

然而,许多人常常把诸如 Internet 电话或者 Internet 电台之类的纯音频媒体也称为多媒体,很显然这是不恰当的。实际上,一个更好的术语是流媒体(streaming media),但我们将沿袭这些人的说法,将实时音频也看作多媒体。在接下来的几小节中,我们将看到计算机是如何处理音频和视频、音频和视频又是如何被压缩的,以及这些技术的一些网络应用。关于网络多媒体的全面(三卷)论述,请参看(Steinmetz and Nahrstedt, 2002; Steinmetz and Nahrstedt, 2003a; and Steinmetz and Nahrstedt, 2003b)。

### 7.4.1 数字音频介绍

音频(声音)波是一维的声(压力)波。当声波进入耳朵时,鼓膜发生振动,从而引起内耳的微细感骨与之一起振动,并向大脑发送神经脉冲。听者感觉到的这些脉冲就是声音。基于类似的方式,当声波碰击麦克风时,麦克风产生电信号,该电信号将声音振幅表示为时间的函数。这种音频信号的表达、处理、存储和传输正是多媒体系统研究工作的一个主要部分。

人耳能听到的声音频率范围是从 20Hz 到 20 000Hz。有些动物,特别是狗,能够听到

更高的频率。耳朵听声音时符合对数关系,因此,功率为 A 和 B 的两种声音的比率,通常按以下公式被表示成分贝(dB,decibels):

$$\text{dB} \approx 10 \log_{10}(A/B)$$

如果我们将 1kHz 正弦波的能听度低限(约 0.0003 达因/cm<sup>2</sup>的压力)定义为 0 分贝,那么普通谈话大约是 50 分贝,耳朵感觉痛的阈值大约是 120 分贝,变化的范围是 1 百万倍。

令人惊讶的是,人的耳朵对于持续时间仅仅几毫秒的声音变化也异常地敏感。相反地,眼睛却察觉不到持续时间为几毫秒的光亮度变化。这一观察的结果是,多媒体传输中仅仅几毫秒的抖动对于可感觉到的声音质量的影响,要大于对可感觉到的图像质量的影响。

通过 ADC(Analog Digital Converter, 模数转换器)可以将音频波转换成数字形式。ADC 以电压作为输入,生成一个二进制数作为输出。在图 7.57(a)中,我们看到了一个正弦波的例子。为了用数字方式来表示该信号,我们可以每隔 T 秒对它采样一次,如图 7.57(b)中的条状高度所示。如果一个声波不是纯的正弦波,而是若干正弦波的线性叠加,并且其中最高的频率成分为 f,那么,根据 Nyquist 定理(参见第 2 章),以 2f 的频率进行采样已经足够了。更高的采样频率是没有意义的,因为此类采样能够检测到的更高频率在这里并不存在。

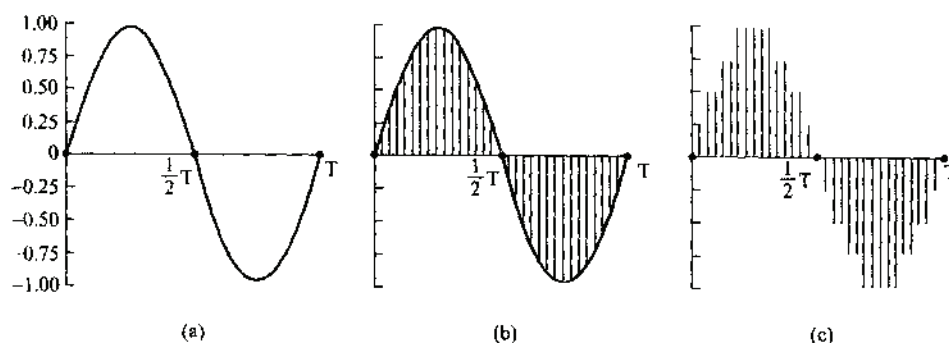


图 7.57

(a) 正弦波; (b) 对正弦波进行采样; (c) 将采样值量化为 4 位

数字采样绝对不会是精确的。图 7.57(c)中的采样只允许 9 个值,从 -1.00 到 +1.00,步长为 0.25。一个 8 位的采样将允许有 256 个不同的值。一个 16 位的采样将允许有 65 536 个不同的值。由于每个采样的位数有限而引入的误差被称为量化噪声(quantization noise),如果量化噪声太大,则耳朵就能觉察得出来。

两个众所周知的使用声音采样技术的例子是电话和音频 CD(compact disc)。电话系统中用到的脉冲编码调制(Pulse code modulation),使用了每秒 8000 次的 8 位采样。在北美和日本,其中 7 位用于数据,1 位用于控制;在欧洲,全部 8 位都用于数据。这个系统的数据率为 56 000 bps 或 64 000 bps。在每秒仅 8000 个采样的情况下,4kHz 以上的频

率将会丢失。

音频 CD 是数字化的,其采样率为每秒 44 100 个采样,这足以捕获高达 22 050Hz 的频率;22 050Hz 对于人来说已经非常好了,但对于犬类音乐爱好者而言却很糟糕。这些采样每个都是 16 位,并且在振幅范围内是线性的。尽管当按照人耳最小可听见的声音作为衡量单位来测量时,耳朵的动态范围大约是 1 百万倍,但是请注意,16 位采样只允许有 65 536 个不同的值。因此,如果每个采样仅使用 16 位,则会引入一定的量化噪声(尽管 CD 并没有覆盖整个动态范围——CD 不应该会刺伤或刺痛人耳)。在每秒 44 100 个采样以及每个采样 16 位的情况下,单声道的音频 CD 需要 705.6kbps 的带宽,立体声的音频 CD 需要 1.411Mbps 的带宽。虽然这低于视频所需要的带宽(见下一小节),但实时地传输未经压缩的 CD 品质的立体声声音仍然会占用几乎整个 T1 信道。

用计算机软件很容易就可以处理数字化的声音。在个人计算机上有许多程序可以让用户对来自不同源的声波进行记录、播放、编辑、混合以及存储操作。事实上,现在所有专业的声音记录和编辑操作都是数字化的。

当然,音乐只是普通音频中的一个特例,但它却是一种重要的音频。另一种重要的特殊音频是语音。人类的语音一般在 600Hz 到 6000Hz 的范围内。语音由元音和辅音组成,两者各有不同的特性。当声道畅通无阻时就会发出元音,并且还会产生共振,共振的基频取决于发声系统的大小和形状,以及说话人舌头和下颚的位置。这些声音几乎都是周期性的,其间隔大约是 30ms。当声道被部分阻塞时发出的声音是辅音。这些声音不像元音那样有规律。

有些语音生成和传输系统利用人类发声系统的模型,将语音信号减少至几个参数(例如,不同腔的大小和形状),而不是仅仅对语音波形进行采样。不过,关于这些声音合成器的工作原理超出了本书的范围,所以不再进一步介绍。

#### 7.4.2 音频压缩

正如我们刚才所看到的,CD 品质的音频需要 1.411Mbps 的传输带宽。很明显,实质性的压缩是非常有必要的,这样才使得可以在 Internet 上传输音频数据。由于这个原因,人们已经研究出了各种音频压缩算法。可能最流行的压缩算法是 MPEG 音频算法,它有三个层次(变种),其中 MP3(MPEG audio layer 3, MPEG 音频层 3)是最强大的,也是最知名的。Internet 上存在大量的 MP3 格式的音乐,其中有些是非法的,这也导致了艺术家和版权所有者们的大量诉讼。MP3 属于 MPEG 视频压缩标准的音频部分。在本章的后面我们还将讨论视频压缩。现在我们先来讨论音频压缩。

音频压缩可以通过两种方法中的任何一种来完成。在波形编码(waveform coding)中,它通过傅立叶变换,将信号从数学上转换成频率分量。图 2.1(a)显示了一个时间函数的例子,以及它的傅立叶变换。然后以最小的方式对每个分量的强度进行编码。这样做的目标是,尽可能在另一端以较少的位数重现该波形。

另一种方法被称为感知编码(perceptual coding),它利用人的听觉系统中某些特定的缺陷来对信号进行编码,这种编码方法保证了,即使通过示波器观察到的信号与原信号有相当大的差异,但人耳朵听起来还是一样的。感知编码方法建立在心理声学



(psychoacoustics)的基础上,即人类是如何感觉到声音的。MP3 正是基于感知编码的。

感知编码的关键特性是,某些声音可以屏蔽(mask)其他的声音。请想象在一个温暖的夏日里,你正在现场直播一场长笛音乐会。突然,附近的一群工人打开他们的手提电钻开始在马路上施工。于是,没有人还能够听得到长笛的声音。长笛的声音被手提电钻屏蔽了。如果仅仅考虑传输的话,则只需对手提电钻所使用的频段进行编码就足够了,因为听众反正也听不到长笛的声音。这种情形被称为频率屏蔽(frequency masking):在某一频段中较大的声音隐藏了另外一个频段中较柔和的声音,这种能力即为频率屏蔽,并且,如果没有这个较大的声音,则较柔和的声音是可以被听到的。实际上,即使手提电钻停下来,也有很短的一段时间内听不到长笛的声音,因为当手提电钻工作的时候耳朵降低了它的放大倍率,它需要一段有限的时间来重新提高放大率。这种效果被称为暂时屏蔽(temporal masking)。

要使这些效果更明显,请想象一下实验 1。一个人在一个安静的房间里,戴着一个连接到计算机声卡的耳机。计算机生成一个较弱的 100Hz 的纯正弦波信号,然后逐渐增加功率。现在给这个人的指示是,当她听到声音的时候就敲击一个键。计算机记录下此时的功率值,然后用 200Hz、300Hz 和其他所有在人类听力范围以内的频率来重复这个实验。把多个人的数据进行平均,从而得到了如图 7.58(a)所示的一个对数(log-log)图,它表示了“任一个音调被人耳听见需要多少功率”。根据图中的曲线,可以直接得出的一个结论是,如果一个频率的功率低于能听度阈值,则永远不需要对该频率进行编码。例如,如果图 7.58(a)中 100Hz 的功率是 20dB,那么它可以在输出中被省略掉而不会有可感觉的品质损失,因为在 100Hz 频率上,20dB 低于能听度水平。

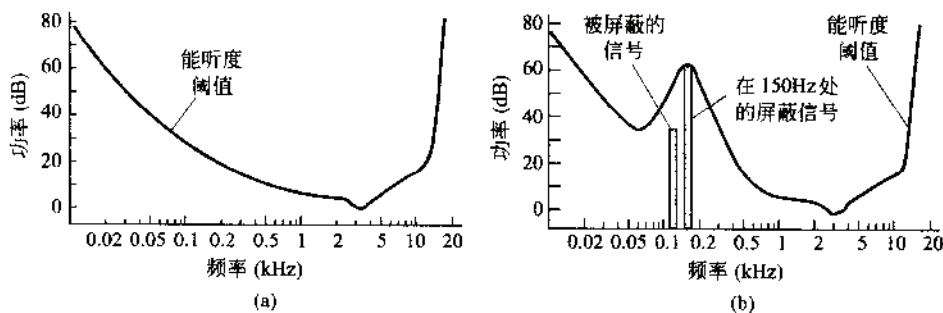


图 7.58

(a) 能听度阈值与频率的函数关系; (b) 屏蔽效应

现在考虑实验 2。计算机再次运行实验 1,但这次在测试频率上又叠加了一个固定振幅的正弦波,比如说它的频率是 150Hz。我们发现,在 150Hz 附近,频率的能听度阈值增加了,如图 7.58(b)所示。

这次实验观测的结果是,通过记录下哪些信号被附近频段中更强的信号所屏蔽,我们就可以在编码信号时省略越来越多的频率,从而节省数据位。在图 7.58 中,125Hz 的信号可以在输出中被完全省略,而且没有人能够听得出差异来。在某个频段中,即使当一个

功率较强的信号停止以后,考虑到它的暂时屏蔽特性,我们可以继续省略被屏蔽的频率一段时间(即耳朵恢复听力的时间)。MP3 的本质是,对声音进行傅立叶变换以得到每个频率处的功率,然后只传输未被屏蔽的频率,并且用尽可能少的位数对它们进行编码。

有了以上这些信息作为背景,我们现在来看一看该如何进行编码。音频压缩的做法是,以 32kHz、44.1kHz 或者 48kHz 频率对波形进行采样。此采样过程可以在一个或者两个通道上完成,采用以下四种配置之一:

- (1) 单声道(单一输入流)。
- (2) 双声道(例如,一个英语声道和日语声道)。
- (3) 分离立体声(每个通道被单独压缩)。
- (4) 联合立体声(充分利用通道之间的冗余)。

首先,选择输出的数据速率。MP3 可以将一个立体声的摇滚音乐 CD 压缩到 96kbps,而在品质上只有极少可以感觉得到的损失,甚至摇滚音乐爱好者也听不出来。对于一个钢琴音乐会,则至少需要 128kbps。之所以有这样的差异,是因为摇滚音乐的信噪比远远高于钢琴音乐会的信噪比(至少从工程的角度来说是这样的)。选择更低的输出数据率也是可以的,但要忍受一些品质上的损失。

然后,按照每一组 1152 个采样值(大约 26ms)进行处理。首先每一组采样被通过 32 个数字滤波器,以得到 32 个频段。同时,输入数据也被传递给一个心理声学模型,来决定被屏蔽的频率。接下来,这 32 个频段中的每一个频段被进一步转换,以便提供更细的频谱分辨率。

下一阶段,在频段之间分配可使用的位数,具有较多未屏蔽功率的频段分配相对较多的位数,而功率较少的未屏蔽频段则分配较少的位数,被屏蔽的频段不分配数据位。最后,使用霍夫曼编码对这些数据位进行编码,它对频繁出现的数值分配较短的编码,不频繁出现的数值分配较长的编码。

实际的过程比这要复杂得多。如果可能,还采用了多种技术来减少噪声、抗混淆、以及利用通道之间的冗余性,但这些技术超出了本书的范围。有关此处理过程的更多形式化数学介绍,请参考(Pan, 1995)。

### 7.4.3 流式音频

现在我们从数字音频处理技术转向它的三个网络应用。第一个应用是流式音频(streaming audio),即通过 Internet 来收听声音,这也被称为音乐点播。在接下来的两个应用中,我们将依次看到 Internet 电台和 IP 语音(voice over IP,简称 VoIP)。

Internet 上充满了音乐 Web 站点,许多音乐站点列出了歌曲的标题,用户只需单击这些标题就可以播放歌曲。有些音乐站点是免费的(例如,新的乐队期望引起公众的关注);其他的音乐站点要求支付一定的费用才可以提供音乐,不过它们往往也会提供一些免费的样品歌曲或片断(例如,一首歌的前 15 秒)。通过 Web 来播放音乐的最直接方法如图 7.59 所示。

当用户单击一首歌曲时这个过程就开始了。于是浏览器开始行动起来,在第 1 步中,浏览器与该歌曲的超链接所指向的 Web 服务器建立一个 TCP 连接。第 2 步是通过

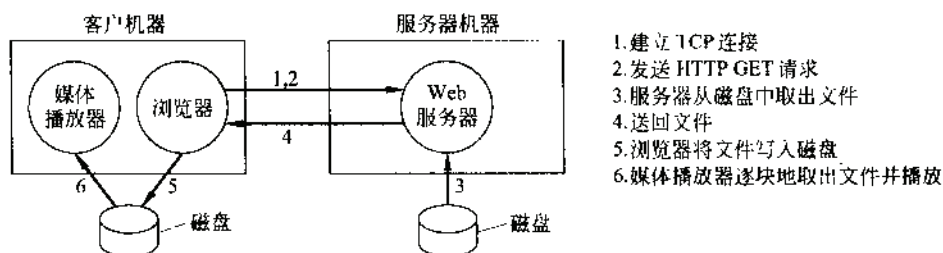


图 7.59 一种在 Web 页面上实现可单击音乐的直接方法

HTTP 发送一个 GET 请求,以请求该歌曲。接着(第 3 步和第 4 步),服务器从磁盘上取出该歌曲(只不过是一个 MP3 或其他某种格式的文件)并将它送回给浏览器。如果该文件大于服务器的内存,则 Web 服务器有可能每次取出一块音乐并发送出去。

由于使用了 MIME 类型,例如 audio/mp3,(或者通过文件扩展名),浏览器查找当前系统以确定该如何显示此文件。通常,系统中存在一个诸如 RealOne Player、Windows Media Player 或 Winamp 之类的辅助应用程序与这种类型的文件关联在一起。由于浏览器与辅助应用程序的一般通信方法是将内容写入到一个临时文件中,因此浏览器首先将整个音乐文件当作一个临时文件保存到磁盘上(第 5 步)。然后,浏览器启动媒体播放器并将临时文件的名称传递给它。在第 6 步中,媒体播放器开始逐块地取出并播放音乐。

原则上,这种方法完全正确并且能够播放音乐。惟一的问题是,在播放音乐之前,必须通过网络将整首歌曲传送到客户端。如果这首歌曲是 4MB 大小(一首 MP3 歌曲的典型大小),并且调制解调器是 56kbps,那么在下载歌曲时,用户将面对差不多 10 分钟左右的寂静。并非所有的音乐爱好者都喜欢这种方式,特别是因为下一首歌曲也将在 10 分钟的下载时间以后才开始播放,再下一首歌仍然这样。

为了避免这个问题并且不改变浏览器的工作方式,音乐站点已经提出并采用了下面的方案。链接到歌曲标题的文件并不是实际的音乐文件,而是一个被称为元文件(metafile)的文件,它非常短,仅仅指定了该音乐的名字。一个典型的元文件可能只有一行 ASCII 文本,看上去像这样:

```
rtsp://joes-audio-server/song-0025.mp3
```

当浏览器得到了这个 1 行长的文件时,它将文件写入到磁盘上的一个临时文件中,再按照辅助应用程序的方式启动媒体播放器,并像往常一样将临时文件的名称传递给它。然后,媒体播放器读取文件,发现它包含了一个 URL。接着,播放器与 joes-audio-server 联系,并请求该歌曲。请注意,浏览器不再介入到此交互过程中。

在大多数情况下,元文件中给出的服务器与 Web 服务器并不相同。事实上,通常它根本就不是 HTTP 服务器,而是一个专门的媒体服务器。在本例中,正如方案名 rtsp 所指示的那样,媒体服务器使用 RTSP(Real Time Streaming Protocol,实时流协议)。RFC 2326 描述了 RTSP 协议。

媒体播放器主要完成以下四项工作:

- (1) 管理用户界面。
- (2) 处理传输错误。
- (3) 解压缩音乐。
- (4) 消除抖动。

现在的大多数媒体播放器都有华丽的用户界面,有时甚至模拟一个立体声音响,面板上带有按钮、旋钮、调节滑块和可视显示窗等。通常播放器的面板可被替换,这样的面板被称为皮肤(skin),用户可以将设计好的皮肤装饰到播放器上。媒体播放器必须管理所有这些事项,并且与用户进行交互。

媒体播放器的第二项任务是处理错误。由于错误和重传有可能导致在音乐中产生不可接受的长时间间隙,所以,实时音乐传输很少使用 TCP。相反,实际的传输往往通过一个类似于我们在第 6 章中学习过的 RTP 的协议来完成。与大多数实时协议一样,RTP 位于 UDP 之上的一层,所以分组可能会丢失。分组丢失的情况由播放器来处理。

在有些情况下,通过交替方式来发送音乐,可使错误处理更加容易。例如,一个分组可能包含 220 个立体声采样,其中每个采样包含一对 16 位的数,这对于 5ms 长的音乐来说通常已经非常不错了。但是传输协议可能在一个分组中发送 10ms 间隔中的所有奇数采样,然后在下一个分组中发送所有的偶数采样。因此,丢失一个分组并不意味着当前音乐中有 5ms 的间隙,而是 10ms 内每隔一个采样就丢失一个采样。媒体播放器可以通过插值的方法,利用前一个采样和后一个采样来估算出丢失的值,从而很容易就可以处理分组丢失的情形。

图 7.60 举例说明了通过交错发送来实现错误恢复的做法。在这里,每个分组包含了 10ms 时间内交替出现的采样值。因此,如图中所示,丢失了分组 3 并不会导致在音乐中产生一个间隙,而只是降低了一段时间间隔内的临时分辨率。通过插值的方法可以计算出这些丢失的值,从而获得连续的音乐。这种特殊的方案只能用于未压缩的采样信号,但是它表明了,通过巧妙的编码可以将丢失分组的影响转变成较低的质量,而不是一段间隙。然而,RFC 3119 给出了一种针对压缩音频的方案。

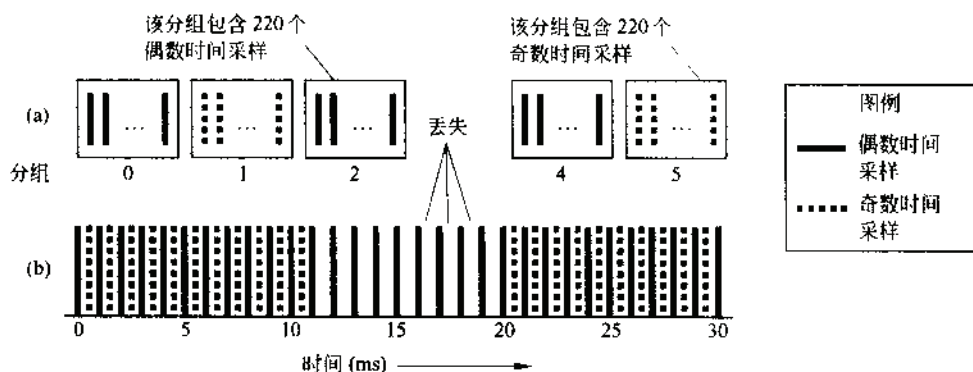


图 7.60 当分组携带的是交替出现的采样时,一个分组的丢失只是降低了临时分辨率,而没有产生时间上的间隙

媒体播放器的第三项任务是解压缩音乐。尽管这项任务是计算密集型的,但它同时也是相当简单而又直接的。

第四项任务是消除抖动,抖动是所有实时系统的致命弱点。如图 7.61 所示,所有的流式音频系统都首先缓存大约 10~15 秒音乐,然后再开始播放。在理想情况下,服务器将持续地以媒体播放器消耗缓冲区数据的速度来填充缓冲区,但是在实践中这可能做不到,因此在播放过程中反馈信息也许会非常有用。

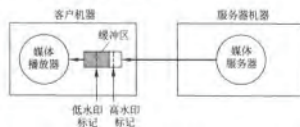


图 7.61 媒体播放器将来自媒体服务器的输入缓存起来,并且播放缓冲区中的数据,而不是直接播放来自网络的数据

通过两种方法可以使缓冲区总是能得到填充。利用拉式服务器(pull server)方法,只要缓冲区中还有空间能存放另一个数据块,则媒体播放器就不停地向服务器发送请求,要求一个新的数据块。它的目标是尽可能地保持缓冲区是满的。

拉式服务器的缺点是产生大量不必要的数据请求。服务器知道它已发送了整个文件,所以,为什么还要让播放器不停地请求呢?由于这个原因,这种方法很少被真正使用。

利用推式服务器(push server)方法,媒体播放器发送一个 PLAY 请求,服务器只是不停地向播放器推送数据。这里有两种可能:媒体服务器以正常的播放速度来运行,或运行得更快。在这两种情况下,媒体播放器在开始播放之前都要先缓存一些数据。如果服务器以正常的播放速度来发送数据,那么,服务器发送过来的数据被添加到缓冲区的尾部,而播放器从缓冲区的前端取出数据来播放。只要一切工作正常,则缓冲区内的数据量始终保持不变。由于在任一方向上都不需要控制消息,因此这种方案非常简单。

另一种推式方案是让服务器发送数据的速度快于播放器需要数据的速度,它的优点是,如果服务器不能保证总是以均匀的速度来发送数据,那么即使它落后一段时间,它仍然还有机会追上来。然而,这里的一个问题是,如果服务器发送数据的速度超过了数据被播放的速度(服务器必须能够这样做,以避免产生间隙),则缓冲区有可能会溢出。

解决的方案是,媒体播放器为缓冲区定义一个低水印标记(low-water mark)和一个高水印标记(high-water mark)。基本而言,服务器往外发送数据直至缓冲区被填至高水印标记,然后媒体播放器告诉服务器暂停发送。在服务器接收到暂停请求以前,数据仍将源源不断地到达客户端,因此,缓冲区的高水印标记与尾部之间的距离必须大于网络的带宽-延迟之乘积。当服务器停止发送数据以后,缓冲区将开始变得空起来。当它到达低水印标记时,媒体播放器告诉服务器继续开始发送数据。在设置低水印标记的位置时必须确保缓冲区不会出现数据供应不足的情况。



为了运行一台推式服务器,媒体播放器需要对它进行远程控制。这正是 RTSP 所提供的功能。RTSP 被定义在 RFC 2326 中,它提供了一些机制可让播放器来控制服务器,但它没有提供数据流机制,通常数据流功能由 RTP 来完成。图 7.62 中列出了 RTSP 提供的主要命令。

命令	服务器的操作
DESCRIBE	列出媒体参数
SETUP	在播放器与服务器之间建立一条逻辑信道
PLAY	开始向客户发送数据
RECORD	客户开始接受数据
PAUSE	暂时停止发送数据
TEARDOWN	释放逻辑信道

图 7.62 播放器发送给服务器的 RTSP 命令

#### 7.4.4 Internet 电台

一旦流式音频在 Internet 上成为可能,商业的广播电台就想到了将它们的内容同时也通过 Internet 广播到世界各地,而不仅限于在空中广播。此后不久,大学的广播电台开始把它们的信号放到 Internet 上。然后,大学生们开始建立他们自己的广播电台。有了现在的技术,事实上任何人都可以建立一个广播电台。Internet 电台是一个非常新的应用,并且仍然处于不断变化的状态之中,但仍然值得对它做一番介绍。

一般有两种实现 Internet 电台的方法。在第一种方法中,节目被预先录制好,并存放在磁盘上。收听者可以连接到电台的资料库中,找到任何一个节目并下载下来收听。实际上,这与我们刚刚讨论过的流式音频完全一样。对于直播节目,也可以在每个节目刚刚被直播之后就把它存储起来,所以,收听者仅仅在实况播出(比如说)半小时或更短的时间以后就可以通过资料库来收听了。这种方法的优点是易于实现,我们已经讨论过的所有技术都可以用在这里,收听者可以精心挑选资料库中的所有节目。

另一种方法是通过 Internet 进行节目直播。有一些电台同时在空中和 Internet 上广播,但越来越多的电台只在 Internet 上广播。一些可用于流式音频的技术也可以用于 Internet 直接广播,但这里也有一些关键的区别。

相同的一点是,都需要在用户端进行缓存以消除抖动。在开始播放之前先收集 10 或 15 秒的广播内容,于是,即使而临网络上的抖动,音频流也可以平稳地继续播放下去。只要所有的分组都在它们被播放之前到达,那么它们何时到达并不重要。

一个关键的区别是,由于当到达缓冲区的高水印标记时接收方可能会停止音频流,因此发送方可以用大于播放速率的速率来向外推出流式音频数据。因此,这就有可能利用这段时间来重传丢失的分组,不过,这种策略并不常用。相反地,直播电台总是完全以节目被生成和播放的速率来进行广播。

另外一个区别是,直播电台通常同时有几百名或几千名收听者,而流式音频却是点到



点的。在这种情况下,Internet 电台应该使用多播以及 RTP/RTSP 协议。这显然是最有效的运行方式。

在当前的实践中,Internet 电台并不是这样工作的。实际的情形是,用户与电台建立一个 TCP 连接,然后在该 TCP 连接上传送直播内容。当然,这引发了许多问题,比如当窗口满了以后数据流就停止了,丢失分组的超时和重传等。

之所以不使用 RTP 多播而使用 TCP 单播的原因有三个方面。首先,很少有 ISP 支持多播,因此这不是一个很实际的选择。其次,RTP 不像 TCP 那样广为人知,广播电台通常都很小,缺乏计算机专业知识,因此,使用一个大家都充分理解并且所有软件包都支持的协议会更加容易一些。第三,许多人在工作时收听 Internet 电台,这在实践中往往意味着他们位于防火墙的后面。大多数系统管理员都会配置他们的防火墙,以保护他们的 LAN 免于遭到不受欢迎的外来者的访问。他们通常允许远程 25 端口的 TCP 连接(用于电子邮件的 SMTP)、远程 53 端口的 UDP 分组(DNS),以及远程 80 端口的 TCP 连接(用于 Web 的 HTTP)。其他的一切(包括 RTP)几乎都被阻止了。因此,使电台信号穿过防火墙的惟一方法是,让电台站点伪装成一个 HTTP 服务器(至少对防火墙而言是这样),然后通过 HTTP 服务器(位于 TCP 之上)来传输音频信号。这些严格的措施仅仅提供了最少的安全性,通常还迫使多媒体应用进入一种非常低效的运行模式之中。

由于 Internet 电台是一种新的传播媒介,所以格式之战正酣。RealAudio、Windows Media Audio 和 MP3 正在该市场中激烈地竞争,它们都期望成为 Internet 电台的主导格式。Vorbis 是一种新的格式,在技术上类似于 MP3,但它是开放源码的,并且非常不同的是,它没有使用 MP3 所依赖的专利。

一个典型的 Internet 电台有一个 Web 页面,其中列出了它的时间表、关于其 DJ 和播音员的信息,以及许多广告。通常还有一个或多个图标列出了它所支持的格式(或者,如果只支持一种格式的话,则仅仅是“现在收听[LISTEN NOW]”)。这些图标或“现在收听”被链接到前面讨论过的元文件上。

当用户单击其中某一个图标时,短小的元文件被发送过来,浏览器根据它的 MIME 类型或文件扩展名来决定合适的辅助应用程序(即媒体播放器)。然后,浏览器将元文件写入磁盘上的一个临时文件中,启动媒体播放器,并将该临时文件的名字传递给播放器。媒体播放器读取临时文件,看到了它里面所包含的 URL(通常采用 http 方案而不是 rtsp 方案以避免防火墙问题,并且一些流行的多媒体应用就是用这种方法来工作的),然后它与服务器联系,并开始像电台一样地工作。顺便提一下,音频只有一个流,因此 http 能够工作,但对于视频,因为它至少有两个流,所以 http 就无法工作了,因此像 rtsp 这样的技术确实是有必要的。

在 Internet 电台领域中,另外一个有趣的发展是,任何人,即使是学生,也可以建立并运行一个广播电台。图 7.63 显示了其中主要的部件。电台的基础是一台带有声卡和麦克风的普通 PC。软件部分包含一个媒体播放器(比如 Winamp 或 Freeamp)、捕获音频的插件,以及所选择的输出格式的编解码器,例如 MP3 或 Vorbis。

该电台生成的音频流被通过 Internet 传送到一台大一点的服务器上,该服务器负责将它分发给大量的 TCP 连接。这台服务器往往支持许多小的电台。它也维护了一个目

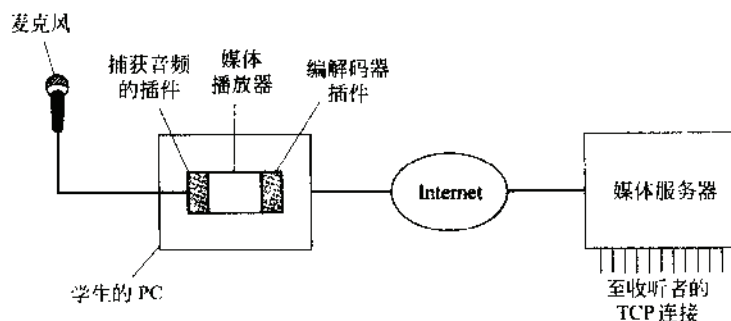


图 7.63 一个学生广播电台

录,其中包含它有哪些电台以及每个电台当前正在播送什么内容。潜在的收听者来到该服务器,然后选择一个电台并得到一个 TCP 回馈。有一些商业软件包可以管理所有这些功能,同时也有一些像 icecast 这样的开放源码软件包。还有一些服务器愿意在收取一定费用后负责分发音频信号。

#### 7.4.5 IP 语音

在很久以前,公共交换电话系统主要用于语音通信,同时各个地方有少量的数据通信。但数据通信增长得非常快,到 1999 年,数据通信的数量(位数)赶上了语音通信的数量(由于在主干线上采用 PCM[脉冲编码调制]来表示语音,所以语音通信也可以用每秒多少位来衡量)。到 2002 年,数据通信量已经比语音通信量多出了一个数量级,并且仍然在以指数级增长,而与此同时,语音通信的增长几乎是平坦的(每年增长 5%)。

这些数字导致的一个结果是,许多分组交换网络运行商突然之间对于在他们的数据网络上承载语音产生了浓厚的兴趣。分组网络是为数据通信而设计的,而语音所要求的额外带宽量又很小。然而,一般人的电话费账单可能高于他的上网费账单,因此数据网络运行商看到了,Internet 电话是一种能赚取大量额外收入的方法,同时又不必在地下铺设任何新的光纤。于是,Internet 电话(Internet telephony)(也被称为 IP 语音,voice over IP)诞生了。

#### H. 323

有一件事情是每个人从一开始就很清楚的,那就是,如果每个厂商都设计它自己的协议栈,那么系统将无法工作。为了避免这个问题,许多感兴趣的团体在 ITU 的倡导下,聚集到一起制定标准。1996 年,ITU 发布了推荐标准 H. 323,其标题是“针对无服务保障的局域网络的可视电话系统与设备[Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-Guaranteed Quality of Service]”。只有电话工业才会想得这样一个名字。1998 年,该推荐标准又进行了修订,修订后的 H. 323 是第一批广泛应用的 Internet 电话系统的基础。

与其说 H. 323 是一个特定的协议,不如说它是 Internet 电话的总体结构描述。在语音编码、呼叫建立、信号机制、数据传输和其他领域中,H. 323 引用了大量特定的协议,而

不是自己重新制定一套新的规定。图 7.64 描述了它的总体模型。在中心处是一个网关 (gateway)，它将 Internet 与电话网络连接起来。网关在 Internet 一端使用 H.323 协议，在电话端则使用 PSTN 协议。通信设备被称为终端 (terminal)。一个 LAN 可能有一个网守 (gatekeeper)，它控制其管辖范围内的端点，此管辖范围被称为区域 (zone)。



图 7.64 H.323 的 Internet 电话结构模型

一个电话网络需要许多协议。首先，需要有一个协议来对语音进行编码和解码。我们在第 2 章中学习过的 PCM 系统被定义在 ITU 的推荐标准 G.711 中。它以每秒 8000 次的 8 位采样对单个语音通道进行编码，在未压缩语音的情况下可以达到 64kbps。所有的 H.323 系统必须支持 G.711。然而，其他的语音压缩协议也是允许的（但不是必需的）。这些协议采用不同的压缩算法，并在质量和带宽之间作不同的平衡。例如，G.723.1 接受一个包含 240 个采样 (30ms) 的语音的数据块，并且使用预测编码以便将它减少为 24 字节或 20 字节。该算法得到的输出率分别为 6.4kbps 或 5.3kbps (压缩因子为 10 和 12)，并且只有少许可觉察的品质损失。H.323 还允许使用其他的编解码器。

由于允许使用多个压缩算法，所以需要有一个协议，以便让多个终端来协商使用哪一个算法，该协议被称为 H.245。它还能协商一个连接的其他方面，比如数据速率。RTCP 被用来控制 RTP 信道，还需要有一个协议来建立和释放连接、提供拨号音、产生响铃音以及完成标准电话的其他功能，这里使用的是 ITU Q.931。终端需要一个协议来跟网守（如果有的话）进行通信，H.225 可作为此用途。它所管理的从 PC 到网关的信道被称为 RAS (Registration/Admission/Status, 注册/许可/状态) 信道。该信道允许终端加入和离开区域、请求和返回带宽、提供状态更新，以及其他一些事情。最后，还需要有一个协议用于实际的数据传输，RTP 可作为此用途。如同往常一样它是由 RTCP 来管理的。所有这些协议的位置关系如图 7.65 所示。

语音	控 制			
G. 7xx	RTCP	H. 225 (RAS)	Q. 931 (呼叫信令)	H. 245 (呼叫控制)
RTP				
UDP			TCP	
IP				
数据链路协议				
物理层协议				

图 7.65 H.323 协议栈

为了看清楚这些协议是如何相互配合的,请考虑这样的情形:一个 LAN(有一个网守)中的 PC 终端呼叫一部远程电话。该 PC 首先必须找到网守,因此它向 1718 端口广播一个 UDP 网守探查分组。当网守有了回应以后,PC 就知道了网守的 IP 地址。现在,PC 在一个 UDP 分组中向网守发送一条 RAS 消息,以便注册到网守中。该消息被接受以后,PC 向网守发送一条 RAS 许可消息以请求带宽。只有当分配了带宽以后,PC 才可以开始建立呼叫。之所以采用事先请求带宽的做法,是为了让网守限制呼叫的数量,以避免引出线被超额使用,从而有助于提供必要的服务质量。

现在,PC 建立一个至网守的 TCP 连接以便开始建立呼叫。在建立呼叫时 PC 使用现有的而向连接的电话网络协议,因此需要使用 TCP。相反地,由于电话系统没有像 RAS 这样的能够让电话机宣布其存在性的协议,所以 H. 323 的设计者可以随便使用 UDP 或 TCP 来实现 RAS,最终他们选择了开销较小的 UDP。

既然该 PC 已经被分配了带宽,它就可以通过此 TCP 连接发送一条 Q. 931 SETUP 消息,该消息指定了被呼叫的电话的号码(如果被呼叫的是一台计算机的话,则是 IP 地址和端口)。网守回应一条 Q. 931 CALL PROCEEDING 消息,以确认它已正确地收到了请求。然后网守将 SETUP 消息转发给网关。

网关的一半是计算机,另一半是电话交换机,它接到了网守转发过来的 SETUP 消息以后,向目标(普通)电话发出一个普通的电话呼叫。目标电话所在的端局使被呼叫的电话发出响铃声,该端局同时也送回一条 Q. 931 ALERT 消息,以告诉呼叫方 PC,电话已经开始响铃了。当另一端的人拿起电话时,端局送回一条 Q. 931 CONNECT 消息,以通知 PC,它已经有了一个连接。

一旦连接已经被建立起来,则网守就不再出现在通信过程中,当然网关仍然还在。后续的分组将绕过网守,直接送达网关的 IP 地址。到此时,我们已经有了一个运行于两方之间的直接管道。这只是一个用来传输数据位的物理层连接,仅此而已。任何一方对另一方都一无所知。

现在使用 H. 245 协议来协商当前呼叫的参数,这里使用的是 H. 245 控制信道,它始终是打开的。每一方都从宣布它的处理能力开始,例如,它是否能处理视频(H. 323 可以处理视频)或会议呼叫、它支持哪些编解码器,等等。一旦每一端知道了另一端具备哪些处理能力以后,两个单向的数据信道被建立起来,并且每个信道被指定一个编解码器和其他一些参数。由于每一端可能有不同的设备,所以前向和逆向信道上的编解码器有可能完全不同。当所有的协商工作都完成以后,就可以开始用 RTP 来传送数据流了。RTP 的管理由 RTCP 来完成,RTCP 在拥塞控制中起着重要的作用。如果有视频的话,RTCP 还要处理音频/视频的同步。图 7.66 显示了各种信道。当任何一方挂断电话时,通过 Q. 931 呼叫信令信道可以拆除连接。

当呼叫结束时,呼叫方 PC 再次用一条 RAS 消息与网守联系,以释放分配给它的带宽。或者,它也可以启动另一次呼叫。

虽然服务质量(QoS)对于 IP 语音的成功是至关重要的,但我们还没有讨论有关服务质量的话题。理由是,QoS 超出了 H. 323 的范围。如果底层网络能够提供一个从发起呼叫的 PC 到网关之间的、稳定而无抖动的连接(例如,使用我们在第 5 章中讨论过的技

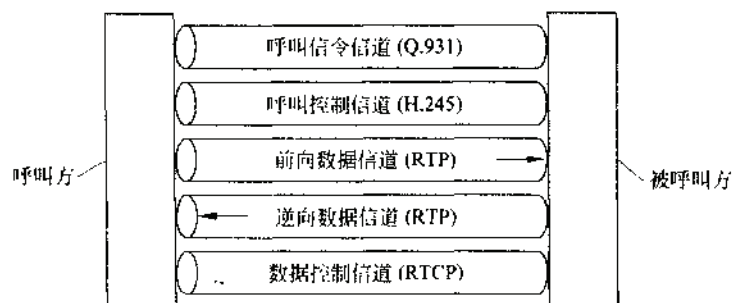


图 7.66 在一次呼叫过程中,呼叫方与被呼叫方之间的逻辑信道

术),那么该呼叫的服务质量就会很好;否则,服务质量就无法保证。电话部分使用了 PCM,并且总是无抖动的。

### SIP—会话发起协议

H. 323 是由 ITU 设计的,Internet 社团中的许多人把它看成一种典型的电信产品:庞大、复杂而且不灵活。因此,IETF 建立了一个委员会,由该委员会设计一种更加简单和模块化的方法来实现 IP 语音。到目前为止的主要成果是 SIP (Session Initiation Protocol,会话发起协议),RFC 3261 描述了 SIP 协议。该协议讲述了如何建立 Internet 电话呼叫、视频会议和其他的多媒体连接。H. 323 是一个完整的协议集合,与此不同的是,SIP 是单个模块,但是它被设计成能很好地与现有的 Internet 应用协同工作。例如,它将电话号码定义成 URL,所以 Web 页面可以包含电话号码,并允许用户单击一个链接就可以发起一次电话呼叫(与 mailto 方案的做法相同:mailto 允许用户单击一个链接就可以启动一个程序来发送电子邮件消息)。

SIP 可以建立两方会话(普通的电话呼叫)、多方会话(每个人都可以听和说),以及多播会话(一个发送方,多个接收方)。这些会话可以包含音频、视频或数据,后者对于诸如多方实时游戏之类的应用非常有用。SIP 只处理会话的建立、管理和终止。诸如 RTP/RTCP 之类的其他协议被用于数据传输。SIP 是一个应用层协议,它可以运行在 UDP 或 TCP 之上。

SIP 支持多种服务,包括查找被呼叫方的位置(他可能不在他家里的计算机旁边)、确定被呼叫方的处理能力,以及处理呼叫建立和终止的细节。在最简单的情形中,SIP 建立一个从呼叫方计算机到被呼叫方计算机之间的会话,所以我们首先来讨论这种情形。

SIP 中的电话号码被表示成 sip 方案的 URL,例如,sip:ilse@cs. university. edu 代表一个由 DNS 名 cs. university. edu 指定的主机上的名叫 Ilse 的用户。SIP 的 URL 也可以包含 IPv4 地址、IPv6 地址或者实际的电话号码。

SIP 协议是一个仿照 HTTP 的基于文本的协议。一方以 ASCII 文本的形式发送一条消息,消息的第一行包含一个方法名,接下来的几行包含一些用于传递参数的消息头。很多消息头都来自于 MIME,以便 SIP 能与现有的 Internet 应用协同工作。图 7.67 中列出了核心规范中定义的六种方法。

方法	说明
INVITE	请求发起一个会话
ACK	确认一个会话已经被启动
BYE	请求终止一个会话
OPTIONS	向一台主机查询它的处理能力
CANCEL	取消一个悬着的请求
REGISTER	将用户的当前位置告诉一个重定向服务器

图 7.67 在核心规范中定义的 SIP 方法

为了建立一个会话,呼叫方要么与被呼叫方建立一个 TCP 连接并在其上发送一条 INVITE 消息,要么在一个 UDP 分组中发送 INVITE 消息。在这两种情况下,第二行和随后各行中的消息头描述了消息体的结构,其中包含呼叫方的处理能力、媒体类型和格式。如果被呼叫方接受该呼叫,那么它回应一个 HTTP 类型的回复码(使用图 7.42 中分类的三位数字,200 表示接受)。在回复码行的后面,被呼叫方还可以提供有关它的处理能力、媒体类型和格式的信息。

利用三步握手方法,连接就完成了,所以,呼叫方回应一条 ACK 消息来结束此协议,并确认已收到了 200 消息。

双方中的任一方都可以通过发送一条包含 BYE 方法的消息来结束当前会话。当另一方确认了该消息时,会话就结束了。

OPTIONS 方法被用于向一台机器查询它的处理能力,通常呼叫方在发起一个会话前使用 OPTIONS 方法来查明该机器是否能处理 IP 语音,或者期望什么类型的会话。

REGISTER 方法涉及到 SIP 能够追踪和连接到一个不在家里的用户。该消息被发送到一个 SIP 位置服务器,位置服务器记录下每个人分别在哪里,所以,以后客户可以向它查询以找到某个用户的当前位置。重定向的过程如图 7.68 所示。在这里,呼叫方将 INVITE 消息发送给一个代理服务器,以便把可能的重定向过程隐藏起来。然后代理服务器查找被呼叫的用户在哪里,并将 INVITE 消息发送至该处。然后,代理服务器为后续的三步握手过程中的消息充当中继节点。LOOKUP 和 REPLY 消息并不是 SIP 的一



图 7.68 与 SIP 一起使用代理服务器和重定向服务器



部分;这里可以使用任何方便的协议,取决于使用了什么类型的位置服务器。

SIP 还有许多其他的特性,其中包括呼叫等待、呼叫屏蔽、加密和认证等,我们这里不再一一描述。如果在 Internet 和电话系统之间有合适可用的网关,则 SIP 还能够从计算机向普通电话发起呼叫。

### H. 323 与 SIP 的比较

H. 323 与 SIP 有许多相似之处,但也有一些不同之处。它们都允许用计算机和电话作为端点来实现两方和多方呼叫。它们都支持参数协商、加密和 RTP/RTCP 协议。图 7.69 总结了它们之间的相似之处和不同之处。

项目	H. 323	SIP
由谁设计的	ITU	IETF
与 PSTN 的兼容性	是	很大程度上
与 Internet 的兼容性	否	是
体系结构	整体的	模块化的
完整性	完整的协议栈	SIP 只处理会话建立
参数协商	是	是
呼叫信令	TCP 之上的 Q.931	TCP 或 UDP 之上的 SIP
消息格式	二进制	ASCII
媒体传输	RTP/RTCP	RTP/RTCP
多方呼叫	是	是
多媒体会议	是	否
地址编址	主机或电话号码	URL
呼叫终止	显式地,或 TCP 连接被释放	显式地或超时
即时消息	否	是
加密	是	是
标准文档的厚度	1400 页	250 页
实现	庞大而复杂	适中
当前状况	已被广泛部署	正在推进,有发展前途

图 7.69 H. 323 与 SIP 的比较

尽管两个协议的特征集是类似的,但它们的基本思想却相差甚远。H. 323 是一个典型的重量级的电话工业标准,它规定了完整的协议栈,并精确地定义了什么是允许的,以及什么是禁止的。这种方法使得在每一层上都有定义完好的协议,从而使互操作性的任务变得容易了。但是,其付出的代价是一个庞大的、复杂的、严格的,而且难以适应未来应用的标准。

与此相反, SIP 是一个典型的 Internet 协议, 它通过交换短短几行 ASCII 文本就可以工作。它是一个轻量级的模块, 与其他的 Internet 协议协同工作得很好, 但是与现有的电话系统信令协议一起工作则稍差一些。因为 IETF 的 IP 语音模型是高度模块化的, 所以它很灵活, 并且很容易适用于新的应用。其不利的方面是潜在的互操作性问题, 不过, 时常有一些会议将不同的实现放在一起以测试这些系统, 从而设法解决它们之间的互操作性问题。

IP 语音是一个正在不断发展并且很有前途的主题。因此, 关于该主题已经有几本书出版了, 例如 (Collins, 2001; Davidson and Peters, 2000; Kumar et al., 2001; Wright, 2001)。2002 年第 5/6 期的 Internet Computing 杂志上有几篇关于该话题的文章。

#### 7.4.6 视频简介

目前我们已经详细地讨论了耳朵, 现在该转移到眼睛上了 (不, 在这一部分后面不再有关于鼻子的讨论了)。人眼有这样一种特性: 当一幅图像出现在视网膜上时, 该图像在衰退下去之前将会保留几毫秒时间。如果一个图像序列以每秒 50 幅图像的速度被逐行绘制, 则眼睛将不会察觉到它看到的是离散的图像。所有的视频 (即电视) 系统都利用这个原理来产生活动的图像。

##### 模拟系统

为了理解视频, 最好从简单的、过时的黑白电视开始讲起。摄像机为了将自己面前的二维图像用一个随时间变化的一维电压函数来表示, 它用一个电子束快速地水平穿越图像并慢慢地向下扫描, 在扫描的同时记录下光的强度。在一次扫描结束时, 电子束又重新折回去, 这样的一次扫描被称为一帧 (frame)。光的强度被作为时间的函数广播出去, 接收者重复此扫描过程以便重建图像。图 7.70 显示了照像机和接收者使用的扫描模式。(顺便提一下, CCD 照像机采用整体照射而不是扫描, 但是有些照像机和所有的监视器都

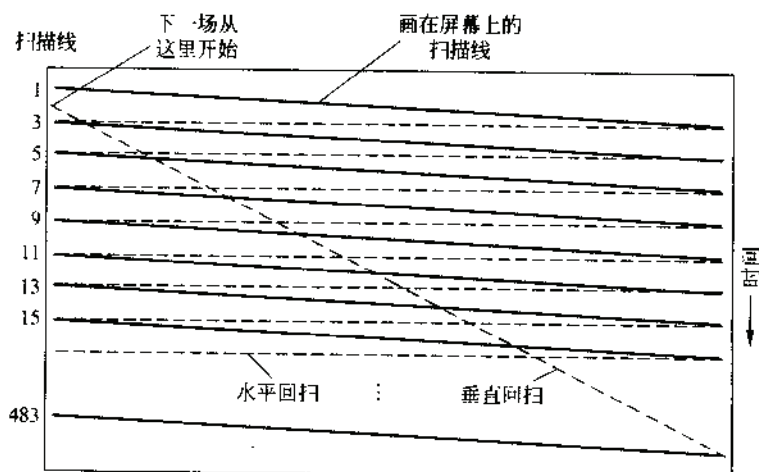


图 7.70 NTSC 视频和电视所使用的扫描模式

采用扫描的方法)。

确切的扫描参数随每个国家而有所不同。北美和南美以及日本使用的系统有 525 条扫描线,水平与垂直的比例是 4:3,而且每秒 30 帧。欧洲使用的系统有 625 条扫描线,水平与垂直的比例同样是 4:3,但每秒 25 帧。在这两种系统中,最上面和最下面的少量扫描线不会被显示出来(以便在原来圆角的 CRT 上近似成一个矩形图像),NTSC 系统的 525 条扫描线中的 483 条(PAL/SECAM 系统的 625 条扫描线中的 576 条)被显示出来。电子束在垂直折回过程中被关闭,所以有些电视台(特别是在欧洲)利用这段时间来广播图文电视(指文本页面,包含新闻、天气、体育、股票价格等信息)。

尽管每秒 25 帧的速度足够获得平滑的运动效果,但是在这种帧速率下,许多人,特别是年纪较大的人,会感觉到图像闪烁(因为在新图像出现以前老图像已经从视网膜上褪去)。为了解决这个问题,当然可以增大帧速率,但是这将要求更多本来就很稀缺的带宽,所以人们想出了另一种方法。这种方法并不是按顺序显示所有的扫描行,而是首先显示所有的奇数行,然后显示偶数行。这里的每半帧图像称为场(field)。实验证明,尽管在 25 帧/秒时人们会感觉到闪烁,但是在 50 场/秒时却感觉不到,这项技术称为隔行扫描(interlacing)。非隔行扫描的电视或者视频称为逐行扫描(progressive)。请注意,电影的运行速度是 24 帧/秒,但是每一帧在 1/24 秒期间是完全可见的。

彩色视频使用的扫描模型与单色(黑白)视频相同,只不过它并不是采用一个电子束,而是采用三个电子束来同步地显示图像。每个电子束分别被用于加色系中的三原色之一:红色、绿色和蓝色(RGB)。这项技术之所以能够奏效,是因为任何颜色都可以用适当强度的红色、绿色和蓝色经线性叠加而构造得来。然而,为了在单一的信道上进行传输,三种颜色信号必须被组合到单个复合(composite)信号中。

当彩色电视刚刚被发明时,有许多种显示颜色的方法在技术上是可行的,所以,不同的国家做出了不同的选择,从而也导致这些系统至今仍互不兼容。(请注意,这些选择与 VHS、Betamax 和 P2000 没有任何关系,它们是录制方法)。在所有的国家中,一个政策性的需求是,在现有的黑白电视机上必须能够收看那些以彩色方式进行传播的节目。因此,最简单的方案,即独立地对 RGB 信号进行编码,是不可接受的。另外,RGB 也不是最有效的编码方案。

第一个彩色电视系统是在美国由国家电视标准委员会(National Television Standards Committee)制定的,这也导致了该标准的首字母缩略形式为:NTSC。几年以后,彩色电视被引入到欧洲,此时技术已经有了充分的提高,从而导致产生了抗干扰性更强和色彩更好的系统。这样的系统有:SECAM(SEquentiel Couleur Avec Memoire,顺序与存储彩色电视系统),它被用于法国和东欧;PAL(Phase Alternating Line,逐行倒相制式),它被用于欧洲的其他地方。NTSC 和 PAL/SECAM 之间的彩色质量差异导致了一个工业界笑话:NTSC 真正代表的是“从来没有两次相同的颜色[Never Twice the Same Color]”。

为了使彩色电视信号可被显示在黑白电视机上,这三种系统都按线性方式把 RGB 信号组合成一个亮度(luminance)(光亮)信号和两个色度(chrominance)(颜色)信号,但是由 RGB 信号来构造这些信号时,它们分别使用了不同的系数。很奇怪的是,眼睛对亮度信号比对色度信号敏感得多,所以后者不需要被精确地传输。因此,亮度信号可以在原来

老的黑白信号所使用的频率上被广播出去,所以它能够被黑白电视机接收。两个色度信号则在较高的频率上用较窄的频段来广播。有些电视机包含三个被标记为亮度、色调和饱和度(或者亮度、色彩和颜色)的控制钮,分别用于控制这三个信号。理解亮度和色度对于理解视频压缩的工作原理是十分有必要的。

在过去几年中,人们对 HDTV(High Definition TeleVision, 高清晰度电视)有非常浓厚的兴趣,它大约增加了一倍扫描线的数量,从而产生出更加精细的图像。美国、欧洲和日本都已经研制出了 HDTV 系统,它们各不相同,而且互不兼容。你还期望有其他的 HDTV 系统吗?从扫描机理、亮度、色度等各个方面来看,HDTV 的基本原理与现有的系统非常相似。然而,所有这三种格式有一个共同的屏幕宽高比 16:9,而不再是 4:3,目的是为了更地吻合电影所使用的格式(电影被录制在 35mm 的胶片上,其宽高比为 3:2)。

### 数字系统

数字视频最简单的表示法是一个帧序列,每个帧包含一个由图像元素或者像素(pixel)组成的矩形网格。每个像素可以是一个单独的位,以便表示黑或者白。这样一个系统的质量与你通过传真发送一幅彩色照片所得到的结果非常类似——效果极差。(如果有条件的话你可以试一试;否则在一台非光栅化的复印机上复印一张彩色照片来看看)。

接下来一步是,每个像素用 8 位来表示 256 个灰度级。这种方案可以表示高质量的黑白视频。对于彩色视频,好的系统使用 8 位来表示 RGB 颜色中的每一个分量,不过几乎所有的系统都把这三个分量混合到复合视频中来传输。由于每个像素使用 24 位来表示,所以总的颜色数量被限制在 16 兆种左右,然而人眼并不能分辨这么多种颜色,更别提再多的颜色了。数字彩色图像是通过三个扫描束而产生的,每种颜色各一个。扫描的路线与图 7.70 所示的模拟系统一样,只不过连续的扫描线被整齐离散像素行所替代罢了。

为了产生平滑的运动,数字视频与模拟视频一样,也必须至少每秒显示 25 帧。然而,由于高质量的计算机监视器通常每秒 75 次或者更多次地重新扫描内存中的图像,所以隔行扫描技术不再有必要了,因此往往不再被使用。只需每秒钟重复三次绘制(即重画)同一帧就足以消除闪烁了。

换句话说,运动的平滑性是由每秒钟显示的不同图像的数量来决定的,而闪烁则是由每秒钟屏幕被重画的次数来决定的。这两个参数是不同的。一幅静止图像以 20 帧/秒的速度被绘制,这并不能表现出运动,但是它将会闪烁,因为在下一帧出现以前,上一帧将从视网膜上消失。如果一部电影每秒钟有 20 个不同的帧,则每一帧的每一行被绘制四次,所以,它不会闪烁,但是运动过程将显得不平滑。

当我们考虑在网络上传输数字视频所需要的带宽时,这两个参数的重要性就会显著地体现出来。当前的计算机监视器大多采用 4:3 的屏幕宽高比,所以它们可以使用专门为消费者电视市场而大规模生产的、便宜的显像管。通常的配置是 1024×768、1280×960 和 1600×1200。即使支持 24 位/像素和 25 帧/秒的最小显示器也需要 472Mbps 带

宽。这需要采用 SONET OC-12 线路才能满足要求,但是把 OC-12 SONET 线路连接到每个人的家里显然还为时尚早。通过使速率加倍来避免闪烁的做法就更没有吸引力了。一种更好的解决方案是每秒传输 25 帧,而让计算机存储每一帧并绘制两次。广播电视并没有使用这种策略,因为电视机没有内存。即使它们有内存,模拟信号如果不先被转换成数字形式的话,也不可能被存储到 RAM 中;而如果要执行转换的话,则要求额外的硬件支持。因此,隔行扫描对于广播电视来说是必需的,但对于数字视频则不是必需的。

#### 7.4.7 视频压缩

到目前为止,很显然的一点是,传输未经压缩的视频是完全不可能的。惟一的希望是有可能大比例地压缩视频数据。幸运的是,过去几十年来大量的研究工作已经产生了许多压缩技术和算法,它们使视频传输变得切实可行。在这一小节中,我们将学习如何实现视频压缩。

所有的压缩系统都要求两个算法:一个在数据源端用于压缩数据,另一个在目标端用于解压缩。在文献中,这两个算法分别被称为编码(encoding)和解码(decoding)算法。在这里我们也将使用这样的术语。

这些算法表现出一定的不对称性,理解这一点是非常重要的。首先,对于许多应用来说,一个多媒体文档,比如说一部电影,只会被编码一次(当它被存储到多媒体服务器上的时候),但是将被成千上万次地解码(当用户观看该电影的时候)。这种不对称性意味着,只要解码算法较快并且不要求昂贵的硬件,则编码算法即使较慢而且要求昂贵的硬件也是可以接受的。毕竟,多媒体服务器的运行商可能很愿意租用一台并行超级计算机几个星期时间,以便对整个视频资料库进行编码,但是,如果要求用户租用一台超级计算机两个小时以便观看一部视频电影,则这种模式不可能成功。许多实际的压缩系统竭尽全力地使解码过程既快速又简单,甚至以编码的慢速和复杂作为代价。

另一方面,对于实时多媒体应用,例如视频会议,慢速的编码过程是不可能被接受的。编码过程必须快速而实时地进行。因此,实时多媒体不同于在磁盘上存储视频,它使用不同的算法或者参数,通常导致略低一点的压缩效率。

另一种不对称性是编码/解码过程不必是可逆的。也就是说,当压缩一个文件并将它传输出去,然后在另一端解压缩的时候,用户期望获得原始的文件,而且必须精确到文件中的每一位。对于多媒体,这种需求并不存在。将视频信号编码之后再进行解码所得到的信号与原始的信号稍微有点差异,这通常是可以接受的。当解码算法的输出不完全等于原始的输入时,该系统被称为是有损的(lossy)。如果输入和输出完全一致,则该系统被称为是无损的(lossless)。有损系统很重要,因为接受少量的信息损失将可以换回在可能压缩率范围内的巨大回报。

#### JPEG 标准

视频只是一个图像序列(加上声音)而已。如果我们能够找到一个好的算法来编码单一的图像,那么,将该算法连续地应用到这个序列中的每一个图像上,我们就可以实现视频压缩。好的静止图像压缩算法是存在的,所以我们先从这里开始有关视频压缩的学习。



用于压缩连续色调的静止图像(比如照片)的 JPEG(Joint Photographic Experts Group, 联合图像专家组)标准是由一批图像专家在 ITU、ISO 和另一个标准组织 IEC 的联合赞助下开发的。它对于多媒体非常重要,因为大致上可以这么说,针对运动图像的多媒体标准 MPEG 只是对每一帧图像单独地进行 JPEG 编码,再加上一些针对帧间压缩和运动检测的额外特性而已。JPEG 被定义在国际标准 10918 中。

JPEG 有四种模式和许多选项。它更像一个购物清单而并非一个算法。就我们的目的而言,只有有损顺序模式才是我们所关心的,图 7.71 演示了这种模式。此外,我们的讨论将集中在 JPEG 通常被用于编码 24 位 RGB 视频图像的方法上,并且为了简便起见,我们在讨论过程中将会忽略一些小的细节。

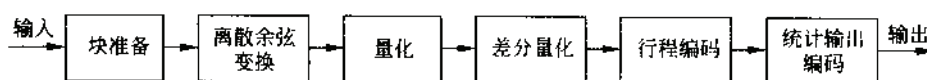


图 7.71 在有损顺序模式下的 JPEG 操作

用 JPEG 来编码一个图像的第 1 步是块准备(block preparation)。为了叙述方便,我们假设 JPEG 的输入是一个  $640 \times 480$  的 RGB 图像,每个像素 24 位,如图 7.72(a)所示。由于使用亮度和色度能够得到更好的压缩效率,所以,我们首先利用下面的公式来计算亮度分量 Y 和两个色度分量 I 和 Q(对于 NTSC):

$$Y = 0.30R + 0.59G + 0.11B$$

$$I = 0.60R - 0.28G - 0.32B$$

$$Q = 0.21R - 0.52G + 0.31B$$

对于 PAL 系统,色度分量被称为 U 和 V,并且系数有所不同,但思想是一样的。SECAM 系统不同于 NTSC 和 PAL。

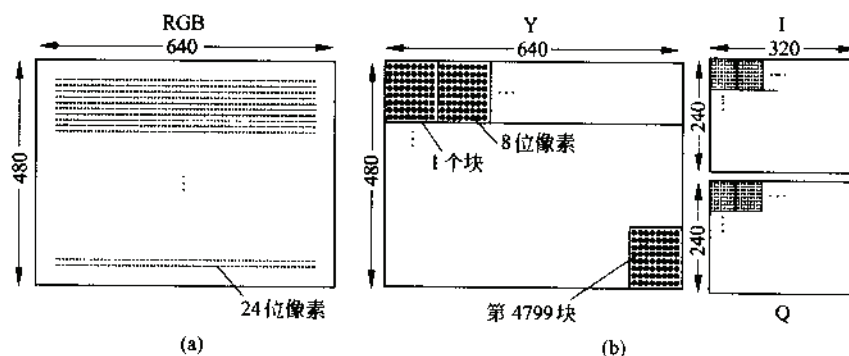


图 7.72

(a) RGB 输入数据; (b) 块准备操作之后

为 Y、I 和 Q 分别构造一个独立的矩阵,每个矩阵元素的范围在 0 至 255 之间。接下来,在 I 和 Q 矩阵中,针对每四个像素组成的方块计算其平均值,因而矩阵被减小至  $320 \times 240$ 。这种缩减是有损的,但是眼睛几乎不会注意到,因为眼睛对亮度比对色度更加敏感。然而,这样做却能够把整个数据量压缩一半。现在从这三个矩阵的每个元素中减



去 128, 以便使元素范围的中间值变成 0。最后, 每个矩阵被分割成  $8 \times 8$  的块。Y 矩阵有 4800 个块; 其他两个矩阵有 1200 个块, 如图 7.72(b) 所示。

JPEG 的第 2 步是对 7200 个块中的每一个块单独做 DCT (Discrete Cosine Transformation, 离散余弦变换)。每个 DCT 的输出是一个  $8 \times 8$  的 DCT 系数矩阵。DCT 元素 (0,0) 是每个块的平均值。其他的元素表明了在每个空间频率处的频谱功率。在理论上, DCT 是无损的, 但是在实践中, 使用浮点数和超越函数通常会引入一些舍入误差, 从而导致少许信息损失。通常, 随着元素离原点 (0,0) 的距离越来越远, 它们的值也迅速地减小, 如图 7.73 所示。

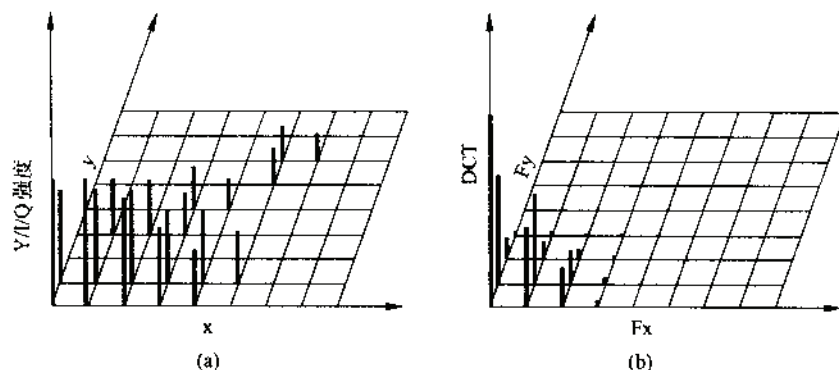


图 7.73

(a) Y 矩阵的一个块; (b) DCT 系数

一旦 DCT 完成了, JPEG 进入第 3 步, 这一步被称为量化 (quantization)。在这一步中, 一些不重要的 DCT 系数被去除掉。这种 (有损) 变换是这样来完成的: 将  $8 \times 8$  DCT 矩阵中的每个系数除以一个权值, 而该权值是通过查表得来的。如果所有的权值都是 1, 则该变换不做任何事情。然而, 如果权值表从原点开始迅速地递增的话, 则较高的空间频率将被快速地丢弃掉。

图 7.74 中给出了这个步骤的一个例子。在这里, 我们可以看到初始的 DCT 矩阵、量化表, 以及每个 DCT 元素除以量化表中对应元素之后得到的结果。量化表中的值不是 JPEG 标准的一部分。每个应用必须提供它自己的量化表, 从而允许它自己来控制“信息损失·压缩效率”之间的平衡。

DCT 系数								量化表								量化后的系数							
150	80	40	14	4	2	1	0	1	1	2	4	8	16	32	64	150	80	20	4	1	0	0	0
92	75	36	10	6	1	0	0	1	1	2	4	8	16	32	64	92	75	18	3	1	0	0	0
52	38	26	8	7	4	0	0	2	2	2	4	8	16	32	64	26	19	13	2	1	0	0	0
12	8	6	4	2	1	0	0	4	4	4	4	8	16	32	64	3	2	2	1	0	0	0	0
4	3	2	0	0	0	0	0	8	8	8	8	8	16	32	64	1	0	0	0	0	0	0	0
2	2	1	1	0	0	0	0	16	16	16	16	16	16	32	64	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	32	32	32	32	32	32	32	64	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	64	64	64	64	64	64	64	64	0	0	0	0	0	0	0	0

