分类号 _____ 密级 _____

UDC _____ 编号 _____

中国科学院大学
硕士学位论文

面向媒体业务的虚拟化网络服务设计与开发

薛娇

指导教师 邓峰 研究员

中国科学院声学研究院

申请学位级别 硕士 学科专业名称 电子与通信工程

论文提交日期 2014年6月 论文答辩日期 2014年6月

培养单位 中国科学院声学研究院

学位授予单位 中国科学院大学

答辩委员会主席 _____

Typeset by LaTeX 2$_\varepsilon$ at April 17, 2014

With package `CASthesis` v0.2 of CTEX.ORG

Multimedia Service Oriented Virtualization of Network Service Design and Development

Jiao Xue

Supervisor:

Prof. Feng Deng

Institute of Acoustic
Academy of Mathematics and Systems Science
Chinese Academy of Sciences

April, 2014

Submitted in total fulfilment of the requirements for the degree of Master in Electronics and Communication Engineering

摘　　要

近年来，随着互联网络的快速发展，网民规模随之增加，互联网普及率越来越高。随着 3G 网络的进一步普及，智能手机和无线网络的持续发展，音频、视频等多媒体业务也随之大幅增长。多媒体业务的大幅增长对基础设施服务提供商提出了巨大的挑战。由于多媒体业务对 QoS 服务质量要求苛刻，如严格的带宽需求保障、较小的延时保障和严格的延时抖动限制等。因此，如何提供合适的服务平台来承载媒体业务，如何保障 QoS 服务质量等等都具有很高的研究价值和广阔的市场前景。

在传统的服务模式下，随着业务的增长和用户的增加，基础设施服务提供商常常通过部署服务器集群、增加服务器设备数量来为用户提供可靠服务。这种传统的服务模式在当今快速发展的互联网络环境下已不再适合。一方面，这种传统的服务模式给基础设施服务提供商带来了巨大的设备成本和部署压力。另一方面，这些部署的大量设备根本得不到充分利用，造成巨大的资源浪费。虚拟化技术对解决这个问题具有很大的优势，它通过建立统一的资源管控系统，管理所有的设备并进行虚拟化，按需为业务分配资源，显著提高了资源利用率，大大减少了服务器设备成本和部署开销。因此，虚拟化技术得到了越来越多的关注。

现有的虚拟化系统，大多是面向硬件资源共享。然而，在实际应用中，常常会用到许多独立的服务，如媒体业务中经常涉及的存储、转码、流化服务。如果将这些独立的服务作为基本服务单元，一方面可以使其可以更好的为不同系统提供高质量服务，另一方面将硬件资源通过多服务类型虚拟，提高了利用率，降低了部署和运营成本，应用和研究价值均较强。

因此，本文以服务虚拟化平台 SuperNova 为依托开展研究和开发工作，研究了面向媒体业务的虚拟化网络服务的关键问题和技术，包括虚拟网络嵌入、资源交互技术、业务流量隔离与控制以及服务虚拟化平台 SuperNova 的功能完善等。通过研究这些问题和技术，提出了相应的技术解决方案，解决了现有技术中的一些问题，取得了以下研究成果：

1、提出了一种面向媒体业务的虚拟网络嵌入算法 MSO-VNE 。MSO-VNE 算法根据虚拟网络请求和底层物理网络的地理位置特征，运用节点聚类算法将

虚拟网络请求聚类分割为规模较小的子请求，充分利用分而治之策略来降低虚拟网络嵌入的时间复杂度。针对所有的虚拟网络子请求，运用并发处理策略进一步降低虚拟网络嵌入处理的时间复杂度。MSO-VNE 算法通过动态服务均衡分析，充分考虑了节点在网络拓扑中的重要性。节点的重要性不仅与节点自身的可用资源有关，还与该节点的邻居节点的可用资源相关。通常，越是重要的节点具有优先的嵌入映射权利。动态服务均衡分析显著提高了虚拟网络嵌入的性能和底层物理网络的资源利用率，增加了基础设施提供商的长期收益。

2、基于面向媒体业务的虚拟网络嵌入算法 MSO-VNE，提出了一种面向媒体业务的资源映射交互框架。资源映射交互框架主要包括用户接口模块、预处理模块、资源映射模块、资源分配模块以及资源监控模块。用户接口模块是资源映射交互框架中面向用户的部分，供用户定义业务和服务请求，提交请求并获取特定的服务。预处理模块通过对用户提交的请求进行预处理，为进一步的快速处理提供保障。资源映射模块是资源映射交互框架的核心模块，它通过利用 MSO-VNE 嵌入映射算法进行资源映射匹配计算，以获得更好的 QoS 服务质量。当用户请求和底层物理网络的可用资源成功映射后，资源分配模块根据资源映射模块的映射结果进行真正的资源分配和管理。同时，资源监控模块通过实时监控底层资源池中的资源使用变化情况并进行必要的同步管理，以辅助资源映射模块和资源分配模块完成它们的工作。资源映射交互框架可有效地保障 QoS 服务质量并提高用户体验质量。

3、基于 Linux 流量控制模块 TC 设计了多层次结构的业务流量隔离与控制系统。通过虚拟局域网 VLAN 实现不可见性隔离，即虚拟局域网中的同一个 VLAN 内的成员可相互通信，而不同 VLAN 之间是相互隔离和不可见的。通过区分数据流量，并规定不同的数据流被发送的方式和顺序实现流量控制和互不影响性隔离。业务流量隔离与控制系统封装了 Linux TC 的控制命令，隐藏了实现细节，简化了用户操作，提高了用户体验。另外，通过开放一定的 API 控制接口，用户可以根据实际业务的需求对流量控制规则进行自定义重配置。业务流量隔离与控制系统可以有效地隔离不同用户之间的业务流量干扰，也为用户提供了一种区分不同业务、实现较高 QoS 服务质量的工具。

4、完善了服务虚拟化平台 SuperNova 的功能实现。服务虚拟化平台 SuperNova 的功能实现主要包括三个部分：首先是虚拟网络嵌入算法在服务虚拟化平台 SuperNova 中的工程实现；其次是业务流量隔离与控制系统在服务虚

拟化平台 SuperNova 中的工程实现；最后是为提高用户体验质量而附加的部分功能实现，包括实例管理，存储管理，安全组管理等。无论是哪个模块组件，在工程实现时都需要充分考虑可实现性和高效率性，因此，可以根据具体情况做相应的简化。譬如，对于虚拟网络嵌入算法，可通过由用户指定集群区域，从而降低聚类算法的复杂度；考虑到链路映射的复杂性，可通过弱化链路映射来提高映射效率等。服务虚拟化平台 SuperNova 最终要面向用户提供服务，因此，平台的用户体验质量是至关重要的。通过对云控制器进行可视化改进，使得用户不再需要面对非常不人性化的终端屏幕，也不再需要输入繁复易错的命令。而是通过在 Web 页面上进行直观的点选操作就可以完成同样的功能，是对用户操作的一种人性化设计，可很好地提高用户体验质量。

关键词： 虚拟化，虚拟网络嵌入，资源映射，流量隔离，流量控制，QoS，媒体业务

Abstract

In recent years, with the rapid development of the Internet, the number of Internet users and the Internet penetration have been increasing. With the sustainable development of the 3G network, smart phones and wireless network, the multimedia services, such as the audio and video business, increased greatly. The surge of multimedia services give the infrastructure service providers a huge challenge. The multimedia services have strict demanding for quality of service, for instance, the strict bandwidth requirements, the smaller delay and delay jitter. Therefore, it has the high research value and broad market prospect to research how to provide appropriate platform to host the multimedia services and how to guarantee the quality of service.

In the traditional service mode, with the growth of the business and the increase of the user number, the infrastructure service providers have to increase the quantity of server equipment to provide users reliable service. While, the traditional service mode is no longer applicable nowadays. On the one hand, the traditional service mode brings infrastructure service provides great equipment and deployment cost. On the other hand, it brings great waste of resources because we can not make full use of the deployed equipments. The virtualization technology has a great advantage to solve the problem. Actually, a unified resource control system was established and all the equipments was managed. The system allocates resource according to the demanding of the service. It is an efficient way to improve the resource utilization and decrease the cost of server equipment and deployment. As a result, the virtualization technology is gaining more and more attention.

Most existing virtualization systems are designed for sharing hardware resource. In practice, most business often use many independent services, such as the storage, transcoding, streaming in the multimedia business. We can make these independent services as basic units. On the one hand, the basic units can provide high quality service for different systems. On the other hand, the mul-

tiple service types of hardware resources can improve the utilization and reduce the cost of operation and deployment. The research has a strong application and research value.

As a result, on the basis of the service virtualization platform, SuperNova, we conducted some research and development work. In the multimedia service oriented virtualization of network service, some key problems including virtual network embedding, interactive technology, traffic control and isolation, and other programming problems were studied. Through studying these techniques, this paper proposed technical solutions to solve some problems in the existing technology. The contributions of this paper are as follows:

1, A multimedia service oriented virtual network embedding algorithm, MSO-VNE, was proposed. According to the location characteristics of the virtual network request and substrate network, the nodes clustering algorithm was applied to split the origin request into smaller ones in the MSO-VNE algorithm. The proposed algorithm make full use of the divide-and-conquer strategy to reduce the time complexity. When embedding all the virtual network sub-requests, the concurrent strategy was applied to further reduce the time complexity. Through the dynamic service balance analysis, the MSO-VNE algorithm take full consideration to the importance of nodes in the network topology. In fact, the importance of a node is not only associated with the available resources of the node itself, but also associated with the available resources of neighbor nodes of the node. In general, the more important the node has the priority right to be embedded. The dynamic service balance analysis gained a significant performance, including improving the performance of the virtual network embedding, increasing the utilization of the substrate network and increasing the revenue of the infrastructure providers in the long term.

Keywords: Chinese Academy of Sciences (CAS), Thesis, LaTeX Template

缩略语

缩略语	英文全称	中文全称
CNNIC	China Internet Network Information Center	中国互联网络信息中心
QoS	Quality of Service	服务质量
CRM	Customer Relationship Management	客户关系管理系统
EC2	Elastic Compute Cloud	弹性云计算服务
AWS	Amazon WEB Service	亚马逊网络服务
IaaS	Infrastructure as a Service	基础设施即服务
PaaS	Platform as a Service	平台即服务
SaaS	Software as a Service	软件即服务
VLAN	Virtual Local Area Network	虚拟局域网
MSO-VNE	Multimedia Service Oriented Virtual Network Embedding	面向媒体业务的虚拟网络嵌入
TC	Traffic Control	流量控制
NIST	National Institute of Standards and Technology	美国国家标准与技术学院
CAS	Could Security Aliance	云计算安全联盟
NASA	National Aeronautics and Space Administration	美国国家航空航天局
CLC	Cloud Controller	云控制器
CC	Cluster Controller	集群控制器
SC	Storage Controller	存储控制器
NC	Node Controller	节点控制器
OSI	Open System Interconnection Reference Model	开放系统互联参考模型
DHCP	Dynamic Host Configuration Protocol	动态主机配置协议

NP	Non-deterministic Polynomial	非确定性多项式
MCF	Multi-Commodity Flow	多商品流
VNE	Virtual Network Embedding	虚拟网络嵌入
VnoM	Virtual Node Mapping	虚拟节点映射
VliM	Virtual Link Mapping	虚拟链路映射
BFS	Breadth-First Search	宽度优先搜索
SN	Substrate Network	底层网络
VNR	Virtual Network Request	虚拟网络请求
GT-ITM	Georgia Tech Internetwork Topology Models	佐治亚互联网络拓扑模型
IP	Internet Protocol	网际协议
TCP	Transmit Control Protocol	传输控制协议
UDP	User Datagram Protocol	用户数据报协议
QDisc	Queuing Discipline	队列规定
FIFO	First-In First-Out	先进先出
TOS	Type of Service	服务类型
RED	Random Early Detection	随机早期探测
SFQ	Stochastic Fairness Queuing	即随机公平队列
TBF	Token Bucket Filter	即令牌桶过滤器
CBQ	Class Based Queuing	基于分类的队列
HTB	Hierarchy Token Bucket	分层的令牌桶
GM	Global Manager	全局管理服务器
Ajax	Asynchronous JavaScript and XML	异步的 Javascript 和 XML 技术
SEDA	Staged Event-driven Architecture	阶段的事件驱动架构
WSDL	Web Services Description Language	Web服务描述语言

目　　录

表　　格

插　　图

第一章　绪论

1.1　研究背景及意义

中国互联网络信息中心（CNNIC）发布的第 33 次《中国互联网络发展状况统计报告》[1] 显示，截至 2013 年 12 月，我国网民规模达到 6.18 亿，全年新增网民 5358 万人，互联网普及率为 45.8%。在 3G 网络进一步普及，智能手机和无线网络持续发展的背景下，视频、音乐等高流量应用拥有越来越多的用户。截至 2013 年 12 月，我国手机端在线收看或下载视频的用户数为 2.47 亿，与 2012 年底相比增长了 1.12 亿，增长率高达 83.8%。图片、音频和视频等多媒体业务的大幅增长对基础设施服务提供商提出了巨大的挑战。由于多媒体业务对 QoS 服务质量要求苛刻，譬如严格的带宽需求保障、较小的延时保障和严格的延时抖动限制等。因此，如何提供合适的服务平台来承载媒体业务，如何保障媒体业务的 QoS 服务质量等等都具有很高的研究价值和广阔的市场前景。

随着海量数据需求的日益增加，云计算技术以及海量数据处理技术得到越来越多的关注和重视。云计算[2, 3]可以算是继个人电脑、互联网后的又一个革命性技术，它彻底改变了我们使用信息技术的方式。时至今日，云计算的影响力已经席卷了 IT 行业的各个角落，云计算相关的产品和服务层出不穷，对人们的生活和工作产生了深远的影响。公认的云计算先驱 Salesforce[4] 通过向企业客户销售基于云的 SaaS 软件及服务产品，CRM[5]，成长为一家年销售额超过 20 亿美元的公司。 Salesforce 第一次将 SaaS 服务大规模地销售给了企业用户，进一步证明了基于云计算的产品和服务不仅仅是大型业务系统的的廉价替代品，它还可以是真正提高企业运营效率、促进业务发展的解决方案，并维持极高标准的可靠性。

在线零售商 Amazon[6] 为提高用户体验，在数据中心配置了大量冗余的服务器设备以缓冲圣诞购物季的高峰时段流量。冗余配置使得数据中心在大部分时间只有不到 10% 的资源利用率，剩下 90% 的资源都是闲置的。因此，Amazon 于 2006 年对外发布了 Amazon EC2[7] 弹性云计算服务，面向公众提供基础架构的云服务，IaaS，将闲置的服务器资源以按需付费的弹性方式开放给公众，以提高资源利用率。 Amazon EC2 是云计算的一个里程碑，它是业界

第一个将基础架构大规模开放给公众用户的云计算服务，用户可以在 Amazon EC2 平台上搭建并部署从 Linux 到 Windows 的任何业务，而且用户可根据业务的实际需求使用云服务资源并支付费用。除了 EC2 之外， Amazon 还发布了 S3[8]、SQS[9] 等云计算服务，组成了完整的 AWS 产品线，合理利用数据中心的闲置服务器资源，提高资源利用率。

继 Amazon AWS 之后，各大厂商都相继涌进云计算的大潮中，各种各样的云计算产品和服务如雨后春笋般不断涌现。 2009 年， Google 开始对外提供 Google App Engine[10, 11] 服务，通过提供完整的 Web 开发环境，用户只需要在浏览器里开发、调试自己的代码，然后部署到 Google 云平台上，就可以直接对外发布并提供服务。 Microsoft 也在云计算的浪潮中发布了自己的云计算产品 Windows Azure[12]。

云计算意味着用户不再需要购置高昂的设备，也不再需要面临各种部署压力和维护成本。用户可直接选择各种各样的云计算服务，并按业务需求支付费用。云计算将物理设备的购置、维护和部署转嫁给云计算服务提供商，尤其是基础设施云服务提供商。通过云计算服务提供商雄厚的资本实力和专业的维护、部署，提供高效率和高可靠性的云计算服务。在私有云服务中，用户可将已购置的或闲置的物理设备搭建成云计算服务平台，整合内部的各种资源和服务并进行统一管理和分配，以最大化资源的利用率。目前，构建云服务平台的虚拟化管理系统层出不穷，譬如 OpenStack[13]、Eucalyptus[14, 15]、OpenNebula[16]、CloudStack[17] 等。这些虚拟化管理系统可以辅助用户高效地搭建云计算服务平台。

目前的虚拟化管理系统大多是面向硬件资源共享的。然而，在实际应用中，常常会用到许多独立的服务，如媒体业务中经常涉及的存储、转码、流化服务。如果将这些独立的服务作为基本服务单元，一方面可以使其更好的为不同系统提供高质量服务，另一方面将硬件资源通过多服务类型虚拟，提高了利用率，降低了部署和运营成本，应用和研究价值均较强。

因此，本文面向媒体业务的虚拟化网络服务设计与开发，以服务虚拟化平台 SuperNova 为依托开展研究和开发工作，研究了面向媒体业务的虚拟化网络服务的关键问题和技术，包括虚拟网络嵌入、资源映射交互技术、业务流量隔离与控制以及服务虚拟化平台 SuperNova 的功能完善等。

1.2　研究内容及面临的挑战

在传统的服务模式下，随着业务的增长和用户的增加，基础设施服务提供商常常通过部署服务器集群、增加服务器设备数量来为用户提供可靠服务。这种传统的服务模式在当今快速发展的互联网络环境下已不再适合。一方面，传统的服务模式给基础设施服务提供商带来了巨大的设备成本和部署压力。另一方面，这些部署的大量设备根本得不到充分利用，造成巨大的资源浪费。虚拟化技术对解决这个问题具有很大的优势，它通过建立统一的资源管控系统，管理所有的设备并进行虚拟化，按需为业务分配资源，显著提高了资源利用率，大大减少了服务器设备成本和部署开销。

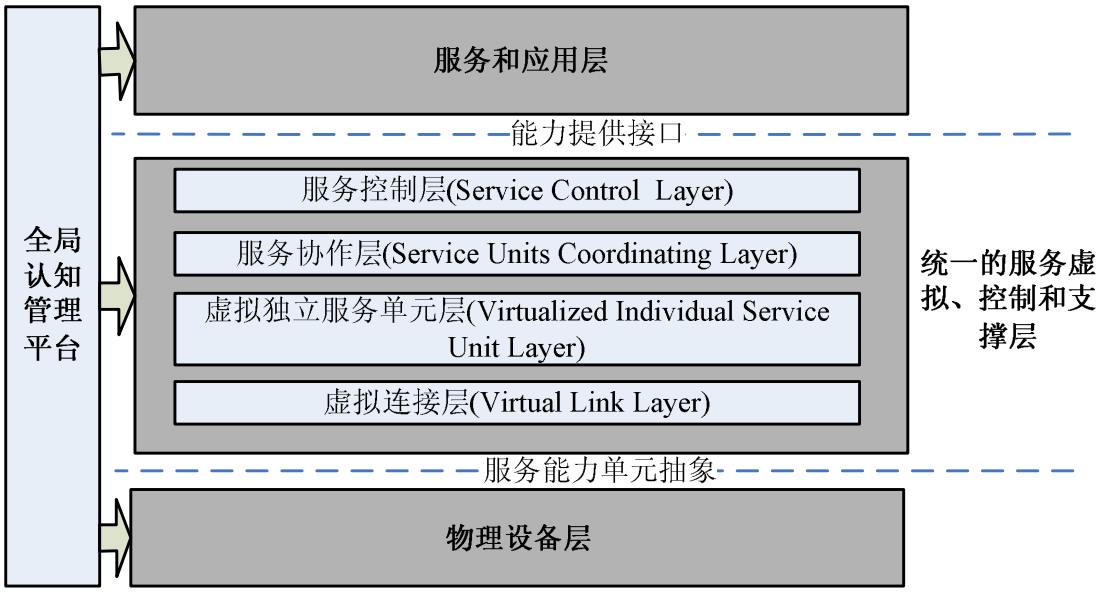

图 1.1: SuperNova系统架构示意图

本文以 863 项目“融合网络业务体系的开发”中服务虚拟化平台 SuperNova 为依托开展研究和开发。服务虚拟化平台 SuperNova 的架构示意图如图1.1所示。服务虚拟化平台 SuperNova 提供一种基于服务虚拟化、具有云服务特征、开放接口的网络系统及实现方法。通过将分布在不同地理位置的物理服务器设备连接起来，采用虚拟化技术以大粒度服务的形式对外提供资源，实现多种业务共享硬件资源，并满足用户对媒体业务的服务质量要求。服务虚拟化平台 SuperNova 将物理设备层屏蔽起来，根据上层应用的业务需求，以接口的方式为上层提供服务，应用无需了解任何网络细节或者硬件情况，可根据业

务需求获得弹性服务。按照功能划分，服务虚拟化平台包括：虚拟连接层、虚拟独立服务单元层、服务协作层、服务控制层和全局认知管理层。

本文在服务虚拟化平台 SuperNova 上开展研究和开发工作，并主要集中在服务控制层和全局认知管理层。本文的研究内容及面临的挑战可归结为以下几个方面：

1）虚拟网络嵌入

虚拟网络嵌入是网络虚拟化的关键内容，通过在共享的物理网络上实例化虚拟网络请求，有效地将虚拟网络请求拓扑中的虚拟节点和虚拟链接嵌入映射到底层物理网络上，最大化从底层物理网络中获得的收益，获取最佳的动态资源分配，从而为最终用户提供定制的端到端保证的服务。虚拟网络嵌入问题是 NP 难问题，其时间复杂度非常高。因此，现有的研究大多都侧重于设计基于启发式的算法来解决。然而，现有的启发式算法均不适合服务虚拟化平台，它们没有充分利用服务虚拟化平台 SuperNova 的独有特征。因此，在服务虚拟化平台 SuperNova 中，设计符合平台特性的虚拟网络嵌入算法是需要考虑的问题。如何对虚拟网络嵌入问题进行建模；如何挖掘服务虚拟化平台 SuperNova 的独有特征；如何减少虚拟网络嵌入的时间复杂度以及如何提高底层物理网络的资源利用率等都是值得研究的问题。

2）业务流量隔离与控制

业务流量隔离与控制系统对于提高业务服务的可靠性，防止数据流量串扰，提高业务的服务质量有着非常重要的作用。它通过利用软件或硬件工具，实现对不同业务的网络流量的隔离与控制，消除不同业务以及不同流量之间的干扰。流量隔离与控制最主要的方法是引入 QoS，通过为不同类型的网络数据包作标记，并决定不同标记的数据包通行的方式以及优先次序。目前，主要的流量控制方式包括流量整形、流量调度、流量监管和丢弃。在服务虚拟化平台 SuperNova 中，汇集了众多的服务器资源，并支撑着大量不同的业务。因此，如何设计业务流量隔离与控制系统是很关键的问题。业务流量隔离和控制系统采用什么样的架构；采用什么流量控制工具；如何划分不同的流量以及如何控制流量等都是值得研究的问题。

3）服务虚拟化平台功能实现

不管是虚拟网络嵌入还是业务流量隔离与控制系统，其理论研究和工程实现都有着巨大的差距。通常，理论研究更注重准确性和高性能，而工程开发和

应用实现更加注重可行性和高效率。因此，如何在服务虚拟化平台 SuperNova 上高效率地实现高性能的虚拟网络嵌入算法是值得研究的问题。具体而言，包括如何对虚拟网络嵌入算法进行简化，怎样进行工程开发和细节简化，以及需要在哪些方面进行折衷等等都是需要仔细推敲和琢磨的。同样，在服务虚拟化平台 SuperNova 上如何实现业务流量隔离与控制系统也需要进行相应的折衷考虑和工程开发设计。另外，作为对外开放使用的服务虚拟化平台 SuperNova，用户体验质量是至关重要的。因此，如何进行呈现设计，开放给用户什么样的 API 接口，如何更好地提高用户体验质量都是值得研究和考虑的问题。

1.3　本文主要贡献

本文面向媒体业务的虚拟化网络服务设计与开发的主要贡献可归结为以下几点：

1、面向媒体业务的虚拟网络嵌入算法 MSO-VNE

本文结合服务虚拟化平台 SuperNova 的应用场景和平台特性，设计并提出了面向媒体业务的虚拟网络嵌入算法 MSO-VNE。MSO-VNE 算法根据虚拟网络请求和底层物理网络的地理位置特征，应用节点聚类算法，充分利用了分而治之思想和并发处理思想，显著降低了虚拟网络嵌入处理的时间复杂度。MSO-VNE 算法充分考虑了节点在网络拓扑中的重要性，通过动态服务均衡分析，显著提高了虚拟网络嵌入的性能和底层物理网络的资源利用率，增加了基础设施提供商的长期收益。

2、面向媒体业务的资源映射交互框架

本文基于面向媒体业务的虚拟网络嵌入算法 MSO-VNE，提出了面向媒体业务的资源映射交互框架。资源映射交互框架主要包括用户接口模块、预处理模块、资源映射模块、资源分配模块以及资源监控模块。其中，资源映射模块是核心模块，它通过利用 MSO-VNE 嵌入映射算法进行资源映射匹配计算，以获得更好的 QoS 服务质量。资源映射交互框架可有效地提高用户体验质量。

3、业务流量隔离与控制系统

本文基于 Linux 流量控制模块 TC 设计了多层次结构的业务流量隔离与控制系统。通过虚拟局域网 VLAN 实现不可见性隔离，即虚拟局域网中的同一个 VLAN 内的成员可相互通信，而不同 VLAN 之间是相互隔离和不可见的。通过区分数据流量，并规定不同的数据流被发送的方式和顺序实现流量控制和互不

影响性隔离。业务流量隔离与控制系统封装了 Linux TC 的控制命令，隐藏了实现细节，简化了用户操作，提高了用户体验。另外，通过开放一定的 API 控制接口，用户可以根据实际业务的需求对流量控制规则进行自定义重配置。业务流量隔离与控制系统可以有效地隔离不同用户之间的业务流量干扰，也为用户提供了一种区分不同业务、实现较高 QoS 服务质量的工具。

4、服务虚拟化平台的工程开发

服务虚拟化平台的工程开发主要包括三个部分：首先是虚拟网络嵌入算法在服务虚拟化平台中的工程实现；其次是业务流量隔离与控制系统在服务虚拟化平台中的工程实现；最后是为提高用户体验质量而附加的部分功能实现，包括实例管理，存储管理，安全组管理等。无论是哪个模块组件，在工程实现时都充分考虑了可实现性和高效率性，以达到较好的用户体验质量。

1.4 本文内容安排

本文共分为六章，对面向媒体业务的虚拟化网络服务进行研究和开发，其中第三、四、五章是本文的核心研究内容。本文各章的内容概述如下：

第一章为“绪论”，介绍了本文的研究背景、研究意义、研究内容及面临的挑战和主要贡献。

第二章为“相关技术背景及技术基础”，介绍了面向媒体业务的虚拟化网络服务相关的技术，包括云计算技术、虚拟化技术以及网络流量控制技术，这些相关技术为本文的研究做了必要的准备和技术支持。

第三章为“面向媒体业务的虚拟网络嵌入研究”。本章通过研究面向媒体业务的虚拟化平台的特性，提出了面向媒体业务的虚拟网络嵌入算法 MSO-VNE，并通过仿真实验验证了该算法的性能。另外，本章基于 MSO-VNE 算法，设计了面向媒体业务的资源映射交互框架，以简化资源映射交互流程并提高用户体验质量。

第四章为“业务流量隔离与控制”，解决了业务之间的流量干扰问题，实现了业务流量的隔离和控制。本章基于 Linux TC 流量控制模块，设计了业务流量隔离和控制框架，并设计仿真实验验证了其隔离和流量控制性能。

第五章为“ SuperNova 系统工程开发”。本章详细阐述了在服务虚拟化平台 SuperNova 上实现流量控制模块、虚拟网络嵌入模块、云控制器模块的方案设计与编程开发。

第六章为“总结和展望”，总结了本文的研究工作，并指出了面向媒体业务的虚拟化网络服务的进一步研究方向。

第二章　相关技术背景及技术基础

2.1　引言

本章将重点介绍虚拟化网络服务的相关技术背景和技术基础，主要包括云计算技术、虚拟化技术以及网络流量控制等相关技术。本章内容安排如下：第 2.2 节详细介绍了云计算相关的概念和技术；第 2.3 节介绍了虚拟化技术，包括单机虚拟化技术和虚拟连接技术；第 2.4 节简要介绍了网络流量控制相关的技术。

2.2　云计算

什么是云计算？这是一个被反复提到、反复回答之后，又反复提出的问题。这种有趣的现象说明云计算是有着强大生命力的技术和发展方向。由于云计算本身是一个非常抽象的概念，要想给出云计算的精确定义是非常困难的。不同的人从不同的角度对云计算的含义给出了解释，而这些解释都各有各的道理。文献 [18, 19] 认为云计算的目标是通过互联网提供各种计算服务和存储服务。云计算安全联盟（CAS）在文献[20]中比较精确地说明了云计算的本质：

“云计算的本质是一种服务提供模型，通过这种模型可以随时、随地、按需地通过网络访问共享资源池的资源，这个资源池的内容包括计算资源、网络资源、存储资源等，这些资源能够被动态地分配和调整，在不同用户之间灵活地划分。凡是符合这些特征的 IT 服务都可以称为云计算服务。”

美国国家标准与技术学院（NIST）提出了一个定义云计算的标准 [21]，认为一个标准的云计算需要具备五个基本元素：通过网络分发的服务、自助服务、可衡量的服务、资源的灵活调度和资源池化。该标准按照服务类型将云计算分为 IaaS、SaaS 和 PaaS 三类；按照部署模式将云计算分为公有云、私有云、混合云和社区云四种。该标准定义的云计算标准展示了如图 2.1 所示的云计算架构。事实上，NIST 提出的云计算标准被业界普遍接受，其中提出的云计算五大要素非常简练地说明了一个云计算系统的特征，只有同时具备这五点的 IT 架构才可以被称为云服务架构，通过这五个特征能够快速地将云计算系统同传统 IT 系统区分开来。

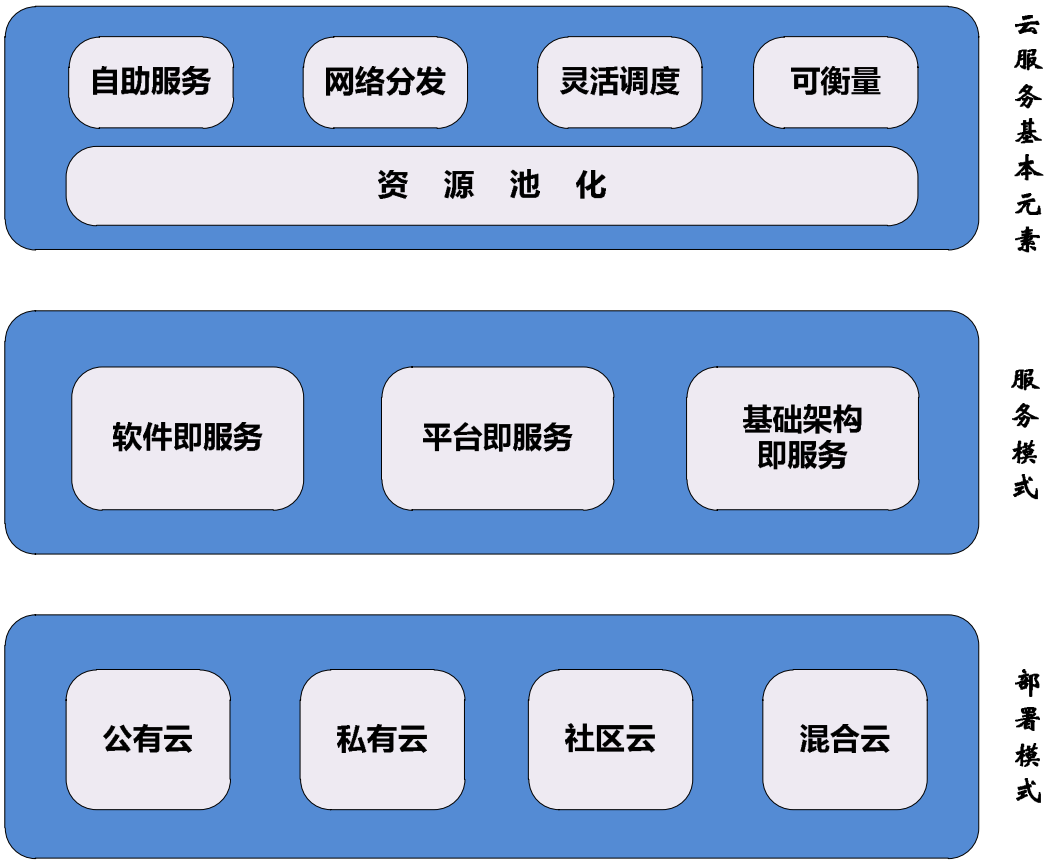

图 2.1: NIST 提出的云计算架构

1、云服务的基本元素

1）自助服务

云计算与传统架构的区别首先在于用户获取服务的方式不同。在云计算系统中，用户通过自助方式获取服务：自助地挑选服务类型，自助地配置服务，自助地开启或关闭服务。整个过程由用户自助完成，无需云服务提供商的人工干预。自助式的服务方式，一方面充分发挥了云计算后台架构强大的运算能力和服务能力；另一方面，用户可更加快捷和高效地获取服务，大大提高用户的体验质量。

