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 March 19, 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

March, 2014

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

摘　　要

本文是中国科学院学位论文的 LaTeX 模板。除了介绍 LaTeX 文档类 CASthesis 的用法外，本文还是一个简要的学位论文写作指南。

关键词： 中科院，学位论文，LaTeX 模板

Abstract

This paper is a thesis template of Chinese Academy of Sciences. Besides that the usage of the LaTeX document class `CASthesis`, a brief guideline for writing the thesis is also included.

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

缩略语

VNE	Virtual Network Embedding	虚拟网络嵌入
VNR	Virtual Network Request	虚拟网络请求
SN	Substrate Network	底层网络

目　　录

表　　格

插　　图

第一章　绪论

1.1　研究背景及意义

1.2　研究内容及面临的挑战

1.3　本文主要贡献

1.4　本文内容安排

第二章　相关技术背景及技术基础

2.1　引言

2.2　云计算

介绍当前的云计算背景，以及云平台

2.3　虚拟化技术

各种虚拟化技术：CPU，网络，主机等虚拟化

2.3.1　网络虚拟化技术

大背景

2.3.2　虚拟网络映射

2.4　网络多媒体

多媒体的背景，特征，应用等

第三章　面向媒体业务的虚拟网络嵌入研究

3.1　引言

1.简要介绍大背景2.简要介绍本部分内容

3.2　相关研究

国内外研究现状

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，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和映射处理阶段，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 。

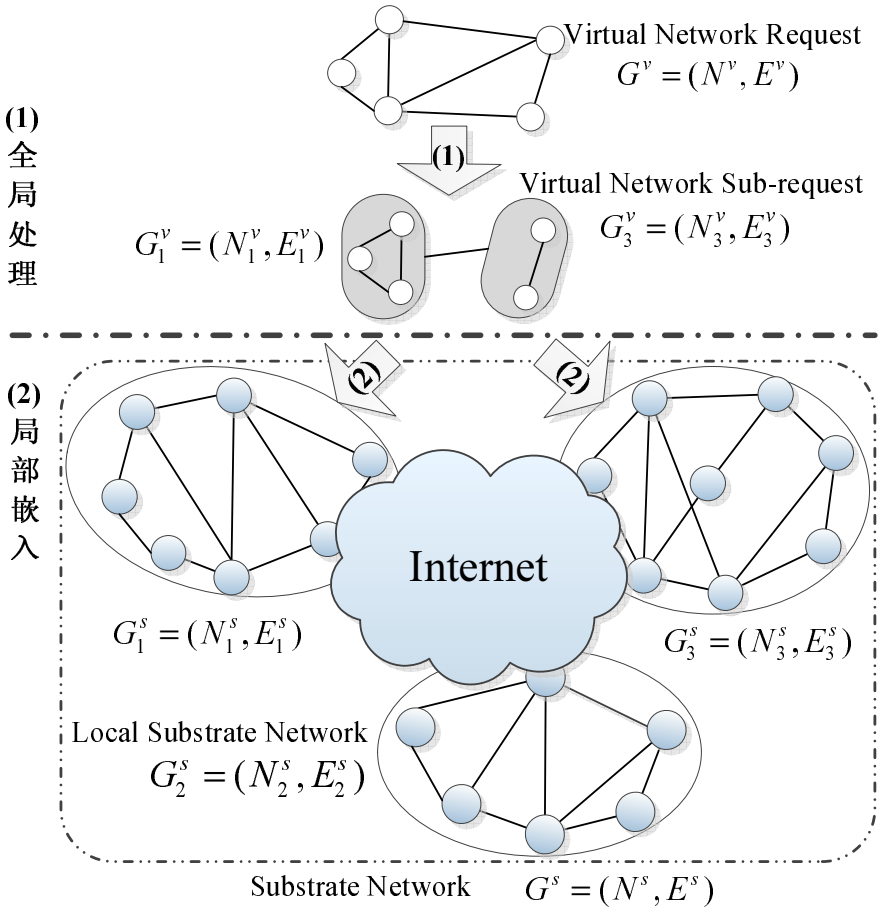

图 3.1: 面向媒体业务的虚拟网络嵌入算法整体架构

3.3.2 网络模型

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

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

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。

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，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是分布在不同的地理位置的，具有明显的区域特性性。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和延时的要求就比较苛刻，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 算法可以有效地评估 Web 页面的质量和受欢迎程度，每个页面的 rank value 排名值暗含了 Web 页面的重要性。通常，一个被很多高排名值网页链接的 Web 网页将获得一个较高的排名值，同时，一个页面链接到越多的页面，它对其他页面的排名值的贡献就越小。

受 PageRank 思想的启发，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。

由此，动态服务均衡算法可以描述如表3.3所示。动态服务均衡算法中的迭代次数是多项式时间复杂度的。

$R = LB(G, \varepsilon_m)$，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
$\varepsilon \leftarrow 0$
for all $n \in N$ do
$r_{i+1}(n) \leftarrow \sum_{m \in N} p(m, n) \cdot r_i(m)$
$\varepsilon \leftarrow \varepsilon + \|r_{i+1}(n) - r_i(n)\|$
$i \leftarrow i + 1$
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

这些原因让我们更加相信那些具有较高服务能力的底层节点，其成功嵌入的概率也会更高。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。

3.3.5 局部子请求映射

如图3.1所示，每个虚拟网络子请求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 $O(max_{0\leq i<|O^s|}(|N_i^v|^2\cdot(|N_i^s|log|N_i^s|+|E_i^s|)))$。

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

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。

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 (Georgia Tech Internetwork Topology Models)，该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。最简单的平面随机图模型是纯随机模型：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，且不能很好地反映现实网络的拓扑结构，所以在此基础上又提出了几种平面随机图模型：

1） Waxman 1 模型

该模型中，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。

2） Waxman 2 模型

该模型和 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。

3） 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。

4）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。

5） 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　性能评估

3.4.2.1　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 算法是协调了节点映射和链路映射的虚拟网络嵌入算法，该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， D-ViNE 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， D-ViNE 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 D-ViNE 算法中，整数线性规划问题是其复杂度最高的部分，它也就是整个 D-ViNE 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。同时，局域嵌入阶段可并发地嵌入多个子请求，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：

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

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 算法相比，很显然，我们提出的 MSO-VNE 算法具有更低的时间复杂度。

3.4.2.2 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 SuperNova 服务虚拟化平台而言，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。

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。

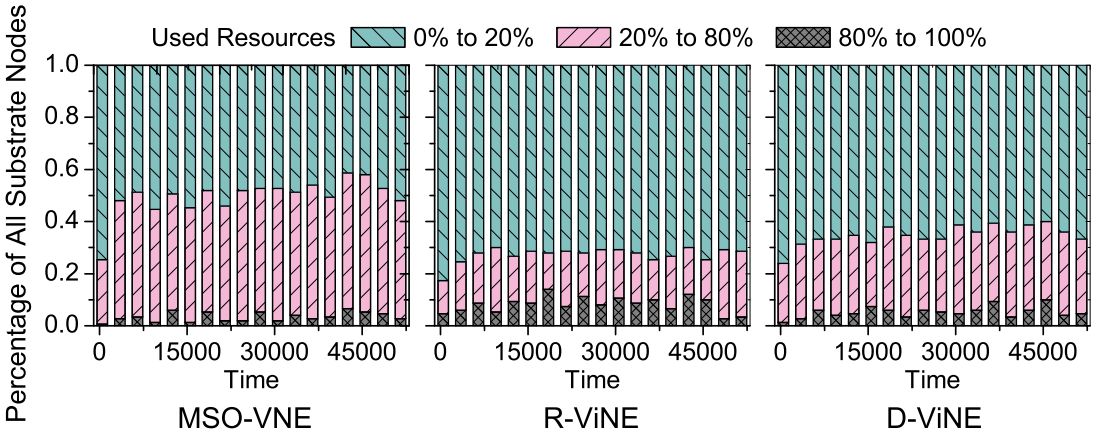

图 3.2: 底层物理网络中的节点负载分布

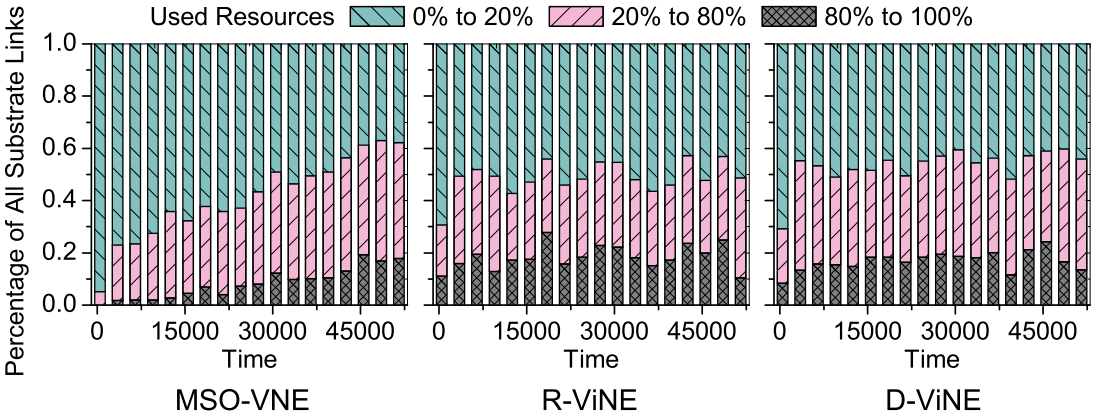

图 3.3: 底层物理网络中的链路负载分布

3.4.2.3　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 1 ，是恒定不变的。

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 $52/57 = 0.78$ 。

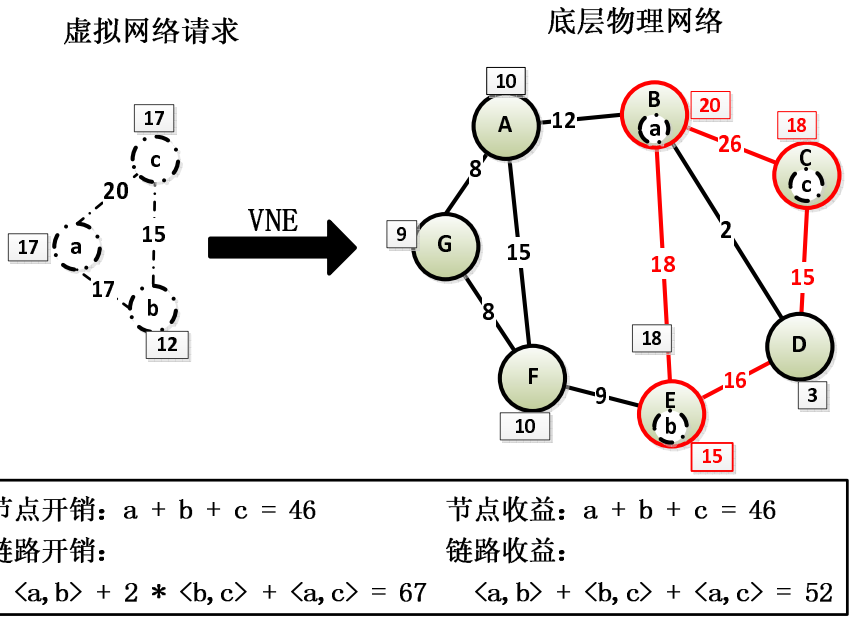

图 3.4: 虚拟网络嵌入中的链路收益开销示意图

对于底层物理网络来说，作为基础设施提供方总是追求利益最大化，希望在相同的底层物理网络上部署更多的虚拟网络请求，即追求收益开销比最大化。我们的仿真实验结果表明，如图3.5 所示，在三个算法中，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 获得相同的

收益只需要更少的开销。实际上，MSO-VNE较高的平均收益开销比来源于虚拟链路的较短的底层路径长度，进一步，这也是受益于动态服务均衡分析。

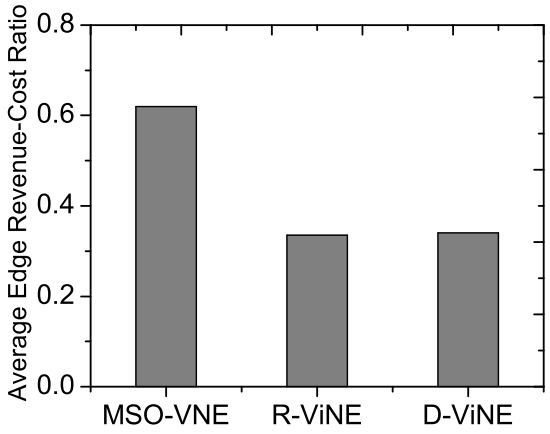

图 3.5: 平均收益开销比对比结果

3.4.2.4　平均接受率

平均接受率衡量的是已接受的请求在总请求中的比例，如图3.6所示，在所有的算法中，MSO-VNE有最高的平均接受率。因此，在相同的环境下，MSO-VNE算法能接受的请求绝不会比其他的算法少。

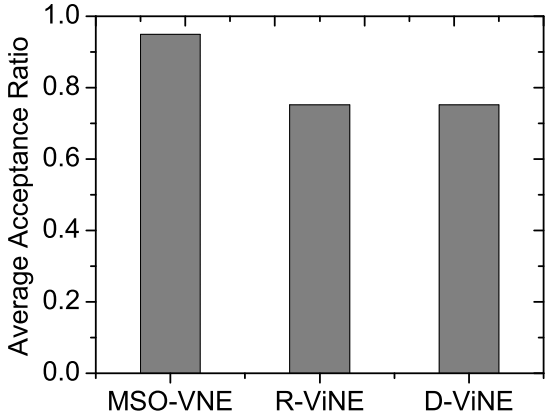

图 3.6: 平均接受率对比结果

3.4.3 小结

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，我们基于MSO-VNE算法提出了面向媒体业务的资源映射交互框架。

