Lab3: QoS Implementation with OvS

发布时间：2022.3.18 23:59

截止时间：2022.4.8 23:59

一、实验介绍

QoS（Quality of Service）即服务质量。指一个网络能够利用各种基础技术，为指定的网络通信提供更好的服务能力。当网络过载或拥塞时，QoS能确保重要业务量不受延迟或丢弃，同时保证网络的高效运行。对于网络业务，服务质量包括传输的带宽、传送的时延、数据的丢包率、网络抖动等。在网络中可以通过保证传输的带宽、降低传送的时延、降低数据的丢包率、降低网络抖动值等措施来提高服务质量。限流是Qos中常用的一种技术手段，本实验基于开源虚拟交换机Open vSwitch和网络仿真器Mininet，从传统网络模式和SDN模式完成限流的功能，对比出各种方式的特点。

二、环境准备

- *tips：*在普通用户下运行可能会遇到权限问题，建议在root权限下实验

系统要求

本实验推荐 **VMware Workstation** + **Ubuntu20.04** 的Linux环境

软件要求

- **安装Open vSwitch**

Open vSwitch(OVS)是运行在虚拟化平台上的虚拟交换机，其支持OpenFlow协议，也支持gre/vxlan/IPsec等隧道技术，并且应用于越来越多的开源项目。

1. 安装编译，可参考OvS官方教程 https://docs.openvswitch.org/en/latest/intro/install/general/）

Bash

```
#安装准备
apt install git build-essential autoconf automake libtool

#获取源代码
git clone https://github.com/openvswitch/ovs.git
cd ovs
git checkout origin/branch-2.13

#编译安装
./boot.sh
./configure
make
make install
make modules_install

#载入ovs模块到内核中
/sbin/modprobe openvswitch
/sbin/lsmod | grep openvswitch
```

完成后可得到如下结果：

Bash

```
openvswitch           147456  0
nsh                    16384  1 openvswitch
nf_conncount           24576  1 openvswitch
nf_nat                 49152  1 openvswitch
nf_conntrack          147456  4
nf_nat,nfnetlink_cttimeout,openvswitch,nf_conncount
nf_defrag_ipv6         24576  2 nf_conntrack,openvswitch
libcrc32c              16384  3 nf_conntrack,nf_nat,openvswitch
```

2. 运行Open vSwitch

```Bash
$ export PATH=$PATH:/usr/local/share/openvswitch/scripts

#配置ovsdb的数据库
$ mkdir -p /usr/local/etc/openvswitch
$ ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
vswitchd/vswitch.ovsschema
$ mkdir -p /usr/local/var/run/openvswitch
$ ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
--remote=db:Open_vSwitch,Open_vSwitch,manager_options \
--private-key=db:Open_vSwitch,SSL,private_key \
--certificate=db:Open_vSwitch,SSL,certificate \
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
--pidfile --detach --log-file

#使用ovs-vsctl初始化数据库
ovs-vsctl --no-wait init

#启动ovs-vswitchd
ovs-ctl --no-ovsdb-server start
```

验证ovs是否启动成功

```Bash
$ ovs-vsctl show
f2a9e1fd-ee07-44a2-9e87-9fbf2d006100
    ovs_version: "2.13.x"
```

- **安装Mininet**

Mininet是一个强大的虚拟化网络仿真工具，它可以创建一个包含主机，交换机，控制器和链路的虚拟网络，其交换机支持OpenFlow，具备高度灵活的自定义软件定义网络。

1. 安装编译

```Bash
git clone https://github.com/mininet/mininet.git
cd mininet/util
./install.sh -3n
```

