

# 指定业务带宽保障

## 目录

1 背景介绍.....	2
2 可行性分析.....	2
3 系统设计方案 .....	3
3.1 转发平面设计 .....	4
3.2 控制平面设计 .....	4
3.3 业务平面设计 .....	5
4 系统实现 .....	5
4.1 Floodlight QoS 模块详述.....	5
4.1.1 REST API 服务模块 .....	6
4.1.2 带宽调度模块 .....	6
4.2 指令配置模块 .....	8
4.2.1 帮助指令 .....	8
4.2.2 状态指令 .....	8
4.2.3 列举指令 .....	9
4.2.4 配置指令 .....	9
4.2.5 退出指令 .....	10
5 系统测试 .....	10
5.1 系统环境搭建 .....	10
5.2 实验拓扑 .....	11
5.3 系统测试 .....	13
5.3.1 网络流量测试工具 .....	13
5.3.2 系统测试步骤 .....	14
6 总结 .....	19

## 1. 背景介绍

网络质量服务是指一个网络能够利用各种技术解决网络延迟和阻塞等问题，向选定的网络通信提供更好服务的一种安全机制。

在传统的互联网体系架构中，网络通常采取尽力而为的传输方式，即把报文转发至目的节点即可。尽管报文头部包含了服务类型(Type of Service, ToS)字段，但在路由器转发时通常被忽略，因此几乎不提供网络质量服务。

因此，在不提供网络质量服务的传统网络中，所有分组处于同等重要的地位。但通常网络中存在着对网络资源的大量竞争，以及网络需要提供给不同用户不同等级的服务。为了解决竞争，必须有可管理的不平等来对待报文转发，在网络的基础设备中定义和部署管理策略，以确保每个节点对单个分组(或者流)的相对重要性做出一致的判断。

本文采用了 SDN 新型网络架构设计了一个系统。该系统在 Floodlight 控制器里实现网络质量服务，通过 OpenFlow 南向接口协议控制底层网络设备的转发行为，通过 REST API 接口与用户交互。该系统能够有效地保障网络中指定业务的带宽需求。

## 2. 可行性分析

(1) Floodlight 提供了对 OpenFlow 协议的支持，从而控制底层 OpenvSwitch 的转发行为。OpenFlow 是 SDN 中第一个也是目前唯一一个定义在控制层和基础设施层之间的标准通信接口，为我们提供了很多重要的消息类型，比如 Packet\_in 消息，使得控制层面的核心模块设计成为可能。

(2) OpenvSwitch 采用 HTB (Hierarchical Token Bucket) 令牌桶机制来保障和限制流量的带宽，如图1所示。这里的“令牌桶”是指 OpenvSwitch 的内部存储池，而“令牌”则是指以给定速率填充令牌桶的数据流量包。交换机在接收每个帧时都将添加一个令牌到令牌桶中，但这个令牌桶底部有一个孔，不断地按你指定作为平均通信速率（单位为 b/s）的速度领出令牌（也就是从桶中删除令牌的意思）。在每次向令牌桶中添加新的令牌包时，交换机都会检查令牌桶中是否有足够容量，如果没有足够的空间，在包上将发生指定监管器中规定的行为（丢弃或标记）。

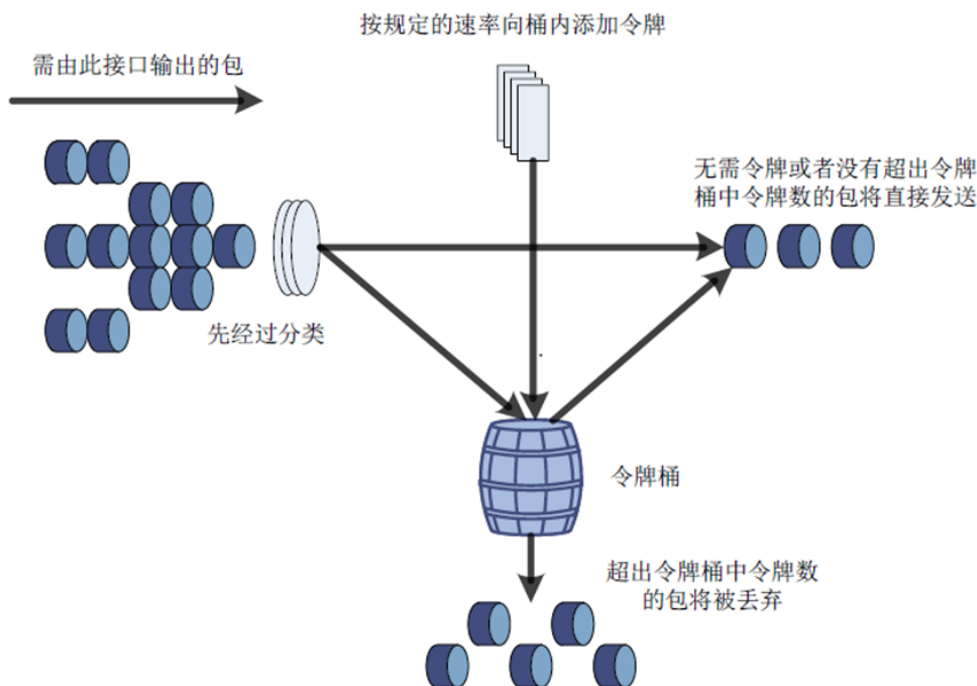


图1：令牌桶机制（HTB）工作原理

### 3.系统设计方案

基于软件定义网络的网络质量服务在实现上主要由指令配置模块（业务层）、带宽调度模块（控制层）和队列管理和调度（转发层）三个部分组成。

用户通过指令配置模块与Floodlight控制器进行交互，可以开启网络质量服务，查询交换机状态，并指定业务带宽保障需求，下发网络质量服务策略（QoS Policy）。带宽调度模块处于核心地位，部署在OpenFlow 控制器上，通过REST API北向接口读取用户指令，并通过查询、配置接口，对底层交换机状态、端口配置、队列配置进行对应的操作。