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，我们将其应用到资源映射交互框架中，以处理资源映射问题。由于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，可以保证媒体服务的QoS服务质量需求。

如图3.7所示，资源映射交互框架主要包括以下几个模块：

- **用户接口模块**是交互框架中面向用户的部分，用户在该模块定义他们的需求并请求特定的服务；
- **预处理模块**的职责是对用户的请求进行预处理，使得该请求不仅是人类可理解的，更是计算机可理解的；
- **资源映射模块**是交互框架的核心，我们采用MSO-VNE算法来辅助处理资源映射；
- **资源分配模块**是在当资源映射成功后，进行真正的资源分配；
- **资源监控模块**的职责是监控资源池中的任何变化并进行同步管理。

从整体来看，面向媒体业务的资源映射交互框架可以看做两层：用户层和系统层。显然，用户层的设计是面向用户的，用户通过定义他们的需求并发起请求，接着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得到处理的，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服务而不是任何其他的琐碎细节，譬如怎样预处理，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。

通常，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提取有用的信息并分析用户的需求。系统层的核心是资源映射，在资源映射

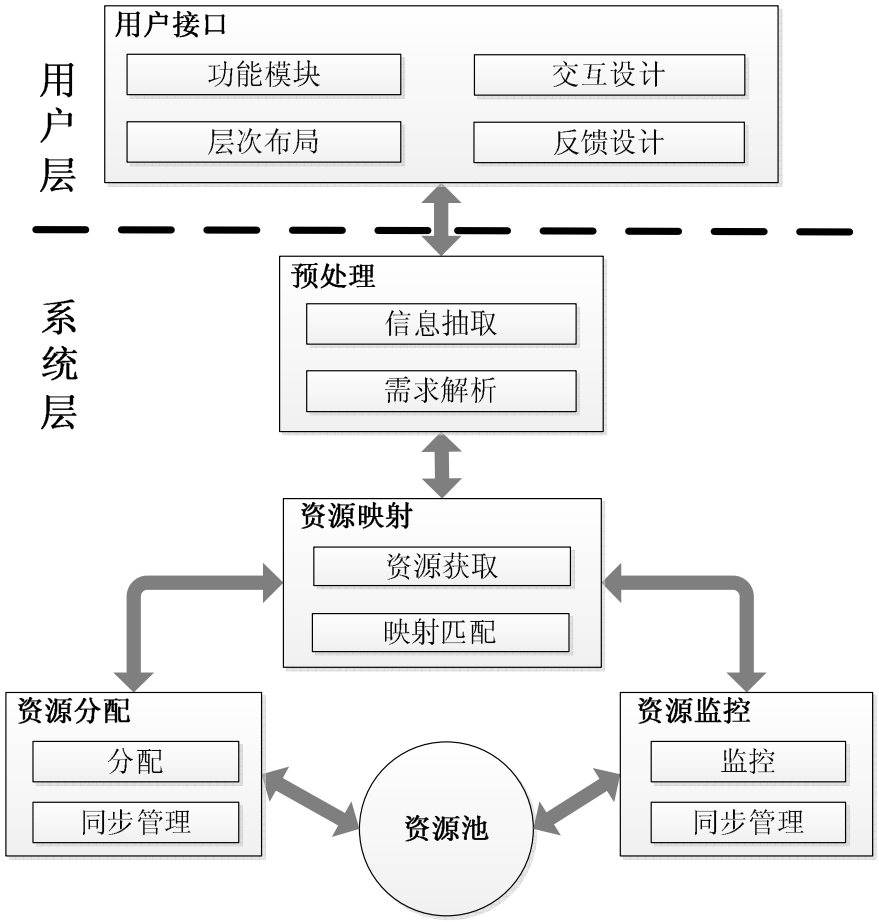

图 3.7: 面向媒体业务的资源交互系统框架

中，我们采用了MSO-VNE算法以执行映射匹配。我们从资源监控模块获取系统可用的资源，如果有足够的资源，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请求，也可以辅助用户做更好的决策。

下面，我们将详细描述交互系统中的各个模块设计。

3.5.2　用户界面

用户界面是面向用户的，其战略目标是简化用户操作，提升用户体验。用户界面模块的设计主要考虑以下几个子模块：

- 功能定义

  功能定义模块主要考虑并定义系统可能提供的功能，很显然，功能定义部分和具体的应用场景密切相关。一个良好的功能定义模块可以清晰地将系统呈现在用户面前，很好地回答“系统是什么”以及“系统能做什么”等类似的问题，留给用户良好的印象。

- 交互设计

  交互设计主要解决用户该如何和系统通信、如何交流等问题。常见的交互方式包括命令行、客户端工具、Web网页交互等。不管如何，人们越来越倾向并习惯于可视化交互。

- 反馈设计

  广义上来说，反馈设计也可以算做是交互设计的一个部分，即是系统将最终的处理结果反馈给用户，以便用户做进一步的决策等。

- 页面布局

  页面布局简单来说就是系统元素的呈现方式，如何将系统功能、交互等合理地组合布局，如何突出系统的重点，如何吸引用户的注意力等。

3.5.3 预处理

预处理是在进行资源映射前，对用户的请求进行适当的整理和预处理。预处理主要的子模块如下：

- 信息抽取

  从用户提交的请求中抽取系统需要的有用的信息，剔除冗余的信息，以减小处理复杂度并提高效率。

- 需求解析

  需求解析包括需求分类以及信息呈现方式等基本预处理，如统一格式化信息。

3.5.4　资源映射

资源映射模块主要包括以下子模块：

- 资源获取

  从资源监控模块获取系统的资源使用情况，主要关注系统的可用资源。

- 映射匹配

  映射匹配是将用户的请求和系统的可用资源进行匹配计算。我们采用的是MSO-VNE算法进行映射匹配。

3.5.5　资源监控

资源监控模块主要包括以下子模块：

- 监控
- 同步管理

顾名思义，监控模块就是监控资源池中资源的变化，并进行同步和更新管理。

3.5.6　资源分配

资源分配模块主要包括以下子模块：

- 分配
- 同步管理

资源分配模块的功能很直接明确：当映射成功的时候，为用户的请求分配真正的资源为其提供服务。并保持资源池中资源的同步和更新操作。

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，充分利用了分而治之策略和并发处理策略的优势。同

时，本章在MSO-VNE算法中应用了动态服务均衡分析，以获得较好的动态服务均衡性能。

虽然，VNE问题起源于网络虚拟化，但它仍是资源管理问题。因此，本章提出了面向媒体业务的资源映射交互框架，在交互框架中采用了MSO-VNE算法作为其资源映射核心部件。

为了评估MSO-VNE算法的性能，我们设计了仿真实验。仿真结果表明MSO-VNE算法具有很好的性能，包括较低的时间复杂度、更好的负载均衡、更高的收益开销比和更高的平均接受率。这些良好的性能进一步给我们更大的信心，使我们更加相信基于MSO-VNE的交互框架可以获得更好的QoS服务质量和更好的用户体验。

第四章　业务流量隔离控制

4.1　引言

1.简要介绍大背景在传统的TCP/IP网络的路由器中，所有的IP数据包的传输都是采用FIFO（先进先出），尽最大努力传输的处理机制。在早期网络数据量和关键业务数据不多的时候，并没有体现出非常大的缺点，路由器简单的把数据报丢弃来处理拥塞。但是随着计算机网络的发展，数据量的急剧增长，以及多媒体，VOIP数据等对延时要求高的应用的增加。路由器简单丢弃数据包的处理方法已经不再适合当前的网络。单纯的增加网络带宽也不能从根本上解决问题。所以网络的开发者们提出了服务质量的概念。概括的说：就是针对各种不同需求，提供不同服务质量的网络服务功能。提供QoS能力将是对未来IP网络的基本要求。2.简要介绍本部分内容

4.2　流量控制原理

4.2.1　Linux TC

Linux TC，即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 利用队列规定建立处理数据包的队列，并定义队列中的数据包被发送的方式，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，TC流量控制是针对出口的流量展开的。 如图4.1所示，接收包从输入接口（Input Interface）进来后，经过入口流量监控（Ingress Policing）丢弃不符合规定的数据包，由输入多路分配器（Input De-Multiplexing）进行判断选择：

1）如果接收包的目的IP是本主机，那么将该包送给上层处理；

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（TCP、UDP等）产生的包也通过转发块进行向外传输。转发块通过查看路由表，决定所处理包的下一跳。然后，对包进行排队（Output Queuing）并传送到输出接口（Output Interface）。

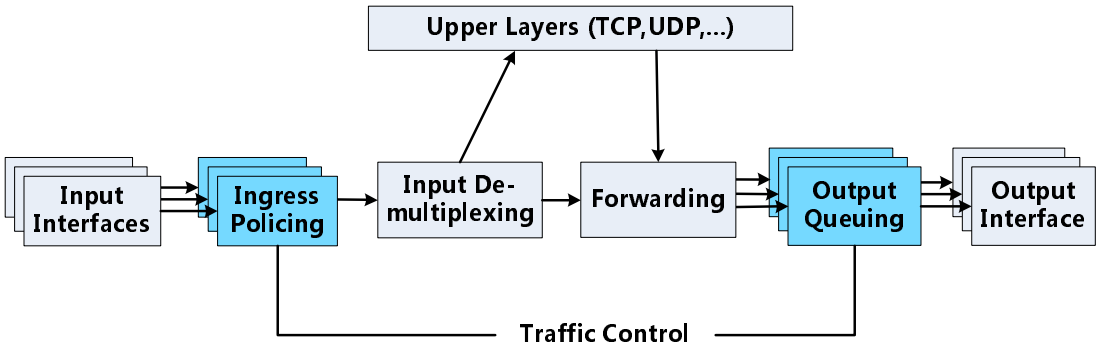

图 4.1: Linux TC流量控制模块原理示意图

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。因此，我们可以通过改变发送次序来控制传输速率。Linux 流量控制主要是在输出接口排列时进行处理和实现的。如图4.1所示，TC流量控制模块包括入口流量限制（Ingress Policing）和输出排队（Output Queuing）子模块。

TC流量控制模块进行流量控制的方式包括以下几种：

1）SHAPING

SHAPING，即流量整形，是一种主动调整流量输出速率的措施。流量整形通过限制流出某一网络的某一连接的流量，有效地控制其传输速率，使得这类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。它通常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，首先在缓冲区中进行缓存，在令牌桶的控制下再均匀地发送这些被缓冲的报文。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。一般而言，SHAPING流量整形只适用于向外的流量。

2）SCHEDULING

SCHEDULING，即流量调度，SCHEDULING通过在输入和输出之间设定一个队列，让数据包在队列中排队，并重新安排数据包被发送的方式和顺序。通过调度数据包的传输，可以在带宽范围内，按照优先级分配带宽。常见的调度策略是FIFO先进先出，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。SCHEDULING只适于向外的流量。

3）POLICING

POLICING，即流量监管，类似于SHAPING，流量监管不仅可以限制从

网络中流出的流量，而且可以限制流入网络中的流量。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，而监管则几乎不引入额外的延迟。

4）DROPPING

DROPPING通过丢弃包、流量或分类来实现对流量的控制，通常在流量超过某个设定的带宽时就会采取丢弃措施。DROPPING不仅可以对从网络中流出去的流量实施管理，还可以丢弃流入网络中的流量，从而实现网络流量的控制和平滑等。

在Linux TC流量控制模块中，主要的组件对象包括：

1）QDisc

2）Class

3）Filter

其中，QDisc即Queuing Discipline，它通过利用队列并决定数据被发送的方式，从而实现流量的控制和管理。QDisc是典型的流量调度器，只适用于向外的流量，通过让数据包在队列中排队，并制定数据包被发送的方式和规则。QDisc分为Classless QDisc和Classful QDisc，即无类别QDisc 和分类QDisc。

在分类QDisc中存在Class，即类别。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QDisc。Filter过滤器是Linux TC流量控制模块中最复杂的组件，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QDisc或Class中，但是数据包会首先进入到队列的根QDisc 中，当根QDisc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，因此，数据包可能会被进一步的细分过滤，直到数据包被发送出去。

4.2.2　QDisc

QDisc，Queuing Discipline即队列规定，它是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接口配置的QDisc队列规定把数据包加入队列。然后，内核会尽可能

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

多地从队列里面取出数据包，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中主要使用以下两种队列规定实现流量控制功能：

1）无类别队列规定（Classless QDisc）

2）分类队列规定（Classful QDisc）

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。使用无类别队列规定形成的队列可以接受、重新编排、延迟或丢弃数据包。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。无类别队列主要包括以下几种：

（1）FIFO

FIFO，First-In First-Out，即先进先出，是最简单的无类别QDisc。FIFO对进入的数据包不做任何处理，数据包按照先进先出的方式通过队列。FIFO的队列有一定的容量大小，可以暂时保存网络接口一时无法处理的数据包。

FIFO可分为三种子类型：pfifo 以数据包为单位区分不同的流量并进行排队处理；bfifo 以字节为单位区分不同的流量并进行排队处理；pfifo_fast 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band，band 0的优先级最高，band 2 的最低。在每个波段里面，使用先进先出规则。如果band 0 里面有数据包，系统就不会处理band 1 里面的数据包，band 1和band 2 之间的关系也是如此。内核遵照数据包的TOS 标记，把不同的数据包分配到三个不同的波段里面。

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。

（2）RED

RED，Random Early Detection，即随机早期探测，它通过监测队列的平

均大小，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，其平均长度也随之增加，导致新传入的数据包被丢弃的概率也增加。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。

