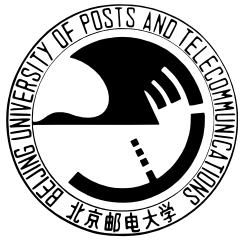
密级：　　保密期限：

北京郵電大學

硕士学位论文



题目：**xxxxxxxxxxxxxxx**

学　　号：**xxxxxxxxx**

姓　　名：**xxxx**

专　　业：信息与通信工程

导　　师：**xxxxx**

学　　院：网络技术研究院

2015 年 12 月 26 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

xxxxxxxxxxxxxxx

摘 要

中、英文摘要位于声明的次页，摘要应简明表达学位论文的内容要点，体现研究工作的核心思想。重点说明本项科研的目的和意义、研究方法、研究成果、结论，注意突出具有创新性的成果和新见解的部分。

关键词是为文献标引工作而从论文中选取出来的、用以表示全文主题内容信息的术语。关键词排列在摘要内容的左下方，具体关键词之间以均匀间隔分开排列，无需其它符号。

关键词：TEX LATEX xeCJK 模板 排版 论文

EXAMPLE OF BUPT GRADUATE THESIS LaTeX 2ε TEMPLATE

ABSTRACT

The Chinese and English abstract should appear after the declaration page. The abstract should present the core of the research work, especially the purpose and importance of the research, the method adopted, the results, and the conclusion.

Key words are terms selected for documentation indexing, which should present the main contributions of the thesis. Key words are aligned at the bottom left side of the abstract content. Key words should be seperated by spaces but not any other symbols.

KEY WORDS: TeX LaTeX xeCJK template typesetting thesis

目 录

第一章 绪论

1.1 研究背景及意义

随着互联网的发展，云数据中心的规模不断扩大，业务流量不断变化，如何为租户提供可编程的云数据中心网络，如何对云数据中心的流量进行有效的控制，提高带宽的利用率，降低成本，成为目前急需解决的问题。

软件定义网络（Software Defined Network，SDN）作为一种新兴的可编程网络架构，有动态配置、可编程及快速响应的特点。其核心思想是将网络控制平面与数据转发平面分离，实现控制平面对数据平面的全局集中化控制；同时对外提供开放的可编程接口，为网络提供可编程能力。控制权的迁移使得底层构架能够抽象出来，各种应用和网络服务因此能将网络当作一个逻辑或虚拟实体，不再依赖于底层网络设备[1]，使得网络配置的自动化程度得到极大提高。通过应用 SDN，除了网络的设计和操作变得简化，网络设备也得到简化，这些设备无需理解或处理成千上万的协议，只需要接受 SDN 控制器的指令即可。利用集中控制，网络管理员可以实时改变网络的行为，并且在几小时或几天内就可以部署新的应用和网络服务。

网络虚拟化[2] 是一种将底层网络中的硬件以及配套的软件资源进行整合，形成统一管理实体的技术，通过虚拟网络资源到物理网络资源的映射，使得多个逻辑虚拟网络共享底层物理网络基础设施, 为用户提供差异化服务。网络虚拟化技术是当今网络革新的重要技术之一。从概念上，网络虚拟化与 SDN 是互相独立的，但随着近几年网络技术的发展与融合，二者之间的联系变得越来越紧密，SDN 的技术相关专题常会提及网络虚拟化技术，网络虚拟化问题的研究也时常会运用到 SDN 的概念，可见基于 SDN 的网络虚拟化技术已经成为网络技术研究领域的一个专门课题。

OpenStack[3] 是由 Rackspace 和美国国家航空航天局（NASA）合作研发的用于搭建 Iaas 平台的云计算管理软件。旨在为公共及私有云的建设与管理提供软件的开源项目，主要提供计算、存储、网络服务。OpenStack 支持几乎所有类型的云环境，项目目标是提供实施简单、可大规模扩展、丰富、标准统一的云计算管理平台。OpenStack 通过各种互补的服务提供了基础设施即服务（Infrastructure as a Service，IaaS）的解决方案，每个服务提供 API 以进行集成[4]。

在现有 OpenStack 云平台中，租户网络的创建与隔离仅限于服务器内部，Open-

Stack 无法进行物理服务器之间数据中心网络的管控，跨服务器的通信通过隧道技术实现，该模式无法满足用户多样性的需求，同时无法实现云数据中心带宽资源的有效利用, 时常会出现有些链路阻塞严重，而有些链路则处于空闲状态。

本文以此作为出发点，提出了一种多租户虚拟网络定制化管理方案，运用虚拟化技术，为租户提供相互隔离的虚拟 SDN 网络（virtual Software Defined Network，vSDN），完成云数据中心物理资源有效利用的同时，SDN 网络由租户自有的控制器实现集中管控，租户可以实时监测当前的流量状况并根据流量状况自定义转发路径，实现对网络的灵活管控，在不降低云平台性能的前提下，实现了 OpenStack 云平台中租户网络的定制化操作，既提高了安全性，又可以根据当下链路的剩余带宽，进行链路的定制化，提高带宽的利用率。对于租户本身而言，真正实现了租户对全局网络的可控性。对于运营商来说，对物理网络的集中控制可以大大较小网络配置的繁琐性，可以对网络故障实现快速的排查，提高可扩展性。

1.2 主要研究内容及创新点

1.2.1 研究内容

本课题为实现基于 SDN 的云平台多租户虚拟网络定制化方案，将 SDN 集成到云数据中心，通过虚拟化技术，为租户提供了可编程的 vSDN 网络，vSDN 网络由租户自由的 SDN 控制器进行集中管控，租户可以根据当前的链路状况，进行定制化流表的下发，真正实现对网络的灵活定制化操作。主要包括以下三个方面的研究内容：

第一、实现虚拟网络的创建：通过虚拟化技术，实现跨物理服务器，数据中心网络的虚拟化，虚拟网络支持 SDN 模式，由租户自己的 SDN 控制器实现集中管控，本文选用网络虚拟化平台（OpenVirteX，OVX）[5] 实现虚拟网络的创建。对于底层物理设备来说，OVX 是一个控制器；对于租户控制器来说，OVX 可以看做是 OpenFlow 交换机的集合，租户控制器看到的只是一张虚拟网络。OVX 最大的优势是紧密结合了 SDN，可以发挥 SDN 的控制与转发分离的强大优势，将创建的虚拟网络指定 SDN 控制器，从而可以运用 SDN 的优势，实现对租户虚拟网络的管控。该部分主要对 OVX 进行二次开发，为其封装手动创建虚拟网络的 API 接口，完成与 OpenStack 的集成，实现 OVX 对 OpenStack 虚拟机的集中管控。

第二、完成虚拟网络的集中控制：针对租户创建的 vSDN 网络，由租户自己的 SDN 控制器进行集中管控，本文为用户提供了集成 Ryu 控[6] 制器的定制化镜像。本文为该控制器开发了剩余带宽、已用带宽、链路时延的测量以及定制化流表的下发

功能。租户可以进行网络的链路带宽、时延，选取一条最优的数据传输链路，下发定制化流表，从而实现数据传输的可控性。提高带宽利用率的同时，数据传输速率得到了很大的改善。

第三、云平台前端可视化操作：本文为实现租户的便捷操作，在 OpenStack 云平台上开放了可视化界面操作。其一是用来显示物理数据中心网络，并在数据中心网络上进行租户虚拟网络的创建操作，以及数据中心网络链路时延的显示工作。其二用于实现租户虚拟网络的拓扑展示，虚拟网络带宽、时延的展示，以及定制化链路的选取和流表的下发操作。使用户切身感受到操作的便捷性、对网络可控性。

1.2.2 创新点

针对现有云平台数据中心的不足，本文主要实现了 SDN 与云平台的结合，将 SDN 的集中控制、网络的虚拟化集成到云平台中，实现了租户网络隔离的同时，租户可以对网络实现自由控制。相应的创新点主要涉及以下几个方面：

1. 对于云平台运营商来说，物理网络的集中控制，可以有效的减小网络配置的繁琐性，为数据中心规模的伸缩性提供了便利，对网络的状况可以实现实时监控，同时大大加速了网络的故障修复速度，通过控制侧的日志输出，以及错误模拟，实现快速故障排除。

2. 将虚拟化技术集成到云数据中心，为租户提供相互隔离的 vSDN 网络，该网络由租户自己的控制器实现集中控制。集中控制的实现，可以让租户指定灵活的数据包转发路径，以及灵活的转发策略。对于自身网络的变化也可以实现即时的应对措施。从而对现有的网络资源进行最有效的利用，减少了网络资源的浪费。租户之间的隔离性，保证了租户数据传输的安全性。

3. 基于包对技术，完成了虚拟网络的链路带宽测量，实现了细粒度的剩余带宽测量，精确度远远高于以前的模式；基于数据包统计的方法，实现了粗粒度的已用带宽的测量；完成了 SDN 架构下的时延测量。租户根据带宽、时延实现定制化链路的选取，可以有效的利用剩余带宽，将降低数据传输时延，大大加快了数据中心网络的传输速度。

综上所述，基于 SDN 的云平台多租户虚拟网络定制化方案，对提高云数据中心的带宽利用率，为租户提供可编程的云网络是非常必要的，有助于实现租户网络的定制化操作，同时为运营商进行云数据中心的维护和配置提供了便利。

1.3 研究生期间主要工作

在攻读硕士研究生期间，本文作者认真学习专业相关课程，积极参与实验室的科研工作。认真学习了网络智能研究中心要求的基本课程并取得了满意的成绩，先后参与了实验室多个科研项目。主要的研究方向为智慧云平台的开发工作，在老师和师兄师姐的指导下，对研究课题进行了深入的探索，通过阅读书籍和学术论文夯实理论基础，并进行系统设计、实现和功能测试。主要科研经历和学术成果如下：

参与了国家重点实验室自主课题仪器设备研制类——面向无线业务的网络智能承载验证环境，开发智慧云平台，为租户提供自主可控的 SDN 云网络；在国家重点基础研究发展计划（973 计划）课题——智慧服务机理与理论中，负责新一代网络架构的调研和撰写工作。

以第一作者的身份发表了两篇 EI 检索论文，具体为：通过对多租户虚拟网络定制化机制的研究，发表了学术论文《MVNC: A SDN-based Multi-tenant Virtual Network Customization Mechanism in Cloud Data Center》，在论文中提出了一种 MVNC 的多租户虚拟网络定制化机制，论文已发表于 IEEE。通过对粒子群算法（Particle Swarm Optimization，PSO）算法及网路虚拟化的研究，撰写了学术论文《A PSO-Based Virtual Network Customization for Multi-tenant Cloud Services》，在论文中，提出了一种基于 PSO 算法的网络虚拟化方案，实验证明相对于传统的基于最短路径的虚拟化算法，性能得到很大提升。同时，在此基础上，完成了多租户云服务中虚拟网络的定制化。论文已被录用，即将发表于 ACM。

1.4 论文组织结构

论文总共包括五章，具体内容组织如下：

第一章主要概述了研究背景、意义以及研究的主要内容，包括 SDN、OpenStack 的背景介绍，本文的主要研究内容以及基于 SDN 云平台多租户虚拟网络定制化的意义、创新点。

第二章主要介绍了 SDN、网络虚拟化及 OpenStack 的相关技术。首先介绍了 OpenFlow 协议,OpenFlow 交换机的组成以及 OpenFlow 流表的字段和匹配过程。对于网络虚拟化技术, 由于本文需要实现 SDN 模式下的网络虚拟化, 所以仅对适用于本文的 SDN 模式下的网络虚拟化技术做了简单的概括。随后阐述了 OpenStack 的模块构成, 而对于本文用到的 Neutron 模块, 进行了详细的架构剖析。最后对本章内容做了

总结。

第三章

第四章

第五章

1.5 本章小结

本章对课题的研究背景进行了描述，阐述了 SDN、OpenStack、网络虚拟化等相关技术，并在此基础上确定了研究内容，对研究生期间的主要工作做了简单的概括，最后对论文的组织结构进行了介绍。

第二章　相关技术概述

SDN 作为新一代的网络架构，具有动态配置、可编程及快速响应的特点，通过解耦控制平面与数据转发平面，实现了对网络底层基础设施的抽象。OpenStack 是当下最流行的开源云平台，为用户提供计算、存储、网络服务。本章对 SDN、OpenFlow、OpenStack 等相关技术做了详细的介绍。

2.1　SDN、OpenFlow 概述

2.1.1　SDN 简介

SDN 是一种时下热门的网络架构，在这种网络架构中，网络的控制与转发解耦，具有很强的可编程行和可扩展性。传统模式下，网络的控制权与网络设备紧密捆绑，现有模式下，控制权的迁移使得底层构架能够抽象出来，各种应用和网络服务因此能将网络当作一个逻辑或虚拟实体。通过应用 SDN，网络的设计和操作变得简化，网络设备也得到简化，这些设备无需理解或处理成千上万的协议，只需要接受 SDN 控制器的指令即可。利用控制集中，网络管理员可以实时改变网络的行为，并且在几小时或几天内就可以部署新的应用和网络服务。

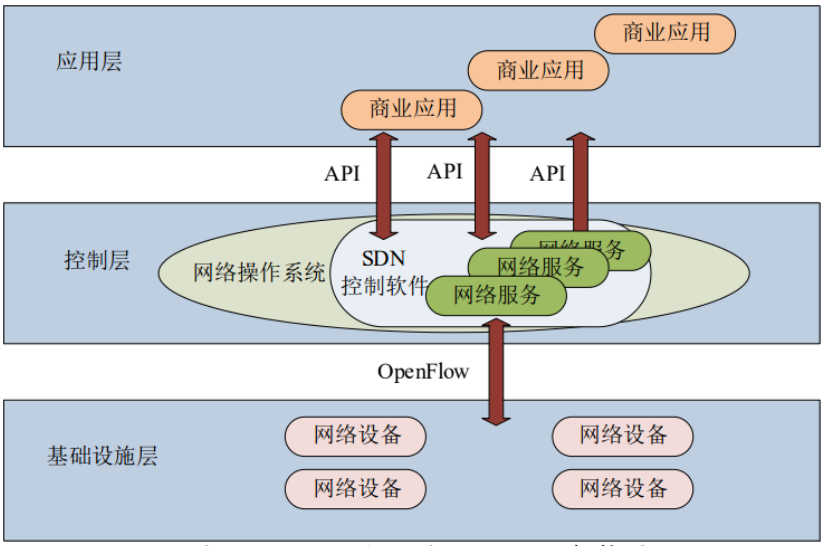


图 2-1　ONF 定义的 SDN 三层架构图

传统网络设备紧耦合的构架在 SDN 体系中被拆分成应用、控制、转发三层分离

的，可编程的开放构架。控制功能被转移到服务器（软件），上层应用、底层转发设施被抽象成多个逻辑实体。图2-1为 ONF 定义的 SDN 三层架构示意图。

应用层为网络各种应用需求，如移动视频、云存储、企业应用商店、桌面云、物联网等，通过北向接口灵活、可编程地调用控制层提供的网络抽象模型与业务功能。北向接口因为涉及业务较多，开放的标准化过程还处于研究阶段。

控制层为整个 SDN 构架的核心，也称为网络操作系统，可实现网络拓扑的集中控制和设备管理，进行流表的控制和下发。其主要功能包括路由优化、网络虚拟化、质量监控、拓扑管理、设备管理、接口适配等。

基础设施层包括标准化的网络设备和虚拟的网络设备，负责多级流表处理和高性能的数据转发，并作为硬件资源池，为控制层提供按需的网络拓扑、数据处理和数据转发。目前主流的网络设备和芯片产商已经提供了支持 OpenFlow 的网络设备。

南向接口定义了控制层与数据转发层（基础设施层）之间的交互协议，通过将转发过程抽象为流表，控制器可直接控制流表、屏蔽硬件，实现了网络虚拟化。物理硬件被淡化为资源池，可按需进行灵活分配和相互隔离。目前主流的控制层与转发层之间交互的协议为 OpenFlow 协议。