当用户开启控制平面提供的网络质量服务（QoS），并指定业务带宽保障需求以后，指令配置模块将用户的业务带宽需求转换成相应的网络质量服务策略（Qos Policy），通过北向接口传给带宽调度模块。带宽调度模块根据全局网络拓扑结构获取路由，生成多条流表策略（Flow Policy），下发给网络拓扑中的各个交换机。当新的数据流量进入SDN网络中时，OpenvSwitch将接收的数据分组与下发的流表策略（Flow Policy）进行匹配。若匹配成功，则根据该流表定义的动作（Action）将其放入指定的端口队列，并由作用于输出队列的队列管理和调度机制按事先设定的带宽、缓冲处理控制每个队列。否则，OVS将该数据分组封装成Packet\_in消息通过安全通道上交给Floodlight Qos模块进行进一

步处理。

### 3.1 转发平面设计

队列管理和队列调度是流量调度的两个关键环节，为了在分配有限的网络资源时，保证业务流之间的公平性。队列管理为入队的报文分配缓存，当缓存溢出时对报文进行丢弃，目的是减小因排队造成的端点间的延迟。队列调度负责通过预设的调度算法将队列中等待处理的分组调度到相应的输出链路上。

本系统采用了HTB（Hierarchical Token Bucket）队列调度算法来实现交换机出端口处的队列调度和队列管理机制。

核心策略HTB（Hierarchical Token Bucket）功能组件如下：

- （一）Shaping：仅仅发生在叶子节点，依赖于其他的Queue
- （二）Borrowing：当网络资源空闲的时候，借点过来为我所用
- （三）Rate：设定的发送速度
- （四）Ceil：最大的速度，和rate之间的差是最多能向其他流量借多少

这些组件在OpenFlow交换机上进行实现。用户搭建好Linux开发平台并配置好OVS（OpenvSwitch）以后，在Linux CLI上输入如下命令，即可创建linux-htb队列：

```
sudo ovs-vsctl -- set port s1-eth1 qos=@defaultqos -- --
id=@defaultqos create qos type=linux-htb other-config:max-
rate=10000000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create queue
other-config:min-rate=10000000000 other-config:max-rate=10000000000 --
--id=@q1 create queue other-config:max-rate=200000000 -- --id=@q2
create queue other-config:max-rate=20000000 other-config:min-
rate=20000000
```

(1)

该命令行为OpenvSwitch s1的端口1创建了三个HTB队列机制（q0, q1, q2），每个队列输出链路的速率可以由用户自己定义。

### 3.2 控制平面设计

Floodlight 控制器中添加的网络带宽保障服务主要用于在控制器中实现对网络带宽的调度，从而保障指定业务的网络带宽。主要包含的功能模块有：

- （1）REST API服务模块：是网络带宽保障服务的REST API提供模块，用户可以通过HTTP 请求调用相应的REST API，以获得控制器中网络带宽保障服务提供的相应服务，如启动和关闭网络质量服务，查询、添加或删除网络质量服务策略（QoS Policy）等。

(2) 带宽调度模块：是本系统最主要的模块，主要负责网络中流量的调度，即将用户添加的调度策略转化为相应的流表下发到网络中，根据策略中指定的不同调度方式，添加指定方式的流表处理网络流量，从而限制一般网络业务的流量，保障指定网络业务的网络带宽。在本模块中主要通过设置队列（Set Queue）来实现网络服务质量。队列设置方式是利用OpenFlow 交换机中的端口队列特性，添加支持Push Queue 动作的流表，将网络中不同业务的流量调度到不同的队列中，从而利用队列的最大最小速率来限制和保留指定业务的网络带宽。

### 3.3 业务平面设计

业务平面包含指令配置模块，主要功能是实现用户与系统的交互。具体交互过程如下：

- 管理员通过CLI（command-line interface，命令行界面）输入配置指令或查询指令；
- 指令配置模块接收管理员的指令，进行必要的转换处理，如将用户输入的JSON格式数据转化为相应的Java 数据结构，包括交换机类（Switch）、策略类（Policy）等，然后将转换后的指令通过REST API接口交付Floodlight控制器。
- Floodlight QoS模块读取指令，通过查询、配置接口对底层交换机状态、端口配置、队列配置进行对应的操作，并返回JSON消息格式的数据。如果是配置指令，则返回配置状态结果；如果是查询指令，则返回用户查询的信息。
- 指令配置模块通过REST API接口接收返回的数据，并通过CLI呈现给管理员，以便进行进一步操作。

## 4. 系统实现

本系统使用 Floodlight 0.9 开源控制器，运行于 Windows7 操作系统下；并在 VMware Workstation 中安装了 Ubuntu 操作系统，在 Ubuntu 环境下安装了 Mininet 虚拟机，构建起 SDN 网络架构。

### 4.1 Floodlight QoS 模块详述

整个 QoS 模块由两个具体的部分组成，分别为 REST API 服务类以及带宽调度类。这两种类实现相互独立的功能，通过暴露的接口耦合，因此这里同样称之为“模块”。下面分别做具体介绍。

