实验2 - 第1部分
在Openflow中应用QoS策略

目标

我们将在Openflow中创建规则，以限制或保证特定流量的带宽。

你将学到什么

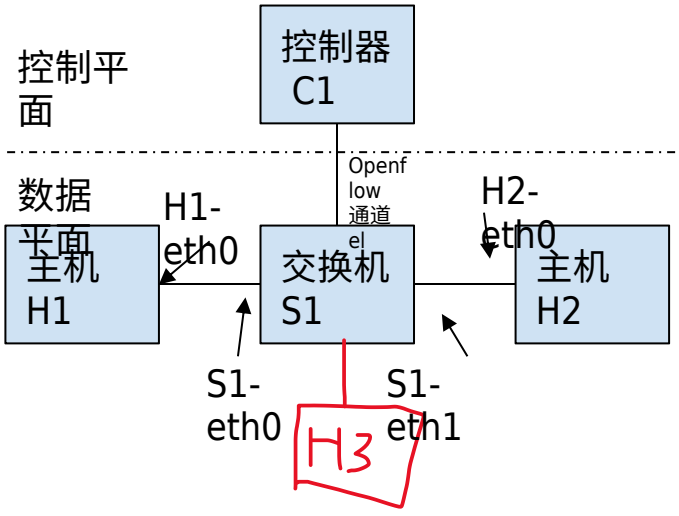
你将使用两种方法来实现QoS，使用队列和使用计量器

队列

这次，使用队列来限制流量速率。 我们实施的场景是通过适当使用队列来限制上行流量。

拓扑

我们创建了一个包含三个主机的简单拓扑。



$ sudo -E mn --topo single,3 --switch ovsk --controller remote --mac

我们使用链接命令在系统上验证拓扑

mininet> 链接
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)

我们还可以使用命令行中的ovs-vsctl命令验证系统上的端口

$ sudo ovs-vsctl show

e7a21c84-4464-4b53-9d84-7ac031b48c46
桥接器 *s1*
控制器 "*ptcp:6654*"
控制器 "*tcp:127.0.0.1:6653*"
故障模式*:* 安全
端口 *s1-eth2*
接口 *s1-eth2*
端口 *s1-eth1*
接口 *s1-eth1*
端口 *s1-eth3*
接口 *s1-eth3*
端口 *s1*
接口 *s1*
类型*:* 内部
*ovs*版本*:* "*2.13.1*"

初步的

首先，我们安装没有队列速率限制的简单转发规则。

sudo ovs-ofctl add-flow s1 dl_type=0x806,actions=output:all
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,actions=output:\"s1-eth2\"
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,actions=output:\"s1-eth1\"
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,actions=output:\"s1-eth3\"

然后我们可以运行iperf
mininet> h3 iperf -s &
mininet> h1 iperf -c 10.0.0.3

--
客户端连接到10.0.0.3，TCP端口5001
TCP窗口大小：340 K字节（默认）
--
[3] 本地10.0.0.1端口37054与10.0.0.3端口5001连接
[ID] 间隔 传输 带宽
[3] 0.0-10.0秒 9.24 G字节 7.94 Gbits/秒

mininet> h2 iperf -c 10.0.0.3
--
客户端连接到10.0.0.3，TCP端口5001
TCP窗口大小：1.38 M字节（默认）
--
[3] 本地10.0.0.2端口45796与10.0.0.3端口5001连接
[ID] 间隔 传输 带宽
[3] 0.0-10.0 秒 14.7 GB 12.6 Gbps

我们发现在TCP传输模式下，从主机h1和h2到h3的吞吐量约为10 Gbps（这取决于您的PC配置，可能更高或更低）

Adding QoS Rules

We can now shape the traffic using queues at the ingress port to the switches. In this instance, we create two queues for the port connected to h3. In one we set maximum rate to 500Mbps and minimum rate to 200Mbps. On the other set we set maximum rate to 100 Mbps and minimum rate to 50Mbps.

$ sudo ovs-vsctl set port s1-eth3 qos=@newqos -- --id=@newqos create qos type=linux-htb queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=200000000 other-config:max-rate=500000000 -- --id=@q1 create queue other-config:min-rate=50000000 other-config:max-rate=100000000

Notice that the firsto queue ID is 0, the second is 1. These are the IDs you will use in the flow rule.