图 7.74 量化的 DCT 系数的计算方法

第4步减小每一块的(0,0)元素的值,做法是,用它与前一个块中对应元素的差值来替代它。由于这些元素是相应块的平均值,所以,它们应该变化很缓慢,因此,采用差值之后可以把它们中的大部分减为很小的数值。其他的元素不计算差值。(0,0)元素的值被称为DC(直流)分量;其他的值被称为AC(交流)分量。

第5步将64个元素排列起来,并且对此列表使用行程编码方法。按照先从左到右再从上往下的方法对一个块进行扫描并不能将0集中到一起,所以JPEG采用了“Z”字形的扫描模式,如图7.75所示。在这个例子中,“Z”字形扫描模式将产生38个连续的0,位于扫描串的末端。然后,这个0串能够被减少为一个计数值,即由该计数值来表明这38个0值,这项技术即为行程编码(run-length encoding)。

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

图 7.75 量化值的传输顺序

现在我们有了一系列数值来代表原图像(在变换空间中)。第6步对这些数值进行霍夫曼编码,以便于存储和传输;在这种编码方法中,越是常见的数值,越被分配较短的编码;相反,不常见的数值则被分配较长的编码。

JPEG 看起来似乎比较复杂,的确是这样的,它确实比较复杂。然而,由于它通常能够达到20:1甚至更好的压缩效率,所以,它得到了广泛的应用。解码一个JPEG图像需要反向运行该算法。JPEG算法基本上是对称的:解码和编码两者所花的时间一样长。正如我们所提到的,并不是所有的压缩算法都具有这种特性。

## MPEG 标准

最后,我们来到了问题的核心部分:MPEG(Motion Picture Experts Group,运动图像专家组)标准。MPEG是用于压缩视频的主要算法,并且从1993年开始已经成为国际标准。因为电影包含图像和声音,所以,MPEG既可以压缩音频也可以压缩视频。我们已经讨论过音频压缩和静止图像的压缩,现在我们来讨论视频压缩。

第一个被定案的标准是MPEG-1(国际标准11172)。它的目标是用1.2Mbps的数据率来产生录像机品质的输出(对于NTSC而言是352×240)。如果图像的大小为352×240,每个像素有24位颜色,并且每秒钟25帧,则所需带宽为50.7Mbps,所以,把带宽要求降低到1.2Mbps是非常有意义的。为了达到这个目标需要40倍的压缩率。MPEG-1视频能够在双绞线上进行中等距离的传输。MPEG-1也被用于在CD-ROM上存储电影。

在 MPEG 家族中,下一个标准是 MPEG-2(国际标准 13818),它最初的设计目标是为了把广播级品质的视频压缩到 4 至 6Mbps,以便能够适应于 NTSC 或 PAL 广播信道。后来,MPEG-2 被扩展到支持更高的分辨率,包括 HDTV。现在它已经非常常用,因为它形成了 DVD 和数字卫星电视的基础。

MPEG-1 和 MPEG-2 的基本原理非常相似,但是细节上有所不同。大致上可以这么说,MPEG-2 是 MPEG-1 的超集,它增加了一些特性、帧格式和解码选项。我们将首先讨论 MPEG-1,然后讨论 MPEG-2。

如图 7.76 所示,MPEG-1 有三个部分:音频、视频和系统,其中系统部分将另外两部分集成在一起。音频和视频的编码器独立地工作,这也引发了音频流和视频流如何在接收端同步的问题。这个问题可以这样来解决:用一个 90kHz 的系统时钟向两个编码器输出当前的时间。这些时间值有 33 位,从而使得连续运行 24 小时也不会发生回绕。这些时间戳被包含在编码输出中,并且被一路传送至接收方,接收方可以用它们来同步音频流和视频流。

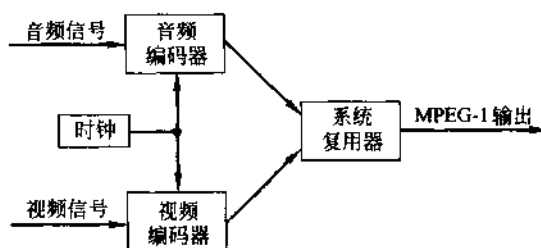


图 7.76 MPEG-1 中音频流和视频流的同步

下面我们来考虑 MPEG-1 的视频压缩。在电影中存在两种冗余:空间的和时间的。MPEG-1 同时使用了这两种冗余。空间冗余度很容易被利用,只需简单地用 JPEG 来独立地编码每一帧即可。这种方法只是被偶尔使用,特别是当需要随机地访问每一帧的时候,例如在视频编辑产品中。在这种模式下,压缩之后的带宽在 8~10Mbps 范围内是可以达到的。

连续的帧往往几乎是相同的,充分利用这样的事实还可以获得更大的压缩率。其效果比初始想象的要差一些,因为许多电影制作人在剪辑电影时差不多每隔 3 或 4 秒就切换场景(对电影计时,并且对场景计数)。然而,即使只有连续 75 个高度相似的帧,也提供了比“简单地利用 JPEG 来独立地编码每一帧”这种做法更多的压缩潜力。

对于那些“摄像机和背景都固定不变,而只有一两个演员慢慢地走来走去”的场景,帧和帧之间几乎所有的像素都是相同的。在这里,只要从前一帧中减去当前帧,并且对两帧之差运行 JPEG 算法,则效果会非常好。然而,对于那些摄像机在移动成者在推拉的场景,这项技术的效果会很差。这就需要一种办法来补偿这种运动。这正是 MPEG 所做的,也是 MPEG 和 JPEG 的主要区别。

MPEG-1 的输出包括四种帧:

- I(帧内编码,Intracoded)帧:自包含的 JPEG 编码的静止图片。

- P(预测, Predictive)帧: 与前一帧之间的逐块差值。
- B(双向, Bidirectional)帧: 与前一帧和后一帧之间的差值。
- D(DC 编码, DC-coded)帧: 用于快速的块平均值。

I 帧只不过是一些静止图片,它采用 JPEG 的一个变种来进行编码,同时沿每一个轴使用全分辨率的亮度和半分辨率的色度。有三个原因使得 I 帧必须周期性地出现在输出流中。首先,MPEG-1 可能被用于多播传输,观众可以随意地收看视频节目。如果所有的帧都依赖于前面的帧,如此一直回溯到第一帧,那么,错过第一帧的人就无法再解码后续的任何帧了。第二,如果任何一帧发生了接收错误,则后续的解码将无法正确地进行。第三,若没有 I 帧,那么,在快速进或者回退时,解码器将不得不计算沿途经过的每一帧,以便得到停止处的那一帧的所有值。由于这些原因,每隔一秒钟 I 帧被插入到输出中一至两次。

相反,P 帧则是对帧间的差值进行编码。P 帧建立在宏块(macroblock)的概念基础上,宏块覆盖了亮度空间中的  $16 \times 16$  像素大小,以及色度空间中的  $8 \times 8$  像素大小。宏块的编码是通过在前一帧中搜索与自己相同或稍有不同的部分来完成的。

图 7.77 显示了 P 帧用法的一个例子。在这里,我们看到三个连续的帧,它们有相同的背景,但是左侧人的位置不同。包含背景的宏块能够精确地匹配,但是包含人的宏块在位置上有一定的未知量偏移,这样的宏块必须被跟踪住。

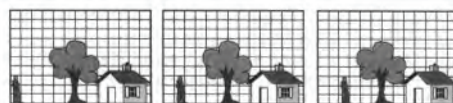


图 7.77 三个连续的帧

MPEG-1 标准没有规定如何进行搜索、搜索的范围有多远,或者如何评价一个匹配的好坏。这些由每个实现自行决定。例如,一个具体的实现可能在前一帧的当前位置上搜索一个宏块,同时也在  $x$  方向偏移  $\pm \Delta x$  范围内、 $y$  方向偏移  $\pm \Delta y$  范围内的所有位置上搜索该宏块。对于每个位置,将计算出亮度矩阵中匹配的数量。如果最高的得分值超过某个预定的阈值,则它所在的位置被宣布为胜者。否则,该宏块被称为已丢失。当然,其他更加复杂的算法也是有可能的。

如果一个宏块已被找到,则使用它与前一帧中对应宏块之间的差值来进行编码(包括亮度和两个色度)。然后,这些差值矩阵被执行离散余弦变换、量化、行程编码和霍夫曼编码,就如同 JPEG 的做法一样。然后,在输出流中,该宏块的值是运动向量(与前一帧中的位置相比较,在每个方向上移动了多少),后面紧跟着霍夫曼编码的数字串。如果在前一帧中没有找到该宏块,则利用 JPEG 对当前的值进行编码,就如同在 I 帧中的做法一样。

很显然,这个算法是高度非对称的。对于任何一个实现,只要它愿意,它总是可以自由地尝试前一帧中每一个可能的位置,因此,不管该宏块如何被移动,只要它还在,则用旁

举的办法总是可以找到原来的位置。这种做法可以使 MPEG-1 的输出流降低到最小,付出的代价是极慢的编码速度。对于电影资料库的一次性编码应用来说,这种方法是非常合适的,但是对于实时视频会议来说则是非常可怕的。

类似地,每一个实现可以自由地决定如何来确定“找到的”宏块。这种自由度允许各个实现者在他们的算法的质量和速度上进行竞争,但总是会产生兼容的 MPEG-1 数据。无论采用何种搜索算法,最后的输出要么是当前宏块的 JPEG 编码,要么是当前宏块与前一帧中离当前宏块有一定偏移的宏块之间的差值的 JPEG 编码。

到目前为止,解码 MPEG-1 是非常直截了当的。解码 I 帧的做法如同解码 JPEG 图像相同。解码 P 帧则要求解码器缓冲前一帧,然后利用完全编码的宏块以及包含与前一帧差值的宏块,在另一个缓冲区中构造出新的一帧。此新帧是由一个一个宏块装配起来的。

B 帧类似于 P 帧,不同之处在于,它允许参考宏块既可以在前一帧中,也可以在后一帧中。这种更大的自由度允许更强的运动补偿,当有些物体在其他物体的前面或者后面通过时,这种方法非常有用。为了进行 B 帧解码,解码器需要同时在内存中保留三个解码的帧:过去的、当前的和将来的帧。尽管 B 帧能够提供最好的压缩效率,但并不是所有的实现都支持 B 帧。

D 帧仅仅被用于在回退或者快进时使得有可能显示低分辨率的图像。实时地完成正常的 MPEG-1 解码已经足够困难了。当以十倍于正常速度来回转视频时也要求解码器做到这一点,则有点强人所难了。替代的办法是,利用 D 帧来产生低分辨率的图像。D 帧中的每一项只不过是一个块的平均值,而且也没有进一步编码,这使得 D 帧很容易被实时地显示。这项设施非常重要,它使得人们可以非常方便地快速扫描一个视频以寻找一个特殊的场景。D 帧通常被放在对应 I 帧的前面,所以,如果快进操作停止了,则马上就可以开始以正常速度进行播放。

在结束了关于 MPEG-1 的讨论之后,下面我们来看一看 MPEG-2。MPEG-2 的编码基本上类似于 MPEG-1 的编码,它也有 I 帧、P 帧和 B 帧。然而,它不支持 D 帧。另外,离散余弦变换使用了  $10 \times 10$  的块,而不是  $8 \times 8$  的块,从而使系数的数量增加了 50%,因而可获得更好的视频质量。由于 MPEG-2 的目标是广播电视和 DVD,所以,它同时支持逐行扫描和隔行扫描的图像,而 MPEG-1 只支持逐行扫描的图像,这是两者不同的地方。这两个标准还有其他一些小的细节差异。

MPEG-2 支持四种而不是一种分辨率等级:低( $352 \times 240$ )、主要( $720 \times 480$ )、高-1440( $1440 \times 1152$ )和高( $1920 \times 1080$ )。低分辨率被用于 VCR,并且向后兼容 MPEG-1。主要级别被用于正常的 NTSC 广播。其他两个级别被用于 HDTV。为了高质量的输出,MPEG-2 通常运行在 4~8Mbps 的速率上。

#### 7.4.8 视频点播

视频点播有时被比喻成电子的录像出租店。用户(消费者)从大量供挑选的录像中选择一个,然后带回家观看。只有通过视频点播,这种选择才可以在家里通过电视机的遥控器来完成,并且选择之后立即开始播放,所以,用户不需要再到店里去了。无需多说,实现

视频点播要比描述它复杂得多。在本节中,我们将从总体上描述视频点播的基本思想以及相应的实现。

视频点播真的像出租录像吗?或者它更像是从一个包含 500 个频道的有线电视系统中挑选出一部电影来观看吗?这个问题的答案有重要的技术含意。尤其是,录像出租店的用户通常习惯于这样的想法:暂停一部录像影片,并快速地跑到厨房或浴室再跑回来,然后从刚才停止的地方继续开始观看。电视观众却从来不希望能按自己的意志来暂停节目。

如果视频点播要在与录像出租店的竞争中取得成功,则它必须允许用户可随意地停止、开始或者回退视频。要想赋予用户这样的能力,实际上相当于迫使视频供应商为每个用户传送一份单独的視頻副本。

另一方面,如果视频点播被看作更像是高级的电视,那么视频供应商可能只需每隔一段时间(比如说 10 分钟)之后开始播放每一个流行的视频节目,并且不间断地运行这些节目就足够了。如果一个用户想要看一个流行的视频节目,则他至多等待 10 分钟就可以从头开始看起。尽管在这里暂停/继续的功能是不可能的,但是观众在经过短暂的休息之后,他可以切换到另一个频道,这个频道正在播放同一份视频,但时间晚了 10 分钟。有一些内容将会重复,但不会漏掉任何内容。这种方案被称为准视频点播(near video on demand)。它用低得多的代价提供了与视频点播类似的潜在能力,因为来自视频服务器的同一份数据可以被同时发送给许多用户。视频点播与准视频点播之间的区别,类似于你自己驾车与乘坐公共汽车之间的区别。

一旦有了宽带网络以后,大量潜在的新服务就成为可能,按(准)点播方式来观看电影只不过是其中之一。许多人使用的通用模型如图 7.78 所示。在这里,我们可以看到,在系统的中心有一个高带宽的(全国的或国际的)广域骨干网络,连接在骨干网络之上的是数以千计的本地分发网络,例如有线电视或者电话公司的分发系统。本地分发系统进入到人们的家中,终止于每户家庭的机顶盒(set-top box),机顶盒实际上是功能强大的专用

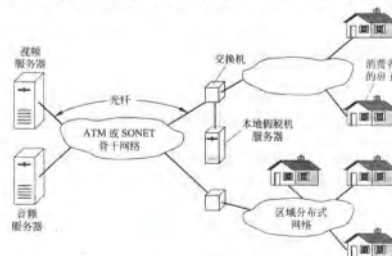


图 7.78 视频点播系统的概况



个人计算机。

通过高带宽光纤连接到骨干网上的是众多的信息供应商。有些供应商提供按逐次观看进行收费的视频(pay-per-view video)或按逐次收听进行收费的音频 CD(pay-per-hear audio CD);其他的则提供一些专门的服务,比如在家购物(让观众旋转一听汤品罐头并放大它的成分列表,或者观看一段关于如何驱动汽油动力割草机的视频片断)。体育、新闻、重新放映“我爱露西”、WWW 访问,以及其他不计其数的各种可能事物,毫无疑问很快都将变成现实。

在系统中还包含了一些本地假脱机服务器,利用这些服务器,供应商可以预先将视频内容放在离用户更近的地方,从而为高峰时间段节省带宽。这些部分将如何配合,以及谁拥有哪些部分,这是工业界中激烈争论的问题。下面我们将讨论系统主要部分的设计:视频服务器和分发网络。

### 视频服务器

为了实现(准)视频点播,我们要求视频服务器(video server)能够同时存储和输出大量的电影。据估计,人们已经制作出来的电影总数大约是 65 000 部(Minoli, 1995)。当用 MPEG-2 来压缩时,一部普通的电影大概占有 4GB 存储空间,因此 65 000 部电影将需要大约 260TB。再加上所有曾经制作的旧电视节目、体育影片、新闻影片、谈论购物的目录片断等等,显然我们立即遇到了一个工业强度的存储问题。

最廉价的存储海量信息的方法是利用磁带。这一直都是这样的,可能将来仍然如此。一盒 Ultrium 磁带可以存储 200GB(50 部电影),其成本是每部电影 1~2 美元。现在已经有商用的大型机械磁带服务器:它们可以存放数千盒磁带,并且有一支机械手可以抓取任何一盒磁带,并将它插入到磁带驱动器中。这些系统的问题是存取时间(特别是对于一盒磁带上的第 50 部电影)、传输率,以及有限的磁带驱动器数目(为了同时提供  $n$  部电影,该服务器将需要  $n$  个驱动器)。

幸运的是,录像出租店、公共图书馆和其他此类组织的经验表明,并非所有的视频内容都同样地受欢迎。根据实验的结果,当有  $N$  部电影可供使用时,受欢迎程度排在第  $k$  位的电影所获得的请求次数占总请求次数的比例大约是  $C/k$ ,这里引入  $C$  是为了将结果总和归一到 1,即:

$$C=1/(1+1/2+1/3+1/4+1/5+\dots+1/N)$$

因此,对于最受欢迎的那部电影,它受欢迎的程度是第七受欢迎电影的 7 倍。这一结果被称为 Zipf 定律(Zipf, 1949)。

有些电影比其他的电影更受欢迎,这样的事实向人们暗示了一种可能的解决方案,其形式是一种存储分层结构,如图 7.79 所示。在这里,性能随着层次的上升而提高。

除了磁带存储,另一种办法是光存储。现在的 DVD 可以存储 4.7GB,适合存放一部电影,但下一代的 DVD 将可以存储两部电影。尽管 DVD 的搜寻时间比磁盘要慢(50 毫秒对 5 毫秒),但由于它们的低成本和高可靠性,因此,对于频繁使用的电影而言,包含有数千片 DVD 的自动点唱机是除了磁带以外的另一种好方案。

接下来是磁盘,它们有较短的存取时间(5 毫秒)、较高的传输率(SCSI 320 的传输率

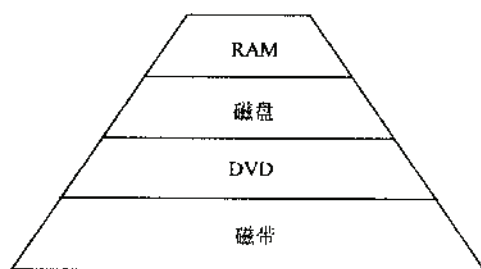


图 7.79 视频服务器的存储分层结构

是 320Mbps),以及相当大的容量(>100GB),这使得它们非常适合于存储那些实际要被传输的电影(与之相对应的是,其他的电影被存储起来只是为了万一有人想要它们)。它们的主要缺点是,对于那些很少被访问的电影来说,其存储代价太高。

图 7.79 的金字塔顶部是 RAM(随机存储器)。RAM 是最快的存储介质,但也是最昂贵的。当 RAM 的价格下降到 50 美元/GB 时,一部 4GB 的电影将占用价值为 200 美元的 RAM 空间,所以,在 RAM 中存放 100 部电影将需要 400GB 内存,价值 20 000 美元。还有,对于一个需要输出 100 部电影的服务器来说,若要将所有的电影都存放在 RAM 中,则按照现在的存储技术,这已经变得切实可行。如果该服务器有 100 个用户,但他们只集中观看 20 部不同的电影,那么这不仅是可行的,而且也是一个好的设计。

由于视频服务器实际上只是一个大容量的实时 I/O 设备,所以它需要一种不同于 PC 或 UNIX 工作站的硬件和软件体系结构。图 7.80 显示了一个典型的视频服务器的硬件体系结构。该服务器有一个或多个高性能的 CPU(每个 CPU 都带有一些局部内存)、一些共享的主内存、大量的 RAM 缓存用于存放那些流行的电影、各种存储设备用于存放电影,以及一些网络硬件,通常有一个连接 SONET 或者 ATM 骨干网的光接口(速度为 OC-12 或更高)。这些子系统通过非常高速的总线(至少 1GB/s)连接起来。

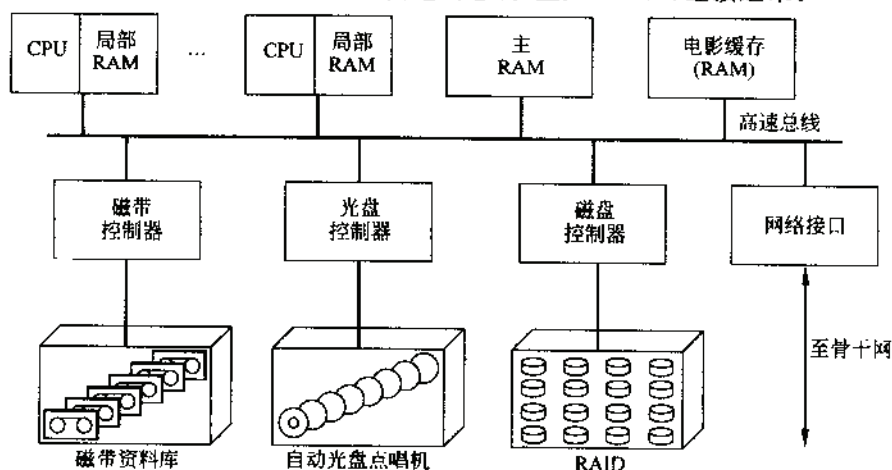


图 7.80 一个典型的视频服务器的硬件体系结构

现在我们来简要地看一下视频服务器软件。CPU 被用于接受用户请求、查找电影、在设备之间移动数据、顾客记账,以及许多其他功能。有一些功能对时间要求并不高,但许多其他功能却有严格的时间要求,所以,有些 CPU(不一定是全部 CPU)必须运行一个实时操作系统,比如一个实时的微内核系统。这些系统通常将工作分解成一些小的任务,每个任务都有一个已知的截止时间。然后,调度程序就可以运行一些算法,比如“下一个最近的截止时间”或者速率单调算法(Liu and Layland, 1973)。

CPU 软件也定义了服务器呈现给客户(假脱机服务器和机顶盒)的接口的本质。有两种流行的设计方案。第一种是一个传统的文件系统,客户可以打开、读、写和关闭文件。除了由于存储分层结构和实时性方面的考虑而带来的复杂性以外,这样的服务器可以使用一个类似于 UNIX 的文件系统。

第二种接口以录像机模型为基础。发送给服务器的命令请求它打开、播放、暂停、快进和回退文件。该模型与 UNIX 模型之间的区别是,一旦服务器接到了 PLAY 命令,它就持续地以恒定的速率输出数据,而不再需要新的命令。

视频服务器软件的核心是磁盘管理软件,它有两项主要的工作:必要时将电影从光存储或者磁带存储中取出来放到磁盘上,以及为各个输出流处理磁盘请求。电影的放置非常重要,因为它可以极大地影响性能。

有两种可能的组织磁盘存储的方法,它们分别是磁盘场和磁盘阵列。在**磁盘场(disk farm)**方法中,每个驱动器存放一定数量的完整电影。由于性能和可靠性的原因,每部电影应该被存放在至少两个,可能更多个驱动器上。另一种存储组织方法是**磁盘阵列(disk array)**或者 RAID(Redundant Array of Inexpensive Disks, 廉价磁盘冗余阵列),每部电影被分布在多个驱动器上,例如,第 0 块在第 0 个驱动器上,第 1 块在第 1 个驱动器上,等等,第  $n-1$  块在第  $n-1$  个驱动器上。此后,又重复循环,第  $n$  块在第 0 个驱动器上,依此类推。这种组织方式被称为**条状化(striping)**。

条状化的磁盘阵列与磁盘场相比有几个优点。首先,所有  $n$  个驱动器可以并行运行,这使性能增加了  $n$  倍。其次,只需为每  $n$  个一组的驱动器组增加一个额外的驱动器,就可以实现一定的冗余度,冗余驱动器包含了其他驱动器的逐块异或(XOR)值,因此,当一个驱动器发生故障时可以完全恢复数据。最后,负载平衡的问题也得到了解决(无需再为了避免使所有流行的电影都位于同一个驱动器上而使用手工放置策略)。另一方面,磁盘阵列的组织方式比磁盘场更加复杂,并且对多处故障的情形极其敏感。它也不适合于某些录像机操作,比如回退或快进电影等。

磁盘软件的另一项任务是为所有的实时输出流提供服务,并满足它们的时间限制要求。仅仅在几年以前,这还需要复杂的磁盘调度算法,但由于内存价格现已如此之低,因此一种更简单的方案开始变得可能。对于所服务的每个流,在 RAM 中保留一个缓冲区,比如说,该缓冲区可以存放 10 秒钟视频(5MB)。磁盘进程填充该缓冲区,网络进程从中提取数据,直到取空为止。若有 500MB 的 RAM,则共有 100 个流可以直接从 RAM 中供应数据。当然,磁盘子系统必须有恒定的 50Mbps 数据率,以便保持缓冲区总是满的,但是,由高端 SCSI 磁盘构造的 RAID 很容易就可以满足这样的要求。

## 分发网络

所谓分发网络,是指数据源与目标之间的交换机和线路的集合。正如我们在图 7.78 中所看到的,它包括一个骨干网络,该骨干网络连接到一个本地分发网络上。通常骨干网络是交换网络,而本地分发网络往往不是。

施加在骨干网络上的主要需求是高带宽。在过去,低抖动也是一个需求,但现在,即使最小的 PC 也能够缓冲 10 秒钟的高质量 MPEG-2 视频,所以低抖动已经不再是一个需求了。

本地的数据分发非常混乱,在不同的地区,不同的公司使用不同的网络。电话公司、有线电视公司,以及新的竞争者如电力公司等都深信,谁第一个进入,谁就是最大的赢家。因此,我们现在看到了各种各样的技术被纷纷采纳进来。在日本,有些下水道公司也从事 Internet 业务,它们认为自己拥有进入每个人家庭的最大管道(它们通过该管道来铺设光纤,但必须非常小心地控制好光纤的精确出口)。针对视频点播,四种主要的本地分发方案按首字母缩写依次是 ADSL、FTTC、FTTH 和 HFC。现在我们将依次介绍每一种方案。

ADSL 是电话工业中第一个介入本地分发竞争的参与者。我们在第 2 章中已经学习过 ADSL,所以这里不再重复那些内容。其想法是,在美国、欧洲和日本几乎每个家庭都已经有一条铜双绞线进入住宅(为了提供模拟电话服务)。如果这些线能够被用于视频点播,则电话公司将大发其财。

当然,问题是,这些线在其典型的 10 公里长度上甚至不可能支持 MPEG-1,更不用说 MPEG-2 了。高分辨率的、全彩色的、全运动的视频需要 4~8Mbps,具体带宽取决于所期望的质量。除了非常短的本地回路以外,ADSL 实际上并没有足够快的速度。

电话公司的第二个设计方案是 FTTC(Fiber To The Curb,光纤到路边)。在 FTTC 中,电话公司从端局出发铺设光纤到每个居民住宅附近,光纤终端的设备被称为 ONU(Optical Network Unit,光纤网络单元)。大约 16 条本地回路铜线可以连接在一个 ONU 上。现在这些回路非常短,以至于有可能在其上运行全双工的 T1 或 T2,从而分别可支持 MPEG-1 和 MPEG-2 电影。此外,由于 FTTC 是对称的,所以对于家庭办公者和小型商务活动而言,视频会议现在也是可能的。

电话公司的第三个解决方案是将光纤铺设到每个人的家里,这被称为 FTTH(Fiber To The Home,光纤到户)。在这种方案中,每个人都可以有一条 OC-1、OC-3 线路,如果需要的话,甚至还可以有更高带宽的线路。FTTH 的代价非常高,最近几年内还不太可能会实现,但是当它最终成为现实时,显然将会开启大量新的可能应用。在图 7.63 中我们看到了每个人如何运营他或她自己的广播电台。你认为家庭中的每个成员都运营自己的个人电视台怎么样? ADSL、FTTC 和 FTTH 都是点到点的本地分发网络,只要知道了当前电话系统的组织结构,这就不足为怪了。

一种完全不同的方法是 HFC(Hybrid Fiber Coax,混合光纤同轴电缆),这是有线电视供应商目前首选的解决方案,如图 2.47(a)所示。事情大致是这样的:当前的 300~450MHz 的同轴电缆被 750MHz 的同轴电缆所替代,其容量从 50 至 75 个 6MHz 的信道

(频道)升级到 125 个 6MHz 的信道。在这 125 个信道中,其中 75 个将被用于传输模拟电视。

对于 50 个新的信道,每个都将使用 QAM-256 进行调制,这样得到的新增带宽为:每个信道大约 40Mbps,总计 2Gbps。头端将被进一步深入到居民住宅附近,从而使得每条电缆仅仅通过 500 户家庭。通过简单的除法运算就可以看出,每个家庭可以分配到专用的 4Mbps 信道,这样就可以传输 MPEG-2 的电影了。

尽管这听起来非常不错,但它要求有线电视供应商用 750MHz 的同轴电缆替换掉所有现有的电缆,并且安装新的头端,拆除所有的单向放大器——简而言之,要替换整个有线电视系统。因此,这里的新基础设施数量与电话公司 FTTC 方案所需的数量相当。在这两种情形下,本地网络供应商都必须将光纤铺设到居民住宅附近。同样地,在这两种情形下,光纤的末端都被连接到一个光电转换器上。在 FTTC 中,最终的一段是一个使用双绞线的本地回路;而在 HFC 中,最终的一段是一根共享的同轴电缆。从技术上而言,这两个系统其实并不像它们各自的支持者所声称的那样有很大的差异。

然而,这里有一个真正的区别值得指出来。HFC 使用了共享的介质,它无需交换和路由。放到电缆上的任何信息都可以被所有的用户不费周折地拿走。FTTC 是全交换的,所以它没有这种特性。其结果是,HFC 的运行商希望视频服务器发送加密的流,这样,那些没有为一部电影付费的用户就不能观看这部电影。FTTC 的运行商却特别不希望加密,因为这将增加复杂性、降低性能,并且没有为它们的系统提供任何额外的安全性。从运行视频服务器的公司的角度来看,加密或者不加密是一个好的做法吗?由电话公司或者它的子公司或合作伙伴来运营的服务器可能有意地不加密它的视频流,并且声称效率是其中的原因,但事实上是为了造成其 HFC 竞争者们的经济损失。

对于所有这些本地分发网络,可能每个居民区都将配备一台或多台假脱机服务器。实际上,这些服务器只不过是上面讨论的视频服务器的较小版本而已。这些本地服务器的巨大优势是,它们将骨干网络上的一部分负载转移到局部范围内了。

通过预定的方式,这些本地服务器可以预先装载一些电影。如果人们提前告诉供应商他们想看什么电影,那么,供应商就可以在非高峰时间段将电影下载到本地服务器上。这种现象很可能导致网络运行商聘请航空公司的管理人员来为他们制定价格。你可以想象这样的价目表:提前 24~72 小时预定周二或周四晚上 6 点以前或 11 点以后的电影,将得到 27% 的折扣;在每个月第一个周日上午 8 点之前预定的并且日期为素数的周三下午的电影,将得到 43% 的折扣,如此等等。

#### 7.4.9 Mbone—多播骨干网

当所有这些产业都在为未来全国性(或国际性)的数字视频点播应用制定并大肆宣扬宏伟计划的时候,Internet 社团已经悄悄地实现了它自己的数字多媒体系统 **MBone** (**Multicast Backbone**, **多播骨干网**)。在本小节中我们将简要地概述一下它是什么以及它是如何工作的。

MBone 可以被看作 Internet 电视。它与视频点播不同,视频点播的重点在于请求和观看预先被存储在服务器上的压缩电影,而 MBone 则被用于通过 Internet 向全世界广播



数字形式的视频实况。从 1992 年初开始,MBone 就已经投入运行了,它已经广播了许多科学会议,特别是 IETF 的会议,甚至还广播了一些像发射航天飞机之类有报道价值的科学事件。作为戛纳电影节的一部分,还曾经在 MBone 上广播了一场滚石音乐会,这是否能当作一件有报道价值的科学事件,倒是尚有疑问。

从技术上来说,MBone 是位于 Internet 之上的一个虚拟层叠网络。如图 7.81 所示,它是由一些通过隧道连接起来的支持多播的岛(island)组成的。在此图中,MBone 包含六个岛,分别标记为从 A 到 F,它们通过 7 条隧道连接起来。每个岛(典型情况下是一个 LAN 或一组互连的 LAN)都支持硬件的多播传输,以便将数据送到它的主机上。岛与岛之间的隧道可以传送 MBone 分组。在将来的某一天,当所有的路由器都能够直接处理多播流量时,这种层叠结构就不再有必要了,但目前需要由它来完成 Internet 多播的任务。

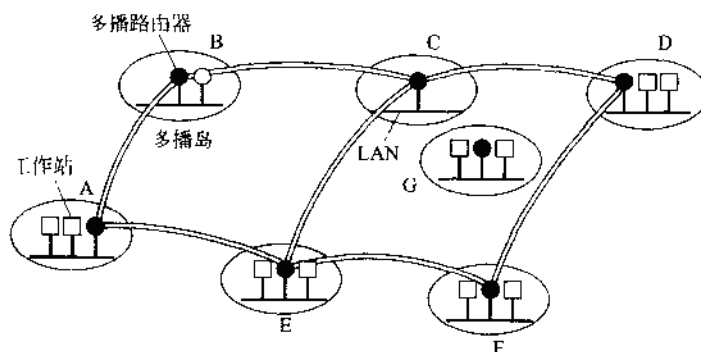


图 7.81 通过隧道连接起来的多播岛构成了 MBone

每个岛包含一台或多台称为 **mrouter**(**multicast router**, 多播路由器)的特殊路由器。有些多播路由器是真正的普通路由器,但大多数只不过是运行了特定的用户态软件(但以 root 身份来运行)的 UNIX 工作站。在逻辑上,这些多播路由器通过隧道连接起来。MBone 分组被封装在 IP 分组中,并被当作常规的单播分组发送至目标多播路由器的 IP 地址。

隧道是由手工来配置的。通常,隧道位于一条存在物理连接的路径上,但这并不是必需的。如果一条隧道下面的物理连接偶然发生了故障,则使用该隧道的多播路由器甚至不会觉察到这种情况,因为 Internet 将自动地重新路由它们之间的 IP 流量,使其经过其他的线路。

如果又出现一个新的岛,并且它想加入 MBone,比如图 7.81 中的 G,那么它的管理员向 MBone 的邮件列表发送一条消息来声明它的存在。然后,附近站点的管理员与他联系,以便设法建立隧道。有时候,现有的隧道也需要被重新配置,以便充分利用新的岛来优化拓扑结构。毕竟,隧道并不是物理存在的,它们是由多播路由器中的内部表格来定义的,管理员只需改变这些表格中的表项就可以增加、删除或者移动隧道。一般地,MBone 上的每个国家都有一个骨干网,该国家中的区域性岛都连接在此骨干网上。通常 MBone 被配置成有一条或两条跨越大西洋和太平洋的隧道,以便使 MBone 变成全球范围的多播网络。



因此,在任何时刻,MBone 都有一个特定的拓扑结构,该拓扑结构由岛和隧道组成,而且与当前正在使用的多播地址的数量以及谁正在收听或观看这些多播地址无关。这种情形与一个普通的(物理)子网非常相似,所以普通的路由算法也可以应用在 MBone 上。因此,MBone 最初使用一个基于 Bellman-Ford 距离矢量算法的路由算法 **DVMRP** (**Distance Vector Multicast Routing Protocol**, 距离矢量多播路由协议)。例如,在图 7.81 中,岛 C 可以通过 B 或 E(或者设法通过 D)路由到 A。为了做出选择,它先获得由这些节点提供的、从它们到 A 的距离值,然后加上从它到这些节点之间的距离。通过这种方式,每个岛就能决定到其他每个岛的最佳路径。但我们很快就会看到,实际上这些路径并没有真正被使用。

现在我们来考虑一下多播传送实际上是如何进行的。为了多播一个音频或视频节目,源主机必须首先获得一个 D 类多播地址,该地址相当于电台的频率或者频道号。D 类地址是通过一个程序在数据库中查找空闲的多播地址来预定的。同一时刻可以有多个多播通信,主机通过监听合适的多播地址,就可以“收听”它所感兴趣的多播。

每台多播路由器定期地在它的岛范围内发送一个 IGMP 广播分组,询问谁对哪个通道感兴趣。如果一台主机期望(继续)接收一个或多个通道,则它送回另一个 IGMP 分组作为回应。这些回应在时间上是错开的,以避免使本地 LAN 超载。每台多播路由器都维护一张表,其中记录了它必须向它的 LAN 转发哪些通道的数据,以免当有些通道没有人想要时它仍然转发该通道的数据,从而浪费带宽。

通过 MBone 来实现多播传送的过程如下所述。当一个音频源或视频源生成一个新的分组时,它利用硬件多播设施将该分组多播到它的本地岛中。本地多播路由器在收到了该分组以后,将它复制到所有与其相连接的隧道上。

然后,凡是通过隧道获得了这样一个分组的的多播路由器都要检查该分组,看它是否沿着最佳路径到来,所谓最佳路径,是指路由器的内部表说明了它使用该路径来到达源主机(就好像它是目标一样)。如果该分组确实沿着最佳路径到来,则多播路由器将它复制到所有其他的隧道上。如果该分组是通过非最佳路径而到来的,则它被丢弃。因此,举例来说,在图 7.81 中,如果 C 的表指明了它应该使用 B 来到达 A,那么,当一个来自 A 的多播分组经过 B 到达 C 时,该分组被复制至 D 和 E。然而,当一个来自 A 的多播分组经过 E(并非最佳路径)而到达 C 时,它被简单地丢弃掉。这个算法就是我们在第 5 章中看到过的逆向路径转发算法(reverse path forwarding algorithm),尽管它并不完美,但已经非常好了,而且实现起来也很简单。

除了使用逆向路径转发算法来防止无节制地扩散到 Internet 上以外,还可以使用 IP 的 Time to live(TTL)域来限制多播的范围。每个分组在出发时都被指定一个特定的 TTL 值(由多播源决定)。每条隧道被分配一个权值。只有当一个分组有足够的 TTL 时,它才可以被通过一条隧道;否则它就会被丢弃。例如,越洋的隧道通常被配置为 128 的权值,所以,只要源主机在发送分组时将它的 TTL 值设定为 127 或者更小,就可以将分组限定在它自己所在的大陆上。当通过一条隧道以后,分组的 TTL 值被减去该隧道的权值。

虽然 MBone 的路由算法可以工作,但还是有许多研究工作致力于对它的改进。其中

一种建议方案继续保留了距离矢量路由的想法,但通过将 MBone 站点组织成一些区域,并首先路由到区域,从而使算法有了层次(Thyagarajan and Deering, 1995)。

另一种建议方案是,用一种修改形式的链接状态路由算法来代替距离矢量路由。尤其是,IETF 的一个工作组修改了 OSPF,以便使它适合于在单个自治系统内部进行多播传送,这样得到的多播 OSPF 被称为 MOSPF(Moy, 1994)。其中所做的修改是,让 MOSPF 建立起来的完整地图不仅记录下常规的路由信息,还记录下多播岛和隧道的信息。有了完整的拓扑结构以后,要计算出从每个岛到其他每个岛的最佳隧道路径就非常容易了,例如,使用 Dijkstra 算法。

第二个研究领域是 AS(自治系统)之间的路由。在这里,另一个 IETF 工作组开发了一个被称为 PIM(Protocol Independent Multicast,协议无关多播)的算法。PIM 有两个版本,划分的依据是岛的疏密程度:密集模式(几乎每个人都想观看)和稀疏模式(几乎没有人想观看)。这两个版本都使用了标准的单播路由表(所以是协议无关的),而不像 DVMRP 和 MOSPF 那样创建一个层叠的拓扑结构。

在 PIM-DM(dense mode,密集模式)中,基本的思想是剪除无用的路径。剪除过程如下所述。当一个多播分组从一条“错误的”隧道上到来时,则通过该隧道送回一个剪除分组,告诉发送方,以后不要再将来自同一个源的分组发送过来。当一个多播分组通过“正确的”隧道到来时,则将它复制到所有在此之前还没有剪除自己的其他隧道上。如果其他所有的隧道都已经剪除了它们自己,并且本地岛内对该多播分组也没有兴趣,则多播路由器通过“正确的”隧道送回一条剪除消息。通过这种方式,多播就能自动适应,并且只到达有人想要它的地方去。

RFC 2362 中描述的 PIM-SM(sparse mode,稀疏模式)的工作方式有所不同。这里的基本思想是,防止因为 Berkeley 的三个人想要通过一个 D 类地址来开一个会就导致 Internet 饱和。PIM-SM 的做法是建立一些汇合点(rendezvous point)。在一个 PIM-SM 多播组中,每个源都将它们的分组发送给汇合点。任何有兴趣加入多播组的站点可以请求其中一个汇合点,要求建立一条从该汇合点到本站点的隧道。这样,所有的 PIM-SM 流量都是通过单播而不是多播来传输的。PIM-SM 正变得越来越流行,而且 MBone 正在向 PIM-SM 迁移。由于 PIM-SM 被应用得越来越广泛,所以,MOSPF 正在逐渐消失。另一方面,MBone 本身看起来也有点停滞了,可能永远也不会大规模流行了。

然而,即使 MBone 并没有获得巨大成功,但网络化的多媒体仍然是一个既令人振奋又快速变化的领域。每天都有新的技术和应用显露出来。正如在(Striegel and Manimaran, 2002)中所讨论的那样,多播和服务质量正在日益融合。另一个热门话题是无线多播(Gossain et al., 2002)。整个多播领域和所有与之相关的事物,可能在未来的几年中仍然很重要。

## 7.5 本章小结

Internet 上的命名机制使用了一种被称为域名系统(DNS)的分层方案。在顶层是众所周知的通用域,包括 com 和 edu 以及大约 200 个国家域。DNS 被实现为一个分布式数

数据库系统,其服务器遍布全球。DNS 保存了各种记录,这些记录包含了 IP 地址、邮件交换和其他信息。一个进程通过查询 DNS 服务器,就可以将一个 Internet 域名映射到一个 IP 地址上,以便与该域进行通信。

电子邮件是 Internet 的两个杀手锏应用之一。从小孩子到老爷爷,现在每个人都在使用电子邮件。世界上绝大多数电子邮件系统都使用了 RFC 2821 和 2822 中定义的邮件系统。在这种系统中发送的消息都使用 ASCII 头来定义消息的属性。通过 MIME 可以发送许多种类的内容。使用 SMTP 可以发送消息,SMTP 的工作方式很简单:首先建立一个从源主机到目标主机的 TCP 连接,然后直接在此 TCP 连接上传递电子邮件。

Internet 的另外一个杀手锏应用是万维网(WWW)。Web 是一个链接超文本文档的系统。最初,每个文档都是一个用 HTML 来编写的页面,页面中包含一些指向其他文档的超链接。现在,XML 正开始逐渐接管 HTML。另外,有大量的内容是动态生成的,可能是由服务器端脚本(PHP、JSP 和 ASP)动态生成的,也可能是由客户端脚本(特别是 JavaScript)动态生成的。浏览器可以显示一个文档,其过程是:与文档所在的服务器建立一个 TCP 连接,并请求该文档,然后关闭 TCP 连接。这些请求消息包含各种各样的头,客户通过这些消息头来提供额外的信息。缓存、复制和内容传输网络被广泛用于提高 Web 的性能。

无线 Web 只是刚刚开始。最初的系统是 WAP 和 i-mode,它们都只有小的屏幕和有限的带宽,但是下一代将会更加强大。

多媒体也是网络天空中正在冉冉上升的一颗新星。它允许音频和视频通过电子方式被数字化,然后被传输和显示。音频要求较少的带宽,所以它可以一路向前。流式音频、Internet 电台、IP 语音现在都已经成为现实,而且新的应用总是在不停地涌现出来。视频点播是一个很有前途的领域,人们对它有极大的兴趣。最后,MBone 是一个实验性的全球数字电视实况服务,它通过 Internet 来发送多媒体数据。

## 习 题

1. 许多商用计算机有三个不同的全球惟一标识符。它们是什么?
2. 根据图 7.3 中给出的信息, little-sister. cs. vu. nl 在一个 A、B 还是 C 类网络上?
3. 在图 7.3 中,在 rowboat 之后没有点号,为什么?
4. 猜测一下微笑图标“:-X”(有时候也被写成“:-#”)可能代表的含义。
5. DNS 使用 UDP 协议而不是 TCP 协议。如果一个 DNS 分组丢失了,这里也没有自动恢复的机制。请问这会产生问题吗? 如果会的话,它是如何被解决的?
6. 除了可能会丢失,UDP 分组还有最大长度限制,甚至可能少到只有 576 个字节。当一个待查找的 DNS 名字超过这个长度时会怎么样? 它可以被放在两个分组中发送吗?
7. 如果一台机器只有一个 DNS 名字,那么它可以有多个 IP 地址吗? 这种情形是如何发生的?
8. 一台计算机可以有两个分别属于不同顶级域的 DNS 名字吗? 如果可以,请给出一个可能的例子。如果不可以,请解释原因。

9. 近年来,拥有 Web 站点的公司的数量爆炸式地增长。结果是,成千上万的公司被注册在 com 域中,从而导致该域的顶级服务器的负载极其繁重。请提出一种建议方案来缓解这个问题,同时又不修改命名方案(也就是说,不引入新的顶级域名)。你的方案可以要求修改客户代码。

10. 有些电子邮件系统支持 Content Return: 头域。它指定了当消息未被递交时是否返回消息体。请问:这个域属于信封还是消息头?

11. 电子邮件系统需要相应的目录,以便于查找人们的电子邮件地址。为了建立这样的目录,名字应该被分割成标准的组成部分(比如说姓和名)以便于搜索。请讨论为了建立一个可接受的全球标准而需要解决的一些问题。

12. 一个人的电子邮件地址是:他的或者她的登录名@带有 MX 记录的 DNS 域的名字。登录名可以是姓、名、首字母缩写,以及其他类型的名字。假设一家大公司认为,由于人们不知道收件人的登录名,因而太多的电子邮件被丢失了。是否有一种办法可以让他们在不改变 DNS 的情况下解决这个问题。如果有,请给出一个建议方案,并解释它是如何工作的。如果没有,请解释为什么不可能。

13. 一个二进制文件的长度为 3072 个字节。每发送 80 个字节,以及在最后一次发送中都要插入一对 CR+LR,请问,如果采用 base64 编码,它将有多长?

14. 请考虑 MIME 的可打印引用编码方案。提出一个在正文中未讨论过的问题,并给出一种解决方案。

15. 指出 5 种本书中没有列出的 MIME 类型。你可以查看你的浏览器或者 Internet 来获得信息。

16. 假设你想要发送一个 MP3 文件给一个朋友,但是你朋友的 ISP 限制进入邮件的大小为 1MB,而该 MP3 文件是 4MB。通过使用 RFC 822 和 MIME,是否有办法可以处理这种情形?

17. 假设某人安装了一个假期守护程序,然后在刚刚退出之前发送了一条消息。不幸的是,接收者已经在休假了(为期一周),而且也运行了一个假期守护程序。那么接下来将会发生什么事情?被封装的回复消息会不会来回地传送,直到某个人休假回来为止吗?

18. 在任何一个标准中,例如 RFC 822 中,需要精确的语法来表示哪些内容是允许的,这样不同的实现才能相互协同工作。即使最简单的条目也必须小心地定义。SMTP 头允许在标记符之间出现空白。请针对标记符之间的空白,给出两种可能的定义。

19. 假期守护程序是用户代理的一部分还是消息传输代理的一部分?当然,它是利用用户代理来建立的,但是用户代理真的发送回复吗?解释你的答案。

20. POP3 允许用户从远程邮箱中获取并下载电子邮件。这是否意味着邮箱的内部格式必须被标准化,以便客户端的任何一个 POP3 程序都能够读取任何邮件服务器上的邮箱吗?讨论你的答案。

21. 从 ISP 的角度来看,POP3 和 IMAP 在一个很重要的方面有所不同。POP3 用户通常每天都会清空他们的邮箱,而 IMAP 用户则无限期地将他们的邮件保存在服务器上。假设现在邀请你为一个 ISP 提出建议,建议它应该支持哪个协议。请问,你会提出哪

些考虑事项?

22. Webmail 使用 POP3、IMAP, 还是两者都不使用? 如果使用了其中之一, 则为什么选择该协议? 如果两者都不使用, 那么从思想上讲, 它更接近于哪一个?

23. 当 Web 页面被发送出去时, 它们的前面被加上了 MIME 头。为什么?

24. 什么时候需要外部查看器(viewer)? 浏览器如何知道该使用哪一个查看器?

25. 是否有可能当用户在 Netscape 中单击一个链接时, 一个特殊的辅助应用程序被启动起来, 但是在 Internet Explorer 中单击同一个链接时, 却启动一个完全不同的辅助应用程序, 尽管在这两种情况下所返回的 MIME 类型是完全相同的? 请解释你的答案。

26. 一个多线程的 Web 服务器被组织成如图 7.21 所示的结构。它需要  $500\mu\text{s}$  来接受一个请求并检查缓存。在一半情况下, 它可以在缓存中找到文件, 并立即返回。另外一半情况下, 该模块必须阻塞  $9\text{ms}$  以等待它的磁盘请求被排队和处理。该服务器应该有多少个模块才能保持 CPU 一直处于忙的状态(假设磁盘不是一个瓶颈)?

27. 标准的 http URL 假设 Web 服务器在 80 端口上监听。然而, Web 服务器在其他的端口上进行监听也是有可能的。请设计一种合理的 URL 语法来支持在非标准端口上访问文件。

28. 尽管在正文中没有提及, URL 的一种替换形式是使用 IP 地址而不是 DNS 名字。一个使用 IP 地址的 URL 例子是 `http://192.31.231.66/index.html`。浏览器如何知道协议名后面的名字是 DNS 名字还是 IP 地址?

29. 想象斯坦福大学计算机系的某个人刚刚编写了一个新程序, 他希望通过 FTP 来分发这个程序。他把程序放在 FTP 目录中: `ftp/pub/freebies/newprog.c`。这个程序的 URL 可能是什么?

30. 在图 7.25 中, `www.aportal.com` 将用户的喜好记录在一个 cookie 中。这种方案的一个缺点是, cookie 的大小限制为 4KB, 所以, 如果这个用户的喜好特别多, 例如许多股票、运动队、各种类型的新闻故事、多个城市的天气、各种类别的产品目录等等, 则可能会到达 4KB 的限制。请另外设计一种方案来记录用户的喜好, 并要求这种方案不再存在同样的问题。

31. Sloth 银行希望它的在线银行更易于为懒惰的客户所使用, 所以, 当用户登录并通过口令认证以后, 银行返回一个包含用户 ID 号码的 cookie。通过这种方式, 用户将来访问在线银行时不必再标识自己或者输入口令。你认为这种想法怎么样? 它可以工作吗? 是不是一个好主意?

32. 在图 7.26 中, ALT 参数被设置在 `<img>` 标签中。浏览器在什么情况下使用这个参数? 如何使用?

33. 在 HTML 中, 如何使一个图像成为可单击的? 给出一个例子。

34. 给出必要的 `<a>` 标签, 使字符串“ACM”成为指向 `http://www.acm.org` 的超链接。

35. 请为一家新公司 Interburger 设计一个表单, 以便客户可以通过 Internet 订购汉堡包。这个表单应该包含客户的名字、地址和城市, 以及对尺寸的选择(巨型的或者大的)和奶酪选项。货送到后通过现金付款, 因此不需要信用卡信息。



36. 设计一个要求用户输入两个数值的表单。当用户单击提交按钮时,服务器返回它们的和。在服务器端用 PHP 脚本来编写。

37. 对于以下每一种应用,请说出它们(1)是否可能使用 PHP 脚本或者 JavaScript;(2)最好使用 PHP 脚本或者 JavaScript,并说明为什么。

(a) 针对用户请求的月份(1752 年 9 月以后)显示日历。

(b) 显示从阿姆斯特丹到纽约的航班安排。

(c) 用用户提供的系数画出一个多项式。

38. 用 JavaScript 编写一个程序,它接受一个大于 2 的整数,并判断这个数是否为素数。请注意,JavaScript 有 if 和 while 语句,其语法与 C 和 Java 一样。取模的运算符是%。如果你需要计算 x 的平方根,请使用 Math.sqrt(x)。

39. 一个 HTML 页面如下:

```
<html> <body>
<a href="www.info-source.com/welcome.html"> Click here for info </a>
</body> </html>
```

如果用户单击超链接,则会打开一个 TCP 连接,并且向服务器发送几行数据。请列出所有被发送的行。

40. If-Modified-Since 头可以被用来检查一个缓存的页面是否还有效。其请求可以针对包含图像、声音、视频以及 HTML 等的页面。你认为这项技术的效率对于 JPEG 图像来说比 HTML 更好还是更差? 仔细考虑“效率”意味着什么并解释你的答案。

41. 在举行某项体育赛事的时候,比如某一流行体育项目的总决赛,很多人会访问它的官方 Web 站点。在这种情形下,是否也会出现与 2000 年佛罗里达州的选举同样的暂时拥挤呢? 为什么是或者为什么不是?

42. 单个 ISP 作为一个 CDN 有意义吗? 如果有,它将如何工作? 如果没有,这种想法有什么问题?

43. 在什么情况下使用 CDN 是一个坏主意?

44. 无线 Web 终端只有很低的带宽,这使得高效的编码方案非常重要。请设计一种方案,使得可以有效地通过无线链路向 WAP 设备传输英文文本。你可以假设 Web 终端有几兆 ROM,有一个中等强度的 CPU。提示:考虑一下你如何传输日语,在日语中每个符号是一个词。

45. 一片 CD 可容纳 650MB 的数据。音频 CD 使用压缩了吗? 解释你的理由。

46. 在图 7.57(c)中,由于采用 4 位采样来表示 9 个信号值,因而产生了量化噪声。第一个采样,在位置 0 处,是精确的,但是接下来的一些采样是不精确的。请问,在周期 T 的 1/32、2/32 和 3/32 处,采样值的百分比误差是多少?

47. 可以利用心理声学的模型来减少 Internet 电话所需的带宽吗? 如果可以,那么必须满足什么条件(如果有的话)该模型才能工作? 如果不可以,请问为什么?

48. 一台音频流式服务器与媒体播放器之间的单程距离为 50ms。服务器的输出速率为 1Mbps。如果媒体播放器有一个 1MB 的缓冲区,你能说出低水印标记和高水印标



记的位置吗?

49. 图 7.60 中的交错算法具有能够抵抗偶尔丢失分组而不会发生播放间隙的优点。然而,当用于 Internet 电话时,它也有一个小的缺点,是什么呢?

50. IP 语音面对防火墙时是否也有与流式音频一样的问题? 请讨论你的答案。

51. 以 40 帧/秒的速率来传输未压缩的彩色图像所需要的位速率是多少? 假设图像的大小为  $800 \times 600$  像素,每个像素的颜色为 8 位。

52. 一个 MPEG 帧中的 1 位错误是否不仅仅影响到发生错误的那一帧? 请解释你的答案。

53. 请考虑一台 100 000 用户的视频服务器,每个用户每个月看两部电影。有一半的电影是在晚上 8 点播放的。在这段时间中,服务器必须同时传输多少部电影? 如果每部电影需要 4Mbps,则服务器需要多少个 OC-12 网络连接?

54. 假定 Zipf 定律对于访问一台包含 10 000 部电影的视频服务器也成立。如果该服务器将最流行的 1000 部电影存放在磁盘上,其他 9000 部放在光盘上,那么,请给出点播到磁盘上的电影的百分比表达式。写个小程序来计算这个表达式。

55. 在计算机领域中也有一些占地者,他们将一些知名公司的误拼写名称注册成 DNS 域名,例如 `www.microsoft.com`。请列出至少 5 个这样的域。

56. 许多人注册了形如 `www.word.com` 的 DNS 名字,其中 word 是一个常见的单词。对于下面给出的每一种类别,请列出 5 个 Web 站点并简要地描述这些站点(例如, `www.stomach.com` 是长岛的一个肠胃病学家)。类别列表为:动物、食物、家庭日常用品和身体部位。对于最后一种类别,请严格使用腰部以上的身体部位。

57. 请使用  $12 \times 12$  的位图来设计你自己的一些 emoji(文字图标)。包括男朋友、女朋友、教授和政治家。

58. 编写一个可以接受以下命令的 POP3 服务器:USER、PASS、LIST、RETR、DELE 和 QUIT。

59. 请重新编写图 6.6 的服务器,使它成为一个可支持 HTTP 1.1 的 GET 命令的真正 Web 服务器。它应该也可以接受 Host 消息。服务器应该维护一个缓存,其中存放了最近从磁盘上获取到的文件,而且在可能的情况下,服务器应该从缓存中读取数据并返回给客户。

## 第8章 网络安全

在最初几十年中,计算机网络主要被大学的研究人员用于发送电子邮件,以及被公司的员工用于共享打印机。在这样的条件下,安全问题并没有得到足够的关注。但现在,成千上百万的普通市民利用网络来完成银行事务处理、购物和填写纳税单等活动,因此,网络安全逐渐成为一个巨大的潜在问题。在本章中,我们将从几个角度来学习网络的安全问题,并指出大量的缺陷,以及讨论许多能使网络更加安全的算法和协议。

安全性是一个范围很广阔的话题,同时也涉及到大量的违法犯罪活动。在最简单的形式中,它关心的是如何保证好事者们不能读取或者(更糟的是)悄悄地修改给其他人的消息。它也涉及到有人企图访问未经授权的远程服务。它还要处理像这样的情形:如何判断一条自称来自 IRS(美国国税局)的“星期五之前付款”的消息确实来自 IRS,而并非来自 Mafia(黑手党)。安全性也涉及到合法消息被捕捉并重放的问题,以及人们企图否认自己曾经收到过某些特定的消息。

大多数安全问题都是由于某些恶意的人企图获得某种收益、引起别人关注,或者伤害他人而有意制造的。图 8.1 列出了一些最为常见的作恶者。从这个列表中应该很清楚地看出,要想使一个网络变得更加安全,需要做的事情很多,而不仅仅是改正编程错误就可以了。这通常包括要与一些非常聪明、专一,甚至还有足够资助的攻击者进行较量。同时也应该很清楚地看到,一些能挫败低水平攻击的手段对于那些专业的攻击者来说并没有

攻击者	目标
学生	喜欢窥探别人的电子邮件
破坏者(cracker)	考验某人的安全系统;偷取数据
销售代表	声称可以代表整个欧洲,而不仅仅是安道尔共和国
商人	想找到竞争对手的市场计划策略
离职雇员	被解雇之后实施报复
会计	挪用公司钱款
股票经纪人	否认自己曾经通过电子邮件对顾客做过承诺
骗子	偷取信用卡号码再转卖
间谍	了解敌方的军事或者工业机密
恐怖分子	偷取细菌战机密

图 8.1 引发安全问题的人群和目标

多大效果。警方的记录表明,大多数攻击事件并不是由外部人员通过搭接电话线而侵入的,而是内部人员因为怀恨在心而实施的报复行为。因此,在设计安全系统时应该牢记这样的事实。

网络安全问题可以被粗略地分成 4 个相互交织的领域:保密、鉴别、不可否认和完整性控制。保密(secretcy)也被称为机密(confidentiality),它的任务是确保信息不会被未经授权的用户访问。这也正是当人们考虑网络安全时首先想到的内容。认证是指当你在展示敏感信息或者进入商务交易之前你必须确定自己在跟谁通话。不可否认性涉及到签名:你的顾客下了一份要购买 1000 万个左手器具的订单,他事后声称每个器具的单价为 69 美分,你如何证明他原来定购的价格是 89 美分呢?或者他事后根本不承认自己曾经下过订单。最后,你如何确定你所接收到的一条消息真的是某个人发送的,而不是被恶意攻击者在传输途中修改的消息或者伪造的消息?