### 4.1.1 REST API 服务模块

REST API 服务模块主要通过北向接口提供 URL 资源，以实现控制层与业务层的交互。QoS 模块主要提供了两种 URL 资源，分别为/wm/qos/tool/{op}/json 和/wm/qos/policy/json。表 1 是 REST API 服务模块中的类及其功能描述：

类名	定义	功能描述
IQoSService.class	public interface IQoSService extends IFloodlightService	向 Floodlight 控制器的模块加载器注册网络质量服务
QoSWebRoutable.class	public class QoSWebRoutable implements RestletRoutable	将处理各种 HTTP 类型消息的类绑定到合适的 URL
QoSResource.class	public class QoSResource extends ServerResource	提供 URL 供用户查询网络质量服务状态、开启或关闭网络质量服务（QoS）
QoSPoliciesResource.class	public class QoSPoliciesResource extends ServerResource	提供 URL 供用户查询、下发和删除网络质量服务策略（QoS Policy）

表 1：REST API 服务模块类

### 4.1.2 带宽调度模块

该模块是 Floodlight QoS 核心模块，主要负责网络中流量的调度。下面进行详细介绍。

（1） QoSPolicy.class：定义了网络质量服务策略（QoS Policy）的消息格式，主要包括策略编号（policyid）、策略名称（name），IP 包头域、交换机通用唯一标识符 dpid （datapath id）以及本条策略的优先级（priority）等等。具体定义如下所示：

```

public long policyid;
public String name;
public short ethtype;
public byte protocol;
public short ingressport;
public int ipdst;
public int ipsrc;
public byte tos;
public short vlanid;
public String ethsrc;
public String ethdst;
public short tcpudpsrcport;
public short tcpudpdstport;

public String sw;
public short queue;
public short enqueueport;
public String service;
public short priority = 0;

```

图 2: QoS Policy 消息格式

(2) QoS.class: 带宽调度模块的核心类。该类注册并实现了网络质量服务 (QoS), 在 Floodlight Controller 启动时作为独立的模块被加载, 在运行过程中监听 6633 端口, 接收和处理 Packet\_in 消息, 与 OpenvSwitch 进行交互。该类中定义了很多重要的方法和接口, 下面对这个类做具体介绍。

方法原型	功能描述
public Collection<Class<? extends IFloodlightService>> getModuleServices()	向模块加载器注册 QoS 服务, 并通过 getServiceImpls()方法提供实现该服务的实例。
protected ArrayList<QoSPolicy> readPoliciesFromStorage()	从存储体当中读取所有已下发的流表策略放入 ArrayList 容器中并返回该 ArrayList 对象
public net.floodlightcontroller.core.IListener.Command receive()	监听 6633 端口并处理 OVS 上交的 Packet_in 消息

public synchronized void addPolicy(QoSPolicy policy)	下发生成的流表策略（Flow Policy），并将其添加到存储体当中
public synchronized void deletePolicy(QoSPolicy policy)	将已下发的流表策略从 OpenvSwitch 中删除，并从存储体中移除记录
public OFFlowMod policyToFlowMod(QoSPolicy policy)	将 QoSPolicy 对象转换成 OFFlowMod 格式消息

表 2: QoS 类主要方法

## 4.2 指令配置模块

下面对指令配置模块为管理员提供的主要输入及其功能进行详细分析。

### 4.2.1 帮助指令

- 语法: -- help
- 功能: 显示帮助信息。提管理员各种输入指令的格式与功能，和可进一步扩展功能的指令。
- 使用范例:

```
wufei@wufei-virtual-machine:~$ sudo ./qosmanager2.py --help
[sudo] password for wufei:
usage: qosmanager2.py [-h] [-p P] [-c C] [-e] [-d] [-s] [-A] [-D] [-M] [-L]
                    [-t {policy,service,policies,services}] [-O OBJ]

Floodlight Quality of Service Manager

optional arguments:
  -h, --help            show this help message and exit
  -p P, --port P
  -c C, --controller C
  -e, --enable
  -d, --disable
  -s, --status
  -A, --add
  -D, --delete
  -M, --modify
  -L, --list
  -t {policy,service,policies,services}, --type {policy,service,policies,service}

Ubuntu Software Center
```

### 4.2.2 状态指令

- 语法: -- status [ enable | disable ]
- 功能: 查看、打开或关闭软件定义网络（SDN）网络质量服务（QoS）功能。当设置为enable 时，开启网络质量服务（QoS），系统读取流规则库





➤ 使用范例:

```
wufei@wufei-virtual-machine:~$ sudo ./qospath2.py --add --name 7878Mbps01-02 -S 10.0.0.1 -D 10.0.0.2 -J '{"queue": "1"}'
add
src good
dest good
Trying to create a circuit from host 10.0.0.1 to host 10.0.0.2...
/home/wufei
Necessary tools confirmed.. circuitpusher.py , qosmanager2.py
create circuit!!!
./circuitpusher.py --controller=127.0.0.1:8080 --type ip --src 10.0.0.1 --dst 10.0.0.2 --add --name 7878Mbps01-02
Process 9666 started to create circuit
Namespace(action='add', circuitName='7878Mbps01-02', controllerRestIp='127.0.0.1:8080', dstAddress='10.0.0.2', srcAddress='10.0.0.1', type='ip')
192.168.150.1:8080
Amazon //192.168.150.1:8080/wm/device/
curl -s http://192.168.150.1:8080/wm/device/
```