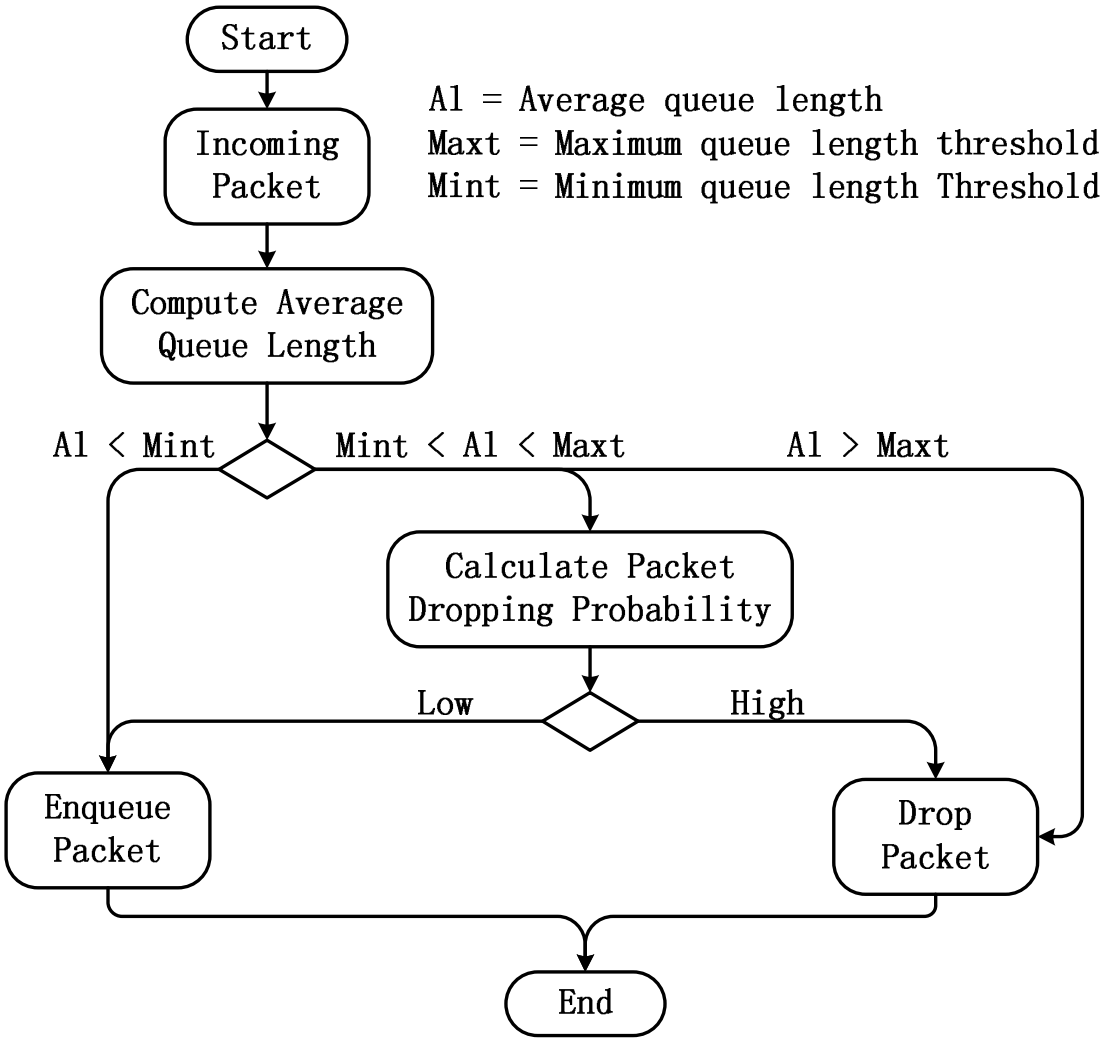

图 4.2: RED的排队规则示意图

RED通常用来防止网络拥塞，是端到端的TCP拥塞控制的补充。它通过预测网络的拥塞，并提前采取丢包动作，而不是等到网络实际拥塞了以后再采取措施。

（3）SFQ

SFQ，Stochastic Fairness Queuing，即随机公平队列，SFQ的关键是“会话”或者“流”，它主要针对一个TCP会话或者UDP流。流量被分成相当多数量的FIFO队列中，每个队列对应一个会话。数据按照简单轮转的方式发送，每个会话都按顺序得到发送机会。因此，SFQ非常公平，它保证了每一个会话都不会没其它会话所淹没。SFQ之所以被称为“随机”，是因为它并不是真的为每一个会话创建一个队列，而是使用一个散列算法，把所有的会话映射到有限的几个队列中去。因为使用了散列，所以可能多个会话分配在同一个队列里，从而需要共享发包的机会，也就是共享带宽。为了不让这种效应太明显，SFQ会频繁地改变散列算法，以便把这种效应控制在几秒钟之内。

在SFQ队列中，只有当出口网卡确实已经挤满了的时候，它才会起作用。否则，机器中根本就不会有队列，SFQ也就不会起作用。

（4）TBF

TBF，Token Bucket Filter，即令牌桶过滤器。TBF只允许以不超过事先设定的速率到来的数据包通过，但可能允许短暂突发流量超过设定值。

TBF的实现在于一个缓冲器，也就是令牌桶。它不断地被“令牌”以特定速率填充。令牌桶最重要的参数就是它的大小，代表了它能够存储的令牌数量。每个到来的令牌从数据队列中收集一个数据包，然后从桶中被删除。在TBF中，令牌流和数据流的速率的不同组合产生不同的情景：

1）数据流的速率==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）数据流的速率< 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，这种情况下会发生突发传输。

3）数据流的速率> 令牌流的速率。这意味着桶里的令牌很快就会被耗尽。导致TBF中断一段时间，称为“越限”。如果数据包持续到来，将发生丢包。

最后一种情景非常重要，因为它可以用来对数据通过过滤器的速率进行整形。令牌的积累可以导致越限的数据进行短时间的突发传输而不必丢包，但是持续越限的话会导致传输延迟直至丢包。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。

以上介绍的是常见的无类别队列规定，下面将介绍常见的分类队列规定。

分类队列规定是对进入网络设备的数据包根据不同的需求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，Class Based Queuing，基于分类的队列。CBQ是一种用于网络调度器的队列技术，它不仅可以用来分类，它本身可以实现流量整形。CBQ的分类可以基于不同的参数，比如优先级、接口、应用程序（端口）等。它允许流量在被按类分组后均衡分享带宽，既有限制带宽的能力，也具有带宽优先级管理的能力。带宽限制，也就是流量整形，是通过计算连接的空闲时间完成的，例如：如果试图把一个10Mbps的连接整形成1Mbps的速率，就应该让链路90%的时间处于闲置状态。然而，闲置时间的测量非常困难，CBQ 采用的是近似值：两个传输请求之间的毫秒数，这个参数可以近似地表征链路的繁忙程度。

（2）HTB

HTB，Hierarchy Token Bucket，分层的令牌桶。HTB基于令牌和桶，实现了一个复杂而又精细的流量控制方法。它允许用户创建一系列具有不同参数的令牌桶，并按需要将这些令牌桶归类，并实现细粒度的流量控制。HTB 最常用的功能就是流量整形，限制出口带宽速率。另外，HTB的一个重要的特性就是租借模型。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。因此，HTB不仅可以保证每个类别的带宽，还可以允许特定的类可以突破带宽上限，占用别的类的带宽。

（3）PRIO

PRIO队列规定，不能限制带宽，从而不能实现流量整形。它仅仅根据事先配置的过滤器把流量进一步细分。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，而不是简单的FIFO。同时，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

在Linux TC中，所有的队列规则、类和过滤器都有标识符ID，可以手工设置，也可以由内核自动分配。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，需要为有子类的队列规则显式地分配一个句柄。

在同一个队列规则里面的类共享这个队列规则的主序列号，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则有关，和父类无关。类的命名习惯和队列规则的相同。

过滤器的ID有三部分，只有在对过滤器进行散列组织才会用到。

如表4.2所示，可以通过使用Linux TC的基本操作命令对QDisc、Class和Filter进行配置和操作。

4.3 业务流量隔离方案

本节将详细介绍流量控制方案的设计，包括整体架构、设计原则以及功能接口。

4.3.1 整体架构

流量控制模块的整体架构如图4.3所示。所有的Class组成一颗流量层次树，每个Class都只有一个父Class，而一个Class可以有多个孩子Class。某些QDisc，如CBQ和HTB，允许在运行时动态添加Class，而其它的QDisc，如PRIO，不允许动态建立Class。允许动态添加Class的QDisc 可以有零个或者多个孩

命令	适用范围	描述
add	QDisc, Class, Filter	在一个节点里加入一个QDisc、类或者过滤器。添加时，需要传递一个祖先作为参数，传递参数时既可以使用ID 也可以直接传递设备的根。如果要建立一个QDisc或者过滤器，可以使用句柄来命名；如果要建立一个类，可以使用类识别符来命名
remove	QDisc	删除某个句柄指定的QDisc，根QDisc也可以删除。被删除QDisc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。handle和parent不能修改，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

子Class，由它们为数据包排队。在流量层次树的叶子类中，必须配置相应的无类别叶子QDisc。默认情况下，系统的叶子QDisc是pfifo_fast，它采用三个不同优先级的波段采用先进先出的方式进行排队，当然，用户也可以配置使用其它类型的QDisc。在我们的业务流量控制模块中，根队列是分类队列规定，目前支持CBQ和HTB 这两种最常用的分类队列规定。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。为了隔离不同的用户业务，避免流量抢占引起的业务干扰，我们为每个用户分配一定的流量，并提供独立的分类和队列规定，通过开放控制接口以方便用户对其业务实现精细控制。对于每个用户，我们整体上将其流量分为内部业务流量和外部服务流量。内部业务流量主要包括用户的业务不同板块之间的内部通信流量，这部分流量不会通过外网。而外部服务流量主要是指为终端用户提供服务而产生的流量。对于一个业务而言，最重要的是为终端用户提供有QoS保证的服务，因此，我们为外部服务流量设置了5个不同的分类。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和相应的QDisc排队规则。

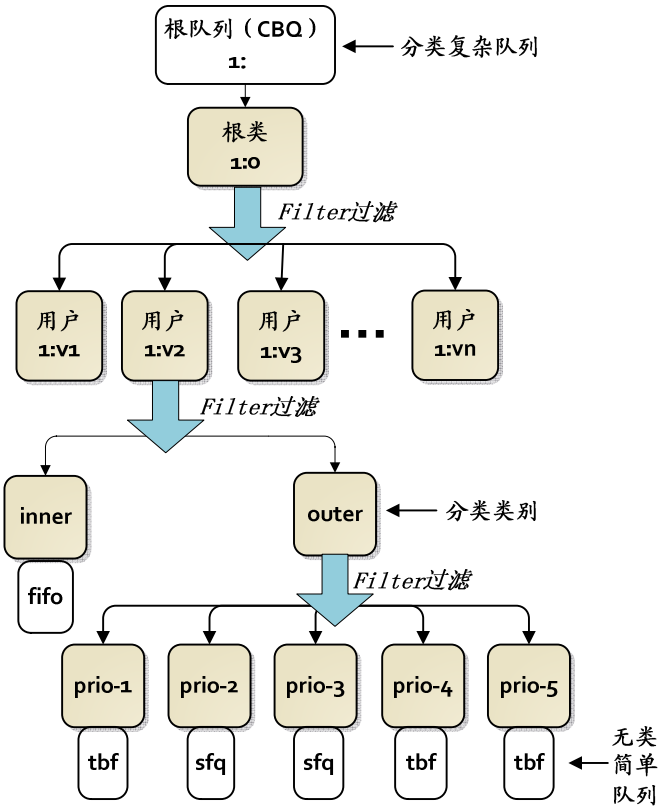

图 4.3: 流量隔离模块整体框架

4.3.2 设计细则

对于业务流量隔离模块，我们的动机主要包括以下两个方面：

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，从上而下细分流量并进行隔离控制以保证较好的QoS。从最顶层，对进入到物理网卡的流量，根据其vlan标签标识不同的用户，并将该流量导向正确的用户Class。对于每个用户而言，我们将其流量分为inner流量和outer流量。其中，inner流量是指业务内部的流量，而outer流量是外部Internet对用户业务的访问流量。根据outer外部流量的数据包特征，我们又进一步细分了5个孩子Class，区分5个不同优先级的流量并提供用户自定义分类规则。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流量的重要性和优先级。因此，给用户一定的自由度去定制分类规则

是有必要的。

我们的业务流量控制方案主要遵循了如下设计原则：

（1）层次性

层次性主要体现在自上而下的分层设计以及不同粒度的流量隔离。这种层次性设计的优势在于系统框架直观，不仅简化了系统的设计，更减轻了系统的工程开发，同时，层次设计大大提高了其横向扩展能力。

（2）封装性

封装性是指整个系统封装了基本的TC命令行操作，如add，remove，change等，开放了统一的功能接口和控制接口，并隐藏了所有TC命令行操作的细节。由于TC流量控制的队列规则、分类规则以及过滤规则的参数众多，如果完全由用户分别输入，其用户体验将非常糟糕。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控制而言，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。

（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自定义的，给用户一定的自由度。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。

4.3.3　功能接口

Linux TC流量控制器作为Linux内核的一个模块，功能非常强大，也非常灵活。其可配置和可调节的参数众多，可塑性很强。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，运转业务并提供服务。

目前，我们开放给用户的功能接口主要包括：

- **euca-setup-trafficcontrol**——启动
- **euca-config-trafficcontrol**——配置
- **euca-describe-trafficcontrol**——查询