所有这些问题(保密、鉴别、不可否认和完整性控制)也发生在传统的系统中,但是有一些重要的区别。在传统的系统中,利用挂号信和文档上锁的做法可以获得完整性和保密性。现在,抢劫邮政列车比过去 Jesse James 时代(美国内战期间铁路巨头们为了在中西部铺设铁路而与当地的人民发生了激烈的冲突)要困难得多。

而且,人们通常能够辨别出原始的纸介文档和复印件之间的区别,这对它们确实很重要。你不妨可以做一个试验,把手头的一张有效支票复制一份,星期一将支票原件拿到银行去兑现,然后星期二用复印件再试着兑现一次。你可以观察到银行的处理会有所不同。若使用电子支票,则原始支票和复制的支票是不可区分的,银行可能需要花一段时间来学习应该如何处理这种情况。

人们通过辨别相貌、声音和笔迹来识别其他的人,通过在信笺纸上签字或者盖印章等手段可以作为签名的证据。篡改行为通常可以被笔迹、墨水和纸张专家检测出来。所有这些手段在电子方式下都是不可用的,因此,我们需要其他的方案。

在讨论电子方案之前,首先值得花一点时间来考虑一下网络安全位于协议栈的什么位置上。可能并不存在单一的位置来容纳网络安全的所有特性。每一层都涉及到安全的某些方面。在物理层上,可以通过将传输线封装在内含高压气体的密封管线中以有效地对付搭线窃听的攻击手段。若企图在管线上钻孔,则气体就会泄漏,从而降低气压,并触发报警。有些军用系统使用了这种技术。

在数据链路层上,对于点到点线路上的分组,它们在离开一台机器的时候可以被加密,当进入另一台机器的时候被解密。所有的细节可以在数据链路层内部来处理,上面的层对于数据链路层上发生的事情一无所知。然而,当分组必须要穿越多台路由器的时候,这种方案可能无法正确地工作,因为在每一台路由器上分组必须被解密,从而使得这些分组容易遭受来自路由器内部的攻击。而且,它也不允许只保护某些会话(比如那些涉及到通过信用卡进行在线购物的会话),而其他的会话不保护。这种方法称为**链路加密(link encryption)**,正如其名称所示,它可以很容易地被应用到任何一个网络中,所以它往往很有用。

在网络层上,可以通过安装防火墙来过滤好的分组和坏的分组,让好的分组正常通过,坏的分组禁止出入。IP 安全(IPSec)也运行在这一层上。

在传输层上,整个连接可以以端到端的方式来加密,也就是说,从进程到进程的加密方式。为了达到最高级别的安全,端到端的安全是必需的。

最后,诸如像用户认证和不可否认性这样的问题只能在应用层上得以解决。

由于安全问题并不恰好吻合于某一层,所以,它也不适合于在本书前面任何一章中讨论。由于这个原因,我们用专门一章来讨论安全问题。

虽然本章很长,技术性很强,而且也比较基础,但是,它相对来说比较独立。有很多很好的材料描述了银行发生安全事件的缘由,比如,由于不合格的雇员、松懈的安全程序,或者内部人员的欺诈行为等,往往并不是因为聪明的罪犯搭接和窃听电话线并且解密所听到的消息而造成的。如果一个人拿着一张在马路上捡到的 ATM 卡随便走进一家银行的分支结构并声称自己忘了 PIN 码,而且当场获得新的 PIN 码(凭着很好的客户关系),那么,世界上所有的密码系统都无法防止这样的事件。从这个角度而言,Ross Anderson 的书是一本真正让人大开眼界的好书,它记录了在各个行业中发生的数百个安全失败的例子,几乎所有这些例子都是由于商务管理松散或者对安全性掉以轻心(这是客气的说法)(Anderson, 2001)。尽管如此,我们还是非常乐观,因为随着电子商务变得越来越普及,企业最终会对它们的操作程序进行测试,并改正其中的错误和不合理之处,消除隐患和漏洞,从而将技术层而上的安全性再次带到中心舞台上。

除了物理层安全以外,几乎所有的安全性都建立在密码学原理的基础上。出于这样的原因,我们在学习安全性的时候首先详细地学习密码学的知识。在 8.1 节中,我们将介绍一些基本的原理;在 8.2 至 8.5 节中,我们讨论一些在密码学中用到的基础算法和数据结构。然后,我们详细地讨论如何使用这些概念来实现网络中的安全性。最后我们将简短地介绍一些有关技术和社会问题的思考来结束本章。

在开始之前,有必要提一下哪些内容本章没有涉及到。我们尽量将焦点集中在与网络有关的问题上,而不是与操作系统和应用系统有关的问题上,但是,线路上的事情往往更加难以描述得清楚。例如,本章不讨论以下内容:基于生物统计学的用户认证、口令安全、缓冲区溢出攻击、特洛伊木马、登录欺骗、逻辑炸弹、病毒、蠕虫等。“Modern Operating Systems”(Tanenbaum, 2001)一书的第 9 章有较大的篇幅介绍了所有这些话题。感兴趣的读者可以参考这本书以便对系统层而的安全性有一个初步的了解。现在让我们开始网络安全之旅。

## 8.1 密 码 学

密码学一词(cryptography)来自于希腊语中的短语“secret writing(秘密地书写)”。它有着辉煌而又悠久的历史,往前可以追溯几千年。在这一节中我们只是勾画出其中一些亮点,然后介绍一些背景信息。有关完整的密码学历史,你可以阅读 Kahn(1995)的书。要想全面地了解安全和密码算法的最新现状、协议和应用,请参考(Kaufman et al., 2002)。有关数学化的方法,请参考(Stinson, 2002);相对数学成分较少的方法,请参考(Burnett and Paine, 2001)。

专业人员对密码和编码作了明确的区分。密码(cipher)是指逐个字符或者逐个位地

进行变换,它不涉及到消息的语言结构。相反,编码(code)则是指用一个词或符号来代替另一个词。尽管编码学有着非常光荣的历史,但是现在已经很少再使用了。在历史上设计最为成功的编码系统是二次世界大战中美国军队在太平洋战场上使用的编码。他们的做法很简单,让纳瓦霍印第安人用专门的纳瓦霍语言来交流军事术语,例如,用 chay-dagahi-nail-tsaidi(字面意思是慢行者之克星)来表示反坦克武器。纳瓦霍语言是一门以音调为主的语言,它非常复杂,而且没有书面的形式。在日本,没有人懂这门语言。

在1945年的9月,圣地亚哥联合会如此描述美国的编码系统:“三年来,无论什么时候当海军登陆的时候,日本人总是听到一连串奇怪的汩汩声,同时夹杂着像是西藏和尚念经的声音和热水瓶倒空时发出的声音。”日本人始终未能破解这套编码系统,纳瓦霍编码的许多配音者由于突出的表现和勇敢精神而获得了很高的军事荣誉。美国破解了日本的编码,而日本却未能破解纳瓦霍编码,这个事实本身在太平洋战争中为美国的胜利扮演了重要的角色。

### 8.1.1 密码学简介

在历史上,有四组人用到了密码学,并且为之作出了贡献,他们是:军事人员、外交人员、写日记者和情侣。在这4种人中,军事人员扮演了重要的角色,而且几个世纪以来他们不断完善着这个领域。在军事组织内部,需要加密的消息通常被交给收入低微的低级译码员来加密和传输。由于消息的数量不大,所以,这项工作不必仰仗少量的专家来完成。

在计算机出现以前,密码学的一个主要限制是译码员执行各种必要的变换的能力,尤其在战场上往往缺乏相应的装备。另一个限制是很难从一种密码方法切换到另一种密码方法,因为这需要重新训练一大批人。然而,由于译码员有可能被敌人俘虏,所以,在必要的时候能及时地更换密码方法变得非常关键。这些相互矛盾的需求导致产生了如图8.2所示的模型。

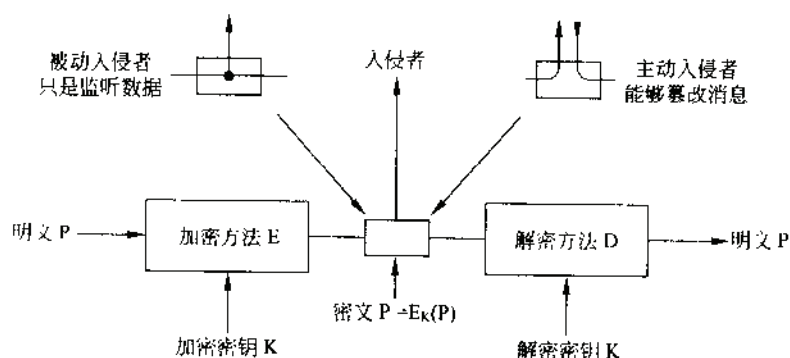


图 8.2 加密模型(假定使用了对称密钥密码)

待加密的消息称为明文(plaintext),它经过一个以密钥(key)为参数的函数变换,这个过程称为加密,输出的结果称为密文(ciphertext),然后,密文被传送出去,往往由通讯员或者无线电方式来传送。我们假设敌人或者入侵者(intruder)听到了完整的密文,并且



将密文精确地复制下来。然而,与目标接收者不同的是,他不知道解密密钥是什么,所以它无法很轻易地对密文进行解密。有时候入侵者不仅可以监听通信信道(被动入侵者),而且还可以将消息记录下来并且在以后某个时候回放出来,或者插入他自己的消息,或者在合法消息到达接收方之前对消息进行篡改(主动入侵者)。破译密码的艺术称为密码分析学(cryptanalysis),它与设计密码的艺术(cryptography)合起来统称为密码学或密码术(cryptology)<sup>①</sup>。

用一种合适的标记法将明文、密文和密钥的关系体现出来,这往往会非常有用。我们将使用  $C=E_K(P)$  来表示用密钥  $K$  加密明文  $P$  得到密文  $C$ 。类似地,  $P=D_K(C)$  代表了解密  $C$  得到明文  $P$  的过程。由此可以得到:

$$D_K(E_K(P)) = P$$

这种标记法也说明了  $E$  和  $D$  只是数学函数,事实上也确实是这样。惟一值得特别注意的地方是,它们都是带两个参数的函数,但是我们将其中一个参数(密钥)写成下标的形式,而不写成实参的形式,从而将它与消息本身区别开来。

密码学的基本规则是,你必须假定密码分析者知道加密和解密所使用的方法。换句话说,密码分析者知道图 8.2 中加密方法  $E$  和解密方法  $D$  的所有工作细节。每次当老的加解密方法被泄漏(或者认为它们已被泄漏)以后,总是需要极大的努力来重新设计、测试和安装新的算法,这使得将加密算法本身保持秘密的做法在现实中并不可行。当一个算法已不再保密的时候仍然认为它是保密的,这将会带来更大的危害。

这正是密钥发挥作用之处。密钥是由一段相对比较短的字符串构成的,它相当于从大量潜在的加密方法中选择了一种加密方法。一般的方法可能几年才会发生变化,与此不同的是,密钥可以根据需要频繁地被改变。因此,我们的基本模型是一个稳定的、广泛公开的通用方法,它用一个秘密的、易改变的密钥作为参数。“让密码分析者知道加解密算法,并且把所有的秘密信息全部放在密钥中”这种思想被称为 **Kerckhoff 原则**(**Kerckhoff's principle**),这是用佛兰德军事密码学家 Auguste Kerckhoff 的名字来命名的,因为他在 1883 年第一次声明了这种思想(Kerckhoff, 1883)。因此,我们有:

**Kerckhoff 原则:** 所有的算法必须是公开的,只有密钥是保密的。

算法的不保密性怎么强调都不过分。企图使算法保持秘密的做法(这种做法也称为含糊的安全性[security by obscurity])永远不会奏效。而且,将算法公开以后,密码设计者可以自由地与大量学院派的密码学家进行交流探讨,这些密码学家总是期望能破译密码系统,从而可以发表论文,来证明自己有多聪明。如果一个密码算法被公开 5 年以后,尽管在此期间许多专家试图破解该算法,但是无人能够成功,那么,这个算法应该是非常可靠的。

由于真正的秘密在密钥中,所以它的长度是一个非常重要的设计要素。请考虑一个简单的号码锁。一般的原则是,你按照顺序输入数字。每个人都知道这一点,但是锁的号码(即密钥)是保密的。如果密钥长度是两个数字,则意味着共有 100 种可能。若密钥长

<sup>①</sup> 译注: cryptography 的确切含义是密码编码学,它与 cryptanalysis(密码分析学)合起来称为密码学。由于本书中基本上没有介绍密码分析学,所以,为了简便起见,我们将 cryptography 也简称为密码学。



度为三个数字,则意味着共有 1000 种可能;同样地,六位数字的密钥意味着一百万种可能。密钥越长,则密码分析者要应对的工作因子(work factor)也越高。通过穷举搜索整个密钥空间来破解密码系统,这种做法的工作因子是密钥长度的指数量级。保密性来自于两个方面,一是强而牢固的(但是公开的)算法,二是长的密钥。为了防止你顽皮的弟弟阅读你的电子邮件,64 位密钥就够了。对于常规的商业用途,至少应该使用 128 位密钥。为了阻挡大的政府机构,则至少需要 256 位密钥,而且越长越好。

从密码分析者的角度来看,密码分析问题有三个主要的变种。当他得到了一定量的密文,但是没有对应的明文时,他面对的是“只有密文(ciphertext-only)”问题。报纸上猜谜栏目中的密码难题就属于这一类问题。当密码分析者有了一些相匹配的密文和明文时,密码分析问题被称为“已知明文(known plaintext)”问题。最后,当密码分析者能够加密某一些他自己选择的明文时,问题就变成了“选择明文(chosen plaintext)”问题。如果密码分析者可以问诸如“ABCDEFGHijkl 经过加密之后是什么?”之类的问题,那么,报纸上的密码难题很容易可被破解。

密码学领域中的新手通常作这样的假设:如果一个密码能够对抗“只有密文”攻击,那么它就是安全的。这个假设是非常幼稚的。在许多情况下,密码分析者可以较为正确地猜测出明文的某些部分。例如,当你向其他的机器发出请求的时候,许多计算机回答的第一件事情是“login:”。一旦有了一些相匹配的明文-密文对以后,密码分析的任务将容易得多。为了获得安全性,密码设计者应该保守一点,并且保证:即使对手能够加密任意数量的选择明文,密码系统也不会被攻破。

在历史上,加密方法被分成两大类:置换密码和转置密码。现在我们简要地介绍这两种密码,正好作为现代密码学的背景信息。

### 8.1.2 置换密码

在置换密码(substitution cipher)中,每个字母或者每一组字母被另一个字母或另一组字母来取代,从而将原来的字母掩盖起来。最古老的密码之一是凯撒密码(Caesar cipher),它因为来源于 Julius Caesar 而得名。在这种方法中,a 变成 D,b 变成 E,c 变成 F,...,z 变成 C。例如,attack 变成 DWDFN。在例子中,明文以小写字母给出,密文则使用大写字母。

凯撒密码的一种略微通用化的方案是,允许明文字母表被移动 k 个字母,而并不总是移动 3 个字母。在此情况下,k 变成了这种循环移动字母表的通用加密方法的一个密钥。凯撒密码也许确实欺骗了庞培(Pompey),但自那以后再也没有骗过别人。

接下来的改进是,让明文中的每个符号(为了简化起见,这里假设为 26 个字母)都映射到其他某一个字母上,例如,

明文: a b c d e f g h i j k l m n o p q r s t u v w x y z

密文: Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

这种“符号对符号”进行置换的通用系统被称为单字母表置换(monoalphabetic substitution),其密钥是对应于整个字母表的 26 字母串。对于上面的密钥,明文 attack 被变换为密文 QZZQEA。

初看起来,这似乎是一个非常安全的系统,因为虽然密码分析者了解通用的系统(即字母对字母的置换),但是,它不知道到底使用哪一个密钥,而密钥的可能性共有  $26! \approx 4 \times 10^{26}$  种。与凯撒密码不同的是,要试遍所有这么多种可能的密钥不是一种可行的做法。即使一台计算机测试每个密钥只需 1ns,试遍所有的密钥也将需要  $10^{10}$  年时间。

然而,只要给出相对少量的密文,就可以很容易地破解该密码。基本的攻击手段利用了自然语言的统计特性。例如,在英语中,e 是最常见的字母,其次是 t、o、a、n、i,等。最常见的两字母组合(或者两字母连字)是 th、in、er 和 an。最常见的三字母组合(或者三字母连字)是 the、ing、and 和 ion。

密码分析者为了破解单字母表密码,他首先计算密文中所有字母的相对频率,然后,他可以试探性地将最常见的字母分配给 e,次常见的字母分配给 t。接下来他查看三字母连字,找到比较常见的形如 tXe 的三字母组合,这强烈地暗示着其中的 X 是 h。类似地,如果模式 thYt 出现得很频繁的话,则 Y 可能代表了 a。有了这些信息以后,他可以查找频繁出现的形如 aZW 的三字母组合,它很有可能是 and。通过猜测常见的字母、两字母连字和三字母连字,并且利用元音和辅音的各种可能组合,密码分析者就可以逐字母地构造出试探性的明文。

另一种做法是猜测一个可能的单词或者短语,例如,考虑以下这段来自于一家会计事务所的密文(分成 5 个字符为一组):

```
CTBMN BYCTC BTJDS QXBNS GSTJC BSWX CTQTZ CQVUJ  
QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ  
DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW
```

在会计事务所的消息中,一个可能的单词是 financial。我们知道在 financial 这个单词中有一个重复的字母(i),并且这两个 i 之间有 4 个其他的字母,根据这样的知识,我们在密文中查找相隔 4 个位置的重复字母。我们可以找到 12 个地方,分别在 6、15、27、31、42、48、56、66、70、71、76 和 82 位置上。然而,只有其中两个地方,即 31 和 42,它的下一个字母(对应于明文中的 n)也在正确的位置上重复。而在这两者之中,只有 31 有正确的 a 位置(考虑在 financial 中有两个 a),所以,我们知道 financial 从位置 30 开始。以此为出发点,利用英语文本的频率统计规律可以很容易地推断出密钥。

### 8.1.3 转置密码

置换密码保留了明文符号的顺序,但是将明文伪装起来了。与此相反,转置密码(transposition cipher)重新对字母进行排序,但是并不伪装明文。图 8.3 给出了一个常见的转置密码:柱形转置。该方案用一个不包含任何重复字母的单词或者短语作为密钥。在这个例子中,密钥是 MEGABUCK。密钥的用途是对列进行编号,第 1 列是指定在密钥的字母中最靠近字母表起始的那个字母的下面,以此类推。明文按水平方向的行来书写,如果有必要的话填满整个矩阵。密文被按列读出,从编号最低的密钥字母开始逐列读出。

为了破解转置密码,密码分析者首先要明白,自己是在破解一个转置密码。通过查看 E、T、A、O、I、N 等字母的频率,很容易就可以看出它们是否吻合明文的常规模式。如果

<u>M</u>	<u>E</u>	<u>G</u>	<u>A</u>	<u>B</u>	<u>U</u>	<u>C</u>	<u>K</u>	
7	4	5	1	2	8	3	6	
p	l	e	a	s	e	t	r	Plaintext
a	n	s	f	e	r	o	n	pleasetransferonemilliondollarsto
e	m	i	l	l	i	o	n	myswissbankaccountsixtwo
d	o	l	l	a	r	s	t	Ciphertext
o	m	y	s	w	i	s	s	AFLLSKSOSELAWAIATOSSCTCLNMOMANT
b	a	n	k	a	c	c	o	ESILYNTWRNNTSOWDPAEDOBUEIRICXB
u	n	t	s	i	x	t	w	
o	t	w	o	a	b	c	d	

图 8.3 一个转置密码

是的话,则很显然这是一种转置密码,因为在这样的密码中,每个字母代表的是自己,从而不改变字母的频率分布。

接下来要猜测共有多少列。在许多情况下,从特定的环境信息中或许可以猜到一个可能的单词或者短语。例如,假定我们的密码分析者怀疑消息中的某个地方出现了明文短语 milliondollars。他观察到在密文中出现的两字母组合 MO、IL、LL、LA、IR 和 OS 是因为这个短语回绕的结果。密文字母 O 跟在密文字母 M 的后面(即在第 4 列的垂直方向上它们是相邻的),这可能是因为它们在短语中被一段等于密钥长度的距离所隔开。如果密钥长度为 7 的话,则两字母组合 MD、IO、LL、LL、IA、OR 和 NS 就会出现。实际上,对于每一个密钥长度,在密文中都会出现一组不相同的两字母组合。通过检查每一种可能性,密码分析者往往很容易就能够确定密钥的长度。

最后的步骤是确定列的顺序。当列数比较小(比如说  $k$ )的时候,则总共有  $k(k-1)$  种可能的列对,你可以对每一个列对进行检查,看它的两字母组合的频率是否与英语文本的两字母组合频率相匹配。假定最佳匹配的那一对已经有正确的位置关系了。现在,用剩下的每一列尝试着跟在这一对的后面,然后检查它的两字母组合和三字母组合的频率,假定最佳匹配的那一列是正确的。通过同样的方式可以陆续找到后继的列。整个过程继续下去,直至找出可能的列顺序关系。到这时候,通过检查明文就可以确定是否破解成功了(比如,如果出现 milloin 的话,很明显就知道错误在哪里了。)

有些转置密码接受一个固定长度的块作为输入,并产生一个固定长度的块作为输出。我们只要给出一个能指明字符输出顺序的列表,就可以完整地描述这样的密码。例如,图 8.3 中的密码可以被看作一个 64 字符块的密码。它的输出是 4、12、20、28、36、44、52、60、5、13、...,62。换句话说,第 4 个输入字符 a 首先被输出,然后是第 12 个字符,以此类推。

#### 8.1.4 一次一密

要想构建一个不可能被攻破的密码其实是非常容易的,相应的技术在几十年前就已经被发掘出来了。首先选择一个随机位串作为密钥,然后将明文转变成一个位串,比如使用明文的 ASCII 表示法。最后,逐位计算这两个串的异或(XOR)值。结果得到的密文不可能被破解,因为即使有了足够数量的密文样本,每个字符的出现概率是相等的,每个两

字母组合的概率也是相等的,三字母组合的概率也相等,以此类推。这种方法被称为一次一密(one-time pad),不管入侵者的计算能力有多么强大,这种密码总是能够对抗所有现在的和将来的攻击。其理由来自于信息论:在消息中没有任何信息,因为在指定长度的情况下,所有可能的明文都是等概率的。

图 8.4 给出了一个一次一密用法的例子。首先,消息 1“I love you.”被转换成 7 位 ASCII 码。然后选择一个一次性密钥 Pad1,并且与消息 1 进行异或(XOR)得到密文。密码分析者可以试验所有可能的一次性密钥,并检查每个密钥所对应的明文。例如,图中列出的一次性密钥 Pad2,可以被用来做试验,结果得到明文 2“Elvis lives”,这个结果有点似是而非(关于这方面的讨论超出了本书的范围)。实际上,对于每一个 11 字符长的 ASCII 明文,就有一个生成此明文的一次性密钥。这也正是我们所说的在密文中没有任何信息的原因:你总是可以得到任何一条长度正确的消息。

```
消息 1: 1001001 0100000 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110
Pad 1: 1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011
密文: 0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101

Pad 2: 1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110
明文 2: 1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011
```

图 8.4 一次一密加密方法的用法举例,换用其他一次性密钥来解密密文可以得到任何可能的明文

一次一密在理论上是非常有意义的,但是在实践中有许多缺点。首先,一次性密钥无法记忆,所以发送方和接收方必须随身携带书面的密钥副本。如果任何一方有可能被敌人捕获的话,则很显然,书面的密钥是一个很大的威胁。而且,可被传送的数据总量受到可用密钥数量的限制。如果一名间谍非常走运,发现了一批极有价值的数据,他可能由于密钥已被用尽而无法将这批数据传送回总部。另一个问题是,这种方法对于丢失字符或者插入字符非常敏感。如果发送方和接收方失去了同步的话,则从失去同步的点开始,后面所有的数据都变成了垃圾数据。

随着计算机的出现,一次一密方法对于某些应用可能会变得实用起来。密钥源可以是一片特殊的 DVD,它包含了几千兆字节的信息,如果它被放在 DVD 电影箱中运输,并且开头部分加上几分钟电影片断,则这样的 DVD 甚至不会招人怀疑。当然,在千兆位网络速度上,每隔 30 秒钟就必须插入一片新的 DVD 可能会非常烦人。由于在发送消息之前,必须首先通过单独的途径将 DVD 从发送方转运到接收方,因此,这极大地降低了一次一密方法的实际使用率。

### 量子密码学(quantum cryptography)

有趣的是,针对如何在网络上传输一次性密钥的问题,可能真的存在一个解决方案。这种方案来源于一种目前还不太可能的源:量子机。这个领域现在仍然是试验性的,但是初始的试验非常成功。如果它能够更加完美一些,而且效率又很好的话,那么,几乎所有的密码系统都可以利用一次一密方法来完成,因为一次一密方法可以被证明是绝对安全的。下面我们将简短地介绍一下这种量子密码系统(quantum cryptography)的工作原

理,尤其是,我们将描述一个被称为 **BB84** 的协议,该协议的名字来源于其作者的名字和发表年份(Bennet and Brassard, 1984)。

有一个用户(比如说 Alice)希望与第二个用户(比如说 Bob)建立一个一次性密钥,这里 Alice 和 Bob 被称为**安全个体(principal)**,他们是我们的故事中的主角。例如,Bob 是一个银行家,Alice 希望与他做一些商务交易。在过去 10 多年中,几乎所有关于密码学的论文和书籍在叙述的时候都使用“Alice”和“Bob”作为安全个体的名字。密码学家都喜欢传统,所以,如果我们使用“Andy”或者“Barbara”作为安全个体名字的话,就没有人会相信这一章的内容了。所以,我们还是沿用传统的做法。

如果 Alice 和 Bob 能够建立一个一次性密钥,那么,他们就可以用该密钥来安全地通信了。问题是:如果不像前面提到的那样交换 DVD 的话,那么,他们该如何建立一次性密钥呢?我们可以假定 Alice 和 Bob 在一根光纤的两端,通过这根光纤他们可以发送和接收光脉冲。然而,一个超级强大的入侵者,比如说她的名字叫 Trudy,能够切断光纤,并且用一个主动式分接头再将光纤连接起来。Trudy 可以读取两个方向上的所有数据,也可以在两个方向上发送假的消息。这种局面对于 Alice 和 Bob 来说,似乎已经没有希望进行安全通信了,但是,量子密码学却为他们带来了一线新的曙光。

量子密码学的物理基础是,光是以一种极小的光包(被称为光子, **photon**)的形式被传递的,并且光子具有某些特殊的属性。而且,光在通过一个偏振滤光器的时候可以被调整到一个方向上,戴太阳镜的人或者摄影师都知道这个事实。如果将一束光(即一个光子流)通过一个偏振滤光器,则该光束中的所有光子都将被偏到滤光器的轴向(比如垂直方向)上。如果现在光束再通过第二个偏振滤光器,则从第二个滤光器出来的光的强度将与两轴之间夹角的余弦平方成正比。如果这两个轴相互垂直的话,则所有的光子都通不过。两个滤光器的绝对方向并不重要,关键是它们之间的夹角。

为了产生一个一次性密钥,Alice 需要两组偏振滤光器,第 1 组滤光器是由一个垂直滤光器和一个水平滤光器组成的,这种选择被称为**直线基(rectilinear basis)**。这里的一个基只是一个坐标系统而已。第二组滤光器也一样,但是旋转 45 度,所以,一个滤光器的方向是从左下至右上,另一个滤光器的方向是从左上至右下。这种选择被称为**对角基(diagonal basis)**。因此,Alice 有两个基,她可以根据需要快速地将这些基插入到她的光束中。在实际的实现中,Alice 并没有 4 个独立的滤光器,而是一个晶体,该晶体的偏振方向可以通过电子方式以极快的速度切换到 4 个允许的方向之一。Bob 也有一套与 Alice 相同的设备。Alice 和 Bob 两个人都有两组可用的基,这对于量子密码系统是非常关键的。

对于每一组基,Alice 现在将一个方向分配为 0,另一个方向分配为 1。在下面介绍的例子中,我们假设她选择垂直方向为 0,水平方向为 1。另外,她也选择从左下至右上方向为 0,从左上至右下方向为 1。她通过明文方式将这些选择发送给 Bob。

现在 Alice 选择一个一次性密钥,比如说她利用一个随机数发生器(这本身就是一个非常复杂的主题)来生成该密钥。然后她逐位地将密钥传送给 Bob,在传送每一位的时候她随机地选择其中一个基。为了发送每一位,她的光子枪发射出来的光子已经被正确地偏振到她为这一位所选择使用的基上。例如,她可能选择对角基、直线基、直线基、对角



基、直线基等。为了用这些基来发送她的一次性密钥 1001110010100110, 她将会发送图 8.5(a) 中所示的光子。给定了一次性密钥和基的序列之后, 用于每一位的偏振方向也被惟一确定下来。像这样每次发送一个光子的数据位被称为量子位(qubit)。

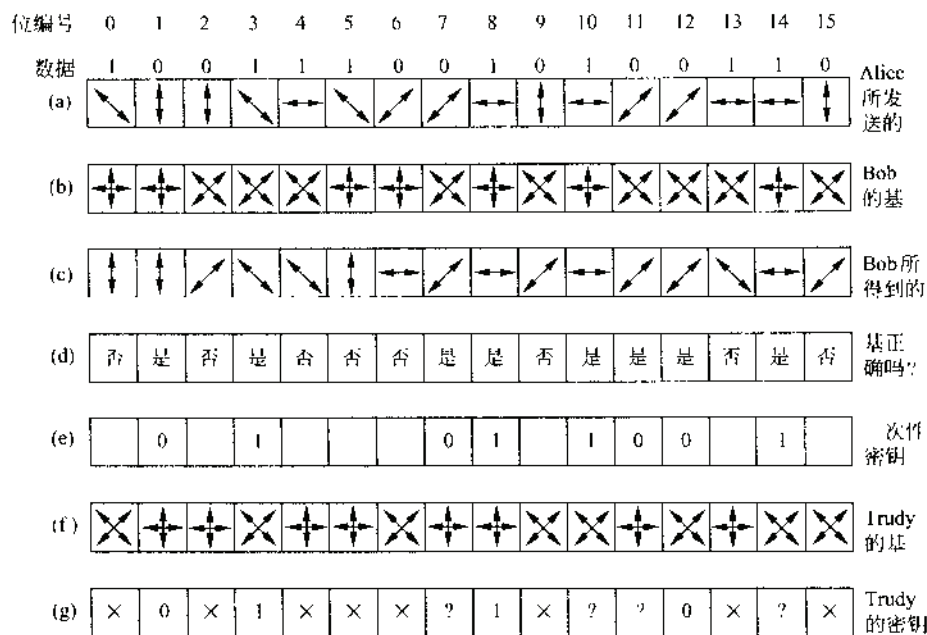


图 8.5 一个量子密码系统示例

Bob 并不知道 Alice 使用了哪些基, 所以他随机地为每一个到来的光子选择一个基, 并使用这个基, 如图 8.5(b) 所示。如果他选择了正确的基, 则他会得到正确的数据位。如果选择的基不正确, 则得到一个随机的位, 因为如果一个光子被发射到一个与它自己的偏振方向成 45 度角的滤光器上, 那么它将会随机地跳到滤光器的偏振方向上, 或者跳到与滤光器方向垂直的偏振方向上, 并且两者的概率相等。光子的这种性质正是量子机的基础。因此, 有些位是正确的, 而有些位是随机的, 但是 Bob 并不知道哪些是正确的、哪些是不正确的。Bob 得到的结果如图 8.5(c) 所示。

那么, Bob 如何知道他选择的哪些基是正确的、哪些基是不正确的呢? 他简单地告诉 Alice, 他为明文中的每一位使用了哪个基, 然后她告诉他, 明文中哪些是正确的, 哪些是不正确的, 如图 8.5(d) 所示。利用这些信息, 双方就可以根据正确的猜测结果获得一个位串, 如图 8.5(e) 所示。平均而言, 这个位串的长度将是原始位串的一半长度, 但是, 由于双方都知道这个位串了, 所以他们可以将该位串用作一次性密钥。Alice 所需要做的事情仅仅是传输一个略微超过期望长度之两倍的位串, 于是她和 Bob 就有了一个期望长度的一次性密钥。问题就这样解决了。

但是请等一等。我们忘了 Trudy 的存在了。假定她对于 Alice 所说的话非常好奇, 所以她割断光纤并插入了她自己的检测器和发射器。对她来说不幸的是, 她并不知道每个光子使用了哪一个基。她所能够做的事情是, 就像 Bob 一样, 随机地为每个光子选择



一个基。图 8.5(f)显示了她所选择的一个例子。当 Bob 后来用明文向 Alice 报告他使用了哪些基,并且 Alice 也用明文告诉他哪些基是正确的时候,Trudy 现在也知道她得到的哪些位是正确的,哪些位是错误的。在图 8.5 中,她知道以下位是正确的:0、1、2、3、4、6、8、12 和 13。但是,从图 8.5(d)的 Alice 应答中,她知道只有 1、3、7、8、10、11、12 和 14 位才是一次性密钥的组成部分。其中只有 4 位(即 1、3、8 和 12 位)她的猜测是正确的,并且也捕获到了正确的位信息。其他的 4 位(7、10、11 和 14 位)她猜测错误,所以她不知道真正传输的位是什么。因此,Bob 从图 8.5(e)中得知,一次性密钥的前 8 位为 01011001,但是,Trudy 仅仅得到了 01? 1?? 0?,如图 8.5(g)所示。

当然,Alice 和 Bob 知道 Trudy 可能已经捕获了他们的一次性密钥的一部分,所以他们希望进一步减少 Trudy 拥有的信息。他们只要对这个一次性密钥做一个变换就可以做到这一点。例如,他们可以将一次性密钥分成 1024 位的块,然后对每一块做平方操作,从而构成 2048 位数值,再将这些 2048 位数拼接起来当作一次性密钥。Trudy 只有一部分位串信息,所以她无法产生位串的平方,因此她什么也得不到。从原始的一次性密钥变换到另一个削弱了 Trudy 所知信息的一次性密钥的过程被称为**秘密放大**(**privacy amplification**)。在实践中,往往使用复杂的变换而不是简单的平方,而且在复杂的变换中,每一个输出的位依赖于所有的输入位。

可怜的 Trudy。她不仅对一次性密钥已经一无所知,而且她的存在也不再是一个秘密了。毕竟,她必须把接收到的每一位转发给 Bob 以便欺骗 Bob,让他以为自己是在跟 Alice 通话。麻烦之处在于,她只能使用她自己选择的偏振方向来传输接收到的每一个量子位,而她选择的偏振方向有一半是不正确的,从而导致 Bob 的一次性密钥中有很多错误。

当 Alice 最后开始发送数据的时候,她使用一种重量级的前向纠错码对数据进行编码。从 Bob 的角度来看,一次性密钥中的 1 位错误与 1 位传输错误是相同的。如果有足够的前向纠错能力的话,即使有这么多错误,他也能够恢复原始的消息,同时他可以很容易地计算出有多少错误被纠正过来。如果这个数值大大超过了设备的期望错误率的话,他就知道 Trudy 已经搭接在线路上了,于是可以根据情况作出响应(比如告诉 Alice 切换到一个无线电信道上,或者打电话给警察局,等等)。如果 Trudy 有办法复制光子,从而她可以监视其中一个光子,并且将另一个同样的光子发送给 Bob,那么,她就可以将自己隐藏起来,避免被 Bob 发现。但是,迄今为止人们尚未发现有理想的方法可以复制光子。不过,即使有一天 Trudy 能够复制光子了,量子密钥学用于建立一次性密钥的价值也不会被减弱。

尽管研究人员已经证明了在超过 60km 距离的光纤上可以运行量子密码系统了,但是装备非常复杂和昂贵。不过,量子密码学的思想仍然很有前途。有关量子密码学的更多信息,请参考(Mullins, 2002)。

### 8.1.5 两条基本的密码学原则

尽管在后文我们将要学习许多不同的密码系统,但是我们首先来学习这些密码系统背后的两条基本原则,这对于理解后面的内容是非常重要的。

## 冗余度

第一条原则是,所有被加密的消息必须包含一定的冗余信息,也就是说,这些信息对于理解这条消息来说是不必要的。通过一个例子或许能看清楚为什么需要冗余信息。请考虑一家名为 Couch Potato(TCP)的邮购公司,它有 60 000 多种产品。想象该公司的效率很高,TCP 的程序员决定,每条订购消息应该包含一个 16 字节的顾客名以及一个 3 字节的数据域(一个字节用于数量,另 2 个字节用于产品编号)。最后 3 个字节用一个非常长的密钥进行加密,并且只有顾客和 TCP 才知道该密钥。

初看起来这种方案似乎非常安全,从某种意义上来看它确实是安全的,因为被动的人侵者不可能解密消息。不幸的是,它有一个致命的缺陷将使得它毫无用处。假定有一个最近刚刚被解雇的雇员想要报复 TCP。在离开之前,她带走了顾客的名单。她连夜编写了一个程序来产生许多编造的订单,但是顾客名是真实的。由于她没有拿到顾客们的密钥,所以她在最后的 3 字节中只是放了一些随机数,然后给 TCP 发送了几百份订单。

当这些订购消息到来的时候,TCP 的计算机利用顾客的名字找到对应的密钥,并解密消息中的最后 3 个字节。对于 TCP 来说,不幸的是,几乎所有的 3 字节消息都是有效的,所以,计算机开始打印发货指令。顾客订购 837 套儿童秋千或者 540 个沙箱,这看起来非常奇怪,不过计算机仅仅知道这些信息而已,但这也并非不可能,比如顾客可能正在筹建一个获准特许经营的游乐场。一个主动的人侵者(离职雇员)即使并不理解她自己的计算机所产生的消息,她也可以通过这种方式造成大量的麻烦。

实际上,通过在所有的消息中增加一定的冗余度就可以解决这个问题。例如,如果订购消息被扩展到 12 个字节,前 9 个字节必须为 0,那么,这种攻击便不再奏效,因为离职雇员不可能再产生大量的有效消息。从这个例子得到的启示是:所有的消息必须包含足够大的冗余度,因而主动入侵者发送的随机垃圾消息不可能再被解释为有效的消息。

然而,增加冗余度也使得密码分析者更加容易破解消息。假设邮购业务的竞争非常激烈,Couch Potato 公司的主要竞争对手 Sofa Tuber 极力想知道 TCP 公司卖出了多少个沙箱。因此,他们搭接在 TCP 的电话线上。在原来的 3 字节消息的方案中,通过密码分析的手段几乎不可能成功,因为在试验了一个密钥之后,分析者无从判断他的猜测是否正确。毕竟,从技术上讲,几乎每一条消息都是合法的。但是,在采用了新的 12 字节的方案以后,分析者可以很容易地区分出有效的消息和无效的消息。因此,我们得到以下原则:

### 密码学原则 1: 消息必须包含一定的冗余度

换句话说,在解密了一条消息以后,接受者必须能够通过简单的检查手段以及可能执行一个简单的计算来判断该消息是否有效。这种冗余是必要的,它可以用来阻止主动入侵者发送垃圾信息,并欺骗接收者解密垃圾信息然后在所谓的“明文”上执行相应的处理。然而,同样这些冗余信息也使得被动入侵者更加容易破解系统,所以这里存在一个平衡的问题。而且,冗余信息不应该采用在消息的开头或者末尾加上  $n$  个 0 的形式,因为在某些密码算法上加解密这样的消息可以得到一些更有预测性的结果,从而使密码分析者的任务更加容易。CRC 多项式编码比加 0 的做法好得多,因为接收者很容易验证消息,但密

码分析者却需要更大的工作量。更好的做法是使用密码学中的散列算法,关于散列算法我们将在本章后面学习。

再回到稍早前学过的量子密码上,我们也可以看到冗余原则在那里扮演了很重要的角色。由于 Trudy 截获了光子,所以造成了 Bob 的一次性密钥中的有些位是错误的。Bob 需要一定的冗余度来判断进来的消息是否出现了错误。一种非常原始的冗余形式是将消息重复两遍。如果两份副本不相同的话,Bob 就知道要么该光纤有噪声,要么有人篡改了传输信息。当然,将所有信息都发送两份这种做法有点矫枉过正了,采用海明码或者 Reed-Solomon 编码可以更加有效地完成检错和纠错的工作。但是,很清楚的一点是,一定量的冗余度对于区分有效消息和无效消息是非常必要的,特别是在面对主动入侵者的情况下。

### 新鲜度

第二条密码学原则是,必须要采取某些措施来保证:每条接收到的消息能被验证是新鲜的,也就是说,它是最近刚被发送出来的。为了预防主动入侵者回放老的消息,这样的措施是必要的。如果不采取这样的措施,那么,我们的离职雇员就可以搭接在 TCP 的电话线上,然后不停地重复以前发送的有效消息。我们可以重新声明这个观点如下:

### 密码学原则 2: 需要采取某种方法来对抗重放攻击

一种措施是在每条消息中包含一个仅仅在一段时间(比如 10 秒钟)内有效的时间戳。然后,接收方只要保留该消息 10 秒钟,并且将新到达的消息与以前的消息进行比较就可以过滤掉重复的消息。10 秒钟之前到达的消息可以被丢弃了,因为凡是在 10 秒钟以后发送的任何重放消息将被当作太老的消息而拒绝。除了时间戳以外的其他措施将在本章后面讨论。

## 8.2 对称密钥算法

现代密码学使用了与传统密码学同样的思想(置换和转置),但是它的重点有所不同。在传统上,密码设计者使用了非常简单的算法。现在,情形完全相反:密码设计的目标是使加密算法尽可能地错综复杂,因而即使密码分析者获得了大量的选择密文,他在没有密钥的情况下也不能够推测出明文。

我们在本章中将要学习的第一类加密算法称为**对称密钥算法**(**symmetric-key algorithm**),因为它们使用同样的密钥来完成加密和解密操作。图 8.2 显示了对称密钥算法的用法。尤其是,我们将焦点集中在**块密码**(**block cipher**)<sup>①</sup>上,这种密码的特点是,它接受一个  $n$  位的明文块作为输入,并利用密钥将它变换成  $n$  位的密文块。

密码算法既可以用硬件来实现(为了速度),也可以用软件来实现(为了灵活)。虽然在本书中我们的主要注意力集中在算法和协议上,而算法和协议本身是独立于它们的具

<sup>①</sup> 译注:有些论文或者书籍将“block cipher”翻译成“分组密码”,本书中为了避免与网络层的“分组(packet)”混淆,因此翻译成“块密码”,它与流密码(stream cipher)相对应。

体实现的,但是,简短地提及一下有关构造密码系统硬件的事项也是饶有趣味的,置换和转置操作可以用简单的电子电路来实现。图 8.6(a)显示了一种被称为 **P 盒(P-box)** 的设备(这里的 P 代表数学中的排列[permutation]),P 盒的效果相当于对一个 8 位输入实现一个转置操作。如果这 8 位被从上到下指定为 01234567,则这个特定的 P 盒的输出为 36071245。通过适当的内部连线方式,P 盒可以执行任何一个转置,而且在实践中可以达到光速的速度,因为这中间没有涉及到任何计算过程,只是信号传播而已。这种设计符合 Kerckhoff 原则:攻击者知道一般化的方法是对输入的位重新排列顺序,但是他不知道哪些位将出现在哪里,这就是密钥。

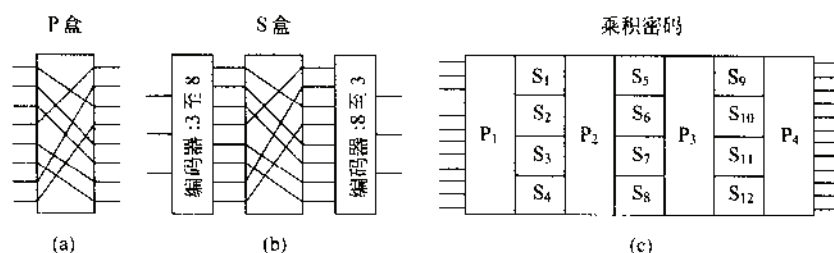


图 8.6 乘积密码的基本元素  
(a) P 盒; (b) S 盒; (c) 乘积

置换是由 **S 盒(S-box)**来完成的,如图 8.6(b)所示。在这个例子中,输入是 3 位明文,输出是 3 位密文。S 盒的执行过程分三个阶段。首先,3 位输入从第一阶段的 8 根输出线中选择其中一根线,并将它设置为 1,所有其他的线均为 0。第 2 阶段是一个 P 盒。第三阶段再将选中的输入线编码成二进制。按照图中显示的连线方式,如果依次输入 8 个八进制数 01234567 的话,则输出序列将是 24506713。换句话说,0 被 2 代替,1 被 4 代替,等等。同样地,通过正确地设置 S 盒内部的 P 盒,任何一种置换都可以用 S 盒来完成。而且,这样的设备可以直接用硬件来构造,从而获得极快的速度,因为编码器和解码器只有一个或者两个(纳秒量级以下)门延迟,而通过 P 盒的传播延迟可能小于  $10^{-12}$  秒。

只有当我们将一系列这样的 S 盒和 P 盒叠加起来构成乘积密码(product cipher)时,这些基本元素的威力才能真正发挥出来。在如图 8.6(c)所示的例子中,第一个步骤( $P_1$ )将 12 根输入线转置(即重新排列)。从理论上讲,将第二步换成一个针对 12 位数的 S 盒变换也是有可能的。然而,对于这样的 S 盒设备,其中间步骤将需要  $2^{12} = 4096$  根交叉线。相反,这里的做法是将 12 位输入分成 4 组,每组 3 位,再将每一组单独做一次置换。虽然这种方法缺乏通用性,但是它仍然非常强大。只要在乘积密码中包含足够多的中间步骤,则它的输入和输出将构成一个极为复杂的函数

对  $k$  位输入执行一个乘积密码算法以产生一个  $k$  位输出,这是非常常见的。通常情况下, $k$  位于 64 到 256 之间。在硬件实现中,通常至少有 8 个物理步骤,而不像图 8.6(c)那样只有 7 个步骤。而在软件实现中,乘积密码可以被编写成一个至少执行 8 次迭代的循环,每次迭代都在 64 至 256 位数据块的子块上执行 S 盒类型的置换操作,然后通过一个转置操作将这些 S 盒的输出全部混合起来。通常在起始处有一个特殊的初始转置操

作,在末尾也有一个转置操作。在许多文献中,这样的迭代称为轮(round)。

### 8.2.1 DES-数据加密标准

1977年1月,美国政府采纳了IBM开发的一个乘积密码作为无密级信息的官方加密标准。这个密码算法,即DES(Data Encryption Standard,数据加密标准),已被工业界广泛应用于安全产品中。它最初的形式已经不再安全了,但是它的一个修订形式仍然非常有用。现在我们来学习DES的工作原理。

DES的基本结构如图8.7(a)所示。明文按64位数据块的单元被加密,生成64位密文。DES算法带一个56位密钥作为参数,它共有19个步骤。第一步是一个与密钥无关的转置操作,它直接作用在64位明文上。最后一步正好是这个转置的逆操作。最后一步之前的那个步骤是交换左32位和右32位。剩下的16步在功能上是完全相同的,但使用了原始密钥的不同函数作为参数。DES算法的设计允许使用同样的密钥来完成解密过程,这正是任何一个对称密钥算法必须满足的一个特性。在DES算法中,解密的步骤只是加密步骤的相反顺序而已。

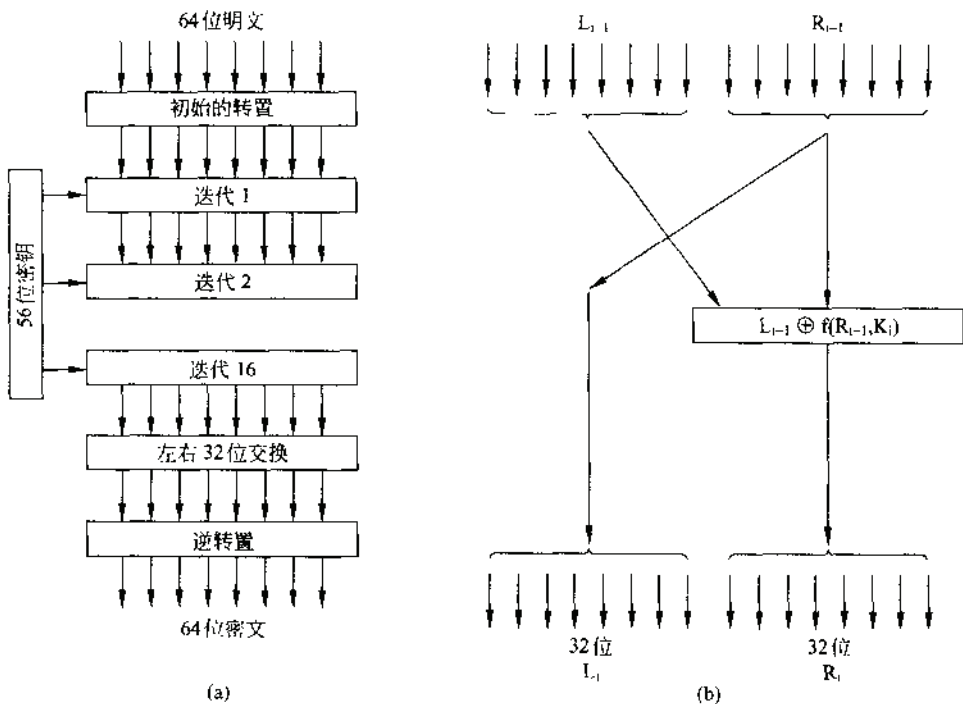


图 8.7 数据加密标准

(a) 一般性的结构; (b) 每次迭代的细节。带圆圈的表示异或

这些中间步骤的操作情况如图8.7(b)所示。每个步骤接受2个32位输入,并产生两个32位输出。左边的输出只是右边输入的一份副本而已。右边的输出是左边输入与一个函数值逐位异或的结果,该函数的输入参数是右边的输入和当前步骤的密钥  $K_i$ 。所



有的复杂性都在这个函数中。

此函数包含 4 个顺序执行的步骤。第一步,根据一个固定的转置和复制规则,将 32 位  $R_{i-1}$  扩展成一个 48 位的数字  $E$ 。第二步, $E$  和  $K_i$  被异或在一起。然后,异或的结果被分成 8 个 6 位组,每个 6 位组被输入到不同的 S 盒中。对于每个 S 盒,共有 64 种可能的输入,每种输入被映射到一个 4 位输出上。最后,将这  $8 \times 4$  位通过一个 P 盒。

这 16 次迭代中的每一次迭代使用了不同的密钥。在算法开始之前,先在 56 位的密钥上执行一个 56 位转置操作。在每一次迭代之前,密钥被分成两个 28 位单元,每个单元向左循环移位,移动的位数取决于当前的迭代号。移位以后再执行另一个 56 位转置即可导出  $K_i$ 。在每一轮上,从这 56 位中提取出不同的 48 位子集并对它进行排列。

有时候可以使用一项特殊的技术来加强 DES 的强度,这项技术称为白化(whitening)。这项技术的做法是,在将每一个明文数据块送给 DES 之前,先用一个随机的 64 位密钥对它执行异或操作,然后对 DES 输出的结果密文用第二个 64 位密钥执行异或,最后再输出异或的结果。去掉白化也非常容易,只要运行逆向的操作即可(前提条件是接收方也拥有这两个白化密钥)。由于这项技术的实际效果是增加了密钥的长度,所以,它使得用穷举法来搜索密钥空间需要消耗更多的时间。请注意,每个明文数据块使用同样的白化密钥(即这里总共只有一个白化密钥[2 个 64 位密钥合起来])。

从 DES 被发布的那一天起,有关 DES 的争议就一直不断。DES 建立在一个由 IBM 开发并拥有专利的加密算法的基础之上,此算法被称为 Lucifer,它使用了 128 位密钥,而不是 56 位密钥。当美国联邦政府希望为无密级的用途建立一个加密算法标准的时候,它“邀请”IBM 与 NSA(美国政府中专门破解密码的机构)“讨论”有关的事项。NSA 雇佣了一支世界上最强大的数学家和密码学家队伍。NSA 是如此之神秘,以至于在工业界流传着这样一个笑话:

问: NSA 代表什么?

答: 代表“没有这样的机构[No Such Agency]”。

实际上,NSA 代表了国家安全局(National Security Agency)。

在经过这些讨论之后,IBM 将密钥从 128 位缩短到 56 位,并且将 DES 的设计过程保密起来。许多人怀疑缩短密钥长度的原因是为了保证只有 NSA 能够破解 DES,而经费预算较少的其他组织却无法破解 DES。这种不公开设计过程的做法使人们猜测 DES 算法中隐藏了一个后门,NSA 利用这个后门可以很容易地破解 DES。当 NSA 的一个雇员非常谨慎地通知 IEEE 取消一个计划好的密码学会议时,这使得人们更加不放心了。NSA 否认这一切。

1977 年,斯坦福大学的两位密码学研究人员 Diffie 和 Helman(1977)设计了一台能破解 DES 的机器,估计它的造价需要两千万美元。给定一小段明文和匹配的密文,这台机器利用穷举法搜索整个密钥空间(共有  $2^{56}$  个密钥),可以在 1 天内找到密钥。现在,这样一台机器的造价已经大大低于一百万美元。

### 三重 DES

1979 年初,IBM 意识到 DES 的密钥长度太短,于是设计了一种方法,利用三重加密



来有效地增加密钥长度(Tuchman, 1979)。所选择的方法如图 8.8 所示,并且它后来已被集成到国际标准 8732 中。这种方法使用两个密钥,共包含三个步骤。第一步按照常规的方式用密钥  $K_1$  执行 DES 加密;在第二步中,DES 以解密方式运行,并使用  $K_2$  作为密钥。最后,再次用  $K_1$  执行 DES 加密。

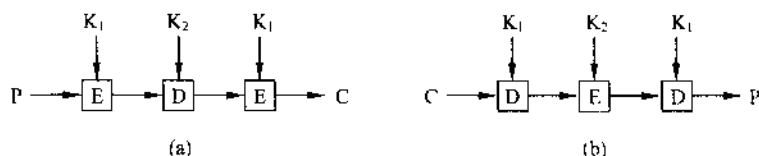


图 8.8

(a) 使用 DES 的三重加密; (b) 解密

这种设计立即引出了两个问题。第一,为什么只使用两个密钥,而不是三个?第二,为什么使用 EDE(加密 解密 加密)模式,而不是 EEE(加密 加密 加密)呢?使用两个密钥的理由是,即使最为多疑的密码学家也认为,就目前而言,112 位对于常规的商业应用来说已经足够了。(对于密码学家来说,多疑往往被认为是一种个性,而不是缺陷。)将密钥长度增加至 168 位只会带来管理和传输另一个密钥的不必要开销,同时又没有任何实际的益处。

采用“加密、解密,然后再加密”的理由是为了与现有的单密钥 DES 系统保持后向兼容性。DES 的加密和解密函数都是 64 位整数集之间的映射关系。从密码学的角度来看,这两个映射是同等强度的。然而,采用 EDE 而不是 EEE 之后,使用三重 DES 的计算机就可以与使用单密钥 DES 的计算机进行通话,做法很简单,只要设置  $K_1 = K_2$  即可。这种特性使得三重 DES 可以被逐步分阶段地应用到实际系统中,不过,学院派的密码学家对此并不感兴趣,但对于 IBM 和它的顾客来说却非常重要。

## 8.2.2 高级加密标准 AES

随着 DES 逐渐走到其生命的尽头,甚至三重 DES 也难以避免这样的厄运,NIST (National Institute of Standards and Technology, 美国标准和技术委员会;它是美国商务部的代理机构,承担着为美国联邦政府核定标准的重任)决定,美国政府需要一个新的密码标准作为无密级用途。NIST 对所有关于 DES 的争议都知道得非常清楚,它同时也知道,如果它直接宣布一个新的标准,则任何一个懂得丰富密码学知识的人很自然就会设想 NSA 已经在标准算法中内置了一个后门,因而 NSA 可以读取一切用该算法来加密的信息。在这样的情况下,可能没有人会使用此标准,从而它最终极有可能悄悄死去。

所以,NIST 采取了一种与政府官僚主义截然不同的做法:它发起了一场密码学比赛(竞赛)。1997 年 1 月,世界各地的研究人员接到邀请,希望为一个新的标准提交方案,这个新标准被称为 AES(Advanced Encryption Standard, 高级加密标准)。比赛的规则如下:

1. 它必须是一个对称的块密码算法。
2. 必须公开所有的设计。
3. 必须支持 128、192 和 256 位密钥长度。

4. 软件实现和硬件实现必须都是有可能的。
5. 算法必须是公有的,或者毫无歧视地授权给大众使用。

NIST 共收到了 15 个提案,于是它组织公开的会议来展示这些提案,并积极鼓励参加会议的人寻找这些提案中的漏洞。1998 年 8 月,NIST 根据这些算法的安全性、效率、简单性、灵活性和内存需求(对于嵌入式系统来说是非常重要的),选出 5 个算法参加最后的比赛。接下来又召开了更多的学术讨论会,同时也发生了更多的激烈论战。在最后一次会议上进行了无记名表决。参加最后决赛的算法以及它们的得分情况如下:

- (1) Rijndael(提出者为 Joan Daemen 和 Vincent Rijmen,86 票)。
- (2) Serpent(提出者为 Ross Anderson、Eli Biham 和 Lars Knudsen,59 票)。
- (3) Twofish(提出者为 Bruce Schneier 所带领的一个小组,31 票)。
- (4) RC6(提出者为 RSA Laboratories[RSA 实验室],23 票)。
- (5) MARS(提出者为 IBM,13 票)。

2000 年 10 月,NIST 宣布 Rijndael 胜出;2001 年 11 月,Rijndael 变成美国政府标准,并作为联邦信息处理标准 FIPS 197 被正式发表。由于整个竞争过程中绝对的开放性、Rijndael 的技术特性,以及“获胜的小组是由两位年轻的比利时密码学家组成的”这样毋庸置疑的事实,所以,可望在未来的至少 10 年以内,Rijndael 将变成世界上最重要的密码标准。名字 Rijndael 是由两位作者的姓氏(Rijmen+Daemen)合成的,其发音为 Rhine-doll(差不多是这样)。

Rijndael 假定密钥长度和块的长度可以从 128 位一直到 256 位(以 32 位为间隔逐步向上递增)。密钥长度和块长度可以单独选择,它们没有必然的关系。然而,AES 规定,块长度必须是 128 位,密钥长度必须是 128、192 或者 256 位。尽管如此,几乎不太可能会有人使用 192 位密钥,所以,事实上,AES 有两个变种:(1) 数据块为 128 位,密钥为 128 位;(2) 数据块为 128 位,密钥为 256 位。

在下面的算法介绍中,我们仅仅讨论 128/128 的情形,因为它有可能会变成商业规范。在 128 位的密钥空间中有  $2^{128} \approx 3 \times 10^{38}$  个密钥。即使 NSA 想办法建造一台内含 10 亿个并行处理器的机器,并且每个处理器每秒钟可以计算  $10^{12}$  个密钥,这样的机器也需要  $10^{10}$  年才能搜索完成整个密钥空间。到那时候太阳的能量已经耗尽了,所以,那时候的人们只好点着蜡烛阅读解密结果了。

### Rijndael

从数学的角度而言,Rijndael 是以 Galois(伽罗瓦)域理论为基础的,Galois 域理论赋予 Rijndael 一些可以证明的安全特性。然而,我们也可以用 C 代码的形式来看待 Rijndael 算法,而不涉及数学理论。

如同 DES 一样,Rijndael 也使用置换和转置操作,并且它也使用了多轮的策略。具体的轮数取决于密钥的长度和块的长度,对于 128 位密钥和 128 位块长度的情形,轮数为 10;对于更长的密钥或者更长的数据块,轮数可以增加至 14。然而,与 DES 不同的是,所有的操作都牵涉到整个字节,从而允许用硬件和软件高效地实现这些操作。Rijndael 的代码结构如图 8.9 所示。

```

#define LENGTH 16                /* # bytes in data block or key */
#define NROWS 4                  /* number of rows in state */
#define NCOLS 4                  /* number of columns in state */
#define ROUNDS 10                /* number of iterations */
typedef unsigned char byte;      /* unsigned 8-bit integer */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
    int r;                        /* loop index */
    byte state[NROWS][NCOLS];    /* current state */
    struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1]; /* round keys */

    expand_key(key, rk);          /* construct the round keys */
    copy_plaintext_to_state(state, plaintext); /* init current state */
    xor_roundkey_into_state(state, rk[0]); /* XOR key into state */

    for (r = 1; r <= ROUNDS; r++) {
        substitute(state);        /* apply S-box to each byte */
        rotate_rows(state);       /* rotate row i by i bytes */
        if (r < ROUNDS) mix_columns(state); /* mix function */
        xor_roundkey_into_state(state, rk[r]); /* XOR key into state */
    }
    copy_state_to_ciphertext(ciphertext, state); /* return result */
}

```

图 8.9 Rijndael 的代码结构

函数 `rijndael` 有三个参数。它们是：`plaintext` 是一个 16 字节的数组，其中包含了输入数据；`ciphertext` 也是一个 16 字节的数组，它包含的是加密之后返回的输出结果；`key` 是 16 字节的密钥。在计算过程中，数据的当前状态被保存在一个字节数组 `state` 中，该数组的长度为  $NROWS \times NCOLS$ 。对于 128 位数据块，该数组为  $4 \times 4$  字节。利用这 16 字节可以存储整个 128 位数据块。

`state` 数组首先被初始化为明文数据，然后在计算过程中每一步都要被修改。在有些步骤中，要执行字节对字节的置换；在其他的步骤中，这些字节被执行数组内的转置操作（即在数组内部重新排列顺序）。Rijndael 还用到了其他的变换。在算法的最后，`state` 中的内容被作为密文返回。

在代码的开始处，首先将密钥扩展到 11 个与 `state` 同样长度的数组中。它们被保存在 `rk` 中，这里的 `rk` 是一个结构数组，其中每个结构元素包含一个状态数组。在这 11 个数组中，有一个被用在计算过程的开始处，其他 10 个被用在 10 轮计算中，每轮一个数组。从加密密钥（即原始密钥）导出每一个轮密钥的过程非常复杂，因此我们这里不讨论这个过程。可以简单地这样来描述，轮密钥是通过反复地对密钥中不同的位组进行循环移位和异或而生成的。关于这些细节，请参考 Daemen and Rijmen, 2002。

接下来的步骤是将明文复制到 `state` 数组中，从而在后续的轮循环中得以处理。复制过程按照列的顺序进行，即前 4 个字节复制到第 0 列中，接下来 4 个字节复制到第 1 列中，以此类推。列和行都从 0 开始编号，但是轮数从 1 开始编号。图 8.10 显示了 12 个  $4 \times 4$  大小的数组的初始化过程。

在主要计算过程开始之前还有一个步骤：`rk[0]` 被逐个字节地异或 (XOR) 到 `state`

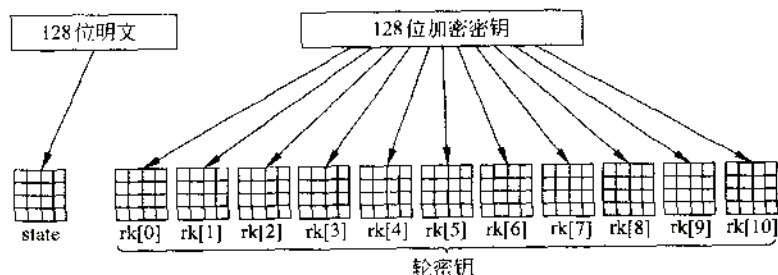


图 8.10 state 和 rk 数组

中。换句话说,在 state 数组的 16 个字节中,每个字节都被它与 rk[0]中对应的字节异或(XOR)之后的结果所取代。

现在该进入到主循环中了。该循环共执行 10 次迭代,每轮一次,并且在每一次迭代中都要变换 state。每一轮中的内容由 4 个步骤组成。第 1 步在 state 上执行一个逐字节的置换。按照顺序,每个字节被当作一个索引引用到一个 S 盒中,然后以该 S 盒中对应项的内容来替换该字节的值。这一步是一个直接的单字母表置换密码。AES 与 DES 不同,DES 有多个 S 盒,但 Rijndael 只有一个 S 盒。

在第 2 步中,4 行中的每一行向左循环移动。第 0 行移动 0 字节(即不改变);第 1 行左移 1 个字节;第 2 行左移 2 个字节;第 3 行左移 3 个字节。这一步起到的作用是将当前的数据内容在整个块中混合起来,其效果类似于图 8.6 中的转置操作。

第 3 步独立地混合每一列,列跟列之间并不相干。这个混合操作是通过矩阵乘法来完成的,在矩阵乘法中,新的列是老的列与一个常量矩阵的乘积,这里的乘法是有限 Galois 域  $GF(2^8)$  上的乘法。虽然这听起来好像很复杂,但实际上存在一个算法可以用两个查表操作和三个 XOR 操作来计算新列的每个元素(Daemen and Rijndael, 2002, Appendix E)。

最后,第 4 步将这一轮的密钥异或(XOR)到 state 数组中。

由于每一步都是可逆的,所以解密过程也很简单,只要反过来运行这个算法的每一步即可。然而,通过使用技巧,也可以用该加密算法来完成解密,只是用到的数据表不相同。

该算法的设计不仅为了极高的安全性,同时也为了极快的速度。在 2GHz 的机器上,一个良好的软件实现应该可以达到 700Mbps 的加密速率,这个速度足以实时地加密 100 部 MPEG 2 录像。硬件实现应该更快。

### 8.2.3 密码算法的使用模式

尽管 AES 算法(或者 DES,或者其他任何块密码算法)如此复杂,但归根到底,它们本质上只是一个单字母表置换密码算法,只不过使用了极大的字符(对于 AES 为 128 位字符,对于 DES 为 64 位字符)而已。任何时候当同样的明文块进入到算法前端时,同样的密文块就从后端输出。如果你用同样的 DES 密钥加密明文 abcdefgh 100 次,那么你就会得到同样的密文 100 次。入侵者可以充分发掘这种特性来协助攻破整个密码系统。

### 电子代码簿模式

为了看清楚如何利用这种“单字母表置换密码算法”的特性来局部地破解密码算法,我们将使用(三重)DES作为例子,因为描述64位数据块比128位数据块更加容易,但是AES也有同样的问题。为了使用DES来加密一段明文,最简单的做法是将其分割成连续的8字节(64位)数据块,然后用同样的密钥逐个加密这些数据块。如果有必要的话,将最后一块明文填充至64位。这项技术称为ECB模式(Electronic Code Book Mode,电子代码簿模式),因为它与老式的代码簿非常类似(在老式的代码簿中,每个明文单词的旁边列出了它的密文,所谓密文往往是5个十进制数字)。

图8.11是一个计算机文件的开始部分,该文件列出了某一公司已确定的奖励给雇员的年终奖金。文件由连续的32字节记录构成,每个雇员一条记录,正如图中所显示的,其格式为:16字节的名字,8字节的职位和8字节的奖金。现在利用(三重)DES将图中显示的16个8字节块(从0到15编号)进行加密。

名字																职位								奖金															
A	d	a	m	s	i	l	e	s	i	i	e	i	i	i	i	C	i	e	r	k	i			9	7	7	7	7	7	7	7								
B	i	l	a	s	k	i	r	d	a	n	i					B	i	s	s	i				5	5	0	0	0	0	0	0								
C	h	i	i	i	n	s		K	i	m					M	a	n	a	s	c	r		5	1	0	0	0	0	0	0									
D	a	v	i	s	i	r	B	a	b	i	c	i			J	a	n	i	t	v	i		5	1	1	1	1	1	1	1	1								
字节																16								8								9							

图 8.11 一个文件的明文被当作 16 个 DES 块进行加密

Leslie 刚刚与老板有过冲突,他估计自己不会得到很多奖金。相反,Kim 则深得老板的喜爱,而且每个人都知道这一点。在这个文件被加密之后,但是在它被送给银行之前,Leslie 有机会访问该文件。请问,在仅仅拿到加密文件的情况下,Leslie 能扭转这种不公平的局面吗?

