

# Lab 4 Report

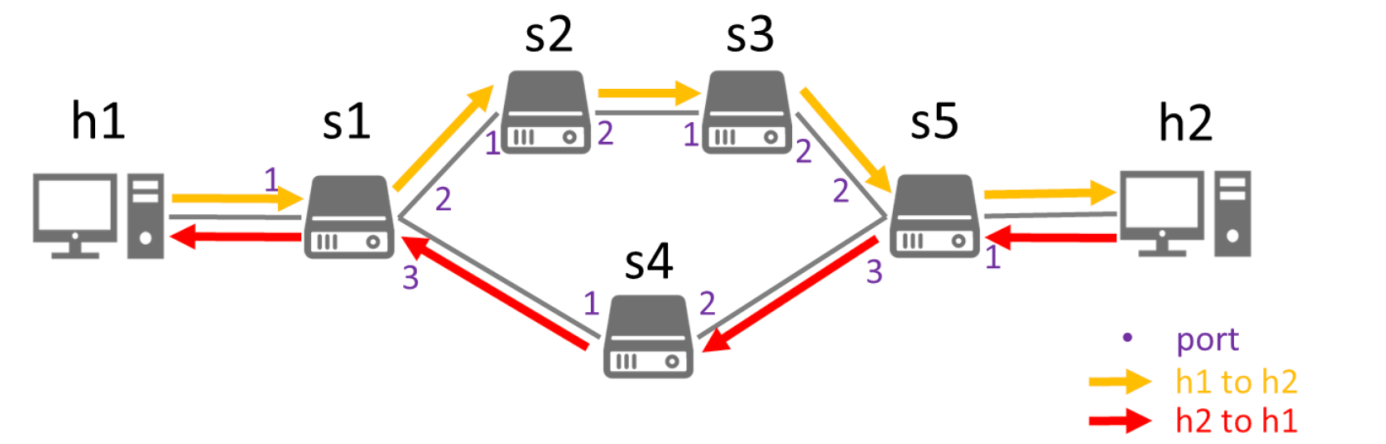
Student Name: 鄭准薰  
Student ID: 313551097

## Introduction

In this lab, I implemented a groupmeter app to control traffic flow and apply rate limits in a ring network topology using SDN. The app configures group tables and meter rules on the switches, allowing me to manage traffic paths and limit bandwidth when needed. This setup helps direct traffic between hosts, reroute it when links go down, and enforce rate limits to control network congestion. Using the groupmeter app, I tested different scenarios to observe how the app affects traffic behavior and flow.

## Test Results

### Workflow1 - s1-s2 link up



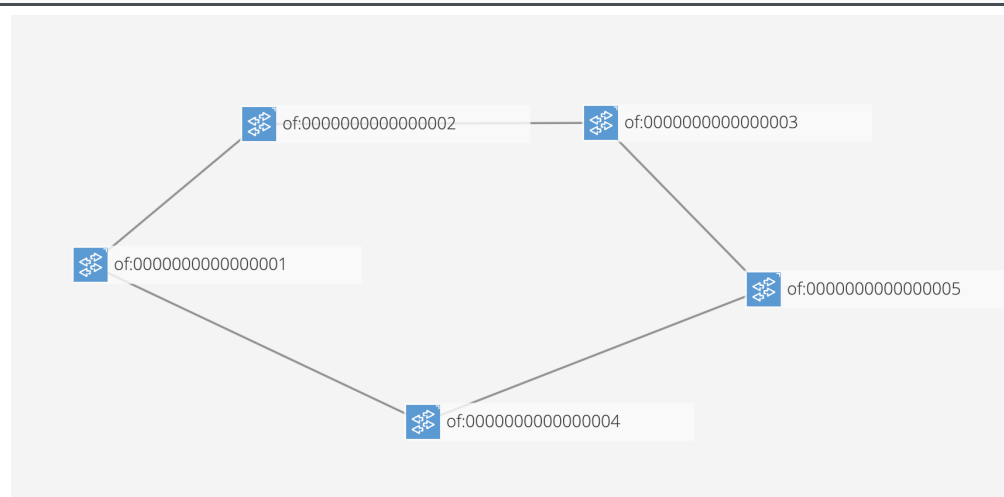
In Workflow1, I set up a ring topology with 5 switches and 5 hosts. I configured h1 as the iperf UDP client and h2 as the iperf UDP server to test network traffic. With s1 connected to s2, the group table on s1 directs traffic from h1 to h2 to flow through s2 and s3. The following steps detail the process and results of Workflow1.