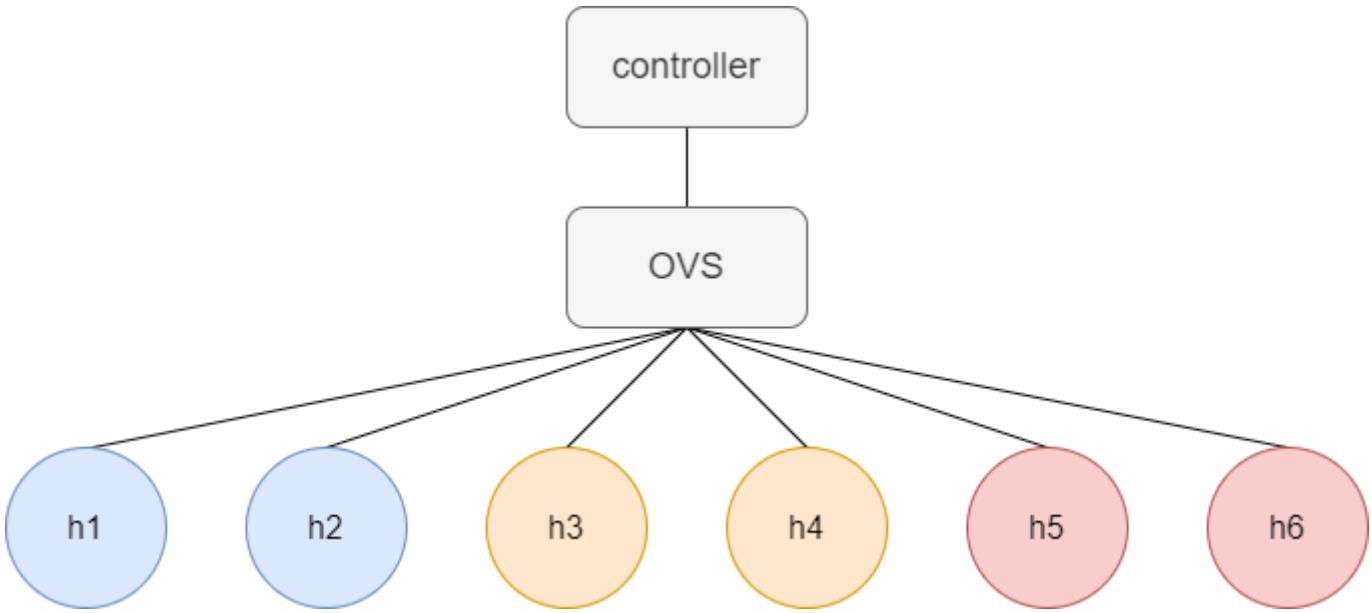
三、实验内容

本次实验共包含**六个Task**以及**两个Question**，请勿遗漏。

Part1：创建网络拓扑

· 拓扑设计与搭建

1. 设计拓扑



2. 搭建拓扑

Bash

```
#使用mininet命令搭建一个如上图所示的拓扑
$ mn --topo single,6 --switch ovs,protocols=OpenFlow13
```

输出如下：

Bash

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

- **连通性测试**

打开虚拟主机h1、h2，使用iperf测试两台虚拟主机之间是否连通

Bash

```
#使用xterm打开虚拟主机的命令行
$ mininet> xterm h1
$ mininet> xterm h2
```

在弹出的Node:h1窗口中

Bash

```
#使用ifconfig查看虚拟主机h1的ip地址
$ ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::a456:a8ff:fef5:5617  prefixlen 64  scopeid 0x20<link>
        ether a6:56:a8:f5:56:17  txqueuelen 1000  (Ethernet)
        RX packets 70  bytes 6511 (6.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

#打开iperf准备收包
$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

在Node:h2窗口中

Bash

```
#使用iperf向h1的ip发包
$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size:  416 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 52186 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  51.0 GBytes  43.8 Gbits/sec
```

同时检查Node:h1的输出

```
Bash
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  6] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 52186
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0-10.0 sec  51.0 GBytes  43.7 Gbits/sec
```

可以看出h1和h2之间已正常连通

Task1 请在你自己的环境中完成上面的连通性测试，并以截图的形式分别记录Node:h1和Node:h2中iperf的输出结果。要求：截图尽可能清晰，且需要连同任务栏的系统时间一起截图，后续的所有截图都要求包含系统时间。

Part2：三种限速方式

- 网卡限速

1. 简单介绍

Open vSwitch的QoS功能是基于Linux内核来完成的，Open vSwitch所做的是对Linux内核的限速功能进行配置。Linux内核中接收数据包使用的方法叫策略（policing），用于限制网卡上接收分组（ingress）的速率，当速率超过了配置速率，就简单的把数据包丢弃。Policing通过简单的丢包机制实现接口速率的限制，它既可以作用于物理接口，也可以作用于虚拟接口。

2. 实验过程

首先，在主机中

Bash