2.1.2 OpenFlow 协议

OpenFlow 协议是 OpenFlow 交换机和控制器之间交互信息的标准，也是 OpenFlow 交换机和控制器之间的接口标准。OpenFlow 协议主要支持三种类型的消息：Controller-to-Switch 消息，Asynchronous 消息和 Symmetric 消息。每种类型消息都有多个子类型。Controller-to-Switch 消息由控制器发起直接管理和监视交换机状态，可能需要交换机做出响应；Asynchronous 消息由交换机发起用于更新交换机的状态和控制器的网络事件；Symmetric 消息可由控制器和交换机任一方发起。具体协议消息的类型及描述如下：

1. Controller-to-Switch 消息

   - Features：在建立传输层安全会话（Transport Layer Security Session）的时候，控制器发送 feature 请求消息给交换机，交换机需要应答自身支持的功能。一般在 OpenFlow 通道建立时使用。
   - Configuration：控制器设置或查询交换机上的配置信息。交换机仅需要应答查询消息。

- Modify-state：控制器管理交换机流表项和端口状态等。
- Read-state：控制器发送该消息来管理交换机状态。主要用于增加、删除或改变组/流表项和设置交换机端口属性。
- Packet-out：控制器通过交换机指定端口发出网包。控制器按指定交换机端口发送数据包, 并且转发经由 Packet-in 消息接收到的包。Packet-out 消息必须包含完整包或一个指向存储在交换机中的完整包的缓冲区 ID。该消息包含一系列按指定顺序执行的行为, 若为空则丢弃该包。
- Barrier：控制器确保消息依赖满足，或接收完成操作的通知

2. Asynchronous 消息

- Packet-in：交换机收到一个网包，在流表中没有匹配项，则发送 Packet-in 消息给控制器。如果交换机缓存足够多，网包被临时放在缓存中，网包的部分内容（默认 128 字节）和在交换机缓存中的的序号也一同发给控制器；如果交换机缓存不足以存储网包，则将整个网包作为消息的附带内容发给控制器。
- Flow-removed：交换机中的流表项因为超时或修改等原因被删除掉，会触发 Flow-removed 消息，通知控制器删除流表。
- Port-status：交换机端口状态发生变化时（例如 down 掉），触发 Port-status 消息。
- Error：交换机发生问题时触发消息。

3. Symmetric 消息

- Hello：交换机和控制器用来建立连接。
- Echo：交换机和控制器均可以向对方发出 Echo 消息，接收者则需要回复 Echo reply。该消息用来测量延迟、带宽、以及是否连接正常等。
- Vendor：交换机提供额外的附加信息功能。为未来版本预留。

OpenFlow 协议的主要交互过程如下所示：

1. 控制器与交换机的连接建立: 通过安全通道建立连接，所有流量都不经过交换机流表检查。当连接建立起来后，两边必须先发送 OFPT_HELLO 消息给对方，该

消息携带支持的最高协议版本号，接收方将采用双方都支持的最低协议版本进行通信。一旦发现共同支持的协议版本，则连接建立，否则发送 OFPT_ERROR 消息，描述失败原因，并中断连接。

2. 控制器与交换机的连接中断：当连接发生异常时，交换机应尝试连接备份的控制器。当多次尝试均失败后，交换机将进入紧急模式，并重置所有的 TCP 连接。此时，所有包将匹配指定的紧急模式表项，其他所有正常表项将从流表中删除。此外，当交换机刚启动时，默认进入紧急模式。

3. 连接加密：安全通道采用传输层安全（Transport Layer Security，TLS）连接加密。当交换机启动时，尝试连接到控制器的 TCP 6633 端口。双方通过交换证书进行认证。因此，每个交换机至少需配置两个证书，一个是用来认证控制器，一个用来向控制器发出认证。

4. 流表项修改：该交互是核心交互过程，通过控制器下发的流表项修改指令完成。每条指令可能触发一系列 OpenFlow 协议消息，主要包括流表的增加、修改和删除。

5. 流表项移除：流表项的移出包括控制器主动模式和被动模式，控制器被动模式下，定时器计时结束：每个表项均有一个 idle_timeout 定时器和一个 hard_timeout 定时器（两者的计量单位都是秒），前者计算的是没有 Flow 匹配发生的时间，而后者则计算的是表项在流表中的总时间。一旦到达时间期限，交换机将自动删除该表项，同时发出一个流删除的消息；控制器主动模式下，控制器通过下发 DELETE_STRICT、DELETE 等指令相关的协议消息主动删除流表项

2.1.3 OpenFlow 交换机

OpenFlow 交换机负责数据转发功能，主要技术细节由流表 (flow table)、安全信道 (secure channel) 组成[7]，如图2-2所示。

OpenFlow 的转发策略主要保存在流表中，每个流表中都有很多表项 (FlowEntry), 这些表项可由外部控制器通过 OpenFlow 协议来写入、更新和删除。对于 OpenFlow v1.3, 每一个流表项都由匹配域、优先级、计数器、指令、超时以及 Cookie 组成，如表2-1所示。当一个流进入交换机后，交换机会选择匹配的流表中优先级最高的一个执行相应的指令并更新计数器、超时等项。

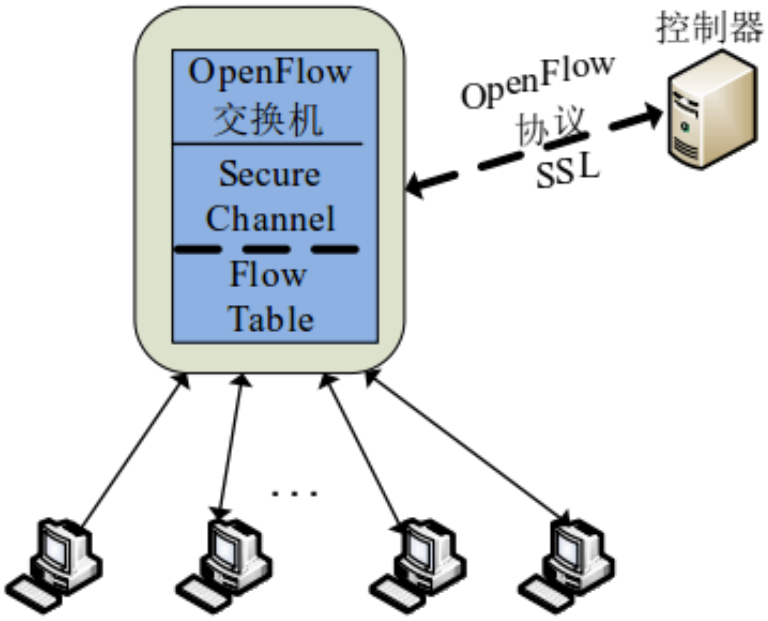


图 2-2 OpenFlow 交换机结构

表 2-1 OpenFlow 流表项结构

匹配域 (Match Field)	优先级 (Priority)	计数器 (Counters)	指令 (Instructions)	超时 (Timeouts)	Cookie

匹配域用于进行对流的匹配。包括 13 个必须支持的域 (Required)：入端口、Ethernet 目的地址、源地址、类型、IP 协议号、IPv4 源地址、目的地址、IPv6 源地址、目的地址、TCP 源地址、目的地址、UDP 源地扯、目的地址。每一个域包含一个确定的值或者任意值 (Any，表示任何值都能匹配)，更准确的匹配可以用位掩码实现，但是每个域对掩码的支持与否不同。

优先级用于在流匹配到多个表项时的选择策略，这里优先级最高的被优先匹配。

计数器用于维护每个流表、每个流、每个队列等的统计数据，例如活动表项、包查找次数、包匹配次数、发送包数、接收字节数等。指令执行于匹配到的包，每个表项包括一系列指令，这些指令可能会直接对包进行改变，也可能改变行动集，或者管道处理。其中行动集是在包被转发前按照指定顺序执行的一系列行动，管道处理是包在各流表之间进行传递时，包与流表之间的互相作用关系。

超时机制用于交换机对于超时表项的自动删除，进而提高交换机的空间利用率。包括硬超时 (指超过该值即删除，绝对时间) 和软超时 (指超过该值也没有匹配的流即删除，相对时间)。

匹配的具体过程如图2-3所示，具体流程如下。

1. 数据匹配字段从数据包中提取。用于表查找的数据包匹配字段依赖与数据包类型，这些类型通常包括各种数据包的报头字段，如以太网源地址或 IPv4 目的地址。除了通过数据包报头中进行匹配，也可以通过入口端口和元数据字段进行匹配。元数据可以用来在一个交换机的不同表里面传递信息。报文匹配字段表示报文的当前状态，如果在前一个表中使用 Apply-Actions 改变了数据包的报头，那么这些变化也会在数据包匹配字段中反映。

2. 数据包匹配字段中的值用于查找匹配的流表项。如果流表项字段具有该值，它就可以匹配包头中的所有可能的值。如果交换机支持任意的的位掩码对特定的匹配字段，这些掩码可以更精确地进行匹配。优先级最高的流表项首先被匹配上，此时与选择流表项相关的计数器也会被更新，选定流表项的指令集也被执行。如果有多个匹配的流表项具有相同的最高的优先级的，所选择的流表项被确定为未定义表项。

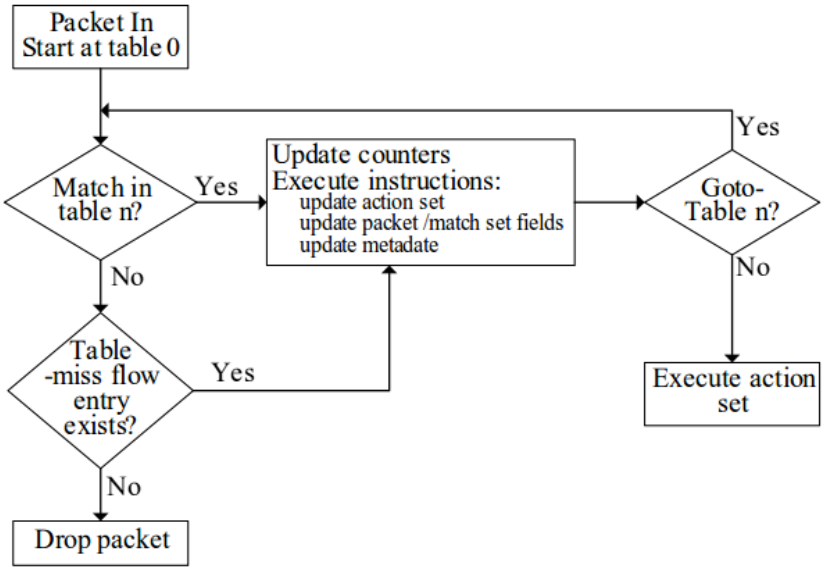


图 2-3　数据包匹配流程图

安全通道是连接 OpenFlow 交换机和控制器的接口，控制器通过这个接口，按照 OpenFlow 协议规定的格式来配置和管理 OpenFlow 交换机，同时接收交换机传来的事件信息并发送数据包等。所有的 OpenFlow 通道信息必须用 OpenFlow 协议打包，OpenFlow 通道通常用 TLS 来加密，但也能直接用 TCP 协议来实现。

目前，基于软件实现的 OpenFlow 交换机主要有两个版本[8]，都部署于 Linux 系统：基于用户空间软件 OpenFlow 交换机操作简单，便于修改，但性能较差；基于内核空间的软件 OpenFlow 交换机[9] 速度较快，同时提供了虚拟化功能，使得每个虚拟机能够通过多个虚拟网卡传输流量，但实际的修改和操作过程较复杂。另外，斯坦福大学基于 NetFPGA 实现了硬件加速的线速 OpenFlow 交换机[10]，而网络硬件厂商如 NEC、HP 等公司也已相继推出了支持 OpenFlow 标准的硬件交换机[7]。

2.1.4 OpenFlow 控制器

控制器是 OpenFlow 网络的核心部分，是整个网络的大脑，网络中所有的控制指令，数据流的转发操作都由它来完成，控制器中组件的好坏直接影响了整个网络的运行效率。

SDN 控制器对网络的控制主要是通过南向接口协议实现，包括链路发现、拓扑管理、策略制定、表项下发等，其中链路发现和拓扑管理主要是控制其利用南向接口的上行通道对底层交换设备上报信息进行统一监控和统计，而策略制定和表项下发则是控制器利用南向接口的下行通道对网络设备进行统一控制。控制器负责控制其所管辖的 OpenFlow 交换机中的流表，包括流表内容的添加、修改以及删除等基本操作。操作的具体逻辑由控制器上运行的控制程序制定。网络使用者可以根据具体需求在控制器上编写控制程序，使得 OpenFlow 网络可以提供丰富的应用，体现了 OpenFlow 网络的灵活性。在数据流处理的过程中，控制器的控制程序决定其所在网络中交换机上的流表内容。网路初始运行时，交换机上的流表为空，控制器必须具有全局网络视图，如网络的拓扑结构、网络的当前运行状态等信息，才能针对不同数据流做出正确的决策。因此，控制器的设计是 OpenFlow 网络整体功能和效率的重点之一。

SDN 北向接口是通过控制器向上层业务应用开放的接口，其目标是使得业务应用能够便利地调用底层的网络资源和能力。通过北向接口，网络业务的开发者能以软件编程的形式调用各种网络资源; 同时上层的网络资源管理系统可以通过控制器的北向接口全局把控整个网网络的资源状态，并对资源进行统一调度。因为北向接口是直接为业务应用服务的，因此其设计需要密切联系业务应用需求具有多样化的特征。同时，北向接口的设计是否合理、便捷，以便能被业务应用广泛调用，会直接影响到 SDN 控制器厂商的市场前景。

控制器负责整个 SDN 网络的集中化控制，对于把握全网置资源视图、改善网络

资源交付都具有非常重要的作用。但控制能力的集中化，也意味着控制器局的安全性和性能成为全网的瓶颈; 另外，单一的控制器也无法应对跨多个地域的 SND 网络问题，需要多个 SDN 控制器组成的分布式集群，以避免单一的控制器节点在可靠性、扩展性、性能方面的问题。

2.2 网络虚拟化技术概述

互联网不断地向前发展, 必然会出现越来越多的应用和服务。然而现有的网络结构比较僵化, 而且可扩展性很差, 随着网络的发展, 这些缺点日益突出, 但是研发和部署新型的网络体系架构比较困难, 难以更新, 而且由于各个设备生产商自身利益的考虑, 更加加剧了这种困难。在这种背景下, 网络虚拟化成为当前互联网面向未来网络体系架构过渡的一种有效解决方案, 同时网络虚拟化也是未来网络体系架构应该具备的关键特性之一。

网络虚拟化技术[2]，通过对共用的底层网络进行抽象并提供统一的可编程接口，将多个相互隔离且具有不同拓扑的虚拟网络映射到共用的基础设施上，为用户提供差异化服务。网络虚拟化允许多个租户占据相同的网络基础设施，每个租户会有一种错觉，认为可以对整个网络进行完整的处理. 传统的网络虚拟化配置和操作都比较复杂, 难以管理。原因就在于路由器和交换机变得越来越复杂，其配置操作的代价越来越大。而 SDN 的出现为网络虚拟化开启了另外一条道路，在 SDN 网络中，集中控制的优势使得对网络的管理变得灵活和高效。基于 SDN 的网络虚拟化是 SDN 思想的典型应用。本文选用 OVX 实现网络虚拟化模块，OVX 通过给每个租户提供访问一个虚拟网络拓扑和一个完整的网络头空间来达到这样的目标，前者允许租户自定义拓扑结构, 而后者保证功能性和流量隔离，即使是在不同的租户选择重叠的寻址方案。OVX 用作控制信道内的一个代理，呈现 OpenFlow 网络给租户，同时经由南向的 OpenFlow 接口控制底层的物理基础设施。通过暴露 OpenFlow 网络给租户，OVX 允许租户使用自己的网络控制器控制自有的虚拟网络资源。换句话说，OVX 基于同一底层物理网络创建多个 vSDN 网络，并且 vSDN 网络之间是相互隔离的，保证了租户虚拟网络的安全性。这样的方法会产生两个后果: 允许 OVX 呈现支持 OpenFlow 的可编程虚拟网络，租户可以使用自己的 NOS 控制虚拟网络，使 OVX 是透明的——从底层网络的角度，OVX 的表现为一个控制器，从租户的角度上看，OVX 作为网络中具有 OpenFlow 能力的交换机集合[11]。OVX 具体的架构图如图3-2所示。