### 1. Build the topology

I run the following command to build a ring topology, as shown in Figure1.

```
$ sudo mn --custom=ring_topo.py --topo=mytopo \
--controller=remote,ip=127.0.0.1,port=6653 \
--switch=ovs,protocols=OpenFlow14
```

Figure1: Topology



## 2. Upload config file to ONOS

Then I upload the config file `hostconfig.json` to ONOS by the following command.

```
$ onos-netcfg localhost hostconfig.json
```

Figure2: Upload config file to ONOS

```

02:22:16.880 INFO [DistributedGroupStore] Group AUDIT: Setting device of:0000000000000005 initial AUDIT completed
02:22:16.880 INFO [DistributedGroupStore] Group AUDIT: Setting device of:0000000000000003 initial AUDIT completed
02:22:16.883 INFO [DistributedGroupStore] Group AUDIT: Setting device of:0000000000000002 initial AUDIT completed
02:22:16.884 INFO [DistributedGroupStore] Group AUDIT: Setting device of:0000000000000004 initial AUDIT completed
02:22:16.907 INFO [DistributedGroupStore] Group AUDIT: Setting device of:0000000000000001 initial AUDIT completed
02:22:31.541 INFO [NetworkConfigManager] Configuration 'informations' queued for subject DefaultApplicationId{id=176, name=nycu.winlab.groupmeter}
  
```

```

$ onos-netcfg localhost hostconfig.json
~/SDNFV-Lab/lab4 main*
$
  
```

## 3. Build, install, and activate your App

After building, installing, and activating the groupmeter App, our app receives the configuration from the uploaded config and installs the necessary group, meter, and flow rules to the switches. Figure3 shows the log of configuration information.

Figure3: Build, install, and activate groupmeter App

```

02:23:08.686 INFO [FeaturesServiceImpl] Changes to perform:
02:23:08.687 INFO [FeaturesServiceImpl] Region: root
02:23:08.687 INFO [FeaturesServiceImpl] Bundles to install:
02:23:08.687 INFO [FeaturesServiceImpl] mvn:nycu.winlab/groupmeter/1.0-SNAPSHOT
02:23:08.688 INFO [FeaturesServiceImpl] Installing bundles:
02:23:08.688 INFO [FeaturesServiceImpl] mvn:nycu.winlab/groupmeter/1.0-SNAPSHOT
02:23:08.698 INFO [FeaturesServiceImpl] Starting bundles:
02:23:08.700 INFO [FeaturesServiceImpl] nycu.winlab.groupmeter/1.0.0.SNAPSHOT
02:23:08.726 INFO [AppComponent] ConnectPoint_h1: of:0000000000000001/1, ConnectPoint_h2: of:0000000000000005/1
02:23:08.727 INFO [AppComponent] MacAddress_h1: 00:00:00:00:00:01, MacAddress_h2: 00:00:00:00:00:02
02:23:08.727 INFO [AppComponent] IpAddress_h1: 10.6.1.1, IpAddress_h2: 10.6.1.2
02:23:08.726 INFO [FeaturesServiceImpl] Done.
02:23:08.729 INFO [ApplicationManager] Application nycu.winlab.groupmeter has been activated

```

---

```

[INFO] Installing nycu/winlab/groupmeter/1.0-SNAPSHOT/groupmeter-1.0-SNAPSHOT.jar
[INFO] Writing OBR metadata
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.428 s
[INFO] Finished at: 2024-11-04T02:22:36+08:00
[INFO] -----
> onos-app localhost install! target/groupmeter-1.0-SNAPSHOT.oar
{"name":"nycu.winlab.groupmeter","id":176,"version":"1.0.SNAPSHOT","category":"default","description":"ONOS OSGi bundle archetype","readme":"ONOS OSGi bundle archetype","origin":"Winlab, NYCU","url":"http://onosproject.org","featuresRepo":"mvn:nycu.winlab/groupmeter/1.0-SNAPSHOT/xml/features","state":"ACTIVE","features":["groupmeter"],"permissions":[],"requiredApps":[]}