没有问题。Leslie 所需要做的事情是,将第 12 个密文块(这是 Kim 的奖金)做一份副本,然后用它替换掉第 4 个密文块(这是 Leslie 的奖金)。虽然 Leslie 并不知道第 12 个密文块到底包含了什么内容,但是他可以期望今年有一个愉快的圣诞节(复制第 8 个密文块也是可以的,但是极有可能被检测出来;另外,Leslie 并不是一个很贪婪的人)。

### 密码块链接模式

为了对抗这种类型的攻击,所有的块密码算法可以按各种不同的方式链接起来,所以,Leslie 所做的密文块替换方法将导致在解密之后,从被替换的块开始的明文都是垃圾数据。一种链接方法是密码块链接(cipher block chaining)。在这种方法中,如图 8.12 所示,每个明文块在被加密之前,先与上一个密文块执行一次异或操作。因此,同样的明文块不再被映射到同样的密文块上,加密操作也不再是一个大的单字母表置换密码了。第一个块与一个随机选取的 IV(Initialization vector,初始向量)执行异或,并且该 IV(以明

文方式)随着密文一起被传输。

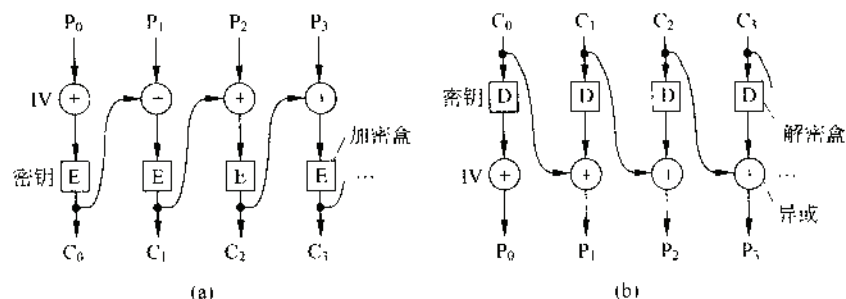


图 8.12 密码块链接模式  
(a) 加密; (b) 解密

通过查看图 8.12 中的例子我们可以看出密码块链接模式的工作原理。首先我们计算  $C_0 = E(P_0 \text{ XOR } IV)$ , 然后计算  $C_1 = E(P_1 \text{ XOR } C_0)$ , 以此类推。解密过程也使用 XOR 来反转整个过程, 有  $P_0 = IV \text{ XOR } D(C_0)$ , 以此类推。请注意, 第  $i$  块的加密操作是从 0 到  $i-1$  块所有明文的一个函数, 所以同样的明文根据它所出现的位置的不同而生成不同的密文。像 Leslie 所做的变换将导致从 Leslie 的奖金域开始的两个块变得没有任何意义。对于精明的安全检查官来说, 这种特点可能正好暗示了应该从哪里开始进行深入的调查。

密码块链接模式还有一个好处, 即同样的明文块并不导致同样的密文块, 这使得密码分析更加困难。实际上, 这也正是它被广泛采用的原因。

### 密码反馈模式

然而, 密码块链接模式也有一个缺点, 即只有当整个 64 位数据块到达以后才可以开始解密。对于交互式终端应用来说, 用户可能输入少于 8 个字符的命令, 然后停下来等待响应, 所以, 很显然这种密码块链接模式并不适合。对于逐字节的加密应用, 可以使用如图 8.13 所示的密码反馈模式 (cipher feedback mode), 它使用了 (三重) DES。要想使用 AES, 基本思想也完全相同, 只是使用 128 位移位寄存器而已。在这个图中, 加密机正好处于“字节 0 至 9 已经被加密并发送出去”之后的状态。当明文字节 10 到来时, 如图 8.13(a) 所示, DES 算法被作用在 64 位移位寄存器上, 并产生一个 64 位密文。该密文最左边的字节被提取出来, 并且与  $P_{10}$  进行异或。然后该字节被传送到输出线路上。而且, 移位寄存器左移 8 位, 使得  $C_2$  被从最左边移走, 而  $C_{10}$  被插入到  $C_9$  右边空出来的位置中。请注意, 移位寄存器的内容依赖于明文整个以前的历史, 所以, 若一个模式在明文中重复多次, 则经过加密以后, 在密文中每次都不相同。如同密码块链接模式一样, 密码反馈模式也需要一个初始向量来启动整个加密过程。

密码反馈模式的解密过程如同加密过程一样也做同样的事情。特别是, 移位寄存器的内容也是被加密, 而不是解密, 所以, 为了生成  $P_{10}$  而被选出来与  $C_{10}$  做异或的那个字节, 与在加密过程中为了生成  $C_{10}$  而与  $P_{10}$  做异或的字节是同一个字节。尽管这两个移位



寄存器保持不变,但是解密过程却是正确的。图 8.13(b)显示了这一过程。

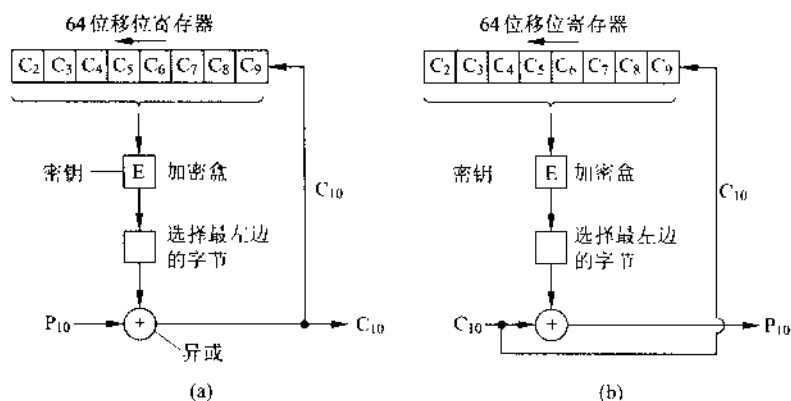


图 8.13 密码反馈模式

(a) 加密; (b) 解密

密码反馈模式的一个问题是,即使密文在传输过程中只有一位偶然发生翻转,则在解密过程中,当坏字节位于移位寄存器中所波及到的 8 个字节都将遭到破坏。一旦坏字节移出了移位寄存器以后,后面产生的明文仍然是正确的。因此,单个位翻转错误只影响相对局部的区域,而不会破坏消息中所有剩下的部分,但是其影响的位数与移位寄存器的宽度相等。

### 流密码模式

然而,1 位传输错误要弄乱 64 位明文,这种影响对于有些应用来说还是太大了。对于它们来说,还可以选择第四种方案,即流密码模式(stream cipher mode)。它的工作方式是用一个密钥来加密一个初始向量以生成一个输出块。然后用同样的密钥对这个输出块进行加密以得到第二个输出块。再对这一块进行加密以生成第三块,以此类推。输出块的序列(可以任意长)被称为密钥流,它就好像被当作一次性密钥那样与明文做异或(XOR)来生成密文,如图 8.14(a)所示。请注意,IV 仅仅被用在第一步中;在此以后,输出数据块被加密。同时也要注意,密钥流与数据是独立的,所以,如果有必要的话,可以将密钥流提前计算出来,而且,它对于传输错误完全不敏感。解密过程如图 8.14(b)所示。

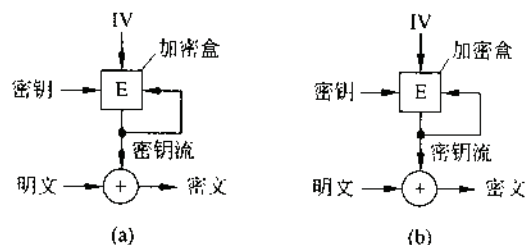


图 8.14 一个流密码

(a) 加密; (b) 解密

为了执行解密,在接收端的机器上也要生成同样的密钥流。由于密钥流仅仅依赖于 IV 和密钥,所以,它不会受到密文中传输错误的影响。因此,密文中的 1 位传输错误仅仅导致被解密出来的明文产生同样的 1 位错误。

很关键的一点是,对于一个流密码,永远不要两次使用同样的(密钥,IV)对,因为一旦这样做以后,则每次都会生成同样的密钥流。两次使用同样的密钥流将会导致密文受到**密钥流重用攻击(keystream reuse attack)**。假定明文块  $P_0$  经过一个密钥流加密之后得到  $P_0 \text{ XOR } K_0$ 。后来,第二个明文块  $Q_0$  也用同样的密钥流加密之后得到  $Q_0 \text{ XOR } K_0$ 。一个人侵者捕捉到这两个密文块以后,他只需将它们异或一下就可以得到  $P_0 \text{ XOR } Q_0$ ,从而消掉了密钥。现在入侵者有了两个明文块的 XOR 结果。如果其中一个明文被入侵者知道或者猜中的话,另一个明文自然也暴露无遗。无论如何,利用消息的统计特性,就更有可能会对两个明文流的 XOR 结果实施破解。例如,对于英语文本,在流中最常见的特征可能是两个空格的 XOR,其次是空格和字母“e”的 XOR,等等。简而言之,有了两个明文的 XOR 信息以后,密码分析者就有更多的机会来推断出这两个明文。

#### 计数器模式

除了电子代码簿模式以外其他所有模式都存在这样一个问题:要想随机访问加密之后的数据是不可能的。例如,假设通过网络传输一个文件,然后以密文的形式将文件存储在磁盘上。如果接收端计算机是一台有可能被偷走的笔记本电脑,那么这可能是一种非常合理的工作方式。使所有关键的文件都以加密的形式来保存可以极大地减小潜在的危险性,因为即使在计算机落入敌手之后秘密信息也不易泄漏出去。

然而,磁盘文件通常是以非顺序的方式来访问的,特别是数据库中的文件。如果用密码块链接模式来加密一个文件,则为了访问一个随机的块,要求解密所有在它之前的数据块,所以这是一个很昂贵的操作。由于这个原因,后来又发明了另一种模式,即**计数器模式(counter mode)**,如图 8.15 所示。在这里,明文并不直接被加密,相反,被加密的是初始向量加上一个常量的值,结果得到的密文再与明文做异或。通过“为每个新的数据块使初始向量递增 1”这种办法,在文件中任何地方的块都可以直接解密,而无需解密所有在它之前的数据块。

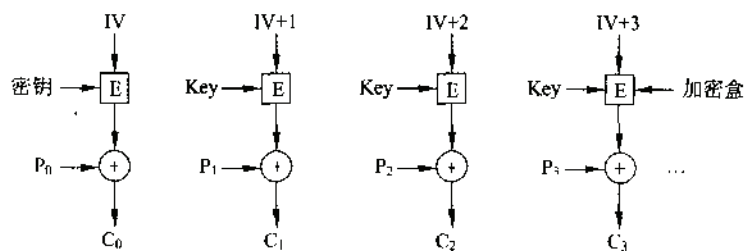


图 8.15 使用计数器模式的加密过程

尽管计数器模式非常有用,但是它的一个弱点也值得在这里指出来。假设同样的密钥  $K$  在将来又被使用了(明文不同,但 IV 相同),并且攻击者获得了这两次运行中所有的

密文。这两种情形下的密钥流相同,从而使密码算法遭受密钥流重用攻击,我们在讨论流密码模式时曾经提到过这种类型的攻击。密码分析者只要将两个密文 XOR 在一起,就可以消除掉所有的密码保护,并获得明文的 XOR 结果。这个弱点并不意味着计数器模式的想法不好,而仅仅意味着密钥和初始向量应该独立地随机选取。即使同样的密钥偶尔被用到了两次,如果这两次的 IV 不相同的话,则明文仍然是安全的。

### 8.2.4 其他密码算法

DES 和 Rijndael 是最著名的对称密钥算法。然而,其他一些设计良好的对称密钥密码算法也值得在这里提一下。其中有些算法已经被嵌入到各种产品之中。图 8.16 列出了最为常见的一些对称密钥密码算法。

密码算法	作者	密钥长度	说明
Blowfish	Bruce Schneier	1-448 位	又老又慢
DES	IBM	56 位	对于现在使用,太弱了
IDEA	Massey 和 Xuejia	128 位	好,但是属于专利算法
RC4	Ronald Rivest	1-2048 位	小心:有一些弱密钥
RC5	Ronald Rivest	128-256 位	好,但是属于专利算法
Rijndael	Daemen 和 Rijmen	128-256 位	最佳的选择
Serpent	Anderson, Biham, Knudsen	128-256 位	很强
三重 DES	IBM	168 位	第二最佳选择
Twofish	Bruce Schneier	128-256 位	很强、被广泛使用

图 8.16 一些常见的对称密钥密码算法

### 8.2.5 密码分析

在结束关于对称密码学主题的讨论之前,起码应该提及一下在密码分析领域中的 4 个进展。第一个进展是差分密码分析(differential cryptanalysis)(Biham and Shamir, 1993)。这项技术可以被用来攻击任何一种块密码算法。它的原理是,首先用一对仅在少量的位上有差异的明文块来执行加密,然后仔细地观察在加密过程中内部每一次迭代时所发生的情况。在许多情况下,有些位模式可能比其他的位模式更经常出现,这种现象可以导致概率攻击。

第二个值得提及的密码分析进展是线性密码分析(linear cryptanalysis)(Matsui, 1994)。它只需用  $2^{41}$  个已知明文就可以破解 DES。它的工作原理是,将明文和密文中特定的位 XOR 在一起,然后检查各种模式的结果。当重复这样的过程时,应该一半的位为 0,一半的位为 1。然而,通常情况下密码算法会在某一个方向上引入偏差,不管这个偏差有多小,它总可以用来降低工作量。有关细节过程,请参考 Matsui 的论文。

第三个进展是通过对电子功率消耗的分析来找到秘密密钥。计算机通常使用 3 伏来

代表“1”位,用 0 伏来代表“0”位。因此,处理“1”比处理“0”需要更多的电能。如果一个密码算法是由一个循环来完成的,并且这个循环按照顺序逐位地处理密钥中的位,那么,攻击者用一个慢的时钟(比如 100Hz)来代替 nGHz 的主时钟,并且用弹簧夹夹住 CPU 的电源和地线就可以精确地监视每条机器指令所消耗的功率。利用这些数据来推断出密钥竟然出奇地容易。这种密码分析方法很容易被挫败,做法很简单,只要用汇编语言小心地编写算法,以确保功率消耗与密钥独立,也与每个轮密钥独立。

第四个进展是时间分析。密码算法往往充满了对轮密钥中的位进行测试的 if 语句。如果 then 和 else 部分需要执行不同长度的话,则只要减慢时钟,并且观察每一步所花的时间,就有可能推断出轮密钥。一旦所有的轮密钥都知道以后,原始的密钥很容易可被计算出来。功率分析和时间分析也可以同时使用,从而使破解工作更加容易。虽然功率和时间分析看起来好像是外来的(即不针对算法本身的特点),实际上它们的威力非常强大,足以破解任何一个未经专门设计来对抗它们的密码算法。

### 8.3 公开密钥算法

在历史上,分发密钥往往是绝大多数密码系统中最薄弱的环节。不管一个密码系统有多强,如果入侵者能够偷取到密钥,则整个系统就毫无价值了。密码学家总是认为加密密钥和解密密钥是一样的(或者很容易从一个推导出另一个)。但是,这个密钥必须被分发给一个系统的所有用户。因此,看起来这里存在一个本身固有的问题:密钥必须被保护起来,以防被偷,但是它们又必须被分发出去,所以,它们不可能仅仅被锁在银行的保险柜中。

1976 年,斯坦福大学的两位研究人员 Diffie 和 Hellman(1976)提出了一种全新的密码系统,在这种密码系统中,加密密钥和解密密钥并不相同,而且不可能很轻易地从加密密钥推导出解密密钥。在他们的提案中,(受密钥控制的)加密算法 E 和解密算法 D 必须满足三个要求。这三个要求可以简述如下:

- (1)  $D(E(P)) = P$ 。
- (2) 从 E 推断出 D 极其困难。
- (3) 用选择明文攻击不可能破解 E。

第一个要求是说,如果我们将 D 作用在一条被加密的消息  $E(P)$  上,则可以恢复原来的明文消息 P。如果没有这个特性的话,则合法的接收者就无法解密密文了。第二个要求不言自明。第三个要求是必要的,因为正如一会儿我们将会看到的那样,入侵者有可能用此算法对他们的核心内容进行试验。具备了这些条件以后,加密密钥就有理由可以被公开了。

这种方法的工作过程如下所述。有一个人,比如说 Alice,希望接收秘密的消息,她首先设计了两个满足以上要求的算法。然后,加密算法和 Alice 的密钥都被公开,因此该系统被命名为公开密钥密码系统(public-key cryptography)。例如,Alice 可以把公开密钥放在她的 Web 主页上。我们将使用标记  $E_A$  来表示以 Alice 的公开密钥作为参数的加密算法。类似地,以 Alice 的私有密钥作为参数的(秘密)解密算法被记为  $D_A$ 。Bob 也做同样

的事情,即公开  $E_B$ ,但是  $D_B$  保密。

现在,我们来看是否能够解决在 Alice 和 Bob(以前两者从来没有联系过)之间建立一个安全通道的问题。假设 Alice 的加密密钥  $E_A$  和 Bob 的加密密钥  $E_B$  位于某些公开可读的文件中。现在, Alice 取出她的第一个消息  $P$ ,并计算  $E_B(P)$ ,然后将它发送给 Bob。Bob 接收到消息以后,利用他的秘密密钥  $D_B$  对消息进行解密[即计算  $D_B(E_B(P)) = P$ ]。其他没有人能够读取已被加密的消息  $E_B(P)$ ,因为假设加密系统  $E_B$  足够强,并且从公开已知的  $E_B$  来推导  $D_B$  是极其困难的。为了发送一个应答  $R$ , Bob 传输  $E_A(R)$ 。现在 Alice 和 Bob 就可以安全地进行通信了。

这里提一下关于术语的说明。公开密钥密码系统要求每个用户拥有两个密钥:一个公开密钥(简称公钥)和一个私有密钥(简称私钥)。当其他人给某个用户发送加密消息的时候,他们使用该用户的公钥;当这个用户需要解密消息的时候,他(或她)使用自己的私钥。我们在下文中将统一地把这两个密钥分别称为公钥和私钥,从而与传统的对称密钥密码系统中用到的秘密密钥区分开。

### 8.3.1 RSA

现在惟一的问题是,我们需要找到满足上述所有三个要求的算法。由于公开密钥密码系统的潜在优势,许多研究人员不遗余力地工作,陆续发表了一些满足要求的算法。一种比较好的方法是由 MIT 的一个小组发现的(Rivest et al., 1978),该方法的名字来源于三个发现者名字(Rivest、Shamir、Adleman)的首字母:RSA。尽管四分之一多个世纪以来,有大量的工作企图要破解该方法,但是它都抵抗住了,所以它被认为是一个非常强的公开密钥算法。在实践中有大量的安全性都建立在它的基础之上。它的主要缺点是,要想达到好的安全性,它要求至少 1024 位长度(相比之下,对称密钥算法只需 128 位),这也使得它的速度非常慢。

RSA 方法基于数论中的一些原理。我们现在只是简要地说明如何使用该方法,有关详细的信息请参考 Rivest 等人的论文。

1. 选择两个大的素数  $p$  和  $q$ (典型情况下为 1024 位)。
2. 计算  $n=p \times q$  和  $z=(p-1) \times (q-1)$ 。
3. 选择一个与  $z$  互素的数,将它称为  $d$ 。
4. 找到  $e$ ,使其满足  $e \times d = 1 \bmod z$ 。

提前计算出这些参数以后,我们就可以开始执行加密了。首先将明文(可以看作一个位串)分成块,使得每个明文消息  $P$  落在间隔  $0 \times P < n$  中。为了做到这一点,只要将明文划分成  $k$  位的块即可,这里  $k$  是满足  $2^k < n$  的最大整数。

为了加密一个消息  $P$ ,只要计算  $C = P^e \pmod n$  即可。为了解密  $C$ ,只要计算  $P = C^d \pmod n$  即可。可以证明,对于指定范围内的所有  $P$ ,加密和解密函数互为反函数。为了执行加密,你需要  $e$  和  $n$ ;为了执行解密,你需要  $d$  和  $n$ 。因此,公钥是由  $(e, n)$  对组成的,而私钥是由  $(d, n)$  对组成的。

这种方法的安全性建立在分解大数的困难度基础之上。如果密码分析者能够分解(公开已知的) $n$ ,那么他就可以找到  $p$  和  $q$ ,从而得到  $z$ 。有了  $z$  和  $e$  之后,只要使用欧几

里德算法就可以找到  $d$ 。幸运的是,数学家们探索大数分解法至少有 300 多年了,几百年积累起来的经验表明这确实是一个极其困难的问题。

根据 Rivest 和同事们的研究工作,若使用穷举的办法,则分解一个 500 位十进制数要求  $10^{25}$  年时间。在计算中,他们假设使用了已知的最佳算法,并且计算机的指令时间为  $1 \times s$ 。即使计算机的速度继续加快,以每 10 年一个数量级的速度增长,则仍然需要几个世纪的时间才能使分解一个 500 位十进制数变得实际可行,到那个时候,我们的后代们只要选择更大的  $p$  和  $q$  就可以了。

图 8.17 给出了一个简单的教学示范例子,它说明了 RSA 算法是如何工作的。在这个例子中,我们选择了  $p=3$  和  $q=11$ ,因此  $n=33$  和  $z=20$ 。由于 7 和 20 没有公因子,所以  $d$  值取  $d=7$  是合适的。有了这些选择以后,通过解方程  $7e=1 \pmod{20}$  可以找到  $e$ ,结果是  $e=3$ 。针对明文消息  $P$  的密文  $C$  可以通过  $C=P^3 \pmod{33}$  来求得。接收方在解密密文的时候,只要利用  $P=C^7 \pmod{33}$  的规则即可。图中的例子显示了明文“SUZANNE”的加密过程。

明文 (P)		密文 (C)			解密之后	
符号	数值	$P^3$	$P^3 \pmod{33}$	$C^7$	$C^7 \pmod{33}$	符号
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E

发送方的计算
接收方的计算

图 8.17 RSA 算法的一个例子

因为这个例子中选择的素数非常小,所以  $P$  必须小于 33,因此,每个明文块只能包含一个字符。其结果是一个单字母表置换,这让人觉得该方法并无稀奇之处。然而,如果我们选择  $p$  和  $q \times 2^{512}$ ,则我们有  $n \times 2^{1024}$ ,所以,每个块可以达到 1024 位,或者 128 个 8 位字节,相比之下,DES 的数据块是 8 个字符,AES 的块是 16 个字节。

应该指出,像以上描述的 RSA 用法有点类似于按 ECB 模式来使用对称密钥算法的做法——即,同样的输入块得到同样的输出块。因此,我们需要使用某种链接形式来加密数据。然而,在实践中,大多数基于 RSA 的系统主要利用公开密钥算法来分发一次性会话密钥,再将这些会话密钥用于某一个对称密钥算法,比如 AES 或者三重 DES。如果真的要利用 RSA 来加密大量的数据则速度太慢了,所以 RSA 被广泛用于密钥分发。

### 8.3.2 其他的公开密钥算法

虽然 RSA 已经被广泛使用了,但它并不是惟一已知的公开密钥算法。第一个公开密钥算法是背包算法(Merkle and Hellman, 1978)。背包算法的思想是,某一个人拥有大量的物品,每个物品的重量各不相同。为了编码一个消息,这个人秘密地选择其中一组物



品并将物品放到一个背包中。背包中物品的总重量被公开,所有可能的物品也被公开列出。但是,背包中物品的明细则是保密的。在特定的附加限制条件下,“根据给定的总重量找出可能的物品明细列表”这个问题被认为是一个计算上不可行的问题,从而构成了此公开密钥算法的基础。

算法的发明者 Ralph Merkle 非常确信这个算法不可能被攻破,所以他悬赏 100 美元奖金给破解算法的人。Adi Shamir(即 RSA 中的“S”)迅速地破解了算法,并领走了奖金。Merkle 并没有气馁,他又加强了算法,并悬赏 1000 美元奖金给破解新算法的人。Ronald Rivest(即 RSA 中的“R”)也迅速地破解了新算法,并领取了奖金。Merkle 不敢再为下一个版本悬赏 10 000 美元奖金了,所以“A”(Leonard Adleman)很是不幸。不管怎么样,背包算法已经不再被认为是安全的,在实践中也没有被采用。

其他一些公开密钥方案建立在计算离散对数的困难度基础之上。El Gamal(1985)和 Schnorr(1991)发明了使用这种原理的公开密钥算法。

另外还存在一些方案,比如基于椭圆曲线的公开密钥算法(Menezes and Vanstone, 1993),但是最主要的两大类算法是:(1)建立在“分解大数的困难度”基础上的算法,以及(2)建立在“以大素数为模来计算离散对数的困难度”基础上的算法。这些问题被认为确实很难解决,因为数学家们已经为之钻研了许多年而未能有所突破。

## 8.4 数字签名

许多法律的、金融的和其他文档的真实性是依据是否出现某一个经授权的手写签名来决定的。影印件是无效的。为了能够用计算机化的消息系统来代替纸和笔墨文档的物理传输系统,必须要找到一种方法来对文档进行签名,并且文档的签名不可被伪造。

设计一个代替手写签名的方案是非常困难的。基本上,我们需要的是这样一个系统,其中一方给另一方发送的签名消息必须满足以下的条件:

- (1) 接收方可以验证发送方所宣称的身份。
- (2) 发送方以后不能否认该消息的内容。
- (3) 接收方不可能自己编造这样的消息。

第一个要求是必需的,例如在金融系统中,当顾客的计算机向银行的计算机订购一吨黄金时,银行的计算机必须要确信:发出订单的计算机真正属于付款账号所属的公司。换句话说,银行必须要认证顾客的身份(顾客也要认证银行的身份)。

第二个要求也是必需的,它可以保护银行不会被欺骗。假设银行买进了一吨黄金,之后黄金的价格立刻跌了下来。一个不诚实的顾客可以起诉银行,宣称自己从来没有发出过购买黄金的订单。当银行在法庭上出示这条消息的时候,顾客否认自己曾经发送过此消息。这种“契约的任何一方以后都不能否认自己曾经签过名”的特性被称为不可否认性(nonrepudiation)。我们现在将要学习的数字签名方案可以用来提供这样的特性。

第三个要求也是必需的,它可以保护顾客的利益不受损害,例如,当黄金的价格暴涨时,银行企图构造一个“顾客要求购买一条黄金(而不是一吨黄金)”的签名消息。在这种欺骗的情形下,银行就可以保留剩余的黄金。

### 8.4.1 对称密钥签名

数字签名的一种做法是设立一个人人都信任并且又熟知一切的中心权威机构,比如 Big Brother(英国电影中的独裁人物),这里简称 BB。然后每个用户选择一个秘密密钥,并且亲手将它送到 BB 的办公室。因此,只有 Alice 和 BB 才知道 Alice 的秘密密钥  $K_A$ ,如此等等。

当 Alice 想要给她的银行业务员 Bob 发送一条签名的明文消息  $P$  时,她生成  $K_A(B, R_A, t, P)$ ,这里  $B$  是 Bob 的标识,  $R_A$  是 Alice 选择的一个随机数,  $t$  是一个时间戳(可用来保证该消息是最新的),  $K_A(B, R_A, t, P)$  是指用她的密钥  $K_A$  加密之后的消息。然后,她按照图 8.18 所示的方式将该消息发送出去。BB 看到该消息来自 Alice,于是解密该消息,并按图中所示给 Bob 发送一条消息。给 Bob 的消息包含了 Alice 的原始消息的明文  $P$  和一条经过签名的消息  $K_{BB}(A, t, P)$ 。Bob 现在执行 Alice 的请求。

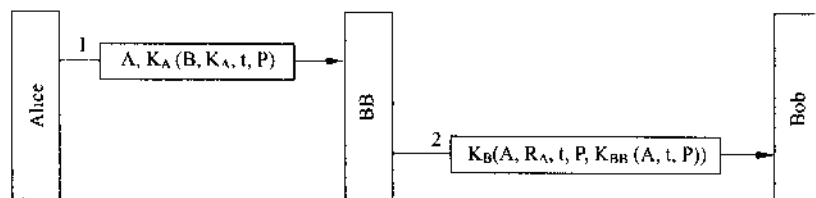


图 8.18 通过 Big Brother(BB)的数字签名

如果 Alice 后来拒绝承认自己曾经发送过该消息,那该怎么办呢? 第 1 步是,人人都可以起诉别人(至少在美国是这样的)。最后,当案子到了法庭上, Alice 顽固地否认自己曾经给 Bob 发送过这条有争议的消息时,法官将会问 Bob,他如何能保证这条有争议的消息确实来自 Alice 而不是 Trudy。Bob 首先指出, BB 在看到来自 Alice 的消息时,除非该消息是用  $K_A$  加密的,否则 BB 是不会接受该消息的,所以, Trudy 向 BB 发送一条谎称来自 Alice 的消息却没有立即被 BB 检测到,这是不可能的。

然后 Bob 在众目睽睽下出示了证据  $A: K_{BB}(A, t, P)$ 。Bob 说,这是一条由 BB 签名的消息,它可以证明 Alice 曾经发送  $P$  给 Bob。然后,法官请 BB(人人都信任 BB)解密证据  $A$ 。当 BB 证实了 Bob 说的是真话时,法官的判决就会有利于 Bob。案子最终结束。

在图 8.18 的签名协议中,一个潜在的问题是 Trudy 可能会重放其中某一条消息。为了使这个问题的危险性降低到最小,可以在每一个环节上用时间戳进行控制。而且, Bob 可以检查所有最近接收到的消息,看一看在这些消息中是否也用到过  $R_A$ 。如果确有消息用到了  $R_A$ ,则他可以将新收到的消息当作重放消息而丢弃。请注意,利用时间戳机制, Bob 将拒绝接收那些很老的消息。为了对付短时间内的重放攻击, Bob 只要检查每条进来消息中的  $R_A$ ,以判断在过去的一小时内是否曾经收到过来自 Alice 的这条消息。如果没有的话,则 Bob 就可以安全地认为这是一个新的请求。

### 8.4.2 公开密钥数字签名

利用对称密钥密码技术来实现数字签名的一个结构性问题是:每个人都必须信任

Big Brother。而且, Big Brother 能够解读所有签名的消息。从逻辑来看, 最有可能运行 Big Brother 服务器的候选机构是政府、银行、会计事务所和律师事务所。不幸的是, 老百姓对所有这些组织并不能寄予足够的信任。因此, 若能在签名文档的时候不要求通过一个可信的权威机构就好了。

幸运的是, 公开密钥密码学为这个领域做出了重要的贡献。我们假设公开密钥的加密算法和解密算法除了具有常规的  $D(E(P))=P$  属性以外, 还具有  $E(D(P))=P$  属性。(RSA 有这样的属性, 所以这个假设并非不合理。) 同时也假设, Alice 为了向 Bob 发送一条签名的明文消息  $P$ , 她传输的是  $E_B(D_A(P))$ 。请注意, Alice 知道她自己的(私有)密钥  $D_A$  以及 Bob 的公开密钥  $E_B$ , 所以, Alice 构造这条消息是可以做得到的。

当 Bob 收到这条消息时, 他像往常一样利用自己的私钥对消息做变换, 从而得到  $D_A(P)$ , 如图 8.19 所示。他将这份信息放在一个安全的地方, 然后通过使用  $E_A$  可得到原始的明文。

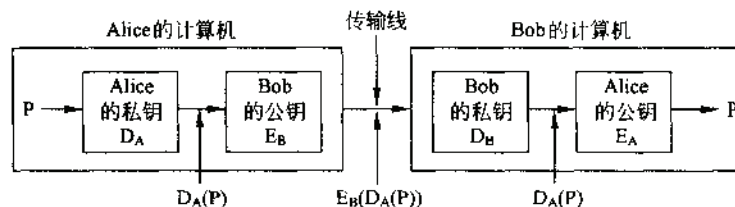


图 8.19 利用公开密钥密码技术的数字签名

为了看清这种签名特性的工作原理, 不妨假设 Alice 后来否认自己曾经给 Bob 发送过消息  $P$ 。当这个案子被提到法庭上的时候, Bob 可以同时出示  $P$  和  $D_A(P)$ 。法官很容易验证 Bob 是否真的拥有一条由  $D_A$  加密的消息, 他只需简单地在消息上应用  $E_A$  即可。由于 Bob 并不知道 Alice 的私钥是什么, 所以, Bob 能获得由 Alice 私钥加密的消息的惟一途径是 Alice 给他发送了这样的消息。当 Alice 因为作伪证和欺骗罪而被监禁起来的时候, 她就有足够的时间来设计新的有趣的公开密钥算法了。

尽管用公开密钥密码技术来实现数字签名是一种非常优美的方案, 但是, 这里还存在一些涉及到运行环境而并非基本算法的问题。首先, 只要  $D_A$  仍然是保密的, 则 Bob 就可以证明一条消息确实是 Alice 发送的。如果 Alice 泄漏了她的秘密密钥, 则这个论点就不再成立, 因为任何人都可以发送这样的消息, 包括 Bob 自己在内。

这时候就可能引发问题, 比如, 假定 Bob 是 Alice 的股票经纪人。Alice 告诉 Bob 购买一支特定的股票或者国库券。很快地, 股票价格急剧下降。Alice 为了否认她曾经给 Bob 发送过消息, 她跑到警察局声称她家遭到抢劫, 保存密钥的 PC 机被偷了。根据她所在州或者国家的法律, 她可能需要, 也可能不需要承担法律责任, 尤其是如果她声称是在下班以后回到家里时才发现被抢了(即发生抢劫几小时以后才发现)。

与这种签名方案有关的另一个问题是, 如果 Alice 决定要改变她的密钥, 那该怎么办呢? 很显然, 这样做是合法的, 而且定期改变密钥可能是一种很好的做法。如果后来发生了法律纠纷, 就像上面描述的那样, 那么, 法官将当前的  $E_A$  作用在  $D_A(P)$  上, 发现其结果

并不等于  $P$ 。这时候 Bob 就会很难堪。

原则上,任何一种公开密钥算法都可以用于数字签名。事实上的工业标准是 RSA 算法。许多安全产品使用了 RSA 算法。然而,1991 年,NIST 建议使用 El Gamal 公开密钥算法的一个变种作为它们新的数字签名标准(Digital Signature Standard, DSS)。El Gamal 算法的安全性建立在计算离散对数的困难度基础之上,而并非建立在分解大数的困难度基础上。

通常情况下,当政府试图推行一个密码学标准时,它总是会招致各种非议。DSS 也受到了批评,理由是:

- (1) 太神秘(这是 NSA 设计的协议,它使用了 El Gamal 算法)。
- (2) 太慢(在检查签名时比 RSA 慢了 10 至 40 倍)。
- (3) 太新(El Gamal 算法还没有被完全分析透彻)。
- (4) 太不安全(固定的 512 位密钥)。

在后续的修订版本中,当允许密钥长度增加到 1024 位以后,第 4 点基本上已经消除了。然而,前面两点仍然有效。

#### 8.4.3 消息摘要

对签名方法的一个批评是,它们通常将两种不相同的功能耦合在一起:认证和保密。通常情况下,认证是必要的,但是保密性并不一定必需。而且,如果一个系统只提供认证而没有提供保密性的话,则往往更加容易申请到出口许可。下面我们描述的认证方案不要求加密整条消息。

这个方案以单向散列函数的思想作为基础,这里的单向散列函数接受一个任意长度的明文作为输入,并且根据此明文计算出一个固定长度的位串。这个散列函数 MD 通常被称为消息摘要(message digest),它有 4 个重要的特性:

- (1) 给定  $P$ ,很容易计算  $MD(P)$ 。
- (2) 给定  $MD(P)$ ,要想找到  $P$  在实践中是不可能的。
- (3) 在给定  $P$  的情况下,没有人能够找到满足  $MD(P')=MD(P)$  的  $P'$ 。
- (4) 在输入明文中即使只有 1 位的变化,也会导致完全不同的输出。

为了满足第 3 条,散列结果应该至少 128 位长,最好还能更长一些。为了满足第 4 条,散列结果必须彻底弄乱明文中的位,譬如像我们在对称密钥加密算法中看到的那样。

从一段明文来计算一个消息摘要必须比用公开密钥算法来加密这段明文要快得多,所以消息摘要可以被用来加速数字签名算法。为了看清楚这个工作过程,请再次考虑图 8.18 的签名协议。BB 现在不再利用  $K_{BB}(A, t, P)$  作为  $P$  的签名,而是计算消息摘要,他将 MD 作用在  $P$  上,得到  $MD(P)$ 。然后 BB 用  $K_{BB}(A, t, MD(P))$  代替原来的  $K_{BB}(A, t, P)$ ,作为他发送给 Bob 的消息(被  $K_B$  加密)中的第 5 项。

如果发生纠纷的话,Bob 可以出示  $P$  和  $K_{BB}(A, t, MD(P))$ 。当 Big Brother 根据法官的要求将后者解密出来以后,Bob 有了  $MD(P)$ ,以及他所宣称的  $P$ 。这里  $MD(P)$  保证是真实的。然而,由于 Bob 不可能有效地找到另一条具有相同散列结果的消息,所以法官很容易就可以确信 Bob 讲的是事实。按照这种方式来使用消息摘要可以节省加密时

间和消息传输的开销。

消息摘要也可以用在公开密钥密码系统中,如图 8.20 所示。在这里,Alice 首先计算明文的摘要,然后她针对摘要进行签名,并且将签名之后的摘要与明文本身一起发送给 Bob。如果 Trudy 中途偷换掉 P 的话,则当 Bob 计算 MD(P)时就可以看出这一点。

#### MD5

目前已经有许多种消息摘要函数被提出来了。其中最为广泛使用的函数是 MD5(Rivest, 1992)和 SHA-1(NIST, 1993)。MD5 是 Ronald Rivest 设计的一系列消息摘要算法中的第 5 个

算法。它通过一种足够复杂的方法来弄乱明文消息中的所有位,每一个输出位都要受到每一个输入位的影响。简短来说,它首先将原始的明文消息填补到 448 位(以 512 为模)的长度。然后,消息的长度被追加成 64 位整数,因而整个输入的长度是 512 位的倍数。最后一个预计算步骤是将一个 128 位的缓冲区初始化成一个固定的值。

现在开始计算。每一轮取出一个 512 字节的输入块,并且将它与 128 位的缓冲区彻底地混淆起来。为了达到更好的混淆效果,还需要引入一个通过正弦函数构造得到的表格。在这里,之所以使用一个像正弦这样的知名函数,并不是因为它比一个随机数发生器更加具有随机性,而是为了避免嫌疑,不至于让人觉得设计者在算法中内置了一个只有他才能进入的精巧后门。前面曾经提到过,IBM 拒绝公开 DES 中 S 盒的设计原理引起了大量关于后门的猜测。Rivest 希望避免这种嫌疑。MD5 对每一个输入块执行 4 轮。这个过程不断进行,直至所有的输入块都被执行完毕。最后,128 位缓冲区中的内容构成了最终的消息摘要。

现在,MD5 已经存在 10 年多了,许多人破解过这个算法。有一些脆弱性已经被发现了,但是某些特定的内部步骤使得它免于被攻破。然而,如果 MD5 内部剩下的屏障也沦陷的话,它最终有可能会被打破。不管怎么样,到本书写作的时候,它仍然还屹立着。

#### SHA-1

另一个主要的消息摘要函数是 SHA-1(Secure Hash Algorithm 1),它由 NSA 开发,而且得到 NIST 的赏识,并被标准化在 FIPS 180-1 中。如同 MD5 一样,SHA-1 也按照 512 位的块大小来处理输入数据,惟一与 MD5 不同的是,它生成一个 160 位的消息摘要。对于 Alice 来说,为了给 Bob 发送一条非保密的、但是签名的消息,一种典型的方法如图 8.21 所示。在这里,她的明文消息被送入到 SHA-1 算法中,以便得到一个 160 位的 SHA-1 散列值。然后,她用自己的 RSA 私钥对散列值进行签名,并且将明文消息和签过名的散列值发送给 Bob。

Bob 在接收到消息以后,他自己计算 SHA-1 散列值,并且利用 Alice 的公钥来解密签过名的散列值,从而得到原始的散列值 H。如果这两者一致的话,则认为该消息是有效的。因为对于 Trudy 来说,她无法在传输过程中既要修改(明文)消息,也要使修改之

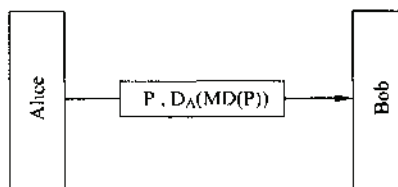


图 8.20 使用消息摘要的数字签名



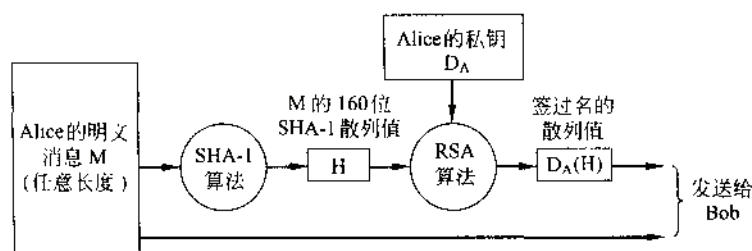


图 8.21 利用 SHA-1 和 RSA 对非保密的消息进行签名

后的消息散列到同一个  $H$  上,所以,Bob 可以很容易地检测到 Trudy 对于消息  $P$  所做的任何改变。对于那些“完整性很重要,但是其内容并不需要保密”的消息,图 8.21 中的方案已被广泛使用。这种方案只需相对较小的计算开销,并且它保证,在传输过程中对明文消息所做的任何修改都将(以极高的概率)被检测到。

现在我们来粗略地看一下 SHA-1 的工作原理。首先它在消息的末尾填补一个 1 位,然后跟上一数量 0 位,0 位的个数必须保证填补之后的消息长度为 512 位的倍数。然后构造一个包含了填补之前消息长度的 64 位整数,并将它与消息的低 64 位进行或 (OR) 操作。在图 8.22 中,填补的位被显示在消息的右侧,因为英语文本和图片是按照从左至右的顺序来书写或绘制的(即,右下角往往被认为是图片的尾部)。对于计算机来说,这种方向特性对应于 big-endian 字节序的机器,比如 SPARC,但是,SHA-1 总是填补在消息的末尾,而不管当前机器使用了哪一种字节序。

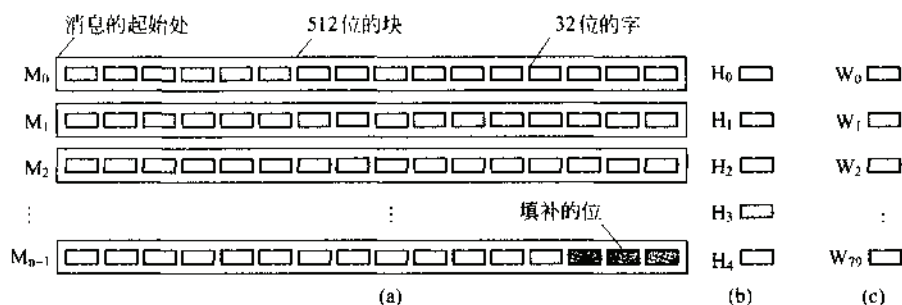


图 8.22

(a) 一个消息被填补至 512 位的倍数; (b) 输出的变量; (c) 字数组

在计算过程中,SHA-1 维护了 5 个 32 位变量,从  $H_0$  至  $H_4$ ,最后散列的结果就放在这些变量中。图 8.22(b)显示了这些变量。它们的初始常量被定义在标准中。

现在按照顺序逐个处理数据块  $M_0$  至  $M_{n-1}$ 。对于当前的块,首先将 16 个字复制到一个 80 字的辅助数组  $W$  的前面部分,图 8.22(c)显示了数组  $W$ 。然后,利用下面的公式来填充  $W$  中其他的 64 个字:

$$W_i = S^b(W_{i-3} \text{ XOR } W_{i-8} \text{ XOR } W_{i-14} \text{ XOR } W_{i-16}) \quad (16 \leq i \leq 79)$$

这里  $S^b(W)$  代表了 32 位字  $W$  循环左移  $b$  位。现在,利用  $H_0$  至  $H_4$  分别初始化 5 个



临时变量(从 A 至 E)。

实际的计算过程可以用下面的伪 C 代码来表达:

```
for (i = 0; i < 80; i++) {
    temp = S5(A) + fi(B, C, D) + E + Wi + Ki;
    E=D; D=C; C=S30(B); B = A; A = temp;
}
```

标准中定义了这里的常量 K<sub>i</sub>。混合函数 f<sub>i</sub>被定义为:

$$\begin{aligned} f_i(B, C, D) &= (B \text{ AND } C) \text{ OR } (\text{NOT } B \text{ AND } D) & (0 \leq i \leq 19) \\ f_i(B, C, D) &= (B \text{ XOR } C) \text{ XOR } D & (20 \leq i \leq 39) \\ f_i(B, C, D) &= (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) & (40 \leq i \leq 59) \\ f_i(B, C, D) &= (B \text{ XOR } C) \text{ XOR } D & (60 \leq i \leq 79) \end{aligned}$$

当 for 循环中的 80 次迭代完成以后, A 至 E 分别被加到 H<sub>0</sub> 至 H<sub>4</sub> 上。

当第一个 512 位的块处理完成以后, 下一个块又接着开始。W 数组又被按照新的块来初始化, 但是, H 数组仍然不变。当这一块完成以后, 下一块又开始, 如此周而复始, 直到所有的 512 位消息块都被处理完毕。当最后一块被处理完以后, H 数组中的 5 个 32 位字即为密码学意义上的 160 位散列值。RFC 3174 给出了 SHA-1 算法的完整 C 代码。

SHA-1 有一些新的版本正在开发之中, 它们分别针对 256、384 和 512 位的散列值。

#### 8.4.4 生日攻击

在密码学领域中, 没有一件事情会像表面上看到的那么简单。有人可能会认为, 为了攻破一个 m 位的消息摘要, 将需要  $2^m$  数量级的操作次数。实际上, Yuval(1979)在他的论文“*How to Swindle Rabin*”中发表了一种生日攻击方法, 用这种方法通常只需要  $2^{m/2}$  量级的操作次数, 现在他的这篇论文已经成为经典。

这种攻击的思想来源于数学教授们通常在他们的概率课堂上采用的一种技术。问题是: 当一个班级中有多少个学生时, 才使得这个班级中有两个学生具有相同生日的概率超过 1/2? 大多数学生都认为答案将会超过 100。实际上, 根据概率论, 答案只是 23。如果不给出严密的分析, 仅仅从直觉上, 我们可以看到, 在 23 个学生的情况下, 我们可以组成  $(23 \times 22)/2 = 253$  对不同的组合, 每一对组合有 1/365 的命中概率。照这样看来, 这个答案实际上也就不那么令人惊奇了。

推广至更为一般的情形, 如果在 n 个输入(人、消息, 等)和 k 个可能的输出(生日、消息摘要, 等)之间存在某一种从输入到输出的映射关系, 那么, 这里共有  $n(n-1)/2$  个输入对, 若  $n(n-1)/2 > k$ , 则至少有一个匹配的机会是非常大的。因此, 近似地, 存在一个匹配的条件是  $n > \sqrt{k}$ , 这个结果意味着只要生成大约  $2^{32}$  条消息, 然后在这些消息中寻找具有相同消息摘要的两条消息就有可能攻破一个 64 位的消息摘要。

现在我们来看一个实际的例子。州立大学的计算机科学系有一个终生教职员的职位, 并且有两个候选人: Tom 和 Dick。由于 Tom 比 Dick 早雇用两年, 所以他是首先被考察的对象。如果他通过了, 则 Dick 就没有机会了。Tom 知道系行政主管 Marilyn 非常欣赏

他的工作,所以他请 Marilyn 为他写一封推荐信给系主任,因为最终的决定权在系主任手中。所有的信件一旦送出去之后,都是保密的。

Marilyn 告诉她的秘书 Ellen 给系主任写封信,并大略地描述了她在信中要表达的内容。当 Ellen 写完信后,Marilyn 将会检查一遍,并计算和签署 64 位摘要,然后发送给系主任。Ellen 可以稍后通过电子邮件发送这封信。

对于 Tom 来说很不幸的是,Ellen 正迷恋着 Dick,因而她想要陷害 Tom,于是,她写下了以下一封有 32 个等价选项的信:

Dear Dean Smith,

This [letter | message] is to give my [honest | frank] opinion of Prof. Tom Wilson, who is [a candidate | up] for tenure [now | this year]. I have [known | worked with] Prof. Wilson for [about | almost] six years. He is an [outstanding | excellent] researcher of great [talent | ability] known [worldwide | internationally] for his [brilliant | creative] insights into [many | a wide variety of] [difficult | challenging] problems.

He is also a [highly | greatly] [respected | admired] [teacher | educator]. His students give his [classes | courses] [rave | spectacular] reviews. He is [our | the Department's] [most popular | best-loved] [teacher | instructor].

[In addition | Additionally] Prof. Wilson is a [gifted | effective] fund raiser. His [grants | contracts] have brought a [large | substantial] amount of money into [the | our] Department. [This money has | These funds have] [enabled | permitted] us to [pursue | carry out] many [special | important] programs, [such as | for example] your State 2000 program. Without these funds we would [be unable | not be able] to continue this program, which is so [important | essential] to both of us. I strongly urge you to grant him tenure.

而且,当 Ellen 写好了以上这封信并录入到计算机中以后,她又写了第二封信:

Dear Dean Smith,

This [letter | message] is to give my [honest | frank] opinion of Prof. Tom Wilson, who is [a candidate | up] for tenure [now | this year]. I have [known | worked with] Tom for [about | almost] six years. He is a [poor | weak] researcher not well known in his [field | area]. His research [hardly ever | rarely] shows [insight in | understanding of] the [key | major] problems of [the | our] day.

Furthermore, he is not a [respected | admired] [teacher | educator]. His students give his [classes | courses] [poor | bad] reviews. He is [our | the Department's] least popular [teacher | instructor], known [mostly | primarily] within [the | our] Department for his [tendency | propensity] to [ridicule | embarrass] students [foolish | imprudent] enough to ask questions

in his classes.

[In addition | Additionally] Tom is a [poor | marginal] fund raiser. His [grants | contracts] have brought only a [meager | insignificant] amount of money into [the | our] Department. Unless new [money is | funds are] quickly located, we may have to cancel some essential programs, such as your State 2000 program. Unfortunately, under these [conditions | circumstances] I cannot in good [conscience | faith] recommend him to you for [tenure | a permanent position].

现在 Ellen 利用她的计算机彻夜地计算出每一封信的  $2^{64}$  个消息摘要。第一封信的消息摘要与第二封信的消息摘要中存在匹配的机会是很大的,如果没有找到匹配的话,她可以再增加一些选项,然后利用周末时间重新计算一次。现在假设她找到了一个匹配。我们将“好的”信称为 A,“坏的”信称为 B。

Ellen 现在以电子邮件的方式将信 A 发送给 Marilyn,请她批准。同时将信 B 完全保密,不让任何人看到。Marilyn 当然同意 Ellen 写的信 A 的内容,于是她计算出这封信的 64 位消息摘要,并且对摘要进行签名,然后将签过名的摘要单独发送给系主任 Smith。而 Ellen 则单独地将信 B 以电子邮件方式寄送给系主任(请注意,她寄送的并不是 Marilyn 所想象的信 A)。

系主任得到了信和签过名的消息摘要以后,他针对信 B 运行消息摘要算法,看一看是否与 Marilyn 送给他的摘要一致,然后解雇了 Tom。系主任并没有意识到 Ellen 生成了两封具有相同消息摘要的信,并且发送给他的信与 Marilyn 看到并批准的信不是同一封。(我们也可以换一个结尾:Ellen 将自己所做的一切全部告诉了 Dick。Dick 非常震惊,并且与她断绝了关系。Ellen 也非常气愤,便向 Marilyn 坦白了一切。Marilyn 打电话告诉了系主任,最终 Tom 获得了职位。)生日攻击对于 MD5 是很难实施的,因为即使每秒钟产生 10 亿个摘要,那也需要花 500 年时间来计算两封信的  $2 \times 2^{64}$  个摘要(每封信有 64 种变化形式),而且即使这样还不能保证总是存在一个匹配。当然,若并行使用 5000 台计算机同时计算,则 500 年变成了 5 个星期。SHA-1 在这方面要好一点(因为它的摘要更长一些)。

## 8.5 公钥的管理

公开密钥密码学使得人们有可能在不共享公共密钥的情况下仍然可以安全地进行通信。它也使得人们有可能在不存在可信第三方的情况下为消息做签名。最后,利用签过名的消息摘要,人们很容易就可以验证自己所接收到的消息的完整性。

然而,请等一等,我们这里还掩盖了一个问题:如果 Alice 和 Bob 彼此之间并不认识,那么,他们如何获得对方的公钥来开始通信过程呢?一种很显然的解决方案是将公钥放在自己的 Web 站点上,但是这种方案并不能工作,理由如下所述。假设 Alice 想要在 Bob 的 Web 站点上寻找他的公钥,那么她该怎么办呢?首先,她在浏览器中输入 Bob 的 URL。然后,她的浏览器找到 Bob 主页的 DNS 地址,并向该地址发送一个 GET 请求,如

图 8.23 中所示。不幸的是,Trudy 截获了这个请求,并且将一个伪造的主页送回给 Alice 作为应答,这个伪造的主页可能是 Bob 主页内容的一份副本,只不过其中 Bob 的公钥被替换成 Trudy 的公钥。当 Alice 现在用  $E_T$  加密她的第一条消息时,Trudy 解密并阅读该消息,然后用 Bob 的公钥重新进行加密,再发送给 Bob,而 Bob 根本不知道 Trudy 已经阅读了他所接收到的消息。更糟的是,Trudy 在为 Bob 重新加密消息之前也可能对消息做一些修改。很显然,这里需要某种机制来确保公钥交换的安全性。

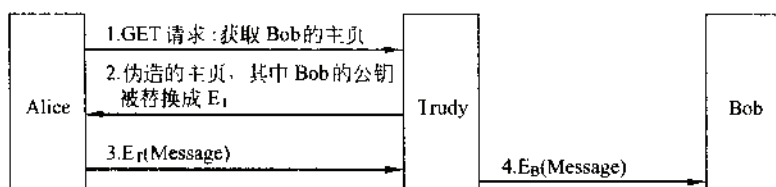


图 8.23 Trudy 破坏公钥加密机制的一种方法

### 8.5.1 证书

为了安全地分发公钥,第一种尝试的方法是,我们想象有一个密钥分发中心一天 24 小时地根据客户的需求提供在线公钥服务。这种方案有很多问题,其中一个问题是它的伸缩性不好,密钥分发中心很快就会变成一个瓶颈。而且,如果它宕机的话,则 Internet 安全性也将立即停止下来。

由于这些原因,人们已经开发出了另一种不同的方案,这种方案并不要求密钥分发中心始终不停地保持在线。实际上,它甚至根本不用在线。相反,它所要做的事情是证明每个公钥属于个人、公司或者其他的组织。这种证明公钥所属权的组织现在被称为 CA (Certification Authority, 证书权威机构)。

作为一个例子,假设 Bob 希望 Alice 和其他的人能够安全地与他进行通信。他可以到一个 CA 那里,出示他的公钥和护照或者驾驶证,请求证明他的公钥。然后,CA 给他颁发一个证书,其中的内容类似于图 8.24 中所示的证书,而且 CA 用自己的私钥对证书的 SHA-1 散列值进行签名。然后,Bob 付给 CA 一定的费用,并获得一片软盘,其中包含了证书和签过名的散列值。

证书的基本任务是将一个公钥与安全个体(个人、公司、等)的名字绑定在一起。证书

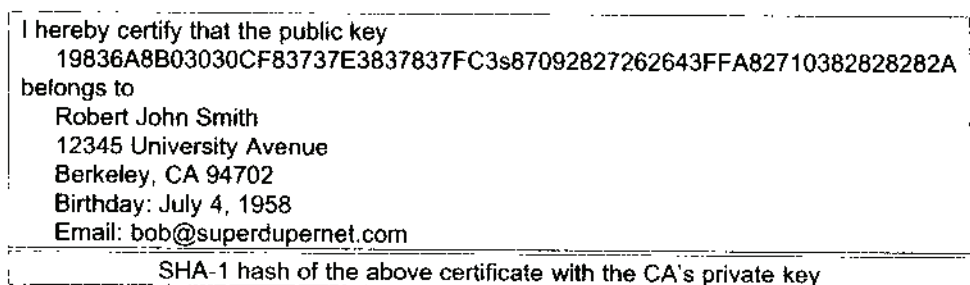


图 8.24 一个可能的证书和签过名的散列值

本身并不是保密的,也没有被保护。例如,Bob可能决定将他的新证书放到Web站点上,他的做法是在主页上使用这样一个链接:点击这里可得到我的公钥证书。当用户单击该链接之后,就可以得到Bob的证书和签名块(即该证书的SHA-1散列值经过签名之后的数据块)。

现在我们再来看一遍图8.23中的场景。当Trudy截获了Alice请求Bob主页的命令时,她该怎么办呢?她可以把她自己的证书和签名块放到伪造的页面上,但是当Alice读取证书的时候,她立即就会发现自己不是在跟Bob通话,因为Bob的名字不在证书中。Trudy可以临时地修改Bob的主页,用她自己的公钥来替换Bob的公钥。然而,当Alice对证书运行SHA-1算法的时候,她得到的散列值与她将CA的公钥(这是众所皆知的)应用在签名块上所得到的散列值不一致。由于Trudy拿不到CA的私钥,所以她无法生成一个包含她的公钥的签名块,也就无法构造出让Alice相信她是Bob的Web页面。通过这种方式,Alice可以确信她得到的是Bob的公钥,而不是Trudy或者其他人的公钥。正如前面我们所承诺的,这种方案不要求CA一直提供在线验证服务,从而消除了潜在的瓶颈。

虽然证书的标准功能是将一个公钥绑定到一个安全个体上,但是,证书也可以被用来将一个公钥绑定到一个特征(attribute)上。例如,一个公钥可以表达这样的含义:该公钥属于某一个年龄超过18岁的人。这样的公钥可以被用来证明私钥的所有者不是一个未成年人,因而他(或她)可以访问少儿不宜的资料,等等,但是这个公钥也没有泄露所有者的身份。通常情况下,持有证书的人往往会把这个证书发送给Web站点、其他的安全个体,或者对年龄较为敏感的进程等。然后,此站点、安全个体或者进程生成一个随机数,并用证书中的公钥对随机数进行加密。如果所有者能够解密随机数并送回来,则证明所有者确实具备了该证书中所声明的特征。或者,此随机数被用来生成一个会话密钥,以保证后续通话的安全。

在证书中可能包含特征的另一个例子是用在一个面向对象的分布式系统中。每个对象通常有多个方法。对象的所有者可以向每一个顾客提供一个证书,在证书中给出一个位图来表明允许该顾客调用哪些方法,并且用一个签过名的证书将位图与一个公钥绑定在一起。同样地,如果证书持有者可以证明自己确实拥有对应的私钥,那么,他将能够执行位图中指定的方法。这个例子用法也有这样的特性:所有者的身份并没有暴露,在非常关注隐私保护的情况下,这种特性是很有用的。

### 8.5.2 X.509

如果每一个想要为某些资料做签名的人都到CA那里申请各种不同类型的证书,那么,管理各种不同的证书格式很快就成为一个问题。为了解决这个问题,ITU已经设计并批准了一个专门针对证书格式的标准。该标准被称为X.509,现在已经广泛应用于Internet上。自从1988年首次被标准化以来,它已经经历了三个版本。下面我们将讨论第3版(V3)。

X.509受到了OSI领域的严重影响,它借用了OSI领域中某些非常糟糕的特性(比如命名和编码)。然而,令人惊奇的是,在几乎所有其他的领域中,从机器地址到传输协

议,再到电子邮件格式,IETF一般都忽略 OSI,而试图按照自己的方式来做得更好,但是在证书格式上,IETF却采纳了 X.509。X.509 的 IETF 版本由 RFC 3280 来描述。

X.509 的核心是提供了一种描述证书的格式。图 8.25 列出了一个证书中最主要的域。这里给出的描述仅仅说明了每个域的一般性用途。有关更多的信息,请参考标准本身或者 RFC 2459。

英文域名	中文	含义
Version	版本	X.509 的哪个版本
Serial number	序列号	序列号加上 CA 的名字可以惟一标识当前证书
Signature algorithm	签名算法	用于为证书做签名的算法
Issuer	颁发者	CA 的 X.500 名字
Validity period	有效期	有效期的起止时间
Subject name	主题名	该证书所证明的密钥所有者实体
Public key	公钥	主题的公钥,以及使用该公钥的算法的 ID
Issuer ID	颁发者标识符	一个可选的 ID,惟一标识了证书的颁发者
Subject ID	主题标识符	一个可选的 ID,惟一标识了证书的主题
Extensions	扩展域	目前已经定义了许多扩展域
Signature	签名	证书的签名(用 CA 的私钥做的签名)

图 8.25 X.509 证书的基本域

例如,如果 Bob 在 Money 银行的信贷部门工作,那么,他的 X.500 地址可能如下:

/C=US/O=MoneyBank/OU=Loan/CN=Bob/

这里 C 代表国家,O 代表组织,OU 代表组织单位,CN 是一个普通名字。CA 和其他的实体也用类似的方式来命名。关于 X.500 名字的一个实质成问题是,如果 Alice 试图通过 bob@moneybank.com 与 Bob 联系,而她拿到的证书中却是 X.500 名字,那么,对于 Alice 来说,该证书是否引用了她想要的那个 Bob 并不是很显然。幸运的是,从第 3 版开始,它也允许使用 DNS 名字来代替 X.500 名字,所以,这个问题最终可能被消除了。

证书的编码使用了 OSI ASN.1(Abstract Syntax Notation 1,抽象语法标记 1),你可以将这种编码想象成与 C 语言的结构(struct)类似的描述形式,除此以外它也包含一些非常特殊和细微的标记。有关 X.509 更多的信息,可以参照(Ford and Baum, 2000)。

### 8.5.3 公开密钥基础设施

由一个 CA 来颁发全世界所有的证书显然是不切实际的。它将会不堪重负,并且成为一个中心失败点。一种可能的解决方案是使用多个 CA,并且所有的 CA 由同一个组织来运行,它们使用同一个私钥来签名证书。虽然这样可以解决负载太重和单失败点的问题,但是它又引入了新的问题:密钥泄漏。如果有几十台服务器遍布在世界各地,并且所有的服务器都持有 CA 的私钥,则私钥被偷窃或者泄漏的机会就会显著地增加。由于该



私钥的暴露将会影响到全世界的电子安全基础设施,所以,使用一个中心 CA 的做法是非常危险的。

另外,由哪一个组织来运行这个 CA 呢? 很难想象有哪一个权威机构能在全球范围内被普遍接受和信任。在有些国家中,人们坚持这个组织应该是一个政府机构,而在其他的国家中,人们又坚持这个组织不应该是一个政府机构。

由于这些理由,人们又发展了另一种证明公钥身份的方法。它的通用名称是 **PKI (Public Key Infrastructure, 公开密钥基础设施)**。在这一小节中,我们将一般性地概述一下 PKI 的工作原理,不过,由于现在有许多关于 PKI 的提案和建议,所以有些细节将来可能会发生变化。

一个 PKI 有多个部件,包括用户、CA、证书和目录。PKI 所做的事情是提供一种方法将这些部件有序地组织起来,并且定义了各种文档和协议的标准。一种特别简单的 PKI 形式是一个 CA 层次,如图 8.26 所示。在这个例子中,我们显示了三个层次,但是在实践中,层次的数目可能更少,也可能更多。最顶级的 CA,即层次的根,它的责任是证明第二级的 CA,这种 CA 我们称为 **RA (Regional Authority, 区域权威机构)**,因为它们可能覆盖某一个地理区域,比如一个国家或者一个洲。然而,这个术语不是标准的;实际上,PKI 中并没有定义标准的术语来表达树结构的不同层次。这些 RA 又依次证明下一级 CA 的真实性;在图中所示的 PKI 结构中,这些下级 CA 才真正为组织和个人颁发 X.509 证书。当根 CA 授权一个新的 RA 时,它生成一个 X.509 证书,并且在证书中声明它已经批准了这个 RA,同时将新 RA 的公钥也包含在这个证书中;根 CA 为证书签过名以后,将它交给 RA。类似地,当 RA 批准一个新的 CA 时,它产生一个证书并为其签名;该证书声明了它已被 RA 批准,并且包含了 CA 的公钥。

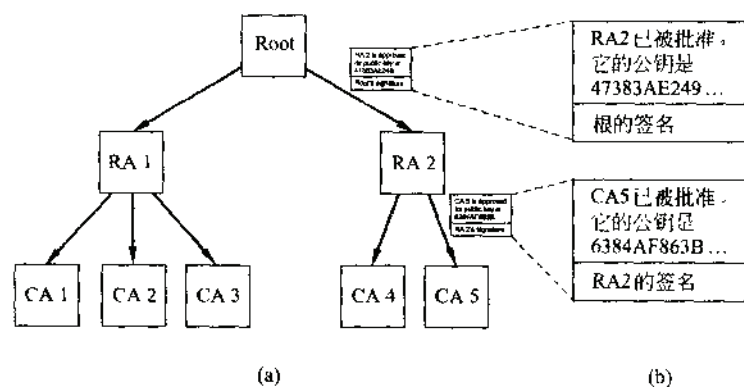


图 8.26

(a) 一个层次状的 PKI; (b) 证书链

我们的 PKI 按照如下的方式来工作。假设 Alice 需要 Bob 的公钥以便与他通信,所以,她搜寻并找到了一个包含 Bob 公钥的证书,该证书已由 CA5 签过名。但是, Alice 从来没有听说过 CA 5。她甚至可能认为 CA5 是 Bob 的 10 岁的女儿。她可以到 CA 5 那里说: 请证明你的合法性。CA 5 用它从 RA 2 那里获得的证书作为回答,该证书包含了

CA 5 的公钥。现在 Alice 拿到了 CA 5 的公钥,所以她会验证 Bob 的证书确实是由 CA 5 签名的,因此是合法的。

除非 RA 2 是 Bob 的 12 岁的儿子,否则接下来的步骤是,Alice 请 RA 2 证明它的合法性。RA2 对此的应答是一个由根 CA 签名的证书,其中包含了 RA 2 的公钥。现在 Alice 可以确信她真正拿到了 Bob 的公钥。

但是,Alice 如何找到根 CA 的公钥呢?这是 PKI 的魔术所在。它假设每个人都知道根 CA 的公钥。例如,她的浏览器可能在发行的时候已经预装了根的公钥。

Bob 是一个非常友好的合作伙伴,他不想给 Alice 增加太多额外的工作。他知道她将要检查 CA5 和 RA2 的合法性,所以,为了省去她的一些麻烦,他把两个必要的证书收集起来,并将这两个证书连同他自己的证书一起交给 Alice。现在,Alice 可以使用她自己所了解到的根证书来验证顶级证书和其中所包含的公钥,进而再验证第二级证书。按照这种方法,Alice 不用跟任何人联系就可以完成验证工作。因为证书全都是经过签名的,所以她会检测出所有对证书内容的篡改行为。像这样由底下回溯到树根的证书链有时候被称为信任链(chain of trust),或者证书路径(certification path)。这项技术已被广泛应用于实践中。

当然,我们还有另一个问题,即谁来运行根 CA 呢?解决的方案是,并非只有一个根,而是有多个根,每个根有它自己的 RA 和 CA。实际上,现代的浏览器在安装的时候预先加载了 100 多个根的公钥,有时候这些根被称为信任锚(trust anchor)。通过这种方式,就可以避免在全球范围内使用单一的可信权威机构。

但是,现在又引出另一个问题,即浏览器厂商如何确定哪些宣称的信任锚是可靠的,哪些是不可靠的。对于用户来说,归根到底是要信任浏览器厂商会作出明智的选择,相信它们不会简单地把那些愿意支付包含费的根 CA 全部变成信任锚。大多数浏览器允许用户检查根密钥(通常以证书的形式出现,并且包含根自身的签名),也允许用户删除掉那些看起来不太让人放心的根密钥。

## 目录

对于任何一个 PKI,另一个问题是在哪里存放证书(以及能回溯到某一个知名信任锚的证书链)。一种可能的方案是让每个用户保存他(或她)自己的证书。虽然这样做很安全(即用户不可能在篡改了签过名的证书以后不会被检测到),但是并不方便。另一种已被提出来的方案是将 DNS 做成一个证书目录。Alice 在与 Bob 联系之前,她或许要通过 DNS 来查找 Bob 的 IP 地址,所以,为什么不让 DNS 在返回 IP 地址的时候也返回 Bob 的完整证书链呢?