2）通过网络分发服务

在云计算服务出现之前，人们使用计算机完成工作的前提是有计算机硬件设备，并在硬件设备上安装了相应的软件程序才能完成相应的工作。这种单一和死板的一对一服务获取方式严重阻碍了计算机能力的最大发挥，不仅限制了

人们对服务的获取，还造成了资源的浪费。云计算通过网络分发服务的方式打破了传统方式的限制，包括地理位置的限制、硬件部署环境的限制。只要有网络就有云计算，革命性地改变了我们使用计算机以及获取服务的方式。

3）资源池化

传统架构中，计算资源以单台服务器为单位。因此，服务器的资源大多时候处于闲置状态，不能得到合理和充分的利用。对用户而言，服务器的设备成本、部署压力等开销都非常大。云计算的资源池化打破了服务器机箱的限制，将所有的 CPU 和内存等资源释放出来，汇集到一起，形成一个个 CPU 池、内存池、网络池。当用户产生需求时，便从各个资源池中配置能够满足需求的资源组合。资源的池化使得用户不再关心计算资源的物理位置以及存在形式，可以以更加灵活的方式使用资源。

4）资源的灵活调度

资源的灵活调度是资源池化的进一步延伸。云计算在资源调度方面的灵活性表现在云计算服务提供商可以非常快速地对资源池进行变更，包括往资源池添加新设备或者从资源池中撤出旧设备。随着业务的增长和用户的增加，只需要简单地往资源池中添加新设备就可以满足用户不断增长的需求。对用户来说，只需要根据自己的业务需求消费相应的资源，并按使用的资源多少付费。

5）可衡量的服务

完整的云计算平台对存储、CPU、带宽等资源都保持实时跟踪，并将这些信息以可量化的指标反映出来。基于这些指标，云计算服务提供商或管理企业内部私有云的 IT 部门，能够快速地对后台资源进行调整和优化。对用户来说，根据业务需求，购置可衡量的服务，并支付相应的费用，显著降低了成本。

2、云计算的服务类型

1）IaaS

IaaS，Infrastructure as a Service，基础架构即服务。通过虚拟化技术将服务器等计算平台同存储、网络资源打包，通过开放的 API 接口提供给用户使用。与传统架构相比，用户不用租借机房，不用维护服务器和交换机，只需要购买与业务适配的 IaaS 基础设施服务就能够获得这些资源。著名的 IaaS 是Amazon EC2 。

2）PaaS

PaaS，Platform as a Service，平台即服务。PaaS 构建在 IaaS 之上，除了提供基础架构外，还提供业务软件的运行环境，以及相应的存储接口 API 供用户调用。PaaS 面向的用户是没有能力或者不愿意维护一个完整运行环境的开发人员和企事业单位，通过使用 PaaS 服务，他们可以从繁琐的环境搭建中解放出来，将更多的时间和精力投入到业务软件的开发中。著名的 PaaS 包括 Google App Engine 和 Microsoft Azure 等。

3）SaaS

SaaS，Software as a Service，软件即服务。SaaS 通过将业务运行的后台环境放入云端，通过瘦客户端，譬如Web浏览器，向终端用户直接提供服务。终端用户直接向云端请求服务，而本地无需维护任何基础架构或软件运行环境。SaaS 面向的用户是软件产品的终端用户。著名的 SaaS 是 Salesforce 的云端 CRM 服务。

由此可知，IaaS 处于最底层，提供给用户可直接访问的底层计算资源、存储资源和网络资源。PaaS 基于 IaaS 实现，提供的是软件业务运行的环境。SaaS 位于 PaaS 之上，将软件以服务的形式通过网络提供给终端用户。这三者的关系如图 2.2 所示。

3、云计算的部署模式

1）私有云

私有云是部署在企业内部，服务于内部用户的云计算类型。私有云的建设、运营和使用都在某个组织或企业内部完成。私有云的服务的对象被限制在这个组织内部，外部用户无法访问私有云，没有对外的公开接口。私有云不对组织外部的用户提供服务，但是私有云的设计、部署与维护可以交由组织外部的第三方完成。

2）社区云

社区云是由多个有共同利益关系或目标的企业和组织共同构建的云计算业务，其服务对象是这几个组织的内部人员。譬如由大学等教育机构维护的教育云就是一个标准的社区云业务，大学和其他高等机构将自身的教育资源放到云平台上，向校内外的用户提供服务。

3）公有云

公有云一般是由云服务运营商搭建，面向公众的云计算类型。公有云通常由一个云服务运营商维护，通过互联网向公众提供服务，任何人都可以申请、

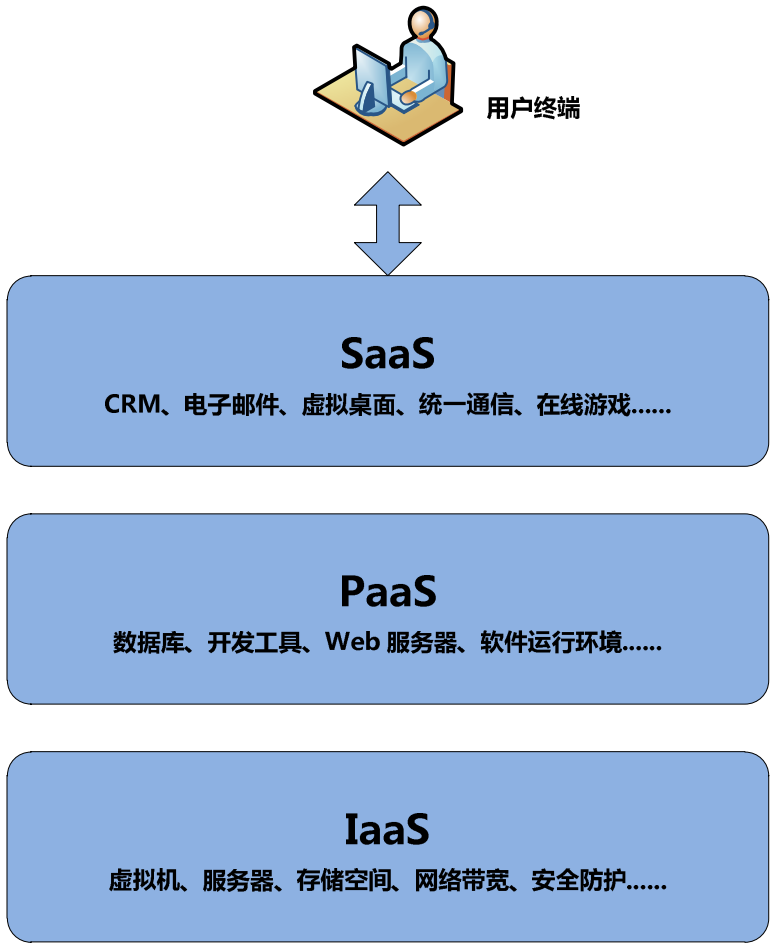

图 2.2: IaaS、PaaS 和 SaaS 的关系

使用公有云资源。因此，公有云的规模一般很大，对可靠性、安全性的要求也更高，其基础架构的组成往往也更加复杂。

4）混合云

混合云则是包含了两种或两种以上类型的云计算形式。混合云可以是公有云与私有云的混合，也可以是私有云与社区云的混合。混合云服务的对象非常广泛，包括特定组织内部的成员，以及互联网上的开放受众。

4、云计算的优势

和传统的 IT 架构相比，云计算具有鲜明的特点和独特的优势，主要包括：

1）低成本

低成本是云计算服务的最显著的特点和优势。在云计算以前，搭建 IT 系统需要购置服务器、网络、存储等设备。在云计算时代，用户不再需要购置任何硬件设备，只需按照业务需求租用云端服务，按需付费。因此，用户租用的 IT 资源随着业务需求的增长而增长、减少而减少，大大降低了成本和开销。

2）高扩展性

高扩展性是云计算架构的另一个重要特点。传统的 IT 架构扩展起来非常困难，任何细小的扩张都会牵动整个系统，不但要购置大量设备，还要兼顾到和原有系统的兼容性等问题。相反，云计算由于其资源池化和自助式服务特点，用户可以随时随地增减计算资源，且对上层业务的影响也被限制在最小的范围。

3）高可靠性

云计算的高可靠性是由云计算服务提供商的高额投资保障的。数据中心的可靠性与投资一般成正比关系：配置了备用电源的机房肯定比普通机房可靠，而在不同物理地点部署备份机房可以提供的灾难恢复时间一定比单一的数据中心短。然而，并不是每个企业都有能力为数据中心做高额的投资。云计算服务提供商为成千上万的企业提供云计算服务，其数据中心的配备和投资是单个企业无法企及的。

4）远程访问

云计算通过网络向用户提供服务，因此，不管用户身处何处，只要有网络遍布，便可对云端服务进行远程访问，非常便捷。

5）高服务质量

与高可靠性一样，云计算服务的高服务质量也是由云计算服务提供商的高额投资保障的。

5、云平台

云平台，又称虚拟化管理系统，通常指的是云计算服务中的基础设施即服务。利用云平台可以构建 IaaS 系统，为用户提供基础设施云服务。本节将重点介绍两个目前广泛使用的云计算平台：OpenStack 和 Eucalyptus 。

OpenStack 是由美国国家航空航天局（NASA）和 Rackspace 合作研发的，以 Apache 许可证授权的自由软件和开放源代码项目。OpenStack 提供了一个部署云的操作平台和工具集，它是与 Amazon EC2 兼容的 IaaS 系统，由一系列松散耦合的模块和组件组成，主要包括：

1）OpenStack Compute（Nova）

OpenStack Compute[22] 是 OpenStack 的云控制器。云控制器可以辅助用户进行业务服务的部署，包括启动虚拟实例、管理网络、控制外界对服务的访问等。OpenStack Compute 构建在底层开源项目 Nova[23] 上，包含 Web 前端、计算服务、存储服务、身份认证服务、存储块设备服务、网络服务、任务调度等多个模块。OpenStack Compute 的不同模块之间不共享任何信息，通过消息传递进行通讯。因此，不同的模块可以运行在不同的服务器上，也可以运行在同一台服务器上。

2）OpenStack Object Storage（Swift）

OpenStack Object Storage[24] 是可扩展的对象存储系统，通过内置冗余和容错机制保证数据的安全性。Swift 对象存储系统支持多种应用，包括复制和存档数据、图像、视频服务，存储静态数据，开发数据存储整合的应用，为 Web 应用创建基于云的弹性存储等。需要注意的是，Swift 对象存储系统不是文件系统，不能保证数据的实时性。因此，它更适用于存储需要长期保存的静态数据。

3）OpenStack Image Service（Glance）

OpenStack Image Service[25] 是 OpenStack 的镜像服务系统，其主要功能是管理硬盘和虚拟机镜像，可利用 OpenStack 对象存储机制来存储镜像，也可利用 Amazon S3 简单存储解决方案直接存储信息，或者将 Amazon S3 存储与对象存储结合起来，作为 Amazon S3 访问的连接器。Glance 镜像服务系统支持多种虚拟机镜像格式，譬如 VMware 镜像 VMDK ， Amazon 镜像 AKI、ARI、AMI 以及 VirtualBox 支持的各种磁盘格式。

4）OpenStack Identity（Keystone）

OpenStack Identity[25] 是 OpenStack 的认证服务系统，提供身份认证服务和相关的权限管理服务。

5）OpenStack Dashboard（Horizon）

Dashboard[26] 是 OpenStack 中提供查看和管理计算、存储、网络资源的图形界面，可简化用户操作，提供用户体验质量。

Eucalyptus，即 Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems，是一个与 Amazon EC2 兼容的私有云服务 IaaS 系统。Eucalyptus 可帮助用户完成对各种基于物理设施的虚拟设备的全局性

掌控，完成整个集群计算能力的动态配置。Eucalyptus 主要包括三个层次的组件，组件间使用安全 SOAP 消息相互通信并保持协作，共同提供所需的云服务。

1）云层次

云层次[27]的组件包括 CLC 云控制器和 Walrus 存储器。其中，CLC 云控制器是整个 Eucalyptus 系统的核心，负责管理整个系统，是所有用户和管理员进入 Eucalyptus 云的主要入口。所有客户机通过基于 SOAP 或 REST 的 API 与 CLC 通信，由 CLC 负责将请求传递给正确的组件，并将这些组件的响应信息反馈给客户机。

Walrus 是一个与 Amazon S3 类似的存储服务，主要用于存储虚拟机映像和用户数据。Eucalyptus 将虚拟机映像文件存储在 Walrus 上。当用户启动一个虚拟机实例的时候，Eucalyptus 首先将相应的虚拟机映像从 Walrus 拷贝到将要运行该实例的计算节点 NC 上。当用户关闭或重启一个虚拟机实例的时候，对虚拟机所做的修改并不会被写回到 Walrus 上原来的虚拟机映像上，所有对该虚拟机的修改都会丢失。如果用户需要保存修改过的虚拟机，就需要将该虚拟机实例保存为新的虚拟机映像。如果用户需要保存数据，则需要利用存储服务器所提供的弹性块设备来完成。

2）集群层次

集群层次[28]的组件包括 CC 集群控制器和 SC 存储控制器。CC 是一个集群的前端，负责协调集群内的计算资源，并且管理集群内的网络流量。因此，CC 负责维护运行在系统内的计算节点 NC 的全部实例信息，并控制实例的生命周期。通常，启动虚拟实例的请求会被 CC 路由到具有可用资源的 NC 上。SC 存储控制器是一个与 Amazon EBS 类似的存储块设备服务，可以用来存储业务数据。

3）节点层次

节点层次[29]的组件包括 NC 节点控制器。NC 是最终的计算节点，通过调用操作系统层的虚拟化技术来启动和关闭虚拟机。在同一个集群内的所有计算节点必须在同一个子网内。在一个集群内通常需要部署一台存储服务器，为该集群内的计算节点提供数据存储服务。Eucalyptus 通过 Agent 代理的方式来管理计算资源。在每个计算节点 NC 上，都需要运行一个 eucalyptus-nc 的服务。该服务在集群控制器 CC 上注册后，云控制器 CLC 即可通过集群控制器 CC 将

需要运行的虚拟机映像文件拷贝到该计算节点 NC 上运行。

2.3　虚拟化技术

虚拟化是一个广泛而抽象的概念。本质上讲，虚拟化都是对底层物理资源进行抽象，形成资源池，并进行按需重新分配。虚拟化技术的主要优势在于资源的共享性和隔离性，多个用户可以共享同一个底层物理资源，且每个用户在使用虚拟资源时又感觉到自己对资源的独占性和隔离安全性。本节将主要介绍单机虚拟化技术和虚拟连接技术。

1、单机虚拟化技术

单机虚拟化技术通过对底层的硬件资源进行虚拟化，以虚拟机的方式提供给用户。因此，用户可以无缝地使用底层的硬件资源，同时屏蔽底层的实现细节。目前的单机虚拟化技术主要包括：

1）完全虚拟化

完全虚拟化通过 Hypervisor 或 Virtual Machine Monitor[30] 软件平台来实现对虚拟机的管理。在完全虚拟化中，Hypervisor 可以看做是运行在底层物理操作系统上的应用软件，它可以构建出一整套虚拟硬件平台，包括 CPU、内存、存储以及适配器等。在 Hypervisor 上，再根据用户需求安装新的操作系统和需要的应用软件，这样宿主主机的操作系统和客户机的操作系统是完全无关化的。完全虚拟化中，虚拟机的应用程序调用硬件资源时需要经过虚拟机内核，再经过 Hypervisor，再经过宿主主机内核，最后到达底层物理硬件，因此完全虚拟化导致其性能的损失很大。

2）OS 层虚拟化

OS 层虚拟化，又称 Single Kernel Image（SKI）或者 Container-Based Virtualization[31] 。这种虚拟化技术不是虚拟化硬件而是虚拟化操作系统，通过在宿主操作系统中模拟出多个容器，以运行宿主操作系统的多个虚拟机实例。由于所有的虚拟机实例共享内核空间，OS 层虚拟化的性能非常好，效率非常高，出现错误的几率也很小。当然，其缺点也是显而易见的，即虚拟机的操作系统单一，虚拟出来的客户机操作系统跟宿主操作系统是一模一样的。

3）硬件层的虚拟化

硬件层的虚拟化通过直接在硬件之上运行虚拟机，虚拟机上再安装操作系统。这种方式隔离性非常好，并且性能出色，通常用在服务器之上。

4）准虚拟化

准虚拟化，又称半虚拟化，Para Virtualization 。这种虚拟化技术使得客户机操作系统必须进行修改，否则不能运行在虚拟环境中。准虚拟化是完全虚拟化的子集，在硬件和修改后的客户机操作系统之间提供软件接口子集。这种虚拟化能够达到很高的性能，非常接近物理硬件，但是其代价是安全性。Xen[32] 是准虚拟化的一个很好的例子。

5）软件虚拟化

软件虚拟化通过运行服务器程序来操作本地资源，模拟虚拟机。

2、虚拟连接技术

虚拟连接技术可以对虚拟机进行连接和路由，目前的虚拟连接方法都是通过在虚拟机上模拟出真实的网络设备来实现的。主要有以下几种方式：

1）桥接方式

桥接方式依据 OSI 网络模型中链路层的地址，对网络数据包进行转发。通常，网桥和交换机都具有桥接功能，被用来隔离不同的局域网。虚拟机的网卡可以通过静态配置或者 DHCP ，分配到一个网络中独立的 IP ，并桥接到主机的物理网卡上形成网桥，所有发送到虚拟机的数据包将经过这个网桥到达虚拟机。因此，虚拟机的所有网络功能和在网络中的真实机器完全一样。

2）网络地址转换方式

网络地址转换方式通过将私有或保留地址转化为合法的 IP 地址。它被广泛应用于各种类型 Internet 接入方式和各种类型的网络中。通过网络地址转换，可以将每个虚拟连接的 IP 地址和端口号映射到真实主机的 IP 地址和随机端口号，所有的数据包在发送或接收时都要再经过修改。在这种连接方式中，连接只能由虚拟机发起且只存在于真实机器和虚拟机之间，虚拟机之间是不可见的。

3）内网方式

内网方式是在同一主机的虚拟机之间形成一个局域网，与外界完全断开。主机与虚拟机处于不同网段，也不能相互连接。

4）主机方式

主机方式在主机中模拟出一张专供虚拟机使用的网卡，所有虚拟机都是连接到该网卡上的，可以通过设置这张网卡来扩展很多功能，比如网卡共享、网卡桥接等。

2.4　网络流量控制

网络流量控制是一种利用软件或硬件的方式来实现对计算机网络流量的控制。它的最主要方法是引入 QoS，通过为不同类型的网络数据包标记，并决定不同标记的数据包通行的优先次序。一般来说，进行网络流量控制的措施主要包括以下几种：

1、流量整形

流量整形是一种主动调整流量输出速率的控制方式。流量整形通过主动限制网络连接向外发送的数据流量，有效地控制网络连接的传输速率，使得该网络连接的报文以比较均匀的速度向外发送。流量整形可以很好地平滑突发数据流量，使网络更加稳定。通常，可利用缓冲区和令牌桶来完成流量整形：当报文的发送速度过快时，首先在缓冲区中缓存数据报文，在令牌桶的控制下再均匀地发送这些被缓冲的数据报文。流量整形的缓存能够对数据包流量的完整性有较好的保存，但缓存也引入了延迟。通常，流量整形只适用于从网络接口向外发送的流量。

2、流量调度

流量调度通过在输入接口和输出接口之间设定一个数据包队列，让数据包在队列中排队，并重新规定数据包被发送的方式和次序。通过调度数据包的传输，可以在带宽范围内，按照优先级为数据包分配带宽。常见的调度策略是 FIFO 先进先出，这种调度策略几乎不用对数据包做任何处理，数据包按照它们到达的顺序依次发送。与流量整形类似，流量调度只适用于从网络接口向外发送的流量。

3、流量监管

流量监管，类似于流量整形，流量监管不仅可以限制从网络接口向外发送的数据流量，还可以限制流入网络接口的流量。流量监控的控制策略主要是丢包和重新标记，它不存在缓冲区或队列。当某个连接的报文流量过大时，流量监管通常是直接丢弃超额流量或是将超额流量标记为低优先级。因此，流量整形和流量监管的重要区别是，流量整形可能会因为对数据包流量的缓存而增加延迟，而流量监管几乎不引入额外的延迟。

4、丢弃

丢弃策略很简单，通过丢弃包、流量或分类来实现对网络流量的控制，通常在流量超过某个设定的带宽时就会采取丢弃措施。丢弃策略不仅可以对从网

络接口向外发送的数据流量实施管理，还可以管理并丢弃流入网络接口的数据流量，从而实现对网络流量的控制和平滑。

2.5 小结

本章介绍了虚拟化网络服务的相关技术背景和技术基础，主要包括云计算技术、虚拟化技术和网络流量控制等相关技术。云计算是有着强大生命力的技术和发展方向，它有着鲜明的特点和独特的优势，包括低成本、高可靠性、高扩展性以及高服务质量。云计算的技术基础是虚拟化技术。虚拟化技术通过对底层物理资源抽象并形成资源池，按需为业务分配资源。虚拟化技术的优势在于资源共享和业务隔离。业务隔离的关键内容是流量的隔离，网络流量控制可很好地实现流量数据之间的隔离和控制。

第三章　面向媒体业务的虚拟网络嵌入研究

3.1　引言

云端应用服务的日益扩张和海量数据需求的日益增长对网络架构提出了新的需求。而传统网络的僵化结构使得它难以变更和演进，无法满足新兴业务的需求。网络虚拟化技术被认为是克服传统网络僵化问题，构建未来网络的关键技术。网络虚拟化技术允许多个异构的虚拟网络在一个共享的底层物理网络上共存。类似于服务器虚拟化技术让多个虚拟机实例宿住在同一台物理主机上，网络虚拟化技术允许多个异构的虚拟网络架构在同一个共享的物理网络上运行而互不影响。各个虚拟网络的运作是相互独立和透明的，就好像只有这一个虚拟网络在物理网络上运行一样。

虚拟网络嵌入 VNE 问题，是网络虚拟化必须要解决的关键问题。虚拟网络嵌入通过在共享的物理网络上实例化虚拟网络，有效地将虚拟网络请求拓扑中的虚拟节点和虚拟链接嵌入映射到底层物理网络上，最大化从底层物理网络中获得的收益。根据业务需求提供动态资源分配，为最终用户提供定制的端到端保证的服务。虚拟网络嵌入问题是 NP 难问题，其时间复杂度非常高。因此，现有的研究大多都侧重于设计基于启发式的算法来解决。

本章针对媒体业务的 QoS 需求特征，结合底层物理网络和虚拟网络请求的地理位置分布属性，提出了面向媒体业务的虚拟网络嵌入算法，MSO-VNE 。基于面向媒体业务的虚拟网络嵌入算法 MSO-VNE，本章设计并提出了面向媒体业务的资源映射交互框架，以提高用户体验质量。

本章的内容安排如下：第 3.2 节介绍了虚拟网络嵌入的相关的研究工作和研究进展；第 3.3 节详细阐述了面向媒体业务的虚拟网络嵌入算法的设计思想和技术细节；第 3.4 节对 MSO-VNE 算法进行仿真验证以评估该算法的性能；第 3.5 节详细描述了基于 MSO-VNE 算法的面向媒体业务的资源映射交互框架的整体架构和各个组件模块的详细设计；最后，第 3.6 节对本章的内容进行总结。

3.2 相关研究

虚拟网络嵌入要解决的问题是在底层物理网络中搜寻合适的拓扑承载虚拟网络请求。因此，虚拟网络嵌入问题可分为两个子问题：虚拟节点映射问题和虚拟链接映射问题。虚拟节点映射，即将虚拟节点嵌入映射到底层物理节点上，由物理节点为虚拟节点提供必要的物理资源。虚拟链路映射，即将连接虚拟节点的虚拟链路嵌入映射到底层物理网络中连接对应节点的路径上。

多路分流问题[33]是典型的 NP 难问题，而虚拟网络嵌入 VNE 问题与多路分流问题密切相关，因此，虚拟网络嵌入 VNE 问题也是典型的 NP 难问题。在虚拟网络嵌入问题中，即使是所有的虚拟节点都被成功映射后，虚拟链路的映射会退化为不可分流问题[34, 35]，这仍然是 NP 难问题。因此，现有的研究大多都致力于设计基于启发式的算法来解决 VNE 问题。譬如，文献[36]提出了一种路径可分离和可移植的虚拟网络嵌入架构，在该架构中，虚拟网络嵌入问题的处理过程被分为两个独立的阶段：节点映射阶段和链路映射阶段。通过组合不同的节点映射算法和链路映射算法，该架构可生成不同的虚拟网络嵌入算法，针对不同的应用场景，可选择不同的算法对虚拟网络请求进行嵌入映射处理。相反，文献[37]通过联合节点映射和链路映射，提出了协作式的虚拟网络嵌入算法。该算法首先在底层物理网络上构建基于虚拟网络请求的增强图，充分考虑节点和链路的资源属性并在增强图中解决虚拟网络嵌入问题。除此之外，文献[38]从问题建模入手，通过反思网络模型，将虚拟网络请求和底层物理网络建模为流量矩阵而不是通用的内部拓扑，以独特的方式解决 VNE 问题。另外，许多新兴的思想和算法也被提出来解决虚拟网络嵌入问题，譬如，文献[39] 提出了基于机会主义资源共享的虚拟网络嵌入算法；文献[40] 提出了可存活的虚拟网络嵌入算法。

随着网络虚拟化技术的发展，许多成熟的虚拟网络嵌入算法被提出来，文献[41]对近几年提出的虚拟网络嵌入算法进行深入调研和综合分析，提出了一种虚拟网络嵌入的分类学。该分类学认为目前的虚拟网络嵌入算法大致可以从以下几个方面进行分类：

1、静态的 vs 动态的

根据是否要变更或迁移虚拟网络请求，以及是否要扩张底层物理网络或重新整合底层碎片资源，可以将虚拟网络嵌入 VNE 方法分为静态的和动态的。静态的 VNE 方法假定底层物理网络和虚拟网络请求都是静态不变的。相反，

动态的 VNE 方法充分考虑到实际情况中虚拟网络请求可能发生的需求变更以及底层物理网络基础设施可能发生的扩展变化，在进行嵌入时会对部分虚拟网络请求进行必要的迁移，以及将底层物理网络资源进行必要的重新分配。通常，静态的虚拟网络嵌入是理想的情况，实际工程应用中的虚拟网络嵌入都是动态的。静态的 VNE 方法假定一切都是事先已知且不变的，但实际情况却远非如此，不管是虚拟网络请求还是底层物理网络，都可能存在变更，具体表现在以下几个方面：

1、用户需求变更。这是用户主动提出的变更，尤其是在已嵌入的虚拟网络请求生命周期结束之前。

2、供应商设施扩展。这类变更是底层物理网络基础设施提供商提出的变更，通常发生在业务壮大时需要补充扩展基础设施的情况。

3、底层物理网络的碎片资源。与内存碎片一样，底层物理网络也会产生各种碎片资源。随着时间的推移，新的虚拟网络请求到达并嵌入，同时，过期的虚拟网络请求逐渐离开并释放资源。这些不断的加入与退出，使得底层物理网络的资源变得支离破碎，底层物理网络中大量的碎片资源将严重影响系统进一步接受新的虚拟网络请求，导致虚拟网络请求接受率下降。

显然，这些变更和不确定性是静态 VNE 方法无法解决的，动态 VNE 方法会根据实际需求试图重新配置已映射的虚拟网络请求，重组资源的分配，优化底层物理网络的资源利用。虚拟网络重配置算法，VNRe[42]，是一种反应式和迭代式的动态 VNE 算法。该算法在虚拟网络请求嵌入失败时开始工作。当虚拟网络请求被拒绝的时候，对已映射的虚拟网络节点按照可迁移性进行排序，优先迁移最容易迁移的节点并重新映射被拒绝的虚拟网络请求。“迁移-嵌入”过程被反复迭代地进行，直到该虚拟网络请求成功映射或者达到一定的迭代次数。为了减少底层物理网络中的碎片资源并合理利用，可以采用周期性的重新配置算法[43]。为了应对用户对虚拟网络请求的需求变更可采取资源重配置算法[44, 45]。在底层物理网络进行进化和扩展时，可以针对变更的底层物理网络进行相关的重新配置[46]。总体而言，动态 VNE 算法可以根据不同的实际情况制定不同的迁移策略，包括针对链路瓶颈的迁移[42]，服务访问位置变化的迁移[47]等。另外，动态 VNE 算法可以追求不同的嵌入部署目标，如节省底层物理网络资源的重新配置[48]。显然，动态 VNE 会有更好的映射效果，譬如更高的接受率，更少的底层物理网络资源开销和更好的资源利用率，但是动态

VNE 意味着迁移和重映射，其必然带来更大的计算开销，目前一些算法也在研究怎样减少动态 VNE 算法的计算开销[43, 49]。

2、分布式的 vs 中心式的

在中心式的虚拟网络嵌入 VNE 方法中，由一个中心映射实体负责执行虚拟网络请求的嵌入操作。该中心映射实体在嵌入映射过程中，熟知整个系统的资源情况，包括网络系统中各个节点的资源使用情况和相应的链路资源使用情况。这些全局的知识和信息将有助于更加优化的嵌入。然而，中心式的嵌入映射的缺陷就是中心映射实体的单点失效性。如果中心映射实体出现故障或失败将导致整个系统中的虚拟网络映射失败。此外，在大型网络中可能有可伸缩性问题，单一的中心映射实体可能面对众多的虚拟网络请求而不知所措，从而严重影响虚拟网络嵌入的效率。

与中心式的虚拟网络映射不同，在分布式的虚拟网络嵌入 VNE 方法中，由多个映射实体共同负责虚拟网络的嵌入，其优势在于它更高的嵌入效率和可伸缩性。分布式的虚拟网络嵌入通过将虚拟网络请求负载分散到多个映射实体，使得每个映射实体都能更好地应对嵌入，并提高虚拟网络嵌入的效率。然而，分布式的嵌入映射面临的问题是其增长的同步开销，多个映射实体需要汇总各自的嵌入映射结果并进行同步更新。另外，为了得到更优化的嵌入，每个映射实体希望获取到关于全局的足够多的系统信息，可用的信息越多，嵌入的效果就越好。但是，如果多个映射实体都留有足够多的系统全局信息，必然导致开销增大。因此，需要在开销和嵌入性能之间进行平衡和折衷。

目前，大多数虚拟网络嵌入 VNE 方法都是中心式的，而分布式的 VNE 一般指的是在不同底层物理网络设施提供商之间的分布式嵌入[50–53]。

3、简洁的 vs 冗余的

在简洁的虚拟网络嵌入 VNE 方法中，只使用必要的底层物理网络资源来承载虚拟网络请求，绝对不会预留额外的冗余资源。因此，可以使用相同的底层物理网络资源去映射嵌入更多的虚拟网络请求，最大化底层物理网络的收益并提高虚拟网络请求的接受率。然而，由于没有任何预留的冗余资源，使得当一些底层物理网络设备失败的时候，不能快速地从错误中恢复，使得系统的稳定性和可用性降低。

相反，冗余的虚拟网络嵌入 VNE 方法在进行嵌入映射时，不仅为虚拟网络请求配备了必要的资源，还分配了额外的冗余资源。在这种情况下，即使部

分底层物理网络设备失效或在运行时失败时，都可迅速激活备份冗余资源并快速恢复使用。因此，冗余的 VNE 方法可以显著提高嵌入映射的可靠性，这种嵌入映射的可靠性是通过牺牲嵌入映射成本而获得的。嵌入可靠性和嵌入成本开销之间存在着平衡折衷：可靠性越高，嵌入成本越高，更多的资源被使用，底层物理网络可接受的虚拟网络请求就更少。

综上所述，冗余的 VNE 方法比简洁的 VNE 方法更复杂，需要考虑更多的影响因素，包括预留多少冗余资源，当失败的时候如何启动后备资源等。文献[54]充分考虑了业务请求的多样性和后备路径的重要性，在进行虚拟网络嵌入映射时，充分挖掘底层物理网络拓扑的信息并寻找有效的后备路径。文献[55–57]在进行虚拟网络嵌入时，不仅在映射的节点间搜寻直接的路径以满足虚拟网络请求的需求限制，还通过构造逐跳路径作为后备路径供虚拟网络请求选择。