```

#### 4. Use h1 as iperf UDP client and h2 as iperf UDP server to test your traffic

Next, I used iperf UDP on h1 to h2 to test the traffic. The result is shown in Figure4. The yellow box in Figure4 shows the intent information log indicating that our app has installed the intent right after udp packet-in.

Figure4: Run iperf UDP on h1 to h2

```

02:23:08.726 INFO [FeaturesServiceImpl] Done.
02:23:08.729 INFO [ApplicationManager] Application nycu.winlab.groupmeter has been activated
02:24:03.225 INFO [AppComponent] Intent `of:0000000000000002`, port `1` => `of:0000000000000005`, port `1` is submitted.
02:24:04.246 INFO [AppComponent] Intent `of:0000000000000005`, port `1` => `of:0000000000000001`, port `1` is submitted.

```

---

```

*** Creating network
*** Adding controller
mininet> h2 iperf -s -u -i 1&
mininet> h1 iperf -c h2 -u -b 2M -i 1

Client connecting to 10.6.1.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 5607.60 us (kalman adjust)
UDP buffer size: 208 KByte (default)

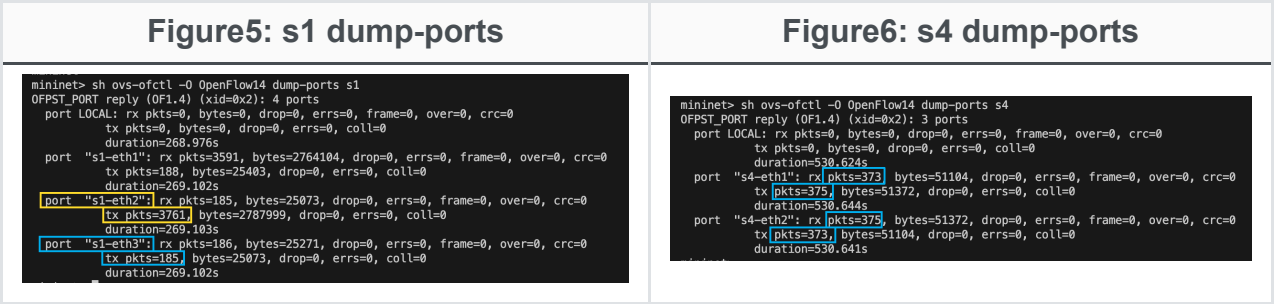
[ 1] local 10.6.1.1 port 38550 connected with 10.6.1.2 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 1] 0.0000-1.0000 sec  258 KBytes   2.12 Mbits/sec
[ 1] 1.0000-2.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 2.0000-3.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 3.0000-4.0000 sec  257 KBytes   2.11 Mbits/sec
[ 1] 4.0000-5.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 5.0000-6.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 6.0000-7.0000 sec  257 KBytes   2.11 Mbits/sec
[ 1] 7.0000-8.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 8.0000-9.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 9.0000-10.0000 sec 257 KBytes   2.11 Mbits/sec
[ 1] 0.0000-10.0097 sec 2.51 MBytes   2.10 Mbits/sec
[ 1] Sent 1788 datagrams
[ 1] Server Report:
[ ID] Interval      Transfer      Bandwidth      Jitter   Lost/Total Datagrams
[ 1] 0.0000-10.0078 sec 2.51 MBytes   2.10 Mbits/sec  0.009 ms 0/1787 (0%)
mininet>