- **euca-stop-trafficcontrol**——终止

下面，我们将详细地介绍这些功能接口。

参数	缩写	描述	默认值
device	d	需要进行流量控制的网卡设备，如eth0，eth1等	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即该用户的vlan网络	当前登录用户
实例	euca-setup-trafficcontrol		

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

1）euca-setup-trafficcontrol

该接口用于启动默认配置的流量隔离控制模块，完整的接口形式如下：

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。

2）euca-config-trafficcontrol

该接口用于配置已启动的流量隔离控制模块，完整的接口形式如下：

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。

实例：

1）修改inner叶子分类的带宽为30mbit

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

2）修改inner叶子分类的无类简单队列为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分类的过滤规则”match u8 0x40 0xf0 at 0”

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

参数	缩写	描述
device	d	需要进行流量控制的网卡设备，如eth0，eth1等，默认是eth0
level	l	所配置的类别，取值为vlanroot，inner，outer，prio1，prio2，prio3，prio4，prio5
qdisc	q	叶子qdisc类型，支持sfq和tbf，须指定level
useree	u	将要启动流量控制的用户即该用户的vlan网络，默认是当前登录用户
bandwidth	b	class的带宽和速率，须指定level参数
prio	p	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

5）为inner分类添加过滤规则”match u8 0x40 0xf0 at 0”

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

3）euca-describe-trafficcontrol

该接口用于查询显示已启动的流量控制模块，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。

4）euca-stop-trafficcontrol

该接口用于停止已启动的流量控制模块TC，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	需要查询流量控制的网卡设备，如eth0，eth1等	eth0
useree	u	要查询流量控制的用户即该用户的vlan网络	当前登录用户
实例	euca-describe-trafficcontrol		

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

参数	缩写	描述	默认值
device	d	需要终止流量控制的网卡设备，如eth0，eth1等	eth0
useree	u	要终止流量控制的用户即该用户的vlan网络	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控制模块的性能，我们设计了实验进行测试。

4.4.1 实验环境

网络问题时非常复杂的问题，尤其是Internet互联网。为了有效地测试和验证业务流量隔离模块的性能，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）通过交换机相连接。

在我们的一系列实验中，52主机是发送机，实现了发送应用程序tcsender，而218主机是接收机，实现了接受应用程序receiver。

1）链路带宽速率测量

即便是如此简单的设计，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。

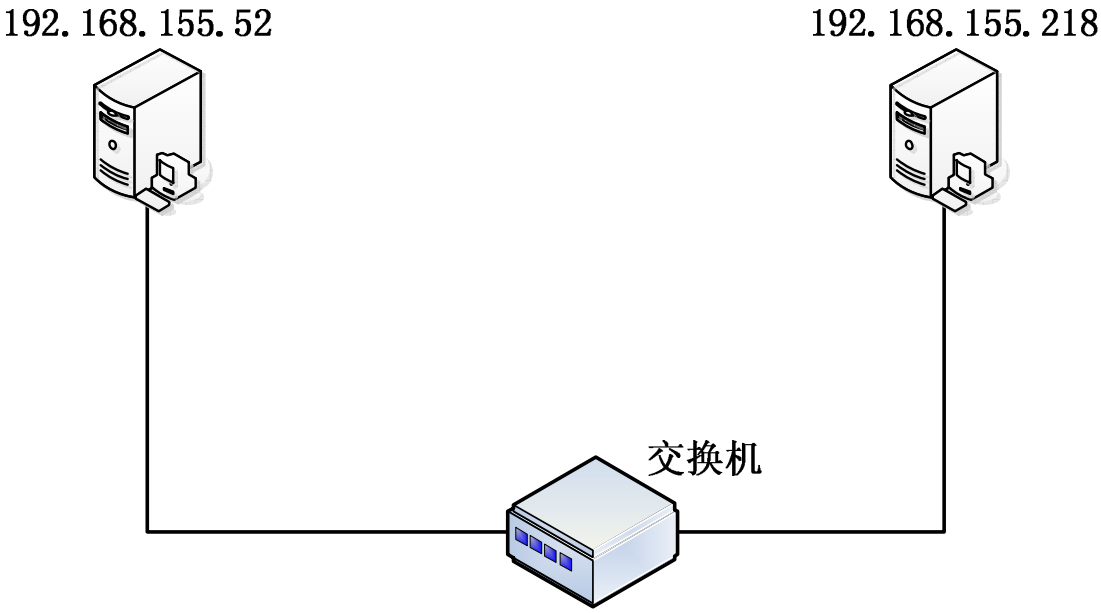

图 4.4: 流量隔离控制实验框架示意图

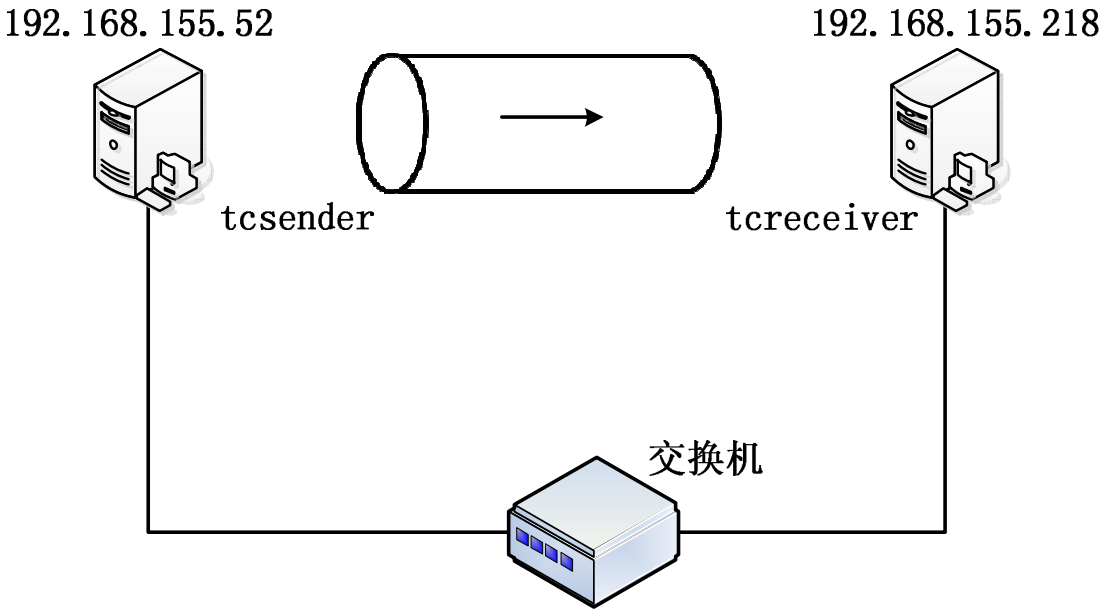

图 4.5: 测量链路带宽的实验架构示意图

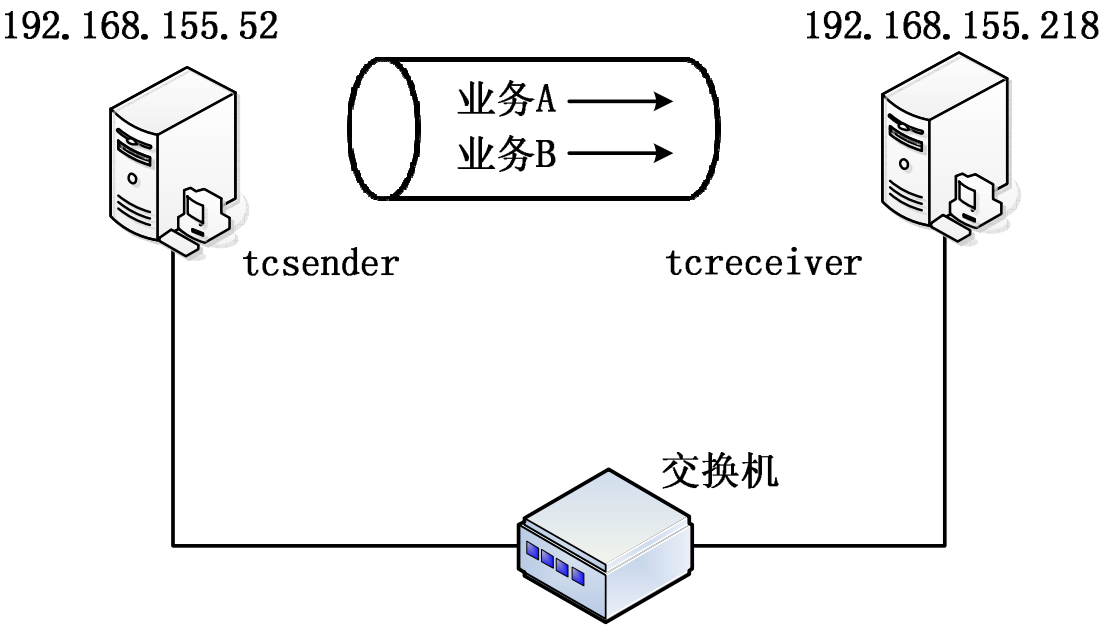

图 4.6: 测量链路上业务之间扰动的实验架构示意图

2）链路业务之间的扰动

在没有业务流量隔离和控制的网络环境中，同时存在的业务可能存在抢占流量等问题，从而导致业务之间相互干扰，使得各业务的可用带宽和数据传输速率受到影响。为此，我们设计实验测量同一链路上不同业务之间的扰动情况。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的外界，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。

3）流量隔离控制性能

为了消除不同业务之间的干扰，我们在52主机和218主机的出口网卡上设置流量隔离和控制。通过规定数据排队和发送的规则，控制出口的流量并隔离不同的业务。因此，我们设计如图4.7所示的实验架构图，测量并评估流量隔离控制系统的性能。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。

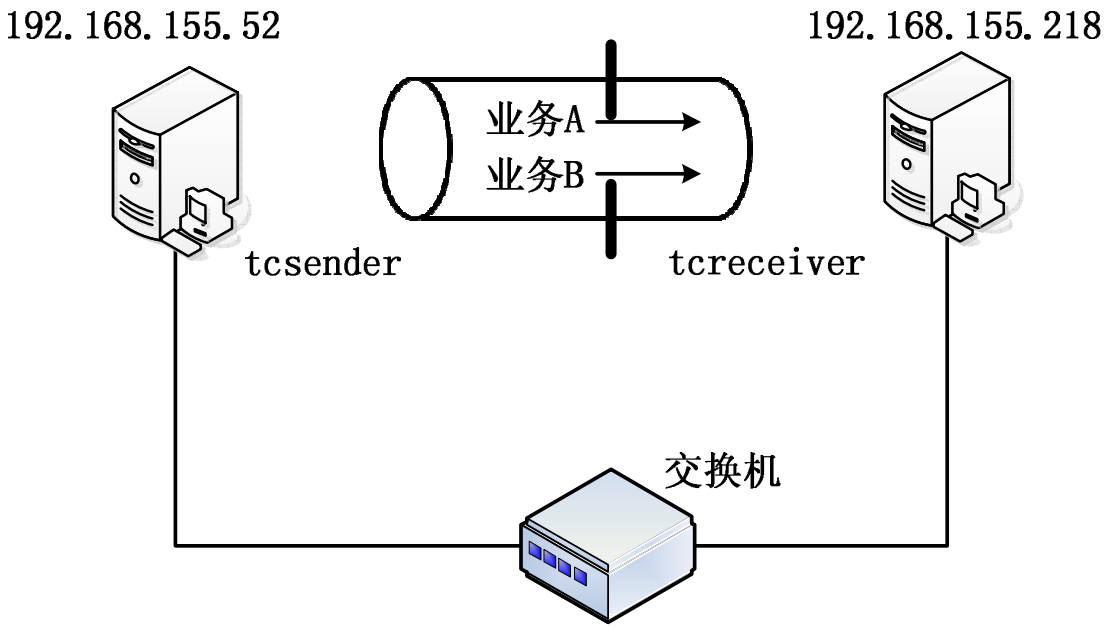

图 4.7: 流量隔离控制性能的实验架构示意图

4.4.2　实验评估

4.4.2.1　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。采用如图4.5所示的网络实验架构，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主机所在的网络是相对封闭和稳定的，因此，其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。这种相对封闭的网络环境更容易进行流量隔离和控制的性能评估。

4.4.2.2　业务干扰特征

如图4.9所示的实验结果是在没有流量隔离控制时，同时存在的业务之间的互相干扰。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。我们通过这种变化的数据传输量来模拟变化多端的外部环境对fixed业务的影响。从实验结果曲线图，我们可以很明显地