4、协作的 vs 非协作的

从虚拟网络的嵌入过程来看，可以分为两种嵌入方式：协作的和非协作的。非协作的 VNE 方法认为虚拟节点映射和虚拟链路映射是独立的和互不影响的，因此，非协作的虚拟网络嵌入 VNE 方法将虚拟网络请求的嵌入映射分为两个独立的阶段： VnoM 虚拟节点映射阶段和 VliM 虚拟链路映射阶段。首先遵循贪心策略进行虚拟节点映射：对于每个虚拟节点，选择节点资源最丰富的底层物理网络节点来承载该虚拟节点。然后，进行虚拟链路映射，在虚拟节点映射到的物理节点之间寻找满足资源限制的底层物理链路或路径。通常，虚拟链路映射可采用单路径映射[58]，如 k 路最短路径[59]，也可采用多路径映射[36]，如多商品流 MCF[60] 等方法解决。

协作的虚拟网络嵌入 VNE 方法通过协调节点映射和链路映射来进行虚拟网络请求的嵌入，有效地降低由于缺乏协调导致的链路开销，从而提高虚拟网络嵌入接受率，并改善长期收益。协作的 VNE 方法又可以分为两阶段的协作嵌入方法和一阶段嵌入方法。

两阶段的协作算法 D-ViNE 和 R-ViNE [61]，通过结合虚拟网络请求在底层物理网络创建增强图，即以虚拟节点为原型在底层物理网络中添加元节点。在增强图中，再对虚拟网络嵌入问题进行建模，并完成虚拟节点映射和虚拟链路映射这两个阶段的映射。

一阶段的协作映射是指虚拟链路和虚拟节点的映射是同时进行的。当第一

个虚拟节点对映射成功时，紧接着对它们之间的虚拟链路进行映射。随着每个虚拟节点的成功映射，连接虚拟节点的虚拟链路也被成功映射。文献[62]是典型的一阶段映射的方法，通过 BFS 算法计算节点在网络拓扑中的重要性，并依次映射每个节点和每条链路。

3.3 面向媒体业务的虚拟网络嵌入算法设计

3.3.1 整体架构

本章提出了面向媒体业务的虚拟网络嵌入算法，MSO-VNE 。该算法的整体架构如图 3.1 所示。MSO-VNE 算法在进行虚拟网络嵌入时，主要分为以下两个阶段:

1、全局处理

在全局处理阶段，采用“中心式”的架构，对原始的虚拟网络请求运用节点聚类分析和分割处理，将原始的虚拟网络子请求划分为规模较小的多个子请求。接着，每个子请求会被分派到相应区域的底层物理网络中进行局部嵌入。相比原始的虚拟网络请求，新的子请求具有更少的节点和更简单的链路，因此，各个子请求的嵌入处理过程会被大大简化，嵌入映射效率也会被大大提升。

2、局部嵌入

局部嵌入阶段是真正的嵌入映射处理阶段，它的处理对象是全局处理阶段分割而成的所有子请求。由于子请求相比原始的请求有更小的规模，因此，局部嵌入计算的复杂度将大大减小。在这个阶段，多个子请求可以“分布式”地在各自区域的底层物理网络中执行真正的嵌入处理，进一步提高了嵌入效率。另外，在局部嵌入中，我们采用了动态服务均衡分析以辅助嵌入，提高了底层物理网络的资源利用率。

在下面的章节中，我们将详细介绍面向媒体业务的虚拟网络嵌入算法 MSO-VNE 。

3.3.2 网络模型

3.3.2.1 虚拟网络请求

虚拟网络请求可建模为带权重的无向图 $G^v = (N^v, E^v)$，其中 N^v 是虚拟节

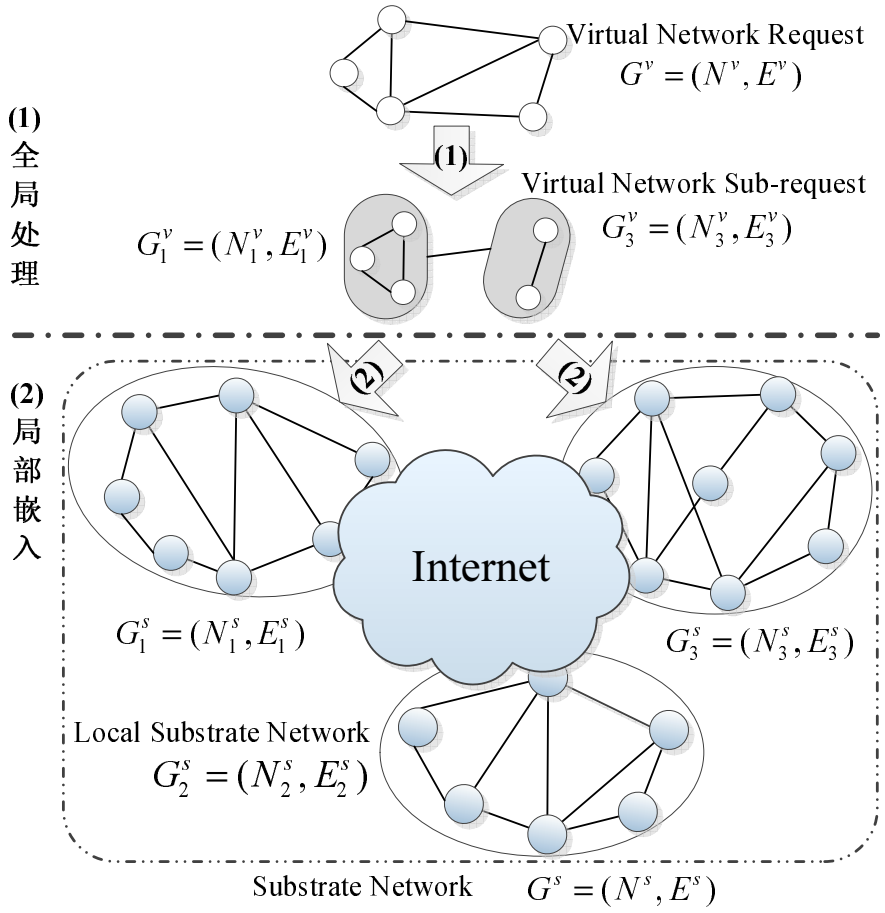

图 3.1: 面向媒体业务的虚拟网络嵌入算法整体架构

点集合，而 E^v 是虚拟链路集合。定义虚拟节点 n^v 的邻居节点集合为 $N^v(n^v)$，定义连接到节点 n^v 的链路集合为 $E^v(n^v)$。虚拟网络请求中的虚拟节点和虚拟链路都与他们的需求限制相关联。我们定义虚拟网络请求中，虚拟节点的需求限制信息包括：地理位置 L^v、存储容量 S^v、磁盘 I/O 速率 I^v、CPU 数量 P^v 和内存大小 M^v，虚拟链路的需求限制包括：带宽 B^v、延迟 D^v 和延迟抖动 J^v。表格 3.1 详细列举出了虚拟网络请求需求限制，并为各个需求限制定义权重影响因子以计算虚拟节点和虚拟链路的归一化需求限制。因此，虚拟网络请求中虚拟节点 n^v 的归一化资源 $c^v(n^v)$ 可表示为:

$$c^v(n^v)=S^v(n^v)\cdot w_S^n+I^v(n^v)\cdot w_I^n+P^v(n^v)\cdot w_P^n+M^v(n^v)\cdot w_M^n$$

虚拟节点需求限制			虚拟链路需求限制		
因素	符号表示	权重因子	因素	符号表示	权重因子
地理位置	L^v		带宽	B^v	w_B^e
存储容量	S^v	w_S^n	延迟	D^v	w_D^e
磁盘 I/O	I^v	w_I^n	延迟抖动	J^v	w_J^e
CPU 数量	P^v	w_P^n			
内存大小	M^v	w_M^n			
归一化资源	c^v		归一化资源	b^v	
$w_S^n + w_I^n + w_P^n + w_M^n = 1$			$w_B^e + w_D^e + w_J^e = 1$		

表 3.1: 虚拟网络请求的需求限制

节点资源限制			链路资源限制		
因素	符号表示	权重因子	因素	符号表示	权重因子
地理位置	L^s		带宽	B^s	w_B^e
存储容量	S^s	w_S^n	延迟	D^s	w_D^e
磁盘 I/O	I^s	w_I^n	延迟抖动	J^s	w_J^e
CPU 数量	P^s	w_P^n			
内存大小	M^s	w_M^n			
归一化资源	c^s		归一化资源	b^s	
$w_S^n + w_I^n + w_P^n + w_M^n = 1$			$w_B^e + w_D^e + w_J^e = 1$		

表 3.2: 底层物理网络的资源限制

虚拟链路 e^v 的归一化资源 $b^v(e^v)$ 可表示为:

$$b^v(e^v) = B^v(e^v) \cdot w_B^e + D^v(e^v) \cdot w_D^e + J^v(e^v) \cdot w_J^e$$

其中需求限制中的地理位置限制主要用于节点聚类分析，不用考虑在节点需求的归一化表示中。

3.3.2.2 底层物理网络

同样地，底层物理网络可建模为带权重的无向图 $G^s = (N^s, E^s)$，底层物理网络的第 i 个区域可表示为无向图 $G_i^s = (N_i^s, E_i^s)$，其中 N_i^s 表示区域 i 内的

节点集合，而 E_i^s 表示区域 i 内的链路集合。定义节点 n^s 的邻居节点集合为 $N^s(n^s)$，定义连接到节点 n^s 的链路集合为 $E^s(n^s)$。底层物理网络中的节点和链路都与他们的资源限制相关联。在底层物理网络中，我们定义节点的资源限制信息包括：地理的位置 L^s、存储容量 S^s、磁盘 I/O 速率 I^s、CPU 数量 P^s 和内存大小 M^s，链路的资源限制包括：带宽 B^s、延迟 D^s 和延迟抖动 J^s。表格 3.2 详细列举出了底层物理网络中的资源限制，并为各个资源限制定义权重影响因子以计算节点和链路的归一化资源限制。因此，底层物理网络中节点 n^s 的归一化资源 $c^s(n^s)$ 可表示为:

$$c^s(n^s) = S^s(n^s) \cdot w_S^n + I^s(n^s) \cdot w_I^n + P^s(n^s) \cdot w_P^n + M^s(n^s) \cdot w_M^n$$

链路 e^s 的归一化资源 $b^s(e^s)$ 可表示为:

$$b^s(e^s) = B^s(e^s) \cdot w_B^e + D^s(e^s) \cdot w_D^e + J^s(e^s) \cdot w_J^e$$

其中资源限制中的地理位置限制主要用于节点聚类分析，不用考虑在节点需求的归一化表示中。

在虚拟网络请求和底层物理网络中，节点与链路的需求限制和资源限制都涉及影响因子的权重设定。根据不同的应用场景，可酌情调整各个影响因素的权重。譬如，在带宽主导的应用场景中，可酌情加大带宽影响因子的权重；在计算主导的应用场景中，可酌情加大 CPU 影响因子的权重等等。一般情况下，可简单地设定为均衡权重，即各个影响因子都是同等重要的。

虚拟网络嵌入问题是动态的。随着时间推移，不同的虚拟网络请求会到达系统并接受嵌入映射处理，同时，系统中生命周期结束的虚拟网络请求会离开系统，释放并归还占用的资源。也就是说，当新的虚拟网络请求到达系统时，虚拟网络嵌入算法发挥作用，在条件允许的情况下，系统中的物理资源会被分配给虚拟网络请求；当虚拟网络请求的生命周期结束时，虚拟网络请求将离开系统，并释放占用的资源。因此，底层物理网络中的资源又分为可用资源 $c_{available}$ 和占用资源 c_{used}。我们定义虚拟网络请求及其占用的资源为一个映射集合 M_{map}，映射集合中的任意一个映射对 $< r, c > \in M_{map}$ 表示虚拟网络请求 r 占用了底层物理网络的 c 资源。

在底层物理网络中，我们为每个区域的底层物理网络定义了一个位置中心，如第 i 个区域的底层物理网络 G_i^s 的位置中心为 O_i^s，即 $O_i^s = (x_i, y_i)$。位置

中心是虚拟的，实际中并不真正存在这个中心点。我们将虚拟的位置中心 O_i^s 定义为该区域内所有节点的位置的几何中心：

$$O_i^s = \sum_{n^s \in N_i^s} \frac{l^s(n^s)}{|N_i^s|}$$

其中，$|N_i^s|$ 表示第 i 个区域的底层物理网络中的节点个数。在实际生活中，每个区域的底层物理网络通常都是一个局域网环境，不同的区域底层物理网络通过 Internet 进行互联互通。由于 Internet 非常复杂，所以，我们假定在 Internet 中的带宽资源是无限的。虽然，上述假定是不成立的，但是我们对 Internet 的资源是没有办法控制的，因此上述假定也是合理的。

3.3.3 节点聚类

在面向媒体业务的虚拟网络嵌入算法，MSO-VNE 中，节点聚类分析是基于地理位置展开的，其原因有如下两点：

1、底层区域性

通常，数据中心或底层物理网络分布在不同的地理位置，具有明显的区域性特征。采用基于地理位置的节点聚类分析，可有效地利用底层物理网络的区域性特征，增加系统的信息量，提高准确率。

2、QoS 需求

一般而言，虚拟网络请求都有地理位置的需求限制，尤其是实时业务的虚拟网络请求，对地理位置的需求限制更加迫切。因为它们需要更高的 QoS 服务质量以提高用户的体验质量，比如多媒体业务对带宽、延时以及延时抖动的要求就比较苛刻，这是业务或服务本身的特质。为了保证这类业务或服务的 QoS 服务质量，通常把服务节点部署在离终端用户较近的地方，以减少延时，改善 QoS 服务质量。

因此，采用基于地理位置的节点聚类分析，一方面，可以充分利用底层物理网络的区域性特征；另一方面，可以提高业务和服务的 QoS 服务质量。在虚拟网络请求 G^v 中，通过对虚拟节点应用基于地理位置的聚类分析，将请求中的节点归聚到若干个不同的类别，从而将原始的虚拟网络请求聚类为多个规模较小的子请求。虚拟网络请求中的任意节点 $n^v \in N^v$，将被聚集到 $C(n^v)$ 中：

$$C(n^v) = \underset{i=0,1,2,\cdot,|O^s|-1}{argmin} \; dis(l^v(n^v), O_i^s)$$

每个虚拟节点 n^v 都被聚类到离它最近的区域底层物理网络 G_i^s 中，其中 $i = C(n^v)$。同时，对于任意的虚拟网络请求 G^v，其聚类后的子请求数量绝不超过区域底层物理网络的总数目 $|O^s|$。聚类分析后，根据聚类结果，可以将虚拟网络请求 G^v 分割为一组虚拟网络子请求集合 Q。该子请求集合中的任意一个子请求 $G_i^v \in Q$ 都比原始的请求 G^v 规模小，同一个子请求 G_i^v 中的虚拟节点属于相同的聚类 i，其中 i 也是第 i 个区域底层物理网络 G_i^s 的索引。由于我们假定不同的区域底层物理网络通过 Internet 互联互通，且 Internet 的链路资源是无限的。因此，任何端点在不同聚类中的虚拟链路都可以忽略不计。这样，每个虚拟网络子请求都可以独立地分配到区域底层物理网络 G_i^s 中进行局域嵌入处理。进而，所有的子请求都可以互不干扰地在不同的区域底层物理网络中进行并发的局域嵌入处理，这样一来，整个虚拟网络请求的嵌入性能便得到大幅度提升。

3.3.4　动态服务均衡

PageRank[63, 64] 网页排名算法是 Google 提出的，用于搜索引擎进行网页排名。PageRank 可以有效地评估网页的相关性和重要性，每个页面的排名值暗含了该页面的质量和受欢迎程度。PageRank 通过互联网络中海量的超链接关系来确定一个页面的排名和等级。如果页面 A 有链接指向页面 B，表明页面 A 向页面 B 的网页排名投票，PageRank 根据投票来源和投票目标的网页排名值来决定新的网页排名。一个页面的网页排名是由所有链向它的页面的网页排名经过递归迭代计算出来的。通常，一个被很多高网页排名的网页链接的网页将获得一个较高的网页排名。然而，一个页面链接到越多的页面，它对其他页面的网页排名的贡献就越小。

受 PageRank 网页排名算法的启发，我们在虚拟网络嵌入中引入了服务能力的概念，服务能力可用来评估特定节点的重要性。在通用网络模型 $G = (N, E)$ 中，对于任意的节点 $n \in N$，其服务能力 $r(n)$ 可以定义为：

$$r(n) = \sum_{m \in N} p(m, n) \cdot r(m)$$

在服务能力的定义中，我们引入了服务因子 $p(m,n)$，它表示节点 m 对节点 n 的服务能力贡献，其定义如下：

$$p(m,n)=\begin{cases} f_1 & m=n \\ \frac{f_2 \cdot r_0(n)}{\sum_{h\in N_{neigh}(m)} r_0(h)} & e(m,n)\in E \\ 0 & other \end{cases}$$

其中，$N_{neigh}(m)$ 表示节点 m 的邻居节点集合，而 f_1 和 f_2 是相应的权重常数，满足 $f_1+f_2=1$。$r_0(n)$ 是节点 n 的归一化资源，定义为节点资源和链接到该节点的链路资源的一半的乘积：

$$r_0(n)=c(n)\cdot \sum_{e\in E_{neigh}(n)} 1/2\cdot b(e)$$

其中，$E_{neigh}(n)$ 表示链接到节点 n 的链路集合，我们假定链路带宽是由链路两端的节点完全共享的，这就是系数1/2的来源。

因此，动态服务均衡算法可以描述如表 3.3 所示。根据 PageRank[64]网页排名算法，动态服务均衡算法可在多项式时间复杂度内完成递归迭代运算，并计算出各个节点的动态服务能力。

$R=LB(G,\varepsilon_m)$，其中 $G=(N,E)$，ε_m 为阈值
Compute each $r_0(n)$ and $p(m,n)$, initialize $i\leftarrow 0$
while $\varepsilon<\varepsilon_m$ do
$\quad\varepsilon\leftarrow 0$
$\quad$for all $n\in N$ do
$\quad\quad r_{i+1}(n)\leftarrow\sum_{m\in N} p(m,n)\cdot r_i(m)$
$\quad\quad\varepsilon\leftarrow\varepsilon+\|r_{i+1}(n)-r_i(n)\|$
$\quad\quad i\leftarrow i+1$
$\quad$end for
end while

表 3.3: 动态服务均衡算法

动态服务均衡算法在虚拟网络嵌入中扮演着非常重要的角色。假设在底层物理网络中存在两个节点，它们可用的资源是完全相同的，然后，它们的邻居

的可用资源，或者它们的邻居的邻居的可用资源却大不相同。此时，为了增加嵌入映射的成功概率，并减少虚拟链路在底层物理网络中的路径长度，我们宁愿选择其邻近节点具有更多资源的节点，也就是具有更高动态服务能力的节点。事实上，一个具有较高服务能力的底层节点至少满足以下两点中的任一点：

1、节点自身有丰富的资源

2、节点的邻居有丰富的资源

不管是节点自身的丰富资源还是节点邻居的丰富资源，都让我们更加相信在那些具有较高动态服务能力的底层节点上成功嵌入虚拟网络请求的概率增加。同时，“能者多劳”的思想使得一个成功的嵌入有更好的负载均衡效果。同样地，我们认为高动态服务能力的虚拟节点具有优先嵌入权，因为当该节点嵌入失败的时候，它可以提前告知整个虚拟网络请求的失败，从而终止虚拟网络请求的嵌入，减少不必要的嵌入开销。

3.3.5　局部嵌入映射

如图 3.1 所示，每个虚拟网络子请求真正的嵌入映射是在局部嵌入阶段完成的。经过全局处理后，虚拟网络请求被聚类分割为多个规模较小的子请求，再经过简化处理后，各个子请求变得相互独立。因此，我们可以并发地处理多个子请求。在对子请求进行局部嵌入时，主要采用贪心策略。贪心策略以节点的动态服务能力为参考依据，优先将子请求中具有最大动态服务能力的虚拟节点映射到对应的区域底层物理网络中具有最大动态服务能力的底层节点上。局部嵌入的具体算法描述如表 3.4 所示。由此可知，所有虚拟网络子请求嵌入处理的时间复杂度为 $O(max_{0\leq i<|O^s|}(|N_i^v|^2\cdot(|N_i^s|log|N_i^s|+|E_i^s|)))$。

3.4　仿真验证

3.4.1　仿真环境

为了评估本章提出的面向媒体业务的虚拟网络嵌入算法，MSO-VNE，本节将详细介绍我们为该算法设计的仿真实验。在仿真实验中，我们设计并实现了离散事件模拟器，以更贴近现实的方式模拟虚拟网络请求的到来和被处理的过程。仿真实验可以有效地评估虚拟网络嵌入算法的性能，也为虚拟网络嵌入算法在工程中的实际应用提供依据和保障。

局部嵌入算法：$LocalEmbed(G_i^s, G_i^v)$	
1.	对 G_i^s 和 G_i^v 应用动态服务均衡分析，计算各节点的动态服务能力，时间复杂度为 $O(\|N_i^s\|log\|N_i^s\|)$，其中 $O(\|N_i^s\|) > O(\|N_i^v\|)$；
2.	从具有最高动态服务能力的虚拟节点 n^v 开始，依次处理所有未嵌入的虚拟节点，直到所有虚拟节点都被嵌入为止；
2.1.	从区域底层物理网络中选择最多 k 个最高动态服务能力的候选节点，这些候选节点都能满足节点 n^v 的需求限制。时间复杂度为 $O(N_i^s)$；
2.2.	找出每个候选节点到已映射节点之间的最短路径，并计算总的路径开销，时间复杂度为 $O(\|N_i^v\| \cdot (\|N_i^s\|log\|N_i^s\|) + \|E_i^s\|)$；
2.3.	在 k 个候选节点中，找出具有最小的总的路径开销的底层节点 n^s，时间复杂度为 $O(k)$；
2.4.	将虚拟节点 n^v 映射到底层节点 n^s 上，同时完成已映射节点到 n^s 节点之间的链路映射，时间复杂度为 $O(\|E_{neigh}^v(n^v)\|)$；
3.	当所有的虚拟节点都成功映射后，分配实际的资源以提供服务，时间复杂度为 $O(\|N_i^v\| + \|E_i^v\|)$；

表 3.4: 虚拟网络嵌入的局部嵌入算法

在实际生活中，真实的底层物理网络基础设施的拓扑是很难获取的，同时，虚拟网络请求的拓扑也很难得到。因此，本节采用人工合成的网络拓扑，我们使用的合成工具是 GT-ITM 拓扑生成器。GT-ITM 拓扑生成器可以用来生成以平面随机图和层次图作为模型的网络拓扑。主要的平面随机图模型包括以下几种：

1、纯随机模型

结点在平面上随机分布，任意两个结点间有边的概率为 a 。纯随机模型非常简单，但不能很好地反映现实网络的拓扑结构。

2、Waxman 1 模型

结点在平面上随机分布，从结点 u 到 v 有边的概率为 $P(u,v) = a * e^{-d/(bL)}$，其中 $0 < a, b \leq 1$，d 是两结点间的距离，$L = \sqrt{2} * scale$ 是平面上任意两结点间的最大距离。a 增大则图中边的数目会增大，b 增大则图中长边数与短边数的比值会增大。

3、Waxman 2 模型

Waxman 2 模型和 Waxman 1 模型很相似，将 Waxman 1 模型中的参数 d 的取值范围变为 0 到 L 之间的随机数即是 Waxman 2 模型。

4、Doar-Leslie 模型

该模型和 Waxman 1 模型很相似，将 Waxman 1 模型中得到的概率值 $P(u,v)$ 乘以一个比例因子 ke/n 便是 Doar-Leslie 模型。其中 e 是结点度数的期望值，n 是结点数，k 是由 a，b 共同决定的常数。

5、指数模型

在指数模型中，从结点 u 到 v 有边的概率为 $P(u,v) = a * e^{-d/(L-d)}$，由此可知，节点间有边的概率随着节点间距离的增加而成指数级的降低。

6、Locality 模型

在 Locality 模型中，根据两结点间的距离将结点对分成不同的等级，不同等级的结点对之间有边的概率不同。譬如在两级模型中的分级概率模型如下：

$$P(u,v) = \begin{cases} a, & d < L * radius \\ b, & d \geq L * radius \end{cases}$$

也就是说，如果节点 u 和 v 之间的距离 d 满足 $d < L * radius$，则节点 u 和 v 之间有边的概率为 $P(u,v) = a$，否则节点间有边的概率为 $P(u,v) = b$，其中 $radius$ 用来确定分级界限。

平面随机图是层次图的基础，层次图一般由多个平面随机图组成，GT-ITM 中的层次图主要包括两类：N-Level 和 Transit-Stub 。其中 N-Level 采用递归的方式来生成网络拓扑图：首先用上述六种随机图模型中的任一种生成一个平面随机图，作为首层图；然后用平面随机图代替首层图中的每一个结点，并且依次替代下去。在 Transit-Stub 中，将结点划入不同类型的域，再将这些域连接起来：首先生成一个平面随机图，图中的每一个结点代表一个 transit 域；然后用平面随机图代替这些 transit 域，表示这些 transit 域的骨干拓扑；对 transit 域中的每个结点，生成一个或多个随机平面图作为 stub 域，并将其和结点连接起来；最后还可以在特定的结点对之间增加一些额外边，这些结点对需满足：一个在 transit 域一个在 stub 域，或者在不同的 stub 域。

在本节的仿真实验中，我们采用 Transit-Stub 模型构建底层物理网络拓扑，采用平面随机图模型合成虚拟网络请求。我们构建了 3 个区域底层物理网络和成千上万个虚拟网络请求。其中，每个区域底层物理网络包含 50 个节点，覆

盖 80×80 的网格区域。同时，我们假定不同的区域底层网络网络通过 Internet 互联互通，考虑到 Internet 的复杂性和不可控制性，我们假定 Internet 有无限的链路资源。为了使我们的仿真实验更接近真实情景，我们设定虚拟网络请求的到达服从泊松分布，平均到达速率是每 100 个时间单位 4 到 8 个请求不等。每个虚拟网络请求的生存周期服从平均时间为 1000 个时间单位的指数分布，且每个虚拟网络请求的节点数量是 2 到 10 的均匀分布。

仿真实验的主要目的是评估面向媒体业务的虚拟网络嵌入算法 MSO-VNE 的性能，因此，实验参照对象是必不可少的。我们在仿真实验中选择的参照对象是现有的 D-ViNE 算法和 R-ViNE 算法，主要的评估指标包括时间复杂度、负载分布、平均收益开销比和平均接受率。为了更真实准确地评估 MSO-VNE 算法的性能，我们在相同的仿真环境下对同样的数据集应用以上三种不同的虚拟网络嵌入算法。

3.4.2 时间复杂度

虚拟网络嵌入问题是典型的 NP 难问题，其时间复杂度非常高。因此，现有的研究大多倾向于设计启发式算法来解决。譬如，D-ViNE 算法和 R-ViNE 算法就是典型的基于启发式的虚拟网络嵌入算法。我们先来详细分析 D-ViNE 算法的时间复杂度。

D-ViNE 算法和 R-ViNE 算法[61]是协调了节点映射和链路映射的虚拟网络嵌入算法，算法首先会结合虚拟网络请求构建增强的底层物理网络拓扑，这一步骤的时间复杂度为 $O(|N^v|)$ 。然后，算法将着力解决整数线性规划问题，其时间复杂度为 $O((|E^s|(1+|E^v|))^{3.5}L^2lnLlnlnL)$。最后，算法将以时间复杂度 $O((|E^s||E^v|)^{3.5}L^2lnLlnlnL)$ 来解决链路映射相关的 MCF 问题。由此可知，在这两个算法中，整数线性规划问题是复杂度最高的部分，它也就是整个算法的时间复杂度。

在面向媒体业务的虚拟网络映射算法 MSO-VNE 中，全局处理阶段将虚拟网络请求划分为多个规模较小的子请求，相比原始的虚拟网络请求，对规模较小的子请求进行嵌入处理具有更低的时间复杂度。由于虚拟网络子请求之间的相互独立性，局部嵌入阶段可并发地嵌入映射多个子请求，便进一步降低了虚拟网络嵌入处理的时间复杂度。如表 3.4 所示，面向媒体业务的虚拟网络嵌入

算法 MSO-VNE 的时间复杂度为：

$$O(\max_{0\leq i<|O^s|}(|N_i^v|^2\cdot(|N_i^s|log|N_i^s|+|E_i^s|)))$$

即使在最坏的情况下，当所有的区域底层物理网络都位于同一个地理位置时，虚拟网络嵌入算法的时间复杂度演变为：

$$O(|N^v|^2\cdot(|N^s|log|N^s|+|E^s|))$$

由于底层物理网络和虚拟网络请求都是连通图，满足 $O(|N|)<O(|E|)$，因此，虚拟网络嵌入算法 MSO-VNE 的时间复杂度可简化为：

$$O(|E^v|^2\cdot|E^s|log|E^s)$$

与现有的 D-ViNE 算法和 R-ViNE 算法相比，很显然，本章提出的面向媒体业务的虚拟网络嵌入算法 MSO-VNE 具有更低的时间复杂度。

3.4.3　负载分布

负载分布包括底层物理网络中物理节点上的负载分布和物理链路上的负载分布。负载分布主要衡量在一段时间内，底层物理网络中各个物理节点和链路上的负载情况，以及全部物理节点和链路上的负载分布情况。

对于虚拟网络嵌入问题，虚拟网络请求就是底层物理网络的负载。具体而言，虚拟网络请求中的节点会根据其需求限制占用底层物理网络中节点的资源，如 CPU 资源、存储资源等。同时，虚拟网络请求中的链路也会根据其需求限制占用底层物理网络中链路的资源，如带宽资源等。由于嵌入算法和部署策略的不同，导致这些虚拟节点负载和虚拟链路负载在底层物理网络中的分布各不相同。因此，衡量负载分布的具体情况，可以窥见虚拟网络嵌入算法的性能。

需要注意的是，虚拟网络嵌入问题是动态问题。随着时间的推移，新的虚拟网络请求会到达底层物理网络并占用底层物理网络资源，同时，过期的虚拟网络请求会随着其生命周期的结束会离开底层物理网络并释放其占用的资源。因此，底层物理网络中的资源使用情况是动态变化的。另外，由于我们假定虚拟网络请求的到达服从泊松分布，虚拟网络请求的生命周期服从指数分布。因

此，底层物理网络中总体资源的使用情况在动态变化中能保持基本平稳，这可从仿真实验结果图 3.2 和图 3.3 中看出。

仿真实验还可反映出虚拟网络嵌入算法的性能。从实验结果图中，我们可以看出，MSO-VNE 算法具有更好的均衡负载分布能力。如图 3.2 和图 3.3 所示，在 MSO-VNE 算法中，底层物理资源使用得最多和最少的底层物理网络节点和链路的比例都要明显低于在其他两个算法。换句话说，在 D-ViNE 算法和 R-ViNE 算法中，存在更多的资源利用不合理的底层物理节点和链路，其利用不合理性在于：这些节点资源和链路的资源要么被过度使用，要么没有得到充分地利用。

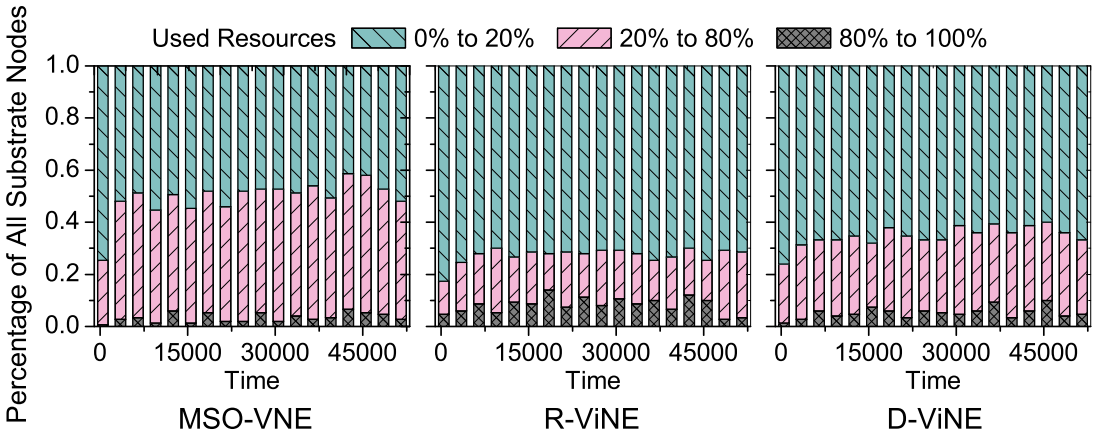

图 3.2: 底层物理网络中的节点负载分布

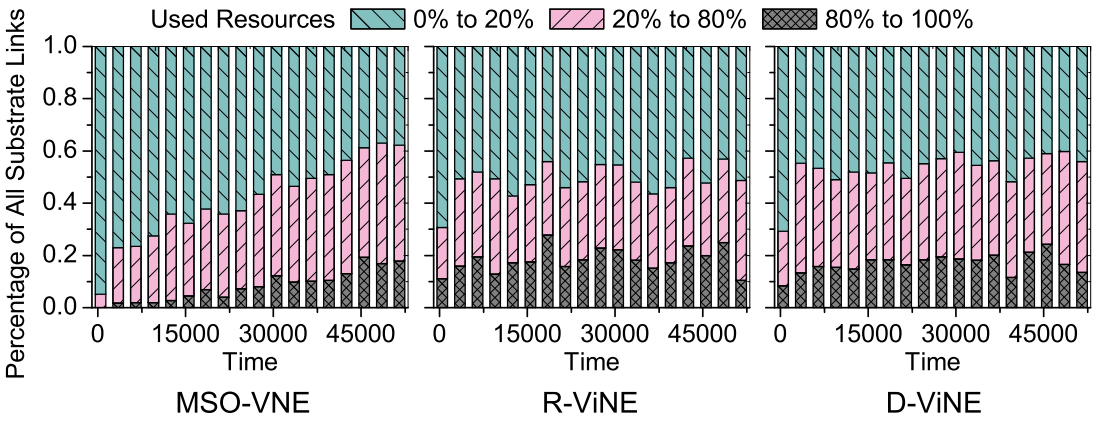

图 3.3: 底层物理网络中的链路负载分布

综上所述，面向媒体业务的虚拟网络嵌入算法 MSO-VNE 相比于 D-ViNE 算法和 R-ViNE 算法具有更好的负载均衡能力。MSO-VNE 算法使得虚拟网络请求中的节点负载和链路负载在底层物理网络中的分布更均衡，从而提升了底层物理网络中的资源利用率。