有些人认为这是正确的解决之道,但是,其他人则希望使用专门的目录服务器来管理 X.509 证书。这样的目录允许用户利用 X.509 名字的属性来提交查询任务。例如,在理论上,这样的目录服务可以回答诸如这样的问题:“将所有在美国或者加拿大销售部门工作的名为 Alice 的人的名单交给我。”LDAP 可能是一个胜任此任务(即保存此类信息)的候选目录标准。

## 撤销

现实世界中充满了各种各样的证书,比如护照和驾驶证。有时候这些证书可以被撤销,例如,对于酒后驾车或者其他的违章驾驶行为,警察局有权吊销驾驶员的驾驶证。在数字世界中,同样的问题也会发生:一个证书的颁发者可能因为证书持有者以某种方式滥用他的证书而决定撤销他的证书。如果主题的私钥已被泄漏,或者更糟糕的情况下,CA 的私钥被泄漏,那么,这时候相关的证书也要被撤销。因此,任何一个 PKI 都需要处理有关证书撤销的事宜。

在这件事情上,第一步是让每个 CA 定期地发布一个 CRL (Certificate Revocation List, 证书撤销列表),该 CRL 列出了所有已被撤销的证书的序列号。由于证书包含了过期时间,所以 CRL 只需包含那些尚未过期的证书的序列号。一旦证书的过期时间已经过去,则该证书自动失效,所以,在那些已经过期的证书与真正被撤销的证书之间并不需要做专门的区分。无论哪一种情形,这些证书都不能再被使用了。

不幸的是,引入 CRL 意味着一个用户在使用证书之前必须先要获得 CA 的 CRL,以确定该证书是否已被撤销。如果它确实已被撤销,则用户就不应该再使用该证书。然而,即使该证书不在 CRL 列表中,它也有可能在 CA 发布 CRL 列表之后刚刚被撤销。因此,真正有保证的惟一途径是询问 CA。而且,当下次再使用这个证书的时候,用户必须再次询问 CA,因为该证书有可能在几秒钟之前刚刚被撤销掉。

另一个复杂之处在于,一个已撤销的证书也可以被恢复为有效证书,例如,如果撤销的原因仅仅是因为用户没有支付必要的费用。由于必须要处理撤销(可能也要处理恢复)的证书,所以,证书的最佳特性之一,即用户在使用证书的时候可以不必与 CA 联系,也不复存在。

那么,CRL 应该被保存在哪里呢?一个理想的选择是证书本身所在的地方。一种策略是,让 CA 主动地定期推出 CRL,然后由各个目录对 CRL 进行处理,它们只需移除掉那些被撤销的证书即可。如果目录本身不保存证书的话,则可以将 CRL 缓存在网络中的各个便利之处。由于 CRL 本身也是一个被签过名的文档,所以,如果它被篡改的话,用户很容易检测出来。

如果证书的有效期很长,则 CRL 也将很长。例如,如果信用卡的有效期为 5 年,则尚在有效期内而被撤销的信用卡的数量,一定比每隔 3 个月发行新卡的情形要多得多。处理这种长 CRL 的一种标准方法是每过一段(较长)时间发行一个全列表,但是期间频繁地发行 CRL 更新消息。这样做可以减少因分发 CRL 而需要的带宽。

## 8.6 通信安全

现在我们已经结束了对各种交易工具的学习。大多数重要的技术和协议也都已经涉及到了。本章剩余部分将讨论如何把这些技术应用到实践中以提供网络安全性,同时在本章末尾还将提到一些有关安全性之社会因素的思考。

在接下来的 4 小节中,我们将讨论通信安全,也就是说,如何将数据位秘密地、未被篡

改地从源端传送到目标端,以及如何将有害的数据位排除在门外。这些问题决不是网络环境中惟一的安全问题,但它们无疑是最重要的,所以,通信安全是一个很好的学习起点。

### 8.6.1 IPSec

多年以来 IETF 一直很清楚,Internet 缺乏安全性。要在 Internet 上增加安全性并不容易,因为曾经爆发过一场关于在哪里加入安全性的争论。绝大多数安全专家相信,为了真正做到安全,加密和完整性检查必须是端到端的(即位于应用层上)。这就是说,源进程对数据进行加密,以及/或者实施完整性保护,然后将数据发送给目标进程;在目标进程中,数据被解密,以及/或者被验证其完整性。若企图在这两个进程之间(包括在操作系统内部)篡改数据,则必定可以被检测到。这种做法的麻烦在于,它要求改变所有的应用系统,以便它们能够感知到安全特性的存在。从这个角度来说,其次最好的方法是将加密功能放到传输层上,或者放到应用层与传输层之间的一个新层上,这样仍然可以做到端到端安全,但是不要求改变应用系统。

相反的观点是,用户并不理解安全性,他们无法正确地使用安全特性,而且没有人愿意以任何一种方式来修改已有的程序,所以网络层应该认证和/或者加密分组,而不要让用户卷入进来。在经过多年的激烈争论之后,这种观点赢得了足够的支持,因此 IETF 定义了一个网络层安全标准。这种论点取胜的部分原因是,在网络层上实施加密操作,并没有妨碍那些了解安全性的用户正确地使用这些安全特性,而且多多少少可以帮助那些对安全性一无所知的用户。

这场争论的结果是一个被称为 IPSec(IP security, IP 安全)的设计方案,RFC 2401、2402 和 2406 等一些 RFC 文档描述了 IPSec。然而,并不是所有的用户都想要加密功能(因为加密操作的计算代价较高)。IPSec 并不是将加密功能做成可选项,相反,它总是要求加密功能,但是允许使用空算法(null algorithm)。空算法因其简单性、易于实现和极高的速度而获得了高度赞赏,RFC 2410 描述了空算法。

完整的 IPSec 设计方案是一个多服务、多算法和多粒度的框架。IPSec 支持多种服务的原因是,并不是每个人都愿意为所有的服务、所有的时间支付费用,所以,IPSec 按照菜单点菜的方式来提供这些服务。最主要的服务是保密性、数据完整性,以及针对重放攻击(即入侵者重放一次会话过程)的保护。所有这些服务都建立在对称密钥密码学的基础上,因为高性能在这里非常关键。

IPSec 支持多个算法的理由是,一个现在被认为安全的算法在将来某个时候可能会被攻破。将 IPSec 设计成与算法无关的好处是,即使某个特殊的算法将来被打破了,这个框架仍然可以幸存下来。

IPSec 支持多种粒度的理由是,这样可使得它既能够保护单个 TCP 连接,也能够保护一对主机之间的所有流量,或者一对安全路由器之间的所有流量,以及其他一些可能性。

关于 IPSec,一个略微使人惊讶的方面是,尽管它位于 IP 层上,但它是面向连接的。实际上,这也无需惊讶,因为为了做到任何一种安全性,建立一个密钥并且在一定长的时间内使用该密钥是不可避免的——从本质上来讲,这正是一种连接。而且,在面向连接的环境中,建立连接的开销可以被分摊到许多分组中。在 IPSec 的环境中,一个“连接”被称

为一个 SA(security association,安全关联)。SA 是两个端点之间的单工连接,它有一个与之关联的安全标识符。如果两个方向上都需要安全通信的话,则要求使用两个安全关联。在这些安全连接上传递的每个分组都携带了相应的安全标识符,而且,当一个分组到达的时候,主机上的 IPSec 利用此安全标识符来查询密钥和其他有关的信息。

在技术上,IPSec 有两个主要部分。第一部分描述了两个新的头,这两个新的头被加入到分组中以便携带安全标识符、完整性控制数据和其他的信息。另一部分是 ISAKMP (Internet Security Association and Key Management Protocol,Internet 安全关联和密钥管理协议),它解决的是如何建立密钥的问题。这里我们不再进一步讨论 ISAKMP,因为(1)ISAKMP 太复杂了;(2)它的主要协议 IKE(Internet Key Exchange)有严重的缺陷,因而需要被更换(Perlman and Kaufman, 2000)。

IPSec 有两种使用模式。在传输模式(transport mode)中,IPSec 头被直接插在 IP 头的后面。IP 头中的 Protocol 域也被做了修改,以表明有一个 IPSec 头紧跟在普通 IP 头的后面(但是在 TCP 头的前面)。IPSec 头包含了安全信息,主要是 SA 标识符、一个新的序列号,可能还包括净荷数据的完整性检查信息。

在隧道模式(tunnel mode)中,整个 IP 分组,连同头部和所有的数据一起被封装到一个新的 IP 分组的数据体中,并且这个 IP 分组有一个全新的 IP 头。当隧道的终点并不是最终的目标节点时,隧道模式将非常有用。在有些情况下,隧道的终点是一台安全网关机器,例如,公司的一个防火墙。在这种模式中,当分组通过防火墙的时候,防火墙负责封装分组,或者解除封装。由于隧道终止于这台安全的机器上,所以公司 LAN 上的机器不必知晓 IPSec 的存在。只有防火墙必须要知道 IPSec。

当许多 TCP 连接被聚集起来,作为一个加密的流被处理的时候,隧道模式也非常有用,这种用法的好处是,入侵者将无法看到谁给谁发送了多少分组。有时候,仅仅知道有多少流量发送到哪里,这本身就是很有价值的信息。例如,如果在一次军事危机中,五角大楼和白宫之间的网络流量突然减少了许多,但是五角大楼与科罗拉多州洛矶山脉(Colorado Rocky Mountains)里某个军事基地之间的流量却增加了同等的数量,则入侵者就有可能从这些数据中推断出某些有用的信息。研究分组的流模式(即使是加密的数据流)被称为流量分析。隧道模式在某种程度上提供了一种应对的方法。隧道模式的缺点是,它加入了一个额外的 IP 头,因此实实在在地增加了分组的长度。与此相反,传输模式不会显著地影响分组的长度。

第一个新的头是 AH(Authentication Header,认证头)。它提供的是完整性检查和防重放安全性,而不是保密性(即没有加密数据)。在传输模式中 AH 的用法如图 8.27 所示。在 IPv4 中,AH 被插在 IP 头(包括可能有的所有选项)和 TCP 头之间;在 IPv6 中,它仅仅是另一个扩展头而已,就如同一般的扩展头那样被处理。实际上,AH 的格式与标准的 IPv6 扩展头的格式非常接近。净荷部分有可能必须被填补到某一个特殊的长度以适应认证算法的要求,如图中所示。

现在我们来讨论 AH 头。Next header(下一个头)域被用来保存原始分组的 IP Protocol 域中原来的值。原始分组中的 Protocol 域现在已被替换成 51 以表明后面紧跟着一个 AH 头。在绝大多数情况下,这里出现的是 TCP 协议代码(6)。Payload length





图 8.27 在 IPv4 的传输模式中的 IPsec 认证头

(净荷长度)域等于 AH 头部中 32 位字的个数减 2。

Security parameters index(安全参数索引)是连接标识符。它是由发送方插入进来的,用来指定接收方数据库中一条特定的记录。该记录包含了这个连接上所使用的共享密钥,以及有关该连接的其他信息。如果这个协议是由 ITU 而不是 IETF 发明的,则这个域将被称为虚电路号(virtual circuit number)。

Sequence number(序列号)域被用来对一个 SA 上发送的所有分组进行编号。每个分组都有一个唯一的编号,即使重传的分组也有唯一的编号。换句话说,重传的分组与原始的分组具有不同的序列号(尽管它们的 TCP 序列号是相同的)。这个域的目的是为了检测重传攻击。这些序列号不会发生回绕的现象,因为如果  $2^{32}$  个序列号全部用完后,IPsec 必须建立一个新的 SA 以便继续进行通信。

最后是 Authentication data(认证数据)域,这是一个可变长度的域,它包含了净荷数据的数字签名。当 SA 被建立的时候,双方协商好它们将使用哪个签名算法。通常情况下,这里不使用公开密钥密码学,因为分组的处理速度必须很快,而所有已知的公钥算法都太慢。由于 IPsec 建立在对称密钥密码学的基础上,而且发送方和接收方在建立 SA 之前协商了一个共享密钥,所以,在签名计算过程中可以利用这个共享密钥。一种简单的办法是在分组和共享密钥合起来的位串上计算散列值。当然,这个共享的密钥并不在网络上传输。像这样的方案被称为 HMAC(Hashed Message Authentication Code,散列的消息认证码)。它的计算速度比先运行 SHA-1 再在结果上运行 RSA 要快得多。

AH 头并不支持数据加密功能,所以,只有当需要完整性检查但不需要保密性的时候,AH 头才十分有用。AH 头有一个值得注意的特性是它的完整性检查也覆盖了 IP 头中的某些域,也即当分组在路由器之间被转发时不随路由器而变化的那些域。例如,Time to live(TTL)域在每一跳上都要改变,所以它不能被包含在完整性检查的范围内。然而,IP 源地址被包含在检查范围内,这就使入侵者无法伪造分组的来源信息。

另一个 IPsec 头是 ESP(Encapsulating Security Payload,封装的安全净荷)。图 8.28 显示了 ESP 用于传输模式和隧道模式的情形。

ESP 头由两个 32 位字组成。它们是 Security parameters index 和 Sequence number 域,我们在 AH 头中已经看到过这两个域了。通常跟在这两个字后面的第三个字是用于数据加密的 Initialization vector(初始向量),但是从技术上来讲,它不属于头部。然而,如



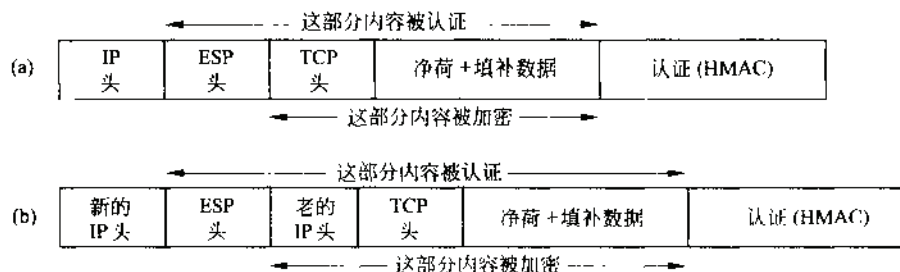


图 8.28

(a) 传输模式中的 ESP; (b) 隧道模式中的 ESP

果使用空加密算法(null encryption)的话,则这个字被省略。

如同 AH 一样,ESP 也提供了 HMAC 完整性检查,但是完整性检查部分并不是被包含在头部,而是跟在净荷之后,如图 8.28 所示。将 HMAC 放在末尾对于硬件实现是有好处的。在数据位被通过网络接口卡往外发送的过程中,同时也可以计算出 HMAC,然后再追加到尾部。这也正是为什么以太网和其他的 LAN 将它们的 CRC(循环冗余校验码)放在尾部而不是头部的原因。若使用 AH,则分组必须被缓冲起来,并且在发送分组之前首先计算出签名,这样有可能会降低每秒钟发送的分组的数量。

既然 ESP 可以完成 AH 所能做的任何事情,甚至还能做得更多,而且效率也更高,那么,一个很自然的问题是:为什么还要引入 AH 呢?这主要是由于历史的原因。最初,AH 只处理完整性,而 ESP 只处理保密性。后来,完整性也被加入到 ESP 中,但是,那些设计 AH 的人不希望在 AH 已经被用起来之后再让它死掉。然而,他们惟一真正的论据是,AH 检查 IP 头的一部分,而 ESP 不检查 IP 头,但这是一个非常弱的论据。另一个也很弱的论据是,如果一个产品支持 AH 但不支持 ESP,那么它或许可以省却有关出口许可的诸多麻烦,因为它不能够做加密操作。AH 在将来有可能被慢慢淘汰。

### 8.6.2 防火墙

能够将任何地方的任何一台计算机与另外一个地方的另一台计算机连接起来,这既有好的一面也有不好的一面。对于家庭个人用户来说,在 Internet 上漫游充满了各种乐趣。但对于公司的安全管理员来说,这是一种梦魇。大多数公司有大量的保密信息放在网络上,例如交易的秘密、产品开发计划、市场策略、财务分析,等等。将这些信息暴露给竞争对手将会带来可怕的后果。

除了信息往外泄漏的危险性以外,同时也还存在信息漏进来的危险性。尤其是,病毒、蠕虫和其他的数字有害物可能会破坏安全性、销毁有用的数据,消耗管理员的大量精力以清理它们留下的混乱局面。这些有害物通常是由于粗心的雇员在玩各种新奇的游戏时引入进来的。

因此,公司的网络需要有一些机制来让“好的”数据位进来,“坏的”数据位出去。一种方法是使用 IPSec。这种方法可以保护安全站点之间传送的数据。然而,IPSec 并不能阻止数字有害物和入侵者进入到公司的 LAN 中。为了看清楚如何实现这个目标,我们需

要看看防火墙。

**防火墙(firewall)**只不过是古代中世纪安全设施(在城堡周围挖一条很深的护城河)的现代数字改编版。在古代,这种设计强迫每个进出城堡的人都要经过一座吊桥,而他们在通过吊桥的时候,I/O 警察可以对他们进行检查。在网络环境中,同样的做法也是有可能的,公司可以按照任意的方式将多个 LAN 连接起来,但是强迫所有进出公司的流量必须通过一座电子吊桥(即防火墙),如图 8.29 所示。

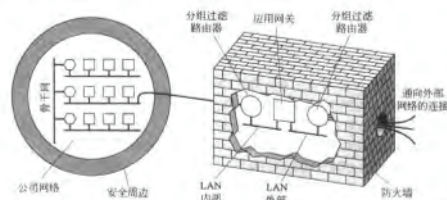


图 8.29 由两个分组过滤器和一个应用网关组成的防火墙

这种配置中的防火墙有两个部件:两台完成分组过滤任务的路由器,以及一个应用网关。更加简单的配置也是可能的,但是这种设计的优点是,每个分组无论是进来还是出去,都必须穿过两个过滤器和一个应用网关。除此以外不存在其他的路径。如果读者认为只需一个安全检查点就已经足够了,那么,显然你最近没有乘坐过固定航线上的国际航班。

每个分组过滤器(packet filter)是一个配置了某些额外功能的标准路由器。这些额外的功能使得每个进来或者出去的分组都要经过检查。只有满足某种准则的分组才被正常转发,而不满足测试条件的分组则被丢弃。

在图 8.29 中,绝有可能在 LAN 内部的分组过滤器负责检查往外发送的分组,而 LAN 外面的分组过滤器负责检查进来的分组。经过了第一道篱笆的分组还要通过应用网关以便做进一步的检查。将两个分组过滤器放在不同的 LAN 上,这样可以确保没有一个分组可以绕过应用网关而传递进来或者出去;除此以外没有别的路径。

分组过滤器通常是以数据表的形式而被驱动的,表中的信息由系统管理员来配置。这些表列出了那些可以被接受的源和目标,以及必须被阻塞的源和目标,并指定一组默认的规则来处理那些来自或者送往其他机器的分组。

在针对 TCP/IP 网络的常见情形中,所谓源或者目标往往是由一个 IP 地址和一个端口组成的。端口表明了期望指定的是哪一个服务。例如,TCP 端口 23 是 telnet,TCP 端口 79 是 finger,TCP 端口 119 则是 USENET 新闻。一家公司可能会阻塞住所有跟这三个端口中任何一个端口组合在一起的 IP 地址。通过这种方式,公司外部没有人能够通过

telnet 登录进来,或者通过 Finger 守护程序查找人员。而且,公司的雇员也不可能整天阅读 USENET 新闻了。

在阻塞往外发送的分组时需要一点技巧,因为虽然绝大多数站点坚持使用标准的端口编号惯例,但是,端口的编号并不是强制性的。而且,对于有些重要的服务,比如 FTP (File Transfer Protocol, 文件传输协议),端口号是动态分配的。另外,虽然阻塞 TCP 连接很困难,但是阻塞 UDP 分组更加困难,因为你很难预先知道它们将会做些什么。许多分组过滤器仅仅被简单地配置成禁止所有的 UDP 流量。

上述防火墙的另一半是应用网关(application gateway)。应用网关并不只是简单地检查原始的分组,而是运行在应用层上。例如,你可以配置一个邮件网关来检查每一条进来或者出去的消息。对于每一条消息,邮件网关根据头部的域、消息的长度,甚至消息的内容来决定是转发该消息,或是丢弃该消息(例如,在一个军事基地,凡是消息中出现了“核”或者“炸弹”之类的词都可能会导致采取某些特殊的措施)。

任何一个组织可以自由地安装一个或者多个针对特殊应用的应用网关,但是,较为常见的情形是,一些敏感的组织允许电子邮件的进和出,可能也允许使用 Web(World Wide Web),但是禁止其他一切可能有风险的流量。这样的应用网关配置与加密功能以及分组过滤器结合起来,可以提供一定程度的安全性,但是牺牲了一些便利性。

即使防火墙的配置极其完美,网络中仍然会存在大量的安全问题。例如,如果一个防火墙被配置成只允许接收来自某些特定网络的分组(比如,来自公司其他分部的流量),那么,防火墙外面的入侵者可以在分组中填上假的源地址,从而绕过防火墙的检查机制。如果一个内部人员想要偷取保密文档的话,他可以加密这些文档,或者先把文档拍摄下来然后以 JPEG 文件的形式将图片转运出来,这样可以绕过防火墙中的内容过滤器。我们一直没有提到,事实上,在所有的攻击事件中,有 70%来自于防火墙的内部,例如,来自于心怀不满的雇员(Schneier, 2000)。

另外,还存在许多其他类型的攻击是防火墙不能处理的。防火墙的基本思想是阻止入侵者进入,或者避免保密数据被转运出去。不幸的是,还有一些人的手段更加低劣,他们不会别的,只会想方设法把目标站点搞瘫痪。他们的做法是,向目标机器发送大量合法的分组,直至目标机器不堪重负而崩溃。例如,为了削弱一个 Web 站点,入侵者可以发送一个 TCP SYN 分组来请求建立一个连接。然后 Web 站点为这个连接分配一个内部表项,并且发送一个 SYN+ACK 分组作为应答。如果入侵者不再响应的话,则这个表项将被占用一定长的时间(比如几秒钟),直至它超时。如果入侵者在短时间内发送数千个连接请求,则所有的表项都将被占用,从而 Web 站点无法再建立起正常合法的连接。在这种攻击中,入侵者的意图是使目标机器停止服务,而不是偷取数据,因此这种攻击被称为 DoS(Denial of Service, 拒绝服务)攻击。一般情况下,请求分组中的源地址是伪造的,所以入侵者不太容易被追踪到。

一个更加糟糕的变种是,入侵者已经攻破了遍布世界各地的数百台计算机,然后他命令所有这些机器同时向同一个目标发起攻击。这种方法不仅增加了入侵者的威力,而且也减小了他被检测到的几率,因为那些攻击分组来自大量的机器,并且这些机器属于普通的可信任用户。这样的攻击被称为 DDoS(Distributed Denial of Service, 分布式拒绝服务)

攻击。这种攻击很难防护，即使被攻击的机器能够快速识别出伪造的请求，它也必须花一定的时间来处理该请求，然后再将它丢弃；如果每秒钟到达的请求是足够多的话，CPU 将不得不花所有的时间来处理这种请求。

### 8.6.3 虚拟私有网络

许多公司的办公室和部门分散在多个城市中，有时候甚至分布在多个国家。过去，在公共数据网络出现以前，对于这些公司来说，最常见的做法是从电话公司租用一些线路，以便将部分场所或者全部场所两两连接起来。现在有些公司仍然这样做。利用公司的计算机和租用的电话线路而建立起来的网络被称为私有网络。图 8.30(a)显示了一个私有网络例子，它将三个场所连接起来。

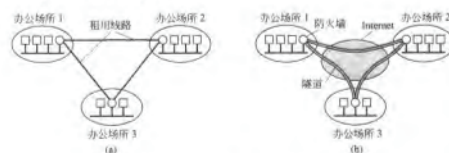


图 8.30  
(a) 通过租用线路建立起来的私有网络；(b) 一个虚拟私有网络

私有网络工作得很好，而且也非常安全。如果可用的线路仅仅是租用线路的话，则所有的流量不会泄漏到公司的各个场所以外，入侵者必须通过物理搭线的方法才能网入，而这种物理搭线的方法并不容易做到。但是，这种私有网络存在的问题是，租用一条 T1 线路的开销是每个月几千美元，而租用一条 T3 线路可能需要几倍的价格。当公共数据网络以及后来的 Internet 出现以后，许多公司希望将它们的数据流量（可能还有语音流量）转移到公共网络上，但是又不希望损失私有网络的安全性。

这种需求很快导致了 VPN (Virtual Private Networks, 虚拟私有网络) 的诞生。VPN 是建立在公共网络之上的层叠网络，但是具有私有网络的绝大多数特性。它们之所以被称为“虚拟的”，是因为它们仅仅是一个假想的网络，就好像虚电路并不是真正的电路、虚拟内存并不是真正的内存一样。

尽管 VPN 可以建立在 ATM (或者帧中继) 之上，但是，日渐流行的做法是直接建立在 Internet 上建立 VPN。一种常见的设计方案是，在每个办公场所配备一个防火墙，并且通过 Internet 在所有这些办公场所两两之间建立隧道，图 8.30(b) 显示了这种设计的例子。如果利用 IPSec 来实现隧道的话，则有可能将任何两对办公室之间的所有流量聚集到一个支持认证和加密功能的 SA 上，从而既提供了完整性控制和保密性，甚至还在很大程度上可避免遭受流量分析的攻击。

当系统启动的时候，每一对防火墙必须协商其 SA 的参数，包括所提供的服务、模式、

算法和密钥。许多防火墙有内置的 VPN 功能,不过有些普通的路由器也有 VPN 功能。但是,由于防火墙的主要任务在于提供安全服务,所以,只要公司与 Internet 之间有非常清晰的隔离边界,则让隧道起止于防火墙是非常自然的做法。因此,防火墙、VPN,以及在隧道模式中采用 ESP 的 IPSec,这三者是一种很自然的组合,目前被广泛应用在實踐中。

一旦建立起 SA,分组流量就可以开始流动了。对于 Internet 上的一台路由器而言,沿着 VPN 隧道传递的一个分组仅仅是一个普通的分组而已。对于这个分组,惟一不同寻常的是 IPSec 头出现在 IP 头的后面,但是,由于这个额外的头不影响转发过程,所以,路由器并不关心这个额外的 IPSec 头。

用这种方式来组织 VPN 的一个关键好处是,它对于所有的用户软件都是透明的。防火墙建立并管理 SA。惟一知道这种配置方案的人是系统管理员,他必须要配置和管理防火墙。对于其他人来说,这样的网络与租用线路的私有网络并无区别。有关 VPN 的更多信息,请参考 Brown, 1999; and Izzo, 2000。

#### 8.6.4 无线网络安全

利用 VPN 和防火墙来设计一个逻辑上绝对安全的系统是出奇地容易,但是,在实践中,这样的系统仍然有可能像筛子一样漏洞百出。例如,如果有些机器是无线的,它们使用无线电波进行通信,这样就会在进和出两个方向上绕过防火墙,则筛子的情形就会发生。802.11 网络的范围通常是几百米距离,所以,如果一个人想要暗中监视一家公司,那么他只需早晨开车到该公司的停车场,并把一台支持 802.11 协议的笔记本电脑留在汽车里以便记录下所有听到的内容,然后他自己离开停车场。到下午晚一点的时候,该计算机的硬盘上就充满了各种有价值的信息。在理论上,这种泄漏是不应该发生的。同样地,在理论上,人们是不应该去抢劫银行的。

大多数安全问题的根源在于无线基站(访问点)的生产商,他们总是试图使自己的产品对用户极为友好。通常情况下,如果用户将设备从包装盒中取出来并插到电源插口上,则它立即就开始运行了——它几乎总是没有任何安全性,而将秘密暴露给无线电波范围内的任何一个人。如果它后来又被插入到一个以太网网络中,则所有的以太网流量也突然出现在停车场内。无线连接使得偷窥者的梦想成为现实:无需做任何工作就可以免费获得数据。因此,毋庸多说,安全性对于无线系统比有线系统更加重要。在这一小节中,我们将看一看无线网络处理安全性的几种方法。要想获得更多的信息,请参考(Nichols and Lekkas, 2002)。

##### 802.11 安全性

802.11 标准规定了一个被称为 WEP(Wired Equivalent Privacy,相当于有线网络的保密性)的数据链路层安全协议,它的设计目标是使得无线 LAN 能够具有像有线 LAN 那样好的安全性。由于有线 LAN 的默认设置也没有任何安全性,所以这个目标很容易达到,正如我们下面将要看到的,WEP 实现了这个目标。

当 802.11 安全功能启用之后,每个无线站与基站共享一个秘密密钥。WEP 标准并



没有规定如何分发这些密钥。这些密钥可以由厂商预先安装,也可以通过有线网络提前相互交换。最后,无论基站还是用户机器都可以选取一个随机密钥,并利用对方的公钥来加密此随机密钥,然后通过空中信道发送给对方。一旦建立起共享密钥之后,它们通常保持数月或数年不变。

WEP 中的加密操作使用了一个基于 RC4 的流密码算法。RC4 是由 Ronald Rivest 设计的,它原来一直是保密的,直到 1984 年才被泄漏并张贴到 Internet 上。正如我们在前面曾经指出的那样,使算法保密几乎是不可能的,即使算法的设计目标是为了保护知识产权(这正是 RC4 的目标)而并非实现含糊的安全性(security by obscurity,这不是 RC4 的目标)。在 WEP 中,RC4 生成一个密钥流,然后这个密钥流与明文异或(XOR)在一起而形成密文。

每个分组的净荷是通过图 8.31 中所示的方法来加密的。首先利用 CRC-32 多项式对净荷部分计算校验和,再将校验和附加在净荷的尾部,两者合起来构成了加密算法的明文。然后,此明文与一个同长度的密钥流做 XOR 操作,结果得到密文。用于启动 RC4 算法的 IV(初始向量)也随着密文被一起发送出去。当接收方得到了分组以后,它从分组中提取出加密的净荷,并且利用共享的秘密密钥和刚刚得到的 IV 生成对应的密钥流,然后用密钥流与净荷做 XOR 操作,恢复出明文数据。然后它检查校验和,以确定分组是否被篡改过。

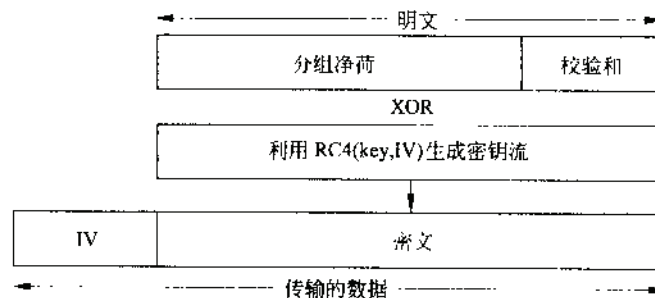


图 8.31 WEP 中的分组加密

虽然这种方法乍看之下好像很好,但是目前已经有人发表了一种能够破解它的方法(Borisov et al., 2001)。下面我们将简要地介绍他们的结果。首先,非常令人惊讶的是,许多无线网络为所有的用户使用同样的共享密钥,在这种情况下每个用户都可以读取所有其他用户的流量。这样的确做到了等同于以太网的效果,但是并不安全。

然而,即使每个用户有单独的密钥,WEP 仍然可以被攻破。由于密钥往往会在一段较长的时间内保持不变,所以,WEP 标准建议(但不是强制)每个分组都应该改变 IV,以避免遭受我们在第 8.2.3 节中讨论过的密钥流重用攻击。不幸的是,许多笔记本计算机的 802.11 接口卡在每次被插入到计算机中时,总是将 IV 重置为 0,然后在发送每个分组的时候将 IV 增加 1。由于人们经常会拔出这些接口卡,再重新插进去,所以,低 IV 值的分组非常常见。如果 Trudy 能够收集到同一个用户发送出来的具有相同 IV 值的几个分组(IV 值本身也被放在分组中,而且以明文的形式),那么她就可以计算出两个明文值的



XOR 结果,从而有可能通过这种方式破解此密码算法。

但是,即使 802.11 接口卡为每个分组选取一个随机的 IV,该 IV 也只有 24 位,所以,在发送了  $2^{24}$  个分组以后,IV 必须被重复使用。更糟的是,若使用随机选择的 IV,则根据我们在 8.4.4 节中介绍过的生日攻击理论,在同一个 IV 被使用两次之前所发送的分组的期望个数大约是 5000(即平均发送了 5000 个分组以后就会重复使用同一个 IV)。因此,如果 Trudy 听上几分钟,她几乎肯定能抓到两个使用同一 IV 和同样密钥的分组。她只要将两个密文做 XOR 操作就可以得到两个明文的 XOR 值。利用各种方法对这个位序列进行攻击可恢复出两个明文来。更进一步,Trudy 还可以得到针对此 IV 的密钥流。Trudy 只要有耐心,按照同样的方法继续做一段时间的工作,就可以构造出一个字典,其中包含了针对各个 IV 的密钥流。一旦某一个 IV 已经被破解了,则所有采用此 IV 的分组,无论是将来的还是过去的,也彻底可被解密出来。

而且,由于 IV 是随机选用的,所以,一旦 Trudy 已经确定了一对有效的(IV, 密钥流),则她就可以用这个 IV 和密钥流来生成所有她想要的分组,从而主动地干扰通信过程。在理论上,接收方可能会注意到突然之间有大量的分组使用同样的 IV,但是(1)WEP 允许这种情形,而且(2)没有人会做这种检查。

最后,CRC 并没有多大价值,因为对于 Trudy 来说,她有可能在改变净荷的同时对 CRC 也做相应的改变,甚至无需去掉加密就可以做到这一点。简而言之,攻破 802.11 的安全性是非常简单的,这里我们甚至还没有列出 Borisov 等人发现的所有攻击。

2001 年 8 月,在 Borisov 等人的论文发表之后 1 个月,针对 WEP 的另一种毁灭性的攻击也被发表出来(Fluhrer et al., 2001)。这种攻击找到了 RC4 本身的密码学上的弱点。Fluhrer 等人发现,许多密钥具有这样的特性:从密钥流有可能推导出某些密钥位。如果这种攻击被反复执行的话,只要经过适当程度的努力,就可能得到完整的密钥。Fluhrer 等人只是从理论上分析了这种攻击,而并没有试图在实践中攻破哪一个 802.11 LAN。

相反,当 AT&T Labs 的一个暑期实习生和两名研究人员知道了 Fluhrer 等人的攻击之后,他们决定在实践中试一试(Stubblefield et al., 2002)。在一周之内,他们在一个生产性的 802.11 LAN 上破解了第一个 128 位密钥,而且在这一周中,他们的大部分时间实际上是在努力寻找最便宜的 802.11 接口卡、申请购买接口卡,然后安装和测试接口卡。真正的程序运行过程只花了两个小时。

当他们宣布他们的结果时,CNN 发表了一篇题为“Off-the-Shell Hack Breaks Wireless Encryption”的文章,在这篇文章中,有些工业界的资深人士极力嘲笑他们的结果,他们认为既然有了 Fluhrer 等人的结果,则 Stubblefield 等人所做的工作并没有多大价值。虽然从技术上来说这种说法是正确的,但事实是,这两个小组所做的工作合起来证明了在 WEP 和 802.11 中存在一个致命的缺陷。

2001 年 9 月 7 日,IEEE 对 WEP 已完全被攻破的事实做出了响应,他们发表了一个简短的声明,这份声明的内容可以被粗略地归纳成以下 6 点:

- (1) 我们告诉你,WEP 的安全性并不比以太网的安全性更好。
- (2) 一个更大的威胁是根本忘记了要启动安全功能。

(3) 请尝试使用其他某种安全设施(比如传输层安全性)。

(4) 下一个版本 802.11i 将会有更好的安全性。

(5) 将来的合格资质将强制要求使用 802.11i。

(6) 我们将试图找到在 802.11i 到来之前的解决办法。

我们已经较为详细地介绍了有关 802.11 安全性的故事,你可以看到,要做到正确的安全性并不容易,即使对于专家也不例外。

### 蓝牙安全性

蓝牙(Bluetooth)的无线范围比 802.11 要短得多,所以它不可能受到来自停车场的攻击,但是,在蓝牙中,安全性仍然是一个问题。例如,想象 Alice 的计算机配备了一个无线蓝牙键盘。在缺少安全性的情况下,如果 Trudy 碰巧在邻近的办公室中,那么她就可以读取到 Alice 敲入的所有信息,包括她往外发送的所有电子邮件。她也可以捕获到 Alice 的计算机发送给不远处的蓝牙打印机的所有信息(比如 Alice 接收到的电子邮件或者机密报告)。幸运的是,蓝牙有一个非常精致的安全方案,它试图让全世界的 Trudy 们的攻击不能得逞。现在我们来概要地描述一下这个安全方案的主要特性。

蓝牙有三种安全模式,从“没有任何安全性”到“完全的数据加密和完整性控制”。如同 802.11 一样,如果安全功能被禁止的话(默认模式),则没有任何安全性。大多数用户关闭了安全功能;直到发生了严重的安全事件以后他们才将安全功能打开。在农业社会中,这种做法相当于等到马跑掉以后,才想起来要锁上马房的门。

蓝牙在多个层上提供了安全性。在物理层上,跳频机制提供了一定程度的安全性,但是由于任何一个蓝牙设备在进入到一个微微网的时候,它必须要被告知跳频序列,所以这个序列很显然不是一个秘密。当一个新到来的从节点向主节点请求一个信道的时候,真正的安全性开始了。假设这两个设备预先已经建立了一个共享的秘密密钥。在有些情况下,两个设备是由同一个生产商搭配在一起的(比如,头戴耳机和移动电话合起来出售)。在其他情况下,一个设备(比如头戴耳机)有一个硬置(hardwired)的密钥,用户必须将这个密钥(十进制数)输入到另一个设备中(比如移动电话)。这些共享的密钥被称为 **passkey**(总密钥)。

为了建立一条信道,从节点和主节点都要检查对方,看它是否知道 **passkey**。如果知道的话,则它们进行协商,以确定该信道是否要加密、是否要完整性控制,或者两者都要。然后它们选择一个随机的 128 位会话密钥,其中有些位可能是公开的。之所以允许弱化这个密钥,是为了遵从不同国家的政府限制,从而避免出口或者使用超出政府破解能力的密钥长度。

蓝牙的加密操作使用了一个被称为 **E<sub>0</sub>** 的流密码;完整性控制使用 **SAFER+**。两者都是传统的对称密钥块密码算法。**SAFER+** 曾经参加了 AES 算法竞赛,但是第一轮就被淘汰了,因为它比其他的候选算法都要慢。蓝牙是在 AES 密码被选定之前就已经定案了,否则的话,它极有可能会使用 Rijndael。

使用流密码的实际加密过程如图 8.14 所示,其中明文与密钥流做 XOR 操作以生成密文。不幸的是,**E<sub>0</sub>** 本身(就像 RC4 一样)可能有致命的弱点(Jakobsson and Wetzel,

2001)。虽然在本书写作的时候它还没有被攻破,但是它与 A5/1 密码(它的失败曾经轰动一时,也危及到了所有的 GSM 电话流量)之间的相似之处吸引了人们的关注(Biryukov et al., 2000)。有时候人们(包括我在内)真的很惊讶,在密码编码学家和密码分析家之间长期的“猫与老鼠”的游戏中,密码分析家总是赢家。

另一个安全问题是,蓝牙只认证设备,而不认证用户,所以,蓝牙设备被偷之后,窃贼有可能访问到用户的金融账户或者其他账户。然而,蓝牙也在上面的层中实现了安全性,所以即使链路层的安全性被突破,其他的安全性仍然存在,尤其是对于那些要求从某种键盘上手工输入 PIN 码才能完成交易的应用来说更是如此。

### WAP 2.0 安全性

在很大程度上,WAP 论坛(WAP Forum)从 WAP 1.0 采用了一个非标准的协议栈的事例中得到了深刻的教训。WAP 2.0 在所有的层上主要使用标准的协议。安全性也不例外。由于它是基于 IP 的,所以它在网络层上完全支持使用 IPSec。在传输层上,TCP 连接可以由 TLS 来保护,TLS 是一个 IETF 标准,本章后面我们将会学习到。再往高层,WAP 2.0 使用了 HTTP 客户认证(RFC 2617 定义了这种安全性)。应用层的密码库提供了完整性控制和不可否认性。总而言之,由于 WAP 2.0 建立在一些非常知名的标准的基础上,所以,它的安全服务(特别是隐私、认证、完整性控制和不可否认性)就有可能比 802.11 和蓝牙更好。

## 8.7 认证协议

认证(Authentication)是指这样的一项技术:一个进程通过认证过程来验证它的通信对方是否是它所期望的实体而不是假冒者。在面对一个恶意的主动入侵者的情况下,要验证远程进程的身份是极其困难的,它要求使用一些基于密码学理论的复杂协议。在本节中,我们将学习许多可被用在不安全的计算机网络中的认证协议。

顺便提一下,有些人可能会混淆授权(authorization)和认证(authentication)这两个概念。认证所针对的问题是,你是否真的在跟一个特定的进程进行通信。而授权关注的是,允许这个进程做什么样的事情。例如,一个客户进程与文件服务器建立联系,并且说:我是 Scott 的进程,我想删除文件 cookbook.old。从文件服务器的角度来看,有两个问题是它必须要回答的:

- (1) 这真的是 Scott 的进程吗(认证)?
- (2) Scott 是否有权删除 cookbook.old(授权)?

只有当这两个问题都被明确地肯定以后,客户所请求的动作才可以执行。前一个问题实际上是一个非常关键的问题。一旦服务器知道了它在跟谁通话,那么,检查授权是非常容易做到的,它只要在本地的数据表或者数据库中查找一些记录即可。由于这个原因,我们在本节中把注意力集中在认证上。

所有的认证协议都使用一个通用的模型,如下所述。Alice 首先发起认证过程,她给 Bob 或者可信的 KDC(Key Distribution Center,密钥分发中心)发送一条消息。假设这里

的 KDC 是诚实可靠的。接下来在各个方向上继续交换其他一些消息。当这些消息被发送出去以后,Trudy 可能截取、修改或者重放这些消息以便欺骗 Alice 和 Bob,或者纯粹捣乱他们的工作。

然而,当协议完成的时候,Alice 确信她是在跟 Bob 通话,而 Bob 也确信他是在跟 Alice 通话。而且,在绝大多数协议中,两者也将建立起一个秘密的会话密钥,以便用于接下来的会话过程。在实践中,出于性能的原因,所有的数据流量都是用对称密钥密码算法(通常为 DES 和 AES)来加密的,不过,公开密钥密码算法被广泛应用于认证协议本身,同时也被用于建立会话密钥。

为每个新的连接使用一个新的随机选取的会话密钥,这样做的好处是,使得利用用户的秘密密钥或者公钥来发送的流量降低到最少,从而也减少了入侵者可能得到的密文数量,而且,如果进程崩溃或者核心转储落入敌手的话也可以将损失降低到最小。期望达到的效果是,发生意外事件后惟一被暴露的密钥是会话密钥。当会话建立起来以后,所有的永久密钥应该谨慎地退出通信过程。

### 8.7.1 基于共享秘密密钥的认证

在第一个认证协议中,我们假设 Alice 和 Bob 已经共享了一个秘密密钥  $K_{AB}$ 。这个秘密密钥有可能是通过电话或者面对面地建立起来的,但无论如何,肯定不是在(不安全的)网络上建立的。

这个协议以下面的一个原理为基础:一方给另一方发送一个随机数,然后后者将这个随机数做一个特殊的变换,再把结果返回给前者。除了本小节的协议以外,在许多其他的认证协议中也都可以找到这个原理的应用。这样的协议被称为质询-回应(challenge-response)协议。在本小节的协议以及后续的认证协议中,我们将使用下面的记号:

A,B 是 Alice 和 Bob 的标识。

$R_i$  是质询,其中下标指明了发起质询的一方。

$K_i$  是密钥,这里  $i$  代表密钥的所有者。

$K_s$  是会话密钥。

在第一个共享密钥的认证协议中,其消息序列如图 8.32 所示。在消息 1 中,Alice 以一种 Bob 能理解的方式将她的标识发送给 Bob。当然,Bob 无法判断这条消息是来自 Alice 还是 Trudy,所以他选择一个质询,即一个大的随机数  $R_B$ ,并且将它以明文方式送

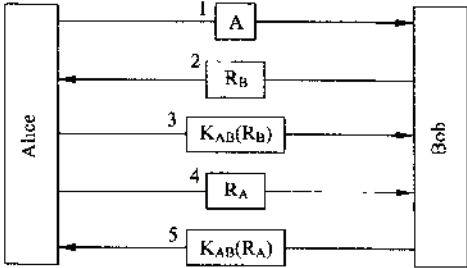


图 8.32 使用质询-回应协议的双向认证过程

回给“Alice”，如图中的消息 2。在这样在质询-回应协议中，这种仅仅使用一次的随机数被称为临时值(**nonce**)。然后 Alice 用她与 Bob 共享的密钥加密此消息，并且在消息 3 中将密文  $K_{AB}(R_B)$  送回来。当 Bob 看到这条消息的时候，他立即知道这条消息来自于 Alice，因为 Trudy 并不知道  $K_{AB}$ ，因此她不可能生成此消息。而且，由于  $R_B$  是在一个很大的空间中被随机选取的(比如说是一个 128 位的随机数)，所以 Trudy 不太可能曾经在以前的会话中看到过  $R_B$  和相应的回应。同样地，她也不可能猜测到任何一个质询随机数所对应的正确回应。

到这时候，Bob 已经确信与他通话的是 Alice，但是 Alice 还不能确定对方的身份。Alice 知道，Trudy 可能已经截取了消息 1 并送回了  $R_B$  作为回应。也许 Bob 昨天晚上已经死了。为了搞清楚与她通话的到底是谁，Alice 选取了一个随机数  $R_A$ ，并以明文方式送给 Bob，见图中的消息 4。当 Bob 以  $K_{AB}(R_A)$  作为回应的时候，Alice 知道她在跟 Bob 通话。如果他们希望现在建立一个会话密钥，则 Alice 可以选取一个  $K_s$ ，并且用  $K_{AB}$  加密之后发送给 Bob 即可。

图 8.32 中的协议包含了 5 条消息，我们来看一看是否可以让它更为精简一些，以便省下几条消息。图 8.33 显示了一种做法。在这里，Alice 主动发起了质询-回应协议，而不是等 Bob 来发起质询。类似地，当 Bob 回应 Alice 的质询时，他也发送自己的质询消息。整个协议可以被减少为 3 条消息，而不再是 5 条消息。

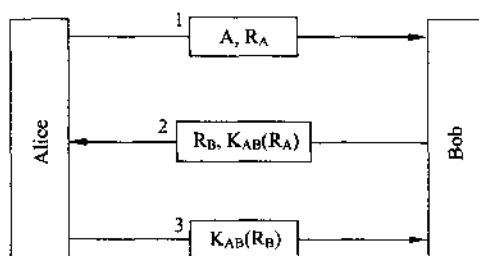


图 8.33 缩减之后的双向认证协议

这个新的协议是否比原来的协议有所改进呢？从某种意义上讲，确实是：新协议更短了。不幸的是，它也是错误的。在特定的条件下，Trudy 利用一种被称为反射攻击(**reflection attack**)的技术可以使该协议失败。尤其是，如果 Trudy 能够与 Bob 一次打开多个会话的话，则 Trudy 就可以攻破这个协议。这种条件是有可能成立的，比如，Bob 是一家银行，他随时准备接受来自柜台机器的多个并发连接。

Trudy 的反射攻击如图 8.34 所示。首先，Trudy 声称她是 Alice，并且发送  $R_T$ ，然后 Bob 如同往常一样，用他自己的质询  $R_B$  作为回应。现在 Trudy 被定住了，她该怎么办呢？她并不知道  $K_{AB}(R_B)$ 。

她可以利用消息 3 打开第二个会话，并且用消息 2 中的  $R_B$  作为她的质询。Bob 平静地对  $R_B$  进行加密，并在消息 4 中送回  $K_{AB}(R_B)$ 。我们为第二个会话上的消息加了阴影，以便突出显示它们。现在，Trudy 有了缺少的信息，所以她可以完成第一个会话，并放弃第二个会话。Bob 现在相信 Trudy 就是 Alice，所以当她想请求查询 Alice 的银行账户余额



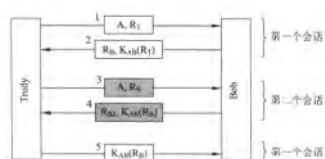


图 8.34 反射攻击

时, Bob 毫无怀疑地告诉了她。然后, 她请求 Bob 将所有的余额转到瑞士银行的一个秘密账户中, 他也毫不犹豫地照办不误。

这个故事的寓意是:

设计一个正确的认证协议要比表面看上去的复杂得多。

下面 4 条一般性的规则通常会有所帮助:

(1) 让发起方首先证明自己是谁, 然后再轮到应答方。在上面的例子中, 在 Trudy 给出证据来证明自己是 Bob 之前 Bob 就给出了有价值的信息。

(2) 让发起方和应答方使用不同的密钥作为证明, 不过, 这意味着要使用两个共享密钥  $K_{AB}$  和  $K'_{AB}$ 。

(3) 让发起方和应答方从不同的集合中选取质询随机数。例如, 发起方必须使用偶数, 而应答方必须使用奇数。

(4) 使协议能够抵抗这种牵扯到第二个并行会话的攻击, 比如像上面给出的反射攻击那样, 在一个会话中得到的信息可以用在另一个会话中。

只要以上规则中有一条被违反了, 则协议通常就会被攻破。在前面的例子中, 所有 4 条都违反了, 所以带来了灾难性的后果。

现在我们回去再仔细看一看图 8.32。这个协议真的不会遭受反射攻击吗? 难说, 那要看情况了。它很微妙。Trudy 能够用反射攻击来挫败我们的协议是因为她有可能打开第二个与 Bob 之间的会话, 然后诱使他自己来回答自己的问题。如果 Alice 是一台能够接受多个会话的通用计算机, 而不是计算机前面的人, 那会怎么样呢? 现在让我们看一看 Trudy 能做什么。

为了看清楚 Trudy 的攻击是如何实施的, 请参考图 8.35。Alice 首先在消息 1 中宣告自己的标识。Trudy 截取了这条消息, 并开始她自己的会话, 声称自己是 Bob, 如图中的消息 2 所示。同样地, 我们用阴影来表示第二个会话的消息。Alice 在消息 3 中间应了消息 2, 她这样说: 你自称是 Bob? 请证明这一点。这时候, Trudy 被定住了, 因为她无法证明自己是 Bob。

现在 Trudy 该怎么办呢? 她又回到第一个会话中, 在这里, 该轮到她发送一个质询, 于是, 她把在消息 3 中得到的  $R_A$  发送出去。Alice 非常友好地做了回应, 即消息 5, 从而给 Trudy 提供了她所急需的信息, 所以, Trudy 发送了消息 6, 此消息属于会话 2。此时,



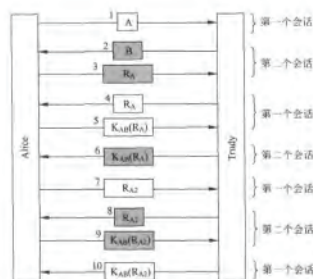


图 8.35 针对图 8.32 中的协议的反射攻击

Trudy 基本上畅行无阻了，因为在会话 2 中，她已经成功地回应了 Alice 的质询。现在她可以取消会话 1 了，对于会话 2 的剩余部分，她发送任何一个老的数值，因此，她获得了一个与 Alice 之间经过认证的会话，即会话 2。

但是 Trudy 比我们想象的还要恶劣，她想彻底地攻进去并且不留任何异常。她并不是马上发送一个老的数值来完成会话 2，而是等待 Alice 发送消息 7，即会话 1 中 Alice 的质询随机数。当然，Trudy 不知道该如何回应，所以她再次利用反射攻击，送回  $R_{A2}$ ，即消息 8。Alice 很自然地加密了  $R_{A2}$ ，并通过消息 9 送给 Trudy。Trudy 现在切换到会话 1，并且在消息 10 中把 Alice 想要的加密结果送给她，实际上她只是将 Alice 在消息 9 中发送过来的加密结果复制回去而已。到这时候，Trudy 与 Alice 之间有了两个完全经过认证的会话。

与图 8.34 中显示的针对三消息协议的攻击相比较，上面的攻击有一个略微不同的结果。这一次，Trudy 与 Alice 之间有了两个经过认证的连接。而在前一个例子中，她与 Bob 之间只有一个经过认证的连接。同样在这里，如果我们应用了前面讨论过的 4 条一般性的认证协议规则，那么，这种攻击就可以被制止。有关这些攻击以及如何阻止这些攻击的详细讨论，请参考 (Bird et al., 1993)。他们也证明了：用系统化的方法来构造出可被证明是正确的协议，这是有可能的。然而，对于这样的协议，即使是最简单的形式也较为复杂，所以，我们现在来介绍他们提出的另外一种类型的协议。

新的认证协议如图 8.36 所示 (Bird et al., 1993)。它使用了 HMAC (关于 HMAC，我们曾经在学习 IPsec 的时候看到过)。Alice 首先给 Bob 发送一个临时值  $R_A$  作为消息 1，Bob 的回应是：选择他自己的临时值  $R_B$ ，并连同一个 HMAC 一起发送回去。这里的 HMAC 是这样形成的：首先建立一个数据结构，其中包含了 Alice 的临时值、Bob 的临时值、他们的标识，以及共享的密钥  $K_{AB}$ ，然后将这个数据结构做散列 (hash) 运算，比如

使用 SHA-1 算法,散列的结果即为 HMAC。当 Alice 接收到消息 2 时,她现在拥有  $R_A$  (这是她自己选取的)、 $R_B$  (这是以明文形式到来的)、双方的标识,以及秘密密钥  $K_{AB}$  (这是她一直知道的),所以她自己也可以计算出 HMAC。如果她计算的 HMAC 与消息中的 HMAC 一致,那么她知道她在与 Bob 通话,因为 Trudy 不知道  $K_{AB}$ ,因此 Trudy 不可能计算出该发送哪一个 HMAC。Alice 回应给 Bob 的也是一个 HMAC,但是此 HMAC 仅仅包含两个临时值和  $K_{AB}$ 。

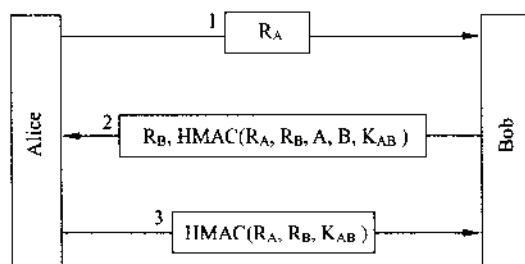


图 8.36 使用 HMAC 进行认证

那么,Trudy 能够破坏这个协议吗?不能,因为她不可能像图 8.34 和 8.35 中那样,强迫任何一方按照她的选择来加密或者散列一个值。这里的两个 HMAC 中包含了双方选择的值,其中部分值是 Trudy 所无法控制的。

采用 HMAC 并不是体现这种思想的惟一做法。除了在一组数据项上计算 HMAC 这种做法以外,另一种常用的方案是利用密码块链接(cipher block chaining)模式顺序地加密所有的数据项。

### 8.7.2 建立一个共享密钥: Diffie-Hellman 密钥交换协议

到现在为止,我们一直假定 Alice 和 Bob 共享一个秘密密钥。现在假设他们之间并没有共享密钥(因为到目前为止还没有一个被普遍接受的 PKI 可用于签名和分发证书)。他们如何才能建立一个共享密钥呢?一种办法是,Alice 打电话给 Bob,并且通过电话将她的密钥告诉 Bob,但是 Bob 可能一上来就这样说:我怎么知道你是 Alice 而不是 Trudy 呢?他们可能会试着安排一次会面,每个人都带上护照、驾驶证和三个主要的信用卡,但如果他们都很忙碌的话,则他们有可能几个月都找不到一个双方都能接受的会面日期。幸运的是,虽然听起来有点难以置信,但是却存在一种办法可以让完全陌生的人在完全公开的情况下建立起一个共享的秘密密钥,即使 Trudy 小心地记录下每一条消息也无妨。

允许陌生人建立共享秘密密钥的协议被称为 Diffie-Hellman 密钥交换协议(Diffie and Hellman, 1976),其工作过程如下所述。Alice 和 Bob 必须就两个大数  $n$  和  $g$  达成一致,这里  $n$  是一个素数,  $(n-1)/2$  也是一个素数,并且  $g$  需要满足一些特殊的条件。这些数可以是公开的,所以,他们两人中的任何一个选取  $n$  和  $g$ ,并且通过公开的方式告诉另一个人。现在 Alice 选择一个大数(比如说 512 位) $x$ ,并将它保密;同样地,Bob 也秘密地选择一个大数  $y$ 。

Alice 发起密钥交换协议,她给 Bob 发送一条消息,其中包含了 $(n, g, g^x \bmod n)$ ,如图 8.37 所示。Bob 给 Alice 发送一条包含了 $(g^y \bmod n)$ 的消息作为应答。现在,Alice 把 Bob 发送给自己的数计算  $x$  次乘方并且以  $n$  为模,得到 $(g^y \bmod n)^x \bmod n$ 。Bob 也执行类似的计算,得到 $(g^x \bmod n)^y \bmod n$ 。根据模算术定理,双方的计算结果都是 $g^{xy} \bmod n$ 。你瞧,Alice 和 Bob 突然之间共享了一个秘密密钥,即 $g^{xy} \bmod n$ 。

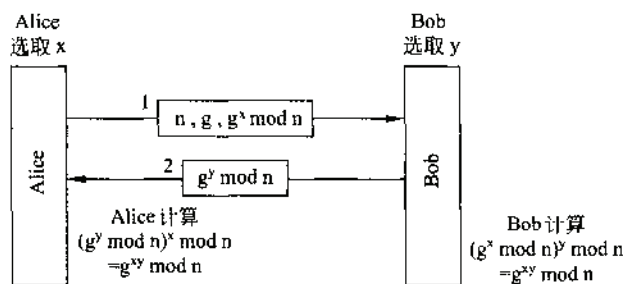


图 8.37 Diffie-Hellman 密码交换协议

Trudy 当然也看到了这两条消息。她从消息 1 中知道了  $g$  和  $n$ 。如果她能够计算出  $x$  和  $y$ ,那么,她就可以得到秘密密钥。问题在于,仅仅知道了 $g^x \bmod n$ ,她不可能找到  $x$ 。在以一个非常大的素数为模的情况下,计算离散对数的实用算法目前尚未发现。

为了使上面的例子更加具体化,我们将使用  $n=47$  和  $g=3$ (这些值完全是不切实际的)来演算一遍。Alice 选取  $x=8$ ,Bob 选取  $y=10$ 。这两个数值都是保密的。Alice 发给 Bob 的消息是 $(47, 3, 28)$ ,因为 $3^8 \bmod 47$ 是 28。Bob 发给 Alice 的消息是 $(17)$ 。Alice 计算 $17^8 \bmod 47$ ,等于 4。Bob 计算 $28^{10} \bmod 47$ ,也等于 4。所以,Alice 和 Bob 现在单独地确定了秘密密钥为 4。Trudy 必须要解方程式: $3^x \bmod 47=28$ ,对于像这里的小数值,利用穷举搜索可以解出  $x$  的值,但是当所有这些数值都是几百位长的时候,这种做法就行不通了。所有当前已知的算法都需要太长的时间才能解出  $x$ ,即使在大型的并行超级计算机上也是如此。

尽管 Diffie-Hellman 算法非常优美,但是它也有一个问题:当 Bob 得到了三元组 $(47, 3, 28)$ 的时候,他怎么知道该消息来自 Alice 而不是 Trudy? 他没办法判断这一点。不幸的是,Trudy 正好可以利用这个事实来同时欺骗 Alice 和 Bob,过程如图 8.38 所示。这里,虽然 Alice 和 Bob 分别选择了  $x$  和  $y$ ,但是,Trudy 也选取了她自己的随机数  $z$ 。Alice 发送消息 1 给 Bob,但是 Trudy 截获了此消息,并且发送消息 2 给 Bob;在消息 2 中,Trudy 使用了正确的  $g$  和  $n$ (不管怎么样,这两个数都是公开的),但是她用自己的  $z$  来代替  $x$ 。她也将消息 3 送回给 Alice。后来,当 Bob 给 Alice 发送消息 4 时,Trudy 再次截获此消息,并保留不转发。

现在,每个人都做模算术运算。Alice 计算出秘密密钥为 $g^x \bmod n$ ,Trudy 也一样(针对所有发送给 Alice 的消息)。Bob 计算出 $g^y \bmod n$ ,Trudy 也做同样的计算(针对所有发送给 Bob 的消息)。Alice 认为她在跟 Bob 通话,所以她建立一个会话密钥(实际上,与 Trudy 共享此会话密钥)。Bob 也如此。Alice 在此加密会话上发送的所有消息都被

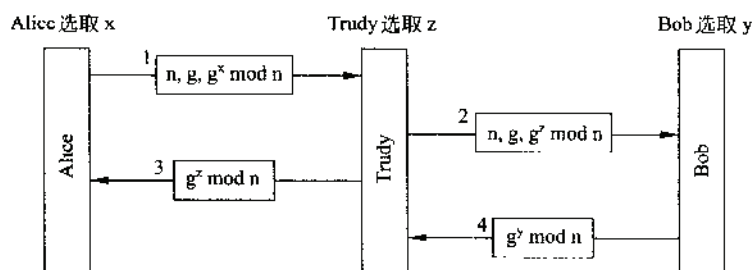


图 8.38 水桶队列攻击或者中间人攻击

Trudy 捕获到,并存储下来,如果愿意的话做适当修改,然后(可选地)传递给 Bob。在另一个方向上也类似。Trudy 看到了所有的秘密,而且可以随意地改动所有的消息,而 Alice 和 Bob 都错误地认为自己与对方有一条安全的信道。这种攻击被称为**水桶队列攻击(bucket brigade attack)**,因为它有点类似于以前的义务消防队的做法,他们从救火车到着火点排成一队传递水桶。这种攻击也被称为**中间人攻击(man-in-the-middle attack)**。

### 8.7.3 使用密钥分发中心的认证协议

与陌生人建立一个共享的秘密差不多是可以工作的,但是并不完全解决问题。另一方面,有可能从一开始就不值得这么做(酸葡萄攻击)。若通过这种方式与  $n$  个人进行通话,那么你将需要  $n$  个密钥。对于普通人来说,密钥管理将变成一份实实在在的负担,尤其当每个密钥都必须被存储在一个单独的塑料芯片卡上的时候。

另一种不同的方法是引入一个可信的密钥分发中心(KDC)。在这种模型中,每个用户与 KDC 共享一个密钥。现在认证和会话密钥管理都通过 KDC 来完成。图 8.39 显示了目前已知的最简单的 KDC 认证协议,它涉及到两方和一个可信的 KDC。

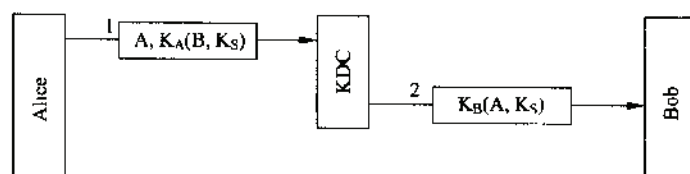


图 8.39 第一个简单的基于 KDC 的认证协议

这个协议背后的思想非常简单: Alice 选取一个会话密钥  $K_S$ , 并且告诉 KDC 她希望利用  $K_S$  与 Bob 进行通话。这条消息用 Alice 与 KDC 之间共享的秘密密钥  $K_A$  (只有 Alice 和 KDC 才知道)来加密。KDC 解密此消息,并提取出 Bob 的标识和会话密钥。然后它构造一条新的消息,其中包含了 Alice 的标识和会话密钥,并且它将这条消息发送给 Bob。这次用 Bob 与 KDC 之间共享的秘密密钥  $K_B$  来加密。当 Bob 解密出该消息时,他就知道 Alice 想与他通话,并且她希望使用  $K_S$  作为会话密钥。

这里的认证是自然而然的。KDC 知道消息 1 一定来自 Alice,因为其他人不可能用

Alice 的秘密密钥来加密此消息。类似地, Bob 知道消息 2 一定来自于 KDC(而 Bob 对 KDC 是信任的), 因为其他没有人知道他的秘密密钥。

不幸的是, 这个协议有一个严重的缺陷。Trudy 需要一些钱, 所以, 她了解到自己可以帮助 Alice 处理一些合法的业务, 于是她向 Alice 提出优厚的条件, 最后得到了这份工作。在完成了工作以后, Trudy 礼貌地请求 Alice 通过银行转账来支付费用。然后 Alice 与她的银行服务员 Bob 建立一个会话密钥。之后她给 Bob 发送一条消息, 请求将一笔钱转到 Trudy 的账户中。

同时, Trudy 又故伎重演, 她在网络上窃听流量, 并且将图 8.39 中的消息 2 和随后的转账请求消息复制下来。后来, 她又把这两条消息重新发送给 Bob。Bob 收到这两条重放消息后, 他会这样想: Alice 一定又雇佣了 Trudy, 很显然 Trudy 干得不错。于是, Bob 又一次将同样数额的一笔钱从 Alice 的账户转到 Trudy 的账户上。经过了 50 次这样的消息重放以后, Bob 走出办公室并找到 Trudy, 表示愿意为她提供一大笔贷款, 以便 Trudy 能够拓展她那看起来显然很成功的业务。这个问题被称为**重放攻击(replay attack)**。

针对重放攻击, 有几种可能的解决方案。第一种方案是, 在每一条消息中包含一个时间戳。之后, 如果有人接收到一条过期的消息, 则丢弃此消息即可。这种做法的麻烦之处在于, 一个网络上的时钟从来都不是精确同步的, 所以, 总是存在某一个时间间隔使得在此期间时间戳是有效的。因此, Trudy 就可以在这段时间间隔之中重放消息, 从而逃脱时间戳检查。

第二种方案是在每条消息之中放置一个临时值。然后, 每一方必须要记住所有此前已经出现过的临时值, 如果一条消息包含了以前用过的临时值, 则拒绝该消息。但是, 这些临时值必须被永久地记录下来, 以免 Trudy 可能会试图重放 5 年前的消息。而且, 如果某台机器崩溃之后丢失了所有的临时值信息, 那么, 它又会再次遭受重放攻击。我们可以将时间戳和临时值结合起来, 以便缩短需要记录临时值的时间范围, 但是很显然, 这将会导致协议变得非常复杂。

一种较为复杂的实现双向认证的做法是使用多路径的质询-回应协议。这种协议的一个著名的例子是 **Needham-Schroeder 认证协议**(Needham and Schroeder, 1978), 图 8.40 显示了它的一种变化形式。

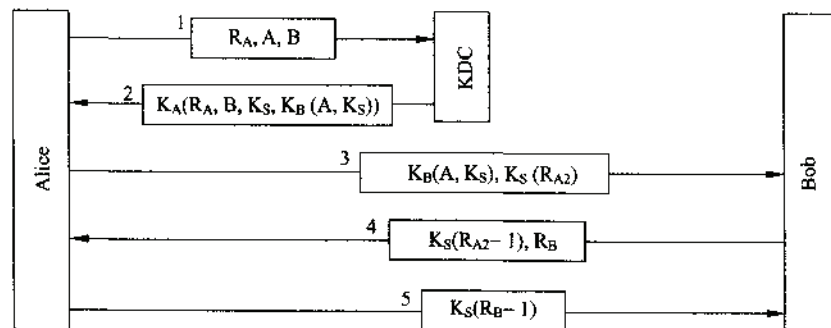


图 8.40 Needham-Schroeder 认证协议

作为协议的开始, Alice 首先告诉 KDC 她想与 Bob 通话。这条消息包含一个大的随机数  $R_A$  作为临时值。KDC 送回消息 2, 其中包含了 Alice 的随机数、一个会话密钥, 以及一个她将来可以发送给 Bob 的票据。这里使用随机数  $R_A$  是为了使 Alice 相信消息 2 是最新的, 而不是重放消息。Bob 的标识也被包含在消息中, 这是为了防止 Trudy 用她自己的标识来替换消息 1 中的 B, 从而导致 KDC 用  $K_T$  而不是  $K_B$  来加密消息 2 末尾的票据; 经  $K_B$  加密的票据也被包含在加密的消息中, 这是为了防止在返回到 Alice 的途中, Trudy 用其他的数据来偷换此票据。

Alice 现在将票据以及一个用会话密钥  $K_S$  加密的新随机数  $R_{A2}$  发送给 Bob。在消息 4 中, Bob 送回  $K_S(R_{A2}-1)$ , 以此向 Alice 证明, 她正在跟真正的 Bob 通话。如果直接送回  $K_S(R_{A2})$  是没有用的, 因为 Trudy 可以直接从消息 3 中偷到  $K_S(R_{A2})$ 。

Alice 在接收到消息 4 以后, 她现在确信自己正在跟 Bob 通话, 而且到现在为止没有任何重放消息。毕竟她仅仅在几毫秒之前才产生了  $R_{A2}$ 。消息 5 的用途是为了使 Bob 确信, 现在与他通话的是真正的 Alice, 而且这里也没有出现任何重放消息。在这个协议中, 每一方都生成一个质询, 并且回应一个质询, 因而消除了任何一种重放攻击的可能性。

尽管这个协议看起来似乎非常坚固, 但它确实有一个微小的弱点。如果 Trudy 曾经设法获得过一个老的会话密钥的明文, 那么她把原来与泄漏的密钥相对应的消息 3 重放给 Bob, 并使 Bob 相信她是真正的 Alice, 从而向 Bob 发起一个新的会话。现在, 她甚至一次都不用为 Alice 提供合法的业务就可以抢劫她的银行账户了。

Needham 和 Schroeder 后来发表的一个协议改正了这个问题 (Needham and Schroeder, 1987)。在同一份杂志的同一期上, Otway 和 Rees (1987) 也发表了一个更加短小的协议, 它同样解决了这个问题。图 8.41 显示了一个略作修改之后的 Otway-Rees 协议。

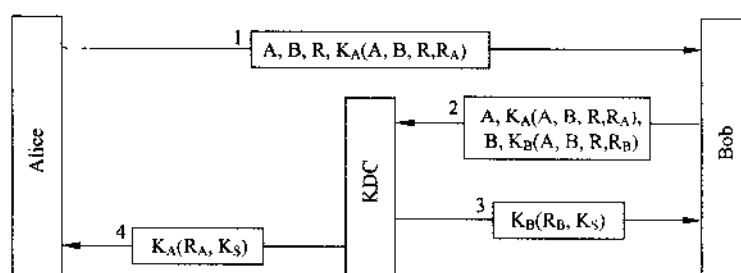


图 8.41 Otway-Rees 认证协议 (略有简化)

在 Otway-Rees 协议中, Alice 首先生成一对随机数  $R$  和  $R_A$ , 这里  $R$  被用作一个公共的标识符, 而  $R_A$  则是 Alice 质询 Bob 的随机数。当 Bob 得到了这条消息时, 他根据 Alice 的消息中的加密部分构造一个新的消息, 以及一条用他自己的信息构造出来的类似消息。用  $K_A$  和  $K_B$  加密的部分标识了 Alice 和 Bob, 同时也包含了公共的标识符和一个质询随机数。

KDC 对这两部分中的  $R$  进行检查, 看它们是否相同。它们有可能不相同, 比如



Trudy 篡改了消息 1 中的 R, 或者替换了消息 2 中的部分数据。如果两个 R 一致的话, 则 KDC 相信 Bob 的请求消息是有效的。然后它生成一个会话密钥, 并且对它加密两次, 一次为 Alice, 另一次为 Bob。在它发送给 Alice 和 Bob 的两条消息中, 每条消息都包含了接收方的随机数, 以此来证明这是 KDC 而不是 Trudy 生成的消息。这时候 Alice 和 Bob 拥有了同一个会话密钥, 于是他们便可以开始通信了。当他们第一次交换数据消息时, 每一方都看到了对方有一个完全相同的  $K_s$ , 所以认证过程便完成了。

#### 8.7.4 使用 Kerberos 的认证协议

在许多实际系统(包括 Windows 2000)中使用的一个认证协议是 **Kerberos**, 它以 Needham-Schroeder 协议的一种变化形式为基础。Kerberos 得名于古希腊神话中一头专门守卫地狱大门的多头狗(大概它也禁止非法离开地狱)。Kerberos 是由 MIT 设计的, 它的设计目标是允许工作站用户以一种安全的方式来访问网络资源。它与 Needham-Schroeder 协议的最主要不同之处在于, 它假设所有的时钟都已经同步好了。Kerberos 协议经历了几次改版, 其中第 4 版被广泛应用于工业界, 所以我们将重点介绍这个版本。之后, 我们也将简略地介绍一下它的后继版本, 即第 5 版。有关更多的信息, 请参看 Steiner et al., 1988。