```

#### 5. Monitor s1 and s4 interface

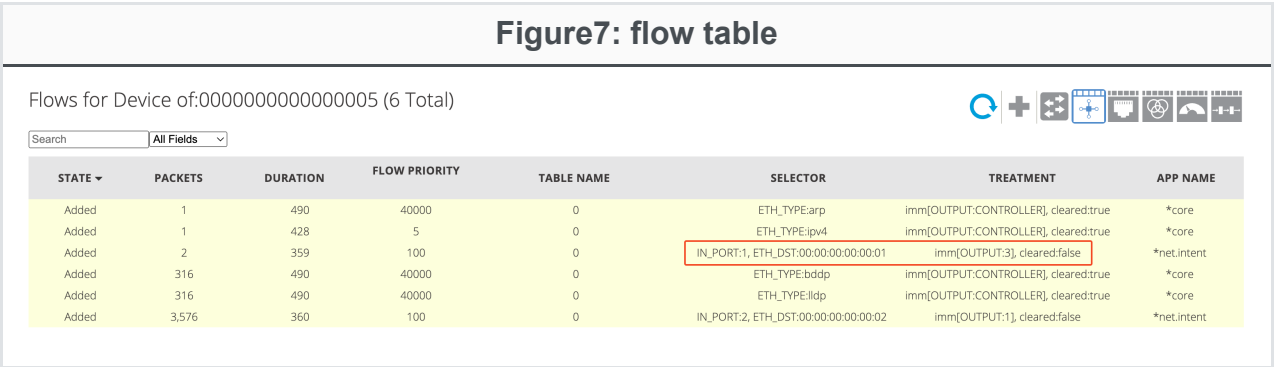
- Check if traffic from h1 to h2 through s2 and s3

Figures 5 and 6 show the port dumps for s1 and s4. About 3,000 packets pass through port "s1-eth2," while only around 100 packets go through port "s1-eth3" toward s4. This shows that the UDP traffic from h1 to h2 primarily goes through s2 and s3.