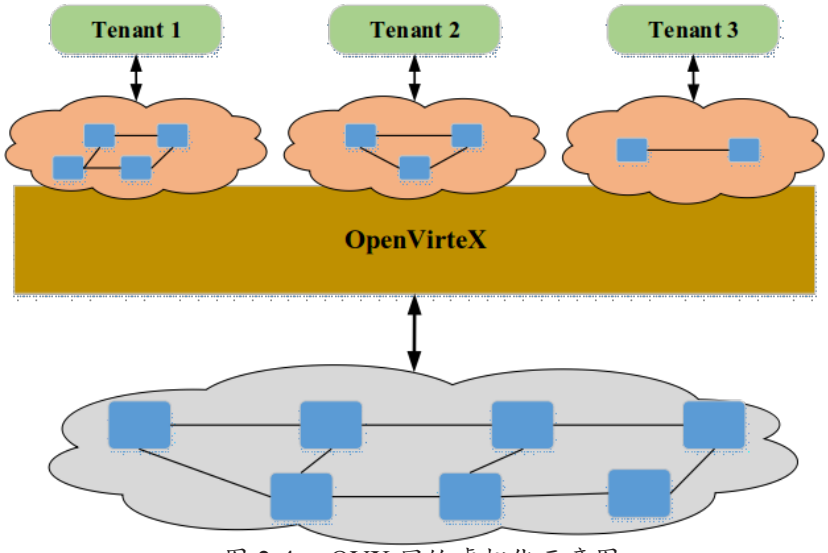


图 2-4 OVX 网络虚拟化示意图

2.3 OpenStack 概述

OpenStack 是一个由 NASA（美国国家航空航天局）和 Rackspace 合作研发并发起的，以 Apache 许可证授权的自由软件和开放源代码项目，主要为企业或者厂商提供了一种云服务的解决方案。OpenStack 既可以被企业使用，在企业局域网内给员工提供私有云；也可以被厂商使用，进行二次开发，给云用户提供通过 Internet 访问的公有云服务。OpenStack 通过一系列相关的服务提供 IaaS 解决方案。每个服务提供一个应用编程的 API，这样有利于集成。管理员可以根据自己的需求安装部分或者全部服务。

2.3.1 OpenStack 架构

OpenStack Juno 版本的构架由 9 个服务组成，分别是 Dashboard、Compute、Networking、Object Storage、Block Storage、Identity、Image Registry and Delivery Service、Telemetry 和 Orchestration。分别对应的各组件及各组件之间的协同关系如图2-5所示。

主要组件的功能介绍如下[12]。

1. Nova：计算服务是 OpenStack 的核心服务，它由 nova-compute 模块通过 libvirt、XenAPI 等管理 hypervisor，从而管理虚拟机，此外它还通过 nova-api 服务向外提供如 EC2 兼容、管控功能等的接口，通过 nova-scheduler 模块提供虚拟机调研逻辑等，这些模块间的通信全部通过消息队列完成

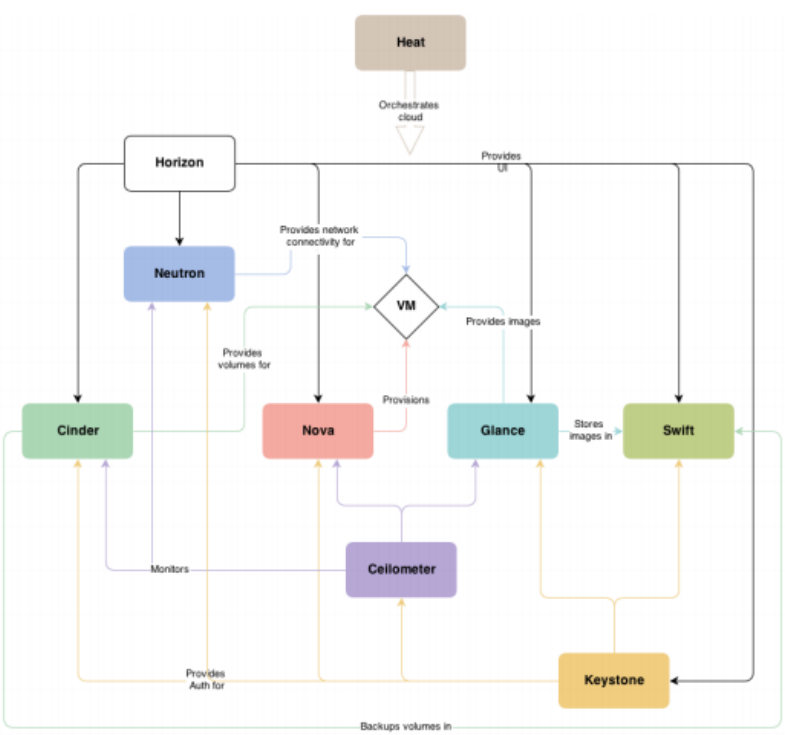


图 2-5　OpenStack 概念架构图

2. Swift：对象存储服务是 OpenStack 最早期的两个服务之一（另一个是计算服务），在 OpenStack 平台中，任何的数据都是一个对象，swift-proxy 模块对外提供如 HTTP(S)、OpenStack Object API 及与 S3 兼容的存取接口。对象的存取经 swift-proxy 接入后，还需要经三个模块进行定位，即 account、container、object，这是因为在 OpenStack 中一个对象被描述为：某个帐户下某个容器中的某个对象。

3. Neutron：经过一定时间的演变，网络管理也抽离成一个独立的服务。在 OpenStack 的网络管理流程中，通常需要经过以下几个步骤：创建一个网络；创建一个子网；启动一个虚拟机，将一块网卡对接到指定的网络上；删除虚拟机；删除网络端口；删除网络。

4. Glance：Glance 的出现是解决虚拟机镜像的管理问题，在生成一个镜像后，需要将镜像注册到系统的数据库中。当要实例化一个虚拟机时，需要将镜像分派到一台具体的实机上用来以启动虚拟机。因而 Glance 最重要的接口是镜像的注册和分派。

5. Cinder：Essex 将 nova 的卷管理 api 独立化后，Folsom 终于将卷管理服务抽离

成了 Cinder，Cinder 管理所有的块存储设备，块设备可以挂接在虚拟机的实例中，然后虚拟机里的 guest 系统可以像操作本地卷一样操作块存储设备。Cinder 需要处理的主要问题应该是接入各种块设备，如本地磁盘、LVM 或各大厂商提供的设备如 EMC、NetApp、HP、华为，还有如 Vmware 提供的虚拟块设备等。

6. KeyStone：身份服务需要进行认证凭证的验证及关于用户、角色等的信息，及所有相关的元数据，这些数据全都由 Keystone 服务管理，并提供 CRUD 的操作方法，另外这些信息可以从另一个认证服务中获取，例如使用 LDAP 做 Keystone 的后端。

7. Heat：Heat 是一套业务流程平台，旨在帮助用户更轻松地配置以 OpenStack 为基础的云体系。利用 Heat 应用程序，开发人员能够在程序中使用模板以实现资源的自动化部署。Heat 能够启动应用、创建虚拟机并自动处理整个流程。

8. Ceilometer：Ceilometer 项目创建时最初的目的是实现一个能为计费系统采集数据的框架。社区后来更新了他们的目标，新目标是希望 Ceilometer 成为 OpenStack 里数据采集（监控数据、计费数据）的唯一基础设施，采集到的数据提供给监控、计费、面板等项目使用。

9. Horizon：Horizon 是一个用以管理、控制 OpenStack 服务的 Web 控制面板，它可以管理实例、镜像、创建密匙对，对实例添加卷、操作 Swift 容器等。除此之外，用户还可以在控制面板中使用终端（console）或 VNC 直接访问实例。

2.3.2 Neutron 模块详解

本文研究的主要内容，涉及最多的是 OpenStack 的 Neutron 模块，故本节对 Neutron 模块做详细的介绍，方便大家对论文的理解。

Neutron 是 OpenStack 项目中负责提供网络服务的组件，它基于软件定义网络的思想，实现了网络虚拟化下的资源管理。在实现上充分利用了 Linux 系统上的各种网络相关的技术。一般的，OpenStack 中网络实现包括 VLAN、GRE、VXLAN 等模式，本文中选取 GRE 模式搭建 Neutron 服务，GRE 模式下的 Neutron 架构图如图2-6

在 OpenStack 中，所有网络有关的逻辑管理均在网络节点中实现，例如 DNS、DHCP 以及路由等。计算节点上只需要对所部属的虚拟机提供基本的网络功能支持，包括隔离不同租户的虚拟机和进行一些基本的安全策略管理（即安全组）。以抽象系

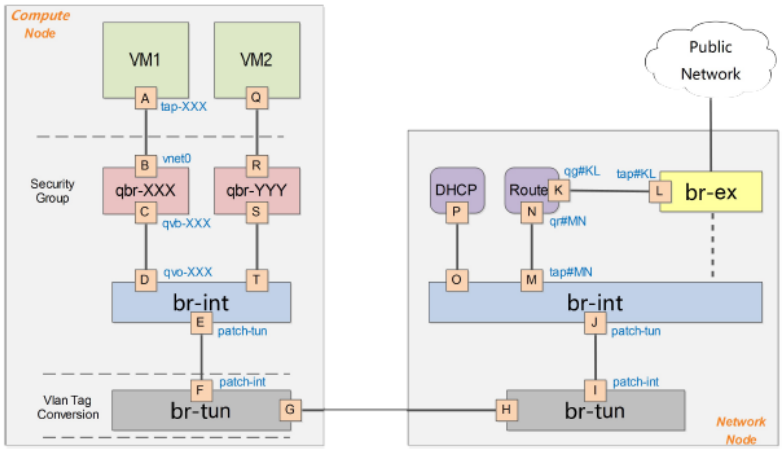


图 2-6　Neutron 组件 GRE 模式架构图

统架构图为例，计算节点上包括两台虚拟机 VM1 和 VM2，分别经过一个网桥（如 qbr-XXX）连接到 br-int 网桥上。br-int 网桥再经过 br-tun 网桥（物理网络是 GRE 实现）连接到物理主机外部网络。网络节点上配置有 DHCP、Router 服务，对于访问外网的数据包，经由 br-ex 发出。各网桥功能及具体的服务如下所示。

1. qbr：在 VM1 中，虚拟机的网卡实际上连接到了物理机的一个 TAP 设（即 A 常见名称如 tap-XXX）上，A 则进一步通过 VETH pair（A-B）连接到网桥 qbr-XXX 的端口 vnet0（端口 B）上，之后再通过 VETH pair（C-D）连到 br-int 网桥上。一般 C 的名字格式为 qvb-XXX，而 D 的名字格式为 qvo-XXX。注意它们的名称除了前缀外，后面的 id 都是一样的，表示位于同一个虚拟机网络到物理机网络的连接上。之所以 TAP 设备 A 没有直接连接到网桥 br-int 上，是因为 OpenStack 需要通过 iptables 实现安全组的功能。目前 openvswitch 并不支持应用 iptables 规则的 Tap 设备。因为 qbr 的存在主要是为了辅助 iptables 来实现 security group 功能，有时候也被称为安全网桥。

2. br-int：br-int 是一个 openvswitch 网桥。br-int 在 GRE 模式中作为一个 NORMAL 交换机使用，因此有效规则只有一条正常转发。如果两个在同一主机上的 vm 属于同一个 tenant 的（同一个 vlan tag），则它们之间的通信只需要经过 br-int 即可。

3. br-tun：br-tun 将带有 vlan tag 的 vm 跟外部通信的流量转换到对应的 gre 隧道，这上面要实现主要的转换逻辑，规则要复杂，一般通过多张表来实现。这些规则组成的整体转发逻辑如图2-7所示。

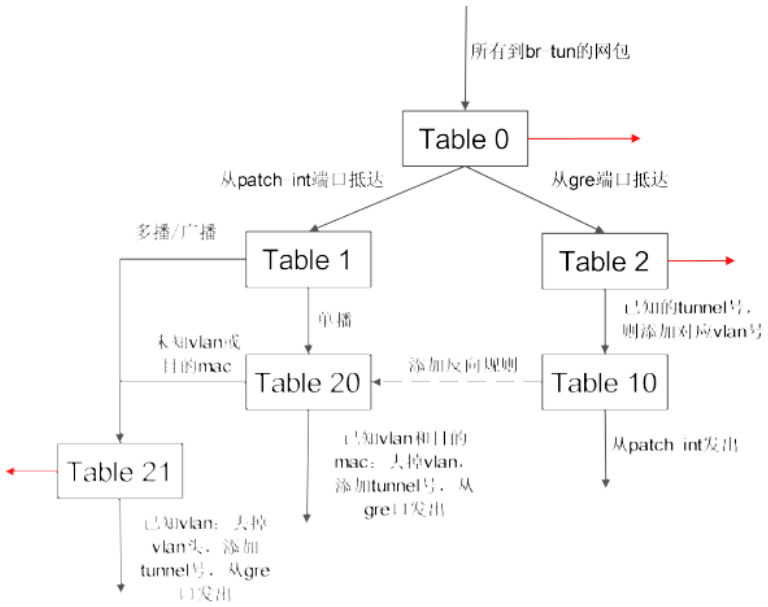


图 2-7 br-tun 网桥转发规则

4. br-ex：br-ex 直接连接到外部物理网络，一般情况下网关在物理网络中已经存在，则直接转发即可。

5. dhcp 服务：dhcp 服务是通过 dnsmasq 进程（轻量级服务器，可以提供 dns、dhcp、tftp 等服务）来实现的，该进程绑定到 dhcp 名字空间中的 br-int 的接口上。可以查看相关的进程。

6. router 服务：首先，要理解什么是 router，router 是提供跨 subnet 的互联功能的。比如用户的内部网络中主机想要访问外部互联网的地址，就需要 router 来转发（因此，所有跟外部网络的流量都必须经过 router）。目前 router 的实现是通过 iptables 进行的。同样的，router 服务也运行在自己的名字空间中。防火墙服务就是在 router 命名空间中实现。

2.4 本章小结

本章主要介绍了 SDN、网络虚拟化及 OpenStack 的相关技术。重点介绍了 OpenFlow 协议，OpenFlow 交换机的组成以及 OpenFlow 流表的字段和匹配过程。对于网络虚拟化技术，由于本文需要实现 SDN 模式下的网络虚拟化，所以仅对适用于本文的 SDN 模式下的网络虚拟化技术做了简单的概括。随后介绍了 OpenStack 的模

块构成，而对于本文用到的 Neutron 模块，进行了详细的架构剖析。本研究的系统，主要在以上各部分的基础上，进行了集成开发。后续会对整体的系统架构做详细的介绍。

第三章 基于 SDN 的云平台架构设计

基于 SDN 的云平台多租户虚拟网络定制化的研究，主要是将 SDN 集中控制和可编程的优势，集成到云数据中心，通过网络虚拟化技术，为租户提供相互隔离的 vSDN 网络，vSDN 网络由租户自由的控制器实现集中管控。物理网络的集中管控，方便运营商对数据中心网络的管理和维护；租户 vSDN 网络的集中控制，便于租户利用自己的控制期开发适合自己的上层应用。本章最该研究的主要架构进行简单的介绍。

3.1 关键技术分析

3.1.1 需求分析

基于 SDN 的云平台多租户虚拟网定制化的实现，首先解决的是物理网络资源的虚拟化，其次是在虚拟化的基础上进行虚拟网络的带宽时延测量，随后根据测量的带宽时延进行链路的定制化，最后为方便用户的操作，完成前端 Web 界面的设计和实现。

