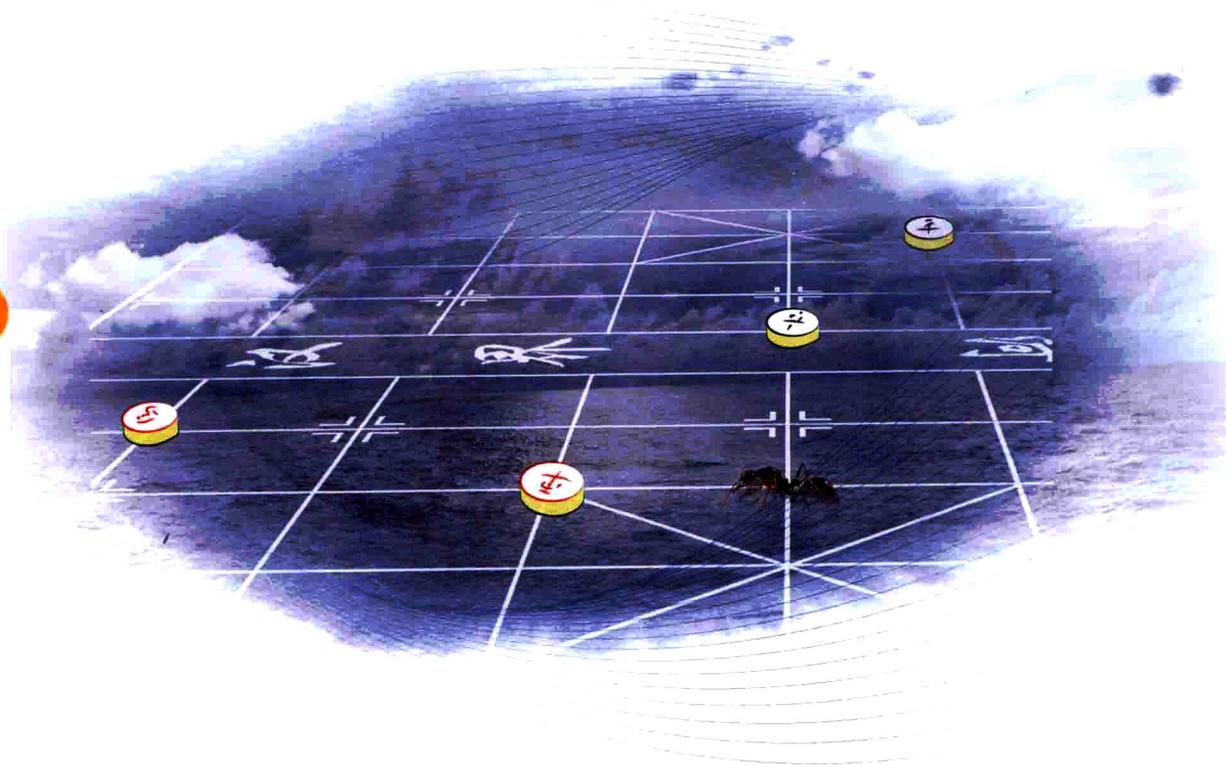


着眼于云计算网络虚拟化中所用到的网络技术原理  
深入解读一些网络新技术和新方案  
快速掌握网络技术必备书

**Broadview**<sup>®</sup>  
www.broadview.com.cn

# 云计算网络珠玑

李俊武 | 著



中国工信出版集团



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
http://www.phei.com.cn

# 云计算网络珠玑

李俊武 | 著

电子工业出版社

Publishing House of Electronics Industry

## 内 容 简 介

本书着眼于讲解云计算网络虚拟化中所用到的网络技术原理，重点展示以太网及 TCP/IP 网络中各种技术内在的关联脉络，包括从传统的 MAC、IP、安全和 QoS 等到新兴的 Trill、LISP、DPI 和 CDN 等技术，从以太网交换机的二层转发、三层路由和 Linux 的 TCP/IP 协议栈到 MAC-in-MAC、VXLAN 和 Neutron 等新方案，以及从传统的数据中心三层架构到扁平化大二层和 SDN/NFV 等新架构。另外，本书对一些网络新技术和新方案（如 SR-IOV、Openflow、DPDK、Serverswitch 等）也进行了阐述，并讨论了在若干技术问题上网络的发展趋势。

本书内容囊括了以太网和 TCP/IP 网络技术的每个方面，并选择大量的实例进行详细描述，其中每个技术点和实例都是经过精心选择的，既兼顾网络技术发展的顺序，也遵循 TCP/IP 网络四层从下到上的划分；另外，还从当前的技术热点上选取了 SDN、Openflow、VXLAN 等新兴技术的发展现状和发展趋势进行详细剖析，便于读者在逻辑思维上顺畅地理解，并向读者提供了一条快速掌握网络技术的学习途径。通览全书后，读者可以尽快建立自己在网络方面的技术知识体系。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有，侵权必究。

### 图书在版编目（CIP）数据

云计算网络珠玑 / 李俊武著. —北京：电子工业出版社，2015.3  
ISBN 978-7-121-25377-5

I. ①云… II. ①李… III. ①计算机网络 IV. ①TP393

中国版本图书馆 CIP 数据核字（2014）第 313547 号

策划编辑：董 英

责任编辑：徐津平

特约编辑：顾慧芳

印 刷：北京中新伟业印刷有限公司

装 订：三河市华成印务有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×980 1/16 印张：19 字数：284 千字

版 次：2015 年 3 月第 1 版

印 次：2015 年 3 月第 1 次印刷

印 数：3000 册 定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 [zltz@phei.com.cn](mailto:zltz@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线：（010）88258888。

# 专家推荐

交换机/路由器技术支撑了近几十年的互联网的发展,然而随着网络复杂度的提升、软件与网络深度结合的需求、云计算与大数据时代对网络新的挑战等新业务场景的激增,传统的交换机/路由器的封闭系统,越来越难以满足需求,因此始于 2006 年的软件定义/控制网络技术(简称 SDN)逐步得到发展,并且渐渐形成事实上的行业标准。

借助于 SDN 技术,技术人员在业务场景发生变化,客户需求变化的情况下无须手动登录交换机进行业务设置,在云计算领域由于多租户的特征,SDN 技术尤为重要。

UCloud 是一家商业化的公有云服务供应商,在 2012 年开始接触、实践自有的 SDN 技术,并利用 SDN 技术实现了多种云计算环境下复杂业务的网络需求,包括转发、容灾、QoS、软交换、硬件 SDN 控制,等等。

《云计算网络珠玑》这本书从网络的基本原理,常用技术、主流网络应用协议、交换的基本原理和 Linux 操作系统的网络实现等内容作为引入,SDN 与 OVS 技术作为基础以及 Neutron 作为实践,由浅入深、从原理到实践地讲解了基础网络向 SDN 网络演进的过程,是网络工作者以及 SDN 软件工作者一本不可多得的启蒙性和工具性书籍。

希望更多的读者能从此书中获得相关知识,努力提升网络的可用性及易用

性，共同为互联网技术的进步添砖加瓦。

UCloud CTO 莫显峰

《云计算网络珠玑》这本书从网络 TCP/IP 技术基本原理开始，陆续讲到了 SDN、云计算及 OpenStack 的网络，我们不仅能从原理中系统地梳理相关知识，而且也能学习到网络技术的发展趋势，尤其是其中大量篇幅涉及 OpenStack 网络的相关知识，对于学习和掌握 OpenStack 也有很大帮助，感谢作者的辛勤付出，也希望读者能从中学习和了解当前网络发展的最新状态

杜玉杰（新浪微博@ben 杜玉杰）

经常听人说，OpenStack 平台中最复杂最难处理的就是网络问题。之所以会这样，除了 OpenStack Neutron 确实存在一些不足之外，一个很重要的原因就是从事 OpenStack 研发和部署的工程师中，很多人对网络不了解，正所谓难者不会，会者不难。而李俊武的这本书可谓是及时雨，根据自己多年的底层网络研发经验，深入剖析了 OpenStack 网络架构，实乃 OpenStack 网络从业者必读书籍。

《深度解析 SDN：利益、战略、技术、实践》作者，盛科网络  
SDN 云计算研发总监 张卫峰（新浪微博@盛科张卫峰）

对我而言，写书是一门非常复杂的手艺，需要有很好的知识积累，最重要的是要有耐心和毅力。认识小 5 是在一个 QQ 群里，那时我在思科，由于工作原因主要研究新的技术和应用，SDN 从我加入群那时开始越来越火，每天的新闻也是层出不穷，仔细研究，确实能从聊天记录中过滤出有用的信息。

群里总有技术讨论，这是一件很好的事情，也让大家最快速地对各种新的技术和解决方案有宏观的印象。但是我也发现了一个比较严重的问题——大部分的



软件工程师根本不懂网络。如果创建一套 Web 系统或者 OA 办公，那么后台业务流转的知识体系相对比较集中，即使没看干过具体项目，相对的也容易理解。但是网络技术是不同的，因为历史原因通信体系过于庞大，从各种不同的载波信号到上层的应用协议，如果没有真正的知识体系积累，即使编程的水平再高也无法交出合适的网络应用。

小 5 决定写这本书的时候，曾经让我看过初稿，其中对传统网络的介绍是我很高兴看到的，这实际上也给想走上这条路的程序员们一个了解网络知识体系的平台，让他们可以创造出更好的上层应用。

———微软（中国）云计算解决方案架构师周博洋  
（花名学霸，新浪微博@种地的天使-重生）

作为一名 SDN 行业的从业人员，我非常看重工程师的基本素质和综合能力，Pica8 公司在选择人才的过程中也以此为标准。在与俊武的多次交流中，我认为俊武展现出非常良好的技术功底和开放的视野，因此我非常高兴看到本书的出版！

俊武从网络研发工程师做起，对硬件和软件都有丰富的实践经验，随着 OpenStack 等开源云平台的发展，俊武又开始关注系统问题，希望为颇受诟病的 Neutron 提供可用的网络平台。我相信他正是对这些实战经验的总结最终促成了本书的出版。与学院派教科书不同，本书更看重知识的层层递进和最终应用，希望读者能够通过本书不仅可以学习到网络的基础知识，更能找到网络知识的用武之地。

今天有机会为俊武的新作《云计算网络珠玑》写推荐序也让我感到新一代的工程师不再是“自学自乐”，他们更希望能够分享知识，希望把自己的经验积累传播给更多的同路者。衷心希望有更多的学有所成者开始行动，像俊武一样为网络行业的生态繁荣贡献自己的力量！

Pica8 公司创始人兼工程副总裁 杜林

起源于美国斯坦福大学的 SDN 技术在工业界引起关注的同时，在教育领域也早已成为研究热点。然而在与很多高校同学的交流中我明显感觉到目前很多同学的基础知识较为薄弱，有些同学想当然地认为有了 SDN 就不需要学习掌握传统网络知识，而更多的同学对 SDN 技术的应用感到困惑，最明显的表现就是常常可以听到同学们探讨学习 SDN 技术毕业后是否可以找到工作？所有这些问题加深了我长久以来有的一个观点：最好的教师不是来自于课堂，而应该来自于工程一线，教育是为科研服务的，只有经历过一线战火的工程师才能做到庖丁解牛，把握系统设计的真谛，将真正的市场需求传递给象牙塔的学子。

Pica8 公司作为一家专注于 SDN 技术的硅谷创业公司，与俊武在职业的不同阶段都有过交集。无论是拜读俊武在网络上留下的网络学习笔记，还是学习俊武关于网络技术的专利，我都可以感受到俊武强烈的分享欲望。今天很高兴看到《云计算网络珠玑》一书的出版，在众多的 SDN 相关书籍中，我认为俊武的最新力作是我心目当中最理想的教学材料。从硬件平台到软件架构，从网络基础知识到最新的 OpenStack 应用，这本来自一线工程师的著作很好地解答了同学们关于 SDN 的种种问题。

最后附上曾经写过的一个段子：学习 SDN，请先带好证件（Tag）上树（STP），只有通过最短路径（OSPF）率先登高望远的同学才能发现自己的边界（BGP），从而有机会启动中控器（Openflow Controller）打败 Boss，最后通过全新的方式就可以出溜（Flow）下树了。有了俊武的《云计算网络珠玑》，落地成功率 100%！

Pica8 公司中国区业务发展总监 杨勇涛

闻好友俊武的《云计算网络珠玑》一书要出版了，心中甚是欣喜，作者这些年在云计算网络方面的研究终于可以和大家分享了。我应作者要求写个序，这还真有点勉为其难，从事云计算这个行业时间也不短了，但都是“搞技术”，写文

章还真不是强项。

从业这些年来我深刻地认识到：网络才是云计算 IT 架构的核心，云平台内部需要设计优良的网络支撑，云提供的各种服务与用户之间依靠设计优良的网络连接，云的安全要靠其网络安全来保证，云的灵活性也要靠网络的灵活性来保证，最终将网络作为一种服务提供给用户。可以说高质、安全、灵活、可靠的网络是云的基础。

从本书的章节设计就可以看出作者的良苦用心，首先是网络基础知识的讲解，然后从硬件层面和操作系统层面分别进行讲解，最后拔高到包括 SDN、Openflow、OpenVswitch 等业界新技术的介绍。当然理论离不开实践，紧接着作者就以 OpenStack 中的 Neutron 为例详细阐述了云计算网络虚拟化的功能点和底层实现，这部分可以说正是作者多年研究之所在。

从传统 IT 时代到云计算时代，网络技术都一直伴随左右，并不断地演进着，相信本书能够从全局到细节带你走进云计算时代的网络世界，成为你网络解决方案新的思想源泉。

曾任世纪互联云计算平台 CloudEx 的研发经理，现在中国  
数码集团旗下北京新网数码信息技术有限公司（www.  
xinnet.com）云计算事业部任高级产品运营经理 常晓东

没有对云计算的热爱，没有对 SDN、NFV、Overlay 等新网络技术的深度理解，小武就不会有这本书。新 IT 时代新技术层出不穷，但追本溯源，唯有网络是上层应用的基础。小武这本书来源于他从业以来的经验积累，通过实际出发帮助网络爱好者打好这个基础，这是很多同类书籍未必能做到的，在这里分享给大家。

H3C 网络及安全产品部副部长 叶航晖（新浪微博@叶航晖）

SDN 网络为沉寂已久的网络世界带来了变革的机会,对网络从业人员的知识体系结构也提出了更高的要求,产业界非常需要大量的专业书籍来加快完善从业人员的知识体系。本书是为数不多的一本专注于 SDN 网络开源技术的专业书籍,对 OpenStack、Openflow 等开源技术进行了深入的讲解,同时对网络设备的转发流程做了深入的讲解,很好地兼顾了网络工程师和软件工程师的需求,也适合从事网络售前、营销工作的专业人士阅读。

华为公司售前工程师 赵博(新浪微博@胖猴信徒赵博)

感谢俊武在他的新书里面留一块地让我写点什么,一直以来我都只在网络技术圈混,是系统和软件的小白,可以有机会认识俊武这样的“跨界高手”实属有缘,当然也要感谢“SDN”可以让更多的软件编程人员和网络人员产生交集。

身为网络从业人员,深知行业变革的迅猛,这一二年中,影响产业最大的就属 SDN,从谷歌等大型数据中心的技术落地,到整个传统 IT 架构的变革,SDN 可谓“云”而动,风生水起,网络从小到大,架构的简单,业务的灵活部署,运维的自动化和智能,一直是客户和网络厂商所追求的目标,而 SDN 恰恰提供了这样一个广阔的舞台,它可以给我们无限想象的空间,让不可能成为可能,让网络可以更好地支撑业务,更大地发挥效能。

理想很丰满,而现实往往并不如我们所愿,当 SDN 充斥在网络的媒体上、活跃在展会中,成为茶余饭后不得不提的“谈资”时,我们却需要在热潮中保持冷静,新兴的技术和理论层出不穷,究竟哪些会真正给业务带来帮助,在 SDN 化的过程中,我们不能只看到愿景的美好,也要看到落地的艰难,并不存在“完美”的技术和设备,只有最适合业务发展的选择。

在 SDN 逐步实现的过程中,我发现一个有趣的现象,在国内,开发人员和网络人员以前往往处在不同的信息孤岛上,彼此在各自的领域耕耘多年,当 SDN

从梦想走进现实时，彼此都发现在交流和沟通上有的时候并不合拍，软件人员对网络的理解和网络人员对软件的理解往往都存在经验主义的主观偏差，而这恰恰可能会导致在合作的过程中不能协调一致，使得“S”和“N”逐渐脱节，南辕北辙。能够让软件开发人员了解到网络的本源，而让网络人员能够学习到软件实现的要素，恰恰是目前整个行业学习者和从业者的迫切需要，也是未来行业对技术人才的必备要求之一。

俊武的作品恰恰捅破了这层窗户纸，俊武从真实的网络场景出发，根据自身经验，把 SDN 的技术逐一梳理，从软件到硬件，从系统到网络，可谓面面俱到，可以让读者看到 SDN 的“全貌”，所以是一本不可多得的学习指南。恰恰也是因为俊武本身的“跨界”能力和经验，才可以使得这本书有别于其他纯软件或纯网络的技术书籍。该书文字言简意赅，浅显易懂，图文并茂，有理有据，在阅读的过程中，带给我们一种技术上的“实在”，相信一定会给致力于 SDN 产业的从业人员带来帮助。

俊武的坚持和勤奋值得敬佩，衷心祝愿在 SDN 的道路上，我们可以有更多的交流，可以学到更多，最后，祝这本书大卖，也祝俊武工作顺利，身体健康。

网络从业者 KkBLuE（新浪微博@KkBLuE 知行合一）

市面上涉及网络技术相关的书不一而足。从最基础的 TCP/IP 到最流行的 SDN，包括当下最热的 OpenStack 中的网络模块，俊武的这本书都做了深入的分析。xNet（网锐）公司是一家在南京注册的开放网络和 SDN/NFV 方案提供商，我会把这本书推荐给我们公司的员工，它应该能帮助各个级别的读者快速理解和认识各种网络技术。

xNet（网锐）公司联合创始人 张立岗

江苏省未来网络创新研究院成立于 2011 年，是由南京市政府、北京邮电大学、中国科学院计算技术研究所、清华大学等作为理事单位组建的事业法人单位。

该研究院是国内最大的第一家专门从事未来网络核心技术研发的科研机构，为省属科研事业单位，致力于建设成为国家级未来网络的协同创新中心。它依托著名的科研院所，通过引进国内外顶级高端人才和技术团队，促进国际合作，为产业发展提供源源不断的动力，增强对信息产业、高技术服务业、经济社会发展的辐射带动作用。

这本书循序渐进地阐述了云网络实践所需的基本概念，着重介绍了围绕 OpenStack 平台 Neutron 组件的研究探索，此举势必会促进业界深化对云计算网络的认知，推动 SDN 和 NFV 的落地实施。作者将其积累的经验倾囊相授，授人以渔，推荐大家一阅！

江苏省未来网络创新研究院 SDNLAB 平台负责人 魏亮 (<http://www.sdnlab.com/>)

记得 2007 年我在实验室构建一个面向生物基因分析的网格计算门户时，当时网格计算还方兴未艾，很快学术圈又出现了一个新词“云计算”，单纯从“资源池化”和“系统可扩展性”这些方面看，很容易产生“论文灌水所创造的伪概念”的错觉。但随着虚拟化等技术成熟和中小企业迫切需求的东风，越来越多的云开始落地：阿里的飞天 5k 项目支撑了天猫的 571 亿双十一单日销售记录，阿里云开始托管政务云和金融云，Ucloud 依靠出色的安全防护能力提供了业内专业的游戏云服务等。积“云”终成雨，润物细无声，人们不知不觉中开始享受到了云计算带来的便利。

2011 年诞生了 OpenStack 项目，更是将云计算中最复杂的 IaaS 系统提供了开源的实现，解决了一般企业部署自有 IaaS 服务的问题，并迅速获得社区和业界的广泛支持。国内如美团、携程和京东等公司利用 OpenStack 提供了云计算服务。尽管如此，维护高效、稳定和易用的云计算系统是有挑战的，尤其是网络虚拟化模块：设计复杂，运行易出错，排错困难。原因在于不同系统的网络环境很可能

大不相同，不同厂商的网络设备间的控制、数据协议也不同，在网络边界上的各类 middlebox，以及出于性能或其他因素所需增加的机制，使得网络模块的设计需要考虑到方方面面，而这往往是从从事网络虚拟化功能开发的程序员所不擅长的，而 IaaS 服务提供商就需要洞察其中之事。我在绿盟研究新型软件定义安全架构，虽然该架构关注安全本身，与虚拟化和 SDN 松耦合，但如果设计者不了解这两个部分，是不可能交付一个适用于不同场景的高效系统的；如果运维者不了解这两个部分，也不可能提供长期稳定的服务。我相信作者编写此书的初衷是他深刻认识到了这点。

这本书从网络基础入手，进而介绍以 OpenStack Neutron 网络虚拟化，深入浅出。第一部分介绍了传统网络的架构、协议和实现，以及相关的网络技术，这些技术往往会被用于网络虚拟化的实现中，如隧道、VLAN 等部分有助于读者理解 Neutron 中节点在 Overlay 网络如何连接，Iptables 有助于理解 Neutron 的安全组和 FWaaS 等机制，openvswitch 和 SDN 部分有助于理解 OpenStack 如何和 SDN 控制器对接。作者俊武对 Neutron 有很深入的研究，他针对 Neutron 高级话题的讨论和 Neutron 发展的分析，都非常值得相关从业人员思考和讨论。

绿盟科技研究院 资深研究员 博士 刘文懋（新浪微博@marvel）

记得我在 2012 年最开始研究 OpenStack 时，网络基础较差，而网络又是 OpenStack 乃至云计算应用中的重要组成部分，因此在对 Neutron 项目的学习中花费了相当大的精力。route, iptables, namespace, OpenvSwitch, Vxlan 等光这些命令、工具和概念就足以让人眼花缭乱。虽然自己也在博客中记录了一些，但终究是不够系统，不够专业。直到看到俊武这本书，我在想，我是不是该删掉那些粗浅的博客了……

孔令贤（新浪微博@孔令贤 HW）

如果说过去十年哪一种事物产生的作用最深远，我会毫不犹豫地选择“网络”。计算机网络自从诞生的那一天开始，就在颠覆复着我们一切的传统思维。它不仅改变了人与人之间的通信方式，让人与人之间距离不再遥远，它更改变了传统的知识获取模式，让一个人不再受限于周围环境给他带来的经验，而可以获取到这个世界上一切的知识，而这些几乎都是免费的。我们甚至可以相信，未来的一天，我们可以从网络中获取任何事物。

知识，或者信息的增长速度，在很大程度上取决于网络发展的速度，这也决定了 IT 系统的模型和架构。当网络发展的速度满足不了数据处理的要求时，系统的设计往往会趋向于选择分布式，就像当年的 PC 时代；而当网络的速度远超过数据处理的能力要求时，系统则倾向于选择将所有的数据通过网络汇聚到核心来处理，这就是当前的大数据时代。进入到二十一世纪之后，光纤网络带来的性能提升远超过了单一业务系统的处理能力要求，人类可以将大量分布式系统集中起来以获得更高的处理能力和处理范围，这就是云计算。大量的数据从相对简单的终端汇聚到云计算数据中心，通过分布式存储、并行计算来构建更大规模以及更高性能的系统，通过数据挖掘和分析来获取海量数据中的个体和群体价值。

计算机网络技术的发展，也是伴随着这个模型进步的，不过却经历了无数次迭代和标准之争，从最早的 AppleTalk，发展到 X.25、帧中继，最后 TCP/IP 战胜了 ATM，一统江湖成为了通信世界的主宰者。在此之后，计算机网络的发展经历了一个相对平缓的阶段，只是在处理能力、复杂度、业务质量这些方面做了一些更新。但是进入二十一世纪之后，云计算带来的变革又一次将计算机网络技术的发展推到了一个新的高度，不仅要求网络能够保障高效的连通性，更要求网络能够理解和适应上层承载业务的需求，更加灵活地提供对服务器虚拟化、资源迁移、用户自定义空间等需求的支持，可以说是一个全新的网络范畴。新技术的不断涌现，各种 SDN、NFV/NV、Trill、Vxlan、Nvgre 之类的专业词汇，让网络工

工程师有点不知所措。

我们可以看见目前 OpenStack 已经成为一个既定的开源云计算标准，拥有非常多的硬件厂商、软件平台开发商、个人开发者的支持，已经可以实现将计算资源和存储资源虚拟化之后，作为统一的资源池进行分配和管理，也提出了将网络虚拟化，通过 SDN 进行统一管理的需求。如何让 OpenStack 更加灵活，网络是一个不可或缺的一环。

这本书是一本非常优秀的计算机网络普及读物，从计算机网络的初期发展一直到新的云计算数据中心的新生态环境，以及 OpenStack 下各种网络配置和实现方式，都给予了非常深入的讲解，无论是想从头开始学习网络技术的小白，还是已经有一定基础的网络工程师，都可以从中找到通往云计算数据中心网络的快速路径。

北京寄云鼎城科技有限公司（寄云科技）

总经理 时培昕（新浪微博@时培昕）

我在运维 SDN 技术社区 SDNAP.com 网站的时候与小武认识，小武在传统网络通信到虚拟云计算的网络开发都有着丰富的实战经验。结合当下最火的 OpenStack 技术与 SDN 技术，小武明显找到了一条快速提升自己职业价值的道路。在 SDNAP 的 QQ 群，有不少传统网络通信毕业的同学们在问，与 SDN 相关的就业岗位太少，这一点当下是事实。遇到这种同学，一般我都建议他们往 OpenStack+SDN 的方向研究，要知道在 SDNAP，2000 人的 QQ 群大概有 1/3 是搞 OpenStack 的，这么多人关注这块，说明这块的职场缺口挺大，当然要求也高。

这本书有利于做云计算开发的，尤其是做 OpenStack 开发的了解基本的网络知识；同时有利于做传统网络通信的同学了解基本的 SDN 及云平台 OpenStack 的网络基本情况，因此我推荐给大家！

SDNAP.com 吴应辉（新浪微博@SDNAP，<http://www.sdnap.com/>）

在云计算和大数据浪潮的当下，越来越多的传统被颠覆。在基础网络领域，SDN 技术带来的变革打破了封闭的格局，传统基础架构正在由此转向更开放、更灵活高效、高度可编程和高度弹性的网络基础设施。海云捷迅（AWCloud）是一家致力于为中国企业提供基于 OpenStack 开源云服务与解决方案的创业公司，本人与这本书的作者有过多次深入的交流，小武根据自身丰富的从业经验，为读者展现了云计算网络里的世界，既涵盖了传统以太网技术，也详细分析了 SDN、网络虚拟化等新兴技术，书中亦不乏作者的真知灼见，值得推荐。

AWCloud 系统架构师 马力

听说云计算有些年头了，我真正开始接触是从 2011 年初开始的，在公司战略项目的推动下，风风火火地投入到了云计算的虚拟化世界，慢慢地明白了，云计算真的不是炒作，而是实实在在地落地生根了，很多领域都急需这样的技术去支撑。但是很不幸，云计算不是一项孤立存在的技术，而是一个庞大的系统工程。那时候计算虚拟化、存储虚拟化都有相对比较成熟的开源项目支持，但云计算离不开网络啊，网络虚拟化连实验室都没有走出去，更何况生产环境呢！从此，网络也不再是孤立的传统网络了，从业人员不能只单纯地面对网络盒子设备了，还有很多系统协议栈相关的东西需要去接触。那么面对这样一个极具挑战的新领域，从哪里获取资料呢，可参考的文档真的很少，网上搜到的又不具有系统性，实在不行就得去看开源项目的代码实现了……

经过这么几年的苦逼发展，现在网络虚拟化也有了相对成熟的方案，也开始落地上线，但和传统网络的成熟度相比，还差得很遥远，网络从业人员任重道远啊！

今天我看到俊武的《云计算网络珠玑》一书心头一热啊，该书从理论基础到实际开发详细地阐述了这几年网工们是怎么探索云计算的虚拟化之路的，是怎么

从深不可测的网络盒子走到服务器操作系统的……

新浪网技术（中国）有限公司基础架构部网络  
资深架构师司迎春（新浪微博@司迎春-Winters）

我们知道从 SDN 诞生至今已经有 5 个年头了，在学术界和产业界引发的轰动有目共睹，没有人质疑 SDN 作为下一代网络体系结构的地位，作为这个领域工程师的我，值得庆幸处在这样一个时代。

但是经常能看到传统网络工程师的困惑，云计算和 SDN 来了，该怎么办？一方面是在继续学习的道路上到底应该怎样抉择？另一方面是 SDN 给程序员打开了一扇进军网络领域的门，对于他们来说，又需要补充哪些网络基础知识？这本书的内容给我们提供了很好的指引，相信能解答心中的困惑。

《零存整取 NetFPGA 开发指南》作者，SDN 初创公司  
南京叠锏联合创始人杨泽卫（新浪微博@杨泽卫-MeshSr）

软件定义网络（SDN）概念从 2006 年被提出以来，常会处于高处不胜寒的处境，只是学术界和少数网络厂商的玩物，直到 2011 年，OpenStack 推出网络管理平台 Neutron 并推出了基于 Linux 的参考实现之后，很多公司和工程师才恍然大悟，原来 SDN 是这么回事。小武同学的《云计算网络珠玑》一书详细介绍了 SDN 网络所依赖的一些基本技术和原理，并以 OpenStack Neutron 为例，深入浅出地介绍了如何用 Linux 网络技术实现一个大规模的 SDN 网络，值得网络初学者和从业人员一读。

UnitedStack 创始人&CEO 程辉（新浪微博@程辉）

我经常在博客（<http://www.chenshake.com/>）上发表关于对 Openstack 的最新进展的文章，读者问及最多的问题还是网络问题，比如用几个网卡可以搭建

Openstack 部署环境，网络节点和其他节点能否共用一台设备，还有些网络方面的排错问题。这些问题大多很基础，但是由于网络是一个专业性和实践性非常强的技术领域，很难从小白入手快速提升成专家。而俊武的这本书，从基本的 TCP/IP 以太网原理开始，用其特有的理解思路，讲述了 SDN/NFV、Neutron 等相关知识以及它们之间的内在关联，并且对很多高级话题的讨论俊武也发表了自己的观点，同样也令该领域的高手们深思。想学习网络的初学者和负责 Neutron 的研发者，建议如果有问题可以先参阅一下这本书。

陈沙克（新浪微博@陈沙克）

# 前 言

## 写作缘由

记得刚毕业那年，找工作的个人简历上写着“精通计算机网络”，而实际上连 IP 报文头部有哪些字段都说不上来，更别说能回答二层转发和三层路由的细节区别等问题。后来虽然有幸进入网络领域从事研发工作，但是总感觉一直对网络还是没有太深入的理解。最开始从事的工作是交换机驱动开发，一年前工作开始转向了云计算网络领域。在这近五年的时间里，我本着学东西就要弄清楚其中所有原理的原则，逐渐积累起自己对网络技术的系统性认识。随着工作经验的积累，参与了业界的很多交流活动，在各种机缘下有幸认识了很多网络领域的专家和大拿，让我对网络的理解更提升了一个层次，尤其是通过 SDNAP 的 QQ 群认识的“网友们”。

在交流的过程中，结合自身的不足也发现了一些问题，比如计算机网络领域是极其博大精深的，有着大量的规范和协议，尤其是现在网络虚拟化、SDN/NFV 技术的出现，更是让很多初学者摸不清头绪，无法抓住理解的要领；还有很多公司的系统工程师肩负着网络工程师的工作，在这种情况下若不深入理解网络设备（比如交换机）底层的工作机制，其设计的网络架构将存在很大的潜在风险。我本人也一直在用“Linux Bridge 与二层交换芯片处理报文的流程上有何不同”这

个问题，来区分应聘者的背景是系统工程师还是网络工程师。如何快速学习网络知识，如何快速在网络领域提升技术能力，面对众多新兴的网络技术如 Vxlan、LISP 等该如何快速理解并应用？面对如此多的问题，于是我萌生了一个念头，何不将自己对传统以太网、网络虚拟化、SDN/NFV 的相关技术的积累和理解进行汇总以串联起来？既方便自己查阅，也能为网络技术的传播尽自己的绵薄之力。

最开始想写这本书的时候是我参加工作的第三年初，写了一段便感觉自己没有足够的积累让自己在写作中选择好一条写作的主线。直到开始从事云计算网络研发，心中萌生的写书念头再次袭来，而这次想法也更加清晰，于是便开始了整理材料和规整思路。因为还要做好公司的工作任务，所以只能每天早晨上班前和晚上下班回家后写作稿件，最终历经三个多月的时间，终于成稿。

## 本书内容

本书内容涉及的网络技术点比较广泛，虽然主要谈及的还是以太网技术，但仍然无法将所有相关技术列举完全；并且根据云计算网络的主线，技术点阐述也有详有略。而网络技术发展也极为迅速，书稿写作时其中讨论的问题，可能在书稿出版时已经有了新技术对其进行了很好的解决。如果将来有精力希望可以对其中的内容进行补充完善，必然会再结合自己的新理解，融合到本书的第二版里，读者也可随时在我的博客上查阅相关更新。希望本书能抛砖引玉，为读者在自己的网络之路上的探索起到一定的帮助作用。

信息技术加快了人类社会的信息化发展进程，互联网尤其是移动互联网的出现为人类分享和获取信息提供了新的方式；网络技术的发展历程中，出现了很多

技术标准或产品方案，尤其是以太网技术和 TCP/IP 协议的出现，以及以太网交换机和路由器等网络设备商的出现，使得网络设备开始标准化大批量生产使用，从而让信息技术的服务受众变得平民化。信息社会的需求不断变化，需要网络技术不断创新，云计算中的网络虚拟化很快成为网络技术的焦点，而 SDN/NFV 技术的出现更是在网络界掀起了一场技术革命。

本书共分为 6 章，各章的主要内容如下。

第 1 章主要介绍网络基础知识。既有传统的 MAC、VLAN、ARP、TCP/IP、路由协议、NAT、MPLS、QoS、CDN、安全监控等技术的介绍，也有 Mac-in-Mac、Trill、DPI、LISP 等新兴技术的讲解，还有负载均衡和数据中心扁平化二层等方面相关的架构知识。让读者对后续网络技术的学习有一个良好的基础。

第 2 章综合了商业交换芯片的网络转发流程的内容，分别详细介绍了交换芯片端口处理、二层转发、三层转发、ACL/QoS、虚拟化和交换机的 CPU，让没有接触过交换机研发的读者对神秘的交换机黑盒子在逻辑上有一个深入理解。

第 3 章则选择当今比较流行的服务器操作系统 Linux 中的 TCP/IP 协议栈来介绍协议栈的功能技术点，还对网络虚拟化常用的 Linux Bridge、TUN/TAP、Iptables、DPDK 和 Dnsmasq 进行了介绍和归纳总结。

第 4 章主要集中于 SDN 和 Openflow 及相关衍生的一些技术，包括 SDN 的部分控制器和 OpenVswitch 等，让读者体会一览新技术的快乐。

第 5 章是本书的重点，在前面介绍网络基础知识的基础上，以 Openstack 中的 Neutron 为例详细阐述了云计算网络虚拟化的功能点和底层实现，并以 VXLAN 隔离环境为例分析 Neutron 底层通信流程和常见问题的解决方法。另外，本章还

关注以前的 Nova-network 技术、热门的 ML2 插件和新的 J 版的 DVR 技术，并进行了一定深度的介绍和探讨。

第 6 章则是对 Openstack 中云计算网络技术 Neutron 的一些高级话题的讨论，包括 SDN 技术的结合硬件设备研发来提高网络性能、云计算的商业模式，以及对云计算的发展个人思考等。

## 本书读者

本书面向的读者对象面较广，可以是没有任何网络基础、但想学习网络技术的零基础人员，也可以是有网络基础知识且想从事云计算网络研发的转型开发者，还可以是想深层理解网络底层细节的云计算网络上层开发者。另外，本书也为具备网络基础知识和云计算网络技能的网络技术专家准备了 SDN/NFV 和云计算网络若干问题的研究讨论。

## 致谢

本书在创作过程中得到了许多朋友的鼓励和帮助，包括高飞、张建勋、周博洋、马力和贾豪杰等人的技术协助，SDNAP 社区负责人吴应辉的推荐，以及为本书写推荐的各位专家的认可，在此俊武表示衷心的感谢！

另外非常感谢电子工业出版社的董英编辑，没有她的辛勤付出，可以说本书很难能够出版！并且在审阅过程中，她能够“容忍”我多次在提交稿件后依然继

续修改，导致已经编排整齐的文档又被我弄乱。

最后感谢我所有的家人们，包括我的父母和岳父母等，尤其是我的妻子王青，在本书的几个月写作过程中，没有足够的时间来陪伴你；如果没有妻子一直的鼓励和默默的支持，可以说我不可能有足够的耐心和毅力来完成本书，谨以此来表示对你永远的爱！

因笔者水平有限，本书难免存在错漏，恳请读者批评、指正。

李俊武

2014年圣诞夜于北京

十载耕耘 奠定专业地位

## 博文视点诚邀精锐作者加盟

以书为证 彰显卓越品质

《C++Primer (中文版) (第5版)》、《淘宝技术这十年》、《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、管辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

十年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金之计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于IT专业出版之巔。

### 英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

### 专业的作者服务

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

**善待作者**——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

**尊重作者**——我们尊重每一位作者的技术实力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

**提升作者**——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



### 联系我们

博文视点官网: <http://www.broadview.com.cn>

投稿电话: 010-51260888 88254368

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

投稿邮箱: [jsj@phei.com.cn](mailto:jsj@phei.com.cn)



@博文视点Broadview



微信公众账号 博文视点Broadview



### 关于本书用纸的温馨提示

亲爱的读者朋友: 您所拿到的这本书使用的是**环保轻型纸!**

环保轻型纸在制造过程中添加化学漂白剂较少,颜色更接近于自然状态,具有纸质轻柔、光反射率低、保护读者视力等优点,其成本略高于胶版纸。给您带来更好的阅读体验并与读者共同支持环保,我们在没有提高图书定价的前提下,使用这种纸张。愿我们共同分享纸质图书的阅读乐趣!

# 博文视点精品图书展台

## 专业典藏



## 移动开发



## 大数据·云计算·物联网



## 数据库



## Web 开发



## 程序设计



## 软件工程



## 办公精品



## 网络营销



# 博文视点 2014 年第一季最新最热图书



## 《淘宝技术这十年》

子柳 著  
定价: 45.00元  
 ○ 淘宝技术大学校长辛辣揭秘  
 ○ 世界最大电商平台、超大型网站,首次全面曝光技术内幕  
 ○ 技术变迁熠熠生辉、产品演进饱含智慧、牛人生涯叱咤风云、圈内趣事令人捧腹



## 《Windows内核原理与实现》

潘爱民 著  
定价: 99.00元  
 第一本用真实的源代码剖析 Windows 操作系统核心原理的原创著作!



## 《软件需求最佳实践——SERU过程框架原理与应用(典藏版)》

徐锋 著  
定价: 69.00元  
 “用户说不清需求”、“需求变更频繁”……,都是在软件需求实践中频繁遇到的问题;本书首先直面这些问题,从心理学、社会学的角度剖析其背后的深层原因,使大家从中获得突破的方法。



## 大数据丛书 《大数据挑战与NoSQL数据库技术》

陆嘉恒 编著  
定价: 79.00元  
 大数据技术的学习指南,突破迷局,厘清思路,拥抱变化。



## 《高性能MySQL(第3版)》

【美】施瓦茨(Schwartz,B.) 【美】扎伊采夫(Zaitsev,P.) 【美】特卡琴科(Tkachenko,V.) 著  
 宁海元 周振兴 彭立勋等 译  
 定价: 128.00元

本书是 MySQL 领域的经典之作,拥有广泛的影响力。第 3 版更新了大量的内容,不但涵盖了最新 MySQL 5.5 版本的新特性,也讲述了关于固态硬盘、高可扩展性设计和云计算环境下的数据库相关的新内容,原有的基准测试和性能优化部分也做了大量的扩展和补充。



## 《收获,不止Oracle》

梁敬彬 梁敬弘 著  
定价: 59.00元  
 颠覆 IT 技术图书的传统写作方式,在妙趣横生的故事中学到 Oracle 核心知识与优化方法论,让你摆脱技术束缚,超越技术。



## 《Clojure编程》

【美】Chas Emerick (蔡司 埃默里克), Brian Carper (布赖恩 卡珀), Christophe Grand (克里斯托弗 格兰德) 著  
 徐明明 杨寿勋 译  
 定价: 99.00元

第一本完整讲述 Clojure 的权威著作。



## 百度认证系列丛书 《百度推广——搜索营销新视角》

百度营销研究院 著  
定价: 59.00元  
 百度营销研究院资深专家团队撰写,百度认证初级教程!



## 《我看电商》

黄若 著  
定价: 39.00元  
 作者近三十年从事零售及电子商务管理的总结和分享。



## 《做自己——鬼脚七自媒体第一季》

鬼脚七 著  
定价: 77.00元  
 本书是鬼脚七自媒体的原创文集,是电商圈第 1 本自媒体著作!书中有关于生活、互联网、自媒体的睿智分享,也有关于淘宝、搜索的独到见解,文章非常耐读,也容易引发读者的思考,是一本接地气、文艺范、充满正能量的电商生活书!

### 欢迎投稿:

投稿邮箱: [jsj@phei.com.cn](mailto:jsj@phei.com.cn)  
[editor@broadview.com.cn](mailto:editor@broadview.com.cn)  
 读者信箱: [market@broadview.com.cn](mailto:market@broadview.com.cn)  
 电话: 010-51260888

### 更多信息请关注:

博文视点官方网站:  
<http://www.broadview.com.cn>  
 博文视点官方微博:  
<http://t.sina.com.cn/broadviewbj>

# 目 录

## 第 1 部分 网络基本原理

第 1 章 TCP/IP 网络技术.....	3
1.1 信息网络.....	6
1.2 以太网技术.....	12
1.3 网络传输设备.....	15
1.4 MAC 和 VLAN.....	18
1.5 MAC-in-MAC.....	22
1.6 STP 和 Trill.....	25
1.7 IP 技术.....	29
1.7.1 IP 地址.....	30
1.7.2 IP 报文格式简介.....	32
1.7.3 TCP 和 UDP.....	34
1.7.4 TCP 与 UDP 检验和.....	39
1.8 DNS 和 DHCP.....	39
1.9 ICMP 报文.....	42
1.10 ARP 和 RARP.....	43
1.11 路由协议.....	46
1.11.1 RIP 和 BGP.....	47
1.11.2 OSPF 协议.....	49
1.12 NAT 技术.....	56
1.13 隧道技术.....	58
1.14 MPLS 和 VPLS.....	66
1.15 QoS 功能.....	69

1.16	网络安全和监控	73
1.17	LB、CDN 和 DPI	77
1.18	LISP 和 LLDP	80
1.19	网络架构	82
第 2 章	以太网交换机	86
2.1	交换机转发流程	87
2.2	交换机端口处理	90
2.3	交换机二层转发	93
2.4	交换机三层转发	100
2.5	交换机 ACL 和 QoS	102
2.5.1	ACL 功能	102
2.5.2	QoS 功能	104
2.6	交换机的虚拟化支持	113
2.7	交换机的 CPU	117
第 3 章	Linux 网络基础	120
3.1	网卡和数据包的收发	121
3.1.1	Linux 网卡收包流程	122
3.1.2	多网卡 Bonding	126
3.1.3	SR-IOV	128
3.1.4	DPDK	129
3.2	TUN/TAP	130
3.3	Linux Bridge 和 VLAN	131
3.4	TCP/IP 协议栈	135
3.5	IPtables	138
3.6	QoS 模块	139
3.7	Dnsmasq	141
第 4 章	SDN 网络架构	143
4.1	什么是 SDN	144
4.2	OpenFlow 与 Open vSwitch	149
4.2.1	OpenFlow 简介	149

4.2.2	Open vSwitch 简介	158
4.3	能为 SDN 做什么	160
<b>第 2 部分 云计算及 OpenStack 的网络</b>		
第 5 章	OpenStack 的网络	167
5.1	云计算及 OpenStack	168
5.2	OpenStack 的网络介绍	173
5.2.1	Nova-Network	175
5.2.2	Neutron 网络	179
5.2.3	OpenStack 存储网络	186
5.3	Neutron 底层网络原理	187
5.3.1	Neutron 组件的构成	189
5.3.2	Neutron 网络的隔离	192
5.3.3	Neutron 网络的互通	196
5.4	Neutron 主要功能	200
5.4.1	互通与隔离功能	201
5.4.2	防火墙与安全组	203
5.4.3	LBaaS 和 VPNaas	204
5.4.4	监控安全和数据中心互联	206
5.4.5	Neutron 中的 QoS 功能	207
5.4.6	Neutron 部署运维	208
5.5	VXLAN 隔离环境通信实例详解	209
5.5.1	VXLAN 报文解析	210
5.5.2	VXLAN 通信流程	214
5.6	Neutron 网络高级话题讨论	219
5.6.1	常见 Neutron 网络问题	220
5.6.2	Neutron 网络性能	225
5.6.3	Neutron 网络稳定性	229
5.6.4	Neutron 在折翼	230
第 6 章	Neutron 网络发展趋势	234
6.1	SDN 的结合	235
6.2	硬件网络设备解决性能问题	236



6.3 安全和监控 .....	238
6.4 虚拟网络中的路由协议 .....	239
6.5 IaaS 上的商业模式 .....	240
6.6 云计算时代的终结 .....	242
附录 .....	245
附录 A Open vSwitch 基本命令 .....	245
附录 B 深入理解 OpenStack 云计算 VLANManager 网络流的六种场景 ..	247
附录 C RDO 配置文件网络部分——VLAN 隔离 .....	262
附录 D VXLAN 通信抓包实例 .....	264
参考文献 .....	274

# 第 1 部分

## 网络基本原理

随着社会信息化程度的不断提高，人们对信息的计算、存储和传输能力的新需求不断被提出，于是需要有相应的方案或产品来满足这些需求。为提高 CPU 的利用率和系统易维护性的云计算是目前比较热门的一个变革性方案，其内容涵盖了计算、存储和网络等技术知识，而在云计算研发中，网络方面的相关工作量保守估计占全部工作量的一半以上，因为三者中的网络技术是最灵活的，而且底层是多变的，所以合理可靠的网络方案便成为云计算领域能否提供可实施产品的关键因素。

虽然云计算中的网络技术有其特殊思维方式和应用场景，但其基本原理与传统的物理网络没有太大区别，只需要在物理网络的基础上提供虚拟网络，并在此条件下对相应应用逻辑有一个新的理解方式。所以，在讨论云计算网络之前，需要先介绍一下相关的基础网络概念和基本原理，正所谓“磨刀不误砍柴工”。

网络技术的发展始于传输文字和语音的电话电报技术，后来出现了手机和无绳电话等无线通信产品，进一步方便了人们信息的传递。此类终端产品的需求导致了一大批电话和手机厂商的诞生与发展，比如摩托罗拉、诺基亚、西门子等。随着时代变迁，仅仅是语音和文字已经无法满足人们对信息传递和处理的需求，

因为越来越多的图像、多媒体等信息形式逐渐成为主流信息载体，尤其是个人计算机的普及和手持移动终端的大量使用，导致对信息处理设备相关的处理器计算速度、网络性能和存储大小都有进一步的要求。这些信息处理终端的大规模出现对网络技术的需求主要体现在传输性能（高带宽、低延迟）上，促进了网络设备（比如交换机和路由器等）的大批量生产，随之出现了思科、华为、Juniper、华三通信等一系列通信巨头，现在移动互联网的出现更体现了对网络的传输带宽和传输延迟等方面的高性能有更迫切的需求。新一代的科技公司（如苹果、小米等）在移动终端上的成功也体现了未来对网络的业务需要的发展趋势。这些都说明现在人们已经不满足于被动地获取信息，而是开始通过使用互联网这种工具，让用户主动检索自己所需的信息。正是互联网的出现，让信息技术改变了人们的学习、工作和生活的方式，也极大地促进了社会进步和生产效率的提高。

以太网（IEEE 802.3）是现在比较通用的局域网网络技术，大带宽、低延迟的以太网交换芯片的出现更是促进了网络的发展。越来越多的信息用以太网进行传输；所以人们平常很多时候所说的网络是指局域网，现在实际场景中讨论的局域网基本上是指在以太网上运行的 TCP/IP 网络，包括云计算中的虚拟网络也基本如此。虽然针对云计算网络的场景提出了一些新技术或新方案来解决其中遇到的某些技术问题，但是，掌握和理解以太网基本的原理及 TCP/IP 的知识仍是深入理解云计算网络的基础。

## 第 1 章

---

# TCP/IP 网络技术

笔者喜欢阅读历史类的书籍，尤其是客观描写的历史，因为它不仅反映了一些重大事件的决策过程，更能让后来者以史为鉴，减小重蹈覆辙的概率。吴军先生所著的《浪潮之巅》汇总了美国硅谷信息技术方面很多曾经有辉煌历史的公司的创办、发展、兴起和部分衰落的过程，国内知名 IT 公司的发展是否也在遵循类似的历史轨迹呢？建议各位可以读一读这本书。周爱民先生的《大道至简》则从很多项目经验和教训的历史中进行总结，从而详细阐述了一个全新的视角——把一个项目的软件工程的体系模型和各步骤之间的内在关系展现在读者面前；同类型的书籍《人月神话》则是从一个浩大的项目中（虽然该项目失败了，但 UNIX 操作系统却从失败的项目中涅槃而生）总结了大量的数据和教训，并推导出了数字化表述和规划 IT 工程的施工进度计算的软件工程管理量化方法。该书作者 Brooks 在 IBM 公司任 System/360 计算机系列及其庞大的软件系统 OS/360 的项目经理，经验非常丰富，但其中也不乏各种血泪历史所积累的数据。学习网络技术也一样，需要不断借鉴以前的技术发展史，“站在巨人肩上，才能看得更远”，对产品设计所用的技术能有较好的把控。

网络技术是从美欧等发达国家起源和发展的，直到现在，国内的很多技术与

欧美相比还有不少差距；在技术的发展脉络探讨和研究上，每个人的精力有限，一方面，每个人无法亲身经历网络技术发展历史中所有事件的发生和结束，另一方面，也无法接触到相应详细的历史材料或各种会议的内容记录，所以，大多数时候笔者无法也不太可能掌握所有网络技术发展过程中的细节，希望将来有人能整理一些相对完整的文章或书籍，让大家对网络的历史尤其是重大事件的过程能有一个清晰的了解，这样在今后的网络技术学习中才能对未来技术的趋势有一个较好的把握。

言归正传。网络技术的产生和发展基于数学、概率论、随机过程、矩阵论、物理学、光学和计算机科学等各门学科的前沿技术；网络技术的发展才几十年，相对于人类的文明历史是非常短暂的，但是其发展速度及对人类文明历史的推动和贡献却有着不可估量的影响，可以说在当今信息技术的推动下，人类社会进步的速度超乎人们自身的发展计划和技术想象。虽然网络技术的标准相对比较完善和统一，且不同厂家的网络设备的兼容性和互通性都是研发厂家产品测试的一项主要内容，但是网络技术的标准制定受到各种技术条件的约束和历史时期的限制，具体如下。

第一，标准在被制定时，很多设计方案并没有考虑后来网络规模的扩大和复杂业务的发展，导致标准的制定或方案的设计存在局限性，很多解决网络难题的技术和方案在网络设备数量扩大后不一定能继续适用，比如，VLAN的个数、IPv4地址的长度、STP、交换机单端口带宽等技术，在物理设备数量规模扩大超过一定范围后，就无法再满足使用需求，这对以后的网络发展造成了进一步的技术限制，犹如网络研发人员自己挖了个陷阱然后再跳进去。通常情况下会有两种方式解决这个问题：一是在原有的基础上进行改良，二是创建新的标准进行替代。两种方式没有好坏之别，只有合适或不合适的新方案之分，其内容具体如下。

- 前一种方式可以被称为革新方式，这种方式下各个协议或标准会有不同的版本，后续版本就是为了纠正以前版本的缺陷或细化明确了以前版本的某些规定等。再如，IGMP 协议有三个版本，OSPF 针对 IPv4 的两个版本，以及针对 IPv6 的第三个版本。
- 后一种方式可以被称为革命方式，即彻底推翻以前解决问题的方法后采用新的方案。比如，虚拟扩展本地局域网（Virtual eXtensible Local Area Network, VXLAN）代替 VLAN、IPv6 代替 IPv4、多链路透明互联（Transparent Interconnection of Lots of Links, Trill, RFC6326）代替 STP 和交换机单端口带宽的标准相继提出来取代原来的方案等。

第二，很多计算机网络方面的技术标准是从实践中产生的，即可能是先有技术和产品之后才形成标准，这样，当多家厂商对同一问题或技术点都有了各自不同的技术和产品后，在采纳哪一家的产品或方案作为标准上就会产生冲突。这种情况下产生的技术标准通常是各商家利益博弈的结果，而未必是技术上最优的产品和技术方案。

第三，一个网络标准需要考虑整个功能内容上的完善性，而这个要求在大多数标准的制定过程中是无法满足的，因为人类对科学技术的认识和掌握处于一个相对不完善的阶段，而且这个不断补充完善的过程是无止境的，当然网络技术的发展也是这样。在这种认知不完善的情况下制定出的解决方案大多数是不太完美的方案，需要研发人员后续在实践中不断地提高认知，纠正方案的不足或者提出新的方案来替代原来的标准。另外，这种认知不完善的情况同样体现在当今的网络技术人员身上，比如因为相关研发人员对网络技术的认知不在同一个高度或不在同一个角度上，而导致的问题处理意见不一致、产品方案不统一等现象，是导致内部冲突和争执的一个很重要的原因。放眼现在，软件自定义网络/网络功能虚

拟化（Software Defined Network/Network Function Virtualization, SDN/NFV）技术则是要对网络架构进行革命，这个过程想必不是一个平静的过渡，而必然是利益相关商家及商家内部不同技术观点人员的一场血雨腥风的技术之争。

---

## 1.1 信息网络

自从在地球上出现人类以来，他们就无时无刻不在和信息打交道，尤其是进入近代以来，新的信息交流工具大大方便了信息的传递，比如电话电报，特别是随着计算机网络的出现，更是为信息的处理和传播提供了强大的媒介工具。21 世纪的社会是信息社会，信息社会通过信息的处理和传递为社会创造了价值，提高了社会生产力，反过来又进一步提高了对信息技术的要求，并促进了信息技术的发展。无论是电报电话通信还是计算机网络通信，当通信规模足够大的时候，除了需要线缆连接通信设备外，还需要各种通信中继的网络设备来互联，一个作用是对承载信息的信号功率进行放大，以防止衰减导致的无法正确识别，另一个作用就是能够对信息帧的转发处理进行集中控制，以减少随着通信终端的增加而引入的运营维护工作。

通信领域技术不断发展，如：通信连接的方式最初是有线方式，现在是无线接入的大规模使用；通信的物理信号最初是模拟通信的调制和解调，后来出现数字通信方式下的数字信号处理技术的发展和應用；最开始传输的信息格式是文本，后来是语音、图像和视频，现代则是集文字、语音、图像、视频等为一体的多媒体通信方式。目前，数据通信的各种信息主要有三种交换方式：电路交换、分组交换和报文交换，有些通信系统也可能需要三种模式中的两种或三种同时存在的混合交换。

笔者认为计算机的计算、通信与存储是属于信息相关的三个学科，计算机计

算是在进行信息的处理和展示，通信网络则进行信息的传递，存储是将载有的信息数据存放到合适的物理介质里，进行保存以增加信息的价值生命周期；大规模的分布式计算或现在盛行的云计算技术极大地增强了信息处理设备的计算能力，以挖掘和处理大数据里所蕴含的信息，这些数据和处理后得到的信息不仅需要拥有大存储量的存储介质，还需要有传递迅速和保障安全的传输网络；这些需求在大型互联网公司如 Google 和 Facebook 等的数据中心里有着尤为明显的体现，也正在被逐步解决。通常，通信专业和计算机网络专业在网络四层划分中处于不同的层次——通信主要关心比特级别的处理，与具体传输介质打交道比较多，无论是网线和网卡，或者是交换芯片硬件或 FPGA，还是无线通信中的 3G 和 LTE，属于通信人的工作；而计算机网络则关心 Packet 级别的通信及系统级的处理实现，是采用以太网还是 PPP 标准，无论是 TCP 还是 UDP 或者是 ICMP 等，都是为了通信而传输报文，无须关心具体用什么硬件转化成多少比特进行传输，这些则属于典型的计算机网络方面的工作。这种区分的优势在于每层工作者都能透明处理信息，且专注于自身负责的工作。

到底什么是网络呢？网络的定义纷繁复杂，不同的人有自身不同的理解和认识。笔者理解的网络是设备上不同进程间的一种通信方式，包括在服务器运行的收发数据的进程（可以属于一个物理或虚拟设备，也可以分别属于不同的物理或虚拟设备）、网络设备和线缆等，其中的网络设备和线缆也包括物理的和虚拟的两种形式。进程间通信的方式有很多种，大多数情况属于同一个设备之间的通信，而跨越不同的设备进行通信时，必须通过网络。当然，同一个物理设备或者虚拟设备的两个进程也可以用网络方式进行通信。为了便于不同厂商的网络设备互联互通，需要为相互通信的计算机网络设备制定相关的通信标准，一方面，可以做到不同厂商设备的互通，另一方面，可以让学术界和产业界的力量集中到一起进

行更深入该方向新技术的研发工作。

国际标准化组织 (International Organization for Standardization, ISO) 为通信网络制定了 OSI (Open System Interconnection), 即开放式系统互联参考模型, 它把网络协议从逻辑上分为七层, 即应用层 (Application Layer)、表示层 (Presentation Layer)、会话层 (Session Layer)、传输层 (Transport Layer)、网络层 (Network Layer)、数据链路层 (Datalink Layer) 和物理层 (Physical Layer), 主要目的是解决异种网络互联时所遇到的兼容性问题, 其最主要的功能就是帮助不同类型的主机实现数据传输。实际产业界所用的网络技术并没有遵循学术界的这个标准化建议, 而是采用了更实用的传输控制协议/因特网互联协议 (Transmission Control Protocol/Internet Protocol, TCP/IP), 又名网络通信协议, 四层模型: 应用层 (Application Layer)、传输层 (Transport Layer)、网络互联层 (Internet Layer) 和网络接口层 (Network Access Layer)。其中, 网络接口层的作用主要是封装和解封装 IP 报文、发送和接收地址转换协议 (Address Resolution Protocol, ARP) 及反向地址转换协议 (Reverse ARP, RARP) 报文等, 这一层随着底层类型的不同而不同; 网络互联层的作用主要是通过寻找合适的网络接口或路径转发, 处理网络报文分组, 完成将报文分发到目的网络或目的主机等; 传输层则是对报文进行分组、重组和格式化信息流, 主要包括传输控制协议 (Transmission Control Protocol, TCP) 和用户数据报协议 (User Datagram Protocol, UDP); 应用层则是向用户提供一组应用程序, 包括文件传输协议 (File Transfer Protocol, FTP)、域名解析服务、DNS (Domain Name Service)、Telnet、简单邮件传输协议 (Simple Mail Transfer Protocol, SMTP)、网络文件系统 (Network File System, NFS)、超文本传输协议 (Hypertext Transfer Protocol, HTTP) 等。

但是现在常见的每种技术标准都是当初从很多技术中竞争胜出的，而这种胜出有时不仅仅是技术的优劣，更重要的是由技术标准的推出者所占的市场份额决定的；而网络界和其他领域不太相同的一点是：学术界的研究内容往往不能直接用于工业界，因为网络中的很多学术研究是基于纯理论的，而网络是一门实践性很强的技术，倒是很多企业里的技术或标准成为了学术界的规范，比如，思科等设备商的很多网络技术在做出良好的产品后会被提交申请为相关的标准。再如，七层模型和四层模型，起初国际标准化组织规定了七层网络模型，但是在企业界生产研发时不太实用，所以企业界通常都用 TCP/IP 四层模型，相信现实中现在用得最多的也是四层模型，而七层模型较多地出现在文献资料和求职者的笔试面试题中，两者的对比如图 1-1 所示。这种工业界指导学术界的现象在 IT 界是比较普遍的，因为 IT 技术是一门实践性比较强的技术，任何标准或规范都需要在大量实际检验的基础上才能制定完成，并且现在学术界和工业界在 IT 领域相比以前有了更大的融合度，尤其是在 IT 领域比较领先的美国，经常会有知名教授创业的信息公司。

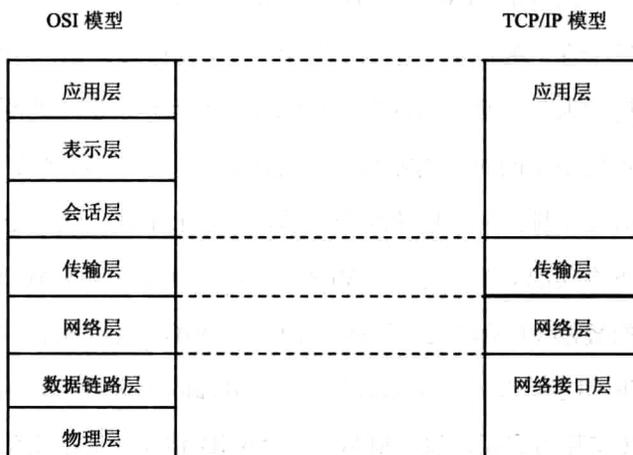


图 1-1 OSI 七层与 TCP/IP 四层的对应关系

目前常见的局域网类型包括以太网（Ethernet）、光纤分布式数据接口（FDDI）、异步传输模式（ATM）、令牌环网（令牌 Ring）等；无线局域网（Wireless Local Area Network, WLAN）包括直接序列扩频（ISM 2.4GHz, 802.11b）、802.11a（5GHz）、正交频分复用 OFDM 调制技术（ISM 2.4GHz, 802.11g）和 802.11z 等标准。后来以太网逐渐成为主流局域网技术，为了加大传输距离出现了各种中继器，主要是对信息的信号进行放大以继续后面的传输。随着以太网通信终端设备的增加，通信终端之间的连线工作也变得繁杂、不可靠且难以维护。以网状型的网络为例进行说明，原来有  $N$  台设备，每增加一台机器（要含  $N$  个网卡），就要新增  $N$  条网线和  $N$  个网卡（其他机器也要增加一块网卡，那个时候网卡即插即用吗？要硬件预留很多网卡插槽，还是换服务器？），其中有些是技术问题，有些是成本问题。所以 Hub 就应运而生，对计算机而言，都只有一块网卡，需要网络的计算机全都用一根网线连到一台 Hub 的一个端口上，给每台计算机分配一个硬件地址称为 MAC（Media Access Control）地址，这样一台计算机发送信息时加一个 MAC 头，便包含两个 MAC 地址：计算机自身的源 MAC 和发送此信息所需到达设备的目的 MAC，即 Destination MAC 和 Source MAC；Hub 的工作方式就像大家坐在一起围着一张桌子开会一样，无论谁说话，其他人都听得见，但只有相关话题的人员才会参与回答，其他人即使听到了也不能回答。这样做带来的问题就是不同小组的讨论内容也会被其他无关小组接收，所以 Hub 的缺陷是广播泛滥且没有安全性。为了解决这个问题，网络上出现了当初令人兴奋而现在云计算多租户下令令人尴尬的技术——VLAN 隔离，就是在 DMAC 和 SMAC 后面再加入 32bits 的数据 VLAN Tag，包括 16bits 的 TPID，其中 3bits 的 Priority 用于二层 QoS，1bits 用于 CFI（Canonical Format Indicator）和 12bits 用于 VLAN ID。基于 MAC 的转发称为二层转发，MAC+VLAN ID 称为二层报文头或以太网头；而 IP 层的转发称为三层转发或路由，IP 头部字段称为三层报文头，而支持路由

的网络设备称之为路由器，后续出现了大吞吐率的三层交换机也可以进行三层转发，而网络设备之间交换 IP 转发路由规则的生成则是通过常见的路由协议或静态路由完成的。这样的网络运转了很长一段时间，起初是运营商（Internet Service Provider, ISP），即互联网服务提供商，为一般用户（包括家庭用户和企业用户）提供网络接入，出现了一段稳定期，因为当时用户的内网规模不是很大，所以一般的网络设备都能实现现有的需求或功能要求，但当互联网公司出现并迅速发展成为大亨、大鳄时，这些大财团发现其为了向互联网客户提供互联业务服务而租用运营商的服务（主要是数据中心和带宽），不仅租用的成本高而且服务也不能满足自身越来越复杂的业务需求，便纷纷开始自建所用的数据中心，国外的如 Google、Facebook 等，国内的主要是 BAT（Baidu、Alibaba、Tencent）等。这些大亨、大鳄凭借雄厚的财力，招揽了大量的人才，根据自己的业务需要，进行了大量的技术尝试创新和方案验证，提出了很多新技术甚至新标准，并且随着需求的不断升级，这些互联网公司发现仅仅是租用空地资源建数据中心的房子，买来通用厂家的设备（服务器和网络设备）进行组装或组建成数据中心已经不能满足自身定制化的需求，便开始自己研发相关的存储、计算和网络等方面的设备及软件产品，所以互联网公司着手参与了根据自身业务定制化需求的服务器及网络设备的硬件和软件自研工作，且已经公布了很多开源的软件代码或者开放的硬件设计，比如国内互联网公司对 Linux 网络协议栈投入大量人力进行开发优化、Facebook 及 Google 则分别自己研发交换机的硬件和软件等，所有这些都无疑极大地促进了网络技术的发展（当然，互联网公司还有很多其他领域的贡献）。

网络技术最初的发展由于当时网络互联的设备是在实验环境下，大家的计算机运行都是可控的、安全的，且数量比较少，也就自然没有网络安全防御方面的需求和相应措施；但是从现在来看，在网络中的每个客户或者大部分使用者是安

全的假设是极其不正确的，甚至和现实是截然相反的，因为现实中无论是主动攻击者还是被控制的肉鸡机器，都表明每台 PC、甚至是每个终端都是很不安全的，也导致了从这样的畸形发展壮大起来的网络会带来后续一系列的安全问题，而为了解决这些安全问题，使得现有的网络陷入了非常复杂的麻烦之中。

---

## 1.2 以太网技术

网络技术的发展最是从通信领域开始的，起源于电报电话等通信，尤其是电话的普及给人类社会的信息传递带来了极大的便利。最开始为了延长通信信号的质量，需要中继器来对衰减的信号进行中继放大，以便进行更远距离的传输。众所周知，最开始发明计算机的时候，真的纯粹是为了数据计算的需要，慢慢才有了广义的计算——数据处理，当然现在已经发展到对文本、图像、视频的综合业务的快速处理。所以 Computer 被翻译成电脑，笔者不认为是可口可乐的那种神译，而是非专业人士引入的另类称呼。计算机一开始进行数据处理基本上也是单机的，随着需求发展出现了需要联机计算的数据，也就是需要计算机网络，这也应该是计算机网络最初的需求模型。

尽管 TCP/IP 的四层协议工作机制比较复杂，但是无论从局域网网络设备还是从服务器的协议栈来说都需要有对应的功能组件来完成相应的网络功能。物理层常见的接口层协议有以太网（Ethernet，IEEE802.3）、令牌总线（Token-Bus，IEEE 802.4）、令牌环（Token Ring，IEEE 802.5）、X.25、Frame relay、HDLC、PPP、ATM 等，这些内容只有进行底层网络的底层开发和设计者才能接触到，通常的网络研发、使用和测试者是无须关注的，大多数情况下仅仅是

查看和配置些参数。在所有的通信网络技术中，局域网中的以太网技术是比较重要的一种类型。以太网（Ethernet）指的是由 Xerox 公司创建并由 Xerox、Intel 和 DEC 公司联合开发的一个基带局域网规范，是当今现有局域网采用的最通用的通信协议标准。在以太网中采用载波监听多路访问及冲突检测（Carrier Sense Multiple Access with Collision Detection, CSMA/CD）技术，支持的速率随着业务的需求从最初的 10Mbps 发展到现在高端交换机单端口的 100Gbps 甚至更大的带宽。

CSMA/CD 的工作机制是：所有发送端发送信息前先侦测线路是否空闲，线路空闲时才可发送数据；如果发送数据时侦测到线路忙则产生了发送冲突，发送端会发送一个干扰信号告知其他发送端链路发送信息已经产生了冲突，这样发送端后续发送数据也采用一个退避算法，即采用一个随机延迟后再发送数据，直到探测到链路空闲才将数据发送出去；如果连续发送 16 次都是探测到链路冲突，那么就认为数据发送失败并返回错误给上层协议，以提示发送失败的错误信息。

以太网帧有两种格式 Ethernet II 和 IEEE802.3 的帧格式，字段都非常类似，区别在于 Ethernet II 的长度值在 0~1500 范围内，而 1536~65535（从 0x0600 到 0xFFFF）的范围保留给 IEEE802.3 的帧格式的类型使用。在数据链路上，以太网的每个数据帧除了自身数据外还都有前后两部分数据，帧前的 8B 数据包括用于位同步的 7B 前导码和用于帧同步的 1B 的定界符，帧后有用于帧间隙（Inter Frame Gap）传输的 12B 数据，原因是帧和帧之间需要有一段时间来让接收服务器做准备接收下一帧。数据帧的长度最小是 64B，如果发送的数据不到这个长度需要补零凑齐。

数据传输速率从最初的以太网单端口 10Mbps 提高到现在数据中心以太网的单端口 100Gbps 甚至更大，无疑是以太网成功的证明之一。最初使用同轴电缆和双绞线作为网络媒体，现在高带宽的光纤逐渐成为数据中心主要的传输线缆，而简单便捷的无线接入网络现已成为人们访问网络的主要接入网技术，包括电视电话在内的信息传输都逐渐转移到了 TCP/IP 网络上。当一种技术刚被提出的时候可能相关产品价格都不菲，但是随着相关产品被大量地生产和销售，相关产品产品的成本就会被分摊到很小，价格的降低更是推动了相关产品的普及；反过来，一定比例的产品利润能被大量应用到该技术更高版本的研发中去，使得该技术的产品被进一步地完善。

在网络方面的工作和学习中，需要熟悉以太网中常见的 EtherType 以及 TCP/IP 里的协议号和端口号等内容。知名的 EtherType 具体可以参见参考文献[21]，知名端口号可以参考 RFC1340，而知名协议号可以参见参考文献[22]里的内容。而知名端口号都是由互联网数字分配机构（The Internet Assigned Numbers Authority, IANA）统一分配的。

以太网报文封装格式有多层次的特点，当 TCP/IP 协议运行在以太网上的时候，应用程序数据较多的时候会将发送内容分成若干段，每段是一个报文的 Payload，然后根据传输协议添加一个 TCP、UDP 或 ICMP 等传输层的协议头，接着在这些字段前面封装 IP 报文头部作为其三层头，最后通过网卡发送出去的时候添加以太网头。当然，如果是 MPLS 报文封装格式，就是另一个封装模板了。这体现了网络的多样性（多种标准和多种格式，可能每种格式都有自己相应的应用场景）。报文的封装从原始的应用数据到添加一层层的报文头，要增加很多信息，这种方式增加了传输应用数据信息的冗余部分，因为对于对端接收者而言，这些数据本身是不需要的，只是为了传输过程的可控才添加的，这也符合信息论

的特性，想获得的信息更多，或保证信息传输得更精确，就必须提供更多的信息比特来传输信息或用于校验。

---

## 1.3 网络传输设备

从物理层到应用层，常见的网络设备有中继器（Repeater）、集线器（Hub）、网桥（Bridge）、交换机（Switch）和路由局域网中器（Router）等。在以太网中对数据报文传输而言，为什么需要这些网络设备？试想最初发明计算机的时候主要用于大型的计算，后来为了提高计算机的性能（虽然现在计算机的性能有了很大提高，但仍不能满足大运算量计算的需求），需要多台计算机同时参与运算，计算过程中的相互协作就需要相互之间能够同步通信。最开始使用的通信方式是多台设备直连，后来通过中继器等网络设备来加强信号功率，完成信号的复制、调整和放大等功能，以此来完成在延长网络传输的长度下，依然保持信息在内容准确的前提下被传递。

如前文所述，中继器属于物理层面的设备，其主要作用是对传输信号进行放大，以减少远距离传输导致信号衰减带来的误码率，属于对信号再生或还原的设备。

集线器与中继器类似，也有对接收到的信号再放大并延长传输距离和减少物理层因信号衰减导致误码的作用。两者不同的是，集线器可以用多条以太网双绞线或光纤将多个节点集合连接在以集线器为中心的同一段物理介质下。集线器也工作于物理层，可被视为多个端口的中继器；在一个广播域里，集线器是在一个以太网中当其侦测到链路发送数据的信号碰撞时，会发送阻塞信号通知整个链路的设备。