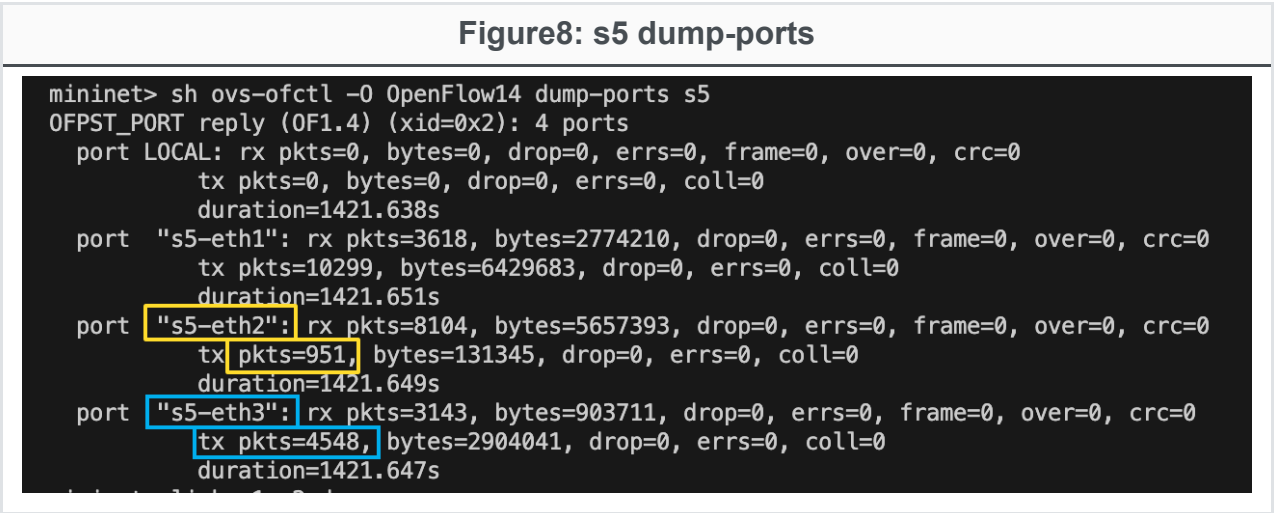


- Check the path from h2 to h1 through which switch

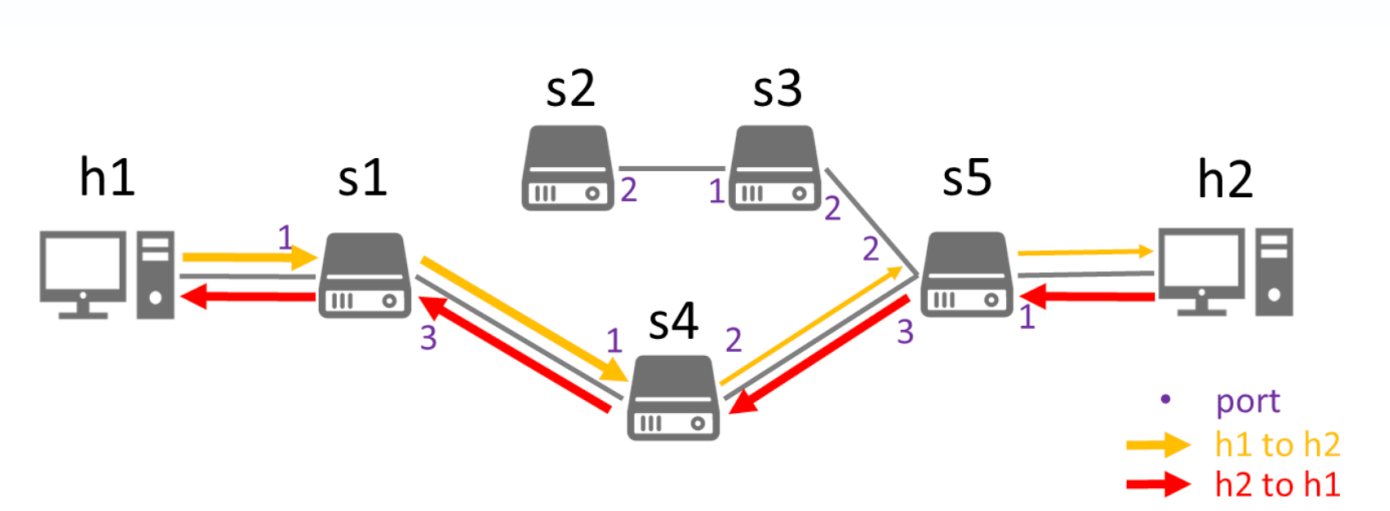
To verify the path from h2 to h1, we first examine the flow table of s5, shown in Figure7. The flow table on s5 includes a rule that forwards packets to port 3, which is connected to s4.



To further confirm this path, I conducted an additional test by running iperf UDP from h2 to h1 while monitoring the port dumps on s5. As shown in Figure8, the traffic from h2 to h1 indeed passes through port "s5-eth3.", confirming that the path from h2 to h1 goes through s4.



Workflow2 - s1-s3 link down

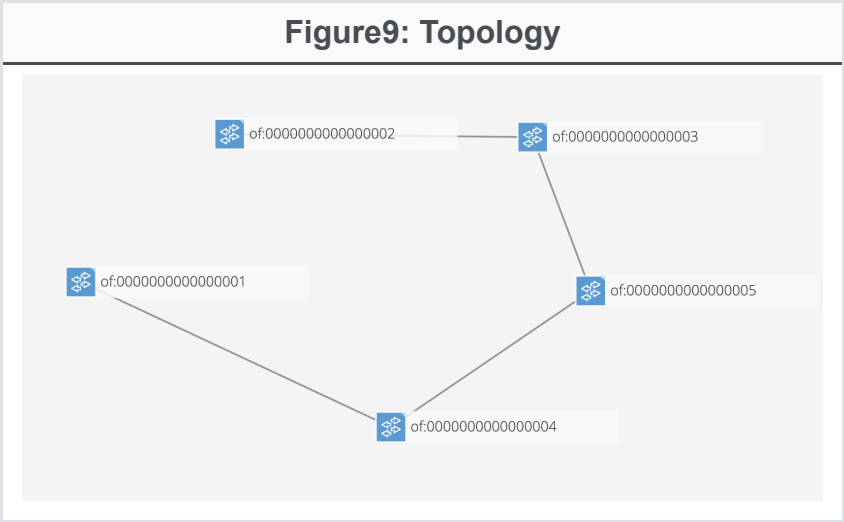


Workflow2 is similar to Workflow1, but with the s1-s2 link down. Thus, s1 will direct traffic from h1 to h2 to flow through s4. Also, the traffic rate will be limited by the meter rule on s4. The following steps detail the process and results of Workflow2.