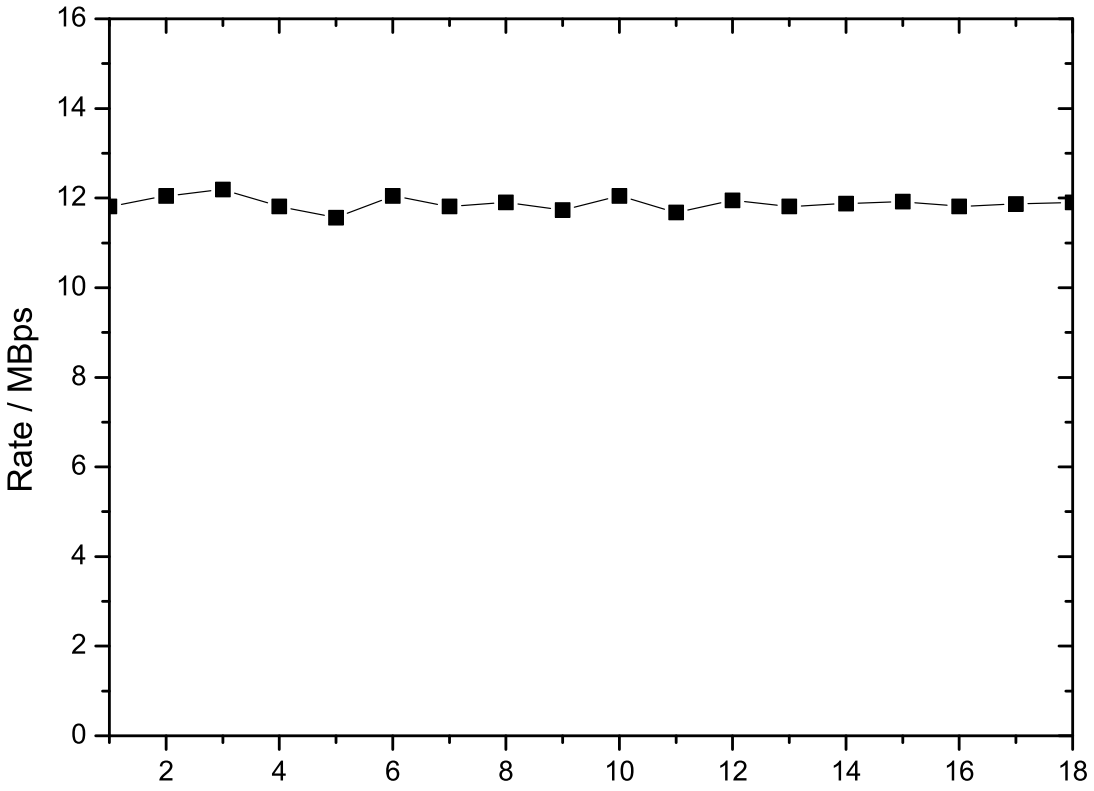

图 4.8: 链路带宽速率测量结果

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，流量隔离控制模块的重要性和急迫性。

4.4.2.3 流量隔离控制性能

如图4.10所示，当实施了流量隔离控制后，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。在流量隔离控制模块的作用下，即使variable标记的业务有着天翻地覆般变化的流量特征，它对fixed业务的影响几乎可以忽略不计。

4.5 小结

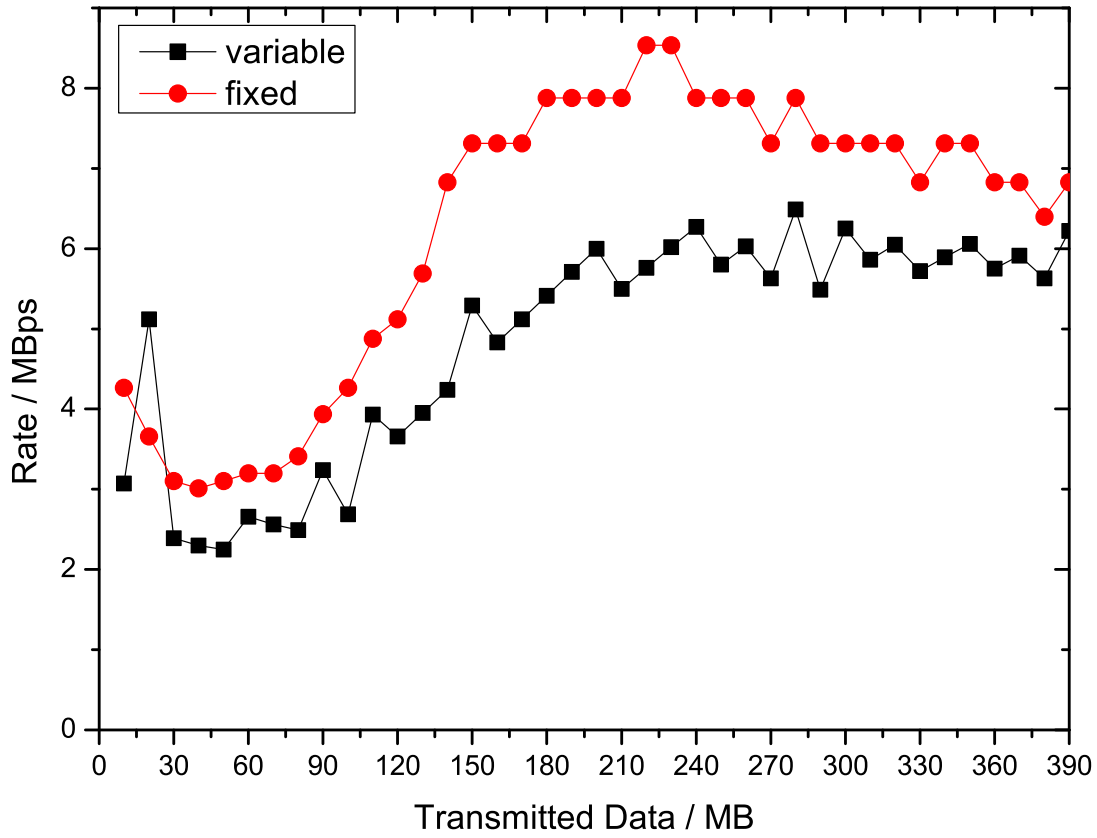

图 4.9: 在无流量隔离控制时，业务之间存在干扰

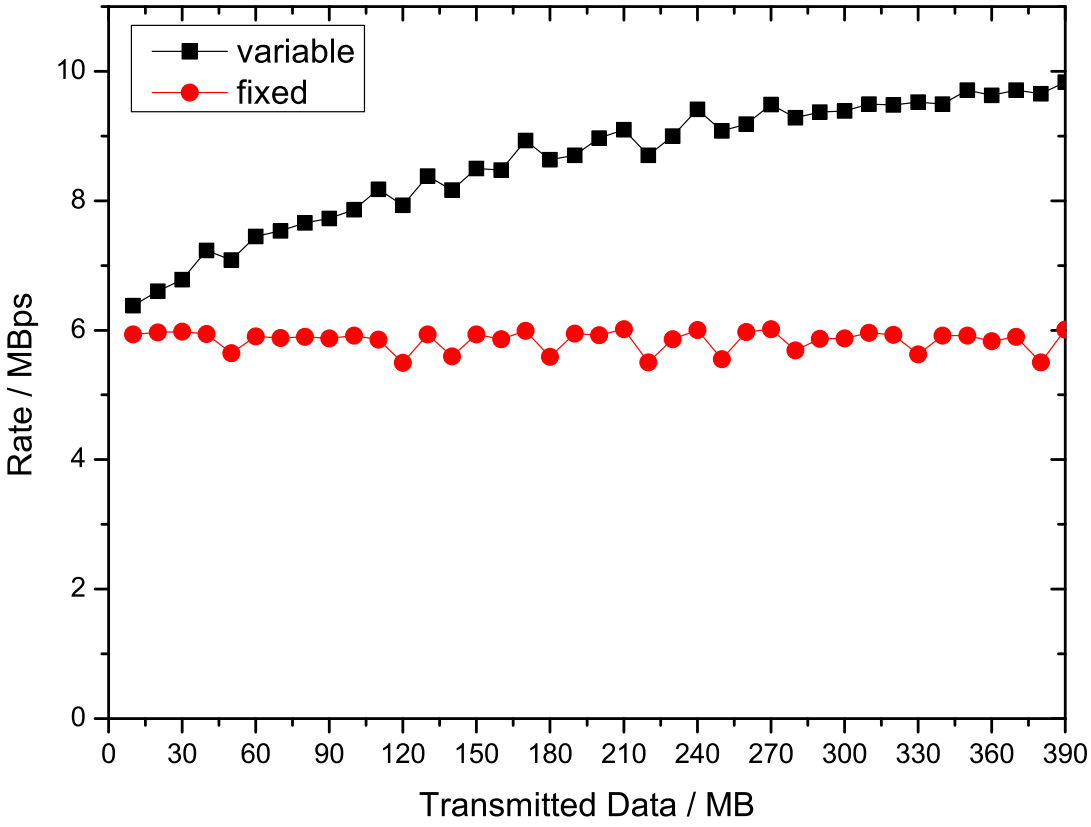

图 4.10: 在流量隔离控制下，业务的隔离情况

第五章　SuperNova系统工程开发

5.1　引言

1.简要介绍大背景2.简要介绍本部分内容

5.2　SuperNova

5.2.1　系统架构

SuperNova是一个多层的服务虚拟化平台，图5.1是其系统架构示意图。

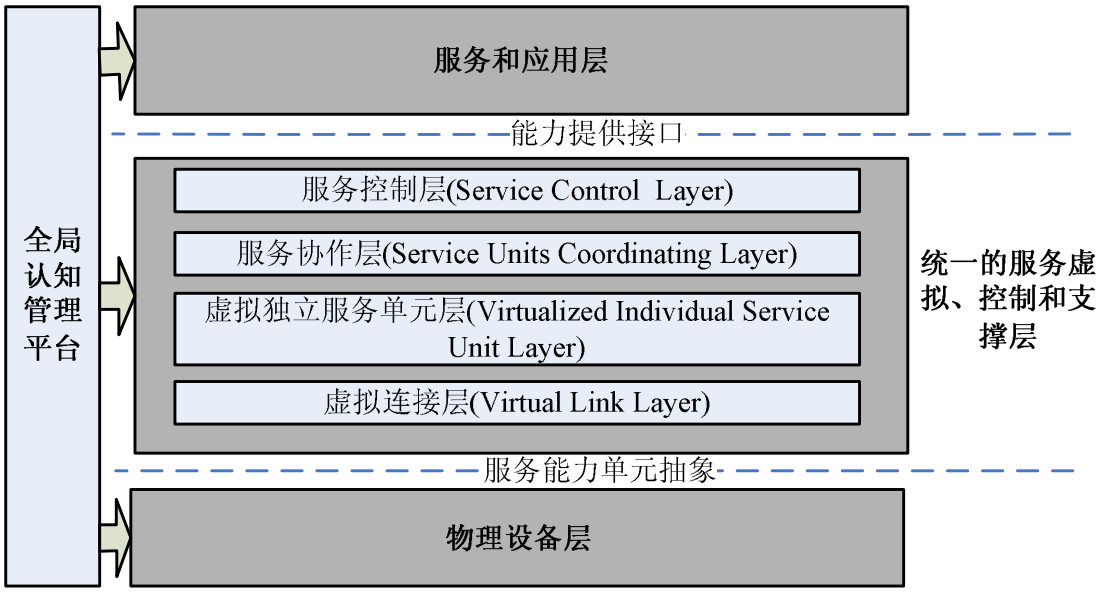

图 5.1: SuperNova系统架构示意图

其中，物理设备层是众多物理设备的集合，如RSPE，服务器等。在SuperNova中，物理设备层统一向上层提供以节点为单位的信息，包括每个节点可承载的不同服务能力单元及其能力状态；服务和应用层是指业务系统的集合，由统一的服务虚拟、控制和支撑层向其提供服务能力；全局认知管理平台具有全局监控、优化和反馈驱动的功能，可以有效动态提高系统性能。在SuperNova中，统一的服务虚拟、控制和支撑层以及全局认知管理平台是研究重点。

服务虚拟化平台的设计目标是将硬件设备进行屏蔽，以接口的方式向服务和应用层提供所需服务。应用无需了解网络细节和硬件情况，可根据需求获得弹性服务。下面将详细介绍各个部分。

5.2.1.1 物理设备层

物理设备层通过对物理设备进行管理和抽象，生成并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，作为虚拟化平台的输入，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

该层的主要任务是组织物理设备层提供的大量节点，以便为不同的应用类型提供服务虚拟化。该层是一个含四个子层的结构，包括虚拟连接层、虚拟独立服务单元层、服务协作层和服务控制层，各层功能详述如下：

1）虚拟连接层

该层主要对物理设备层提供的所有节点进行组网，使其成为一个全局可路由、可快速定位、高扩展性的自组织网络，并在物理资源之间安全交换数据。

虚拟连接层采用一种基于地理位置信息编码的网络结构，包括两层：局域管理层和连接层。局域管理层主要负责不同地理区域（也可能是根据链路状态组成的区域）节点的检索、管理和索引等工作；虚拟连接层中所有节点通过结构化P2P方式（DHT）进行组织，主要负责物理设备层所提供的节点的路由和查询，同时，地理位置信息的引入，保证了P2P网络中节点的区域聚集性；另外，P2P技术的应用，便于该层的扩展，降低了管理服务器的负载，提升了用户体验。其中，局域管理服务器由虚拟层中的某个节点担任。该虚拟连接层的结构如图5.2所示。

该结构中，全局管理服务器（Global Manager，GM）是一个全局设备，每个加入虚拟化平台的节点都首先在全局监管服务器上注册，将其IP地址、地理位置信息以及相应的能力信息进行报告，如存储空间、可支持流化路数等。如果虚拟连接层中没有节点，则该节点为虚拟连接层的第一个节点，同时，该节点也被选为所在区域的局域管理服务器。如果虚拟连接层中已有节点，GM则为节点返回一些节点信息，帮助该节点加入虚拟连接层。另外，在同一个区域