系统的整体需求如图3-2所示。

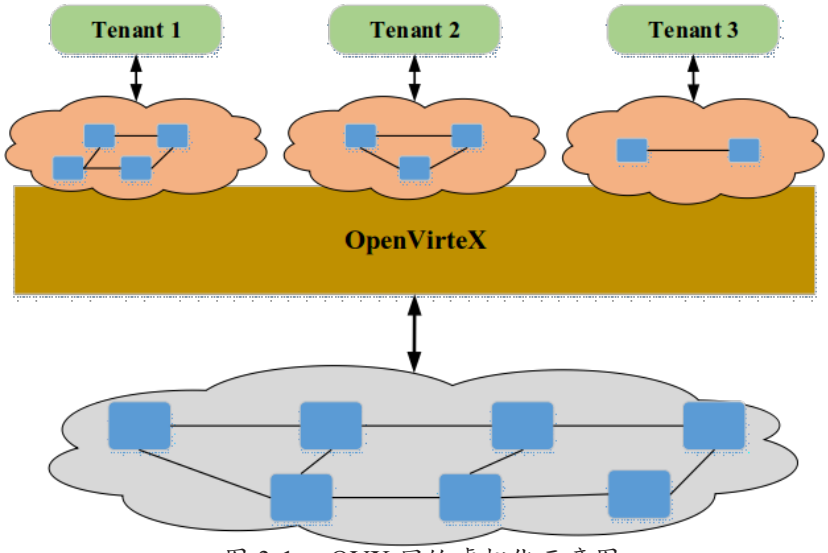


图 3-1 OVX 网络虚拟化示意图

网络的虚拟化，需要实现资源隔离的同时，还要满足虚拟网络支持 SDN 模式，

本文选用支持资源虚拟化的 SDN 控制器来完成该功能，通过对底层物理网络中下发的流表信息中，添加租户 ID 的相关信息，完成底层网络中租户的隔离工作。通过自身维护的物理网络与虚拟网络的映射关系图，为上层 SDN 交换机提供支持 OpenFlow 协议的虚拟网络，此时租户便可利用自身的 SDN 控制器完成对 vSDN 网络的集中管控。

在虚拟资源之上进行链路带宽时延等信息的测量工作，需要实现 SDN 模式下链路的带宽、时延的精确测量，由于 SDN 的集中控制和可编程的先天优势，控制器端可以看到全局的网络拓扑，与此同时，OpenFlow 协议的规范性和便捷性，方便了我们实现可用带宽、已用带宽、时延的精确测量工作，测量的数据成为用户做出决策的考量值。

链路的定制化，需要用户根据链路的时延、带宽等信息，选取便于传输的最优链路，进行定制化流表的下发工作，此时，租户便可以对自己在数据中心的虚拟机，进行远程的操控，可以自定义数据的传输路径。

前端的 Web 界面方便了用户的操作，用户可以在其 web 界面，通过简单的点击操作，完成对数据中心网络的定制化操作。包括物理、虚拟网络的拓扑获取，链路带宽、时延的前端展示，虚拟网络的创建，定制化链路的前端选取等功能。一系列的操作，需要实现租户之间的隔离工作，我们需要通过认证功能，对租户的请求进行认证。实现租户操作的隔离行。

3.1.2 关键技术

本文旨在构建一个基于 SDN 的云平台多租户虚拟网络定制化的平台，该平台一方面实现对云平台数据中心网络的集中控制，方便运营商对数据中心网路的管理和运维，加快了新业务的引入速度，运营商可以通过可控的软件部署相关的功能，该自动化部署和运维故障诊断，减少了网络的人工干预，降低了网络的运营费用，也降低了出错率。与此同时，通过虚拟化技术，基于底层的统一物理资源，虚拟出相互隔离的租户 vSDN 网络，vSDN 网络由租户自己的 SDN 控制器实现集中管控，实现了租户对数据中心网络的集中管控，租户可以为自己的控制器开发特有的北向应用，从而定制化数据中心业务需求，当然租户对数据中心虚拟网络的集中管控，可以实现租户对自有网络的监控工作，并可根据当前负载定制化链路的转发路径。为实现上述功能，需要突破以下关键技术。

1. 基于 SDN 的云平台系统搭建。将 SDN 集成到云平台数据中心，系统的搭建尤

为重要，如何在实现相应功能，满足租户需求的情况下，搭建一套性能稳定的系统成为本文的关键所在。为此，本文创新性地提出了基于 SDN 的云平台多租户虚拟网络定制化机制，在实现对数据中心网路集中控制的同时，利用虚拟化技术虚拟出相互隔离的 vSDN 网络，该网路由租户自有的控制器实现集中管控，这样，租户便可以利用自己的 SDN 控制器完成对自有网络的灵活控制。为方便与 OpenStack 云平台的集成，本文在对 OpenStack 云平台改动最小、性能降低最低的情况下，完成了整体系统的搭建，系统将虚拟化平台与 OpenStack 内部的网络模块 Neutron 分工合作，Neutron 主要负责 OpenStack 内部网络资源的管理和控制，而本文中支持 SDN 模式的虚拟化平台主要负责物理服务器之间的物理网络的管理和控制，两者协同工作，共同完成 SDN 与 OpenStack 的集成工作，开放控制器给租户，实现租户对数据中心虚拟网络的灵活管控。

2. 链路带宽、时延测量。SDN 控制器开发给租户后，利用 SDN 控制器实现对 vSDN 网络的负载情况的精确测量，是每一个租户希望看到的，测量数据的精确性，直接决定了数据转发策略的制定。因此，全局网络的带宽、时延数据的测量，在本文中起到关键作用。针对带宽、时延，本文创新性地提出了 SDN 模式下的测量方法。首先针对链路可用带宽，本文采用包对技术，精确地测量了链路的可用带宽，测试证明，该方法在低带宽模式下具有精确的测量结果。针对时延，本文基于三点模式的数据报传输，通过统计时间差的模式，完成了数据的精确测量。带宽和时延数据，成为了租户进行链路定制化的重要依据，租户根据当前链路的负载情况，可以选取一条最优的链路进行数据的传输，在提高数据中心带宽利用率的同时，极大地提高了租户网络的数据传输速率。

3.2 系统架构设计

3.2.1 系统总体设计图

系统主要由四个模块组成。分别为网络虚拟化模块、控制器模块、通信模块和 GUI 模块。总体的架构图如图3-2所示。

如图中所示，架构底层主要包含支持 OpenFlow 协议的交换机，以及 OpenStack 内部的虚拟机资源，交换机既包含 OpenStack 内部的虚拟网桥，亦包含连接物理服务器之间的物理 OpenFlow 交换机。将这部分网络开放给租户，主要是为了便于租户实现对数据中心物理网络的可控性。租户可以根据链路时延、带宽定制化最优数据传

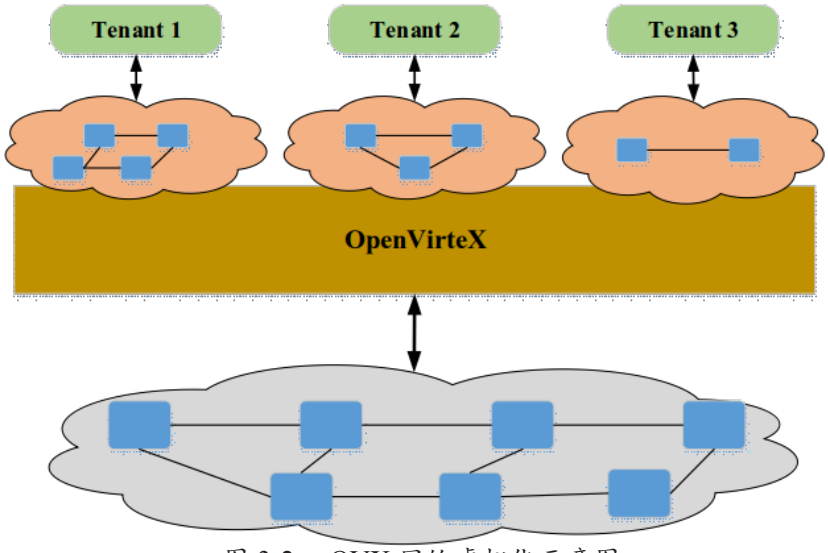


图 3-2　OVX 网络虚拟化示意图

输链路。支持网络虚拟化的 SDN 控制器，实现对底层网络和计算资源的集中控制。由该模块，可以创建相互隔离的租户虚拟网络，该虚拟网络支持 SDN 模式，租户可利用控制器实现对自由虚拟网络的集中管控。自主开发北向应用，实现特有的功能。

控制器模块主要实现对租户控制器的管理。租户的控制器实现对自有 vSDN 网络的集中管控，该控制器运行于特定虚拟机之中，租户可以登录虚拟机，进行相应的配置和开发工作。每一个 vSDN 网络对应唯一的 SDN 控制器。现有的控制器镜像，已经开发了对链路带宽、时延的测量功能，租户可以登录控制器所在的虚拟机，开发自己的北向应用。

通信模块主要实现异步通信，前端对网络拓扑的获取、带宽时延数据的获取以及虚拟网络的创建请求，均通过通信模块，分别和控制器、网络虚拟化模块完成数据的请求和获取。通信模块完成了进程间的异步通信。

前端 GUI 界面，主要为了方便用户的操作，用户可以在其 web 界面，通过简单的点击操作，完成对数据中心网络的定制化操作。包括网络拓扑的显示，链路带宽、时延的前端展示，虚拟网络的创建操作，定制化链路的前端选取等功能。前端模块的极大的提高了用户体验效果。

3.2.2　系统详细架构图

在 OpenStack 云平台数据中心网络中采用 SDN 架构，核心思想是用控制器对网路运营模式实现统一管控。现有云数据中心网络的模式均为单一节点控制，该模式下的 SDN 网络对于租户是不可见的。本文提出了多租户虚拟化网络的定制和管理方

案，实现了租户自有控制器对虚拟网络的灵活控制。多租户对应多控制器的机制便于租户对自有网络进行带宽时延查询、定制化流表下发、链路切换、控制器北向接口开发实验等自定义操作。系统的详细架构图如图3-3所示。

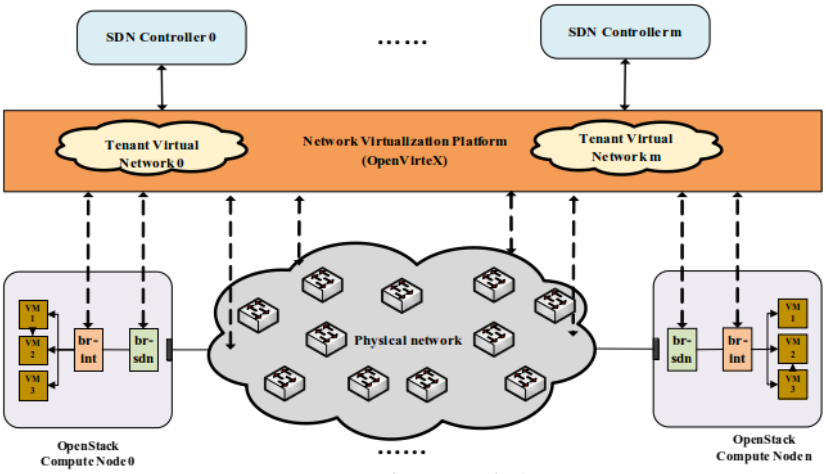


图 3-3 系统详细架构图

在兼容现有 OpenStack 网络模式的前提下，为 OpenStack 平台的每个计算节点添加 br-sdn 网桥，由于 br-int 网桥实际就是一个 NORMAL 交换机，不管其受 SDN 控制器控制与否，不会影响其作用，故我们将该网桥以及连接物理服务器之间的支持 OpenFlow 协议的交换机均受 OVX 控制器进行管控，OVX 控制器实现对网络的集中控制，与此同时，OVX 支持 SDN 模式下的网络虚拟化，可以虚拟出相互隔离的 vSDN 网络，vSDN 网络由租户自己的控制器进行集中控制。前端 GUI 模块基于 OpenStack 的 Horizon 模块进行二次开发，为其添加两个 Panel，分别实现对物理网络和虚拟网络的拓扑展示，全局链路带宽、时延展示，以及虚拟网络创建和删除的前端操作。

在该架构图中，OVX 与 OpenStack 中的 Neutron 组件并列存在，Neutron 主要管控 OpenStack 内部的虚拟网络，包括各种防火墙、router、DNS 服务，而 OVX 主要负责连接服务器之间的物理网络的连接状态。两者并列存在。相互协同工作。这种实现的模式可以在对 OpenStack 云平台改动最小、性能影响最低的情况下完成 SDN 与 OpenStack 的集成。

对于数据传输，现有架构下，有两种模式可以选择。租户可以选择使用传统网络模式进行数据的传输，数据经 br-int 和 br-tun，由传统模式交换机，进行数据的传输。与此同时，租户亦可通过流表的下发，实现 SDN 模式下的数据传输，该模式下，数据经由 br-int 和 br-sdn，途径支持 OpenFlow 协议的交换机，完成数据的转发工作。

两种模式的并存，租户可以自主选择切换模式。现有的 SDN 模式，相比于传统模式，SDN 模式下租户可以开发自己的北行应用，实现数据中心网络的相关功能的自定义。当然 SDN 现有模式仅仅实现了二层转发，跨路由的传输现阶段只能通过传统模式实现。

3.2.3 系统流程图

系统的整体流程图如图3-4所示。流程图主要对系统整体的工作流进行了阐述。详细讲解了从虚拟网络创建到链路定制化的整个流程。

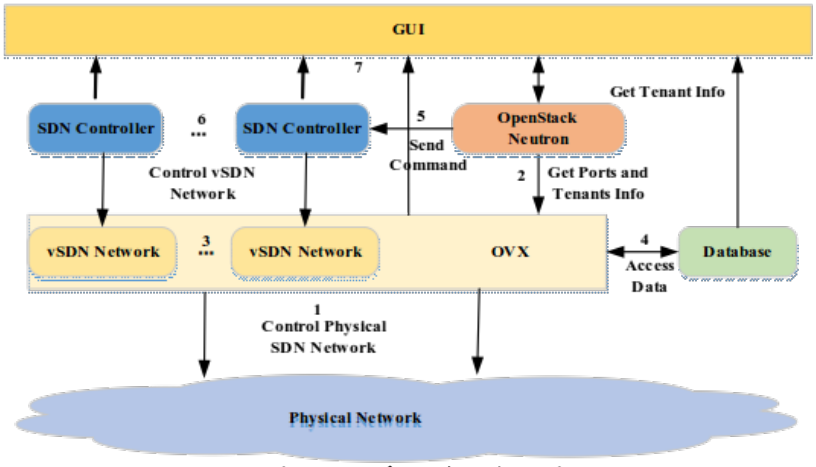


图 3-4　系统详细流程图

1. 网络虚拟化平台 OVX 实现对底层物理网络管理，完成物理网络的集中控制。本研究为其开发了创建虚拟网络的 API 接口 *createCustomNetwork*，租户可以调用该接口，实现虚拟网络的创建。OVX 自身维护虚拟网络与物理网络的映射表，完成南北向指令的映射工作。

2. 从 OpenStack Neutron 侧，获取虚拟机以及虚拟机绑定 br-int 网桥的端口信息，OVX 本身只能获取到交换机的连接信息，两者结合，得到全网的完整拓扑信息图。基于此物理网络信息，进行虚拟网路的创建。

3. 本研究对 OVX 封装了创建虚拟网路的 *createCustomNetwork* 接口，调用该 API 接口，完成虚拟网络的创建。该虚拟网络基于底层的物理资源，可以是底层物理资源的同构子图，也可以是自定义的虚拟网络拓扑图。

4. 在网络虚拟化平台 OVX 侧，每个虚拟网络由 *virtualnetworkid* 进行标识。在 OpenStack 中，每个租户有自己的 ID 号 *tenantid*。本文将两者的对应关系存储到数据库中。另外，对创建的虚拟网络以及虚拟网络和物理网络的对应关系，亦存储到数据库中，方便用户进行数据的读取和修改。本研究采用 MongoDB 数据库完成上述操作。由于 MongoDB 为非关系型数据库，处理 json 格式的数据非常便捷。