Now we need to create new rules for forwarding packets form H1 and H2 to H3 to use the appropriate queue.

sudo ovs-ofctl del-flows s1 out_port=3 *# delete the flow rule towards h3*
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,dl_src=00:00:00:00:00:01,actions=enqueue:3:0
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,dl_src=00:00:00:00:00:02,actions=enqueue:3:1

mininet> h1 iperf -c 10.0.0.3

--
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 518 KByte (default)
--
[3] local 10.0.0.1 port 41956 connected with 10.0.0.3 port 5001
[ID] 间隔 传输 带宽
[3] 0.0-10.0 sec 484 MBytes 406 Mbits/sec

mininet> h2 iperf -c 10.0.0.3

--
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 264 KByte (default)
--
[3] local 10.0.0.2 port 46452 connected with 10.0.0.3 port 5001
[ID] 间隔 传输 带宽
[3] 0.0-10.0 sec 109 MBytes 90.9 Mbits/sec

IMPORTANT FOR THE ASSIGNMENTS!!

In the assignment you will have tow files, one is the controller application and the other is the python script running the entire mininet emulation.

The commands for creating queues should be created in the python mininet script. You can use the function **os.system(‘*COMMAND*’)** (where ***COMMAND*** is the command line text you want to execute).
In the python controller application you will need instead make sure that the appropriate flow rule (where required) sends the packets to a specific port and queue (i.e., use the command **flow.actions.append(of.ofp_action_enqueue(port =** *…variable identifying the destination port…***,queue_id=** *…variable identifying the queue id…* **)**)

You can see that the port has now a qos rule:

mininet> sudo ovs-vsctl list Port s1-eth3

```
_uuid               : 83120616-c0c3-405e-80cd-573620935f39
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [7db0864c-8a65-413e-9bda-618b6991b67b]
lacp                : []
mac                 : []
name                : s1-eth3
other_config        : {}
protected           : false
qos                 : 60f547c9-d672-4090-b22b-c8d2b19d4a01
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
```

you can see all qos rules by typing

mininet > *sudo ovs-vsctl list qos*

```
_uuid               : 60f547c9-d672-4090-b22b-c8d2b19d4a01
external_ids        : {}
other_config        : {}
queues              : {0=20491d88-0614-45eb-82cc-f2bcf1bff77b, 1=76b55b3d-3973-
4fec-9abc-5973d6adcd57}
type                : linux-htb
```

and the corresponding queues by typing

mininet > sudo ovs-vsctl list queue

```
_uuid               : 20491d88-0614-45eb-82cc-f2bcf1bff77b
dscp                : []
```

external_ids : {}
other_config : {max-rate="500000000", min-rate="200000000"}

_uuid : 76b55b3d-3973-4fec-9abc-5973d6adcd57
dscp : []
external_ids : {}
other_config : {max-rate="100000000", min-rate="50000000"}
the remove the qos rule

You can remove the qos rule from port s1-eth3 by typing:
mininet > *sudo ovs-vsctl clear Port s1-eth3 qos*

You will see that the qos is not anymore associated with port s1-eth3
mininet> sudo ovs-vsctl list Port s1-eth3
_uuid : 83120616-c0c3-405e-80cd-573620935f39
bond_active_slave : []
bond_downdelay : 0
bond_fake_iface : false
bond_mode : []
bond_updelay : 0
cvlans : []
external_ids : {}
fake_bridge : false
interfaces : [7db0864c-8a65-413e-9bda-618b6991b67b]
lacp : []
mac : []
name : s1-eth3
other_config : {}
protected : false
qos : []
rstp_statistics : {}
rstp_status : {}
statistics : {}
status : {}
tag : []
trunks : []
vlan_mode : []

However, the queues and qos rule will remain in the system unless they are manually removed

The rule is that you first need to remove the qos reference from the port (as just done above), then destroy the qos rule and then destroy the queue.
You need to do it in this exact order, otherwise the system won't let you remove an object which is still being referenced by another!!