3.4.4　平均收益开销比

在虚拟网络嵌入问题中，存在“收益”和“开销”两个概念。收益是指当成功嵌入一个虚拟网络请求后，底层物理网络因承载该虚拟网络请求获得的收益。开销是指当成功嵌入一个虚拟网络请求后，底层物理网络因承载该虚拟网络请求的资源开销。通常，对于节点而言，其收益和开销是相同的。譬如虚拟节点需要 50M 存储空间，为了承载该虚拟节点，底层物理网络需要分配 50M 的存储空间给该虚拟节点，不管是从一个底层节点上分配 50M 存储空间，还是多个底层节点共同凑齐 50M 存储空间。底层物理网络的收益和开销都是 50M 的存储空间，因此，其收益和开销是相同的，收益开销比为：收益/开销 = 1，且保持恒定不变。

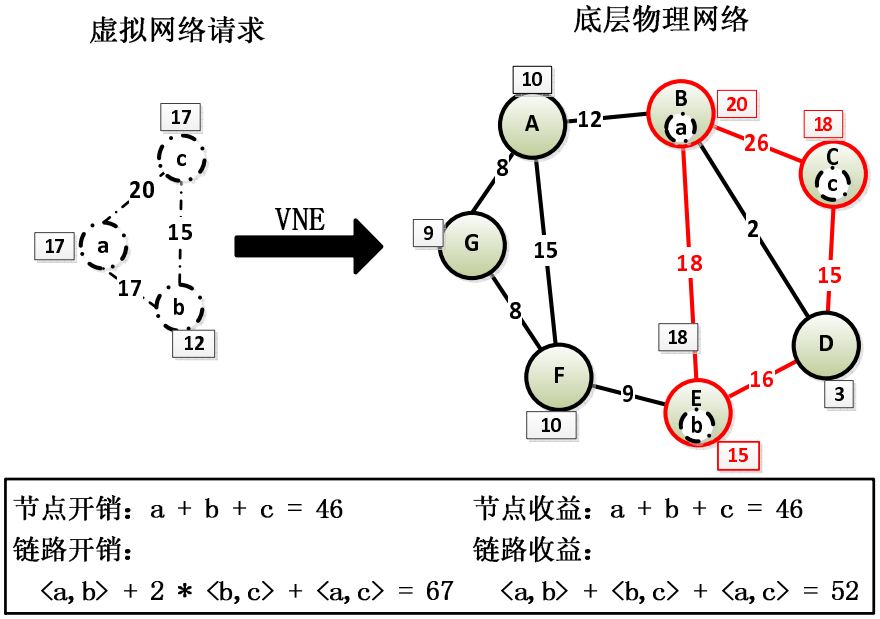

图 3.4: 虚拟网络嵌入中的链路收益开销示意图

对于链路而言，收益和开销就不一定相同了。在虚拟网络嵌入中，可能存在跨过节点的链路映射，即虚拟链路可能会被映射到由多条底层物理

链路组成的路径上，该路径中的某些节点并没有被用于承载任何虚拟节点。在这种情况下，底层物理网络中的链路开销将大于该底层物理网络的链路收益。如图 3.4 所示，当虚拟网络请求嵌入到底层物理网络中时，物理路径$< E, D, C >$上的节点 D 并没有用于承载任何虚拟节点，因为其节点资源不足以承载虚拟网络请求中的任何一个虚拟节点。此时，底层物理网络中的链路开销为 $< a, b > +2 * < b, c > + < a, c >= 67$，而链路收益只有 $< a, b > + < b, c > + < a, c >= 52$。因此，底层物理网络中的链路收益开销比为 $52/57 = 0.78$。

作为基础设施提供者，追求的是将底层物理网络的利益最大化，希望在相同的底层物理网络上嵌入部署更多的虚拟网络请求，追求收益开销比最大化。如图 3.5 所示的仿真实验结果表明，在三个虚拟网络嵌入算法中，我们提出的 MSO-VNE 算法有最高的平均收益开销比，这就意味着 MSO-VNE 算法获得相同的收益只需要更少的开销，或者说在相同的开销下可以获得更多的收益。

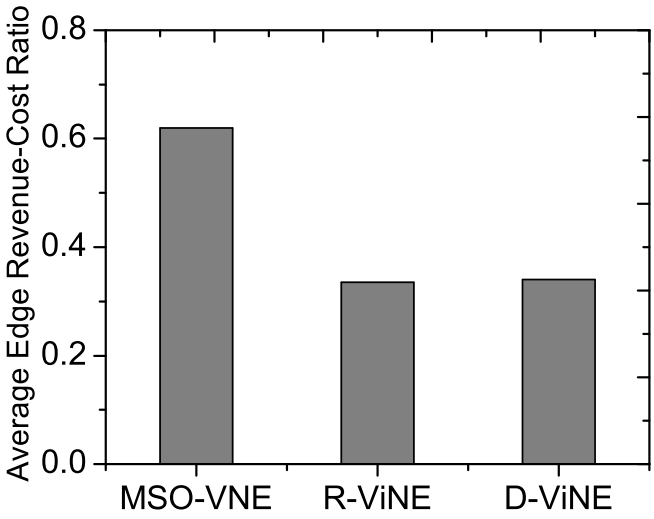

图 3.5: 平均收益开销比对比结果

MSO-VNE 算法的高平均收益开销比来源于局部嵌入阶段的动态负载均衡分析。在动态负载均衡分析中，通过引入动态服务能力来评估节点的重要性，由于动态服务能力不仅考虑了节点自身的资源，还考虑了邻居节点的资源，再加上贪心部署策略的实施，使得虚拟网络嵌入能有效地避免孤立嵌入问题。孤立嵌入是指将虚拟网络请求中的某个虚拟节点嵌入到底层物理网络中孤立的物

理节点上的情形。孤立的物理节点是指那些自身拥有丰富的资源，但是其邻居节点的资源几被消耗殆尽的节点。当存在孤立嵌入时，为了完成整个虚拟网络请求的嵌入，必然存在很多跨越节点的链路映射，从而增加了虚拟链路映射的底层路径长度，使得链路的开销增大。

在 MSO-VNE 算法中，局部嵌入阶段引入的动态服务能力和贪心策略可保证虚拟链路尽可能被映射到底层物理网络中的较短路径上。因此，底层物理网络在承载该虚拟网络请求时，采用 MSO-VNE 算法将导致虚拟链路映射的底层物理路径长度减少。由于链路收益是不变的，链路开销的降低使得底层物理网络的链路收益开销比得到增加。

3.4.5　平均接受率

平均接受率衡量的是在一段时间内被底层物理网络接受的请求在总请求中的比例。由于虚拟网络嵌入的动态性，底层物理网络中的资源使用情况是动态变化的。动态性即随着时间的推移，新的虚拟网络请求会到达底层物理网络并占用底层物理网络资源，同时，过期的虚拟网络请求会随着其生命周期的结束而离开底层物理网络并释放其占用的资源。另外，虚拟网络嵌入又是基本平稳的，因为虚拟网络请求的到达服从泊松分布，同时，虚拟网络请求的生命周期服从指数分布。因此，平均接受率在足够长的时间后也是保持基本平稳的。

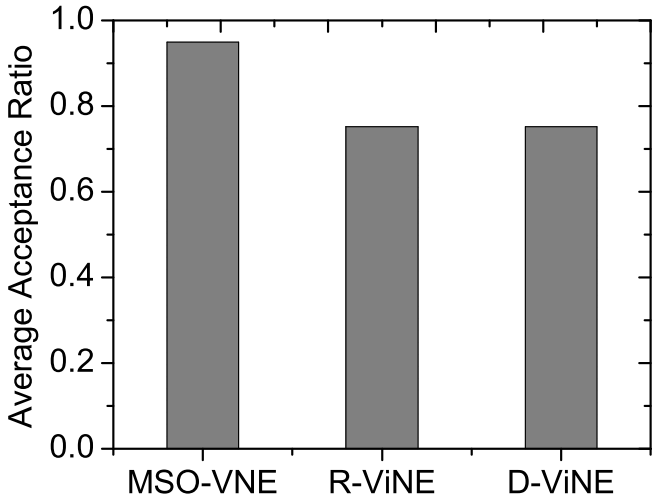

图 3.6: 平均接受率对比结果

如图3.6所示的仿真实验结果图表明，在所有的算法中， MSO-VNE 算法具有最高的平均接受率。即在相同的环境下， MSO-VNE 算法能够接受并容纳的虚拟网络请求数绝不会比其他的算法少。

MSO-VNE 算法的高平均接受率是很容易理解的。直观地， MSO-VNE 算法的高平均收益开销比说明该算法在相同的收益下，只需要更少的资源开销。这就意味着剩余的资源可以用来嵌入更多的虚拟网络请求，久而久之，底层物理网络便能接受并容纳更多的虚拟网络请求，即具有更高的平均接受率。

仿真实验结果表明，相比于 D-ViNE 算法和 R-ViNE 算法，本章提出的面向媒体业务的虚拟网络嵌入算法 MSO-VNE 具有很出色的性能表现：更低的时间复杂度、底层物理网络中更均衡的负载分布、更高的平均收益开销比和更高的平均接受率。 MSO-VNE 算法的出色性能，一方面得益于全局处理阶段采用的分而治之思想和并发处理思想：通过将虚拟网络请求聚类分割为规模较小的子请求，并将各个子请求分派到离它最近的区域底层物理网络中进行并发处理，从而大大降低了时间复杂度。另一方面得益于局部嵌入阶段的动态负载均衡分析：通过引入动态服务能力来评估节点的重要性，不仅考虑节点自身的资源承载能力，还兼顾邻居节点的资源承载能力。动态均衡分析为整个虚拟网络请求的成功和优化嵌入提供了保障，在改善了底层物理网络的负载分布情况和资源利用率的情况下，大大减少了链路映射开销，提高了平均收益开销比和平均接受率。

3.5 面向媒体业务的资源映射交互框架

基于面向媒体业务的虚拟网络嵌入算法 MSO-VNE，本节提出了面向媒体业务的资源映射交互框架，MSO-VNE 算法负责处理该交互框架的资源映射，是该交互框架的核心模块。本节将详细介绍面向媒体业务的资源映射交互框架，包括系统框架和各部分的组件模块。

3.5.1 系统框架

在面向媒体业务的资源映射交互框架中，MSO-VNE 算法是其核心模块。尽管虚拟网络嵌入问题起源于网络虚拟化，但它仍然属于资源管理领域，因此，可将其应用到资源映射交互框架中处理资源映射问题。本章提出的 MSO-VNE 算法是面向媒体业务的，因此，将其应用到面向媒体业务的资源映射管

理中，可以有效保障媒体服务的 QoS 服务质量。如图 3.7 所示，资源映射交互框架主要包括以下几个模块：

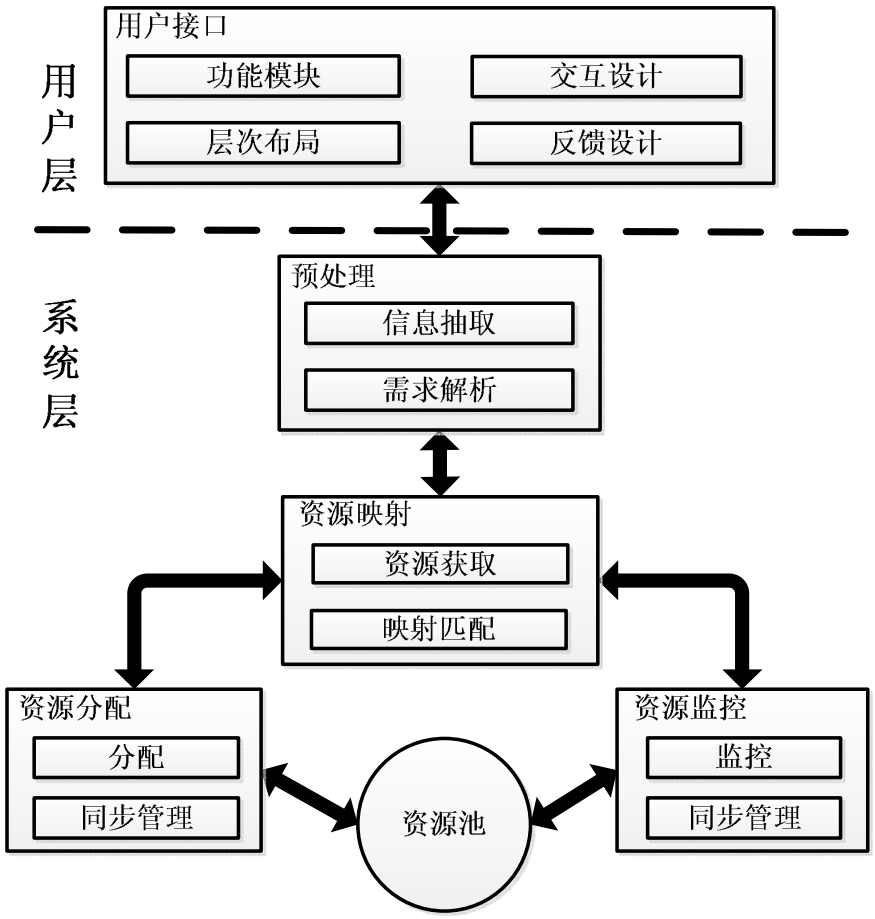

图 3.7: 面向媒体业务的资源映射交互系统框架

1、用户接口模块

用户接口模块是资源映射交互框架中面向用户的部分，用户通过该模块中的相应组件以命令行方式或 Web 页面方式定义其业务需求和服务请求，通过提交请求以获取特定的服务。

2、预处理模块

预处理模块的主要职责是对用户提交的请求进行预处理，使得该请求不仅是人类可理解的，更是计算机可理解的。

3、资源映射模块

资源映射模块是资源映射交互框架的核心模块。在该模块中，面向媒体业务的虚拟网络嵌入算法 MSO-VNE 被采用以处理该框架中的资源映射问题，通过将用户的业务请求和底层物理网络的可用资源进行匹配计算和映射处理，完成资源的映射。

4、资源分配模块

资源分配模块是资源映射模块的补充。当用户提交的业务请求和底层物理网络的可用资源成功映射后，资源分配模块便开始工作，它根据资源映射模块的映射结果进行真正的资源分配工作。

5、资源监控模块

资源监控模块是辅助模块，它可以辅助资源映射模块和资源分配模块完成它们的工作。资源映射模块通过资源监控模块获取底层物理网络中的可用资源数量，以辅助资源映射工作。资源分配模块通过资源监控模块获取并分配实际的资源。资源监控模块通过实时监控底层资源池中的资源使用变化情况并进行必要的同步管理。

从整体来看，面向媒体业务的资源映射交互框架可以分为两层：用户层和系统层。其中，用户层的设计是面向业务用户的，用户通过定义他们的业务需求并提交请求，然后，这些业务需求会被分析和处理。当然，任何请求的具体的分析和处理都是在系统层进行的，而系统层对用户是透明的。用户层的设计宗旨是让用户专注在他们所关心的业务和服务上，而不是任何其他的琐碎细节，如怎样进行请求预处理，如何执行映射抑或如何分配或监控资源。这些琐碎的细节都是有系统层来处理的。

当用户的请求到达时，系统层开始工作。首先，系统层执行预处理以从用户提交的业务需求请求中提取有用的信息并分析用户的业务需求。系统层的核心是资源映射，在资源映射模块中，我们采用了 MSO-VNE 算法以执行映射匹配。通过资源监控模块，可以获取系统的可用资源，如果系统有足够的资源，用户的请求就会得到成功的匹配。当成功匹配后，资源分配子模块会为用户分配实际的资源。当然，不管是否能够成功映射，系统层都会将处理结果反馈给用户层。反馈对系统是有用的，不仅可以告诉用户请求的处理结果，还可以辅助用户做更好的决策。

下面，我们将详细描述资源映射交互框架中的各个模块设计。

3.5.2　设计细则

3.5.2.1　用户接口模块

用户接口模块是面向用户的，其战略目标是简化用户操作，提升用户体验质量。用户接口模块的设计主要包括以下几个子模块：

1、功能定义

功能定义子模块的主要用途是定义系统可以提供的功能。实际上，功能定义子模块和具体的应用场景密切相关。一个良好的功能定义子模块可以清晰地将系统及其提供的功能呈现在用户面前，很好地回答“系统是什么”以及“系统能做什么”等问题，留给用户良好的第一印象，并以提高用户体验质量为宗旨。

2、交互设计

交互设计子模块主要解决以下问题：用户如何与系统进行通信和交流、如何实现良好的人机交互等问题。常见的交互方式包括命令行客户端工具、Web 网页交互等。相比较而言，命令行客户端工具具有简洁准确和快速的特点，但他稍显复杂的操作方式显然比不上 Web 页面的可视化交互方式。不管采用哪种交互方式，都必须做到意图明了、操作简单和快速响应。

3、反馈设计

广义上来说，反馈设计子模块也可以算做是交互设计的一部分。反馈设计的重点在于系统将处理结果告知并反馈给提交请求的用户。通过将最终的处理结果反馈给用户，以指导用户做进一步的决策。

4、页面布局

页面布局子模块只针对 Web 网页交互方式。简单来说，页面布局子模块就是合理安排组件元素的呈现方式，如何将系统功能、交互元素等合理地组合布局，如何突出系统的重点，如何吸引用户的注意力等。

3.5.2.2　预处理模块

从资源映射交互系统的框架示意图3.7中可以看出，预处理模块被划归到系统层，实际上，预处理模块对接了用户层和系统层。预处理模块将用户层中用户提交的业务请求进行适当的整理和预处理，为系统层的进一步处理奠定了基础。预处理模块主要包含如下几个子模块：

1、信息抽取

信息抽取子模块通过从用户提交的业务请求中抽取出系统层需要的有用的信息，剔除冗余和无用的信息，以减小处理复杂度并提高效率。

2、需求解析

计算机和人类不同，计算机可理解的只是简单符号。因此，需求解析子模块通过对提取出来的信息进行分类处理或者合理地编排呈现，使得计算机可以很好地理解用户的业务请求，为资源的映射处理奠定了基础。

3.5.2.3 资源映射模块

资源映射模块是资源映射交互框架的核心，经过上述几个模块的处理，用户提交的业务需要被合理的解析和处理，能够被计算机识别和理解。此时，资源映射模块获取系统的可用资源，并进行映射匹配计算。资源映射模块主要包括以下几个子模块：

1、资源获取

资源获取子模块在资源监控模块的辅助下，访问获取系统的资源使用情况和可用资源情况，为资源映射匹配做好充分的准备。

2、映射匹配

映射匹配子模块的主要任务是将用户的业务请求和系统的可用资源进行匹配计算。在这里，我们采用的是本节提出的面向媒体业务的虚拟网络嵌入算法MSO-VNE，上节的仿真实验已验证了该算法的出色性能。

3.5.2.4 资源分配

当资源映射成功后，资源分配模块根据映射处理结果从底层物理系统中为用户的业务请求分配实际的真正的物理资源以为用户提供服务。资源分配模块的成功执行使得底层物理系统中的资源使用情况发生变化，因此，需要及时更新资源池中的资源情况和可用资源情况。因此，资源分配模块主要包括以下两个子模块：

1、分配

2、同步管理

3.5.2.5　资源监控

资源监控模块作为资源映射模块和资源池组件的辅助模块。一方面，资源监控模块为资源映射模块提供资源访问接口，查询访问底层物理系统的资源使用情况和可用资源情况；另一方面，资源监控模块随时监控资源池中资源的变化情况，譬如当请求到达系统时占用资源以及当请求生命周期结束后释放资源，无论什么情况，都要定期检测和同步资源池中资源使用情况和可用资源情况。资源监控模块主要包括以下两个子模块：

1、监控

2、同步管理

3.6　小结

本章提出了面向媒体业务的虚拟网络嵌入算法 MSO-VNE，并详细阐述了算法的主要思想和技术实现细节。MSO-VNE 算法根据虚拟网络请求和底层物理网络的地理位置特征，应用节点聚类算法，充分利用了分而治之思想和并发处理思想的优势，显著地降低虚拟网络嵌入处理的时间复杂度。同时，本章提出的 MSO-VNE 算法在局域嵌入阶段应用了动态服务均衡分析，充分考虑了节点在网络拓扑中的重要性，引入动态服务能力，通过贪心策略进行嵌入映射。

为了评估本章提出的面向媒体业务的虚拟网络嵌入算法 MSO-VNE 的性能，我们设计了仿真实验，将 MSO-VNE 算法与已有的 D-ViNE 算法和 R-ViNE 算法进行性能对比。仿真实验的结果表明本章提出的 MSO-VNE 算法具有很出色的性能：更低的时间复杂度、更好的负载均衡、更高的平均收益开销比和更高的平均接受率。

虽然，VNE 问题起源于网络虚拟化，但它仍属于资源分配和管理领域。因此，本章提出了面向媒体业务的资源映射交互框架，在该资源映射交互框架中，本章提出的 MSO-VNE 算法被采用，作为该资源映射交互框架的资源映射核心组件模块。MSO-VNE 算法在仿真实验中表现出的良好性能，给了我们很大的信心，使我们更加相信基于 MSO-VNE 算法的资源映射交互框架可以获得更好的 QoS 服务质量和更好的用户体验质量。

第四章　业务流量隔离和控制

4.1　引言

在传统的 TCP/IP 网络中，所有 IP 数据包的传输都是采用 FIFO 先进先出的方式，通过“尽力而为”的传输处理机制，尽可能地传输数据包。随着计算机网络的发展，数据量的急剧增长，以及多媒体数据对 QoS 服务质量的高要求，使得研究同一个系统中的业务流量隔离和控制变得越来越重要。

通常，流量隔离包含双重含义：

1、流量之间相互不可见

2、流量之间相互不影响

流量之间的不可见隔离可以通过 VLAN 来实现。VLAN 将局域网内的设备根据应用的不同，以软件的方式逻辑地划分并管理工作组。在 VLAN 虚拟局域网中，同一个 VLAN 内的成员可相互通信，而不同 VLAN 之间是相互隔离和不可见的。

流量之间的不影响隔离以及流量控制都可通过引入 QoS 来实现。QoS 可区分不同的数据流量，根据业务数据的重要程度、紧急程度以及优先级而区分对待。针对各种不同的需求，提供不同服务质量的网络服务。不同的数据流量根据其不同的 QoS 等级，获得对应等级的网络带宽等网络资源保障，并保证不干扰、不抢占其他的数据流量。

本章结合 Linux TC 流量控制模块的工作原理和关键技术，通过对 Linux TC 流量控制模块进行层次性设计和封装，提出了业务流量隔离和控制系统。

本章的内容安排如下：第 4.2 节简要介绍了 Linux TC 流量控制的原理和核心技术；第 4.3 节详细阐述了本章设计提出的业务流量隔离和控制系统的整体架构以及设计原则；第 4.4 节设计仿真实验对本章提出的业务流量隔离和控制系统的性能进行评估；第 4.5 节对本章的内容进行总结。

4.2 流量控制原理

4.2.1 Linux TC

4.2.1.1 Linux TC 简介

Linux TC[65, 66]，Linux Traffic Control，是 Linux 操作系统中的流量控制器，Linux 内核网络协议栈从 2.2.x 开始提供该流量控制器模块。Linux TC 通过建立处理数据包的队列缓冲区，并限制和规定队列中的数据包被发送的方式和次序，从而实现对出口流量的控制。根据 Internet 的工作方式，我们无法直接控制别人向我们发送什么数据。因此，Linux TC 流量控制是针对从网络接口向外发送的数据流量。

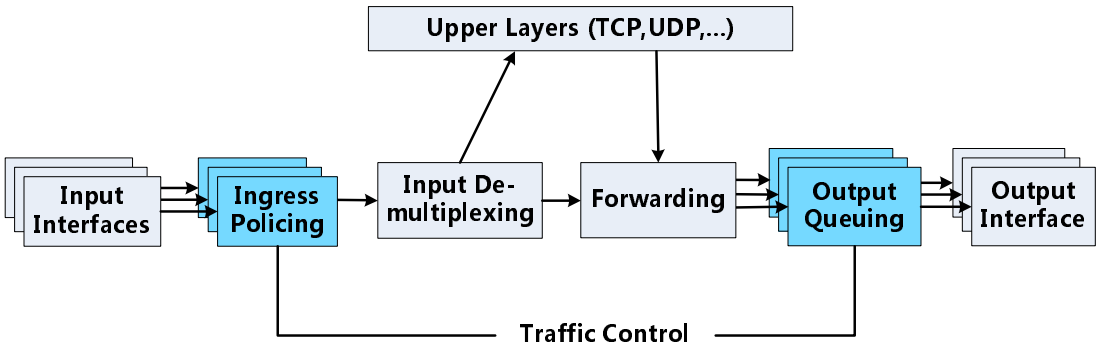

图 4.1: Linux TC 流量控制模块原理示意图

如图 4.1 所示，网络中的数据包从输入接口 Input Interface 进来后，经过入口流量监控模块 Ingress Policing，丢弃不符合规定的数据包，剩下的数据包进过输入多路分配器 Input De-Multiplexing 进行判断和选择：

1）如果数据的目的 IP 是本主机，那么将数据包送给上层处理；

2）否则，将接收包交到转发块 Forwarding Block 进行转发处理。

另外，本主机上层应用，譬如 TCP、UDP 应用，产生的数据包也通过转发块进行向外传输。转发块通过查看路由表，决定所处理数据包的下一跳。然后，对数据包进行排队 Output Queuing 并传送到输出接口 Output Interface，由输出接口将数据包传送到网络中。

由于 Internet 的工作方式，我们只能限制从网卡发送出去的数据包，不能限制从网卡接收进来的数据包。通过改变数据包被发送的方式和次序，可以控

制数据包的传输速率。Linux 流量控制工作在输出接口，在对数据包进行排队时实现流量控制。如图 4.1 所示，TC 流量控制模块包括入口流量限制和输出排队两个子模块。

4.2.1.2　Linux TC 流量控制方式

Linux TC 流量控制模块进行流量控制的方式包括以下几种：

1、流量整形

流量整形是一种主动调整流量输出速率的控制方式。流量整形通过主动限制网络连接向外发送的数据流量，有效地控制网络连接的传输速率，使得该网络连接的报文以比较均匀的速度向外发送。流量整形可以很好地平滑突发数据流量，使网络更加稳定。通常，可利用缓冲区和令牌桶来完成流量整形：当报文的发送速度过快时，首先在缓冲区中缓存数据报文，在令牌桶的控制下再均匀地发送这些被缓冲的数据报文。流量整形的缓存能够对数据包流量的完整性有较好的保存，但缓存也引入了延迟。通常，流量整形只适用于从网络接口向外发送的流量。

2、流量调度

流量调度通过在输入接口和输出接口之间设定一个数据包队列，让数据包在队列中排队，并重新规定数据包被发送的方式和次序。通过调度数据包的传输，可以在带宽范围内，按照优先级为数据包分配带宽。常见的调度策略是 FIFO 先进先出，这种调度策略几乎不用对数据包做任何处理，数据包按照它们到达的顺序依次发送。与流量整形类似，流量调度只适用于从网络接口向外发送的流量。

3、流量监管

流量监管，类似于流量整形，流量监管不仅可以限制从网络接口向外发送的数据流量，还可以限制流入网络接口的流量。流量监控的控制策略主要是丢包和重新标记，它不存在缓冲区或队列。当某个连接的报文流量过大时，流量监管通常是直接丢弃超额流量或是将超额流量标记为低优先级。因此，流量整形和流量监管的重要区别是，流量整形可能会因为对数据包流量的缓存而增加延迟，而流量监管几乎不引入额外的延迟。

4、丢弃

丢弃策略很简单，通过丢弃包、流量或分类来实现对网络流量的控制，通

常在流量超过某个设定的带宽时就会采取丢弃措施。丢弃策略不仅可以对从网络接口向外发送的数据流量实施管理，还可以管理并丢弃流入网络接口的数据流量，从而实现对网络流量的控制和平滑。

4.2.1.3 Linux TC 流量控制组件

在 Linux TC 流量控制模块中，主要的流量控制组件对象包括：

1、QDisc

QDisc 队列规定，它对数据被发送的方式和顺序都进行了限制和规定，以此实现对数据流量的控制和管理。QDisc 是典型的流量调度器，只适用于从网络接口向外发送的数据流量，它通过让数据包在队列中排队，并按照事先制定的发送方式和次序发送数据。QDisc 分为无类队列规定和分类队列规定。

2、Class

在分类队列规定中存在 Class 类别，以对不同的数据流量进行区分对待。一个类别可以是内部类别，也可以是叶子类别。内部类别包括很多子类别，而叶子类别不能有任何的子类别，而且必须包含一个简单队列规定或无类队列规定。

3、Filter

Filter 过滤器是 Linux TC 流量控制模块中的粘合组件，它能够很方便地粘合流量控制中的关键组件要素。过滤器可以被附加到分类队列规定或类别中，根据数据流量的特征将数据流量导向不同的队列或分类中。数据包首先进入到根队列规定中，当根队列规定的过滤器被遍历后，数据包会被引导流入到某一个子类中，该子类可以有自己的过滤器规则，继续对数据流量进行分类导流，直到最终数据包被发送出去为止。

4.2.2 QDisc

QDisc 队列规定是 Linux TC 流量控制模块的关键组件。无论什么时候，内核如果需要通过某个网络接口发送数据包，它都需要按照为这个网络接口配置的队列规定把数据包加入队列进行排队。然后，内核根据网络接口的队列规定所制定的数据发送方式和次序，尽可能多地从队列里取出数据包，把它们交给网络适配器的驱动模块，从而将数据包发送出去。 Linux TC 中的队列规定包括两种：无类别队列规定和分类队列规定。

4.2.2.1　无类别队列规定

无类别队列规定是对进入网络设备的数据流不加区分、统一对待的队列规定。这类队列规定形成的队列可以对整个网络设备的流量进行整形，但不能细分各种情况，不能对网络流量进行区分对待。无类别队列主要包括以下几种：

1、FIFO

FIFO[67] 先进先出队列规定是最简单的无类别队列规定。FIFO 对进入网络接口的数据包不做任何额外处理，数据包按照先进先出的方式通过队列。队列即是缓冲区，有一定的容量大小，可以暂时保存网络接口一时无法处理的数据包。 FIFO 队列规定可分为三种子类型：pfifo[68] 以数据包为单位区分不同的流量并进行排队处理；bfifo[69] 以字节为单位区分不同的流量并进行排队处理；pfifo_fast[70] 是系统默认的无类别队列规定，它包括三个不同优先级的波段，波段 0 的优先级最高，波段 2 的优先级最低。在每个波段里面，使用先进先出的排队规则。如果波段 0 里面有数据包，系统就不会处理波段 1 里的数据包，波段 1 和波段 2 之间的关系也是如此。内核遵照 TOS 字段对数据包进行标记，把不同的数据包分配到三个不同的波段中。

TOS，Type of Service，即服务类型，它包含 4 个 bit。TOS 字段的字节定义如表 4.1 所示。

二进制	十进制	含义
1000	8	最小延迟 Minimum Delay
0100	4	最大吞吐量 Maximum Throughout
0010	2	最大可靠性 Maximum Reliability
0001	1	最小成本 Minimum Costing
0000	0	正常服务