网桥可以说是一个智能的中继器，也有对接收到的信号放大的作用，主要通过数据帧进行转发，将多个数据链路层的区域互联起来。网桥的名字非常形象，正如走路过河的桥一样，有两个端口，从一个端口进入网桥，并穿过桥，从另一个端口转出，反之也可以。

交换机可以理解为多端口的网桥，为其任意连接的两个网络节点提供通达的信号链路。现在通常说的交换机是以太网交换机，单端口从 10Mbps 到 100Gbps，甚至有更大的带宽（如没有特殊提及，交换机端口和芯片带宽的单位都是比特每秒，即 bps），单芯片数据带宽从每秒兆比特到现在的每秒几太比特，如果通过多个芯片构成机架设备，则整机带宽更大。因为交换机有多个端口，所以所支持的网络规模和带宽范围较大，数据帧转发时如果有其目的 MAC 表项则根据 MAC 表项单播转发，否则进行广播到所有端口，所以引入了泛洪的问题，为解决泛洪问题引入了 VLAN 的概念来隔离不同的广播域。最初的交换机都工作在基于 MAC 或 MAC+VLAN ID 的数据链路层转发，后来引入了三层交换机转发以解决路由器路由带宽和性能受限的问题。路由器则是在互联网中将网络范围大小不一的局域网和广域网互联、将各种封装格式不同的网络进行互联（以太网、SLIP 和 PPP 等）、进行大量路由计算后将报文根据 IP 地址进行转发的网络设备，工作在网络层。

路由器和三层交换机到底有什么区别？这个问题需要从很多角度考虑，下面给出几个不同点。

- 三层交换机可以二层转发，也可以三层路由；路由器则是通常只作为三层路由的设备。
- 三层交换机基本采用商业交换芯片来提高传输性能，IP 转发时吞吐量一般

比路由器大；路由器对报文处理主要靠 NP (Network Processor)，路由器的功能丰富，比如服务质量 (Quality of Service, QoS) 特性较多等。

- 路由器可以实现数据报文分片和组合、压缩/解压缩、加密/解密以及 NAT 等功能，而交换机只能变相支持或部分支持这些功能，现在传统商业交换芯片只有部分型号结合附加/加解密模块来支持 DES 等加/解密处理，也只有很少的型号可以支持 NAT，并且表项数量远远不足，这也是在云计算的某些场景下仅使用交换机某些需求无法满足的原因之一。
- 现在的交换机多是针对以太网研发的网络设备，所以常称为以太网交换机，解析其他格式的封装报文一般需要做特殊配置；而路由器不仅是以太网也是多种不同类型链路层网络的路由区域交接点，所以除了以太网，还可以对多种格式封装的报文进行解析和封装，即支持多种接口的通信，比如 PPP、PPPoE、SLIP 等。
- 不同网络设备的東西价格自然不同，以前的交换机中，CPU 通常是 PPC、ARM 和 MIPS 居多，现在随着硬件价格的下降和性能要求的提升，用 x86 作为交换机的 CPU 依然是一种趋势；云计算中使用 ServerSwitch (参见文献[9]) 必然也是一种方案的热点。

随着网络发展和新技术的不断出现，导致在技术下二层和三层之间的界限也越来越模糊，比如后续有专门章节介绍的 MPLS 和 LISP 等技术。随着数据中心设备规模的增長，大数据存储和分析、数据挖掘、移动互联网和云计算技术的盛行，数据中心物理设备的快速上架、应用程序的便易部署和问题故障的快速排查与解决是实施运维工作的普遍需求，所以计算、存储和网络一体机设备成为各个设备商为计算高密度和虚拟化提供的一种解决方案，这种方案可以减少设备间的复杂连线和提高资源利用率，减少空间占用、能源消耗和安装部署的时间。这类

方案的典型产品有思科的 UCS (Unified Computing System)、华为的云计算 FusionCube 一体机和华三通信 UIS (统一基础架构系统) 等。

---

## 1.4 MAC 和 VLAN

为了让计算机或者说服务器之间能相互找到通信的双方,需要给每台计算机赋予一个名字或者分配一个地址,名字是现在的主机名 (HostName),而这个地址则是 MAC 地址。MAC 地址是收录在 Network Interface Card (NIC, 网卡) 里的用于标明网卡位置的信息符号。MAC 地址也叫硬件地址,是由 48bits (6B, 1byte=8bits) 的 6 组十六进制的数字组成的。0~23 位叫作组织唯一标志符 (Organizationally Unique Identifier, OUI), 是识别厂商的标识; 24~31 位是设备的 ID, 32~47 位是厂家的 Serial ID, 其中, 第 1 字节的末位置 1 是组播地址标志位。网卡的物理地址通常是由网卡生产厂家烧入网卡的 EPROM (一种闪存芯片, 通常可以通过程序擦写, 存储计算机传输的数据) 用来标识发出该数据的设备地址, 与接收该数据的主机地址 (不一定是最终处理该数据帧的设备 MAC 地址) 一起封装在数据帧的二层头里。相互通信的双方通过 MAC 地址寻找接收信息对端的方式需要网桥这类设备来完成, 网桥和交换机的作用就是先学习报文的 SMAC, 然后根据 DMAC 进行二层转发。后来随着计算机数量的进一步增多, 报文的广播非常多 (比如 ARP 请求和未知目的报文都会广播), 一方面导致了广播泛滥影响通信效率, 另一方面也使网络变得不安全了, 因为任何人都有可能接收到其他人的信息。于是提出了 VLAN 隔离技术来划分不同的广播域, 所以现在二层交换机的 MAC 表项通常是用 MAC+VLAN ID 的关键词来存储和查找的。VLAN Tag 的格式如图 1-2 所示。交换机中 MAC+VLAN ID 与端口 Port 的对应关

系用 FDB (Forwarding DataBase) 表来记录。

目的 MAC (6B)	源 MAC (6B)	类型 (2B)	数据 (46~1500B)	FCS (4B)
----------------	---------------	------------	------------------	-------------

(a) 不带 VLAN Tag 的数据帧格式

目的 MAC	源 MAC (6B)	Type (2B)	VLAN Tag (2B)	报文数据 (若干)	FCS (4B)
-----------	---------------	--------------	------------------	--------------	-------------

(b) 带 VLAN Tag 的数据帧格式

SType (2B)	STag (2B)	CType (2B)	CTag (2B)
---------------	--------------	---------------	--------------

(c) QinQ 双 VLAN Tag 格式

User Priority	CFI (1bit)	VLAN-ID (12bit)
------------------	---------------	--------------------

(d) VLAN Tag 内部字段

图 1-2 报文的 VLAN Tag 格式

在一个 VLAN Tag 里, 前 2 字节是 16bits 的标签协议标识 (Tag Protocol Identifier, TPID), 表示报文 VLAN Tag 的识别标记, 后面依次为 3bits 的 COS(Class of Service), 用于二层 QoS 分类作用 (又可以被称为 Priority Code Point, PCP); 1bits 的 CFI, 用于表示报文的封装是否是标准的封装格式和 12bits 的 VLAN ID。为了扩大 VLAN ID 的数量, 屏蔽私有 VLAN 透明地在公网上传输, 便提出了二层 VPN 的 QinQ 技术 (IEEE 802.1ad), QinQ 还有些别的称谓, 比如 Dot1q-tunneling、Tag in Tag、VLAN VPN、Double VLAN、Stack VLAN 等。IEEE 802.1q 协议规定该字段的取值为 0x8100; 各大通信设备商交换机设备默认采用协议规

定的 TPID 值(0x8100),其他常用的 VLAN TPID 包括 0x9100、0x9200 或者 0x9300 等值,而根据 IEEE 802.1ad (Virtual Bridged Local Area Networks Amendment),由于运营商的外层 VLAN TPID 被定义为 0x88a8,所以当前常见的各厂商实现的 QinQ 的外层 TPID 值和该标准有较大的不同,但各厂商的实现均大同小异,主要原因是各设备商采用的交换芯片商可选择的余地不多;外层 VLAN Tag 称为 S-tag (Service tag) 或 Outer Tag,内层 VLAN Tag 称为 C-Tag (Customer Tag) 或 Inner Tag。随着虚拟化技术的出现,虚拟机之间以及虚拟网络设备之间的通信和隔离对网络提出了新的要求,虽然技术基本原理变化不大,但是云计算中的网络有其独特之处,比如大二层的需求等。

最初的网络范围比较小,所以都在同一个广播域里。后来随着网络规模的扩大,一方面,报文根据 DMAC 查找不到出端口时默认广播的做法造成了广播泛滥的问题,另一方面,在网络范围内,如果将网卡设置成混杂模式,就能监听到所有网卡接收到的报文,无论这个报文 DMAC 是不是自己,两个方面都引入了安全问题。为了解决这两方面的问题引入了 VLAN 的概念。这个需求导致了新的网络产品交换机的出现;数据帧转发时,进入交换机内的报文都要先获取相应的 VLAN Tag,然后到交接的 MAC 表项里学习报文的 SMAC+VLAN ID,再根据 DMAC+VLAN ID 进行转发,查不到表项默认就向该 VLAN 内的所有端口广播;响应报文回来时,根据原先学习到的表项进行转发回来;这样就完成了 VLAN 的广播域隔离(不能转给其他 VLAN)和安全组的划分(不同的组通信用不同的 VLAN)。但是随着网络规模的扩大,可能会导致出现环路、MAC 表查找和维护工作复杂度较大等问题,如果一个 VLAN 内部的转发路径一旦成环,就会导致环路上的流量被不停地转发,从而导致阻塞,影响业务数据的正常通信,所以使用

防环协议如 STP 等就显得特别重要。

VLAN 出现后,在局域网内使用 VLAN 进行隔离域内进行本地通信,如果两个 VLAN 间的设备成员想通信怎么办?为了实现跨 VLAN 通信,采用了 IP 转发技术,即在报文中添加 IP 报文头,新增字段有源 IP 地址和目的 IP 地址,三层 QoS 所用字段 TOS,并吸取了二层环路经验添加了三层防环 TTL 字段等,这样主机通信在同一局域网(比如一个 VLAN)内用 MAC,不同的 VLAN 之间通信用 IP,形成了接入层、汇聚层与核心层等传统的三级层次的网络架构模型。在二层,如果两个 VLAN 互通需要使用 Trunk 技术(这样表述可能不是很精确,后续章节会详细讨论)。

专用虚拟局域网(Private VLAN, PVLAN)常用于企业内私有网络,每个 PVLAN 包含两种 VLAN:主 VLAN(Primary VLAN)和辅助 VLAN(Secondary VLAN)。其中辅助 VLAN 包含两种:隔离 VLAN(Isolated VLAN)和团体 VLAN(Community VLAN)。支持 PVLAN 的交换机的物理端口也被分为两类:混杂端口(Promiscuous Port)和主机端口(Host Port),其中,主机端口也包含两类:隔离端口(Isolated Port)和团体端口(Community Port)。混杂端口隶属于主 VLAN,主机端口隶属于辅助 VLAN,隔离端口属于隔离 VLAN,团体端口属于团体 VLAN。通信规则如下:

- 在隔离 VLAN 中,隔离端口只能和混杂端口通信,不能与除它之外的任何其他端口通信;
- 在团体 VLAN 中,团体端口可以和混杂端口通信,同一团体 VLAN 的团体端口之间也可以互相通信,但不能与其他端口通信;

- 不同主 VLAN 之间的任何端口都不能互相通信。

PVLAN 用于解决一个 VLAN 内部隔离网络流量的需求，这样可以突破 VLAN 数目 4096 的限制，以减少对 STP 的依赖以及减少 IP 地址的浪费的优势，PVLAN 的应用对于保证接入网络的数据通信的安全性是非常有效的。

从当时来看 12bits 的 VLAN ID，有 0~4095 等 4096 个隔离域范围足够使用了；VLAN ID 的概念经过 PVLAN 等的改进和补充，在现在以太网中有着广泛的应用。交换机对未知 DMAC 的报文在一个 VLAN 内广播的处理从最初来看是非常好的一个决策，但是从对现在网络技术的影响来看，尤其是针对云计算网络中的需求，这种做法并不是一种完美的策略。为了解决云计算虚拟网络中租户对应 VLAN 不够的问题，提出了 VXLAN、NVGRE、STT 和 Geneve 等技术，并且还要虚拟网络和物理网络的兼容工作，虽然暂时从数量上解决了 VLAN 不够的问题，但随之带来的是报文头部开销带来的带宽损耗、ECMP 路径选择时 HASH 值的影响和需要购买新的硬件设备才能支持新标准等影响。或许这时候，我们需要回到技术的起点，回到技术上仍然面对广播泛滥还没提出 VLAN 的出发点，再来深刻思考如何解决广播产生的泛洪和引入的安全问题。

---

## 1.5 MAC-in-MAC

1.4 节中提到为了解决 VLAN 数不够的问题，早期提出了 QinQ 技术来解决。QinQ 技术设想的模型是客户的设备在客户局域网（Customer Bridged Local Area Network）内，本地通信不需要扩展到其他网络里。如果客户有设备需要

连接到服务供应商网络 (Provider Bridged Network), 首先要经过客户桥 (Customer Bridge, CB) 打上客户 VLAN Tag 的标记, 然后通过供应商桥 (Provider Bridge, PB) 再打上一层 Tag 标记在服务供应商网络里的隔离范围。将 CB 打的 VLAN Tag 称为 C-Tag, 而 PB 打上的 VLAN Tag 称为 S-Tag; 所以服务供应商网络内数据报文的传输对于客户局域网络来说是透明不可见的。如果要开启 QinQ 功能, 一方面需要整个链路的支持, 另一方面, 开启后会对安全 (比如防火墙规则)、监控 (Remote Switched Port Analyzer, 远程端口镜像功能 RSPAN) 等功能造成不必要的影响, 最后也会给部署和运维等方面带来很多棘手的问题。

为了克服 QinQ 的局限性, 提出了 MAC-in-MAC 技术, 又称为 PBB (Provider Backbone Bridge), 对应的技术标准是 IEEE 802.1ah, 是针对骨干网流量工程支持提出的利用 MAC 头进行堆栈的桥接技术, 具体来讲就是将用户的报文封装在运营商的 MAC 帧里; 在云计算环境里, 因为所有 VM 的 MAC 地址生成都是可以控制的, 所以在云计算的租户网络隔离的时候可以采用这种 MAC-in-MAC 的方案, 从报文封装的角度来说, 这也是一种 Overlay 的方案。PBB-TE (Provider Backbone Bridge-Traffic Engineering, 对应的技术标准是 IEEE 802.1 Qay) 通过在 PBB 的基础上针对流量工程的运营商骨干桥接技术, 可以在 PBB 基础上关闭组播和 STP 协议, 不同客户的流量相互隔离, 增强了以太网业务的安全性, 因为通过预先规划防止了内部环路的生成, 所以大大提高了链路的利用率, 同时也能对各种流量的转发进行很好的控制, 比如, 负载均衡不均的问题可以得到很好的解决。MAC-in-MAC 的报文封装格式如图 1-3 所示。

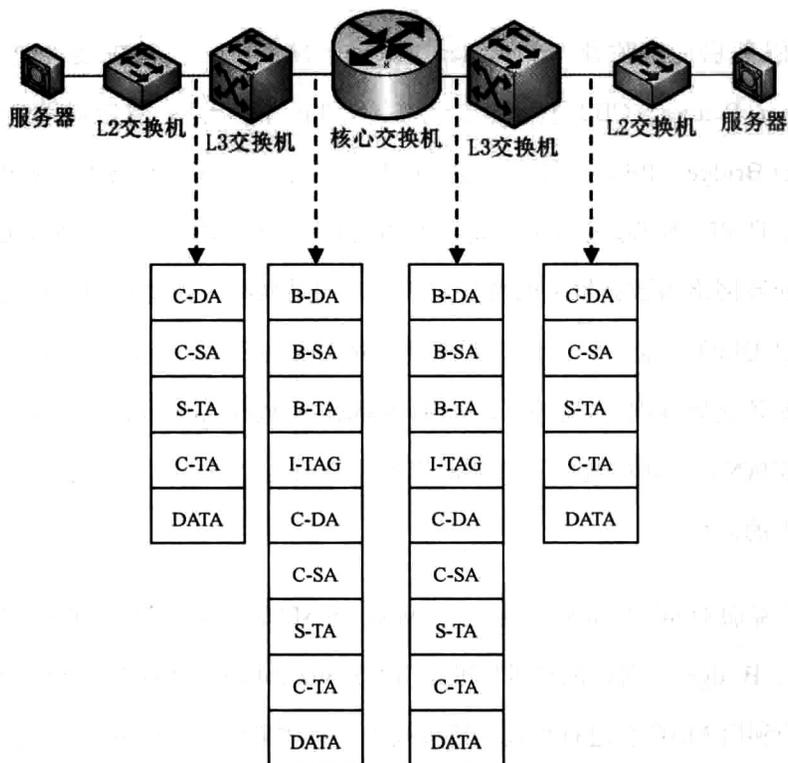


图 1-3 MAC-in-MAC 报文封装格式

在 PBB 的报文封装格式里，因为报文封装了两层 MAC 头，所以报文通常有两个 VLAN Tag。外层 MAC 头后的一个 VLAN Tag 称为 ETag 或 BTag，对应骨干网的 MAC 地址和桥接 VLAN ID，EtherType 值为 0x880a，在外层 Tag 后面 EtherType 为 0x88e7 的 ITag（Instance Tag）作为 MAC-in-MAC 封装的标志；内层 MAC 头后有一层 Tag，EtherType 值也是 0x880a，并且 VLAN Tag 对应的是 QinQ 的双层 Tag，外层是 Service Tag，内层是 Customer Tag；采用双层结构可以使用户和运营商的 MAC 地址学习互相隔离，一方面，避免了用户和运营商各自在 MAC 地址和 VLAN ID 方面使用的重叠而导致的影响，另一方面，为用户在透明传输业务的同时，运营商可以对网络数据报文的转发进行控制，从而不影响

运营商骨干网的稳定，减少广播帧以实现有效地利用带宽的目的。

---

## 1.6 STP 和 Trill

除了广播泛滥的问题外，在二层还有环路的问题，因为数据链路层报文字段中没有防止报文环路转发的字段（IP 报文中 TTL 字段的添加，就是为了规避二层报文在转发时没有防环字段的问题），所以报文转发的链路一旦成环，对正常通信的带宽占用是破坏性的，也必然导致正常的业务通信无法进行。

为了解决环路的问题，在网络方面提出了生成树协议（Spanning Tree Protocol, STP）及改进的快速生成树协议（Rapid Spanning Tree Protocol, RSTP）和多生成树协议（MultiPle Spanning Tree Protocol, MSTP）等。其中，STP 协议的运行过程大致如下。

① 将整个网络拓扑搭建完毕后，配置每个交换机的网桥 ID，以及根据各个端口速率等信息所决定的端口开销；各交换机开始发送 STP 的 BPDU（Bridge Protocol Data Unit）报文（目的 MAC 为 0x0180c2000000）来交换传递各自的 STP 信息。

② 选举根网桥（Root Bridge）。选举过程初始，每个网桥都认为自己是根网桥，收到 BPDU 报文的交换机比较网桥 ID 是否比自己的小，如果接收到的报文网桥 ID 比自己的小，则该网桥是根网桥，并将之作为后续发送 BPDU 报文的网桥 ID；如果接收到的网桥和自己的 ID 一致，则继续比较各个网桥的 MAC 地址，小的是根网桥；否则继续认为自己是根网桥，直到全网收敛即所有交换机对谁是

根网桥的结论有统一的结论。

③ 根端口 (Root Ports) 的选举, 在每一个非根网桥上都有唯一的一个根网桥, 选择依据是端口到根网桥的路径开销最小, 如果一个非根网桥到根网桥有多个端口到根网桥的路径开销一样小, 那么将端口 ID 最小的作为根端口。

④ 为每一个网段指定一个端口 (Designated Ports), 选择依据是除根端口外到根网桥的路径开销最小, 如果开销相同, 则用网桥 ID 最小的端口。

⑤ 之后其他端口都被设置成阻塞端口, 只能接收 BPDU 报文, 除了链路 Down 和阻塞状态的端口, 其他端口最终则可以正常转发业务报文。这一步虽然中断了环路的形成, 但没有造成链路带宽的浪费。

⑥ 一旦已经收敛的网络拓扑有变化, 整个网络需要重新进行 STP 的收敛计算过程: 检测到拓扑变化的交换机通过根端口向上层交换机传递 TCN (Topology Change Notification) 信息, 直到收到拓扑改变确认 (Topology Change Acknowledgement, TCA) 信息; 然后上层交换机按照此过程继续传递, 直到根网桥; 根网桥得到网络拓扑有变化的信息后广播 TC (Topology Change) 比特位被设置的配置 BPDU 报文, 所有的交换机都知道网络拓扑有变化后, 便开始新的 STP 收敛计算过程。在这个收敛的过程中, 如果网络范围较大, 时间久, 就会变得很长, 也就是 STP 的收敛时间会比较大, 这也是 STP 协议的一个很大缺陷。

STP 协议的端口类型分为以下 5 种。

① Disabled: 属于端口链路 Down 的情况, 不收发任何报文。

② Blocking: 仅可转发 BPDU, 不学习报文 MAC 地址。

- ③ **Listening:** 仅可收发 BPDU, 不学习报文 MAC 地址。
- ④ **Learning:** 仅可收发 BPDU, 学习报文的 MAC 地址。
- ⑤ **Forwarding:** 可收发 BUPD 报文, 还收发业务数据报文, 并学习 MAC 地址。

从 STP 的运行过程来看, 都会出现浪费链路带宽和在规模稍大且拓扑改变时收敛慢的缺点, 虽然后续为解决这些问题相继提出了 RSTP 和 MSTP 的协议作为修正补充, 但是仍然无法满足大规模运营的需求。现在在大型的数据中心, 可以通过网络结构 (三层网络架构包括核心、汇聚和接入三层, 也可以用只有核心和大二层的二层扁平化网络) 来规避环路, 从物理环路上避免环路的生成, 尤其是互联网企业的数据中心为了节省成本和减少带宽等的浪费, 大多数对 STP 协议的使用方案是不接受的。

由于 STP 等协议的缺陷, IETF 在交换上最新提出了 Trill 技术来克服 STP 原有的缺陷。Trill 是将原先浪费的链路进行了充分有效的利用, 并且支持 ECMP 的技术, 具有部署简单、防止环路、无带宽浪费和相对于 STP 快速收敛的特征, 从参考文献<sup>[21]</sup>中可以看到, Trill 的 Ethertype 为 0x22F3。Trill 的运行是在 IS-IS 协议扩展的基础上进行的, 支持运行 Trill 的交换机称为 RBridge (Router Bridge), 每个由 RBridge 构建的 Trill 网络称为 TrillCampus, 并通过扩展的 IS-IS 协议使用最短路径优先算法 (Shortest Path First Algorithm, SPF) 生成从该 RBridge 到其他 RBridge 的路由表。当一个以太网的数据帧经过 RBridge 时, 首先将整个报文封装一个包含本地 RBridge 信息和目的 RBridge 信息的 Trill 头, 然后在外面添加新的以太网头转换成已知单播报文发送出去。当目的 RBridge 接收到该报文通过目

的 RBridge 的信息得知是给自己的话, 就会将 Trill 报文的 Trill 头部等字段去除并解封装成以太网报文, 然后该以太网报文再从本地以太网中根据相应的转发规则进行转发。在 Trill 报文的转发过程中, 可以根据网络情况支持 ECMP 等转发方式, 最大限度地支持链路带宽的利用。

在云计算环境或者大二层通信中, Trill 技术被寄予厚望, 因为无论是在数据中心的设备搬迁还是在云计算网络中虚拟机迁移的需求场景下, 都希望被迁移的对象在迁移后其 MAC 和 IP 地址都保持不变, 以保证持续服务的不间断, 而 Trill 技术正是将二层的无环灵活配置和三层的路由规模进行了结合, 其头部格式如图 1-4 所示。

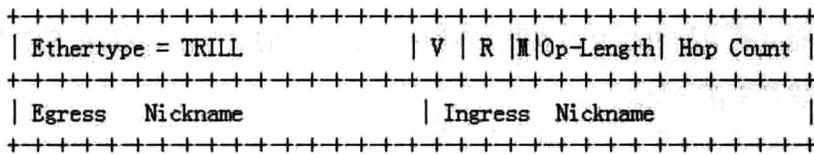


图 1-4 Trill 头部格式

对数据中心来说, Trill 技术的支持需要新的硬件设备, 即购买新硬件上的开销。另外, Trill 和 STP 一样受 VLAN 个数 4096 的限制, 所以, 在云计算的大二层环境下, 租户个数依然受限。Trill 的配置虽然很简单, 但只是相对于 STP 协议来讲的, 新加的报文头进行数据传输必然会增加额外的带宽开销, 并且真正到实施的时候, 可升级性、可操作性、安全性、稳定性和可运维性等都需要考虑。另外, 思科设备中和 Trill 比较接近的一个概念是 Fabric Path, 也是一个采用了新增加头和扩展的 IS-IS 协议进行封装报文, 可以说是思科替代 STP 的一个私有的解决方案。

## 1.7 IP 技术

数据传输中以太网承载的高层协议既可以是 TCP/IP 协议，也可以是 IPX 协议 (NetWare)、NetBEUI (NetBios Enhanced User Interface) 协议等。NetBEUI 在局域网中有着较高的通信效率，比如，在原来的 Windows 系统中发现网络邻居等。IPX/SPX 协议本身就是 Novell 开发的专用于 NetWare 网络中的协议，因其地址可以自动配置，现在局域网中基本被限于在联机的游戏网络中使用。而 TCP/IP 则被广泛用于大型互联网中，因而 IP 技术运行于以太网上也成为局域网技术的首选。

为了实现本地的跨 VLAN 间能够二层通信，提出了端口的 Access、Trunk 和 Hybrid 属性，这些内容后续在 2.3 节有详细说明。随着网络设备的大规模出现和网络互联区域的范围不断扩大，不同区域的网络之间互相传输信息提出了三层基于 IP 地址通信的概念，即两个 VLAN 之间可以用三层转发实现远距离通信，技术上通常通过路由器和三层交换机来实现这个功能。之所以保留原有 MAC 地址新提出 IP 地址进行通信，有技术也有历史的原因。

第一，转发表项若基于 MAC，则无法进行汇聚，因为无法确保某个网络范围区域中计算机网卡的 MAC 地址都是一个厂家供应且数字上是连续的，这就导致实际情况里很大可能是这些 MAC 地址分布在离散型比较大的一个范围里，那么 MAC 表项就无法汇聚，所以，如果此时采用交换机进行大规模网络互联，其需要记录的 MAC 表项就非常大，报文转发查找 MAC 表项的效率或导致的延迟等性能也可能会下降。

第二，IP 技术的提出当时正值美苏冷战时期，处理重要信息的计算机或网络设备尤其是军工网络的设备很容易遭到对方打击。这种情形下就要求网络有一定

的自愈性，即网络设备遭到部分破坏后，剩余的设备能通过信息自主交换的方式来自动寻找到新的全网互通链路，这个功能由后来的路由协议来实现。

第三，MAC 表项无法汇聚导致的 MAC 表项过大，为自动生成类似 IP 网络路由的网络转发规则带来了困难，包括 MAC 表项的存储、MAC 表项硬件转发表项设计和 MAC 表项下发到硬件时软件大量计算等方面；并且网络设备的 MAC 地址现在很多是可以随意更改的，运营商无法对 MAC 地址重复的客户进行有效管理，这体现在 MAC-in-MAC 技术的解决方案上。

综上所述，最终的 IP 地址被提出来作为广域网的一种信息传递技术，而对于 IP 地址和 MAC 地址的映射，网络设备则使用 ARP 或 RARP 协议来实现。

### 1.7.1 IP 地址

IPv4 地址有 32bits，除了支持类似 MAC 查找全匹配的主机路由外，还支持地址汇聚的远端路由，这样大大减少了网络设备中 IP 地址转发查找路由的路由条目，并且增加了路由设备路由表项的容纳能力。当主机路由或远端路由都没有匹配时，通常会由经配置的默认路由或策略路由进行处理。IP 地址为了实现上述功能，将 IP 地址分为两段：网络地址和主机地址，并且掩码可通过是有类路由还是无类路由来区分。有类路由不需要掩码，是按照预先的规定来区分一个 IP 地址中网络地址和主机地址的位数，它可分为 5 种，每种路由的网络数目和其每个网络中主机的数目是不同的。

#### 1. A 类地址

一个 A 类 IP 地址仅使用第一个 8 位组表示网络地址，剩下的 3 个 8 位组表示主机地址。A 类地址的第一个位总为 0，地址后面的 24 位（3 个点，十进制数）表示可能的主机地址，A 类网络地址的范围从 1.0.0.0 到 126.0.0.0。127.0.0.0 也是一个

A 类地址，但是它已被保留作闭环 (look back) 测试之用，而不能分配给一个网络。A 类网络地址有 126 个，每个 A 类网络地址能支持  $2^{24}-2=16777214$  个不同的主机地址。其中，从 10.0.0.0 到 10.255.255.255 的范围是作为私有网络地址来使用的。

## 2. B 类地址

B 类网络地址的范围从 128.1.0.0 到 191.255.0.0。B 类地址的计算逻辑是相当简单的。一个 B 类 IP 地址使用两个 8bits 的位组表示网络号，另外两个 8 位位组表示主机号。B 类网络有 16382 个地址，每一个 B 类网络地址能支持  $2^{16}-2=64534$  个唯一的主机地址。其中，从 172.16.0.0 到 172.31.255.255 的范围为私有网络地址，169.254.X.X 是保留地址，如果服务器 IP 地址是自动获取 IP 地址，而在网络上又没有找到可用的 DHCP 服务器，就会自动分配得到保留地址中的一个 IP。这个地址在云计算网络中也被用于在 VM 里获取 Metadata 信息。

## 3. C 类地址

C 类地址用于支持大量的小型网络，它使用三个 8 位位组表示网络地址，仅用一个 8 位位组表示主机号。因此，C 类网络地址范围从 192.0.1.0 至 223.255.254.0。C 类地址有 2097150 个网络地址，每个网络地址可以支持 254 个不同的主机地址。其中，192.168.0.0 和 192.168.255.255 为内网私有地址。

## 4. D 类地址

D 类地址用于 IP 网络中的组播 (Multicasting)，又称为多播。D 类地址空间的范围从 224.0.0.0 到 239.255.255.254。二层组播协议 (Internet Group Management Protocol, IGMP) 已经有三个版本，V1 的版本支持组播成员的加入通告和查询；V2 加入了组播成员离开通告以及 IGMP 查询器由 IGMP 协议产生，而不是由 V1 中的路由协议产生；V3 版本新添了组播组可以允许或拒绝某些组播源的数据。另外，

组播数据帧在二层没有开启组播协议时或组播协议里没有相应组播组时基本被广播，为了节约带宽、减少广播、增强安全和便于计费，可以在交换设备上开启 IGMP Snooping 功能，这样做可以将已知组播组的数据二层转发给指定的接收者。

## 5. E 类地址

E 类地址保留作研究之用，因此原本 Internet 上没有可用的 E 类地址，但随着 IP 地址耗尽，E 类地址也开始在一些方案中被使用。E 类地址的前 4 位恒为 1，因此，有效的地址范围从 240.0.0.0 至 255.255.255.255。

对无类地址来说，在有类的基础上通过掩码进一步划分子网范围，使得子网范围的掩码长度可以自由变换，以节省 IP 地址的分配和使用，从而更方便、灵活、有效地利用 IP 地址来组建网络。

### 1.7.2 IP 报文格式简介

IP 报文常见的封装格式有两个版本：IPv4 和 IPv6，以封装格式中的版本号来区分，两种报文的报文头格式有非常大的区别。IPv4 报文格式头部如图 1-5 所示。

0	4	8	16	19	24	31
版本	头部长	Tos	报文总长度			
标识			标志	片偏移		
生存时间 TTL		协议号	头部检验和			
32 位源 IP 地址						
32 位目的 IP 地址						
可选部分字段						
数据部分						

图 1-5 IPv4 报文格式头部

其中，4bits 的版本号是 4；4bits 的首部长度仅仅是 IP 报文头部的长度，以 4B 为单位；8bits 的 ToS (Type of Service) 用于差分 QoS 服务中的数据分类；16bits 的报文总长度是包括首部长度的整个 IP 报文的长度，单位是字节 (B)；16bits 的标识符代表发出该报文服务器所有发出报文的唯一标记；3bits 标记和 13bits 片偏移用于对报文分片和接收时重组；8bits 生存时间 (Time To Live, TTL) 是借鉴二层成环隐患的教训引入的三层防环字段；8bits 的协议号表示不同的 IP 报文的种类，常见的有 TCP、UDP、ICMP 等。16bits 校验和仅是对 IP 报文头部的检验，然后是 32 位的源 IP 地址和 32 位的目的 IP 地址；IP 包头后面可能还有可变长度的选项，填充区域随选项长度变化，用于确保长度为整 4B 的倍数。

IPv6 报文头部格式如图 1-6 所示。

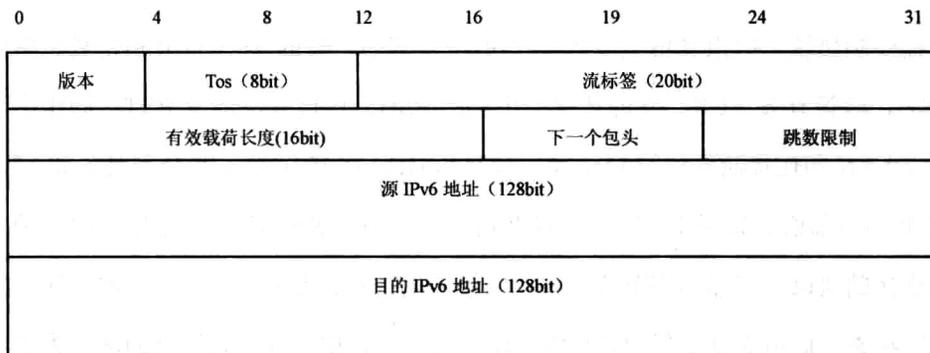


图 1-6 IPv6 报文头部格式

从图 1-6 的 IPv6 报文头部格式中，可以看出 IPv6 的设计比 IPv4 精简很多，仅保留了重要的几个字段；前 4bits 和 IPv4 一样，用于标志版本号，值恒为 6；接着是 8bits 的 ToS (Type of Service) 用于差分 QoS 服务中的数据分类；然后是

20bits 的流标签 (Flow Label) 用于标志在网络层不同流的特定报文, 也可用于 QoS 处理; 然后是 16bits 的有效净荷长度 (Payload Length) 用于表示该报文除去 IPv6 基本头部后剩余的有效长度, 包括 IPv6 头部的扩展部分; 再接着是 8bits 的 IPv6 下一个头部字段 (Next Header), 作用类似于 IPv4 的协议号; 然后是 8bits 的跳数限制 (Hop Limit), 其作用类似于 IPv4 的 TLL; 最后是长度均为 128bits 的 IPv6 的 SIP 地址和 DIP 地址。

在后续的介绍中, 除非有特殊说明, 所属内容基本上是在 IPv4 环境下的相关技术。需要注意, IPv5 是有这个协议的, 但基本与 IPv6 等没有任何关系, 主要用于实验性的资源预留协议 (详见 RFC 1190 ST2 和 RFC 1819 ST2+)。

### 1.7.3 TCP 和 UDP

TCP 和 UDP 是传输层的两种重要协议, TCP (Transmission Control Protocol, 传输控制协议) 的报文格式如图 1-7 所示, 其中, 6bits 标志位分别是紧急指针 URG、应答有效 ACK、立即传输应用程序 PSH、连接复位请求 RST、同步序号标志 SYN 和连接断开标记 FIN 等。TCP 和 UDP 在 IP 的协议号分别是 6 和 17, UDP 的头部格式如图 1-8 所示。TCP 协议是在不可靠的 IP 协议基础上的一种面向连接的协议、可靠的字节流协议。所谓面向连接, 是指 TCP 需要维持连接的前后状态, 而可靠则是指 TCP 协议保证发送的数据能确认到达目的端。为了实现这个面向连接的可靠传输功能, TCP 协议设计了复杂的状态机制, 并且 TCP 还是全双工服务的协议。

另外, 与 UDP 发送数据长度固定的方式不同, TCP 发送端的报文通过流量控制机制来发送合适大小的数据块, 并且对发送数据建立定时器等待目的端确认

收到这个数据包，如果定时器超时，则重发此份数据。所以，TCP 接收端除了对报文进行检验和校验外，还需要对报文进行去重和排序后才能传递给应用程序。去重和排序也是影响 TCP 报文接收性能的重要因素。

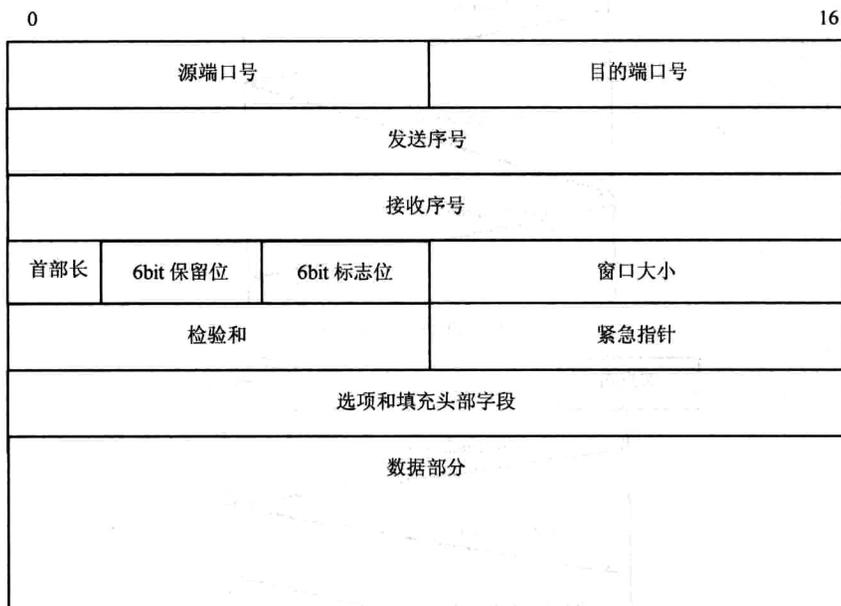


图 1-7 TCP 头部格式

用户数据报协议（User Datagram Protocol, UDP）头部有 8B，具体的头部格式如图 1-8 所示，这里字段的含义和 TCP 协议基本一致。

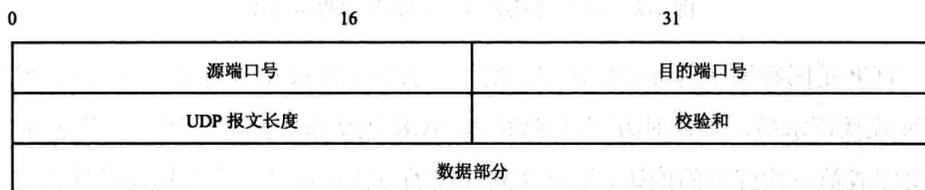
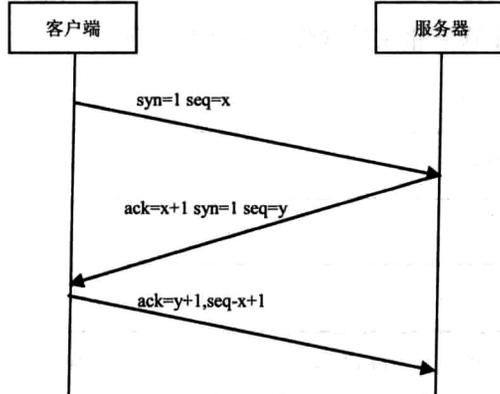
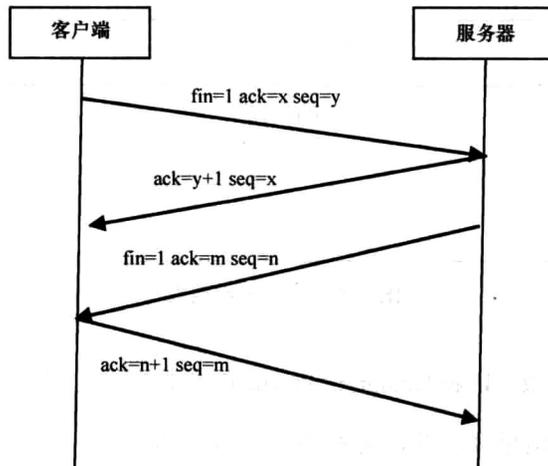


图 1-8 UDP 头部格式

TCP 连接的建立和断开过程如图 1-9 所示，其中，TCP 报文建立连接需要三次握手，断开 TCP 连接需要四次握手。



(a) TCP 建立连接三次握手



(b) TCP 断开连接四次握手

图 1-9 TCP 连接的建立和断开过程示意图

TCP 是因特网中的传输层协议，使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 SYN 和 ACK 确认报文到达，然后再发送 ACK 报文完成第三次握手的确认。这种建立连接的方法可以防止产生错误的连接或者连接的不完整，TCP 使用的流量控制协议是可变大小的滑动窗口协议。

三次握手的详细步骤如下。

第一次握手：建立连接时，HostA 发送 SYN 包 (SEQ=a) 到 HostB，并进入 SYN\_SEND 状态，等待 HostB 确认。

第二次握手：HostB 收到 SYN 包后，必须确认 HostA 的 SYN (ACK=a+1)，同时自己也送一个 SYN 包 (SEQ=b)，即 SYN+ACK 包，此时 HostB 进入 SYN\_RECV 状态。

第三次握手：HostA 收到 HostB 的 SYN+ACK 包，向 HostB 发送确认包 ACK (ACK=b+1)，此包发送完毕，HostA 和 HostB 进入 Established 状态，完成三次握手，建立 TCP 双向连接。

当数据传输完毕后，需要经过四次握手来断开 TCP 连接，其步骤如下。

① HostA 要终止连接，发送序列号为  $y$  且 FIN 置位的数据报文，同时确认此前收到的报文。

② HostB 收到 HostA 发送的段后，发送 ACK 段，确认号为  $y+1$ ，同时关闭连接。

③ 同时 HostB 发送序列号为  $n$  的且 FIN 置位的报文，通知对端连接需要关闭。

④ HostA 收到 HostB 发送的段后，发送 ACK 段，确认号为  $n+1$  的报文，同时关闭连接，这样就完成了 TCP 连接的双向关闭。

为什么建立连接需要三次握手呢？两次或四次行不行？简单地说，TCP 建立连接通过一次握手显然是不行的，若客户端仅仅发送了 SYN 报文就开始发送数据的处理方式，对于对方端来说很大可能是没有做好接收或处理数据的准备。两

次握手情况下，当客户端给服务器发送了 SYN 请求后，对端服务器也返回了 ACK+SYN 请求，如果此时服务器就开始发送数据，那么在服务器的 ACK+SYN 报文因某种原因丢弃的情况下，导致客户端一直在等待建立连接，而服务器端等待接收数据响应这样的死锁状态。当然，在三次握手已经满足的情况下，使用四次握手则明显有多余的步骤来浪费网络带宽。

TCP 还有很多种定时器，这些都有可能导致 TCP 连接状态的迁移，下面列举几种。

- 网络设备在发送 SYN 报文建立连接后会启动定时器，在对方端超时没有回应后将断开 TCP 连接。
- 当网络设备通过 TCP 连接发送某个数据后，若在某个时间范围内没有收到对方端确认接收该数据帧的确认帧时，发送端就会重新发送该数据。
- 当网络设备收到对方端窗口大小为 0 的通知后停止发送数据，并启动定时器，当定时器超时后，服务器会发送 1B 数据来探测对方端窗口是否已经打开。
- 空闲的 TCP 连接会建立定时器，当空闲时间超过定时器时间后，将断开 TCP 连接。

总之，TCP 在无连接的 IP 网络层上，发送端通过三次握手建立面向连接的可靠全双工连接，通过窗口大小来进行拥塞控制保证 QoS 服务传输质量，通过建立一系列的定时器的机制保证 TCP 状态能正常迁移，并通过在发送端分片和在接收端进行排序和去重的方法来发送大数据块的内容，最终为业务提供了面向连接的、可靠的字节流服务。

UDP 协议使用 IP 协议提供不可靠的传输层服务；在传输业务时，很多情况

需要对数据报文分片传输；另外，当 UDP 的数据报文到达服务器却没有端口监听时，那么服务器的协议栈将返回一个 ICMP 端口不可达的差错报文。相应的场景 TCP 报文则使用 RST 复位标记。

#### 1.7.4 TCP 与 UDP 检验和

TCP 和 UDP 报文的检验和字段算法与 IP 报文采用 CRC 校验和一样，只是计算时所采用的字段不同；TCP 校验和是一个端到端的校验和，由发送端计算，然后由接收端验证，其目的是为了发现 TCP 头部和数据在发送端到接收端之间发生的任何改动。如果接收方检测到校验和有差错，则 TCP 报文会被直接丢弃。TCP 校验和覆盖 TCP 头部和 TCP 数据，而 IP 头部中的校验和只覆盖 IP 的头部，不覆盖 IP 数据报中的任何数据。TCP 的校验和是必需的，而 UDP 的校验和是可选的，并且如果需要计算 UDP 的校验和也是使用了包含 UDP 头部和 UDP 数据的内容计算得出的校验和值。另外，检验和计算 TCP 和 UDP 计算校验和时，都要加上 12B 的伪头部。伪头部的格式如图 1-10 所示。



图 1-10 TCP/UDP/ICMP 校验和伪头部格式

## 1.8 DNS 和 DHCP

互联网名称与数字地址分配机构 (The Internet Corporation for Assigned Names and Numbers, ICANN) 是一个非营利性的国际组织，负责互联网协议 (IP)

地址的空间分配、协议标识符的指派、通用顶级域名（gTLD）、国家和地区顶级域名（ccTLD）系统的管理，以及根服务器系统的管理。这样每个提供网络服务的设备只需要一个简单的 IP 地址供客户端访问就可以。但是这么多 IP 数字让用户每次使用时靠记忆力来输入写 IP 地址组成的地址对用户来讲是非常痛苦的，所以网络界对服务器提出了域名的概念，即对服务器取一个在网络中唯一的容易记住的名字，用户可以通过服务器的名字对其进行访问。在底层网络中，用 DNS 协议解析域名和 IP 地址的对应关系。DNS 是一个以递归的方式进行解析的服务器，当服务器访问某个域名时首先访问最近的 DNS 服务器，如果已经有相应的域名对应 IP 地址的记录，则直接返回该 IP 地址；如果没有，则向上一级查询，在上一级 DNS 服务器也执行同样的上述行为，如果所有的 DNS 服务器都没有记录则最终向域名根服务器查询；当域名服务器从上级返回查询结果时，会先进行本地的记录保存以备后来服务器再查询用，然后再发给下级服务器，这样持续一直到发出查询请求的确切服务器为止。目前全球共有 13 台域名根服务器。其中，1 个为主根服务器，放置在美国，其余 12 个均为辅根服务器，有 9 个放置在美国、欧洲 2 个（位于英国和瑞典）、亚洲 1 个（位于日本），由 ICANN 统一管理。DNS 报文通常使用 UDP 报文，但也可以使用 TCP 报文，端口都是 53。当域名长度超过 512B 或主辅服务器同步时，需要使用 TCP 来传输 DNS 报文。

在 DNS 方面，网络安全中有一个典型的网络攻击行为是 DNS 劫持，又称为 DNS 钓鱼或域名劫持。它是在网络系统中拦截 DNS 的请求报文，分析请求的域名，将某些特定的域名返回虚假的 IP 地址或使 DNS 流程中达到失去正常响应的效果。DNS 劫持在被用于安全方面时，是针对某些不安全或内容不良的网站进行封锁，但是也会被某些黑客用来提供某些域名的虚假 IP 地址以欺骗客户的信息或钱财，比如，使用虚假银行地址骗取用户账号和密码、虚假购物或募捐网站

骗取用户支付钱财等，所以 DNS 劫持一旦被某些不法分子使用，其危害非常大，防止 DNS 劫持的方法是杜绝不良内容网站的访问及减少未知软件的安装；手工指定 DNS 解析服务器，比如国外的 Google 的 DNS 服务器 8.8.8.8 和国内的 114.114.114.114 常用域名服务器、定期对网络设备域名解析工作情况进行检查等，若有可能，可以直接以 IP 地址的形式访问网站。

动态主机配置协议（Dynamic Host Configuration Protocol, DHCP）是一个局域网协议，用于主机动态获取 IP 地址，使用 UDP 传输交互信息，其中，UDP67 和 UDP68 为正常的 DHCP 服务端口，分别作为 DHCP Server 和 DHCP Client 的服务端口；DHCP 为局域网内的服务器和网络设备自动分配 IP 地址，便于对内网设备进行管理。

局域网内的服务器可以手工配置静态 IP 地址，也可以通过 DHCP 获取 IP。DHCP 有三种分配 IP 地址的方式：自动分配（Automatic Allocation）、动态分配（Dynamic Allocation）和手工配置（Manual Allocation），最常用的是动态分配的方式，因为这种方式可以重复利用空闲的 IP 地址。

DHCP 的交互过程先是 DHCP Client 发送广播的 DHCP Discover 报文，然后 DHCP Server 收到 Client 的 Discover 报文后回复 DHCP Offer 报文，报文中含有给 Client 的 IP 地址，并且 MAC 和 IP 的对应信息会被 Server 端记录；接着 Client 接收到 Offer 报文后，将 IP 地址提取出来，发送免费 ARP 报文以检验环境中此 IP 地址是否被使用，如果没有被使用，则 Client 可以使用该 IP 地址，如果已经被使用，则继续发送 Discover 报文，当环境中有多多个 DHCP Server 时，通常会使用第一个接收到的 Offer 报文里的 IP 地址；然后 DHCP Client 发送 DHCP Request 报文给 Server 请求使用该 IP 地址，DHCP Server 收到 Request 请求后检验该 IP

是否被使用，如果空闲，则发送 ACK 报文，否则发送 Deny 报文；DHCP Client 收到 ACK 报文表示请求的 IP 地址可以使用，DHCP 交互过程结束，如果收到 Deny 报文，说明 IP 地址已经被占用，需要继续发送 DHCP Discover 报文进行下一轮的交互过程。

有时候，网络环境 DHCP 中会因为各种误操作或攻击因素导致额外的 DHCP Server 被启动，这样就导致了内网设备获取 IP 地址的混乱。为了防止此类场景的发生，可以在交换机上开启安全特性 DHCP Snooping 功能，一方面，会对 DHCP 的 ACK 报文进行监听，以提取 MAC 和 IP 对应关系，另一方面，最重要的是将交换机的端口分为信任端口和非信任端口两种，只有信任端口可以转发 DHCP Offer 报文，非信任端口则丢弃 Offer 报文，这样就能屏蔽假冒或者非信任 DHCP Server 对内网环境获取 IP 地址的影响。

另外，在大型数据中心里为了便于部署和管理，使得主机能够跨网段获取 IP 地址，需要使用 DHCP Realy 功能，该功能需要在三层交换机上开启才可以生效。

有一类特殊的 IP 地址范围是 169.254.\*.\*，这是原来 Windows 系统在 DHCP 获取地址失败的情况下自动分配给系统网卡的，后来 169.254.169.254 在亚马逊的 AWS 中作为 VM 里获取 metadata 信息的地址，OpenStack 里为了兼容 AWS 的接口，继承了这一做法，其底层实质上是通过 IPtables 将其映射到真实地址的。

---

## 1.9 ICMP 报文

ICMP 报文是协议号为 1 的 IP 报文，被用于在网络设备之间传递差错信息或其他重要的信息。ICMP 报文能传递信息的种类用类型及其相应的代码来表示，

类型和相应代码都用 8bits 来表示，即每种都有 256 类值，主要包括不可达、重定向、请求回显、时间戳请求与应答、地址掩码请求与应答和 TTL 生存时间为 0 等类型。其中，请求回显用于常用的 Ping 命令中，经常用于判断网络的畅通情况。不过需要注意的是，很多情况下纵然无法 Ping 通某台服务器，但此时很多服务还是可以从服务器获得的，因为 Ping 报文和其他服务的报文有很多特征上的区别。虽然 ICMP 报文被拦截，但是这些服务的报文可能被放行；而 traceroute 功能则是结合了 TTL 字段和 ICMP 报文来跟踪整个链路设备的（Linux 下称为 tracepath，Windows 下称为 tracert），不过不排除有些网络设备为了安全因素禁止了对 traceroute 功能的回应。另外需要注意，ICMP 差错报文、报文目的 IP 地址或源 IP 地址不是单播的报文、分片的非第一片报文等不会再产生 ICMP 差错报文。ICMP 报文头部格式如图 1-11 所示。对于差错报告报文类型的 ICMP 报文，数据字段包括 ICMP 差错信息和触发 ICMP 的整个原始数据报文内容，但原始数据报文长度不超过 576B。

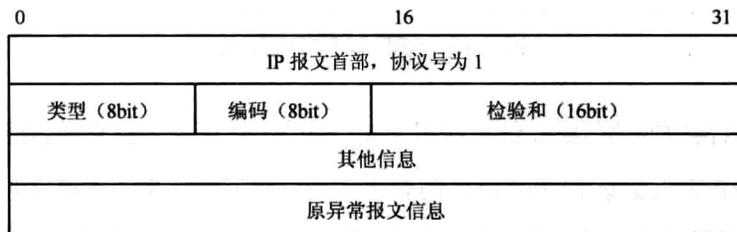


图 1-11 ICMP 头部格式

## 1.10 ARP 和 RARP

地址解析协议（Address Resolution Protocol，ARP）是根据 IP 地址获取物理地址的一个 TCP/IP 协议；反向地址转换协议（Reverse Address Resolution Protocol，

RARP)则相反,是通过网络设备物理地址来获取其对应的 IP 地址。ARP 和 RARP 非常重要,因为两个协议将二层转发和三层转发进行有机的结合,从而实现了二层本地网络转发与三层远距离传输的完美结合。ARP 和 RARP 的数据帧格式基本一致,在以太网类型中为了区分,ARP 的以太网类型值是 0x0806, RARP 的以太网类型值是 0x0835。ARP 和 RARP 的格式如图 1-12 所示,其中,硬件类型为 1,表示以太网;协议类型为 0x0800,表示 IP 地址;操作类型为 1,表示 ARP 请求,2 表示 ARP 应答,3 表示 RARP 请求,4 表示 RARP 应答。

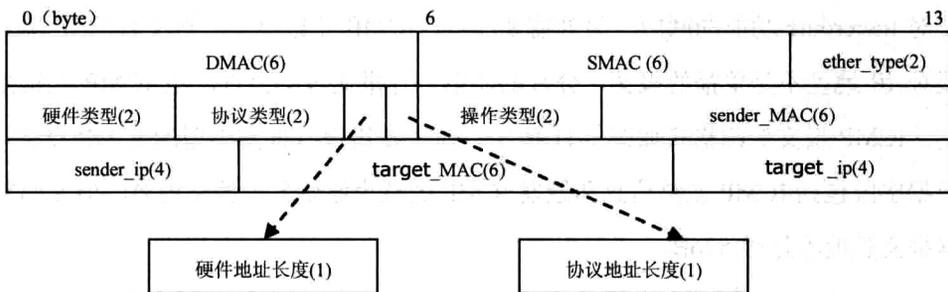


图 1-12 ARP 和 RARP 格式

在以太网中的通信需要经常使用 ARP 功能,在三层交换机或者路由器上都有一张 ARP 表来记录 IP 和 MAC 的对应关系。在下面步骤中将描述服务器发送报文的流程以及 ARP 报文在这个过程中所起的作用。

① 当网络设备发送一个数据报文时,首先会对该报文的 DIP 加上网络掩码的信息进行路由查找,并判断是否与发送端的 SIP 地址在同一个网段。

② 如果两者在一个网段,则查找 ARP 表的记录,得出对应 DIP 的 MAC 地址,如果已经有 ARP 表项记录,则返回 MAC 地址;如果查找不到,则发送 ARP 请求报文,以等待接收端进行 ARP 应答。

③ 如果本地网络范围内有网络设备对 ARP 请求进行应答并返回 MAC 地址，则记录相应 MAC 和 IP 地址对应的 ARP 关系；如果一直没有 ARP 应答，到超时后将返回发送报文失败，比如报文不可达等信息。

④ 然后对于有 ARP 应答的情形，发送端将报文封装帧目的 MAC 填充为对应的 MAC 地址，并通过服务器协议栈中相应的网段主机路由发送出去。

⑤ 如果报文的 DIP 不是本网段内的 IP 地址，那么直接查找协议栈的路由表项，分别是主机路由、远端路由和默认路由，直到查找到匹配项。大多数情况下会被默认路由表项发送到默认网关，再转到其他网络。

⑥ 如果 DIP 是其他网段的地址，又查找不到转发出去的路由，即默认路由也没有配置，那么将返回网络不可达的错误。

⑦ 如果报文在 WAN 网传输时，达到某个网络设备查找不到路由时，也会返回不可达的 ICMP 错误。

⑧ 直到发送出去的报文到达 DIP 的目的主机，然后开始对报文进行处理，包括对报文的回应，报文也按照上述处理步骤反方向回到源主机。

ARP 报文有两种特殊类型需要注意，一类是免费 ARP (Gratuitous ARP)，即主机发送一个请求自己 IP 地址的 ARP 请求报文，主要用途是确认局域网内没有重复的 IP 地址，DHCP 里主机动态获取 IP 地址的过程就有一个步骤是发送免费 ARP；另一类是代理 ARP (Proxy ARP)，其用途主要是当网关的网段范围属于连接某个主机的网段范围中的某一段，那么当这台主机发送的数据报文 DIP 不在网关地址范围，但在主机网段地址范围时，就会发送 ARP 请求，为了能正常通信，需要网关配置 ARP 代理代为应答，这样才能完成完整的通信过程。

---

## 1.11 路由协议

随着网络规模的逐步扩大，尤其是世界范围的互联网的出现，更是将世界的网络设备组成了一个超大规模的网络。为了能让这些跨洲际的 WAN 主机之间进行相互远距离通信，通信链路上的网络设备需要在双方向都下发一些路由规则。但面对如此大规模的网络都采用静态路由是不可能的，一方面是配置方式效率非常的低下，另一方面就是人工配置或脚本配置容易出错或无法及时响应网络的变化，因为如果网络一旦发生了变化，那么整个网络的静态路由很可能就需要跟随全局做相应的管理和调整，手工或脚本的静态路由显然是无法做到对变化的及时响应的。所以，为了能在网络自治系统内部和不同的网络自治系统可以正常地进行数据通信，需要使用动态的路由协议来在网络设备之间自动交互路由信息，并扩散相关的路由信息给其他网络设备，这样让每个进行路由计算的网络设备都能在尽可能短的时间内获取相关的路由信息及拓扑变动信息，然后通过统一的路由算法计算得出全网络路由信息，即达到全网路由的一个收敛，为网络各个设备点的网络互访提供全链路转发规则。常用的路由协议有路由信息协议（Routing Information Protocol, RIP）、开放式最短路径优先（Open Shortest Path First, OSPF）、IS-IS（Intermediate System to Intermediate System Routing Protocol）和边界网关协议（Border Gateway Protocol, BGP）等几种。在三层交换机或路由器上可能运行一种或同时运行多种路由器协议，路由协议获取的路由表信息最终形成 RIB（Route Information Base）表，经过一定规则的优先级选择，把选择的结果加入 FIB（Forward Information Base）表，FIB 供路由器的报文转发使用或在三层交换机上下发对应的转发规则到硬件交换芯片里。

最开始, 网络常用的路由协议是基于距离矢量 (Distance Vector Routing Protocol) 的 RIP (Routing Information Protocol) 协议, 后来提出基于链路状态 (Link State Routing Protocol) 的 OSPF 和 IS-IS (Intermediate System to Intermediate System Routing Protocol) 路由协议。上述三种路由协议属于自治系统内的路由协议, 通常称为内部网关协议 (Interior Gateway Protocol, IGP)。OSPF 和 IS-IS 相比, 两者都支持网络分层, IS-IS 路由协议是 ISO 的标准协议, 后来才增加对 IP 协议报文的路由计算的支持, 所以 IS-IS 在内部网关协议领域的 IP 转发应用是后来才可以的。OSPF 起初就是 IETF 为 IP 网络中的报文转发设计的, 应用于 IP 报文转发的完善程度和标准化程度也相对较高, 所以对 IP 协议中进行路由的计算和发布来讲 OSPF 的支持较为成熟。对 IS-IS 协议来说, 一个路由器是 Intermediate System (IS), 一个主服务器就是 End System (ES), 在一个服务器和路由器之间运行的协议叫作 ES-IS, 路由器与路由器之间运行的协议是 IS-IS 协议。后来为跨自治系统 (Autonomous System, AS) 间的路由选择提出了 BGP 路由协议, 其运行于 TCP 协议的 179 端口的连接上, 用作 Internet 上不同独立自治系统间连接的路由选择协议, 也可以用于自治系统内部。TCP/IP 网络发展初期规模较小, 路由协议使用的是 RIP 协议, 随着 AS 网络规模的扩大采用了 OSPF 或 IS-IS 协议来满足扩展需求。最近某些大型互联网公司的数据中心网络规模进一步扩大, 基于 IP 报文的 OSPF 协议的 LSA 广播泛滥等缺陷也暴露了出来, 所以使用 BGP 协议逐渐开始在数据中心内部成为趋势。

### 1.11.1 RIP 和 BGP

RIP 是基于 UDP 报文的 520 端口进行 RIP 进程间的通信, 使用 Bellman-Ford 算法的距离矢量算法的路由协议, 是由 Xerox 公司在 20 世纪 70 年代开发的, 最大跳数只有 15 跳, 所以其应用的网络规模有限; 路由更新报文 RIPv1 使用广播,

而 RIPv2 使用组播，并且 RIPv2 支持无类域间路由（Classless Inter-Domain Routing, CIDR），而 RIPng 是用在 IPv6 网络环境里的。RIP 路由协议的工作过程大致为：运行 RIP 协议的网络设备启动，先进行内部数据基于本地网卡地址信息的初始化，然后从该网络设备的所有运行 RIP 协议的网口上发送请求路由表项的报文，请求邻居设备的完整路由表项；邻居接收到路由请求报文后作出响应，将自己的路由表项发送给请求的网络设备；然后网络设备接收到多个邻居的路由表项后，逐条加入到自己的路由表中，或者对于设备中已经存在的路由表项，通过基于跳数来选择是否进行更新。这样就完成了与邻居路由表项的同步，即靠这种同步方式实现 RIP 路由协议域内所有的网络设备路由表项的收敛。然后每个运行 RIP 协议的网络设备每隔一段时间重新发送包含自身所有路由表的应答信息给邻居，以不间断地实现同步更新，默认每隔 30 秒将自己的路由表发送给相邻路由器；如果 180 秒内没有收到相邻路由器的路由更新，则将该路由置为不可用，再经 60 秒仍没有更新，则删除该条目路由；如果有网络拓扑变化等，还会触发网络设备重新进行网络路由的收敛计算工作。从 RIP 的工作过程可以看出，在网络中某个设备到另外一个设备的网络路径只会选择路径代价较小的那个，即使有多个等价路由也无法形成多路径转发，即无法支持 ECMP（Equal Cost MultiPath）；并且每隔固定时间周期的路由广播，会引入大量的广播报文。

BGP 也是一种基于距离矢量的路由协议，用来提供不同自治系统 AS 之间的路由选择，并确保自治系统之间能无环地交换路由选择信息。不同于 AS 系统内部的 OSPF 和 IS-IS 路由协议使用 Hello 报文进行保活，BGP 使用基于 TCP 连接的 keepalive 报文周期性地发送保活报文。BGP 的路由器有三张表：邻居关系表、转发数据库和路由表，其中邻居关系表用于存放所有的 BGP 邻居信息，转发数据库用来记录所有的近路网络及相应路径的属性，而路由表则由最佳路径组成，

包括 EBGP 路由和 IBGP 路由两部分。BGP 的邻居关系建立要经历 Idle（等待连接状态）、Connect（完成建立两端的 TCP 连接，发送 Open 分组给对方）、Opensent（TCP 连接建立后发送 Open 分组）、Active（Open 分组发送建立 BGP 邻居关系失败）、OpenConfirm（Open 分组后对方端成功地进行响应）和 Establised（BGP 邻居关系建立，开始交换路由信息）。BGP 路由器之间的交互信息类型有 Open（用于建立邻居关系）、Keepalive（对等路由器实体之间周期性地交互信息以保活）、Update（对等路由器实体间交换网络可达性信息）和 Notification（通知错误信息或路由器关闭信息）等。BGP 是一种基于策略的路由选择协议，BGP 选择最佳路由是根据路由的很多属性来进行选择的，并根据这些属性对 BGP 的路由进行汇总和决策。BGP 路由可以有很多属性，包括源头属性、下一跳属性和权重属性等。BGP 有三个重要的概念：路由波动（Router Flaps）、路由惩罚（Route Dampening）和路由反射器。其中，路由波动是指由于链路不稳定、设备接口故障和人为的施工及配置错误引起的每台路由器重新计算路由的现象；路由惩罚是为了减少不稳定并防止路由持续抖动，以增强稳定性；路由反射器（Route Reflector, RR）是为了解决 IBGP 域全网连接的一个问题——全连接时网络资源和 CPU 资源消耗过大，这样对一组 BGP 路由器，可以选择一个作为路由反射器，允许其他路由器通过它与组外的路由器建立连接，组内的其他路由器称为客户，组外的路由器与路由反射器交互信息的称为对等体，组内的其他机器称为该路由发射器的客户。

另外，BGP 可以结合 MPLS VPN 技术和 VRF（Virtual Routing and Forwarding）技术，来解决公网中进行私密数据传输的问题。

### 1.11.2 OSPF 协议

开放式最短路径优先（OpenShortest Path First, OPSF）是一种链路状态协议，即 OSPF 范围内的每个路由器含有整个网络拓扑的路由信息，这样可以计算选择

比较优的路径，路由表中的表项里考虑网络成本的因素是传输速度和延迟级等属性。OSPF 目前有三个版本，其中，OSPFv3 属于 IPv6 路由协议，OSPFv2 是现在较为成熟和常用的 IPv4 路由协议。下面对 OSPF 分别从邻居关系建立、路由信息同步机制、支持的网络类型和层次化等方面内容进行总体性介绍。

IP 转发模型中的原则是不同 IP 子网的主机通信必须经过一个或多个路由设备，两个相邻的路由设备通信必须各有一个相同 IP 子网的接口，所以 OSPF 协议的工作过程是首先和邻居通过邻居状态机建立邻接（Adjacency）关系，然后才能获取整个网络的链路信息。邻居状态机分为以下几个阶段。

- **Down:** 路由设备邻居状态机初始状态，表示 Dead-Interval 时间内没有接收到对方的 Hello 报文。
- **Attemp:** 它只有 NBMA 类型网络才有，表示具有 DR 选取资格的设备向配置的邻居发送 Hello 报文。
- **Init:** 表示收到对方的 Hello 报文，并发送一个填充自己 Router-ID 的 Hello 报文回应。
- **2-way:** 表示收到对方端含有 Router-ID 的 Hello 报文。路由器的 Router-ID 一般是接口地址中的一个，这不是 OSPF 标准的硬性规定。
- **Exstart:** 双方通过 DD 报文根据 Router-ID 的大小确定主从关系，以及用于数据交换的初始数据库描述数据包的序列号，以保证后面数据库同步地可靠传输，Router-ID 以大的为主。
- **Exchange:** 路由器将本地的路由用链路状态数据库描述报文摘要（DDP）发送给对方端。

- **Loading**: 路由器接收到对方端发来的 DDP, 会与本地的 LSDB 进行比较, 当发现不一致时会开始两端的链路状态数据库同步, 将缺少的和比较新的 LSA 发送 LSRequest 报文给对方端, 对方端收到 LSR 后会回复 LSUUpdate 报文, 本地将新的 LSA 更新到本地链路状态数据库中, 然后给对方端发送 LSACK 报文。这个过程一直进行, 直到两端的链路状态数据库同步为止。
- **FULL**: 当两边的链路状态数据库中 LSA 同步后, 就建立起了邻接关系。OSPF 的两个路由邻居如何判断两端是否拥有一致的状态数据库信息呢? 方法是首先判断两者所有 LSA 的条目是否一样, 如果一样再判断所有的 LSA 检验和的综合是否相同, 同时满足这两条的相邻路由器的链路状态数据库就是同步的状态。

随后每个路由器之间通过每 10 秒 (默认值, 本节中的数据没有特殊说明都是 OSPF 的默认设置值, 通常可以通过命令或其他配置方式进行修改) 发送一次 Hello 报文保活, 当 40 秒没有收到对端的 Hello 报文, 就宣告该链路失效, 并更新自己的链路状态数据库。其他路由器收到该链路失效通告后, 除了会更新自己的链路状态数据库外, 还会继续将该链路失效通告继续泛洪, 直到所有的路由器都收到该通知。

当相邻路由器建立好邻居关系后, 就各自拥有了自己的链路状态数据库, 路由器会在每 30 分钟将自己的 LSA 通过泛洪给相邻的所有路由器, 来通告一遍自己的链路状态数据库, 通过这种方式实现全网 OSPF 路由器的数据库同步。如前文介绍中提及的, OSPF 路由器间的通信报文分组有 5 种: (1) Hello 报文, 用于邻居间的保活和发现, TTL 通常为 1, 目的 IP 一般为邻居的 IP 地址或组播地址 224.0.0.5/224.0.0.6; (2) 链路状态描述报文 (Database Description Packet, DDP), 用来对链路状态数据库的 LSA 进行描述; (3) 链路状态请求报文 (LSRequest); (4) 链路状态更新报文 (LSUpdate); (5) 链路状态确认报文 (LS Ack), 主要用

于链路状态数据库的更新。在邻接关系中，相邻的设备不接收区域 ID 不匹配、路由死亡间隔不匹配、子网不匹配（地址和掩码）、Hello 报文间隔不匹配等的 Hello 报文，这样邻居关系就无法建立。

LSA 的概念在 OSPF 中非常重要，因为当一个路由器运行 OSPF 协议时，会产生一个或多个 LSA（链路状态通告），OSPF 的链路状态数据库就是所有路由器产生的 LSA 的集合，而相应的链路状态路由选择协议则是一个分布式的、冗余的数据库，用该数据库作为输入，采用 Dijkstra 算法一次性地计算出到达所有目的端的最短路径。LSA 通过可靠的泛洪过程被发送到其他路由器。LSA 有很多类型，并且有着不同的作用。图 1-13 所示是一个 LSA 报文的头部格式。LSA 的类型及其用途有：

- type 为 1 的是 routerLSA，用来描述路由器自己有效的端口和邻居；
- type 为 2 的是 networkLSA，用来描述一个网段及与该网络相连的路由器的标识；
- type 为 3 的是 NetworkSummary LSA，由 ABR（区边缘路由器）创建来描述区内 1 类和 2 类 LSA 中包含的子网；
- type 为 4 的是 ASBR Summary LSA，用来通告一条用于前往 ASBR 的主机路由；
- type 为 5 的是 ASEExternal LSA，由 ASBR 创建，用于描述被注入 OSPF 中的外部路由；
- type 为 6 的是 GroupMembership LSA，主要用于 MOSPF 的三层组播来宣告组播成员信息；

- type 为 7 的是 NSSA External LSA，用于 NSSA 外边区域 LSA 的引入，通过边缘路由器转换成 type 为 5 的 LSA 泛洪到整个 OSPF 区域；
- type 为 8 的是 ExternalAttributes LSA，用来替代内部 BGP 承载跨越 OSPF 路由选择域的 BGP 路径信息；
- type 为 9~11 的是 Opaque LSA，主要用来对通用 LSA 的扩展。

8bits	8bits	8bits	8bits
LS 时间		选项	类型
链路状态 ID			
Router-id			
LS 序列号			
LS 校验和			
长度			

图 1-13 LSA 报文头部格式

一个路由器通过 type 类型和链路状态 ID 来区分自己发出的不同 LSA，并通过 LS 序列号区分相同 LSA 的不同 LSA 实例。OSPF 里 LSA 序列号是从 0X80000001~07ffffff 的范围线性空间变化，默认情况下，几百年才会有重复值（一个路由器 5 秒内不能更新自己发出的 LSA 实例）。对于接收 LSA 的路由器来说，接收者不转发且不存储那些不能被识别为已知类型的 LSA。