## 4.2.5 退出指令

- 语法: exit
- 功能: 退出程序和当前指令配置界面。

## 5. 系统测试

### 5.1 系统环境搭建

如图5.1所示, 本方案选择的SDN控制器是免费开源的floodlight, 将其运行于win7系统下eclipse中; 虚拟交换机使用开源的openvSwitch; 虚拟网络的创建使用安装于Ubuntu12.04下的mininet。

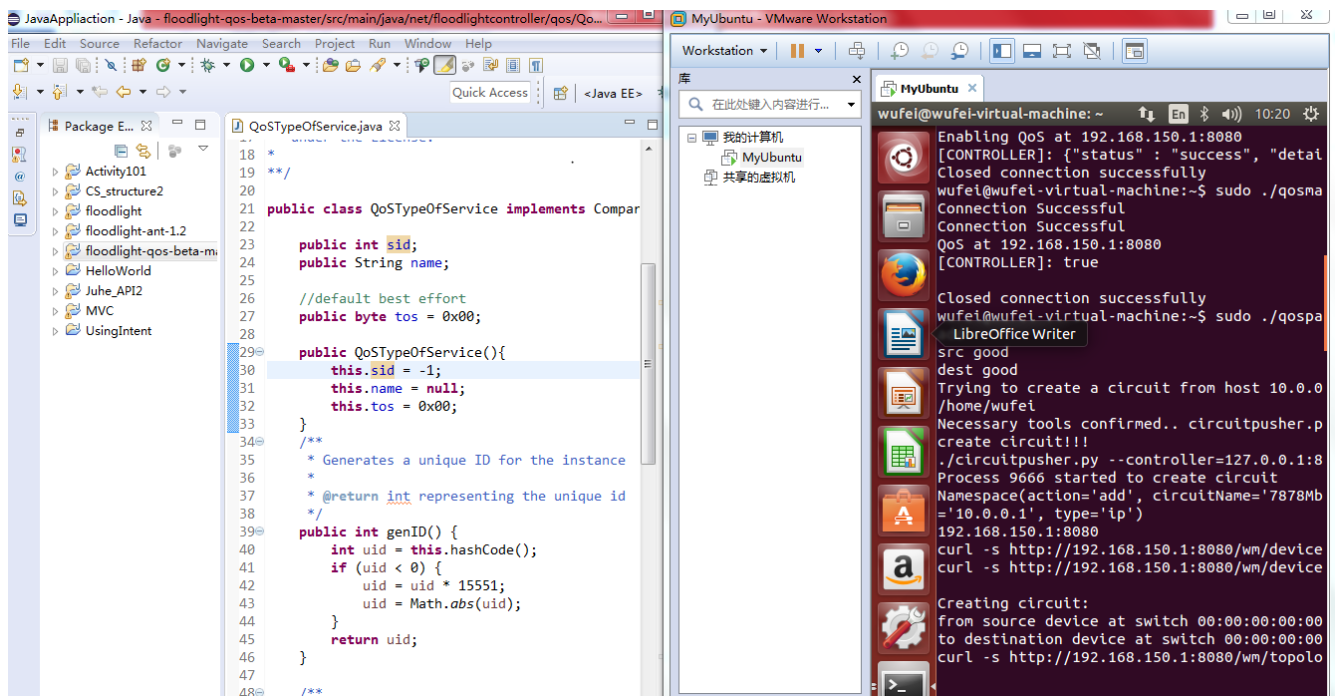


图5.1: 系统环境搭建

## 5.2 实验拓扑

用户有两种常用的方式搭建网络拓扑结构。一种是通过 Mininet 的 custom 文件夹下的 Python 文件自定义网络拓扑结构，如图 5.2 所示：

```
#!/usr/bin/python
from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.net import Mininet
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun
from mininet.node import OVSSwitch, OVSController, Controller, Remote
from mininet.topo import Topo

class MyTopo( Topo ):
    def __init__( self ):
        Topo.__init__( self )
        h1 = self.addHost( 'h1' , ip="10.0.0.1" , mac="00:00:00:00:"
        h2 = self.addHost( 'h2' , ip="10.0.0.2" , mac="00:00:00:00:"
        h3 = self.addHost( 'h3' , ip="10.0.0.3" , mac="00:00:00:00:"
        s1 = self.addSwitch( 's1' )
        self.addLink( s1, h1 )
        self.addLink( s1, h2 )
        self.addLink( s1, h3 )
        mytopo': ( lambda: MyTopo() ) }
```

图 5.2：自定义网络拓扑

一种是使用 Mininet 开发工具提供的标签选项快速搭建网络拓扑：

```
sudo mn --topo tree,3 --switch ovsk --controller=remote,ip=192.168.150.1,port=6633 --mac
```