表 4.1: TOS 字段的 bit 定义

2、RED

RED[71, 72] 随机早期探测队列规定，通过监测队列的平均大小并基于统计概率随机地丢弃数据包。如图 4.2 所示，当带宽的占用接近于规定的带宽时，系统会随机地丢弃一些数据包。当缓冲区或队列为空时，所有传入的数据包都会被接受。随着缓冲区或队列被数据包填充，平均长度随之增加，因此，新传

入的数据包被丢弃的概率就增加。当缓冲区或队列满的时候，任何新传入的数据包都将被丢弃。RED 队列规定通常用来防止网络拥塞，是端到端的 TCP 拥塞控制的补充。它具有一定的网络拥塞预测能力，当预测到网络拥塞时，提前采取丢弃策略，而不是等到网络真正拥塞之后再采取补救措施。

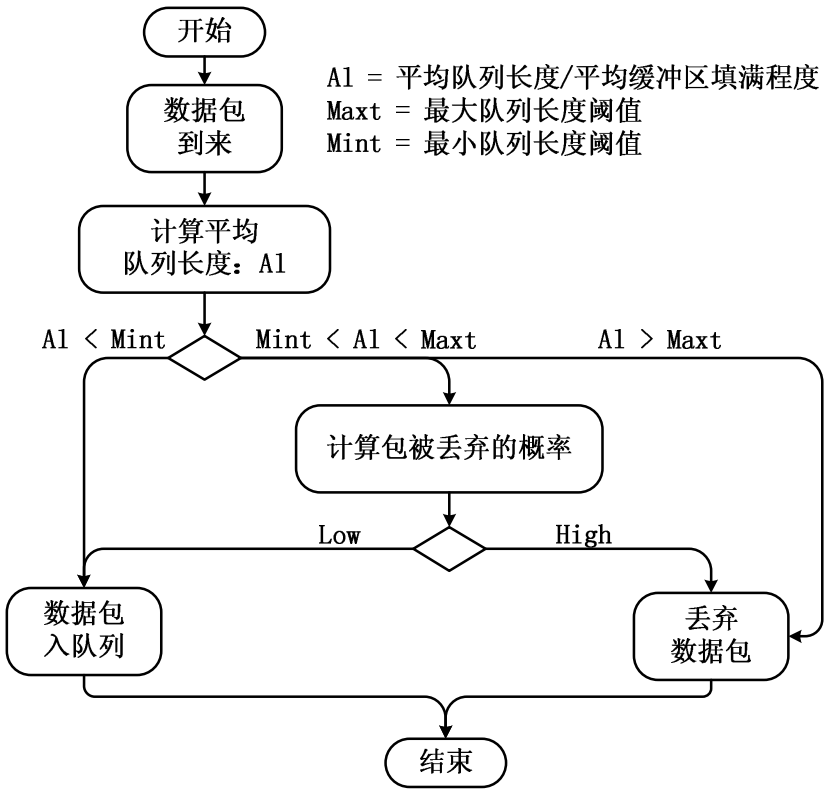

图 4.2: RED 队列规定的排队规则示意图

3、SFQ

SFQ[73] 随机公平队列，它的关键概念是“会话”或者“流”，即 TCP 会话或者 UDP 流。在 SFQ 随机公平队列中，数据流量按照其会话或流特征被分成一个个 TCP 会话或 UDP 流。每个会话或流的数据流量被导向到一个 FIFO 队列中，也就是说每个队列对应一个会话或流。数据按照简单轮转的方式被发送，每个会话都按顺序得到数据发送机会。因此， SFQ 非常公平，它保证了每个会话的数据发送不会被其它会话淹没。SFQ 的“随机”体现在它使用散列算法把所有的会话映射到有限的几个队列中去，而不是真正的为每个会话创建一个队列。由于散列操作可能使得多个会话被分配到同一个队列里，使得同一

个队列里的数据包共享数据发送机会，即共享带宽。SFQ 通过频繁地改变散列算法控制这种共享效应带来的性能损失。需要注意的是，SFQ 随机公平队列只有当出口网卡确实已经被挤满时才会起作用。否则，网络接口中根本就不会有队列，SFQ 也就不会起作用。

4、TBF

TBF[74] 令牌桶过滤器队列规定，它只允许以不超过事先设定的速率到来的数据包通过，但可以允许短暂突发流量超过设定值。TBF 通过令牌桶缓冲器来实现。令牌桶不断地被“令牌”以特定速率填充，每个令牌都可以从数据队列中携带一个数据包发送出去，然后该令牌从令牌桶中移除，被携带的数据包也从数据队列中移除。令牌桶过滤器队列规定中最重要的变量是令牌桶的容量大小，它代表了能够存储的令牌数量。在 TBF 令牌桶过滤器中，令牌流速率V_{ticket}和数据流速率V_{data}的不同组合将产生以下不同的情形：

1）$V_{ticket} = V_{data}$，数据流的速率等于令牌流的速率。这种情况下，每个到来的数据包都能对应一个令牌，然后无延迟地通过队列。

2）$V_{ticket} < V_{data}$，数据流的速率小于令牌流的速率。通过队列的数据包只消耗了一部分令牌，剩下的令牌会在桶里积累下来，直到桶被装满。剩下的令牌可以在需要以高于令牌流速率发送数据流的时候消耗掉，这种情况下即是突发流量传输。

3）$V_{ticket} > V_{data}$，数据流的速率大于令牌流的速率。这意味着桶里的令牌很快就会被数据包耗尽，导致数据流的发送被短时间中断。此时，如果数据包持续到来，将发生丢包。

最后一种情形正是令牌桶过滤器队列规定的整形限流作用。当数据流速率较低时，所有到来的数据包都能通过队列，并只消耗了部分令牌，剩余令牌慢慢积累起来；当数据流速率较高时，快速到来的数据流可在短时间内消耗掉令牌，出现流量突发传输；此时，如果数据流仍然保持高速率到达，由于令牌已被耗尽，数据包不再被发送，数据流被限制整形，使得数据包传输延迟甚至被丢弃。通常，实际的实现是针对数据的字节数而不是数据包进行的。

4.2.2.2　分类队列规定

分类队列规定是对进入网络设备的数据包以分类的方式区分对待的队列规定。数据包进入一个分类的队列后，它就需要被送到某一个类中，也就是说数

据包需要进行分类处理。对数据包进行分类的工具是过滤器，过滤器通过检测数据包的基本信息并返回一个决定，队列规定就根据这个决定把数据包送入相应的类或丢弃。在每个子类中，可以继续使用过滤器对数据包进行进一步细致的分类。最终，当数据包进入到叶子类时，数据包就不会再被过滤器分析，也不会被分类。此时，数据包根据叶子类的无类别队列规定进行排队并发送。分类队列主要包括以下几种：

1、CBQ

CBQ[75] 基于分类的队列，是一种基于网络调度的队列规定，它不仅可以用来对数据包进行分类，还可以实现数据流量整形。CBQ 的分类可以基于不同的参数：优先级、应用程序类型、端口等。CBQ 允许数据流量在被分类分组后共享带宽，既有限制带宽的能力，也有带宽优先级管理的能力。带宽限制，也就是流量整形。CBQ 的流量整形是通过计算连接的空闲时间完成的，如果试图把一个 10Mbps 的连接整形成 1Mbps ，就应该让链路在 90% 的时间处于闲置状态。由于链路闲置时间非常难于测量，CBQ 采用近似值，即两个传输请求之间的毫秒数，以近似表征链路的繁忙程度。

2、HTB

HTB[76] 分层的令牌桶，基于令牌和桶，实现了一个复杂而精细的流量控制方法。HTB 允许用户创建一系列简单令牌桶并对令牌桶归类，实现细粒度的流量控制。HTB 通过控制令牌桶中令牌的速率实现流量整形。令牌的速率是一定的，而数据以变化的速率到达数据队列缓冲区。数据队列中的数据必须在有令牌的情况下才可以被发送，通过这种方式来限制数据的出口带宽速率。另外，HTB 的租借模型，可以在保证每个类别的带宽的同时，允许特定的类可以突破其带宽上限，占用别的类的闲置带宽。譬如，当子类的流量超过了设定的速率后，它可以向父类借用令牌，直到子类的可用令牌数量使得它达到其最大速率为止。

3、PRIO

PRIO 分类队列规定，根据事先配置的过滤器把流量进一步细分并进行后续处理。PRIO 队列规定可以视为 pfifo_fast 的衍生物，区别在于 pfifo_fast 的每个波段在 PRIO 队列规定中都是一个单独的类，可以为该类配置各种不同的无类别队列规定，而不是固定的 FIFO 先进先出队列规定。另外， pfifo_fast 不能使用 TC 配置命令进行自定义配置，而 PRIO 则可以由用户根据自己的需要进

行自定义配置和修改。PRIO 不能限制带宽，从而不能对流量进行整形，因为属于不同类别的数据包是顺序离队的。但 PRIO 可以很容易对流量进行优先级管理，只有属于高优先级类别的数据包全部发送完毕，才会发送属于低优先级类别的数据包。

4.2.3　配置管理

命令	适用范围	描述
add	QDisc, Class, Filter	在一个节点里添加一个队列规定、类别或过滤器。添加时，需要传递 parent 参数，该参数可使用 ID 或设备的根来标识。如果要建立一个队列规定或过滤器，可使用句柄来命名；如果要建立一个类别，可使用类识别符来命名
remove	QDisc	删除某个句柄指定的队列规定，根队列规定也可以删除。被删除的队列规定上的所有子类以及附属于各个类的过滤器都会被自动删除
change	QDisc, Class, Filter	以替代的方式修改条目。句柄 handle 和 parent 条目不能修改，也不能移除任何节点
replace	QDisc, Class, Filter	添加或删除一个节点，是原子操作。如果节点不存在，则建立节点
link	QDisc	替代一个存在的节点

表 4.2: TC的基本操作命令

在 Linux TC 中，所有的队列规则、类别和过滤器都有标识符 ID 的，该标识符可以由用户手动设置，也可以由内核系统自动生成和分配。ID 由一个主序列号 $MajorID$ 和一个从序列号 $MinorID$ 组成，两个数字用一个冒号分开，其基本格式如公式 4.1 所示。

$$MajorID : MinorID \tag{4.1}$$

队列规则的主序列号，也可以称作句柄 $handle$，队列规则的从序列号是类的命名空间。句柄采用象 $MajorID:$ 一样的表达方式。习惯上，需要为含子类的队列规则显式地分配一个句柄。在同一个队列规定的类共享该队列规定的主序列号，但是每个类都有自己的从序列号，即类识别符 $ClassID$。类识别符只与父队列规定有关，和父类无关。类的命名习惯和队列规定的命名习惯相同。

在 Linux TC 中，所有的配置管理都是采用终端命令进行的。如表4.2所示的是 Linux TC 的基本操作命令，可对 QDisc、Class 和 Filter 进行相应的配置管理。

4.3 业务流量隔离和控制方案

本节将详细介绍业务流量隔离和控制系统的设计方案，包括整体架构、设计原则以及功能接口。

4.3.1 整体架构

业务流量隔离和控制系统的整体架构如图 4.3 所示。在该架构中，所有的流量控制组件组成一棵分层的树形结构。其中，树的根是分类复杂队列，目前支持 CBQ 和 HTB 两种分类队列规定。在根队列下，是一个根类别，所有将要从网络接口往外发送的数据包都将经过根队列和根类别。根队列的标识符或句柄是 1 :，根队列下的根类的标识符为 1 : 0 。在根类别下，我们为每个用户的数据流量指定一个分类别，以有效地隔离用户，避免用户之间的流量抢占现象。同时，我们还提供了独立的分类和队列规定，并面向用户开放一定的控制接口以方便用户对其业务实现精细控制和适度调整。当数据包从根类别离开时，会根据数据包的 IP 地址区分出数据流量所属的用户，从而将不同用户的流量导入正确的分类别中。

对每个用户而言，我们将每个用户的流量分为两个类别：inner 内部流量类别和 outer 外部流量类别。内部流量主要包括用户业务的不同模块之间的内部通信流量，这部分流量不会通过外网流进互联网中。而外部流量主要是指为终端用户提供服务而产生的流量。对用户业务而言，最重要的是为终端用户提供有 QoS 保障的服务，因此，我们为外部服务流量预设了 5 个不同的分类别。用户可根据自己的业务需求，选择性地配置其中的若干个分类别和相应的队列规定或排队规则。

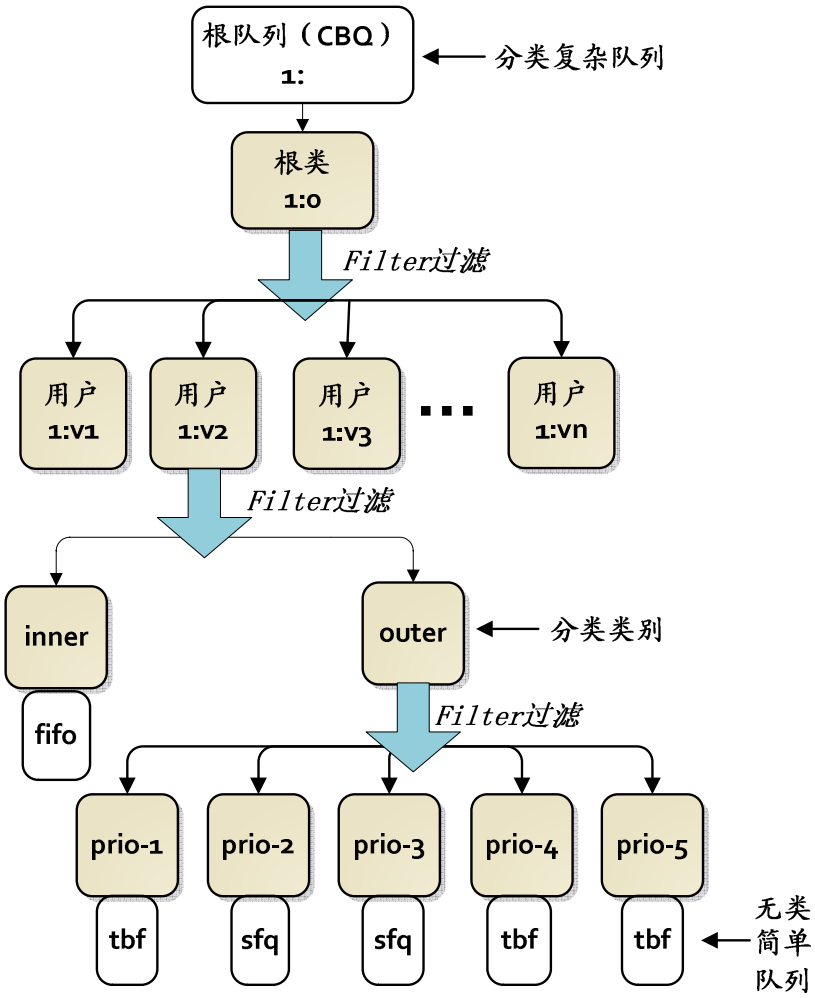

图 4.3: 业务流量和隔离模块整体框架

4.3.2　设计细则

对于业务流量隔离和控制系统的整体框架设计，我们主要遵循以下两个大原则：

1、隔离不同用户之间的业务

2、隔离同一用户的不同流量

因此，我们采用了层次设计，自上而下细分流量并进行隔离控制以保障较好的 QoS 服务质量。从根队列开始，进入到网络接口的数据流量会直接通过根类，然后，根据数据包的 IP 地址区分并标识不同的用户数据流量，并将

流量导向正确的用户类别。对于每个用户而言，我们将其数据流量分为 inner 流量和 outer 流量。 inner 流量是指业务内部的流量，而 outer 流量是从外部 Internet 访问用户业务的数据流量。根据 outer 外部流量的数据包特征，我们又进一步细分了 5 个子类别，以区分出 5 个不同优先级的流量，并给用户开放了自定义分类规则的接口。通常，用户对自己的业务具有绝对的知悉和掌握，他可以很准确地区分不同数据流量的重要程度和优先级情况。因此，给用户一定的自由度去定制分类规则是很有必要的。

具体来讲，本章的业务流量隔离和控制系统遵循了如下设计细则：

1、层次性

层次性主要体现在自上而下的分层设计和不同粒度的流量控制。这种层次性设计的优势在于直观明朗的系统框架。层次性设计不仅简化了系统的设计，更减轻了系统的工程开发工作。同时，层次性设计大大提高了系统的横向扩展能力，当用户增加时，横向扩展非常方便，几乎不用修改系统框架。

2、封装性

封装性是指整个系统封装了基本的 TC 终端命令行操作，如 add，remove，change 等操作。通过统一的功能接口和控制接口，隐藏了所有 TC 终端命令行的繁复参数和细节。由于 TC 流量控制的队列规定、分类规则以及过滤规则的参数众多，如果完全由用户分别一一输入，其用户体验质量将非常糟糕。另外，TC 流量控制的众多参数使得它非常灵活，可塑性很强。但对于业务流量隔离和控制系统而言，我们并不需要太过复杂和花哨的细致调节。因此，我们通过封装流量隔离控制模块，改善用户的操作方式，并将开放给用户的配置参数进行一定的精简，从而简化了系统的设计和开发，以提高用户体验质量。

3、开放性

开放性是指系统中各个分类的规则以及排队的规则都是可以由用户通过开放的接口进行自定义的管理和配置的。通过给用户一定的自由度，让系统更加丰富和人性化。通常，用户是对其业务最熟悉的人，他明确地知道自己的不同业务流量之间的重要性和优先级，所以，将系统的微调配置开放给用户是最有效的。

4.3.3　功能接口

Linux TC 流量控制器作为 Linux 内核的一个功能模块，其功能非常强大，也非常灵活。当然，Linux TC 可配置和可调节的参数众多，可塑性很强。但在我们的应用场景下，我们最关心的是带宽限制。我们希望不同的用户在其可“活动”的带宽范围内发送数据包，安全地运转业务并提供服务，避免不同用户的相互干扰，也避免不用业务的相互干扰。

目前，我们开放给用户的功能接口 API 主要包括以下四个：

1、启动——euca-setup-trafficcontrol

2、配置——euca-config-trafficcontrol

3、查询——euca-describe-trafficcontrol

4、终止——euca-stop-trafficcontrol

下面，我们将详细地介绍这些功能接口 API 。

4.3.3.1　euca-setup-trafficcontrol

接口 euca-setup-trafficcontrol 用于启动业务流量隔离和控制模块。如果系统从未启动过该模块，则采用系统默认的配置；如果系统曾经启动过该模块并实施过重新配置，则采用最新的配置。其完整的接口 API 形式如下：

euca-setup-trafficcontrol [-d –device] [-q –qdisc] [-u –useree]

该接口的具体的参数说明如表 4.3 所示。

参数	缩写	描述	默认值
device	d	需要进行流量控制的网络接口	eth0
qdisc	q	TC 的根队列类型，支持 CBQ 和 HTB	CBQ
useree	u	将要启动流量控制的用户	当前登录用户
实例	euca-setup-trafficcontrol		

表 4.3: euca-setup-trafficcontrol 的详细参数说明

4.3.3.2　euca-config-trafficcontrol

接口 euca-config-trafficcontrol 用于配置已启动的业务流量隔离和控制模块，其完整的接口 API 形式如下：

euca-config-trafficcontrol [-d –device] [-l –level] [-q –qdisc] [-b –bandwidth] [-p –prio] [–limit] [–latency] [–perturb] [–quantum] [–burst] [–addfilter] [–rmfilter] [–bandwidth-list] [–qdisc-list] [-u –useree]

该接口的具体参数说明如表 4.4 所示。

参数	缩写	描述
device	d	需要进行流量控制的网络接口，默认为 eth0
level	l	分类类别，包括 vlanroot，inner，outer，prio1 到 prio5
qdisc	q	叶子队列规定的类型，支持 SFQ 和 TBF ，须指定 level
useree	u	将要启动流量控制的用户，默认为当前登录用户
bandwidth	b	分类类别的带宽和速率，须指定 level 参数
prio	p	分类类别的优先级，须指定 level 参数
limit		TBF 队列的参数 limit，须指定 level 且 qdisc 为 tbf
latency		TBF 队列的参数 latency，须指定 level 且 qdisc 为 tbf
burst		TBF 队列的参数 burst，须指定 level 和 qdisc 为 tbf
perturb		SFQ 队列的参数 perturb，须指定 level 且 qdisc 为 sfq
quantum		SFQ 队列的参数 quantum，须指定 level 和 qdisc 为 sfq
addfilter		增加过滤规则，必须指定 level
rmfilter		删除过滤规则，必须指定 level

表 4.4: euca-config-trafficcontrol 的详细参数说明

下面列举了几个使用接口 euca-config-trafficcontrol 进行实际配置的实例。

（1）修改 inner 叶子分类类别的带宽为 30mbit

euca-config-trafficcontrol -l inner -b 30mbit

（2）修改 inner 叶子分类类别的无类简单队列为 tbf，并指定参数 latency 和 burst

euca-config-trafficcontrol -l inner -q tbf –latency 50ms –burst 5kb

（3）修改 inner 叶子分类类别的无类简单队列为 sfq，并指定参数 perturb 和 quantum

euca-config-trafficcontrol -l inner -q sfq –perturb 8 –quantum 1514

（4）移除 inner 叶子分类类别的过滤规则 match u8 0x40 0xf0 at 0

euca-config-trafficcontrol -l prio5 –rmfilter “match u8 0x40 0xf0 at 0”

（5）为 inner 叶子分类类别添加过滤规则 match u8 0x40 0xf0 at 0

euca-config-trafficcontrol -l prio5 –addfilter “match u8 0x40 0xf0 at 0”

4.3.3.3 euca-describe-trafficcontrol

接口 euca-describe-trafficcontrol 用于查询显示已启动的流量控制模块，其完整的接口形式如下：

euca-describe-trafficcontrol [-d –device] [-u –useree]

该接口具体的参数说明如表 4.5 所示。

参数	缩写	描述	默认值
device	d	需要查询流量控制的网络接口	eth0
useree	u	要查询流量控制的用户	当前登录用户
实例	euca-describe-trafficcontrol		

表 4.5: euca-describe-trafficcontrol 的详细参数说明

4.3.3.4 euca-stop-trafficcontrol

接口 euca-stop-trafficcontrol 用于停止已启动的流量控制模块，其完整的接口形式如下：

euca-stop-trafficcontrol [-d –device] [-u –useree]

该接口的具体的参数说明如表 4.6 所示。

参数	缩写	描述	默认值
device	d	需要终止流量控制的网络接口	eth0
useree	u	要终止流量控制的用户	当前登录用户
实例	euca-stop-trafficcontrol		

表 4.6: euca-stop-trafficcontrol 的详细参数说明

4.4 实验验证

为了测试业务流量隔离与控制模块的相关性能，我们设计了仿真实验进行测试。

4.4.1 实验环境

通常，网络问题都是非常复杂的，尤其是涉及到 Internet 互联网的网络问题。为了有效地测试和验证业务流量隔离与控制模块的性能，我们必须选择合适的网络结构。在实验中，我们设计了相对封闭的网络结构，如图 4.4 所示。其中，两台主机位于同一个局域网络中，使得互联网的复杂性和不确定性大大减小。主机 192.168.155.52，简称 52 主机，和主机 192.168.155.218，简称 218 主机，通过交换机直接相连。

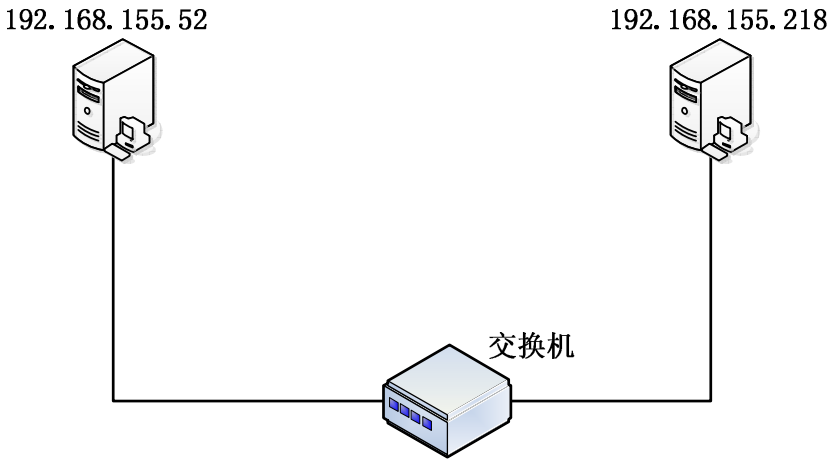

图 4.4: 业务流量隔离和控制实验框架示意图

在一系列的实验中，我们让 52 主机作为发送主机，实现并安装了发送应用程序 tcsender，218 主机是接收主机，实现并安装了接收应用程序 receiver。

4.4.2 实验内容

为了充分评估和验证业务流量隔离和控制模块的性能，我们设计了如下三个实验内容以全面评估业务流量隔离和控制模块。

4.4.2.1　链路带宽速率测量

如图 4.4 所示的业务流量隔离和控制实验架构是相对封闭和简单的，即便是这样简单的设计，也不能完全排除网络自身特征中的复杂性和不确定性。52 主机和 218 主机之间的链路带宽不仅与交换机有关，还与链路网线、主机网卡等众多因素相关。因此，我们设计实验方案来测量 52 主机和 218 主机之间的链路带宽。我们采用了如图 4.5 所示的网络架构进行链路带宽的测量。通过 52 主机不断地向 218 主机发送数据包，统计并计算数据发送的平均速率。为了使结果更真实可靠，我们设计了多组对比实验，tcsender 通过发送不同数量的数据，并查看 tcreceiver 接收完数据所需要的时间。

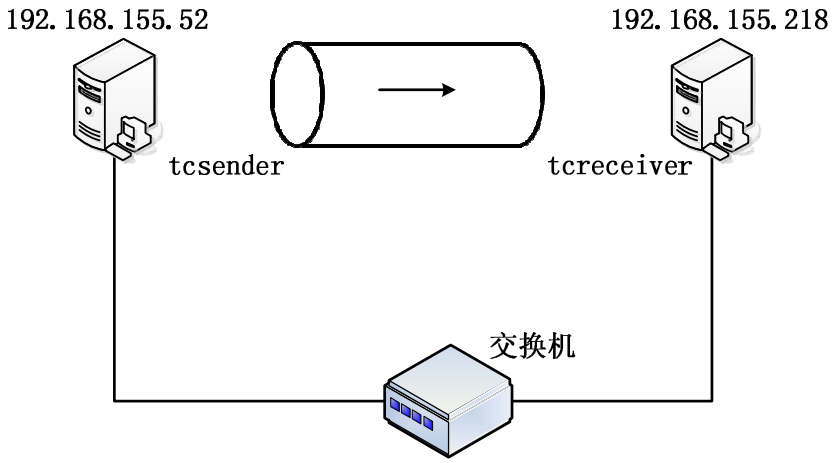

图 4.5: 测量链路带宽的实验架构示意图

4.4.2.2　业务之间的相互干扰

在没有业务流量隔离和控制系统的网络环境中，同时存在的业务可能存在抢占流量等问题，从而导致业务之间的相互影响和干扰，使得各业务的可用带宽和数据传输速率受到影响而上下波动，严重影响了业务服务的可用性和稳定性。为此，我们设计实验测量同一链路上不同业务之间的相互干扰和影响情况。如图 4.6 所示，通过在 52 主机上运行两个不同的业务，测量并统计各个业务的数据传输速率。在具体实验中，我们假定业务 A 是相对稳定的业务，需要传输的数据都趋于平稳，而业务 B 是多变的业务，其需要传输的数据随着时间的推移而不断变化。从宏观上来说，我们可以把业务 B 当做是业务 A 的变化

的外界网络环境，通过不断变化业务 B 的数据传输从而模拟复杂多变的外界环境对业务 A 的影响。

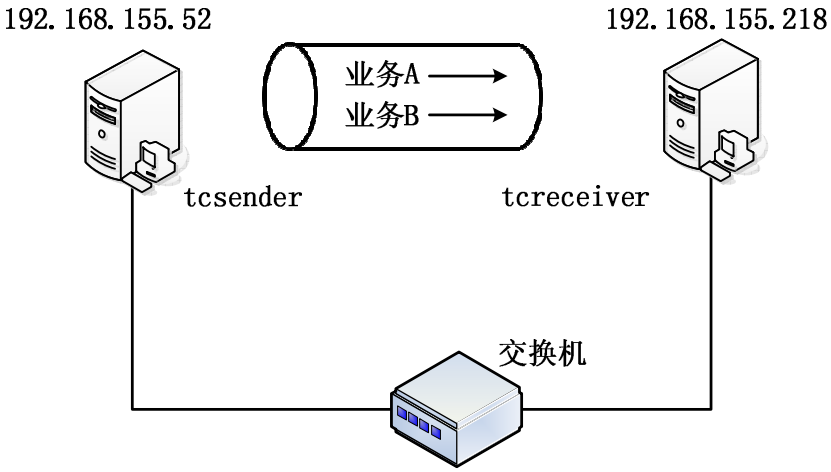

图 4.6: 测量链路上业务之间扰动的实验架构示意图

4.4.2.3 流量隔离控制性能

为了消除不同业务之间的数据流量的干扰影响，我们在 52 主机和 218 主机的网络接口上设置并开启业务流量隔离和控制模块。通过规定数据流量排队和发送的方式和次序，可控制网络接口向外发送的流量速率并隔离不同的业务。因此，我们设计如图 4.7 所示的实验架构图，测量并评估业务流量隔离和控制系统的性能。流量控制就相当于在链路上设置了栅栏，只允许一定流量的数据通过，且优先级高的数据优先通行，以此来达到隔离和控制流量的目的。

4.4.3 实验评估

4.4.3.1 链路带宽

如图 4.8 所示，是 52 主机和 218 主机之间的链路带宽策测量结果。采用如图 4.5 所示的实验网络架构，通过在 52 主机上向 218 主机发送不同的数据量，统计并计算数据被发送的平均速率。在链路带宽的测量中，链路是开放的，没有设置任何的排队规则和限流规则。实验测量结果如图 4.8 所示，52 主机和 218 主机之间的链路带宽约为 12MBps 。从图中可以看出，52 主机和 218 主机之间的数据传输速率是相对较稳定的，没有太大的波动，也没有任何的突变。

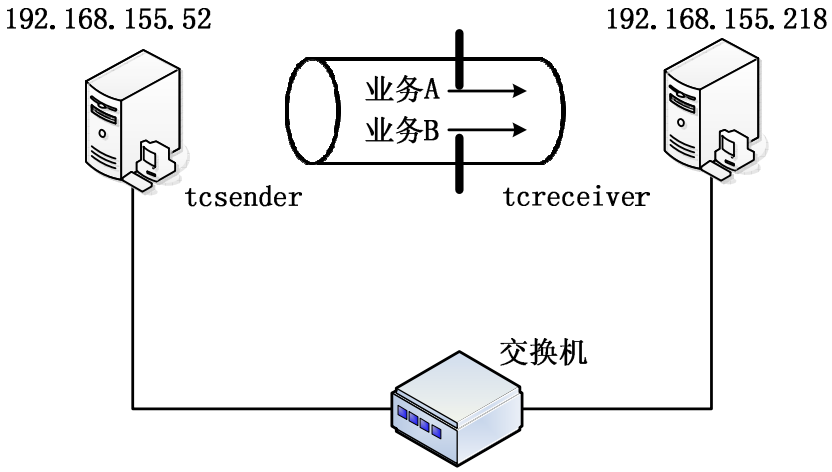

图 4.7: 流量隔离控制性能的实验架构示意图

由此可知，52 主机和 218 主机所在的网络环境是相对封闭和稳定的。这种相对封闭的网络环境使得业务流量隔离和控制的性能评估具有较高的准确性。

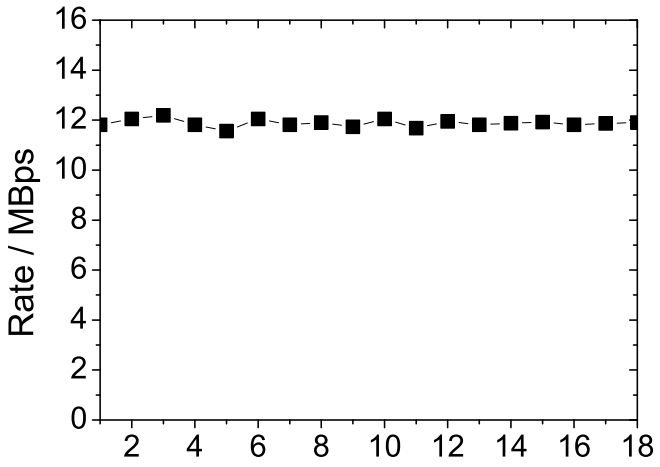

图 4.8: 链路带宽速率测量结果

4.4.3.2　业务干扰特征

如图 4.9 所示的实验结果是在没有业务流量隔离和控制时，同时存在的两个业务之间的互相干扰情况。实验中，fixed 标记的业务有相对稳定的数据发送量，而 variable 标记的业务，其数据传输量是变化的。我们通过 variable 这种