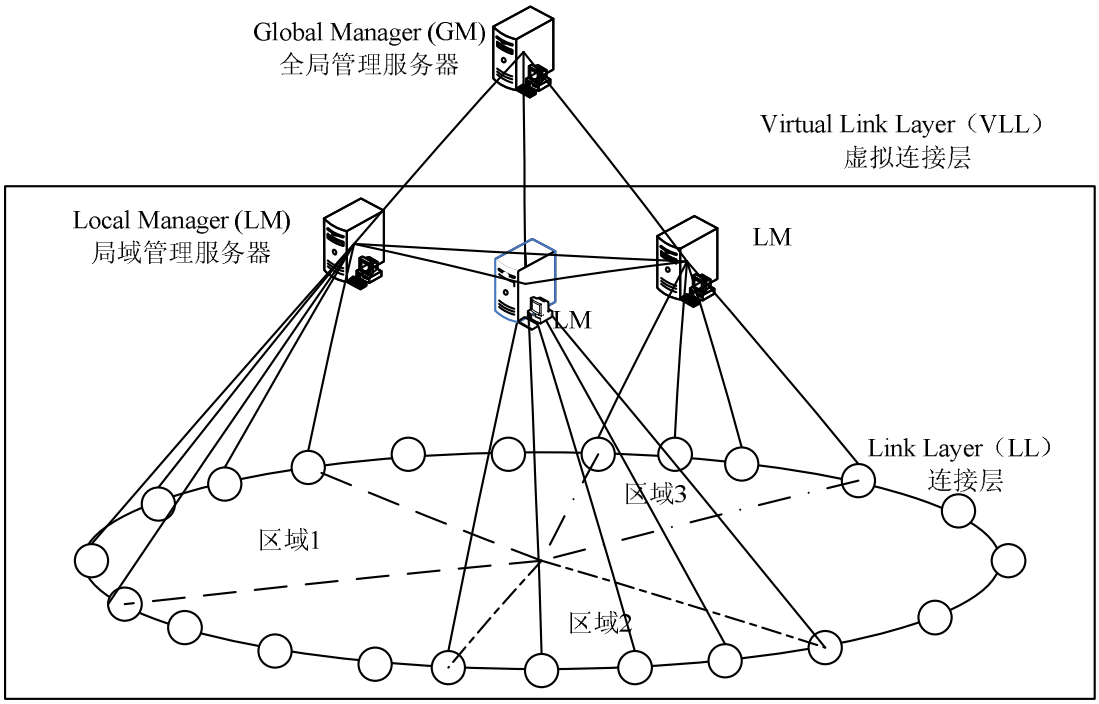

图 5.2: 虚拟连接层的结构示意图

中，如果某个节点能力在一个较长时间内远大于局域管理服务器，则将区域管理服务器中的内容同步到该节点上，并由该节点替换原局域管理服务器，保证局域管理的高效性。

这里，也提出一种基于地理位置信息的服务单元自动聚类的管理层次划分方法。在区域划分标准上，具有强制与自动两种划分指标。所有服务单元首先按照强制标准进行区域划分，当一个原有区域内的服务单元数量达到一定限度后，按照基于地理位置信息的聚类/分类方法，将原区域进行拆分，拆分后的每个子区域中选出该新区域的区域管理服务器。当区域过小时，相近区域进行合并。这样形成多层次的区域划分结构。该方法可避免某区域过大时区域管理服务器的负载过重，以及区域过小时的管理资源浪费问题。

当有服务请求时，根据服务的目标地理区域信息和全局，任意一个节点可以路由到所有区域的节点，进而知道该区域的局域服务器地址，向局域服务器发起查询请求，局域服务器对任务进行分析，计算出服务方式，为所请求的服务返回合适的入口节点。如果局域服务器不可用，通过DHT的路由机制，也可以将该请求递交给对应区域的某个节点，这个节点为该请求进行服务需求计算

执行。

节点周期性查询自身可提供的服务能力状况，同时，通过带宽和延迟探测方法探测网络中链路状况，向区域管理服务器报告，区域管理服务器也向全局管理服务器报告，由全局管理服务器进行分析整合，将分析结果返回给局域管理服务器，便于其进行服务分配计算时的决策。节点依据一定的算法从系统中所有节点中选择一部分节点作为本节点的优先邻居节点，并且彼此间定时的交换邻居信息。

2）虚拟独立服务单元层

该层通过对物理资源、服务能力进行封装，使其成为最小粒度服务单位，利用虚拟连接层提供的接口进行通信，并对上层开放管理接口。

在这一层，我们提出了一个基于服务单元的属性封装模型，将单节点提供的服务虚拟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。这些基本服务单元只提供单一的功能，如存储、流化、计算能力等。

3）服务协作层

通过虚拟独立服务单元层的处理，所有的服务单元相关能力和属性都封装其中，每个服务单元除了保证DHT正常运行的路由表外，还需维护一个在网络上可以优先协作的服务单元列表，包括这些服务单元所在节点、以及相关链路信息。因此，在服务协作层，我们要将根据不同的服务类型和应用特征，对服务单元进行组织并协作。该层可以根据用户需求，提供具有弹性能力的网络服务，同时屏蔽底层细节。服务协作层的主要功能包括：1) 索引服务：维护服务单元的占用和回收信息，同时提供查询已注册服务单元的功能；2) 分池和调度：将服务单元按特定的用途聚合起来，并提供不同任务之间的调度功能；3) 监视和诊断：监视虚拟硬件，包括功能性损失、入侵检测、过载检测等；4) 能力监控：管理数据、内存和存储，最大化访问性能(例如响应时间、可靠性、冗余、成本等)。

在该层中，根据不同服务单元的特征，充分考虑底层网络特征，进行不同的服务协作同时，使能力结构符合实际硬件分布状况，并兼顾系统容错和负载均衡多个指标，有效提高系统服务性能。

4）服务控制层

提供给终端用户的最后一层，对服务协作层进行屏蔽，用户不需知道服务协作层的实现细节，仅通过这一层定义的服务接口即可提出需求，并获得服务。其功能包括：1) 服务调度：根据用户需求调度所需服务；2) 隐私保护：构建一个隐私框架，来保护终端用户的隐私、数据机密性和完整性；3) 交换和优化支持：在终端用户和服务拥有者之间充当仲裁者的身份，在服务拥有者尽力而为的情况下，保证终端用户的需求。

5.2.1.3　全局认知管理层

在上述的结构中，受限于层次协议结构，各层很少了解其他层的信息，服务单元的调度和控制只能得到次优的结果。因此引入了全局认知管理平台(Global Cognitive Management Plane)，其目的为：承担网络服务管理器的功能，通过不断的观察学习，以达到更好的实施结果；另外，也使虚拟化平台更好的理解和符合用户需求。

1) 全局多目标优化：融合各个层次信息，使系统达到全局优化；2) 信息采集和模型决策：收集物理资源的状态和信息、用户目的和用户模式，通过训练一个感知引擎模型组件，对运行计划进行决策，并根据做出的决定指导资源共享层。3) 感知引擎模型学习和更新：需要有人工智能的方法进行自学习，如通过神经网络、进化规划等；并不断的通过环境数据库、先验行为和他们的反馈来调整自己的行为。

该层可以通过一个单独的服务器进行实现。各层系间通过接口向该层告知当前状态、用户访问情况以及硬件设备情况等信息。认知平台周期性根据所搜集的信息计算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，促进该层优化。

5.2.1.4　服务和应用层

对应用提出的任务调用“统一的服务虚拟、控制和支撑层”的接口并执行。

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

在上一节，我们介绍了服务虚拟化平台的系统层次架构，详细介绍了系统的层次组成以及各层的功能。系统架构对我们认识系统、理解系统等有很大的帮助，但是在进行实际开发工作时，需要更深入地理解系统的编程架构。本节将详细介绍服务虚拟化平台的编程架构。

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，实现功能。下面将详细介绍服务虚拟化平台中涉及的组件。

5.2.2.1 Boto

Boto是一个Python包，它提供了访问AWS（Amazon Web Service）服务的接口，包括Amazon S3、Amazon EC2、Amazon DynamoDB等AWS服务。Boto库通过提供Python API，降低了代码编程复杂性，提高了代码复用率。其特点如下：

1）Python语言

基于Python语言编写，简洁

2）一致的服务接口

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访问体验。

3）简化代码复杂性

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。

在SuperNova的工程开发工作中，我们使用Boto Python API的主要工作包括与AWS EC2建立连接、发送HTTP请求并获取响应。

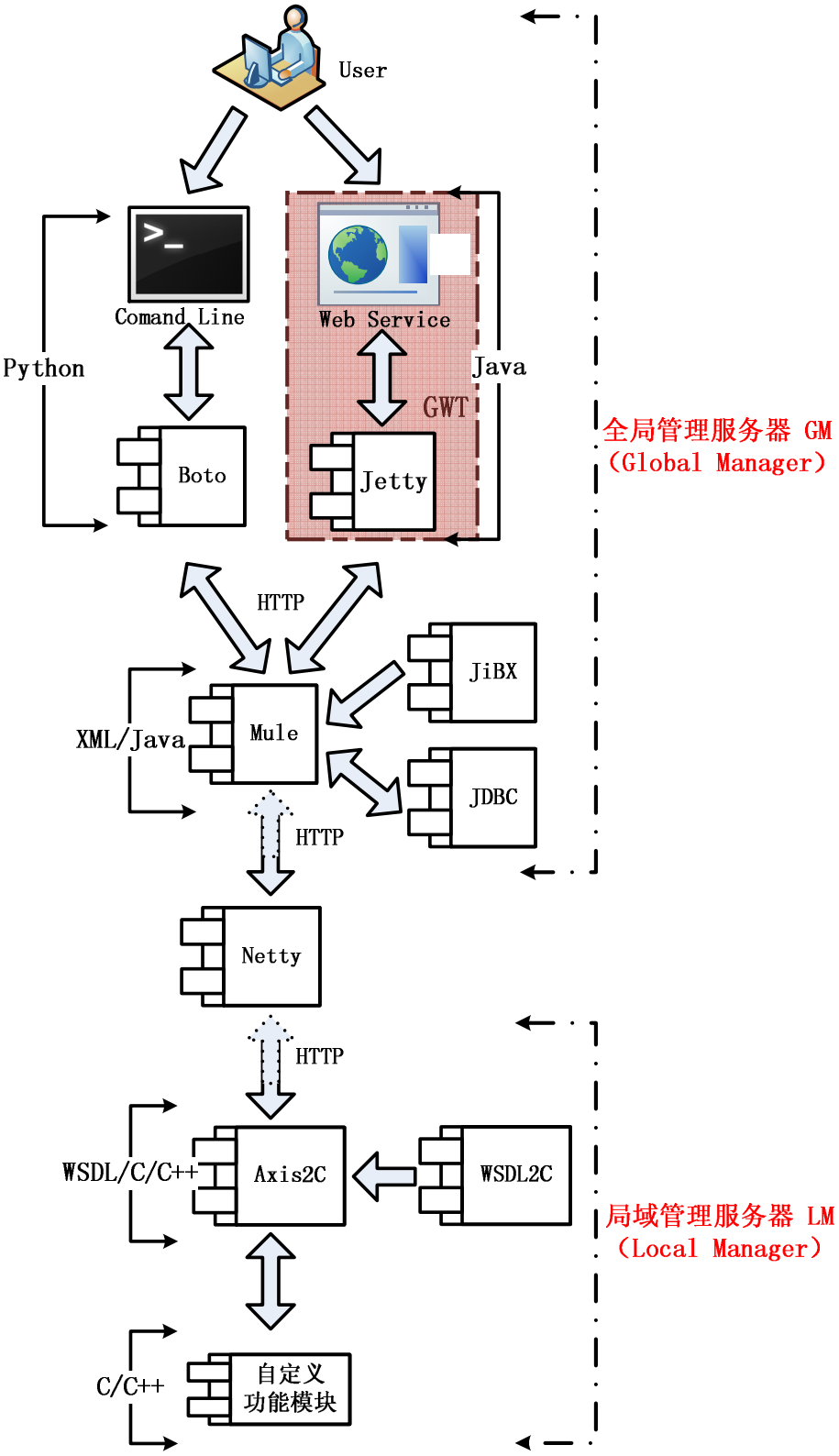

图 5.3: SuperNova服务虚拟化平台编程架构示意图

5.2.2.2 GWT

Ajax，Asynchronous JavaScript and XML，是由XHTML、CSS、JavaScript、XMLHttpRequest、XML等技术组合而成，是当前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以辅助支持Ajax应用的开发和调试，即使开发者对于Ajax技术非常熟悉，其开发和调试过程也很困难。与此相反，Java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，Google Web Toolkit，是Google推出的Ajax应用开发包，它支持开发者使用Java语言开发Ajax应用。GWT解决了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，为Ajax大规模应用创造可能。另一方面，又可以充分发挥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包和java.util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虚拟机的支持了。

5.2.2.3 Jetty

Jetty 是一个开源的servlet容器，它为基于Java的web内容，例如 JSP 和 servlet 提供运行环境。Jetty是使用Java语言编写的，它的API以一组JAR包的形式发布。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或者API来对进行配置，同时，Jetty可作为嵌入式服务器。

5.2.2.4 Mule

Mule是一种Mulesoft推出的开源轻量级的ESB消息框架。Mule的核心概念是SEDA （Staged Event-driven Architecture），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事件驱动的，采用异步通信模式。

5.2.2.5 JiBX

5.2.2.6 JDBC

5.2.2.7 Netty

Netty是由JBOSS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。也就是说，Netty 是一个基于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相当简化和流线化了网络应用的编程开发过程，例如TCP和UDP的socket服务开发。