图 5.3：快速搭建网络拓扑

本系统采用第二种方法搭建网络拓扑结构。搭建的网络拓扑结构如图 5.4 所示：

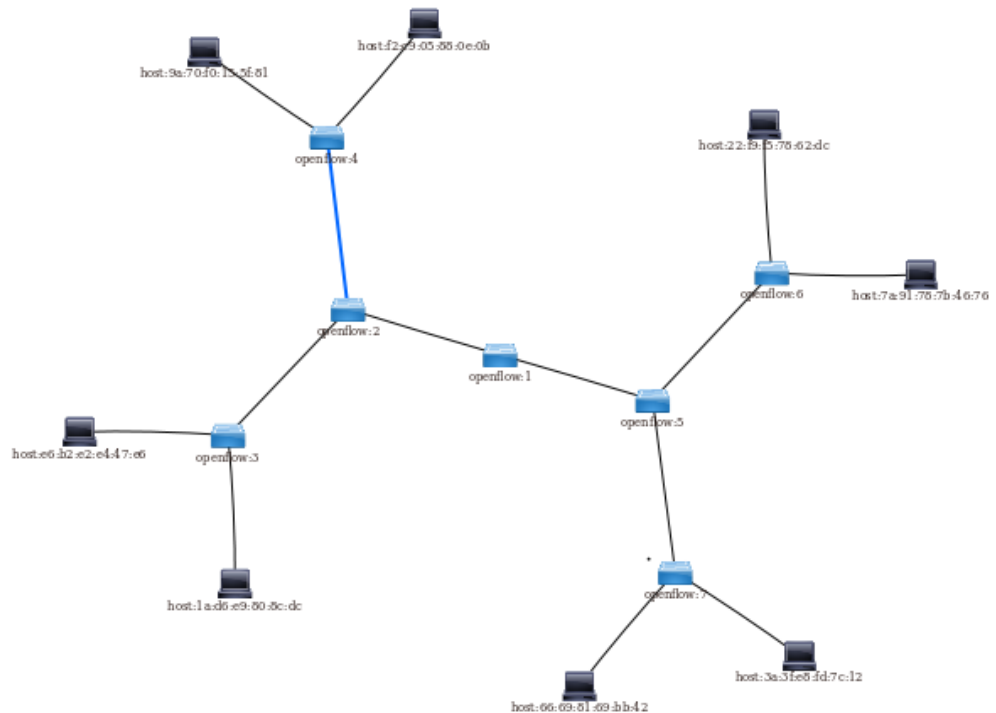


图5.4：系统搭建的网络拓扑结构

该网络拓扑呈二叉树状结构，交换机呈树杈状分散开来，最下面一层交换机挂两个主机。树的深度为3层，共7个交换机（OpenvSwitch），8个主机（Host）。具体的链路结构如图5.5所示：

```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo: s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo: s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo: s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo: s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo: s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
      :h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo: s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
c0
```

图5.5：网络链路结构

此时，打开网址<http://localhost:8080/>，查看本机8080端口，如图5.6所示，可以看到，Floodlight Controller提供两种工具，Quality of Service 和 Firewall，且两种工具均处于” false ” 状态（disable）。

**(2) Network Tools**

Click on the tool for more information, and to enable it

Tool	Version	Base URI	Is Enabled
<a href="#">Quality of Service</a>	1	/wm/qos/	false
<a href="#">Firewall</a>	1	/wm/firewall/	false

图5.6: 通过REST API查看控制器状态

## 5.3 系统测试

### 5.3.1 网络流量测试工具

本次实验使用Iperf作为网络流量测试工具。

Iperf 是一个网络性能测试工具，可以测试最大TCP和UDP带宽性能。该工具有多种参数和UDP特性，根据需要可以调整。Iperf还可以报告带宽，延迟抖动和数据包丢失。因此Iperf可以用来测试一些网络设备如路由器，防火墙，交换机等的性能。

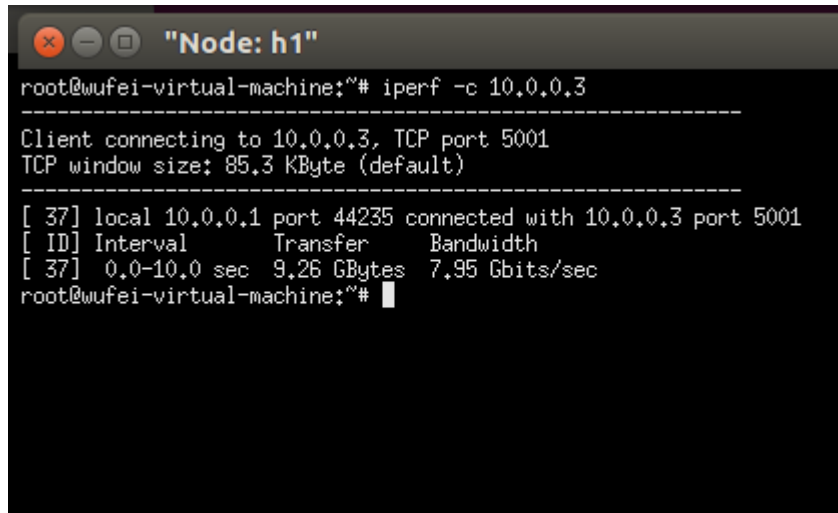
TCP带宽测试命令如下：

(1) 服务器端启动并监听TCP端口：

```
"Node: h3"
root@wufei-virtual-machine:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 kByte (default)
-----
```

图5.7: 服务器监听TCP端口

(2) 客户端启动并连接指定的服务器端：

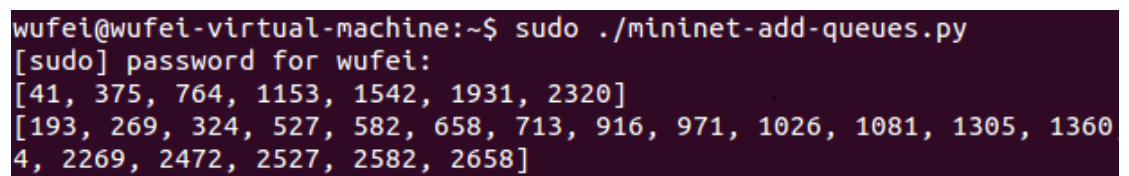


```
root@wufei-virtual-machine:~# iperf -c 10.0.0.3
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.1 port 44235 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 37] 0.0-10.0 sec  9.26 GBytes 7.95 Gbits/sec
root@wufei-virtual-machine:~#
```