除了 Alice 作为一个客户工作站以外, Kerberos 还涉及到三个服务器:

- 认证服务器(AS): 在用户登录过程中验证用户的身份。
- 票据发放服务器(Ticket-Granting Server, TGS): 发放“身份票据的证明”
- Bob 服务器: 真正完成 Alice 所请求的工作。

AS 与 KDC 非常类似, 它与每个用户共享了一个秘密的口令。TGS 的任务是发放票据, 它发放的票据可使真正的服务器相信: TGS 票据的持有者确实是他或她所声称的那一位。

为了开始一个会话, Alice 在任意一台公共的工作站前面坐下来, 并输入她的名字。该工作站将她的名字以明文方式发送给 AS, 如图 8.42 中所示。AS 送回来的是一个会话密钥和一个票据  $K_{TGS}(A, K_s)$ , 这里的票据是给 TGS 看的。这两个数据项被包装在一起, 并且用 Alice 的秘密密钥进行加密, 所以, 惟有 Alice 才能解密此消息。只有当消息 2

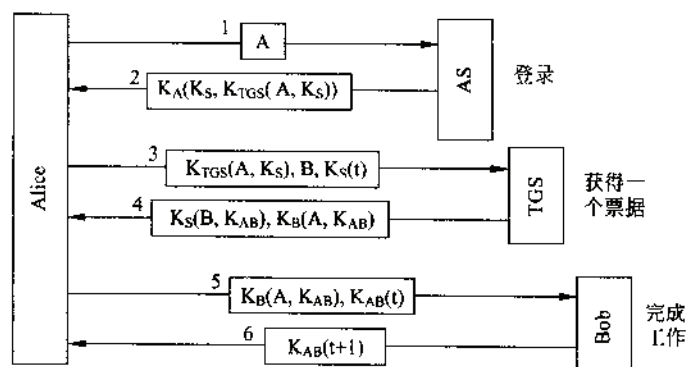


图 8.42 Kerberos V4 的操作过程

到来时,工作站才会请求 Alice 输入口令。然后工作站利用该口令生成  $K_A$ ,从而解密消息 2,并且获得该消息内部的会话密钥和 TGS 票据。这时候,工作站覆盖掉 Alice 的口令,以确保口令的明文至多在工作站内部只停留几毫秒时间。如果 Trudy 试图以 Alice 的身份来登录的话,那么,她输入的口令将是错误的,工作站将会检测到这种错误,因为消息 2 的标准部分是不正确的。

当 Alice 登录进来以后,她可能告诉工作站,她想要联系 Bob 文件服务器。然后工作站给 TGS 发送消息 3,请求一个使用 Bob 文件服务器的票据。这个请求中的关键要素是  $K_{TGS}(A, K_S)$ ,它是用 TGS 的秘密密钥来加密的,它的用途是证明发送方真的是 Alice。TGS 的响应是,创建并送回一个会话密钥  $K_{AB}$ ,以供 Alice 和 Bob 通信时使用。该会话密钥的两个版本都被送回来,第一个版本仅仅用  $K_S$  加密,所以 Alice 可以读取  $K_{AB}$ 。第二个版本用 Bob 的密钥  $K_B$  来加密,所以 Bob 也可以读取此会话密钥。

Trudy 可以复制消息 3,并再次发送此消息,但是,她将会失败,因为连同消息 3 一起发送的有一个加密的时间戳  $t$ 。Trudy 不可能用一个更新的时间戳来代替  $t$ ,因为她不知道  $K_S$ ,即 Alice 用来与 TGS 通话的会话密钥。即使 Trudy 极为快速地重发了消息 3,她所得到的也只不过是消息 4 的另一份副本而已,既然第一次她不能解密消息 4,那么第二次她同样也不能解密。

现在 Alice 可以将  $K_{AB}$  发送给 Bob 以便与他建立一个会话。这次消息交换也使用了时间戳。其中 Bob 的应答消息向 Alice 证明了,她确实是在与 Bob 而不是 Trudy 通话。

经过这一系列的消息交换以后,Alice 就可以在  $K_{AB}$  的保护下与 Bob 进行通信了。如果她后来决定要跟另一个服务器 Carol 进行通话,那么她只要向 TGS 再次发送消息 3,只不过这一次指定 C 而不是 B。TGS 将迅速地回送一个用  $K_C$  加密的票据,以后 Alice 可以将此票据发送给 Carol,而 Carol 则凭此票据相信她收到的消息确实来自 Alice。

所有这些工作的要点在于,现在 Alice 可以以一种安全的方式来访问网络上所有的服务器了,而且她的口令从来都不会出现在网络上。实际上,即使在她自己的工作站上,口令也只是停留了几毫秒而已。然而,请注意,每个服务器都有它自己的授权规则。当 Alice 向 Bob 出示她的票据时,此票据仅仅向 Bob 证明了这是谁发送过来的。至于到底允许 Alice 做什么事情,则要取决于 Bob。

由于 Kerberos 的设计者们并不期望全世界都信任同一个认证服务器,所以他们提供了对多个域(realm)的支持,每个域有自己的 AS 和 TGS。为了得到一个针对远程域中某个服务器的票据,Alice 需要向她自己所在域的 TGS 请求一个票据,并且指定此票据的目标接受者是远程域中的 TGS。如果远程 TGS 已经向本地的 TGS 注册过了(其做法就如同本地服务器的注册方法一样),则本地 TGS 将给 Alice 发送一个可用于远程 TGS 的票据。然后她就可以在那边(即远程域)完成各种事务,比如获取针对该域中各个服务器的票据。然而,请注意,为了让两个域中的多个参与方共同完成一些工作,每一个域必须信任另一个域的 TGS。

Kerberos V5 比 V4 更加复杂,也需要更多的开销。它也使用 OSI ASN.1 (Abstract Syntax Notation 1)来描述数据类型,并且在协议中有一些小的变化。而且,它有更长的票据生存期,允许票据被重新使用(renew),并且也允许发送未来才有效(postdated)的票

据。另外,至少在理论上而言,V5 并不依赖于 DES(而 V4 则依赖于 DES),并且通过将票据的生成过程委托给多个票据服务器的方式来支持多个域。

### 8.7.5 使用公开密钥密码学的认证协议

我们也可以利用公开密钥密码学来完成双向认证。首先,Alice 需要得到 Bob 的公钥。如果有一个 PKI 目录服务器可以提供公钥证书的查询服务,那么 Alice 可以请求 Bob 的公钥证书,如图 8.43 中的消息 1。目录服务器在消息 2 中回应一个 X.509 证书,其中包含了 Bob 的公钥。当 Alice 验证了证书中的签名无误以后,她给 Bob 发送一条消息,该消息包含了她的标识和一个临时值。

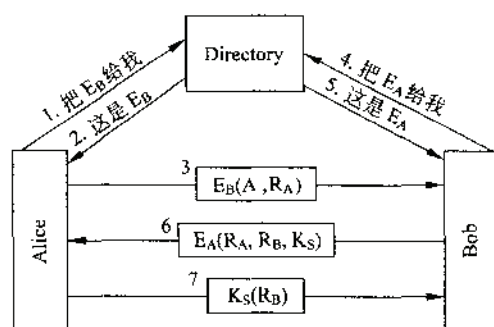


图 8.43 使用公开密钥密码学的双向认证

当 Bob 接收到这条消息时,他无从判断这条消息到底来自于 Alice 还是 Trudy,但是他可以等待一下,并且向目录服务器请求 Alice 的公钥(消息 4),很快他便获得了 Alice 的公钥(消息 5)。然后,他给 Alice 发送一条消息,即图中的消息 6,其中包含了 Alice 的  $R_A$ 、Bob 自己的临时值  $R_B$  和一个建议的会话密钥  $K_S$ 。

当 Alice 得到了消息 6 以后,她用自己的私钥将它解密出来。她看到了里边的  $R_A$ ,这让她有一种很温暖的感觉。这条消息一定是 Bob 发送过来的,因为 Trudy 无法确定  $R_A$ 。而且,它也一定是最新的,而不是一条重放消息,因为她刚刚把  $R_A$  发送给 Bob。Alice 送回消息 7 以表示她同意这个会话。当 Bob 看到了  $R_B$  是用他刚刚生成的会话密钥来加密时,他知道 Alice 已经得到了消息 6 并且验证了  $R_A$ 。

Trudy 怎样才能破坏这个协议呢? 她可以虚构消息 3,以此来引诱 Bob 向 Alice 发送消息 6,但是 Alice 将会发现这个  $R_A$  不是她发送的,所以不会进行下去。Trudy 不可能伪造出送回给 Bob 的消息 7,因为她不知道  $R_B$  或者  $K_S$ ,在没有 Alice 的私钥的情况下,她是不可能确定  $R_B$  和  $K_S$  的。因此,当面对这个协议时她的运气就太不好了。

## 8.8 电子邮件安全

当电子邮件消息在两个远程站点之间传递时,一般情况下该消息沿途要经过几十台机器。这些机器中的任何一台都可以阅读和记录下该消息,以备将来可能之用。在实践

中,不管人们怎么想,隐私是不存在的。然而,许多人希望自己发送的电子邮件只有目标接收者才能阅读,其他人都无法阅读,即使他们的老板或者他们的政府也不能阅读。这种愿望刺激了一些人和组织将前面学习过的密码学原理应用到电子邮件上,从而形成安全的电子邮件。在本节中,我们将首先学习一个已经得到广泛应用的安全电子邮件系统: PGP,然后再简短地介绍另外两个安全电子邮件系统: PEM 和 S/MIME。有关安全电子邮件的更多信息,请参看 Kaufman et al., 2002; and Schneier, 1995。

### 8.8.1 PGP—Pretty Good Privacy

我们的第一个例子 PGP (Pretty Good Privacy, 相当好的隐私)本质上是 Phil Zimmermann 一个人的作品(Zimmermann, 1995a, 1995b)。Zimmermann 是一个极其崇尚隐私的人,他的人生格言是:如果隐私是非法的,那么只有违法者才会有隐私。PGP 于 1991 年被发布,它是一个完整的安全电子邮件软件包,提供了私密性、认证、数字签名和压缩功能,而且所有这些功能都非常易于使用。此外,完整的软件包中也包含了所有的源代码,并通过 Internet 自由分发。由于 PGP 的质量、价格(为 0),以及在 UNIX、Linux、Windows 和 Mac OS 平台上的易用性,今天 PGP 已得到了广泛的使用。

PGP 使用一个被称为 IDEA(International Data Encryption Algorithm)的块密码算法来加密数据。该算法使用 128 位密钥,它是在瑞士被设计出来的,当时 DES 已经被认为不够安全了,而 AES 尚未发明。从概念上讲,IDEA 非常类似于 DES 和 AES;它也运行许多轮,在每一轮中将数据位尽可能地混合起来,但是,它的混合函数与 DES 和 AES 中的不同。PGP 的密钥管理使用 RSA,数据完整性使用 MD5,这些话题我们前面都已经讨论过了。

PGP 自从第一天生以来,就被卷入到一场持久的争论之中(Levy, 1993)。因为 Zimmermann 并没有阻止其他人把 PGP 放到 Internet 上,所以,全世界的任何人都可以得到 PGP,因此,美国政府宣称 Zimmermann 违反了美国的军用武器出口法。美国政府对 Zimmermann 进行了长达 5 年的调查,但是最终放弃了,这里可能有两个原因。第一, Zimmermann 自己并没有把 PGP 放到 Internet 上,所以他的律师声称他没有出口任何东西(至于是否创建了一个包含出口物品的 Web 站点,这是一个小问题了)。第二,政府最终意识到,要想赢得这个案子,意味着要使陪审团相信:一个可下载隐私程序的 Web 站点也被涵盖在军火交易法的适用范围内,因而对 Web 站点的限制就如同禁止出口战争武器(比如坦克、潜艇、军用飞机和核武器)一样。多年来的反面宣传也没有起到作用。

顺便提一下,将出口法应用在 PGP 上也是非常不恰当的。政府认为把代码放在一个 Web 站点上是一种非法的出口,并为此侵扰了 Zimmermann 达 5 年之久。另一方面,如果某个人以一本书的形式来发表 PGP 完整的 C 代码(采用比较大的字体,并且每一页带一个校验和,因此扫描识别非常容易),然后出口这本书,那么,政府认为这没有什么关系,因为图书并不是军品。剑的威力比笔要大得多,至少对于山姆大叔(即美国政府)来说是这样的。

PGP 的另一个问题是,它牵连到了专利侵犯案。拥有 RSA 专利的公司是 RSA Security 公司,它宣称 PGP 使用 RSA 算法侵犯了它的专利,但是从 2.6 版本以后这个问

题已经解决了。此外,PGP 还使用了另一个受专利保护的加密算法,即 IDEA,由此曾经还引起了一些问题。

由于 PGP 是源码开放的,所以,所有的人和组织都可以对它进行修改,并且再产生各种版本。其中有些版本的设计目标是为了回避军品出口法的问题,而其他有些版本则是为了避免使用专利算法,另外还有一些版本则是为了将它变成一个源码封闭的商业产品。虽然军品出口法现在有些放宽了(否则的话,使用 AES 的产品将不可能从美国出口),而且,RSA 算法已于 2000 年 9 月到期,但是,所有这些问题遗留下来的结果是,有许多不兼容的 PGP 版本流传在各处,而且名字也不尽相同。下面的讨论将集中在经典的 PGP 上,这是最老的、也是最简单的版本。另一个流行的版本为 OpenPGP,RFC 2440 描述了这个版本。还有一个版本是 GNU Privacy Guard。

PGP 特意使用现有的密码学算法,而不是发明新的算法。它主要基于那些已经经受了广泛的审查并且未受任何政府机构影响(政府机构总是试图削弱它们)的算法,对于那些不太信任政府的人来说,这个特性是一个很大的吸引点。

PGP 支持正文压缩、私密性和数字签名,也提供了可扩展的密钥管理工具,但奇怪的是,它没有提供电子邮件工具。它只是一个预处理器:接受明文输入,并产生签过名的密文作为输出,其输出格式为 base64。当然,这个输出结果然后可以被通过电子邮件发送出去。在有些 PGP 实现中,最后一步调用一个用户代理以便真正将消息发送出去。

为了看清楚 PGP 是如何工作的,我们来考虑图 8.44 中的例子。这里,Alice 想要通过一种安全的方式给 Bob 发送一个签名的明文消息。Alice 和 Bob 都有私有的( $D_x$ )和公有的( $E_x$ )RSA 密钥。我们假设每一方都知道对方的公钥;我们还将扼要地介绍一下 PGP 的密钥管理机制。

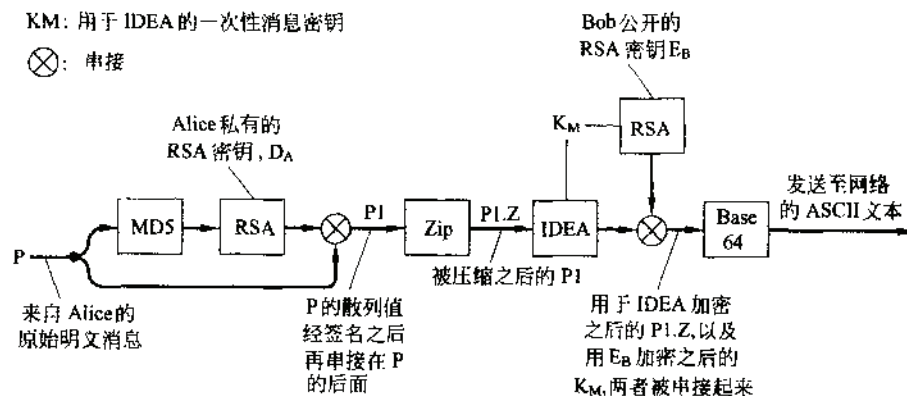


图 8.44 在 PGP 中,发送一个消息的操作过程

作为开始,Alice 调用她自己机器上的 PGP 程序。PGP 首先使用 MD5 算法对她的消息 P 做散列运算,然后用 Alice 的私有 RSA 密钥  $D_A$  加密此结果散列值。当 Bob 最终得到了该消息时,他可以用 Alice 的公钥解密此散列值,并且验证散列值是否正确。即使此时其他人(比如 Trudy)也能够获得这个散列值,并且也可以用 Alice 的公钥对它解密,



但 MD5 的强度保证了,要产生另一个具有相同 MD5 散列值的消息在计算上是不可行的。

经过加密的散列值和原始的消息现在被串接到一条消息 P1 中,并且使用 ZIP 程序进行压缩。ZIP 程序使用了 Ziv-Lempel 算法(Ziv and Lempel, 1977)。我们将这一步的输出结果称为 P1.Z。

接下来,PGP 提示 Alice 输入一些随机的信息。Alice 输入的内容和敲键的速度被用来生成一个 128 位的 IDEA 消息密钥  $K_M$ (在 PGP 的文献中该密钥被称为会话密钥,但是,这种用法实际上不太恰当,因为这里并没有会话)。现在按照密码反馈模式,并利用 IDEA 算法和  $K_M$  来加密 P1.Z。另外,也使用 Bob 的公钥  $E_B$  来加密  $K_M$ 。然后,这两部分内容被串接起来,并转换成 base64 编码,正如我们在第 7 章中讨论 MIME 时所介绍的那样。然后,结果得到的消息只包含字母、数字和符号 +、/ 和 =,这意味着该消息可以被放到一个 RFC 822 的消息体中,并且期望能未被修改地到达另一方。

当 Bob 得到此消息时,他做一个反向的 base64 编码,并且利用他自己的私有 RSA 密钥解密出 IDEA 密钥。利用这个密钥,他解密此消息以得到 P1.Z。经过解压缩之后,Bob 将明文与加密的散列值分离开,并且使用 Alice 的公钥来解密此散列值。如果所得到的明文散列值与他自己计算的 MD5 散列值一致,那么,他知道,P 是正确的消息,而且它确实来自于 Alice。

值得注意的是,在这里只有两个地方用到了 RSA: 为了加密 128 位的 MD5 散列值,以及加密 128 位 IDEA 密钥。虽然 RSA 非常慢,但是,它只需加密 256 位数据,而不是大量的数据。另外,所有这 256 位明文数据是高度随机的,所以,Trudy 需要做相当大量的工作才能确定一个猜测的密钥是否正确。在 PGP 的操作过程中,繁重的加密工作是由 IDEA 来完成的,而 IDEA 比 RSA 快了几个数量级。因此,PGP 不仅提供了安全性、压缩和数字签名的功能,而且它完成这些功能的效率比图 8.19 中显示的方案要高得多。

PGP 支持 4 种 RSA 密钥长度。它可以由用户来选择一种最为合适的长度。这些长度选项如下:

- (1) 临时的(384 位): 今天很容易被破解。
- (2) 商用的(512 位): 可以被三字母组织破解。
- (3) 军用的(1024 位): 地球上的任何人都无法破解。
- (4) 星际的(2048 位): 其他行星上的任何人也无法破解。

由于 RSA 仅仅被用在两个小的计算过程中,所以,每个人在任何时候都应该使用星际强度的密钥。

经典 PGP 消息的格式如图 8.45 所示。在实践中还有其他许多格式。图中显示的消息有三个部分,分别包含了 IDEA 密钥、签名和消息体。密钥部分包含的不仅仅是密钥,还有一个密钥标识符,因为 PGP 允许每个用户有多个公钥。

签名部分包含一个头,这里我们并不关心这个头。头的后面紧跟一个时间戳、发送方公钥的标识符、某些类型信息以及加密的散列值本身。其中,公钥标识符指定了用来解密签名散列值的公钥,而类型信息则标识了所使用的算法(目的是为了当 MD6 和 RSA2 发明出来时可以直接支持它们)。



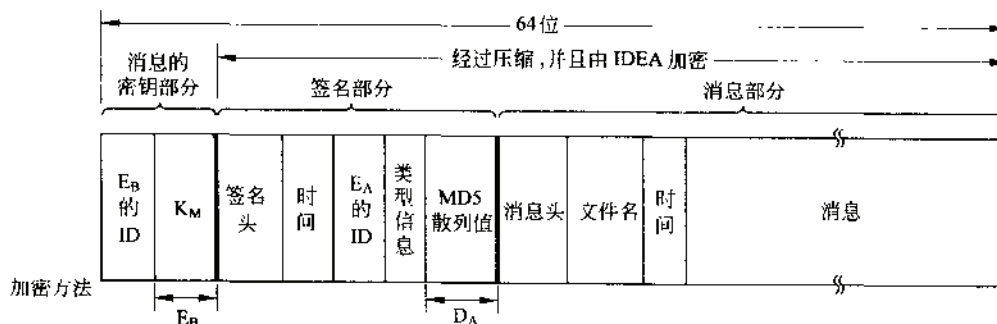


图 8.45 PGP 消息

消息部分也包含一个头,以及一个默认的文件名,如果接收方要将文件写到磁盘中的话,则可以使用此默认的文件名。另外,消息部分还包含了一个用来标识该消息被创建时刻的时间戳,最后是消息本身。

在 PGP 中,密钥管理部分是最为受到关注的,因为它是所有安全系统的要害所在。PGP 的密钥管理模型如下所述。每个用户在本地维护两个数据结构:一个私钥环和一个公钥环。**私钥环(private key ring)**包含一个或者多个本人的公-私钥对。为每个用户支持多对公-私钥的原因是允许用户定期地改变他们的公钥,或者当用户认为某一个私钥已经泄漏时,他无需将当前已准备好或者正在传送过程中的消息统统作废就可以改变公钥。每一对公-私钥都有一个标识符与其关联在一起,所以,消息的发送方可以告诉接收者自己是使用哪一个公钥来对它加密的。该标识符是由公钥的低 64 位构成的。避免公钥标识符发生冲突,这是用户的责任。磁盘上的私钥必须使用特殊的口令(任意长)来加密,以避免它们被偷窥。

**公钥环(public key ring)**包含了与当前用户进行通信的其他用户的公钥。为了加密与每条消息相关联的消息密钥,这些公钥是必要的。公钥环上的每个条目不仅包含了公钥,而且也包含它的 64 位标识符,以及一个代表用户信任此密钥的强烈程度的指示值。

在这里需要解决的问题如下所述。假设公钥被存放在公告板上。Trudy 为了阅读 Bob 的秘密电子邮件,一种办法是攻击公告板,并且用她自己选择的公钥来代替 Bob 的公钥。当 Alice 后来获得了这个自称属于 Bob 的公钥时,Trudy 就可以对 Bob 实施水桶队列攻击。

为了防止这样的攻击,或者至少将攻击的后果减到最小,Alice 需要知道在她的公钥环中,她对这个自称是“Bob 的公钥”有多大的信任程度。如果她知道该公钥来自于 Bob 亲自交给她的软盘,那么她就可以将信任值设置到最大。正是这种非中心化的、由用户控制的公钥管理方法使得 PGP 无需再采用中心化的 PKI 方案。

然而,有时候人们还是要通过查询一个可信的密钥服务器来获得其他用户的公钥。由于这个原因,在 X.509 被标准化以后,PGP 除了支持传统的 PGP 公钥环机制以外,也支持 X.509 证书。所有当前的 PGP 版本都支持 X.509。

### 8.8.2 PEM—Privacy Enhanced Mail

PGP 最初是由一个人推动起来的,与此相反,我们的第二个例子,即 **PEM**(**Privacy Enhanced Mail**,增强隐私的邮件),则是一个正式的 Internet 标准,它是在上个世纪八十年代后期被开发出来的,并且被定义在 4 个 RFC 文档中:从 RFC 1421 至 RFC 1424。粗略地说,PEM 覆盖的范围与 PGP 相同:为基于 RFC 822 的电子邮件系统提供私密性和认证功能。然而,在具体方法和技术上,它与 PGP 有所不同。

使用 PEM 发送的消息首先被转换成一种标准的形式,所以它们针对空白符(比如制表符、结尾符等)有同样的约定。接下来,使用 MD2 或者 MD5 计算出消息的散列值。然后,将散列值和消息串接起来,并且用 DES 进行加密。众所周知,DES 的 56 位密钥是比较弱的,所以 PEM 的这个选择很令人怀疑。然后,利用 base64 编码方法对加密之后的消息进行编码,再发送给收件人。

如同在 PGP 中一样,每条消息都使用一次性密钥进行加密,并且该密钥也被包装到消息中。这个密钥既可以用 RSA 来保护,也可以用 EDE 方式的三重 DES 来保护。

PEM 采用了比 PGP 更加结构化的形式来管理密钥。PGP 密钥的真实性可以通过由 CA 颁发的 X.509 证书来证明,而且这些 CA 被安排成一个严格的、只有一个根的层次结构。这种方案的优点是,有可能让根 CA 定期地发行 CRL 来实现证书的撤销。

PEM 的惟一问题是,由于一直没有人使用 PEM,所以,它早已被束之高阁了。这个问题很大程度上是政治性的:谁来运行根 CA,又在什么样的条件下运行呢?在实践中并不缺乏候选者,但是许多人很害怕将整个系统的安全性全盘托付给任何一家公司。最有兴趣的候选者是 RSA Security 公司,它希望为每个颁发的证书收取一定的费用。然而,有些组织反对这种想法。尤其是,美国政府可以免费使用所有的美国专利,而美国之外的公司已经习惯于免费使用 RSA 算法(RSA Security 公司忘了在美国以外的其他地方为 RSA 算法申请专利)。对于他们来说,本来已经习惯于免费使用的东西,现在突然之间要给 RSA Security 公司支付费用,他们当然没有积极性了。最终,由于找不到可以运行根的机构,所以 PEM 失败了。

### 8.8.3 S/MIME

IETF 接下来又冒险进军安全电子邮件的领域,其结果是 **S/MIME**(**Secure/MIME**,安全的 MIME),RFC 2632 至 2643 描述了 S/MIME。如同 PEM 一样,它也提供了认证、数据完整性、保密性和不可否认性。而且,它也非常灵活,支持大量的密码学算法。无需惊奇的是,既然它使用了这样的名字,那么它一定与 MIME 集成得很好,从而可以保护各种类型的消息。此外,它也定义了许多新的 MIME 头,比如,用来存放数字签名的 MIME 头。

毫无疑问,IETF 从 PEM 的经历中学到了不少经验和教训。S/MIME 并没有一个严格的、从单个根开始的证书层次结构。相反,用户可以有多个信任锚。只要一个证书能够被回溯到当前用户所相信的某一个信任锚,则它就被认为是有效的。S/MIME 使用了标准的算法和协议,关于这些算法和协议,我们前面都已经学习过了,所以这里不再进一步

讨论它们。有关 S/MIME 的细节,请参考相应的 RFC。

## 8.9 Web 安全

我们前面已经学习了两个非常需要安全性的重要领域:通信和电子邮件。你可以将它们想象成汤羹和开胃小吃。现在是该上主菜的时候了,即 Web 安全。Web 是如今的 Trudy 们的主要工作场所,他们也正是在 Web 上完成他们卑劣的工作。在本节中,我们将讨论与 Web 安全有关的一些问题和事项。

我们可以粗略地将 Web 安全分成三个部分。第一,如何安全地命名对象和资源?第二,如何建立起安全的、真实的连接?第三,当一个 Web 站点向客户发送一段可执行代码时,又会怎么样呢?下面我们首先看一些潜在的威胁,之后我们再详细地讨论每一部分。

### 8.9.1 威胁

人们几乎每个星期都可以在报纸上看到有关 Web 站点安全的问题。这种状况确实非常严酷。我们现在看一些已经发生过的例子。首先,许多组织的主页受到过攻击,被代之以破坏者(cracker,有时也被称为骇客)所选择的新主页(流行的出版物将这些侵入别人计算机的人称为“黑客[hacker]”,但是许多程序员把“黑客”这个术语留给了那些超级程序员。本书中我们将这些人称为“破坏者[cracker]”)。曾经遭受过这种破坏的站点包括 Yahoo、美国军方、CIA、NASA 和纽约时报。在绝大多数情况下,破坏者只是放置一些稀奇古怪或者调侃的文字,因而遭受破坏的站点可以在几小时内被修复。

现在我们看一些更加严重的案例。有许多站点曾经因为遭受拒绝服务攻击而停止运行,在这些攻击中,破坏者用大量的流量淹没掉 Web 站点,使它不能对合法的请求做出响应。通常这种攻击是这样形成的:已被破坏者侵入的大量机器向一个目标聚集,从而将它淹没(即 DDoS 攻击)。这样的攻击非常常见,所以它们甚至已经不再是新闻了,但是,它们却可以使被攻击的站点造成成千上万美元的损失。

1999 年,一名瑞典的破坏者侵入了 Microsoft 的 Hotmail Web 站点,并创建了一个镜像站点,该镜像站点允许任何人输入 Hotmail 用户的名字,然后读取所有人当前的和已经存档的电子邮件。

在另一个案例中,一个名叫 Maxim 的 19 岁俄罗斯破坏者侵入了一个电子商务 Web 站点,并且偷到了 300 000 个信用卡号码。然后,他联系上站点的所有者们,并告诉他们,如果他们不付给他 100 000 美元的话,他将把所有的信用卡号码张贴到 Internet 上。他们并没有屈从于他的敲诈勒索,结果他真的张贴了信用卡号码,因而极大地损害了许多无辜者的利益。

在另一个不同类型的事件中,一名 23 岁的加利福尼亚学生以电子邮件的形式给一家通讯社发了一则新闻稿,谎称 Emulex 公司将要发布有关季度亏损的消息,而且 C. E. O 也要立即辞职。在几小时以内,该公司的股票暴跌了 60%,从而导致股票持有者损失了 20 亿美元以上。而作案者在发送公告之前不久卖出了他的股票,他赚了二十多万美元。虽然这起事件不属于 Web 站点入侵,但是很显然,这样的公告放在任何一家大公司的主

页上,也会有类似的效果。

我们可以用许多页的篇幅来介绍类似这样的事件(这真的很不幸)。但现在是该介绍与 Web 安全相关的技术要素的时候了。有关各种安全问题的更多信息,请参考 Anderson, 2001; Garfinkel with Spafford, 2002; and Schneier, 2000。你也可以在 Internet 上搜索到大量的实际案例。

### 8.9.2 安全的命名机制

我们首先从一些最基本的问题入手: Alice 希望访问 Bob 的 Web 站点。她在她的浏览器中敲入 Bob 的 URL,几秒钟以后,一个 Web 页面出现了。但是,这真的是 Bob 的主页吗? 可能是,也可能不是。Trudy 可能又重施故技了。例如,她可能截获了 Alice 发出来的所有分组,并查看了这些分组的内容。当她捕捉到有一个 HTTP GET 请求指向 Bob 的 Web 站点时,她可以到 Bob 的 Web 站点上将页面取过来,然后根据她的意愿进行修改,再将修改之后的假页面返回给 Alice。Alice 对此一无所知。更糟的是,Trudy 可以大幅降低 Bob 电子商场中的货物价格,以便使他的货物看起来非常有诱惑力,从而诱使 Alice 将她的信用卡号码发送给“Bob”以购买商品。

这种经典的中间人攻击的一个缺点是,Trudy 必须位于某一个能够截取到 Alice 的输出流量以及能够伪造她的输入流量的位置上。在实践中,她必须搭接在 Alice 的电话线上,或者 Bob 的电话线上,因为搭接到光纤骨干线路上是相当困难的。虽然主动搭线这种方法肯定是行得通的,但是它需要一定的工作量;另外,虽然 Trudy 非常聪明,但同时她也非常懒惰。除了这种做法以外,还有其他更加容易的方法在等着 Alice 去尝试。

#### DNS 欺骗

例如,假设 Trudy 能够攻破 DNS 系统(有可能只是控制了 Alice 的 ISP 上的 DNS 缓存),然后,Trudy 用她自己的 IP 地址(比如说 42.9.9.9)来替换 Bob 的 IP 地址(比如说 36.1.2.3)。这将会导致下述的攻击发生。图 8.46(a)显示了设想中的工作过程。这里,(1)Alice 向 DNS 请求 Bob 的 IP 地址;(2)得到了地址;(3)向 Bob 请求他的主页;(4)得到了 Bob 的主页。当 Trudy 将 Bob 的 DNS 记录修改成包含她自己的 IP 地址而不再是 Bob 的地址时,我们得到了如图 8.46(b)所示的情形。这里,当 Alice 查询 Bob 的 IP 地址时,她得到了 Trudy 的 IP 地址,所以她期望发送给 Bob 的流量全部转向 Trudy。现在 Trudy 就可以实施中间人攻击,而无需陷入到搭接任何电话线的麻烦之中。不过,她必须要侵入一台 DNS 服务器,并改变其中一条记录,这可能会容易得多。

Trudy 如何欺骗 DNS 呢? 这实际上是比较简单的。简而言之,Trudy 可以诱使 Alice 的 ISP 中的 DNS 服务器送出一个查询请求,请求查找 Bob 的 IP 地址。由于 DNS 使用的是 UDP,所以对于 DNS 服务器来说,非常不幸的是,它实际上并没有办法来检查到底谁提供了答案。Trudy 可以充分利用这种特性,她的做法是,伪造一个预设的应答分组,从而将一个假的 IP 地址插入到 DNS 服务器的缓存中。为了简化起见,我们将假设 Alice 的 ISP 初始时并不包含与 Bob 的 Web 站点 bob.com 相对应的 DNS 条目。如果已经存在的话,则 Trudy 可以等待一段时间,直到该条目超时然后再重试(或者使用其他的

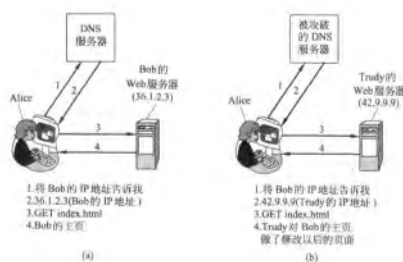


图 8.46

(a) 正常的情况; (b) 一种攻击, 侵入 DNS 并修改 Bob 的记录

技巧)。

Trudy 发起攻击的做法是, 她向 Alice 的 ISP 发送一个查询请求, 请求 bob.com 的 IP 地址。由于不存在与这个 DNS 名字相对应的条目, 所以缓存服务器询问顶级的 com 域服务器, 以便得到此 DNS 名字的 IP 地址。然而, Trudy 抢在 com 服务器之前先送回一个假的应答“bob.com 是 42.9.9.9”, 这里的 IP 地址正是她自己的地址。如果她的假应答首先到达 Alice 的 ISP 处, 则她的 IP 地址将被缓存起来, 而真正的应答则变成一个无人期待而主动发送过来的应答, 从而被拒绝。诱使 DNS 服务器装入一个假 IP 地址的技术被称为 **DNS 欺骗 (DNS spoofing)**。像这样存放了一个刻意的假 IP 地址的缓存被称为已中毒的缓存 (**poisoned cache**)。

实际上, 事情并没有这么简单。首先, Alice 的 ISP 要检查该应答是否包含顶级服务器的正确 IP 源地址。但是, 由于 Trudy 可以在这个 IP 域中设置任何数值, 所以她很容易挫败这项测试, 因为顶级服务器的 IP 地址是公开的。

第二, 为了允许 DNS 服务器辨别哪个应答对应于哪个请求, 所有的请求都携带一个序列号。为了欺骗 Alice 的 ISP, Trudy 必须知道它的正确序列号。要想了解当前的序列号, 对于 Trudy 来说, 最简单的办法是自己注册一个域, 比如说 trudy-the-intruder.com, 我们假定它的 IP 地址也是 42.9.9.9。她也为这个新谋划的域创建一个 DNS 服务器: dns.trudy-the-intruder.com。而且, 该服务器仍然使用 Trudy 的 IP 地址 42.9.9.9, 因为 Trudy 只有一台计算机。现在, 她必须让 Alice 的 ISP 知道她的 DNS 服务器。这很容易做到。她只需向 Alice 的 ISP 请求 foobar.trudy-the-intruder.com 即可, 这会导致 Alice 的 ISP 询问顶级 com 服务器, 谁在运行 Trudy 的新域的 DNS 服务。

随着 dns.trudy-the-intruder.com 安全地进入到 Alice 的 ISP 的缓存中, 真正的攻击开始了。现在 Trudy 向 Alice 的 ISP 请求查询 www.trudy-the-intruder.com。该 ISP 很



自然地，Trudy 向 com 的 DNS 服务器发送一个同样的查询请求。该请求携带的序列号正是 Trudy 所努力寻找的。Trudy 高兴得跳了起来，她立即请求 Alice 的 ISP 查询 Bob 的地址，并迅速地发送一个伪造的应答来回答自己的问题，而且谎称该应答来自顶级 com 服务器，该应答的内容是：“bob.com 是 42.9.9.9”。这个伪造的应答中所携带的序列号，比她刚刚接收到的请求中的序列号大 1。虽然她已经算得很准确了，但是她也可以再发送第二个伪造的应答，其中序列号比刚才接收到的大 2，甚至可能发送几十个伪造的应答，其中的序列号逐渐增加。这些应答中总有一个会匹配到 Alice 的 ISP 所发送的 DNS 查询请求，剩下的将被丢弃。当这个伪造的应答到达时，它被缓存起来；而当真正的应答稍后到达时，它被拒绝，因为那时已经没有正在等待回应的查询了。

现在，当 Alice 查询 bob.com 时，它被告知应该使用 42.9.9.9，即 Trudy 的地址。Trudy 在她自己舒适的房间里，已经成功地实施了中间人攻击。图 8.47 显示了这种攻击的各个步骤。使事情更加糟糕的是，这里展示的技术并不是唯一的欺骗 DNS 的方法。实际上还存在许多其他的方法。

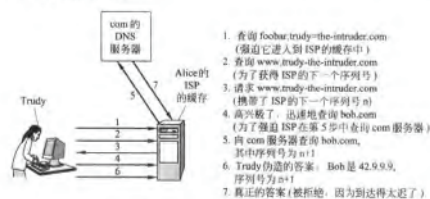


图 8.47 Trudy 如何欺骗 Alice 的 ISP

#### 安全的 DNS

前面介绍的攻击很容易被挫败，只要让 DNS 服务器在它们发出的查询请求中使用随机的 ID 而不是按顺序递增的 ID，但事情总是这样的，每当一个漏洞被补上的时候，另一个漏洞又冒出来。真正的问题在于，当 DNS 被设计出来时，Internet 只是一个仅供几百所大学使用的研究工具，而不是让 Alice、Bob 或者 Trudy 都加入进来的聚会场所。那时候安全并不是一个问题，使 Internet 能真正工作起来才是最为重要的。随着这几十年来，环境已经有了很大的变化，所以，1994 年，IETF 成立了一个工作组以努力使 DNS 从本质上变得更加安全。这个项目被称为 **DNSsec(DNS Security)**，关于该项目的成果，请参考 RFC 2535。不幸的是，DNSsec 尚未被完全地部署起来，所以，目前有大量的 DNS 服务器仍然会遭受各种欺骗攻击。

从概念来看，DNSsec 极其简单，它建立在公开密钥密码学的基础之上。每一个 DNS 区域（按照图 7.4 中的含义）有一个公/私钥对。DNS 服务器发送的所有信息都经过



---

了签名,签名所用的私钥是发起区域的私钥,所以接收方可以验证它的真实性。

DNSsec 提供了三个基本的服务:

(1) 证明数据是从哪里发出来的。

(2) 公钥分发。

(3) 对事务处理和请求的认证。

主要的服务是第一个,它证实了所返回的数据确实是经过区域的所有者同意的。第二个服务对于安全地保存和获取公钥非常有用。第三个服务对于预防重放和欺骗攻击是非常必要的。请注意,DNSsec 并没有提供保密性服务,因为 DNS 中的所有信息都应该是公开的。按照预定的期望,DNSsec 需要几年时间才能被部署起来,所以,具有安全功能的 DNS 服务器与不支持安全功能的服务器之间的协同工作能力是非常必要的,这意味着 DNSsec 不能改变协议。现在我们来看一看其中一些细节。

DNS 的记录被组织成一些称为 **RRSet(Resource Record Sets,资源记录集)** 的集合,所有具有同样名字、类别和类型的记录被集中到一个集合中。一个 RRSet 可能包含多个 A 记录,例如,倘若一个 DNS 名字被解析到一个主要的 IP 地址和一个次要的 IP 地址。RRSet 也允许扩展新的记录类型(后面将讨论到)。每个 RRSet 都有一个密码学意义上的散列值(比如使用 MD5 或者 SHA-1),然后又利用该区域的私钥对散列值进行签名(比如使用 RSA 算法)。发送给客户的传输单元是经过签名的 RRSet。当客户接收到一个签过名的 RRSet 时,它可以验证此 RRSet 是否已被发起区域签过名。如果签名没有问题的话,则 RRSet 中的数据可被接受。由于每个 RRSet 都包含它自己的签名,所以,RRSet 可以被缓存在任何地方,即使被缓存在不可信的服务器上也不会危及安全性。

DNSsec 引入了几种新的记录类型。第一种是 KEY 记录。这种记录包含了某一个区域、用户、主机或者其他个体的公钥,以及用于签名的密码算法、所使用的传输协议,还有其他一些数据位。这里的公钥是完全裸露的,DNSsec 没有使用 X.509 证书,因为它太庞大。若算法域被设置为 1,则代表了签名方案使用 MD5/RSA 算法组合(这是首选的方案);其他的算法组合使用其他的值。协议域表明了使用 IPsec 或者其他的安全协议(如果有的话)。

第二种新的记录类型是 SIG 记录。它存放的是经过签名的散列值,其中所用的算法由 KEY 记录来指定。这个签名作用在该 RRSet 中所有的记录上(即签名的范围),包括所有出现的 KEY 记录,但是不包括 SIG 记录自身。SIG 记录也包含了该签名有效周期的开始时间和过期时间、签名者的名字以及其他一些信息。

在 DNSsec 的设计方案中,每个区域的私钥可以一直保持离线的状态。每天一次或者两次将一个区域的数据库通过手工方式(比如通过 CD-ROM)传输到一台未联网的机器(即私钥所在的机器)上。所有的 RRSet 在那里被签名,由此而产生的 SIG 记录被传回到该区域的主服务器中。通过这种方式,私钥可以被保存在 CD-ROM 上,除了每天签名新的 RRSet 时需要将 CD-ROM 插入到未联网的机器中以外,其他时间该 CD-ROM 被锁在保险箱里。当签名完成以后,内存中和磁盘上所有的私钥副本都被擦除掉,并且 CD-ROM 又放回保险箱里。这个过程实际上把电子安全还原为物理安全,人们对于物理安全总是比较容易理解和处理。

这种预先为 RRSet 做签名的方法大大加速了 DNS 服务器响应客户查询的过程,因为这样就不需要在处理过程中做密码学运算了。所付出的代价是,在 DNS 数据库中需要大量的磁盘空间来存储所有的密钥和签名。由于引入了签名,有些记录的长度将增加十倍以上。

当客户进程获得了一个签过名的 RRSet 时,它必须利用发起区域的公钥来解密散列值,同时自己计算一遍散列值,然后比较这两个值。如果它们相等,则认为此 RRSet 有效。然而,这个过程又引出了另一个问题,即客户如何得到一个区域的公钥。一种方法是利用一个安全的连接(比如使用 IPsec)通过一台可信的服务器来获得区域的公钥。

然而,在实践中,DNSsec 假定客户已经预先配置了所有顶级域的公钥。如果 Alice 现在想要访问 Bob 的 Web 站点,则她可以向 DNS 请求 bob.com 的 RRSet,该 RRSet 中包含了 Bob 的 IP 地址和一个 KEY 记录(其中包含了 Bob 的公钥)。此 RRSet 已被 com 顶级域签过名,所以 Alice 很容易验证它的有效性。图 8.48 中显示了这个 RRSet 可能包含的内容。

域名	生存期(Time to live)	类别	类型	值
bob.com	86400	IN	A	36.1.2.3
bob.com	86400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com	86400	IN	SIG	86947503A8B848F5272E53930C...

图 8.48 bob.com 的 RRSet 例子: KEY 记录是 Bob 的公钥;

SIG 记录是顶级 com 服务器对 A 和 KEY 记录的散列值的签名,以证明它们的真实性

现在 Alice 拥有了 Bob 公钥的一份副本,并且已经通过了验证,于是她向 Bob 的 DNS 服务器(由 Bob 运行)请求 www.bob.com 的 IP 地址。Bob 返回的 RRSet 将用 Bob 的私钥来做签名,所以 Alice 可以验证这个 RRSet 上的签名。如果 Trudy 设法将一个假的 RRSet 插入到任何一个缓存中,则 Alice 很容易就可以检测出它是不真实的,因为 RRSet 中的 SIG 记录将是不正确的。

然而,DNSsec 也提供了一种密码学机制来把应答和特定的查询绑定起来,从而防止像图 8.47 中 Trudy 所做的那种欺骗。这种(可选的)反欺骗手段的做法是,应答方利用自己的私钥对查询消息的散列值做签名,然后把签名后的散列值也加入到应答消息中。由于 Trudy 不知道顶级 com 服务器的私钥,所以她不可能为 Alice 的 ISP 所发送的查询伪造一个应答。她当然可以让她的应答先回到 Alice 的 ISP 处,但是它将被拒绝,因为她对查询的散列值所做的签名是无效的。

DNSsec 也支持其他一些记录类型。例如,CERT 记录可以被用来保存证书(比如 X.509 证书)。之所以提供这个记录是因为有些人希望将 DNS 变成一个 PKI。这种变化是否真的会发生还有待于进一步观察。我们对于 DNSsec 的讨论到此为止,有关更多的细节,请参考 RFC 2535。

### 自证明的名字

安全的 DNS 并不是惟一种保护名字的方法。另一种完全不同的方法被用于安全

的文件系统(Secure File System)中(Mazières et al., 1999)。在这个项目中,作者们设计了一个安全的、可扩展的、全球范围的文件系统,它不需要修改(标准的)DNS,也不使用证书或者假设存在一个 PKI。这一部分我们将介绍一下他们的想法是如何被应用到 Web 上的。因此,在下面的描述中,我们将使用 Web 术语而不是论文中所使用的文件系统术语。但是为了避免可能的混淆,这里说明一点,虽然这种方案可以被应用到 Web 上从而获得很高的安全性,但是,它当前并没有真正被使用,而且,为了把它引入进来,将需要对软件做实质性的修改。

作为开始,我们假设每个 Web 服务器都有一个公/私钥对。这种思想的本质是,每个 URL 包含了服务器名字和公钥的一个密码学散列值,也就是说,此散列值成为 URL 的一部分。例如,在图 8.49 中,我们看到了 Bob 的图片的 URL。它以常规的“http:”作为开头,后面跟着服务器的 DNS 名字,即 www.bob.com。然后是一个冒号和 32 字符的散列值。最后也跟平常一样,是文件的名字。除了散列值以外,这是一个标准的 URL。由于有了散列值,它变成了一个自证明的 URL(self-certifying URL)。

服务器	SHA-1(服务器名字, 服务器的公钥)	文件名
<hr/>		
http://www.bob.com:2g5hd8bfjkc7mf6hg8dgany23xds4pe6/photos/bob.jpg		

图 8.49 一个自证明的 URL,其中包含了服务器名字和公钥的散列值

一个很显然的问题是:这个散列值的用意是什么?该散列值的计算方法是,首先将服务器的 DNS 名字与服务器的公钥串接起来,然后运行 SHA-1 函数得到一个 160 位的散列值。在这种方案中,散列值用一个由数字或小写字母(共 32 个选择)组成的序列来表示,为了避免混淆,不使用字母“l”和“o”,以及数字“1”和“0”,所以还剩下 32 个数字和字母。由于有 32 个字符可以使用,所以,每个字符可以编码一个 5 位的串。由 32 个字符组成的字符串可以编码 160 位的 SHA-1 散列值。实际上,使用散列值并不是必要的,使用密钥本身也是可以的。使用散列值的好处是缩短名字的长度。

为了看到 Bob 的图片,在最简单(但最不方便)的方法中,Alice 只要在她的浏览器中输入图 8.49 中的字符串。浏览器给 Bob 的 Web 站点发送一条消息,向 Bob 请求他的公钥。当 Bob 的公钥到来时,浏览器将服务器名字和公钥串接起来,并运行散列算法。如果算出来的结果与安全 URL 中 32 字符的散列值一致,那么,即使 Trudy 截获了这个请求,并且伪造了应答消息,但她也无法找到一个适当的公钥能给出期望的散列值。因此她所做的任何干扰和破坏都能够被检测到。Bob 的公钥可以被缓存起来以备将来使用。

现在 Alice 必须要验证 Bob 拥有对应的私钥。她构造一条消息,其中包含一个建议的 AES 会话密钥、一个临时值和一个时间戳。然后她用 Bob 的公钥来加密此消息,并且发送给 Bob。由于只有 Bob 才拥有对应的私钥,所以只有 Bob 才能解密此消息,然后他用 AES 密钥加密临时值,再把加密之后的临时值送回给 Alice。当 Alice 接收到用 AES 加密的正确的临时值时,她知道自己在跟 Bob 通话。而且,Alice 和 Bob 现在有了一个 AES 会话密钥,它可被用于后续的 GET 请求和应答。

一旦 Alice 有了 Bob 的图片(或者任何一个 Web 页面),她可以将这个页面加入到书

签表中,所以她下次不用再手工输入完整的 URL 了。而且,嵌入在 Web 页面中的 URL 也可以是自证明的,所以用户只需单击这些 URL 就可以使用它们,因此,用户不用做额外的工作,但是却获得了额外的安全性(知道所返回的页面一定是正确的)。为了避免第一次手工输入自证明的 URL,其他还有一些方法,比如通过一个安全连接(指向某一个可信服务器)来获得这些 URL,或者将 URL 放到由 CA 签名的 X.509 证书中。

另一种获得自证明 URL 的方法是连接到一个可信的搜索引擎上,其做法是,输入该搜索引擎的自证明 URL(第一次),并且经过如上面所描述的协议交互过程,从而与可信的搜索引擎服务器之间建立一个安全的、真实的连接。然后让搜索引擎执行各种查询,在结果得到的签过名的页面上充满了自证明的 URL,用户只需单击这些 URL 而无需再敲入长长的字符串了。

现在我们来查看一下这种方法如何很好地对抗 Trudy 的 DNS 欺骗攻击。如果 Trudy 设法毒害 Alice 的 ISP 的缓存,那么 Alice 的请求有可能被错误地递交到 Trudy 处,而不是 Bob 处。但是,现在这个协议要求初始消息的接收者(即 Trudy)返回一个能产生正确散列值的公钥。如果 Trudy 返回她自己的公钥,则 Alice 立即就能检测出来,因为算出来的 SHA-1 与自证明的 URL 不匹配。如果 Trudy 返回 Bob 的公钥,则 Alice 将检测不到攻击,但是 Alice 将使用 Bob 的密钥来加密她的下一条消息。Trudy 能够获得这条消息,但是她无法解密该消息,从而不可能提取出 AES 密钥和临时值。无论怎么样,DNS 欺骗攻击所能够达到的最佳效果也就是拒绝服务攻击。

### 8.9.3 安全套接字层 SSL

安全的命名机制是一个很好的起点,但是,于 Web 安全而言,这里还有更多的问题。下一步是安全的连接。接下来我们来看一看如何实现安全的连接。

当 Web 最初出现在公众面前时,它仅仅被用于发布静态的页面而已。然而,不久之后,有些公司就想到了将 Web 用于金融交易,比如用信用卡购物、在线银行和电子股票交易。这些应用创造了一种新的需求,即迫切需要安全的连接。1995 年,Netscape 公司(Netscape Communications Corp)作为当时占主导地位的浏览器厂商,对此迅速做出了响应,他们引入了一个被称为 SSL(Secure Sockets Layer,安全套接字层)的安全软件包来迎合这种需求。这个软件包和它的协议现在已被广泛使用,甚至也为 Internet Explorer 所采用,所以,我们在这里有必要详细地做一番介绍。

SSL 在两个套接字之间建立一个安全的连接,其中包括以下功能:

- (1) 客户与服务器之间的参数协商。
- (2) 客户和服务器的双向认证。
- (3) 保密的通信。
- (4) 数据完整性保护。

以前我们曾经看到过这些功能的确切含义,所以这里不再细致地加以阐述。

图 8.50 显示了 SSL 在通常的协议栈中的定位。实际上,它是位于应用层和传输层之间的一个新层,它接受来自浏览器的请求,再将请求转送给 TCP 以便传输到服务器上。一旦安全的连接已经被建立起来,则 SSL 的主要任务是处理压缩和加密。在 SSL 之上使

用的 HTTP 尽管也是标准的 HTTP 协议,但是它被称为 **HTTPS(Secure HTTP, 安全的 HTTP)**。有时候它使用一个新的端口(443),而不是标准的端口(80)。顺便提一下,尽管 SSL 并不仅限于被用在 Web 浏览器中,但这仍然是它最常见的应用。



图 8.50 家庭用户使用 SSL 进行 Web 浏览时的协议栈

SSL 协议经历了几个版本。下面我们只讨论第 3 版,这是最为广泛使用的版本。SSL 支持许多种不同的算法和选项。这些选项包括是否使用压缩功能、使用哪些密码学算法,以及一些与密码产品出口限制有关的事项。最后一项的意图是为了确保高强度的密码学技术仅用于当连接双方都在美国国内时的情形。在其他情况下,密钥被限制在 40 位,许多密码学家对此非常不屑一顾。为了从美国政府这里拿到出口许可,Netscape 被迫加入这项限制。

SSL 由两个子协议组成,一个可用来建立安全的连接,另一个使用安全的连接。我们首先来看一看如何建立安全的连接。图 8.51 显示了连接建立子协议。子协议从消息 1 开始,此时 Alice 向 Bob 发送一个建立连接的请求。该请求指定了 Alice 所用的 SSL 版本,以及她优先选择的压缩算法和密码学算法。它也包含一个临时值  $R_A$ ,后面将会用到此值。

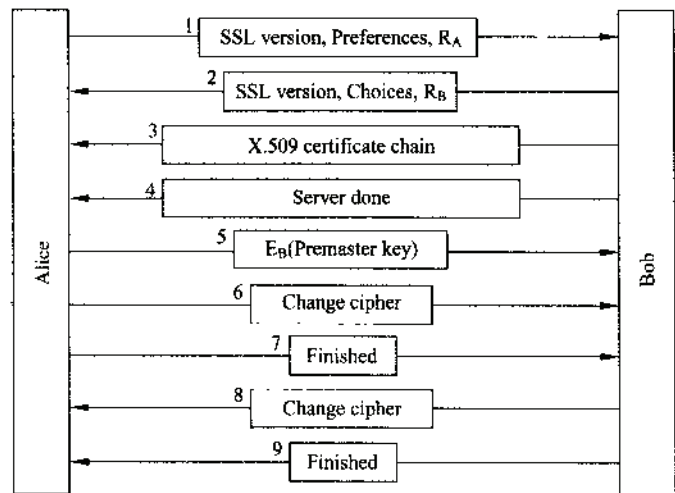


图 8.51 SSL 连接建立子协议的简化版本



现在轮到 Bob 了。在消息 2 中, Bob 在 Alice 可能支持的各种算法之中做出选择, 并且将他的临时值  $R_B$  发送给 Alice。然后, 在消息 3 中, 他发送一个证书, 其中包含了他的公钥。如果这个证书并没有被某一个知名的权威机构签过名, 那么, 他也发送一个证书链, 通过这条证书链可以回溯到某一个由权威机构签名的证书。所有的浏览器, 包括 Alice 的浏览器, 都预装了大约 100 多个公钥, 所以, 如果 Bob 能够建立起一条证书链, 并且此证书链以这 100 多个证书中的某一个作为信任锚, 那么, Alice 将能够验证 Bob 的公钥。这时 Bob 可以发送其他一些消息(比如请求 Alice 的公钥证书)。当 Bob 完成以后, 他发送消息 4 以便告诉 Alice 该轮到她了。

Alice 的响应是, 选择一个随机的 384 位**预设主密钥**(premaster key), 并且用 Bob 的公钥加密之后发送给 Bob(消息 5)。用于加密数据的实际会话密钥是从这个预设主密钥和两个临时值, 通过一种复杂的方法推导得来的。当 Bob 接收到消息 5 以后, Alice 和 Bob 都能够计算会话密钥。由于这个原因, Alice 告诉 Bob, 现在请切换到新的密码(消息 6), 并且告诉他她已经完成了建立连接子协议(消息 7)。然后, Bob 对这两条消息分别进行确认(消息 8 和 9)。

然而, 虽然 Alice 知道了 Bob 是谁, 但是, Bob 并不知道 Alice 是谁(除非 Alice 有一个公钥和相应的证书, 而这对于普通个人用户来说是不太可能的)。因此, Bob 的第一条消息极有可能是一个登录请求, 请求 Alice 用一个预先建立的名字和口令登录进来。然而, 此登录协议完全超出了 SSL 的范围。一旦 SSL 的安全连接已经建立起来, 不管怎么样, 数据传输就可以开始了。

正如上面所提到的, SSL 支持多种密码学算法。最强的一种方案使用三个独立密钥的三重 DES 来加密数据, 使用 SHA-1 来保护数据完整性。这种做法相对比较慢, 所以它主要用于银行和其他一些对安全性要求比较高的应用。对于普通的电子商务应用, 则使用 128 位密钥的 RC4 算法来完成数据加密, 使用 MD5 来实现消息认证。RC4 将这个 128 位的密钥用作一个种子, 并且将它扩充成一个非常大的数以便内部使用。然后, 它利用这个内部数来生成一个密钥流。该密钥流与明文做 XOR 操作, 从而实现了一个经典的流密码算法(如图 8.14 中所示)。出口版本也使用了 128 位密钥的 RC4 算法, 但是其中 88 位是公开的, 以使该密码算法易于被破解。

为了实际传输数据, 我们需要使用第二个子协议, 如图 8.52 所示。来自浏览器的消息首先被分割成最多 16KB 的单元。如果当前连接支持压缩功能的话, 则每个单元被单独压缩。之后, 根据两个临时值和预设主密钥推导出来的一个秘密密钥被串接到压缩之后的文本中, 然后再利用已经协商好的散列算法(通常是 MD5)对串接之后的结果做散列运算。这个散列值被附加在每一个分段的尾部, 作为它的 MAC(消息认证码)。然后, 再使用协商好的对称加密算法(通常是与 RC4 的密钥流进行 XOR 运算)对压缩之后的分段和 MAC 进行加密。最后, 每个分段被附上一个分段头, 再通过 TCP 连接被传送出去。

然而, 有一个警告值得在这里提出来。由于研究人员已经证明了 RC4 算法存在一些弱密钥, 即很容易被分析的密钥, 所以, 使用 RC4 的 SSL 其安全性将是不可靠的(Fluhrer et al., 2001)。如果浏览器允许用户来选择密码算法组合, 则用户应该将它配置成使用 168 位密钥的三重 DES 和 SHA-1, 尽管这种组合比 RC4 和 MD5 要慢一些。



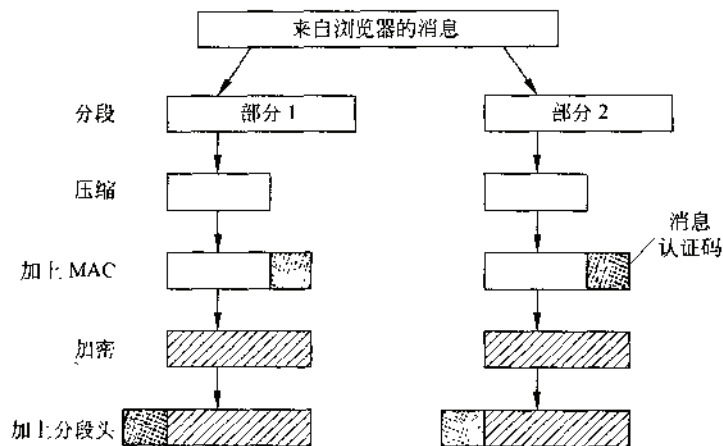


图 8.52 使用 SSL 的数据传输

SSL 的另一个问题是,通信个体可能没有证书,或者,即使他们有证书,他们也并不总是验证所用的密钥是否与证书相匹配。

1996 年, Netscape 公司将 SSL 移交给 IETF 进行标准化。最终的结果是 TLS (Transport Layer Security, 传输层安全)。RFC 2246 描述了 TLS。

尽管 IETF 对 SSL 所做的变化非常小,但即使这些小小的变化也足以使 SSL 第 3 版和 TLS 无法互操作。例如,从预设主密钥和临时值推导出会话密钥的方法也被改变了,从而使得密钥更强了(即难以进行密码分析)。TLS 版本也被称为 SSL 3.1 版。1999 年出现了第一个 TLS 实现,然而,尽管 TLS 比 SSL 有了略微的增强,但目前还不清楚在实践中 TLS 是否会取代 SSL。不过,RC4 弱密钥的问题仍然存在。

#### 8.9.4 移动代码的安全

命名和连接是与 Web 安全有关的两个主要关注领域。但是,除此以外还有其他一些领域也值得关注。在早期时候,当时 Web 页面只是一些静态的 HTML 文件,它们不包含任何可执行的代码。现在,Web 页面通常包含小的程序,例如 Java applet、ActiveX 控件和 JavaScript 等。下载并执行这样的移动代码(mobile code)很显然存在极大的安全风险,所以人们已经设计了各种方法来尽可能地降低这种风险。现在我们来快速地看一看某些因为移动代码而引起的问题,以及处理这些问题的一些做法。

##### Java Applet 的安全

Java applet(也被称为 Java 小程序)是小的 Java 程序,它们已被编译成一种面向栈的机器语言,即 JVM(Java Virtual Machine, Java 虚拟机)。Java applet 可以被放到 Web 页面上,连同 Web 页面一起被下载到用户机器上。当 Web 页面被加载到浏览器中以后,这些 Java applet 也被插入到浏览器内部的 JVM 解释器中,如图 8.53 所示。

在经过编译的代码上再来运行解释性代码的好处是,解释器在执行每一条指令之前

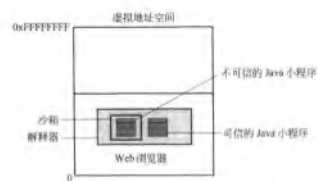


图 8.53 Web 浏览器可以解释 Java applet 程序

可以先对指令进行检查。这使得解释器有机会检查每条指令的地址是否有效。另外，系统调用也可以被捕获到，并且翻译过来。如何处理这些调用是一件与安全策略有关的事情。例如，如果一个 Java applet 是可信的（比如它来自于本地的硬盘），则解释器可以毫不犹豫地执行它的系统调用。然而，如果一个 applet 是不可信的（比如它是通过 Internet 而进入到本地机器中的），则解释器可能要将它封装到一个称为沙箱(sandbox)的环境中以便限制它的行为，并且捕捉住它要使用系统资源的企图。

当一个 applet 试图使用某一个系统资源时，它的调用被传递给一个安全监视器，以期能通过检查。监视器根据本地的安全策略对该调用进行检查，然后做出允许或者拒绝的决定。通过这种方式，就有可能让 applet 访问某些资源，但不是所有的资源。不幸的是，在现实中，这种安全模型工作得非常差，它总是不停地出现各种错误。

#### ActiveX 控件

ActiveX 控件是针对 Pentium 处理器的二进制程序，它也可以被嵌入到 Web 页面中。当浏览器遇到一个 ActiveX 控件时，它就要对该控件进行检查，看它是否应该被执行。如果通过了测试的话，则执行该控件。ActiveX 控件并不是解释执行的，也没有以任何方式被封装在沙箱环境中，所以，它具有跟其他用户程序一样的威力，因而有可能造成更大的危害。因此，所有的安全性仅在于一项简单的决定上，即决定是否运行 ActiveX 控件。

为了做出这项决定，Microsoft 选择的方法建立在代码签名(code signing)的思想基础上。每个 ActiveX 控件都伴随一个数字签名——由它的创建者利用公开密钥密码技术对代码的散列值进行签名。当浏览器看到一个 ActiveX 控件时，它首先验证该控件的签名，以确定它是否在传输途中被篡改了。如果签名是正确的，于是浏览器检查它的内部表，看该程序的创建者是否是可信的，或者是否存在一条信任链可以回溯到某一个可信的创建者。如果该创建者是可信的，则程序被执行，否则它不被执行。Microsoft 用来验证 ActiveX 控件的系统被称为认证码(Authenticode)。

将 Java 和 ActiveX 这两种方法做一下对比是非常有用的。若采用 Java 方法，则用户就无法确定一个 applet 是谁写的，但是，运行时的解释器可以保证，这个 applet 不会做出

该机器主人明确指明不允许做的事情。与此相反,若采用代码签名的方法,则浏览器不可能监视移动代码的运行时行为。如果移动代码来自于一个可信的源,并且在传输途中没有被修改过,那么它直接运行即可。这时就不必看这份代码是否是恶意的。如果程序员有意地编写这段代码来格式化硬盘,并且擦除闪存(flash ROM)中的内容,从而使计算机无法再重新启动,而且,如果程序员已经被证明是可信的,那么,这段代码将被运行,并且毁坏计算机(除非浏览器中已经禁止掉 ActiveX 控件)。

许多人觉得,信任一个不熟悉的软件公司是一件可怕的事情。为了演示这个问题,西雅图(Seattle)的一名程序员组建了一个软件公司,并且使它通过鉴定,成为一个可信任的公司,这是不难做到的。然后,他编写了一个 ActiveX 控件,该控件所做的事情是彻底关闭机器,然后他广泛发布此控件。它关闭了许多机器,但是这些机器只需重新启动即可,所以没有造成其他任何危害。这名程序员只是想把这个问题暴露给全世界。官方的回应是,撤销了这个 ActiveX 控件所使用的证书,从而结束了这一短暂的窘迫事件,但是,根本的问题仍然存在,并且有可能被恶意的程序员所利用(Garfinkel with Spafford, 2002)。成千上万的软件公司都有可能编写移动代码,要想监管所有这些公司是不可能的,所以,代码签名技术实际上是一场待之以发的灾难。

### JavaScript

JavaScript 没有任何正式的安全模型,但是各种有漏洞的实现的历史却已经很长了。每个厂商处理安全问题的方式都不相同。例如,Netscape Navigator 第 2 版使用了类似于 Java 模型的方式,但是到第 4 版时,它又放弃了原来的模型,改而使用代码签名模型。

基本的问题是,让外部代码在你的机器上运行,这等于在自找麻烦。从安全的角度来看,这就类似于邀请一个窃贼到你家里,然后企图谨慎地看守住他,以便不让他从厨房溜到客厅里。如果一旦发生意料之外的事情,你稍不留神,坏事就有可能发生。这里的安全压力在于,移动代码允许各种闪烁的图形和快速的交互过程,而且许多 Web 设计者认为,这些比安全性更加重要,尤其是当处于危险境地的是别人的机器而不是自家机器的时候。

### 病毒

病毒是另一种形式的移动代码。惟一与上述的移动代码例子不同的是,病毒并不是被邀请进来的。病毒与普通移动代码之间的区别是,病毒具有繁殖能力,它可以不断地复制自己。当一个病毒不管是通过 Web 页面,或者电子邮件的附件,或者其他某种方式到来时,它通常首先会感染硬盘上的可执行程序。当任何一个被感染的程序运行起来的时候,控制权被传递给病毒,它通常试图将自己传播到其他的机器上,例如,通过电子邮件将自身的副本寄送给受害者的电子邮件地址簿中的每一个人。有些病毒也会感染硬盘的启动扇区,所以当机器启动的时候,病毒就有机会运行。病毒已经变成了 Internet 上的一个大问题,并且导致了数十亿美元的损失。关于病毒并没有显而易见的解决方案。可能一个全新的操作系统将有助于抑制病毒,这个新的操作系统将建立在安全的微内核基础之上,并且严格地区分用户、进程和资源。