5.2.2.8 Axis2C

5.2.2.9 WSDL2C

5.3 流量控制模块

5.3.1 流量控制简介

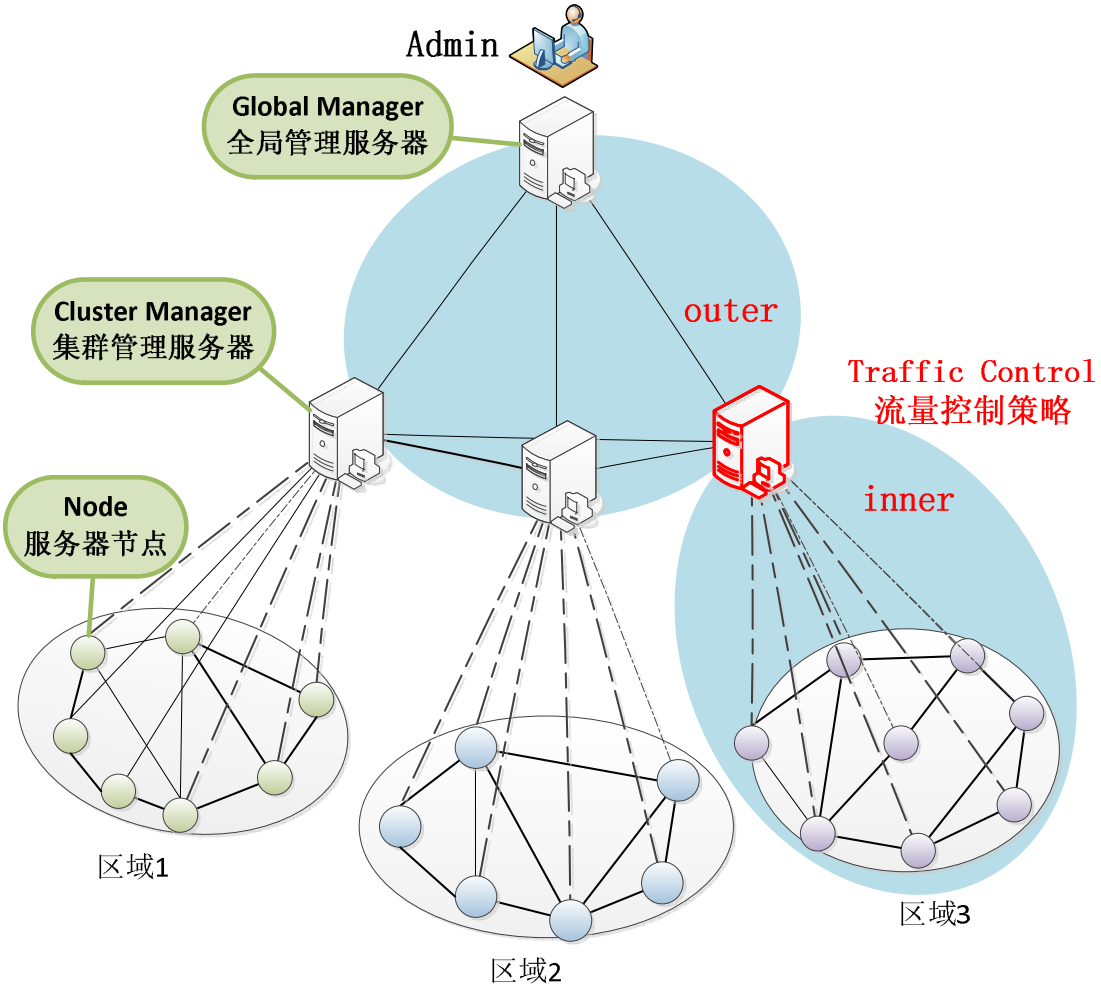

图 5.4: SuperNova中的流量控制模块

如图4.3和图5.4所示， SuperNova 中的流量控制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 和 inner 两大部分。事实上，对于不同用户而言，流量控制模块隔离了他们之间的流量，屏蔽了不同用

户的流量之间的相互干扰。 inner 流量是指用户部署的业务内部之间的相互通信带来的流量，而 outer 流量是外部 Internet 对该业务的访问流量。根据 outer 外部流量中数据包的特征，我们又进一步细分了 5 个孩子Class，以区分 5 个不同优先级的流量并提供用户自定义分类规则。

5.3.2　编程架构

如图5.5所示，SuperNova中流量控制模块编程架构示意图，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文件。首先，用户提出请求“我想启动流量控制模块”。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 EC2 建立连接，并发送请求消息。在 Web 方式下，编写GWT Java代码，包括用户界面 EucalyptusWebInterface.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。

进入到Mule中的消息，根据其消息类型，经由不同的Mule component组件处理。对于流量隔离控制模块，主要的 Mule component 是 TrafficControlEndpoint.java 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来进行处理，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。在局域管理服务器中，通过WSDL文件定义数据类型以及操作类型，如supernova-cc.wsdl，再通过WSDL2C将其转换为C语言定义的代码。Axis2C能够将接受到的SOAP消息转换以调用C程序代码。

5.3.3　功能和实现

下面将详细介绍流量隔离控制的各个功能定义。

5.3.3.1　启动流量控制

启动流量隔离控制模块的方式有两种：

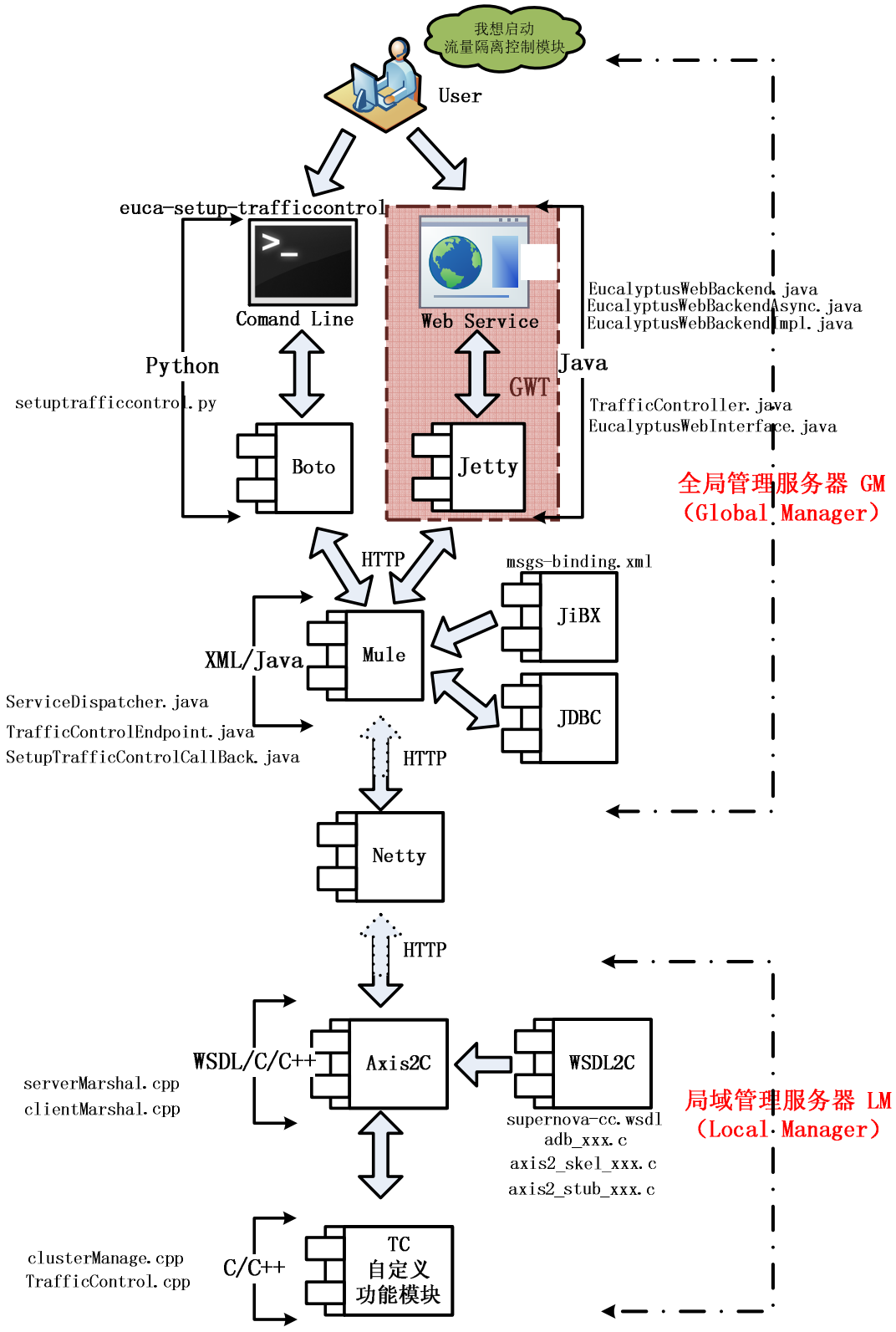

图 5.5: SuperNova中流量控制模块编程架构示意图

1）命令行工具

直接在终端输入命令：euca-setup-trafficcontrol，即可启动流量隔离控制模块。

2）Web 方式

如图5.6所示，点击勾选checkbox，即可启动流量隔离控制模块。

图 5.6: Web方式启动流量控制

不管是采用命令行工具，还是采用Web方式，都可以很容易地启动流量控制模块。事实上，如果系统从来没有启动过流量控制，则流量控制模块会从配置文件中读取默认的流量控制参数，以实现如图4.3所示的流量控制架构。另一方面，如果系统曾经启动过流量控制模块，并重新进行相应的配置和规则修改，这些新的规则都会被保存下来。所以，当用户再次启动流量控制时，会自动恢复到用户的最新规则。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，提高用户体验。