图5.8：客户端连接服务器并测速

### 5.3.2 系统测试步骤

(1) 执行mininet--add-queues.py文件，为网络拓扑中的OpenvSwitch设置HTB (Hierarchical Token Bucket) 队列，如图5.9所示：

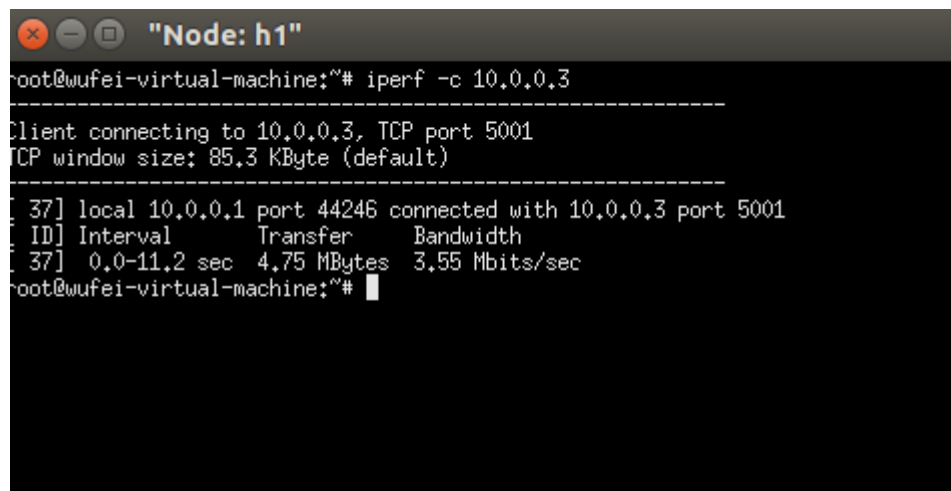


```
wufei@wufei-virtual-machine:~$ sudo ./mininet-add-queues.py
[sudo] password for wufei:
[41, 375, 764, 1153, 1542, 1931, 2320]
[193, 269, 324, 527, 582, 658, 713, 916, 971, 1026, 1081, 1305, 1360]
[4, 2269, 2472, 2527, 2582, 2658]
```

图5.9：执行队列配置文件

该Python文件为每个OpenvSwitch的链路端口设置了三个linux-htb 队列，分别为q0, q1, q2，带宽分别限定为2Mbps, 10Mbps和50Mbps。

(2) 用户在不指定业务带宽保障需求的情况下，使用iperf进行TCP网络带宽测速，其中host h1作为客户端，host h3作为服务器端。测试结果如图5.10所示：



```
root@wufei-virtual-machine:~# iperf -c 10.0.0.3
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.1 port 44246 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 37] 0.0-11.2 sec  4.75 MBytes 3.55 Mbits/sec
root@wufei-virtual-machine:~#
```

图5.10: TCP测速

通过Linux CLI输入 `xterm s3`, 为OpenvSwitch s3开启终端, 查看队列状况。如图5.11所示, 可以看到, 此时的网络流量皆通过默认队列queue0 传输:

```
oot@wufei-virtual-machine:~# ovs-ofctl queue-stats s3
FPST_QUEUE reply (xid=0x2): 9 queues
port 3 queue 0: bytes=4757157, pkts=3646, errors=0, duration=?
port 3 queue 1: bytes=0, pkts=0, errors=0, duration=?
port 3 queue 2: bytes=0, pkts=0, errors=0, duration=?
port 1 queue 0: bytes=736620, pkts=4878, errors=0, duration=?
port 1 queue 1: bytes=0, pkts=0, errors=0, duration=?
port 1 queue 2: bytes=0, pkts=0, errors=0, duration=?
port 2 queue 0: bytes=635392, pkts=3344, errors=0, duration=?
port 2 queue 1: bytes=0, pkts=0, errors=0, duration=?
port 2 queue 2: bytes=0, pkts=0, errors=0, duration=?
```

图5.11: 查看OVS队列状态

(3) 调用 `qosmanager2.py` 文件提供的命令行指令 (CLI) 开启网络质量服务 (QoS), 如图5.12所示:

```
wufei@wufei-virtual-machine:~$ sudo ./qosmanager2.py -e
Connection Successful
Connection Successful
Enabling QoS at 192.168.150.1:8080
[CONTROLLER]: {"status": "success", "details": "QoS Enabled"}
Closed connection successfully
```

图5.12: 开启网络质量服务

此时, 打开浏览器网址 `http://localhost:8080/`, 查看本机8080端口, 如图5.13所示, Quality of Service已经开启 (enable), 默认提供的服务是尽最大努力交付 (Best Effort), ToS (Type of Services) 字段标记为0。Policies一栏为空, 表明目前用户没有指定业务带宽保障需求, 因此网络质量服务策略为空。

Quality of Service Enabled: true

/wm/qos/

A Quality of Service Module that allows Quality of Service state to be pushed into the network through REST API's. The QoS follows queuing QoS for max-min rate limits and a DiffServ model Type of Service model. Allows for users to define a QoS rule to be a queuing policy or a type of service/ diffserv policy. Configuration done through REST API