一个路由器接到一个 LSA 后，如何判断该 LSA 是否是一个新实例呢？首先判断该 LSA 校验和是否正确，校验和不正确的 LSA 将被丢弃（路由器也会定期对自己的链路状态数据库进行校验和校验，以防止硬件和软件出错，默认为 5 分

钟)；其次 OSPF 里有相关规定，即特定链路和特定类型的 LSA 更新频率默认最高不超过 1 秒，也就是说，一个 LSA 实例距离该 LSA 上次更新不超过 1 秒，就会忽略该 LSA 更新；如果更新频率正常，返回判断序列号的大小，新的实例 LSA 序列号较大；如果序列号一致，就判断两个 LSA 实例的时间差是否在 15 分钟内，如果链的各个 LSA 的时间差值在 15 分钟内，则认为是同一个 LSA 实例，否则就是一个新的 LSA 实例。

如果一个路由器的某条 LSA 一个小时仍没有收到更新报文，则认为达到了 Max-Age；路由器含有 Max-Age 的 LSA 时，会先将自己链路状态数据库中的该 LSA 副本删除，然后进行泛洪到其他相邻路由器，其他路由器收到 Max-Age 的 LSA 也会将本地的副本删掉，然后继续泛洪。所以，一条链路的通达，若要能在各个路由器计算路由时被使用，条件是链路两端的路由器都必须宣告该链路有效。而且，只有在邻居建立邻近关系和新链路状态通告时才会导致邻居两端的链路状态数据库同步。LSA 计时生效通过 LSA 报文的 LS 时间字段来起作用。

OSPF 的网络类型有点对点、广播、非广播多点访问 (NBMA) 和点到多点，上述内容对点到点的网络基本都适用。如果任何连接到一个数据链路上的点发送单一分组都能被连在该链路上的其他节点收到，就是广播子网。广播子网的 Hello 报文目的 IP 是 224.0.0.5，通过指定路由器 (DR) 和备用指定路由器 (BDR) 来进行链路状态数据库同步；NBMA 网络支持两个以上的路由器且允许任意两个路由器直接通信，但是不支持链路广播，它必须通过配置来实现邻居发现，并且指定路由器中必须配置其他路由器的身份及是否适合做指定路由器，NBMA 也是通过指定路由器来实现链路状态数据库的同步，其健壮性和可靠性取决于下层数据链路的服务，主要用于帧中继、X.25 和 ATM 等类型的网络；点对多点与 NBMA

的模型区别是没有指定路由器，这样需要两两之间发生 Hello 报文保活与链路数据库同步。

OSPF 的区域 (Area) 是一种层次化路由选择方案，在构建大型网络和实现网络快速收敛方面有着重要的作用。OSPF 通过层次化路由通过区实现两层路由选择，路由知道本区的所有网段，对于其他分区只有通过分发才能取到远程目的地址的概要信息，这样可以实现路由表的增长速度与网段的数目的对数成正比的复杂关系，但是对路由信息进行了压缩可能导致非最优化转发。OSPF 的每个区有自己的链路状态数据库，该数据库由描述区路由器及网段互联的 router-LSA 和 network-LSA 组成，位于两个区的路由器称为边界路由器 (ABR)；当引入两个 ABR 时，就有两个默认的路由被注入区域，通过向默认的路由指定代价以解决多条缺省路由的选择问题。区和区之间采用距离矢量算法交换路由选择信息。OSPF 会有一个骨干区，Router-ID 一般为 0.0.0.0。AS 的边界路由器 (ASBR) 产生 AS-external-LSA 将外部路由信息 (OSPF 从其他路由协议获取到的、路由目的地址位于 OSPF 路由域外的路由信息称为外部路由信息) 引入到 OSPF 区。每一个 AS-external-LSA 通告一个网络地址前缀，并且用 type 为 4 的 ASBR summary-LSA 来通告 ASBR 的位置。为了支持某些低端小型路由器和较低带宽链路的区，引入了几个特殊的区：stub 区 (有的叫根区、末梢区等) 的特点是不支持 ASBR，没有 AS-external-LSA 和 ASBR summary-LSA，路由选择需要配置静态路由；另一个是 NSSA (not so stubby area)，它通过使用 type 为 7 的 NSSAexternal-LSA 将外部路由引入 OSPF 区，并在边界将 type 为 7 的 NSSAexternal-LSA 转换成 type 为 5 的 ASexternal-LSA 并泛洪到 OSPF 区。

OSPF 的扩展方式有新的应用、添加新的 LSA 类型、添加新的路由选择算法或改变链路状态数据库的同步方式等。OSPF 可以拒绝与某个路由器为邻居。现

在的扩展方式有：基于 TOS 的路由选择、存根区、需求线路扩展、NSSA 区、数据库溢出、external-attributes-LSA 和 MOSPF 等。MOSPF 大量被用于 IP 组播用途的私网上，它结合 IGMP 等组播协议为组播报文选择合理的路由。

OSPF 的优点在于能够快速收敛，采用区划分和 Hello 保活方式的通信报文占用链路带宽相对较低，安全性高，支持 ECMP，可用于大型网络。但也有一些缺点，例如，配置复杂，网络规模大的时候会有严重的 LSA 泛洪问题等。

---

## 1.12 NAT 技术

类似 VLAN ID 只有 12bits 一样，制定相关标准的时候人们认为 32bits 的 IPv4 地址范围已经大得不可想象，而现实情况是随着互联网的大规模发展，IPv4 地址枯竭已经来临。为了解决 IPv4 地址枯竭的问题，一方面除了提出 IPv6 的新体系（地址用 128bits 表示，现在来看可以满足需求，但是随着物联网和星际网络互联的出现，可能这个范围在将来某个时刻仍然需要不断扩充）外，还提出了一种利用私有网络地址到外网映射的缓冲过渡技术，称为网络地址转换（Network Address Translation, NAT）技术。

RFC1918 规定了三个保留私有地址段：10.0.0.0~10.255.255.255；172.16.0.0~172.31.255.255；192.168.0.0~192.168.255.255。NAT 技术除了缓解 IPv4 地址枯竭问题外，还被用于隐藏服务器真实 IP 地址或者实现两个内网地址是重叠网络的设备之间的通信。

NAT 有三种：静态（一个内网地址映射一个公网地址）、动态（内网到外网的 IP 地址映射是随机的）和 PAT（多个内网地址对应一个公网地址）等。一对一

就是一个私有地址对应一个公网 IP，这种方式并没有节约 IP 地址，当私网主机访问外网时，到 NAT 网关处将私有地址映射到公网地址，然后响应报文回来时进行逆映射；一对多的方式则是多个私网地址对应一个公网地址，这种方式也称为网络地址端口转换（Network Address Port Translation, NAPT）或端口地址转换（Port Address Translations, PAT），是现实中较多的方式，常见的包括公司企业家庭内部设备访问外网等场景，但这种方式需要用协议层解决外网服务器响应报文如何找到内网中发出请求的真实服务器，因为现在的映射是一对多，常用的是采用 TCP 或 UDP 的端口号。在云计算 OpenStack 中，两种模式都有对应的场景，刚建立虚拟机的时候，默认并没有绑定 Floating IP，但是 VM 在只有内网地址的条件下依然可以访问外网，比如能 Ping 通 114.114.114.114 服务器等，此时就利用一对多的 NAT 功能，转换成的外网地址是外网网关处的公网地址，即内网连接的虚拟路由器连接外网的 IP 地址；当租户给 VM 绑定浮动 IP 地址后，那么这台 VM 访问外网报文的源 IP 就在虚拟路由器处被映射成 Floating IP，且外网服务器访问这台 VM 服务的报文目的 IP 地址也需要用 VM 的 Floating IP，然后在虚拟路由器处进行逆映射到 VM 的私网地址。大部分报文用 TCP 或 UDP 通信问题不大，但是有些报文没有端口号就需要进行特殊处理。比如，VM 访问外网的 ICMP 报文的 NAT 通过 ICMP 报文 Identifier 字段的值来区分不同 VM 的 ICMP 报文，当然，Linux 和 Windows 不同的操作系统具体的处理技术细节也不尽相同。

NAT 技术可以进行动态映射，也可以进行静态映射。NAT 技术为 IPv4 网络带来巨大收益的同时，也有以下的弊端。

- 映射需要在 NAT 网关进行记录和删除等管理工作，增加了 NAT 网关的工作量，也增加了网络延迟。比如，云计算 OpenStack 里 NAT 的操作主要是依赖 Linux IPtables 的 NAT 实现，而当有大量 IPtables 规则的时候，报文

的转发延迟可能会有明显增加。

- IP 报文的 NAT 处理会导致网络定位（如 `traceroute tracepath` 等命令）可能失效或诡异的现象发生，比如，在外网的某台 Linux 系统服务器里对 OpenStack 的 VM 的 Floating IP 执行 `tracepatch` 操作，此时虚拟路由器的 NAT 功能可能会导致返回两次。这是因为确实在外网网关返回一次，到 VM 后返回的又被转换成外网网关的地址。
- NAT 功能可能会对一些服务造成影响，因为 NAT 功能需要对报文的 IP 地址或端口号改变，而修改报文头部字段信息的行为可能造成报文在安全保证设备处无法被正确检验，这样给网络安全带来隐患。

---

## 1.13 隧道技术

隧道技术是网络进行互联传递数据的一种基本方式，两个基础设施之间建立通道，通过通道将其他协议或封装格式的数据帧进行重新封装后发送或接收，以实现不同协议、不同封装格式或通过不同层次的网络进行通信，达到安全可靠、互通隔离和其他用途的目的。

隧道技术有主要以下几方面的应用。

### 1. VPN 技术和安全访问

可以利用 PPTP（Point to Point Tunneling Protocol）协议为客户机通过 PPTP 服务器提供加密的点对点通信，让 PPTP 的客户机单点接入方式连接 VPN 服务；或者通过 IP 安全协议（IP Security, IPSec）这一协议来提供 VPN 网关，以实现

多点对多点之间的 VPN 连接服务, IPsec 提供了 3 个基本协议: AH 协议为 IP 包提供信息源验证和完整性保证; ESP 协议提供加密机制; 密钥管理协议 (ISAKMP) 提供双方交流时的共享安全信息。

另外隧道或 VPN 是现在多数据中心安全互联的一种技术方式, 为了实现异地灾备或 CDN (Content Delivery Network), 需要在多地建设数据中心; 大型互联网公司的数据中心存储的信息数据可以说是人类社会文明的重要财富, 如果数据有丢失不仅对互联网公司有影响, 对社会文明进步来说也是一种极大的损失; 实现灾备功能需要多个数据中心的不断数据同步, 而这些数据在公网上传输就需要隧道的方式来保证安全, 经常用的技术有 MPLS VPN 或 IP+GRE 等方。

## 2. 通用路由封装

在 RFC1701 和 RFC1702 中定义通用路由封装协议 (Generic Routing Encapsulation, GRE), 主要解决用一种网络层协议来封装另一个网络层协议的方法步骤, 内容包括标识封装的协议号、检验和、密钥、序列号和包括两端 SIP 和 DIP 的定義的路由信息等, 通常依靠外层封装来解决路由问题; 允许用户使用 IP 封装 IP、IPX、AppleTalk, 并支持全部的路由协议, 如 RIP、OSPF 等; GRE 的封装没有提供加密和防止窃听的技术, 所以通常和 IPSEC 结合对数据加密提供安全服务。除了 GRE 外还有表 1-1 所示的通过隧道来封装报文的几种格式, 现在应用的一个场景是在 IPv6 替代 IPv4 的过渡环境中:

第一, 当主干核心网络用 IPv6 替换 IPv4 时, 可以在服务器本地两端都为 IPv4 网络, 再通过隧道将 IPv4 报文封装在 IPv6 报文中, 从而实现在 IPv6 的骨干网络上实现 IPv4 范围地址服务器的互相通信;

表 1-1 IPv4 和 IPv6 隧道的类型

外部头	隧道类型	协议号
IPv4	IPinIP	4
	6to4	41
	6to4Secure	41
	ISATAP	41
	GRE	47
	UDP AMT	17
	PIM SIM	103
IPv6	IPv4 in IPv6	4
	IPv6 in IPv6	41
	GRE	41
	UDP AMT	47
	PIM SIM	103

第二，当服务器端和客户端配置了 IPv6 地址时而骨干网络仍为 IPv4，与上述第一点中类似，可以通过将服务器的 IPv6 地址封装到 IPv4 报文里，实现两端 IPv6 地址的服务器通过 IPv4 的骨干网进行通信。

### 3. Overlay 技术

Overlay 技术也是隧道技术的一种，与通常隧道技术不同的是，以 GRE 为例来说隧道技术的二层头通常在隧道头前面，隧道头里只封装 IP 报文；Overlay 技术则通常是将整个以太网报文作为 DATA 封装在新报文中的，隧道报文封装新的二层头和三层头甚至会有新的 TCP 或 UDP 报文头。这些技术有 VPLS、VXLAN、

STT、NVGRE 和 Geneve 等标准,通过这些技术可以将分布在不同地域数据中心的已经互通的设备扩展为简化的二层传输技术,进行数据中心之间的通信(在思科技术和方案里称之为 Data Center Interconnect, DCI)。

Geneve 协议已经发布到 02 版本,笔者已经将其翻译成中文([http://blog.csdn.net/night\\_elf\\_1020/article/details/41967385](http://blog.csdn.net/night_elf_1020/article/details/41967385)),该协议欲将该类技术进行统一,在该书写作过程中暂未发现有硬件设备或软件方案支持。在这几种技术里,介绍一下笔者认为在趋势上将会最常用的 VXLAN 技术。VXLAN 技术是由 VMware、思科、Arista、Broadcom、Citrix 和红帽共同提出的 IETF 草案,用于解决数据中心多租户间通信和隔离时解决 VLAN 不够的问题;VXLAN 草案里给出的封装格式是 MAC-in-UDP,即将原始报文封装在 UDP 报文里,其本意是在用于不同数据中心之间的 VM 之间进行通信时数据报文的封装格式。但是这种技术是否是一种完美的解决方案呢?

下面对 VXLAN 技术提出的标准里主要想解决的几个问题进行说明。

### 1) STP 协议缺陷和 VLAN ID 数目少带来的限制

对于国内互联网的公司如 BAT 的数据中心,STP 协议多数是不开启的,明显的缺陷是 VXLAN 标准中所提及的,STP 会导致链路带宽的大量浪费,并且一旦拓扑规模增加到二三百台甚至有时候几十台的时候,STP 收敛性能会明显变慢;所以大型数据中心会通过对网络的合理规划,来规避环路,以避免对 STP 功能的使用。

### 2) 云计算中多租户环境下的通信和隔离

对于为了提高 CPU 的使用率以及方便服务器维护/迁移等操作的云计算来说,不同租户的云主机需要进行私网隔离,对于同一个租户的多个私网内的云主机,需要二层数据隔离而可以通过三层转发或路由进行通信,并且当云主机在多

个数据中心之间进行热迁移时，为了保持其提供服务的不间断，需要保证迁移前后在一个 VLAN 内，即多个数据中心组成的二层某个 VLAN 网络有多大，那么里面的 VM 就能迁移有多远，也就是所谓的大二层技术。然而一个数据中心的物理服务器规模十万级在国内已经出现，国外 Google 等已有百万级规模的数据中心，此类环境下仅有 12bits 的 VLAN 数目对于云计算的租户数目来讲是远远不够的；其实 VLAN 数目不够很早就被意识到，并提出了 QinQ 之类的技术（还有灵活 QinQ 等），为什么这些技术不直接被拿来在云计算虚拟网络环境里使用呢？因为虽然可以在数据中心内部控制报文的 Tag 情况，但是一旦报文到了公网，其路径经过的网络设备是路由器还是交换机是无法得知的，根本无法保证让带各种不同 Tag 的报文安然通过公网达到对端的数据中心；然后就有了 VXLAN 等类似的技术被提出和应用。

### 3) TOR 交换机的 MAC 地址表不充足

再讨论一下云计算环境下 TOR (Top of Rack) 交换机 MAC 地址表不足的问题，在以前 TOR 上的报文走三层转发或二层转发，只会学习到与其直连设备及二层相连设备的 MAC；而随着大二层的出现，不仅要学习本数据中心二层的设备 MAC 地址，还要学习更多其他数据中心二层范围内的设备 MAC 地址需要记录，包括海量的虚拟机的 MAC 地址，这无疑大大增加了 TOR 的 MAC 表项容量的要求。对于传统的商业交换芯片如 Broadcom 的 ESW 系列芯片，基本上都有 128KB 的 MAC 表项大小，这个容量相对于现代一个数据中心十万级甚至百万级设备的数据量真不算太大，因为现在的一台网络设备有多个网卡是比较常见的。

### 4) 针对 VXLAN 技术的内容有以下几点讨论。

① 没有说明同一个租户下不同 VNID 之间如何通信；以前对于同一个 VLAN

或不同的 VLAN 来说两个主机通过交换机进行通信，需要添加或删除报文的 VLAN Tag；但是对于同一个 VNID 或不同的 VNID 的主机进行通信，封装 VXLAN 报文还需要额外提供一个从 L2 以太网到 L4 UDP 的头部，而为了传递更多信息的新加封装头部必然也占用了一定的数据带宽，VXLAN 报文中 VXLAN 头部格式如图 1-14 所示。

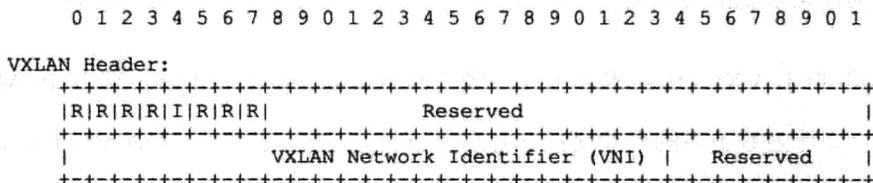


图 1-14 VXLAN 报文头部格式

② VLXAN 技术将隔离区域数目从原来的 12bits 的 VLAN ID 扩展到现在的 24bits 的 VNI，可能犹如原先 VLAN 提出时看待其 12bits 的大小人们认为已经很多了，但是现在网络规模越来越大，随着业务的需求扩展，比如多个行星之间的星际通信发展后，可能很快又无法满足需求；这种扩展的方式始终是有缺陷的。

### ③ 对原有 VLAN 技术的兼容

VXLAN 技术如其他革命性的网络技术的出现（OSPF 代替 RIP 或 IPv6 代替 IPv4），应该取代 VLAN，并且兼容 VLAN，这点 VXLAN 技术标准文档里也有提及，即如何让 VXLAN 里的设备与 VLAN 里的设备进行通信，或者也可以考虑将 VLAN 作为 VXLAN 一个子域的 PVLAN 技术；但是兼容性其他方面依然是个值得考虑的问题，比如三层交换机现有的 IP 转发是基于 VLAN 创建三层接口的，

可以简单理解为 DIP+VLAN 进行查找表项，然后更改报文的 VLAN ID 号转发到另一个 VLAN 里转出；当 VXLAN 代替 VLAN 后，芯片基于 VXLAN 的 VNID 如何进行三层转发是 VXLAN 标准里没有提及的内容，还有如 QoS 中 VLAN shapPing 的映射等技术也存在这个问题；而且封装后还会有报文转发效率、对转发负载均衡性能等方面的影响；尤其是对于 VLAN 和 VXLAN 并存的网络，需要添加翻译层来记录对应的关系，这点在云计算里也是一个很重要的功能点。

④ 对于 VXLAN 技术的出现，现有网络设备支持度也不够；仅有 BCM 的 Trident 2 等为数不多的商业交换芯片可以支持，值得一提的是 Open vSwitch 是支持的；VXLAN 技术应该与常规的传统三层转发芯片的 VLAN 间通信类似，应提供同一个 VNI 范围内的主机二层互通，不同 VNI 范围内的主机通过 IP 地址三层转发互通（一个租户有两个 VNID），跨数据中心的主机流量进行二层或三层通信时需要添加含有公网 IP 地址的 outer IP header 来走外网进行通信，当然有需求可以提出在 VXLAN 下 Trunk、Hybrid、Access 的端口属性，但是这些都没有做出规定；这个功能按照基于 VLAN 的三层转发来讲不是什么高级的特性；但是这项技术都必须在新的交换芯片上来做，而新出的芯片都是比较高级的，比如 Broadcom 的 Trident 2 系列，如果在上面添加这些功能，暂时没有出现 VXLAN 与某些技术，比如 MPLS、Trill 等技术兼容的方案，即如果用了 VXLAN，这些技术虽然被芯片支持，但很大可能是芯片的这些相关功能的表项就被浪费了；所以硬件支持工作还有相当大的工作要做。

⑤ VXLAN 并不是数据中心互联技术，而是在数据中心互联后将多个数据中心进行大二层划分的技术；所以数据中心的互联仍然需要借助传统的隧道或 VPN 技术等技术方式；比如 OpenStack 的部署如果是在多个数据中心的服务器设备，

那么多个数据中心的互联对于 OpenStack 是无法感知到的，OpenStack 的网络所能看到的仅是互联互通的一个大的服务器和网络设备的集群。

⑥ 大规模的部署 VXLAN 方案，因为对于组播报文、广播报文和未知单播都会封装到对应的组播报文里；当 VTEP（Virtual Tunnel End Point）的设备点特别多时，势必造成原来 VLAN 出现之前广播泛滥的情形，最终对 VTEP 的性能和稳定性造成影响，这个问题称之为 BUM 问题（Broadcast, Unknown unicast, Multicast，即二层广播，未知单播和组播）；解决该方法从底层来讲是通过风暴抑制和代理的方式减少 VTEP 接收或发送 BUM 报文的数量，从控制层面来讲可以结合 SDN 或其他控制手段来配置静态 ARP 表项或其他规则来减少 BUM 报文的产生和需求。

为了解决 VLAN 不够的问题，类似 VXLAN 的技术还有微软的 NVGRE 和 STT 等，笔者的意见是解决这类技术问题方式是非常不彻底的，尽管是革命性替代技术，但是兼容性不足且治标不治本。当初利用 VLAN 技术对网络进行隔离的选择可能是网络技术发展史上最大的一个错误决策，现在因为在其基础上建立的三层转发和受其限制而引发的一系列挽救技术都显得蹩脚，尤其在现在的云计算背景下；是不是应该考虑从根本上解决该问题而不是每次都做“补丁”的方式，因为这样做不仅导致了技术的复杂性越来越大，而且报文的这种添加头部的多次封装方式又导致了传输效率的降低，还会引入新的使用限制从而影响新技术被应用到各个领域，比如逐渐成熟的移动互联网。所以让问题回到二层，回到 VLAN 没有出现的时候所遇到的网络广播风暴和安全隔离的问题上，可能会提出更好的解决办法，衍生更多的好技术，以避免今天网络的痛苦和无奈。

## 1.14 MPLS 和 VPLS

多协议标签交换 (Multi-Protocol Label Switching, MPLS) 的产生有其特殊的历史背景, 当时因特网迅速发展产生的路由表项过大问题和 QoS 难以保障的难题对 IP 技术提出了更高的技术要求, 而 ATM 技术又存在成本高和效率低的缺陷。于是 IETF 结合两者的优势提出了 MPLS 技术, 旨在继续 IP 技术前提下结合 ATM 解决查找路由器效率低下和 QoS 流量工程困难的问题, MPLS 技术借鉴 ATM 表签转发机制的原理, 提高了 IP 报文转发的性能, 也为实现流量工程提供了基础。当时随着网络规模的日益增大, IP 转发也体现出了一些不足, 具体如下。

(1) IP 报文进行路由转发时, 每次查找时都需要先查找主机路由, 然后是远端路由, 最后是静态路由或策略路由。当骨干网的路由器表项非常多时, 在早先的设备里查找路由表给报文传输带来的延迟是有影响的; 而且 IP 报文中基于 DSCP 的 QoS 策略在整个网络拓扑中没有机制来保证整个链路的一致性 (Google 公司的 B4 网络改造, 在 QoS 整个网络拓扑所有链路的部署机制方面是很大的一个特色); 这种情况下考虑 ATM 技术的标签查找是非常高效的方式, 于是将两种技术结合而产生的 MPLS 报文。

(2) 互联网包括网络设备和服务器在内的网络规模越来越大, 如前文所述, 网络设备数量的急剧增加, 导致 IP 报文为如此多的业务提供 QoS 功能上无法满足要求, 具体原因如下:

① IP 转发最初只考虑了最短路径或最优路径转发, 没有顾全整个网络的链路带宽利用率的指标; 比如当多个业务的流量都非常大时, 数据报文可能都会按

照路由协议计算出的最短路径进行转发，导致最短路径流量带宽非常大，甚至超过链路带宽发生阻塞和丢包的现象，但是非最短或非最优路径却很空闲，其带宽一直被浪费；

② 随着业务类型的日益丰富，要求网络设备能根据不同的流提供相应的服务，但是 IP 的 QoS 基本上是没有服务质量的，或者说采用的是第一种服务模型 Best Effort（尽力而为的服务），虽然有 DSCP 字段，但是远远不能满足网络业务分类需求和完整链路 QoS 中端到端服务的保障。而面向连接的 ATM 技术则提供了优化网络的流量工程技术，在这种背景下，MPLS 才不得不采纳 ATM 的思想，利用资源预留协议（Resource Reservation Protocol, RSVP）提供一种较为全面良好的质量服务保证体系，从而为 IP 的多业务提供了可控的业务保障功能。因为 MPLS 报文里的 EXP 相当于 VLAN Tag 里的 COS 字段用于 QoS 功能。虚拟专用局域网服务（Virtual Private Lan Service, VPLS）和虚拟专用线服务（Virtual Private Wire Service, VPWS）是基于 IP/MPLS 的二层 VPN 技术，VPLS 和 VPWS 的区别是 VPLS 可以支持点到点、点到多点、多点到多点的业务类型，但是 VPWS 只支持点对点的业务类型。

MPLS 的 EtherType 是 0X8848，一个 MPLS 头部字段的格式如图 1-15。

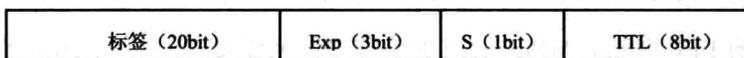


图 1-15 MPLS 头部字段格式

其中 20bits 的 label 流表签用于转发，3bits 的 Exp 相当于 VLAN Tag 里的 Cos 字段，1bits 的 S 字段代表该 MPLS 头是否是帧中最后一个 (Bottom of Stack)，后面 8bits 的 TTL 字段和 IP 报文里的 TTL 字段有一样的作用，防止数据报文在网络环路里持续转发。

MPLS VPN 是三层 VPN，其报文封装格式如图 1-16。

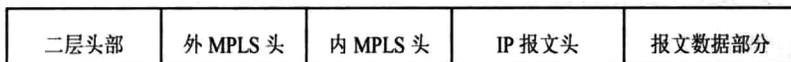


图 1-16 MPLS 封装格式

VPLS VPN 是二层 VPN，其报文封装格式如图 1-17。

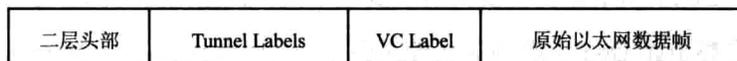


图 1-17 VPLS 封装格式

MPLS 的部署方式是在服务器端的边缘标记交换路由器（Label Edge Router, LER）上根据相应的组网策略配置转发等效类（Forwarding Equivalence Class, FEC），目的是在 LER 上将 IP 报文安装 FEC 的规则封装成 MPLS 报文，然后 MPLS 报文通过标记交换路由器（Label Switched Router, LSR）依据 MPLS 报文的 lable 标签进行转发到目的地；FEC 是将某些有共同特征的数据流打上同一个集合的流表签，这样保证具有该特征的数据会在 LSR 中都有相同的处理方式，和 IP 报文的路由协议类似，MPLS 报文使用标签分发协议（Label Distribution Protocol, LDP）来协调 LSR 之间的 FEC 绑定，并把每个 LSR 的 FEC 绑定通告给相连的 LSR，使得全局每个 LSR 能对全网有一个统一的 FEC 绑定信息集合；当 MPLS 报文到达目的地的 LER 处时，MPLS 报文的头部将被全部剥离，恢复成相应普通 IP 报文，相应通信的回复报文也按照相同方式进行处理。

后来随着三层交换机的出现，MPLS 技术渐渐失去其优势，因为交换芯片属于硬件转发，无论对于 IP 报文还是 MPLS 报文转发的速率都是一样的，比如万兆交换机，两者的处理速度都在一个物理端口上都是万兆的速率。所以 MPLS 曾经一度被认为是将被弃用的技术。但是后来的数据中心互连和 VPN 技术为

MPLS 的应用带来了再一次的辉煌。将 MPLS 技术根据封装的方式和转发时 VPN 路由处理方式不同分为二层 VPN (VPLS VPN) 和三层 VPN (MPLS VPN), 这种技术在骨干网上和数据中心互连上被广泛应用。现在 MPLS VPN 也是一种很常用的云计算隔离方案, 相应的产品有 Juniper 的 OpenContrail (OpenContrail 是 Juniper 的开源版, 还有商业版 Juniper Networks Contrail), 其优势在于能结合传统的支持 MPLS 的网络设备实现网络隔离, 客户不用更新网络设备来支持新的技术, 这也是很多客户在节省成本上非常容易接受的方案。

---

## 1.15 QoS 功能

网络的流量因为链路带宽、时延等因素的影响在路由协议计算出的可能不是最优的路径或对某些流量不是最优路径, 需要引入 QoS 功能来提升用户体验。QoS 从广义面来讲很多内容都属于 QoS 的范畴, 比如等价多路径(Equal-Cost Multi Path Routing, ECMP)/链路汇聚(Link Aggregation, LAG)等技术为了增大带宽实现链路不丢包等目的; 通常来讲 QoS 是为了减少报文传输延迟、降低时延抖动、丢包率从而提高链路带宽利用率和吞吐量的基础技术。

QoS 有如下三种服务模型。

(1) 尽力而为服务模型 (Best-Effort Service) 是最简单的现实中很多场景下默认采用的服务模型, 对时延大小不提供保证, 仅是提供最大努力下一种基础 QoS 服务; 默认情况下采用先入先出 (first in first out, FIFO) 的队列机制实现。

(2) 综合服务模型 (Integrated Service, Int-Serv) 是一种端到端的服务模型, 需要一种全局资源的控制支配权, 即当有一种业务需要传输报文所需的各种网络

资源时，先申请整个链路的带宽消耗等资源，申请成功后通信双方再通过该网络资源进行业务的交互；综合服务模型通过 RSVP 协议进行汇总系统资源和监控每个业务流对网络资源的使用情况；这种模型从业务层次来讲一方面需要底层带宽足够大才能按需分配，可扩展性和配置灵活性很差，另一方面申请网络资源的过程对网络设备要求高且会导致业务传输的延迟；MPLS 的流量工程可以结合 RSVP 协议采用综合服务模型。

(3) 区分服务模型 (Differentiated Service, Diff-Serv) 是在二层或三层转发时网络设备基于报文的某些字段来区分不同的传输流或业务，从而后续提供不同的 QoS 服务，这种差异化的服务即是区分服务模型；可用的字段二层通常是 VLAN Tag 里的 COS 字段，三层则是 IP 报文里的 DSCP 字段；区分服务模型是在现今带宽受限的情况下一种容易部署、扩展性好的实用模型，但是也存在业务区分度粒度粗和全局服务不统一的缺陷。

在网络设备中，QoS 的处理流程大概包括 Classifying (分类)、Policing (策略)、ReMarking (重标记)、Queueing (队列)、Scheduling (调度) 等步骤。分类步骤中，报文中可以用于对报文分类的方法有二层报文的 COS 字段或 MPLS 的 EXP 字段、IP 报文的 DSCP 字段以及根据 ACL 对报文进行分类等；策略则是根据对报文的分类进行相应的处理，包括通过 meter 染色或限速等；重标记则是根据策略的结果对报文的某些用于 QoS 的字段比如 COS 或 DSCP 进行设置特定的值，供后续路径中进行特定的 QoS 分类和策略用。入队列则是根据报文的分类或策略进出端口的不同队列；调度则是对不同队列的报文从出端口发送出去的不同方式，包括严格优先级 (SP)、轮询 (RR)、基于权重的轮询 (WRR) 等。加权随机先期检测 (Weighted Random Early Detection, WRED) 功能当某个接口

开始出现拥塞时，它有选择地丢弃较低优先级的通信，而不是简单地随机丢弃分组；WRED 功能可有效避免 TCP 连接慢同步问题。

需要提及的是根据 IP 报文的 TOS 字段进行分类时，有很多种分类方法。在 TCP/IP 协议栈里，对报文的 QoS 分类也主要是靠 IP 头部的 8bits 的 TOS 字段来完成的，主要应用于 IETF 在 1998 年 12 月发布的 Diff-Serv 的模型。TOS 字段有两种解析方式分别如图 1-18 的上下图所示；图 1-18 下图是利用 IP 报文的前 6 个 bits 作为差分服务代码点（Differentiated Services Code Point, DSCP）值，后 2 个 bits 最初未使用保留，后来这两个 bits 被用于显示拥塞通知（Explicit Congestion Notification, ECN）。DSCP 就是可以将流分为 64 种不同的类型或者优先级来进行分别处理；ECN 则是在整条链路开启该功能后，当网络设备传输 TCP 报文出现拥塞时，就会使用下一个 ACK 报文将 TOS 的最后两个 bits 置位，以这种方式显示的通知发送方链路发生拥塞现象，然后发送方做出相应的减小拥塞窗口，以缓解堵塞降低丢包。

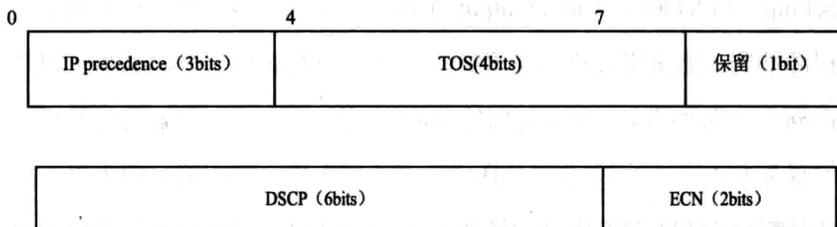


图 1-18 TOS 字段解析

从图 1-18 上图可以看出 IP 报文头部 TOS 字段的前三个字段称为 IP precedence，主要是与二层 QoS 的字段 VLAN Tag 的 COS 值相对应的 8 种类型分流；中间 4 个 bits 也称为 TOS 值，主要用于如表 1-2 所示的服务类型。

表 1-2 TOS 字段 4bits 的含义

二 进 制	十 进 制	意 义
1000	8	最小延迟 (md)
0100	4	最大 throughput (mt)
0010	2	最大可靠性 (mr)
0001	1	最小成本 (mmc)
0000	0	正常服务

在商业交换芯片中, QoS 方面涉及很多功能比如流量监管 (Traffic Policing)、流量整形 (Traffic Shaping)、带宽控制 (Bandwidth-Control) 和风暴抑制 (Storm-Control) 的实现使用了令牌桶 (令牌-Bucket) 和漏桶机制 (Leaky-Bucket) 等。Linux 内核协议栈也有很多 QoS 的功能, 大部分是基于 TC (Traffic Control) 机制来实现的。这些都会在后面由第 2 章和第 3 章里结合交换机及 Linux 内核功能做进一步的介绍。

谈到 QoS 设备商或芯片商的具体实现, 需要解释两个概念——HOL (Head of Line Blocking) 和 VOQ (Virtual Output Queue)。在网络数据报文传输时, 在 QoS 处理流程环节里, 通常需要进入某个队列 (对于没有队列区分的基本可以认为所有报文在同一个队列里), 当该队列的前面报文阻塞时, 可能该队列的后续报文和前面的报文并不是一个目的地, 但是后面的报文也会被阻塞, 因为同一个队列里的报文是按照 FIFO 的顺序进行转发的, 这种以为特定队列的对头前面报文阻塞而导致后面队尾报文也阻塞的现象称之为 HOL; 比如图 1-19 VOQ 原理示意图中的交换机有 A、B、C 和 D 四个端口, A 端口和 B 端口作为入端口都没有区分队列, 即各自端口的所有报文在同一个队列里, A 端口的流量各有 50% 的线速流量转发到 C 端口和 D 端口, B 端口分别有 40% 的线速流量和 60% 的线速流量转发到 C 端口和 D 端口, 这个时候可以算出 D 端口的流量超过了带宽, 但是 C 端

口流量带宽还有剩余，但是因为 D 端口的带宽耗尽有阻塞会影响 C 端口的流量也会被部分丢弃。VOQ 是为了避免 HOL 问题提出的一种解决方案，它是通过将上述例子中 A 和 B 两个入端口分别到 C 和 D 的流量划入到不同的虚拟队列，通过端口的 Flow Control 机制来通知是哪部分流量有 HOL，从而只引起该部分流量进行丢包，对其他流量则无影响，在硬件里支持 VOQ 功能的交换芯片的型号也已越来越多。

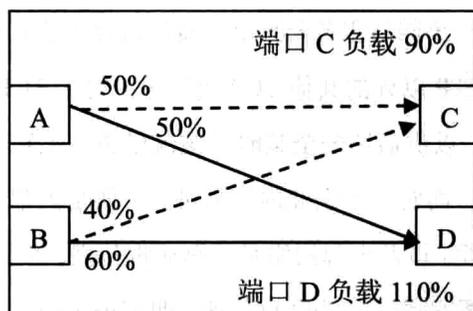


图 1-19 VOQ 原理示意图

## 1.16 网络安全和监控

计算机网络的最初设计只是为了内部进行大规模计算和数据共享来使用，后来随着规模的扩大和互联网的出现，网络安全问题成为焦点，因为最初的网络设计并没有考虑网络会出现安全的问题，包括数据安全和隔离安全；这个现象一方面说明网络技术发展历程中网络设计技术的不完善，另一方面也为后续无数做安全的厂商提供了生存和发展的机会，这些安全厂商也为 IT 人员的就业提供了大量的职位。需要说明的是，网络技术本身没有好坏之别，区分之处在于如何使用网络技术以及使用网络技术的人员的动机。

安全问题具体包括外界访问安全的控制及拒绝服务 (Denial of Service, DoS) 攻击识别和清洗、内部安全的病毒传播及用户密码的安全等方面。网络安全能得以实施的基础是网络监控, 网络监控的目的是对网络中的数据流进行分类和统计, 利用分类和统计的结果根据先验的网络安全知识进行模式识别, 从而达到网络安全中防止入侵、DDoS 攻击和病毒传播的目的。

网络监控在数据中心及企业内部主要是指内部网络的访问外网的监视和内部流量的监控等; 对一个网站服务器来讲, 监控内容主要有访问流量类型, 比如 HTTP、UDP 和除 HTTP 以外的其他 TCP 流量等, 对于 TCP 还需要区分 SYN、ACK 等标记的报文, 以供后续安全策略对该流量进行识别, 来判断是正常流量还是异常的攻击流量; 而对一个企业内部来讲, 主要是内部网络访问外网的交互数据是否违反安全规定, 以及内部网络是否有病毒传播、木马入侵等行为的控制。网络流量的监控大多数部署方式为旁路部署, 即在正常网络拓扑里增加一条新的数据链路, 用于将正常流量镜像或抽样到安全策略控制器处, 安全策略控制器对流量进行分析和识别来判断当前流量是正常流量还是攻击流量, 如果是异常流量, 安全策略控制器会对网络设备下发安全策略将异常的流量清洗掉。从控制网络流量到安全策略控制器的技术有很多, 主要包括 SPAN (Switched Port Analyzer) 和 sFlow 等; SPAN 技术包括本地 SPAN (Local SPAN)、远端 SPAN (Remote SPAN) 和 ERSPAN (Enhanced Remoted SPAN) 等, 本地 SPAN 包括端口镜像、流镜像、VLAN 镜像等几种; 下面针对每种流量提取技术详细介绍如下。

1. **端口镜像:** 这种技术是在支持端口镜像的交换机进行配置, 选择一个或几个端口作为镜像的源端口, 再选择另外的一个或几个端口连接分析镜像数据的流量分析仪, 这个端口称之为镜像的目的端口; 对镜像源端口入方向的数据流进行的镜像称之为 Ingress Mirror, 反之对镜像源端口出方向的数据流镜像称之为

Egress Mirror，镜像源端口可以同时配置出入方向的镜像。

2. **流镜像**：流镜像和端口镜像概念基本一致，只是流镜像是针对流的，而不是具体的端口；详细配置过程是通过 ACL 规则匹配一类流，将 ACL 的动作设置为镜像（Mirror），然后所有匹配该 ACL 规则的流量数据报文就会被复制一份到镜像目的端口；当然从技术上来讲 ACL 规则可以匹配端口的任意出入方向。

3. **VLAN 镜像**：VLAN 镜像是以 VLAN 为对象的镜像，即将交换机上某个 VLAN 的所有流量全部镜像到镜像目的端口，具体实现可以通过 ACL 规则匹配 VLAN ID 来完成；这里需要提及的是，以上三种镜像的源端口和目的端口需要在一个交换机内部，且镜像出的数据在镜像目的端口会受 VLAN 出方向过滤的影响。

4. **RSPAN**：这种技术主要是将镜像的源端口和目的端口分布在不同的交换机上，方便网管人员对多远程设备的管理，除了镜像源端口和目的端口外，还需要配置源交换机、中间交换机和目的交换机；在源交换机上，将镜像数据转发给中间交换机的端口可以固定也可以不固定，如果不固定需要配置反射端口（Reflector Port）并将反射端口设置成 Loopback 模式，这样配置比较灵活；为了防止受 VLAN 过滤的影响，镜像出的数据流量的转发时流经的各个交换机端口需要被设置为 Trunk 模式。

5. **ERSPAN**：前面所述的几种 Mirror 方式都是限制在二层网络的范围内，如果需要将 Mirror 报文数据传输到跨三层网络的远端流量分析仪，就需要用到 ERSPAN；这种技术是将 Mirror 出的报文封装在 GRE 隧道里，然后再经过三层转发跨越三层网络到目的地。

**6. sFlow:** sFlow (RFC 3176) 是一个使用端口镜像进行旁路探测和监控的技术标准, 主要用于将网络流量的统计信息进行格式标准化。配置步骤包括根据用户监控需求操作交换机或路由器等网络设备进行配置, 将被监控数据流进行分类和统计并形成流记录, 然后发送到网络监控流量分析仪, 并对流记录进行存储、分析实现网络监控目的。与 sFlow 类似的监控协议或厂家私有方案还有很多, 包括思科的 NetFlow (从出现时间来看是先有 NetFlow, 之后才提出 sFlow 标准的, 其中 NetFlow 的 V10 版称为 IPFIX, 即 IP Flow Information Export, 相关 RFC 是 RFC3917 和 RFC3955)、Juniper 的 cFlow 和 H3C 的 NetStream 等。

部署了良好的网络流量监控技术后, 就可以通过模式识别、神经网络或 1.17 节的 DPI (Deep Packet Inspection) 等技术对网络流量进行分析和识别, 并为后续的计费、统计和网络安全技术做准备。

为了解决安全性访问问题, 即防止外面的入侵以及内部病毒传播等安全行为, 在信息安全领域引入了防火墙 (Firewall) 的概念; 防火墙处于内部网络和外部网络之间, 通过在硬件或软件商配置一系列规则对报文进行过滤, 让报文根据对规则的匹配结果决定其被处理的行为, 动作是通过或被丢弃。防火墙有网络层防火墙、应用层防火墙之分; 配置方式包括在硬件上配置生效比如交换机上的 ACL, 或在操作系统软件里实现比如 Linux 的 IPtables。另外在交换机中, ACL 底层硬件规则还可以用于 QoS 的分类作用或者策略路由指定报文的下一跳。在交换机上的防火墙规则还可以用于很多的认证功能, 比如 dot1x/webportal 等。

除了对提供服务的网络系统进行入侵获取未授权数据或等破坏数据等网络行为外, 还能基于互联网或其他信息公司的信息系统或数据中心的服务能力发出正常的服务请求, 只是发出请求的量大于网络系统能提供的服务能力的话, 就会

导致正常用户无法得到服务，从而引起拒绝服务攻击，即 DoS；因为单站点发动的拒绝服务攻击一方面在系统通常性能下流量速率有限，另一方面也非常容易被识别出来而被进行防御，所以为了克服通常 DoS 攻击的缺陷而发动的这种攻击往往攻击发动者需要控制大量肉鸡来共同访问同一个提供服务的网络系统，这样就形成了分布式拒绝服务（Distributed Denial of Service, DDoS 攻击）；在数据中心或者其他方式提供网络服务的系统边缘都有防止 DDoS 攻击的流量清洗系统。DDoS 攻击是一种常见的网络攻击方式，为了应对这些攻击，安全系统通常以前文所说的旁路的方式部署，即所有访问流量被镜像一份到流量识别系统，一旦监测到有流量攻击就将所有流量牵引到流量清洗系统里，进行攻击流量清洗后将流量再回注到原来的链路里提供正常客户访问的服务。

另外，为了保证数据的安全通常需要对业务数据进行加密处理，而对认证或检验用的用户的密码进行 MD5 或 SHA-1 之类的信息摘要算法，以实现用户的密码保存不采用明文形式；还有对病毒、蠕虫、木马等网络隐患的识别与处理等内容也是网络安全的一个重要方面；将来的趋势是将网络安全进行智能化及硬件化处理。随着 SDN 和 OpenFlow 技术的发展，网络安全领域必然会有一系列新技术或新方案被提出。

---

## 1.17 LB、CDN 和 DPI

前面 1.13 节也提及，从广义上来讲负载均衡（Loading Balance, LB）也属于 QoS 的内容。负载均衡在交换芯片中的主要功能包括 Lag 口的端口选择、ECMP 的路径选择以及根据 ACL 规则或 OpenFlow 流规则来选择出端口的方式，所以交换芯片只能到 4 层 LB，而在 Linux 里主要实现负载均衡的开源软件有 Linux 虚拟服务器

(Linux Virtual Serve, LVS)、Haproxy 和 Nginx 等, 其中 LVS 仅支持 4 层负载均衡。负载均衡的通用模式是访问某个服务点有很多条路径或者多个服务器提供同一种服务, 做法根据报文的头部信息利用一定的哈希算法算出报文应走的多条服务路径中的一条, 以提高服务能力及链路带宽, 减少服务延迟及服务得不到满足的异常情况。

LVS 是一个虚拟的服务器集群系统, 现在有四种方式实现 IP 负载均衡技术: VS/NAT 技术 (Virtual Server via Network Address Translation)、VS/TUN (Virtual Server via IP Tunneling)、VS/DR (Virtual Server via Direct Routing) 和 FNAT 技术。这四种技术各有优缺点, 比如 VS/DR 的高效率但是不安全。其中 FNAT 是国内大型互联网主要采用的方式, 其实现架构如图 1-20 所示。

在 FNAT 模式中, 交换机通过 OSPF 与多个 DR 模块形成 ECMP 路由, 当有请求报文过来时通过 ECMP 的哈希分发到不同的 DR 上; 报文达到 DR 后再通过 SNAT+DNAT 的方式修改报文的 SIP 和 DIP, 为后续在 DR 里将请求再哈希分给不同的服务器, 以规避两次哈希带来的重叠影响; 虽然单个 DR 大量的 SNAT+DNAT 规则导致了效率的降低, 但多个 DR 不仅抵消了效率的降低还为服务器的集群规模扩大提供了非常有效的方式。

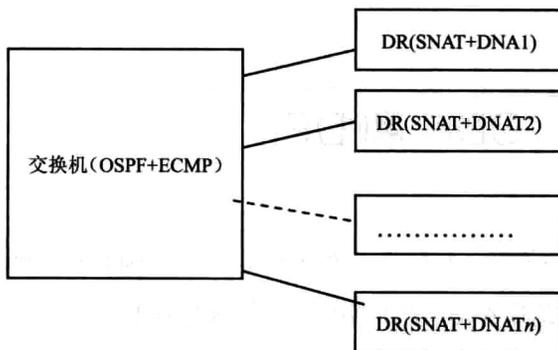


图 1-20 LVS 的 FNAT 架构图

在计算机系统内部，为了加快 CPU 对外部设备的读/写速率，用到了内存的设备来对服务器上的一块数据进行缓存，依据的原理是当某个数据被需要时那么该附近的数据在很大概率上会随之被需要；这个思想还被运用到了 DNS 和 HTTP 上；对于 DNS 来说，当有个域名被访问时，就需要在最近一级的 DNS 服务器上保存该域名对应的 IP 地址，以提高访问网络地址的解析效率；而 HTTP Cache 是对网页数据的缓存，当重复访问某个页面时需要到服务器上去检验是否有新的数据，如果没有就无须下载数据展示缓存中的数据即可，如果有更新则需要更新页面。无论对于内存还是 DNS 或 HTTP 的缓存，更新数据的同步都需要有相应的可靠机制。

内容分发网络（Content Delivery Network, CDN）类似于 HTTP Cache 等机制的大批量数据缓存和同步技术，即当某个用户访问一个网站时，比如淘宝等，那么该用户访问的网站域名就被解析成最近数据中心服务器的地址，这里面需要用到 DNS 的技术，和 DNS 劫持的原理类似；如果最近的服务器有该用户需要的数据则立刻返回，如果没有需要则从核心数据中心的服务器上将数据取回并本地保存然后返回给用户，以后用户或本地其他用户再访问该数据只需从本地服务器获取即可，这样大大提高了访问网站的速率和稳定性，大大降低了核心服务器的压力；而为了让本地数据中心和核心数据中心有尽可能多的一致服务数据，需要通过避开互联网数据高峰的方式来解决数据中心同步的网络瓶颈问题，以减少对数据传输速率和稳定性的影响。

因为跨数据中心的同步在大型互联网公司的 CDN 需求和异地灾备的需求中非常重要，而实现方式往往需要非常昂贵的光纤专线和大带宽等资源，很可能还需要海底光缆的协助实现洲际数据中心的同步。所以在传统的同步方式中，因为不同的业务之间数据需要及时同步但又无法进行有效的流量控制与引导，导致每

年需要花费大量的资金在租用网络带宽等资源上，增加了公司的运营成本，现在 SDN 的出现能够让数据中心的数据同步更加灵活和智能化，最大化地利用网络带宽，节省成本，并且通过 QoS 对数据同步的网络性能有很好的保障。所以随着 CDN 的需求和实施案例越来越多，SDN 的应用领域也越来越多。

深度报文解析 (Deep Packet Inspection, DPI) 是现在网络安全领域的一个技术热点，在识别报文五元组的基础上 (TCP 协议等还需要 session 的概念) 对报文数据内容进行深度检测，而不是传统的仅对报文头部进行分析，通过对报文 1~7 层每一部分内容的解析来分析报文是否具有木马病毒、恶意程序和间谍软件等危险攻击，以保证内网安全不受互联网外界攻击的侵害，常用于的领域有访问控制、QoS 和日志分析等方面。DPI 深度检测为互联网应对外部检测提供了强有力的新工具，尤其是对于检测新的侵害程序提供了强大的方法，但也并不是能应对所有的外部攻击，而且 DPI 技术由于对报文的深入分析可在某些私密场景下使用或者被故意用于分析用户的报文内容，那么可能导致用户的信息会被泄露，从而引起新的安全事故。Broadcom 公司已经在 DPI 方面进行了基于处理芯片内存检查的千兆设备的开发 (<http://www.broadcom.com/products/Security/Deep-Packet-Inspection>)，DPI 的开源软件有 OpenDPI (<http://www.opendpi.org/>) 等。

---

## 1.18 LISP 和 LLDP

在网络技术的发展之路上，LISP 协议可以说是一种革命性的协议。自从网络技术出现以来，尤其是互联网的出现，导致 IPv4 地址被大量使用，宿主机的个数不断上升，加上非聚合路由的使用等原因，互联网核心路由表的规模也越来

越大；伴随而来的流量工程问题、安全性和移动性问题迫切需要有效而统一的解决方案。这些问题的根本原因是因为 IP 地址设计和使用过程总的语义双重性，即包含标识和位置的双重信息；这个问题在实际中本来可以在 IP 地址出现的时候利用 MAC 地址和 IP 地址的形式来解决，但是很不幸当时没有这样来设计，并且在现有网络下再来改造是很困难的。为此 IETF 提出了 RFC6830 的位置标识分离协议（Location-ID Separation Protocol, LISP），来实现网络设备位置和标识的分离，以解决现在遇到的上述问题。

在 LISP 协议中，将原来网络中用于标识和位置的 IP 地址分为两部分 EID（end-identifier）和 RLOC（routing locator），其中 EID 用于标志主机而 RLOC 用于全网路由，并且在标识和位置分离后需要全局的 EID-to-RLOC 的映射来记录两者的对应关系，犹如 IP 地址和 MAC 地址的 ARP 对应关系。这种方案得到包括思科在内的众多厂商和高校研究人员的支持，在 OpenDaylight 中支持基于 LISP 的 NFV 功能。

链路层发现协议（Link Layer Discovery Protocol, IEEE802.1ab, LLDP）是一个用于以太网网络设备（比如无线局域网的接入点 AP、以太网交换机、客户端、路由器等）在本地子网中通告自身标识（比如设备 ID、设备类型和端口号等）和性能的一个与厂商无关的二层标准协议，用于替代各个厂家私有的协议，比如思科的 CDP（Cisco Discovery Protocol）等，可用于网络中确定网络拓扑结构及数据流量的能力。LLDP 采用在链路层发现协议数据单元（Link Layer Discovery Protocol Data Unit, LLDPDU）中以类型/长度/值（Type/Length/Value, TLV）的格式来封装设备的描述信息，包括设备系统名称、设备描述、系统功能及已使能功能、管理地址、电源信息等，端口的信息还可以包括端口的 VLAN ID、名称、速

率、MTU 和双工等；报文封装格式有两种：Ethernet II 和子网访问协议（Subnetwork Access Protocol, SNAP）。运行 LLDP 协议的设备端口有四种模式：Disable（不发送也不接收 LLDP 报文）、TXRX（同时接收发送 LLDP 报文）、TX（只发送 LLDP 报文）和 RX（只接收 LLDP 报文）。在 LLDP 报文中四种 TLV 是必须携带的，包括 Chassis ID TLV、Port ID TLV、TTL TLV 和 End TL，其中通过对 TTL 字段的设置来表明邻居设备的老化时间。总之，LLDP 协议满足了不同厂商设备之间的邻居精确定位。

在 SDN 中 LLDP 协议用于控制器发现其所控制的 OpenFlow 交换机并形成控制层面的网络拓扑，LLDP 的信息被以“packet in”规则转发给控制器，SDN 控制器通过这些信息就可以建立网络拓扑结构的数据库信息，但是这种方案仅限于在支持 LLDP 协议的 OpenFlow 交换机的直连二层范围内才能被支持。

---

## 1.19 网络架构

普通家庭用户通过运营商提供的网络连接到互联网所用的技术称为接入网，最开始是在电话线基础上实现的，这种方式需要利用调制解调器（MODEM，中文常称为“猫”，调制器和解调器英文分别是 Modulator 和 Demodulator）实现模拟信号和数字信号的转换来实现用户到互联网的连接，之后又提出了综合业务数字网（Integrated Services Digital Network, ISDN）、非对称数字用户线路（Asymmetric Digital Subscriber Line, ADSL）、光纤接入 FTTx（比如 FTTH/FTTB/FTTC, Fiber To The Building/ Fiber To The Home/Fiber To The Curb）的无源光纤网络技术（Passive Optical Network, PON），包括基于 ATM 的无源光网络 APON

和基于 Ethernet 的无源光网络 EPON 等，这时的猫称之为“光猫”)、混合光纤同轴网络 (Hybrid Fiber-Coaxial, HFC) 和无线接入网络等。

相比于家庭 PC 直接用非屏蔽双绞线 (Unshielded Twisted Pair, UTP, 两头用 RJ45 水晶头, 一共 8 根线, 布线规则是 1236 线有用, 4578 线闲置) 的便宜和便捷, 数据中心的服务器连接网络设备的线缆则价格要高出很多, 并且布线也是一门非常有讲究的学问, 因为数据中心设备众多, 拓扑复杂, 设备之间的互联如果没有很好的规划与记录, 一旦发生线缆问题需要更换, 从如此多的线缆中找出到底有问题的那一根是非常困难的, 所以数据中心的线缆经常用颜色、线缆头附加标签等手段来区分; 而如果某个线缆商的线缆经常出现问题比如折弯导致的折断、外层保护层的损坏、线缆头部的损伤、线缆误码率的异常等都会给数据中心带来灾难。原来常用的线缆是直接两根捆绑在一起的光纤线缆, 在弯曲程度大时就非常容易折断, 并且和光模块 (Small Form-Factor Pluggable, SFP) 是分开的, 光纤头插拔光模块也非常容易出现困难; 所以现在数据中心使用较多的是直接电缆 (Direct Attach Cable, DAC) 和有源光缆 (Active Optical Cable, AOC) 替代传统光纤。DAC 有两种, 一种是两头均是 10G SFP+, 另外一种是一头是 40G 的 QSFP (Quad Small Form-factor Pluggable) 而对头是 4 根 SFP+, 这种线缆常用于将 40G 的交换机端口流量分到 4 个 10G 的交换机端口上; 因为 AOC 是有源的, 光纤和光模块是一个整体, 便于部署, 而且传输距离误码率要比 DAC 低很多, 支持的传输距离要远一些, 线缆重量相比也轻很多。对于光缆传输来说, 当数据报文进入交换机的时候, 需要变成电信号, 然后交换机再从电信号识别出是 bits1 还是 bits0, 从而生成数据帧进入交换机进行转发流程; 而在发送时, 也需要 SFP 实现电光转换将电信号转成光信号在光纤上传输。

这时候家庭使用网络的模式经常用 C/S 架构 (Client/Server)，网络在层次上也分为接入层 (Access Layer)、汇聚层 (Convergence Layer) 和核心层 (Core Layer)；这种三层架构也被广泛用于运营商网络、企业内部网络和数据中心网络。传统数据中心通常用的三层网络架构如图 1-21 所示，这里有几个概念或俗语需要先解释一下。

- 上行和下行：经常提及的上行和下行分别是指用户端发出数据的方向和接收数据的方向。
- 南北流量和东西流量：在如图 1-21 三层架构上，按照地图上北下南左西右东的规则，各层之间的流量称为南北流量，同层之间的流量称为东西流量；有时候这张图可能被上下倒置，但是南北流量和东西流量的称谓不会改变，旋转 90 度或 270 度的网络拓扑图的画法不是不可以，只是就像地图一样，这种表示方向的做法有违常理而已。
- 收敛比：这个概念是现在网络架构设计的一个重要内容，可用于三层架构中某层或某个交换机上，计算方法是输入带宽与输出带宽的比值，用于表明是否有部分端口阻塞而丢包；在图 1-21 的中，汇聚层的收敛比通常是指汇聚层交换机接收交换机的带宽与汇聚层交换机连接核心交换机的带宽之间的比值。

在数据center里三层架构有着容易扩展业务和安全的优点，但是也有着降低网络性能、扩容成本大、维护成本高的不足，所以现在很多设备商提出了二层架构的方案，二层方案需要核心层端口为 10G 并且在收敛比大的时候也不太适用，但是随着数据中心万兆交换设备成为发展趋势，二层架构已成为数据中心主流架构方式。对于云计算的网络环境来讲，大二层的含义一方面是指二层的隔离域不受 VLAN 4096 个限制，另外一层意思则是相对于数据中心互联和灾备的层面上，同

一个隔离域的地理范围可能分布在两个甚至几个数据中心内，实际上彼此可能距离非常远。大二层的概念也常被说成网络的扁平化，主要是减少服务器网卡的数目，节省成本，在虚拟化环境里实现 VM 更大范围的迁移，实现方便的运维和数据中心的灾备；从理论上来说大二层有多大，虚拟机的迁移才能被迁移多远，因为虚拟机动态迁移后要求 MAC 和 IP 地址不改变，这样才能持续不断地提供业务的服务；但跨数据中心的大二层在现有网络实现的拓扑下，大多是通过隧道的 L2 over L3 技术来实现的。

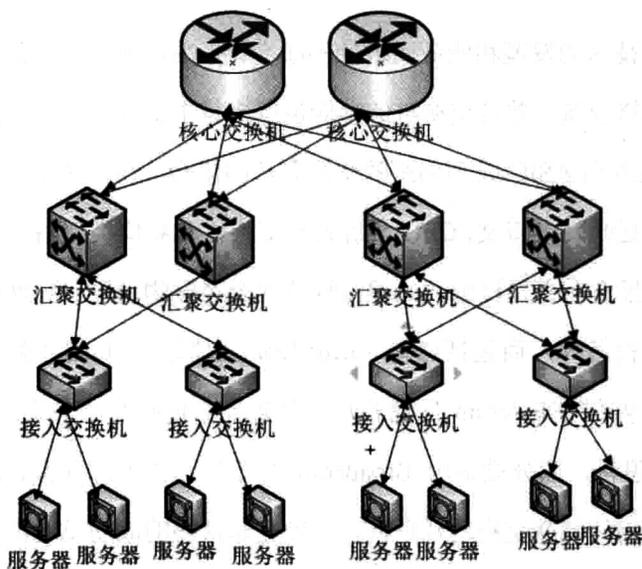


图 1-21 三层网络结构图

现在又新提出了软件定义网络（Software Defined Network, SDN）的概念，这是一种新型的网络架构，包括一个备受争议的基本原则是转发和控制相分离，并需要提供开发的南向接口（Southbound Interface, SBI）和北向接口（Northbound Interface, NBI）。后续的第 4 章会详细介绍这些内容。

## 第 2 章

---

# 以太网交换机

随着网络技术和业务需求的不断变化，在各种网络设备中交换机成为了最常用的网络设备，并且交换机的功能也越来越丰富和完善。交换机硬件是将 CPU 连接或嵌入到 ASIC 或 FPGA 的交换芯片的集合，在这种结构下，交换芯片负责高速低延迟地转发报文，CPU 负责处理交换机和其他类型网络设备之间的协议交互报文。报文在交换机中的处理流程依次要经过物理层、数据链路层和网络层。商用交换机芯片厂商包括盛科、Broadcom（博通）、Intel（英特尔，主要交换芯片产品是从收购 Fulcrum 厂商芯片的基础上，再研发而来的）、Marvell（美满）、Cisco（思科，部分设备用 Broadcom 的芯片）和 华为海思等；其中盛科和 华为海思是中国的国产交换芯片厂商，已经有非常多的成功案例被实施。传统的以太网交换机只能支持解析报文到 TCP 或 UDP 的四层，虽然现在已经有七层交换机的报道，可能这种交换机应用于安全领域较多，所以笔者暂未看到部署案例；本书中通常提及的交换机称呼如果没有特殊情况的数目，通常是指支持三层转发的以太网交换机。本章后续内容将针对报文在交换机上不同层次的功能，详细介绍报文被交换机处理的芯片内部流程。

## 2.1 交换机转发流程

现在三层交换机基本上都用支持三层转发的交换芯片来开发，三层交换机功能丰富，报文处理流程比较复杂，通常还包括隧道、MPLS 等处理环节；所以从处理流程上来说，综合起来交换机的主要功能主要有以下几点。

- 端口接收报文，并形成帧，进行端口统计、MTU 检查，对报文的端口信息进行标注，分配报文缓存用的芯片 Buffer。
- 能够解析端口接收的报文二层、三层和四层报文头部的具体内容，包括每个二层的 MAC 地址及 VLAN Tag、三层的 IP 地址及协议号和 TCP 或 UDP 的源 PORT 和目的 PORT 等字段。
- 接着进行一些防 DoS 攻击的检查，可能支持的有 SMAC=DMAC，DIP=SIP 等。
- 对报文的 VLAN Tag 进行识别和获取，包括基于端口 PVID、基于 QinQ 功能、基于 VLAN Translation 功能等方式来获取 VLAN Tag，用于后续二层转发功能，并在入口进行 VLAN 过滤功能。
- 接着进行 SMAC+VLAN ID 的地址学习，以及依据 DMAC+VLAN ID 的二层转发等处理流程；若 DMAC 是系统的 MAC 地址或三层接口的 MAC 地址，则进行三层转发。
- 如果报文需要走三层转发，则对报文先进行隧道识别和解封装，比如 IPv4-IN-IPv4 的报文，可能需要先剥去外层的隧道头用内部的 DIP 进行查找路由。

- 若报文是 MPLS 需要走 MPLS 的报文处理流程；若是 VPLS 则走 VPLS 的流程；都有需要满足的条件进行判断报文体具体走哪种转发流程；类似的处理流程还包括 MAC-in-MAC 报文、Trill 报文或 Vxlan 报文等。
- 对报文进行 ACL 表项的查找，如果存在报文匹配的规则且 ACL 规则的动作是丢弃动作，那么就需要将报文做丢弃处理。
- 根据报文的 VLAN ID、COS、DSCP 等值进行入口队列映射，以再出口进入不同优先级的转发队列。
- 经过 Buffer 的队列调度算法被转发到出端口，进行出口 VLAN 检查、翻译等功能。
- 根据前面处理流程对报文产生的动作，进行相应的 QoS 字段修改（QoS Remark）。
- 报文再经过出口 ACL 表项的匹配，并执行相应的动作，比如丢弃或 QoS Remark 等。
- 最后报文被转发到交换芯片的出端口，进行出口 VLAN 过滤和 MTU 检查，如果报文可以通过这些检查，报文则转出交换机，经过线缆传输进入下一个网络设备处理流程。

交换机大概的处理流程基本上如上述所说，这里面还有很多细节没有提及，若想进一步学习和理解，只能再参阅相应厂家交换芯片的内部资料。不过需要说明的是不同的交换芯片对上面应用表现的功能基本一致，但是对报文的底层处理流程却有很大区别，这是因为不同的商业交换芯片处理流程细节上有很大的区别。熟悉交换芯片的人，判断和解决交换机的故障问题，无须参考具体交换芯片

驱动资料或平台具体的代码，仅仅根据交换机的功能配置，然后将报文在交换芯片转发流程中对照一遍是如何处理的，就可以确定报文走到哪里被卡住了；因为交换机网络方面的问题，可以大概分为以下几类。

- 交换机下发配置信息报错，或下发配置没有报错但是不生效，这类问题跟踪配置信息下发流程，很容易就可以找到问题根源和解决方法。
- 下发配置没有报错，但是需要被处理的报文并没有按照相应流程被处理，具体体现是该被转发的报文被丢弃了，或者应该被丢弃的报文没有被丢弃，或者报文的某些字段没有被修改成预定的值，也可能是报文没有从指定的路由或出端口转出，这个时候就是考验定位人员对交换芯片及配置下发实现的熟悉程度了；需要根据交换芯片处理流程来具体加以定位。
- 还有一类是无法必然复现的问题或没有找到必然复现步骤的问题，即交换芯片下发配置和使用功能都没有问题，仅仅在某些场景下出现问题或偶尔会触发问题发生，比如全端口无法线速转发、交换机间歇性丢包甚至重启等，这些问题基本上都是设备商需要解决的重大问题，对于研发人员来说可以考虑换新的设备来对比试验，也可能是特定设备的硬件问题可以通过更换硬件组件的方式来解决。

交换机有盒式机和机架的区分，并且放在服务器机架上的盒式交换机通常称之为 TOR。交换机从架构上说，盒式机基本上就是 CPU 加交换芯片，机架则是机架各个插槽里有主控卡和线卡，主控卡相当于盒式机的 CPU 起到控制作用，并且运行着包括路由协议在内的上层系统，多个线卡之间用高带宽的背板 HIG 口和 Crossbar 技术进行无阻塞的互联；一个机架可能有双主控进行主备方式的高可用，线卡主要用于转发业务数据报文，主控卡可能本身也有转发业务数据报文的

处理能力。Juniper 的 QFabric 系统就根据这些架构，将交换机的基本模块组件分离开来，成为 QFabric Node、QFabric Interconnect 和 QFabric Director 三部分，为数据中心的网络设备提供高扩展性。

---

## 2.2 交换机端口处理

交换机的端口主要负责对数据报文的接收和发送，以及必要的分类统计和流控功能。

TCP/IP 网络四层模型中的最底层是网络接口层，该层以比特位为单位传输和接收数据，并以数据帧为单位向数据芯片提供处理单元。网络设备最开始传输数据的速率相比现今的设备是非常慢的，香农提出了信息论的几个基础性概念的定义和定理。然而随着网络需求的新增和新技术的出现，100Gbps 的双工端口标准是最近的趋势。端口有电端口和光端口之分，不同的端口用的传输线缆介质也会有不同，而线缆的价格也是不菲的；好的线缆不仅仅是传输速率的大带宽，还有传输的低误码率、便于运维的线缆信息的标准性、传输性能的稳定性和较长的使用寿命等因素。端口的标准在维基百科（[http://en.wikiPedia.org/wiki/10-gigabit\\_Ethernet](http://en.wikiPedia.org/wiki/10-gigabit_Ethernet)）有基本概念的解释可以参考。不过需要注意的是，交换机端口速率中的 M 就是  $10^6$ ，G 则代表是  $10^9$ ，这里的进制不是 1024，而是 1000。

交换机的每一个端口都有一个网卡，这个网卡由 MAC 和 PHY 组成，现在通常两者是集成在外观上看是一体的一个芯片里。通俗来说，PHY 就是将电信号或者光信号转换后的电信号转换成比特位的零和壹，MAC 层再根据这些比特位来从其中筛选某些比特位组成数据帧，即通常说的一个报文。

相连的两个端口通常还需要支持自协商 (Auto-Negotiation) 或端口速率的设置, 因为可能两个端口支持的最大速率是不同的, 两者需要协商出一个合适的速率进行通信, 比如一个百兆口和十兆口相连, 最终通常自协商的结果是按照 10Mbps 来进行速率传输的, 当然, 这种情况下也可以人为地将百兆口设置成十兆速率的工作模式, 并关闭两端的自协商功能以按照十兆的速率传输数据, 值得一提的是万兆口与千兆或者百兆进行相连时万兆端口无法支持自协商时, 需要人工模式强制万兆口为千兆或者百兆模式才可以工作。

交换机的单物理端口有可能存在带宽不够的情形, 就出现了汇聚端口 (AgGREGate-port) 的概念来解决这个问题, 也称之为 port-channel 或 port group。汇聚端口的主要作用是增加带宽、高可用及负载均衡; 交换芯片在汇聚口的负载均衡和 ECMP 的负载均衡在大部分交换芯片上哈希算法的支持度是一样的, 即在哈希算法种类及哈希算法所能使用的报文字段基本一致。对于三层转发, 还有以 VLAN 为单位建立的三层接口, 每个接口配置一个或多个三层 IP 地址, 以供跨网段转发或接收外来报文给下联的主机。

当两个端口传输数据时, 端口速率被确定后, 如何确定一个数据帧的到来和结束呢? 或者说特定端口速率的端口到底能传输报文的有效 bits 的速率是多少? 其实当发送数据报文的设备将报文封装成 bits 位后在传输介质上传输时, 前面需要有 8B 的前导帧, 前 7B 为 0xAA, 最后 1B 为 0xAB, 对接收设备而言标志着一个帧的开始; 报文结束后还需要加上 4B 的 FCS, 来确保设备接收的报文从 MAC 到数据段的正确性, 此部分算在报文长度之内; 而在传输过程中还有 12B 的帧间隙来对各个数据帧进行物理隔离, 因为现在以太网的传输基本上是 CSMA/CD 方式。并且, 数据报文在被编码成物理比特位的时候, 这样万兆端口在 1 秒内能传输 64B 报文的个数为 (这个速率的计算单位一般为 PPS):

$$10 \times 10^9 / (8 \times (64 + 8 + 12)) = 14880952 \text{PPS}$$

所以从理论计算上也可以证明，当数据报文 MTU 支持得越大时，传输过程中所用的 20Bytes 所占的带宽比例也就越小，这样带宽利用率也就越高；所以整个传输路径如果都能保证开启 Jumbo 功能，且整条链路 MTU 一致，那么某些场景下比如存储网络里等，就可以尽可能高地提高有效带宽利用率来传输长报文的数据内容。所以端口的速率不一定对所有报文传输效率是一致的。