## 8.10 社会问题

Internet 和它的安全技术是一个使社会问题、公共政策和技术交汇在一起的广阔领域,在这个领域中,这种交汇往往又导致重要的后果。下面我们将简短地讨论三个领域:隐私、言论自由和版权。毋庸多说,我们这里只是涉猎表面而已。有关另外的读物,请参看 Anderson, 2001; Garfinkel with Spafford, 2002; and Schneier, 2000。Internet 本身也充满了各种材料。你只需在任何一个搜索引擎中输入诸如“privacy(隐私)”、“censorship(审查)”和“copyright(版权)”之类的关键字即可。另外,在本书的 Web 站点上你也可以看到一些链接。

### 8.10.1 隐私

人们有隐私权吗? 很好的问题。美国宪法的第四次修订版本中禁止政府在毫无恰当理由的情况下搜索人们的房屋、文件和私人财产,并且继续限制搜查许可证的发放条件(即在哪些情况下才应该发放搜查许可证)。因此,隐私被提到公开的议程中已经有 200 多年的历史了,至少在美国是如此。

在过去 10 年中所发生的变化有两个方面:政府监视老百姓更加容易了,而老百姓预防这种监视也更加容易了。在 18 世纪,政府为了搜索一个公民的文件,它必须派出一名骑警到该公民的农场去查看某些文档。这是一个非常繁琐的过程。现在,只要出示了搜查许可证,则电话公司和 Internet 供应商就会提供窃听装置。这使得警察们的生活更加容易了,而且也不用再冒着从马上掉下来的危险了。

密码学技术的发展改变了这一切。任何人只要下载并安装了 PGP,而且使用星际强度的密钥,他就可以保证宇宙间的任何一个人都不可能读取他的电子邮件,不管他有没有搜查许可证。政府很清楚这一点,他们并不喜欢这样的局面。对于政府来说,真正的隐私意味着他们将很难监视各种各样的罪犯,而且也很难监视新闻记者和各种持不同政见者。因此,有些政府限制或者禁止密码技术的使用或出口。例如,在法国,1999 年以前,所有的密码系统都是禁止的,除非已经把密钥交给政府。

法国并不是惟一个禁止密码技术的国家。1993 年 4 月,美国政府宣布它计划要制造一种硬件密码处理器,称为 **clipper chip**(剪切芯片),以作为所有网络通信的标准。据说用这种方式就可以保证公民的隐私。该计划同时也提到了,这种芯片采用一种被称为 **key escrow**(密钥托管)的方案(它允许政府可以访问所有的密钥),从而为政府提供了解密所有流量的能力。然而,该计划中也承诺,只有当拥有合法的搜查许可证时政府才使用这种能力。无需多说,随之发起了激烈的抗议,隐私倡导者们公开指责这整个计划,而法律执行官则极力赞美它。最终,政府取消了这个计划,并放弃了这种想法。

在电子边境基金会(Electronic Frontier Foundation)的 Web 站点 [www.eff.org](http://www.eff.org) 上,有大量关于电子隐私的信息可以参考。

## 匿名邮件中继器

PGP、SSL 和其他的一些技术使得任何两方之间有可能建立起安全的、真实的通信,从而避免受到第三方的监视和干扰。然而,有时候,隐私的真正含义是不要认证,实际上也就是使通信变成匿名的。对于点到点的消息、新闻组或者两者兼而有之的应用场合,人们可能非常期望这种匿名性。

现在我们来举一些例子。第一,生存在独裁政治体制下的持不同政见者通常希望进行匿名的通信,以避免被监禁或谋杀。第二,企业、教育、政府以及其他机构中的不正当行为通常被一些勇敢的正义人士揭发出来,而这些揭发者通常希望能保持匿名,以免遭到报复。第三,持有非主流的社会、政治或宗教观点的人可能希望在跟其他人通过电子邮件或者新闻组进行通信时不暴露他们的身份。第四,人们可能希望在新闻组中匿名地讨论酗酒、精神病、性骚扰、虐待儿童,或者受迫害的少数派的成员信息。当然,除此以外,还有许多其他的例子。

我们来考虑一个特殊的例子。在 20 世纪 90 年代,一个非传统宗教组织的有些批评人士利用一个匿名邮件中继器(anonymous remailer),将他们的观点张贴到一个 USENET 新闻组中。这个服务器(即匿名邮件中继器)允许用户创建假的名字,并且给服务器发送邮件,然后,该服务器使用这个假名字将邮件寄送或者张贴出去,所以,没有人能够知道消息的确切来源。他们的有些帖子泄露了宗教组织所宣称的内部机密以及受版权保护的文档。该宗教组织对此做了回应,它告诉本地的权力机构,它的内部机密被暴露,并且它的版权受到了侵犯,这两者都是转发服务器所在地的犯罪行为。这起事件被闹到法庭上,最终,服务器运行商被迫将那些隐含了发贴人真实身份的历史记录信息交出来(顺便提一下,这并不是第一起因为有人泄露秘密而惹怒一个宗教的事件:1536 年 William Tyndale 因为将圣经翻译成英文而被绑在树桩上烧死)。

在 Internet 社会中,有相当一部分地区笼罩着这种侵犯隐私行为的阴影。每个人的结论是,那些保存了从真实邮件地址到假名字之间映射关系的匿名邮件中继器(称为第一类型邮件中继器)不再有任何价值。这种情形刺激了很多人来设计可抵抗法庭传票攻击的匿名邮件中继器。

这些新的邮件中继器通常被称为翻译型的邮件中继器(cypherpunk remailer),其工作过程如下所述。用户产生一个电子邮件消息,其中含有完整的 RFC 822 头(当然除了“From:”以外),然后用邮件中继器的公钥对它加密,并发送给邮件中继器。在邮件中继器上,外部的 RFC 822 头被剥掉,内容被解密出来,然后里边的消息被重新发送出去。邮件中继器没有任何账号,也不维护日志,所以,即使邮件转发服务器事后被查抄,它也不会保留任何从它这里经过的消息的痕迹。

许多希望发送匿名消息的用户让他们的请求穿越多个匿名邮件中继器,如图 8.54 所示。这里,Alice 想要给 Bob 发送一张完完全全匿名的情人节卡片,所以她使用三个邮件中继器。她写好一个消息 M,再把消息放在一个含有 Bob 邮件地址的 RFC 822 头中。然后,她用 3 号邮件中继器的公钥  $E_3$  对整个消息进行加密(如图中水平阴影线所示)。此时她又在整个消息的外面套上一个明文头,其中包含 3 号邮件服务器的邮件地址。这是图



中 2 号和 3 号邮件中继器之间所显示的消息。

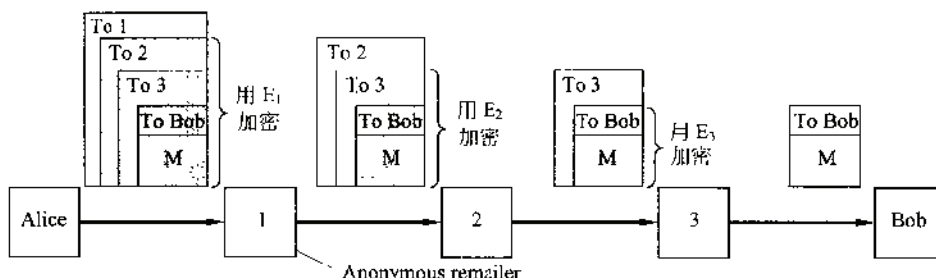


图 8.54 Alice 如何使用 3 个邮件中继器来给 Bob 发送一个消息

然后,她用 2 号邮件中继器的公钥  $E_2$  来加密这个消息(如图中垂直阴影线所示),并且在外面套上一个含有 2 号中继器邮件地址的明文头。这是图 8.54 中 1 号和 2 号邮件中继器之间所显示的消息。最后,她再用 1 号邮件中继器的公钥  $E_1$  来加密整个消息,并且套上一个含有 1 号中继器邮件地址的明文头。这是在图 8.54 中 Alice 右侧所显示的消息,而且这也是她真正传送的消息。

当消息到达 1 号邮件中继器时,首先外部的头被剥掉。消息体被解密,然后将其中的邮件消息发送给 2 号邮件中继器。在其他两个邮件中继器上也执行类似的步骤。

任何人要想从最终的消息追踪到 Alice 将非常困难,尽管如此,许多邮件中继器还采取了附加的安全预防措施。例如,它们可能将消息保留一段随机的时间,在消息的末尾增加或者删除一些无关紧要的信息,对消息重新排序;它们所做的这一切动作都是为了使得“辨别一个邮件中继器的输出消息对应于哪一个输入消息”变得更加困难,即为了抵抗流量分析攻击。有一个系统展示了匿名电子邮件技术的最新发展状况,关于该系统的描述,请参考(Mazières and Kaashoek, 1998)。

匿名需求并不仅限于电子邮件。在实践中,也存在一些允许匿名 Web 冲浪的服务。用户可以对他的浏览器进行配置,让浏览器使用一个匿名代理。以后,所有的 HTTP 请求都转向匿名代理,由它请求真正的页面,并送回给用户。Web 站点将匿名代理看作请求的源,而不是把用户看作请求的源。只要匿名代理没有记录日志,则事后没有人知道谁请求了哪些页面。

### 8.10.2 言论自由

隐私涉及到的内容是,每个人希望限制其他人能够观察到自己的信息。第二个关键的社会问题是言论自由,以及它的对立面:审查。所谓审查是指政府希望限制每个公民所能阅读和发表的内容。由于 Web 包含了数千万以上的页面,所以,Web 成了审查员的天堂。根据政治体制的种类和意识形态的不同,Web 站点如果包含了以下的内容则可能会被禁止:

- (1) 少儿不宜的材料。
- (2) 对各种种族、宗教、性别或者其他组织有歧视倾向。
- (3) 有关民主和民主价值的信息。



(4) 与政府的版本有抵触的历史事件报道。

(5) 介绍撬锁、制造武器、加密消息等技术的参考材料。

通常的响应措施是禁止这样的坏站点。

有时候结果是不可预测的。例如,有些公开的图书馆在它们的计算机上安装了 Web 过滤器,并且阻塞色情站点,以便允许儿童使用。这些过滤器一方面禁止黑名单中所有的站点,另一方面,在显示页面之前也要检查是否其中包含脏字。在弗吉尼亚州 Loudoun 县的一个案例中,当一名赞助人通过 Web 搜索有关乳房癌的信息时,过滤器阻止这一搜索请求,因为它看到了“乳房”一词。这名赞助人控告了 Loudoun 县。然而,在加利福尼亚州的 Livermore,一名赞助人控告公开图书馆的理由是他们没有安装过滤器,所以她那 12 岁的儿子在那里被抓到正在浏览色情内容。图书馆该怎么办呢?

许多人都知道,万维网(World Wide Web)是一个全球范围的 Web,它覆盖了整个世界。然而,对于哪些内容可以出现在 Web 上,并不是所有的国家都有统一的规定。例如,在 2000 年 11 月,法国的法院命令美国加利福尼亚州的 Yahoo 公司阻止法国的用户观看 Yahoo 的 Web 站点上有关纳粹纪念物的拍卖活动,因为按照法国的法律,拥有这样的物品是违法的。Yahoo 向美国的法院提出上诉,而美国法院没有这样的禁令,因此问题就变成了谁的法律适用于哪里,这在 Web 上是很难解决的。

你可以继续想象。如果美国犹他州的一个法院命令法国阻塞所有做酒生意的 Web 站点,因为犹他州关于酒的法律非常严格,这些站点不符合它的法律,那么会怎么样呢?假设中国要求所有涉及到民主的 Web 站点都被禁止,因为它们不符合国家的利益。伊朗关于宗教的法律适用于更加自由的瑞典吗?沙特阿拉伯能够阻塞所有涉及妇女权利的 Web 站点吗?整个事情将变成一个真正的潘多拉盒子。

一条来自 John Gilmore 的很中肯的评论是“网络将审查行为看作一种破坏,并且设法绕过它。”要想了解一个具体的实现,请考虑 eternity 服务(Anderson, 1996)。它的目标是保证已被发表的信息不可能被取消或改写。为了使用 eternity 服务,用户需要指定自己的材料将被保留多久,并且支付与材料的长度和保留时间成正比的费用,然后上载材料。之后,没有人可以移除或者编辑这些材料,即使上载者也不例外。

这样的服务应该如何来实现呢?最简单的模型是使用一个对等系统(peer-to-peer system),所存放的文档被放在几十台参与到对等系统中的服务器上,并且每台服务器获得一部分费用,以此来鼓励服务器加入到对等系统中。系统应该使这些服务器尽可能地遍布到许多法律的管辖范围。10 台随机选择的服务器的列表被安全地保存在多个地方,所以,即使有些地方被攻破了,其他的仍然还在。如果一个机构下定决心要毁掉某一个文档,那么它可能永远也无法确定自己是否找到了该文档所有的副本。该系统可以被做成自修复的,也就是说,如果它知道有些副本已经被毁坏了,则剩下的站点将设法找到新的藏宝处,以便替换那些被毁坏的副本。

Eternity 服务是第一个能对抗审查的系统方案。自那以后,还有其他一些系统也被提出来了,而且有几个已经被实现了。各种新的特性也被加入进来,比如加密、匿名性和容错性。保存在系统中的文件通常被分割成多个片断,每个分段被保存在许多台服务器上。这样的系统有 Freenet(Clarke et al., 2002)、PASIS(Wylie et al., 2002)和 Publius

(Waldman et al., 2000)。有关其他工作的报告请参考(Serjantov, 2002)。

现在的发展趋势是,越来越多的国家试图对无形作品的出口进行管制,所谓无形的作品通常包括 Web 站点、软件、科技论文、电子邮件、电话服务台等。即使像英国这样有几个世纪之久的言论自由传统的国家,现在也在考虑制定更加严格的法律,例如,剑桥大学的英国教授和他的外国留学生之间的技术讨论也被定义为需要得到政府许可的、受管制的出口行为(Anderson, 2002)。无需多言,这样的政策是极有争议的。

#### 信息隐藏学(Steganography)

在审查制度比较严格的国家中,持不同政见者通常企图使用技术手段来躲避审查。密码学技术使得秘密消息仍然可以被发送出去(不过这样做可能不合法),但是如果政府认为 Alice 是一个坏人,那么,仅仅凭着 Bob 与她进行通信这一事实就可能使他也归到坏人这一类别中,因为即使高压政府中缺乏数学家,他们也会理解传递闭包的概念。匿名邮件中继器也许有所帮助,但如果它们被禁止只能在国内通信,并且通向国外中继器的消息都要求政府的出口许可,那么它们也不会再有任何帮助了。但是 Web 仍然可以。

希望进行保密通信的人总是试图隐藏所有的通信痕迹,包括发生了通信本身这一事实。隐藏消息的学科被称为信息隐藏学(steganography),它来源于希腊语中的词“covered writing”。实际上,古希腊人自己也使用这种技术。希罗多德(Herodotus, 希腊历史学家)写到了这样一个故事:一名将军剃光了一个信使的头,再将消息刺在他的头皮上,等到他的头发长出来之后再送他出去。现代的技术在概念上也是一样的,只不过它们有更高的带宽和更低的延迟。

作为一个例子,请考虑图 8.55(a)。这幅照片是本书作者在肯尼亚拍摄的,图中有三头斑马凝望着棵橡胶树。图 8.55(b)看起来也有同样的三头斑马和一棵橡胶树,但是它加入了额外的有趣的东西。它内嵌了 5 部莎士比亚戏剧的完整文本:Hamlet(哈姆雷特)、King Lear(李尔王)、Macbeth(麦克白)、The Merchant of Venice(威尼斯商人)和 Julius Caesar(凯撒大帝)。这些喜剧合起来总共超过了 700KB 的文本。



图 8.55

(a) 三头斑马和一棵树; (b) 三头斑马、一棵树和 5 部莎士比亚戏剧的完整文本

这种隐藏信道是如何做到的呢? 原始的彩色图像有  $1024 \times 768$  个像素。每个像素是

由三个 8 位整数构成的,分别对应于该像素的红、绿和蓝三种颜色的强度。像素的颜色是由这三种颜色经线性叠加而形成的。这里的信息隐藏编码方法使用每个 RGB 颜色值的最低位作为隐藏信道。因此,每个像素有 3 位秘密信息空间:红色值中 1 位、绿色值中 1 位,以及蓝色值中 1 位。对于像例子中大小的一幅图像,它可以存储至多  $1024 \times 768 \times 3$  位或者 294 912 字节的秘密信息。

这 5 部戏剧的完整文本以及一份简短的评论累加起来总共有 734 891 字节。首先使用一个标准的压缩算法将所有这些文本信息压缩到大约 274KB 大小。然后利用 IDEA 算法对压缩之后的输出进行加密,接下来再将密文插入到每个颜色值的最低位中。当这幅图像被显示出来时,隐藏信息的存在是完全不可见的(实际上,插入的信息是不可能看得到的)。在一幅大的、全彩色的图像中,隐藏信息的存在同样是不可见的。肉眼不可能很容易地区分 21 位彩色和 24 位彩色。

在低分辨率下观看两幅黑白图像并不能体现出这项技术是多么有效。为了更好地感觉信息隐藏技术的效果,本书作者专门准备了一个演示,其中包括图 8.55(b)中内嵌了 5 部戏剧的全彩色高分辨率图像。你可以从本书的 Web 站点上找到这个演示,其中包括用于在图像中插入和提取文本的工具。

为了将信息隐藏技术用于未能被发现的通信中,持不同政见者可以创建一个 Web 站点,并在 Web 站点上放置各种无任何政治问题的图片,比如领导人的照片、本地的体育赛事、电影、电视明星,等等。当然,这些图片中包含有隐藏的消息。如果这些消息首先被压缩,然后再被加密的话,则即使有人怀疑存在这些消息,他们也很难将这些消息与白噪声区分开。当然,这些图像应该是重新扫描的;如果直接从 Internet 上复制一个图片,再改变其中的一些位,这将会泄漏其中的机密。

隐藏消息的载体并不仅限于图像。音频文件也可以工作得很好。视频文件有更加巨大的隐藏信息带宽。即使 HTML 文件中的布局和标签顺序也可以携带信息。

虽然我们是在讨论言论自由的上下文环境中来介绍信息隐藏学的,但是实际上它还有许多其他的用途。一种常见的用途是,图像的所有者可以编制一些秘密的消息来表达他们的版权所有声明。如果这样的图像被别人偷走并放在一个 Web 站点上,则合法的所有者可以在法庭上出示隐藏的消息,以此来证明这到底是谁的图像。这项技术被称为数字水印(watermarking)。(Piva et al., 2002)中讨论了这项技术。

有关信息隐藏学的更多信息,请参看 Artz, 2001; Johnson and Jajoda, 1998; Katzenbeisser and Petitcolas, 2000; and Wayner, 2002。

### 8.10.3 版权

隐私和审查是“技术与公共政策彼此消长”的两个领域。第三个领域是版权。版权(copyright)是对 IP(Intellectual Property, 知识产权)的创造者的一种授权,这样的创造者包括作家、画家、作曲家、音乐家、摄影师、电影摄影师、舞蹈动作设计师,等等,这种授权是排他性的,以允许他们在一定长的时间里充分使用他们的 IP,典型的时间长度为作者的一生再加上 50 年,如果是法人所有权的话,则为 75 年。当一件作品的版权过期以后,它便变成公有财产,任何人都可以随意地使用或者出售。例如, Gutenberg Project(古滕堡

工程, [www.promo.net/pg](http://www.promo.net/pg)) 在 Web 上放置了数千件公有的作品(比如莎士比亚、马克·吐温、狄更斯等人的作品)。1998 年, 美国国会接受好莱坞的请求, 将作品在美国的版权延长了 20 年, 好莱坞声称, 如果不延长的话, 则没有人再愿意制作任何作品了。与此形成鲜明对照的是, 专利只持续 20 年, 但人们仍然在发明各种事物。

当 Napster(一个交换音乐的服务)拥有 5 千万成员时, 版权便成了引人注目的焦点。虽然 Napster 并没有实际复制任何音乐, 但是法庭认为, 它拥有的中心数据库记录了谁有哪一首歌, 这有助于侵犯版权, 也就是说, 他们帮助其他人侵犯版权。虽然没有人正式地宣称版权是一个不好的概念(不过, 许多人声称, 这个术语太长了, 它更偏向于大的公司而不是公众), 但是, 下一代音乐共享技术已经引发了一些主要的道德问题。

例如, 考虑在一个对等网络中, 人们共享合法的文件(公有的音乐、家庭视频片断、非机密的宗教宣传册, 等等), 可能还有一些受版权保护的内容。假设每个人通过 ADSL 或者有线电视全天候地保持在线连接。每台机器都对自己硬盘上的内容做了索引, 同时还有一个其他成员的列表。当有人要查找某一特定的作品时, 他可以随机选择一个成员, 看它是否有这件作品。如果没有的话, 他可以检查此人列表中的所有成员, 以及他们的列表中的所有成员, 以此类推下去。计算机很擅长做这种工作。一旦找到了作品之后, 请求者只要复制过来即可。

如果这件作品是受版权保护的, 则请求者便侵犯了版权(不过, 对于跨国家的传输, 到底应用谁的法律, 这个问题尚不清楚)。但是, 对于提供作品的人又怎么样呢? 你付钱购买了音乐并且将它合法地下载到自己的硬盘上, 但别人有可能在你的硬盘上找到这件音乐作品, 那么, 你的所作所为算是一种犯罪行为吗? 如果在乡下你有一间没锁门的小屋, 一个 IP 小偷带着一台笔记本电脑和一台扫描仪, 偷偷地溜进来复制了一本版权书, 然后又溜出去了, 那么, 你是否犯了“未能保护他人版权”的罪行呢?

但是, 在版权的前沿阵地上, 还有更多潜在的麻烦。好莱坞和计算机工业界之间有一场巨大的斗争正在进行。前者希望对所有的知识产权都进行严格的保护, 而后者并不愿意成为好莱坞的警察。1998 年 10 月, 美国国会通过了 **DMCA (Digital Millennium Copyright Act, 数字千禧年版权法案)**, 这使得“破解或绕开一件版权作品中出现的任何保护机制, 或者告诉别人怎样破解或绕开”这样的行为也是一种犯罪。同样地, 欧盟也制定了类似的法律。几乎没有人认为远东的盗版商复制那些受版权保护的作品应该是允许的, 但是, 许多人认为, DMCA 完全改变了在版权所有者利益和公众利益之间的平衡。

举一个相关的例子。2000 年 9 月, 一个音乐界团体负责建立了一个不可破解的系统, 用于在线销售音乐。它同时发起了一场竞赛, 邀请人们来破解这个系统(对于任何一个新的安全系统来说, 这种做法是相当正确的)。由 Princeton 大学的 Edward Felten 教授领导的一个小组聚集了来自多所大学的安全研究人员, 他们接受了这个挑战, 并且最终攻破了系统。然后他们写了一篇论文来介绍他们的研究结果, 并且将论文提交给 USENIX 的安全会议, 经过同行评议之后, 这篇论文被接受了。在论文发表前夕, Felten 收到了来自美国唱片业协会的一封信, 信中威胁他, 如果他们坚持要发表这篇论文的话, 则唱片业协会将在 DMCA 的框架下控告论文的作者。

他们的回应是, 起草一份诉讼文件请求联邦法庭来裁定“发表关于安全研究的科技论

文是否仍然是合法的”。慑于联邦法庭可能会做出不利的裁定,唱片业协会撤回了它的威胁,法庭也就驳回了 Felten 的上诉。毫无疑问,唱片工业界肯定被它的案例中暴露出来的弱点所刺激了:他们邀请人们来攻击他们的系统,然后又威胁要起诉某些接受了挑战的人。由于威胁被撤回了,所以论文得以发表(Craver et al., 2001)。因此,新一轮的对峙已是必定无疑了。

一个相关的问题是合理使用原则(fair use doctrine)的度,所谓合理使用原则,是在各个国家中由法庭裁决建立起来的。这条原则说明了,一件版权作品的购买者拥有一定的、受限制的权利来复制这件作品,包括引用部分内容作为科学研究之用、用作学校或学院的教材,以及在某些情况下,当原始介质不能使用时可以做一些副本以供自己使用。合理使用的判断准则包括(1)是否出于商业目的;(2)被复制部分所占的百分比;(3)这种复制对于该作品销售的影响。由于 DMCA 和欧盟内部类似的法律禁止破解或者绕开作品的复制保护方案,所以这些法律也禁止合法的合理使用。实际上,DMCA 将一些历史上原本属于用户的权利给了内容销售商,从而赋予销售商更多的权利。一场大的冲突已经不可避免了。

尽管 DMCA 改变了版权所有者和用户之间的平衡,然而,另外还有一些工作使 DMCA 也相形见绌,其中之一是由 Intel 和 Microsoft 领导的 TCPA(Trusted Computing Platform Alliance,可信计算平台联盟)。TCPA 的思想是让 CPU 芯片和操作系统谨慎地通过各种方法来监视用户的行为(比如播放盗版音乐),并禁止有问题的行为。甚至该系统允许内容的所有者可以远程操纵用户的 PC,以便当他们认为有必要的时候改变规则。无需多说,这种方案的社会影响是极为广泛的。工业界最终注意到了安全性,这是好事,但令人惋惜的是,它把所有的焦点都瞄准在加强版权法上,而不是对付病毒、破坏者、入侵者和其他一些大多数人都很关心的安全问题上。

简而言之,在接下来的几年中,立法者和律师们将不停地忙于平衡版权所有者的经济利益与公众的利益。数字空间与社会空间并没有不同:它经常挑起一个组与另一个组之间的竞争,从而导致权力斗争、诉讼,以及(希望)最终是某种形式的决议案,至少在某一新的破坏技术出现之前是这样的。

## 8.11 本章小结

密码学是一个可被用来保密信息以及确保信息完整性和真实性的工具。所有现代的密码系统都建立在 Kerckhoff 原则的基础上,即算法是公开的,但密钥是保密的。许多密码学算法使用了内含置换和转置的复杂变换,从而将明文变换成密文。然而,如果量子密码学变得切实可行的话,则使用一次一密方法也许真的可以实现牢不可破的密码系统。

密码算法可以被分成对称密钥算法和公开密钥算法。对称密钥算法将数据位通过一系列用密钥作为参数的轮变换,从而将明文变成密文。三重 DES 和 Rijndael(AES)是目前最为常见的对称密钥算法。这些算法可被用在电子代码簿模式中,也可以被用在密码块链接模式、流密码模式、计数器模式,以及其他一些模式中。

公开密钥算法有这样的特性:加密和解密使用不同的密钥,并且从加密密钥不可能



推导出解密密钥。这些特性使得公开一个密钥(即公钥)成为可能。主要的公开密钥算法是 RSA,它的强度建立在分解大整数非常困难的基础之上。

许多法律的、商业的和其他的文档需要有签名。因此,人们设计了许多种数字签名方案,有的使用对称密钥算法,有的使用公开密钥算法。通常,需要被签名的消息首先使用像 MD5 或者 SHA-1 这样的算法来计算散列值,然后对此散列值进行签名,而不是对原始的消息进行签名。

公钥管理可以通过证书来完成,所谓证书是指将一个安全个体与一个公钥绑定到一起的文档。证书需要由可信的权威机构来签名,或者由某一个得到可信权威机构批准的实体来签名(可以如此递归下去)。在使用时,这条链的根必须已被提前获取到,通常浏览器已经内置了许多根证书。

这些密码学工具可以被用来保护网络的流量。IPSec 运行在网络层上,它加密主机之间的分组流。防火墙可以屏蔽进出一个组织的流量,屏蔽的依据通常是所用的协议和端口。虚拟私有网可以模拟老式的租用线路的网络,从而提供某些期望的安全特性。最后,无线网络需要好的安全性,但 802.11 的 WEP 并没有提供足够的安全性,不过 802.11i 应该会有大的改进。

当两方建立一个会话时,他们必须相互认证对方的身份,如果有必要的话,还要建立一个共享的会话密钥。现在已经有了许多认证协议,其中有的使用了可信第三方,还有一些使用了 Diffie-Hellman、Kerberos 或者公开密钥密码学。

将本章中学习过的技术组合起来就可以实现安全的电子邮件。例如,PGP 首先压缩邮件消息,然后利用 IDEA 对消息进行加密。PGP 利用接收方的公钥来加密 IDEA 密钥,再将它发送出去。此外,它也计算消息的散列值,并且将签过名的散列值发送出去以便验证消息的完整性。

Web 安全也是一个重要的话题,起点是安全的命名机制。DNSsec 提供了一种预防 DNS 欺骗的方法,利用自证明的名字也可以预防 DNS 欺骗。大多数电子商务 Web 站点使用 SSL 来为客户和服务端建立起安全和真实的会话。另外,有许多技术被用来应对移动代码,特别是沙箱和代码签名技术。

Internet 引发了许多导致技术与公共政策强烈交汇的问题。这样的领域包括隐私、言论自由和版权。

## 习 题

1. 请破解下面的单字母表密码。明文仅由字母组成,它是从 Lewis Carroll 的诗中摘录下来的名句:

```
kfd ktbd fzm eubd kfd pzyiom mztix ku kzyg ur bzha kfthcm
ur mftnm zhx mfudm zhx mdzythc pzq ur ezsscedm zhx gthcm
zhx pfa kfd mdz tm sutythc fuk zhx pfdkfdi ntem fzld pthcm
sok pztz z stk kfd uamkdim eitdx sdruid pd fzld uoi efzk
ruu mubd ur om zid uok ur sidzki zhx zyy ur om zid rzk
```



hu foia mztz kfd ezindhkdi kfda kfzhgdx fit boef rui kfzk

2. 破解下面的柱形转置密码。明文是从一本流行的计算机教材中摘录的,所以“computer”是一个可能的单词。明文全是由字母组成的(没有空格)。为了方便阅读,密文被分割成 5 个字符的块。

aaauan cvlre rurnn dlme aeepb ytust iceat npmey iiego gorch stsoc  
nntii imiha oolpa gsivt upsit lbolr otoex

3. 从图 8.4 的密文中,找到一个能产生文本“Donald Duck”的 77 位一次性密钥。

4. 量子密码学要求有一个能根据需要激发单个光子(携带 1 位信息)的光子枪。在这个习题中,请计算一下,在一条 100Gbps 的光纤链路上,一位携带多少个光子。假设光子的长度等于它的波长,并且在这个习题中,波长为  $1\mu\text{m}$ 。光纤中的光速为  $20\text{cm/ns}$ 。

5. 假设一个系统使用了量子密码技术,如果 Trudy 能够捕获并重新生成光子,那么,它将会得到一些错误的位,从而导致在 Bob 的一次性密钥中也会出现错误。请问,平均而言,Bob 的一次性密钥中错误的位占多大的比例?

6. 一条基本的密码学原则声明了所有的消息必须有冗余度。但是,我们也知道,消息中的冗余可以帮助一个入侵者来识别一个猜测的密钥是否正确。请考虑两种冗余形式。第一,明文的前  $n$  位包含一种已知的模式;第二,消息的最后  $n$  位包含了该消息的一个散列值。从安全的角度来看,这两种形式是等价的吗?请解释你的答案。

7. 在图 8.6 中,P 盒和 S 盒交替出现。尽管从美学角度上来看这种安排是非常令人愉悦的,但是,比起“首先通过所有的 P 盒再通过所有的 S 盒”这种形式,它会更加安全吗?

8. 假设 DES 的明文中只包含大写 ASCII 字母,以及空格、逗号、句号、冒号、回车符和换行符,请根据这样的知识来设计一种破解 DES 的方法。关于明文的奇偶校验位并不知道。

9. 在正文中我们计算过,假设一个处理器每  $10^{-12}$  秒能够分析一个密钥,那么,一台具有  $10^9$  个处理器的密码破解机器将需要  $10^{10}$  年时间才能破解 128 位版本的 AES。然而,当前的计算机可能有  $10^{24}$  个处理器,并且每  $1\text{ms}$  可以分析一个密钥,所以,我们需要在性能上提高  $10^{15}$  倍才能获得可破解 AES 的机器。如果摩尔定律(Moore's law,即每 18 个月计算能力翻一番)持续有效的話,则需要多少年才能制造出这样的机器?

10. AES 支持 256 位的密钥。AES-256 有多少个密钥?看一看你是否可以在物理、化学或者天文学中找到同样规模的数值。你可以使用 Internet 来帮助查找大的数值。根据你的调查研究,请给出一个结论。

11. 假设一条消息已被使用 DES 和密码块链接模式进行了加密。块  $C_i$  中的一位密文在传输过程中被偶然地从 0 变成了 1。请问结果将有多少明文被弄乱?

12. 现在再来考虑密码块链接模式。这次不再是一个 0 位被转换成了 1 位,而是一个额外的 0 位被插入到块  $C_i$  之后的密文流中。同样地,请问结果将有多少明文被弄乱?

13. 请从传输一个大文件所需要的加密操作次数的角度,来比较一下密码块链接模式与密码反馈模式。哪一个效率更高?高多少?

14. 使用 RSA 公开密钥密码系统,并且  $a=1, b=2$ ,以此类推,
- (u) 如果  $p=7, q=11$ ,请列出 5 个合法的  $d$  值。
  - (v) 如果  $p=13, q=31, d=7$ ,请找出  $e$ 。
  - (w) 使用  $p=5, q=11, d=27$ ,请找到  $e$ ,并加密“abcdefghij”。
15. 假设一个用户 Maria 发现了她的私有 RSA 密钥( $d_1, n_1$ )与另一个用户 Frances 的公开 RSA 密钥( $e_2, n_2$ )相同。那么, Maria 应该考虑改变她的公开密钥和私有密钥吗?请解释你的答案。
16. 请考虑如图 8.15 中所示的计数器模式的用法,但是  $IV=0$ 。一般而言,这里使用 0 是否会危及到该密码的安全性?
17. 图 8.18 中的签名协议有下述的弱点。如果 Bob 崩溃了,他可能会丢失 RAM 中的内容。这会引起哪些问题?他又该如何来防止这些问题呢?
18. 在图 8.20 中,我们看到了 Alice 如何向 Bob 发送一条签过名的消息。如果 Trudy 替换了  $P$ ,则 Bob 能检测出来。但是,如果 Trudy 同时替换了  $P$  和签名,则会怎么样呢?
19. 数字签名有一些由于懒惰的用户而引起的潜在弱点。在电子商务交易中,有可能会形成一份合同,并要求用户对它的 SHA-1 散列值进行签名。如果用户没有真正对这份合同和对应的散列值进行验证的话,那么他有可能无意中签了另一份完全不同的合同。假设黑手党试图利用这个弱点来套一笔钱。他们建立了一个付费的 Web 站点(比如色情或者赌博站点,等等),并且要求新的顾客提供信用卡号码。然后,他们发送了一份合同,说明顾客希望使用他们的服务并通过信用卡来支付费用,而且他们请顾客对这份合同进行签名,实际上,他们知道大多数顾客根本不检查合同和散列值的一致性,就会对散列值进行签名。请说明黑手党如何从一家合法的 Internet 珠宝商处购买一批钻石,并且将费用记到那些轻信的顾客身上。
20. 一个数学班有 20 名学生。请问,至少两名学生具有相同生日的概率为多少?假设这个班上没有人在闰日(即 2 月 29 日)出生,所以,总共有 365 种可能的生日。
21. 当 Ellen 向 Marilyn 坦白了自己在 Tom 的终身职位事件中所做的欺骗行为以后, Marilyn 决定用一台录音机录下她想要发送的消息的内容,然后让她的新秘书仅仅完成键盘输入工作,以避免再发生这样的问题。然后 Marilyn 打算在消息被输入以后,她在自己的终端上检查这些消息,以确保消息中的内容是本意。请问,新秘书仍然能够使用生日攻击来伪造消息吗?如果能的话,她该怎么办?提示:她能够。
22. 考虑图 8.23 中 Alice 未能获得 Bob 公钥的情形。假设 Bob 和 Alice 已经共享了一个秘密密钥,但是 Alice 仍然想要 Bob 的公钥,现在 Alice 有办法安全地获得 Bob 的公钥吗?如果可以的话,她该怎么办?
23. Alice 想要利用公开密钥密码技术与 Bob 进行通信。她与某个人建立了一个连接,她希望这个人就是 Bob。她请他出示他的公钥,然后他以明文方式将公钥和一个由根 CA 签名的 X.509 证书一起发送给 Alice。Alice 为了证明她在跟真正的 Bob 进行通信,需要执行哪些步骤?假设 Bob 并不关心他在跟谁通话(比如, Bob 是某一种公共的服务)。

24. 假设系统使用了一个基于树状 CA 层次结构的 PKI。Alice 希望与 Bob 进行通信;在与 Bob 建立起通信信道以后,她接收到 Bob 发送过来的一个证书,该证书的签名者是一个 CA,我们称为 X。假设 Alice 从来没有听说过 X,那么,她该执行哪些步骤来验证自己确实是在跟 Bob 通话?

25. 如果有一台机器位于一个 NAT 盒的后面,那么,若 IPSec 使用了 AH,则它能够在传输模式下使用吗?请解释你的答案。

26. 与用 RSA 来签名 SHA-1 散列值的做法相比较,请说出 HMAC 的一个优点。

27. 请说出一个理由,为什么可以对防火墙进行配置以检查进来的流量。再说出一个理由,为什么也可以对防火墙进行配置以检查出去的流量。你认为这两种检查可能会成功吗?

28. 图 8.31 显示了 WEP 分组的格式。假设校验和是 32 位,它是将净荷中所有的 32 位字 XOR 在一起而计算得来的。同时也假设 RC4 的问题已经改正了,比如我们用一个无弱点的流密码来替换它;同时也假设 IV 已被扩展到 128 位。在这种情况下,入侵者还有办法来监听或者干扰流量而不会被检测到吗?

29. 假设一个组织使用了 VPN,以便通过 Internet 将它的多个站点安全地连接起来。对于这个组织中的一个用户 Jim 来说,为了与同一组织中的另一个用户 Mary 进行通信,Jim 是否还需要使用加密或者其他的安全机制?

30. 对图 8.34 所示的协议中的一条消息做一点小的修改,以使它能够抵抗反射攻击。请解释为什么你的修改能起作用。

31. Diffie-Hellman 密钥交换协议可以被用于在 Alice 和 Bob 之间建立一个秘密密钥。Alice 发送给 Bob 的是(719, 3, 191)。Bob 的回应是(543)。Alice 的秘密数  $x$  是 16。请问秘密密钥是什么?

32. 如果 Alice 和 Bob 从未碰过面,也没有任何共享的秘密,并且都没有证书,然而,他们可以使用 Diffie-Hellman 算法来建立一个共享的秘密密钥。请解释为什么他们很难抵抗中间人攻击。

33. 在图 8.39 的协议中,为什么 A 以明文方式连同加密的会话密钥一起被发送出去?

34. 在图 8.39 的协议中,我们曾经指出,每条明文消息都以 32 个 0 位作为开始是一个安全风险。假设每条消息的开始处是一个随每个用户而改变的随机数,实际上,这个随机数相当于用户与 KDC 之间的第二个秘密密钥。这样能够消除已知明文攻击吗?为什么?

35. 在 Needham-Schroeder 协议中,Alice 生成了两个质询  $R_A$  和  $R_{A2}$ 。这看起来有点多余了。难道只用一个就不能完成同样的工作吗?

36. 假设一个组织使用 Kerberos 作为认证协议。请从安全性和服务可用性的角度来看,如果 AS 或者 TGS 停机了,则会有什么样的影响?

37. 在图 8.43 的公开密钥认证协议中,在消息 7 中, $R_B$  是用  $K_S$  来加密的。这里的加密操作有必要吗?或者说,是否以明文方式送回来就足够了?请解释你的答案。

38. 使用磁卡和 PIN 码的销售点终端有一个致命的缺陷:一个恶意的商家可以修改

他的读卡器,以便将用户磁卡上的所有信息以及 PIN 码捕捉并保存下来,从而将来可以寄送(post)另外的(伪造的)交易。下一代销售点终端使用的卡上将具有完全的 CPU、键盘和微显示器。请为这种系统设计一个协议,要让恶意的商家无法攻破系统。

39. 请说出两个理由,为什么 PGP 要压缩消息。

40. 假设 Internet 上的每个人都使用 PGP,请问,一条 PGP 消息能被发送到任意一个 Internet 地址,并且被所有涉及到的人正确地解码吗?请讨论你的答案。

41. 图 8.47 中显示的攻击遗漏了一个步骤。如果仅仅为了使欺骗能够奏效,则这一步骤并不必要,但是,加入这一步以后,可能会减小事后被怀疑的可能性。请问漏掉的步骤是什么?

42. 正文中已经提到了可以让 DNS 服务器采用随机的 ID 而不是计数器的方案,来对抗采用 ID 预测的 DNS 欺骗。请讨论这种做法的安全性。

43. SSL 数据传输协议涉及到两个临时值和一个预设主密钥。请问,使用这两个临时值有什么价值(如果有的话)?

44. 图 8.55(b)的图像包含了莎士比亚 5 部戏剧的 ASCII 文本。请问,有可能在斑马中隐藏音乐而不是文本吗?如果可能的话,该怎么做?在这幅照片中你能够隐藏多少音乐?如果不能的话,请问为什么不能?

45. Alice 是一个大量使用第一类型匿名邮件中继器的用户。她在自己喜欢的新闻组 alt.fanclub.alice 上张贴了许多消息,每个人都知道这些消息来自 Alice,因为它们有同样的笔名。假设邮件中继器的工作一切正常,Trudy 不可能模仿 Alice。在第一类型邮件中继器全部停机以后,Alice 切换到一台翻译型的邮件中继器上,并在她的新闻组中开始一个新的讨论话题。请为她设计一种方法来阻止 Trudy 模拟 Alice 在新闻组中张贴新的消息。

46. 请在 Internet 上搜索一个涉及到隐私的有趣案例,并就此案例写一份 1 页左右的报告。

47. 请在 Internet 上搜索某一个涉及到版权和合理使用原则的法庭案例,并写一份 1 页左右的报告来总结你的研究结果。

48. 编写一个程序,它用一个密钥流对输入数据做 XOR 操作,以此来加密输入数据。寻找或者编写一个尽可能好的随机数发生器来生成密钥流。这个程序应该可以被用作一个过滤器,它接受来自标准输入的明文,并产生密文显示在标准输出上(或者反之)。该程序应该有一个参数:作为随机数发生器种子的密钥。

49. 编写一个过程,它为一段数据计算 SHA-1 散列值。该过程应该有两个参数:一个指向输入缓冲区的指针和一个指向 20 字节输出缓冲区的指针。要想查看 SHA-1 的严格规范,请在 Internet 上搜索 FIPS 180-1,这是 SHA-1 的完整规范。

---

## 第9章 阅读书目和参考文献

---

我们已经结束了关于计算机网络的学习,但是,这仅仅是一个开端而已。许多有趣的话题并没有得到它们应有的篇幅来详细地介绍,甚至其他有些话题由于篇幅的原因而完全被忽略了。在本章中,我们提供了关于进一步阅读的一些建议,以及一份参考文献列表,以方便那些想继续学习计算机网络的读者。

### 9.1 进一步阅读的建议

关于计算机网络的各个方面有大量的文献资料。有三份期刊经常发表这个领域中的论文,它们是: *IEEE Transactions on Communications*、*IEEE Journal on Selected Areas in Communications* 和 *Computer Communication Review*。许多其他杂志偶尔也发表关于这个主题的论文。

IEEE 还出版了三本杂志: *IEEE Internet Computing*、*IEEE Network Magazine* 和 *IEEE Communications Magazine*, 内容包括网络领域中的综述、指导教程和案例研究。前两本杂志的重点在于体系结构、标准和软件,最后一本倾向于通信技术(光纤、卫星等)。

除此以外,还有许多一年一次或者一年两次的会议吸引了大量有关网络和分布式系统的论文,比较突出的有 *SIGCOMM Annual Conference*、*The International Conference on Distributed Computer Systems* 和 *The Symposium on Operating Systems Principles*。

下面列出一些辅助阅读的建议,并且按照本书的章节对它们进行排列和组织。其中大多数是相关主题的指导教程或者综述。有些是教科书的章节。

#### 9.1.1 简介和综合论著

Bi et al., “Wireless Mobile Communications at the Start of the 21st Century”

一个新的世纪,一项新的技术。听起来非常不错。这篇论文在介绍了无线网络的历史以后,讲述了一些重要的话题,包括标准、应用、Internet 和技术。

Comer, *The Internet Book*

任何一个希望快速进入 Internet 大门的人都应该阅读这本书。Comer 以初学者能理解的方式介绍了 Internet 的历史、成长、技术、协议和服务,由于该书覆盖了如此多的素材,所以,即使对于技术型的读者来说,该书阅读起来也是饶有趣味的。

Garber, “Will 3G Really Be the Next Big Wireless Technology?”

第3代移动电话的目标是将语音和数据结合起来,并且提供可达 2Mbps 的数据率。



但是它的启动过程太慢了。在这篇很易于阅读的文章中,作者介绍了与宽带无线通信有关的承诺、缺陷、技术、政策和经济。

*IEEE Internet Computing, Jan.-Feb. 2000*

*IEEE Internet Computing* 杂志在新千年的第一期中做了人们非常期待的一件事情:请上个世纪在 Internet 发展过程中做出过贡献的那些人展望一下 Internet 在下世纪的发展方向。这些专家是 Paul Baran、Lawrence Roberts、Leonard Kleinrock、Stephen Crocker、Danny Cohen、Bob Metcalfe、Bill Gates、Bill Joy 等。要想看到真正的结果,请等上 500 年,那时再来阅读他们的预测。

*Kipnis, "Beating the System: Abuses of the Standards Adoption Process"*

标准委员会总是试图在他们的工作中保持公平以及厂商独立,但不幸的是,有些公司总是企图违反游戏规则。例如,像这样的情形屡见不鲜:一家公司协助开发了一个标准,等到该标准通过以后,它就宣布这个标准建立在它所拥有的一个专利的基础之上,因此,它授权给自己喜欢的公司,但是不授权给不喜欢的公司,而且价格也由它随意决定。要想了解标准化过程中的阴暗面,这篇文章是一个很好的起点。

*Kyas and Crawford, ATM Networks*

ATM 曾经一度被吹捧为未来的网络互连协议,但它在电话系统中仍然很重要。本书是一本反映 ATM 当前状态的最新指南,它详细地介绍了 ATM 协议,以及如何与 IP 网络集成到一起。

*Kwok, "A Vision for Residential Broadband Service"*

如果你想知道 1995 年时 Microsoft 是如何看待视频点播技术的,那么请读一读这篇文章。5 年以后,他们的观点彻底过时了。这篇文章的价值在于,它说明了即使学识极为丰富并且又有极强推动力的人也难以精确地看清楚 5 年以后的未来状况。这对于我们所有的人来说都是一个教训。

*Naughton, A Brief History of the Future*

到底谁发明了 Internet? 许多人声称获此殊荣。可以这样说,许多人以不同的方式为 Internet 做出了贡献。Internet 的历史告诉我们这一切是如何发生的,这个过程非常有趣,甚至充满了幽默和趣闻轶事,比如 AT&T 反复地声明它的信仰“数字通信是没有前途的”。

*Perkins, "Mobile Networking in the Internet"*

这篇论文从每个协议层的角度概述了移动网络。它讲述了从物理层至传输层,以及中间件、安全性和 ad hoc 网络。

*Teger and Waks, "End-User Perspectives on Home Networking"*

家庭网络不同于企业网络。所运行的应用不相同(家庭网络上有更多的多媒体),所用的设备来自于更大范围的硬件供应商,而且用户缺乏技术训练,面对失败时没有足够的耐心。要想了解更多的信息,请看这篇论文。



*Varshney and Vetter, "Emerging Mobile and Wireless Networks"*

这是另一篇介绍无线通信的论文。它讲述了无线 LAN、无线本地回路、卫星通信,以及一些软件和应用。

*Wetteroth, OSI Reference Model for Telecommunications*

虽然 OSI 协议本身已不再被使用了,但 7 层模型已经非常知名了。这本书除了介绍 OSI 模型以外,也将该模型应用到(与计算机网络对立的)电信网络上,并且说明了常见的电话协议和其他语音协议应该适合于在网络栈中的哪个位置上。

### 9.1.2 物理层

*Abramson, "Internet Access Using VSATs"*

小的地面站变得越来越流行了,无论对于发达国家中的农村电话还是企业 Internet 访问都是如此。然而,在这两种情形下,流量的本质有极大的差异,所以,需要不同的协议来处理这两种情形。在这篇文章中,ALOHA 系统的发明者讨论了许多可被用于 VSAT 系统的信道分配方法。

*Alkhatib et al., "Wireless Data Networks: Reaching the Extra Mile"*

若想快速地浏览一下无线网络的术语和技术,包括扩频技术,则这篇指导性的论文是一个不错的起点。

*Azzam and Ransom, Broadband Access Technologies*

这本书介绍了各种网络接入技术,包括电话系统、光纤、ADSL、有线电视网络、卫星,甚至电线。其他的话题还包括家庭网络、服务、网络性能和标准。该书最后一章介绍了主要的电信公司和网络业务公司的传记,以及工业界的发展历程,也许这一章内容比其他的章节更加使这本书具有珍藏价值。

*Bellamy, Digital Telephony*

这本权威的论著中包含了所有你想知道的关于电话系统的知识。有些章节尤为有趣,譬如关于传输和复用、数字交换、光纤、移动电话和 DSL。

*Berezdivin et al., "Next-Generation Wireless Communications Concepts and Technologies"*

这些人的思想比其他人超前了一大步。题目中的“下一代”是指第 4 代无线网络。这些网络将提供无处不在的 IP 服务,并且与 Internet 无缝地连接起来,而且具有很高的带宽和极佳的服务质量。通过智能频谱分配方案、动态资源管理和自适应的服务,这些目标都是可以实现的。现在所有这些听起来像是幻想,但是同样地,在 1995 年,那时候移动电话听起来也像是一个遥远的梦想。

*Dutta-Roy, "An Overview of Cable Modem Technology and Market Perspectives"*

有线电视(Cable TV)经历了从简单的 CATV,到融合了 TV、Internet 和电话的复杂分发系统的发展历程。这些变化也极大地影响了电缆基础设施。这篇文章讨论了电缆工

广、标准、市场,并且重点在于 DOCSIS,对于有兴趣的读者,该文章值得一读。

*Farserotu and Prasad, "A Survey of Future Broadband Multimedia Satellite Systems, Issues, and Trends"*

天空中或者画板上有许多数据通信卫星,包括 Astrolink、Cyberstar、Spaceway、Skybridge、Teledesic 和 iSky。它们使用各种各样的技术,比如弯曲管道和卫星切换等。这篇论文概述了不同的卫星系统和技术,对于想了解卫星通信的读者来说,这是一个很好的起点。

*Hu and Li, "Satellite-Based Internet: A Tutorial"*

通过卫星来访问 Internet 不同于通过电缆线来访问 Internet。两者之间不仅有延迟方面的差异,而且也有路由和交换方面的区别。在这篇论文中,两位作者讨论了一些与通过卫星来访问 Internet 有关的问题。

*Joel, "Telecommunications and the IEEE Communications Society"*

这篇文章简洁而又全面地介绍了电信的历史,从电报开始,一直讲述到 802.11 为止,对于有兴趣的读者,该文章值得一读,它覆盖了无线电广播、电话、模拟和数字交换、海底电缆、数字传输、ATM、电视广播、卫星、有线电视、光纤通信、移动电话、分组交换、ARPANET 和 Internet。

*Metcalf, "Computer/Network Interface Design: Lessons from Arpanet & Ethernet"*

尽管工程师们很多年以前就已经建立了网络接口,但人们常常想知道他们是否从设计网络接口的经历中学到了什么东西。在这篇论文中,以太网的设计者告诉我们如何设计一个网络接口,以及一旦你设计出来之后又该怎么做。他没有自吹自擂,既给出了他失败的地方,也说明了他的成功之处。

*Palais, Fiber Optic Communication, 第三版*

有关光纤技术的著作通常都是写给专家们看的,但这本书却甚是通俗易懂。它介绍了波导、光源、光检测器、耦合器、调制、噪声和许多其他话题。

*Pandya, "Emerging Mobile and Personal Communications Systems"*

这是一篇关于手持个人通信系统的短文,共 9 页,其中一页列出了在其他 8 页中出现的 70 个缩写词。对手持个人通信系统有兴趣的读者,本论文值得一读。

*Sarikaya, "Packet Mode in Wireless Networks: Overview of Transition to Third Generation"*

第三代蜂窝网络的总体思想是无线数据传输。这篇论文全面地概述了第二代网络是如何处理数据的,以及第三代网络将会有哪些新的发展,对于有兴趣的读者,这是一个很好的起点。它涵盖了 GRPS、IS-95B、WCDMA 和 CDMA2000。

### 9.1.3 数据链路层

*Carlson, PPP Design, Implementation and Debugging, 第 2 版*

如果你对 PPP 套件中所有协议(包括 CCP[压缩]和 ECP[加密])的细节信息感兴趣的话,则这是一本非常好的参考书。该书尤其重点介绍了一个流行的 PPP 实现:ANU PPP-2.3。

*Gravano, Introduction to Error Control Codes*

几乎所有的数字通信都可能会产生错误,人们已经设计了许多种编码方法来检测和纠正错误。这本书介绍了一些最为重要的编码方法,从简单的线性海明码到比较复杂的 Galois 域和卷积编码。它试图以尽可能少的代数知识来讲解这些编码,但即便如此,该书仍然涉及到非常多的代数知识。

*Holzmann, Design and Validation of Computer Protocols*

对数据链路(和类似的)协议的形式化论述有兴趣的读者应该看一看这本书。它介绍了此类协议的规范、建模、正确性和测试等内容。

*Peterson and Davie, Computer Networks: A Systems Approach*

第 2 章包含了有关数据链路层上各种事项的材料,包括成帧、错误检测、停-等协议、滑动窗口协议和 IEEE 802 LAN。

*Stallings, Data and Computer Communications*

这本书第 7 章涉及到数据链路层,它讨论了流控制、错误检测和基本的数据链路协议,包括停-等协议和回退 n 帧协议。这一章还介绍了 HDLC 类型的协议。

#### 9.1.4 介质访问控制子层

*Bhagwat, "Bluetooth: Technology for Short-Range Wireless Apps"*

这篇论文简单明了地介绍了蓝牙系统,对于想要快速了解蓝牙系统的读者来说,这是一个很好的起点。该文讨论了蓝牙系统的核心协议和应用轮廓、无线电频谱、微微网和链路,后面部分还介绍了各种协议。

*Bisdikian, "An Overview of the Bluetooth Wireless Technology"*

如同上述 Bhagwat 的论文类似,这篇论文也是一个学习蓝牙系统的好起点。它讨论了微微网、协议栈和应用轮廓,以及其他一些话题。

*Crow et al., "IEEE 802.11 Wireless Local Area Networks"*

要想快速简要地了解 802.11 的技术和协议,这是一个好的起点。这篇论文的重点在于 MAC 子层。它既介绍了中心化的控制模型,也介绍了分布式的控制模型。论文最后给出了一些模拟研究的结果,显示了在各种条件下 802.11 的性能情况。

*Eklund et al., "IEEE Standard 802.16: A Technical Overview of the Wireless MAN Air Interface for Broadband Wireless Access"*

802.16 是无线本地回路的标准化方案,它是由 IEEE 于 2002 年形成的。802.16 有可能彻底改变电话服务,因为它将宽带引入到人们的家里。在这份综述中,作者们介绍了与该标准有关的主要技术事项。

---

Kapp, "802.11: Leaving the Wire Behind"

这份关于 802.11 的简短介绍包括 802.11 的基础、协议和相关的标准。

Kleinrock, "On Some Principles of Nomadic Computing and Multi-Access Communications"

通过共享信道进行无线访问,这比让有线的用户站共享一条信道要复杂得多。其中一些特殊的问题包括动态拓扑结构、路由和电源管理。这篇论文讨论了这些问题,以及其他一些与无线移动设备访问信道有关的问题。

Miller and Cummins, *LAN Technologies Explained*

是否有必要知道更多的 LAN 技术(即可以在 LAN 中使用的技术)呢?这本书涵盖了绝大多数 LAN 技术,包括 FDDI 和令牌环,以及一直非常流行的以太网。虽然现在很少再有人会重新安装前面两种网络,但是,许多原有的网络仍然在使用这些技术,而且环网络仍然是常见的,比如 SONET。

Perlman, *Interconnections*, 第 2 版

这是一本权威性的,但又饶有趣味的书,它一般性地介绍了网桥、路由器和路由机制,如果读者想了解这些内容,则 Perlman 的书绝对值得一读。IEEE 802 生成树网桥中使用的算法是该书作者设计的,她是世界上有关网络互连方面的主要权威之一。

Webb, "Broadband Fixed Wireless Access"

这篇论文介绍了固定宽带无线访问的“为什么”和“怎么做”的问题。作者在“为什么”部分中的论点是,人们并不希望这样的局面:一个家庭 e-mail 地址、一个工作 e-mail 地址,三个电话号码(家庭电话号码、工作电话号码和移动电话号码),一个即时消息账户,可能有一个或者两个传真号码;他们希望有一个在任何地方都能够工作的集成系统。技术部分的重点在物理层上,包括诸如 TDD 与 FDD、自适应调制与固定调制、载波数量等话题。

### 9.1.5 网络层

Bhatti and Crowcroft, "QoS Sensitive Flows: Issues in IP Packet Handling"

在一个网络中,一种获得更好服务质量的方法是,小心地调度好每个路由器上分组的转发任务。这篇论文相当详细地讨论了各种分组调度算法,以及相关的事项。

Chakrabarti, "QoS Issues in Ad Hoc Wireless Networks"

对于那些偶然碰到一起的笔记本电脑计算机所构成的 ad hoc 网络,即使在不考虑服务质量的情况下其路由问题也是相当困难的。然而,人们确实非常在意服务质量,所以,这个话题需要引起研究人员的关注。这篇文章讨论了 Ad hoc 网络的本质,以及与路由和服务质量相关的一些事项。

Comer, *Internetworking with TCP/IP*, 第 1 卷,第 4 版

Comer 的著作是关于 TCP/IP 协议套件的权威论著。第 4 至 11 章讨论了 IP 以及网

络层上有关的协议。其他的章节主要介绍了网络层以上的层,这些内容同样也值得一读。

*Huitema, Routing in the Internet*

如果你想了解有关 Internet 路由的方方面面,那么,这本书正适合于你。无论是可以直接发音的算法(比如 RIP、CIDR 和 MBone),还是不能直接发音的算法(比如 OSPF、IGRP、EGP 和 BGP),在这本书中都有详细的讨论。同时,该书中也介绍了一些新的特性,比如多播、移动 IP、资源预留等。

*Malhotra, IP Routing*

作为一份介绍 IP 路由的详细指南,这本书包含了大量的材料。涉及到的协议有 RIP、RIP-2、IGRP、EIGRP、OSPF 和 BGP-4。

*Metz, "Differentiated Services"*

对于许多多媒体应用来说,服务质量保证是非常重要的。综合服务和区分服务是实现服务质量的两种可能方法。这篇论文讨论了这两种方法,但重点在区分服务上。

*Metz, "IP Routers: New Tool for Gigabit Networking"*

第 5 章绝大多数其他的参考资料都是关于路由算法的。但这一篇文章有所不同:它讲述了路由器实际上是如何工作的。路由器已经经历了一个很长的变化发展过程,从早期通用的工作站发展成为高度专用的路由机器。如果你想了解更多的信息,那么这篇文章无疑是一个很好的起点。

*Nemeth et al., UNIX System Administration Handbook*

现在换一下口味,这本书第 13 章讨论了以一种更加实用的方法来进行网络互连,这是它不同于这里给出的其他参考材料的地方。这一章不只是介绍一些抽象的概念,同时也给出了许多关于如何管理一个实际网络的忠告和建议。

*Perkins, "Mobile Networking through Mobile IP"*

随着移动计算设备变得越来越普及,移动 IP 也正在变得越来越重要。这是一份很好的介绍移动 IP 以及相关话题的指南。

*Pearlman, Interconnections: Bridges and Routers, 第 2 版*

在第 12 至 15 章中,Pearlman 描述了许多涉及到单播和多播路由算法设计的事项,既有针对 WAN 的,也有针对 LAN 的网络,同时她还描述了这些算法在各种协议中的实现。但是,这本书的最精彩部分是第 18 章,它既包含大量的信息又非常有趣,作者将她数年来关于网络协议的经验精炼出来写成了这一章。

*Puzmanova, Routing and Switching: Time of Convergence?*

在上个世纪九十年代后期,有些网络设备厂商开始将所有的设备都称作交换机,同时,许多大型网络的管理员声称,他们正在从路由器向交换机转换。正如这本书的标题所暗示的,它预测了路由器和交换机的未来,并且质疑它们是否真的会走到一起。

*Royer and Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless*

*Networks*”

我们在第 5 章中讨论的 ad hoc 路由算法 AODV 并不是惟一的 ad hoc 路由算法。其他还有许多这样的算法,包括 DSDV、CGSR、WRP、DSR、TORA、ABR、DRP 和 SRP,这篇论文讨论了这些算法,并且对它们作了比较。很显然,如果你打算设计一个新的 ad hoc 路由协议,则第 1 步是先想好一个 3 字母或者 4 字母的缩写词。

*Stevens, TCP/IP Illustrated*, 第 1 卷

第 3-10 章全面地介绍了 IP 和相关的协议(ARP、RARP 和 ICMP),而且作者通过一些例子来演示这些内容。

*Striegel and Manimaran, “A Survey of QoS Multicasting Issues”*

随着诸如 Internet 电台和 Internet 电视之类的服务开始兴起,多播和服务质量成为两个日益重要的话题。在这篇综述论文中,两位作者讨论了路由算法如何将多播和服务质量也考虑进去。

*Yang and Reddy, “A Taxonomy for Congestion Control Algorithms in Packet Switching Networks”*

这篇论文的作者设计了一种针对拥塞控制算法的分类方法。主要的类别有:源端控制的开环算法、目标端控制的开环算法、有显式反馈的闭环算法,以及无显式反馈的闭环算法。他们使用这种分类方法来描述 23 种已有的算法,并对它们进行了分类。

### 9.1.6 传输层

*Comer, Internetworking with TCP/IP*, 第 1 卷,第 4 版

正如前面所提到的,Comer 的著作是关于 TCP/IP 协议套件的权威论著。第 12 章讨论了 UDP;第 13 章讨论了 TCP。

*Hall and Cerf, Internet Core Protocols: The Definitive Guide*

如果你喜欢来自于直接源头的信息,那么,这是学习 TCP 的最佳之处。毕竟,Cerf 是发明者之一。第 7 章是一份很好的 TCP 参考材料,它显示了如何对协议分析工具和网络管理工具所提供的信息进行解释。其他的章节讨论了 UDP、IGMP、ICMP 和 ARP。

*Kurose and Ross, Computer Networking: A Top-Down Approach Featuring the Internet*

第 3 章讨论了传输层,其中包含了相当数量的关于 UDP 和 TCP 的材料。它也讨论了我们在本书第 3 章中看到的停-等协议和回退 n 步协议。

*Mogul, “IP Network Performance”*

尽管这篇文章的标题是说 IP 网络性能,但是,它至少会介绍 TCP 和一般意义上的网络性能,而不会专门介绍 IP 性能。该文章给出了大量有用的指导原则和各种实践经验。

*Peterson and Davie, Computer Networks: A Systems Approach*

第 5 章介绍了 UDP、TCP 和一些相关的协议。同时也简要地讨论了网络性能的



问题。

*Stevens, TCP/IP Illustrated*, 第 1 卷

第 17-24 章全面地介绍了 TCP, 并通过例子做了演示。

### 9.1.7 应用层

*Bergholz, "Extending Your Markup: An XML Tutorial"*

这是一份针对初学者的关于 XML 的简短、浅显的介绍。

*Cardellini et al., The State-of-the-Art in Locally Distributed Web-Server Systems"*

随着 Web 的日益流行, 有些 Web 站点需要大型的服务器场来处理其流量。构建一个服务器场的困难之处在于如何在机器之间分发负载。这篇介绍性的论文用相当的篇幅讨论了这个主题。

*Berners Lee et al., "The World Wide Web"*

Web 的发明者以及他在 CERN 的一些同事展示了 Web 的前景以及它的走向。这篇文章将焦点集中在 Web 的结构、URL、HTTP 和 HTML 以及未来的发展方向上, 同时也比较了 Web 与其他的分布式信息系统。

*Choudbury et al., "Copyright Protection for Electronic Publishing on Computer Networks"*

尽管有大量的书籍和文章描述了密码学算法, 但是鲜有提及如何利用这些算法来阻止用户进一步分发他们自己可以解密的文档。这篇论文描述了许多可在电子领域中保护文档作者版权的机制。

*Collins, "Carrier Grade Voice over IP"*

如果你已经阅读过 Varshney 等人的论文, 并且现在想知道所有关于用 H. 323 来实现 IP 语音的详细信息, 那么, 这本书很值得一读。尽管这本书又长又涉及大量的细节, 但本质上它是指南性的, 所以并不要求读者具备电话工程方面的预备知识。

*Davison, "A Web Caching Primer"*

随着 Web 的增长, 缓存对于性能越来越重要了。这篇论文简要地介绍了 Web 缓存, 对于有兴趣的读者, 本论文值得一读。

*Krishnamurthy and Rexford, Web Protocols and Practice*

要想找出一本比这本书更详细地介绍 Web 方方面面的书是非常困难的。正如你可能期望的, 这本书覆盖了 Web 客户、服务器、代理和缓存。除此以外, 它还有一些章节涉及到 Web 流量和测量手段, 以及关于 Web 的当前研究状况和如何改进 Web。

*Rabinovich and Spatscheck, Web Caching and Replication*

这是一本全面地介绍 Web 缓存和复制的书, 如果读者对此有兴趣, 则不妨一读。该书相当细致地介绍了代理、缓存、预先获取(prefetching)、内容传输网络、服务器选择等更

多内容。

*Shahabi et al. "Yima: A Second-Generation Continuous Media Server"*

多媒体服务器是极为复杂的系统,它们必须管理 CPU 调度、磁盘文件的放置、流同步等。随着时间的推移,人们已经逐渐知道了该如何将这些工作做得更好。这篇论文展示了一个最新系统的总体结构。

*Tittel et al., Mastering XHTML*

这是两本书合成的一个大卷,它覆盖了 Web 新的标准标记语言。该书首先是讲述 XHTML 的正文,其焦点主要集中在它与普通的 HTML 如何有所不同。然后是一份全面的参考指南,其中包括标签、编码和 XHTML 1.0 中用到的特殊字符。

*Varshney et al., "Voice over IP"*

IP 语音是如何工作的?它将来是否会取代公共交换电话网络?读一读这本书,你就会知道答案。

### 9.1.8 网络安全

*Anderson, "Why Cryptosystems Fail"*

根据 Anderson 的观点,银行系统的安全性非常之糟,但其中的原因并不是由于聪明的人侵者会用他们的 PC 来破解 DES。真正的问题包括不诚实的雇员(银行的雇员可以将顾客的邮政地址改成他自己的地址,从而截取银行信用卡和 PIN 码)和程序错误(为所有的顾客赋以相同的 PIN 码)。尤其有趣的是当银行面对错误时其傲慢的反应:"我们的系统是非常完善的,因此所有的错误一定是由于顾客的错误或者欺骗而引起的。"

*Anderson, Security Engineering*

在某种程度上,这本书是 *Why Cryptosystems Fail* (即上一本书)的 600 页版本。这本书的技术性比 *Secrets and Lies* 更强,但是比 *Network Security* (见下面)更弱。该书在简单地介绍了基本的安全技术以后,其他所有的章节都在介绍各种各样的应用,包括银行系统、核命令和控制系统、安全打印、生物测量学、物理安全性、电子战争、电信安全、电子商务和版权保护。该书的第三部分涉及到政策、管理和系统评估。

*Artz, "Digital Steganography"*

信息隐藏学又回到了古希腊时代,那时候人们在空白的板上融上一层蜡,因而秘密消息可以被涂抹在底层的木板上,然后再重新抹上蜡。现在人们使用的方法有所不同,但是目标是相同的。在这篇论文中作者讨论了各种现代的信息隐藏技术,通过这些技术可以将信息隐藏在图像、音频或者其他的载体中。

*Brands, Rethinking Public Key Infrastructures and Digital Certificates*

这本书不仅仅全面地介绍了数字证书,同时也极力推崇这项技术。作者相信,当前基于纸质的身份鉴别系统已经过时了,而且也极为低效;同时作者认为,数字证书可以被用于诸如电子投票表决、数字版权保护,甚至取代现金之类的应用中。他还警告说,如果没

有 PKI 和加密,则 Internet 可能会变成一个大范围的监视工具。

*Kaufman et al., Network Security*, 第 2 版

这本权威而诙谐的书给出了大量的有关网络安全算法和协议的技术信息,如果读者正在寻找这些信息,则这本书当属首选。该书以相当的篇幅审慎地介绍了对称密钥算法和协议、公开密钥算法和协议、消息散列算法、认证、Kerberos、PKI、IPSec、SSL/TLS 和电子邮件安全性,同时也给出了许多例子。第 26 章关于安全的故事是一处真正的精华所在。在安全性方面,邪恶总是存在于细微之处。任何人若要设计一个实际的安全系统,他总可以在这一章中从来自于实践的忠告中学到许多东西。

*Pohlmann, Firewall Systems*

防火墙是大多数网络的第一道(和最后一道)对抗攻击者的防线。这本书讲述了防火墙是如何工作的,以及它们做了些什么事情,它从最简单的、基于软件的、被设计用于保护单台 PC 的防火墙开始讲起,一直到位于一个私有网络和其 Internet 连接之间的高级防火墙设备。

*Schneier, Applied Cryptography*, 第 2 章

这本不朽的密码学著作是 NSA 的梦魇:它描述了几乎所有已知的密码学算法。更糟的是(或者说更为理想的是,这要看你持什么样的观点),这本书包含了大多数算法的可运行程序(以 C 语言的形式)。而且,该书也提供了超过 1600 条密码学参考文献。它不是针对初学者的,但如果你真的想让自己的文件保密的话,则建议读一读这本书。

*Schneier, Secrets and Lies*

如果你从头至尾阅读了“Applied Cryptography”一书,那么,你已经知道了有关密码学算法的所有内容。如果你再从头到尾地阅读了这本“Secrets and Lies”一书(阅读这本书所需要的时间比上一本少得多),那么,你就会知道密码学算法并不是所有的一切。大多数安全弱点并不是由于算法有错误,或者密钥太短,而是由于安全环境中存在缺陷。该书中展示了大量的例子,其内容涉及到威胁、攻击、防御、反击等。作为一本从广阔意义上讨论计算机安全的非技术性书籍,这本引人入胜的书值得一读。

*Skoudis, Counter Hack*

阻止黑客的最好办法是从黑客的角度来思考问题。这本书显示了黑客们如何来看待一个网络,作者认为,安全性应该是整个网络设计的一个函数,而不是基于某一项特定技术的事后结果。该书几乎覆盖了所有常见的攻击,包括那些利用“用户通常并不熟悉计算机安全手段”这一事实的“社会工程”类型的攻击。

## 9.2 按字母顺序的参考文献

ABRAMSON, N.: “Internet Access Using VSATs,” *IEEE Commun. Magazine*, vol. 38, pp. 60-68, July 2000.