Enable

Disable

Manage QoS

Services

Id	Service	ToS
1935906109	Best Effort	0

Policies

Id	Policy	Match Fields	Switches	Enqueue #/s	Service	Priority
----	--------	--------------	----------	-------------	---------	----------

Add QoS Policies

Remove QoS Policies

图5.13: 通过8080端口查看控制器



调用 qospath.py 文件提供的命令行指令（CLI）指定业务带宽保障需求，生成网络质量服务策略（QoS Policy）：

```
wufei@wufei-virtual-machine:~$ sudo ./qospath2.py --add --name 10Mbps01-02 -S 10.0.0.1 -D 10.0.0.3 -J '{"queue": "1"}'
add
src good
dest good
Trying to create a circuit from host 10.0.0.1 to host 10.0.0.3...
/home/wufei
Necessary tools confirmed.. circuitpusher.py , qosmanager2.py
create circuit!!!
./circuitpusher.py --controller=127.0.0.1:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name 10Mbps01-02
Process 11381 started to create circuit
Namespace(action='add', circuitName='10Mbps01-02', controllerRestIp='127.0.0.1:8080', dstAddress='10.0.0.3', srcAddress=
10.0.0.1', type='ip')
Amazon 1:8080
curl -s http://192.168.150.1:8080/wm/device/
curl -s http://192.168.150.1:8080/wm/device/
```

图5.14：指定业务带宽保障需求

如上图5.14所示，该命令行创建了一条实际的QoS Policy策略，名称为 10Mbps01-02 。该网络质量服务策略（QoS Policy）指定了 host h1 到 host h3之间的业务带宽保障需求。此时，打开qos-state.json文件，可以看到已经写入的多条流表策略（Flow Policy），其中每条流表策略的名称结构如下：

*Name of QoS Policy.DPID of OpenvSwitch.Serial Number* (2)

打开浏览器，输入网址http://localhost:8080/，查看本机8080端口，如图 5.15所示，可以看到Policies一栏显示的已生成的多条流表策略（Flow Policy），表明该QoS Policy已成功创建：

Policies						
Id	Policy	Match Fields	Switches	Enqueue #:#	Service	Priority
2048011597	10Mbps01-02.00:00:00:00:00:00:04.2	ethernet-type=-1, protocol=-1, ingress-port=1, ip-dest=167772161, ip-src=167772163, tos-bits=-1, vlan-id=-1, ethsrc=null, ethdst=null, tcpdstport=-1, tcpsrcport=-1,	00:00:00:00:00:00:04	Enqueue 1:3	null	32767
1664604442	10Mbps01-02.00:00:00:00:00:00:04.1	ethernet-type=-1, protocol=-1, ingress-port=3, ip-dest=167772163, ip-src=167772161, tos-bits=-1, vlan-id=-1, ethsrc=null, ethdst=null, tcpdstport=-1, tcpsrcport=-1,	00:00:00:00:00:00:04	Enqueue 1:1	null	32767
400858930	10Mbps01-02.00:00:00:00:00:00:02.2	ethernet-type=-1, protocol=-1, ingress-port=2, ip-dest=167772161, ip-src=167772163, tos-bits=-1, vlan-id=-1, ethsrc=null,	00:00:00:00:00:00:02	Enqueue 1:1	null	32767

图5.15：通过8080端口查看生成的流表策略

(4) 通过Linux CLI输入 xterm h1,h3，为host h1,host h3开启终端，使用 iperf工具进行TCP网络带宽测速：



```
"Node: h1"
root@wufei-virtual-machine:~# iperf -c 10.0.0.3
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.1 port 35039 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 37] 0.0-10.2 sec  13.6 MBytes 11.2 Mbits/sec
root@wufei-virtual-machine:~#
```

```
"Node: h3"
root@wufei-virtual-machine:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 38] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 35039
[ ID] Interval      Transfer    Bandwidth
[ 38] 0.0-12.1 sec  13.6 MBytes 9.45 Mbits/sec

```

图5.16: TCP测速

如图5.16所示，对客户机 client h1与服务器 server h3之间的数据流量进行网络测试，平均带宽达到了10Mbps，与用户指定的业务带宽保障需求（QoS Policy）一致。

同时，为消除实验结果的误差，通过Linux CLI开启多个主机终端，进行多组TCP网络测速：

① Client: host h2; Server: host h5

```
"Node: h2"
root@wufei-virtual-machine:~# iperf -c 10.0.0.5
-----
Client connecting to 10.0.0.5, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.2 port 45780 connected with 10.0.0.5 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 37] 0.0-10.5 sec  4.38 MBytes 3.48 Mbits/sec
root@wufei-virtual-machine:~#

"Node: h5"
root@wufei-virtual-machine:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 38] local 10.0.0.5 port 5001 connected with 10.0.0.2 port 45780
[ ID] Interval      Transfer    Bandwidth
[ 38] 0.0-19.3 sec  4.38 MBytes 1.90 Mbits/sec

```

② Client: host h7; Server: host h1

```
"Node: h7"
root@wufei-virtual-machine:~# iperf -c 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.7 port 46371 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 37] 0.0-10.6 sec  4.25 MBytes  3.35 Mbits/sec
root@wufei-virtual-machine:~#

"Node: h1"
root@wufei-virtual-machine:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ ID] local 10.0.0.1 port 5001 connected with 10.0.0.7 port 46371
[ ID] Interval      Transfer      Bandwidth
[ ID] 0.0-10.8 sec  4.25 MBytes  1.89 Mbits/sec
```