```
#使用ifconfig查看h1和h2对应的网卡
$ ifconfig

...
s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8c20:d7ff:fe94:f1f8  prefixlen 64  scopeid 0x20<link>
        ether 8e:20:d7:94:f1:f8  txqueuelen 1000  (Ethernet)
        RX packets 860391  bytes 56786070 (56.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1250073  bytes 54794581912 (54.7 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::2c2d:41ff:fe35:572e  prefixlen 64  scopeid 0x20<link>
        ether 2e:2d:41:35:57:2e  txqueuelen 1000  (Ethernet)
        RX packets 1249962  bytes 54794572648 (54.7 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 860468  bytes 56793066 (56.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
...

#使用vsctl设置两张网卡的ingress rate，当收包速率超过5000Kbps时，将多余的包直接丢掉
ovs-vsctl set interface s1-eth1 ingress_policing_rate=5000
ovs-vsctl set interface s1-eth2 ingress_policing_rate=5000
```

在h1中

Bash

```
#打开iperf收包
$ iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

在h2中

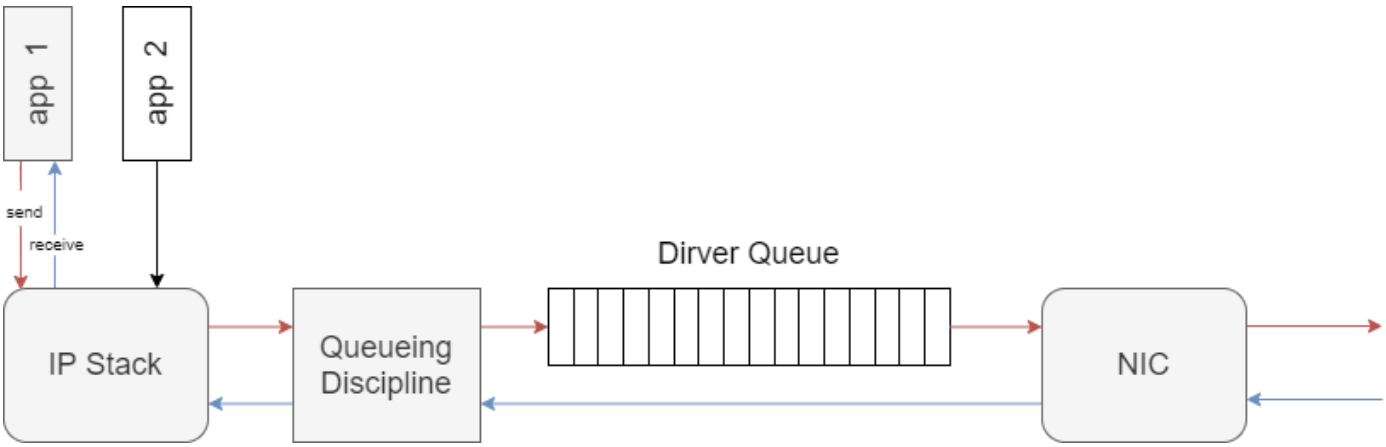
Bash

```
#向h1发包，将发送带宽设置为限速的两倍，10M
$ iperf -u -c 10.0.0.1 -b 10M
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1121.52 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
...
```

Task2.1 请截图记录输出结果，截图要求同**Task1**，并着重关注其中的带宽、抖动、丢包率等数据。

- **队列限速**

1. 简单介绍



数据包队列是任何一个网络栈的核心组件，数据包队列实现了异步模块之间的通讯，提升了网络性能。Linux可以将网络数据包缓存起来,然后根据用户的设置,在尽量不中断连接(如 tcp)的前提下来平滑网络流量。内核通过某个网络接口发送数据包,它都需要按照这个接口的队列规则把数据包加入队列。由于Linux对接收队列的控制不够好,所以一般只用发送队列,即”控发不控收”。shaping用于实现出口流量的控制，使用队列queue，可以缓存和调度数据包发送顺序，比policing更加精确和有效。在ovs的数据表中主要使用QoS和Queue两张表。