5.3.3.2 查询流量控制规则

```
[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 5.7: 查询流量控制规则

查询流量控制及其规则可通过命令euca-describe-trafficcontrol进行，如图5.7所示。实际上，查询操作是直接在全局管理服务器上开展的。在 SuperNova 流量控制模块编程架构5.5中，查询消息在Mule结束后便终止传递了。由于在SuperNova系统的全局管理服务器中，存在内存数据库表，其中存储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。这种设计是典型的以空间换取时间。

5.3.3.3 配置流量控制规则

配置流量控制规则可通过命令进行 euca-config-trafficcontrol 进行，该命令功能强大，有众多参数可配置选择，如表4.4所示。在配置流量规则的时候，一方面，要修改全局管理服务器的内存数据库表信息以保持一致性；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配置。

5.3.3.4 取消流量控制

取消流量控制可以通过以下两种方式：

1）命令行

直接在终端命令行工具输入命令：euca-stop-trafficcontrol，即可取消并停止流量控制。

2）Web方式

如图5.8所示，取消Web页面中的checkbox勾选，即可取消并停止流量控制。

图 5.8: 取消流量控制

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第三章所述的MSO-VNE算法，进行工程简化和实施。下面将详细描述虚拟网络嵌入模块在SuperNova上的工程实现。

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

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。不仅如此，任何因素的微小变化都会使链路特征发生质的变化。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 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。

工程开发实现与理论算法研究不同，它必须在准确性和可实现性上作出折衷，可以视实际情况牺牲一方而加强另一方。在SuperNova的虚拟网络嵌入模块中，我们牺牲准确性以获得较高的可实现性。在进行实际的VNE匹配计算时，我们只考虑节点的属性特征，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5.9所示，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VNE匹配计算，如果成功映射，则使用开放API将用户的请求部署到SuperNova虚拟化平台中。

SuperNova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5.9和图5.5 流量控制模块的编程架构，虚拟网络嵌入模块的编程实现要简单容易得多，其开发流程也非常简洁明了，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5.9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API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

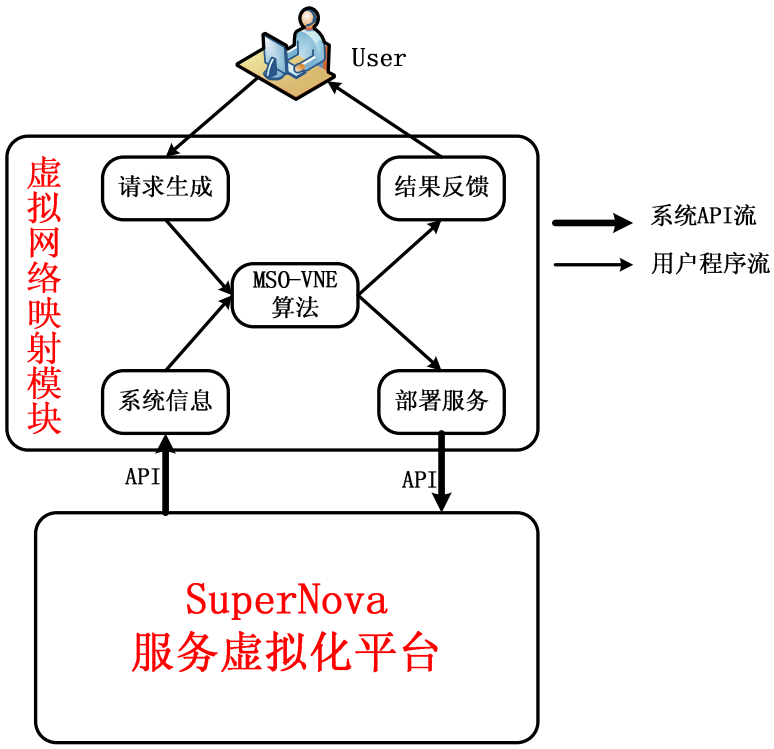

图 5.9: SuperNova中虚拟网络嵌入模块结构

值（max），ram的可用值（free），以及disk，cluster zone信息，如图5.10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 5.10: 由API查询的系统信息

SuperNova的流量控制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，便通过下一个API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

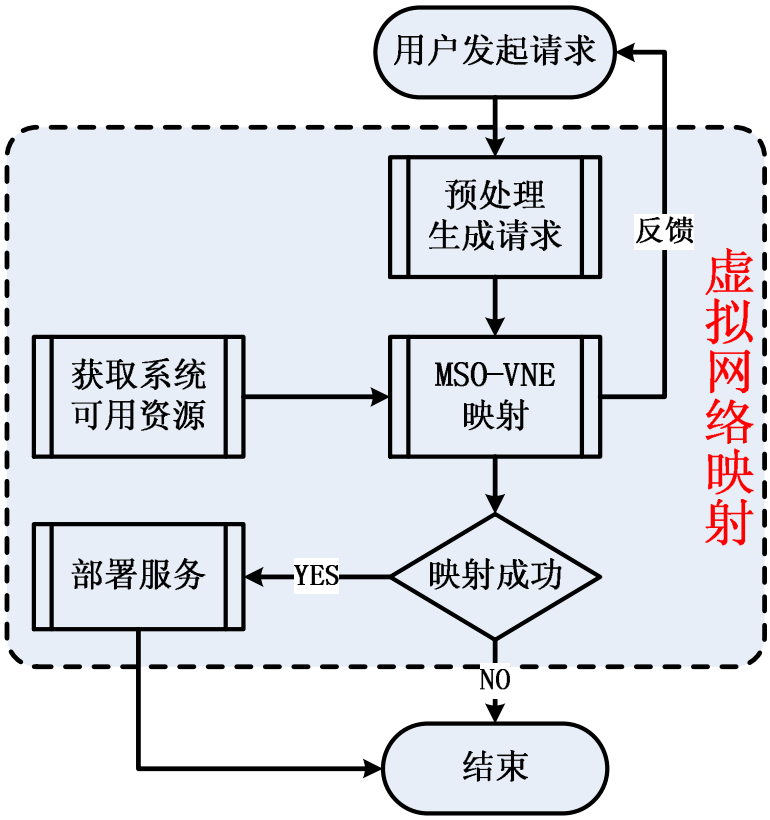

图 5.11: SuperNova中虚拟网络映射模块的实现流程

如图5.11所示，SuperNova虚拟网络映射的关键处理包括：

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VNE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5.12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平台上的服务的管理入口。通过云控制

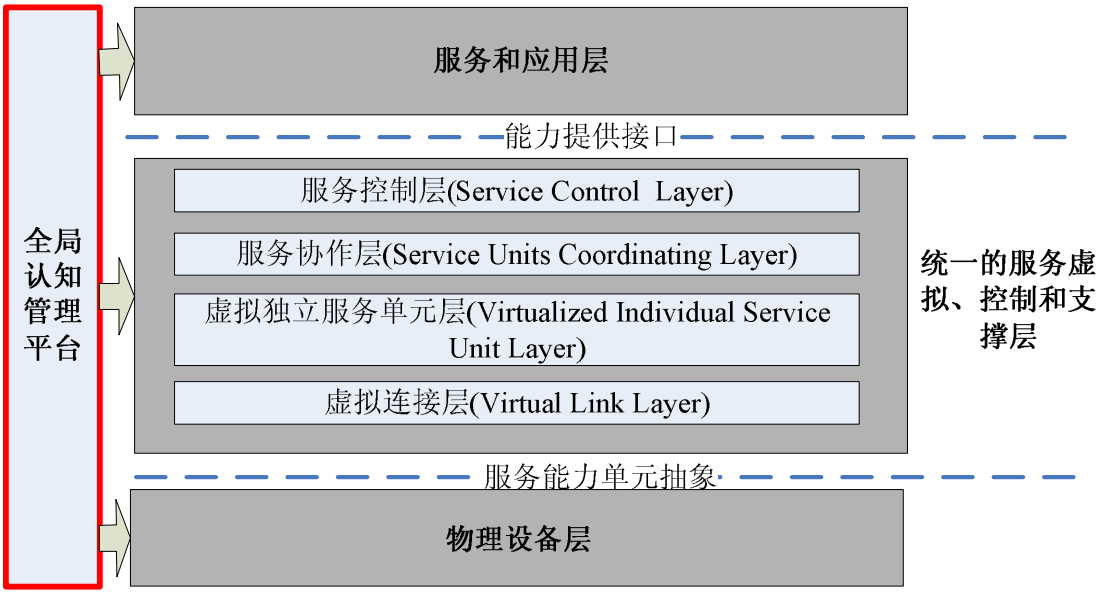

图 5.12: 云控制器在SuperNova中的位置

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

功能完善工作主要是在已有的Euca2ools客户端基础上，对已有的管理操作进行重编码，剔除euca2ools库中对客户端命令的解析并转换为Web Service的过程。这样做的好处有以下两点：

1）简化代码处理流程。

原来的代码处理流程需要经过euca2ools的解析，将通过终端命令行工具输入的命令解析并转换为Web Service的这个过程省去了，直接通过GWT工具库构建Web页面，从而生成Web Service。

2）提高用户体验

对于用户来说，最大的改进就是不在需要面对非常不人性化的终端屏幕，也不用输入繁复易错的命令。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5.13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 。

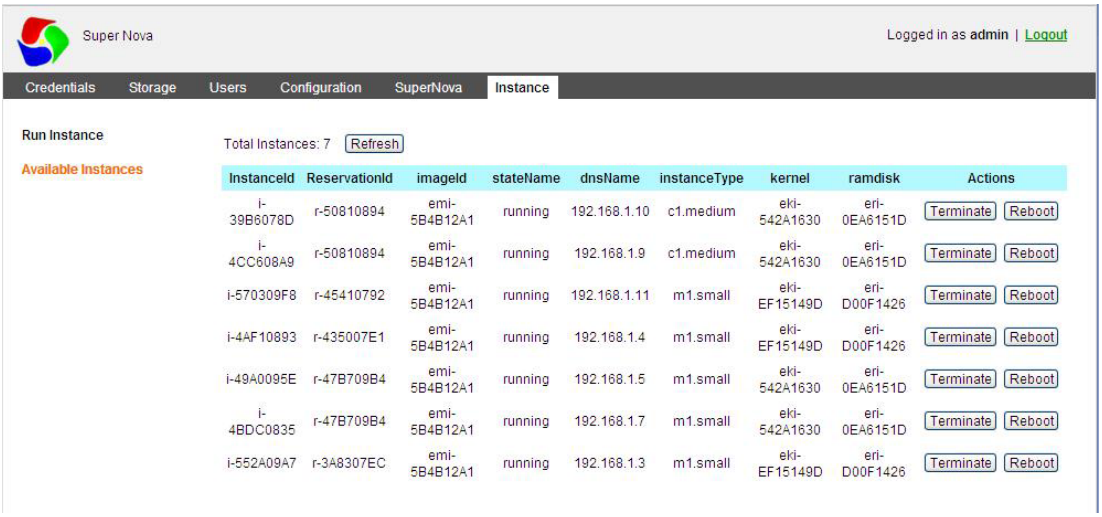

图 5.13: Available Instance

如图5.14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 5.14: 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5.15 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。

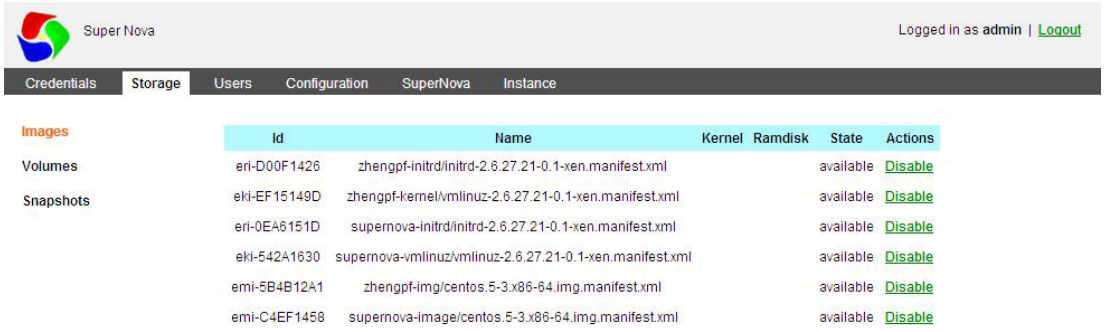

图 5.15: SuperNova服务虚拟化平台的Images

如图5.16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。

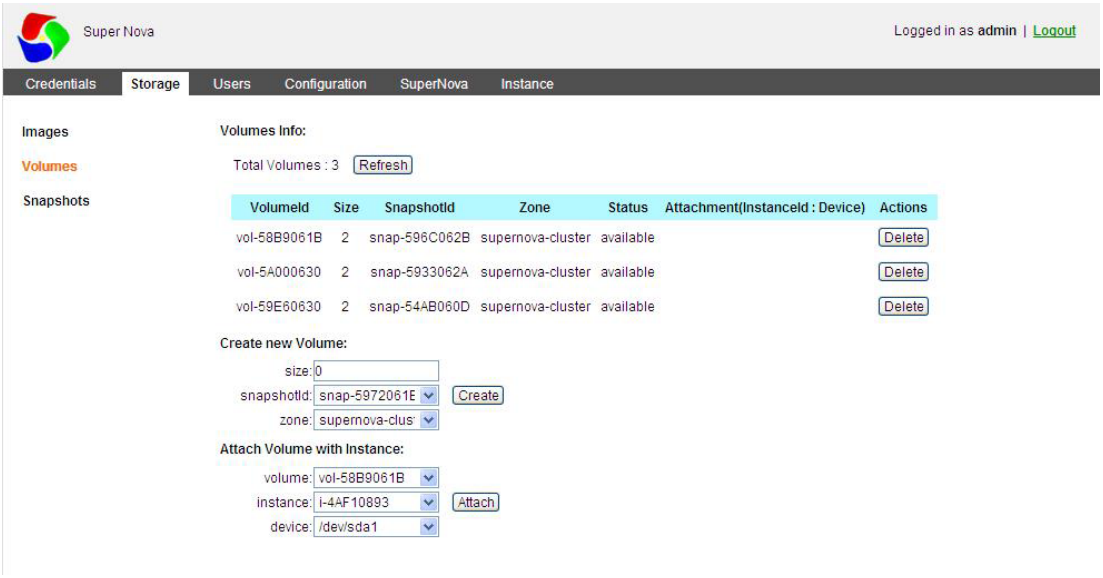

图 5.16: SuperNova服务虚拟化平台的Volumes

如图5.17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。

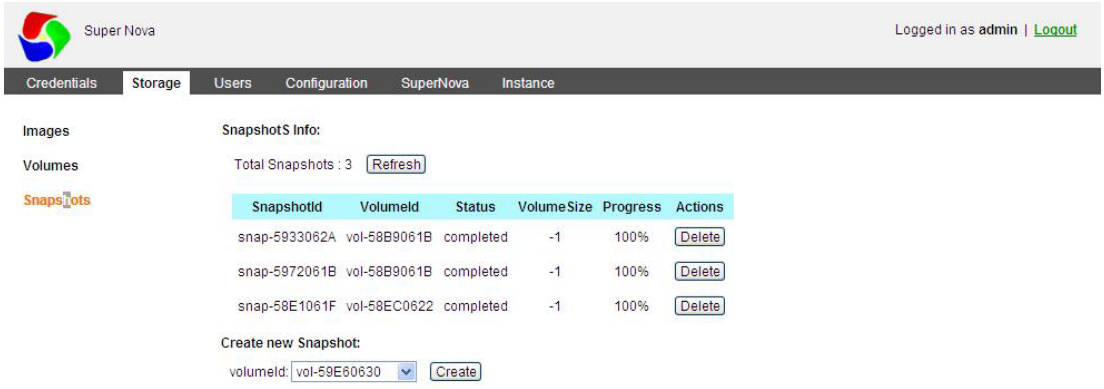

图 5.17: SuperNova服务虚拟化平台的Snapshots

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。

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。

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

在SuperNova服务虚拟化平台中，安全组相关的Web界面如图5.18 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

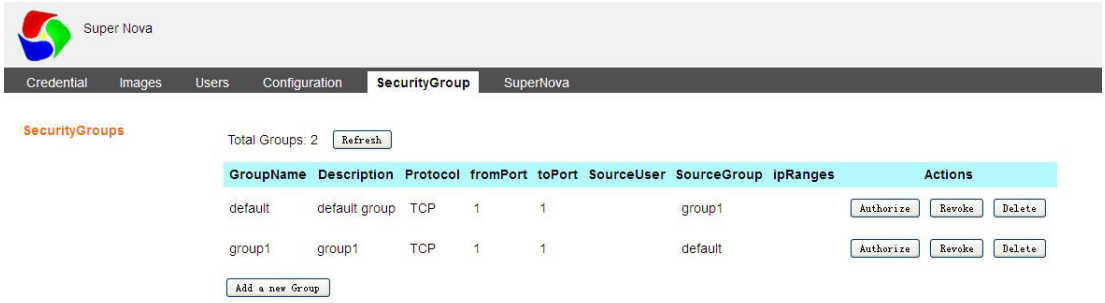

图 5.18: SuperNova中的安全组

第六章　总结和展望

6.1　工作总结

6.2　今后工作展望

参考文献

[1] 邓建松, 彭冉冉, 陈长松. *LaTeX 2ε* 科技排版指南. 科学出版社, 书号: 7-03-009239-2/TP.1516, 北京, 2001.

[2] 王磊. *LaTeX 2ε* 插图指南. 2000.

[3] 张林波. 关于新版 *CCT* 的说明. 2003.

[4] CTEX 翻译小组. *lshort* 中文版 *3.20*. 2003.

[5] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

[6] Donald E. Knuth. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

[7] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

[8] Donald E. Knuth. *TeX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

[9] Donald E. Knuth. *The TeXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

[10] Leslie Lamport. *LaTeX — A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1985.

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”，计算机工程.（已录用，2013年6期）

[2] **Jiao Xue**, Jiali You, Jinlin Wang and Feng Deng, "Nodes Clustering and Dynamic Service Balance Awareness Based Virtual Network Embedding", 2013 IEEE Tencon Conference

[3] Jiali You, **Jiao Xue** and Jinlin Wang, "A Behavior Cluster Based Availability Prediction Approach for Nodes in Distribution Networks", 2013.5 ICASSP

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

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!