You can either destroy one by one by using the uuid
mininet > sudo ovs-vsctl destroy qos *60f547c9-d672-4090-b22b-c8d2b19d4a01*

or all of them
mininet > sudo ovs-vsctl --all destroy qos

Then you can destroy the queues, by name:
mininet > sudo ovs-vsctl destroy queue 20491d88-0614-45eb-82cc-f2bcf1bff77b 76b55b3d-3973-4fec-9abc-5973d6adcd57

or all of them
mininet > sudo ovs-vsctl --all destroy queue

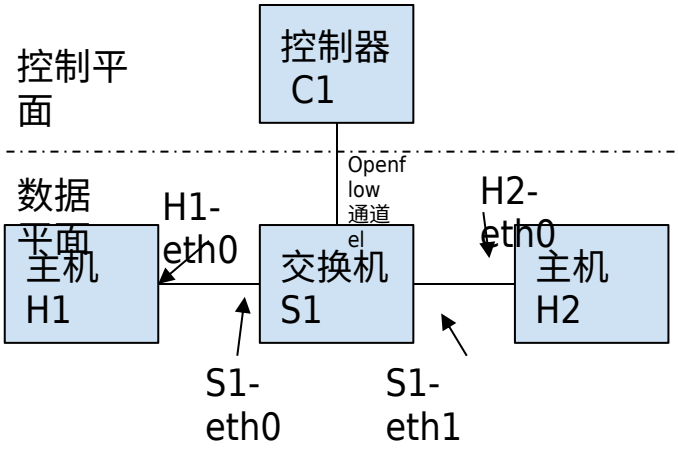
You can get more info on the ovs-ocftl in:
https://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt

Meters

Another approach is to make use of meters. They can be applied as part of an action on a flow entry, however they are not supported in POX, so you will not be using them for the assignments. They are based on a given threshold expressed in kbps or packets per second.

拓扑

The topology is composed of 2 hosts H1 and H2,, an openflow switch S1 and a controller C1.



Start mininet with single OVS switch and dummy controller. Simplify mac addresses

sudo -E mn --switch ovsk --controller remote --mac

Adding flows and rules

Use meters to restrict the traffic rate. Firstly, we remove all flows and meters
$ sudo ovs-ofctl del-flows s1
$ sudo ovs-ofctl -O OpenFlow13 del-meter s1 meter=1

Notice the -O OpenFlow13 which defines the use of OpenFlow 1.3 (which makes use of meter tables)

Next we insert the meter rules manually into the flow tables.

$ sudo ovs-ofctl -O OpenFlow13 add-meter s1 meter=1,kbps,band=type=drop,rate=30000

$ sudo ovs-ofctl -O OpenFlow13 add-flow s1 in_port=1,priority=100,actions=meter:1,output:2

$ sudo ovs-ofctl -O OpenFlow13 add-flow s1 in_port=2,priority=100,actions=output:1

You can get more info on the ovs-ocftl in: https://www.openvswitch.org/support/dist-docs-2.5/ovs-ofctl.8.txt

Testing

mininet> h2 iperf -s &
This commands starts an iperf server in node h2 and keeps it running in the background

mininet> h1 iperf -c 10.0.0.2
This commands starts an iperf client in node h1 towards h2
--
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 340 KByte (default)
--
[3] local 10.0.0.1 port 56934 connected with 10.0.0.2 port 5001
[ID] 间隔　　传输 带宽
[3] 0.0-10.0 sec 43.2 MBytes 36.1 Mbits/sec

Now you can try to modify the meter and run it again:
$ sudo ovs-ofctl -O OpenFlow13 mod-meter s1 meter=1,kbps,band=type=drop,rate=300000

You should see a rate 10 times higher:
--
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 850 KByte (default)
--
[3] local 10.0.0.1 port 58756 connected with 10.0.0.2 port 5001
[ID] 间隔　　传输 带宽
[3] 0.0-10.1 sec   428 MBytes   357 Mbits/sec

Validating / debugging meters

We can validate or debug the use of the meters.
$ sudo ovs-ofctl -O OpenFlow13 meter-stats s1

OFPST_METER reply (OF1.3) (xid=0x2):
meter:1 flow_count:1 packet_in_count:4374 byte_in_count:50483292 duration:586.887s bands:
0: packet_count:353 byte_count:4867026
This first shows packet and byte count for the entire flow, then for the specific band.

sudo ovs-ofctl -O OpenFlow13 meter-features s1
OFPST_METER_FEATURES reply (OF1.3) (xid=0x2):
max_meter:4294967295 max_bands:1 max_color:0
band_types: drop
capabilities: kbps pktps burst stats

Here the max meter shows the maximum available data rate to be set for metering (i.e. a UINT32 number). Only one band is implemented and only drop can be carried out.

—----------