端口另一个重要的功能就是端口统计，统计内容包括端口接收和发送报文分别的总个数、错误帧数、丢弃帧数以及各个长度区间范围内的报文数目，这方面的相关标准有若干个 RFC 来定义，但是交换芯片可以选择其中多种不同的功能项来支持或者有不同的支持程度。报文个数统计比较容易，识别出来数据帧后进行一次累加和即可，比较难以处理的是速率的数据解析和生成。和通常理解的不太一样，无论是何种厂家的传统商用交换芯片，端口的速率并没有用测速的 meter 来进行支持（meter 其实不能用来测速，只能用来限速），而很多是上层通过软件来计算获取的速率值。计算的方法是固定时间间隔内（通常是 5 秒，有的高级交换机可以有配置命令，支持在一定范围内设置该值）利用软件主动读取交换芯片的端口统计计数，然后在特定的时间间隔后再读取端口计数器一次，用前后两次读取的计数差除以这个特定的时间间隔，就获取了该时间间隔内的平均端口速率。当然这样的计算处理方法不是十分的精确，但是基本上是能满足客户的实际需求。

交换机端口另一个重要功能是流控（FLOW CONTROL, FC），其实流控属于宽泛的 QoS 功能；值得提及的是流控功能属于入口的功能，因为流控的产生原因虽然有可能大多是因为出口的数据转发阻塞，比如 HOL，但是却是由于入口接

收的数据帧过多或无法及时转出而导致入口的 Buffer 耗尽产生的, 此时接收报文的交换机入端口就会向发送报文的交换机发送流控帧, 来告诉对方交换机端口降低速率转发甚至停止发送报文到此端口, 等到一段时间后接收报文的交换机端口有可用 Buffer 了, 再给对方发送继续发送报文的的通知, 这样就可以防止通信中有因为阻塞而发生丢包。对交换机测试流控时, 数据帧中的两个值需要注意, 一方面是流控帧的标识是 DMAC 为 01-80-C2-00-00-01、操作符为 1 且 ETHERTYPE 为 0X8808, 另一方面是通知对方停止发包的时间参数, 即在操作参数里的数字一定要被设上合理的值, 而且这个值是以当前传输介质传输速率传 512 位的时间为单位, 范围是 0-65535。需要说明的是流控功能在数据中心很多时候会被关闭, 因为和 POE 功能、STP 协议类似这会使得数据链路的传输效率降低, 而靠网络规划来规避网络阻塞或者特定场景下容忍一定限度的丢包。QoS 在同一个端口上可以针对不同的流基于差异服务模型进行分类处理, 如果一个端口开启了流控, 即使较低优先级的流发生了阻塞, 也会导致较高优先级的数据流报文因为接收端口的流控功能而停止发送报文, 这种情况是不太合理的, 所以提出了 PFC (Priority FC) 的概念, 就是阻塞交换机的入口根据阻塞报文的优先级发送流控帧, 发送端交换机根据 PFC 帧来确定哪种优先级的报文被停止转发, 而其他优先级的报文则可以继续转发。

---

## 2.3 交换机二层转发

通常来说, 物理层的一串比特位信号到数据链路层后, 会被组成帧, 并做检验和的校验——只有检验和通过的数据帧才会被后续处理。二层转发的概念对于交换机来讲就是基于 MAC 地址的转发, 最初的二层转发设备是仅仅基于 MAC

地址的，用于局域网（LAN）内部的主机间通信，后来由于广播泛滥和通信安全性的考虑，采用 VLAN 对不同的通信域进行隔离。虚拟局域网（Virtual Local Area Network, VLAN）是不是网络历史上第一个网络虚拟化的技术笔者无法确定，不过可以确认的 VLAN 是笔者接触到的第一个网络虚拟化技术。

VLAN 技术是通过将原来的报文在 MAC 地址头后加入 4B 的数据来识别和标志 VLAN，具体格式如图 2-1 所示。

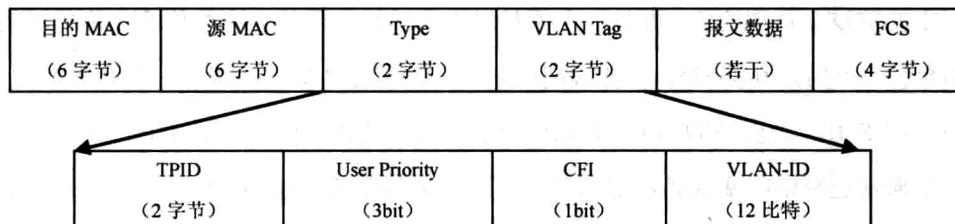


图 2-1 VLAN Tag 格式

其中，TYPE ID（简称 TPID）例子中是 0x8100，其实这个不是必需的，稍微高级点的交换芯片都支持更改该配置为其他值，这个功能在 QinQ 中需要用到；因为处于二层隧道或者 VLAN ID 不够用的需求场景下，可以给一个报文打上两个 VLAN Tag，并且两个 VLAN Tag 的 TPID 可以不同，也可以相同。而对于操作报文能灵活删添或修改 VLAN Tag 的功能称之为灵活 QinQ。

后面 3bits 的 User Priority 用于一般称之为 COS 值，用于二层 QoS 功能，再后面的 1 比特位是前文所说的 CFI（Canonical Format Indicator），为 0 表示该帧格式为用于 802.3 或 EthII 封装，常用于以太网类网络和令牌环类网络之中，如果在以太网端口接收的帧具有 CFI，那么设置为 1，表示该帧不进行转发。

自从 VLAN 技术出现后，二层转发在交换机中便不再是仅仅基于 MAC 地址

的学习和转发，而是基于 DMAC+VLAN ID 的学习和转发，下面结合传统商业交换芯片厂商的处理过程介绍下大致流程，如图 2-2 所示。

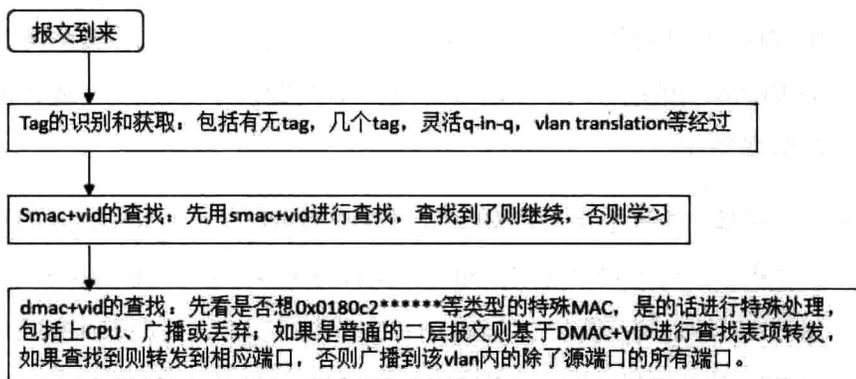


图 2-2 以太网交换机二层报文转发流程

其实二层转发的过程比较简单，需要说明以下几点。

- 对于 SMAC=DMAC 的攻击报文，在高级的交换芯片里可配置直接丢弃。
- VLAN Tag 的识别和获取过程很复杂，后续将详细介绍。
- SMAC+VLAN ID 的地址学习或 DMAC+VLAN ID 的地址查找做法通常可以是正常处理流程，但是某些芯片内部一个没有获取任何 VLAN Tag 的数据报文怎么处理？对这个问题不同芯片处理方式不同，即使是同一个厂商的，有的是丢弃，有的是广播，有的是可灵活配置。
- DMAC 为 0x0180c2 开头的报文主要用于一些二层协议的 BPDU 报文，包括流控、LLDP、LACP、STP 等。
- 交换机芯片通常进行报文单播和广播，默认不开启组播功能，但是从数学值范围的角度讲，单播和广播都可以看作是组播的两个边界值；这和物理

学的很多定理类似，从普通日常生活现象到理论学习推广到相应公式的过程可能非常困难或很难理解，但是实际上经常发生的情形却只是理论公式上极为简单的几种特殊情况；另外，如果 DMAC+VLAN ID 查找不到相应的表项，默认动作是广播该报文，将报文广播到除入端口以外的所有其他同 VLAN 内的端口，通常交换芯片也可以配置该行为成丢弃、或者转发到 CPU 等动作。

- 二层组播的协议主要有 IGMP 等，相应内容可以参考其 RFC，现在主要有三个版本；对于 IPv4 来讲，组播报文目的 MAC 地址以 0x01005E 开始，组播 IP 地址的范围是 224.0.0.0~239.255.255.255；对应于 IPv6 地址中 0XFF/8 的是组播地址，而对应的组播 MAC 地址是 0X3333 开头。IPv4 里二层组播和三层组播的 MAC 和 IP 对应关系如图 2-3 所示。

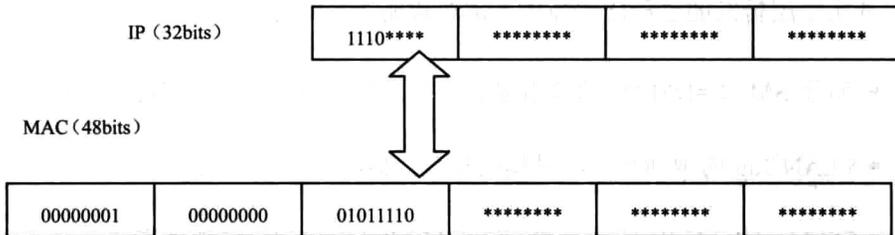


图 2-3 组播报文 MAC 地址和 IP 地址的对应关系

- 二层里报文没有防止成环的字段，这就导致一旦链路有环，将会发生极其灾难的事故，随之提出 STP 协议来解决这个问题，但是 STP 协议随着网络规模的扩大，到 100 台网络设备的规模的时候，一旦网络拓扑发生变化，其响应网络变化的能力就会有很明显的降低；虽然后来提出了 RSTP、MSTP 等改进，但是依然摆脱不了对链路带宽的浪费等缺陷，直至提出了复杂的 Trill 协议该缺陷才得以解决；所以三层报文 IP 头部设计的时候就考虑了 TTL 这个防环的字段。

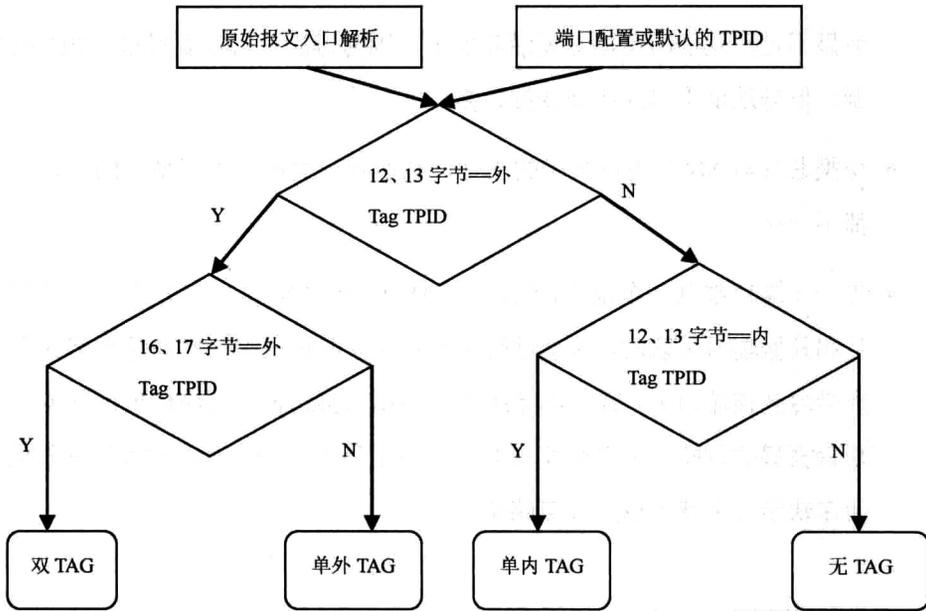
- 每个端口都有自己的出入方向的 MTU，当 CPU 通过端口发包时，在协议栈是可以对报文分片的，这个时候分片的依据是该 VLAN 内的所有端口的最小 MTU 值或 CPU 协议栈的 MSS 值；需要明确的是交换芯片本身并不支持报文的分片和重组功能。
- 为了实现不同 VLAN 的端口能相互通信，引入了端口的 Trunk 属性的概念，可以实现一个端口属于多个 VLAN，从而实现多个 VLAN 间的通信；这个概念是将端口分为三类：Access，Trunk，Hybrid；交换机所有端口默认是 Access 口，此口出入只转发 VLANID 和端口 Native VLAN 相同的报文；Trunk 口时能出入转发所有 VLAN ID 的报文，这里的所有是指交换机上已经创建的 VLAN，但是出去的时候如果报文的 VLAN ID 和 Native VLAN ID 相同，那么则删掉报文的 VLAN Tag 将报文转出去，否则就带 Tag 转发出去；Hybrid 则是可以任意单独配置端口所允许进入的各个 VLAN ID 报文，以及转出报文是否带 Tag；这个功能从数学上来讲和组播类似，可以将 Access 和 Trunk 看成是 Hybrid 的两个边界值。
- 所有端口默认所属的 VLAN 是 VLAN 1，该属性称为 PVID，交换机支持对端口的所属 VLAN 进行更改；同样地对于二层 QoS 字段 COS 值，那些没有 Tag 的报文，给其打 Tag 的时候赋予什么样的 COS 值也是可以配置的。
- 高级的交换芯片为了实现 VLAN 内部用户的隔离，提出了私有 VLAN 的概念，即 PVALN，现在仅有部分交换芯片支持；具体信息可以参见 1.4 节中的介绍。
- 同 QinQ 一样，MAC 地址也有 MAC-in-MAC 的用途，又称 PBB，技术标准是 IEEE802.1ah；具体详见前文 1.5 节的介绍。

- MAC 地址学习到硬件中的表项，需要通过中断的方式通知 CPU；这样软件中便可以有硬件 MAC 表的副本，便于显示和查询，以及通过 ARP 的概念将 MAC 和 IP 进行关联，决定三层路由表项是否下发硬件。
- MAC 地址都有一个老化的过程，即一段时间内如果没有报文的 SMAC 地址学习也没有报文的 DMAC 地址转发匹配到这个 MAC 表项，那么这个 MAC 地址的表项会在硬件表项中被删除，并通知上层软件；MAC 表项老化时间默认为 300 秒，这个值是可以配置的，且根据芯片支持程度而不同，一般支持范围为 60~1800 秒。

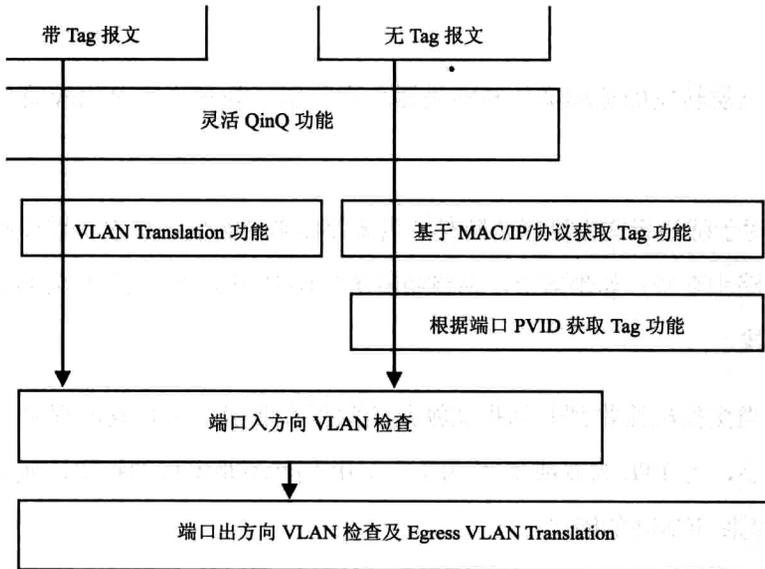
VLAN Tag 的识别和获取是一个复杂的过程，因为涉及很多功能。下面是一个简单的流程，如图 2-4 所示。

图 2-4 里面描述的流程中需要注意以下一些问题。

- 不同厂商的交换芯片处理流程稍有不同，对各种 VLAN 功能的支持程度也不太一样。
- 灵活 QinQ 是通常基于 TCAM 来实现匹配内层或外层的 VLAN ID，设置动作是删除、添加或修改 Tag 字段。
- 传统芯片的二层表项的查找大部分表项是哈希方式，少部分是 TCAM 表项；在现在 OpenFlow 的趋势下，也出现了大量用 TCAM 来做二层转发的交换芯片。
- MAC 表项的学习可以基于硬件，也可以是软学习；硬件学习在成熟的商业芯片可以是完全线速的，但在其他某些不成熟的芯片里，这方面支持度



(a) VLAN Tag 识别流程



(b) 报文 VLAN Tag 获取流程及出入方向过滤

图 2-4 报文 VLAN Tag 识别和获取流程

略显不足；软学习可以更灵活和安全，因为每次学习都要经过 CPU 的控制，但是地址学习速率要变慢很多。

- 交换芯片对 MAC 表项多少的支持，在不同厂商或者不同型号的芯片基本都不一样。
- 当一个端口收到一个报文，但是其 SMAC+VLAN ID 的硬件表项已经存在且和其他端口关联时，这个时候交换芯片需要根据配置判断是否将这个地址学习到该端口上，这个功能称之为 MAC 地址漂移；MAC 地址学习具体是否漂移的判断，大多数交换芯片厂商是基于一些寄存器的设置来做的，而在软学习中就可以通过策略来控制。

---

## 2.4 交换机三层转发

报文三层转发的流程类似与路由器非常类似，处理也相对比较简单，做法如下。

(1) 对于隧道报文先进行判断是否需要终结隧道封装，不需要则用外层 IP 头进行路由查找；如果需要，则终结隧道外层 IP 头，用内层 IP 头部进行后续的路由查找。

(2) 当交换机接收到数据报文帧判断除报文要走三层转发流程时，先检验 TTL 的大小，当 TTL 为 0 或 TTL 为 1 但 DIP 仍然不是本机的报文，就会丢弃该报文并且返回 ICMP 的错误。

(3) 然后根据报文的 SIP 进行可配置性检验，即这些功能是可以开启或者不开启的，这些检验包括当报文 SIP 是组播地址时、是 127/8 的地址范围内的地址

时、单播逆向路径转发 (Unicast Reverse Path Forwarding, URPF)) 检查, 根据检查结果对报文进行丢弃或报文继续被后续流程处理; 其中 URPF 检查可以用于 TCP SYN 攻击或 UDP 及 ICMP 的泛洪攻击。

(4) 如果上述 (1) 和 (2) 中的检查通过, 则进行 DIP 的路由查找, 先是基于全匹配的主机路由查找 (基本是 HASH 方式), 然后是基于 DIP 加掩码的远端路由查找 (基于 TCAM 的查找), 最后则是默认路由或策略路由的查找; 一旦匹配则终止查找流程, 根据查找到的路由来获取出端口等信息。

(5) 根据 (3) 查找到的出端口信息, 来获取报文的 SMAC、DMAC 和 VLAN Tag 的修改, 以及交换机具体出端口等信息; 对于隧道报文需要获取隧道的 ID。

(6) 将报文二层头包括 MAC 和 VLAN 及减少 TTL 等信息进行修改; 重新计算校验和信息。

(7) 根据 (5) 中的隧道信息, 如有需要封装隧道则通过隧道 ID 获取外层隧道头部信息进行重新封装报文, 包括计算新的校验和。

(8) 将封装好的报文通过 (4) 中的出端口信息将报文发送出交换芯片。

这样就完成了报文的根据 IP 地址的三层转发; 当有多条等价路由到达目的地址时, 那么就会形成 ECMP 路由 (Equal-Cost Multi-path Routing Protocol), 报文会根据配置的字段比如二层头部某些字段或三层头部某些字段 (通常是报文五元组 SIP/DIP/PROTOCOL NUMBER/SPORT/DPORT) 进行 HASH 运算, 再对 ECMP 路由数目进行 HASH 运算得到出端口路径。

如果 TCP 或 UDP 的五元组对应的某条业务流带宽非常大, 那么就会形成通

常说的大象流：大象流的危害是很可能超过单端口带宽而导致接收端的 TCP 或 UDP 的重排序，从而引起延等性能急剧下降，还会导致某些小的业务流（称之为老鼠流）因大象流长期占用带宽服务受到影响。现在商用的大多数交换芯片大多数对大象流还没有非常好的解决方案，通常的做法是增大整个链路上所有单端口的带宽。盛科即将推出的新一代交换芯片将支持大象流的识别，大象流识别算法基于参考文献<sup>[23]</sup>；盛科该芯片的代号是 GoldenGate，全万兆端口，芯片带宽 960G，并且在延续现有芯片支持 NVGRE 的基础上增加了对 VXLAN 特性的新支持。

---

## 2.5 交换机 ACL 和 QoS

除了二层转发和三层转发外，主要还有 ACL（Access Control Lists）和 QoS 功能。交换机交换机的 QoS 功能主要是为了保证服务质量，比如为用户提供相应付费的带宽、限制某些用户的最大可用带宽、对报文根据某些字段进行分类进入有优先级的队列、对各个队列进行队列调度、WRED 功能、修改报文中的 QoS 字段以形成整个链路的有机处理流程等。

### 2.5.1 ACL 功能

ACL 功能是交换芯片的一个重要部分，可以用于实现交换机的防火墙、认证（比如 DOT1X 认证、Webportal 认证等，其中 Webportal 是为了实现无法安装客户端的打印机等设备的认证，现在很常用的一种机制如 AAA）、对报文进行细粒度 QoS 分类（以实现限速功能）、计数等功能，ACL 表项匹配后产生的动作有丢弃报文、让报文通过继续转发、修改报文某些字段的数值、指定报文的下一跳、指定报文的出端口（称为重定向功能）、指定报文副本的出端口（称之为基于流的

镜像功能)；ACL 的大体流程如图 2-5 所示。



图 2-5 ACL 功能报文处理流程

ACL 的主要功能是提供一系列规则，让匹配上规则的报文按照规则的动作进行转发或丢弃等操作，从而完成防火墙、报文重定向、报文镜像等功能。ACL 功能对报文的处理流程包括如下步骤。

(1) 对报文头部进行解析，通常交换芯片的解析能力是固定的，传统的交换芯片大多可以解析报文的前 128B 的数据。

(2) 根据报文解析出的各层字段对规则进行匹配，如果报文有匹配上规则则终止该处理流程，否则未能匹配上规则的话则继续匹配下一条。

- 大部分交换芯片都是将 ACL 规则分层若干个组，同一组内的规则顺序匹配。
- 不同组的规则报文是进行并行匹配，每个组最多产生一个匹配中的规则，一条规则可以有多个不冲突的动作；所有组可能产生多个匹配结果。

(3) 收集根据 (2) 中报文匹配到的所有规则的规则动作，以对报文进行处理，如果匹配的所有规则的动作不冲突，则所有动作都执行，如果动作有冲突则按照动作的优先级或规则的优先级进行抉择来选择执行哪个规则的动作。

(4) 当报文被匹配规则的动作标记为丢弃标记时则丢弃报文，实现 ACL 的防火墙功能；通过 ACL 规则还可以实现其他比如重定向、流镜像等功能。

上面描述的处理流程是非常简略的，只是一个大体的流程，而且一方面 ACL

表项对于不同厂商甚至不同型号的芯片实现机制可能都有不同，另一方面不同芯片对于报文能匹配的字段多少、表项的多少、支持修改报文字段的动作类型的支持度都是不同的。但是这部分绝对可以说是 OpenFlow 协议的最初模型。因此将 OpenFlow 用于开发防火墙、QoS 的细粒度分类器、接入端交换机的认证等功能，是很容易支持的，又加上现在交换机可以统一控制，那么 QoS 中综合服务模型里的 RSVP 协议就可以得到非常好的实现，当然这个前提依然是要有足够的带宽资源够分配。

认证功能的通常做法是首先下发一条让所有报文都丢弃的默认规则（这条默认规则使得交换机中默认所有用户都是安全的情况得以改变，使之接近现实中每个用户都可能是攻击者的情况，但是使用的通常拓扑依然不是很灵活），然后当有用户开始使用通信时，首先通过把用户名和密码等信息发给认证服务器鉴别身份，唯有合法用户才能获得认证服务器提供相应级别服务的许可，然后通知网络设备下发一条优先级高于默认规则的规则，让这个用户的相应级别的服务报文可以转发以完成认证和授权动作，并且开始相应的统计和计费功能，其他未有认证的用户依然无法访问相应资源。用 OpenFlow 交换机实现这些功能是轻而易举的事情，而且 OpenFlow 交换机有 ControllerController 的概念，完全可以起到认证服务器的作用，而 OpenFlow 规则实现报文的丢弃和转发动作是再通常不过的规则了。常用的 AAA 协议是 Radius，参见 RFC 2865，RFC 2866。另外还有 HWTACACS（Huawei Terminal Access Controller Access Control System）协议。HWTACACS 是华为对 TACACS 进行了扩展的协议。

## 2.5.2 QoS 功能

在常用的 Diff-Server QoS 机制下，网络整条链路里，每一个交换机上的 QoS 都是按照 PHB（Per Hop Behavior）方式来执行处理的。每个节点的交换机的 QoS

功能在交换芯片里大致分为以下这么几个处理流程：入口映射，内部队列入队，限速和监管，出口队列调度，出口限速和重映射等，不同的芯片流程和支撑的功能类型不太一样，图 2-6 是通常的一种 QoS 芯片处理流程，但不是所有的交换芯片厂商的处理流程都严格按照这个流程的。

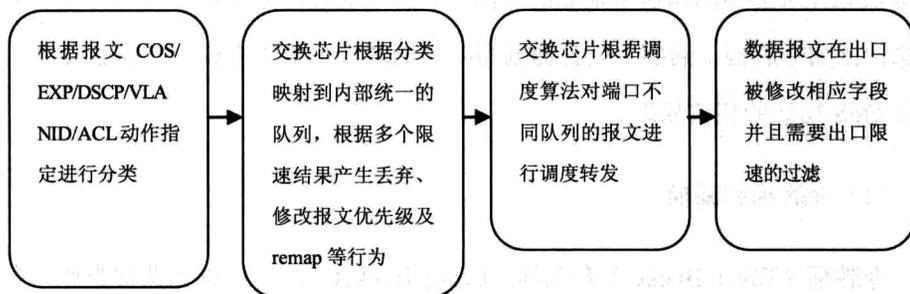


图 2-6 QoS 功能报文处理流程

为了限制广播或组播的风暴，有的交换芯片在端口的出入方向都有对未知单播、组播报文和广播报文的基于 PPS 的风暴抑制功能。用于限速的底层实现器件在交换机芯片中多称之为 **meter**，即限速器，但是该器件的运行返回的结果不是用于测量数据流的速率，因为它的行为是对于超过限制的报文进行丢弃而不是统计；然后为了实现对不同报文进行不同处理，**meter** 利用令牌桶或漏桶等实现对流经的报文进行标记，称之为丢弃优先级，最初分为绿色报文和红色报文，也就是说如果需要丢弃的话会先丢弃红色报文；但是因为硬件 **meter** 对于限制速率是有误差的，而且速率非常大的时候这个误差的绝对值是不小的，为了能让管理员决定对这部分误差的处理机制，将这个误差范围内的称为黄色报文，后来扩展了绿色、黄色和红色报文的速率范围都是可以配置的，以进一步增加限速的灵活性；这样做相当于对报文的转发速率用两个阶梯分了三类，最低的绿色报文，中间的属于黄色报文，最高的属于红色报文；一旦有需要丢弃的时候，和两种颜色区分

的情况类似，红色报文最先被丢弃，黄色次之，绿色报文基本是允许通过的；这种对报文颜色分类的方法有：单速单桶，单速双桶，双速双桶，改进的双速双桶和改进的单速双桶；另外结合 TCP 重传机制，为了避免 TCP 全局同步的发生，引入了 WRED 等功能，减少或避免丢包引起的 TCP 全局同步问题。交换机的端口可以配置风暴抑制和速率限制的功能，下面分别通过交换机常用的几种功能或概念，比如令牌桶、漏桶、风暴抑制和队列调度等，依次来说明交换芯片是如何实现 QoS 功能的相关原理。

### (1) 令牌桶和漏桶

令牌桶 (Token Bucket) 和漏桶 (Leaky Bucket) 的工作机制非常类似，令牌桶常用于流量监管和端口限速，而漏桶机制适用于最大带宽限制和最小带宽保证的功能。

令牌桶有这么几个概念：令牌桶大小 (BUCKETSIZE)、令牌桶刷新闻隔 (T\_REFRESH)、令牌桶刷新速度 (REFRESHCOUNT)、令牌桶内令牌数 (BUCKETCOUNT) 和令牌桶粒度 (GRANULARITY) 等。工作原理大概是：系统在每间隔 T\_REFRESH 的时间往令牌桶一次性地放入了 REFRESHCOUNT 个令牌，每个令牌代表 GRANULARITY 个自己的报文；当有报文过来时，需要按照报文的字节数大小从令牌桶内取对应的令牌数，如果此刻令牌桶内的令牌数 BUCKETCOUNT 比报文所取的令牌数大，则报文被染绿色，否则就是红色；后续有相应的策略分别对红绿报文进行有区分的处理，比如绿色报文通过而红色报文丢弃实现限速功能等。与令牌桶的机制相反，漏桶是报文来了往漏桶里添加相应字节数目的令牌，而系统在每间隔 T\_REFRESH 的时间从漏桶里一次性取出固定数目的令牌数，通过这样做来保证最小带宽和最大带宽限制。

令牌桶还有另外一个配置参数——Burst，其含义是代表允许流量突发的最大值。这个参数值决定了令牌桶本身的大小，为什么要引入这个参数呢？表面上看，对于令牌桶限定的速率，只要系统往令牌桶足够快地添加令牌，到来的数据包都能取得足够的令牌后转发，而不会因取不到令牌被丢弃，但实际上不是这样的。这里要明确几个情况：

第一，假设一个 128B 大小数据报文，特别是按照字节测速的时候，只要无法从限速令牌桶里取到 128 个足够的令牌报文就会被丢弃，就是说恰好令牌桶里只有 127B 的令牌了，报文仍然会被丢弃；

第二，令牌桶的限速是速率值是一个匀速值，然后实际情况除了测试中很少有这样的流，尤其是有限带宽的用户上网时，其流量有很多时候速率很小，比如打开网页后浏览的一段时间内是速率比较低的，但也会有突发流量的时候，比如连续点击打开大量网页尤其是视频几秒后再关闭的时候，用户的流量就会产生如图 2-7 的结果。图 2-7 竖线表示用户的实际所占用的带宽，横轴表示时间。图中虚线表示运营商给用户提供的网络带宽。当用户进行流量很少的网络操作时，能满足用户需求，这个时候用户的带宽资源并没有全部占用。但是一旦用户有很大的突发流量，比如打开迅雷视频并打开很多网页等待视频，这个速率是大大超过其带宽的，这个时候如果对应的限速令牌桶很小的情况下，一下子就将桶的令牌取尽了，导致在这突发的瞬间很多流量数据包因为取不到令牌要被丢失，这样用户的平均带宽肯定就达不到运营商给用户分配的带宽标准了，相当于用户缴纳了相应的带宽费用，却只能使用平均低于缴费标准的带宽，带来的必然是投诉电话甚至是客户流失。这一点在平常上网时尤其是学校宿舍的网络会有很深的体会——往往开始上网或点击视频开始速率很快，但是网速马上会慢下来，这是因为网络在

之前不用或之前速度很慢的使用情况下，令牌桶的令牌被加满，这时候的突发流量会得到足够多的令牌而通过限速，但如果发送报文的速率一直很大直到桶内的令牌被取尽，网速就会慢下来。

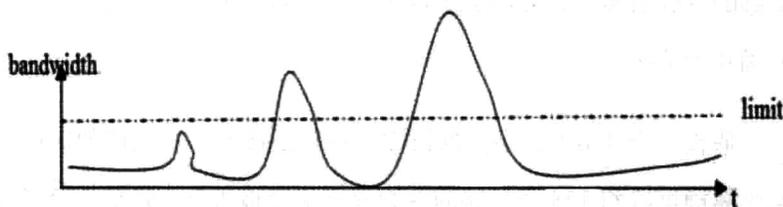


图 2-7 用户流量图

第三，还有一种情况也会在令牌桶不大的情况下链路上经常有单个报文将桶内的令牌耗尽，就是端口 MTU 特别大的时候，超过了令牌桶的大小，虽然数据包速率不是很大，但是包长都超过 4000B，但限速桶的大小只有 2000B，那么每个数据包到来都因取不到足够的令牌被丢弃。这也是研发人员或客户所不能容忍的。所以需要交换机管理员对令牌桶配置 Burst 参数，来规定好令牌桶的突发流量带宽限制，以满足配置需求，即令牌桶的大小最小要等于交换芯片所支持的最大 MTU。

## (2) 令牌桶染色机制

令牌桶的工作原理是令牌桶里会被以一定的特定时间间隔放一个令牌（令牌），直到放满令牌桶为止，后续便开始丢弃多余的令牌不再放进令牌桶；根据用户的配置，一个令牌代表一定的数据包长字节数或数据包数，当数据包到来时就到令牌桶内根据数据包的大小（按字节）等取令牌，取到令牌的数据报文和没有取得令牌的数据报文分别进行不同的处理，比如取到令牌的可以被提高优先级进行优先转发等，没有取到令牌的报文可能被丢弃或被降低转发的优先级。

进一步，在通用惯例上，技术上将取到令牌桶内令牌的报文（in-profile）称为被染成了绿色的报文（简称绿色报文，green），而没有取到令牌桶内令牌的报文（out-profile）为被染成了红色的报文（red）。ACL 规则可分别对绿色报文或红色报文设置相应的 action。在实际应用中，网络流量并不是均匀的，可能会有突发流量的情况，因此引入了向桶中放置令牌的速率称为承诺信息速率，即允许的平均速度，（Committed Information Rate, CIR）和令牌桶的容量称为承诺突发尺寸，即每次突发所允许的最大的流量尺寸，（Committed Burst Size, CBS）两个概念，同时需保证设置的突发尺寸必须大于最大报文长度（即保证 CBS 大于端口的 MTU）。

为了在更复杂的情况实施更灵活的调控策略，仅仅将报文分为绿色报文和红色报文是不够的，所以又引入了通过两个令牌桶（简称 C 桶和 E 桶）实现对报文染成三种颜色（引入另一种颜色-黄色，yellow）的机制，即通过向两个令牌桶放入令牌，当报文到来时，分别从两个令牌桶里取令牌，根据是否取到令牌的结果，判断出报文染成的颜色。这方面有一系列的算法支持：如 single-rate Three-ColorMarker（srTCM, RFC2697），Modified srTCMColor-Blind，Modified srTCMColor-Blind two-rate Three-Color Marker（trTCM, RFC2698），and modified two-rate Three-ColorMarker（modified trTCM, RFC4115）等算法。总之，meter 有这么几种工作机制：DEFAULT mode（报文全是绿色）、Flow Mode（报文只可以被分成红色或绿色）及通过两个 meter 染成三种颜色的上述四种机制。而针对三种颜色染色算法提出了一些新的概念，CIR 和 CBS 对应到 C 桶，相应 E 桶中的称为峰值信息速率（Peak Information Rate, PIR）和超出突发尺寸（Excess Burst Size, EBS）。染成三种颜色的机制基本原理是一致的，即 C 桶中取到报文为绿色报文，如果 C 桶中没有令牌则再到 E 桶中取令牌，取到的话为黄色报文，如果两

桶都没有令牌的报文是红色报文。另外，几种类型的三色机制都可被用于色盲机制（Color-Blind，认为原始报文没有颜色）和有色机制（Color-Aware，根据 DSCP 或 COS 映射到交换机芯片内部的标志，再根据内部标志被识别成不同颜色标记的报文，包括绿色、黄色和红色）。

令牌桶的应用领域主要在流量监管和流量整形方面，入口和出口都可以配置，流量分类主要是基于 ACL 的规则，然后在 ACL 规则中开启具有测速功能的 meter；目的是根据 ACL 规则进行流量分类，符合分类的流量进入令牌桶里取令牌，然后再根据是否取到令牌执行相应的动作，比如流量监管功能中，如果数据包中取到令牌则转发，如果令牌桶中没有令牌即数据包取不到令牌则被丢弃或降低优先级，以保护网络资源不受到损害；流量整形和流量监管的区别是如果数据包取不到令牌则先进行缓存（放到缓存区或队列内，超出缓存时依然会丢弃报文），当令牌桶内有令牌时再发送，以避免不必要的报文丢弃和延迟，使这类报文采用比较均匀的速度向外发送，以便使流量适配下游设备可供的网络带宽等资源，为网络整条链路的流量提供一个稳定的 QoS 服务。

### （3）风暴抑制

风暴抑制功能有三种：广播、组播和未知单播，度量单位是数据包个数/秒，或帧个数/秒，即 PPS，具体则是指端口（包括物理端口或汇聚端口）分别每秒转发三种报文的 PPS 值，主要是为了减少三种报文广播过多时对正常单播业务造成的影响。风暴抑制在 VXLAN 里是解决 BUM 问题的一种技术手段。

### （4）队列调度

报文经过队列映射、入队、流量整形、入口带宽限制等处理，仍然能被转发出的报文在出端口处要进行队列调度，即每个队列的报文根据优先级按照预设的

队列规则或队列调度算法相继被转出端口。端口调度算法分为有队列的调度算法和无队列的调度算法，端口默认的调度算法是无队列调度算法先入先出队列（First Input First Output, FIFO），该算法是将端口的所有报文视作进入了同一队列，根据先到先服务的顺序进行转出；两外，在有队列的调度算法里，同队列的报文也是按照 FIFO 的方式进行转出的。有队列的调度算法大概有以下几种。

- SP（Strict Priority）是严格按照队列优先级，高优先级队列的报文优先转出，只要有高优先级队列的报文时低优先级报文就暂时缓存，当超出缓存时进行丢弃。
- RR（Round Robin）是按照队列轮询调度，调度的对象是每个队列的报文；因为每个队列的报文是大小不固定的，即使是同一队列的报文长度也是不固定的，所以这种方式只能保证报文数目的轮询调度服务，而不是严格意义上的带宽。
- WRR（Weighted Round Robin）是基于权重的调度，这种调度算法的对象也是基于每个队列的报文而不是字节或 bits 数目；具体做法是给每个队列赋值一个权重，然后不同的队列按照赋予的权重进行调度，比如两个队列权重比是 2: 7，那么调度的结果是一个队列每转出 2 个数据报文，另一个队列就会转出 7 个数据报文；当然交换机上有很多队列，需要配置更多的权重值；从调度效果对比上看，RR 是 WRR 所有队列权重为 1 的特殊情形。
- WDRR（Weighted Deficit Round Robin）调度算法并不是所有的设备都支持的，该调度算法克服了 WRR 的缺陷，是针对字节或比特的调度，所以可以做到不同队列之间以严格的带宽比例进行转出报文；配置方法同 WRR 一样，只是调度效果不同；这种调度算法不受报文帧长度不固定的影响，所以更加公平。

另外还有 SP+WRR 和 SP+WDRR 调度算法，这些是前面几种调度算法同时应用在不同的队列上，即高优先级的队列按照严格优先级调度，剩余队列按照 WRR 或 WDRR 的调度算法转出端口。

#### (5) 拥塞避免

在报文进入交换机某个端口后，需要被映射到交换机内部的某个队列；在报文被转发到出端口的队列时，可能因为 HOL 而导致该队列的某些报文被丢弃。默认的特定队列里的报文，是按照尾丢弃（Tail Drop，TD）的方式进行随机丢弃的，这种尾丢弃的方式容易对 TCP 的业务流造成影响，具体体现是大量的 TCP 业务流到来时，因为 TCP 数据量会按照其窗口机制上升，当这些业务数据流带宽超过交换机端口或该队列的带宽时，就会对 TCP 丢包，导致 TCP 业务流的所有发送端重新从最小窗口开始发送数据，这个过程会一直重复，从而造成带宽利用率曲线的波动，也就是带宽利用率不高，这种 TCP 业务不断从最小窗口开始的现象称之为 TCP 全局启动。为了避免 TCP 全局启动的问题，引入了随机早期检测机制（Random Early Detection，RED），RED 机制是为 TCP 的业务流设置两个队列长度门限  $L_{min}$  和  $L_{max}$  以及一个丢弃概率  $P_d$ ，为到来的报文都分配一个随机数，当设备的网络设备中该数据报文队列长度小于  $L_{min}$  时不丢弃报文，当数据报文队列长度在  $L_{min}$  和  $L_{max}$  之间时对该队列的报文根据分配的随机数按照  $P_d$  的概率进行丢弃，如果数据报文的队列进一步增长超过了  $L_{max}$  值，那么就开始采用 TD 方式进行丢弃，这样就能有效地避免 TCP 全局同步问题。但是 RED 的方式灵活性比较差，所以该类别的报文都被采用同样的方式被对待，所以引入了加权随机早期检测（Weighted Random Early Detection，WRED）的控制；WRED 是在 RED 的基础上，结合报文的 DSCP 等字段对该队列的报文区分对待，分别配置不同的丢弃策略，并且可以结合前面介绍的调度算法实现基于流的 WRED。

通过以上几种机制综合协作，才能实现交换机芯片里 QoS 机制的各种功能，达到对网络服务质量的控制或调整。

---

## 2.6 交换机的虚拟化支持

交换机虚拟化的支持包括硬件交换机的虚拟化和软件实现交换机的虚拟化两种方式，即服务器内部网络虚拟化的 Server Virtualization 和硬件形式的 Virtual Interface Switch (VIS)。这两种方案从不同厂家的产品视角来看也不尽相同，无论从何种角度或技术，现在物理网络和虚拟网络都应作为一个整体来部署、管理和为上层服务提供所需的资源。

服务器内的网络虚拟化通常需要相关的软件来支撑，比如用 Linux Bridge 或 Open vSwitch 作为虚拟交换机 (vSwitch)，来完成 VM 与外部网络、VM 与 VM 间的流量交换，这种方案称为 VEB (Virtual Ethernet Bridge)，即为服务器内部的 VM 之间和服务器到物理网络之间提供了第一层软件实现的网络，能为 VM 之间提供本地的桥接或转发服务，使得虚拟化网络的规模变大，且自动完成 VM 迁移后相关变化的网络策略的部署，具有节省接入层物理交换机设备和对外网兼容性好的优势，但所提供的网络性能会受系统配置如 CPU、内存等参数和软件实现方式的限制，另外安全方面及扩展管理、网络流量监控和流量转发控制策略管理等方面也存在不足。软件交换机的虚拟化典型例子是 Open vSwitch 后续将会详细介绍。

后来为解决 VEB 中存在的问题，提出了 EVB (802.1Qbg Edge Virtual Bridging) 的概念，以 VEPA (Virtual Ethernet Port Aggregator) 为实现方案，处理方法是将在 VM 产生的所有网络流量全部交于与其所在物理服务器相连的交换机来完成转发。

VEPA 可以借助硬件实现，也可以通过软件实现。802.1Qbg 还规定了多通道 (Multi-Channel) 技术来让用户选择虚拟网络与外界网络的互通技术方案。

传统的使用场景基本上都是一个物理服务器连接一个交换机的端口，从单个数据报文转发路径来看，报文从服务器内部网络转出到 TOR 或其他接入层网络设备，为了防止环路是绝对不允许再从该 TOR 的入端口转出的；举个例子，在二层交换机广播时，传统的转发流程中是严禁报文从进入交换机的端口再转发出一份的，因为不可能服务器发给交换机的报文后是转给自己的。但是在云计算网络环境下会与传统相比有很大不同；为了提高虚拟网络的转发性能，让更多的服务器资源用于计算，而网络功能或性能问题交予网络设备来解决或实现，可以让原本在服务器内部转发的流量直接转发到 TOR，再让 TOR 来决定报文的走向；这种报文的转发流程有很大改变甚至和以前是冲突的，因为一个物理服务器内可能有多个 VM，如图 2-8 所示当租户 A 的 VM1 要访问在同一个服务器上租户 B 的 VM2 所提供的服务时，报文就会从该服务器所连接的交换机端口进入后再从该端口转发出来。需要物理交换机支持的一种转发模式称之为 RR 反射转发模式，又称为发夹式转发模式 (hair-pin mode)，采用这种模式的方案称之为 VEPA 方案。

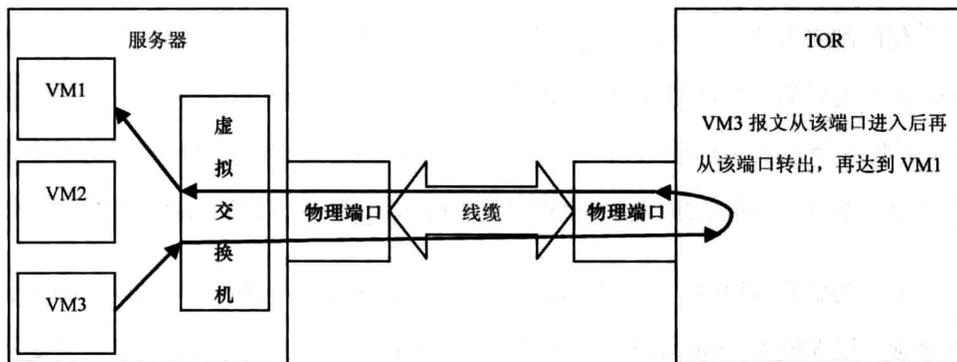


图 2-8 VEPA 中 RR 模式示意图

802.1Qbg 的 Multi-Channel 技术是对 VEPA 的增强, 每个逻辑通道可由用户根据需要定义成 VEB、VEPA 或 Director I/O 的任何一种。Multi-Channel 是将 VM 以逻辑通道的模式连接到物理交换机的虚拟端口上 (Vport), 然后再进行转发, 当报文广播或组播需要复制时由服务器的外部物理网络来实现, 对于 VEPA 来说简化了网络结构, 进一步减少了服务器网络处理的负担, 具体工作原理如图 2-9 所示。802.1Qbg 的 Multi-Channel 则采用了 QinQ 数据封装格式, 外层 S-TAG 表示所连接的本地通道号, 只在服务器和交换机间的网络被识别和生效, 而内层 C-TAG 表示标识 VM 所提供的服务; 这样就减少了客户升级设备的成本, 技术方案也更成熟。将传统的技术应用到新环境是比较常见的网络技术创新模式。VEPA 和 Multi-Channel 都属于 802.1Qbg 标准的技术, 两种方案都需要相应的交换芯片支持才可以实现。

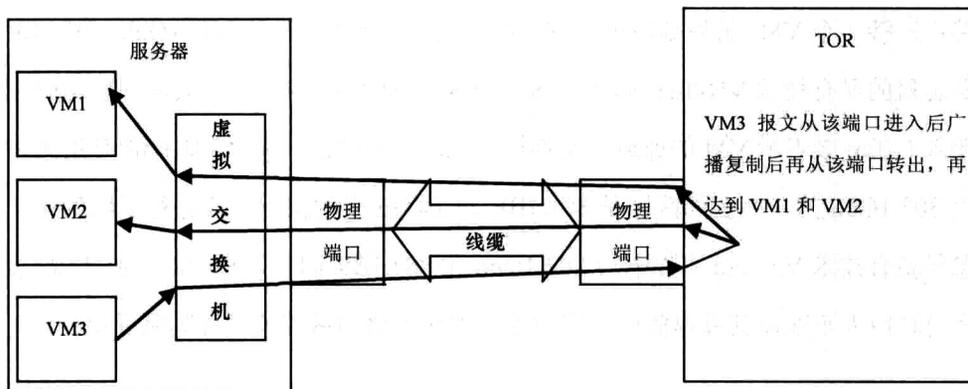


图 2-9 VEPA 中 Multi-Channel 示意图

在网络虚拟化方面, 各家厂商积累的产品不同, 从各家产品向网络虚拟化延伸的角度也不一样, 比如惠普认为应是服务器向网络设备的延伸所以称之为端口扩展 (Port Extender, PE) 技术, 而一直从事网络设备研发的思科则根据网络设

备向服务器扩展且能配置管理服务器的虚拟网络的思维称之为 FE (Fabric Extender), 但是对用户来讲都是同类的虚拟网络和物理网络互通的技术。硬件交换机的虚拟化取决于交换芯片的支持程度, 这里的交换机基本上是指连接服务器机架的 TOR 或边缘交换机 (Edge Switch)。虚拟化技术与交换机的发展紧密相连, 在传统的交换机功能里比如 VLAN 功能将同一个物理区域虚拟成多个广播域的过程, 是典型的端口一虚多技术; 同一个交换机的链路汇聚 (LinkAggregation) 是典型的端口多虚一技术, 而且思科提出了不同交换机之间端口的汇聚技术 (Virtual Port Channel, vPC, 具体见参考文献<sup>[19]</sup>)。随着网络技术的进一步发展, 尤其是云计算技术的出现, 对交换机的使用场景提出了更多新的需求。为了支持这些网络虚拟化的功能交换芯片提出了网络端口虚拟化 (Network Interface Virtualization, NIV) 功能, 将交换芯片的一个端口虚拟化为多个端口, 每个虚拟端口连接一个 VM。另外端口扩展方面相关的标准或方案还有 802.1Qbh、802.1BR 及思科的私有技术 VN-Tag 等, 其中 802.1BR 和 802.1Qbh 都在以太网帧的基础上插入 E-Tag 来表示 VM 的通道与交换机虚拟端口的映射, 并且 802.1BR 是用来替代 802.1Qbh 的, 两者的区别是 802.1BR 对 E-Tag 里的保留字段进行了明确化; 思科私有技术 VN-Tag 可以兼容 802.1Qbh 和 802.1BR 同时有相应的产品实现, 总之思科巨人正在以其深厚的研发实力在虚拟化网络领域继续发挥其领导地位。

网络虚拟化在实际应用领域里, 苏州盛科的设备里有一种类似于 VEPA 的被称之为 Offload 的方案, 已经被一些云计算公司所使用, 比如 Ulcloud 等来提升网络的性能。在网络虚拟化技术盛行的今天, 尤其在云计算应用场景下, 网络交换芯片商和网络设备商正将其网络控制领域延伸到虚拟网络、存储、计算等领域, 而传统的系统厂商依据其强大的系统研发经验积累想将其业务以虚拟网络产品

的形式延伸到网络领域，所以现在可以看到虚拟网络无论是网络设备商还是系统商，尤其是在网卡和 CPU 卓有建树的 Intel 更是志在必得地加快在虚拟网络领域的整体布局，以加快公司的业务拓展，通过收购交换芯片公司 Fulcrum，整合资源提供网络和系统一体机 ServerSwitch 方案，抢占数据中心设备和云计算方案的市场份额。网络设备商提供的功能必须基于交换芯片厂商提供交换芯片硬件所支持的功能，才可能为客户提供功能的形式有所创新，而交换芯片商博通在继续占据老大地位的同时，还不断创新网络设备的新技术，也正逐步走向需要和 Intel 对决的云计算领域。无论各家设备商采用何种标准都需要进行互通性测试工作，但是对于使用的客户来讲，使用新的技术就需要升级新的设备或系统，甚至需要购买新的硬件，以便换取所需的功能或性能。

---

## 2.7 交换机的 CPU

类比于一个人体来讲，交换机的 CPU 是交换机的大脑，交换芯片可以说是四肢；交换机的 CPU 通常来说可以是 ARM、PowerPC 和 MIPS 等类型，所以性能不是非常强大，现在交换机设备商慢慢开始使用 X86，这也包括各家厂商的存储、计算和网络集成在一起的一体机硬件设备；交换芯片和 CPU 的通道在交换芯片上的端口俗称为 CPU 口，CPU 口是交换芯片将协议报文送至 CPU 的通道，基本上是 DMA 方式，并且由于交换机的 CPU 性能不是很强大，处理协议报文的速率相对于交换机 CPU 口的硬件转发速度通常相差几个数量级，所以对报文上 CPU 的速率进行限制，不同协议之间报文上 CPU 的相互影响也需要加以考虑；这点在现在大规模 OSPF 部署时就存在一个很大的制约，因为 OSPF 报文在

固定间隔同时广播 LSA 需要转发到交换机 CPU 被处理时，如果得不到及时合理的处理，就会造成协议上的不一致性而导致不可预知的问题。其实交换机上可以看出是每个交换机都有一个控制端，然后为了实现多个交换机的协同工作，交换机 CPU 作为控制端进行交互信息；交换机的产品最初是普通的盒式机，即一个 CPU 控制一两个交换芯片进行报文转发；后来发展成多个交换芯片集合在一起被交换机 CPU 控制的机框式结构，CPU 称为主控卡，来控制几个甚至更多的交换芯片进行业务流转发，主控卡上也可以支持数据报文的转发；为了让主控卡工作更稳定，往往有备用主控卡，就是一个机架上有两个或多个主控卡，当 **master** 主控卡异常或者挂掉的时候，备用主控卡可以立即起作用，保证整个机架的使用正常。

盒式交换机与机框式结构设备相比，有一个很重要的缺点就是当端口数目不足时无法扩展。为了在盒式交换机上实现机架的这种组合效应，增加总带宽及均衡和管理能力，提出了对盒式交换机的堆叠；堆叠的交换机一般彼此间的物理位置都比较近，需要一个类似于 OSPF 的广播网络类型选举过程来产生主备交换机和成员等角色的一个流程，但是现在不同厂商的交换机往往是无法一起堆叠的；组成堆叠的设备扩展了端口后，只需要连接到 **master switch** 上，就可以通过 **telnet/web/snmp** 各种方式对交换机的堆叠系统进行统一的管理。还有一个概念是虚拟堆叠，其实质是通过快速以太网（Fast Ethernet，100Mbps）端口或千兆以太网（Giga-Ethernet）端口进行的级联，实现成本低，便于设备的网络管理。

从堆叠和机架产品架构可以看出，控制和转发相分离不是 SDN 架构新提出的，也不是 SDN 的核心思想，只是其一个争议较大的原则，做到控制和转发相分离的不一定是 SDN，而没有将转发和控制相分离的也不一定就不是 SDN 的架

构，即转发和控制是否相分离不是判断该网络方案或产品是不是属于 SDN 架构的根本标准，因为 SDN 还需要标准开放的南北向接口和反馈机制。

现在有些交换芯片厂商比如 Intel 等为了解决交换芯片不能 NAT 等不足，正在实施结合强大的 CPU 比如 X86 加交换芯片的集成工作，这种设备通常称为 ServerSwitch，这样 CPU 将进行 NAT 等交换芯片不擅长的处理工作，而交换芯片将进行大流量数据的转发工作；交换芯片也可以用 FPGA 来代替，以让客户定制开发自己的应用；笔者认为后续这类网络产品在云计算里将会有非常好的应用前景。

## 第 3 章

# Linux 网络基础

对于在数据中心内部向外界提供业务的服务器现在基本都是运行在 Linux 系统上，所以对于服务器的网络协议栈，本章将拿开源操作系统 Linux 的实现作为例子阐述网络协议栈实现的就几个主要模块，而且 Linux 的 TCP/IP 协议栈从网络各个层次上也都有对应的相应功能来支持，包括网卡收发包驱动、Linux Bridge、IPtables、QoS、IP 转发等。

Linux 操作系统诞生于 1991 年，创始者是里纳斯·托瓦兹（Linus Torvalds）在赫尔辛基大学上学时出于个人爱好而开始编写 Linux 内核源代码的；Linux 操作系统是开源的类 UNIX 操作系统，支持多用户、多进程和多 CPU。在 Linux 系统里的 TCP/IP 协议栈源代码是一套经典的网络协议实现，Linux 的源码中网络相关部分主要有网卡收发包驱动模块、虚拟设备 TUN/TAP、Linux Bridge、TCP/IP 协议栈部分、IPtables、QoS 等几个模块。从 Linux 内核的 3.3 版本开始 Open vSwitch 也是属于 Linux 操作系统网络相关部分的一个处理模块，其相关内容在后续第 4 章内进行介绍；另外 Linux 操作系统还有包括 Dnsmasq 在内的很多开源的网络相关服务。Linux 内核网络相关的报文处理模块很多是和系统实现相关的，包括进程线程、进程调度、页读写、I/O 性能、中断机制等，这些可能并不完全是网

络标准或通用做法相关的技术内容，所以 Linux 协议栈源代码这部分的学习和使用需要系统方面的经验和知识积累。

---

## 3.1 网卡和数据包的收发

网卡 (Network Interface Card, NIC) 也叫网络适配器，虚拟网络缩写为 vNIC (virtual NIC)；网卡根据隶属总线类型分为 PCI 网卡、USB 网卡、PCMCIA 网卡和 ISA 网卡等类型，按照带宽分为 10M、100M、1000M 和万兆网卡等种类，大部分网卡有端口自协商的功能 (Auto Negotiation)，即设备双方的联接端口根据彼此速率、双工等信息共同协商，以达到双方工作在合适的连接速度和双工模式的目的。网卡的几个知名制造商包括 Marvell、Broadcom、Intel、Realtek 等，这些信息一般在物理网卡的厂商 ID、设备 ID 等信息里都有记录。网卡在服务器启动的时候，需要注册并被挂载到 Linux 的内核上，之后被 Linux 内核进行初始化，这样网卡才能在收发包时与内核进行通信，这个挂载的过程是通过 Linux 内核的网卡驱动程序的主动探测和识别过程来实现的。

任何从服务器外部进入服务器内部的报文都要从物理网卡处传输进来，网卡接收到数据帧后放入缓存中，以中断的方式通知 CPU 或 CPU 采取轮询的方式读取网卡接收到的数据帧数据，驱动程序将数据包缓存数据插入到接收队列，然后 Linux 内核协议栈收包程序依次从收包队列处读取报文，并通过软中断将报文内容作为信息传送到协议栈进行后续的处理流程；当 CPU 向网卡发送数据时，也是以软中断的方式传递给发包队列，然后网卡驱动程序将数据包通过物理网卡将数据报文发送出服务器。网卡同任何其他 Linux 硬件设备一样，是需要与之匹配

的驱动程序才能让 CPU 控制与使用的网络设备；所以服务器的网卡在能被正常使用前需要从芯片厂商官方网站下载网卡驱动程序，并可以直接安装或者使用第三方驱动工具安装驱动到服务器内核里。也就是说，每种新的网卡也都需要在 Linux 内核的源代码中有对应的网卡驱动才能在 Linux 的服务器里工作。Linux 系统的设备分为字符设备 (char device)，块设备 (block device) 和网络设备 (network device) 三种。由于网络设备种类繁多，所以网络设备驱动程序是在 Linux 内核里源代码需要不断更新，才能跟进并支持新的网络设备的使用。以前大部分的网卡收包都是单队列的，通过过滤器再分成很多类进行不同的处理，现在多队列网卡已经开始被使用，也提供了更多 QoS 和收发包方面的新功能。

### 3.1.1 Linux 网卡收包流程

如前文所述，Linux 系统服务器的网卡对数据报文的收和发送依赖于 Linux 网络设备驱动程序完成，即物理网卡通过驱动程序将接收到的数据帧传递给 Linux 内核协议栈，而协议栈也是用网络设备驱动程序将封装好的数据帧发送出去。Linux 几乎将所有对象都当作文件来对待，虽然网络应用程序也是用套接字 (Socket) 并且也有 read 和 write 等操作函数，但是网络设备驱动程序与字符设备、块设备的读/写函数相比有非常大的区别。另外，从数据流来说，字符设备和块设备多是 Linux 系统内部数据的交互，而网络设备则多是 Linux 系统与其他系统的交互。

网卡接收报文时会根据网卡被设置得不同模式及报文的目的 MAC 地址来判断是否接收该报文；网卡的模式有混杂模式 (Promiscuous Model)、直接模式 (Direct Model)、组播模式 (MultiCast Model) 和广播模式 (BroadCast Model) 四种。直接模式是网卡只接收目的 MAC 是网卡自身 MAC 的数据帧的工作模式，组播模式是网卡接收所有的组播报文数据帧的工作模式，而广播模式网卡可以接

收广播帧的工作模式；混杂模式是网卡可以接收所有任意目的 MAC 报文的工作模式，即无论报文的目 MAC 是否是自身网卡的地址，无论报文是单播还是非单播。通常情况下网卡正常工作时，只接收目的 MAC 为网卡自身地址的数据帧和广播的数据帧，以及自己所在组播组的组播报文。需要注意的是，当网卡在 Windows 操作系统下用 Wairshark 工具抓包或者在 Linux 操作系统用 tcpdump 命令抓包时，相应网卡都会被设置成混杂模式；当网卡加入到 Open vSwitch 里作为其一个端口时，也会被设置成混杂模式。

当物理网卡收到数据帧时，可以使用中断或轮询的方式查收报文；通常情况下因为数据量速率不是很稳定，所以中断的收包方式的效率相对较高；但是当有大量数据报文到来时，轮询（NAPI, New API）的方式查询浪费的时间就比较少，接收报文的效率就会大大提高，从而网卡收包的性能也相应会有很大的提高；因此采用何种方式收包要视业务具体场景的情况而定。Linux 网络设备驱动程序不仅可以处理网络层 IP 报文或纯以太网格式的二层报文，还能处理 IPX 网络协议或令牌环格式的数据帧。物理网卡为提高网协议栈对报文的收发等处理效率，提供了一系列如下有意思的功能：TSO、UFO、GSO、LRO、GRO 和 RSS。

整个网络系统中，包括服务器的物理端口和交换机的端口，都有一个对报文的可支持的最长字节数的限制，即 MTU（Maximum Transmission Unit）的大小；当网络端口接收到一个超过其 MTU 大小的数据帧时，网卡就会丢弃该报文并在丢弃报文计数器里增加计数。在 TCP 报文里 MTU 的值大小，基本上是 MSS（Maxitum Segment Size，数据包每次能够传输的最大数据分段）的值加上报文各层报文头部的字节数和尾部 CRC 校验和的字节数。

而对 Linux 系统的服务器来讲，报文的接收 Buffer 大小可以说从 8K 到 64K 之间，这个范围内的值都可以采用。那么如果端口 MTU 太小，一方面某次业务报文分片就会很多，而接收端的服务器就需要很多的中断次数 Linux 协议栈才能完全接收到这些报文，这就使得 Linux 系统的收包性能有所下降；相同的，发包时的过程亦是如此；另一方面，报文分片过多时，经过协议栈处理的次数就会多，也会导致协议栈性能下降。如果能将这些分片报文在物理网卡处合并或者在协议栈查找路由前进行合并，那么协议栈对报文的处理效率必定将会大大提高。所以网卡收发包出现了以下这些技术。

- LSO (Large Segment Offload) 是一种通过网卡对 TCP 或 UDP 的发送报文进行分片的技术，需要硬件支持检验和计算；TCP 的数据报文称之为 TSO (TCP Segmentation Offload)，UDP 的对应功能称之为 UFO (UDP Fragment Offload)。
- LRO (Large Receive Offload) 和 LSO 相似，L 服务器的网卡接收报文时，在网卡处对 TCP 或 UDP 将分片报文进行组合，然后再统一传送到协议栈，以提高协议栈的处理性能。
- GSO (Generic Segmentation Offload) 比 LSO 更通用，针对任意协议，这点只是内核特性，方法是发送报文时尽可能在协议栈减少对报文的分片，而是只经过协议栈一次处理后，如果网卡支持 GSO，则直接在数据报文的发送网卡处进行分片，否则协议栈完成数据报文的分片。
- GRO (Generic Receive Offload) 功能也是对协议栈报文处理的一种减负处理方式。它不限于 TCP/IP 协议，同 GSO 一样也是内核的特性，比 LRO 更通用，但 GRO 功能只是针对 NAPI 类型的驱动。

在支持多队列网卡里，可以针对 CPU 单核或多核、多中断和多队列之间寻找某种映射，通过挖掘 CPU 的剩余资源来提高 Linux 系统对报文的收发处理性能；现有的一些技术已经可以使用，比如 RSS（Receive Side Scaling）是一项将具备多队列网卡的新特性，能够将数据报文根据队列分别分配到不同的 CPU 核上去处理，并且保证每个流的报文都会被通过计算哈希值（比如利用 Toeplitz 算法对报文四元组源 IP、目的 IP、源端口和目的端口进行哈希）的方式来被分到同一个队列，以保证同一条流的报文会被同一个 CPU 处理，减少 CPU 的 Cache 抖动；在硬件上 RSS 的实现通常对应一个 entry 组，每个 entry 对应网卡的一个队列。RPS（Receive Packet Steering）和 RFS（Receive Flow Steering）是 RSS 的两种软件相应实现，由 Google 贡献，分别是针对报文级别和数据流级别的软件实现。RPS 在内核的接收报文处理流程中和 RSS 的描述基本一致，具体是将 RPS 选择队列的过程实现放在了收包中断的后半部处理里，报文根据哈希计算的值来选择被放入的队列，然后这些队列具体被绑定到哪个 CPU 去处理，这样做是因为即使没有增加收包中断的次数，也能被任意网卡使用，并且软件实现中可以很容易增加对新协议选择队列的策略。RFS 通常情况下对接收报文处理流程与传统的收包方式没有太大区别，但当某个应用程序有多个线程，且这些线程分布在不同的 CPU 上执行时，就会给这个应用程序上的报文如何分配到这些 CPU 上的处理机制带来问题，因为一旦同一个流的报文在不同 CPU 上来回切换引起的 CPU 抖动，并会导致性能下降；为了解决 RFS 的这个问题，引入了两个表：rps\_sock\_flow\_table 和 rps\_dev\_flow\_table，在网卡处对 TCP 或 UDP 流来进行处理，rps\_sock\_flow\_table 表用以全局记录该报文所属的流期望到哪个 CPU 上被处理，而 rps\_dev\_flow\_table 表用来记录这个包所属的数据流正被哪个 CPU 处理。另外还有两个相关的 CPU 称谓：期望 CPU（Desired CPU）和当前 CPU（Current CPU），

期望 CPU 表示数据报文到来时根据计算应该处理该报文的 CPU，而当前 CPU 表示处理过该报文所属数据流的 CPU，分别从前两个表查询得出；工作流程是数据报文到了网卡后，根据报文计算哈希值查询 `rps_sock_flow_table` 表得到期望 CPU，然后用这个索引值与 `rps_dev_flow_table` 表的值进行比较，如果 `rps_dev_flow_table` 表中显示该数据流没有被哪个 CPU 处理过或被某个 CPU 处理过但上次处理的报文队列已经完毕，以及表中当前 CPU 有该流报文处理但处于离线状态（Offline）或有数据流被当前 CPU 处理但当前 CPU 正是期望 CPU 时，则该数据流报文被期望 CPU 处理，否则表示当前数据流中，有报文正被与期望 CPU 不同的当前 CPU 处理，为防止报文重排序，该数据流报文应该被送往当前 CPU 进行处理；通过这种方式实现了流级别的数据报文与 CPU 处理之间的绑定关系，大大提高了服务器报文对业务应用程序的处理能力。

在发包方面，提出 XPS（Transmit Packet Steering）是在支持发包多队列网卡上用于智能地选择不同 CPU 的发送报文到不同的队列的功能。多个不同的 CPU 发包和不同队列的对应关系，用一个全局表来记录即可，这样做不仅避免了对不同 CPU 操作不同发包队列锁的使用，并且发包队列的 Cache miss 速率会有显著的下降。

网卡的特性正在日新月异地被推出，可能很多时候在内核协议栈的大量优化会被物理网卡硬件点滴的优化所替代。可以用 `ethtool` 命令来看看端口对这些特性的支持，比如 `ethtool -k eth*` 等。可以用如下类似命令来关闭网卡的上述某个功能，比如 `ethtool -K eth0 tso off`。

### 3.1.2 多网卡 Bonding

Linux 的多网卡 Bonding 和交换机的汇聚端口（Aggregate-port）是同等概念，即将多个网卡虚拟成一个逻辑的端口，该多块网卡共用一个 IP 地址和 MAC

地址,从而实现网卡的高可用和负载均衡功能,只是一个 Bonding 端口能支持的网卡成员数目受 Linux 实现的限制。Bonding 网卡的 MAC 地址如没有特殊配置,默认是从第一个网卡成员的 MAC 地址来获取的,也可以通过 `ifconfig bond0 hardware` 的命令来修改。

网卡 Bonding 有如下七种工作模式。

(1) 设置 `mode=0` 对应的是 Load Balancing Round-Robin 的负载均衡方式,这种模式下两块网卡都工作,传输数据包时依次分配到各个网卡进行传输;该模式不需要对端交换机配合,缺点是可能造成同一个 TCP 流或 UDP 流的数据包被分到不同链路上,造成接收端服务器的数据包重排序问题。

(2) 设置 `mode=1` 对应的是 Fault-tolerance (Active-backup) 方式,该方式提供了网卡冗余备份的能力,并且网卡之间分主备状态且同一时刻只有一个网卡处于主的活动状态,其他网卡处于备的冗余状态,具有较高的冗余性,但是会造成带宽的浪费。

(3) 设置 `mode=2` 对应的是 Balance-xor 平衡策略模式,通过配置不同的哈希策略算出一个相应的值,然后再对网卡数目取模来决定从哪一个网卡转出去,这点和交换芯片的静态配置的汇聚口选择具体出端口的做法非常类似,可以提供负载均衡和高可用的功能。

(4) 设置 `mode=3` 对应的是 Broadcast 为广播策略模式,让每个报文都能从所有端口转发出去,这种模式虽然提供了容错能力,但是配置也会造成通信带宽的浪费,甚至有可能需要接收端服务器进行 TCP 流的去重工作。

(5) 设置 `mode=4` 表示网卡 Bonding 与对端设备端口通过 IEEE 802.3ad

Dynamic Link Aggregation 协议 (IEEE 802.3ad) 建立动态链接聚合。同模式 mode=2, 也可以通过设置哈希策略来改变报文选择出端口的算法; 另外该模式需要每个 Bonding 网卡支持相同的速率和双工模式, 对端交换机和本地 Linux 系统支持 IEEE 802.3ad Dynamic Link Aggregation 协议。

(6) 设置 mode=5 表示网卡 Bonding 工作在 Adaptive Transmit Load Balancing 模式, 即适配器传输负载均衡模式; 该模式根据负载情况分配外出的流量; 这种模式和 mode=2 的平衡策略模式的区别与前文提及的交换芯片调度算法 WRR 和 WDRR 之间的区别非常类似, 一种是基于报文的, 另一种则是基于比特流的。

(7) 设置 mode=6 表示网卡 Bonding 工作在 Adaptive Load Balancing 模式, 即适配器适应性负载均衡模式; 该模式不需要对端交换机的支持, 与 mode=0 的区别是在该模式下流量先都从第一个端口成员转出, 直到第一个超过了第一个端口成员的带宽数据报文才从第二个成员端口转出。

Bonding 网卡的工作模式多样, 具体选择哪种需要根据不同的场合和需求来决定, 并且还要取决于对端设备是否需要配合或支持。需要提及的是 Open vSwitch 的端口也是支持 Bonding 网卡作为其一个端口的, 并且可以和交换机之间进行 LACP 信息交互, 以完成动态绑定的实现, 这点也为网络虚拟化提高网络性能提供了一种方式。

### 3.1.3 SR-IOV

为提高服务器里虚拟机收包报文的性能和可伸缩性, 解决 I/O 虚拟化最后一公里的问题, 提出了基于硬件的 SR-IOV 虚拟化解决方案。SR-IOV 标准允许在虚拟机之间高效共享 PCIe (Peripheral Component Interconnect Express, 快速外设

组件互连)设备,并且它是在硬件中实现的,可以获得能够与本机性能接近的 I/O 性能。

SR-IOV 中的两种功能类型是物理功能 (Physical Function, PF) 和虚拟功能 (Virtual Function, VF), PF 用于支持 SR-IOV 的 PCI 功能,拥有完全配置或控制 PCIe 设备资源的能力;而 VF 是一种轻量级的 PCIe 功能,与 PF 相关联,可以与物理功能以及与同一物理功能关联的其他 VF 共享一个或多个物理资源。

SR-IOV 功能需要硬件和软件都支持时才能使用,并且该功能可以提高性能,节省成本和耗能,简化与网络设备的适配、布线的工作。

#### 3.1.4 DPDK

DPDK (Data Plane Development Kit, <http://www.DPDK.org/>) 是 Intel 在 x86 平台上快速处理报文的一个驱动和库,它工作在 Linux 的用户态,但 DPDK 并不能提供路由、防火墙、IPsec 等网络功能。主要的库包括:

```
Multi-core Framework  
Huge Page Memory  
Ring Buffers  
Poll-mode Drivers
```

DPDK 可被用于用最少的 CPU cycles 来接收和发送报文,可基于其开发类似于 T cpdump 的抓包工具,也可以结合第三方的协议栈对报文进行快速路由或转发处理。其官网的一个性能数据为在两个 PCIe 千兆网卡下达到了 160 MFPS (million frames per second, 64B 报文测试) 的速率。但是在各种系统参数 (多核和 CPU 性能) 和不同的数据流场景 (TCP 短连接长连接、UDP 及其他不同长度的数据包甚至较多分片报文的情况) 下 DPDK 是否能满足开发者的需求,需要具体进行相应的测试才能得到结论。