ABRAMSON, N.: “Development of the ALOHANET,” *IEEE Trans. on Information Theory*, vol. 16, pp. 726-729, 1969.

IT-31, pp. 119-123, March 1985.

Computer Communications

ADAMS, M. , and DULCHINOS, D. : "OpenCable," IEEE Commun. Magazine, vol. 39, pp. 98-105, June 2001.

ALKHATIB, H. S. , BAILEY, C. , GERLA, M. , and MCRAE, J. : "Wireless Data Networks: Reaching the Extra Mile," Computer, vol. 30, pp. 59-62, Dec. 1997.

ANDERSON, R. J. : "Free Speech Online and Office," Computer, vol. 25, pp. 28-30, June 2002.

ANDERSON, R. J. : Security Engineering, New York: Wiley, 2001.

ANDERSON, R. J. : "The Eternity Service," Proc. First Int'l Conf. on Theory and Appl. of Cryptology, CTU Publishing House, 1996.

ANDERSON, R. J. : "Why Cryptosystems Fail," Commun. of the ACM, vol. 37, pp. 32-40, Nov. 1994.

ARTZ, D. : "Digital Steganography," IEEE Internet Computing, vol. 5, pp. 75-80, 2001.

AZZAM, A. A. , and RANSOM, N. : Broadband Access Technologies, New York: McGraw-Hill, 1999.

BAKNE, A. , and BADRINATH, B. R. : "I-TCP: Indirect TCP for Mobile Hosts," Proc. 15th Int'l Conf. on Distr. Computer Systems, IEEE, pp. 136-143, 1995.

BALAKRISHNAN, H. , SESHAN, S. , and KATZ, R. H. : "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," Proc. ACM Mobile Computing and Networking Conf. , ACM, pp. 2-11, 1995.

BALLARDIE, T. , FRANCIS, P. , and CROWCROFT, J. : "Core Based Trees (CBT)," Proc. SIGCOMM '93 Conf. , ACM, pp. 85-95, 1993.

BARAKAT, C. , ALTMAN, E. , and DABBOUS, W. : "On TCP Performance in a Heterogeneous Network: A Survey," IEEE Commun. Magazine, vol. 38, pp. 40-46, Jan. 2000.

BELLAMY, J. : Digital Telephony, 3rd ed. , New York: Wiley, 2000.

BELLMAN, R. E. : Dynamic Programming, Princeton, NJ: Princeton University Press, 1957.

BELSNES, D. : "Flow Control in the Packet Switching Networks," Communications Networks, Uxbridge, England: Online, pp. 349-361, 1975.

BENNET, C. H. , and BRASSARD, G. : "Quantum Cryptography: Public Key Distribution and Coin Tossing," Int'l Conf. on Computer Systems and Signal Processing, pp. 175-179, 1984.

BEREZDIVIN, R. , BREINIG, R. , and TOPP, R. : "Next-Generation Wireless Communication Concepts and Technologies," IEEE Commun. Magazine, vol. 40, pp. 108-116, March 2002.

BERGHEL, H. L. : "Cyber Privacy in the New Millennium," Computer, vol. 34, pp. 132-134, Jan. 2001.

BERGHOLZ, A. : "Extending Your Markup: An XML Tutorial," IEEE Internet Computing, vol. 4, pp. 74-79, July-Aug. 2000.

BERNERS-LEE, T. , CAILLIAU, A. , LOUTONEN, A. , NIELSEN, H. F. , and SECRET, A. : "The World Wide Web," Commun. of the ACM, vol. 37, pp. 76-82, Aug. 1994.

BERTSEKAS, D. , and GALLAGER, R. : Data Networks, 2nd ed. , Englewood Cliffs, NJ: Prentice Hall, 1992.

BHAGWAT, P. : "Bluetooth: Technology for Short-Range Wireless Apps," IEEE Internet Computing, vol. 5, pp. 96-103, May-June 2001.

BHARGHAVAN, V. , DEMERS, A. , SHENKER, S. , and ZHANG, L. : "MACAW: A Media Access Protocol for Wireless LANs," Proc. SIGCOMM '94 Conf. , ACM, pp. 212-225, 1994.

BHATTI, S. N. , and CROWCROFT, J. : "QoS Sensitive Flows: Issues in IP Packet Handling,"

---

IEEE Internet Computing, vol. 4, pp. 48-57, July-Aug. 2000.

BI, Q., ZYSMAN, G. I., and MENKES, H., "Wireless Mobile Communications at the Start of the 21st Century," IEEE Commun. Magazine, vol. 39, pp. 110-116, Jan. 2001.

BIHAM, E., and SHAMIR, A., "Differential Cryptanalysis of the Data Encryption Standard," Proc. 17th Ann. Int'l Cryptology Conf., Berlin: Springer-Verlag LNCS 1294, pp. 513-525, 1997.

BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P. A., KUTTEN, S., MOLVA, R., and YUNG, M., "Systematic Design of a Family of Attack-Resistant Authentication Protocols," IEEE J. on Selected Areas in Commun., vol. 11, pp. 679-693, June 1993.

BIRRELL, A. D., and NELSON, B. J., "Implementing Remote Procedure Calls," ACM Trans. on Computer Systems, vol. 2, pp. 39-59, Feb. 1984.

BIRYUKOV, A., SHAMIR, A., and WAGNER, D., "Real Time Cryptanalysis of A5/1 on a PC," Proc. Seventh Int'l Workshop on Fast Software Encryption, Berlin: Springer-Verlag LNCS 1978, p. 1, 2000.

BISDIKIAN, C., "An Overview of the Bluetooth Wireless Technology," IEEE Commun. Magazine, vol. 39, pp. 86-94, Dec. 2001.

BLAZE, M., "Protocol Failure in the Escrowed Encryption Standard," Proc. Second ACM Conf. on Computer and Commun. Security, ACM, pp. 59-67, 1994.

BLAZE, M., and BELLOVIN, S., "Tapping on My Network Door," Commun. of the ACM, vol. 43, p. 136, Oct. 2000.

BOGINENI, K., SIVALINGAM, K. M., and DOWD, P. W., "Low-Complexity Multiple Access Protocols for Wavelength-Division Multiplexed Photonic Networks," IEEE Journal on Selected Areas in Commun., vol. 11, pp. 590-604, May 1993.

BOLCSKEI, H., PAULRAJ, A. J., HARI, K. V. S., and NABAR, R. U., "Fixed Broadband Wireless Access: State of the Art, Challenges, and Future Directions," IEEE Commun. Magazine, vol. 39, pp. 100-108, Jan. 2001.

BORISOV, N., GOLDBERG, I., and WAGNER, D., "Intercepting Mobile Communications: The Insecurity of 802.11," Seventh Int'l Conf. on Mobile Computing and Networking, ACM, pp. 180-188, 2001.

BRANDS, S., Rethinking Public Key Infrastructures and Digital Certificates, Cambridge, MA: M. I. T. Press, 2000.

BRAY, J., and STURMAN, C. F., Bluetooth 1.1: Connect without Cables, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2002.

BREYER, R., and RILEY, S., Switched, Fast, and Gigabit Ethernet, Indianapolis, IN: New Riders, 1999.

BROWN, S., Implementing Virtual Private Networks, New York: McGraw Hill, 1999.

BROWN, L., KWAN, M., PIEPRZYK, J., and SEBERRY, J., "Improving Resistance to Differential Cryptanalysis and the Redesign of LOKI," ASIACRYPT '91 Abstracts, pp. 25-30, 1991.

BURNETT, S., and PAINE, S., RSA Security's Official Guide to Cryptography, Berkeley, CA: Osborne/McGraw-Hill, 2001.

CAPETANAKIS, J. I., "Tree Algorithms for Packet Broadcast Channels," IEEE Trans. on Information Theory, vol. IT-25, pp. 505-515, Sept. 1979.

CARDELLINI, V., CASALICCHIO, E., COLAJANNI, M., and YU, P. S., "The State-of-the-Art in Locally Distributed Web-Server Systems," ACM Computing Surveys, vol. 34, pp. 263-311, June 2002.

- CARLSON, J. : PPP Design, Implementation and Debugging. 2nd ed. , Boston: Addison Wesley, 2001.
- CERF, V. , and KAHN, R. : "A Protocol for Packet Network Interconnection," IEEE Trans. on Commun. , vol. COM-22, pp. 637-648, May 1974.
- CHAKRABARTI, S. : "QoS Issues in Ad Hoc Wireless Networks," IEEE Commun. Magazine, vol. 39, pp. 142-148, Feb. 2001.
- CHASE, J. S. , GALLATIN, A. J. , and YOCUM, K. G. : "End System Optimizations for High-Speed TCP," IEEE Commun. Magazine, vol. 39, pp. 68-75, April 2001.
- CHEN, B. , JAMIESON, K. , BALAKRISHNAN, H. , and MORRIS, R. : "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," ACM Wireless Networks, vol. 8, Sept. 2002.
- CHEN, K.-C. : "Medium Access Control of Wireless LANs for Mobile Computing," IEEE Network Magazine, vol. 8, pp. 50-63, Sept./Oct. 1994.
- CHOUDBURY, A. K. , MAXEMCHUK, N. F. , PAUL, S. , and SCHULZRINNE, H. G. : "Copyright Protection for Electronic Publishing on Computer Networks," IEEE Network Magazine, vol. 9, pp. 12-20, May/June, 1995.
- CHU, Y. , RAO, S. G. , and ZHANG, H. : "A Case for End System Multicast," Proc. Int'l Conf. on Measurements and Modeling of Computer Syst. , ACM, pp. 1-12, 2000.
- CLARK, D. D. : "The Design Philosophy of the DARPA Internet Protocols," Proc. SIGCOMM '88 Conf. , ACM, pp. 106-114, 1988.
- CLARK, D. D. : "Window and Acknowledgement Strategy in TCP," RFC 813, July 1982.
- CLARK, D. D. , DAVIE, B. S. , FARBER, D. J. , GOPAL, I. S. , KADABA, B. K. , SINCOSKIE, W. D. , SMITH, J. M. , and TENNENHOUSE, D. L. : "The Aurora Gigabit Testbed," Computer Networks and ISDN Systems, vol. 25, pp. 599-621, Jan. 1993.
- CLARK, D. D. , JACOBSON, V. , ROMKEY, J. , and SALWEN, H. : "An Analysis of TCP Processing Overhead," IEEE Commun. Magazine, vol. 27, pp. 23-29, June 1989.
- CLARK, D. D. , LAMBERT, M. , and ZHANG, L. : "NETBLT: A High Throughput Transport Protocol," Proc. SIGCOMM '87 Conf. , ACM, pp. 353-359, 1987.
- CLARKE, A. C. : "Extra-Terrestrial Relays," Wireless World, 1945.
- CLARKE, I. , MILLER, S. G. , HONG, T. W. , SANDBERG, O. , and WILEY, B. : "Protecting Free Expression Online with Freenet," IEEE Internet Computing, vol. 6, pp. 40-49, Jan. Feb. 2002.
- COLLINS, D. : Carrier Grade Voice over IP, New York: McGraw-Hill, 2001.
- COLLINS, D. , and SMITH, C. : 3G Wireless Networks, New York: McGraw-Hill, 2001.
- COMER, D. E. : The Internet Book, Englewood Cliffs, NJ: Prentice Hall, 1995.
- COMER, D. E. : Internetworking with TCP/IP, vol. 1, 4th ed. , Englewood Cliffs, NJ: Prentice Hall, 2000.
- COSTA, L. H. M. K. , FDIDA, S. , and DUARTE, O. C. M. B. : "Hop by Hop Multicast Routing Protocol," Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Commun. , ACM, pp. 249-259, 2001.
- CRAVER, S. A. , WU, M. , LIU, B. , STUBBLEFIELD, A. , SWARTZLANDER, B. , WALLACH, D. W. , DEAN, D. , and FELTEN, E. W. : "Reading Between the Lines: Lessons from the SDMI Challenge," Proc. 10th USENIX Security Symp. , USENIX, 2001.
- CRESPO, P. M. , HONIG, M. L. , and SALEHI, J. A. : "Spread-Time Code Division Multiple Access," IEEE Trans. on Commun. , vol. 43, pp. 2139-2148, June 1995.



- CROW, B. P. , WIDJAJA, I. KIM, J. G. , and SAKAI, P. T. ; "IEEE 802.11 Wireless Local Area Networks," IEEE Commun. Magazine, vol. 35, pp. 116-126, Sept. 1997.
- CROWCROFT, J. , WANG, Z. , SMITH, A. , and ADAMS, J. ; "A Rough Comparison of the IETF and ATM Service Models," IEEE Network Magazine, vol. 9, pp. 12-16, Nov./Dec. 1995.
- DABEK, F. , BRUNSKILL, E. , KAASHOEK, M. F. , KARGER, D. , MORRIS, R. , STOICA, R. , and BALAKRISHNAN, H. ; "Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service," Proc. 8th Workshop on Hot Topics in Operating Systems, IEEE, pp. 71-76, 2001a.
- DABEK, F. , KAASHOEK, M. F. , KARGER, D. , MORRIS, R. , and STOICA, I. ; "Wide-Area Cooperative Storage with CFS," Proc. 18th Symp. on Operating Systems Prin. , ACM, pp. 202-215, 2001b.
- DAEMEN, J. , and RIJNDEN, V. ; The Design of Rijndael, Berlin: Springer-Verlag, 2002.
- DANTHINE, A. A. S. ; "Protocol Representation with Finite State Models," IEEE Trans. on Commun. , vol. COM-28, pp. 632-643, April 1980.
- DAVIDSON, J. , and PETERS, J. ; Voice over IP Fundamentals, Indianapolis, IN: Cisco Press, 2000.
- DAVIE, B. , and REKHTER, Y. ; MPLS Technology and Applications, San Francisco: Morgan Kaufmann, 2000.
- DAVIS, P. T. , and MCGUFFIN, C. R. ; Wireless Local Area Networks, New York: McGraw-Hill, 1995.
- DAVISON, B. D. ; "A Web Caching Primer," IEEE Internet Computing, vol. 5, pp. 38-45, July-Aug. 2001.
- DAY, J. D. ; "The (Un)Revised OSI Reference Model," Computer Commun. Rev. , vol. 25, pp. 39-55, Oct. 1995.
- DAY, J. D. , and ZIMMERMAN, H. ; "The OSI Reference Model," Proc. of the IEEE, vol. 71, pp. 1334-1340, Dec. 1983.
- DE VRIENDT, J. , LAINE, P. , LEROUGE, C. , and XU, X. ; "Mobile Network Evolution: A Revolution on the Move," IEEE Commun. Magazine, vol. 40, pp. 104-111, April 2002.
- DEERING, S. E. ; "SIP: Simple Internet Protocol," IEEE Network Magazine, vol. 7, pp. 16-28, May/June 1993.
- DEMERS, A. , KESHAV, S. , and SHENKER, S. ; "Analysis and Simulation of a Fair Queueing Algorithm," Internetwork: Research and Experience, vol. 1, pp. 3-26, Sept. 1990.
- DENNING, D. E. , and SACCO, G. M. ; "Timestamps in Key Distribution Protocols," Commun. of the ACM, vol. 24, pp. 533-536, Aug. 1981.
- DIFFIE, W. , and HELLMAN, M. E. ; "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," Computer, vol. 10, pp. 74-84, June 1977.
- DIFFIE, W. , and HELLMAN, M. E. ; "New Directions in Cryptography," IEEE Trans. on Information Theory, vol. IT-22, pp. 644-654, Nov. 1976.
- DIJKSTRA, E. W. ; "A Note on Two Problems in Connexion with Graphs," Numer. Math. , vol. 1, pp. 269-271, Oct. 1959.
- DOBROWSKI, G. , and GRISE, D. ; ATM and SONET Basics, Fuquay-Varina, NC: APDG Telecom Books, 2001.
- DONALDSON, G. , and JONES, D. ; "Cable Television Broadband Network Architectures," IEEE Commun. Magazine, vol. 39, pp. 122-126, June 2001.
- DORFMAN, R. ; "Detection of Defective Members of a Large Population," Annals Math.

Statistics, vol. 14, pp. 436-440, 1943.

DOUFEXI, A., ARMOUR, S., BUTLER, M., NIX, A., BULL, D., MCGEEHAN, J., and KARLSSON, P.: "A Comparison of the HIPERLAN/2 and IEEE 802.11A Wireless LAN Standards," IEEE Commun. Magazine, vol. 40, pp. 172-180, May 2002.

DURAND, A.: "Deploying IPv6," IEEE Internet Computing, vol. 5, pp. 79-81, Jan.-Feb. 2001.

DUTCHER, B.: The NAT Handbook, New York: Wiley, 2001.

DUTTA-ROY, A.: "An Overview of Cable Modem Technology and Market Perspectives," IEEE Commun. Magazine, vol. 39, pp. 81-88, June 2001.

EASTTOM, C.: Learn JavaScript, Ashburton, U.K.: Wordware Publishing, 2001.

EL GAMAL, T.: "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," IEEE Trans. on Information Theory, vol. IT-31, pp. 469-472, July 1985.

ELHANANY, I., KAHANE, M., and SADOT, D.: "Packet Scheduling in Next-Generation Multiterabit Networks," Computer, vol. 34, pp. 104-106, April 2001.

ELMIRGHANI, J. M. H., and MOUFTAH, H. T.: "Technologies and Architectures for Scalable Dynamic Dense WDM Networks," IEEE Commun. Magazine, vol. 38, pp. 58-66, Feb. 2000.

FARSEROTU, J., and PRASAD, R.: "A Survey of Future Broadband Multimedia Satellite Systems, Issues, and Trends," IEEE Commun. Magazine, vol. 38, pp. 128-133, June 2000.

FIORINI, D., CHIANI, M., TRALLI, V., and SALATI, C.: "Can we Trust HDLC?," Computer Commun. Rev., vol. 24, pp. 61-80, Oct. 1994.

FLOYD, S., and JACOBSON, V.: "Random Early Detection for Congestion Avoidance," IEEE/ACM Trans. on Networking, vol. 1, pp. 397-413, Aug. 1993.

FLUHRER, S., MANTIN, I., and SHAMIR, A.: "Weakness in the Key Scheduling Algorithm of RC4," Proc. Eighth Ann. Workshop on Selected Areas in Cryptography, 2001.

FORD, L. R., Jr., and FULKERSON, D. R.: Flows in Networks, Princeton, NJ: Princeton University Press, 1962.

FORD, W., and BAUM, M. S.: Secure Electronic Commerce, Upper Saddle River, NJ: Prentice Hall, 2000.

FORMAN, G. H., and ZAHORJAN, J.: "The Challenges of Mobile Computing," Computer, vol. 27, pp. 38-47, April 1994.

FRANCIS, P.: "A Near-Term Architecture for Deploying Pip," IEEE Network Magazine, vol. 7, pp. 30-37, May/June 1993.

FRASER, A. G.: "Towards a Universal Data Transport System," in Advances in Local Area Networks, Kummerle, K., Tobagi, F., and Limb, J. O. (Eds.), New York: IEEE Press, 1987.

FRENGLE, N.: I-Mode: A Primer, New York: Hungry Minds, 2002.

GADECKI, C., and HECKERT, C.: ATM for Dummies, New York: Hungry Minds, 1997.

GARBER, L.: "Will 3G Really Be the Next Big Wireless Technology?," Computer, vol. 35, pp. 26-32, Jan. 2002.

GARFINKEL, S., with SPAFFORD, G.: Web Security, Privacy, and Commerce, Sebastopol, CA: O'Reilly, 2002.

GEIER, J.: Wireless LANs, 2nd ed., Indianapolis, IN: Sams, 2002.

GEVROS, P., CROWCROFT, J., KIRSTEIN, P., and BHATTI, S.: "Congestion Control Mechanisms and the Best Effort Service Model," IEEE Network Magazine, vol. 15, pp. 16-25, May-June 2001.

GHANI, N., and DIXIT, S.: "TCP/IP Enhancements for Satellite Networks," IEEE Commun.

---

Magazine, vol. 37, pp. 64-72, 1999.

GINSBURG, D. : ATM: Solutions for Enterprise Networking, Boston: Addison-Wesley, 1996.

GOODMAN, D. J. : "The Wireless Internet: Promises and Challenges," Computer, vol. 33, pp. 36-41, July 2000.

GORALSKI, W. J. : Optical Networking and WDM, New York: McGraw-Hill, 2001.

GORALSKI, W. J. : SONET, 2nd ed. , New York: McGraw-Hill, 2000.

GORALSKI, W. J. : Introduction to ATM Networking, New York: McGraw-Hill, 1995.

GOSSAIN, H. , DE MORAIS CORDEIRO, and AGRAWAL, D. P. : "Multicast: Wired to Wireless," IEEE Commun. Mag. , vol. 40, pp. 116-123, June 2002.

GRAVANO, S. : Introduction to Error Control Codes, Oxford, U. K. : Oxford University Press, 2001.

GUO, Y. , and CHASKAR, H. : "Class-Based Quality of Service over Air Interfaces in 4G Mobile Networks," IEEE Commun. Magazine, vol. 40, pp. 132-137, March 2002.

HAARTSEN, J. : "The Bluetooth Radio System," IEEE Personal Commun. Magazine, vol. 7, pp. 28-36, Feb. 2000.

HAC, A. : "Wireless and Cellular Architecture and Services," IEEE Commun. Magazine, vol. 33, pp. 98-104, Nov. 1995.

HAC, A. , and GUO, L. : "A Scalable Mobile Host Protocol for the Internet," Int'l J. of Network Mgmt, vol. 10, pp. 115-134, May-June, 2000.

HALL, E. , and CERF, V. : Internet Core Protocols: The Definitive Guide. Sebastopol, CA: O'Reilly, 2000.

HAMMING, R. W. : "Error Detecting and Error Correcting Codes," Bell System Tech. J. , vol. 29, pp. 147-160, April 1950.

HANEGAN, K. : Custom CGI Scripting with Perl, New York: Wiley, 2001.

HARRIS, A. : JavaScript Programming for the Absolute Beginner, Premier Press, 2001.

HARTE, L. , KELLOGG, S. , DREHER, R. , and SCHAFFNIT, T. : The Comprehensive Guide to Wireless Technology, Fuquay-Varina, NC: APDG Publishing, 2000.

HARTE, L. , LEVINE, R. , and KIKTA, R. : 3G Wireless Demystified, New York: McGraw-Hill, 2002.

HAWLEY, G. T. : "Historical Perspectives on the U. S. Telephone System," IEEE Commun. Magazine, vol. 29, pp. 24-28, March 1991.

HECHT, J. : "Understanding Fiber Optics," Upper Saddle River, NJ: Prentice Hall, 2001.

HEEGARD, C. , COFFEY, J. T. , GUMMADI, S. , MURPHY, P. A. , PROVENCIO, R. , ROSSIN, E. J. , SCHRUM, S. , and SHOEMAKER, M. B. : "High-Performance Wireless Ethernet," IEEE Commun. Magazine, vol. 39, pp. 64-73, Nov. 2001.

HELD, G. : The Complete Modem Reference, 2nd ed. , New York: Wiley, 1994.

HELLMAN, M. E. : "A Cryptanalytic Time-Memory Tradeoff," IEEE Trans. on Information Theory, vol. IT-26, pp. 401-406, July 1980.

HILLS, A. : "Large-Scale Wireless LAN Design," IEEE Commun. Magazine, vol. 39, pp. 98-104, Nov. 2001.

HOLZMANN, G. J. : Design and Validation of Computer Protocols, Englewood Cliffs, NJ: Prentice Hall, 1991.

HU, Y. , and LI, V. O. K. : "Satellite-Based Internet Access," IEEE Commun. Magazine, vol. 39, pp. 155-162, March 2001.

- HU, Y.-C., and JOHNSON, D. B.: "Implicit Source Routes for On-Demand Ad Hoc Network Routing," *Proc. ACM Int'l Symp. on Mobile Ad Hoc Networking & Computing*, ACM, pp. 1-10, 2001.
- HUANG, V., and ZHUANG, W.: "QoS-Oriented Access Control for 4G Mobile Multimedia CDMA Communications," *IEEE Commun. Magazine*, vol. 40, pp. 118-125, March 2002.
- HUBER, J. F., WEILER, D., and BRAND, H.: "UMTS, the Mobile Multimedia Vision for IMT-2000: A Focus on Standardization," *IEEE Commun. Magazine*, vol. 38, pp. 129-136, Sept. 2000, nr u 0
- HUI, J.: "A Broadband Packet Switch for Multi-rate Services," *Proc. Int'l Conf. on Commun.*, IEEE, pp. 782-788, 1987.
- HUITEMA, C.: *Routing in the Internet*, Englewood Cliffs, NJ: Prentice Hall, 1995.
- HULL, S.: *Content Delivery Networks*, Berkeley, CA: Osborne/McGraw-Hill, 2002.
- HUMBLET, P. A., RAMASWAMI, R., and SIVARAJAN, K. N.: "An Efficient Communication Protocol for High-Speed Packet-Switched Multichannel Networks," *Proc. SIGCOMM '92 Conf.*, ACM, pp. 2-13, 1992.
- HUNTER, D. K., and ANDONOVIC, I.: "Approaches to Optical Internet Packet Switching," *IEEE Commun. Magazine*, vol. 38, pp. 116-122, Sept. 2000.
- HUSTON, G.: "TCP in a Wireless World," *IEEE Internet Computing*, vol. 5, pp. 82-84, March-April, 2001.
- IBE, O. C.: *Essentials of ATM Networks and Services*, Boston: Addison-Wesley, 1997.
- IRMER, T.: "Shaping Future Telecommunications: The Challenge of Global Standardization," *IEEE Commun. Magazine*, vol. 32, pp. 20-28, Jan. 1994.
- IZZO, P.: *Gigabit Networks*, New York: Wiley, 2000.
- JACOBSON, V.: "Congestion Avoidance and Control," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 314-329, 1988.
- JAIN, R.: "Congestion Control and Traffic Management in ATM Networks: Recent Advances and a Survey," *Computer Networks and ISDN Systems*, vol. 27, Nov. 1995.
- JAIN, R.: *FDDI Handbook-High-Speed Networking Using Fiber and Other Media*, Boston: Addison-Wesley, 1994.
- JAIN, R.: "Congestion Control in Computer Networks: Issues and Trends," *IEEE Network Magazine*, vol. 4, pp. 24-30, May/June 1990.
- JAKOBSSON, M., and WETZEL, S.: "Security Weaknesses in Bluetooth," *Topics in Cryptology, CT-RSA 2001*, Berlin: Springer-Verlag LNCS 2020, pp. 176-191, 2001.
- JOEL, A.: "Telecommunications and the IEEE Communications Society," *IEEE Commun. Magazine*, 50th Anniversary Issue, pp. 6-14 and 162-167, May 2002.
- JOHANSSON, P., KAZANTZIDIS, M., KAPOOR, R., and GERLA, M.: "Bluetooth: An Enabler for Personal Area Networking," *IEEE Network Magazine*, vol. 15, pp. 28-37, Sept.-Oct 2001.
- JOHNSON, D. B.: "Scalable Support for Transparent Mobile Host Internetworking," *Wireless Networks*, vol. 1, pp. 311-321, Oct. 1995.
- JOHNSON, H. W.: *Fast Ethernet-Dawn of a New Network*, Englewood Cliffs, NJ: Prentice Hall, 1996.
- JOHNSON, N. F., and JAJODA, S.: "Exploring Steganography: Seeing the Unseen," *Computer*, vol. 31, pp. 26-34, Feb. 1998.
- KAHN, D.: "Cryptology Goes Public," *IEEE Commun. Magazine*, vol. 18, pp. 19-28,

---

March 1980.

KAHN, D. : *The Codebreakers*, 2nd ed. , New York: Macmillan, 1995.

KAMOUN, F. , and KLEINROCK, L. : "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks," *Computer Networks*, vol. 3, pp. 337-353, Nov. 1979.

KAPP, S. : "802.11: Leaving the Wire Behind," *IEEE Internet Computing*, vol. 6, pp. 82-85, Jan.-Feb. 2002.

KARN, P. : "MACA-A New Channel Access Protocol for Packet Radio," *ARRL/CRRL Amateur Radio Ninth Computer Networking Conf.* , pp. 134-140, 1990.

KARTALOPOULOS, S. : *Introduction to DWDM Technology: Data in a Rainbow*, New York, NY: IEEE Communications Society, 1999.

KASERA, S. K. , HJALMTYSSON, G. , TOWLSEY, D. F. , and KUROSE, J. F. : "Scalable Reliable Multicast Using Multiple Multicast Channels," *IEEE/ACM Trans. on Networking*, vol. 8, pp. 294-310, 2000.

KATZ, D. , and FORD, P. S. : "TUBA: Replacing IP with CLNP," *IEEE Network Magazine*, vol. 7, pp. 38-47, May/June 1993.

KATZENBEISSER, S. , and PETITCOLAS, F. A. P. : *Information Hiding Techniques for Steganography and Digital Watermarking*, London, Artech House, 2000.

KAUFMAN, C. , PERLMAN, R. , and SPECINER, M. : *Network Security*, 2nd ed. , Englewood Cliffs, NJ: Prentice Hall, 2002.

KELLERER, W. , VOGEL, H.-J. , and STEINBERG, K.-E. : "A Communication Gateway for Infrastructure-Independent 4G Wireless Access," *IEEE Commun. Magazine*, vol. 40, pp. 126-131, March 2002.

KERCKHOFF, A. : "La Cryptographie Militaire," *J. des Sciences Militaires*, vol. 9, pp. 5-38, Jan. 1883 and pp. 161-191, Feb. 1883.

KIM, J. B. , SUDA, T. , and YOSHIMURA, M. : "International Standardization of B-ISDN," *Computer Networks and ISDN Systems*, vol. 27, pp. 5-27, Oct. 1994.

KIPNIS, J. : "Beating the System: Abuses of the Standards Adoptions Process," *IEEE Commun. Magazine*, vol. 38, pp. 102-105, July 2000.

KLEINROCK, L. : "On Some Principles of Nomadic Computing and Multi-Access Communications," *IEEE Commun. Magazine*, vol. 38, pp. 46-50, July 2000.

KLEINROCK, L. , and TOBAGI, F. : "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," *Proc. Nat. Computer Conf.* , pp. 187-201, 1975.

KRISHNAMURTHY, B. , and REXFORD, J. : *Web Protocols and Practice*, Boston: Addison-Wesley, 2001.

KUMAR, V. , KORPI, M. , and SENGODAN, S. : *IP Telephony with H. 323*, New York: Wiley, 2001.

KUROSE, J. F. , and ROSS, K. W. : *Computer Networking: A Top-Down Approach Featuring the Internet*, Boston: Addison-Wesley, 2001.

KWOK, T. : "A Vision for Residential Broadband Service: ATM to the Home," *IEEE Network Magazine*, vol. 9, pp. 14-28, Sept./Oct. 1995.

KYAS, O. , and CRAWFORD, G. : *ATM Networks*, Upper Saddle River, NJ: Prentice Hall, 2002.

LAM, C. K. M. , and TAN, B. C. Y. : "The Internet Is Changing the Music Industry," *Commun. of the ACM*, vol. 44, pp. 62-66, Aug. 2001.

- LANSFORD, J. , STEPHENS, A, and NEVO, R. : "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," IEEE Network Magazine, vol. 15, pp. 20-27, Sept.-Oct 2001.
- LASH, D. A. : The Web Wizard's Guide to Perl and CGI, Boston: Addison-Wesley, 2002.
- LAUBACH, M. E. , FARBER, D. J. , and DUKES, S. D. : Delivering Internet Connections over Cable, New York: Wiley, 2001.
- LEE, J. S. , and MILLER, L. E. : CDMA Systems Engineering Handbook, London: Artech House, 1998.
- LEEPER, D. G. : "A Long-Term View of Short-Range Wireless," Computer, vol. 34, pp. 39-44, June 2001.
- LEINER, B. M. , COLE, R. , POSTEL, J. , and MILLS, D. : "The DARPA Internet Protocol Suite," IEEE Commun. Magazine, vol. 23, pp. 29-34, March 1985.
- LEVINE, D. A. , and AKYILDIZ, I. A. : "PROTON: A Media Access Control Protocol for Optical Networks with Star Topology," IEEE/ACM Trans. on Networking, vol. 3, pp. 158-168, April 1995.
- LEVY, S. : "Crypto Rebels," Wired, pp. 54-61, May/June 1993.
- LI, J. , BLAKE, C. , DE COUTO, D. S. J. , LEE, H. I. , and MORRIS, R. : "Capacity of Ad Hoc Wireless Networks," Proc. 7th Int'l Conf. on Mobile Computing and Networking, ACM, pp. 61-69, 2001.
- LIN, F. , CHU, P. , and LIU, M. : "Protocol Verification Using Reachability Analysis: The State Space Explosion Problem and Relief Strategies," Proc. SIGCOMM '87 Conf. , ACM, pp. 126-135, 1987.
- LIN, Y.-D. , HSU, N.-B. , and HWANG, R.-H. : "QoS Routing Granularity in MPLS Networks", IEEE Commun. Magazine, vol. 40, pp. 58-65, June 2002.
- LISTANI, M. , ERAMO, V. , and SABELLA, R. : "Architectural and Technological Issues for Future Optical Internet Networks," IEEE Commun. Magazine, vol. 38, pp. 82-92, Sept. 2000.
- LIU, C. L. , and LAYLAND, J. W. : "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, vol. 20, pp. 46-61, Jan. 1973.
- LIVINGSTON, D. : Essential XML for Web Professionals, Upper Saddle River, NJ: Prentice Hall, 2002.
- LOSHIN, P. : IPv6 Clearly Explained, San Francisco: Morgan Kaufmann, 1999.
- LOUIS, P. J. : Broadband Crash Course, New York: McGraw-Hill, 2002.
- LU, W. : Broadband Wireless Mobile: 3G and Beyond, New York: Wiley, 2002.
- MACEDONIA, M. R. : "Distributed File Sharing," Computer, vol. 33, pp. 99-101, 2000.
- MADRUGA, E. L. , and GARCIA-LUNA-ACEVES, J. J. : "Scalable Multicasting: the Core-Assisted Mesh Protocol," Mobile Networks and Applications, vol. 6, pp. 151-165, April 2001.
- MALHOTRA, R. : IP Routing, Sebastopol, CA: O'Reilly, 2002.
- MATSUI, M. : "Linear Cryptanalysis Method for DES Cipher," Advances in Cryptology? Eurocrypt '93 Proceedings, Berlin: Springer-Verlag LNCS 765, pp. 386-397, 1994.
- MAUFER, T. A. : IP Fundamentals, Upper Saddle River, NJ: Prentice Hall, 1999.
- MAZIERES, D. , and KAASHOEK, M. F. : "The Design, Implementation, and Operation of an Email Pseudonym Server," Proc. Fifth Conf. on Computer and Commun. Security, ACM, pp. 27-36, 1998.
- MAZIERES, D. , KAMINSKY, M. , KAASHOEK, M. F. , and WITCHEL, E. : "Separating Key Management from File System Security," Proc. 17th Symp. on Operating Systems Prin. , ACM, pp. 124-139, Dec. 1999.



- MCFEDRIES, P.: Using JavaScript, Indianapolis, IN: Que, 2001.
- MCKENNEY, P. E., and DOVE, K. F.: "Efficient Demultiplexing of Incoming TCP Packets," Proc. SIGCOMM '92 Conf., ACM, pp. 269-279, 1992.
- MELTZER, K., and MICHALSKI, B.: Writing CGI Applications with Perl, Boston: Addison-Wesley, 2001.
- MENEZES, A. J., and VANSTONE, S. A.: "Elliptic Curve Cryptosystems and Their Implementation," Journal of Cryptology, vol. 6, pp. 209-224, 1993.
- MERKLE, R. C.: "Fast Software Encryption Functions," Advances in Cryptology? CRYPTO '90 Proceedings, Berlin: Springer-Verlag LNCS 473, pp. 476-501, 1991.
- MERKLE, R. C., and HELLMAN, M.: "On the Security of Multiple Encryption," Commun. of the ACM, vol. 24, pp. 465-467, July 1981.
- MERKLE, R. C., and HELLMAN, M.: "Hiding and Signatures in Trapdoor Knapsacks," IEEE Trans. on Information Theory, vol. IT-24, pp. 525-530, Sept. 1978.
- METCALFE, R. M.: "On Mobile Computing," Byte, vol. 20, p. 110, Sept. 1995.
- METCALFE, R. M.: "Computer/Network Interface Design: Lessons from Arpanet and Ethernet," IEEE Journal on Selected Areas in Commun., vol. 11, pp. 173-179, Feb. 1993.
- METCALFE, R. M., and BOGGS, D. R.: "Ethernet: Distributed Packet Switching for Local Computer Networks," Commun. of the ACM, vol. 19, pp. 395-404, July 1976.
- METZ, C.: "Interconnecting ISP Networks," IEEE Internet Computing, vol. 5, pp. 74-80, March-April 2001.
- METZ, C.: "Differentiated Services," IEEE Multimedia Magazine, vol. 7, pp. 84-90, July-Sept. 2000.
- METZ, C.: "IP Routers: New Tool for Gigabit Networking," IEEE Internet Computing, vol. 2, pp. 14-18, Nov. Dec. 1998.
- MILLER, B. A., and BISDIKIAN, C.: Bluetooth Revealed, Upper Saddle River, NJ: Prentice Hall, 2001.
- MILLER, P., and CUMMINS, M.: LAN Technologies Explained, Woburn, MA: Butterworth-Heinemann, 2000.
- MINOLI, D.: Video Dialtone Technology, New York: McGraw-Hill, 1995.
- MINOLI, D., and VITELLA, M.: ATM & Cell Relay for Corporate Environments, New York: McGraw-Hill, 1994.
- MISHRA, P. P., and KANAKIA, H.: "A Hop by Hop Rate-Based Congestion Control Scheme," Proc. SIGCOMM '92 Conf., ACM, pp. 112-123, 1992.
- MISRA, A., DAS, S., DUTTA, A., MCAULEY, A., and DAS, S.: "IDMP-Based Fast Handoffs and Paging in IP-Based 4G Mobile Networks," IEEE Commun. Magazine, vol. 40, pp. 138-145, March 2002.
- MOGUL, J. C.: "IP Network Performance," in Internet System Handbook, Lynch, D. C. and Rose, M. T. (eds.), Boston: Addison-Wesley, pp. 575-675, 1993.
- MOK, A. K., and WARD, S. A.: "Distributed Broadcast Channel Access," Computer Networks, vol. 3, pp. 327-335, Nov. 1979.
- MOY, J.: "Multicast Routing Extensions," Commun. of the ACM, vol. 37, pp. 61-66, Aug. 1994.
- MULLINS, J.: "Making Unbreakable Code," IEEE Spectrum, pp. 40-45, May 2002.
- NAGLE, J.: "On Packet Switches with Infinite Storage," IEEE Trans. on Commun., vol.

COM-35, pp. 435-438, April 1987.

NAGLE, J. : "Congestion Control in TCP/IP Internetworks," *Computer Commun. Rev.*, vol. 14, pp. 11-17, Oct. 1984.

NARAYANASWAMI, C. , KAMIJOH, N. , RAGHUNATH, M. , INOUE, T. , CIPOLLA, T. , SANFORD, J. , SCHLIG, E. , VENTKITESWARAN, S. , GUNIGUNTALA, D. , KULKARNI, V. , and YAMAZAKI, K. : "IBM's Linux Watch: The Challenge of Miniaturization," *Computer*, vol. 35, pp. 33-41, Jan. 2002.

NAUGHTON, J. : "A Brief History of the Future," Woodstock, NY: Overlook Press, 2000.

NEEDHAM, R. M. , and SCHROEDER, M. D. : "Authentication Revisited," *Operating Systems Rev.*, vol. 21, p. 7, Jan. 1987.

NEEDHAM, R. M. , and SCHROEDER, M. D. : "Using Encryption for Authentication in Large Networks of Computers," *Commun. of the ACM*, vol. 21, pp. 993-999, Dec. 1978.

NELAKUDITI, S. , and ZHANG, Z. L. : "A Localized Adaptive Proportioning Approach to QoS Routing," *IEEE Commun. Magazine* vol. 40, pp. 66-71, June 2002.

NEMETH, E. , SNYDER, G. , SEEBASS, S. , and HEIN, T. R. : *UNIX System Administration Handbook*, 3rd ed. , Englewood Cliffs, NJ: Prentice Hall, 2000.

NICHOLS, R. K. , and LEKKAS, P. C. : *Wireless Security*, New York: McGraw-Hill, 2002.

NIST, "Secure Hash Algorithm," U. S. Government Federal Information Processing Standard 180, 1993.

O'HARA, B. , and PETRICK, A. : 802.11 Handbook: A Designer's Companion. New York: IEEE Press, 1999.

OTWAY, D. , and REES, O. : "Efficient and Timely Mutual Authentication," *Operating Systems Rev.*, pp. 8-10, Jan. 1987.

OVADIA, S. : *Broadband Cable TV Access Networks: from Technologies to Applications*, Upper Saddle River, NJ: Prentice Hall, 2001.

PALAIS, J. C. : *Fiber Optic Commun.*, 3rd ed. , Englewood Cliffs, NJ: Prentice Hall, 1992.

PAN, D. : "A Tutorial on MPEG/Audio Compression," *IEEE Multimedia Magazine*, vol. 2, pp. 60-74, Summer 1995.

PANDYA, R. : "Emerging Mobile and Personal Communication Systems," *IEEE Commun. Magazine*, vol. 33, pp. 44-52, June 1995.

PARAMESWARAN, M. , SUSARLA, A. , and WHINSTON, A. B. : "P2P Networking: An Information-Sharing Alternative," *Computer*, vol. 34, pp. 31-38, July 2001.

PARK, J. S. , and SANDHU, R. : "Secure Cookies on the Web," *IEEE Internet Computing*, vol. 4, pp. 36-44, July-Aug. 2000.

PARTRIDGE, C. , HUGHES, J. , and STONE, J. : "Performance of Checksums and CRCs over Real Data," *Proc. SIGCOMM '95 Conf.*, ACM, pp. 68-76, 1995.

PAXSON, V. : "Growth Trends in Wide-Area TCP Connections," *IEEE Network Magazine*, vol. 8, pp. 8-17, July/Aug. 1994.

PAXSON, V. , and FLOYD, S. : "Wide-Area Traffic: The Failure of Poisson Modeling," *Proc. SIGCOMM '94 Conf.*, ACM, pp. 257-268, 1995.

PEPELJAK, I. , and GUICHARD, J. : *MPLS and VPN Architectures*, Indianapolis, IN: Cisco Press, 2001.

PERKINS, C. E. : *RTP: Audio and Video for the Internet*, Boston: Addison-Wesley, 2002.

PERKINS, C. E. (ed.) : *Ad Hoc Networking*, Boston: Addison-Wesley, 2001.

- PERKINS, C. E. : *Mobile IP Design Principles and Practices*, Upper Saddle River, NJ: Prentice Hall, 1998a.
- PERKINS, C. E. : "Mobile Networking in the Internet," *Mobile Networks and Applications*, vol. 3, pp. 319-334, 1998b.
- PERKINS, C. E. : "Mobile Networking through Mobile IP," *IEEE Internet Computing*, vol. 2, pp. 58-69, Jan.-Feb. 1998c.
- PERKINS, C. E. , and ROYER, E. : "The Ad Hoc On-Demand Distance-Vector Protocol," in *Ad Hoc Networking*, edited by C. Perkins, Boston: Addison-Wesley, 2001.
- PERKINS, C. E. , and ROYER, E. : "Ad-hoc On-Demand Distance Vector Routing," *Proc. Second Ann. IEEE Workshop on Mobile Computing Systems and Applications*, IEEE, pp. 90-100, 1999.
- PERLMAN, R. : *Interconnections*, 2nd ed. , Boston: Addison-Wesley, 2000.
- PERLMAN, R. : *Network Layer Protocols with Byzantine Robustness*, Ph. D. thesis, M. I. T. , 1988.
- PERLMAN, R. . and KAUFMAN, C. : "Key Exchange in IPSec," *IEEE Internet Computing*, vol. 4, pp. 50-56, Nov.-Dec. 2000.
- PETERSON, L. L. , and DAVIE, B. S. : *Computer Networks: A Systems Approach*, San Francisco: Morgan Kaufmann, 2000.
- PETERSON, W. W. , and BROWN, D. T. : "Cyclic Codes for Error Detection," *Proc. IRE*, vol. 49, pp. 228-235, Jan. 1961.
- PICKHOLTZ, R. L. , SCHILLING, D. L. , and MILSTEIN, L. B. : "Theory of Spread Spectrum Communication—A Tutorial," *IEEE Trans. on Commun.* , vol. COM-30, pp. 855-884, May 1982.
- PIERRE, G. , KUZ, I. , VAN STEEN, M. , TANENBAUM, A. S. : "Differentiated Strategies for Replicating Web Documents," *Computer Commun.* , vol. 24, pp. 232-240, Feb. 2001.
- PIERRE, G. , VAN STEEN, M. , and TANENBAUM, A. S. : "Dynamically Selecting Optimal Distribution Strategies for Web Documents," *IEEE Trans. on Computers*, vol. 51, pp. , June 2002.
- PISCITELLO, D. M. , and CHAPIN, A. L. : *Open Systems Networking: TCP/IP and OSI*, Boston: Addison-Wesley, 1993.
- PITT, D. A. : "Bridging—The Double Standard," *IEEE Network Magazine*, vol. 2, pp. 94-95, Jan. 1988.
- PIVA, A. , BARTOLINI, F. , and BARNI, M. : "Managing Copyrights in Open Networks," *IEEE Internet Computing*, vol. 6, pp. 18-26, May-June 2002.
- POHLMANN, N. : *Firewall Systems*, Bonn, Germany: MITP-Verlag, 2001.
- PUZMANOVA, R. : *Routing and Switching: Time of Convergence?*, London: AddisonWesley, 2002.
- RABINOVICH, M. , and SPATSCHECK, O. : *Web Caching and Replication*, Boston: Addison-Wesley, 2002.
- RAJU, J. , and GARCIA-LUNA-ACEVES, J. J. : "Scenario-based Comparison of Source-Tracing and Dynamic Source Routing Protocols for Ad-Hoc Networks," *ACM Computer Communications Review*, vol. 31, October 2001.
- RAMANATHAN, R. , and REDI, J. : "A Brief Overview of Ad Hoc Networks: Challenges and Directions," *IEEE Commun. Magazine*, 50th Anniversary Issue, pp. 20-22, May 2002.
- RATNASAMY, S. , FRANCIS, P. , HANDLEY, M. , KARP, R. , and SHENKER, S. : "A Scalable Content-Addressable Network," *Proc. SIGCOMM '01 Conf.* , ACM, pp. 1161-1172, 2001.
- RIVEST, R. L. : "The MD5 Message-Digest Algorithm," RFC 1320, April 1992.

- 
- RIVEST, R. L. , and SHAMIR, A. : "How to Expose an Eavesdropper," *Commun. of the ACM*, vol. 27, pp. 393-395, April 1984.
- RIVEST, R. L. , SHAMIR, A. , and ADLEMAN, L. : "On a Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Commun. of the ACM*, vol. 21, pp. 120-126, Feb. 1978.
- ROBERTS, L. G. : "Dynamic Allocation of Satellite Capacity through Packet Reservation," *Proc. NCC, AFIPS*, pp. 711-716, 1973.
- ROBERTS, L. G. : "Extensions of Packet Communication Technology to a Hand Held Personal Terminal," *Proc. Spring Joint Computer Conference, AFIPS*, pp. 295-298, 1972.
- ROBERTS, L. G. : "Multiple Computer Networks and Intercomputer Communication," *Proc. First Symp. on Operating Systems Prin. , ACM*, 1967.
- ROSE, M. T. : *The Simple Book*, Englewood Cliffs, NJ: Prentice Hall, 1994.
- ROSE, M. T. : *The Internet Message*, Englewood Cliffs, NJ: Prentice Hall, 1993.
- ROSE, M. T. , and MCCLOGHRIE, K. , *How to Manage Your Network Using SNMP*, Englewood Cliffs, NJ: Prentice Hall, 1995.
- ROWSTRON, A. , and DRUSCHEL, P. : "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. 18th Symp. on Operating Systems Prin. , ACM*, pp. 188-201, 2001a.
- ROWSTRON, A. , and DRUSCHEL, P. : "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Storage Utility," *Proc. 18th Int'l Conf. on Distributed Systems Platforms, ACM/IFIP*, 2001b.
- ROYER, E. M. , and TOH, C.-K. : "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks," *IEEE Personal Commun. Magazine*, vol. 6, pp. 46-55, April 1999.
- RUIZ-SANCHEZ, M. A. , BIRSACK, E. W. , and DABBOUS, W. : "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network Magazine*, vol. 15, pp. 8-23, March-April 2001.
- SAIRAM, K. V. S. S. S. , GUNASEKARAN, N. , and REDDY, S. R. : "Bluetooth in Wireless Communication," *IEEE Commun. Mag.*, vol. 40, pp. 90-96, June 2002.
- SALTZER, J. H. , REED, D. P. , and CLARK, D. D. : "End-to-End Arguments in System Design," *ACM Trans. on Computer Systems*, vol. 2, pp. 277-288, Nov. 1984.
- SANDERSON, D. W. , and DOUGHERTY, D. : *Smileys*, Sebastopol, CA: O'Reilly, 1993.
- SARI, H. , VANHAVERBEKE, F. , and MOENECLAHEY, M. : "Extending the Capacity of Multiple Access Channels," *IEEE Commun. Magazine*, vol. 38, pp. 74-82, Jan. 2000.
- SARIKAYA, B. : "Packet Mode in Wireless Networks: Overview of Transition to Third Generation," *IEEE Commun. Magazine*, vol. 38, pp. 164-172, Sept. 2000.
- SCHNEIER, B. : *Secrets and Lies*, New York: Wiley, 2000.
- SCHNEIER, B. : *Applied Cryptography*, 2nd ed. , New York: Wiley, 1996.
- SCHNEIER, B. : *E-Mail Security*, New York: Wiley, 1995.
- SCHNEIER, B. : "Description of a New Variable-Length Key, 64-Bit Block Cipher [Blowfish]," *Proc. of the Cambridge Security Workshop, Berlin: Springer-Verlag LNCS 809*, pp. 191-204, 1994.
- SCHNORR, C. P. : "Efficient Signature Generation for Smart Cards," *Journal of Cryptology*, vol. 4, pp. 161-174, 1991.
- SCHOLTZ, R. A. : "The Origins of Spread-Spectrum Communications," *IEEE Trans. on Commun. , vol. COM-30*, pp. 822-854, May 1982.
- SCOTT, R. : "Wide Open Encryption Design Offers Flexible Implementations," *Cryptologia*, vol. 9, pp. 75-90, Jan. 1985.

- 
- SEIFERT, R. : *The Switch Book*, Boston: Addison-Wesley, 2000.
- SEIFERT, R. : *Gigabit Ethernet*, Boston: Addison-Wesley, 1998.
- SENN, J. A. : "The Emergence of M-Commerce," *Computer*, vol. 33, pp. 148-150, Dec. 2000.
- SERJANTOV, A. : "Anonymizing Censorship Resistant Systems," *Proc. First Int'l Workshop on Peer-to-Peer Systems*, Berlin: Springer-Verlag LNCS, 2002.
- SEVERANCE, C. : "IEEE 802.11: Wireless Is Coming Home," *Computer*, vol. 32, pp. 126-127, Nov. 1999.
- SHAHABI, C. , ZIMMERMANN, R. , FU, K. , and YAO, S.-Y. D. : "YIMA: A Second-Generation Continuous Media Server," *Computer*, vol. 35, pp. 56-64, June 2002.
- SHANNON, C. : "A Mathematical Theory of Communication," *Bell System Tech. J.*, vol. 27, pp. 379-423, July 1948; and pp. 623-656, Oct. 1948.
- SHEPARD, S. : *SONET/SDH Demystified*, New York: McGraw-Hill, 2001.
- SHREEDHAR, M. , and VARGHESE, G. : "Efficient Fair Queueing Using Deficit Round Robin," *Proc. SIGCOMM '95 Conf.*, ACM, pp. 231-243, 1995.
- SKOUDIS, E. : *Counter Hack*, Upper Saddle River, NJ: Prentice Hall, 2002.
- SMITH, D. K. , and ALEXANDER, R. C. : *Fumbling the Future*, New York: William Morrow, 1988.
- SMITH, R. W. : *Broadband Internet Connections*, Boston: Addison Wesley, 2002.
- SNOEREN, A. C. , and BALAKRISHNAN, H. : "An End-to-End Approach to Host Mobility," *Int'l Conf. on Mobile Computing and Networking*, ACM, pp. 155-166, 2000.
- SOBEL, D. L. : "Will Carnivore Devour Online Privacy," *Computer*, vol. 34, pp. 87-88, May 2001.
- SOLOMON, J. D. : *Mobile IP: The Internet Unplugged*, Upper Saddle River, NJ: Prentice Hall, 1998.
- SPOHN, M. , and GARCIA-LUNA-ACEVES, J. J. : "Neighborhood Aware Source Routing," *Proc. ACM MobiHoc 2001*, ACM, pp. 2001.
- SPURGEON, C. E. : *Ethernet: The Definitive Guide*, Sebastopol, CA: O'Reilly, 2000.
- STALLINGS, W. : *Data and Computer Communications*, 6th ed., Upper Saddle River, NJ: Prentice Hall, 2000.
- STEINMETZ, R. , and NAHRSTEDT, K. : *Multimedia Fundamentals. Vol. 1: Media Coding and Content Processing*, Upper Saddle River, NJ: Prentice Hall, 2002.
- STEINMETZ, R. , and NAHRSTEDT, K. : *Multimedia Fundamentals. Vol. 2: Media Processing and Communications*, Upper Saddle River, NJ: Prentice Hall, 2003a.
- STEINMETZ, R. , and NAHRSTEDT, K. : *Multimedia Fundamentals. Vol. 3: Documents, Security, and Applications*, Upper Saddle River, NJ: Prentice Hall, 2003b.
- STEINER, J. G. , NEUMAN, B. C. , and SCHILLER, J. L. : "Kerberos: An Authentication Service for Open Network Systems," *Proc. Winter USENIX Conf.*, USENIX, pp. 191-201, 1988.
- STEVENS, W. R. : *UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTI*, Upper Saddle River, NJ: Prentice Hall, 1997.
- STEVENS, W. R. : *TCP/IP Illustrated, Vol. 1*, Boston: Addison-Wesley, 1994.
- STEWART, R. , and METZ, C. : "SCTP: New Transport Protocol for TCP/IP," *IEEE Internet Computing*, vol. 5, pp. 64-69, Nov.-Dec. 2001.
- STINSON, D. R. : *Cryptography Theory and Practice*, 2nd ed., Boca Raton, FL: CRC Press, 2002.

- STOICA, I. , MORRIS, R. , KARGER, D. , KAASHOEK, M. F. , and BALAKRISHNAN, H. : "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. SIGCOMM '01 Conf. , ACM, pp. 149-160, 2001.
- STRIEGEL, A. , and MANIMARAN, G. : "A Survey of QoS Multicasting Issues," IEEE Commun. Mag. , vol. 40, pp. 82-87, June 2002.
- STUBBLEFIELD, A. , IOANNIDIS, J. , and RUBIN, A. D. : "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," Proc Network and Distributed Systems Security Symp. , ISOC, pp. 1-11, 2002.
- SUMMERS, C. K. : ADSL: Standards, Implementation, and Architecture, Boca Raton, FL; CRC Press, 1999.
- SUNSHINE, C. A. , and DALAL, Y. K. : "Connection Management in Transport Protocols," Computer Networks, vol. 2, pp. 454-473, 1978.
- TANENBAUM, A. S. : Modern Operating Systems, Upper Saddle River, NJ; Prentice Hall, 2001.
- TANENBAUM, A. S. , and VAN STEEN, M. : Distributed Systems: Principles and Paradigms, Upper Saddle River, NJ; Prentice Hall, 2002.
- TEGER, S. , and WAKS, D. J. : "End-User Perspectives on Home Networking," IEEE Commun. Magazine, vol. 40, pp. 114-119, April 2002.
- THYAGARAJAN, A. S. , and DEERING, S. E. : "Hierarchical Distance Vector Multicast Routing for the MBone," Proc. SIGCOMM '95 Conf. , ACM, pp. 60-66, 1995.
- TITTEL, E. , VALENTINE, C. , BURMEISTER, M. , and DYKES, J. : Mastering XHTML, Alameda, CA; Sybex, 2001.
- TOKORO, M. , and TAMARU, K. : "Acknowledging Ethernet," Compcon, IEEE, pp. 320-325, Fall 1977.
- TOMLINSON, R. S. : "Selecting Sequence Numbers," Proc. SIGCOMM/SIGOPS Inter-process Commun. Workshop, ACM, pp. 11-23, 1975.
- TSENG, Y.-C. , WU, S.-L. , LIAO, W.-H. , and CHAO, C.-M. : "Location Awareness in Ad Hoc Wireless Mobile Networks," Computer, vol. 34, pp. 46-51, 2001.
- TUCHMAN, W. : "Hellman Presents No Shortcut Solutions to DES," IEEE Spectrum, vol. 16, pp. 40-41, July 1979.
- TURNER, J. S. : "New Directions in Communications (or Which Way to the Information Age)," IEEE Commun. Magazine, vol. 24, pp. 8-15, Oct. 1986.
- VACCA, J. R. : I-Mode Crash Course, New York; McGraw-Hill, 2002.
- VALADE, J. : PHP & MySQL for Dummies, New York; Hungry Minds, 2002.
- VARGHESE, G. , and LAUCK, T. : "Hashed and Hierarchical Timing Wheels; Data Structures for the Efficient Implementation of a Timer Facility," Proc. 11th Symp. on Operating Systems Prin. , ACM, pp. 25-38, 1987.
- VARSHNEY, U. , SNOW, A. , MCGIVERN, M. , and HOWARD, C. : "Voice over IP," Commun. of the ACM, vol. 45, pp. 89-96, 2002.
- VARSHNEY, U. , and VETTER, R. : "Emerging Mobile and Wireless Networks," Commun. of the ACM, vol. 43, pp. 73-81, June 2000.
- VETTER, P. , GODERIS, D. , VERPOOTEN, L. , and GRANGER, A. : "Systems Aspects of APON/VDSL Deployment," IEEE Commun. Magazine, vol. 38, pp. 66-72, May 2000.
- WADDINGTON, D. G. , and CHANG, F. : "Realizing the Transition to IPv6," IEEE Commun.



Mag., vol. 40, pp. 138-148, June 2002.

WALDMAN, M., RUBIN, A. D., and CRANOR, L. F.: "Publius: A Robust, Tamper-Evident, Censorship-Resistant, Web Publishing System," Proc. Ninth USENIX Security Symp., USENIX, pp. 59-72, 2000.

WANG, Y., and CHEN, W.: "Supporting IP Multicast for Mobile Hosts," Mobile Networks and Applications, vol. 6, pp. 57-66, Jan.-Feb. 2001.

WANG, Z.: Internet QoS, San Francisco: Morgan Kaufmann, 2001.

WARNEKE, B., LAST, M., LIEBOWITZ, B., and PISTER, K. S. J.: "Smart Dust: Communicating with a Cubic Millimeter Computer," Computer, vol. 34, pp. 44-51, Jan. 2001.

WAYNER, P.: Disappearing Cryptography: Information Hiding, Steganography, and Watermarking, 2nd ed., San Francisco: Morgan Kaufmann, 2002.

WEBB, W.: "Broadband Fixed Wireless Access as a Key Component of the Future Integrated Communications Environment," IEEE Commun. Magazine, vol. 39, pp. 115-121, Sept. 2001.

WEISER, M.: "Whatever Happened to the Next Generation Internet?," Commun. of the ACM, vol. 44, pp. 61-68, Sept. 2001.

WELTMAN, R., and DAHBURA, T.: LDAP Programming with Java, Boston: AddisonWesley, 2000.

WESSELS, D.: Web Caching, Sebastopol, CA: O'Reilly, 2001.

WETTEROTH, D.: OSI Reference Model for Telecommunications, New York: McGraw-Hill, 2001.

WILJAKKA, J.: "Transition to Ipv6 in GPRS and WCDMA Mobile Networks," IEEE Commun. Magazine, vol. 40, pp. 134-140, April 2002.

WILLIAMSON, H.: XML: The Complete Reference, New York: McGraw-Hill, 2001.

WILLINGER, W., TAQQU, M. S., SHERMAN, R., and WILSON, D. V.: "Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," Proc. SIGCOMM '95 Conf., ACM, pp. 100-113, 1995.

WRIGHT, D. J.: Voice over Packet Networks, New York: Wiley, 2001.

WYLIE, J., BIGRIGG, M. W., STRUNK, J. D., GANGER, G. R., KILICCOTE, H., and KHOSLA, P. K.: "Survivable Information Storage Systems," Computer, vol. 33, pp. 61-68, Aug. 2000.

XYLOMENOS, G., POLYZOS, G. C., MAHONEN, P., and SAARANEN, M.: "TCP Performance Issues over Wireless Links," IEEE Commun. Magazine, vol. 39, pp. 52-58, April 2001.

YANG, C.-Q., and REDDY, A. V. S.: "A Taxonomy for Congestion Control Algorithms in Packet Switching Networks," IEEE Network Magazine, vol. 9, pp. 34-45, July/Aug. 1995.

YUVAL, G.: "How to Swindle Rabin," Cryptologia, vol. 3, pp. 187-190, July 1979.

ZACKS, M.: "Antiterrorist Legislation Expands Electronic Snooping," IEEE Internet Computing, vol. 5, pp. 8-9, Nov.-Dec. 2001.

ZADEH, A. N., JABBAR, B., PICKHOLTZ, R., and VOJCIC, B.: "Self-Organizing Packet Radio Ad Hoc Networks with Overlay (SOPRANO)," IEEE Commun. Mag., vol. 40, pp. 149-157, June 2002.

ZHANG, L.: "Comparison of Two Bridge Routing Approaches," IEEE Network Magazine, vol. 2, pp. 44-48, Jan./Feb. 1988.

ZHANG, L.: "RSVP: A New Resource ReSerVation Protocol," IEEE Network Magazine, vol. 7, pp. 8-18, Sept./Oct. 1993.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.



- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.



---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.



- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

---

ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.

ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.

ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.

ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.

ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

- 
- ZHANG, Y. , and RYU, B. : "Mobile and Multicast IP Services in PACS; System Architecture, Prototype, and Performance," *Mobile Networks and Applications*, vol. 6, pp. 81-94, Jan.-Feb. 2001.
- ZIMMERMANN, P. R. : *The Official PGP User's Guide*, Cambridge, MA; M. I. T. Press, 1995a.
- ZIMMERMANN, P. R. : *PGP: Source Code and Internals*, Cambridge, MA; M. I. T. Press, 1995b.
- ZIPF, G. K. : *Human Behavior and the Principle of Least Effort; An Introduction to Human Ecology*, Boston; Addison-Wesley, 1949.
- ZIV, J. , and LEMPEL, Z. : "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT-23, pp. 337-343, May 1977.