③ Client: host h3; Server: host h6

```
"Node: h6"
root@wufei-virtual-machine:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 38] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 50480
[ ID] Interval      Transfer      Bandwidth
[ 38] 0.0-20.3 sec  4.62 MBytes  1.91 Mbits/sec

"Node: h3"
root@wufei-virtual-machine:~# iperf -c 10.0.0.6
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.3 port 50480 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 37] 0.0-10.9 sec  4.62 MBytes  3.55 Mbits/sec
root@wufei-virtual-machine:~#
```

由以上多组测试结果可以看出，未指定业务带宽保障需求时，主机之间TCP通信平均带宽为3Mbps。而指定了业务带宽保障需求的主机之间通信流量达到10Mbps。网络质量服务得到了有效的实现，同时也证明了QoS系统对底层交换设备流量控制功能的正确性。

(5) 调用 qospath.py 文件提供的命令行指令（CLI）删除名称为10Mbps01-02的网络质量服务策略（QoS Policy）：

```
wufei@wufei-virtual-machine:~$ sudo ./qospath2.py --delete --name 10Mbps01-02
delete
Trying to delete QoSPath 10Mbps01-02
Deleting circuit
Namespace(action='delete', circuitName='10Mbps01-02', controllerRestIp='127.0.0.1', type='ip')
192.168.150.1:8080
{'outPort': 3, 'Dpid': '00:00:00:00:00:00:00:03', 'name': '10Mbps01-02', 'type': 'ip', 'v': 30:42 2017'}
{'outPort': 2, 'Dpid': '00:00:00:00:00:00:00:02', 'name': '10Mbps01-02', 'type': 'ip', 'v': 30:42 2017'}
{'outPort': 1, 'Dpid': '00:00:00:00:00:00:00:04', 'name': '10Mbps01-02', 'type': 'ip', 'v': 30:42 2017'}
```

图5.17：删除网络质量服务策略

打开浏览器，输入网址http://localhost:8080/，查看本机8080端口，可以看到Policies一栏再次为空，表明名称为10Mbps01-02的网络质量服务策略（QoS Policy）对应的多条流表项（Flow Entry）已经删除。

此时，使用Iperf对host h1 与host h3之间进行TCP网络带宽测速，可以看到，网络通信流量恢复到3Mbps。

```
"Node: h1"
root@wufei-virtual-machine:~# iperf -c 10.0.0.3
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.1 port 52238 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 37] 0.0-10.6 sec  4.38 MBytes 3.47 Mbits/sec
root@wufei-virtual-machine:~#
```

图5.18: TCP测速

(6) 用户可以同时指定多个主机之间的业务带宽保障需求，从而下发多条网络质量服务策略。另外，如果用户有更高的业务带宽需求，可以指定队列queue3作为流量通道。例如，在客户端 host h2 与服务器端 host h5之间指定更高的业务带宽保障需求，如图5.19所示：

```
wufei@wufei-virtual-machine:~$ sudo ./qospath2.py --add --name 50Mbps02-05 -S 10.0.0.2 -D 10.0.0.5 -J '{"queue": "2"}'
[sudo] password for wufei:
add
src good
dest good
Trying to create a circuit from host 10.0.0.2 to host 10.0.0.5...
/home/wufei
Necessary tools confirmed.. circuitpusher.py , qosmanager2.py
create circuit!!!
./circuitpusher.py --controller=127.0.0.1:8080 --type ip --src 10.0.0.2 --dst 10.0.0.5 --add --name 50Mbps02-05
Process 31334 started to create circuit
Namespace(actions='add', circuitName='50Mbps02-05', controllerRestIp='127.0.0.1:8080', dstAddress='10.0.0.5', srcAddress='10.0.0.2', type='ip')
```

图5.19: 指定更高的业务带宽保障需求

打开浏览器，输入网址<http://localhost:8080/tools>，可以看到生成的多条流表策略。使用Iperf进行TCP网络带宽测速：

```
"Node: h5"
root@wufei-virtual-machine:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 38] local 10.0.0.5 port 5001 connected with 10.0.0.2 port 34759
[ ID] Interval      Transfer    Bandwidth
[ 38] 0.0-12.4 sec  64.6 MBytes 43.7 Mbits/sec

"Node: h2"
root@wufei-virtual-machine:~# iperf -c 10.0.0.5
-----
Client connecting to 10.0.0.5, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 37] local 10.0.0.2 port 34759 connected with 10.0.0.5 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 37] 0.0-10.5 sec  64.6 MBytes 51.7 Mbits/sec
root@wufei-virtual-machine:~#
```

图5.20: TCP测速

可以看到，host h2和host h5之间的通信流量达到50Mbps，与用户指定的网络质量服务策略（QoS Policy）一致，从而实现了更高的业务带宽保障需求。

## 6. 总结

本文通过分析 QoS 技术要求，结合 SDN 网络架构控制和转发分离以及易于编程的特点，设计并实现了一个适用于软件定义网络架构的网络质量服务系

统。该系统使用 OpenVSwitch 作为底层网络设备，在 Floodlight 控制器上开发 Qos 模块，实现了基于 HTB 队列管理和队列调度机制算法的流量控制机制。本系统控制灵活，扩展性高，有效地实现了指定业务带宽保障需求，不足之处在于没有在 Mininet 上模拟视频流，以提供更加直观的业务带宽保障服务。另外，控制平面缺乏对网络流量的实时监控和资源利用率等信息的采集，不能根据网络环境提供动态的 QoS 策略，缺乏自适应性，因此更加智能和自适应的 QoS 管理体系是下一步的主要研究方向。