DPDK 现在已经有不少厂家的产品中正在做集成工作，对于 OpenStack 云计算 Neutron 网络组件来说，亮点在于 Plugin 使用 Open vSwitch 时能把 DPDK 和 Open vSwitch 进行集成（Open vSwitch 的 2.3.0 版中已经支出了 DPDK），以提高 Open vSwitch 的收发包性能，此环境下对 VLAN、NVGRE 和 VXLAN 的隔离技术也需要相应支持。另外，Intel 在其交换芯片和 X86 集成的方案里做 ServerSwitch<sup>[9]</sup>方案，这个估计也希望能集成 DPDK 技术，来加快 X86 收发报文的处理以提高整个系统处理报文的能力。

---

## 3.2 TUN/TAP

TUN/TAP 是 Linux 下的虚拟网络设备，从对报文的处理来讲，TUN 处理 IP 报文的点对点设备，能够处理 IP 数据的封装等；而 TAP 则是工作在第二层处理以太网数据帧的虚拟以太网设备，应用程序需要通过 ioctl 函数设置其虚拟网络设备的工作模式。

TUN/TAP 可以作为内核空间与用户空间交互的一种方式，即内核通过 TUN/TAP 设备向其所属的用户控件的程序发送信息数据，反之用户空间的程序也可以通过 TUN/TAP 设备向内核空间发送数据，这点在内核空间来看像是内核网卡驱动收发报文一样。虚拟网卡 TUN/TAP 驱动也是开源的，Openvpn 即是在其基础之上进行隧道封装的开源项目。在云计算中，虚拟机每一个网卡在宿主机里对应一个 TAP 设备。

## 3.3 Linux Bridge 和 VLAN

Linux Bridge 模拟了物理网络中网桥的概念，即将若干个服务器的端口加入到网桥中，网桥端口对端相连的设备通过发送报文给 Linux Bridge，并通过 Linux Bridge 学习报文 SMAC 和查找报文的 DMAC 转发到相应的目的地，这点和普通的二层交换机非常类似。Linux Bridge 的端口可以是物理网卡端口也可以虚拟的端口。

Linux Bridge 本身是没有 VLAN 功能的，需要 VLAN 模块协作实现 VLAN 过滤的功能。当端口加入到 Linux Bridge 里面时，默认的是不属于任何 VLAN 的或者说是属于所有 VLAN 的，因为能对所有 VLAN 的报文接收和转发；这点和交换机里的 Trunk 口还是有很大区别的，因为交换机里的 Trunk 口是针对 VLAN 的端口属性，需要交换机配置了 VLAN 才能转发报文通过；而 Linux Bridge 没有配置 VLAN 端口就会让所有报文通过。对于配置了 VLAN 属性的端口，则该端口接收报文和发送报文时都会进行相应属性的 VLAN 检查，不同的是，入端口要识别并摘除报文的 VLAN Tag，以将带 VLAN Tag 的报文变成普通报文对上层进行透明传输，而转发出服务器网卡时需要再对数据报文添加上 VLAN Tag，以标记其转发到所属 VLAN 的隔离范围。当有 VLAN Tag 的报文进入服务器被 Linux Bridge 处理时，需要在收包的网卡处对报文的 VLAN Tag 进行识别和过滤，并去除报文的 VLAN Tag 字段，为上层协议栈对报文提供透明的处理方式。

Linux Bridge 上端口的 MAC 地址的相关表项，在端口加入到 Linux Bridge

时自动生成，并且被表项中的 `is_local` 设置成 `yes`，来标记此 MAC 地址是本地地址，当有 DMAC 是此类的 MAC 地址时，需要将报文送交协议栈来处理。当收到 SMAC 是此类 MAC 地址的报文时，会产生错误信息，并将该报文丢弃。在某个 Linux Bridge 上用 `brctl` 命令显示 MAC 表项如图 3-1 所示。



图 3-1 Linux Bridge 的 MAC 地址表项

Linux Bridge 在 OpenStack 的云计算网络中有着非常重要而广泛的应用，Nova-network 即是用 Linux Bridge 和 VLAN 做租户隔离的，即使后来网络相应的 Neutron 组件在用 Open vSwitch 做 Plugin 时也大量用到 Linux Bridge 和 VLAN 的概念，这些在后面的云计算网络中会有详细阐述和分析。

提起 Linux Bridge 就需要提及 veth pair；veth pair 设备类似于一根网线，可以完成两个 network namespace 的数据通信；当多个 network namespace 需要通信时，就需要 Linux Bridge 来协助。

Linux 下创建 Bridge 的常用命令如下。

- `brctl addbr br`：表示创建一个名为 `br` 的 Bridge。
- `brctl addif br eth0`：表示将端口 `eth0` 添加到网桥 `br` 里。

- `ifconfig br 192.168.1.100 up`: 表示为 br 配置 IP 地址为 192.168.1.100 并 UP 起来。
- `brctl show`: 用来对网桥配置的显示。
- `brctl hairpin br port1 {on|off}`: 对 br 的端口 port1 开启或关闭发夹功能。

Linux 的 namespace 也可以被称之为 container, 是一种资源隔离方案, 是基于容器的虚拟化技术的基础, 和 C++里的 namespace 非常类似, 所以 namespace 不是 Linux 里属于网络部分的概念, 而是属于 Linux 系统层次资源管理和使用的技术内容。在 Linux 的系统里调用进程创建函数 clone 设置了 CLONE\_NEWPID 参数后就会创建一个 namespace, 在每个 namespace 里, 如同一个新系统一样, 进程号也从 1 开始创建, 有独立的系统 PID 及 IPC 通信机制, 并且会有其独自的网络环境, 包括协议栈、路由表和防火墙规则等; 但是在不同的 namespace 里这些资源相互之间都是隔离的, 两者之间的信息互通只能通过网络来完成。namespace 这种方案部署起来开销更小且部署非常便捷, 但是这种技术导致内核开发复杂; 与其他的虚拟化技术或云计算平台相比, 不能任意指定想用的操作系统并部署相应的软件。在 Linux 里 namespace 的配置和显示可以通过 ip netns 命令来实现, 比如 ip netns add/delete/list 等; 如果想在对应的 namespace 里进行命令操作可以用 ip netns exec namespace\_name command parameters 类似的命令来操作, 比如查看 qrouter-c6e38a5a-2adf-42a5-8c6f-5eab99208869 命名空间里的 IPtables 规则, 则可以用 ip netns exec qrouter-c6e38a5a-2adf-42a5-8c6f-5eab99208869 iptables-nL 来实现。两个 namespace 下的通信命令举例如下:

```
ip netns add ns0
ip netns add ns1
ip link add type veth //产生 veth0 veth1 一对接口
```

```
ip link set veth0 netns ns0
ip link set veth1 netns ns1
ip netns exec ns0 ip link set veth0 up
ip netns exec ns0 ip address add 10.0.0.2/24 dev veth0
ip netns exec ns1 ip link set veth1 up
ip netns exec ns1 ip address add 10.0.0.3/24 dev veth1
```

执行 `ip nets ecec ns0 ping 10.0.0.3` 或 `ip nets ecec ns1 ping 10.0.0.2` 命令，在各自的 namespace 里可以 Ping 通对方的地址。

谈起 namespace 就需要介绍另一个资源控制和管理的攻击 Cgroups。Cgroups 是 Controller Groups 的缩写，可以通过 Linux 对包括 CPU、系统内存、网络带宽、系统 IO 等方面的系统物理资源进行记录、限制和隔离，以实现在 Linux 系统下进程组数量和进程组之间的优先级、进程组的资源使用与隔离和进程组所有进程的挂起和恢复等目的。Linux 系统的每一个进程都对应 Cgroup 里的一个任务 (Task) 的概念，进程默认都属于 Root Cgroups；当某个进程属于新的 Cgroups 后，其子进程默认也属于整个 Cgroups；某个进程属于哪个 Cgroups 是按照需求根据相应的配置来分别划分的，这个配置下对应的一组进程称之为控制组 (Control Group)；控制组的进程可以被按照树的形式进行组织，树里父节点的所有限制同样对该父节点下的所有子节点有效，这种控制组的控制方式称之为分层 (Hierarchy)；在分层的每一级上可以配置很多控制资源的控制策略，这些策略为了管理和使用的方便进行了分类处理，每一类称之为子系统 (Subsystem)，所以控制组的分层每一级上都可以附加配置一个或多个子系统；通过如此的方式，来实现不同进程组对系统资源的限制使用与隔离。Cgroups 技术同 namespace 一样属于系统层次的内容，不是 Linux 网络的专属技术，这种方式非常类似于网络中的 QoS 模块，对资源进行分类限制，每个分类可以分为子类，且子类中又可以使用控制策略，如同 QoS 的流量被分成了很多的队列进行不同的限速，而队列又被分成了很多子队列，子队列又可以有各自的限速，另外需要说明的是，Cgroups

仅仅是起到了对资源使用的控制和隔离作用，并不是一种增加系统资源或提高资源利用效率的算法或工具；在资源不足的时候通过 Cgroups 可以减少资源的竞争，但是绝不能做到增加资源满足需求。举个例子，当一个物理服务器建立了大量虚拟机的時候，可以通过 Cgroups 来控制系统资源被各个 VM 都能使用，但是一旦 VM 过多会导致系统资源不足，这种技术是无法解决这个问题的，所以 VM 纵然可以得到一些系统资源来使用，但是资源不足的问题依然存在。

---

## 3.4 TCP/IP 协议栈

Linux 协议栈网卡驱动程序通过软中断将报文从网卡送到 Linux 协议栈进行处理；之后在 Linux 内核协议栈里数据报文的大致处理流程是根据报文的类型调用不同的处理函数，主要是以太网 ARP 报文和 IP 报文；但两者的处理流程有很大的区别。

如果是 ARP 报文则首先对 ARP 报文进行解析，并检查协议类型是否是 IP，然后进行解析报文 ARP 格式的各字段信息；接着类似于报文 MAC 地址的学习和转发，然后对该报文所含的 SMAC 和 SIP 进行 ARP 学习，最后根据 ARP 类型是请求或应答进行相应的建立 ARP 新表项或作出 ARP 应答。

而对于 IP 报文，则会将报文分为 TCP、UDP、ICMP 和 IGMP 等几个类型，分别进行不同的处理流程；这几种类型中 TCP 的处理最为烦琐，因为 TCP 是面向连接的，TCP 连接的建立、断开以及需要维护等相关状态迁移处理工作比较多。经过上述的处理步骤后，会将报文或根据接收报文产生的应答报文进行查找路由，包括主机路由、远端路由和默认路由，如果数据报文匹配的是主机路

由，即服务器网卡发送的是同网段的报文，可能还需要 ARP 的查询过程；然后再根据查找到的路由表项进行转发，接着 Linux 系统通过软中断的方式传递数据信息给 Linux 网卡驱动程序，最终 Linux 系统的网卡驱动程序通过物理网卡发送出主机。

socket 接口是 TCP/IP 网络的 API，socket 接口设计者最先是将接口放在 UNIX 操作系统里面的，Linux 系统里实现了一套类似的接口，同样称为 socket 接口。socket 也具备一个类似于打开文档的函数调用 socket ()，该函数返回一个整型的 socket 描述符，随后的连接建立、数据传输等操作都是通过该 socket 实现的。常用的 socket 类型有 SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET，等等。其中 SOCK\_STREAM 是一种面向连接的 socket，针对面向连接的 TCP 服务应用；数据报式 socket 是一种无连接的 socket，对应于无连接的 UDP 服务应用。Socket 的流类型主要有以下几类。

- SOCK\_STREAM=1：表示提供双向连续且可信赖的数据流，即 TCP。
- SOCK\_DGRAM=2：表示为不连续不可信赖的数据包连接。
- SOCK\_RAW=3：表示为原始网络协议提供存取。
- SOCK\_RDM=4：表示为数据包提供可信赖的连接。
- SOCK\_SEQPACKET=5：表示为数据包提供连续可信赖的连接。
- SOCK\_PACKET=10：表示提供和网络驱动程序直接通信方式的连接。

在 Linux 内核中，Socket 通信流程如图 3-2 所示，TCP 和 UDP 的 socket 通信流程分别如图 3-3 和图 3-4 所示。

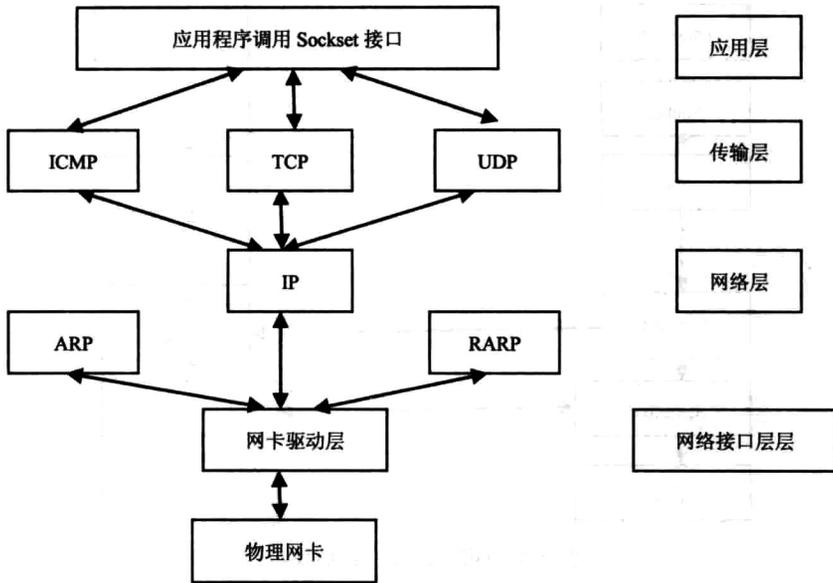


图 3-2 socket 通信流程

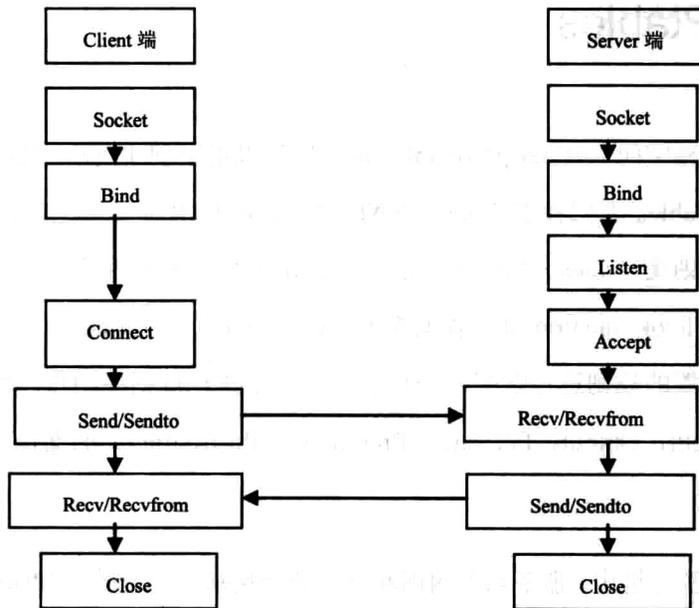


图 3-3 TCP socket 编程流程

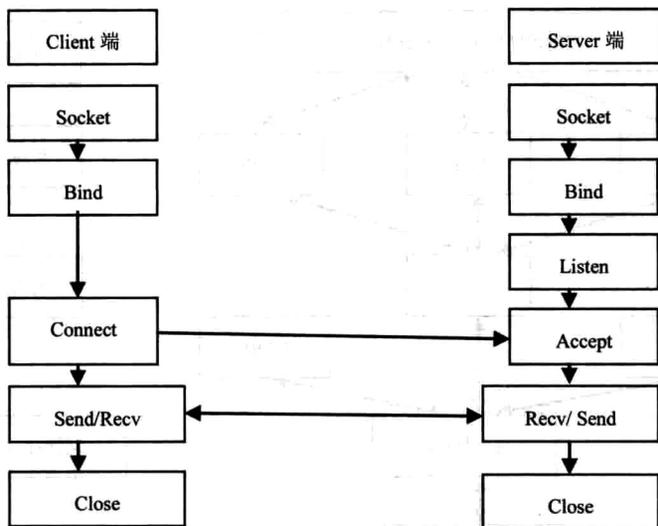


图 3-4 UDP socket 编程流程

## 3.5 IPtables

基于内核空间 netfilter 的 IPtables 防火墙可以用于创建过滤 (filter) 与 NAT 规则等。IPtables 有四种表 Filter, NAT, Mangle 和 Raw。其原理是将控制 IP 包处理的规则集 (rules) 放到不同的链 (chain) 中, 然后在内核中选取五个位置放了五个 hook function 来挂载配置不同链 (chain) 中的规则; 数据包进来后根据不同位置的规则进行匹配, 一旦匹配中, 则执行规则中动作。四种表和五个位置 (Input, Output, Forward, Prerouting, Postrouting) 的支持对应关系如表 3-1 所示。

在云计算环境中, 服务器的内网和外网的转换基本上是通过 IPtables 的 NAT 转换的; 虚拟网络里的安全访问组和防火墙的底层也可以通过 IPtables 来实现。

但是当大量 IPtables 的规则被使用时，可能导致因规则匹配开销引起的性能下降问题，在云计算网络中这点可能的影响需要稍加注意。

表 3-1 IPtables 对应关系表

	Input	Prerouting	Forward	Postrouting	Output
Filter	支持		支持		支持
NAT		支持		支持	支持
Mangle	支持	支持	支持	支持	支持
Raw		支持			支持

## 3.6 QoS 模块

Linux 的 TCP/IP 协议栈里依据 QoS 的几个 RFC 实现了自身的 QoS 功能，其系统实现与交换机的硬件交换芯片功能体现方式上有些不同。Linux 的 QoS 功能基本步骤分为过滤器（Filter）、分类器（class）和排队（QDisc）服务等，其协同工作的简单模型如图 3-5 所示。Linux 的 QoS 服务流程是过滤器依据分类器对报文进行分类进入不同的队列，然后依据队列的不同被不同的 QoS 策略处理，这个过程称为 TC（Traffic Control，流量控制器）。过滤器可以是结合 IPtables 的防火墙或路由来做，而分类器是不同队列的细分。QDisc 又分为无类队列和有类队列，因为队列内部又可以分为多个子队列，而原来的队列称之为父队列；所以无类队列类似于数据结构中树里叶子的概念。无类队列技术包括 Pfifo\_fast、SFQ、RED 等几种，而有类队列技术分为 HTB、CBQ、PRIO 和 WRR 几种。值得一提的是 Open vSwitch 中的端口入方向限速用的是 HTB 机制，Pfifo\_fast 机制在 OpenStack

Neutron 里物理端口和虚拟端口默认使用一种 QoS 方式。

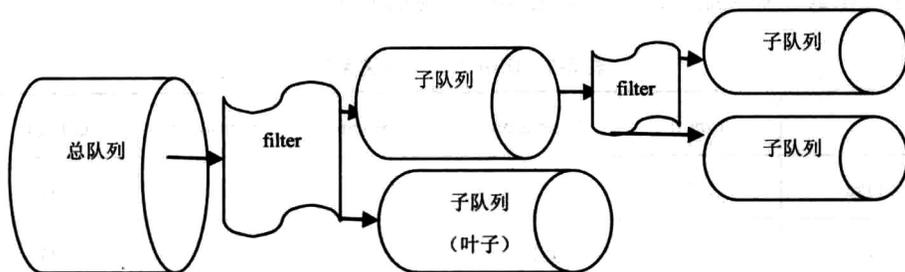


图 3-5 流量控制实现模型

下面列举几种 Linux 里针对网卡或虚拟端口限速的 tc 命令：

(1) 基于 tbf 的方式：

```
tc qdisc add dev eno0 root tbf rate 2000kbit latency 10ms burst 1600
```

(2) 基于 htb 的方式：

```
tc qdisc del dev eno0 root
tc qdisc add dev eno0 root handle 1: htb default 30
tc class add dev eno0 parent 1:0 classid 1:1 htb rate 2mbit ceil 2mbit burst
15k
tc qdisc add dev eno0 parent 1:1 handle 10: sfq perturb 10
tc filter add dev eno0 protocol ip parent 1:0 prio 1 u32 match ip src 0.0.0.0/0
flowid 1:1
```

(3) 基于 cbq 的方式：

```
tc qdisc del dev eno0 root
tc qdisc add dev eno0 root handle 1:0 cbq bandwidth 1000Mbit avpkt 1000
cell 8
tc class add dev eno0 parent 1:0 classid 1:1 cbq bandwidth 1000Mbit rate
100Mbit weight 50Kbit prio 8 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded
tc class add dev eno0 parent 1:1 classid 1:4 cbq bandwidth 1000Mbit rate
100Mbit weight 50Kbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000
tc qdisc add dev eno0 parent 1:4 handle 40: sfq
tc filter add dev eno0 parent 1:0 protocol ip prio 1 u32 match ip src 0.0.0.0/0
flowid 1:4
```

(4) 基于 IPtables 命令也可以实现端口限速功能，限速命令示例如下：

```
iptables -I FORWARD 1 -i eno4 -s 0.0.0.0/0 -m limit --limit 1000/s
--limit-burst 10 -j ACCEPT
```

删除命令用：

```
iptables -D FORWARD 1
```

---

## 3.7 Dnsmasq

Dnsmasq 是一个可以在小型网络中提供 DNS 和 DHCP 配置服务的开源工具，DHCP 可以静态也可以动态，并且能提供本地 DNS 域名缓存或与系统中的 `/etc/resolv.conf` 结合的 DNS 服务。目前在 Linux 系统上 CentOS 和 Ubuntu 里都有很好的支持。具体参考资料可以详见官网地址 <http://www.thekelleys.org.uk/Dnsmasq/doc.html>。

Dnsmasq 的启动选项有很多，所以功能还算丰富。进程开启 DHCP 服务的时候，影响 Dnsmasq 工作性能或功能的因素有很多，一方面是 Dnsmasq 所跑的服务器性能，Linux 系统能为其提供的系统资源对其性能有很大影响，另一方面 Dnsmasq 进程启动的配置参数很重要，所以在 Dnsmasq 的性能问题上需要多加注意。Dnsmasq 的几个主要参数介绍如下：

- `--dns-forward-max=<queries>`来指定当前支持的同时最大请求数，默认是 150；
- `--dhcp-lease-max=<number>`来指定当前 Dnsmasq 进程所能提供租赁的 IP 个数；
- `--dhcp-range` 里有很多参数，其中 `<start-addr>` 和 `<end-addr>` 来指定 DHCP 服务器能分配 IP 地址的始末范围，`lease time` 则是指客户端对 IP 地址的租约时间，对 DHCP 功能来讲客户端如果继续使用其分配的 IP 地址，需要在租约时间到达过期时间一半的时候向 DHCP 服务器发信息完成续租流程。如果服务器没有应答，则该 IP 地址租期到期。

举一个 Dnsmasq 进程启动参数配置的例子。OpenStack 中的 Dnsmasq 是在不同的 namespace 里启动的，因为不同的 Neutron 建立的虚拟网络每个网段基本上都对应一个 namespace，dhcp 的租期默认是一天时间，网络地址最大支持数目是根

据配置的网段及掩码来自动计算的。可以从 OpenStack 环境下网络节点利用 `ps-aux|grep dnsm` 的命令来 dump 具体一个网段的配置如下：

```
Dnsmasq --no-hosts --no-resolv --strict-order --bind-interfaces
--interface=tap836570a9-2c --except-interface=lo
--pid-file=/var/lib/neutron/dhcp/436bb327-0ab9-441d-bc15-07634b17fb3b/p
id
--dhcp-hostsfile=/var/lib/neutron/dhcp/436bb327-0ab9-441d-bc15-07634b17
fb3b/host
--dhcp-optsfile=/var/lib/neutron/dhcp/436bb327-0ab9-441d-bc15-07634b17f
b3b/opts
--leasefile-ro --dhcp-range=set:tag0, 10.0.1.0, static, 86400s
--dhcp-lease-max=256 --conf-file= --domain=openstacklocal
```

## 第 4 章

---

# SDN 网络架构

传统的网络架构中，底层流量无论是靠二层 MAC 地址转发还是靠三层 IP 地址路由，这些转发规则对上层业务来说都是透明的；这种透明性可以说是传统网络架构的特性并为上层服务提供了方便，但是也为上层业务按需使用底层网络资源造成了障碍，同时也给网络的维护和扩展性带来了不便。网络底层技术的进步为上层的应用提供了很强大的支持，而上层的应用需求是无穷尽的，反过来又对底层技术的发展提出了更高的要求。举一个例子来说，某家互联网公司为用户提供图片检索服务，分别在北京和纽约设有两个数据中心，两地的信息需要不断同步才能对外提供一致的服务，但是数据中心的同步流量往往是比较大的且具有突发性，这种远距离的洲际数据传输需要专用的海底光缆，价格非常高，而且带宽利用率也比较低；那么可以利用 SDN 技术对流量的传输进行引导和控制，按照业务需求在合理的时间段进行数据同步，必然能提高数据同步的效率并节省成本。这就出现了一个呼之欲出的网络新架构软件定义网络（Software Defined Network, SDN）。SDN 是一种新型网络架构，旨在实现上层业务应用对底层网络资源的直接控制与使用，可以用来提高网络资源的使用效率，降低网络方案的成本。SDN 的新架构为运营商骨干网络、互联网的数据中心、移动通信网甚至企业内部网络的有效使用网络资源、更安全稳定提供了新的研究方向。

## 4.1 什么是 SDN

SDN 自从产生以来到现在都是非常火热的技术，但是真正深刻理解并掌握 SDN 核心思想的专业人士并不太多，很多人理解 SDN 就是将物理网络全部功能用软件模拟实现或者将 SDN 的概念等同于将控制和转发相分离的架构，其实这些理解认识或观点思想都是不合适的，因为这种将物理网络的功能用软件在虚拟环境下模拟实现的做法称为网络虚拟化（Network Virtualization, NV）。因为网络虚拟化是指将各种不同的软件和硬件网络资源进行整合成一个基于软件统一管理的统一体的过程。有的云计算公司技术人员认为使用硬件的网络就绝不是 SDN，这个观点同样是不正确的，因为其混淆了 SDN 的软件不是指相对于传统硬件设备的实现网络功能的软件，而是指应用程序，即 APP；这里的定义也不能简单地理解为软件决定网络或虚拟化网络，而是通过开放的南向和北向接口来控制的意思，并且需要有反馈机制。

那么 SDN 的核心到底是什么呢？从 SDN 的定义理解可知，是实现应用程序直接控制和使用底层的网络资源，并且有一套开放的控制平面协议和开放的控制平面接口，包括南向接口和北向接口，用来实现应用程序对底层网络资源的控制和使用，另外还需要控制器通过北向接口向应用程序进行反馈。为了实现底层网络资源的分配，一方面是必须为应用程序提供使用网络资源的开放 API，另一个方面是需要对底层网络资源能统筹兼顾地使用，只有这样才能使得各种应用程序对底层网络资源的使用不冲突、不重叠以达到互不影响的目的；所以 SDN 采用了集中控制和转发分离的基本原则；这样集中控制部分称之为控制器，控制器为应用程序提供的控制使用底层网络的 API 称为北向接口，而 SDN 控制器对底层网络资源的使用所调用的接口称为南向接口，现在认为比较常用的南向接口标准

是 OpenFlow; SDN 的南向和北向接口都应该是 Open 的。所以 SDN 的核心思想不是转发和控制相分离, 并且控制和转发相分离的实现方案, 在第 2 章提及的交换机机架产品里早已经得到实现, 在通信网络里的智能网也实现了控制协议和业务数据转发的分离, 但是这些与 SDN 的控制转发分离的目的都是不同的。SDN 的架构图如图 4-1 所示。

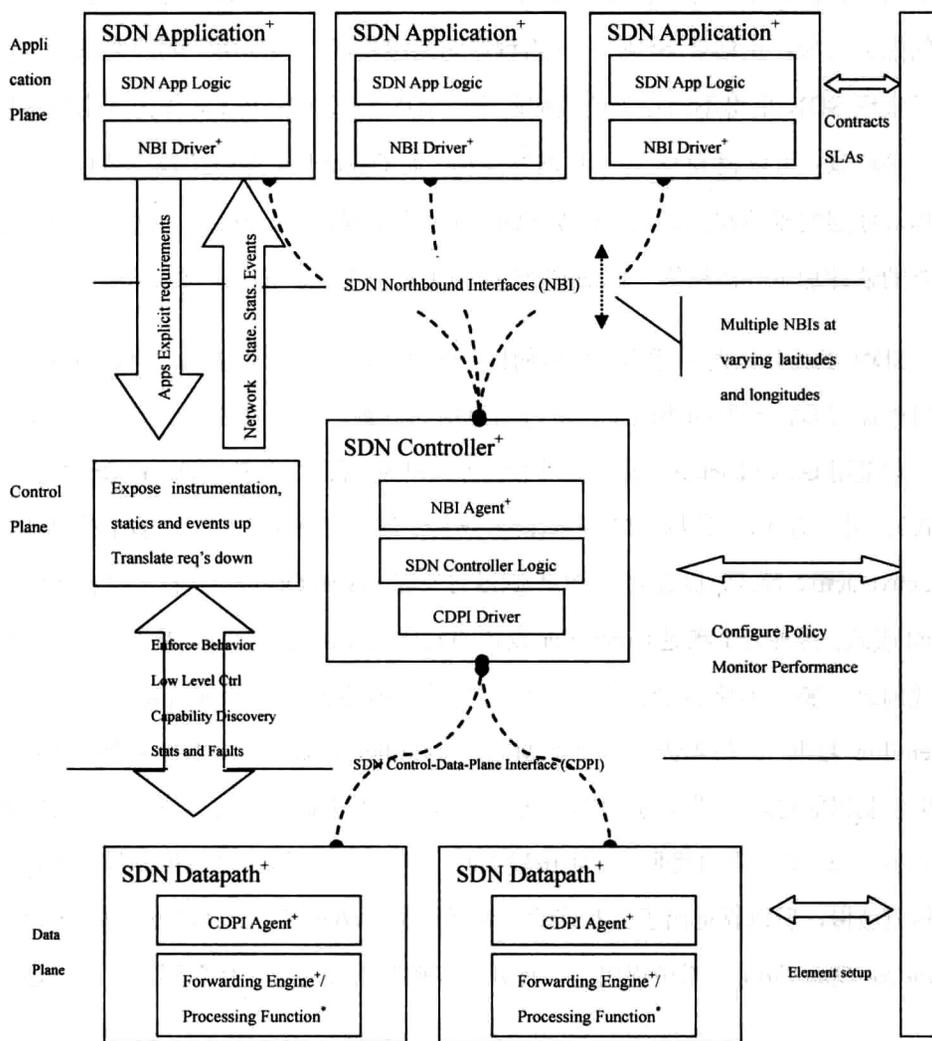


图 4-1 SDN 概览图 (摘自参考文献<sup>[20]</sup>)

SDN 的出现是针对业务的不同,通过控制器的接口为业务提供不同的网络服务,笔者比较早接触到的一种类似 SDN 的场景是在硬件交换机上提供 API 实现的策略路由,即通过对报文的某些字段的匹配指定报文下一跳或其他动作,但是毕竟当时还没有控制器的概念,也没有应用程序直接调用 API 的案例;后来为了对不同业务流进行区分和控制,提出了 OpenFlow 的协议,其能对各种业务流分类的强大之处,正切合 SDN 里应用程序对网络资源直接使用控制的需求,因此为了实现 SDN 里将不同业务的匹配起来,SDN 便将 OpenFlow 作为其南向接口的一种协议;可以说 SDN 是一种网络架构,而 OpenFlow 是一个控制协议,属于具体对底层实现的相关规定,但是 SDN 可以完全脱离 OpenFlow,这种思想不是简单的软件加单纯的网络。OpenFlow 的相关内容在 4.2 节会有详细介绍。

SDN 的控制器对于适当范围内的控制还是可以有性能保证的,但是当规模较大时就需要多控制器的协同工作进行高可用的方案,一方面提高了控制器的可靠性,即采用 active/backup 的模式(比如 OpenFlow 标准里的控制器 master 和 slave 模式);另一方面,可以对 SDN 的请求或其他信息进行负载均衡,多个控制器采用 active/active 模式,以提高 SDN 控制器的性能(比如 OpenFlow 标准里的控制器 equal 模式;再者为了满足 Controller 设备的安全性和稳定性要求,因为一旦控制器出现问题,整个网络就会异常,甚至可能造成网络的不可用或者崩溃,这也是 OpenFlow 标准 1.2 版本提出需要有 Multi-Controller 的原因。纵然有多个控制器,如果控制器的地理位置较为集中,一方面导致控制器集群在应对自然灾害方面的能力有所下降,另一方面就是控制范围较大时,容易因物理距离产生控制信息的延迟和相应缓慢,所以就提出了分布式的控制器概念, *HyperFlow: A distributed control Plane for OpenFlow* 一书中提出了一种分布式控制器并对其工作原理进行了阐述。

现在 SDN 部署实践的经典例子是基于 TTP (Table TyPing Patterns, 后来称

之为 NDM, Negotiable Data-plane Model) 方式的 Google B4 网络的 SDN 优化, 大大提高了 Google 不同数据中心之间数据同步的链路带宽利用率, 从而节省了成本, 需要注意这里 Google 的 B4 网络改造用的是 OpenFlow 协议的 TTP, 而且在博通传统交换芯片上凭借其雄厚软件实力, 开发了基于 TTP 方式的 SDN 新架构; 这一实践为现在南向接口中采用 OpenFlow 的 TTP 提供了良好的实践积累和基础经验, 并且在多控制器开发、控制器之间的选举以及控制和转发平面的分离实践等都做了大量尝试, 所以近来 ONF 也为 TTP 发布了第一个标准 *OpenFlow Controller-Switch NDM Synchronization 1.0* (<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow/OpenFlow%20Controller-Switch%20NDM%20Synchronization%20v1.0.pdf>)。

SDN 的控制器现在有很多, 包括 OpenDaylight (<http://www.opendaylight.org/>)、Juniper 的 OpenContrail (<http://opencontrail.org/>)、思科的 OnePK (<http://www.cisco.com/c/en/us/products/ios-nx-os-software/onepk.html>)、Floodlight (<http://www.projectfloodlight.org/floodlight/>)、NOX (<http://www.noxrepo.org/>)、POX (<http://www.noxrepo.org/>)、Ryu 等。另外值得一提的是, OpenStack 的 Neutron 本身不是 SDN 控制器, 仅仅是相当于一个对底层网络的配置层, 虽然能对上层应用提供开放 API, 但不能直接对底层网络资源进行控制; 与 Neutron 类似, 现在的公有云平台虽然可能有着开放的北向接口, 但并不是有北向接口的都是 SDN, 而且最重要的是无法给应用程序提供反馈机制, 所以如果没有与应用程序控制的 SDN 控制器结合, 现在的公有云平台都还算不上 SDN, 从划分来看只能算是网络虚拟化 (Network Virtualization)。

I2RS (路由系统接口, Interface to the Routing System, <http://datatracker.ietf.org/wg/i2rs/charter/>) 是 IETF 参与 SDN 提出的一个草案, 旨在为运行路由协议的传

统路由设备的控制提供一个北向接口，相对于完全取代现有控制层面的技术，I2RS 是实现在传统路由设备上 SDN 的一个折中方案。

随着互联网和运营商数据中心的单个数据中心服务器的规模越来越大，大量的应用程序被部署到多种不同的服务器厂商、存储厂商和网络设备商的计算、存储和网络设备上。如果想在上面支持一种新的网络功能，必须要更换软件版本甚至是要更新或增加硬件，此类的模式不仅会导致数据中心空间、供电和散热等成本的增加，也必然会导致数据中心创新的困难，并且网络技术不断更新而导致的设备不断更替必然会缩短设备生命周期和降低硬件投资的收益率甚至浪费。因此与 SDN 相接近的另一个概念是提出了 NFV（网络功能虚拟化，Network Function Virtualization，<http://www.etsi.org/technologies-clusters/technologies/NFV>），该功能目的就在于通过整合标准化的虚拟化技术将各种不同的网络设备嵌入到大容量的服务器、网络 and 存储里去。解决的问题包括但不限于以下内容：

- 通过统一和标准化来降低设备的成本及能耗；
- 通过减少网络设备创新的周期来加快其市场化进程；
- 允许多种应用程序、不同版本的同一种程序和不同租户的程序同时被部署；
- 应用程序可以被快速部署和删除；
- NFV 不仅适用于数据中心网络，对移动通信网络等同样适用。

NFV 的概念旨在通过虚拟化技术来实现数据中心网络设备的低成本、较高性能、不同厂商的兼容性与可替代性，以降低能耗并加快数据中心的创新速度；NFV

与 SDN 完全是两个互相独立的概念。但是 SDN 与 NFV 是有交集的，并且 SDN 可以让 NFV 工作得更流畅和高效。

最后，SDN 这种新型网络架构有其独特的优势还在越来越多的业务中被应用。随着 SDN 网络的越来越多，如何让 SDN 网络与传统的网络协调工作是一个值得关注和思考的问题。

---

## 4.2 OpenFlow 与 Open vSwitch

OpenFlow 是 SDN 的南向接口标准之一，而 Open vSwitch 则是一款开源且支持 OpenFlow 的产品，两者的出现对于云计算和 SDN/NFV 里开发新的产品或设计新的方案都有重要的实用价值和参考意义。

### 4.2.1 OpenFlow 简介

2008 年，Nick McKeown 教授等人提出了 OpenFlow 的概念，并于当年发表了题为 *OpenFlow: Enabling Innovation in Campus Networks* 的论文，首次详细地介绍了 OpenFlow 的概念，该文指出在对网络进行创新时新提出的协议往往仅在具有小规模网络设备的实验室里是不够的，但是为了实验而投入大规模的设备需要很大的资金，不可避免地造成浪费；于是便提出了在校园网内通过支持 OpenFlow 标准的交换机上进行实验，该交换机的强大之处就是能够识别任何流，包括已有标准的数据流和未知新协议的实验流，然后对不同的业务数据流通过流表项进行匹配再进行不同的处理，实现实验流和实际业务流互不影响的效果，从而完成在校园内部借助校园网的网络设备来实施新协议网络实验的目的。该篇论文除了阐述 OpenFlow 的工作原理外，还列举了 OpenFlow 其他几大应用场景。这种基于

流的方式无疑为 SDN 架构中应用业务直接控制使用底层网络资源的目标有非常大的契合点，很快 OpenFlow 成为了 SDN 的一个南向接口标准，因为只有底层网络中能对数据流（每种数据流都可以用一些报文字段的特征来表征，对应的往往是上层的一种业务）进行识别和处理后，才能针对不同的业务实现相应配额的网络资源的分配和使用。因此用于流分类的 OpenFlow 协议在 QoS 和安全方面等领域里有着天然的独特应用优势。现在 OpenFlow 的标准由 ONF (<https://www.opennetworking.org/>) 来制定和发布，最新的 OpenFlow 是 1.4 版本。

在 OpenFlow 的协议中，实质是对 OpenFlow 的交换机进行了阐述，包括 OpenFlow 交换机的端口类型及流表项、OpenFlow 交换机与控制器的安全通道和 OpenFlow 协议信息格式三部分。OpenFlow 的交换机端口有很多类型，不同的端口有不同的作用，有必须支持的也有可选的，大体从报文转发方向可分为 ingress port 和 output port；OpenFlow 必须支持的三种 standard ports 分别为 physical ports, logical ports 和 reserved ports。OpenFlow 交换机和控制器之间可以通过 TCP 或安全加密的 TLS (Transport Layer Security, 传输层安全协议) 两种安全的方式连接。OpenFlow 交换机分两种：只支持 OpenFlow 的交换机和同时支持 OpenFlow 和通常以太网转发的两种类型，在最新的 OpenFlow 标准中分别称之为 OpenFlow-only 和 OpenFlow-Hybrid。

在 OpenFlow 1.4 的标准里，表项由若干个流表 (Flow Tables)、一个组表 (Group Table) 和 Meter 表构成，其中每个流表由很多流表项 (Flow Entry) 组成，每条流表项的结构如图 4-2 所示。

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie
--------------	----------	----------	--------------	----------	--------

图 4-2 流表项的组成元素

流表项对应上层下发的流规则，每条流规则下发到一条流表项里。流表项里各字段的含义如下。

- **Match Fields:** 匹配域，主要内容有入端口信息、报文头部解析得到的数据和可选的前面流表传递的元数据 (metadata) 信息，大部分字段在匹配中可以通过值加掩码的方式实现。
- **Priority:** 优先级，流规则的匹配顺序，优先级值越大，表示优先级越高，要优先匹配；报文对在一个流表里进行匹配，只会会有一个流表项会最终被匹配到；匹配域和流规则是流表里区分一个流表项的标记。
- **Counters:** 计数器，按照 packet 级别对匹配到该流表项的报文进行计数；计数的内容有很多如图 4-3 所示。

Counter	Bits	
Per Flow Table		
Reference Count (active entries)	32	Required
Packet Lookups	64	Optional
Packet Matches	64	Optional
Per Flow Entry		
Received Packets	64	Optional
Received Bytes	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Port		
Received Packets	64	Required
Transmitted Packets	64	Required
Received Bytes	64	Optional
Transmitted Bytes	64	Optional
Receive Drops	64	Optional
Transmit Drops	64	Optional
Receive Errors	64	Optional
Transmit Errors	64	Optional
Receive Frame Alignment Errors	64	Optional
Receive Overrun Errors	64	Optional
Receive CRC Errors	64	Optional
Collisions	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Queue		
Transmit Packets	64	Required
Transmit Bytes	64	Optional
Transmit Overrun Errors	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Group		
Reference Count (flow entries)	32	Optional
Packet Count	64	Optional
Byte Count	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Group Bucket		
Packet Count	64	Optional
Byte Count	64	Optional
Per Meter		
Flow Count	32	Optional
Input Packet Count	64	Optional
Input Byte Count	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Meter Band		
In Band Packet Count	64	Optional
In Band Byte Count	64	Optional

图 4-3 流表项中 counters 的统计内容

- **Instructions:** 指令，修改流表项的动作设置或流水处理流程；指令的类型有 Meter (指明一个 meterid 来对报文 remark 或 drop 限速)、Apply-Actions (立即应用 Action List 里的动作到报文，对 Action Set 的内容无影响)、Clear-Actions (清空 Action Set 里的所有动作)、Write-Actionsaction (合并

指定的动作到 Action Set 里)、Write-Metadata (以值加掩码的形式将元数据信息写入到元数据域) 和 Goto-Table (表明处理流程的下一个流表)。如果交换机不支持该指令, 配置时应返回 unsupported 错误。

- **Timeouts:** 老化时间, 流表项处于空闲时间 (即没有任何报文匹配时该规则的时间) 的最大值; 流表项老化时间超过后需要被从流表中删除, 设置为 0 表示永远不老化; 每个流表中必须有一个特殊的表项被称为 Table-miss 表项, 该表项老化时间被设置为 0, 匹配所有的报文, 动作可以是转给控制器、丢弃或将报文重定向到后续的一个流表中处理; 如果流表中没有 Table-miss 表项则对没有匹配到任何表项的数据报文按照丢弃处理, 控制器可以添删以及编辑 Table-miss 表项。
- **Cookie:** 不用于对报文的处理, 可以被控制器来选择设置一些没有明确含义的数据, 比如可用于流表项的过滤报文、流规则修改和流规则删除等标志。

流表项的删除有三种方式: 老化机制、控制器主动删除和可选的交换机主动清理方式。老化机制是每个流表都有对应的空闲老化 (idle\_timeout) 值和硬件老化 (hard\_timeout) 值, 具体做法是当硬件老化值非零时, 流规则无论是否有报文匹配, 时间一到立刻将流规则删除; 当硬件老化值为零且空闲老化值非零时, 交换机需要根据该流规则最后一个匹配报文到现在的时间来计算空闲时间, 一旦超过空闲老化时间值则删除该表项。SDN 控制器可以主动通过发送流规则删除信息的指令来删除某个交换机中对应流表的流规则。交换机清除流表项的做法是当交换机的流表显示使能该功能时才会以回收资源的形式进行处理, 这种清理的方式会受限于流表项参数、OpenFlow 交换机资源回收机制和 OpenFlow 交换机其他内在的限制。

组表也是由很多组表项构成的，主要完成广播、组播、ECMP 等功能，每个组表项的结构如图 4-4 所示，各组表项的含义介绍如下。

Group Identifier	Group Type	Counters	Action Buckets
------------------	------------	----------	----------------

图 4-4 组表项的组成

- **Group Identifier:** 组标识，在组表里的唯一标示符号，暂定为 32bits 的值。
- **Group Type:** 组类型，主要用来标识该组表项的用途，包括 ALL（广播到所有出端口或组播）、select（用于 ECMP 和 LACP 这类的端口选择，选择方式可以是用户配置的哈希或者是 RR 轮询方式）、indirect（用于多个流表项或组表，来实现更快更有效的聚合，诸如 IP 报文策略路由等功能）、fast failover（动作是选择 action buckets 里第一个 live 的，以关联某个特定端口或组表项）。
- **Counters:** 与流表项的 counters 类似，当组表项匹配到某个报文时，对统计内容进行相应的更新。
- **Action Buckets:** 动作桶，由一个有序的链表形式的多个动作桶组成，每个动作桶由一个动作集合（a set of actions）和相关参数组成。

这里需要解释 Action List 和 Action Set 的含义及相互的区别。Action List 是一组动作的列表且同类型的动作可以有重复的多个，执行的顺序是按照在列表里的位置依次执行，动作之间的效果是可以叠加的；Action List 存在于 Apply-Actions 指令和 Packet-out 信息中，当接收的报文传到该流表项中的指令是 Apply-Actions 类型时，会立即执行 Action List 里的动作，报文在后续流表中的处理基于按照

Action List 里的动作对报文的修改。Action Set 则最多只含有一个类型动作，对于 set-field 动作如果是不同类型字段则可以有多多个动作，可以通过 Write-Action 指令或 Clear-Action 指令对初始为空的 Action Set 进行内容修改；Action Set 里的每一个 action 元素无论其相互之间添加的顺序如何，都按照固定的动作类型来执行，动作执行的优先级顺序是 Copy TTL inwards>pop>push-MPLS>push-PBB>push-VLAN>Copy TTL outwards>decrement TTL>set>QoS> group>output。

Meter 表同样由 Meter 表项组成，主要完成 QoS 的限速和重标记功能，Meter 表项结构如图 4-5 所示，各表项的含义如下。

Meter Identifier	Meter Bands	Counters
------------------	-------------	----------

图 4-5 Meter 表项的组成

- Meter Identifier: 用一个无符号的 32bits 的数唯一标识 Meter 的 ID。
- Counters: 对于该 Meter 处理的报文进行统计计数。
- Meter Bands: 用一个无序的 Meter Band 列表来指定速率限制和处理报文的动作，比如 remark dscp 或 drop 等。

每个 Meter Band 的组件如图 4-6 所示。

Band Type	Rate	Counters	Type specific arguments
-----------	------	----------	-------------------------

图 4-6 Meter Band 示意图

Meter Band 内各组件的含义如下。

Band Type: 定义报文的处理方式，比如 remark dscp 或 drop 等。

**Rate:** 用于 Meter 选择 Meter band, 表明可用于该 Band 的最低速率。

**Counters:** 根据被该 Meter Band 处理的报文进行计数。

**Type specific arguments:** Meter Band 的一些可选参数。

每个 Meter 有一个或多个 Meter Band, 每个 Meter 根据配置的 rate 大小来选择特定的 Meter Band 来生效, 选择的方式是从所有 rate 低于配置限速值的 Meter Band 中选择最大 rate 的那个, 如果配置的限速值比每一个 Meter Band 中的 rate 都小, 则没有任何一个 Meter Band 生效。

OpenFlow 交换机通过一个安全通道 (OpenFlow Channel) 与 SDN 控制器相连, 安全通道可以直接使用 6633 端口的 TCP 方式或加密的 TLS 的安全方式。OpenFlow 与 SDN 控制器交互的协议报文信息分为三个种类: Controller-to-Switch, Asynchronous 和 Symmetric, 每一个类型都有多个子类型; Controller-to-Switch 信息由控制器发起并且直接用于检测交换机的状态, Asynchronous 信息由交换机发起并通常用于更新控制器的网络事件和改变交换机的状态, Symmetric 信息可以在没有请求的情况下由控制器或交换机发起。OpenFlow 的标准里还对多控制器进行了规定, 用于快速的故障恢复或负载均衡, 将控制器的角色分为 ROLE\_EQUAL、ROLE\_SLAVE 和 ROLE\_MASTER。所有的控制器默认是 ROLE\_EQUAL, OpenFlow 的交换机能接受其 Asynchronous 信息并发送 Controller-to-Switch 信息改变交换机的状态; ROLE\_MASTER 和 ROLE\_EQUAL 的控制器没有太大区别, 只是全局只能有一个这样角色的控制器; ROLE\_SLAVE 只有对 OpenFlow 交换机的读权限。当一个控制器变成 ROLE\_MASTER 的角色后, 除了 ROLE\_EQUAL 的控制器没任何影响外, 其他的控制器必须全部都是

ROLE\_SLAVE 角色。

对于在 SDN 网络架构下使用 OpenFlow 交换机的来说,可以看作是堆叠或者机架设备的进一步升级,将更大范围的网络设备的控制层集中到一个设备上,但不仅仅是控制器范围的简单扩大;这样做的好处就是可以对此范围的所有网络设备进行统一控制和集中管理,对于网络带宽等资源进行合理的调度和分配,比如 QoS 中综合服务模型的 RSVP 协议,在 OpenFlow 交换机 SDN 这种架构下就相对容易实现各种类型业务流的识别和处理。SDN 控制器对流表的增删修改可以通过 Reactively 和 Proactively 两种模式进行,其中 Reactively 模式是当报文到达时默认没有其对应的转发表项,全都通过默认规则以“packet in”的信息发送给控制器,然后 SDN 控制器再下发相应的流表;而 Proactively 做法相反,需要提前了解业务流量需求,将所需的规则提前下发,当报文到来时可以直接被使用;两种方式各有优缺点和适用场景。

QoS 主要是基于不同的业务对业务数据流进行详细的分类,在 OpenFlow 交换机下发送流规则要按照分类进行业务识别,以提供不同的 QoS 服务。另外,还可以结合 SDN 控制器进行全局网络资源的统一分配和利用,大大提高了网络带宽利用率,节省昂贵的成本;安全方面则是基于攻击报文特性对报文进行分类识别,甚至进行 DPI (Deep Packet Inspect),将攻击报文提取后进行清洗工作,保障正常服务不受影响。

无论 OpenFlow 最终是否能解决传统网络的各种棘手问题而得以长足发展,还是只是一种中间技术形态很快被其他新技术所代替,这些标准的确立和采用在很大程度上更取决于各个利益相关者的竞争,标准的最终形态是一个博弈过程,更是各个相关厂家瓜分相关领域网络市场最终讨价还价的结果。将来 SDN 将发

展成何种形态现在还无法得知，但是 SDN 的发展经过必然是一个血雨腥风的过程，有可能网络市场被重新洗牌，一些芯片商、设备商和云计算公司发展壮大，当然还有更多的被淘汰。

SDN 和 OpenFlow 关系，两者是不能划等号的。SDN 是一种网络架构的新思想，也是一种站在用户角度、站在管理员角度可以参与管理和控制网络底层转发决策需求的满足和实现，而不是一种方案或者功能。底层转发的实现是否是纯软件或者有多大程度的硬件参与并不重要。OpenFlow 作为 SDN 底层南向接口实现的一个理想选择，在转发层面确实有很多优势，但并不是 SDN 南向的唯一选择，笔者认为还有以下一些需要注意的地方。

(1) 如前文所述，将控制平面和转发平面进行分离，减少了控制点，增加了控制平面的负担，需要比以往有更强的 CPU 计算能力的设备才能堪重负或者通过高可用技术提高可靠性。

(2) SDN 上层应用对网络资源的使用和控制，这个过程需要借助于管理员在管理网络方面的能力和对网络底层资源的熟悉程度，这些无疑对管理员的技能和对所管理的业务熟悉度有了更高一步的要求，这样才能有合理的网络资源使用方式，这一点与网络设备智能化、自动化、傻瓜化的发展思想多少是有点相悖的趋势；因此如何让 SDN 控制器对网络底层更智能地自动判断资源并合理使用资源是一个问题。

(3) OpenFlow 的多表项为业务实现提供了灵活性，但是这无疑增加了设备的成本，并且为交换芯片驱动的开发人员在软件中记录和维护配置表项、容错等研发工作提供相当大的复杂度；无论升级已有的设备支持 OpenFlow，还是购买支持 OpenFlow 的新硬件，都是不小的投资，需要客户根据实际需求来决定。

(4) OpenFlow 在转发层面相比于传统方式有了很大提高，理论上应该是先兼容以前所有的转发方式，但是现在的标准中，还有很多传统转发方式支持的功能在 OpenFlow 中是不支持的，比如要实现某个出端口中 SIP=1.1.1.1 的报文全丢弃，标准中暂时没有直接出现匹配出端口的特性，希望 OpenFlow 标准尽快得到完善。

(5) 同 (4)，出口方向的 QoS 限速功能甚至整个 OpenFlow 的 QoS 功能都不是很强大，离传统交换芯片的支持完善程度还有很大的差距。

(6) 传统交换机中还有很多转发层面的功能或报文处理动作方面的功能，比如流控、队列调度、WRED 等方面都要涉及交换芯片 MMU，而这些 OpenFlow 是很难对各个厂家通过标准而统一的，所以 OpenFlow 标准化方面还有很多的工作要做。

## 4.2.2 Open vSwitch 简介

Open vSwitch (<http://Open vSwitch.org/>) 是开源的 Apache2.0 许可下的产品，比一般开源的工程代码质量要高很多，在 XEN 和 KVM 的开源平台上都可以支持，支持的功能包括用于隔离的 VLAN/VXLAN/NVGRE、用于监控的 NetFlow sFlow/IPFIX、SPAN/RSPAN/BFD、支持端口的 LACP 及 TRUNK 模式、这次 MIRROR 功能、支持 STP 和 QoS 的入口限速及队列、支持连接 SDN 控制器 OpenFlow 流表项配置等。

Open vSwitch 对报文的转发流程大致如下：

① 交换机起来后，添加或变更交换机所包含的端口，这里的端口可能是物理端口也可能是虚拟网卡；

② 根据数据包的触发、CLI 命令下发或者 SDN 控制器的指令下发流表项，

可能是转发表项，也可能是其他动作的表项；

③ 从交换机端口接收报文后，根据端口类型对报文进行解封装处理，比如 VXLAN 或 NVGRE 的端口需要将相应报文的隧道头去掉后再解析报文的头部字段信息，包括二层头的 MAC 和 VLAN，三层头部 IP 字段信息、四层头部 TCP/UDP 等包头信息及还可以用 NXM 自定义便宜解析；

④ 根据解析得到的字段组成 key，去匹配 Open vSwitch 中已有的流表项；

⑤ 根据匹配结果转发到对相应的出端口，也可能是被丢弃或者被送上控制器；

⑥ 到出端口根据相应端口类型可能进行字段修改或隧道封装转发出 Open vSwitch。

在虚拟环境下的虚拟机之间进行的网络行为，包括内网通信、外网通信、QoS 服务、安全监控和管理部署等，都给网络设计和运维带来了困难，因为 VM 之间的流量在服务器内部，并不是所有的流量都到服务器之间的物理交换机上；此时 Open vSwitch 的出现给这些问题的解决带来了希望，因为 Open vSwitch 支持 VLAN/VXLAN/NVGRE 等隔离方式，并且支持 QoS 服务和 SFLOW 等监控功能，最可贵的是还能支持 OpenFlow 协议；这些不仅让虚拟机的网络提供丰富的网络功能成为可能，将来还可以让支持 OpenFlow 南向接口的 SDN 的控制器非常方便地整合到云计算的网络中去。但是 Open vSwitch 里支持的 OpenFlow 与 OpenFlow 协议里的确切内容可能有所不同，这个现象很多是系统实现的缘故导致的。毕竟 Open vSwitch 是一个软件实现的虚拟交换机，所以其性能和系统实现有很大的关系，不同的报文可能导致软件处理的流程不一样，性能也就可能有非常大的区别，这点是和硬件交换机区别很大的地方，在实际部署中也非常值得注意。

## 4.3 能为 SDN 做什么

SDN 从提出到现在已经有好几年了，大家都在提 SDN，讨论 SDN 的概念，探索 SDN 的应用场景，进行 SDN 领域的创新，很多的设备厂商（Pica8、南京叠锶、思科、Juniper、华为、苏州盛科、南京 xNet 等）、互联网公司（Google/Facebook/Amazon 等）、云计算业务公司（海云捷迅，Unitedstack 等）以及高校等都在研究、开发、应用和推广 SDN 相关技术。SDNAP 高级群（qq: 279796875）刚升级，可以支持到 2000 人规模，相信将来不久群就又会爆满。大家都在讨论和学习 SDN 的技术，但是我们能在 SDN 里做什么？或者高尚点说我们能为 SDN 的发展做些什么？

笔者认为想对 SDN 研究，请先熟悉传统网络，再来谈什么是 SDN，才能理解 SDN 相关技术的产生背景、适合应用的领域及解决问题有哪些优势；如果一个人不懂什么是网络，不了解传统网络遇到的问题，而直接跟着别人火热地呼喊便全身投入身心地要做 SDN 领域的东西，很可能做出来的是个四不像。唯有熟悉了网络技术的历史，掌握了现在网络技术应用到某些场景所遇到的难题，其中哪些问题用 SDN 来解决是非常彻底的，这样才能做到盛科张卫峰（文献<sup>[1]</sup>的作者）说的“不是为了 SDN 而 SDN”。就好比吃过一道好菜，但是只有吃过难吃的菜的食客，才知道好菜为什么好吃；如果整天吃的都是好菜或者根本没有机会吃到好菜，是绝无法体会到好菜的味道到底好在哪儿的。

SDN 的发展，首先要有一个功能完善和标准化的过程。任何新鲜事物的发展都有一个完善的过程，SDN 技术的发展过程也是如此，无论是已经成为标准的 OpenFlow 技术（该标准的完善仍然还有很长的路要走）；还是还没有统一标准的

Controller, 以及现在仅看到通过 OpenFlow 安全通道用 OpenFlow 协议与 OpenFlow 交换机通信的南北向接口等少数标准化, 这些不同厂商或发布者之间的 Controller 能相互通信吗? 控制器之间通信的协议或标准需要遵循什么标准或格式? 或者说 Controller 之间东西方向的通信流量需要吗? 还有很多其他比如 Open vSwitch、OpenDayLight 等 SDN 相关的产品也需要大量地开发和完善。因此, 为了 SDN 发展好, 先要把它完善好。

SDN 的发展, 需要把 SDN 控制器范围内的网络设备和传统的网络设备进行协同工作。SDN 的基础网络原理都来源于传统网络, 并没有对网络原理提出很大的创新; 熟悉传统网络, 将传统网络和 SDN 网络的协作过程及需要的标准、产品、工具做好, 也必定能促进 SDN 的发展。这方面工作的一个很重要的体现就是, ONF 的 OpenFlow 标准里 Hybrid 交换机上对于 NORMAL 和 FLOOD 等端口的支持。

SDN 的发展, 必须不断探索 SDN 应用的新场景; 找到 SDN 的用武之地, 为 SDN 的发展带来技术驱动力, 从解决现实问题上给使用者带来经济效益, 必然能促进 SDN 相关技术的发展到一个新的阶段。在计算机相关研究领域上, 笔者相信工业界指导学术界, 所以一个只是研究书本和老旧知识但从来没有参加过实践工作或者不与产业界保持紧密交流的研究者, 其所出的成果, 只能增加初学者搜集有效学习材料的工作难度。

SDN 的发展, 也需要不断研究 SDN 的不足, 就是找出 SDN 技术不适用的领域或缺点有哪些方面, 笔者认为这也是为 SDN 发展做贡献, 就可以避开 SDN 的死角, 使其走上更加光明的道路。就算哪一天有人通过理论和测试证明 SDN 道路必死, 那也是非常值得称赞的, 因为他让大家都不再花费冤枉的精力到 SDN 领域。

但是 SDN 的发展，绝不是前面说的为了 SDN 而 SDN；把 SDN 技术用于传统解决不了或者解决不好的而 SDN 能很好地解决的问题才是其目的；现在出现了很多不好现象，没有弄明白什么是 SDN，就把传统的网络转发方式在 OpenFlow 上“实现一遍”，把传统已经解决的问题用 OpenFlow 再解决一遍，甚至作为专利或者论文来申请和发表，这些行为是学术腐败的可耻行径，应该被严厉批评和坚决禁止。

顺便说一下，讨论问题的沟通需要艺术。无论是工作学习的研究成果的还是仅仅作为个人知识的汇总，如果能分享的希望尽量分享，SDNAP 已经为大家搭建了一个很好的分享平台。而作为读者，关注点应该是笔者的观点，是不是自己没有领会到，或者领会到了有不同意见需要相互讨论，而有些专门指责别人错误却说不清自己理由或者不理睬笔者观点而揪着读者文字的瑕疵来抬高自己的行为笔者觉得都是不可取的，更有甚者发展到了人身攻击。有不同意见可以私下甚至当面相互讨论都是提倡的，但是只能从学术的角度来进行。

从最本地说，熟悉 SDN，并把自觉研究和推广 SDN 作为一种兴趣，提高自身能力和经验的同时扩大自己影响力，而又通过提高的自身影响力来推广 SDN 去解决相应领域的问题。这应该是每一个对 SDN 技术感兴趣人士的基本职责，但是却是非常难于做到的，真的很不容易。

从 SDN 群的讨论来看，2014 年感觉 SDN 有点淡了，其实是大家对 SDN 都开始沉下心来思考，冷静下来也好，这样才能让 SDN 有更多的产品落地；经过前几年的火热，是该到冷静下来思考的时候了，这样能出更好的作品。希望对 SDN 感兴趣的所有人都在 SDN 之路上结出硕果。

有关 SDN 的书籍笔者推荐张卫峰的《深度解析 SDN 利益.战略.技术实践》

和雷葆华的《SDN 核心技术解析和实战指南》等。另外 SDNAP (SDN Associated Press) 为 SDN 从业者提供一个综合报道的博客 (媒体) 平台, 任何个人或商家都可以在此平台上投稿发布自己对 SDN 行业的看法、分析、评论及相关产品信息, 并且可以随意关联投稿者的微博、博客网站或邮件地址等。目前 SDNAP 有个官方 QQ 群: 初级群 (群号: 337157076) 和高级群 SDNAP ((群号: 279796875), 对应的 sina 微博 @SDNAP, 相应 SDN 的论坛 <http://www.SDNAP.com>。吴总 (群名片 @北京 sdnap-吴厘头) 为 SDN 交流群和论坛付出了巨大的心血, 希望能被大家好好利用。



## 第 2 部分

# 云计算及 OpenStack 的网络

云计算平台的本质就是将计算、网络和存储的相关设备虚拟化为一个资源池进行统一控制和划分给租户（Tenant）作为可管理的服务来使用的系统，租户可以在其租用的虚拟化基础设施上快速部署业务平台和应用软件，对业务相关的数据进行存储、管理、分析挖掘并提供相应业务服务，减少租户的运营设备成本、维护工作和企业运作成本；而云计算系统里的云平台设备间的管理信息传递、租户虚拟机提供服务的业务数据传输、业务数据通过存储网络到存储设备的数据传输和 VM 跨设备甚至跨数据中心的迁移等，都需要通过云计算中的网络技术来承载传输的数据；云计算中的网络技术是为租户提供网络互通和隔离功能、租户业务提供带宽的 QoS 功能、为租户上层业务提供服务功能（包括防火墙、负载均衡和 VPN 等）、为租户虚拟机迁移提供传输功能、为租户提供高吞吐的存储网络功能、为租户网络提供防止攻击的安全功能、为云计算系统的设备（包括物理设备和虚拟设备）实时运行情况的监控功能等云计算系统里设备间通信所用的一些相关技术。云计算的基础是虚拟化，而网络的虚拟化技术是云计算的重要组成部分。网络虚拟化中比较早的技术从 VLAN 开始，即在物理域中虚拟几个广播隔离域，典型的“一虚多”技术，PVLAN（Private VLAN）技术则是对 VLAN 的再

一次一虚多；汇聚口则是为了解决单端口带宽不足和负载均衡流量的多虚一技术；现在已有很多商业交换芯片支持单物理端口可以虚拟成多个 Vport 的技术，这种技术也是典型的一虚多技术，相应的是一个物理交换机可以虚拟成多个交换机的一虚多技术；网络虚拟化可以说在现代网络技术里已经很常见。云计算研发里计算、存储和网络等三方面相关工作同等重要，缺一不可，但是网络技术研发方面的工作量至少占到云计算研发所有工作量的一半以上。

## 第 5 章

---

# OpenStack 的网络

云计算平台是一种以虚拟化技术为基础，以提高服务器 CPU 和网络资源利用率及其带来的能耗降低、维护简易、便于扩展、成本下降为目的的新型服务系统和运营模式。云计算按照服务的层次可以划分为三种，即 IaaS（Infrastructure-as-a-Service）、PaaS（Platform-as-a-Service）和 SaaS（Software-as-a-Service）三类。现在常见的 IaaS（Infrastructure as a Service）开源平台有 OpenStack、CloudStack、Eucalyptus 和 OpenNebula，但 OpenStack 的火热程度远远高于后三者，且国内基于 OpenStack 再研发的互联网公司和创业公司非常之多，很多有研发实力的互联网公司则是在 OpenStack 的基础上进行了大量适合自己业务的定制开发。随着云计算系统里虚拟机（Virtual Machine, VM）的数量增多，IaaS 平台中 VM 之间的通信，以及 VM 和外网设备之间的通信，导致了网络虚拟化的基本需求，后续的存储网络虚拟化也随之而来。随着云计算系统里设备间通信环境变得越来越复杂，监控、安全和运营维护问题也日益突出，这些问题都是现在需要结合云计算网络来调研和解决的，因为云计算系统管理员不仅需要控制管理整个云计算平台的服务器和网络资源，还需要实现对设备故障的监控和防护攻击的报警与保护，而且云计算的租户也需要时刻能掌控其部署在 VM 上的业务运行是否正常。

现在云计算中用到的网络虚拟化技术主要有 VLAN/VXLAN、Bridge、IPtables、TUN/TAP、Openv Switch 等。本章将以 OpenStack 的 H/I/J 版网络为例，深入分析云计算网络中所采用的底层技术，及将 Neutron 组件产品化过程中会有哪些需要注意的技术点。现在 Neutron 中提供的网络功能，主要提供了在云计算环境下为不同的租户的虚拟机建网组网、分配带宽和 IP 地址、网络隔离、外网 NAT、FWaaS、LBaaS、VPNaaS 等。在后续的 Neutron 技术讨论中，如果配置环境没有特殊说明，都是指以 Open vSwitch 为 Plugin 的底层网络原理和特性的研究。

---

## 5.1 云计算及 OpenStack

谈起云计算，就要涉及虚拟化的内容，因为虚拟化是云计算平台实现的技术基础。从概念上看，最早的虚拟化的起源可能要追溯到 1959 年 6 月，在国际信息处理（International Federation for Information Processing, IFIP）大会上，美国的克里斯托弗·斯特雷奇（Christopher Strachey）在《大型高速计算机中的时间共享》（*Time Sharing in Large Fast Computers*）一文中，最先提到了“Virtualization”这个对应中文虚拟化的词语。1965 年，IBM 发布了最早在商业系统上实现虚拟化的产品——IBM 7044，但是基于 PC 服务器的虚拟化技术一直进展缓慢，直到 1999 年，VMware（2003 年 12 月被 EMC 收购）才发布了它的第一款产品 VMware Workstation，2001 年通过发布 VMware GSX Server（托管）和 VMware ESX Server（不托管）宣布进入服务器市场。VMware Workstation 可以工作于 Linux 和 Windows 操作系统上。后来 AMD 和 Intel 的处理器都在内核里设计了硬件虚拟化功能。Intel 的 VT（Virtualization Technology）技术和 AMD 的 SVM（Secure Virtual

Machine) 使得虚拟化的领域扩展到了硬件。2006年8月9日, Google 首席执行官埃里克·施密特(Eric Schmidt) 在搜索引擎大会(SES San Jose 2006) 上首次提出了“云计算”(Cloud Computing) 的概念, 从那时起到现在, 云计算一直为炙手可热的概念。随着互联网巨头的业务兴起, 各家互联网公司在对数据中心计算服务器、数据存储设备和网络设备的扩充过程上, 为了在设备故障的时候保证有冗余设备可用的及时性, 一直存在着冗余甚至浪费, 因为为了满足新业务的上线需求, 必须提前准备大量的设备, 这样很多设备处于闲置状态, 尽管这些设备可能没有上电。这个问题在亚马逊(Amazon) 等大型互联网巨头的数据中心里尤其突出。后来为了让这部分资源得到充分利用, 便出现了当今成熟的 IAAS 平台 AWS (Amazon Web Services), 紧接着 Google、IBM 和微软一系列国外 IT 巨头相继宣布了各自的云计算平台产品。虽然大小企业和开发者个人都在提云计算, 但是每个人对云计算的理解与对目前网络的新架构 SDN 技术的理解一样, 都不尽相同。总体来讲, 云计算是以对数据中心计算、存储与网络资源进行一虚多或多虚一的虚拟化技术为基础, 为租户提供统一管理使用和监控运维数据中心内部基础设施的新型资源服务模式。

云计算中所用到的网络技术与传统的网络技术原理一样, 只是将原有的传统网络技术运用到云计算下的虚拟环境里。但是其特殊性在于当云计算中有多个租户(Multi-tenant) 时, 每个租户的 VM 之间需要做隔离, 此处的隔离是指 VM 之间用每个租户的私有 IP 地址是不能通信的, 当 VM 都配置或绑定了公网 IP 地址后, 可以相互之间进行通信。类似于 A 和 B 两个公司的办公内网, A 内的网络相连设备可以采用内网互相通信, 但是不能直接用公司 A 内网的私有地址和 B 公司内网的任何设备通信(当然需要排除两个公司之间架设了 VPN 类似的特殊情况), 同时 B 公司也同样不能直接用其私网地址和 A 公司内的任何设备进行通信, 纵然两者可能有相同的内网网段地址。但在 A 公司或 B 公司里, 同一个公

司的两个部门之间是可以利用内网通信的，甚至同一个公司分布在不同城市的办公场所之间也可以通过 VPN 等方式直接使用内网地址进行通信。

云计算研发方面的国外公司有前文提及的 IBM、微软、Google，以及亚马逊的 AWS 等，国内则有 Ucloud、海云捷迅、UnitedStack、EasyStack、金山云、阿里云、寂云科技、易云捷迅等。现在比较流行的四个云平台是 CloudStack、Eucalyptus、vCloud Director 和 OpenStack，其中，OpenStack 占了绝对的份额优势。OpenStack 可以说是一个社区（OpenStack 社区官网：<http://www.OpenStack.org/>，国内对 OpenStack 的报道网站有 <http://www.OpenStack.cn> 等）；OpenStack 社区的官网资料包括开源的代码（源代码网址：<https://github.com/OpenStack>）、多个组件的项目开发与发布等内容；由美国国家航空航天局和 Rackspace 合作研发，以 Apache 许可证授权的一个自由软件和开放源代码项目，现在大约有超过 120 家公司和 3600 名社区开发人员奉献于 OpenStack 社区。OpenStack 是一个云平台管理的项目，它不是一个单纯的开源软件，每年的 4 月和 10 月左右各有一次发布新版本的峰会，版本号以英文字母为序，选取相应的一个单词作为版本名称，比如 H 版是 Havana，I 版是 Icehouse，J 版是 Juno，将来 K 版是 Kilo。OpenStack 随着组件的数据不断增加，支持的功能也日益丰富；其版本号现在到了 I 版，J 版马上也要推出。OpenStack 的版本号和发布一个软件产品类似，包括软代码以及相应的技术部署文档等说明信息；只是随着功能的增加和完善用版本号作为一个个里程碑来区分记录下而已。

OpenStack 为了实现云计算的各项功能，将计算、存储、监控和网络划分为几个项目分别来开发，每个项目对应 OpenStack 的一个或几个组件；OpenStack 的组件及其作用如表 5-1 所示，组件之间的关系如图 5-1 所示。