6. Turn down s1-s2 link

I run the following command to turn down the s1-s2 link.

```
mininet> link s1 s2 down
```



7. Run iperf UDP on h1 to h2

I tested the traffic by running iperf UDP on h1 to h2. The result is shown in Figure10. The yellow box in Figure10 shows the intent information log indicating that our app has installed the intent right after udp packet-in.

Figure10: Run iperf UDP on h1 to h2

```
02:27:53.110 INFO [DeviceManager] Device of:0000000000000001 port 2 status changed (enabled=false)
02:27:53.110 INFO [DeviceManager] Device of:0000000000000002 port 1 status changed (enabled=false)
02:27:53.134 INFO [TopologyManager] Topology.DefaultTopology(file=273076880703368, creationTime=1730658473131, computeCost=373802,
02:28:04.225 INFO [AppComponent] Intent 'of:0000000000000005', port '3' => 'of:0000000000000005', port '1' is submitted.
[]
PROBLEMS 9 OUTPUT DEBUG CONSOLE PORTS 1 COMMENTS TERMINAL
mininet> h1 iperf -c h2 -u -b 2M -i 1
Client connecting to 10.6.1.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 5607.60 us (kalman adjust)
UDP buffer size: 208 KByte (default)

[ 1] local 10.6.1.1 port 40180 connected with 10.6.1.2 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 1] 0.0000-1.0000 sec  258 KBytes   2.12 Mbits/sec
[ 1] 1.0000-2.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 2.0000-3.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 3.0000-4.0000 sec  257 KBytes   2.11 Mbits/sec
[ 1] 4.0000-5.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 5.0000-6.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 6.0000-7.0000 sec  257 KBytes   2.11 Mbits/sec
[ 1] 7.0000-8.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 8.0000-9.0000 sec  256 KBytes   2.09 Mbits/sec
[ 1] 9.0000-10.0000 sec 257 KBytes   2.11 Mbits/sec
[ 1] 0.0000-10.0098 sec 2.51 MBytes   2.10 Mbits/sec
[ 1] Sent 1788 datagrams
[ 1] Server Report:
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-10.0361 sec 563 KBytes   459 Kbits/sec   1.696 ms 1396/1788 (78%)
mininet>
```

## 8. Monitor s1 and s4 interface

- Check if both traffic go through s4

Figures 11 and 12 display the port dumps for s1 and s4. Originally, as see in Figure 5, port "s1-eth3" was handling around 100 packets, but after the s1-s2 link goes down, this number increases to 3,000 packets. This indicates that the traffic from h1 to h2 is now being rerouted through s4.

Figure11: s1 dump-ports (link down)

```
mininet> sh ovs-ofctl -O OpenFlow14 dump-ports s1
OFPST_PORT reply (OF1.4) (xid=0x2): 4 ports
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=0, errs=0, coll=0
duration=500.304s
port "s1-eth1": rx pkts=7375, bytes=5531752, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=356, bytes=55375, drop=0, errs=0, coll=0
duration=500.430s
port "s1-eth2": rx pkts=261, bytes=35637, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=3838, bytes=2798633, drop=0, errs=0, coll=0
duration=500.431s
port "s1-eth3": rx pkts=353, bytes=55201, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=3916, bytes=2813251, drop=0, errs=0, coll=0
duration=500.430s
```

Figure12: s4 dump-ports (link down)

```
mininet> sh ovs-ofctl -O OpenFlow14 dump-ports s4
OFPST_PORT reply (OF1.4) (xid=0x2): 3 ports
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=0, errs=0, coll=0
duration=531.799s
port "s4-eth1": rx pkts=3937, bytes=2816138, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=374, bytes=58088, drop=0, errs=0, coll=0
duration=531.920s
port "s4-eth2": rx pkts=376, bytes=58366, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=2540, bytes=819132, drop=0, errs=0, coll=0
duration=531.920s
```