2. 实验过程

在主机中

Bash

```
#为h4的网卡创建队列

#为s1-eth4创建qos，名称为newqos

#创建qos，名称为newqos，类型为linux-htb，指定处理队列为q0

#创建队列q0，设置最大的速率为5000000bps，即5M
$ ovs-vsctl set port s1-eth4 qos=@newqos -- \
--id=@newqos create qos type=linux-htb queues=0=@q0 -- \
--id=@q0 create queue other-config:max-rate=5000000



#查看队列信息

#qos的参数信息，最重要的是处理队列的uuid
$ ovs-vsctl list qos
_uuid               : da72773b-b39e-47ab-8b8c-e01c8b31b38f
external_ids        : {}
other_config        : {}
queues              : {0=5148fd23-aa4e-413e-80b9-659bd242f402}
type                : linux-htb

#队列的信息，最重要的是队列的最大速度
$ ovs-vsctl list queue
_uuid               : 5148fd23-aa4e-413e-80b9-659bd242f402
dscp                : []
external_ids        : {}
other_config        : {max-rate="5000000"}
```

h4:

Bash

```
#查看ip
$ ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.4  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::64dc:18ff:fe6e:298f  prefixlen 64  scopeid 0x20<link>
        ether 66:dc:18:6e:29:8f  txqueuelen 1000  (Ethernet)
        RX packets 98  bytes 8571 (8.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1076 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

#打开iperf
$ iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

h3:

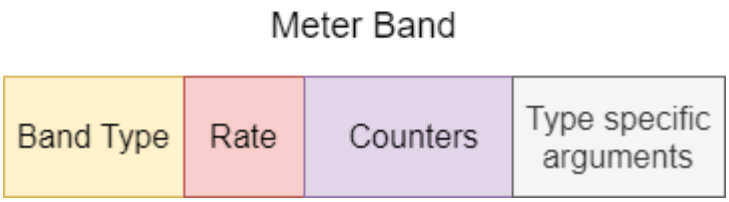
Bash

```
#打开iperf
$ iperf -u -c 10.0.0.4 -b 10M
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1121.52 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
...
```

Task2.2 同上，此处也需要截图记录实验结果。

- Meter表限速

1. 简单介绍



Meter表限速是SDN的限速方式。OpenFlow 1.3版本中增加了对单个数据流的计量功能，使得OpenFlow能够实现简单的QoS服务（例如流量限速），并且可以结合每个端口队列来实现更复杂的QoS框架（例如DiffServ）

- Meter Identifier 32位的无符号整数，用来唯一识别该计量表项
- Merter Bands 由计量带组成的无序列表，其中每个计量带都指明了其速率及处理数据包的方式
- Counters 用于在报文被计量表项处理时更新相关计数

每个计量表项可能具有有一个或多个计量带，每个计量带都指定了其所适用的速率和数据被处理的方式。每个Meter Band指明了带宽速率以及对数据包的处理行为。数据包基于其当前的速率会被其中一个Meter Band来处理。

- Band Type 定义了数据包怎样被处理（drop,dscp remark）
- Rate 用于选择计量带，定义了带可以运行的最高速率
- Counters 当数据报文被计量带处理时，更新计数器
- Counters 当数据报文被计量带处理时，更新计数器

Band Type

- Drop 通过丢弃数据包，定义带宽速率限制
- Dscp remark 降低数据包的IP头中的DSCP字段丢弃的优先级

2. 实验过程

主机中：

Bash

```
#查看交换机中的流表
$ ovs-ofctl dump-flows s1 -O openflow13


#设置交换机的工作模式和协议版本
$ ovs-vsctl set bridge s1 datapath_type=netdev
$ ovs-vsctl set bridge s1 protocols=OpenFlow13


#下发meter表并查看
$ ovs-ofctl add-meter s1 meter=1,kbps,band=type=drop,rate=5000 -O OpenFlow13


#查看流表转发端口，为下发流表准备
$ ovs-ofctl show s1 -O openflow13


#下发流表并查看
$ ovs-ofctl add-flow s1 in_port=5,action=meter:1,output:6 -O openflow13
$ ovs-ofctl dump-flows s1 -O openflow13
```

Question 1 尝试理解Line19,20两条指令，指出每条指令的具体工作是什么，并逐个分析其中各个参数的具体含义。

h6:

Bash