表 5-1 OpenStack 各组件及其作用

服 务	工 程 名 字	描 述
Dashboard	Horizon	让用户与 OpenStack 各种服务交互
Compute	Nova	创建和管理虚拟机
Networking	Neutron	用流行的网络供应商的网络技术以 Plugin 形式作为服务为用户提供各服务或虚拟机的网络互联功能
Object Storage	Swift	提供对象存储服务
Block Storage	Cinder	提供块存储服务
Identity Service	Keystone	为 OpenStack 的各个服务提供认证服务
Metering/Monitoring Service	Ceilometer	通过对 OpenStack 的服务进行监控和限速, 实现计费、扩展和统计的需要
Orchestration Service	Heat	通过 REST API 或兼容的查询 API, 以及模板的形式为云计算各种组件的应用提供模板服务
A data-intensive application cluster	Sahara (曾经为 Savanna)	OpenStack 在 Openstack 上提供一个数据密集型应用集群 (Hadoop or Spark)

OpenStack 的各个组件之间耦合是非常松的, 其中 Keystone 是各个组件之间通信的核心; 每个组件都需要向 Keystone 注册; 当一个组件开启后先向 Keystone 注册, 并获取可以通信的组件的地址 (IP 地址及对应端口号代表的服务), 然后再做些辅助型工作就可以实现和其他组件的通信任务了。

对 OpenStack 的学习如何入门? 学习 OpenStack 时, 请先参考官网了解一下 OpenStack 的概念及其大概历史, 然后在官网中查看安装操作或原理方面的文档,

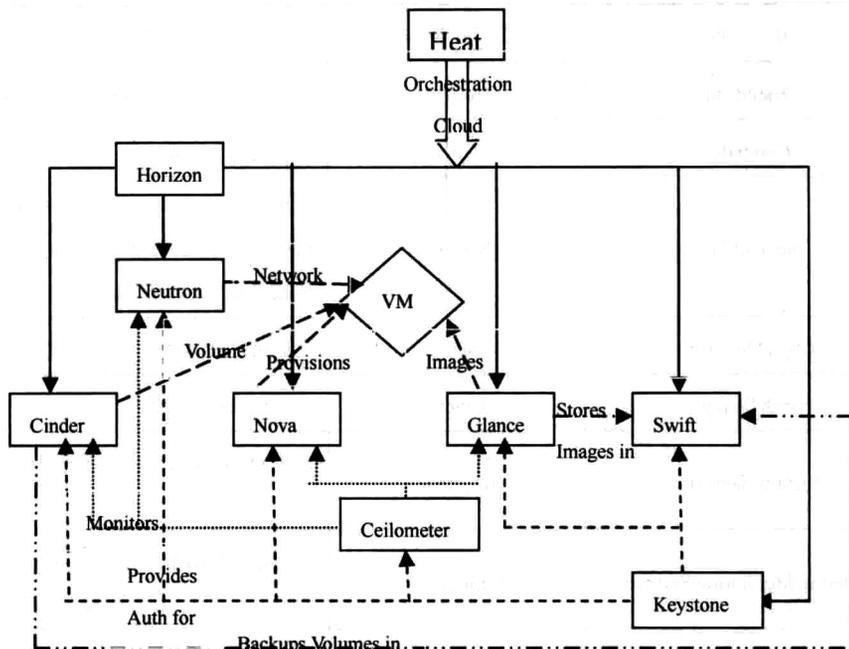


图 5-1 OpenStack 各组件相互作用关系（引自 OpenStack 官网）

如果读者想走捷径，可以先查阅国内外著名人士的博客（比如陈沙克老师的 <http://www.chenshake.com/technology/>）。接着结合学习使用和负责工作的内容，可以找各个组件的作用和内部机理的资料来深入学习。通过网络博客学习 OpenStack 需要注意的是，很多这方面其他人的博客写的内容比较简单，还有些技术博客的内容是不全面甚至部分是错误的，需要读者进一步研究理解和把控。建议不要急于动手实践，因为如果不学习原理就直接安装实践，必然会遇到很多困难。另外，学习 OpenStack 还需要紧跟社区进展和相关产业发展，充分理解客户的需求，及时掌握新技术的应用场景和 OpenStack 社区的发展方向。

有了一定的基础知识后，便可以实践 OpenStack 的相关内容了。OpenStack

的安装文件可以在官网 (<http://docs.OpenStack.org/>) 中找到, 建议一步步地用源码安装, 这样必然会学习到 OpenStack 的很多内容。当然, 也可以采用一键部署的方式——Ubuntu 下有 devstack(官网详见 <http://devstack.org/>), Centos 或 Redhat 系列操作系统里可以采用 RDO 的 packstack 工具(官网详见 [http://openstack.redhat.com/Main\\_Page](http://openstack.redhat.com/Main_Page)), 具体内容可以参见官网的介绍; 但是同一个新的功能 RDO 的 packstack 比 Ubuntu 的 Devstack 集成发布时间要晚一些, 集成进去的 OpenStack 的功能也稍少一些(现在来看 VPNaas 暂时 RDO 的 packstack 工具还没有一键部署的安装配置)。总之, 学习 OpenStack 自身需要的内容大都可以在官网中找到, 少数好的技术博客也很有参考价值。但是 OpenStack 的官网文档也有很多不足甚至错误的地方, 需要甄别理解再吸收。所以本书除了详细阐述 Neutron 所用到的网络底层技术, 只会对 H 版和 I 版 Neutron 的功能进行简单介绍, 并简单讨论 J 版 Neutron 的一些重要特性, 比如 DVR 等, 不会对 Neutron 本身的代码做太多讨论, 因为其代码架构会随着后续的优化和开发在 J 版及以后的版本中都会有很大的调整或改变, 但是底层使用的网络基本原理是不会变的, 理解这些是掌握 Neutron 万变不离其宗的钥匙。

---

## 5.2 OpenStack 的网络介绍

涉及云计算网络就不能不谈起 AWS (Amazon Web Services) 的 VPC (Virtual Private Cloud) 技术, VPC 是亚马逊提供的云计算网络服务, VPC 也将网络分为私网和公网, 并且都有子网的概念, 并提供了防火墙和 VPN 的功能, AWS 也采用 NAT 技术作为内网和外网的转换技术, 可以通过 Web 界面来使用, 同时有 CLI 的命令和开发用的 API 接口, 不同租户 VPC 的网络资源是相互隔离的。但是 AWS

是闭源的，无法对底层细节详细深入研究，所以这里选择一款常用的云计算平台——OpenStack 的 Neutron 组件为例来详细介绍下云计算网络用到的技术原理。

OpenStack 的网络最初是由 Nova-network 来提供的，租户间的隔离采用 VLAN 技术，并且支持 Multi-Host 方案；OpenStack 在 Diablo 版中 Quantum 以发展项目的形式首次出现，在 Essex 版本中有了试用版的 Quantum，直到 Folsom 版本 Quantum 才正式发布；Grizzly 版本里网络功能得到了极大的增强，因为名称 Quantum 已经被一个公司 Quantum 注册，所以在 Havana 版本中更名为 Neutron，现在 Icehouse 版、Juno 版中网络组件里都是以 Neutron 冠名的。

OpenStack 的 Neutron 网络非常类似于实际当中的网络结构；拿公司企业的网络架构来讲，将企业自己的办公环境全部转移到云计算环境中，那么公司内部的服务器则是 VM，服务器之间的交换机属于云计算网络环境下的 L2 层虚拟交换机，而服务器通过 DHCP 获取的网络地址则是 VM 的固定 IP (Fixed IP)，公司内网通往外网的路由器则是虚拟网络中的默认网关设备；为了向外界用户提供公司的产品服务，当某台物理服务器需要被外界直接访问时需要通过 NAT 技术给这台物理服务器绑定一个公网 IP，在云计算网络中也是一样，VM 的公网 IP 称之为浮动 IP (Floating IP)；运营商会根据公司需求分配一定的带宽，而在云计算网络中也有基于租户带宽和基于 IP 地址带宽限制的需求；为了保证公司内网的网络安全，需要在路由器上配置防火墙以阻挡外面的入侵和内部服务器访问外网不安全的网站，好比在现实网络中一样云平台还提供防火墙服务 (FWaaS)；如果公司提供服务的访问量非常大，单台服务器设备无法提供足够的能力来处理这些请求就需要有多台服务器设备通过负载均衡服务 (LBaaS) 来分担单个服务器的压力，并提高整个系统的承受能力；另外，一个公司可能会分布在多个城市，为了让一个公司的多个地点的内部网络实现互通就需要采用 VPN 服务

(VPNaaS), 包括接入 VPN 和端到端的 VPN 两种形式; 而在云计算中这些需求也有对应的 FWaaS、LBAas 和 VPNaaS 三种服务来满足, 从安全角度和资源抽象角度, 还有 IDS-aaS 和 data-center-interconnect-aaS 的服务。

### 5.2.1 Nova–Network

在 2010 年 OpenStack 第一个版本 Austin 正式发布时, Nova-network 便已被作为组件发布。Nova-network 的主要功能是为 VM 提供私有 IP 地址 (Fixed IP) 和浮动 IP (Floating IP), 通过 IPtables 和 Ebtables 来提供安全功能, 并为网络管理提供三种网络模型: 扁平网络 (Flat Network)、DHCP 扁平网络 (Flat DHCP Network) 和 VLAN 网络 (VLAN Network)。

在 Nova-network 的网络里, 物理机上同一租户的虚拟机通过 Linux Bridge 与服务器的物理网卡相连, 以此来将数据信息通往外网或发送到其他服务器上的 VM, 每个租户至少占用一个采用 VLAN 技术划分的隔离域, 每个租户内的 VM 若是通过 DHCP 获取 IP 地址, 则必须有对应子网的 Dnsmasq 进程。

Openstack 的 Nova-network 中有三种网络模型中的 FLAT 模式需要管理手工创建网桥并把虚拟机关联到该网桥, 所有的虚拟机处于同一个子网下; FLATDHCP 模式也是所有的虚拟机在同一个子网, 但是 Nova-network 自动创建网桥, 通过 Dnsmasq 为 VM 自动分配 Fixed 的 IP 地址, 并且维护已经分配的 Floating IP; VLAN 网络模式为每个租户分配一个 VLAN, 租户之间的二层网络相互隔离, 并且 Nova-network 自动创建网桥, 然后将网桥连接 VM 的虚拟网卡, 通过 Dnsmasq 分配 Fixed IP, 并维护已分配的 Floating IP 地址等。当同一物理机上同一租户的 VM 之间通信时, 只需要在物理机内部网络通信即可; 如果是 VM 访问外网的通信, 在 Multi-Host 模式下则可以从本地物理机的外网网卡直接通往

外网进行通信；不同物理机上的同一租户的 VM 之间通信也只需要通过物理网络环境的内网才能进行通信，而不同租户之间的 VM，无论是否在同一个物理服务器上，都需要经过外网地址即 Floating IP 才可以进行通信。

在云计算网络环境里，每一个计算节点里的 VM 都需要通过内部网络交互信息，并通过外部网络获取资源或者为外界以间接和直接两种方式提供可访问的资源，这里的网络是指云计算系统的物理网络；那么 Nova-network 的技术环境下，VM 具体是如何访问外部网络呢？Nova-network 提供了 Multi-Host 方案，这种方案是对云计算系统所拥有的计算节点服务器资源而言的，每个物理服务器上的 VM 都能通过其所在的计算节点，将与外网服务器交互的流量信息直接转发到外网上，而无须通过类似 I 版或 H 版中的 Neutron 中网络节点的集中转发设备的方式才能互通互联网。具体如图 5-2 (a) 中，两台物理服务器上的 VM 通过 eth0 连接交换机进行内网互通，而相应的每个物理服务器上的 VM 可以通过 eth1 直接与外网交互。这种方式在 H 版和 I 版的 Neutron 里还不支持，到了 J 版 Neutron 组件有了变相的支持方案 DVR，和 Multi-Host 有很大的不同，DVR 后续 Neutron 网络相关章节将再详细介绍。

Mirantis 是一家具有俄罗斯背景且专注于提供 OpenStack 相关的技术支持的硅谷公司，其官网有很多 OpenStack 的学习技术资料。对于 Nova-network 的底层通信流程有一篇博客 ([https:// www.mirantis.com/ blog/ VLANmanager-network-flow-analysis/](https://www.mirantis.com/blog/VLANmanager-network-flow-analysis/)) 讲解得非常详细，这篇文章里对 Nova-network 的通信流程和几个经典场景做了很好的分析与总结，附录 B 对这篇博客进行了翻译；在 OpenStack 使用 Nova-network 提供网络服务时，如果一个租户有两个子网，那么这两个子网之间应该内网可以直接互通，这点需求会带来不少复杂的技术点，并且在上述这篇技术博客里的场景 6 有相应的描述；场景 6 后面的那种拓扑下是 Nova-network 功



两个 VLAN 内的 VM 是无法直接使用内网互通的，需要在租户间配置策略路由：

- tenant1: Nova secgroup-add-rule default TCP 1 65535 10.1.0.0/24
- tenant2: Nova secgroup-add-rule default TCP 1 65535 10.0.0.0/24

这样两者之间的 TCP 通信就被放行。

但是图 5-2 (a) 和图 5-2 (b) 的区别在于图 5-2 (b) 中左边物理服务器只有 VLAN100 的 VM，所以这里 br102 是没有必要建立的。那么如果采用这种添加安全组规则的方式，当 VM\_1 里执行 Ping 10.1.0.4 即 Ping VM\_6 时无法通信，因为报文被默认路由转发到了外网。

其实这篇博客里提及的解决方案只考虑了软件解决方案，也可以结合连接服务器的物理交换机，采用硬件网络设备的方案来解决；第一种结合硬件交换机的方案是在同样如图 5-2 的拓扑下，想实现当 VM\_1 通过内网直接和 VM\_6 直接通信，不能通信的原因是因为在所有服务器的物理网卡 eth0 上有 VLAN 隔离，那么可以在物理交换机上利用灵活 QinQ 功能将 VLAN 100 通往 VLAN102 的报文进行 VLAN Tag 的 VLAN ID 进行修改转换，当响应报文从 VLAN102 回复到 VLAN 100 时需要灵活 QinQ 对 VLAN ID 进行逆转换，这样就能实现两个 VLAN 内 VM 的互通。这时候需要考虑当图 5-2 (b) 中 VM\_4 Ping VM\_6 时需要交换机支持发夹 (Hair-pin) 功能，即交换机连接 VM\_4 和 VM\_6 所在服务器的端口能支持 VM\_4 的 Ping 报文进入交换机后完成 VLAN ID 修改后从该端口再被转回来。

再有一种方案就是结合厂商的硬件交换机将所有报文都 Offload 到网络设备交换机 TOR 上，虚拟机的通信报文 Offload 到物理交换机后，统一在交换机上做控制，那么对这时候让两个 VLAN 间的报文进行通信就变得可控性强些，这个时候可以将物理服务器所连的端口都配置成报文转出时带 Tag 的 Hybrid 口，然后不同 VLAN 间的报文进行允许跨 VLAN 的三层转发，并且同时 VLAN ID 被修改成目的主机所在的 VLAN，这样就能实现同一租户不同 VLAN 的转发；这种方案也

利用了 Nova-network 不支持地址重叠的缺点，不然在物理交换机上针对有相同网段的两个 VLAN 的情况，实现上述方案会是个很大的难题。

这种方案也有其限制，一方面需要硬件对需要的功能有所支持，因为灵活 QinQ、三层转发或发夹的转发并不是所有交换芯片都支持，而且不同网络设备的相关功能表项大小或支持度也不尽相同；另一方面如果每次配置新的租户就需要手工去更改交换机配置是不可行，需要在 Neutron 添加先关的代码，但是现在交换机厂商的不同 CLI 命令是不同的，也暂时没有统一的 API 接口，这个时候如果所有厂商都是统一的 OpenFlow 交换机其优势就不言而喻了；从这个角度也说明了在 SDN 中统一南向接口的意义，虽然这个统一的标准不一定是 OpenFlow 或者说可能不是只有一个唯一的标准。

所以 Nova-network 的优势就是拓扑简单、支持 Multi-Host 方案、报文转发流程少而转发效率损耗较低等优势，但是也存在功能较少、只支持二层转发、不便于扩展、不支持重叠地址的多个私有网络、不能提供网络高级服务(FWaaS、LBaaS、VPNaaS)、隔离方式只支持 VLAN 所以租户数最多 4094 个等缺陷，而 VXLAN 和 NVGRE 方案可以支持更多租户之间的隔离，但是存在着相对于 VLAN 隔离方案性能稍低、产品方案成熟度有待验证的问题。

## 5.2.2 Neutron 网络

Neutron 是 OpenStack 的网络组件，通过 Neutron 的 CLI 或者 Horizon 的页面调用 Restful API 配置了 OpenStack 某个网络功能后，就会调用 Neutron 的 Python 代码的 API，将配置信息下发到底层网络；比如建立 VM 连接某个私有网络时，当 Plugin 是 Open vSwitch 的情况下则生成对应的 Open vSwitch 的规则命令下发到底层 Open vSwitch 上，并将相关的 VM 的虚拟网卡和对应网桥等网络设备等进行互联，底层就提供了相应的网络转发功能的连通链路（如果支持的话则正常下发，有

的功能不支持可能也不报错，比如 H 版对 IPv6 转发的支持)。这些网络功能包括二层 VLAN 或 VXLAN 等租户之间的隔离、Floating IP 之间的三层转发和 NAT 转换、防火墙、VPN 和负载均衡等，根据官网介绍中还有 IDS 和 DC 互联等功能。

图 5-3 是 OpenStack 平台上用 Neutron 网络建立的一个网络拓扑图，如果初学者能配置出如图 5-3 中的网络拓扑且 VM 都能访问外网和租户间相互隔离，说明对 Neutron 的底层网络技术有了入门级的掌握。如果用 Open vSwitch 作为 Plugin 的话需要注意，里面的一个虚拟路由器并不是直接对应创建一个 Open vSwitch，很有迷惑性，在底层其仅仅是在 Open vSwitch 上创建了一个该租户 VLAN 的端口，配置添加了一个三层接口地址，并且新建了一对和这个网段相关联的虚拟 Port。另外对于 VM 直接挂在公网有时候会有需求（即如图 5-3 中 VMA01-CA、VMA02-CA 和 VMA03-CB 等连接网络的方式），这种网络连接方式通过非常少的底层配置，在任意拓扑或隔离技术下都可以很容易实现。

Open vSwitch 是 Nicira（后被 VMWare 于 2012 年收购）一个开源 Apache 2.0 的虚拟交换机软件，最初虚拟机之间的隔离是采用 Nova-network，其底层采用的是 Linux Bridge 技术，虚拟网络的组建和配置非常的不灵活而且很多情况下网络配置信息无法模板化，这些缺陷导致使用 Nova-network 是无法搭建复杂的虚拟网络的；并且在 Nova-network 下对于虚拟网络的调试、监控、故障定位都有一些不方便的情况；Open vSwitch 的出现解决了上述这些问题，因为 Open vSwitch 支持 QoS 功能、镜像功能、CFM 功能、netFlow 功能等特性，而且它还可以支持 OpenFlow 协议以比较容易的方式和某些 SDN 控制器的网络融合；Open vSwitch 里最重要的是采用了一套用高可移植性 C 语言开发的开源实现，导致其在网络虚拟化领域里得到了极大的重视，Open vSwitch 现已经可以在 FreeBSD、Windows 甚至在 non-POSIX 等嵌入式平台里运行了。

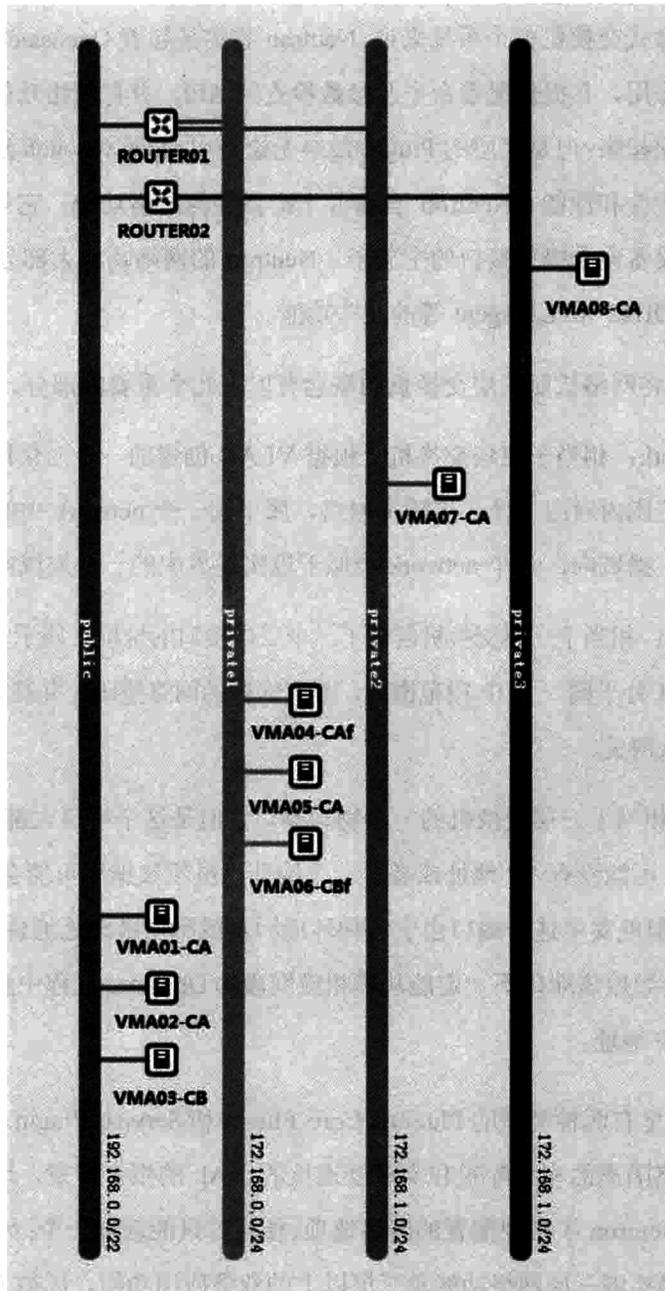


图 5-3 Neutron 搭建某租户的网络拓扑实例

相对于盒式交换机整个系统来说, Neutron 其实是起着 Openstack 网络系统中管理平台的作用, 其提供配置命令及参数检查的 API, 并把网络功能用一种业务上的逻辑组织起来; 但是底层的 Plugin 最终无论是用 Open vSwitch 还是硬件交换机来提高稳定性和性能, Neutron 自身并不提供任何网络功能, 它只是一个负责为底层网络设备向上提供接口的空架子。Neutron 的网络功能大部分是 Plugin 提供的, 除了 DHCP 和 L3 Agent 等的某些功能。

Neutron 将网络按照三层交换机的概念有以下几个重要的部分。

- **Network:** 相当于三层交换机上根据 VLAN 创建的一个三层网络接口, 该接口范围内对应一个二层隔离网络, 属于同一个 network 中的 VM 处于同一个广播域内, 一个 network 类似于现实世界中的一个局域网。
- **Subnet:** 相当于三层交换机创建了一个三层接口的地址, 属于一个 subnet 中的 VM 处于同一个 IP 段范围内, 具有同样的网络掩码, 并且具有相同的网络默认网关。
- **Port:** 相当于三层交换机的一个物理端口, 但是这个端口大都有一个 MAC 地址, 可能没有 IP 地址或者有一个地址, 虽然数据库可能会指定一个 IP 地址, 但现实中这个端口由于各种环境因素或网络链路连通性异常的原因, 可能会导致该端口不一定能从其相应网段的 Dnsmasq 进程中获取到该网段中的 IP 地址。

Neutron 里有两种类型的 Plugin: Core Plugin 和 Service Plugin。Core Plugin 完成二层网络隔离的相关配置和实现互通性的 VM 的报文转发, 现在大约有近 20 种, 属于 Neutron 中必须配置的网络选项, 使用时只能选择一个; 另一种 Service Plugin 主要是实现三层网络功能及三层以上的业务应用功能, 比如 Meter/Router/FWaaS/LBaaS/VPNaaS 等底层所用的 Plugin Agent; 每种应用的底层 Service Plugin

也可能会有多种, 和 Core Plugin 一样, 基本只能选择其中的一个来使用。Neutron 支持的 Core Plugin 有以下种类:

```
Open vSwitch Plugin
Cisco UCS/Nexus Plugin
Linux Bridge Plugin
Modular Layer 2 Plugin
Nicira Network Virtualization Platform (NVP) Plugin
Ryu OpenFlow Controller Plugin
NEC OpenFlow Plugin
Big Switch Controller Plugin
Cloudbase Hyper-V Plugin
MidoNet Plugin
Brocade Neutron Plugin Brocade Neutron Plugin
PLUMgrid Plugin
Mellanox Neutron Plugin Mellanox Neutron Plugin
Embrane Neutron Plugin
```

还有一些非官方维护的:

```
OpenContrail Plugin
Extreme Networks Plugin
Ruijie Networks Plugin
Juniper Networks Neutron Plugin
```

Neutron 支持如此众多种类的 Core Plugin, 那么开发或者部署过程中如何选择哪种 Plugin 就成了技术或方案上的难题, 每种 Plugin 都有其优势和劣势, 但是如果每种 Plugin 都部署环境来测试对比下相关性能数据, 这种做法几乎是不太可能的, 因为除去时间和精力上的问题, 还有一个原因就是测试方案还需要大量投资到服务器和相关网络设备的硬件上, 这个对于调研工作者来说是一笔不小的花费; 对于实际中的研发工作的选择来讲, 就有一些因素可以参考, 比如 Plugin 的被采纳的使用情况、对网络功能的支持程度、实际部署中的容易程度和后续实施方案中是否会被特定厂家设备所绑定等因素。

对于这些 Plugin 的在实际使用部署中的具体汇总情况, 会对调研者选择使用哪一种 Plugin 有一定的参考价值, 因为当很多云计算研发企业或云计算项目实施方案中, 某种 Plugin 被大量使用的时候, 使用者共同积累的使用经验会比较多;

这时候使用者一旦遇到问题，可以通过网络社区或技术交流等途径及时得到解决问题的答案或技术方案；另外使用者对该种 Plugin 的使用时间越久，被纠正的缺陷和优化的技术点就会越多，那么该 Core Plugin 也就越成熟，并更适用于实际的部署方案的需求；在 <https://www.OpenStack.org/summit/portland-2013/session-videos/presentation/OpenStack-user-committee-update-and-survey-results> 里有一些有意思的统计信息，比如不同用户对不同 Plugin 的选择大致比例如图 5-4 所示。

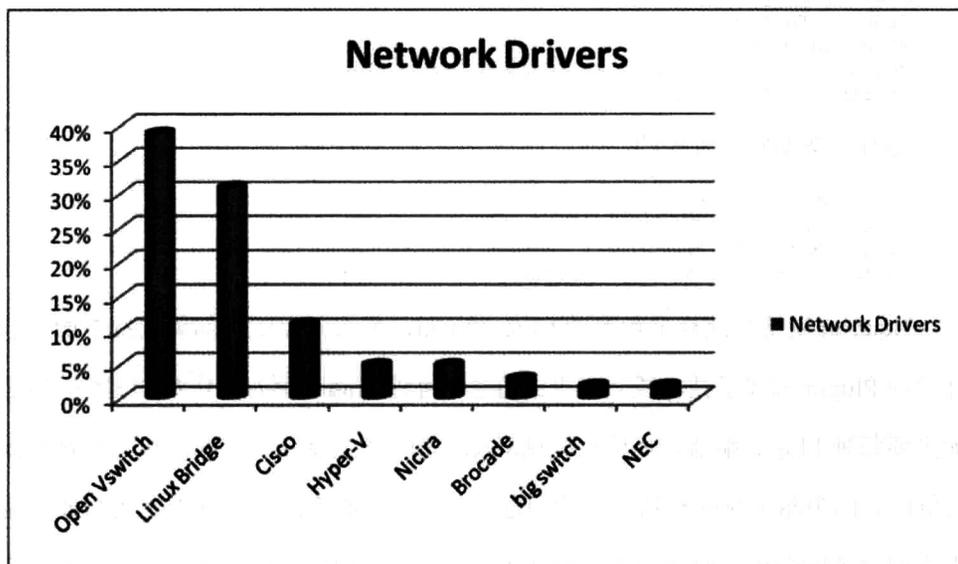


图 5-4 Neutron 中 Core Plugin 的使用统计

通过图 5-4 可以看出在实际部署和调研中 Openstack 的 Neutron 组件 Open vVwich 是被使用最多的 Core Plugin，其次是 Linux Bridge 以其简洁高效赢得了大批量的客户，这两个 Plugin 赢得约市场上 70% 的用户；笔者觉得主要原因是两者都是用开源软件实现的，因为软件代码的开源是一个趋势，在实施中投资会相对节省，尤其是针对某些客户已有用于部署云计算的服务器设备和网络设备的情况下，这种方案更容易被接受；从这一点上也体现出网络研发相关软件的开源也越来越多的趋

势，其中甚至盛科的交换芯片 SDK 源代码也已经开源，这个是网络设备研发界非常巨大的进步；并且云私有云计算的客户和云计算网络技术开发者的方案都不太想被一些厂商的网络设备所绑定，所以白牌交换机在云计算网络里将继续会发挥优势。另外一种需要被提及的 Plugin 是 ML2 (Modular Layer 2)，从 Havana 开始支持，其出发点是考虑现实中这些 Plugin 所对应的设备，在现有的数据中心物理网络中的使用是可以同时存在的情况，所以其目的是想实现原来各种 Plugin 同时协调工作，并且通过非常少的初始化和持续努力且很容易地添加对新 Plugin 的支持，具有良好的扩展性支持多种 Agent 实现多种隔离技术，又可为 L3 Agent 的扩展整合提供一种 Service Plugin 的机制；这种机制为 Neutron 多种隔离技术并存或从 VLAN 扩展到 VXLAN 提供了思路。ML2 有两个重要的概念：Type Drivers 和 Mechanism Drivers，Type Drivers 是从 Neutron 的网络拓扑结构来为用户分配网络、提供网络验证、管理可用网络类型和维护网络的相应网络状态，具体来讲就是决定了 Neutron 网络的拓扑形式，比如可以是 Local、FLAT、VLAN、GRE 和 VXLAN 等几种网络类型。Mechanism Drivers 主要靠 Core Plugin 来实现，用于网络工作机制的管理和底层网络隔离的实现；支持的主要有 Open vSwitch Agent、Linux Bridge Agent、Hyper-V Agent、L2 Population、Cisco Nexus 和 Arista 等。后续 Juno 发布的 DVR 即是基于 ML2 的 Open vSwitch 这种 Plugin 来实现。ML2 可以有效提高网络的可靠性和性能，可以预见是未来生产环境中非常重要的一个特性，目前来看 ML2 Plugin 是有前景的 Plugin，但是其在实际部署中对云计算厂商来将带来了一定的复杂度，实际业务有这种需求的情况可能开始于 Neutron 网络的隔离技术的升级服务。

Neutron 中各种 Plugin 的性能和稳定性如何？这个暂时没看到确切的比对测试数据，但是毋庸置疑的是，硬件的 Plugin 使用方式合理的话应比软件方式的 Plugin 性能上有所提升，毕竟购买硬件设备是有相应成本上的投资。仅从 Linux Bridge 和 Open vSwitch 两种软件方式的 Plugin 上来说，肯定也会有所区别，因为

毕竟不同的技术环境里，数据报文虚拟机和物理服务器转发出去的流程有非常大的区别，而且 Open vSwitch 自身的隔离方式不同（比如用 VLAN、NVGRE 或 VXLAN 等），其数据报文传输性能的测试结论也肯定不尽相同，仅从性能上来说 Linux Bridge 应更好一些，但是从功能丰富性来说 Open vSwitch 是趋势。

至于 Neutron Plugin 是路由器还是交换机，这个也无法明确，确切来说是看所用的 Plugin 以及其能提供的功能，毕竟 Neutron 有 L2 Agent 和 L3 Agent，这些功能对于底层的 Plugin 需要相互协作才能完成二层转发和三层路由；但是选择的 Plugin 满足自己的需求才是最重要的，至于工作在几层似乎是个多余的问题，应该要问的是自己的知识储备和需求是什么，两者的结合点在哪里。

### 5.2.3 OpenStack 存储网络

网络存储技术（Network Storage Technologies）技术有很多种，其中存储区域网络（Storage Area Network, SAN）是将服务器连接到光纤通道上与存储设备互联，用于存储数据流量通常情况下都比较大的情况。存储网络的技术主要包括 NAS（Network Attached Storage）、SAN（Storage Area Network）和 RAID5（Redundant Array of Independent Disks），以及 iSCSI（Internet Small Computer System Interface）。

NAS 是通过以太网将存储节点上的文件系统作为计算节点的一个网络访问元素来使用，存储节点的文件系统可以是 Network File Systems）和 CIFS（Common Internet File Systems）；SAN 则是将 NAS 中的以太网换成了光纤（Fiber Channel）网络，让计算节点看来是本地的一个存储系统一样，因为光纤带宽较高，这里需要提及的一个标准是 FCoE（Fiber Channel over Ethernet），是通过增强的 10Gbps 以太网技术在支持 FCoE 的交换机的环境下，使用隧道协议允许将光纤通道的信息封装到以太网信息内，将本来运行于光纤通道的交互信息和数据信息通过以太网来传输，融合 LAN 和 SAN 的数据类型，减少线缆设备及耗电、网络收敛便于管理，但是这种技

术毕竟是一种新技术，为了保证 FCoE 在以太网中不丢包引入了 DCB (Data Center Bridging) 的概念，包含 IEEE 802.1Qbb Priority-based Flow Control (PFC)、IEEE 802.3bd Frame Format for RFC、IEEE 802.1Qaz Enhanced Transmission Selection (ETS) and Data Center Bridging eXchange (DCBX)、IEEE 802.1Qau Congestion Notification、IEEE 802.1Qbh Port Extender 等一系列技术标准，需要购买新的网络设备来支持，而新设备的稳定性相对于现在成熟的方案值得推敲，所以在国内某些大的互联网公司的数据中心是对该技术有排斥的。RAIDS 是将一系列物理磁盘作为一个存储单元，数据可以在这个单元里复制和分发，并且用其冗余和高可靠性能对计算节点的错误做到容忍；iSCSI 并不能算是一种存储网络类型，只是一种基于 IP 技术的接口标准，由 IBM 公司研究开发的，包括一套指令集，可以与以太网技术或光线网络进行融合，比如 SAN 就是使用了 iSCSI 技术和光纤网络，达到硬件成本低、扩展性强、操作容易和高性能的优势。

另外，OpenStack 存储里需要提及的技术是 Ceph (<http://ceph.com/>)，这是一种大容量、通过负载实现高性能和高性能的分布式文件系统，也是 OpenStack 里寄予厚望的开源存储解决方案，UnitedStack 的生产环境里采用了 Ceph 技术。

---

## 5.3 Neutron 底层网络原理

OpenStack 的安装首先需要考虑物理服务器的角色分工，在一个通过若干个交换机等网络设备互联的服务器集群中，哪些服务器做控制节点、哪些服务器做网络节点、哪些服务器做计算节点、哪些服务器做监控服务器和哪些服务器部署服务的管理节点等都需要提前规划好，这样才能在物理网络中实现管理网络 (Management network)、数据网络 (Data network)、外部网络 (External network)

等角色；其中管理网络是为 OpenStack 的控制节点管理其他计算节点和网络节点使用的网络，通常实验环境下管理网络可以与外网或内网归并成一个网络。数据网络其实就是内部网络，主要负责云计算中部署的 VM 之间通信的数据流传输；另外还有一个访问 OpenStack API 的 API network，这个通常和外网合并到一起。管理网络、内部网络、存储网络和外部网络的概念是针对整个 OpenStack 的，底层网络即使采用 Nova-network 方案也有这四个概念，需要提及的是这四种网络的划分只是逻辑的，至于对应物理网络里是否是一一映射的不太重要，只要逻辑对应清晰即可。图 5-5 是 OpenStack 官网资料里的几种网络的拓扑示意图。

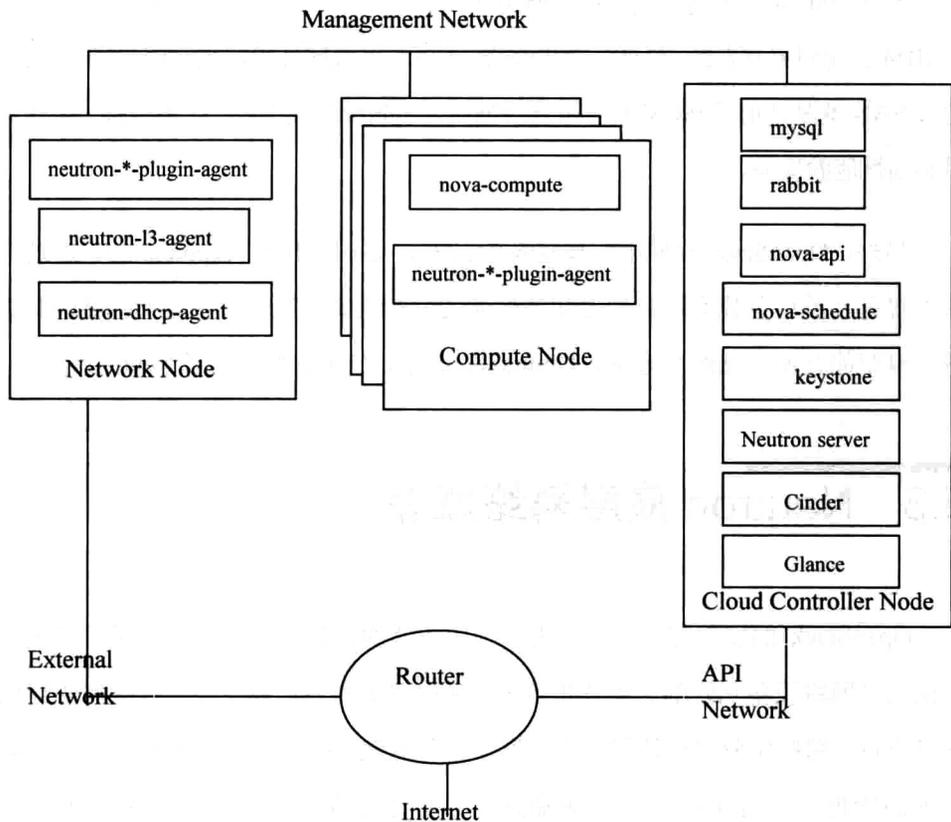


图 5-5 OpenStack 节点划分网络示意图

### 5.3.1 Neutron 组件的构成

从 OpenStack 的网络组件来看，Neutron 其实并没有实现任何网络功能，仅是对底层 Plugin 的配置提供 API；即 Neutron 对 OpenStack 的虚拟网络来说仅是提供了配置的接口并下发配置信息到底层网络上，并没有提供二层转发、三层路由、安全和 NAT 等任何网络功能，所有的网络功能都是由底层比如 Open vSwitch、Linux Bridge 和 IPtables 等网络模块来提供。在 Neutron 中有几个重要的 Agent 来实现对以下这些底层软硬件的配置和控制：

- 各种网络服务的 Agent（名称类似于 Neutron-\*-Agent）与 Neutron-Server 之间主要通过 RPC 消息进行交互。上层调用 API 传进来的参数用于对 Neutron 的 Plugin 进行配置和操作，Plugin 会根据这些配置和操作做一些处理（如增删改数据库），如 Neutron-Server 需要通知信息到 Agent 的话，将通过 RPC 消息传递到 Agent 侧，完成对底层软硬件的配置和配置信息数据的下发；
- DHCP Agent 的用途主要为处于网络中的 VM 获取 IP 地址提供 DHCP 服务，但 DHCP Agent 进程本身并不具备 DHCP 能力，它是利用 Dnsmasq 进程完成 DHCP 的功能，通过对 Dnsmasq 配置和管理来为 VM 提供灵活的 DHCP 服务和 DNS 服务的；
- L3 Agent 主要是为所有 VM 访问外网提供三层路由服务，包括为处于不同网段的 VM 之间提供路由服务、为需要外网服务的 VM 提供 NAT 服务等；但 L3 Agent 并不完成路由功能和 NAT 功能，而是通过解析参数转换成底层 Plugin 可识别命令的操作来对底层完成配置和管理，真正实现路由功能和 NAT 功能的应分别是底层的 Core Plugin 和 IPtables。

Neutron 能支持非常多的 Plugin，从可以提供网络功能的丰富性和项目实施

的方案成本来计算，笔者认为 Open vSwitch 是比较合适的一种，当然也已经有很多云计算公司在实际中已经采用了这种 Plugin，比如 UnitedStack 等。但是这种 Plugin 也不是十全十美的，毕竟是开源软件实现的 Plugin 方案，所以其性能和稳定性依赖于软件的具体实现和 Openstack 部署的设备系统特性，而且对不同的业务流或应用场景下测试出的性能指标数据可能也不同。

Neutron 的安装可以采用每个组件分步安装的方式，也可以采用某些厂商的一键部署的方式，比如前文提及的 Devstack 和 RDO 等；分步安装对于熟悉 OpenStack 的每个组件以及每个组件的作用和组件之间的关联会起到很好的作用，但是比较耗时和容易出错，尤其对于对 OpenStack 还没有清晰概念的初学者来讲是比较痛苦的，所以先对 OpenStack 安装成功再详细学习无疑极大增加了初学者的信心，建议初学者从 RDO 的安装方式入手，因为 RDO 的系统包和 OpenStack 包都相对比较稳定，安装文档和原理文档比较丰富；安装前需要先配置好 yum 源，这里建议将 RDO 的源做成本地源，这样能提高安装的速度以及因公司外网因素导致的安装不稳定问题。下面以两个物理节点为例子（一个节点只做计算节点，另一个节点是计算节点、网络节点和管理节点的混合）简要介绍一下 OpenStack 的安装过程，附录 C 给出了 RDO 安装的应答文件中网络部分配置的全部内容。

两台节点的物理机拓扑图如图 5-6 所示。

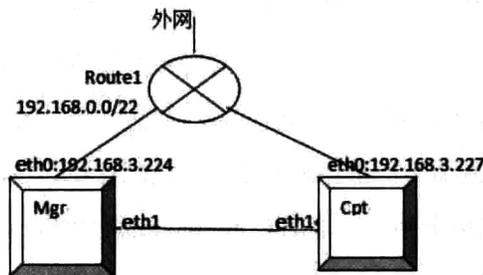


图 5-6 OpenStack 安装物理机拓扑举例

(1) 配置好 yum 源后, 用 `yum install openstack-packstack` 安装 OpenStack 的安装包。

(2) 在安装 OpenStack 后, 用 `packstack --gen-answer-file=ans.conf` 生成 RDO 安装需要的应答文件, 即 OpenStack 各个组件安装的配置文件; 附录 C 是 I 版的 RDO 某个版本上针对拓扑图 5-6 的一个配置文件中网络部分内容的实例; 这里面做的修改主要如下几方面:

- 管理节点、计算节点和网络节点的安装物理机 IP 地址, 管理员 admin 的账户密码;
- 各个组件安装与否的选择;
- 网络的使用类型是 Nova-network 还是 Neutron, 如果是 Neutron 还需要附录 C 中的详细配置;
- 如果网络选择了使用 Neutron 组件, 那么还需要配置底层的 Plugin 是什么, 比如选择 Open vSwitch 等, 以及网络采用的隔离方式, 包括 VLAN、VXLAN 和 GRE 等, 选择了隔离方式还需要配置相应的内网端口、网络节点的外网端口、支持的租户数目及租户使用的隔离号 ID 范围等; 这几种隔离技术都可以直接配置好后安装, 但是配置选项较多且比较复杂, 不懂底层网络的安装者往往配置冲突或缺少配置项, 导致 OpenStack 安装失败或安装成功但是 VM 的网络异常甚至 Neutron 无法使用;

(3) 配置文件修改完毕, 执行 `packstack --answer-file=ans.conf` 命令会根据 ans.conf 里的配置安装 OpenStack;

(4) 根据 (2) 中步骤对网络的配置, 修改物理机的网络配置, 让 OpenStack

的网络和物理机的互连网络成为一个统一的整体。

### 5.3.2 Neutron 网络的隔离

这里分析一下当 RDO 采用的是 VLAN 隔离方式时,连接物理服务器的交换机是否需要支持 VLAN 功能?答案是不需要的,从前文中介绍的网络传输设备的相关章节中可以知晓,有些物理设备是不支持 VLAN 功能的,虽然这类设备现在不是太多,但这部分设备也确实存在的。首先在服务器内部,不同的网段等资源是通过 namespace 的方式进行隔离的,所以 Neutron 可以支持网络地址重叠等功能,这种隔离方式到了物理设备上对应的就是需要 VLAN/VXLAN 等方式的隔离,从而实现服务器内部和服务器外部的统一隔离;VM 的数据报文转发出物理机时,采用 VLAN 方式时,会带上 VLAN Tag 来区分对应的隔离域,如果连接物理服务器的交换机不支持 VLAN,会对带有 VLAN Tag 的报文进行透明传输,直接通过 MAC 地址或者广播转发出去到所有物理服务器上,此时相应的目的服务器里肯定会被接收到该报文,而对于不在同一个 VLAN 的设备接收到该报文时会因 VLAN 过滤把报文丢弃;而在一个 VLAN 内的设备接收到该报文后才进行后续处理;对于支持 VLAN 的连接物理服务器的交换机来说,需要配置交换机让带 VLAN Tag 的报文通过交换机并继续带着 Tag 转发到相应的对端,通常的实现方式就是在交换机上建立相应的 VLAN 并将相关端口配置成 Trunk 模式,这样报文就可以带着 VLAN Tag 达到目的地的物理服务器。在实际 OpenStack 的部署中,有的厂商的交换机受硬件芯片限制并不能支持所有的 4096 个 VLAN,而只能支持其中的一部分,或者另外一种常见情况也有可能是物理交换机正在生产环境中使用,不允许对其进行更多的配置,只允许使用现有已经配置的一个 VLAN 的范围,这样就会产生一个问题:就是如何将 OpenStack 虚拟网络中的 VLAN ID 范围和实际物理网络中的 VLAN ID 相对应?这就用到了 VLAN-TranslationTranslation 的功能,即将 OpenStack 所用的

VLAN 隔离号范围和对物理网络支持 VLAN 的范围的值进行一一映射，在附录 C 中对应到配置文件里配置项对应的是 CONFIG\_NEUTRON\_OVS\_VLAN\_RANGES 和 CONFIG\_NEUTRON\_OVS\_Bridge\_MAPPINGS。从上面分析可以看出，Open Stack 用 VLAN 做隔离时，物理服务器的互联交换机可以支持 VLAN，也可以不支持 VLAN 功能，但对于支持 VLAN 功能的交换机需要做好配置以使得 OpenStack 的虚拟网络和物理网络有一个统一的映射和融合。当交换机不支持 VLAN 时可以仅靠二层 MAC 地址进行转发，这个实现的前提是所有 VM 的 MAC 地址都是不同的；有的云计算厂商将 VM 产生 MAC 地址产生机制做了修改，不同租户可能产生相同 MAC 地址的虚拟机，这种情况不支持 VLAN 的交换机就无法使用了。所以当所有 VM 的 MAC 地址都不同时，也可以通过一些规则仅通过 MAC 地址来对不同的租户进行隔离和通信。

那么采用 VXLAN 或 GRE 隔离方式时是否需要购买支持 VXLAN 或 GRE 的网络物理设备呢？比如交换机或路由器。答案在 Open vSwitch 作为 Plugin 的环境下依旧是不需要。因为在 Open vSwitch 上已经实现在其端口上对 VXLAN 或 GRE 的封装及解封装，当 VM 的业务数据报文转发出物理服务器时，报文就是正常的带 VLAN Tag 的 IP 报文，会根据 MAC 地址等字段进行转发到目的地，但是用 Open vSwitch 作为 VTEP 的方式，确实性能和稳定性方面值得商榷。如果需求中有 VXLAN 网络的云计算环境和 VLAN 隔离的云计算网络环境进行互通和隔离，那么在这种拓扑下需要借助物理交换机来做些 VLAN Translation 工作。

图 5-7 是按照上述配置方法针对两个计算节点和一个网络节点组成的底层网络拓扑图。从图 5-7 可以看出，采用 Open vSwitch 为 Plugin 的安装时，底层拓扑中用到了 Dnsmasq 来实现 DHCP /DNS 功能，用 IPtables 规则实现了 NAT、防火墙和安全组功能，用 Open vSwitch 的流表实现了二层和三层转发的相应功能，包

括根据 MAC 地址转发、根据 IP 地址报文转发、VLAN-Translation 和 ARP 报文的转发等。

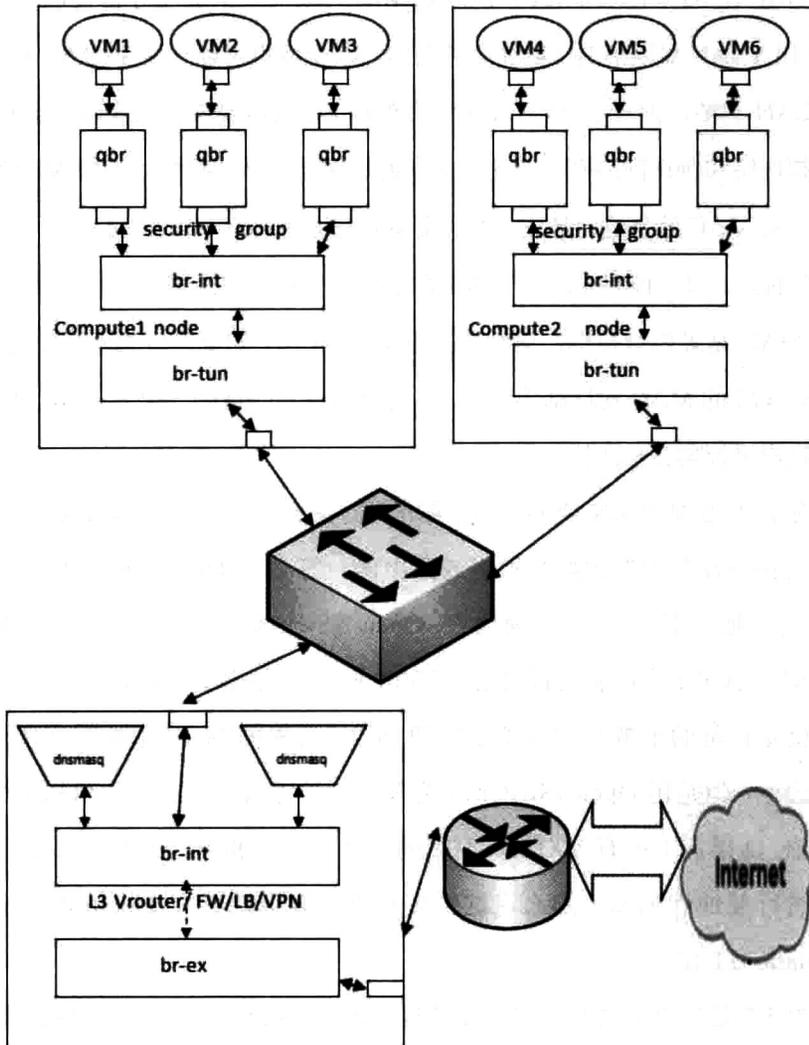


图 5-7 I 版本 Neutron 底层网络拓扑实例图

采用 VLAN 方式做隔离时底层转发报文转发原理是同一个租户下在相同路由

器上且在同一物理机上的 VM 直接通过 Open vSwitch 进行二层通信, 数据报文无须到物理机外面转发, 只有同一个租户下在相同路由器连接的同一个私网网段里, 但分布在不同物理机上的两个 VM 之间通信, 才会通过服务器之间的物理内网进行传输; 而不同租户下或同一租户不同路由器下的两个 VM 通信, 都需要通过网络节点进行外网通信, VM 与外网互联网设备的通信同样需通过网络节点进行传输。

可以看出, OpenStack 的 VM 在 Neutron 提供的 VLAN 隔离的功能下可以实现如下通信方式:

(1) 对于同一个租户的同一个路由下同一网段的 VM 之间可以通过内网通信, 如果都有 Floating IP 也可以通过 Floating IP 进行通信;

(2) 对于同一个租户的同一个路由下不同网段的 VM 之间可以通过内网通信, 如果都有 Floating IP 也可以通过 Floating IP 进行通信; 一个路由器下可以建立多个网络, 每个网络又可以创建数个子网, 这样做的目的是为了便于在对原来网络无任何影响的前提下对网段地址范围大小进行扩充, 以解决内网地址耗尽的问题;

(3) 对于同一个租户的不同一个路由下无论内网是否相同, 各 VM 之间都不可以通过内网直接通信, 如果两路由器下的 VM 都有 Floating IP 地址, 那么可以通过 Floating IP 进行通信;

(4) 对于不同租户下的无论内网是否相同的 VM, 它们之间都不可以通过内网通信, 如果都有 Floating IP 可以通过 Floating IP 进行通信;

(5) 两个租户下的 VM 的网络之间或同一个租户不同路由器下的 VM 网络之间, 当配置开启了 VPN 功能时, 那么可以实现用内网通过 VPN 互通。

那么为什么不将 VM 直接挂到 Open vSwitch 上, 而是将先连接到 Linux Bridge

上,再通过 Linux Bridge 连接到 Open vSwitch 上? 这个从网络原理上来说也是可行的,但是这样 IPtables 就无法继续使用,需要将 IPtables 实现的 NAT 和安全功能转移到 Open vSwitch 上来实现,仅从实现安全和 NAT 功能上来说 Open vSwitch 可以胜任,但是 Open vSwitch 毕竟还需要进行 VLAN-Translation、IP 报文和 ARP 报文转发以及 VXLAN 或 GRE 报文的封装与解封装,这就会增加 Open vSwitch 功能配置的逻辑复杂性;随着 Open vSwitch 功能的日益强大和完善,将这些功能集中到 Open vSwitch 来实现和控制,笔者认为是一种 OpenStack 底层网络的技术趋势。另外,将 VM 连到 Linux Bridge 上,然后多个 Linux Bridge 再连接到汇聚用的一个名为 br-int 的 Open vSwitch 上,再通过路由及 NAT 连接到外网的三层架构非常符合实际中的三层物理网络拓扑的设计,即接入层、汇聚层和核心层。这种方式在实际物理网络中的使用经验和教训已经积累了很多,可以根据 OpenStack 虚拟网络的需要进行适当借鉴,尽量减少虚拟网络发展路途中的未知缺陷。

### 5.3.3 Neutron 网络的互通

创建 VM 的时候,底层需要做很多工作才能保证 VM 可被控制地访问外网或实现同内网间 VM 的通信,这里面用到了前文所述的很多的技术或机制。这里根据 OpenStack 的 I 版下的 Neutron,以 Open Vswitch 为 Core Plugin 下的 VLAN 隔离为例,如图 5-7 中,来详细剖析一下 VM 是如何从计算节点连接到网络节点,以及如何通过网络节点联通外网的。具体流程为:

(1) 当一个 VM 被创建的时候,其 VM 的网卡在计算节点来看是一个虚拟的以太网 TAP 设备,该网络虚拟设备和普通的网卡一样,有 MAC 地址等信息;当 VM 创建好后可以在 VM 所在的计算节点里,根据 VM 网卡的 MAC 地址寻找对应的 TAP 设备;TAP 设备的命名规则是 tap 作为命名字符串的前三个字符,后

面再附加该设备对应 UUID 前 11 位，名字字符串总共 14 个字符；

(2) VM 网卡对应到计算接点的虚拟网卡 TAP 设备并没有直接连接到 Open Vswitch 的虚拟交换机上，而是中间连接了一个网桥 qbr；原因是租户的安全组规则需要使用 IPtables 规则来实现，如果直接连接到 Open Vswitch 的虚拟交换机上则会导致安全组规则的功能失效；

(3) VM 网卡对应到计算接点的虚拟网卡 TAP 设备直接作为 qbr 网桥的一个端口成员，而不是通过其他的设备或虚拟网络；

(4) 租户的网桥设备 qbr 通过创建一对 veth pair 的虚拟端口连接到直接连接到 Open Vswitch 的虚拟交换机上，qbr 侧的端口通常以 qvb 作为命名字符串的三个字符，而 Open Vswitch 的虚拟交换机上的端口通常以 qvo 作为命名字符串的前三个字符；同 tap 设备命名，qbr、qvb 和 qvo 三种虚拟设备都是设备类型三字节作前缀，后面添加 UUID 的前 11 位字符串来命名；并且，qvo 设备在虚拟交换机上都是配置带有 VLAN ID 的，这里也是不同租户的业务报文在 VLAN 隔离技术下被打上 VLAN Tag 的地方；

(5) qbr 网桥连接的 Open Vswitch 的虚拟交换机通常称之为 br-int，代表汇聚层设备的意思，与物理网络中的汇聚层交换机作用非常类似，而 qbr 则相当于实际物理网络中的二层交换设备；

(6) 虚拟交换机 br-int 是每个计算节点和网络节点都需要存在的，而且 Openstack 的 Neutron 组件安装部署完毕后，就会被自动创建；在 br-int 上，从 br-tun 方向转发过来的报文，需要实现 VLAN Translation 功能，以完成云计算设备物理内网租户到虚拟网络的 VLAN ID 之间的映射；

(7) 虚拟交换机 `br-int` 并没有直接将计算节点作为内网的物理端口作为其端口成员，而是通过一对 `veth pair` 连接了另一个虚拟交换机 `br-tun`；该虚拟交换机 `br-tun` 和 `br-int` 一样，也是在每个计算节点和网络节点都需要存在的，而且 Openstack 的 Neutron 组件安装部署完毕后，就会被自动创建；

(8) 虚拟交换机 `br-int` 和虚拟交换机 `br-tun` 之间的 `veth pair`，位于 `br-int` 上的端口通常命名为 `int-br-tun`，而位于 `br-tun` 上的端口通常命名为 `phy-br-tun`；这对 `veth pair` 也是在系统部署时自动被创建并配置好的；

(9) `br-tun` 的命名是在部署时可以任意提前指定的，只是 `br-tun` 比较符合其作用；因为该虚拟交换机直接将计算节点作为内网的物理端口作为其端口成员，通过 `br-tun` 虚拟交换机，报文在不同隔离技术下将被封装到不同的隧道里，比如 VLAN 隔离是进行 VLAN Translation 的作用，将从 `br-int` 方向转发过来的报文，完成从租户虚拟网络到云计算设备内网的 VLAN ID 之间的映射；而在 VXLAN 或 GRE 的隔离技术下，则是对隧道报文进行封装和解封装的作用；并且这些功能的配置和添加计算节点内网物理端口作为其端口成员都可以通过配置，完成部署安装 OpenStack 时自动完成相关的配置；

(10) 在图 5-7 的 VLAN 隔离下，网络节点的虚拟交换机 `br-tun` 和结算节点的基本一致；而网络节点虚拟交换机 `br-int` 配置，则比结算节点多了些相应隔离域内 `Dnsmasq` 进程连接的 TAP 端口；当一个隔离域内有多个网段时，即对应 Horizon 页面一个网络对应了多个子网时，该隔离域内的 `Dnsmasq` 进程就负责对所有这些子网的 DHCP 服务；对于新建网络的 DHCP 服务的 `Dnsmasq` 进程，起初是没有启动的，仅是在第一次创建 VM 的时候，才根据具体的相应配置信息启动 `Dnsmasq` 进程；`Dnsmasq` 的进程在启动以后的运行过程中，不会因创建或删除

VM 而被无故关闭；

(11) 这样创建完毕后，整个云计算网络的租户网络环境就实现了连通；租户可以创建任意的虚拟网络，并且同租户 VM 之间可以通过内网互相通信；

(12) 只有在网络节点才有通往外网的虚拟交换机，通常命名为 `br-ex`，也可以在安装部署时配置其他名称；并且每个网络节点可以有多个通往外网的虚拟交换机，但是这些通外网的虚拟交换机都只能连接同一个外网；即每个网络节点的 L3 Agent 值只可以绑定到一个外网；当有多个网络节点时，可以分属于不同的多个外网，每个外网有多个网络节点时，可以分别各自做各自网络的 L3 Agent 的 HA；

(13) 当建立一个虚拟路由器时，底层对应创建了一个以 `qrouter` 开头的命名空间，在这个命名空间里，创建了两个分别以 `qg` 和 `qr` 开头的两个虚拟网卡，命名以 `qg` 开头的虚拟网卡属于虚拟交换机 `br-ex` 的一个端口，相应的命名以 `qr` 开头的虚拟网卡属于虚拟交换机 `br-int` 的一个端口，这样 `br-int` 和 `br-ex` 才能互相通信，才能连通整个内网和外网；

(14) 每个外网虚拟交换机 `br-ex`，需要一个网络节点的外网物理网卡作为端口成员，才能实现与外网的互通；且运行环境中，`br-ex` 可以配置公网 IP 地址，也可以不配置具体的 IP 地址，VM 里的数据报文照常可以被转发到外网；

(15) 网络节点的虚拟交换机 `br-int` 和虚拟交换机 `br-ex` 之间需要有一对相应 `qrouter` 的 namespace 里的虚拟端口，这对端口命名以 `qg` 开头的虚拟网卡的 IP 地址是虚拟路由器设置公网为默认网关时获取的公网 IP 地址，而命名以 `qr` 开头的虚拟网卡的 IP 地址是租户内网网关的 IP 地址；这样，内网和外网通过该虚拟

路由器的命名空间里的一些 IPtables 和路由实现租户网络与外网的互通；相应 qrouter 里端口命名以 qg 开头虚拟网卡有公网 IP 地址，是 br-ex 无需具体 IP 地址即可实现 VM 与外网通信的根本原因；但是每个虚拟路由器一个公网 IP 地址，确实有点浪费紧缺的 IPv4 地址资源；

(16) 经过 (1) ~ (15) 步骤的连接，在云计算物理内网上建立的 VM 的租户虚拟网络，已经可以和云计算的外网进行互通；另外，云计算租户的内网虚拟内网可以任意命名和使用任意规划的网段，但是云计算的虚拟外网必须和云计算物理设备的外网保持一致，并且虚拟路由器无法同时连接两个有地址重叠范围的两个网络；

(17) 在此基础上，Floating IP 和 FWaaS 服务是通过网络节点相应虚拟路由器对应命名空间里的 IPtables 实现的；LBaaS 和 VPNaaS 也是通过相应命名空间和所需的功能模块来实现的。这样就完成了整个 Neutron 底层网络的互相连通。如果想对报文的转发进行控制，只需在相应网络处理流程处设置合理的控制功能即可实现。

所以，从上述云计算网络环境里 VM 通信的整个报文转发流程来看，云计算的网络技术涉及的技术点比较多，流程比较复杂，需要对照实际的云计算网络环境一步步实践操作才能完全理解和掌握。这些流程中对数据报文通信的性能和稳定性都有着多多少少的影响。

---

## 5.4 Neutron 主要功能

OpenStack 通过 Neutron 提供的网络功能，需要为 VM 之间的通信提供保障，这里的保障不仅仅包括互通，也包括多种类型的隔离技术，以及提供 FWaaS、

LBaaS 和 VPNaas 等高级业务服务的功能。并且随着 Neutron 的发展，支持的功能会越来越多，当然也会势必导致 Neutron 的代码结构和底层架构越来越复杂，而信息技术产品的功能和性能、稳定性大多时候是有相互影响，产品的功能越多，通常会造成性能下降或稳定性降低。

### 5.4.1 互通与隔离功能

不同租户、不同路由器及不同网段内的 VM 可以互通及隔离是 Neutron 的基本功能，互通包括二层互通和三层互通，而这里的隔离基本都是指二层隔离，包含了两层含义，一方面是实现底层网络通过隔离有不同的广播域，另一方面则是指不同租户下或同租户不同路由器下各自的内网区域里，需要实现彼此无法直接互通。同租户的 VM 就像一个公司内的办公 PC 或内网服务器，可以访问私网内的设备以及互联网上的网络资源；同租户不同路由器下的子网类似于分布在不同地理位置的子公司，两者的内网是无法直接互通的，需要通过 Floating IP 才能互相访问，或需要 VPN 才能互访内网的资源；这点类似于不同租户下私网连接的 VM 一样，无法直接用内网地址进行通信，需要借助于 Floating IP 或者 VPN 进行互通。同租户不同路由器网段属于不同的 namespace，因为可以地址重叠。租户 VM 的 Floating IP 地址的 NAT 是通过底层的 IPtables 来实现的，这些转换规则下发在每个租户的 router 对应网络节点的 namespace 里。

当云计算的租户非常多超过 4096 的个数时，采用 VLAN 的隔离方式就无法提供如此多的隔离域；当云计算网络环境中的隔离域数目被耗尽的时候，比如说安装 OpenStack 时 Neutron 只配置了两个隔离域，那么通过 Horizon 建第三个隔离域的时候，即创建虚拟路由器并设置外网为默认网关的情形下，页面会提示错误，因为隔离域已经没有可用资源。而所谓的大二层通常意义是为了 VM 迁移的

需要，一个 VLAN 的物理范围有多大，那么这个大二层就多大，VM 迁移就能迁移有多远，因为 VM 的迁移需要保证其 MAC 和 IP 迁移前后一致，虽然已经提出了三层迁移的概念，但是很多技术上的实现细节还不明确；所以提高隔离域的数目，采用新的隔离方式如 VXLAN 或 NVGRE 等技术就成为了技术趋势；但是由于 NVGRE 是点对点的，效率相对于 VXLAN 来说比较低，鉴于此，VXLAN 成为了实际 Neutron 产品化所用技术的趋势，并且现在很多交换芯片商和设备商也比较推崇 VXLAN 技术，也已经有很多的云计算研发公司采用了这种技术。需要明确的是 VLAN 等隔离技术所做的隔离是指二层隔离，对于三层转发的报文通过 VLAN 等技术是无法阻挡的，前面第 2 章交换机部分也阐述了三层转发接口也是基于 VLAN 做的抽象端口，交换机上的三层转发或三层路由就是为了实现不同 VLAN 的转发以扩大虚拟网络的通信范围；这点可能对很多不熟悉网络的读者来说有点难理解甚至误解。另外类似于物理网络，网络上的设备之间的通信受限于防火墙的规则。

OpenStack 一个租户可以有多个路由器，也可以在没有路由器的情况下将新创建的 VM 直接挂在公网上，这样 VM 直接获取公网 IP 地址与其他设备进行通信，只是这种情形下，FWaaS 将不再对该 VM 的通信流量起安全过滤作用；每个租户的一个路由器可以有多个网络，每一个网络里也可以有多个不同网段 subnet；一个路由器下不能同时连接两个拥有相同网段或有重叠地址范围的子网的网络。每个网络可以有多个子网的原因有两个，一是可以使用多个不连续的网段地址来对内部的 VM 做逻辑划分，二是当一个子网地址范围不够时还想继续使用该隔离域，则可以进行扩展 IP 地址范围用。这个与物理交换机同一个三层接口有多个网段一个道理，无须二层隔离，因为使用的是同一个隔离域，只是不同的网段；

但是无论是使用 VLAN 隔离还是用 VXLAN 等隔离方式,与一个路由器的多个网落不在一个广播域内的情形不同,一个网络里的多个子网属于同一个隔离域。另外,任意网络的子网,对于该网段的 IP 地址可以分成若干个不连续的段,配置给该网段的 dnsmasq 进程,这样可以预留一些地址做他用;这点在外网网段中有很大的使用价值。在 Neutron 环境的一个子网网段地址范围是 10.0.3.0/24,那么配置页面里可以为 DHCP 分配的 IP 范围指定两段不连续的地址范围,如图 5-8 所示。

**激活DHCP:**

**分配地址池:**  
10.0.3.20,10.0.3.100  
10.0.3.120,10.0.3.200

**DNS域名解析服务:**  
114.114.114.114  
8.8.8.8

**主机路由:**

图 5-8 Neutron 子网分配两段可用地址范围

## 5.4.2 防火墙与安全组

Neutron 的安全功能分为两部分,一部分是安全组的规则,另一部分是防火墙规则(Fwaas),两部分的实现底层均是采用了 IPtables,但是安全组的规则是下发在计算节点的物理服务器内,而防火墙规则则是下发在网络节点相应的 vrouter namespace 空间里。所以对于直接挂在外网的 VM,安全组的规则是可以生效的,而防火墙的规则在此种场景下失效。另外对于 Horizon 的界面来说,安

全组是默认打开的，所以有相应的配置页面，但是 RDO 安装的 Openstack 里防火墙则是一种高级服务，默认页面时没有该功能的相应配置项，需要在安装时或安装后开启页面开关，并且配置正确底层的选项才能被使用。在现阶段 Neutron 的安全功能截止到 I 版的实现依赖于 IPtables，路由的 NAT 功能也依赖于 IPtables，然而 IPtables 的规则比较多时非常可能造成报文处理性能的下降，增大报文的网络延迟；当 Core Plugin 为 Open vSwitch 时，对于使用 Open vSwitch 规则和 IPtables 规则哪一种对报文处理性能影响更大，需要测试数据来给出结论，而且不同的应用场景或不同的数据流下的测试结论可能也不同。

和租户 VM 的 Floating IP 一样，租户的防火墙规则也是下发到相应的 Router namespace 里；但是 H 版和 I 版的防火墙实现有一个问题就是租户的防火墙对所有虚拟路由器下的网络都生效，在底层实现对应来看，是防火墙的所有规则都下发到了该租户每一个路由器的 namespace 里；这种做法不仅与现实中的物理网络部署相比非常不合理，而且还会导致规则下发多份导致额外系统开销，尤其是当一个租户多个虚拟路由器下的网段地址有 Overlapping 时，在现有版本下则无法实现对不同虚拟路由器下采用不同的安全策略控制，希望社区尽快有成熟的方案和实现，即实现针对路由器对象的防火墙，这样才能比较符合实际网络的需求。

### 5.4.3 LBaaS 和 VPNaas

LBaaS (Load Balancer as a Service) 和 VPNaas (VPN as a Service) 是 Neutron 除防火墙外提供的其他两个高级网络服务功能，这两种功能在现实物理网络中实际部署应用是必不可少的选择。

负载均衡的作用就是在已有的网络架构基础上，为了提高对网络数据请求服务的服务能力，及增强虚拟网络中服务器提供的的数据吞吐量和网络服务可用性的

技术方案，从实现的网络层次来说有四层负载均衡和七层负载均衡；四层负载均衡主要是针对 TCP 和 UDP 的负载均衡，七层负载均衡主要是针对应用的 URL 的负载均衡，四层负载均衡是七层负载均衡的基础。现在比较知名的负载均衡技术方案或设备有 Lvs、Haproxy、Nginx 以及硬件的 F5。从性能上来说 F5 最优，是 F5 Networks 公司研发的在服务器负载均衡、链路负载均衡、多站点负载均衡等领域的硬件产品，虽然 F5 出名的是负载均衡方面但是也提供很多其他方面的网络功能，比如应用交付网络（Application Delivery Networking, ADN）、本地流量管理、灾难备份、广域网传输优化、SSL VPN、不同 ISP 互访互通、远程安全接入/访问、文件存储虚拟化、多链路接入、远程安全访问等，这种方案的缺点就是投资成本高。Lvs 性能次之，且和 F5 一样只有四层负载均衡的功能，属于 Linux 的开源项目。Nginx 和 Haproxy 都支持 L4-L7 的负载均衡但性能不如前两种方案，其中 Nginx 仅能支持 HTTP、FTTPS 和 Email 协议适用范围小，Haproxy 可以支持虚拟主机及保持 Session 等功能，另外 HAProxy 负载均衡策略非常多，包括简单轮询的 RoundRobin、基于权重的 static-rr、基于连接数的 leastconn 等 8 种。负载均衡技术在 OpenStack 的 Neutron 里截至 I 版的实现暂时只有 Haproxy 这个 Plugin 的支持。但是在从 H 版到 I 版，甚至在 J 版中，结合 L3 Agent 的网络节点 HA，LBaaS Agent 的高可用也是一项复杂的技术点，还包括需要针对 LB 服务的总限速实现、应对 LB 高可用时脑裂的机制、多种业务同时部署相同 VM 里的 LB 机制等统筹方案来实施部署。

VPN 技术在实际物理换联网中有着非常广泛的用途，是一种重要的数据安全通信方式，提供了用户验证、数据加密和密钥管理等功能，比如公司的子公司互访互访、个人办公的接入和数据中心的数据中心的数据同步等。能通过 Neutron