变化的数据传输量来模拟变化多端的外部网络环境对 fixed 业务的影响。从实验结果曲线图，我们可以很明显地看出，两个业务之间的干扰是很明显的。这也说明了在同一个系统中，业务流量隔离和控制模块的重要性和急迫性。

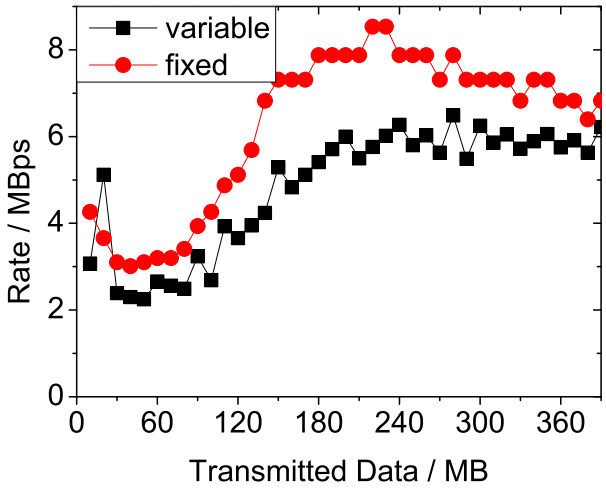

图 4.9: 在无流量隔离控制时，业务之间存在干扰

4.4.3.3 流量隔离控制性能

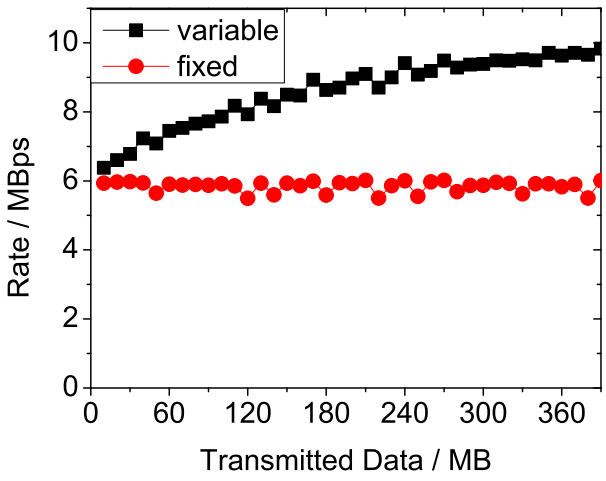

图 4.10: 在流量隔离控制下，业务的隔离情况

如图 4.10 所示，当实施了业务流量隔离和控制后，业务之间的流量得到了很好的隔离。fixed 标记的业务有相对稳定的数据发送量，而 variable 标记的

业务具有变化的数据传输量，可用 variable 流量来模拟变化多端的外部网络环境。在业务流量隔离和控制模块的作用下，即使 variable 标记的业务流量发生天翻地覆般变化，对 fixed 业务的影响几乎都可以忽略不计。

4.5　小结

本章通过详细阐述 Linux TC 流量控制模块的原理和技术细节，设计并提出了基于 Linux TC 的业务流量隔离和控制框架。本章提出的业务流量隔离和控制模块采用层次结构设计，封装了 Linux TC 流量控制模块的繁复的终端命令和参数，并开放一定的 API 接口用于用户根据其业务需要自定义配置和管理业务流量隔离和控制模块。该业务流量隔离和控制模块在面对用户规模扩张时能够从容应对，另外，该模块在设计中充分考虑了用户接口的设计以尽可能提高用户体验质量。

为了评估本章提出的业务流量隔离和控制模块的性能，我们设计了仿真实验，通过构造相对封闭的网络环境，测试网络链路的带宽速率，并对比实施业务流量隔离和控制模块前后的网络性能和多个业务各自的数据流量特征。实验结果表明，本章提出的业务流量隔离和控制模块具有很好的流量隔离性，对于营造安全隔离的网络通信环境有重要作用。

第五章　SuperNova 系统工程开发

5.1　引言

SuperNova 是一个层次结构的服务虚拟化平台，它通过屏蔽底层硬件设备并隐藏实现细节，以开放接口 API 的方式向应用提供所需的服务，而应用无需了解底层的硬件情况和具体的实现细节，根据业务的需求便可获得弹性服务。SuperNova 服务虚拟化平台提供一种基于服务虚拟化、具有云服务特征、开放接口的网络系统及实现方法，通过将分布在不同地理位置的物理设备连接起来，采用虚拟化技术以大粒度服务的形式对外提供资源，从而达到多种业务共享硬件资源，满足用户对媒体业务的服务质量需求。

SuperNova 系统的工程开发工作主要包括以下几个部分：

1、将第三章所述的面向媒体业务的虚拟网络嵌入系统应用到 SuperNova 服务虚拟化平台中，以有效地提高资源嵌入映射的效率和底层物理资源的利用率以及长期收益。

2、将第四章所述的业务流量隔离和控制系统应用到 SuperNova 服务虚拟化平台中，以隔离系统中不同的业务流量，并区分不同特征的数据流量，通过流量控制以提供有 QoS 保障的服务。

3、完善 SuperNova 服务虚拟化平台的功能，提供可视化操作方式，简化业务消息处理流程，提高用户体验质量。

本章的内容安排如下：第 5.2 节介绍了 SuperNova 服务虚拟化平台的系统架构和编程架构；第 5.3 节详细阐述了 SuperNova 服务虚拟化平台的业务流量隔离和控制模块，不仅包括编程架构和实现细节，还包括功能接口和使用方式；第 5.4 节详细阐述了虚拟网络嵌入模块在 SuperNova 服务虚拟化平台上的工程实现，主要包括对该模块的工程设计简化，工程开发实现流程，以及功能接口和使用方式。第 5.5 节介绍了 SuperNova 服务虚拟化平台的云控制器模块，通过完善云控制器的功能以简化消息处理流程并提高用户体验质量；第 5.6 节对本章的内容进行总结。

5.2 SuperNova

5.2.1 系统架构

SuperNova 服务虚拟化平台的系统架构示意图如图 5.1 所示。由此可知，该平台主要包括 4 大层次部分：物理设备层，统一的服务虚拟、控制和支撑层，服务和应用层，全局认知管理平台。

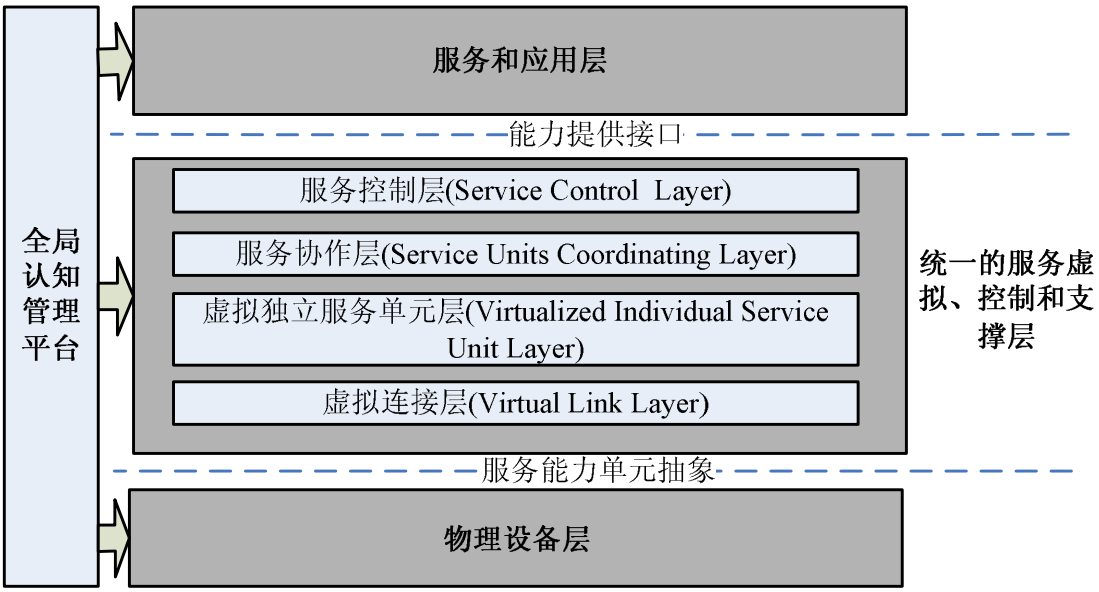

图 5.1: SuperNova 系统架构示意图

其中，物理设备层是众多物理服务器的集合。物理设备层统一向上层提供以节点为单位的信息，包括每个节点的资源状态和使用情况。服务和应用层是指业务和应用系统的集合，由统一的服务虚拟、控制和支撑层向其提供服务能力。统一的服务虚拟、控制和支撑层是 SuperNova 的核心层，它连接了物理设备层和服务应用层，通过屏蔽物理设备层的服务器硬件设备，并以接口的方式向服务和应用层提供所需服务，因此，应用无需了解底层的网络细节和硬件情况，可根据具体的业务和应用需求获得弹性服务。在 SuperNova 中，全局认知管理平台具有全局监控、优化和反馈驱动的功能，对于优化并提高系统性能有重要的作用。

下面将详细介绍 SuperNova 服务虚拟化平台中的各个层次组件。

5.2.1.1　物理设备层

物理设备层对物理服务器硬件设备进行管理和抽象，生成、实现不同的服务类型，并向统一的服务虚拟、控制和支撑层提供具有不同能力的节点信息。这些信息作为服务虚拟化平台的输入，服务虚拟化平台将这些节点组织为一个可针对不同服务类型进行扩展的虚拟服务网。

5.2.1.2　统一的服务虚拟、控制和支持层

统一的服务虚拟、控制和支持层的主要任务是组织物理设备层提供的大量节点信息，以便为不同的业务和应用类型提供服务。该层包括四个子层：虚拟连接层、虚拟独立服务单元层、服务协作层和服务控制层。

1、虚拟连接层

虚拟连接层对物理设备层提供的所有节点进行连接组网，形成一个全局可路由、可快速定位、高扩展性的自组织网络，从而可在物理资源之间安全地交换数据。

虚拟连接层的结构示意图如图 5.2 所示。虚拟连接层采用基于地理位置信息编码的网络结构，包括局域管理层和连接层。局域管理层主要负责管理不同地理区域内的节点，或是根据链路状态组成的区域内的节点。局域管理服务器由其中的某个节点担任。连接层通过结构化 P2P 方式将所有节点组织连接起来。虚拟连接层的主要工作是负责对物理设备层所提供的节点进行准确路由和查询。在虚拟连接层中，地理位置信息的引入，保证了 P2P 网络中节点的区域聚集性，提高了虚拟连接层的可扩展性，降低了管理服务器的负载，提升了用户体验质量。

如图 5.2 所示，全局管理服务器是一个全局设备，每个新加入到服务虚拟化平台的节点都需要首先在全局管理服务器上注册，汇报其 IP 地址、地理位置以及相应的服务能力信息。事实上，服务虚拟化平台中的节点会周期性查询自身可提供的服务能力状况，并通过带宽和延迟探测方法探测网络中链路状况，向局域管理服务器汇报，局域管理服务器再向全局管理服务器汇报。全局管理服务器会对其收集到的信息进行分析和整合，并将分析结果返回给局域管理服务器，以便局域管理服务器进行服务分配计算时作出更好的决策。另外，节点会从系统中所有节点中选择一部分节点作为该节点的优先邻居节点，并且定时相互交换邻居信息。

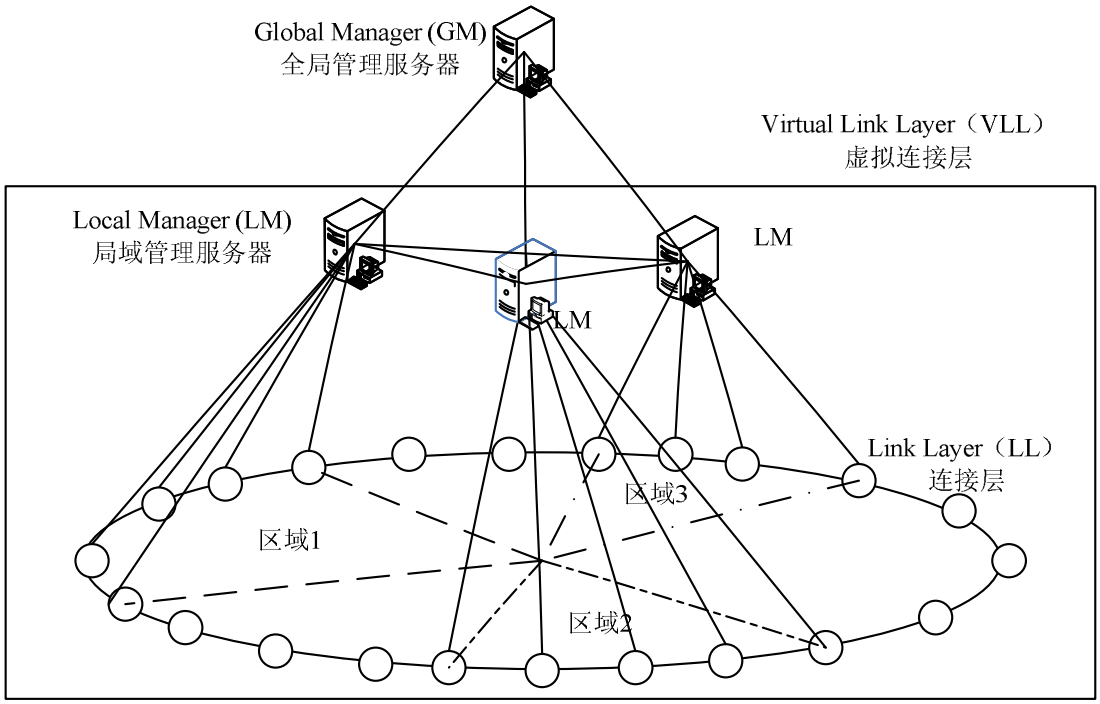

图 5.2: 虚拟连接层的结构示意图

当服务请求到达时，全局管理服务器根据服务请求的目标地理区域信息，将该服务请求分派到某个局域管理服务器，局域管理服务器对该服务请求任务进行分析，计算出服务方式，并为所请求的服务返回合适的入口节点。

2、虚拟独立服务单元层

虚拟独立服务单元层通过采用基于服务单元的属性封装模型，将单个节点提供的服务虚拟化为多个具有不同属性的服务单元。服务单元属性封装模型隐藏了不同硬件服务器设备之间的差异，提取出这些设备之间的共同特征并进行抽象，为上层提供统一的调用方式。在虚拟独立服务单元层中，服务单元即是最小粒度服务单位，是对节点物理资源、服务能力的封装和抽象。通常，基本服务单元只提供单一的功能，譬如存储、流化、计算等。另外，虚拟独立服务单元层还对网络链路状态进行探测，并将探测的信息作为节点属性进行封装。虚拟独立服务单元层位于虚拟连接层和服务协作层之间，通过利用虚拟连接层提供的接口进行通信，并对服务协作层开放管理接口。

3、服务协作层

服务协作层，顾名思义，是对虚拟独立服务单元层提供的最小粒度服务单

元进行组织协作，以为不同用户的不同需求、不同的服务类型以及不同的应用特征提供具有弹性能力的网络服务，同时屏蔽底层细节。服务协作层不仅要充分考虑不同服务单元的特征，还要兼顾底层网络的实时特征，以便进行有效的服务组织和协作。服务协作层使得能力结构能更好地配合实际硬件的分布状况，兼顾系统容错和负载均衡等多个指标，有效地提高系统服务性能。

4、服务控制层

服务控制层对服务协作层进行屏蔽，为服务和应用层提供服务。因此，用户的服务和应用无需了解服务协作层的实现细节，仅通过服务控制层定义的服务接口即可提出需求，并从系统中获得服务。

5.2.1.3　全局认知管理层

服务虚拟化平台 SuperNova 的层次结构使各层之间得到很好的屏蔽，各层可以独立地扩展或改变而互不影响。因此，系统的兼容性和扩展性都非常好，然而，这种屏蔽和隔离也有它的缺点。由于各层之间缺乏了解和信息交流，缺乏全局知识，使得服务单元的调度和控制只能得到次优的结果。因此，在服务虚拟化平台 SuperNova 中引入了全局认知管理层，来承担网络服务管理器的功能，通过不断的观察和学习，以达到全局优化的实施结果。同时，使服务虚拟化平台 SuperNova 更好地理解和符合用户需求。

通常，全局认知管理层可用一个单独的服务器来实现，其余各层通过开放的接口周期性地向全局认知管理层汇报当前状态、用户访问情况以及硬件设备情况等信息。全局认知管理平台根据所搜集的信息计算出不同用户账号下的文件访问情况和趋势、用户行为变化函数、硬件使用规则等行为特征，并反馈给服务协作层，使不同应用可以根据这个信息调整和优化其网络结构，均衡负载。另外，为每个物理设备建立马尔科夫模型，估计其在时域上的负载和能力变化状态，将该信息通知服务协作层，促进该层的进一步优化。

5.2.1.4　服务和应用层

服务和应用层是面向用户的层次结构，直接面向用户。用户应用提出服务需求，服务和应用层对该服务需求任务调用“统一的服务虚拟、控制和支撑层”的开放接口并执行。

在服务虚拟化平台 SuperNova 中，统一的服务虚拟、控制和支撑层是系统

的研究重点，而本文的开发工作主要集中在该层的服务控制层子层，以及全局认知管理层。

5.2.2 编程架构

如图 5.3 所示，SuperNova 服务虚拟化平台的编程架构主要分为两个部分：全局管理服务器编程和局域管理服务器编程。全局管理服务器是面向用户的，在该层，用户通过命令行客户端或 Web Service 网页与系统进行交互，提出消息请求，然后经由 Mule ESB 消息框架进行相关的处理。如果用户的消息请求需交由局域管理服务器处理，则通过 Netty 异步通信机制与局域管理服务器通信，由局域管理服务器完成相应的请求。下面将详细介绍服务虚拟化平台 SuperNova 的编程架构中涉及的组件。

5.2.2.1 Boto

Boto[77] 是一个 Python[78] 包，它提供了访问 AWS 服务的接口，包括 Amazon S3、Amazon EC2、Amazon DynamoDB 等 AWS 服务。Boto 库提供了开放的 Python API，且所有的服务客户端都采用相同的接口，以便在所有支持的服务上都提供一致的用户访问体验。Boto 库通过封装代码细节，提供简洁的接口，大大简化了代码的编写，降低了编程复杂性，提高了代码复用率。

在 SuperNova 的工程开发工作中，我们使用 Boto 库来与 AWS EC2 建立连接，根据业务需求发送定制的 HTTP 请求并获取响应。

5.2.2.2 GWT

Ajax[79]，是由 XHTML、CSS、JavaScript、XMLHttpRequest、XML 等技术组合而成的 Web 应用开发领域的热门技术，用于创建更加动态和交互性更好的 Web 应用程序，提升用户的浏览体验。然而，Ajax 应用的开发和调试是非常繁复的，由于没有合适的开发工具辅助支持 Ajax 应用的开发和调试，使得 Ajax 应用的开发和调试过程很困难。与此相反，Java[80] 语言作为企业应用开发的主流语言，有各种各样开发工具的支持，如 Eclipse、NetBeans 等，其开发和调试过程变得非常简单。

GWT[81] 是 Google 推出的 Ajax 应用开发包，支持开发者使用 Java 语言开发 Ajax 应用，能够将 Java 语言开发的应用转化为 Ajax 应用。GWT 解决了

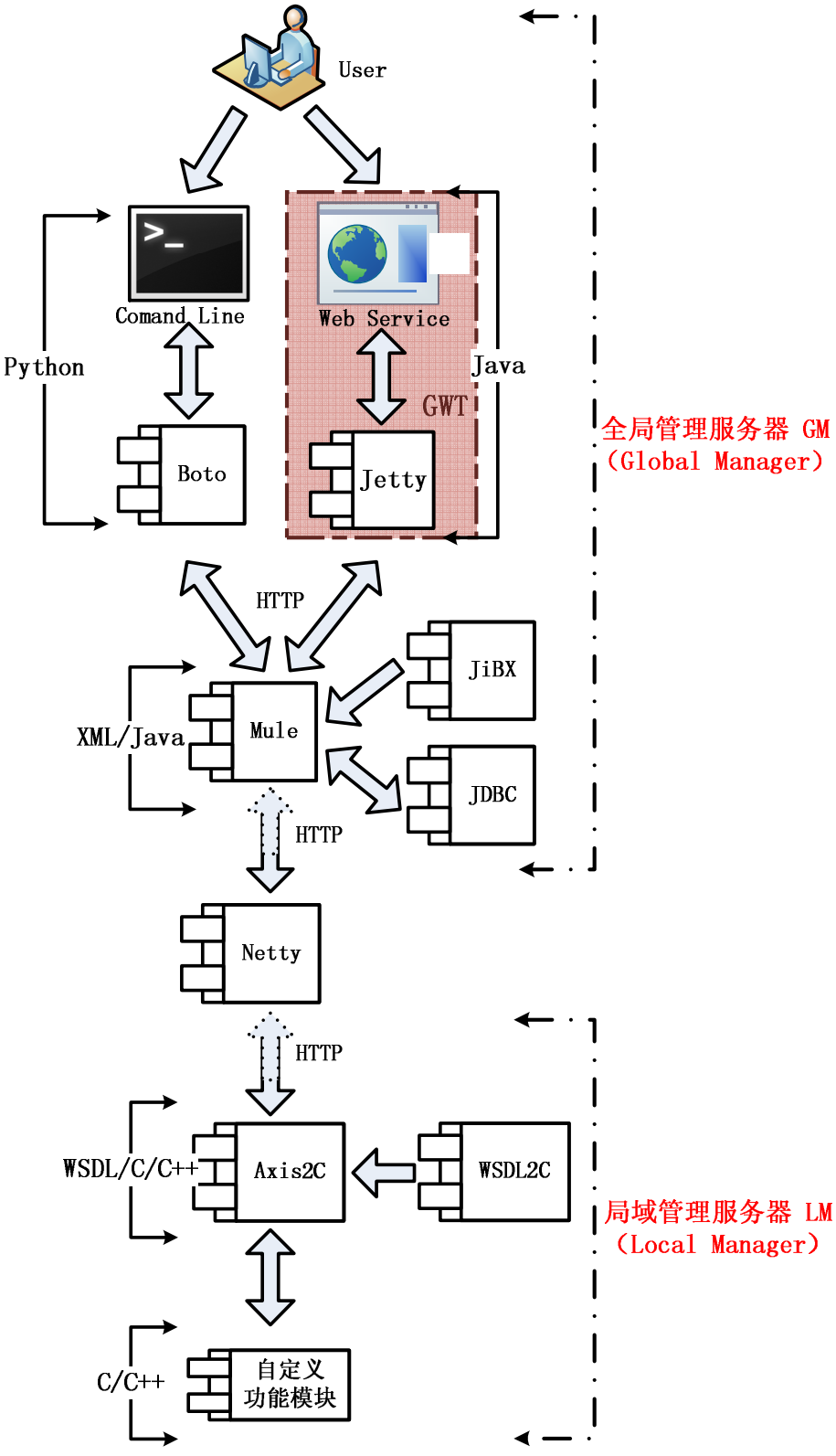

图 5.3: SuperNova服务虚拟化平台编程架构示意图

Ajax 应用开发中开发工具缺失的难题，开发者可以充分利用 Java 语言的开发优势，降低 Ajax 应用开发的难度，加快 Ajax 应用的开发速度。因此，可充分发挥 Ajax 技术的优势，创建更加动态和交互性更好的 Web 应用程序，提升用户的浏览体验。使用 GWT 开发的 Java 应用将由 GWT 开发包提供的编译工具编译后生成对应的、应用了 Ajax 技术的 Web 应用，Java 应用中和服务器之间的交互动作也被自动生成的异步调用代码所代替。

GWT 支持将 Java 语言开发的应用转化为 Ajax 应用，并提供了更多的高级特性：

1、GWT 编译器

GWT 编译器是 GWT 开发工具包的核心，负责将 Java 代码翻译成 Ajax 内容。GWT 编译器能够翻译 Java 语言的大部分特性，包括支持 Java 语言中的基本类型，支持 java.lang[82] 包和 java.util[83] 包中的大部分类和接口，支持正则表达式和序列化。

2、跨平台支持

GWT 提供了很多的组件元素，如 Button、VerticalPannel 等，GWT 编译器能够将这些组件翻译成浏览器内置的类型，使得由 GWT 编译生成的 Ajax 应用能够支持大部分的浏览器和操作系统，如 Internet Explorer、Firefox 等浏览器和 Linux、Windows 等操作系统。

3、宿主模式

宿主模式是指我们和 GWT 应用交互的状态。当我们开发和调试时，我们就处于宿主模式下。此时，Java 虚拟机使用 GWT 内置的浏览器运行 GWT 应用编译后的 class 内容，因此能够提供“编码、测试、调试”过程的最佳速度。

3、Web模式

Web 模式是指我们和成功转换的 Ajax 应用的交互状态。此时，GWT 应用已经成功编译转换为 Ajax 应用了，我们可以使用 Web 方式来访问 Ajax 应用了，因此，不再需要 GWT 工具包或者 JVM[84, 85] 虚拟机的支持了。

5.2.2.3 Jetty

Jetty[86] 是一个开源的 servlet 容器。Jetty 为基于 Java 的 Web 内容提供运行环境。Jetty 使用 Java 语言编写，以 JAR 包的形式对外发布开放的 API。开发人员可以将 Jetty 容器实例化成一个对象，可以迅速为一些独立运行的 Java

应用提供网络和 Web 连接。通常，Jetty 可通过 XML 文件或者 API 接口进行配置。另外，Jetty 可作为嵌入式服务器。

5.2.2.4　Mule

Mule 是由 Mulesoft[87] 推出的开源轻量级的 ESB 企业服务总线消息框架。Mule 的核心概念是 SEDA，即分阶段和事件驱动。分阶段指的是将一个复杂的处理过程分解成几个不同的阶段，可以由不同的线程分别进行处理；事件驱动意味着所有的处理过程都是基于事件驱动和异步通信模式。

5.2.2.5　JiBX

JiBX[88] 是对 XML 数据和 Java 对象进行绑定的工具框架。JiBX 通过绑定定义文档来定义 XML 数据与 Java 对象间的绑定和转换规则，绑定定义文档是联系 XML 数据与 Java 对象之间的桥梁。在 JiBX 中，通过 marshal 绑定，可由 Java 对象生成 XML 数据文档；通过 unmarshal 绑定，可根据 XML 数据文档建立 Java 对象。

5.2.2.6　JDBC

JDBC[89] 即 Java 数据库连接，是标准的 Java API 接口集合，用于访问数据库。JDBC 对数据库的操作与具体的数据库类型无关，它是一种规范，提供了一套完整而统一的编程接口，使开发人员可以使用纯 Java 的方式来连接并操作任意类型的数据库。如图 5.4 所示为 JDBC 的体系架构示意图。从图中可以看出，JDBC 由两层组成：

1、JDBC API

JDBC API 使用一个驱动程序管理器和数据库特定的驱动程序，从而提供透明的异构数据库的连接。因此，应用程序可直接使用 JDBC API 完成数据库相关的操作，而无需了解特定的数据库细节。

2、JDBC Driver API

JDBC Driver API 用于支持 JDBC 管理器，确保使用正确的驱动程序来访问每个数据源的 JDBC 驱动程序管理器。驱动程序管理器能够支持多个并发连接到多个异构数据库的驱动程序。

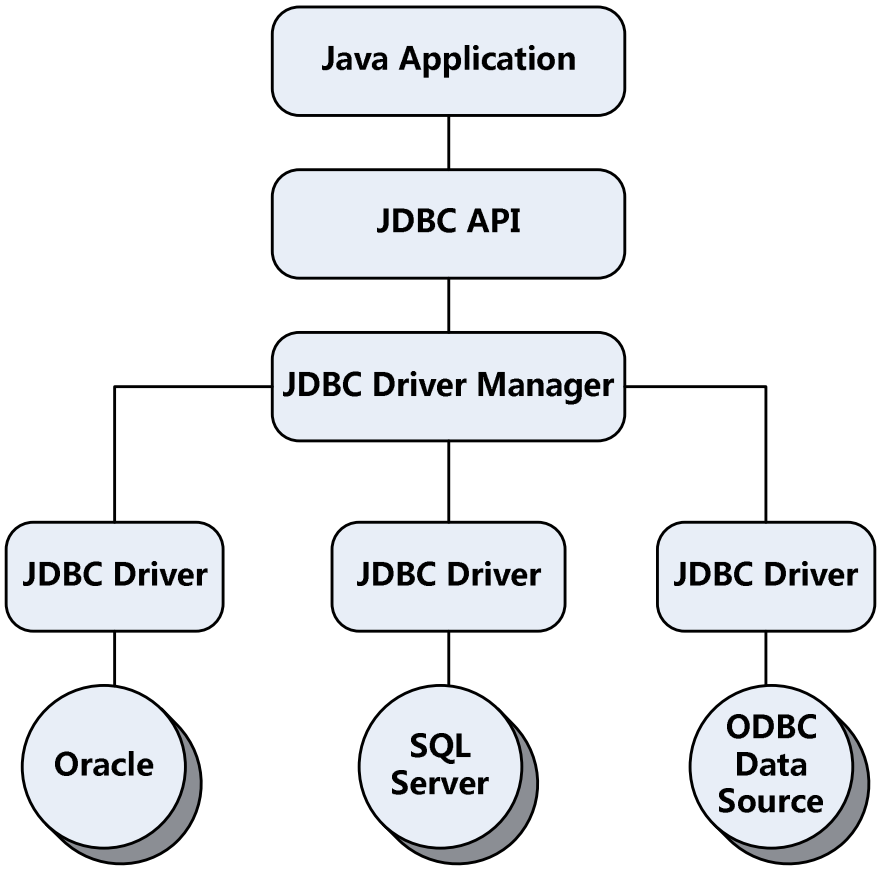

图 5.4: JDBC 的体系架构示意图

5.2.2.7 Netty

Netty[90] 是由 JBOSS[91] 提供的一个 Java 开源框架。Netty 提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序。因此，Netty 是基于 NIO 的客户、服务器端编程框架，使用 Netty 可以确保快速和简单的开发出一个网络应用，例如实现了某种协议的客户、服务端应用。Netty 简化和流线化了网络应用的编程开发过程。

5.2.2.8 Axis2C

Axis2C[92] 是基于 Axis2 架构，用 C 语言实现的 Web Service 引擎，具有良好的可移植性，可作为其他应用的子模块来提供 Web 服务。Axis2C 支持 SOAP 协议，也支持 RESTful 风格的 Web 服务。用户可以基于 Axis2C 开发

Web 服务提供给其他客户端调用，也可通过调用 Axis2C 提供的 C 语言库，开发客户端程序，访问其他的 Web 服务。

5.2.2.9　WSDL2C

Axis2C 提供了 WSDL2C 工具，可根据 WSDL 语言编写的用于描述整个 Web Service 的 XML 数据文档，生成 C 语言描述的 Axis2C 的服务器 Stub。因此，WSDL2C 相当于翻译机，将 XML 语言翻译为 C 语言。WSDL 是 Web 服务描述语言，可精确地描述 Web 服务。WSDL 的文档结构图如图 5.5 所示，通常，WSDL 文档的包含以下几个重要的元素：

1、Types：数据类型定义的容器，通常使用某种类型系统，如 XML Schema 中的类型系统。

2、Message：通信消息的数据结构的抽象类型化定义。使用 Types 所定义的类型来定义整个消息的数据结构。

3、Operation：对服务中所支持的操作的抽象描述，可准确描述一个访问入口的请求和响应消息对。

4、Binding：特定端口类型的具体协议和数据格式规范的绑定。

5、Port：服务访问点，是一个协议/数据格式绑定与具体 Web 访问地址的组合。

6、PortType：服务访问点类型。描述了某个访问入口点类型所支持的操作的抽象集合。

7、Service：由相关的服务访问点集合所组成的 Web 服务。

5.3　业务流量隔离和控制模块

5.3.1　流量控制简介

如图 4.3 和图 5.6 所示，SuperNova 中的业务流量隔离和控制模块主要应用在局域管理服务器上，并将流经局域管理服务器的流量划分为 outer 数据流量和 inner 数据流量两大部分。通常，inner 数据流量是指用户部署的业务内部之间的相互通信产生的数据流量，而 outer 流量则是外部终端用户通过 Internet 对该业务的访问流量。根据 outer 外部流量的特征，我们又进一步预设了 5 个不同优先级的类别，用户可根据业务需求选择其中的若干个进行配置以提供自