5. OpenStack Neutron 侧发送指令到 SDN 控制器。现阶段，指令主要包括带宽/延迟测量以及定制化流表的下发。未来我们的平台将提供更多的功能。

6. SDN 控制器接收到消息，通过我们开发的应用，完成相应的测量工作。最后返回对应的数据。租户可以登录到 SDN 控制器所在的虚拟机，开发自己的应用程序，以实现更多的功能。

7. GUI 界面完成物理网络、虚拟网络、链路带宽、时延等信息的获取，在前端界面完成展示工作。

3.3 模块详解

3.3.1 网络虚拟化模块

网络虚拟化允许不同需求的用户组访问同一个物理网络，但从逻辑上对它们进行一定程度的隔离，以确保安全。凭借网络虚拟化技术，能在单一物理基础设施上部署多个封闭用户组，并在整个网络中保持高标准的安全性、可扩展性、可管理性和可用性。通过网络虚拟化可实现弹性、安全、自适应、易管理的基础网络，充分满足服务器虚拟化等虚拟技术对基础网络带来的挑战，达到提高数据中心的运行效率、业务部署灵活、降低能耗、释放机架空间的目的。

本文选用的网络虚拟化平台，通过给每个租户提供访问一个虚拟网络拓扑和一个完整的网络头空间来完成虚拟网络的创建，呈现 OpenFlow 网络给租户，同时经由南向的 OpenFlow 接口控制底层的物理基础设施，该模式下租户可以自定义虚拟网络的拓扑结构，而且，完整的网络头空间保证了租户虚拟网络功能性和流量隔离。与网络切片进行虚拟化相比，该模式允许租户使用相同的 IP 或 TCP/UDP 端口，而网络切分会造成流量泄露。与此同时，网络切片必须限制为底层网络的同构子图。该虚拟网络平台在底层物理设备看来，为一个 SDN 控制器，而对于北向租户控制器而

言，其为网络中具有 OpenFlow 能力的交换机集合，所以，从某种意义上讲，网络虚拟化平台可以理解为控制平面和基础物理设施之间 OpenFlow 消息的多路复用器、解复用器。网络虚拟化模块主要由四部分组成，面向网络的南向接口、面向租户的北向接口、全局映射表和 API 服务器。具体架构如图3-5所示。

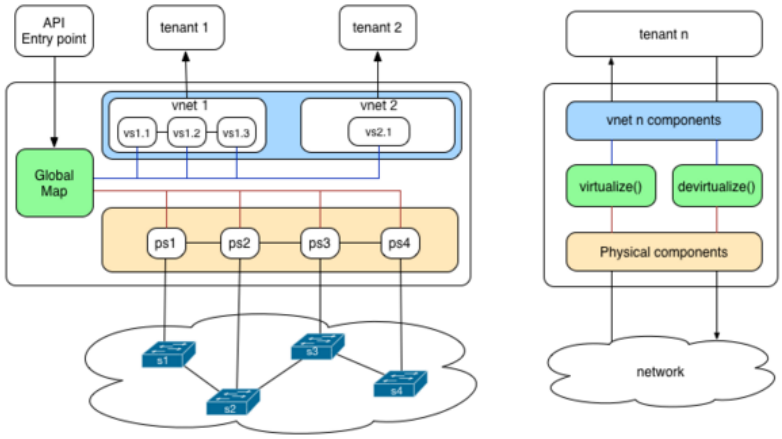


图 3-5　网络虚拟化模块架构

面向网络的南向接口，主要用建立和维护与基础设施的连接，即对底层物理网络实现集中管控。管理虚拟平台和基础设施之间的 OpenFlow 信道。面向租户的北向接口，主要是呈现 vSDN 网络给租户，管理租户控制器和 vSDN 网络之间的 OpenFlow 信道。全局映射表，完成物理网络和虚拟网络之间的相互映射，并且连接这两个网络。API 服务器，主要是监听 JSONRPC 调用系统配置和系统/网络状态信息，以此同时，租户亦可进行虚拟网络的创建、删除、修改工作。

虚拟化平台不仅拥有底层物理网络拓扑，而且还有每个租户的虚拟网络拓扑。底层的物理网络拓扑是通过虚拟化平台下发 LLDP 报文获取到的，而上层的虚拟网络拓扑则是由其本身生成的。即租户的控制器下发的 LLDP 报文不会直接送到底层物理网络，而是在虚拟化平台中通过查询底层物理拓扑，从而模拟了 LLDP 的过程，向上提供网络拓扑。虚拟化模块通过映射算法，解耦了底层网络和租户网络。即南向上，虚拟化模块作为控制器，接收底层物理设备的信息，并下发流表等报文到底层设备。北向上，其作为交换机集合，向租户的控制器发送 OF 报文。从而将一条完整的控制通路，变成了两段相互独立的控制通路。而这两者之间的映射关系，保存在映射表中。在图3-5中，蓝色部分是面向租户的，而橙色部分是面向交换机的。两者之间互不可见，即租户看不到物理的交换机数据。而绿色部分是两者结合的纽带，保存了网络映射的数据。在数据上行、下行的过程中通过查表修改的方式，翻译成

正确的数据。如图3-5右侧所示，上行数据需要调用 virtualize() 函数，而下行数据需要调用 devirtualize() 函数。

虚拟化平台中的租户隔离，作为虚拟化的一个核心需求，本研究中通过在 MAC 字段中添加额外的信息，使不同租户的流量得到隔离。具体来讲，虚拟化平台在对底层 OpenFlow 网络下发流表时，在数据包的入口交换机处，对数据包的 MAC 做了修改，添加了流 ID、租户 ID、虚拟链路 ID。流 ID 主要用于查询流入虚拟链路的特定流表；租户 ID 用于隔离不同租户；虚拟链路 ID 是我们能够跟踪该条虚拟链路。当数据报传输至目的交换机时，虚拟化平台下发特定流表，将数据包中的 MAC 地址字段修改为原始地址，数据报顺利到达目的主机。

综上所述，本文中选用的虚拟化平台主要实现以下四个功能。

- 创建了一个特定拓扑的独立虚拟网络。
- 租户使用自己的控制器对虚拟网络进行集中控制。
- 为每个租户使用整个地址空间，租户虚拟网络可以使用相同的 IP 地址。
- 可以动态地改变运行中的虚拟网络，并且能够在物理失效的情况下，自动恢复。

3.3.2 通信模块

通信模块基于消息队列实现，消息队列是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。消息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信。队列作为消息代理，给应用程序提供一个共同的平台，用于存储发送/接受的消息。直到消息从队列中被接收端收到为止。具体的通信模块如图3-6所示。

由图中可以看出，生产者绑定 Exchange，Exchange 是一个消息交换机，它指定消息按什么规则，路由到哪个队列。具体的路由规则由 RoutingKey 设定，RoutingKey 作为路由关键字，Exchange 根据这个关键字进行消息投递。队列用于消息的存储，直到消息被消费者从队列中读取完毕。

本文基于消息队列实现通信模块，带来诸多好处。消息队列降低了进程间的耦合度，即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。而这种允许重试或者延后处理请求的能力保证了平台的健壮性。消息队列提

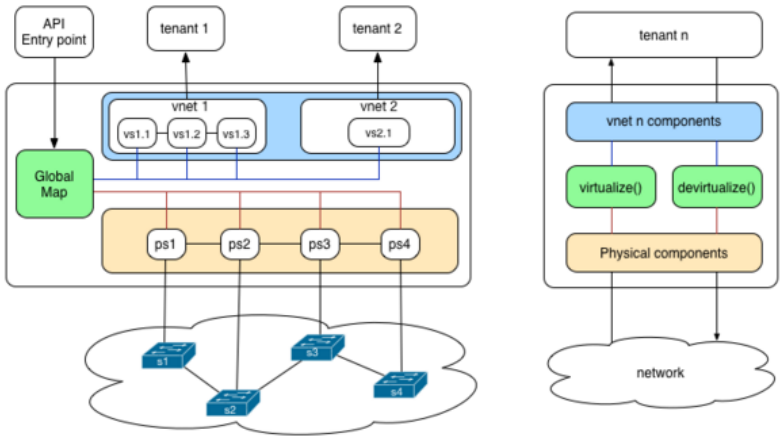


图 3-6　通信模块架构图

供的冗余机制保证了消息能被实际的处理，只要一个进程读取了该队列即可。在此基础上，消息队列提供了一个” 只送达一次” 保证。无论有多少进程在从队列中领取数据，每一个消息只能被处理一次。这之所以成为可能，是因为获取一个消息只是” 预定” 了这个消息，暂时把它移出了队列。除非客户端明确的表示已经处理完了这个消息，否则这个消息会被放回队列中去，在一段可配置的时间之后可再次被处理。很多时候，你不想也不需要立即处理消息。消息队列提供了异步处理机制，允许你把一个消息放入队列，但并不立即处理它。你想向队列中放入多少消息就放多少，然后在你乐意的时候再去处理它们。

3.3.3　控制器模块

控制器是 SDN 的重要组成部分，其设计与实现是 SDN 最为关键的技术环节之一。作为数据控制分离的 SDN 的操作系统，控制器具有举足轻重的地位，它是连接底层交换设备与上层应用的桥梁。一方面，控制器通过南向接口协议对底层网络交换设备进行集中管理、状态监测、转发决策以处理和调度数据平面的流量：另一方面，控制器通过北向接口向上层应用开放多个层次的可编程能力，允许网络用户根据特定的应用场景灵活地制定各种网络策略。

本研究中，我们提供了基于 Ryu 控制器的 Linux 镜像，租户可以基于次镜像进行控制器的创建，默认情况下控制器占用 TCP 的 6633 端口。租户只需要讲自己创建的虚拟网络指向自己的 SDN 控制器，即可实现自由控制器对虚拟网络的集中管控。租户可以根据自身需求进行控制器北向应用的相关开发工作。本文主要实现了链路的时延、带宽的测量，以及定制化流表的下发实验。

本文选用的 Ryu 控制器，是由日本最大的电信服务提供商 NTT 主导开发的基

于 Python 语言的开源控制器，是一个基于组件的软件定义网络架构，是一个基于 Apache 协议的完全开源的 SDN 控制器，对于软件组件提供了良好的 API 接口，便于开发人员创建新的网络管理与控制中的应用，Ryu 支持管理网络设备的多种网络协议，诸如：OpenFlow,Netconf,OF-config 等，租户 Ryu 架构和主要组件如图3-7所示。OF-conf、Netconf 等组件主要提供了对 OpenFlow 交换机的控制能力，Topology 用于对 SDN 网络的拓扑进行管理,REST 提供了上层的 API 接口。

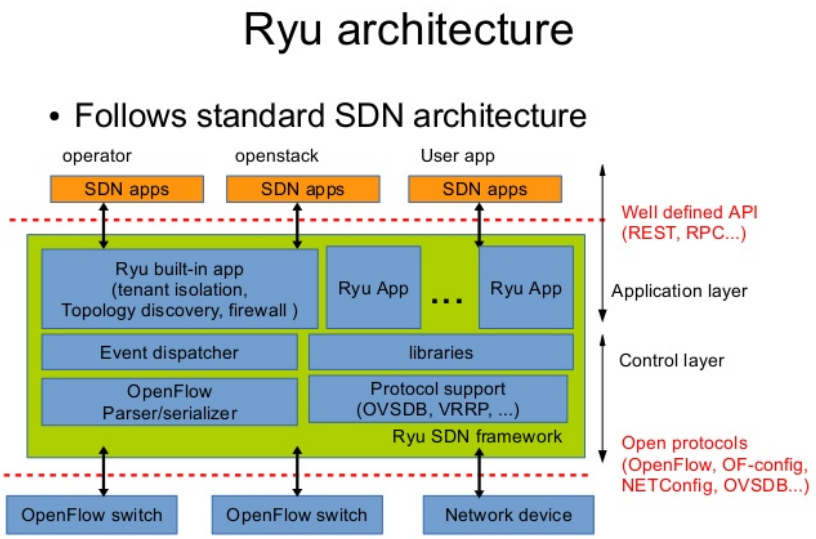


图 3-7 Ryu 架构图

3.3.4 GUI 模块

为方便用户对虚拟网络的操作，本研究为用户开放了前端 GUI 接口，用户可以在前端实现物理网络、虚拟网络的拓扑获取，虚拟网络的创建和删除，全局网络带宽、时延的测量，以及数据传输链路的选取等操作。开放的 GUI 界面，可以在用户不理解底层实现的基础上，进行便捷行的操作。

针对 GUI 模块，本文基于 OpenStack 的 Horizon 模块进行二次开发，为其添加了物理网络和虚拟网络显示的模块，在显示拓扑的基础上，实现虚拟网络创建、删除，以及网络负载查询等功能。

为了租户请求的安全性，我们为其添加了认证机制，在租户发出某一请求时，首先会向通过自己的户名和密码向认证模块申请 token，认证成功后，会返回该用户的唯一 token 标识，token 中包含租户的基本信息，以及租户的权限信息，如果认证

失败，该次请求结束，并返回错误信息。如果成功，租户会将刚才的请求与现有的token值合并，再一次发送至后端服务器进行请求，服务器端对token验证成功后，执行相应的请求，并将请求结果返回给前端。具体的流程如图3-8所示。

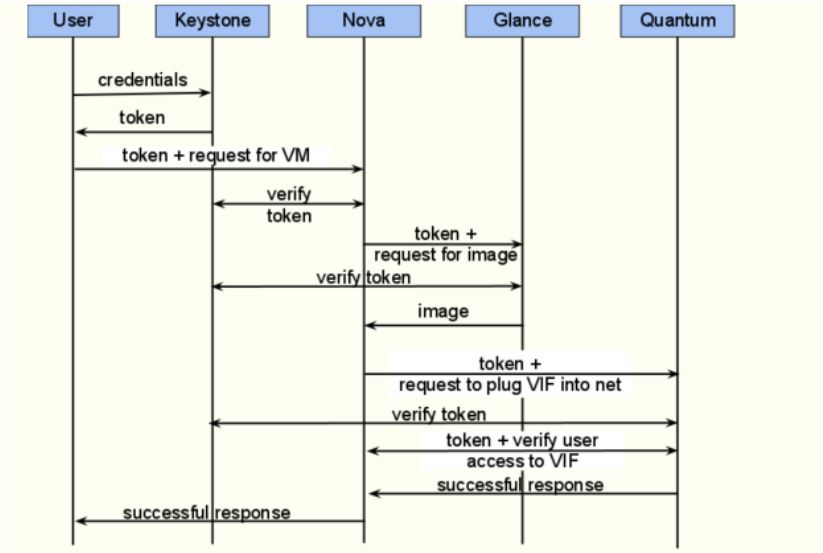


图 3-8　请求认证图

3.4　本章小结

本章主要对系统的架构做了详细的说明，为后面系统功能的实现做了铺垫。首先论文介绍了需求分析和关键技术，对本文研究内容的需求做了简要阐述，介绍了网络的虚拟化、链路负载测量、链路定制化、前端web界面的简单实现。通过对关键技术的分析说明，阐述了本文系统实现的关键点和难点，对系统搭建、链路负载测量的关键性做了分析。随后介绍了系统的整体架构图和流程图，最后对系统的各模块架构做了简单介绍，分别对网络虚拟化模块、通信模块、控制器模块、GUI模块的架构做了阐述。

第四章　系统功能的具体实现

基于 SDN 的云平台多租户虚拟网络定制化研究的实现，需要完成系统的搭建、系统各模块功能的实现，以及相互之间的协调工作。系统架构中，网路虚拟化模块，完成虚拟网络的创建、删除工作；通信模块实现进程间的异步通信；控制器模块主要进行带宽、时延的测量工作，已经通过定制化流表的下发完成链路的选取；GUI 模块主要实现拓扑信息、虚拟网络创建、测量数据展示的功能。各模块之间协同工作，实现基于 SDN 的云平台多租户虚拟网络的定制化。

4.1　网络虚拟化模块

4.1.1　虚拟化和去虚拟化流程

网络虚拟化模块，主要包含虚拟化和去虚拟化两个方面，如第三章图3-5所示。主要涉及到交换机的转换、OpenFlow 字段转换（包括 Cookie、Buffer ID, XID）、地址的虚拟化。其中对于交换机的转换，包括针对租户控制器发送的南向消息，进行去虚拟化，由 VirtualSwitch 映射成 PhysicalSwitch；针对底层物理网络的北向消息，进行虚拟化操作，由 PhysicalSwitch 映射成 VirtualSwitch。地址的虚拟化，是针对主机创建 VirtualIP 和 PhysicalIP，以避免租户流量之间存在地址空间冲突，图4-1描述了地址虚拟化的具体过程。