中提供的 Vpnaas 应用场景也非常与之类似。现在 VPN 的技术方案主要有点对点的 PPTP (Point to Point Tunneling Protocol, 点对点隧道协议)、IPSEC 隧道、L2TP 隧道协议 (Layer 2 Tunneling Protocol, 第二层隧道协议)、MPLS/VPLS 的 VPN 等。现在 Neutron 中的 I 版支持基于 IPsec 方式的开源项目 Openswan (<https://www.openswan.org/>) 作为 Plugin 和 CiscoCsrIPsec 设备作为 Plugin 的两种方式, 还有基于 OpenVPN 开源项目的 SSL VPN 也已经逐步得到支持和完善。VPNaas 的建立属于底层通信, 在 Horizon 页面的配置开启 LBaaS 和 VPNaas 功能, 还需要对 Neutron 底层的配置文件做适当的修改, 并安装合适的 Service Plugin 后才能被使用。

#### 5.4.4 监控安全和数据中心互联

安全从互联网诞生起就是无时无刻不在讨论的课题, 由于网络最初架构没有考虑安全的因素, 随着网络非法入侵、病毒木马盛行、DoS 攻击等网络事件时有发生, 会给提供信息服务的互联网公司以及公司的客户带来很大的威胁甚至导致经济利益上的损失。在现实的 IDC 数据中心里或大型企事业单位或公司的内网与外网互联的出口处, 都会有防止 DDoS 攻击的流量清洗系统, 也会部署大量的防火墙规则来阻止非法入侵和木马病毒的渗透。现在网络虚拟化技术实际部署越来越多, 但是网络虚拟化的监控和安全技术发展相对缓慢, 针对网络虚拟化产品容易部署和功能强大的安全产品市场需求非常大, Neutron 也提出了 IDS-aas 的概念, 安全厂商绿盟等也关注网络发展趋势及网络虚拟化, 并结合 SDN/NFV 技术提出了虚拟化网络或云计算网络安全的产品和方案。但是在这种云计算的网络中需要考虑这么一个场景, 因为大多数的云计算厂商不一定有自己的数据中心, 所以在云计算公司采用租用 IDC 机房服务器的方式来部署其云平台向租户提供云计算服务, 那么这个时候 IDC 机房会有其部署的流量清洗系统和防火墙功能, 但是这些功能对云计算公司的云平台来说不一定是恰好满足需求的, 有可能云计算

平台需要放行的流量被 IDC 的防火墙给拦截了，而攻击云平台的流量又没有被 IDC 机房的流量清洗系统 IDS 识别出来，而且 IDC 的防 DoS 攻击或防火墙功能基本不会给云计算厂商配置的权限，所以如何能提供两者在适当权限下的安全联动与结合机制的方案或产品就成了一个难题。另外，网络的监控不仅仅需要监控虚拟机或虚拟交换机，也需要监控物理服务器和物理交换机的运行参数（网络流量大小、磁盘 I/O、CPU 使用率、存储容量使用率、资源使用统计分析）等，才能全面地反映整个 OpenStack 系统和用户业务的运行情况，以及时发现报警并尽快排除故障。

数据中心为了异地灾备或 CDN 的需求要求不同地区甚至不同大洲的数据中心进行互联互通，OpenStack 部署的时候考虑到这点也需要此类技术，来保证 OpenStack 的 VM 上所提供的服务有高性能和高可靠性。所以 Neutron 从网络角度提出了一个解决该问题的 IDC-aas 的概念。

#### 5.4.5 Neutron 中的 QoS 功能

在 Openstack 中提及 QoS 的概念，大多数时候想到的都是限速功能，因为这个功能需求比较强烈，然而需要明确的是限速功能仅是 QoS 的一个功能点，还有很多诸如入口队列映射、流量监管、流量整形和出口队列调度等内容。目前 Neutron 中已经慢慢开始支持 QoS 的功能（<https://wiki.OpenStack.org/wiki/Neutron/Metering/Bandwidth>），但仅限于 QoS meter 等基本的概念，主要是旨在 router 和每条 traffic 的级别上通过 IPtables 来标记以协助完成限速的功能，另外在 Cisco 和 NVP 的插件上支持了部分 QoS 功能，但是还没有普及到所有的 Plugin 上；希望社区能尽快发布成熟通用的 QoS 方案。而在 Neutron 实际部署中，对于公网带宽、端口带宽、广播域中的风暴抑制、报文转发延迟保证甚至基于业

务流的队列调度等都是有一定急迫需求的，包括 VXLAN 的 BUM 问题。公网带宽可以通过虚拟化技术中的端口带宽限制或开发 Neutron 中的相应功能来提供，但是通过 VM 虚拟化技术的端口限速来限制带宽的使用做法太过彻底，在限速南北流量的同时将 VM 之间的东西流量也进行了限速，从而影响了某些东西方向业务流量的正常传输。QoS 这点是 Neutron 中需要加强的地方，虽然所需 QoS 功能在传统物理网络中大都比较基础且有成熟技术方案，但是在云计算网络里，这些功能却是比较费周折才能实现。如何保证 Neutron 根据业务提供适合的 QoS 功能是将来 OpenStack 产品化的一个重点，这个点的需求和 SDN 有非常好的切合之处。可以结合 OpenDayLight 等 SDN 控制器，通过 SDN 架构对 Neutron 的底层网络流量进行控制，来实现流量均衡和流级别的限速。

#### 5.4.6 Neutron 部署运维

Neutron 作为 Openstack 的一个组件，Openstack 的部署方式直接决定了 Neutron 功能配置复杂度。因为网络功能配置特别复杂，首先私有云需要考虑项目实施中客户所需求的功能点和各种资源的需求量，而公有云则需要考虑现有环境下对客户的业务是否均可正常运行；其次在任何种类的云项目实施部署时，需要提前对服务器和网络设备的整体拓扑做合理的规划，这样才能做好系统方面的降低成本和便于运营；最后还要考虑整个系统后续在故障维护和扩展节点方面的便利性，达成对用户承诺的稳定性和可用性指标，不会因为系统故障或系统升级而给租户带来业务数据的丢失或业务经济上的损失。

现在能够比较成熟的部署工具还不是太多，比较常用的有 Puppet、Devstack、Chef、Packstack、Crowbar 等，大都需要结合服务器的 PXE 功能；packstack 这类工具大多底层也采用的是 Puppet，Chef 工具使用的是 Ruby 语言。但是 packstack

等仅仅是一个搭建实验性环境的部署工具，对于生产环境中的运营部署灵活性和界面化配置便捷性还欠缺；现在大规模可用的有 Foreman 和 Mirantis 公司推出了 Fuel 等部署工具，其中 Fuel 提供了 GUI 图形化界面和 CLI 两种配置方式，并基于心跳机制提供了一些功能，比如管理节点和网络节点的 HA 部署，网络节点默认是主备的 HA。但是对于 Neutron 的某些特性比如 LBaaS 和网络节点 HA 等方面，还需要加强支持。

总的来说，Neutron 的功能越来越丰富，一个原因是因为现实中网络的技术和功能就非常的复杂；云计算提供了在租户网络 VPN（Virtual Private Network）的基础上提供 VPC（Virtual Private Cloud）的功能，很多物理网络的技术就有需求被移植到虚拟网络中。希望将来 Neutron 能在保证性能和稳定性的前提下提供更多新的特性。

---

## 5.5 VXLAN 隔离环境通信实例详解

采用 VXLAN 技术在云计算网络进行隔离的方式，是现在交换芯片厂商、网络设备商和云计算厂商都比较看重和正深入研究的一种大二层 Overlay 技术。这里针对两个节点的设备搭建了使用 VXLAN 隔离的 OpenStack 的 Neutron 网络环境，并对 Neutron 底层网络的流量，通过在各个通信流程关键点处抓包来分析的方式，阐述底层信息交互的过程。本节先通过一个 VXLAN 报文分析实例来说明 VXLAN 报文的具体格式，然后对 VXLAN 环境下的两台 VM 之间通信报文封装和解封装流程进行分析说明，最后对实验环境下的各个关键点来抓包，具体说明一下 VM 在 Neutron 的 VXLAN 隔离环境下报文转发的详细处理过程。

## 5.5.1 VXLAN 报文解析

掌握 VXLAN 封装和解封装较好的学习方式是抓取通信流程的几个报文来解析，这样能有助于加深对 VXLAN 协议的理解。笔者在实验室机房里用 Neutron 的 VXLAN 隔离搭建了 OpenStack 平台，并在上面新建了一个租户和该租户下分布在两台物理机上的两个 VM，然后对这两个 VM 之间的 Ping 通信主要报文用 tcpdump 命令进行了抓取和分析，包括 ARP 的请求和应答报文、ICMP 的请求和应答四种报文。实验环境是在 Fixed IP 地址为 10.10.10.2 的 VM 里与 Fixed IP 地址为 10.10.10.3 的 VM 执行 Ping 命令，然后在计算节点的内网对应的网卡抓取报文，具体报文格式解析如下：

1) 在计算节点内网端口，用 tcpdump 命令抓取的相应从 10.10.10.2 里 Ping 地址 10.10.10.3 的 VM 的十六进制报文数据为：

```
0x0000: 000c 29d2 873d 000c 29e0 0994 0800 4500
0x0010: 004e 97bb 4000 4011 598d c0a8 6402 c0a8
0x0020: 6403 8b14 12b5 003a 0000 0800 0000 0000
0x0030: 0200 ffff ffff ffff fa16 3e90 752f 0806
0x0040: 0001 0800 0604 0001 fa16 3e90 752f 0a0a
0x0050: 0a02 0000 0000 0000 0a0a 0a03
```

解析如下：

```
0x0000: 000c 29d2 873d //外层以太网 DMAC
          000c 29e0 0994 //外层以太网 SMAC, 此处 VTEP 通信采用多个单播
          0800 //外层 Ethertype
          4500 //外层 IP 的版本及头部长度、TOS
0x0010: 004e //外层 IP 报文总长度
          97bb 4000 //外层 IP 报文标识、标志、片偏移
          4011 598d //外层 IP 报文 TTL、协议号和头部校验
          c0a8 6402 //外层 IP 报文 SIP
          c0a8
0x0020: 6403 //外层 IP 报文 DIP
          8b14 12b5 //UDP 报文的源 PORT 和目的 PORT
          003a 0000 //UDP 长度、无检验和
          0800 0000 //VXLAN 32bits 保留位
```

```

0000
0x0030: 0200 //VXLAN 的 VNI 及 8bits 保留位
        ffff ffff ffff //内层 ARP 请求的 DMAC
        fa16 3e90 752f //内层 ARP 请求的 SMAC
        0806 //内层 ARP 的 Ethertype
0x0040: 0001 0800 //内层 ARP 的硬件类型、协议类型
        0604 //内层 ARP 的硬件长度、协议长度
        0001 //内层 ARP 的操作码为 ARP 请求
        fa16 3e90 752f //内层 ARP 报文的 Sender MAC
        0a0a
0x0050: 0a02 //内层 ARP 报文的 Sender IP
        0000 0000 0000 //内层 ARP 报文的 Target MAC
        0a0a 0a03 //内层 ARP 报文的 Target IP

```

## 2) 从 10.10.10.3 的 VM 回复的 ARP 应答十六进制报文数据为:

```

0x0000: 000c 29e0 0994 000c 29d2 873d 0800 4500
0x0010: 004e af76 4000 4011 41d2 c0a8 6403 c0a8
0x0020: 6402 c90b 12b5 003a 0000 0800 0000 0000
0x0030: 0200 fa16 3e90 752f fa16 3ead b70d 0806
0x0040: 0001 0800 0604 0002 fa16 3ead b70d 0a0a
0x0050: 0a03 fa16 3e90 752f 0a0a 0a02

```

### 解析如下:

```

0x0000: 000c 29e0 0994 //外层以太网 DMAC
        000c 29d2 873d //外层以太网 SMAC
        0800 //外层 Ethertype
        4500 //外层 IP 的版本及头部长度、TOS
0x0010: 004e //外层 IP 报文总长度
        af76 4000 //外层 IP 报文标识、标志、片偏移
        4011 41d2 //外层 IP 报文 TTL、协议号和头部校验
        c0a8 6403 //外层 IP 报文 SIP
        c0a8
0x0020: 6402 //外层 IP 报文 DIP
        c90b 12b5 //UDP 报文的源 PORT 和目的 PORT
        003a 0000 //UDP 长度、无检验和
        0800 0000 //VXLAN 32bits 保留位
        0000
0x0030: 0200 //VXLAN 的 VNI 及 8bits 保留位
        fa16 3e90 752f //内层 ARP 应答的 DMAC
        fa16 3ead b70d //内层 ARP 应答的 SMAC

```

```

0806 //内层 ARP 的 Ethertype
0x0040: 0001 0800 //内层 ARP 的硬件类型、协议类型
0604 //内层 ARP 的硬件长度、协议长度
0002 //内层 ARP 的操作码为 ARP 应答
fa16 3ead b70d //内层 ARP 报文的 Sender MAC
0a0a
0x0050: 0a03 //内层 ARP 报文的 Sender IP
fa16 3e90 752f //内层 ARP 报文的 Target MAC
0a0a 0a02 //内层 ARP 报文的 Target IP

```

### 3) 从 10.10.10.2 的 VM 到 10.10.10.3 的 ICMP 请求十六进制报文数据为:

```

0x0000: 000c 29d2 873d 000c 29e0 0994 0800 4500
0x0010: 0086 97bc 4000 4011 5954 c0a8 6402 c0a8
0x0020: 6403 adb1 12b5 0072 0000 0800 0000 0000
0x0030: 0200 fa16 3ead b70d fa16 3e90 752f 0800
0x0040: 4500 0054 0000 4000 4001 1291 0a0a 0a02
0x0050: 0a0a 0a03 0800 3d13 0e02 0000 aeb9 fe30
0x0060: 0000 0000 0000 0000 0000 0000 0000 0000
0x0070: 0000 0000 0000 0000 0000 0000 0000 0000
0x0080: 0000 0000 0000 0000 0000 0000 0000 0000
0x0090: 0000 0000

```

#### 解析如下:

```

0x0000: 000c 29d2 873d //外层以太网 DMAC
000c 29e0 0994 //外层以太网 SMAC
0800 //外层 Ethertype
4500 //外层 IP 的版本及头部长度、TOS
0x0010: 0086 //外层 IP 报文总长度
97bc 4000 //外层 IP 报文标识、标志、片偏移
4011 5954 //外层 IP 报文 TTL、协议号和头部校验
c0a8 6402 //外层 IP 报文 SIP
c0a8
0x0020: 6403 //外层 IP 报文 DIP
adb1 12b5 //UDP 报文的源 PORT 和目的 PORT
0072 0000 //UDP 长度、无检验和
0800 0000 //VXLAN 32bits 保留位
0000
0x0030: 0200 //VXLAN 的 VNI 及 8bits 保留位
fa16 3ead b70d //内层 IP 报文的 DMAC
fa16 3e90 752f //内层 IP 报文的 SMAC
0800 //内层 IP 报文的 Ethertype

```

```

0x0040: 4500 //内层 IP 的版本及头部长度、TOS
         0054 //内层 IP 报文总长度
0000 4000 //内层 IP 报文标识、标志、片偏移
         4001 1291 //内层 IP 报文 TTL、协议号和头部校验
         0a0a 0a02 //内层 IP 报文 SIP
0x0050: 0a0a 0a03 //内层 IP 报文 DIP
         0800 //ICMP 的类型为回显请求、代码
         3d13 //ICMP 的检验和
         0e02 0000 //ICMP 的标识符和序号
         aeb9 fe30
0x0060: 0000 0000 0000 0000 0000 0000 0000 0000
0x0070: 0000 0000 0000 0000 0000 0000 0000 0000
0x0080: 0000 0000 0000 0000 0000 0000 0000 0000
0x0090: 0000 0000 //ICMP 请求的选项数据

```

#### 4) 从 10.10.10.3 的 VM 到 10.10.10.2 的 ICMP 应答十六进制报文数据为:

```

0x0000: 000c 29e0 0994 000c 29d2 873d 0800 4500
0x0010: 0086 af77 4000 4011 4199 c0a8 6403 c0a8
0x0020: 6402 bd00 12b5 0072 0000 0800 0000 0000
0x0030: 0200 fa16 3e90 752f fa16 3ead b70d 0800
0x0040: 4500 0054 e756 0000 4001 6b3a 0a0a 0a03
0x0050: 0a0a 0a02 0000 4513 0e02 0000 aeb9 fe30
0x0060: 0000 0000 0000 0000 0000 0000 0000 0000
0x0070: 0000 0000 0000 0000 0000 0000 0000 0000
0x0080: 0000 0000 0000 0000 0000 0000 0000 0000
0x0090: 0000 0000

```

#### 解析如下:

```

0x0000: 000c 29e0 0994 //外层以太网 DMAC
         000c 29d2 873d //外层以太网 SMAC
         0800 //外层 Ethertype
         4500 //外层 IP 的版本及头部长度、TOS
0x0010: 0086 //外层 IP 报文总长度
         af77 4000 //外层 IP 报文标识、标志、片偏移
         4011 4199 //外层 IP 报文 TTL、协议号和头部校验
         c0a8 6403 //外层 IP 报文 SIP
         c0a8
0x0020: 6402 //外层 IP 报文 DIP
         bd00 12b5 //UDP 报文的源 PORT 和目的 PORT
         0072 0000 //UDP 长度、无检验和
         0800 0000 //VXLAN 32bits 保留位
         0000

```

```

0x0030:  0200 //VXLAN 的 VNI 及 8bits 保留位
         fa16 3e90 752f //内层 IP 报文的 DMAC
         fa16 3ead b70d //内层 IP 报文的 SMAC
         0800 //内层 IP 报文的 Ethertype
0x0040:  4500 //内层 IP 的版本及头部长度、TOS
         0054 //内层 IP 报文总长度
         e756 0000 //内层 IP 报文标识、标志、片偏移
         4001 6b3a //内层 IP 报文 TTL、协议号和头部校验
         0a0a 0a03 //内层 IP 报文 SIP
0x0050:  0a0a 0a02 //内层 IP 报文 DIP
         0000 //ICMP 的类型为回显应答、代码
         4513 //ICMP 的检验和
         0e02 0000 //ICMP 的标识符和序号
         aeb9 fe30
0x0060:  0000 0000 0000 0000 0000 0000 0000 0000
0x0070:  0000 0000 0000 0000 0000 0000 0000 0000
0x0080:  0000 0000 0000 0000 0000 0000 0000 0000
0x0090:  0000 0000 //ICMP 请求的选项数据

```

从以上 Neutron 环境下 Openvswitch 的 VXLAN 四种数据格式的解析可以看出，Open vSwitch 对 VXLAN 环境下的通信报文格式头封装严格遵循了 VXLAN 标准的封装要求。

## 5.5.2 VXLAN 通信流程

如图 5-9 的环境下，VXLAN 在底层的隔离技术对服务器里面的 VM 来说是透明的，所以 VM 发送和接收的都是普通的以太网报文；对于同租户下位于 compute node1 上的两个 VM 而言，两者的通信也无须 VXLAN 的封装和解封装，而是直接的局域网以太网二层转发的方式，但对于位于两台物理服务器上的 VM1 和 VM3 来讲，两者的通信则分别需要两次封装和两次解封装，也就是只有 VM 的报文要离开物理服务器时或从物理服务器外界接收报文时才会涉及 VXLAN 的封装和解封装。下面针对 VM1 和 VM3 之间的 Ping 通信流程来解释下 VXLAN 的工作原理。

在如图 5-9 的实验环境按照图 5-10 的虚拟网络拓扑建立好虚拟机后，VM 的

分布对应图 5-9 所示的拓扑。那么当 VM1 里对 VM3 的 IP 地址 10.0.1.7 执行 Ping 命令时，流程如下：

(1) VM1 通过数据报文的的目的 IP 地址和自己网卡的 IP 地址及掩码可以明确这是同一个隔离域的 VM 的 IP 地址，所以直接通过二层的 MAC 地址转发即可到达目的主机，但是不知道该 IP 地址对应的 MAC 地址，会首先发送一个 ARP 请求报文来查询 IP 地址 10.0.1.7 的主机即 VM3 的 MAC 地址，报文的 SMAC 是 VM1 的网卡 MAC，DMAC 是广播地址全 0xff 的 MAC 地址；

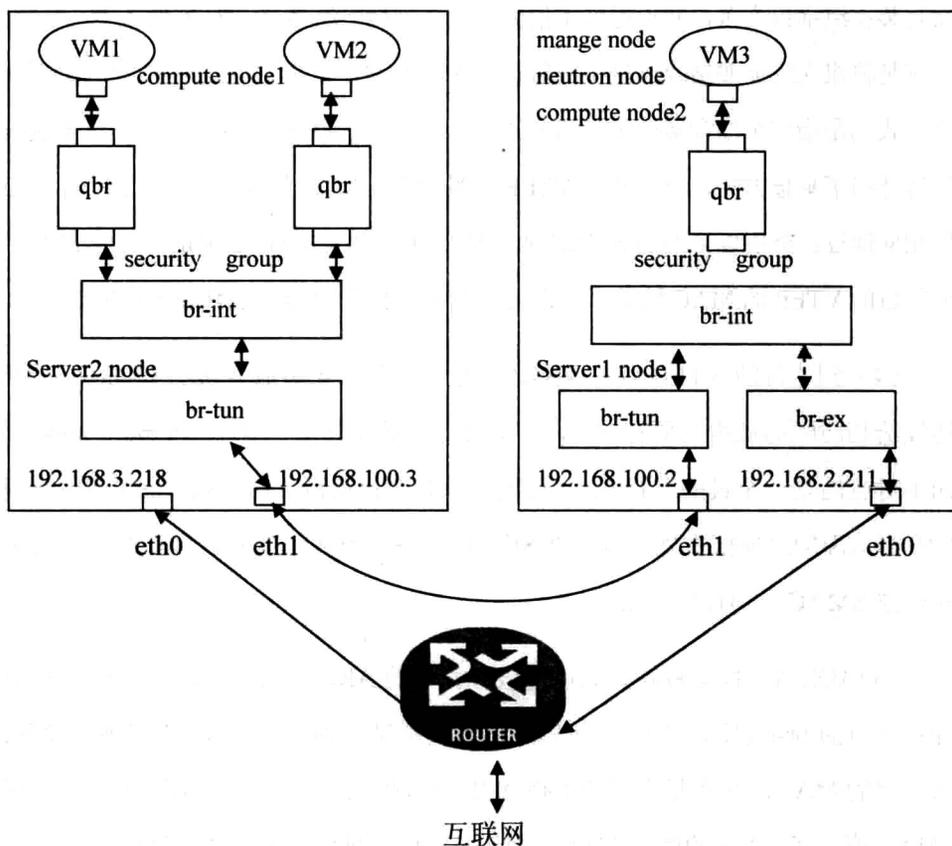


图 5-9 VXLAN 环境物理服务器拓扑示意图

(2) 该 ARP 请求报文在 br-int 会被广播到 VM2 上, 但是 VM2 不会做出 ARP 应答, 因为 ARP 请求报文不是对 VM2 上某个网卡 IP 地址的请求;

(3) ARP 请求的广播报文里会有一份报文被送到了 compute node1 的 br-tun 上; 在 br-tun 上有一个 VXLAN 类型的成员端口, 该成员端口报文指定了 VTEP 的 VXLAN 报文封装外层 SIP 的地址, 然后将 VM1 发送来到 ARP 报文封装到 VXLAN 格式中; VXLAN 封装时会往其他的 VTEP 上都发一份。针对非单播报文可以有三种方式与其他 VTEP 通信, 组播、单播和混合方式, 组播就是将非单播报文封装在组播报文里, 然后通过组播协议发送到相应组播组里的 VTEP 上, 单播方式则是将报文给需要接收该报文的每一个 VTEP 采用单播的方式发送一份相应的混合方式则是指将组播和多个单播的方式结合起来。在 OpenStack 的 Neutron 安装时, 笔者选择了单播方式, 所以此处 VTEP 的每份报文的外层 DIP 地址则是目的 VTEP 的相应地址; 外层以太网的 SMAC 地址是 VTEP 的网卡 MAC 地址, 目的 MAC 地址用目的 VTEP 的 MAC 地址; 该报文被当成普通的 IP 报文被转发到目的地;

(4) 到达目的 VTEP 后对 VXLAN 报文进行解封装成普通的 IP 报文, 然后转发给相应隔离域内的所有 VM, 在如图 5-9 的拓扑里, 就是 compute node2 的 eth1 对应的是一个目的 VTEP; 并且该 VTEP 会记录报文内层以太网的 SMAC 和 VNI 与该 VXLAN 报文外层以太网 SIP 对应关系, 当然还需要学习外层以太网 SIP 和外层 SMAC 的 ARP 关系;

(5) VXLAN 报文被解封装后变成了普通的 ARP 请求报文, 然后被转发到该计算节点的 br-int 上, 并在相应 VNI 区域内继续广播, 只有 VM3 发现有设备请求自己的 MAC, 就回复相应的单播 ARP 应答报文, 此时封装 ARP 应答报文时, SMAC 变成了 VM3 的网卡 MAC, 目的 MAC 地址则是变成了 VM1 的网卡地址 MAC, 并将 VM3 的 MAC 和 IP 对应关系封装在内;

(6) ARP 封装好后转到 br-int, 并根据单播转发查询将该报文转到该 VM 所在服务器的 VTEP 上进行 VXLAN 封装, 封装格式是外层 SIP 是 compute node2 上 VTEP 的 IP 地址, SMAC 是 VTEP 所对应网卡的 MAC 地址, 外层目的 IP 是根据步骤 (4) 中学习的 MAC+VNI 与其所在 VTEP 端 IP 地址的对应记录来获取的, 而外层 DMAC 则是根据 IP 地址转发路由相应的 ARP 关系来进行封装; 然后将报文转发到目的 VTEP;

(7) ARP 应答的 VXLAN 报文达到 compute node1 后进行解封装, 同样, 该节点的 VTEP 需要学习 VM3 的 MAC+VNI 与 compute node2 上 VTEP 的 IP 地址的对应关系, 以便于后续数据发送时查询使用; 然后 ARP 应答报文被转发到相应的 VNI 隔离域内;

(8) 在对应的隔离域内, 根据报文的 DMAC 将 ARP 应答报单播转发给 VM1, 然后 VM1 就获取了 VM3 的 IP 和 MAC 的 ARP 对应关系, 开始发送数据;

(9) 然后 VM1 发出 Ping 的 ICMP 请求报文, SMAC 和 SIP 分别是 VM1 的网卡 MAC 地址和 IP 地址, DMAC 和 DIP 是 VM3 上网卡的 MAC 地址和 IP 地址;

(10) ICMP 请求报文根据 DMAC 经过 br-int 到 br-tun 后, 在 br-tun 被封装成 VXLAN 报文, 外层 SMAC 和 SIP 是本地 VTEP 的网卡对应的 MAC 和 IP 地址, DIP 是根据步骤 7) 中记录的 VM3 的 MAC+VNI 查找到的对端 VTEP 的 IP 地址, DMAC 是从 Linux 的 TCP/IP 协议栈里 ARP 的对应关系中查到的该对端 VTEP 的 IP 地址的相应 MAC 地址; 然后从 compute node1 单播发送到对端 compute node2 的 VTEP 处;

(11) ICMP 请求 VXLAN 封装后的报文到达 compute node2 的 VTEP 后被 VXLAN 解封装, 不再需要步骤 4) 中报文 SMAC+VNI 与外层 SIP 的对应关系, 因为已经存在了该记录, 只需将 VXLAN 解封装后的报文转发到相应 VNI 的隔离域即可;

(12) 在相应隔离域内, ICMP 请求报文通过 DMAC 将报文转发给 VM3, 并且 VM3 给出 ICMP 应答报文到 compute node2 的 br-int 处后再被转发到 br-tun 上;

(13) 在 compute node2 里 VTEP 上按照步骤 (10) 的方式进行 VXLAN 的封装, 并转发到对端的 VTEP 的处;

(14) 在 compute node1 的 VTEP 上对 ICMP 应答的 VXLAN 报文进行解封装, 然后转到相应的隔离域, 类似于步骤 11), 该处也无须重新生成 MAC+VNI 与 VTEP 的 IP 地址对应关系;

(15) 然后被解封装的 ICMP 应答报文在相应的隔离域内通过 DMAC 被转发到 VM1 处, VM1 就得到了 ICMP 请求的应答报文, 表示通信链路畅通; 至此, 也完成了 VM1 对 VM3 的 IP 地址执行 Ping 命令的整个流程。VXLAN 隔离实施环境网络拓扑图如图 5-10 所示。

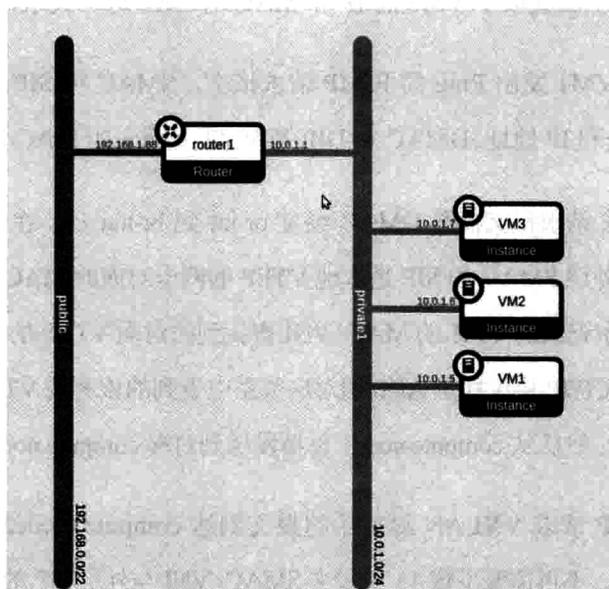


图 5-10 VXLAN 隔离实验环境网络拓扑图

(16) 对于 VM 访问外网的情况，因为目的 IP 是远端路由，所以不再需要 ARP 的学习过程，而是直接被转发到 br-tun 上在 VTEP 处进行 VXLAN 的封装，再被转发到网络节点服务器对 VXLAN 报文解封装；然后报文转发出去的 SNAT 过程和 VLAN 隔离的环境下的流程就没有差别了；

(17) 外网访问 VM 的流量，先在网络节点进行 DNAT 转换，然后在网络节点上 br-tun 的 VTEP 处进行 VXLAN 的封装，然后被转到相应 VM 所在物理服务器上，在相应计算节点的 VTEP 处进行 VXLAN 解封装，并最终通过二层转发到达目的 VM 完成通信流程。

笔者对 VXLAN 环境下的几种通信情况通过 Linux 的 tcpdump 命令进行了详细抓包，具体可以请参见附录 D。

---

## 5.6 Neutron 网络高级话题讨论

Neutron 的使用和部署已经有很多的案例，比如 Unitedstack 的公有云和托管云的底层网络 Core Plugin，使用的就是 Neutron 中 Open vSwitch 的方案，但是在云计算中 Neutron 网络的研究和部署中确实都遇到了很多问题，确实还有很多云计算公司倾向于采用简单稳定的 Nova-network 技术，毕竟对于生产环境的产品，稳定性压倒一切；即使功能再丰富，系统不定期的故障导致客户数据丢失或业务的间歇性服务故障的现象，必然结果是没有客户再信任该云计算方案或平台的提供商；所以就像网络历史中任何新技术的出现一样，必然会有一个产生、发展和成熟的时间历程。Neutron 现在经历了若干个版本的迭代，已经修复了大量 BUG，并且提供的网络功能在大量测试后不断完善，随着云计算厂商研发人员深入的研究和改进，及使用 Neutron 相关的云计算平台的不断增多，会有越来越多的部署

案例使用 Neutron 技术，并且在 J 版中 Nova-network 已经被标注。在 OpenStack 的技术交流 qq 群里（群号：347195472）也有非常多新入门或不熟悉网络的初学者咨询各种网络问题。本节对常见的问题及通用的解决方法进行下汇总。

### 5.6.1 常见 Neutron 网络问题

搭建完 OpenStack 的实验环境后，需要对网络进行做些修改配置才能使得 OpenStack 的虚拟网络为 VM 提供正常服务。由于 OpenStack 的安装者或网络配置者不熟悉网络原理、其他原因操作不当或配置不完整等原因导致的网络问题时时有发生。这里对常见的 Ping 不通外网 Ping 不同 VM，VM 无法获取 DHCP 地址等常见问题进行汇总，剖析导致问题发生或出现异常的可能原因，为解决这些问题提供思路 and 定位问题的手段。下面讨论的情形大部分是针对用 Open vSwitch 做 Plugin 且使用 VLAN 隔离环境下 VM 采取 DHCP 的方式获取 IP 地址的场景。

通常情况下 VM 启动时，会和相应网段的 DHCP 功能的进程 Dnsmasq 进行交互，基本就是 DHCP 的交互流程，并获取 DNS 的配置信息。当 OpenStack 安装完毕建立 VM 后，如果 VM 能够启动但是无法获取 IP 地址时，即 VM 在内部用 Linux 系统的 ifconfig 命令或 Windows 系统用 IPconfig 等查看网卡信息，可以看到相应的网卡下没有 IP 地址或没有有正确的 IP 地址，这种情况的发生必然是因为 DHCP 的交互过程不完整导致的，具体问题原因定位可以分这么几步来进行：

(1) 首先需要到网络节点看下相应网段的 Dnsmasq 进程是否启动正常，如果没有启动需要分析下没有自动启动的原因，排除错误让 Dnsmasq 能正常启动后再尝试 VM 获取 DHCP 的流程，以保证 Dnsmasq 可以正常为该网段的 VM 提供 DHCP 服务；

(2) 如果 Dnsmasq 的进程正常，VM 仍然还是无法通过 DHCP 的流程获取

IP 地址，则需要根据 VM 的启动日志判断分析 DHCP 的交互过程是卡在了哪一步，大部分都是 Discover 这个报文发出后没有得到 Offer 报文，可以初步判断是因为底层网络通路配置不正确导致的；底层网络不通的原因有很多，可能是 VM 所在物理服务器的网卡网线松动（用 `ethtool eth0` 格式的命令查看网卡是否 active 即可）；如果是在采用 VLAN 技术的隔离环境里，更重要的原因有可能是因为 VLAN 在交换机上配置的范围和 OpenStack 的不匹配，因为 OpenStack 安装时会指定物理交换机的 VLAN 范围，那么在 Open vSwitch 会配置 VLAN Translation 的功能，并且物理交换机上要配置添加 OpenStack 指定的 VLAN 范围内的所有 VLAN ID，连接服务器的交换机端口还要配置成 Trunk 模式。

(3) 如果 Dnsmasq 进程没有问题且 VLAN 物理交换机和 OpenStack 的虚拟网络配置都匹配，VM 仍然无法通过 DHCP 获取 IP 地址的话，这个时候基本上是交换机的 ACL 或者服务器的 IPtables 规则将 DHCP 的交互报文丢弃导致的，就需要根据 DHCP 交互报文的特征去查看交换机的 ACL 规则和 IPtables 规则，来确认是否有规则匹配到报文且把相应的报文丢弃了，采用 VXLAN 的方式隔离时就需要对 IPtables 做配置以放行 VXLAN 格式封装的报文（主要是匹配 VTEP 的地址和 UDP 的目的端口 4789）。通过以上三步，基本可以定位出 VM 无法通过 DHCP 获取 IP 地址的确切原因。

当 VM 建立后，后续的工作就是想让其能访问公共网络或 OpenStack 所指定的 public 网络，测试方式往往是 Ping `www.baidu.com` 或 Ping `114.114.114.114`，如果遇到 Ping 不通外网设备的情况就需要按照如下步骤对问题进行定位：

(1) 如果 Ping `114.114.114.114` 能够通信，仅是 Ping `www.baidu.com` 不通，基本可以断定是该网段的 DNS 或 VM 的 DNS 配置有问题，修改下相应的 DNS

配置信息即可；

(2) 如果 Ping 114.114.114.114 和 Ping www.baidu.com 都不通，那么应该是确切的 VM 无法访问外网，这时候先看 VM 是否有正确的 IP 地址，如果没有正确的 IP 地址可以参考前文所述的 VM 如何获取正确获取 IP 地址的步骤进行定位；

(3) 如果 VM 有正确的 IP 地址但仍然无法访问外网，先判断 VM 所连内网对应的虚拟路由器是否设置了 public 网为外网网关，并且 public 网是可以与互联网通信的；很多时候该步骤的原因是租户或管理员将外网直接在虚拟路由器中进行加入接口操作，而不是将其设置为默认网关；

(4) 如果 VM 能获取 IP 地址且整个链路都没有问题，那么可以通过 tracepath 或 traceroute 某个 public 网中某个设备的 IP 地址的方式来判断报文是那里丢弃的，然后确切定位到链路不通的原因；

(5) 链路不通原因基本是物理链路不通或者交换机的 ACL 规则及服务器的 IPtables 规则把报文丢弃了，只需要修改相应的规则或打通链路即可解决问题；

(6) 另外还有配置错误的原因，比如路由器添加外网时设置外网为默认网关而不是添加接口到外网，或者忘记在网络节点将 br-ex 里包含物理服务器的外网网卡作为其一个端口成员，这些都是可能导致 VM 不通外网的原因。

OpenStack 的组件 Neutron 在 OpenStack 虚拟网络中所起的作用和机架交换机中的主控卡作用类似，虽然有网络控制和转发的分离，但功能上仅有对网络的连通性进行配置和控制，并不能为应用程序提供对网络资源的使用和 API，所以 Neutron 不能归类为 SDN 的控制器，如果其他的云平台无论是开源或者闭源，仅仅是起到了 Neutron 类似的功能，同样不能归为 SDN 控制器，否则只能是云厂家的概念炒作。如果 Neutron 像与 SDN 结合使用，可以使用 ODL (OpenDaylight)

和 OpenContrail 等控制器配合来控制网络。OpenDaylight 有三个版本，Base Edition、Virtualization Edition 和 Service Provider Edition；只有 Virtualization Edition 支持 Neutron 的集成，Neutron 在 ODL 中的位置可以参考 ODL 官网提供的资料，OpenDaylight 的 Virtualization Edition 架构图如图 5-11 所示。

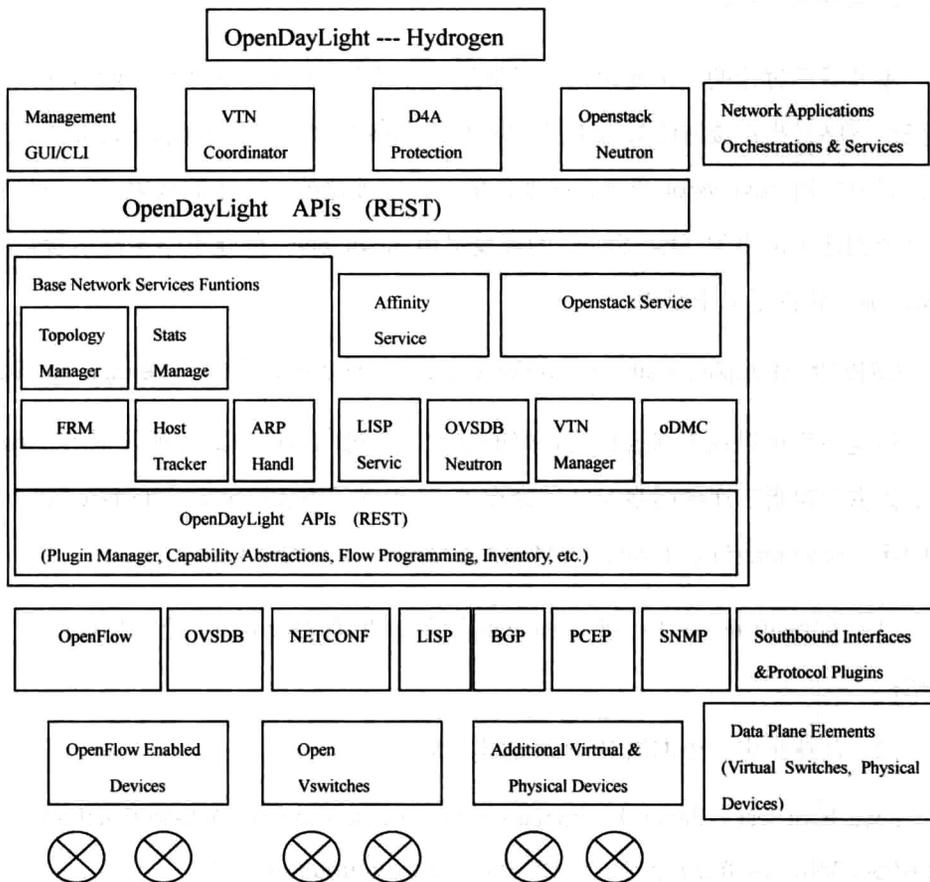


图 5-11 OpenDaylight 的 Virtualization Edition 架构图

从 OpenStack 的 Neutron 网络资料中可以看到，网络节点的 br-ex 和 br-int 之间是有一条由 VLAN 隔离比 VXLAN 多了 phy-br-ex 和 int-br-ex 组成的 veth 互联，

这个虚拟连线在真实部署的环境中可能是不存在的，原因可能是因为 Open vSwitch 之前版本需要后来 Open vSwitch 版本升级后不再需要就去掉了但是资料没有更新，当然通过这种方式实现 VM 内网和外网环境的互通是确实可以的，因为 Neutron 的底层网络是丰富多变的，为了实现连通的目的，通常可以有多种配置方式或实现方案。

本节最后讨论的一个问题是如何创建一个没有任何网卡的 VM？然后任意指定任何网络且指定该网段 IP 地址连接到相应网络？这个需求在 Openstack 的原生态代码中想用 nova boot 命令直接实现的话，需要修改一些限制代码，因为如果在已经创建了网络的 OpenStack 的环境里用 nova boot 创建不指定网络参数的 VM，那么将提示如下错误：

```
ERROR: Multiple possible networks found, use a Network ID to be more specific
```

但是我们可以在不修改任何代码的基础上通过间接的方式实现，因为 Nova 除了提供创建指定连接网络 VM 的命令外，还提供了添加(nova interface-attach)和删除(nova interface-detach) VM 端口的命令；具体命令流程如下：

(1) 用 neutron net-list 或 nova net-list 列出所有 Openstack 已经创建的租户网络；

(2) 任意指定一个网络 IP 地址创建 VM；

```
nova boot test --flavor 1 --image cirros --nic net-id=08f4e4ab-8a97-4e0f-8999-f7f0a950d36b, v4-fixed-ip=10.10.10.100 --security-groups default
```

(3) 用 nova interface-list test 查看新创建 VM 的端口 IP；

(4) 用 nova interface-detach server\_name port\_id 的格式删除新创建的 VM 的网卡；

(5) 然后可以用 `nova interface-attach` 可以连接到任何网络。

通过这种间接的方式，便可以轻易地实现创建 VM 不带任何网卡，且可以后续连接到任意该租户内的网络，并且连接时可以指定任意没有被使用的该网段的 IP 地址。

## 5.6.2 Neutron 网络性能

Neutron 支持了丰富的网络功能，这点在一定程度上导致了网络性能的降低，但是这个不是 Neutron 网络性能的决定因素；从 Neutron 的发展历史和现在存在的问题来看有如下几个性能问题。

(1) 公网网络瓶颈问题：因为 OpenStack 将节点分为网络节点、管理节点和计算节点，网络节点是所有 VM 与公共互联网通信的关键之处，如果网络节点的性能不够必然导致整个 OpenStack 集群中某些 VM 与外网通信的质量受到影响，尤其是单节点 L3 Agent 的公网带宽使得 Neutron 更是成为 OpenStack 产品化的最大障碍。现在解决这个问题的办法有网卡 Bonding, Multi-L3-Agent、分布式虚拟路由器 (Distributed Virtual Router, DVR) 和将网络节点设备替换为 ServerSwitch 等或者几种技术的结合等方案。在 J 版的 OpenStack 中已经支持了 DVR 这种 Multi-L3-Agent 技术，但这种技术还并不是十分的完善，具体有以下几个方面需要注意。

① 从网络安全上讲，这种方案与 Nova-network 一样存在计算节点的安全问题，这种方案需要将计算节点也暴露于公网之上，对于客户来说云计算的网络会有安全隐患，导致很多客户不接受这类方案。

② 从 DVR 解决带宽瓶颈的原理来讲本质也是 Multi-L3-Agent, 这种方式在 H 版和 I 版来讲也是已经实现了的，只是 DVR 方案采用的是对那些配置 Floating IP 的 VM 南北流量直接从计算节点出去，而没有配置浮动 IP 的南北流量仍然要从

网络节点出去；而且 DVR 方案只处理 VM 通过 Floating IP 网络流量的瓶颈问题，至于那些没有配置 Floating IP 的 VM 仍然需要通过网络节点的默认网关访问外网业务，而这部分网络节点的带宽瓶颈解决工作仍然需要。

③ 从 DVR 解决网络 HA 来讲，虽然 H 版和 I 版还不是很成熟，但是在此基础上进行一定的开发基本可以满足需求；DVR 的方案必定还有很多未知的缺陷或限制，也需要经过大量的实践测试和验证才能完善。

④ 另外 DVR 并不能完全解决 Neutron 的公网瓶颈问题；首先从原理上来说，DVR 和原来的 Nova-network 不同，DVR 是在某个计算节点的外网上，作为一个租户或几个租户的网络节点，该节点所对应的租户所有的 VM 的外网流量要经过该节点，并且存在该租户 VM 的所有节点都会通过大量静态 ARP 表来减少广播带来的网络性能损耗，而且当 VM 没有 Floating IP 时与外网的交互还需要通过网络节点才行；这样当有某个租户的外网带宽或者同一个网络节点几个租户的外网带宽之和超过物理服务器网卡带宽时依然存在网络公网瓶颈，这种场景下就无法保证完全外网带宽没有问题。

⑤ DVR 可能增加设备成本或导致运维复杂化；因为 DVR 解决带宽瓶颈不彻底引起的网络堵塞现象是无法提前预知的，一旦发生也无法进行配置修改，另外若是通过增加服务器外网带宽而扩展网卡还导致了服务器网卡硬件配置的不确定性，并且每个物理服务器都需要预留一定数额的外网带宽，随着计算节点的增多必然造成云计算厂商对于租用外网的大量浪费，且给运维团队的工作带来了诸多不便，而对于将网络节点和计算节点分开的 Multi-L3-Agent 方案就不存在这个问题，一方面可以对网络节点提前规划或者后续添加，也能对网络节点的设备进行定制，比如使用 ServerSwitch 等。

⑥ 最后，DVR 下 FWaaS、LBAas 和 VPNaas 都需要做些相应的修改才能继

续使用，即给 FWaaS、LBaaS 和 VPNaaS 的部署带来了不必要的麻烦。

所以对于社区支持将 DVR 代码合并近主流的抉择，笔者观点是不太支持的，毕竟网络性能的问题靠单纯的软件持续开发和优化始终有瓶颈，想实现性能方面的实质性发展，必须需要依赖网络设备商的方案和设备来解决这些问题；而 DVR 方案则是为了采用一种不彻底的解决方式来解决一个重大问题，则引入了数个令人懊恼的问题，使用这种方案的云计算厂家有点得不偿失。有些时候或许硬件的点滴改进或，可以抵过单纯软件方面的大量的优化，Neutron 的性能问题还需要通过适配 Neutron 组件的网络设备硬件来解决。

(2) 不同的网络隔离技术比如 VLAN、VXLAN、MPLS、NVGRE/Geneve 或其他方案因为报文封装格式和封装效率不同，为虚拟网络带来的性能损耗也不尽相同，并且也性能问题与具体软件的系统级实现有非常大的关系，所以需要根据测试结果进行尝试优化；在 Open vSwitch 的 Plugin 里网络性能的决定因素很大程度上取决于 Open vSwitch 的确切实现；所以对客户的实施方案需要在性能和支持租户数量上进行一个折中，不过 ML2 Plugin 的出现对于解决该问题会有很大的帮助，因为其允许多种网络类型的同时存在，这为以后从 VLAN 到 VXLAN 的扩展提供基础。

(3) 现在 Neutron 的网络底层经过各层虚拟化后，处理流程增加很多，导致了报文在服务器内部处理的复杂性，必然影响报文的转发效率和 CPU 的使用率，为了最大可能提高报文转发速度，可以和 LSO/LRO、Intel 的 DPDK 技术或 SR-IOV 技术结合，加快物理服务器和 VM 收发包的速率，从而极大的提高虚拟网络里报文的转发速率和降低延迟；但是 X86 CPU 的处理能力毕竟受系统实现的限制，比起硬件交换机的转发能力还相差很多。

(4) IPtables 性能也是众所周知的问题，当 IPtables 的规则比较多时，一方面报文进入后随着 IPtables 的一条条规则进行匹配，直到找到匹配项执行相应的动作，这个报文与 IPtables 的执行匹配的过程需要耗时，也会导致报文转发延迟加大，且延迟抖动变大；另一方面，当 IPtables 配置变化时尤其是配置大量规则时，需要对 IPtables 的规则进行先后排序的逻辑就会复杂，容易产生功能混乱，并且同一个 chain 里一组互不影响的规则排列先后顺序可能导致某条规则的报文转发性能的不同，从而影响性能不稳定。

(5) DHCP 的性能问题是指当一个网络的 Dnsmasq 进程工作时，如果同时有大量 VM 新建时或很多的 VM 同时 DHCP 租约到期时，会发送大量的 DHCP Discovery 报文或 DHCP Request 续租报文；如果 Dnsmasq 进程性能不足够强大，很可能导致对 DHCP Discovery 报文处理不过来，从而某些 VM 获取 IP 地址不正确或部分 VM 的 IP 地址续租不成功，这个影响也是非常大的；另外 DHCP 性能问题中还包括租约时间问题，如果租约周期太短会导致 VM 的续租频率很高，占用一定的 CPU 使用，而租约周期太长一旦 VM 或其网络发生异常，如果这种情况下的 VM 的 IP 仍然被占用的话，DHCP 进程会一直等到租约周期到达时才将该 IP 地址释放，这就可能会影响新建 VM 对该 IP 地址的租用，这个问题可以结合 SDN 的方案解决：①把 DHCP 服务部署到仅提供 DHCP 服务的物理服务器上，以此来提高 DHCP 服务的性能；②全局联动 VM 挂掉或销毁后，尽快通过控制器发送相关的指令将 IP 地址释放掉。结合 SDN 解决该方法，有的云计算公司已经开始着手采用这种方案进行研发。

(6) ARP 的表项在 Linux 系统是有确切 ARP 表项大小的限制，这个值可以配置但是有上限，而且在 ARP 表项非常多的时候，维护起来也是个极其复杂的工作；在 Neutron 的 J 版中 DVR 方案中需要配置大量的静态 ARP；Linux 的 ARP

表项容量问题在笔者原来做交换机研发时也曾遇到过，可以适当在 Linux 系统里放开限制并做好全局规划，暂不确定在 VEPA 方案中是否有厂家的设备可以实现部分租户流量到 TOR 的方案，这样做也可以增加些 ARP 表项。

### 5.6.3 Neutron 网络稳定性

Neutron 网络的稳定性主要是指 Neutron 的各种服务可扩展性和异常发生时能够及时正确处理保持服务正常的指标，包括 L3 Agent HA、单路由器的 HA、DHCP 的 HA 及管理节点 Neutron-service 的 HA，DNS 的持续服务等，以及各种应用场景数据流下无论用 TCP 还是 UDP 都存在网络稳定性方面的问题，这些问题对不同业务的特性表现都不太一样，对网络服务的要求也不太一样，而虚拟网络很多是用软件的系统实现，在各种服务流量下的虚拟网络稳定性非常依赖于系统实现的细节。比如在 I 版和 J 版的现有网络节点高可用方案里，在网络节点故障时，该节点上租户路由器下的 VM 提供的 TCP 服务都会失去连接。对于路由器的 HA 在 Openstack 的 Juno 版本中有 L3-ha 的新特性，是基于 VRRP 的 Keepalived 工具实现 Active/Backup 的 HA 模式，这种方案还会因为 VRRP 的实现导致每个租户只能最多有 256 个虚拟路由器的限制。在 J 版中，DVR 和 L3-ha 还无法同时支持。

系统的实现包括开发云平台的开发环境所基于的操作系统版本是否稳定、所使用 Neutron 的版本是否稳定以及 Neutron 所使用的网络底层的 Core Plugin 和 Service Plugin 是否稳定等因素；而这些系统或工具的稳定与否，很大程度上取决于对其各个版本的深入理解和大量测试得出的结论。所以 Openstack 就理解和掌握其各个组件的架构和元代代码而言，技术知识已经非常不简单，如果再加上有需求实现对计算、存储和网络等模块专业的研发和部署设计等要求，将 Openstack 产品化将是一场非常专业和浩大的复杂工程，没有大量的专业领域的人才和一段时间持续的积累，是不可能实现的。

稳定性方面也需要考虑云计算产品或功能升级的问题，因为 OpenStack 产品化是基于开源社区的代码来实现的，现在 OpenStack 每半年一个版本，每次都会有大量的 BUG 被修复，也会有大量的新功能被提交，甚至还有大量的原有方案或代码被废弃，如果不及时更新肯定很快被技术发展的浪潮所淘汰。Neutron 中的升级还包括底层 Core Plugin 和 Service Plugin 的版本升级或 BUG 修复升级；另外 Neutron 的升级中，从 Nova-network 到 Neutron、从 VLAN 到 VXLAN 或从 I 版本升级到 J 版，都有大量的工作需要做，社区也没有统一成熟的方案来支持。这些都将是 Neutron 研发者所面临的技术问题。

与此同时，就为基于 OpenStack 进行产品化工作的云计算公司带来一个需要思考的问题，就是对 OpenStack 产品化中所做的定制化工作，如何与后续升级的 Openstack 代码快速的合并和联调。因为产品化过程中有些代码是公司定制化开发的功能，排除某些功能无法上传到社区的原因，还有很多是社区不可能接收的代码，比如目的是替换 Horizon 组件而重新开发的管理端等，而这些都需要在 OpenStack 进行产品化工作中有很好的产品设计和架构规划。

#### 5.6.4 Neutron 在折翼

如今 OpenStack 已经在今年 4 月份发布了 I 版，J 版马上也要在今年 10 月份发布，所以 OpenStack 的版本更新导致的 Neutron 版本迭代非常的迅速；而且不仅仅 Neutron 安装的时候要选 Plugin（比如说 OpenVswitch），就连 FWaaS、LBaaS、VPNaaS 也需要选择不同的 device\_driver 作为其实现功能的底层 Plugin，比如 VPNaaS 的 device\_driver 不仅支持 Openswan 还有对一种思科设备驱动的支持，研发或实施部署时，就需要对这些技术方案提前规划好。从功能实现上讲，Neutron 组件只是一个框架，具体的功能都由各个组件的底层 Plugin 来完成，好比人的一

副骨架只能支撑起整个形态，具体的网络功能是需要各种各样作为“血肉”的 Plugin 来具体实现的。这种架构有其自身的优势，但是笔者个人认为更多是 Neutron 发展的阻碍和 OpenStack 作为产品定制时的陷阱。而且不仅仅对 Neutron 组件里，Nova 组件里也存在这种现象。

为什么会出现这个问题？根本原因是 OpenStack 社区想赢得最广泛商家的支持，对于 Neutron 来说，就是各种 Plugin 的形式，让各个厂家能将其自己的虚实网络设备与 Neutron 对接，这样在 OpenStack 商业化中获取一定的利益。或许有的 OpenStack 使用者会有这样的疑问：“无论选择何种 Plugin，其他的 Plugin 代码都不会被使用了，如何还会有影响”？这个问题的回答需要先看下 Neutron 现在的代码结构，之后这些读者就不再疑惑了；H 版、I 版和 J 版中，Neutron 的代码架构如图 5-12 所示。

相信能够将 Neutron 项目的建立的大牛们不会不懂网络的基本原理，这种代码架构对于适应各家厂商将 Neutron 和其自身的 Plugin 进行对接非常的合适，但这样也会导致以下问题：纵然实际使用环境中已经选择了某种 Plugin，但是各种 Plugin 的中间层代码依然在运行，导致了 Python 程序运行代码增多而执行效率降低，还会因潜在的缺陷带来运行的不稳定因素；而由于各种不同的 Plugin 的差异，所以对各种 Plugin 的调用需要单独实现，比如 Open vSwitch 则是将上层的功能参数解析成对应的 Open vSwitch 的 CLI 命令在调用下发，这是何等的效率低下，并且如 Open vSwitch 直接用 CLI 配置有了错误时也无法详细反映到上层，这无疑为定位底层错误制造了障碍；而且如果 Open vSwitch 的 CLI 风格做了修改，此部分代码相关部分就需要相应重写。即使是现在的物理交换机配置的时候，直接调用命令行进行配置的自动化工具也是不被看好的，其中也包括某个公司开发交换机时由于技术水平的低劣在 Netconf 中使用了直接调用 CLI 的方式。

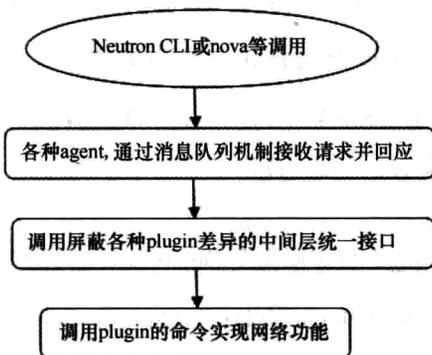


图 5-12 Neutron 代码调用层次结构图

将 Neutron 的上层和底层的开发工作进行了隔离，然而底层和上层的接口定制并没有很好地进行沟通或者有详细的需求及设计文档进行约束，相反，基本是一种盲目的无约束行为；这就造成了现在一种现象，很多参与 Neutron 项目开发的社区人士仅仅关注在上层，懂得如何使用简单的 Ping 和 TCPdump 命令，但对于网络基础知识不甚了解，甚至连交换机的二层和三层的区别都不清楚，也就是被称为的 OpenStack 社区中 Neutron 的“Python 写手”。这类现象刚开始笔者也不太相信是真的存在，直到后来与一位跨国 IT 公司的 Neutron 开发人员交流，笔者“才感觉膝盖中了一箭”；虽然这位开发者声称对 Neutron 的底层和上层技术都非常熟悉，但是讨论到 H 版用 Open vSwitch 作为 Plugin 时却连虚拟机 Ping 8.8.8.8 的时候，相关数据报文从计算节点的虚拟机网卡到网络节点出去的过程根本无法给出清晰描述或解释，仅仅知道在网络节点有一个虚拟交换机 br-ex，更别说能清楚的理解二层转发和三层转发的区别，以及 VLAN Translation 等功能早 Neutron 里使用的意图是什么了。

Neutron 因为多种 Plugin 的架构问题导致了不仅仅是代码臃肿与运行效率低下，带来的更大问题是社区力量分散，大家都在为各自的 Plugin 做着重复的工作，

而且每种 Plugin 都有着各自的缺陷；如果能通过合理的投票和技术抉择方式，每种网络功能都选择只一种 Plugin，开发的时候中间屏蔽各种 Plugin 差异的中间层代码也不再需要，这样代码架构也会更加清晰，维护也必然会更加容易；而且开发者都在为一种底层开发，相信对完善这种底层技术必将大大加速，也会有更多更好更稳定的新功能被开发出来。

摒除多种 Plugin 的做法必会涉及很多网络设备商厂商的利益，导致设备商的设备很难与 Neutron 组件对接，因为都想用特定的 Plugin 与自己厂商的利益对 OpenStack 之 Neutron 组件进行绑定，然而这个绑定对 Neutron 的发展是一种阻碍。笔者个人认为，选择某种 Core Plugin 作为 Neutron 组件的底层后，着力与 Neutron 对接的厂商设备，无论是做虚拟交换机还是物理交换机的 Plugin 厂商，一定要提供这种底层技术使用的统一 CLI 或者 API，这样便可以进行二者的有机统一与调试结合。

类似 Linux 发展历程，现在需要能独断专行的几位牛人带领大家，凭借着远大的眼光、强大的影响力和高效的执行力，言之有理，行之有效，最终为 Linux 的一次次飞跃发展做出卓越性贡献。快速发展的 OpenStack 初期为了赢得最广泛的支持采取了痛苦的策略，现在的开发者也在经历着这种痛苦，而 OpenStack 产品化的厂商也面对着这些 Plugin 做着痛苦的抉择；那让疯狂的技术人员来一次对 Neutron 革命，集合所有 Neutron 项目 OpenStack 社区的力量，对 Neutron 做一次大手术，以实现 Neutron 的再一次飞跃提供动力。此次 J 版为了解决 Multi-Host 的问题做了妥协，只在 ML2 Plugin 的 Open vSwitch 上实现算是在这方面有了很大的进步（但是 DVR 方案依旧是笔者不太看好的 Neutron 公网瓶颈解决方案），OpenStack 社区 Neutron 项目的大牛们，你们会为商业利益所累还是会为技术而狂热，让我们拭目以待。

## 第 6 章

---

# Neutron 网络发展趋势

OpenStack 是目前非常热门的一个开源社区，官方有大量的资料和社区研发人员贡献的代码及文档。但是从国内的博客分享来看，大部分是基于如何搭建出简单的实验环境或某个高级功能的配置步骤，从介绍来看绝大多数分享者对 OpenStack 的网络底层技术并不了解，甚至出现了两台云主机在两个 VLAN 就无法 Ping 通的实验结论，尤其是对网络这块如何搭建部署以保证能建立虚拟机后在虚拟机内部可以正确地访问各种网络，大多数人没有思路或基本的网络概念，也没有对出现问题时排查纠错的能力。因为网络虚拟化在 OpenStack 进行产品化中提供服务的重要性，每年的 OpenStack 会议都有大量的会议在讨论网络相关的技术，而针对 Neutron 的不足来解决相应问题也是 SDN 体现价值的一个重要着眼点。OpenStack 的缺点已经被各种吐槽，但是依然挡不住 OpenStack 对云计算相关人员的吸引力；对于网络组件 Neutron 笔者认为 OpenStack 为了获取最广泛的支持，采取了吸纳百家的策略，就是现在项目众多，耦合性也越来越大；最麻烦的是，对于计算、网络和存储的虚拟化，它不是经过考虑讨论后慎重选择了一种最有前途的技术，然后进行专注和持续积累的方式，而是选择了每种技术都支持且经过抽象封装了一层中间层，这就导致 OpenStack 的专注无法集中，虽然持续但是开发者的力量被分散，代码需要经过多层的封装，自然也就效率非常低

下，比如网络的 Neutron 中每修复一个 BUG 就需要保证对所有的 Plugin 都修改正确等。各家基于 OpenStack 的云计算公司发行的商业版本除了跟随线上不断修正其固有的 BUG 外，还必须关注和解决上述问题。

---

## 6.1 SDN 的结合

SDN 的核心思想和目标在于实现应用程序对底层网络资源的直接控制和使用，控制与转发的分离的原则很大程度上迎合了 SDN 的这种需求；在云计算网络中，比如 OpenStack 通过 Neutron 组件来集中控制对底层网络的资源进行控制并且有开发的 API 接口，这点为 Neutron 和 SDN 的结合提供了先天的优势。需要注意的是，SDN 控制器在云计算网络里，不仅仅需要控制虚拟交换机，也需要将物理交换机一起控制，才能对整个系统的网络资源有个全局判断和最优使用，这点来讲现有的控制器大部分还都需要继续完善。

云计算随着部署规模的不断扩大，租户的数目日益增多，租户的 VM 也会随着业务的扩展急剧增加，而不同业务的种类也会随着上升，这个时候在底层网络来看就是各种特征放入数据流量的种类不断复杂多样化；这些现象都体现了云计算网络流量的部分特征：底层网络逻辑拓扑变化频率高、业务流量种类多及流量变化率大，而这些都要求云计算网络在为租户提供了网络高性能和高稳定性外，还要能适应云计算网络业务不断频繁变化带来的 QoS 和全局敏捷感应等功能的灵活要求，这点属于对 SDN 的一个迫切需求；另外云计算平台的网络架构也会存在各种缺陷，比如 OpenStack 中 Neutron 组件的网络节点的公网瓶颈，可以结合 SDN 技术将 Neutron 的底层架构进行适当颠覆或替换，来弥补 Neutron 网络的缺陷，以满足用户的定制化需求和种类繁多业务的需要。由此来看 OpenDaylight 和 Juniper 的 Open

Contrail 等控制器与 Neutron 的结合将是 SDN 和云计算网络发展的良好契机,为 IAAS 实现通过 SDN 控制 Neutron 底层网络提供新的应用场景,也必将因两种火热技术的结合而带来网络界历史的新革命。

现在已经有很大云计算公司开始此方面的研究,包括海云捷迅和 Unitedstack 等,还有积极参与 SDN 应用到云计算领域的国内芯片商苏州盛科网络 (Centec),及网络设备生产商华为、华三 (H3C)、Pica8 (<http://www.pica8.com>)、Xnet (<http://www.xnetec.com>) 和 DCN (神州数码网络有限公司) 等。其中 Pica8、盛科和 Xnet 等几家公司是软件定义网络 (SDN) 的领先创新者,给网络行业带来协作和革新。Ucloud 是基于自研的商业化比较早的云计算平台,在业务迅速扩张的情况下,可以遇到同行中某些方面还没有涉及的技术场景,所以在应用 SDN 解决云计算的问题上,所积累的解决网络方面问题的经验也较为丰富。

Neutron 结合 SDN 在结构上有三种方式: Neutron AS APP, Neutron AS part of Controller 和 Neutron AS Underlay: 三种方式各有优缺点,解决的问题和引入的缺陷也都有所不同,需要根据业务特点和系统架构进行合理选择。另外,需要注意的是,网络安全问题上也需慎重考虑。著名的网络安全厂商绿盟 (<http://www.nsfocus.com/>) 的刘文懋等,专注于云计算网络中结合 SDN,提供了相应的方案和架构来解决云计算网络安全方面的问题。

---

## 6.2 硬件网络设备解决性能问题

云计算以虚拟化为基础的特征决定了网络部分也需要一定的虚拟化,但是这部分虚拟化网络需要物理服务器在数据中心内的物理网络有统一的协调配置,不

能存在冲突的配置；而最重要的是虚拟化网络到底在整个网络工作量中占用多大比重，哪些转发的步骤必须通过网络虚拟化来实现都是值得思考的问题，因为网络虚拟化的实现依赖于系统的软件实现，这部分的性能和稳定性相比于硬件交换机实现的物理网络来讲都是不可同日而语的；而且如果虚拟化网络的转发步骤太多时，需要在物理服务器内占用很多系统的资源，包括 CPU、内存、PCI 总线和网卡等，但是这些资源中的 CPU 和内存本应尽可能多地给计算部分来使用，现在虚拟化后不仅仅要在对网络和计算两部分的工作上进行适当的资源分配，另外还需要考虑整个物理设备投资的成本上相比于所获取的功能孰轻孰重。

如果仅仅考虑虚拟网络的性能和稳定性来讲，将虚拟网络的部分功能尽可能地转移到物理交换机来实现是一种非常好的优化方式，这个方案的经典例子就是苏州盛科和 Ucloud 一起合作的 Offload 方案；但是这个方案，需要使用者考虑现有设备是否存在硬件限制或采用 Offload 方案后，功能是否有缺失等方面的问题。

另外，针对 OpenStack 中 Neutron 中间网络节点性能问题，Intel 提出了 ServerSwitch 的方案可以较好地解决这个问题，它是使用 X86 CPU+交换芯片来硬件实现网络节点并提升网络节点的网络带宽和稳定性，交换芯片进行大规模数据的转发，CPU 来实现常规路由器中 NAT 等交换芯片不擅长的工作，并且 Intel 的交换芯片转发表项用大规模的 TCAM 来实现，支持 OpenFlow 标准，非常容易后续做 SDN 的融合；这个方案一方面还没有成熟的产品发布和使用案例，另一方面就是在投资成本上可能在私有云或公有云都需要值得考虑，但是笔者认为在解决 OpenStack 的 Neutron 网络性能方面是一种比较完美的解决方案，因为性能问题最终还需要硬件的参与来解决。

## 6.3 安全和监控

安全和监控是网络领域一直重大的研究课题，监控是对网络的各种流量进行统计识别，安全则是在监控的基础上对异常流量的适当处理，并保证正常流量的转发。

从业务安全上，如何说服客户将自己的业务尤其是核心业务的敏感数据搬到云平台，尤其是公有云平台上，本身就是个巨大的难题，这不仅仅涉及信息安全问题也涉及商业核心机密方面的保障问题；虽然可以用客户的财政资金存进了银行此类例子来试图说服客户将业务搬到云平台上，但是从隐秘性而言，客户存钱到银行只是一部分业务类似，因为有些信息对客户而言可能根本上就不想让其他公司的人甚至非公司核心人员掌握，甚至于这些信息的存在都不想被低信任级别的人员知晓；所以从这点来看，云计算业务市场肯定是公有云与私有云同在，公有云跑用户不敏感的业务以减少维护和运营的成本，而私有云上则布置核心业务来保障对公司核心业务数据的安全性，那么如何将公有云和私有云统一管理和底层网络的互通则是摆在人们面前的难题，所以将来的云市场在混合云方面的发展和需求是趋势。

从技术的实现上，传统网络无论是企业内网还是数据中心的网络，在入口处都需要防火墙防止外网的入侵和木马病毒的肆虐，对于提供向外网用户提供服务的网络设备还需要用流量清洗系统对 DDoS 攻击进行识别和防御；这些安全问题在云计算的虚拟网络一样存在，而且虚拟网络里的安全和物理网络有些不同，最大的差异在于物理网络可以随时通过 MIRROR 等方式进行监控，而虚拟网络的流量监控因为是在一个服务器内部，无法通过插拔线缆方式来监控和定位，虽然现在 Neutron 中可以通过 Open vSwitch 等支持 MIRROR 和 sFlow 等流量监控功

能，但是监控流量转发到物理服务器外部也需要占用网卡的带宽和 CPU 等资源，势必对物理服务器的资源规划产生影响；另外为了让监控平台尽量少占用云计算的资源，监控分析部分的系统会尽量布置到云计算系统之外的物理服务器上，这样做也是对监控系统稳定和性能的一个重要保障，但是云计算网络的某些流量尤其是内网流量，比如某个租户在同一个服务器的两台 VM 之间的东西流量，这些流量需要被识别和监控到就会有点复杂，一方面是采集服务器的内网流量（比如一个租户分布在同一个计算节点里同一个隔离域内的两台 VM 之间的通信流量）非常困难，并且流量采集信息大时会占用外卡数据带宽，并可能导致无法进行流量统计分析，另一方面则是流量采集的方式很可能引起租户对其业务安全和数据保密的担忧、从而导致无法部署流量采集，最后则是当流量采用 Overlay 新技术进行隔离时，若要对这种网络环境里的流量实现监控就需要对 OTV 数据流进行识别和统计，这就需要后买新的硬件，而在物理设备上加大投资。

这些问题都是向云计算研发厂商、云计算网络设备商、云计算监控设备商或云计算安全厂商抛出的挑战，如何保证云计算网络中用户 VM 里的业务稳定运行前提下，向用户提供包括物理服务器、租户虚拟机、物理交换机和虚拟交换机等硬件设备的监控和保证租户虚拟机运营业务的网络安全的服务，值得每个云计算网络监控和安全相关研发从业者思考。

---

## 6.4 虚拟网络中的路由协议

现在的路由协议用得较多的是 RIP、OSPF、IS-IS 及 BGP 等。在传统网络中路由协议对网络的互通互联起到了非常大的作用，对于像互联网这么大规模的网

络，如果没有路由协议而靠手工配置来实现互通互联其工作量和维护复杂度将是不可想象的，甚至一个只有十万级服务器内部的网络，没有路由协议来做到互联互通和运营维护都是不可能的。当云计算网络的 VM 和虚拟路由器等设备不太多且三层架构配置清晰时，暂时不用网络路由协议无可厚非，但是随着云计算网络技术研发力量的日益增多，网络虚拟化的技术越来越成熟，支持的功能也就会越来越丰富，那么租户的数目、VM 的数目、Vrouter 的数目和网段的数量都会急剧增加，当某个租户的 VM 达到几万级别的数量级来实现某个数据中心的整个服务时（虽然这个假设近期内看来不太现实），就需要运行路由协议来实现跨网络的互联互通，那么这个时候路由协议在虚拟网络中就需要被引入，现在 Neutron 的实现基本是靠 ipv4\_forward 的静态路由+IPtables 等技术，相信将来对 BGP 等路由协议的支持也会慢慢的实现（[https://wiki.openstack.org/wiki/Neutron/BGP\\_MPLS\\_VPN](https://wiki.openstack.org/wiki/Neutron/BGP_MPLS_VPN)），比如结合 Quagga 来做等。但是虚拟网络中跑大量路由协议的性能和稳定性如何，现在几乎没有任何经验积累。云计算里不同数据中心之间 L2 over L3 的通信的方案很多是 OTV（Overlay Transport Virtualization）的方案。