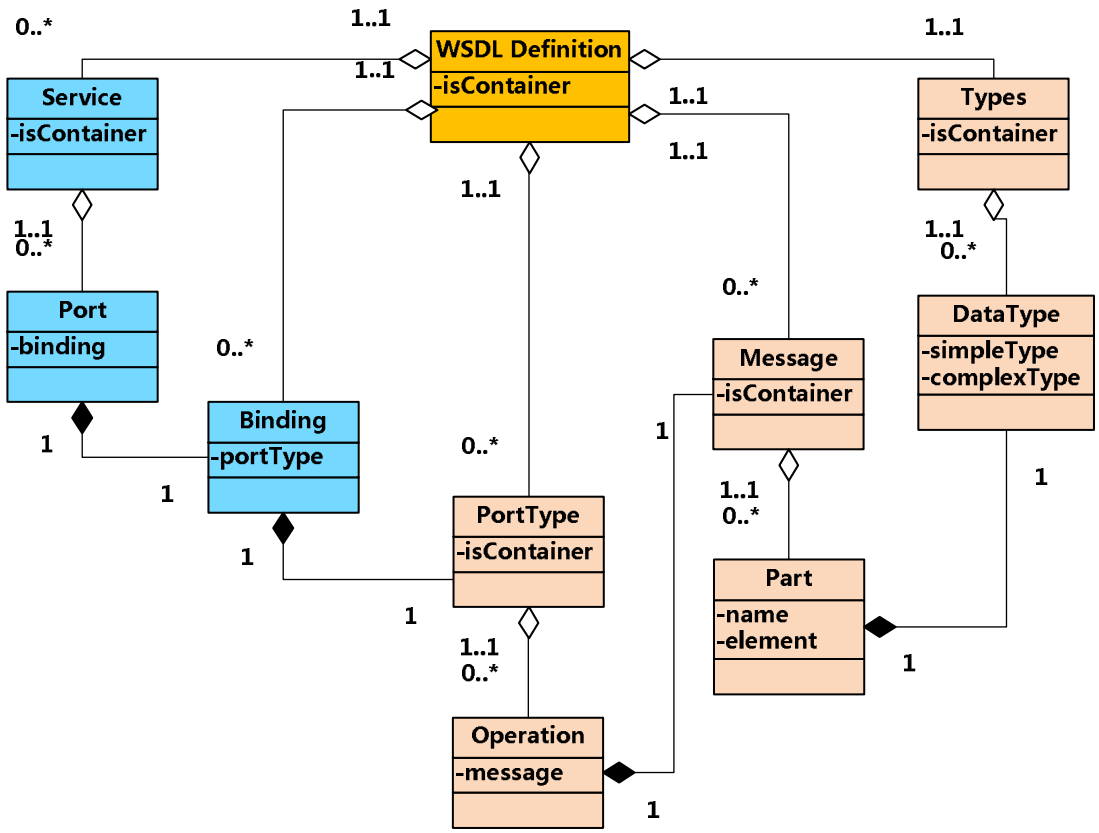

图 5.5: WSDL 文档结构图

定义的流量控制方案。在系统层面，业务流量隔离和控制模块通过隔离不同用户之间的数据流量，屏蔽了不同用户的数据流量干扰。在用户层面，业务流量隔离和控制模块根据用户需求区分对待用户的不同业务流量，避免业务之间的数据流量干扰。

5.3.2 编程架构

如图 5.7 所示，SuperNova 中业务流量隔离和控制模块的编程架构示意图。图中自上而下展示了整个开发过程，并明确标出了需要添加或修改的主要文件。首先，用户提出请求“我想启动业务流量隔离和控制模块”。请求的方式包括命令行工具和 Web Service 两种。在命令行工具下，需要编写 Python 代码 setuptrafficcontrol.py ，调用 Boto 库与 AWS EC2 建立连接，发送请求消息并等待响应。在 Web 方式下，编写 GWT Java 代码，包括用户界面 Eu-

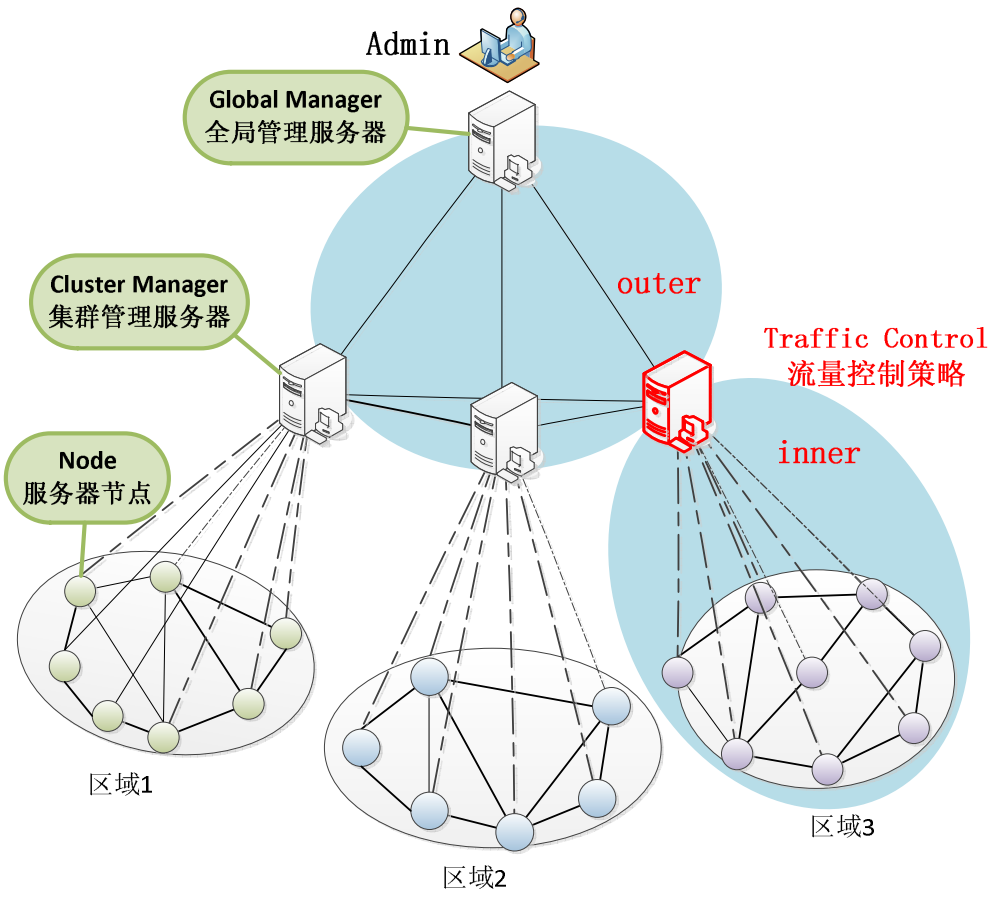

图 5.6: SuperNova中的流量控制模块

calyptusWebInterface.java 文件，异步调用代码 EucalyptusWebBackend.java，EucalyptusWebBackendAsync.java，EucalyptusWebBackendImpl.java，以及功能实现 TrafficController.java 等文件。

无论是采用命令行方式还是 Web Service 方式，访问请求消息都会进入到 Mule 组件中。Mule 组件根据消息类型，将消息导向不同的 Mule Component 组件进行处理。对于业务流量隔离和控制模块，主要的 Mule Component 是 TrafficControlEndpoint.java 所实现的 TrafficControlEndpoint 组件，根据 msgs-binding.xml 中的配置定义，该组件接受所有关于流量隔离控制的消息，包括 SetupTrafficControlType，StopTrafficControlType，ConfigTrafficControlType 等消息。在 Mule 组件 TrafficControlEndpoint 中，还存在与内存数据库表的交互。内存数据库表的数据是在系统空闲时，全局管理服务器从所有局域管理服

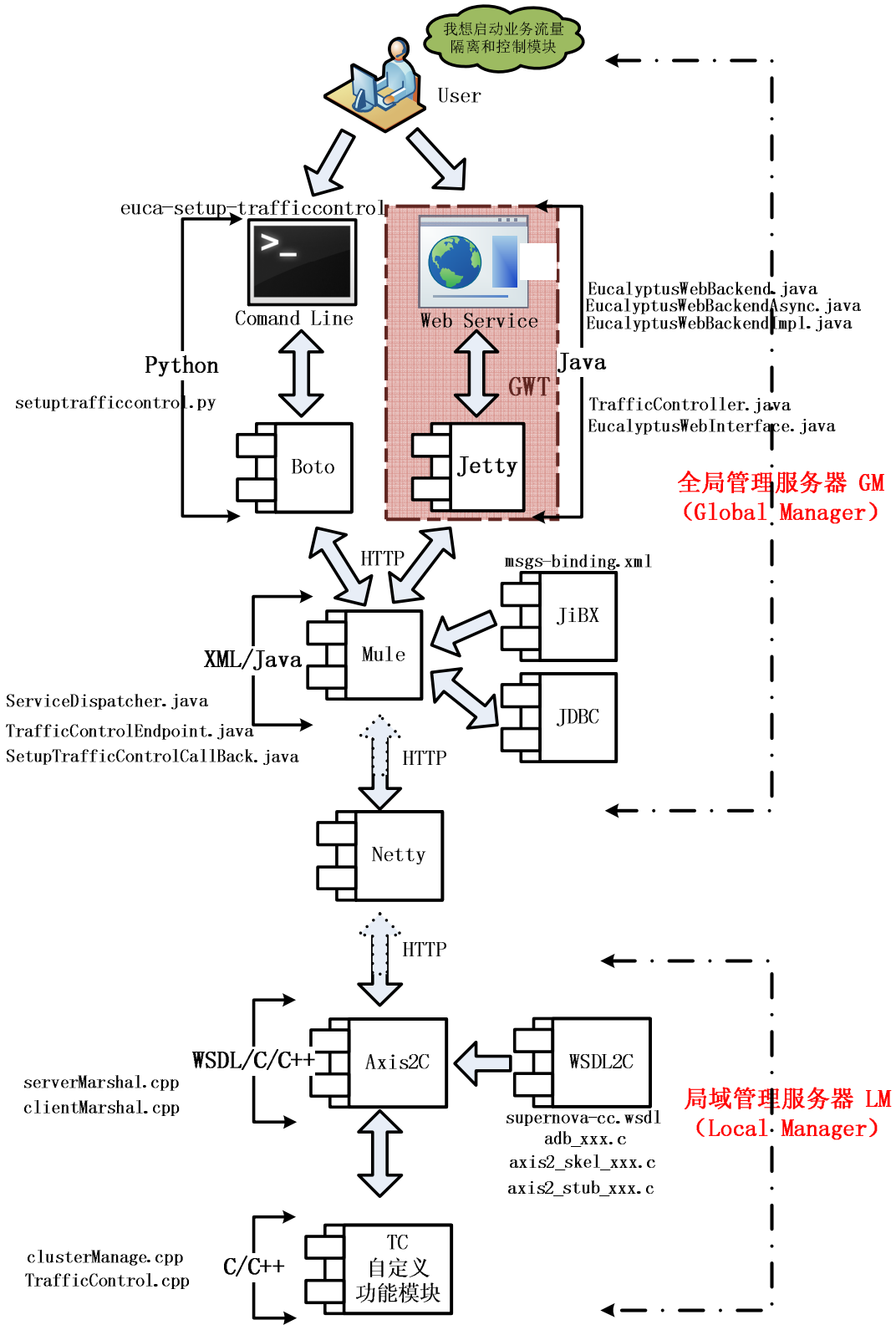

图 5.7: SuperNova中流量控制模块编程架构示意图

务器收集的缓存数据，具有快速访问获取的特点。

如果用户的请求需要局域管理服务器进行处理，则通过 Netty 框架实现全局管理服务器和局域管理服务器的异步通信，全局管理服务器将请求消息传给局域管理服务器，局域管理服务器在处理结束后把结果异步地返回给全局管理器。在局域管理服务器中，通过 WSDL Web 服务描述文件定义数据类型以及操作类型，如 supernova-cc.wsdl，再通过 Axis2C 的工具 WSDL2C 将其转换为 C 语言定义的代码，进而调用具体的功能实现函数，如文件 TrafficControl.cpp 中定义的函数。

5.3.3　功能和实现

5.3.3.1　启动

启动业务流量隔离和控制模块的方式有两种：

1）命令行方式

启动终端工具，在终端输入命令：euca-setup-trafficcontrol，即可启动业务流量隔离和控制模块。

2）Web 方式

如图 5.8 所示，点击并勾选 Traffic Control 复选框，通过“保存簇配置”使配置生效，即可启动业务流量隔离和控制模块。

不管是采用命令行方式，还是采用 Web 方式，都可以很容易地启动业务流量隔离和控制模块。事实上，如果系统从来没有启动过业务流量隔离和控制模块，则该模块会从系统配置文件中读取默认的业务流量隔离和控制参数，以实现如图 4.3 所示的流量控制架构。相反，如果系统曾经启动过业务流量隔离和控制模块，并重新进行过相应的配置和规则修改，这些新的规则都会被保存下来。当用户再次启动业务流量隔离和控制模块时，会自动恢复到用户最后一次修改后的最新规则。这种设计方式更加符合用户的行为习惯，能够减轻用户繁复的配置操作负担，提高用户体验质量。

5.3.3.2　查询

查询包括查询系统是否启动了业务流量隔离和控制模块，还包括查询系统已启动的业务流量控制规则。可通过命令 euca-describe-trafficcontrol 进行，如

图 5.8: Web 方式启动业务流量隔离和控制模块

图 5.9 所示。实际上，查询操作是直接在全局管理服务器上开展的。因此，在 SuperNova 流量控制模块编程架构 5.7 中，查询消息在经过 Mule 组件后，便终止了消息传递。这是因为在 SuperNova 系统的全局管理服务器中，存在内存数据库表，其中缓存了相关的流量控制规则。因此，流量控制规则的查询无需向局域管理服务器发出请求，可直接从内存数据库表中查询数据。这种设计是典型的以空间换取时间，牺牲一定的存储空间以达到快速响应，从而提高用户体验质量。另外，查询消息不需要再传递到各个局域管理服务器，从而屏蔽了网络的不确定性带来的不稳定性能损失。

5.3.3.3 配置

配置流量控制规则可通过命令：euca-config-trafficcontrol 进行，该命令功能强大，有很多参数可供选择以对已启动的流量隔离和控制模块进行微调和重

```
[root@supernova-2 euca2-admin-x509]# euca-describe-trafficcontrol
Traffic Control Info
Device   RootQdisc Level    Bandwidth Prio     Qdisc    QdiscParams              Filter
eth0     cbq      vlanroot 100mbit  1                  None
eth0     cbq      inner    50mbit   1         sfq      perturb=10;quantum=1514   match ip dst 10.1.0.0/16
eth0     cbq      outer    50mbit   1                  None
eth0     cbq      prio1    10mbit   1         tbf      latency=50ms;burst=1540   match ip tos 0x10 0xff
eth0     cbq      prio2    10mbit   2         tbf      latency=50ms;burst=1540   match ip tos 0x04 0xff
eth0     cbq      prio3    10mbit   3         tbf      latency=60ms;burst=1514   match ip tos 0x02 0xff
eth0     cbq      prio4    10mbit   4         sfq      perturb=8;quantum=1514    match ip tos 0x08 0xff
eth0     cbq      prio5    10mbit   5         sfq      perturb=10;quantum=1514   match u8 0x60 0xf0 at 0;
[root@supernova-2 euca2-admin-x509]#
```

图 5.9: 查询流量控制规则

新配置，主要的参数如表 4.4 所示。在配置流量规则时，一方面，要修改全局管理服务器的内存数据库表信息以记录最新的配置信息并保持系统的一致性；另一方面，要将配置消息传递到局域管理服务器以执行真正的修改和重配置，从而使新的配置生效。

5.3.3.4　取消

取消业务流量隔离和控制模块可以通过以下两种方式：

1）命令行

直接在终端输入命令：euca-stop-trafficcontrol，即可取消并停止业务流量隔离和控制模块。

2）Web 方式

如图 5.10 所示，在 Web 页面上点击并取消 Traffic Control 复选框的勾选，通过“保存簇配置”使配置生效，即可取消并停止业务流量隔离和控制模块。

5.4　虚拟网络嵌入模块

SuperNova 服务虚拟化平台上的虚拟网络嵌入模块是基于本文第 三 章所述的 MSO-VNE 算法，进行工程简化并编程开发实施的。下面将详细描述在 SuperNova 上的虚拟网络嵌入模块的工程实现。

5.4.1　工程设计

5.4.1.1　算法简化

在第 三 章所述的 MSO-VNE 算法中，我们做了如下假设：

1）链路的特征是可测量的

图 5.10: Web 方式取消业务流量隔离和控制模块

2）Internet 链路带宽是无限的

然而，这两条假设在实际生活中都是不成立的。首先，网络是非常复杂的，它受各种各样的因素影响，包括线路材质、温度等气候条件。通常，这些因素中，任何一个因素的微小变化都会使网络链路特性发生本质的变化。因此，网络链路充满了不确定性，链路特征是很难测量的，尤其是延迟、延迟抖动等参数。正是由于网络的复杂性和不确定性，任何涉及到网络链路的特征的假设都是不符合实际的。对于 Internet 互联网而言，其庞大的规模更是加重了这种不确定性和复杂性。所以，上述两条假设在现实生活中都是不成立的。

工程开发实施与理论算法研究不同，工程开发实施必须在准确性和可实现性之间折衷考虑，可以视实际情况牺牲一方而加强另一方。在 SuperNova 服务虚拟化平台中，对虚拟网络嵌入模块，我们可牺牲准确性以获得较高的可实现性。在进行实际的虚拟网络嵌入匹配计算时，我们只考虑节点的属性特征匹

配，而忽略链路的属性特征匹配。一方面，这种简化使得工程开发较容易实现；另一方面，这种简化使得系统能快速响应用户的服务部署请求。

5.4.1.2　编程简化

如图 5.11 所示，我们对 SuperNova 中的虚拟网络嵌入模块进行了编程简化，主要通过构建开发独立于 SuperNova 服务虚拟化平台的虚拟网络嵌入模块实现。该模块接受用户的请求，通过开放的编程接口 API 从 SuperNova 服务虚拟化平台获取可用的资源信息，在模块中进行虚拟网络嵌入匹配计算，如果成功映射，则使用开放的接口 API 将用户的请求部署到 SuperNova 服务虚拟化平台中。

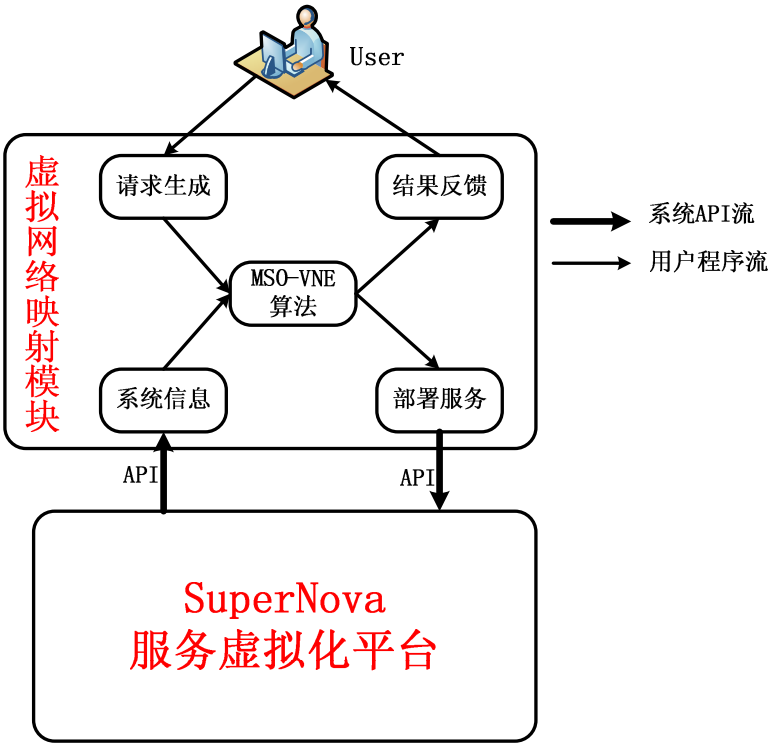

图 5.11: SuperNova中虚拟网络嵌入模块结构

在 SuperNova 服务虚拟化平台中，虚拟网络嵌入模块的编程实现不同于流量控制模块的编程实现，对比图 5.11 和图 5.7 ，我们可以看出，虚拟网络嵌入模块的编程架构比流量控制模块的编程架构要简明得多，编程实现和开发流程也要简洁明了很多，这种简洁明了的编程开发使得其可扩展性很强。如果我们

提出了新的虚拟网络嵌入算法，只需要修改图 5.11 中的虚拟网络嵌入模块，而无需改动SuperNova平台中的一行代码。

与流量控制模块不同，SuperNova 中的虚拟网络嵌入模块是独立于 SuperNova 系统的，它通过 SuperNova 服务虚拟化平台开放的 API 接口从系统获取必要的信息、与系统进行必要的交互并反馈处理结果。这里涉及到的开放 API 接口主要包括：

1）euca-describe-availability-zones *verbose*

euca2ools 命令 euca-describe-availability-zones *verbose* 用于获取 SuperNova 系统的资源信息，包括 cpu 的最大值（max），cpu 的可用值（free），ram 的最大值（max），ram 的可用值（free），以及 disk，cluster zone 等信息，如图 5.12 所示。

```
[root@supernova-2 euca2-admin-x509]# euca-describe-availability-zones verbose
AVAILABILITYZONE        52cluster       192.168.155.52
AVAILABILITYZONE        |- vm types     free / max   cpu   ram  disk
AVAILABILITYZONE        |- m1.small     0016 / 0016   1    128     2
AVAILABILITYZONE        |- c1.medium    0016 / 0016   1    256     5
AVAILABILITYZONE        |- m1.large     0008 / 0008   2    512    10
AVAILABILITYZONE        |- m1.xlarge    0007 / 0007   2   1024    20
AVAILABILITYZONE        |- c1.xlarge    0003 / 0003   4   2048    20
[root@supernova-2 euca2-admin-x509]#
```

图 5.12: 由API查询的系统信息

SuperNova 的虚拟网络嵌入模块将该命令获取的系统资源信息和用户的请求进行匹配计算。若系统有丰富的资源，足以支撑部署用户的请求和业务，便通过以下开放的 API 接口在 SuperNova 系统中部署业务。

2）euca-run-instances *param*

euca2ools 命令 euca-run-instances *param* 用于部署业务。用户提出虚拟网络请求后，如果 SuperNova 系统有丰富的资源，足以支持用户的请求在系统中部署，则虚拟网络请求会成功映射，即虚拟网络请求中每个虚拟节点和虚拟链路都能匹配映射到实际的物理节点和物理链路上。成功映射后，可使用 euca-run-instance 接口进行真正的业务部署，以分配实际的物理资源。

5.4.2 开发实现

如图 5.13 所示，SuperNova 服务虚拟化平台中虚拟网络嵌入映射的关键处理步骤包括：

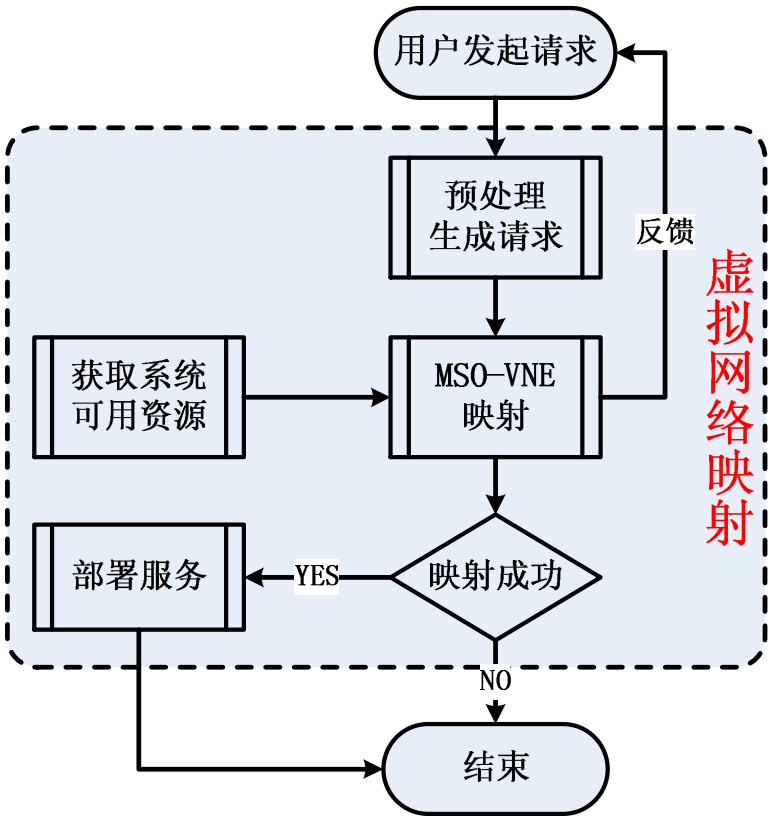

图 5.13: SuperNova中虚拟网络映射模块的实现流程

1）当用户发起请求时，虚拟网络映射开始；

2）虚拟网络映射模块首先对用户的请求进行预处理，生成系统可方便处理的格式；

3）虚拟网络映射模块从 SuperNova 系统获取系统可用的资源，包括 CPU，内存，磁盘等信息；

4）将系统的可用信息和用户的请求进行虚拟网络嵌入映射，并将映射结果反馈给用户；

5）若系统有足够的资源支撑用户的请求，映射成功后，使用系统开放的部署API将用户的请求部署到系统中；

6）第4）步映射失败以及第5）步部署完成都会结束虚拟网络映射。

5.4.3 功能接口

SuperNova 服务虚拟化平台中，虚拟网络映射的部署接口 API 为：

euca-vne-deploy *params*

该 euca2ools 命令通过调用 Boto 库接口与 AWS EC2 建立连接，进行用户认证，根据用户的请求，结合系统可用的资源进行虚拟网络嵌入匹配计算和映射处理，成功映射的虚拟网络请求会被分配实际的物理资源，从而为终端用户提供服务。该命令的具体参数如表 5.1 所示。

参数	缩写	描述
node-num	n	虚拟节点的数量
life-time	l	虚拟网络请求的生存时间，分钟
cpu	c	CPU 资源的需求量
ram	r	RAM 资源的需求量，MB
disk	d	DISK资源的需求量，MB
prefer-cluster	p	用户希望部署的集群区域
help	h	显示帮助信息
example	euca-vne-deploy -n 1 -l 2 -c 1 -r 128 -d 5 -p 1	

表 5.1: euca-deploy-vne 的参数说明

5.5 云控制器模块

5.5.1 云控制器简介

如图 5.14 所示，云控制器隶属于 SuperNova 服务虚拟化平台的全局认知管理平台，是用户对其部署在 SuperNova 平台上的业务服务的管理入口。通过云控制器，用户可查询已部署的业务，也可以阐述业务需求，并请求系统提供服务，还可以管理已部署的业务，包括虚拟实例、存储组件、安全组等的管理。

5.5.2 功能完善

对 SuperNova 服务虚拟化平台的功能完善工作主要是在已有的 euca2ools 命令行客户端工具基础上，对已有的管理操作进行重编码，省去对 euca2ools

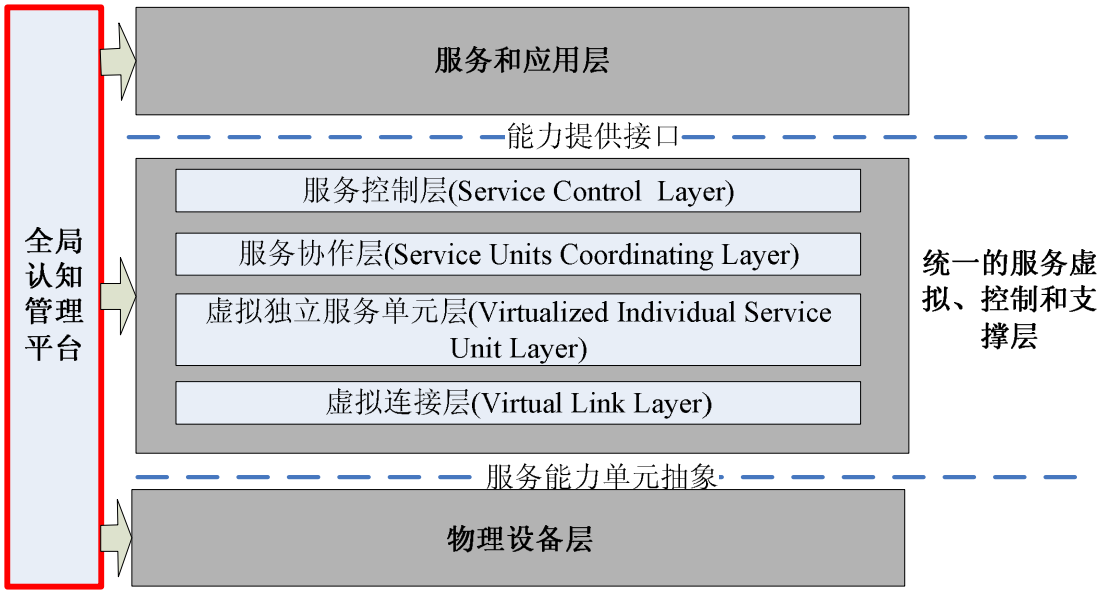

图 5.14: 云控制器在 SuperNova 中的位置

客户端命令的解析，直接采用 Web Service 工作方式。这样做的好处有以下两点：

1、简化代码处理流程。

在 SuperNova 服务虚拟化平台中，euca2ools 客户端命令的对应的代码代码处理流程如下：首先，用户在终端输入 euca2ools 客户端命令；然后，euca2ools对该命令进行解析，并转换为以 Web Service 方式呈现的请求消息；最后，将请求消息传递给 Mule 组件进行进一步的处理。我们对这个处理流程的改进是删除其中第 2 个步骤的解析和转换工作，简化代码处理流程。我们通过 GWT 框架，直接构建 Web 页面，用户通过在该 Web 页面上简单地点击便以 Web Service 的方式生成请求消息，从而简化了代码处理流程。

2、提高用户体验

SuperNova 服务虚拟化平台最终要面向用户提供服务，因此，平台的用户体验质量是至关重要的。我们通过对云控制器进行可视化改进，使得用户不再需要面对非常不人性化的终端屏幕，也不再需要输入繁复易错的命令。而是通过在 Web 页面上进行直观的点选操作就可以完成同样的功能，是对用户操作的一种人性化设计，可很好地提高用户体验。

5.5.2.1 实例管理

在 SuperNova 服务虚拟化平台中，虚拟实例的管理包括查询已经创建的虚拟实例，对已存在的虚拟实例进行重启、终止操作，以及重新创建并启动新的实例。如图 5.15 所示，在 Web 页面的 Instance 选项卡下的 Available Instances 显示当前用户所创建的实例，用户可对他们进行相应的操作，包括 Terminate 和 Reboot 。

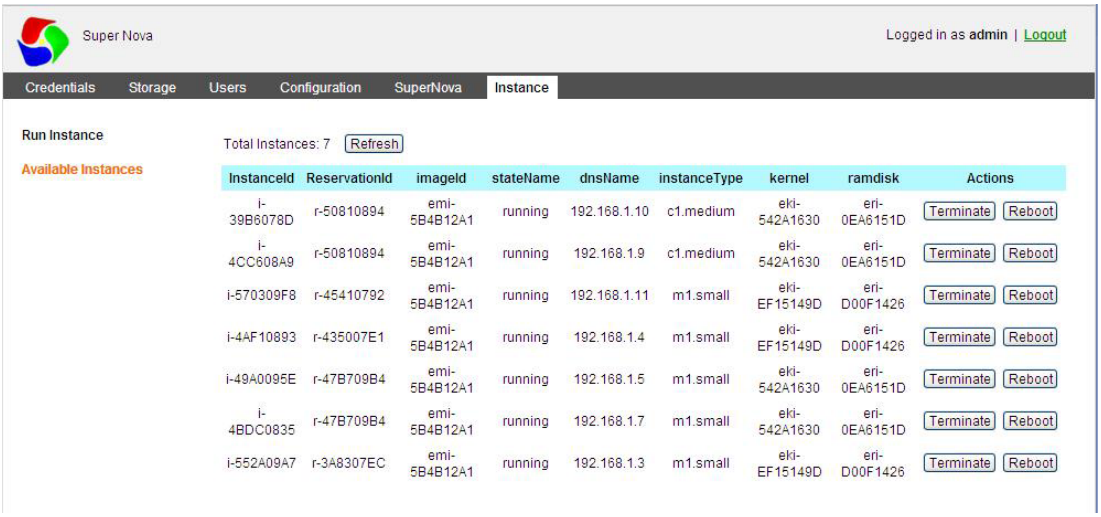

图 5.15: Available Instance

如图 5.16 所示，Instance 选项卡下的 Run Instance 允许用户指定相应的参数以创建用户希望的虚拟实例。

图 5.16: Run Instance

其中的参数含义如表 5.2 所示。

参数	描述
count	需要创建的虚拟实例的数量，默认是1
security group	虚拟实例所属的安全组，默认是default
key	公钥名称
address	虚拟实例的地址（可选）
instanceType	虚拟实例的类型
zone	虚拟实例所属的局域管理服务器
kernel	kernel内核镜像
ramdisk	ramdisk镜像
imagId	镜像Id