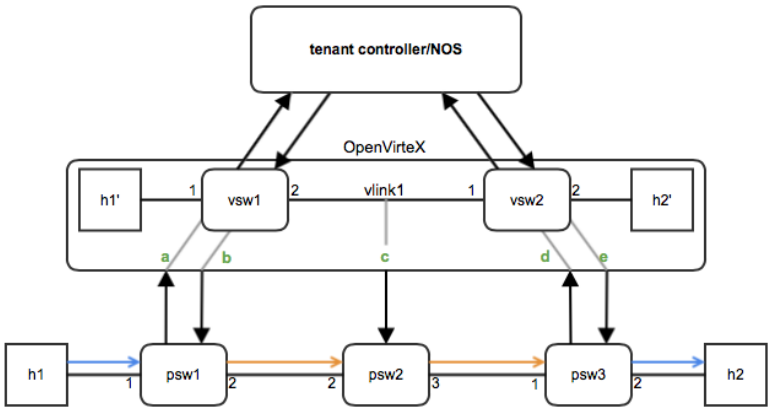


图 4-1　地址虚拟化过程

虚拟化流程如下：

a) PacketIn 消息携带 VirtualIP，未经修改直接发送给租户控制器

b) 相应的 FlowMod 匹配 VirtualIP，执行的 OFAction 为将值修改为 PhysicalIP。

c) 虚拟链路通过核心侧交换机 psw2 映射回两跳路径。

d) 目的地边缘的 PacketIn 与核心网络中的转换行为一致。

e) 虚拟化平台下发与 PhysicalIP 匹配的 FlowMod 消息。执行的 OFAction 为重写回 VirtualIP。

转换的过程主要对 PacketIn、PacketOut 和 FlowMod 消息的虚拟化与去虚拟化。针对 PacketIn 消息，由于该消息是来自南向物理交换机，所以消息的处理过程为虚拟化过程，具体的虚拟化过程如图4-2所示。首先判断该数据报是否来自边缘侧交换机，如果是，直接查看租户 ID 和虚拟交换机，如果不是，通过对数据域的匹配，获取租户的 ID 和虚拟交换机, 进而检查该租户的控制器是否以建立连接，如果建立连接，直接将消息发送给租户控制器，否则丢弃。针对 L3 层的数据报，首先检查 PhysicalIP 的合理性，在合理的情况下，查看虚拟 IP 和交换机，并将数据包中的 IP 地址进行重写。

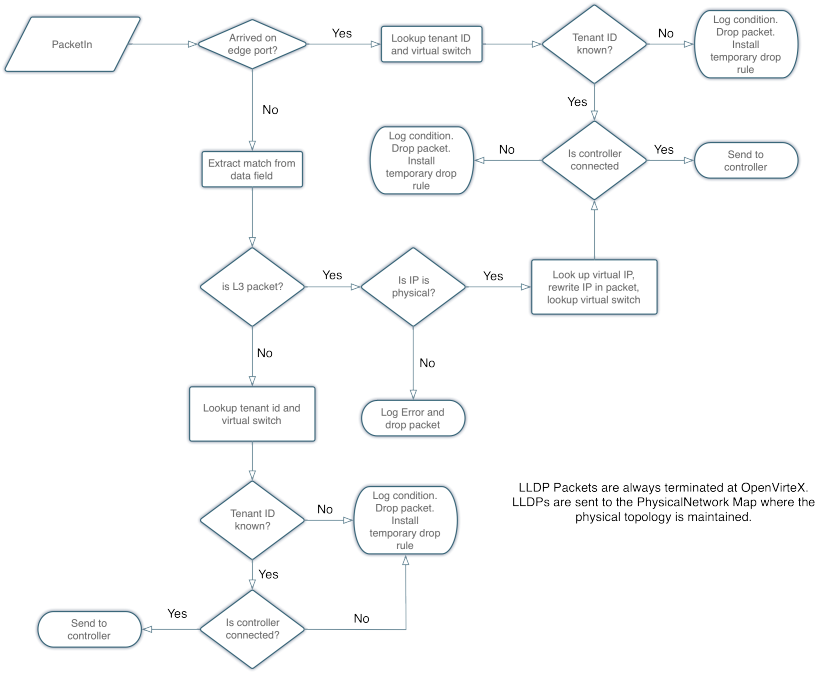


图 4-2　PacketIn 虚拟化

针对 PacketOut 消息，由于该消息是由北向租户控制器下发给交换机，所以该消息的处理过程是去虚拟化过程，具体的去虚拟化过程如图4-3所示，首先减产 bufferID 的合理性，进而从 data 中提取匹配域，重写端口以及 bufferID 等信息，执行的动作是修改地址为 PhysicalIP。

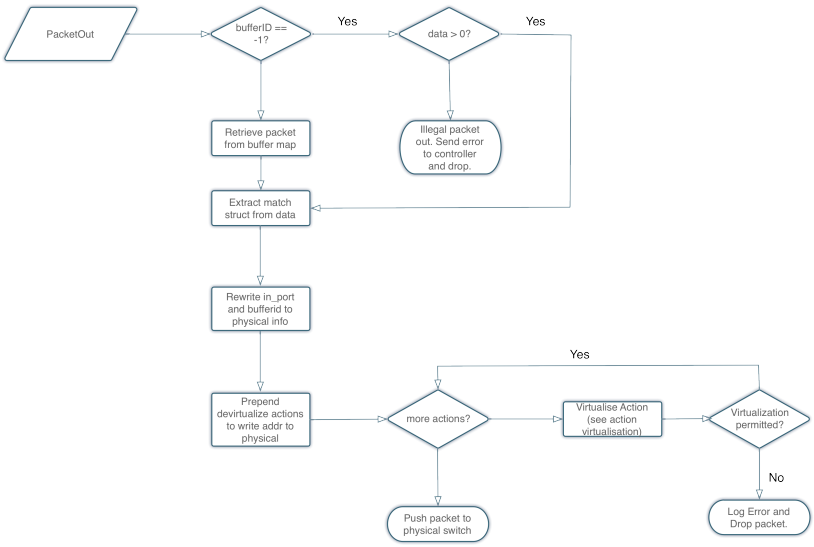


图 4-3　PacketOut 去虚拟化

针对 FlowMod 消息，由于该消息是由北向租户控制器发送给交换机，因此，该过程也是一个去虚拟化的过程。首先需要判断该条流是来自入口交换机、出口交换机，还是核心交换机，针对不同的情况进行具体的操作。如果该消息来自出口交换机或者核心交换机，需要将匹配域重写为 PhysicalIP，然后，依次将执行的动作重写为物理地址，然后将消息下发给物理交换机。

4.1.2　虚拟网络创建

针对虚拟网络的创建，本研究中为其封装了 API 接口 *createCustomNetwork*，该接口的输入数据为 Json 格式的物理网络拓扑图，具体的输入信息如下：

```
topo = {
  'subnet':'type:str', #虚拟网络的网络地址/位数，例如'192.168.0.0/24'
  'switches':[
    {
      'dpid':'type:str'
    }
  ], #物理交换机列表，列表中的每个交换机由dpid标识
  'hosts':[
    {
      'mac':'type:str',
      'port':'type:int',
      'dpid':'type:str'
    }
```

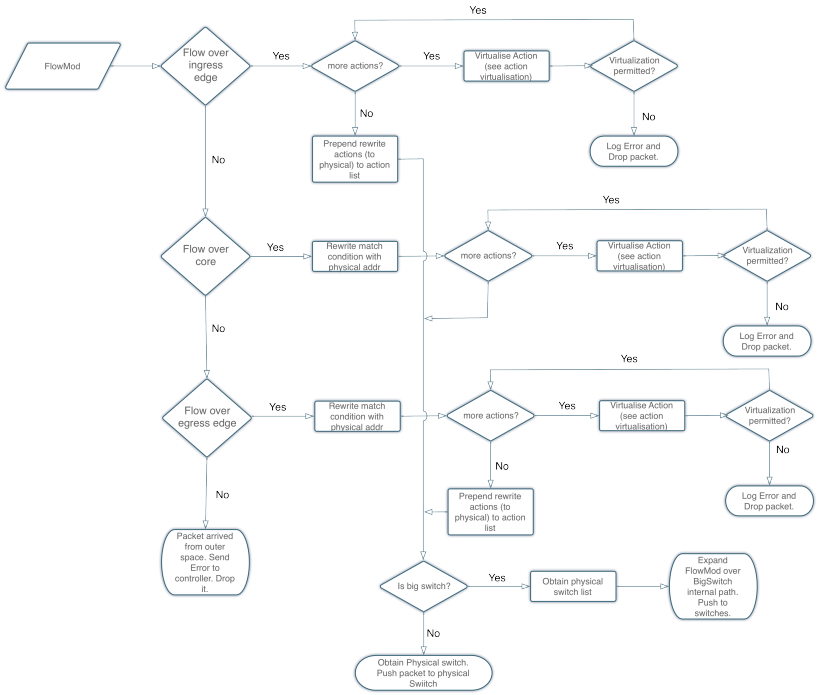


图 4-4　FlowMod 虚拟化

```
  ], #list, 物理主机列表, 列表中每条主机信息为dict, 存储主机的mac地址、连接物理交换机的dpid
                                                和port
  'ctrls':'type:str', #租户控制器信息, 包括租户控制器所在IP以及占用的TCP端口, 例如'tcp:10.0
                                                .0.2:6633'
  'method':'type:str', #请求的函数方法名,
  'routing':{
    'algorithm':'type:str'
  }, #路由算法, 例如'spf'
}
```

进行虚拟网络创建的伪代码如伪代码 1 所示，主要分 4 个步骤完成，首先基于租户控制器信息以及创建虚拟网络的子网信息，创建一个逻辑的虚拟网络，返回虚拟网络的 ID 号；进而基于交换机信息，创建一一对应的虚拟交换机信息；然后对主机信息，依次创建虚拟端口，绑定到特定虚拟交换机，最后，对链路信息，依次创建端口映射，进而对相连接的端口建立虚拟链路。至此，整个虚拟网络创建成功。

4.2　通信模块

通信模块主要实现进程见得异步通信，本文基于消息队列实现，由发送方和接收方组成。发送放主要实现的功能是将需要发送的数据，准确无误的发送至消息队列中，消息队列用于存储通信双方需要传送的数据，直到接收方从队列中读取完成。

发送方的具体实现代码如下：

```
#!/usr/bin/env python
```

伪代码 1 虚拟网络创建

输入： 物理网络拓扑图，$topo$

输出： 租户虚拟网络 ID 号，$tenantId$

function createCustomNetwork($topo$)
    $ctrls \leftarrow topo['ctrls']$
    $subnet \leftarrow topo['subnet']$
    $route \leftarrow topo['routing']['algorithm']$
    $switches \leftarrow topo['switches']$
    $hosts \leftarrow topo['hosts']$
    $links \leftarrow topo['links']$
    $tenantId \leftarrow createvNetwork(ctrls, subnet)$
    for $dpid$ **in** $switches$ **do**
        $createvSwitch(tenantId, dpid)$
    end for
    for $host$ **in** $hosts$ **do**
        $(vdpid, vport) \leftarrow createvPort(tenantId, host['dpid'], host['port'])$
        $connectvHost(tenantId, vdpid, vport, host['mac'])$
    end for
    for $link$ **in** $links$ **do**
        $(srcdpid, srcport) \leftarrow (ink['src']['dpid'], link['src']['port'])$
        $(dstdpid, dstport) \leftarrow (link['dst']['dpid'], link['dst']['port'])$
        $(srcvdpid, srcvport) \leftarrow createvPort(tenantId, srcdpid, srcport)$
        $(dstvdpid, dstvport) \leftarrow createvPort(tenantId, dstdpid, dstport)$
        $connectvLink(tenantId, srcvdpid, srcvport, dstvdpid, dstvport, route)$
    end for
    $startvNetwork(tenantId)$
    return $tenantId$
end function

```
#coding=utf8
import pika

class Sender(object):
    def __init__(self, host):
        """
```

```
        self.connnection:连接rabbitmq服务器
        self.channel:定义队列
        self.result:定义返回队列
        """
        self.connection = pika.BlockingConnection(pika.ConnectionParameters(host=host))
        self.channel = self.connection.channel()
        self.result = self.channel.queue_declare(exclusive=True)
        self.callback_queue = self.result.method.queue
        self.channel.basic_consume(self.on_response,no_ack=True,queue=self.callback_queue)

    def on_response(self, ch, method, props, body):
        self.response = "success"

    def request(self, body):
        self.response = None
        self.channel.basic_publish(exchange='',
                                   routing_key='test',
                                   properties=pika.BasicProperties(
                                         reply_to = self.callback_queue,
                                         ),
                                   body=body)
        #接收返回的数据
        while self.response is None:
            self.connection.process_data_events()
        return self.response

sender = Sender('127.0.0.1')
response = sender.request("request")
print " [x] Requesting request"
print " [.] Got %s"%response
```

接受方具体实现代码如下：

```
#!/usr/bin/env python
#coding=utf8
import pika

class Receiver(object):
  def __init__(self, host):
    """
    self.connnection:连接rabbitmq服务器
    self.channel:定义队列
    self.result:定义返回队列
    """
    self.connection =  pika.BlockingConnection(pika.ConnectionParameters(host=host))
    self.channel = self.connection.channel()
    self.channel.queue_declare(queue="test")
    self.channel.basic_qos(prefetch_count=1)
    self.channel.basic_consume(self.request, queue='test')

  def request(self, ch, method, properties, body):
    print "Receive %s"%body
    response = "success"
    ch.basic_publish(exchange='',
             routing_key=properties.reply_to,
                      body=response)
    ch.basic_ack(delivery_tag = method.delivery_tag)

receiver = Receiver("127.0.0.1")
receiver.channel.start_consuming()
```

4.3　控制器模块

4.3.1　可用带宽测量

本文基于包对算法，实现细粒度的链路可用带宽的测量，包对技术的核心思想是，连续发送两个背靠背的数据包，由交换机 A 发送至交换机 B，根据两个数据包到达交换机 B 的时间差，用数据包的长度/时间差，得到精确的链路可用带宽。经测试证明，该方法在低带宽模式下，具有相当精确的测量值。

该技术的实现，要求精确的时间差，对此本文拟定的方案是：首先 SDN 控制器对交换机下发两条特殊的流表，用于进行带宽测量的数据包的转发工作。对入口交换机下发流表执行的动作是连续两次转发该数据报，从而在链路中产生两个背靠背传输的数据报，对于出口交换机下发流表执行的动作是将数据报转发回控制器。具体如图4-5所示，为了在数据包中存储到达目的交换机的时间，本文对虚拟交换机(Open Virtual Switch，OVS）进行源码变更，在匹配到特殊数据包（用于测量带宽的特定数据包，用特殊的 IP 地址标识）时，对数据包做改动，将数据包 IPv6 字段替换成当下时间，而后将数据包发送给控制器，控制器从数据报中解析出到达交换机的时间，即可得到背靠背数据包到达目的交换机的时间差，数据包的大小与时间差的比值，即为当前链路的可用带宽。而对于测量用的数据包，本文通过 SDN 控制器模拟一个满足需求的特定数据包下发给交换机，用于在交换机之间进行传输。

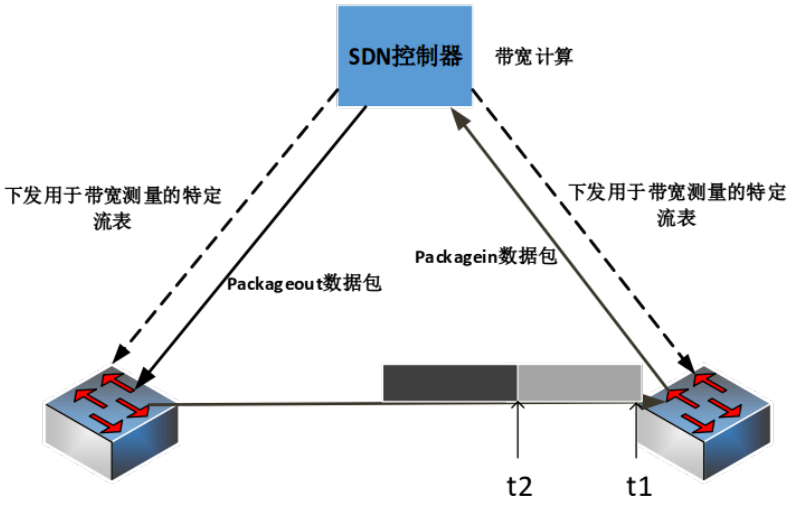


图 4-5　链路可用带宽测量模型

对于 OVS 的开发工作，主要涉及到对特定数据包的修改工作，即将数据包中的 IPv6 字段替换为当前的时间，具体如伪代码 2 所示。由伪代码可以看出，本文对特定数据包的要求为源、目的 IP 均为 0，由于该数据包由 SDN 控制器模拟产出，而且