---

## 6.5 IaaS 上的商业模式

无论采用哪云计算平台，或者云计算凭条无论是采用开源方案比如 Openstack 还是完全基于自研，云计算的服务都可以分三个等级：IaaS（Infrastructure-as-a-service）、PaaS（Platform-as-a-Service）和 SaaS（Software-as-a-service），现在 IaaS 和 PaaS 融合是趋势，从国内国外提供 IaaS 云计算厂商提供的服务来看基本都提供了 PaaS 的服务，但是从事 SaaS 的厂商还比较少，国外比较出名的 Intercloud 和国

内的寄云科技。不过值得一提的是，AWS 的研发者们坚信的是 Web Service 可以满足一切 IT 人的需求，而不关心其所提供的服务到底是 IaaS、PaaS 和 SaaS 的哪一种。

从国外的技术研发模式上，通常是技术人员对市场做了一定的调研基础上，看到某项技术或方案在不久的将来有很大的市场前景，然后就会用几年的时间专注于该项技术或方案的积累；而国内商业模式有很大不同，急功近利的比较多，因为国内外创业的环境不同，如果不能短期盈利，融资商往往就不太看好，这样给创业公司的压力非常大，必须在研发新技术的同时考虑尽快盈利；国内公司在这种历程中成长起来，即使成为了大公司也是基本上无利不为。

现在云计算提供 IaaS 服务的创业公司在国内如雨后春笋般的出现，IaaS 和 PaaS 融合也是一种趋势；对于提供 SaaS 服务的公司如果没有自己的 IaaS，起初可能因为商业盈利不大或者说商业模式前景不明朗，这些做 IaaS 技术的公司也着眼于 IaaS 技术的研发，暂时不会关注 SaaS 方面的业务；但一旦 IaaS 的公司做起来成熟后，并且发现了 SaaS 的盈利点，就可以非常容易地在自己 IaaS 平台上做起 SaaS 的服务，与其他的 SaaS 服务的公司争抢 SaaS 市场的蛋糕，而 IaaS、PaaS 和 SaaS 一起提供服务的成本肯定要比在其他 IaaS 上提供 SaaS 服务的商业模式成本要低，这无疑会给没有自己 IaaS 服务的 SaaS 厂商带来致命灾害；虽然说业务客户和经验的积累非常重要，但是当一家公司在互联网某项业务上独大甚至到垄断的地步时，其所想做的业务起点都将是一个很高的出发点，对那些规模不是很大正处于发展期公司冲击来讲是不可估量的。所以依据笔者浅陋的见识在国内不看好只提供 SaaS 服务的厂商，如果做 SaaS 必须有自己的 IaaS，但 SaaS 在云计算里如同移动互联网里各家互联网公司要控制移动终端的入口一样重要。

另外，云计算的三种服务模式中无论是哪种运营模式，都需要在一个垂直行业里深入挖掘，而不是泛泛的没有特色。这方面 UCloud 是做得比较好的一个示例。UCloud 云计算平台已经为游戏、移动互联网、大数据、电子商务、SaaS 等多个领域提供 IT 基础架构支撑，得益于业内领先的云计算技术和技术团队的专业经验，UCloud 已成功为国内外上万家企业用户提供服务，大幅降低了用户使用 IT 基础设施的成本及技术门槛，国家工信部首批认证通过的“可信云计算企业”。

---

## 6.6 云计算时代的终结

基于虚拟化的云计算的盛行根本原因一方面是基于提升了 CPU 的使用率节省了数据中心空间、物理服务器和网络设备的成本，另一方面提高了对设备的运维的方便简易性；从提升 CPU 使用率优势上来说，暂时没有太多的技术对其现在造成威胁，从对设备的维护性上来说对创业小公司是来部署自己的服务可谓方便了，对于 IDC 服务商和大型互联网公司的数据中心来说都有很丰富的运维经验，随着新的技术出现，运维方便性这点优势可能会慢慢变弱。所以云计算的技术有以下问题。

- 单台 VM 所能占用的最大核数目受限于单台物理机的 CPU 核数目，虽然一台物理服务器可以虚拟出很多核，但毕竟有一定的比例，而且只提高这个比例，最终会在性能和稳定性之间做折中，比如，Nova 里默认的比例值是 16。试想：将来在网络技术高度成熟的情况下，仅从技术而言是否可能单台 VM 享用不同物理服务器上的 CPU？
- 物理服务器本来是应该最大限度的用于计算方面的，网络方面的任务交由

网络设备来完成；但是现在虚拟化环境下，网络虚拟化才能满足各种需求，这点不仅是业务的需要，也为不同厂家重新瓜分 IT 行业硬件和软件的市场份额提供又一次大好的机会，也为一批创业公司提供了创业技术路线；但是云计算环境里绝大多数物理网络设备尤其是 TOR 工作承载是不饱和的，如何在不影响现有云计算系统正常运行的前提下将网络工作所占物理机的计算资源归还给计算任务，尽可能利用网络设备 TOR 来提供功能丰富、稳定性和性能都较高的网络功能，除了现在的 VEPA 或者盛科的 Offload 方案，是否还有新技术出现达到另一个网络性能极限？

- 曾经看到有人在微博或者其他信息渠道说云计算普及后运维及其他相关人员的就业将是一个不可估计的负面影响，将会造成大量的失业，这个笔者认为危言耸听，真是“语不惊人死不休”；常规意义上云计算普及后所说的方便了运维，这个运维是针对云计算的租户来讲的，对于那些提供云计算服务的厂商来说这么大规模的包括物理和虚拟化在内的服务器、网络设备和存储设备，运维工作量只会更多，当然运维技术和运维工具能简便运维工作是另外的话题；从技术进展的变革历史来看，每一次的技术变革，只是将原来陈旧的技术淘汰，只要有学习能力紧跟技术潮流和信息时代发展方向，就不会被时代所抛弃。
- 在传统网络中，某种以五元组（源 IP、目的 IP、协议号、源端口和目的端口）为标记的 TCP 或 UDP 的某一业务流，如果服务器接收报文出现了乱序，对服务器来讲需要进行对这些报文进行去重和重排序的处理，这个工作非常消耗 CPU 的性能，尤其是在现在云计算环境下对服务器的性能影响更是需要关注；现在的做法通常是避免服务器的重排序工作，暂时还

没有有效的解决方案；如何让对报文重排序的解决方案更有效，以应对报文业务流的重排序导致服务器的性能下降问题，一直以来非常重要但是进展缓慢；如果这个问题能解决那么不仅仅对服务器对整个互联网和数据中心内部的网络都将产生深远的变革。

如果将来某些技术尤其是硬件的革新或软件方案的优化导致 CPU 在物理机上也能飙升到满意值，以及云计算技术自身不同方案的迭代比如现在的 Docker 等技术，那么云计算技术或某种方案是否还需要？云计算里所涉及的所有计算、存储和网络技术或许也都只是 IT 技术发展史上的一个阶段，后续将会有多少新的问题出现来驱动产生新的技术或方案，永远是个说不清的谜底。一句话，没有永远流行的技术，CPU 的摩尔定律还没有停息，云计算的技术和 SDN/NFV 技术也许都只会辉煌一时，但终究都会成为过去，这也正是《人月神话》里所讲的“没有银弹”的从另一个角度的理解。

# 附录

---

## 附录 A Open vSwitch 基本命令

### 1. 查看 ovs 版本

```
ovs-appctl --version
ovs-appctl (Open vSwitch) 2.0.0
Compiled Feb 11 2014 16:52:28
```

### 2. 添删查看 Open vSwitch

```
ovs-vsctl add-br br-ovs
ovs-vsctl list-br
ovs-vsctl del-br br-ovs
```

### 3. 添删查看端口

```
ovs-vsctl add-port br-ovs port1
ovs-vsctl list-ports br-ovs
ovs-vsctl del-port br-ovs port1
```

### 4. 添加动态汇聚端口

```
ovs-vsctl add-bond br-ovs bond0 enp6s0 enp7s0 lacp=active
ovs-vsctl del-port br-ovs bond0
```

### 5. 设置端口的 VLAN ID

```
ovs-vsctl set port0 eth0 Tag=1//VLAN ID
```

### 6. 查看 MAC 地址表

```
ovs-appctl fdb/show br-ovs
```

### 7. 查看端口号（可用于流规则匹配项）

```
ovs-ofctl show br-int
```

## 8. 查看 Open vSwitch 配置

```
ovs-vsctl show
```

## 9. 添删和查看流规则

```
ovs-ofctl add-flow br-ovs
idle_timeout=0,in_port=2,dl_type=0x0800,dl_src=00:88:77:66:55:44,
dl_dst=11:22:33:44:55:66,nw_src=1.2.3.4,nw_dst=5.6.7.8,nw_proto=1,tp_src=1,t
p_dst=2,actions=drop
ovs-ofctl del-flows br-ovs in_port=2 //in_port=2 的所有流规则被删除
ovs-ofctl dump-flows br-ovs
```

流规则支持的匹配字段还有 `nw_tos`、`nw_ecn`、`nw_ttl`、`dl_VLAN`、`dl_VLAN_pcp`、`IP_frag`、`ARP_sha`、`ARP_tha`、`IPv6_src`、`IPv6_dst` 等；支持流动作还有 `output:port`、`mod_dl_src/mod_dl_dst`、`set field` 等。

## 10. 添删 sflow 功能

```
ovs-vsctl -- --id=@s create sFlow Agent=enpls0f0
target="\192.168.10.2:6343\" header=128 sampling=64 polling=10 -- set Bridge
br-ovs sflow=@s
ovs-vsctl -- clear Bridge br-ovs sflow
```

## 11. 设置 SDN 控制器地址

```
ovs-vsctl set-Controller br-ovs ssl:192.168.100.1:6633
ovs-vsctl set-Controller br-ovs tcp:192.168.100.1:6633
ovs-vsctl get-Controller br-ovs
ovs-vsctl del-Controller br-ovs
```

## 12. 配置和删除 QoS 功能

```
ovs-vsctl -- set port enpls0f0 QoS=@QoSq -- --id=@QoSq create QoS
type=Linux-htb other-config:max-rate=1000000 queues:0=@queue1 -- --id=@queue1
create queue other-config:min-rate=1000000 other-config:max-rate=10000000
ovs-vsctl list QoS
ovs-vsctl clear Port enpls0f0 QoS
```

## 13. 镜像的配置和删除

```
ovs-vsctl -- set Bridge br-ovs mirrors=@m1 -- --id=@enpls0f0 get Port
enpls0f0 -- --id=@enpls0f1 get Port enpls0f1 -- --id=@enp6s0 get Port enp6s0
-- --id=@m1 create Mirror name=mirror1 select-dst-port=@enpls0f0,@enpls0f1
select-src-port=@enp6s0,@enpls0f1 output-port=@enpls0f1
ovs-vsctl remove Bridge br-ovs MIRRORS mirror1
```

## 14. 其他更多的信息请参见 `man ovs-vsctl` 和 `man ovs-ofctl` 等。

## 附录 B 深入理解 OpenStack 云计算 VLANManager 网络流的六种场景<sup>①</sup>

在最近我（这里指 Piotr Siwczak）的两篇博文里主要介绍了 OpenStack 的一些重要概念，包括 VLANManager 和 Floating IPs。这篇博客在前面内容的基础上主要来说明在不同场景下流量传输是如何进行的。在阅读本篇博客之前强烈建议先阅读下前面的 VLANManager 和 Floating IPs 等博文。另外阅读之前需要注意的是本文后续所提及的所有场景都是基于 Multi-Host 网络模式。Single-Host 网络是被忽略的，因为这种模式被视为是可能单点故障的（Single Point of Failure, SPOF），而且在现实的产品部署中应该被避免这样做。

在每一次 OpenStack 的部署中，对于相关通信和配置的需求都是有非常大的不同。大多数情况下，下面几项条件是至少需要被满足的：

- VM 被创建以后，相应实例必须能从内网即 Fixed 网段中获取一个 IP 地址；
- 基本上通常情况下实例需要能够访问互联网。比如可以下载安全更新；
- 通常实例必须能够与 Fixed 网络里其他的 IP 地址进行通信；
- 一些实例需要用公网路由的 IP 地址形式来暴露给网络世界。

第三件事就是需要注意租户的隔离，这个问题涉及面很广，从每一个租户都

---

<sup>①</sup> 本附录是对 Piotr Siwczak 于 2012 年 8 月 14 日发表的一篇文章（网址为 <https://www.mirantis.com/blog/vlanmanager-network-flow-analysis/>）的翻译，其中稍有修改。——作者注

被封闭隔离在各自的 Fixed 网络里，到另一个极端是这些网络都 100% 的可见性。因此第一个场景是典型的公有云中的一个场景（租户们默认不想把他们的服务器暴露给其他人），后续还会遇到企业云，这里因为所有的租户都属于一个企业组织（例如不同的研发组之间）并且通常对他们的虚拟机实例互通存在需求。

考虑如图 B-1 所示的网络拓扑。

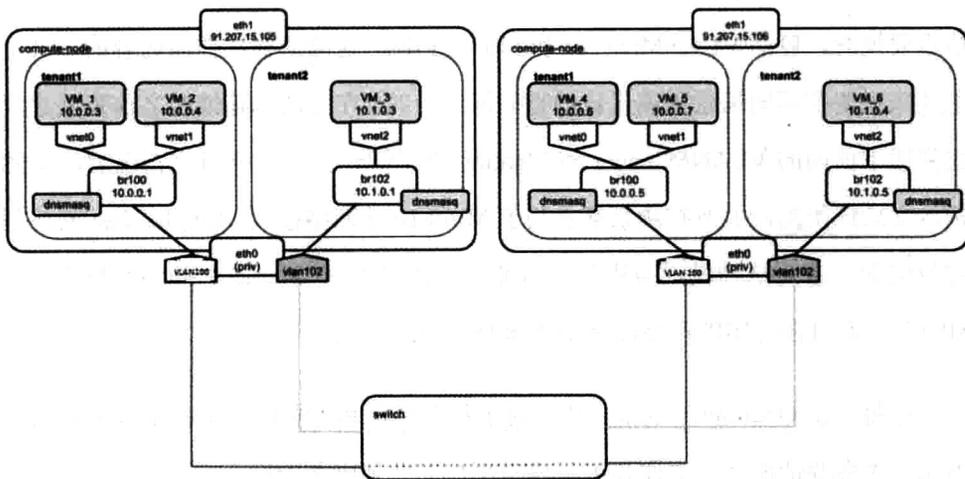


图 B-1 两个计算节点通过网络交换连接通信示意图

图 B-1 显示了两个计算节点通过一个网络交换连接进行通信的示意图。我们已经有租户 1 和租户 2，两个各自有自己不同的私有网络，分别隶属于不同的 VLAN 内（100 和 102）。因为我们是在一个 Multi-Host 网络配置上运行，所以每一个计算节点可以直接的访问外网（比如互联网）。为了实现这个目的，有一个端口 eth1 被使用。基于这张图我将展示 6 种场景一来说明 OpenStack 网络不同场景下是如何产生差异的。

## 1. 场景 1

租户 1 的实例启动并且有了分配到的 IP 地址，流程如图 B-2 所示。

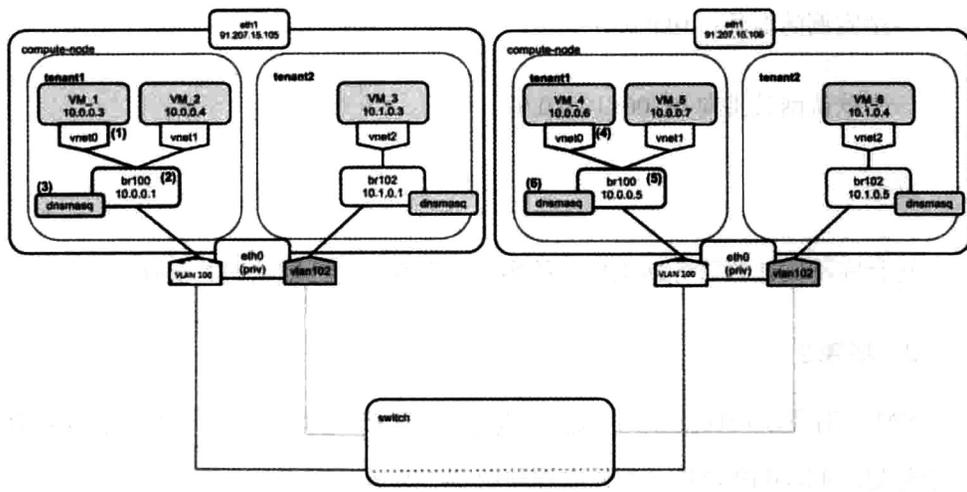


图 B-2 VM 实例 DHCP 通信流程示意图

实例 VM\_1 启动后接着发送了一个 DHCPDiscovery 广播信息在自己的本地网络里。这个信息会在 br100 里被广播传送。

Dnsmasq 服务器监听于 br100 的地址范围内（配置参数 “-listen-address 10.0.0.1”）。它也会通过包含在 DHCPOffer 里的信息为 VM\_1 进行一个静态租赁配置。

一个实例的地址：10.0.0.3；

一个默认网关指向 br100: 10.0.0.1；

实例 VM\_4 启动后接着发送了一个 DHCPDiscovery 广播信息在自己的本地网络里。这个信息会在 br100 里被广播传送。

Dnsmasq 服务器监听于 br100 的地址范围内（配置参数 “-listen-address 10.0.0.5”）。它也会通过包含在 DHCPOffer 里的信息为 VM\_4 进行一个静态租赁配置。

一个实例的地址：10.0.0.6；

一个默认网关指向 br100: 10.0.0.5；

注意：

运行在不同计算服务器主机上的实例，会有不同的默认网关配置。

## 2. 场景 2

VM\_1 有需求访问互联网（比如 Google's DNS: 8.8.8.8 地址）并且它仅仅有一个分配的 Fixed IP 地址。具体流程如图 B-3 所示。

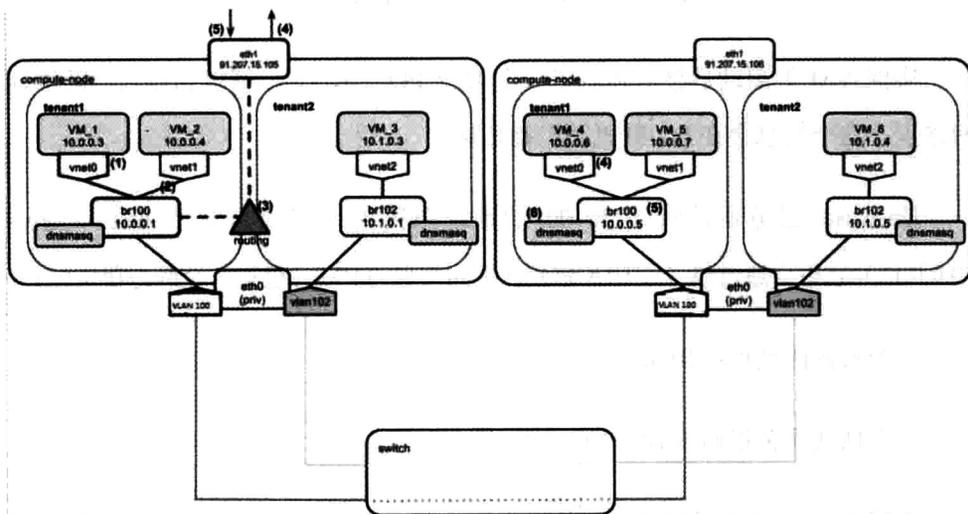


图 B-3 租户 VM 访问外网通信示意图

VM\_1 发送一个目的地址是 Google's DNS 8.8.8.8 的 Ping 报文。

因为 www.google.com 的 DNS 地址不在本地网络内，所以 VM\_1 会将 Ping 报文直接通过默认网关转发出去（在本例中默认网关是 10.0.0.1）。

计算节点的做了路由查找后的转发决定。8.8.8.8并不在任何直连网络的范围内，所以计算节点决定将这个报文通过计算节点的默认网关转发出去(91.207.15.105)。

在报文被转出过程中，通过 SNAT 方式转成了 eth1 的 IP 地址 91.207.15.105。在 Nova-计算 IPtables 的 nat 表里会有一条相应的规则处理这个报文：

```
Nova-network-snat -s 10.0.0.0/24 -j SNAT --to-source 91.207.15.105
```

当 8.8.8.8 会给 91.207.15.105 一个回复时，在 Nova.conf 里对这条规则行为的控制的设置是路由时 source\_IP=91.207.15.105，并且内核里的 NAT 表被用来将报文送回给 VM\_1。

### 3. 场景 3

让我们假设租户 1 里想从 VM1\_1 来 Ping VM\_2。这里有两个重要的事情需要注意：

两个实例都属于租户 1；

两个实例在同一个计算节点里；

两个 VM 实例都属于租户 1。

流量转发看起来将会是如图 B-4 所示。

VM\_1 发送一个报文给 VM\_2。VM\_2 在 VM\_1 的相同网络上。VM\_1 还不知道 VM\_2 的 MAC 地址，所以它会先发一个 ARP 广播报文。这个广播报文通过 br100 被转发到其他租户 1 网络里所有设备上，包括 VM\_2，并且只有 VM\_2

会回复一个 ARP 应答报文给 VM\_1。

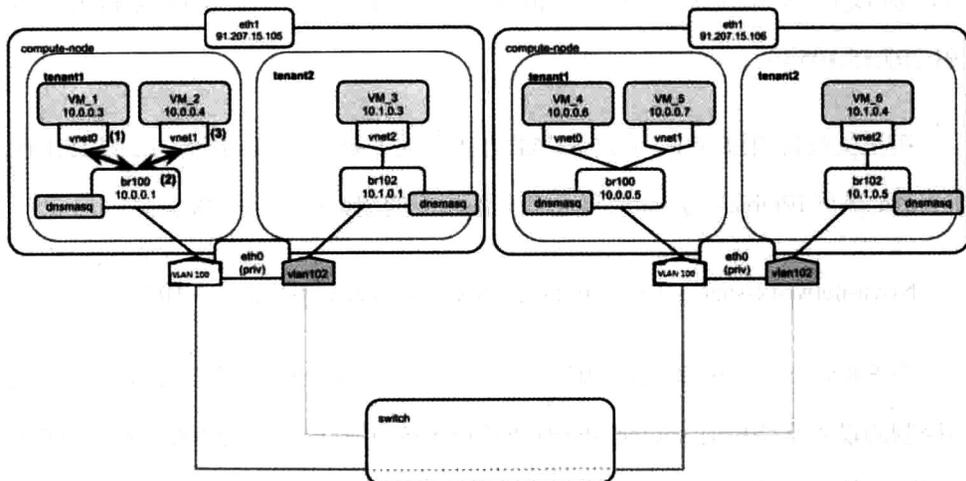


图 B-4 同租户同计算节点的 VM 实例之间通信示意图

一旦 VM\_2 的 MAC 被确定后，IP 数据包就可以直接从 VM\_1 发出。

注意：

相同租户的实例在同一个 L2 的广播域内。这个广播域在计算节点上通过 VLAN100 的接口和交换机来扩展范围，这个交换机支持 802.1Q VLAN 流量。所以租户 1 在所有创建了 VLAN100 接口的计算节点网络上，所有的 ARP 广播在本地网络上都是可见的，更不用说 VM\_1 和 VM\_2 在同一个服务器上，并且在第一点中 ARP 广播的发送也能被 VM\_4 和 VM\_5 所接收到。

#### 4. 场景 4

现在让我们假设从 VM\_1 来 Ping 虚拟机 VM\_5。两台虚拟机都属于租户 1，但是却分布在不同的物理计算节点上，如图 B-5 所示。

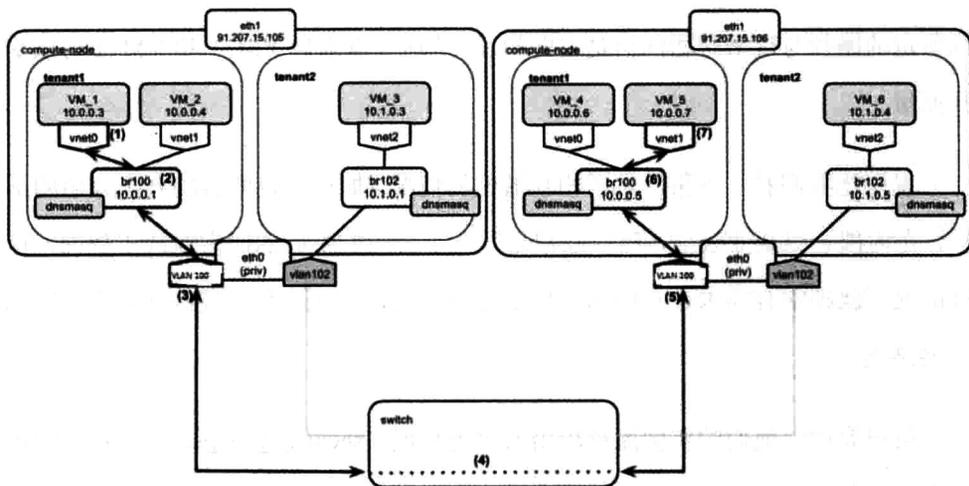


图 B-5 同租户不同计算节点上 VM 之间通信示意图

VM<sub>1</sub> 想发送一个报文给位于不同计算节点的 VM<sub>5</sub>, 首先会发送一个 ARP 广播报文来确定 VM<sub>5</sub> 的 MAC 地址。

广播报文将被网桥 br100 传送给其所有相连的接口, 包括 VLAN100。

这个报文被在计算节点的端口被加上一个 802.1Q VLAN 的 Tag 100。

带 Tag 的报文经过交换机转发时, 需要交换机的端口都被配置成 Trunk 模式。Trunking 模式允许信息报文带着 VLAN Tag 在两个计算节点之间传输。

随后带 Tag 报文到达另一个计算节点的物理网络接口。因为报文带着 Tag 100, 所以报文将被进一步转发到 VLAN 100 的接口。并且报文的 Tag 也会从这里删除。

报文穿过 br100 继续被转发。

VM<sub>5</sub> 接收这个广播报文并用其 MAC 地址答复 ARP 请求, ARP 应答报文返

回所走的路径与过来时相同但是方向相反。从这一点来说 VM\_1 和 VM\_2 可以交互流量。

现在让我们看一下租户之间相互通信的情况。如在这篇博客的开头介绍的那样，在内部云或企业云里，租户经常是同一个公司的不同的研发组或工程部，这种情况下就经常有需求让不同的工程部之间互通，甚至它们可能在不同的 Fixed IP 网络里。

用户允许对他们的实例流量应用安全组功能 (security groups.)。在我们的场景里，在用户 1 和租户 2 的云里可以通过调整如下的安全组规则简单的实现二者实例的通信：

租户 1: Nova secgroup-add-rule default TCP 1 65535 10.1.0.0/24

租户 2: Nova secgroup-add-rule default TCP 1 65535 10.0.0.0/24

租户 1 让从租户 2 网络 (10.1.0.0) TCP 所有端口范围 1-65535 的流量都方向，反之亦然。

根据实例在 OpenStack 中所处的位置，可以分为如下两种场景：

两个租户的不同实例位于同一个计算节点；

两个租户的不同实例位于不同的计算节点。

## 5. 场景 5

这次的 Ping 从 VM\_1 到 VM\_3。在这个情况下，两个租户之间会有数据报文的交互，如图 B-6 所示。这点对于理解两个网络之间是如何进行路由起作用的

非常关键。

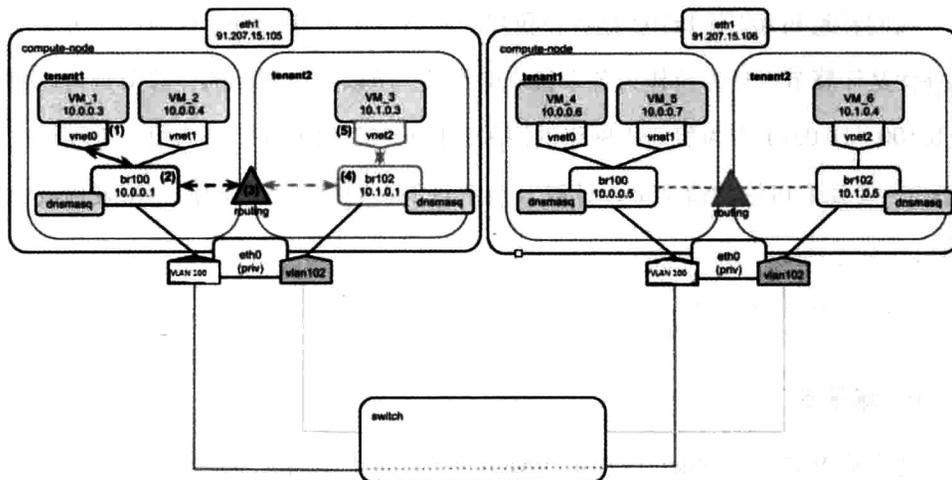


图 B-6 不同租户同计算节点通信示意图

VM<sub>1</sub> 想达到位于同一个计算节点的租户 2 的 VM<sub>3</sub>。VM<sub>1</sub> 能判断出 VM<sub>3</sub> 在各自不同的网段上 (10.0.0.0/24 vs 10.1.0.0/24)，因此它直接将报文转发给地址为 10.0.0.1 的网关。

报文到达 br100。

计算节点根据内部的额路由表将这个报文路由到 br102。

这个报文达到 br102。VM<sub>3</sub> 的 MAC 地址通过 ARP 广播来确定。

VM<sub>3</sub> 应答其 MAC 地址。因为 VM<sub>1</sub> 是在与 VM<sub>3</sub> 不同的网段上，VM<sub>3</sub> 回复应答报文转发给 IP 地址为 10.1.0.1 的网关 br102。报文然后被 br100 路由回到租户 1 的网络内。

注意：

我们看到 br100 和 br102 被各自租户的网络赋予了 IP 地址。这点的一方面的影响就是计算节点网桥作为各个租户的网关。所以左边开始的计算节点可以感知 br100 (10.0.0.1 作为租户 1 网络的网关) 和 br102 (10.1.0.1 作为租户 2 网络的网关)。在我们讨论的情形下，计算节点上对于不同的网络之间的路由表项就像这样：

```
10.0.0.0/24 dev br100 proto kernel scope link
10.1.0.0/24 dev br102 proto kernel scope link
```

## 6. 场景 6

这次从 VM\_1 里 Ping 虚拟机 VM\_6，它们属于不同的租户并且在不同的计算节点上，如图 B-7 所示。请特别注意观察“非对称路由”(asymmetric routing)，比如从 VM\_1 到 VM\_6 的请求走一条路，但是从 VM\_6 到 VM\_1 的相应报文需流经不同的路由。

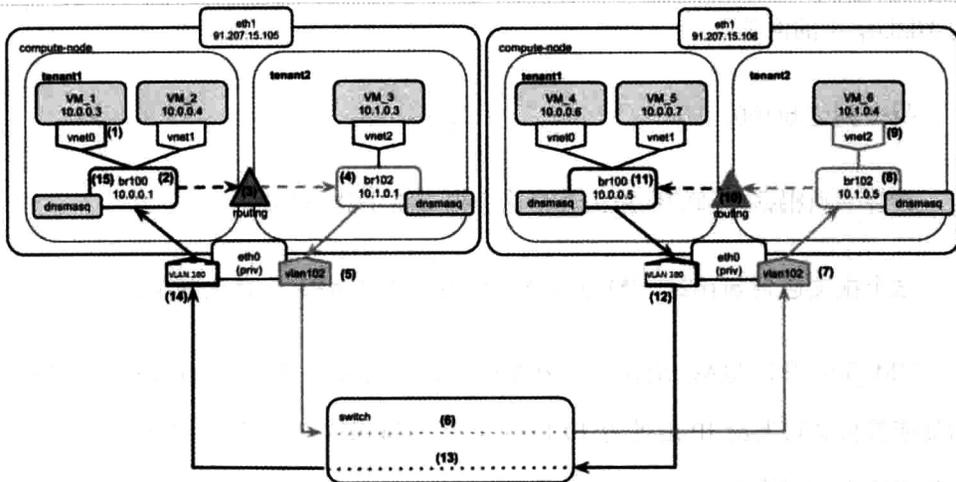


图 B-7 不同租户不同计算节点 VM 之间内网通示意图 情形一

VM\_1 想和 VM\_6 交互数据报文。VM\_6 属于另一个租户并且与 VM\_1 不在同一个计算节点上。因为 VM\_6 不在 VM\_1 的本地网络里，所以在我们的情形下 VM\_1 发送一个目的之地为 10.1.0.4 的报文直接给 IP 地址为 10.0.0.1 的网关

报文从 VM\_1 到达 br100。

计算节点看到了目标网络（10.1.0.0/24 属于租户 2）在 br102 上，所以计算节点将报文路由到 br102。

报文现在应在租户 2 的二层网段内。

然后报文获得了 802.1Q VLAN Tag 头。

接着通过交换机的 Trunk 端口。

带 Tag 的报文达到物理网络另一个计算节点的接口。因为带了 102 的 Tag，所以被进一步转发到了 VLAN102 的接口。并且报文的 Tag 在此处被删除掉。

报文流经 br102 并且达到 VM\_6。

VM\_6 发送目的 IP 地址为 10.0.0.3 的应答报文给 VM\_1。因为 VM\_1 是在不同网段上的，所以应答报文直接被转发到默认网关，在示例当中 IP 地址是 10.1.0.5。

计算节点看到目标网络（10.0.0.0/24 术语租户 1）在 br100 上所以它将报文路由到 br100 上。

报文现在到了租户 1 的二层网络段上。

然后获得了 802.1Q VLAN Tag。

报文接着通过交换机的 Trunk 端口。

报文达到物理网络左边计算节点的接口。因为报文带着 Tag 100，所以被进一步转发到 VLAN100 的接口。并且报文 Tag 在这里被摘除。

报文经过 br100 到达 VM\_1 后 VM\_1 就得到了回来的应答报文。

但是这种情况下如何处理那？

OpenStack 期望网络能满足要求。这就意味着如果租户在 OpenStack 里创建了一个新的网络，它的配置（例如声明网桥是一个额外的 VLAN 接口）是在默认情况不被所有的计算节点所感知的。通常情况下仅仅在有某个计算节点上有实例挂在到这个新网络上的时候才会将该网络的相关内容在这个计算节点上被启动。例如，在一个计算节点上应该有 br100，br102，br103 的被设置为 UP，然而另一个计算节点暂时只有看到 br102 被建立。

这种行为可能导致一些迷惑的情况。再想象一下我们想从 VM\_1 来 Ping 虚拟机 VM\_6 的情形，如图 B-8 所示。但是这次在左边计算节点上我们没有存在租户 2 的虚拟机实例（因此没有网桥 br102 且也没有 VLAN102 的内容）。但是等一分钟后，因此网桥 br10 作为网段 10.1.0.0 的一个默认网关，我们如何到达租户 2 的网络哪？答案很简单——默认下我们做不到。

VM\_1 想发送到 VM\_6 的 Ping 报文，两者不在同一个网络里。

因此 VM\_1 发送的 Ping 报文会走默认网关（br100，10，0，0，1）。

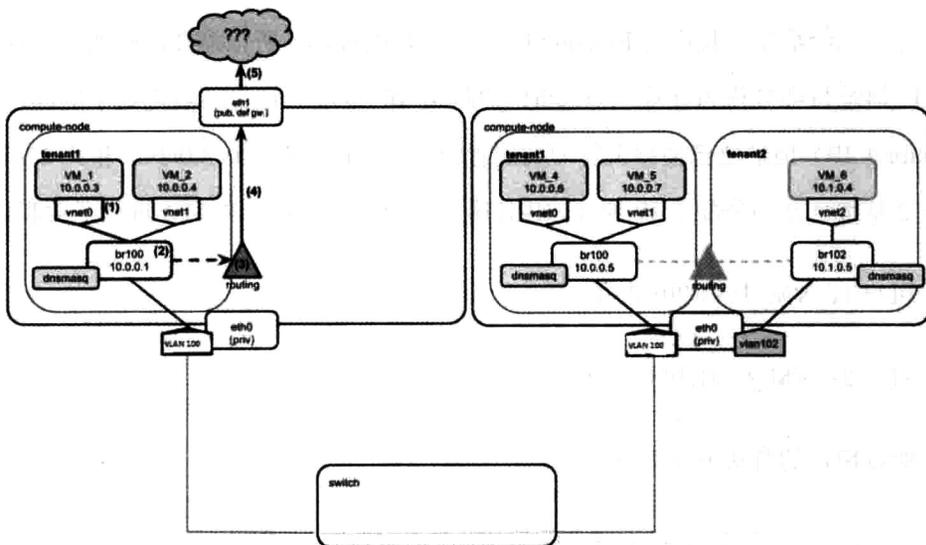


图 B-8 不同租户不同计算节点 VM 之间通信示意图——情形二

在计算节点上这里没有到达 10.1.0.0 网段的直达路由（这里没有本可以提供一个此类网段路由的 br102）。

因此计算节点发送这个报文到默认的网关，而这个默认网关现在恰巧是 eth1。

因为 eth1 位于一个与 eth0 完全不同的网络里，没有任何机会来找到到达 10.1.0.0 网段的路由。我们会得到一个“主机不可达”（“Host unreachable”）的信息。

我们现在可以看到租户间的互通仅仅依赖于 fixe IP 网络不能得到保障。针对这个问题有大量潜在的补救措施，当然最简单的方法（以我来看也是最好的方法）将是在租户间通信时用 Floating IP 通信而不是用 Fixed IP 地址。

## 7. 场景 7

VM\_1 想发送 Ping 虚拟机 VM\_6 的报文，两个实例都有已经分配的 Floating IP 地址，如图 B-9 所示。在先前的博文中（译者注：这里指 Piotr Siwczak 博客中

的文章) 已经给出了我们对 Floating IP 工作原理的理解, 所以需要清晰的明白这个 IP 地址将会被作为计算节点 eth1 的从属 IP 地址。让我们假定如下的浮动 (Floating IP) 地址池已经被云管理员配置为如下网段: 91.208.3.0/24。租户 1 和租户 2 从地址池中分配了 IP 地址, 并且用手工的方式将它们和 VM 进行了关联:

租户 1: VM\_1: 91.208.23.11

租户 2: VM\_6: 91.208.23.16

所以现在的解决方案如下:

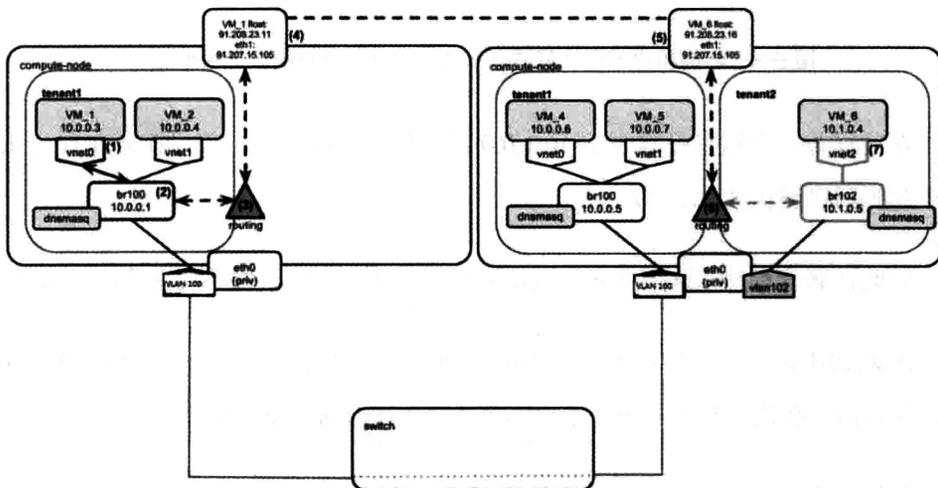


图 B-9 不同租户之间 VM 公网地址通信示意图

让我们设定 VM\_1 想 Ping 虚拟机 VM\_6。租户们使用它们的 Floating IP 地址代替 Fixed 地址进行通信。

VM\_1 想 Ping 虚拟机 VM\_6, VM\_6 的 Floating IP 地址是 91.208.23.16。因此 Ping 报文从源主机 10.0.0.3 到达目的 IP 地址 91.208.23.16。

因为 91.208.23.16 不是一个 VM\_1 的本地网络地址，所以报文将被转发到默认网关 10.0.0.1 处。

计算节点根据转出路由认为将报文通过端口 eth1 转发出去。

报文的源 NAT (Source NAT-ting , SNAT-ting) 被施行 (重写报文的源 IP 地址 10.0.0.3 -> 91.208.23.11)。因此现在报文的源和目的 IP 地址看起来应该这样：

源 IP 地址为 91.208.23.11 和目的 IP 地址为 91.208.23.16。

源 IP 地址为 91.208.23.11 和目的 IP 地址为 91.208.23.16 的报文到达目的计算节点。目的 NAT (Destination Network Address Translation , DNAT) 的转换此处进行，即将报文的源 IP 地址从 91.208.23.16 转换为 10.1.0.4。此时报文的源 IP 地址和目的 IP 地址看起应该分别是 91.208.23.11 和 10.1.0.4。

基于目的 IP 地址，计算节点的路由将报文通过 br102 进行路由。

报文到达目的地址：VM\_6。

VM\_6 给 VM\_1 的响应报文走一样的路径，但是顺序相反。当然也有一个不同点。因为 ICMP 应答报文是和之前的 ICMP 请求报文相关的，但是没有清晰的 DNAT-ting 来完成，因此应答报文通过 eth1 返回到左边的计算节点里。相反地，内核内在的 NAT 表需要被查询。

对其他基于 Fixed IP 地址确保租户间互通的方式 (就我现有的网络知识来考虑，请注意，仅是我个人对这个问题的思考) 需要对 OpenStack 的代码打补丁。

你应该也需要确保当一个新的网络被添加时，Linux 的 Bridge 和相应的 VLAN 接口都会在计算节点被量产时被创建 (现在进在虚拟机被创建的过程中完成)。

一个上游的路由器可以被挂在到 OpenStack 的私有网络组 (eth0)，这个网卡应被视为不同租户网络间互通的网关。然而一些代码需要被添加到 OpenStack 里面去以确保这些路由流量能被有合适的 802.1q Tagging，这个也基于 OpenStack 的网络都配置而来什么 VLAN。

如果 FlatDHCPManager 合适就用它。这种方式仅仅在每一个计算节点仅创建一个网络 Linux Bridge，这从开始就是正确的作做法。

本附录分析了各种场景下的虚拟机实例流量，从虚拟机的产生和启动开始，结束与租户间的互通。现在网络的模型尤其自身的限制并不能总提供用户可能预期的功能。所以，OpenStack 是高度灵活和扩展性的软件。它提供给用户通过合适的配置改变来处理这些限制或通过用户代码扩展满足相应的定制化需求。因此，Quantum 在下一个版本发布时即将到来，并会导致整个网络概念的小革新。

---

## 附录 C RDO 配置文件网络部分——VLAN 隔离

```
[general]
# Path to a Public key to install on Servers. If a usable key has not
# been installed on the remote Servers the user will be prompted for a
# password and this key will be installed so the password will not be
# required again
# The IP addresses of the Server on which to install the Neutron
# Server
CONFIG_NEUTRON_SERVER_HOST=192.168.3.224

# The password to use for Neutron to authenticate with Keystone
CONFIG_NEUTRON_KS_PW=1923a2ea5f8c4210

# The password to use for Neutron to Access DB
CONFIG_NEUTRON_DB_PW=7e22138c645a40bc
```

```
# A comma separated list of IP addresses on which to install Neutron
# L3 Agent
CONFIG_NEUTRON_L3_HOSTS=192.168.3.224

# The name of the Bridge that the Neutron L3 Agent will use for
# external traffic, or 'provider' if using provider networks
CONFIG_NEUTRON_L3_EXT_Bridge=br-ex

# A comma separated list of IP addresses on which to install Neutron
# DHCP Agent
CONFIG_NEUTRON_DHCP_HOSTS=192.168.3.224

# A comma separated list of IP addresses on which to install Neutron
# LBaaS Agent
CONFIG_NEUTRON_LBAAS_HOSTS=192.168.3.224

# The name of the L2 Plugin to be used with Neutron
CONFIG_NEUTRON_L2_PLUGIN=Open vSwitch

# A comma separated list of IP addresses on which to install Neutron
# metadata Agent
CONFIG_NEUTRON_METADATA_HOSTS=192.168.3.224

# A comma separated list of IP addresses on which to install Neutron
# metadata Agent
CONFIG_NEUTRON_METADATA_PW=43acc9a56a7b4dec

# The type of network to allocate for tenant networks (eg. VLAN,
# local, GRE)
CONFIG_NEUTRON_LB_TENANT_NETWORK_TYPE=VLAN

# A comma separated list of VLAN ranges for the Neutron LinuxBridge
# Plugin (eg. physnet1:1:4094, physnet2, physnet3:3000:3999)
CONFIG_NEUTRON_LB_VLAN_RANGES=

# A comma separated list of interface mappings for the Neutron
# LinuxBridge Plugin (eg. physnet1:br-eth1, physnet2:br-eth2, physnet3
# :br-eth3)
CONFIG_NEUTRON_LB_INTERFACE_MAPPINGS=

# Type of network to allocate for tenant networks (eg. VLAN, local,
# GRE)
CONFIG_NEUTRON_OVS_TENANT_NETWORK_TYPE=VLAN

# A comma separated list of VLAN ranges for the Neutron Open vSwitch
# Plugin (eg. physnet1:1:4094, physnet2, physnet3:3000:3999)
CONFIG_NEUTRON_OVS_VLAN_RANGES=physnet1:1:4050

# A comma separated list of Bridge mappings for the Neutron
# Open vSwitch Plugin (eg. physnet1:br-eth1, physnet2:br-eth2, physnet3
# :br-eth3)
CONFIG_NEUTRON_OVS_Bridge_MAPPINGS=physnet1:br-tun
```

```
# A comma separated list of colon-separated OVS Bridge:interface
# pairs. The interface will be added to the associated Bridge.
CONFIG_NEUTRON_OVS_Bridge_IFACES=br-tun:eth1

# A comma separated list of tunnel ranges for the Neutron Open vSwitch
# Plugin
CONFIG_NEUTRON_OVS_TUNNEL_RANGES=

# Override the IP used for GRE tunnels on this hypervisor to the IP
# found on the specified interface (defaults to the HOST IP)
CONFIG_NEUTRON_OVS_TUNNEL_IF=
.....
```

## 附录 D VXLAN 通信抓包实例

针对图 5-9 的物理和拓扑和 5-10 的 Neutron 环境对几种通信情况进行了抓包，包括了同租户同物理机之间 VM 通信、同租户不同物理机上 VM 之间的通信和 VM 与外网的通信等。

### 1. 服务器网络配置

下面是 Server1 服务器里 ifconfig 看到的网络信息：

```
Server1:
ifconfig
br-ex  Link encap:Ethernet HWaddr 00:25:90:C3:B2:EA
       inet addr:192.168.2.211 Bcast:192.168.3.255 Mask:255.255.252.0
       inet6 addr: fe80::c842:b0ff:feaf:5b14/64 Scope:Link
       UP BROADCAST RUNNING MTU:1500 Metric:1
       RX packets:298241 errors:0 dropped:0 overruns:0 frame:0
       TX packets:229907 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:73058174 (69.6 MiB) TX bytes:47725497 (45.5 MiB)
br-int  Link encap:Ethernet HWaddr 1A:56:80:CC:0B:41
       inet6 addr: fe80::4489:fcff:fe54:8744/64 Scope:Link
       UP BROADCAST RUNNING MTU:1500 Metric:1
       RX packets:121 errors:0 dropped:0 overruns:0 frame:0
       TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:11128 (10.8 KiB) TX bytes:468 (468.0 b)
br-tun  Link encap:Ethernet HWaddr AE:F4:CE:C9:05:48
       inet6 addr: fe80::fcec:cdff:fedd:a3ed/64 Scope:Link
```

```
UP BROADCAST RUNNING MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 b) TX bytes:468 (468.0 b)
eth0 Link encap:Ethernet HWaddr 00:25:90:C3:B2:EA
inet6 addr: fe80::225:90ff:fec3:b2ea/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:340622 errors:0 dropped:0 overruns:0 frame:0
TX packets:233207 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:76128145 (72.6 MiB) TX bytes:47948257 (45.7 MiB)
Memory:df980000-dfa00000
eth1 Link encap:Ethernet HWaddr 00:25:90:C3:B2:EB
inet addr:192.168.100.2 Bcast:192.168.100.255
Mask:255.255.255.0 inet6 addr: fe80::225:90ff:fec3:b2eb/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:750 errors:0 dropped:0 overruns:0 frame:0
TX packets:627 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:110891 (108.2 KiB) TX bytes:101247 (98.8 KiB)
Memory:df900000-df980000
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:2435155 errors:0 dropped:0 overruns:0 frame:0
TX packets:2435155 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:785440896 (749.0 MiB) TX bytes:785440896 (749.0 MiB)
qbr87816d39-b0 Link encap:Ethernet HWaddr 82:56:D3:3F:0C:E7
inet6 addr: fe80::9496:elff:fe8d:9df1/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:15 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:1668 (1.6 KiB) TX bytes:468 (468.0 b)
qvb87816d39-b0 Link encap:Ethernet HWaddr 82:56:D3:3F:0C:E7
inet6 addr: fe80::8056:d3ff:fe3f:ce7/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:279 errors:0 dropped:0 overruns:0 frame:0
TX packets:327 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:31398 (30.6 KiB) TX bytes:32301 (31.5 KiB)
qvo87816d39-b0 Link encap:Ethernet HWaddr 92:F2:28:19:83:17
inet6 addr: fe80::90f2:28ff:fe19:8317/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:327 errors:0 dropped:0 overruns:0 frame:0
TX packets:279 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:32301 (31.5 KiB) TX bytes:31398 (30.6 KiB)
tap87816d39-b0 Link encap:Ethernet HWaddr FE:16:3E:2F:7C:89
inet6 addr: fe80::fc16:3eff:fe2f:7c89/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
```

```

RX packets:318 errors:0 dropped:0 overruns:0 frame:0
TX packets:282 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:31623 (30.8 KiB) TX bytes:31608 (30.8 KiB)

```

下面是 Server1 服务器里 ovs-vsctl show 看到的网络配置信息:

```

ovs-vsctl show
171b52d0-f528-43fd-ba46-ec7d06ff35fc
  Bridge br-ex
    Port "qg-009eb02f-6d"
      Interface "qg-009eb02f-6d"
        type: internal
    Port "eth0"
      Interface "eth0"
    Port br-ex
      Interface br-ex
        type: internal
  Bridge br-tun
    Port patch-int
      Interface patch-int
        type: patch
        options: {peer=patch-tun}
    Port br-tun
      Interface br-tun
        type: internal
    Port "Vxlan-1"
      Interface "Vxlan-1"
        type: Vxlan
        options: {in_key=flow, local_IP="192.168.100.2",
out_key=flow, remote_IP="192.168.100.3"}
  Bridge br-int
    Port patch-tun
      Interface patch-tun
        type: patch
        options: {peer=patch-int}
    Port "tapd4894cd5-08"
      Tag: 1
      Interface "tapd4894cd5-08"
        type: internal
    Port br-int
      Interface br-int
        type: internal
    Port "qvo87816d39-b0"
      Tag: 1
      Interface "qvo87816d39-b0"
    Port "qr-87fa0951-54"
      Tag: 1
      Interface "qr-87fa0951-54"
        type: internal
    Port "tap5b507a21-32"
      Tag: 2
      Interface "tap5b507a21-32"
        type: internal
  ovs_version: "1.11.0"

```

下面是 Server2 服务器里 ifconfig 看到的网络配置信息:

```
ifconfig
br-int Link encap:Ethernet HWaddr BE:28:0B:91:E7:46
inet6 addr: fe80::d841:94ff:feab:481b/64 Scope:Link
UP BROADCAST RUNNING MTU:1500 Metric:1
RX packets:97 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:9986 (9.7 KiB) TX bytes:468 (468.0 b)
br-tun Link encap:Ethernet HWaddr FE:F2:7B:43:BE:43
inet6 addr: fe80::ac71:62ff:fe12:315e/64 Scope:Link
UP BROADCAST RUNNING MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 b) TX bytes:468 (468.0 b)
eth0 Link encap:Ethernet HWaddr 00:25:90:C3:B0:6E
inet addr:192.168.3.218 Bcast:192.168.3.255 Mask:255.255.252.0
inet6 addr: fe80::225:90ff:fec3:b06e/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:245452 errors:0 dropped:0 overruns:0 frame:0
TX packets:203847 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:43915710 (41.8 MiB) TX bytes:61552906 (58.7 MiB)
Memory:df980000-dfa00000
eth1 Link encap:Ethernet HWaddr 00:25:90:C3:B0:6F
inet addr:192.168.100.3 Bcast:192.168.100.255
Mask:255.255.255.0
inet6 addr: fe80::225:90ff:fec3:b06f/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:621 errors:0 dropped:0 overruns:0 frame:0
TX packets:759 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:101247 (98.8 KiB) TX bytes:110989 (108.3 KiB)
Memory:df900000-df980000
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:15 errors:0 dropped:0 overruns:0 frame:0
TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:1458 (1.4 KiB) TX bytes:1458 (1.4 KiB)
qbr0da6db3e-d8 Link encap:Ethernet HWaddr 9A:6E:D0:0C:59:D2
inet6 addr: fe80::b05e:2aff:febf:43b6/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:47 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:4580 (4.4 KiB) TX bytes:468 (468.0 b)
qbr431b34bc-17 Link encap:Ethernet HWaddr 96:14:4E:93:AF:10
inet6 addr: fe80::1c88:fdff:fea8:b668/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
```

```

RX packets:30 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:3056 (2.9 KiB) TX bytes:468 (468.0 b)
qvb0da6db3e-d8 Link encap:Ethernet HWaddr 9A:6E:D0:0C:59:D2
inet6 addr: fe80::986e:d0ff:fe0c:59d2/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:195 errors:0 dropped:0 overruns:0 frame:0
TX packets:213 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:23183 (22.6 KiB) TX bytes:21534 (21.0 KiB)
qvb431b34bc-17 Link encap:Ethernet HWaddr 96:14:4E:93:AF:10
inet6 addr: fe80::9414:4eff:fe93:af10/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:183 errors:0 dropped:0 overruns:0 frame:0
TX packets:217 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:21943 (21.4 KiB) TX bytes:21671 (21.1 KiB)
qvo0da6db3e-d8 Link encap:Ethernet HWaddr 6A:AC:74:19:47:9A
inet6 addr: fe80::68ac:74ff:fe19:479a/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:213 errors:0 dropped:0 overruns:0 frame:0
TX packets:195 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:21534 (21.0 KiB) TX bytes:23183 (22.6 KiB)
qvo431b34bc-17 Link encap:Ethernet HWaddr B6:B3:AF:81:A9:31
inet6 addr: fe80::b4b3:afff:fe81:a931/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:217 errors:0 dropped:0 overruns:0 frame:0
TX packets:183 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:21671 (21.1 KiB) TX bytes:21943 (21.4 KiB)
tap0da6db3e-d8 Link encap:Ethernet HWaddr FE:16:3E:19:E4:D1
inet6 addr: fe80::fc16:3eff:fe19:e4d1/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:203 errors:0 dropped:0 overruns:0 frame:0
TX packets:171 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:20778 (20.2 KiB) TX bytes:20095 (19.6 KiB)
tap431b34bc-17 Link encap:Ethernet HWaddr FE:16:3E:EE:7C:BE
inet6 addr: fe80::fc16:3eff:feee:7cbe/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:208 errors:0 dropped:0 overruns:0 frame:0
TX packets:171 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:20993 (20.5 KiB) TX bytes:20263 (19.7 KiB)

```

下面是 Server2 服务器里 `ovs-vsctl show` 看到的网络配置信息:

```

o0356736a-53b2-4d5a-ad43-2994f3b728ee
  Bridge br-tun
    Port patch-int
      Interface patch-int
        type: patch
        options: {peer=patch-tun}

```

```
Port br-tun
  Interface br-tun
    type: internal
Port "Vxlan-2"
  Interface "Vxlan-2"
    type: Vxlan
    options: {in_key=flow, local_IP="192.168.100.3",
out_key=flow, remote_IP="192.168.100.2"}
Bridge br-int
  Port "qvo0da6db3e-d8"
    Tag: 2
    Interface "qvo0da6db3e-d8"
  Port patch-tun
    Interface patch-tun
      type: patch
      options: {peer=patch-int}
  Port br-int
    Interface br-int
      type: internal
  Port "qvo431b34bc-17"
    Tag: 2
    Interface "qvo431b34bc-17"
ovs_version: "1.11.0"
```

## 2. 同租户同物理机 VM 通信

VXLAN 隔离环境下通信流程抓包数据如下:

一个租户下的同一物理服务器内的两个 VM 内网通信, 内网网关 10.0.1.1, 从 MAC 地址来看 VM1 的网卡对应 tap0da6db3e-d8, VM2 的网卡对应 tap431b34bc-17; VM1 (10.0.1.5) Ping VM2 (10.0.1.6) 报文转发流程, 在 VM1 内执行 Ping 10.0.1.6 命令在 Server2 内路径各点处抓包:

```
(1) TCPdump -i tap0da6db3e-d8 -vv
10.0.1.5 > 10.0.1.6: ICMP echo request, id 3842, seq 71, length 64 08:30:40.
620212 IP (tos 0x0, ttl 64, id 61104, offset 0, flags [none], proto ICMP (1),
length 84)
10.0.1.6 > 10.0.1.5: ICMP echo reply, id 3842, seq 71, length 64
(2) TCPdump -i qbr0da6db3e-d8 -v
10.0.1.5 > 10.0.1.6: ICMP echo request, id 3842, seq 140, length 64
08:31:49.625600 IP (tos 0x0, ttl 64, id 61173, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.6 > 10.0.1.5: ICMP echo reply, id 3842, seq 140, length 64
(3) TCPdump -i qvb0da6db3e-d8 -v
```

```

10.0.1.5 > 10.0.1.6: ICMP echo request, id 4866, seq 2, length 64
11:21:52.496739 IP (tos 0x0, ttl 64, id 2791, offset 0, flags [none], proto
ICMP (1), length 84)
10.0.1.6 > 10.0.1.5: ICMP echo reply, id 4866, seq 2, length 64
(4) TCPdump -i qvo0da6db3e-d8 -v
10.0.1.5 > 10.0.1.6: ICMP echo request, id 3842, seq 180, length 64
08:32:29.629276 IP (tos 0x0, ttl 64, id 61213, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.6 > 10.0.1.5: ICMP echo reply, id 3842, seq 180, length 64
(5) TCPdump -i br-int -v
TCPdump: WARNING: br-int: no IPv4 address assigned
TCPdump: listening on br-int, link-type EN10MB (Ethernet), capture size
65535 bytes
(6) TCPdump -i tap431b34bc-17 -v
10.0.1.5 > 10.0.1.6: ICMP echo request, id 3842, seq 1149, length 64
08:48:38.721389 IP (tos 0x0, ttl 64, id 62182, offset 0, flags [none], proto
ICMP (1), length 84)
10.0.1.6 > 10.0.1.5: ICMP echo reply, id 3842, seq 1149, length 64
(7) TCPdump -i br-tun -v
TCPdump: WARNING: br-tun: no IPv4 address assigned
TCPdump: listening on br-tun, link-type EN10MB (Ethernet), capture size
65535 bytes
(8) VM 的报文没有出物理服务器
TCPdump -i eth0 -v| grep icmp
TCPdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535
bytes
^C21 packets captured
TCPdump -i eth1 -v| grep icmp
TCPdump: listening on eth1, link-type EN10MB (Ethernet), capture size 65535
bytes
^C0 packets captured

```

### 3. 同租户不同物理机 VM 通信

一个租户下的不同物理服务器内的两个 VM 内网通信，内网网关为 10.0.1.1，从 MAC 地址来看 VM1 的网卡对应 tap0da6db3e-d8，VM1 位于 Server2 内；VM3 的网卡对应 tap87816d39-b0，VM3 位于 Server1 内；VM1 (10.0.1.5) Ping VM3 (10.0.1.7) 报文转发流程，在 VM1 内执行 Ping 10.0.1.7 命令在 Server2 和 Server1 内路径各点处抓包：

```
(1) Server2 内 TCPdump -i tap0da6db3e-d8 -v
```

```
10.0.1.5 > 10.0.1.7: ICMP echo request, id 4610, seq 32, length 64
11:15:45.919544 IP (tos 0x0, ttl 64, id 11125, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 4610, seq 32, length 64
(2) Server2内TCPdump -i qbr0da6db3e-d8 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 4610, seq 163, length 64
11:17:56.931662 IP (tos 0x0, ttl 64, id 11256, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 4610, seq 163, length 64
(3) Server2内TCPdump -i qvb0da6db3e-d8 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 5378, seq 2, length 64
11:22:33.352315 IP (tos 0x0, ttl 64, id 11490, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 5378, seq 2, length 64
(4) Server2内TCPdump -i qvo0da6db3e-d8 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 5378, seq 28, length 64
11:22:59.354484 IP (tos 0x0, ttl 64, id 11516, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 5378, seq 28, length 64
(5) Server2内TCPdump -i eth1 -v
192.168.100.3.49727 > 192.168.100.2.4789: UDP, length 106
11:23:41.357901 IP (tos 0x0, ttl 64, id 11558, offset 0, flags [DF], proto
UDP (17), length 134)
192.168.100.2.53195 > 192.168.100.3.4789: UDP, length 106
(6) Server1内TCPdump -i eth1 -v
192.168.100.3.49727 > 192.168.100.2.4789: UDP, length 106
11:24:40.103895 IP (tos 0x0, ttl 64, id 11597, offset 0, flags [DF], proto
UDP (17), length 134)
192.168.100.2.53195 > 192.168.100.3.4789: UDP, length 106
(7) Server1 TCPdump -i qvo87816d39-b0 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 5378, seq 286, length 64
11:27:37.116996 IP (tos 0x0, ttl 64, id 11774, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 5378, seq 286, length 64
(8) Server1TCPdump -i qvb87816d39-b0 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 5378, seq 308, length 64
11:27:59.118632 IP (tos 0x0, ttl 64, id 11796, offset 0, flags [none],
proto ICMP (1), length 84)
(9) Server1内TCPdump -i qbr87816d39-b0 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 5378, seq 350, length 64
11:28:41.122206 IP (tos 0x0, ttl 64, id 11838, offset 0, flags [none],
proto ICMP (1), length 84)
```

```
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 5378, seq 350, length 64
(10) Server1内 TCPdump -i tap87816d39-b0 -v
10.0.1.5 > 10.0.1.7: ICMP echo request, id 5378, seq 377, length 64
11:29:08.124335 IP (tos 0x0, ttl 64, id 11865, offset 0, flags [none],
proto ICMP (1), length 84)
10.0.1.7 > 10.0.1.5: ICMP echo reply, id 5378, seq 377, length 64
```

#### 4. VM 与外网通信

一个租户下的计算节点上的 VM 进行外网通信，内网网关 10.0.1.1，从 MAC 地址来看 VM1 的网卡对应 tap0da6db3e-d8，VM1 (10.0.1.5) Ping 8.8.8.8 报文转发流程，在 VM1 内执行 Ping 8.8.8.8 命令在 Server2 和 Server1 内路径各点处抓包：

```
(1) Server2内 TCPdump -i tap0da6db3e-d8 -v
10.0.1.5 > google-public-dns-a.google.com: ICMP echo request, id 5634, seq
12, length 64 11:37:51.034347 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
proto ICMP (1), length 84)
google-public-dns-a.google.com > 10.0.1.5: ICMP echo reply, id 5634, seq
13, length 64 11:37:52.034438 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
proto ICMP (1), length 84)
(2) Server2内 TCPdump -i qbr0da6db3e-d8 -v
10.0.1.5 > google-public-dns-a.google.com: ICMP echo request, id 5634, seq
66, length 64 11:38:44.525895 IP (tos 0x0, ttl 31, id 0, offset 0, flags [none],
proto ICMP (1), length 84)
google-public-dns-a.google.com > 10.0.1.5: ICMP echo reply, id 5634, seq
66, length 64 11:38:45.039500 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
proto ICMP (1), length 84)
(3) Server2内 TCPdump -i qvb0da6db3e-d8 -v
10.0.1.5 > google-public-dns-a.google.com: ICMP echo request, id 5634, seq
106, length 64 11:39:24.532342 IP (tos 0x0, ttl 31, id 0, offset 0, flags [none],
proto ICMP (1), length 84)
google-public-dns-a.google.com > 10.0.1.5: ICMP echo reply, id 5634, seq
106, length 64 11:39:25.043053 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
proto ICMP (1), length 84)
(4) Server2内 TCPdump -i qvo0da6db3e-d8 -v
10.0.1.5 > google-public-dns-a.google.com: ICMP echo request, id 5634,
seq 128, length 64 11:39:47.044803 IP (tos 0x0, ttl 64, id 0, offset 0, flags
[DF], proto ICMP (1), length 84)
google-public-dns-a.google.com > 10.0.1.5: ICMP echo reply, id 5634, seq
130, length 64 11:39:49.044982 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
```

```
proto ICMP (1), length 84)
  (5) Server2内 TCPdump -i eth1 -v
    192.168.100.3.53949 > 192.168.100.2.4789: UDP, length 106 11:40:27.535961
IP (tos 0x0, ttl 64, id 19337, offset 0, flags [DF], proto UDP (17), length
134)
  192.168.100.2.40110 > 192.168.100.3.4789: UDP, length 106
  (6) Server1内 TCPdump -i eth1 -v
    192.168.100.3.53949 > 192.168.100.2.4789: UDP, length 106 11:43:45.289186
IP (tos 0x0, ttl 64, id 19406, offset 0, flags [DF], proto UDP (17), length
134)
  192.168.100.2.40110 > 192.168.100.3.4789: UDP, length 106
  (7) Server2内 TCPdump -i eth0 -v|grep ICMP
    192.168.1.88 > google-public-dns-a.google.com: ICMP echo request, id 5634,
seq 410, length 64 11:44:48.292681 IP (tos 0x0, ttl 32, id 0, offset 0, flags
[none], proto ICMP (1), length 84)
    google-public-dns-a.google.com > 192.168.1.88: ICMP echo reply, id 5634,
seq 410, length 64
```

这里对 VXLAN 环境下的 ARP 封装 VXLAN 报文格式和同一租户的 VM 之间通信及 VM 访问外网的数据流进行了抓取,通过这些数据可以详细看到报文是如何被封装、转发和解封装的,加深对 VXLAN 环境下 VM 通信流程的理解。

# 参考文献

- [1] 张卫峰.深度解析 SDN: 利益.战略.技术实践.北京: 电子工业出版社, 2014.
- [2] Homas D. Nadeau Ken Gray.软件定义网络: SDN 与 OpenFlow 解析.毕军译.北京: 人民邮电出版社, 2014.
- [3] Gary.Wright, W.Richard Stevens.TCP/IP 卷一. 范建华, 胥光辉, 张涛等译. 北京: 机械工业出版社, 2000.
- [4] Robert Love.Linux 内核设计与实现.3 版. 陈莉君, 康华译.北京: 机械工业出版社, 2011.
- [5] Jonahan Corbet.LINUX 设备驱动程序.3 版.魏永明, 耿岳, 钟书毅等译.北京: 中国电力出版社.2006;
- [6] John T.Moy. OSPF 协议剖析.皮学贤, 李铭译.北京: 中国电力出版社, 2002.
- [7] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon ,Martin Casado.Extending Networking into the Virtualization Layer[J]. Nicira, Inc. (Palo Alto, CA, US).
- [8] Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado.VirtualSwitching in an Era of Advanced Edges[J]. Nicira Networks, Palo Alto, CA, 650 473-9777.
- [9] Guohan Lu, Chuanxiong Guo, Yulong Li, Zhiqiang Zhou Tong Yuan, Haitao Wu, Yongqiang Xiong, Rui Gao, Yongguang Zhang.ServerSwitch: A Programmable and High Performance Platform for Data Center Networks[J]. Microsoft Research Asia, Beijing, China Tsinghua University.
- [10] Amin Tootoonchian, Yashar Ganjali.HyperFlow A distributed control plane for OpenFlow [J]. University of Toronto.
- [11] <http://www.OpenStack.org/ation/OpenStack-user-committee-update-and-survey-results>
- [12] <http://www.OpenStack.org/>
- [13] <https://www.opennetworking.org/>
- [14] <http://www.DPDK.org/>
- [15] <http://Open vSwitch.org/>
- [16] [http://portal.etsi.org/NFV/NFV\\_White\\_Paper.pdf](http://portal.etsi.org/NFV/NFV_White_Paper.pdf)
- [17] <http://deeppacketinspection.net/>
- [18] <http://www.opencloudblog.com>
- [19] [http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexne5000/sw/operations/n5k\\_vpc\\_ops.html](http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexne5000/sw/operations/n5k_vpc_ops.html)

- [20] <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf>
- [21] <http://en.wikIPedia.org/wiki/EtherType>
- [22] [http://en.wikIPedia.org/wiki/List\\_of\\_IP\\_protocol\\_numbers](http://en.wikIPedia.org/wiki/List_of_IP_protocol_numbers)
- [23] Tom Edsall, Andy Fingerhut, Terry Lam, Rong Pan, George VargheseFlame: Efficient and Robust Hardware Load Balancing for Data Center Routers. University of California, San Diego, Cisco Systems.
- [24] Kevin Jackson, Cody Bunch. OpenStack 云计算实战手册（第二版）黄凯，杜玉杰等.北京：人们邮电出版社 .2014.

# 云计算网络珠玑

## 第1部分 网络基本原理

### 第1章 TCP/IP网络技术

- 1.1 信息网络
- 1.2 以太网技术
- 1.3 网络传输设备
- 1.4 MAC和VLAN
- 1.5 MAC-in-MAC
- 1.6 STP和Trill
- 1.7 IP技术
- 1.8 DNS和DHCP
- 1.9 ICMP报文
- 1.10 ARP和RARP
- 1.11 路由协议
- 1.12 NAT技术
- 1.13 隧道技术
- 1.14 MPLS和VPLS
- 1.15 QoS功能
- 1.16 网络安全和监控
- 1.17 LB、CDN和DPI
- 1.18 LISP和LLDP
- 1.19 网络架构

### 第2章 以太网交换机

- 2.1 交换机转发流程
- 2.2 交换机端口处理
- 2.3 交换机二层转发
- 2.4 交换机三层转发
- 2.5 交换机ACL和QoS
- 2.6 交换机的虚拟化支持
- 2.7 交换机的CPU

### 第3章 Linux网络基础

- 3.1 网卡和数据包的收发

### 3.2 TUN/TAP

### 3.3 Linux Bridge和VLAN

### 3.4 TCP/IP协议栈

### 3.5 IPtables

### 3.6 QoS模块

### 3.7 Dnsmasq

### 第4章 SDN网络架构

#### 4.1 什么是SDN

#### 4.2 OpenFlow与Open vSwitch

#### 4.3 能为SDN做什么

## 第2部分 云计算及OpenStack的网络

### 第5章 OpenStack的网络

#### 5.1 云计算及OpenStack

#### 5.2 OpenStack的网络介绍

#### 5.3 Neutron底层网络原理

#### 5.4 Neutron主要功能

#### 5.5 VXLAN隔离环境通信实例详解

#### 5.6 Neutron网络高级话题讨论

### 第6章 Neutron网络发展趋势

#### 6.1 SDN的结合

#### 6.2 硬件网络设备解决性能问题

#### 6.3 安全和监控

#### 6.4 虚拟网络中的路由协议

#### 6.5 IaaS上的商业模式

#### 6.6 云计算时代的终结

## 附录

### 附录A Open vSwitch基本命令

### 附录B 深入理解OpenStack 云计算VLANManager 网络流的六种场景

### 附录C RDO配置文件网络部分——VLAN隔离

### 附录D VXLAN通信抓包实例



博文视点Broadview



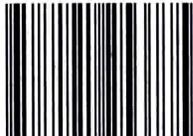
@博文视点Broadview



策划编辑：董英  
责任编辑：徐津平  
封面设计：李玲

上架建议：计算机/云计算

ISBN 978-7-121-25377-5



9 787121 253775 >

定价：69.00元