```
#查看ip
$ ifconfig
h6-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.6  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::a849:b6ff:fe77:1125  prefixlen 64  scopeid 0x20<link>
        ether aa:49:b6:77:11:25  txqueuelen 1000  (Ethernet)
        RX packets 99  bytes 8613 (8.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1076 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

#打开iperf
$ iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

h5:

Bash

```
#查看网卡
$ ifconfig
h5-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.5  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::60e8:93ff:fe58:a72f  prefixlen 64  scopeid 0x20<link>
        ether 62:e8:93:58:a7:2f  txqueuelen 1000  (Ethernet)
        RX packets 101  bytes 8697 (8.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17871  bytes 26996478 (26.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

#关闭网卡的tx校验
$ ethtool -K h5-eth0 tx off
Actual changes:
tx-checksumming: off
        tx-checksum-ip-generic: off
        tx-checksum-sctp: off
tcp-segmentation-offload: off
        tx-tcp-segmentation: off [requested on]
        tx-tcp-ecn-segmentation: off [requested on]
        tx-tcp-mangleid-segmentation: off [requested on]
        tx-tcp6-segmentation: off [requested on]

#使用iperf向h6发包
$ iperf -u -c 10.0.0.6 -b 10M
------------------------------------------------------------
Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1121.52 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
...
```

Task2.3 同上，请将此处的实验结果按要求截图。

Question 2 到这里，你已经完成了三种限速方式的实验，并获得了三组测试数据，请你就三组数据中的带宽、抖动和丢包率等参数，对三种限速方式进行横向比较，并适当地分析原因。

Part3：拓展与应用

- **场景介绍**

经过上述三个主机间交互的限速实验，你已经对使用Open vSwitch和Mininet的操作有了基本的了解。现在我们将研究在多台主机间交互时的QoS实现。

小明是上海交通大学的一名学生，在疫情封校期间他倍感无聊，准备在周末好好放松自己。周五晚上他打算为周末的计划准备一下，在玩《绝地求生》的同时又在后台下载《艾尔登法环》，并且还在下载一部自己很早就想看的电影。但是他发现边玩边下载让自己的游戏很卡，并且下载速度也很慢，这样自己晚上玩不好，并且明天的计划也要泡汤了！请你帮助小明设计一套QoS方案，以保证他的计划能够实现。

小明的想法是在保证游戏稳定的前提下，尽量将更多的带宽资源分配给《艾尔登法环》下载，电影是在最后一天看，慢慢下载也可以接受。

- **实验过程**
- *tips*：记得清理先前实验过程中遗留的配置，防止对该部分造成影响

1. 场景模拟

在该拓展部分使用先前Mininet创建的网络拓扑中从h1至h4四台虚拟主机。h1作为iperf的Server端，h2至h4为Client（假定Server端和Client端的网络带宽均为10Mb）。

在Server中运行下列指令：

```Bash
iperf -u -s
```

在三个Client中同时运行下列指令：

```Bash
iperf -u -c <ip of h1> -b 10M -t 20 -i 1
```

Task3 在限制Server端（h1）的带宽为10Mb的前提下，观察稳定后的三个Client的带宽，将结果截图并简单分析。

2. QoS设计

在该部分中，当Client同时运行时，你需要保证总带宽资源在他们之间分配方式如下：

h2：5Mb及以上

h3：3Mb及以上

h4：在保证h2和h3的前提下尽量多

Task4 你可以通过上述三种限速的方法来达成目标，请记录你的设计过程（思路及运行指令），并将你稳定后的三个Client的带宽结果截图。

四、提交要求

- 你应当在Canvas上提交一个gzip压缩包，命名为{Your student ID}.zip。其中Task完成结果分文件夹保存，Question回答以word或pdf文档的形式提交。
- 带有实验结果的截图请保证清晰度，并且规范命名（拒绝电脑随机生成的名字）。截图应带有系统时间。

五、评分标准

- Task：1，2.1，2.2，2.3各10%；3，4各20%。
- Question：1，2各10%。
- 若发现文档中的错误并提出修改意见，可获得**额外加分**。
- 杜绝抄袭现象，一经发现将严肃处理！