在实际的生活中是不存在的，所以不影响整个系统的运行。

伪代码 2 OVS 为数据包添加当下时间

输入： 数据包，*packet*

输出： 包含当前时间的数据报，*newpacket*

```
1:  function modifyIPv6totime(packet)
2:      srcip ← packet.get('srcipv4')
3:      dstip ← packet.get('dstipv4')
4:      if srcip == 0 and dstip == 0 then
5:          localtime ← time.time
6:          packet.update(ipv6,localtime)
7:      end if
8:      newpacket ← packet
9:      return newpacket
10: end function
```

对于控制器端主要开发的功能包括，第一、模拟源、目的 IP 都为 0 的 IP 数据包。第二、为两个交换机下发用于测量带宽的流表。第三、对于控制器接收到的数据包，如果匹配数据包是用来测量可用带宽的，对数据包进行解析，获取时间差，计算当前链路的可用带宽。具体如伪代码 3 所示。

4.3.2 已用带宽测量

对于已用带宽的测量，本文基于数据包统计的方案实现。由第二章中的表2-1可以看出，流表中具有计数器字段，计数器用于统计流表被匹配的次数，以及匹配的数据包的大小。本文基于此，提出了基于计数器的已用带宽测量，通过统计某段时间内，从交换机某端口发出/接受的数据包数，取平均值进行已用带宽的测量工作，具体实现如伪代码 4 所示。

4.3.3 时延测量

时延的测量，本研究采用最基本的基于三点的测量方法。即控制器与两台交换机组成最小的测量单元，分别测出控制器到交换机的时延以及三边的总时延，做差值即为链路的时延。模型如图5-6所示，图中链路的时延即为 t3-(t1+t2)/2。

伪代码 3 SDN 控制器测量可用带宽

输入： 链路的源、目的交换机信息、交换机上报的数据包，$srcswitchdstswitch, packets$

输出： 当前链路的可用带宽，$bandwidth$

function add_flow($srcswitchdstswitch$)
    $srcaction \leftarrow [OFPActionOutput(2), OFPActionOutput(2)]$
    $dstaction \leftarrow [OFPPCONTROLLER]$
    $macth \leftarrow OFPMatch(src = \text{"0.0.0.0"}, dst = \text{"0.0.0.0"})$
    $addFlowMod(srcswitch, match, srcaction)$
    $addFlowMod(dstswitch, match, dstaction)$
end function
function simulatePacket($srcswitch$)
    $pkt \leftarrow packet.Packet()$
    $pkt.update(ipv4(srcip = \text{"0.0.0.0"}, dstip = \text{"0.0.0.0"}))$
    $sendpacketout(pkt, srcswitch)$
end function
function calculateBandwidth($packets$)
    $starttime \leftarrow packets[0].get(ipv6)$
    $endtime \leftarrow packets[1].get(ipv6)$
    $bandwidth \leftarrow (packets[0].size()/(endtime - starttime))$
    return $bandwidth$
end function

具体实现如伪代码 5 所示。主要包含流表下发、数据报模拟、时延计算模块，首先计算数据包从控制器到交换机的时延，需要为交换机下发流表，匹配到特定数据包以后，返回给控制器，而后控制器模拟出符合该匹配规则的数据包，发送个控制器，记录当前的时间，数据包到达交换机后，匹配上特定流表，数据包返还给控制器，至此控制器根究当前时间，利用时间差求解出控制器到交换机的时延。同理可以得到控制器 -> 源交换机 -> 目的交换机 -> 控制器的时延，利用前面的公式即可得到链路的时延信息。

伪代码 4 SDN 控制器测量已用带宽

输入： 链路的源、目的交换机信息，以及交换机的端口号，$srcswitchsrcport, dstswitch, dstport$

输出： 当前链路的已用，$bandwidth$

```
1:  function get_counters(srcswitchsrcport, dstswitch, dstport)
2:      inputBytes ← getCounters(srcswitch, inport = srcport)
3:      outputBytes ← getCounters(srcswitch, outport = srcport)
4:      allBytes ← (inputBytes + outputBytes)
5:      return allBytes
6:  end function
7:  function calculateBandwidth(srcswitchsrcport, dstswitch, dstport)
8:      bytes1 ← getcounters(srcswitchsrcport, dstswitch, dstport)
9:      time.sleep(sometime)
10:     bytes2 ← getcounters(srcswitchsrcport, dstswitch, dstport)
11:     bandwidth ← ((bytes2 − bytes1)/sometime)
12:     return bandwidth
13: end function
```

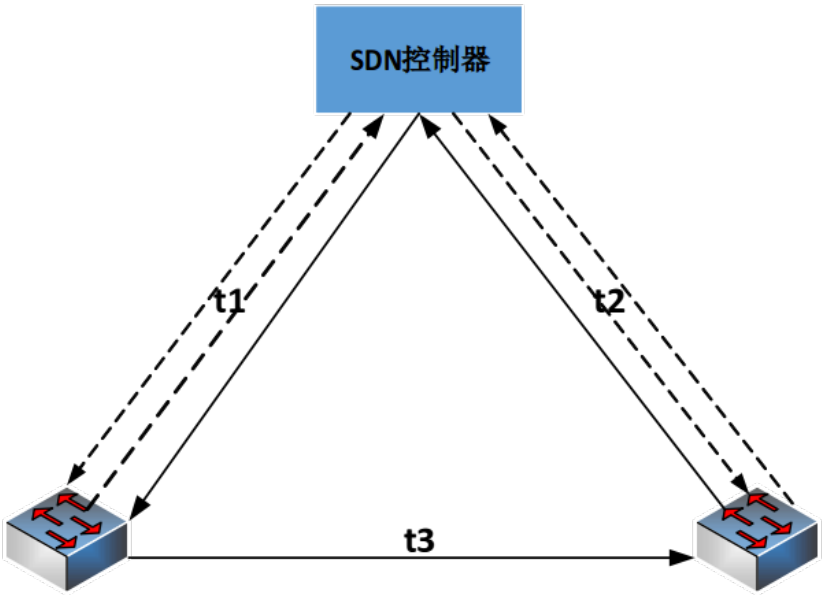


图 4-6　链路时延测量模型

伪代码 5 SDN 控制器测量链路时延

输入： 链路的源、目的交换机信息，以及交换机的端口号，$srcswitch, srcport, dstswitch, dstport$

输出： 当前链路的时延，$delay$

function add_flow($srcswitch, dstswitch, srcaction, dstaction$)
    $macth \leftarrow OFPMatch(srcmac = "00:00:00:00:00:00", dstmac = "00:00:00:00:00:00")$
    $addFlowMod(srcswitch, match, srcaction)$
    $addFlowMod(dstswitch, match, dstaction)$
end function
function simulatePacket($switch$)
    $pkt \leftarrow packet.Packet()$
    $pkt.update(mac(src = "00:00:00:00:00:00", dst = "00:00:00:00:00:00"))$
    $sendpacketout(pkt, switch)$
end function
function calculateDelay($packets$)
    $t1 \leftarrow time(controller->srcswitch->controller)$
    $t2 \leftarrow time(controller->dstswitch->controller)$
    $t3 \leftarrow time(controller->srcswitch->dstswitch->controller)$
    $delay \leftarrow (t3-(t1+t2)/2)$
    return $delay$
end function

4.3.4　定制化流表下发

定制化流表的下发，主要是根据用户定制化的链路信息，进行流表的下发工作，从而实现数据传输路径的定制化，即租户可以根据当前拓扑的负载情况，选择一条最优的链路，通过定制化流表的下发，实现数据的高效传输。输入的数据为 Json 格式的链路信息。具体数据如下。

```
link = {
  'hosts':{
    'src':{
      'mac':'type:str'
    },
    'dst':{
      'mac':'type:str'
```

```
    }
  }, #链路两端的主机信息，包括源主机的mac地址，目的主机的mac地址
  'switches':[
    {
      'dpid':'type:str',
      'in_port':'type:int',
      'out_port':'type:int'
    }
  ] #链路的交换机列表集合，每个元素每一个交换机数据，主要包括交换机的ID号、输入端口、
                                                    输出端口、
}
```

具体实现如伪代码 6 所示，针对用户发送的数据信息，分别对交换机下发定制化的流表。这里针对每个交换机，均要添加反向流表。从而实现主机的双向通信。

伪代码 6 SDN 定制化流表下发

输入： 定制化链路信息，$link$

function add_flow($switch, match, out\_port$)
    $action \leftarrow [OFPActionOutput(out\_port)]$
    $addFlowMod(switch, match, action)$

end function
function add_customflow($link$)
    $hosts \leftarrow link['hosts']$
    $switches \leftarrow link['switches']$
    $srcmac \leftarrow hosts['src']['mac']$
    $dstmac \leftarrow hosts['dst']['mac']$
    for $switch$ **in** $switches$ **do**
        $match1 \leftarrow OFPMatch(srcmac = srcmac, in\_port = switch['in\_port'])$
        $match2 \leftarrow OFPMatch(srcmac = dstmac, in\_port = switch['out\_port'])$
        $add\_flow(switch, match1, switch['out\_port'])$
        $add\_flow(switch, match2, switch['in\_port'])$
    end for
end function

4.4　GUI 模块

本模块主要实现前端可视化的操作，为租户提供操作的便捷性，本文中基于力导向图，实现了拓扑图的自动布局。该模式下，会对全网的整个拓扑图进行自动定位，节点与节点之间的互斥力，保证了节点的不可重合性。与此同时，拓扑元素可以实现可拖拽功能和手动定位功能，极大地提高了用户体验性。对于拓扑展示，本文主要实现了 SDN 物理网络、SDN 虚拟网络的拓扑展示，在各自的拓扑图基础上，实现相应功能的具体操作和展示。具体如下。

4.4.1　物理网络显示图

物理网络的拓扑显示如图4-7所示，黑色的交换机代表云平台内部的 br-int 网桥，蓝色交换机是用 OVS 模拟的物理服务器之间的交换机集群，由图可以看出，租户可以进行物理网络的初始化，力导向图可以对拓扑中的元素进行自动布局。在次基础上，租户可以进行全局网络可用带宽的测量，测量数据会在相应的链路之上进行显示。通过对链路的点击选取，在输入框输入租户自由控制器的 IP 地址，可以实现 vSDN 网络的创建，创建的虚拟网路由租户自由的控制器进行集中管控。除此之外，租户以可以进行删除虚拟网络的操作。

图 4-7　物理网络拓扑显示图

4.4.2 虚拟网络显示图

虚拟网络的拓扑显示图如图5-3所示，该界面主要是实现对租户虚拟网络的拓扑展示，由图中可以看出租户可以进行虚拟网络图的初始化。现阶段，云平台提供的SDN 控制器镜像，实现了已用带宽、时延的测量以及通过定制化流表进行选路的北向应用。未来租户可以自由开发 SDN 北向应用。测量的数据会在拓扑图的相应链路上进行显示，对于选路功能，租户通过点击链路的方式进行数据传输链路的定制化，后台通过解析链路信息，下发定制化流表，实现特定数据传输路径的设置。

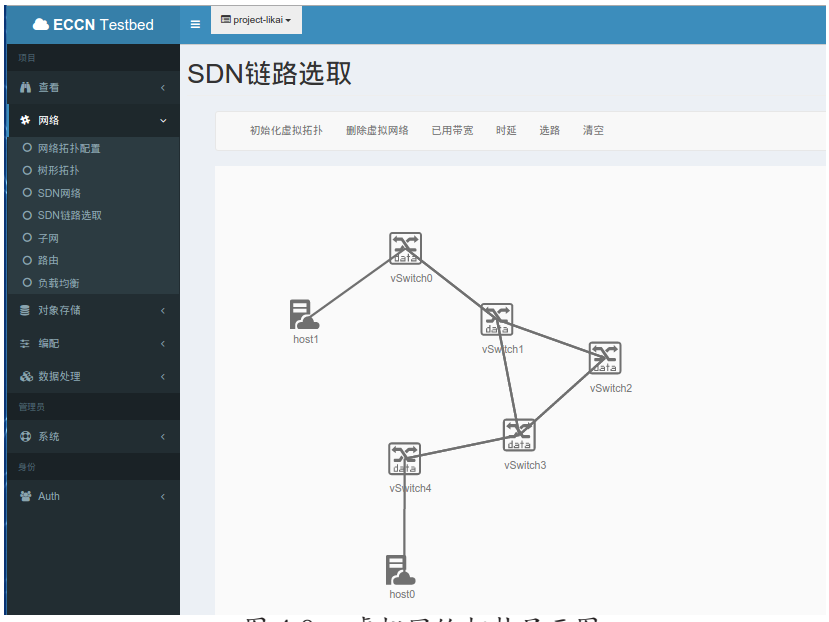


图 4-8　虚拟网络拓扑显示图

4.5 本章小结

本章主要对系统功能的具体实现做了详细的介绍。首先论文介绍了网络虚拟化模块，对虚拟化和去虚拟化的流程做了详细的解读，分别对 PacketIn、PacketOut、FlowMod 的虚拟化和去虚拟化的流程进行了讲解。针对虚拟网络的创建，主要分 4 个步骤，在伪代码中做了详细介绍。通信模块主要介绍了基于 RabbitMQ 实现进程间通信的代码实现。对于控制器模块，本文对 SDN 模式下，可用带宽、已用带宽、时延定制化流表下发进行了详细的介绍，对测量功能的伪代码做了分析。最后在 GUI 模块中，介绍了物理网络、虚拟网络拓扑图的前端显示，以及在相应模块上实现的不同功能。

第五章 系统测试

在实现系统功能的基础上，需要对系统的功能、性能进行测试。本文搭建了多节点云平台，在此基础上进行了功能测试和性能测试。功能测试验证了租户虚拟网络之间的隔离性，带宽、时延的测量功能，以及租户数据传输的链路切换功能。性能测试对 SDN 模式下流表的下发性能进行测试，验证 SDN 模式下集群大小对流表下发性能的影响。通过对 SDN 模式和传统模式下数据传输效率的测试。验证 SDN 模式下数据传输的效率是否有所降低。

5.1 实验环境

本文搭建的基于 SDN 的云平台，包含 4 台物理服务器，其中有一台单独配置为网络节点，主要负责云平台中网络资源的管理。两台单独配置为计算节点，一台配置为控制节点和计算节点。计算节点主要负责虚拟机的创建，控制节点实现集中调度的功能。具体各节点以及服务器硬件信息如表5-1所示。

表 5-1 云平台实验环境

属性	服务器配置	操作系统
计算节点	ubuntu1(内存：64G，CPU：24 核，硬盘：2.3T) ubuntu5(内存：64G，CPU：24 核，硬盘：2.4T) ubuntu6(内存 64G:，CPU：24 核，硬盘：450G)	Ubuntu-14.04
网络节点	ubuntu-1404-1(内存：24G，CPU：4 核，硬盘：1.4T)	Ubuntu-14.04
控制节点	ubuntu5(内存：64G，CPU：24 核，硬盘：2.4T)	Ubuntu-14.04

在云平台中，本文选用 project-likai 租户进行相关测试。该租户共计包含 11 台虚拟机，其中 10 台虚拟机基于 CirrOS 镜像进行创建，1 台虚拟机基于 Ryu 镜像进行创建，该镜像集成 Ryu 控制器，本文对该控制器开发了北向应用，可以实现链路带宽、时延、定制化流表下发的功能。与此同时，为该镜像集成了开机自启动脚本，在虚拟机启动时，自动运行 Ryu 控制器及相关应用。默认情况下，该控制器占用 TCP 的 6633 端口，租户可以登录该控制器虚拟机，自定义控制器的启动端口。虚拟机的详细列表如图5-1所示，图中详细介绍了虚拟机的名字、镜像、IP 地址、虚拟机的运行状态，以及虚拟机的配置信息。图中的 m1.tiny，代表的配置信息为内存 512M、硬盘