- Check if the iperf traffic rate is limited

To verify that the traffic rate is limited by the meter on s4, I ran an iperf UDP test from h1 to h2 while monitoring the port dumps on s4. Figure 14 shows that when traffic flows through s4, the meter enforces a 1 Mbps cap, resulting in reduced throughput and potential packet drops once the rate exceeds the limit. The meter on s4 monitors traffic and allows bursts up to 1024 KB, but once the burst buffer is exceeded and traffic surpasses 512 KB/sec, the meter begins dropping excess packets. In contrast, Figure 13 shows traffic passing through s2 without any rate limit, allowing higher sustained bandwidth and no packet loss.

Figure13: h2 jobs (no rate limit)

```
mininet> h2 jobs
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)

[ 1] local 10.6.1.2 port 5001 connected with 10.6.1.1 port 38550
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-1.0000 sec  258 KBytes   2.12 Mbits/sec   0.015 ms 0/180 (0%)
[ 1] 1.0000-2.0000 sec  256 KBytes   2.09 Mbits/sec   0.014 ms 0/178 (0%)
[ 1] 2.0000-3.0000 sec  257 KBytes   2.11 Mbits/sec   0.008 ms 0/179 (0%)
[ 1] 3.0000-4.0000 sec  256 KBytes   2.09 Mbits/sec   0.009 ms 0/178 (0%)
[ 1] 4.0000-5.0000 sec  256 KBytes   2.09 Mbits/sec   0.026 ms 0/178 (0%)
[ 1] 5.0000-6.0000 sec  257 KBytes   2.11 Mbits/sec   0.012 ms 0/179 (0%)
[ 1] 6.0000-7.0000 sec  256 KBytes   2.09 Mbits/sec   0.011 ms 0/178 (0%)
[ 1] 7.0000-8.0000 sec  256 KBytes   2.09 Mbits/sec   0.010 ms 0/178 (0%)
[ 1] 8.0000-9.0000 sec  257 KBytes   2.11 Mbits/sec   0.011 ms 0/179 (0%)
[ 1] 9.0000-10.0000 sec 256 KBytes   2.09 Mbits/sec   0.010 ms 0/178 (0%)
[ 1] 0.0000-10.0078 sec 2.51 MBytes   2.10 Mbits/sec   0.010 ms 0/1787 (0%)
[1]+  Running                  iperf -s -u -i 1 &
```

Figure14: h2 jobs (rate limit)

```
mininet> h2 jobs
[ 2] local 10.6.1.2 port 5001 connected with 10.6.1.1 port 39594
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 2] 0.0000-1.0000 sec  169 KBytes   1.39 Mbits/sec   0.005 ms 60/178 (34%)
[ 2] 1.0000-2.0000 sec  41.6 KBytes   341 Kbits/sec   0.008 ms 144/173 (83%)
[ 2] 2.0000-3.0000 sec  44.5 KBytes   365 Kbits/sec   0.006 ms 151/182 (83%)
[ 2] 3.0000-4.0000 sec  44.5 KBytes   365 Kbits/sec   0.008 ms 151/182 (83%)
[ 2] 4.0000-5.0000 sec  44.5 KBytes   365 Kbits/sec   0.010 ms 146/177 (82%)
[ 2] 5.0000-6.0000 sec  43.1 KBytes   353 Kbits/sec   0.008 ms 143/173 (83%)
[ 2] 6.0000-7.0000 sec  43.1 KBytes   353 Kbits/sec   0.013 ms 147/177 (83%)
[ 2] 7.0000-8.0000 sec  45.9 KBytes   376 Kbits/sec   0.011 ms 150/182 (82%)
[ 2] 8.0000-9.0000 sec  47.4 KBytes   388 Kbits/sec   0.005 ms 149/182 (82%)
[ 2] 9.0000-10.0000 sec 44.5 KBytes   365 Kbits/sec   0.014 ms 147/178 (83%)
[ 2] 0.0000-10.0443 sec 570 KBytes   465 Kbits/sec   2.191 ms 1391/1788 (78%)
[1]+  Running                  iperf -s -u -i 1 &
```