表 5.2: 创建虚拟实例的参数说明

5.5.2.2　存储管理

在 SuperNova 服务虚拟化平台中，存储组件包括镜像（Images），存储卷（Volume）和存储卷快照（Snapshot）。因此，存储管理即是对相关组件的管理。对于镜像 Images 的管理，其 Web 界面如图 5.17 所示。Storage 下的 Images 选项卡展示了可用的 Images 以及相关的操作，包括 Disable 或 Enable。

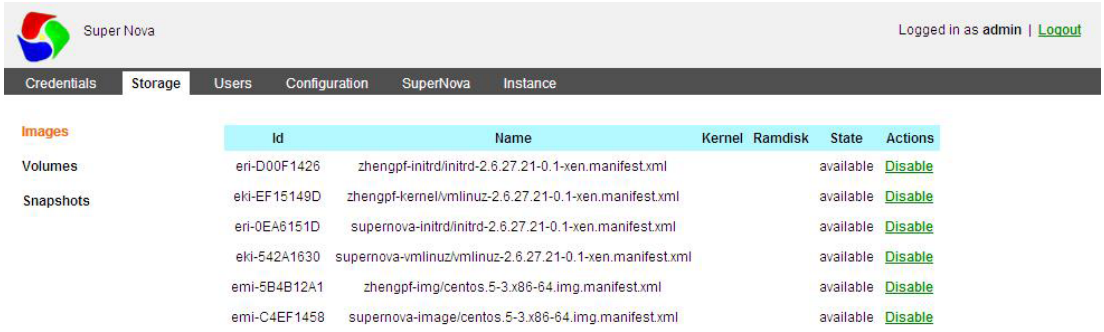

图 5.17: SuperNova 服务虚拟化平台的 Images

如图 5.18 所示，Storage 下的 Volumes 选项展示了可用的 Volume 和相应的挂载和卸载情况以及对应的操作，包括删除、卸载、挂载。同时，用户可以通过指定 Volume 的空间大小，快照 Id 和集群管理器以创建新的 Volume；也可以把存在的存储卷和虚拟实例关联起来，即存储卷的挂载。

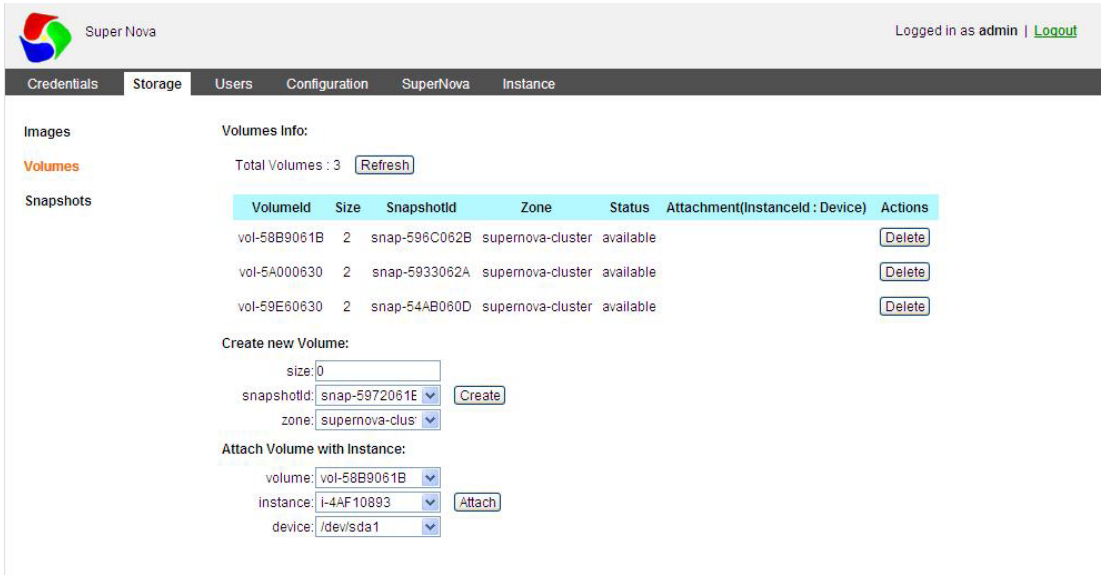

图 5.18: SuperNova 服务虚拟化平台的 Volumes

如图 5.19 所示，Storage 下的 Snapshots 是对存储卷 Volume 进行相应的备份快照。与 Snapshot 相关的操作包括创建指定 Volume 的 snapshot 以及删除指定的 snapshot 。

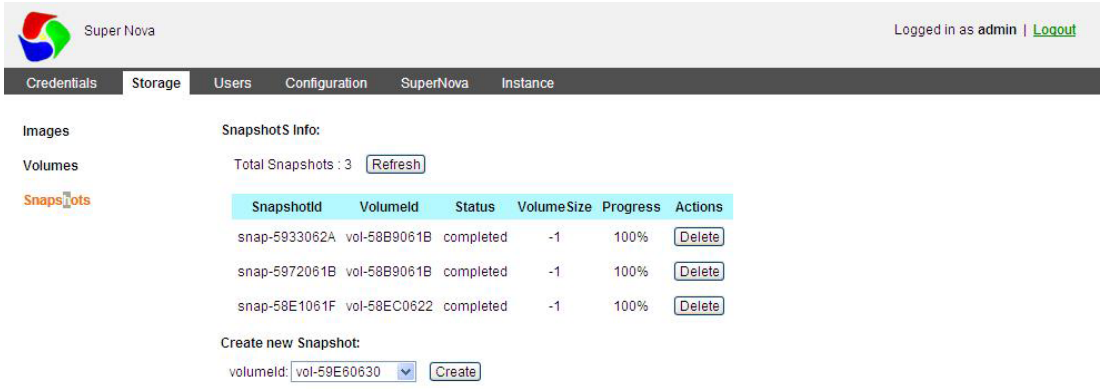

图 5.19: SuperNova 服务虚拟化平台的 Snapshots

5.5.2.3　安全组管理

Security Groups，网络安全组，相对于防火墙，它对流入实例的流量进行控制和管理。一个安全组可以有多个实例与之关联，同样，一个实例可以关联到多个安全组。通过安全组的规则可以控制到达实例的入境流量，只有符合实例关联的安全组的规则的入境流量才能进入到实例中，其他的任何入境流量都将被丢弃。安全组的规则可以在任何时刻对其进行修改，修改之后，新的安全组规则将自动地应用到与安全组相关联的所有实例中。

操作类型	euca2ools命令	描述
创建安全组	ec2-create-group	用当前账户创建新的安全组
描述安全组	ec2-describe-group	描述与账户相关的安全组信息
添加安全组规则	ec2-authorize	向指定安全组添加一个或多个规则
移除安全组规则	ec2-revoke	从指定安全组中移除一个或多个规则
删除安全组	ec2-delete-group	删除和账户关联的安全组

表 5.3: 安全组相关的操作和 euca2ools 命令

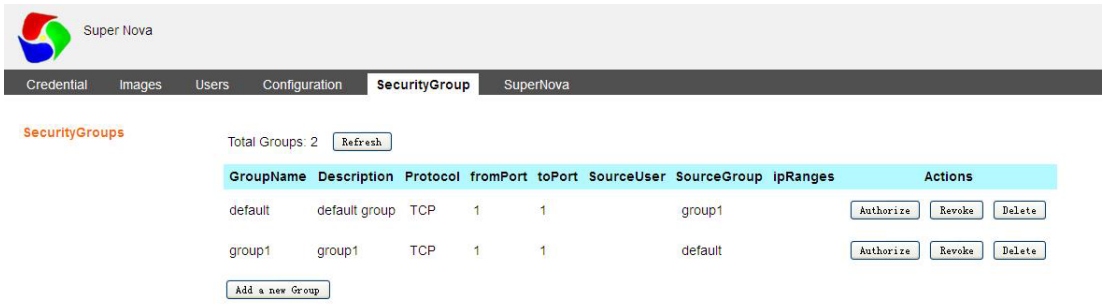

图 5.20: SuperNova 中的安全组

在 SuperNova 中，有默认的安全组 default。在启动实例时，可以显式地为实例指定安全组，否则该实例将关联到默认的安全组。默认安全组的初始设置如下：

1）不允许入境流量；

2）允许所有的出境流量；

3）允许关联到安全组的实例之间相互通信。

用户可以使用如表 5.3 所示的 euca2ools 命令来创建新的安全组，或修改安全组的规则，或删除安全组等。

在 SuperNova 服务虚拟化平台中，安全组相关的 Web 界面如图 5.20 所示。在 SecurityGroups 选项卡中，我们可以使用 Add a new Group 创建新的安全组，对已经存在的安全组，可以使用点击 Authorize 添加安全组规则，或通过 Revoke 移除安全组规则，或通过 Delete 完全删除安全组。

5.6 小结

本章详细阐述了在 SuperNova 服务虚拟化平台上开展的工程开发工作，主要包括三大部分：业务流量隔离和控制模块的工程开发实现，虚拟网络嵌入模块的工程开发实现，云控制器模块的功能完善。

本章首先详细介绍 SuperNova 服务虚拟化平台，包括系统架构和编程架构。然后，本章对 SuperNova 服务虚拟化平台的业务流量隔离和控制模块的工程开发实现进行详尽的阐述，包括该模块在 SuperNova 中的位置、实现方式、编程实现架构、功能接口以及使用方式。在 SuperNova 服务虚拟化平台中实现业务流量隔离和控制模块，可有效隔离屏蔽不同的业务流量，并根据业务数据的 QoS 特征，区分不同的数据流量，并提供有 QoS 保障的服务。另外，对于具体的业务数据，用户可根据其数据流量特征，进行自定义配置，以提高服务质量，达到更好的服务效果。

对于 SuperNova 服务虚拟化平台的虚拟网络嵌入模块，本章主要从工程设计、开发实现和功能接口这三个方面进行阐述。由于虚拟网络嵌入问题是 NP 难问题，时间复杂度非常高，而工程实现需要在准确性和可实现性上寻求折衷平衡，以提高工程效率和系统性能，进一步提高用户体验质量。因此，我们对 SuperNova 服务虚拟化平台的虚拟网络嵌入模块进行相应的工程简化设计，在开发实现时，充分考虑了时间和空间性能以及系统扩展性。

另外，本章为提高 SuperNova 服务虚拟化平台的用户体验质量，对全局认知管理层的云控制器进行相应的功能完善和补充。主要包括对部分 euca2ools 命令的可视化改进，如实例管理、存储管理、安全组管理等。对系统而言，可视化改进大大简化了代码处理流程，提高了系统的效率和性能。对用户而言，可视化的操作方式让用户从繁复的终端命令行中解放出来，显著提高了用户体验质量。

第六章　总结和展望

本章对本文的工作进行总结，并对今后的工作进行展望。本文的内容安排如下：第 6.1 节对本文的主要工作进行总结；第 6.2 节对今后的研究方向进行展望。

6.1　工作总结

在 3G 网络进一步普及、智能手机和无线网络持续发展的背景下，视频、音乐等媒体应用的普及率逐步攀升。与此同时，虚拟化技术的快速发展以及云计算技术的普遍应用，使得面向媒体业务的虚拟化服务研究具有较强的研究价值和广阔的市场前景。本文面向媒体业务的虚拟化网络服务设计与开发研究了面向媒体业务的虚拟网络服务的关键问题和技术，包括虚拟网络嵌入、资源交互技术、业务流量隔离与控制等。通过研究这些问题和技术，提出了相应的技术解决方案，解决了现有技术中的一些问题，取得了以下研究成果：

1、面向媒体业务的虚拟网络嵌入算法 MSO-VNE

本文结合服务虚拟化平台的应用场景和平台特性，设计并提出了一种面向媒体业务的虚拟网络嵌入算法 MSO-VNE 。MSO-VNE 算法根据虚拟网络请求和底层物理网络的地理位置特征，运用节点聚类算法将虚拟网络请求聚类分割为规模较小的子请求，充分利用分而治之策略来降低虚拟网络嵌入的时间复杂度。针对所有的虚拟网络子请求，运用并发处理策略进一步降低虚拟网络嵌入处理的时间复杂度。MSO-VNE 算法通过动态服务均衡分析，充分考虑了节点在网络拓扑中的重要性。节点的重要性不仅与节点自身的可用资源有关，还与该节点的邻居节点的可用资源相关。通常，越是重要的节点具有优先的嵌入映射权利。动态服务均衡分析显著提高了虚拟网络嵌入的性能和底层物理网络的资源利用率，增加了基础设施提供商的长期收益。

2、面向媒体业务的资源映射交互框架

本文基于面向媒体业务的虚拟网络嵌入算法 MSO-VNE ，提出了一种面向媒体业务的资源映射交互框架。资源映射交互框架主要包括用户接口模块、预处理模块、资源映射模块、资源分配模块以及资源监控模块。用户接口模块是

资源映射交互框架中面向用户的部分，供用户定义业务和服务请求，提交请求并获取特定的服务。预处理模块通过对用户提交的请求进行预处理，为进一步的快速处理提供保障。资源映射模块是资源映射交互框架的核心模块，它通过利用 MSO-VNE 嵌入映射算法进行资源映射匹配计算，以获得更好的 QoS 服务质量。当用户请求和底层物理网络的可用资源成功映射后，资源分配模块根据资源映射模块的映射结果进行真正的资源分配和管理。同时，资源监控模块通过实时监控底层资源池中的资源使用变化情况并进行必要的同步管理，以辅助资源映射模块和资源分配模块完成它们的工作。资源映射交互框架可有效地保障 QoS 服务质量并提高用户体验质量。

3、业务流量隔离与控制系统

本文基于 Linux 流量控制模块 TC 设计了多层次结构的业务流量隔离与控制系统。通过虚拟局域网 VLAN 实现不可见性隔离，即虚拟局域网中的同一个 VLAN 内的成员可相互通信，而不同 VLAN 之间是相互隔离和不可见的。通过区分数据流量，并规定不同的数据流被发送的方式和顺序实现流量控制和互不影响性隔离。业务流量隔离与控制系统封装了 Linux TC 的控制命令，隐藏了实现细节，简化了用户操作，提高了用户体验。另外，通过开放一定的 API 控制接口，用户可以根据实际业务的需求对流量控制规则进行自定义重配置。业务流量隔离与控制系统可以有效地隔离不同用户之间的业务流量干扰，也为用户提供了一种区分不同业务、实现较高 QoS 服务质量的工具。

4、服务虚拟化平台的工程开发

服务虚拟化平台的工程开发主要包括三个部分：首先是虚拟网络嵌入算法在服务虚拟化平台 SuperNova 中的工程实现；其次是业务流量隔离与控制系统在服务虚拟化平台 SuperNova 中的工程实现；最后是为提高用户体验质量而附加的部分功能实现，包括实例管理，存储管理，安全组管理等。无论是哪个模块组件，在工程实现时都需要充分考虑可实现性和高效率性，因此，可以根据具体情况做相应的简化。譬如，对于虚拟网络嵌入算法，可通过由用户指定集群区域，从而降低聚类算法的复杂度；考虑到链路映射的复杂性，可通过弱化链路映射来提高映射效率等。服务虚拟化平台 SuperNova 最终要面向用户提供服务，因此，平台的用户体验质量是至关重要的。通过对云控制器进行可视化改进，使得用户不再需要面对非常不人性化的终端屏幕，也不再需要输入繁复易错的命令。而是通过在 Web 页面上进行直观的点选操作就可以完成同样的

功能，是对用户操作的一种人性化设计，可很好地提高用户体验质量。

6.2　今后工作展望

本文围绕面向媒体业务的虚拟化网络服务涉及的虚拟网络嵌入、资源映射交互、业务流量隔离与控制等方面做了一定的研究和探索，通过设计仿真实验验证，得到了一些有意义的方法和结论。但面向媒体业务的虚拟化网络服务还有很多方面需要进一步研究，主要包括：

1、面向媒体业务的虚拟网络嵌入算法 MSO-VNE 还没有充分利用媒体业务的特征，譬如媒体类型，服务类型等特征。进一步的研究可针对不同的服务请求制定不同的虚拟网络嵌入策略，因地制宜，以获取更高的性能。另外，为了进一步提高底层物理网络的资源利用率，可以对已嵌入的虚拟网络请求进行迁移，减少底层物理网络中的碎片资源。

2、面向媒体业务的资源映射交互框架还可以通过更优化的用户接口操作方式来提高用户体验质量。一方面，可以提供多样化的交互操作方式；另一方面，可以加强直观可见的操作方式，譬如图形化界面、自定义组件拖拽操作等。另外，面向媒体业务的资源映射交互框架的核心组件是资源映射模块，在进一步的研究中，可以实现多种资源映射算法供用户选择，以区别对待用户的不同类型请求，提供更高的性能和更好的体验质量。

3、本文提出的业务流量隔离与控制系统只适用于从物理网卡往外发送的数据包和数据流量。然而，从整个网络数据来看，从物理网卡接收的数据包也占了相当一部分。如果外部流量蜂拥般地流向系统，同样会引起业务之间的干扰，导致业务的不稳定。因此，进一步的研究可以对从物理网卡进入系统的流量进行隔离和控制。

参考文献

[1] 中国互联网络信息中心. 中国互联网络发展状况统计报告 [EB/OL]. http://www.cnnic.cn/hlwfzyj/hlwxzbg/hlwtjbg/201403/P020140305346585959798.pdf. 2014.

[2] 罗军舟，金嘉晖，宋爱波等. 云计算: 体系架构与关键技术[J]. 通信学报，2011，32(7)：3－18.

[3] 陈康，郑纬民. 云计算: 系统实例与研究现状[J]. 软件学报，2009，20(5)：1337 －1348.

[4] Salesforce[EB/OL]. http://www.salesforce.com/. 2014.

[5] CRM 客户关系管理[EB/OL]. http://www.salesforce.com/cn/crm/what-is-crm.jsp. 2014.

[6] Amazon [EB/OL]. http://www.amazon.com/. 2014.

[7] Amazon EC2 [EB/OL]. https://aws.amazon.com/cn/ec2/. 2014.

[8] Amazon S3 [EB/OL]. http://aws.amazon.com/cn/s3/. 2014.

[9] Amazon SQS [EB/OL]. http://aws.amazon.com/cn/sqs/. 2014.

[10] Google App Engine [EB/OL]. http://code.google.com/appengine/. 2014.

[11] A. Zahariev, Google App Engine, in TKK T-110.5190 Seminar on Internetworking, 2009, pp. 1-5.

[12] MicroSoft Azure [EB/OL]. http://www.microsoft.com/windowsazure/. 2014.

[13] OpenStack [EB/OL]. http://www.openstack.org/. 2014.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System [C]. In Cloud Computing and Applications 2008 (CCA08), 2008.

[15] Eucalyptus [EB/OL]. http://www.eucalyptus.com/. 2014.

[16] OpenNebula [EB/OL]. http://opennebula.org/. 2014.

[17] CloudStack [EB/OL]. http://incubator.apache.org/cloudstack/. 2014.

[18] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities[C]. In Proc. 10th IEEE Int. Conf. High Performance Computing and Communications. pp. 5-13. 2008.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing[EB/OL]. EECS Dept., Univ. California, Berkeley, No. UCB/EECS-2009-28. https://radlab.cs.berkeley.edu/.

[20] Cloud Security Alliance. Security Guidance for Critical Areas of Focus in Cloud Computing V3.0[EB/OL]. http://www.cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf. 2011.

[21] U.S. National Institute of Standards and Technology. NIST Working Definition of Cloud Computing/NIST 800-145[EB/OL]. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. 2011.

[22] OpenStack Compute[EB/OL]. http://www.openstack.org/software/openstack-compute/. 2014.

[23] Nova[EB/OL]. https://github.com/openstack/nova. 2014.

[24] OpenStack Storage[EB/OL]. http://www.openstack.org/software/openstack-storage/. 2014.

[25] OpenStack Shared Services[EB/OL]. http://www.openstack.org/software/openstack-shared-services/. 2014.

[26] OpenStack Dashboard[EB/OL]. http://www.openstack.org/software/openstack-dashboard/. 2014.

[27] Eucalyptus Cloud Level[EB/OL]. https://www.eucalyptus.com/eucalyptus-cloud/iaas/architecture/cloud-level. 2014.

[28] Eucalyptus Cluster Level[EB/OL]. https://www.eucalyptus.com/eucalyptus-cloud/iaas/architecture/cluster-level. 2014.

[29] Eucalyptus Node Level[EB/OL]. https://www.eucalyptus.com/eucalyptus-cloud/iaas/architecture/node-level. 2014.

[30] S. Devine, E. Bugnion, and M. Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture[P]. US Patent, 6397242, Oct. 1998.

[31] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors[C]. In Proc. EuroSys, pp. 275 – 287, 2007.

[32] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization[C]. In Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19), pp. 164-177. ACM Press, 2003.

[33] D. G. Andersen. Theroretical approaches to node assignment[Z]. unpublished Manuscript. 2002.

[34] J. Kleinberg. Approximation algorithms for disjoint paths problems[D]. Massachusetts Institute of Technology. Ph.D. dissertation. 1996.

[35] S. Kolliopoulos and C. Stein. Improved approximation algorithms for upsiplitable flow problems[C]. In Proc. Foundations of Computer Science, IEEE. pp. 426-436. 1997.

[36] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration[J]. ACM SIGCOMM Computer Communication Review. 38(2) : 17-29. 2008.

[37] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual Network Embedding with Coordinated Node and Link Mapping[C]. In INFOCOM 2009, IEEE. pp. 783-791. 2009.

[38] W. Cong, S. Shanbhag, and T. Wolf. Virtual network mapping with traffic matrices[C]. In Communications (ICC), 2012 IEEE International Conference on, pp. 2717-2722. 2012.

[39] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein. Virtual Network Embedding with Opportunistic Resource Sharing[J]. Parallel and Distributed Systems, IEEE Transactions on, pp. 1-1. 2013.

[40] M. Rahman and R. Boutaba. SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization[J]. Network and Service Management, IEEE Transactions on, pp. 1-14. 2013.

[41] A. Fischer, J. Botero, M. Beck, H. De Meer, and X. Hesselbach. Virtual Network Embedding: A Survey[J]. Communications Surveys & Tutorials, IEEE. pp. 1-19. 2013.

[42] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann. VNR algorithm: A greedy approach for virtual networks reconfigurations[C]. In Global Telecommunications Conference (GLOBECOM2011), 2011 IEEE. pp. 1-5. 2011.

[43] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components[C]. In INFOCOM 2006. 25th IEEE International Conference on Computer Communications. pp. 1-12. 2006.

[44] G. Sun, H. Yu, V. An, and L. Li. A cost efficient framework and algorithm for embedding dynamic virtual network requests[J]. Future Generation Computer Systems. 29(5) : 1265-1277. 2013.

[45] C. Marquezan, L. Granville, G. Nunzi, and M. Brunner. Distributed autonomic resource management for network virtualization[C]. In Network Operations and Management Symposium (NOMS), 2010 IEEE. pp. 463-470. 2010.

[46] Z. Cai, F. Liu, N. Xiao, Q. Liu, and Z. Wang. Virtual network embedding for evolving networks[C]. In Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE. pp. 1-5. 2010.

[47] M. Bienkowski, A. Feldmann, D. Jurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer. Competitive analysis for service migration in vnets[C]. In Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architecture, sser. VISA ’ 10. New York, NY, USA: ACM. pp.17-24. 2010.

[48] Z. Shun-li and Q. Xue-song. A novel virtual network mapping algorithm for cost minimizing[J]. Cyber Journals: J. Selected Areas in Telecommunications (JSAT). 02(01) : 1-9. 2011.

[49] J. Fan and M. H.Ammar. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies[C]. In INFOCOM 2006. 25th IEEE International Conference on Computer Communications. pp. 1-12. 2006.

[50] Y. Xin, I. Baldine, A. Mandal, C. Heermann, J. Chase, and A. Yumerefendi. Embedding virtual topologies in networked clouds[C]. The 6th International Conference on Future Internet Technologies, ser. CFI ’ 11. New York, NY, USA: ACM. pp. 26-29. 2011.

[51] B. Lv, Z. Wang, T. Huang, J. Chen, and Y. Liu. Virtual resource organization and virtual network embedding across multiple domains[C]. In Multimedia Information Networking and Security (MINES), 2010 International Conference on, pp. 725 – 728. 2010.

[52] A. Leivadeas, C. Papagianni, and S. Papavassiliou. Efficient resource mapping framework over networked clouds via iterated local search based request partitioning[J]. IEEE Trans. Parallel Distrib. Syst., 99 : 1-1. 2012.

[53] I. Houidi, W. Louati, W. B. Ameur, and D. Zeghlache. Virtual network provisioning across multiple substrate networks[J]. Computer Networks, special Issue on Architectures and Protocols for the Future Internet. 55(4) : 1011-1023. 2011.

[54] X. Zhang, C. Phillips, and X. Chen. An overlay mapping model for achieving enhanced qos and resilience performance[C]. In Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2011 3rd International Congress on, pp. 1-7. 2011.

[55] J. Shamsi and M. Brockmeyer. Qosmap: Qos aware mapping of virtual networks for resiliency and efficiency[C]. In Globecom Workshops, 2007 IEEE. pp. 1-6. 2007.

[56] J. Shamsi and M. Brockmeyer. Efficient and dependable overlay networks[C]. In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, pp. 1-8. 2008.

[57] J. Shamsi and M. Brockmeyer. Qosmap: Achieving quality and resilience through overlay construction[C]. In Internet and Web Applications and Services, 2009. ICIW ’ 09. Fourth International Conference on, pp. 58-67. 2009.

[58] A. Razzaq and M. Rathore. An approach towards resource efficient virtual network embedding[C]. Evolving Internet (INTERNET), 2010 Second International Conference on, pp. 68-73. 2010.

[59] D. Eppstein. Finding the k shortest paths[J]. SIAM J. Comput., 28 : 652 – 673. 1999.

[60] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems[J]. SIAM Journal of Computing. 5 : 691-703, 1976.

[61] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping[C]. In IEEE INFOCOM. IEEE Infocom. 2009.

[62] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking[J]. SIGCOMM Comput. Commun. Rev., 41 : 38 – 47. 2011.

[63] PageRank[EB/OL]. http://en.wikipedia.org/wiki/PageRank. 2014.

[64] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web[R]. Stanford InfoLab, Technical Report 1999-66, 1999.

[65] Hubert Bert, et al. Linux Advanced Routing & Traffic Control HOWTO[EB/OL]. http://www.lartc.org/lartc.pdf. 2014.

[66] Hubert Bert. Linux advanced routing and traffic control[C]. In Proceedings of Ottawa Linux Symposium, Ottawa, Canada, pp. 213 – 222. 2002.

[67] K. van der Laan. FIFO and LIFO incognito. Nederlandstalige TEX Gebruikersgroep[J]. pp. 121-121, 1992.

[68] TC pfifo[EB/OL]. http://linux.die.net/man/8/tc-pfifo. 2014.

[69] TC bfifo[EB/OL]. http://linux.die.net/man/8/tc-bfifo. 2014.

[70] TC pfifo_fast[EB/OL]. http://linux.die.net/man/8/tc-pfifo_fast. 2014.

[71] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance[J]. Networking, IEEE/ACM Transactions on, vol. 1, pp. 397-413. 1993.

[72] D. Lin and R. Morris. Dynamics of random early detection[C]. In ACM SIGCOMM Computer Communication Review, pp. 127-137. 1997.

[73] P. E. McKenney. Stochastic fairness queueing[C]. In INFOCOM'90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration, IEEE. pp. 733-740. 1990.

[74] S. S. V. Appala and B. Erimli. Weighted fair queuing approximation in a network switch using weighted round robin and token bucket filter[P]. Google Patents. 2005.

[75] C. Fragouli, V. Sivaraman, and M. B. Srivastava. Controlled multimedia wireless link sharing via enhanced class-based queuing with channel-state-dependent packet scheduling[C]. In INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE. pp. 572-580. 1998.

[76] J. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, and O. Sallent. A hierarchical token bucket algorithm to enhance QoS in IEEE 802.11: proposal, implementation and evaluation[C]. In Vehicular Technology Conference, pp. 2659-2662. 2004.

[77] Boto [EB/OL]. https://github.com/boto/boto. 2014.

[78] M. F. Sanner. Python: a programming language for software integration and development[J]. J Mol Graph Model, vol. 17, pp. 57-61, 1999.

[79] Garrett JJ. Ajax: A New Approach to Web Applications[EB/OL]. 2005. https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf. 2014.

[80] J. Gosling. The Java language specification[M]. Addison-Wesley Professional, 2000.

[81] GWT, Google Web Toolkit [EB/OL]. http://www.gwtproject.org/. 2014.

[82] Package java.lang [EB/OL]. http://docs.oracle.com/javase/7/docs/api/java/lang/package-summary.html. 2014.

[83] Pakage java.util [EB/OL]. http://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html. 2014.

[84] Meyer, J. and T. Downing. Java virtual machine[M]. O'Reilly & Associates, Inc. 1997.

[85] Stark, R., J. Schmid, and E. Borger. Java and the Java Virtual Machine: definition, verification, validation[M]. Springer-Verlag. 2001.

[86] Jetty [EB/OL]. http://www.eclipse.org/jetty/.

[87] Mule [EB/OL].http://www.mulesoft.org/. 2014.

[88] Sosnoski, D., JiBX: Binding XML to Java Code[EB/OL]. http://jibx.sourceforge.net/. 2014.

[89] Reese, G., Database Programming with JDBC and JAVA[M]. O'Reilly Media, Inc. 2000.

[90] Haddad, Z.T.M., Implementing a TCP hole punching NAT traversal solution for P2P applications using netty[D]. University of Stirling Masters Thesis. 2010.

[91] Jamae, J. and P. Johnson. JBoss in action: configuring the JBoss application server[M]. Manning Publications Co. 2009.

[92] Axis2C [EB/OL]. http://axis.apache.org/axis2/c/core/. 2014.

发表文章目录

[1] **薛娇**，孙鹏，邓峰，王劲林. 基于触摸屏的手势遥控系统[J]. 计算机工程.（2013年5月录用）

[2] **Jiao Xue**, Jiali You, Jinlin Wang, and Feng Deng. Nodes Clustering and Dynamic Service Balance Awareness Based Virtual Network Embedding[C]. In TENCON 2013 - 2013 IEEE Region 10 Conference (31194). 2013. pp. 1-4.

[3] **Jiao Xue**, Jiali You, Jinlin Wang, and Feng Deng. Multimedia Service Oriented Virtual Network Embedding[J]. Journal of Computer Science and Technology.（审稿中）

[4] Jiali You, **Jiao Xue**, and Jinlin Wang. A Behavior Cluster Based Availability Prediction Approach for Nodes in Distribution Networks[C]. In the 38th International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013, pp. 2810-2814.

[5] 宋军，赵志强，尤佳莉，**薛娇**. 基于P2P在线信息聚合的存储策略研究[J]. 网络新媒体技术. 2012.

申请专利目录

[1] 孙鹏，**薛娇**，王劲林，朱小勇，尤佳莉，吕阳，程钢. 一种基于传感器的手势遥控方法及系统. 申请号：201210464933.7.

[2] 孙鹏，**薛娇**，王劲林，朱小勇，尤佳莉，李晓林，程钢. 一种智能终端的控制信息输入方法及系统. 申请号：201210465024.5.

[3] 尤佳莉，**薛娇**，郑鹏飞，卓煜. 一种基于分类的虚拟网络映射方法及系统. 申请号：201310247047.3.

[4] 尤佳莉，**薛娇**，郑鹏飞，卓煜. 基于虚拟网络映射算法的交互方法及系统.（申请中）

[5] 王劲林、宋军、尤佳莉、苏杭、李晓林、郑鹏飞、**薛娇**、吕阳. 一种基于P2P在线信息聚合的副本存储系统及方法. 申请号：201110374604.9.

[6] 尤佳莉、宋军、彭飞、李晓林、郑鹏飞、**薛娇**、吕阳. 一种面向区域的机顶盒P2P-Vod系统及数据预部署方法. 申请号：201210573077.9.

[7] 尤佳莉，李晓林，王劲林，郑鹏飞，宋军，吕阳，**薛娇**. 一种基于请求转发的 p2p 流媒体系统. 申请号：2012105530880.

[8] 尤佳莉，李晓林，王劲林，郑鹏飞，宋军，吕阳，**薛娇**. 一种基于反馈控制的 p2p 流媒体系统. 申请号：2012105530397.

致　　谢

值此论文完成之际，谨在此向多年来给予我关心和帮助的老师、同学、朋友和家人表示衷心的感谢!

……

谨把本文献给我最敬爱的父亲!