1G、vCPU 为 1 核。物理服务器之间的

图 5-1　租户虚拟机列表

5.2　功能测试

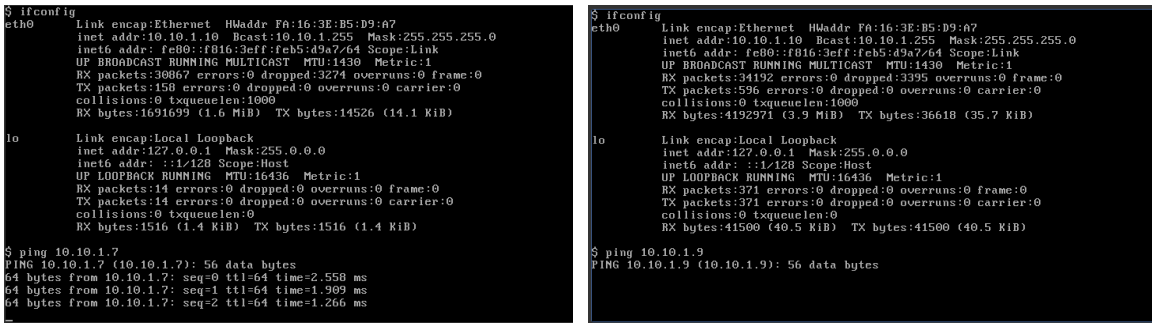
5.2.1　租户隔离性

本文基于底层的物理网络，进行了虚拟网络的创建，具体如图5-2所示，由图中可以看出，本文选取 test-3,test-4,switch0,switch2,switch3,switch5,switch7 进行虚拟网络的创建，控制器为运行在 Ryu-controller 虚拟机中，IP 地址为 192.168.1.100，租户创建的虚拟网络由运行在该虚拟机中的 Ryu 控制器进行集中管控。

图 5-2　虚拟网络创建图

创建生成的虚拟网络如图5-3所示，与上图之间是一一对应的关系。

图 5-3　虚拟网络拓扑图



(a) test-3 与 test-4 测试结果　　(b) test-3 与 test-5 测试结果

图 5-4　租户隔离性测试图

为了进行租户隔离性的测试，本文选取 test-3 虚拟机，让其分别与 test-4 和 test-5 进行 ping 测试，由图5.4(a)可以看出，test-3 与 test-4 属于同一个虚拟网络，可以进行正常通信。而对于 test-3 和 test-5，由图5.4(b)可以看出，两虚拟机不属于租户创建的同一个虚拟网络内，故不能直接进行通信。实验表明，租户创建的虚拟网络之间是相互隔离的，数据传输是安全可靠的。

5.2.2　带宽、时延测量

为方便租户在前端进行网络负载的查看，本文实现了对物理网络和虚拟网络带宽、时延的测量以及前端界面的可视化。对于物理网络的带宽测量，测试结果如图5-5所示，实验表明，租户可以查看到物理网络当前的带宽使用情况。而对于虚拟

网络的带宽测量，后续章节进行详细介绍。

图 5-5　物理网络带宽测量图

对于时延，本文仅仅实现了对虚拟网络的时延测量工作，通过为租户控制器开发的北向应用，实现虚拟网络的时延测量工作，测试结果如图5-6所示，实验表明，租户控制器可以实现对虚拟网络时延数据的测量工作。

图 5-6　虚拟网络时延测量图

(a) 定制化链路 1　　(b) 定制化链路 2

图 5-7　虚拟网络链路切换图

5.2.3　链路切换

对于租户创建的虚拟网络，本文实现了链路切换的功能，可以让租户实现对数据传输路径的定制化，如图5-3所示，host0 和 host1 之间有存在两条传输路径。分别为链路 1：host1-vswitch0-vswitch1-vswitch3-vswitch4-host0，和链路 2：host1-vswitch0-vswitch1-vswitch2-vswitch3-vswitch4-host0，本文首先选取链路 1 进行数据传输，通过选路按键，进行定制化流表下发后，host0 与 host1 进行 ping 测试，得到的带宽测量图如图5.7(a)所示，由图中可以看出，该模式下，数据按照链路 1 进行数据传输。随后，选取链路 2，并通过选路下发定制化流表，进而进行带宽测试，发现数据传输的通路已然变为链路 2，具体如图5.7(b)所示。实验表明，租户可以自定制数据转发路径，通过选路操作进行定制化流表的下发，从而实现对数据转发路径的控制操作。

5.3　性能测试

5.3.1　流表下发性能测试

引入 SDN 后，在 SDN 模式下，对单条流表的下发时间做了测试，验证随着数据中心网络的扩大，流表的下发效率是否会急剧下降，对数据的传输是否有较大的影响，本文分别对租户 A 和租户 B 的网络进行了测试。测试结果如图5-8所示。时延表明，随着云数据中心网络的扩大，单条流表的下发时间随之增长，但增长的趋势逐渐变缓，效率并未大幅下降，仍能满足用户对数据传输效率的要求。

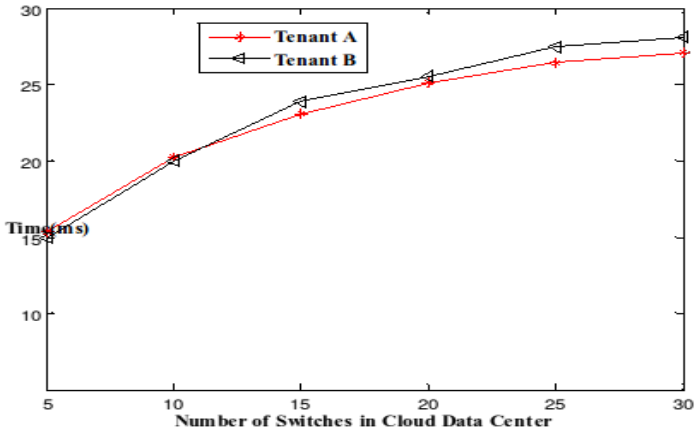


图 5-8　数据中心单条流表下发时间

5.3.2　数据传输效率测试

本文分别对 SDN 架构和传统 OpenStack 架构下的数据传输进行测试，对比引入 SDN 后，数据的传输速率是否受到了影响。本文以租户 A 的网络作为实验对象，在其中的两台虚拟机之间传输固定大小的数据包，分别测试在传统架构和 SDN 架构下的数据传输时间，数据包的大小在 200MB-600MB 之间，实验结果如图5-9所示，结果表明，引入 SDN 架构后，数据的传输速率并未得到大幅下降，仍能很好的满足用户的需求。

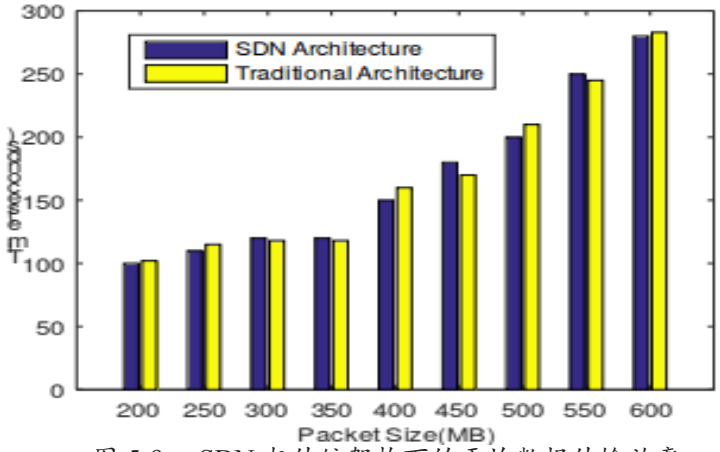


图 5-9　SDN 与传统架构下的平均数据传输效率

5.4 本章小结

本章主要是对系统进行了测试，首先介绍了实验环境，对云平台的服务器信息做了简单概述。测试主要分功能和性能测试，功能测试主要对租户的隔离性，链路带宽、时延的测量，以及链路切换进行了验证。性能测试主要包括流表的下发效率以及 SDN 模式与传统模式数据传输效率。结果表明，SDN 模式并未降低数据传输效率，能够很好的满足用户的需求。

第六章 总结和展望

6.1 论文工作总结

6.2 存在的问题及展望

附录 缩略语表

IaaS Infrastructure as a Service，基础设施即服务

OVS Open Virtual Switch，虚拟交换机
OVX OpenVirteX，网络虚拟化平台

PSO Particle Swarm Optimization，粒子群算法

SDN Software Defined Network，软件定义网络

TLS Transport Layer Security，传输层安全

vSDN virtual Software Defined Network，虚拟 SDN 网络

参考文献

[1] Porras P, Shin S, Yegneswaran V. A security enforcement kernel for Open Flow networks[A]. // The 1st Workshop on Hot Topics in Software Defined Networks[C]. New York: ACM Press, 2012: 121–126.

[2] Wang A, Iyer M, Dutta R, et al. Network Virtualization: Technologies, Perspectives, and Frontiers[J]. Journal of Lightwave Technology, 2013, 31(4): 523–537.

[3] Hua-YingChang, Shie-YuanWang. Using SDN technology to mitigate congestion in the OpenStack data center network[A]. // (ICC)[C]. IEEE, 2015: 401–406.

[4] Feller E, Rilling L, Morin C, et al. A scalable and autonomic virtual machine management framework for private clouds[A]. // Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster[C]. IEEE Computer Society, 2012: 482–489.

[5] OpenVirteX web site[EB/OL]. http://ovx.onlab.us/.

[6] Ryu web site[EB/OL]. https://osrg.github.io/Ryu/.

[7] 左青云, 陈鸣. 基于 OpenFlow 的 SDN 技术研究 [J]. 软件学报, 2013, 24(5): 1081–1083.

[8] OpenFlow setup website[EB/OL]. [4]http://www.openflow.org/wp/deploy-labsetup.

[9] Pfaff B, Pettit J, Koponen T, et al. Extending networking into the virtualization layer[A]. // Proc. of the 7th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)[C]. New York: ACM Press, 2009.

[10] Naous J, Erickson D, Covington G, et al. Implementing an OpenFlow switch on the NetFPGA[A]. // Proc. Of the 4th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)[C]. San Jose: ACM Press, 2008: 1–9.

[11] Al-Shabibi A, Leenheer M, Gerola M, et al. OpenVirteX: A Network Hypervisor[A]. // Open Networking Summit 2014[C]. 2014.

[12] OpenStack web site[EB/OL]. http://docs.openstack.org/.

[13] 北京邮电大学研究生院培养与学位办公室. 关于研究生学位论文格式的统一要求 [EB/OL]. http://www.bupt.edu.cn/, 2014–11.

[14] 北京邮电大学研究生院培养与学位办公室. 关于研究生学位论文格式的统一要求 [EB/OL]. http://www.bupt.edu.cn/, 2004.

致 谢

从 2013 年 9 月份开始，我在北京邮电大学网络技术研究院网络管理中心的研究生生活便拉开了序幕。时光如指间流沙，两年半的研究生生涯转眼间就要结束，这段充实有意义的时光让我终身难忘。在这两年半的时间里，我从一个稚嫩的大学生一步一步成长为逻辑严谨、成熟稳重的硕士研究生，即将精神饱满地进入社会的摇篮。在学习和生活中，我不仅学到了很多计算机通信相关的专业基础知识，学会了如何在科学研究领域做学术研究，如何提升自己的理论水平和实践开发能力，更懂得了许多做人做事的道理。经历过研究生生活的酸甜苦辣，在此，我想对身边的每一个老师、同学、朋友和家人表达深深的感谢，谢谢你们对我给予的帮助和鼓励。

衷心感谢我的导师——廖建新老师。在我攻读硕士期间，感谢熊老师在学习、科研和生活中给予我的帮助、鼓励与指导。熊老师认真的工作精神、高效的办事效率、严谨的治学态度和渊博的知识不断地影响到我。读研期间，在任何时候您都能给予我及时的回复，并且每次都能让我受益匪浅。同时，尽管您业务繁忙，但您仍然对我的科研进展和学习生活情况关怀备至，令我十分感动。在今后的工作中，我会以您为榜样严格要求自己，积极向上。

衷心感谢负责我实验室日常工作的指导老师——王敬宇。王老师主要负责我科研方面的指导工作，从本科生毕业设计开始到研究生各阶段，王老师一点点开启了我的学术科研之路。不论是论文和专利撰写，还是标准制定、项目调研等一系列日常科研工作，王老师都给予我悉心指导。王老师具有踏实的工作态度和亲切的处事待人方式，是我们大家学习的榜样。王老师积极组织科研小组定期进行会议讨论，检查大家的科研进度，促进沟通和交流，使得我们获得了广阔和多层次的研究视角。浓郁的学术氛围极大地加快了大家的科研工作进度，使我们受益匪浅。王老师指导我参与了多项国家科研项目的研究，一直信任我、鼓励我，在我迷茫的时候指引我正确的研究方向。王老师就像一位辛勤的园丁，又像一位贴心的母亲，从您的身上我学到了温柔的力量。

衷心感谢负责我实验室日常工作的指导老师——戚琦老师。戚老师主要负责我科研方面的指导工作，从本科生毕业设计开始到研究生各阶段，戚老师一点点开启了我的学术科研之路。不论是论文和专利撰写，还是标准制定、项目调研等一系列日常科研工作，戚老师都给予我悉心指导。戚老师具有踏实的工作态度和亲切的处

事待人方式，是我们大家学习的榜样。戚老师积极组织科研小组定期进行会议讨论，检查大家的科研进度，促进沟通和交流，使得我们获得了广阔和多层次的研究视角。浓郁的学术氛围极大地加快了大家的科研工作进度，使我们受益匪浅。戚老师指导我参与了多项国家科研项目的研究，一直信任我、鼓励我，在我迷茫的时候指引我正确的研究方向。王老师就像一位辛勤的园丁，又像一位贴心的母亲，从您的身上我学到了温柔的力量。

感谢实验室的同窗们。感谢龚军、孙海峰、王金柱、薛瑞等师兄师姐，在本科毕设和研究生前两年期间，你们帮助我走进网络管理中心的生活，让我感受到轻松愉快的学习氛围，走进科研的殿堂。感谢同组的李涛、何昱泽、陆中豪、陈良章，还有庄子睿、包剑楠、李钰剑、庞旭东、武莹等师弟师妹，有幸能和你们一起讨论学术问题，一起学习和玩耍，共同进步和成长。感谢李呈同学，虽不在同一实验室，他的博文成为我学习 SDN 的指路明灯。感谢网络管理中心的其他同窗，舍友亦或同学，大家在网管这个大家庭里相聚相知，谢谢你们的理解和支持。

衷心感谢我的父母和家人，你们是我力量的来源。陪伴你们的时间太少，你们默默奉献无私的爱和帮助，润物细无声，让我无忧无虑地完成研究生学业。感谢我的朋友们，谢谢你们给予我学习上的鼓励和帮助，以及生活上的包容和理解。

感谢相聚的时光，感谢出现在研究生岁月里的人和事。

最后，衷心感谢在百忙之中抽出宝贵时间对论文进行评阅的各位专家和老师！

脚注使用带圈数字的表示方法，此处为示例 1① 和示例 2②。参考文献可以使用[13] 和 [14] 的表示方法。

① 测试脚注一
② 测试脚注二

攻读学位期间发表的学术论文目录

期刊论文

[1] **Zhang San** , Newton I, Hawking S W, et al. An extended brief history of time[J]. Journal of Galaxy, 2079, 1234(4): 567–890. （SCI 收录，检索号：786FZ）.

会议论文

[2] McClane J, McClane L, Gennero H, et al. Transcript in Die hard[A]. // Proc. HDDD 100th Super Technology Conference (STC 2046)[C]. Eta Cygni, Cygnus: 2046: 123–456. （EI 源刊）.

专利

[3] 张三 , 李四. 一种进行时空旅行的装置 [P]. 中国: 1234567, 2046–01–09.