

ELEC5517 Project 2

Group 19

Student	SID
Youhao Zheng	490278310
Chao Wang	480080978
Mengzhen Chen	480462613
ShuTing Li	460016168
Xiaoyi Xu	490126549

Contents

1. Scenario Design	1
2. Task1	2
2.1 Create Topology	2
2.2 Use Wireshark to Catch OpenFlow packets	3
2.3 Show and Assign the Bandwidth Paths	5
3. Task 2	5
3.1 Initial Sample	6
3.2 Network Delays	7
3.2.1 Transmission Delay	7
3.2.2 Propagation delay	7
3.2.3 Queuing Delay	8
3.2.4 Processing Delay	10
3.3 Similarity	10
4. Task 3	11
4.1 Implementation the Topology in the ONOS	11
4.2 Isolating the Media Switch From the Network	12
4.3 Get the statistical data	15
4.4 Add a New Host on S2	15
Reference	18

1. Scenario Design

AT&T WorldNet is planning to update its software-defined network system in Australia and provide related IT services to various customers in the telecommunications, banks, and media industries. Before doing so, their engineers designed a simple testing tree model with a fan out of three.

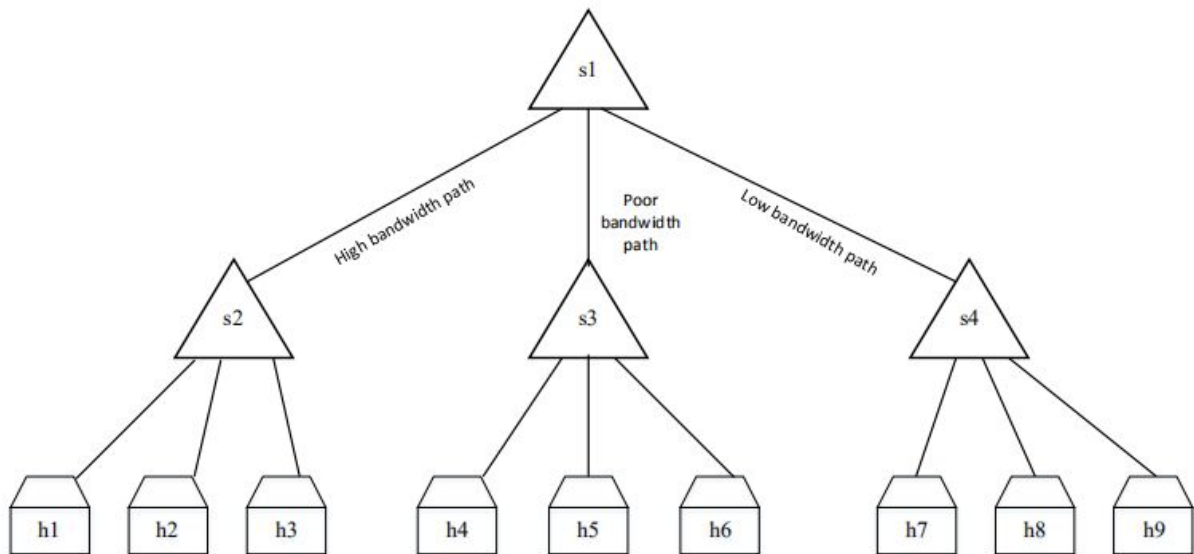


Figure 1.1 Tree network topology example

The host in these industries refers to different IT services, the details are shown in the following table.

Table 1. Industry and bandwidth [1][2][3]

Industry	No.	Services	Bandwidth (Mbps)
Telecommunication	h1	Message	2
	h2	Call	64
	h3	Live	1000
Media	h4	Text	32
	h5	Audio	64
	h6	Vlog (Video)	600
Bank	h7	Internal Message	2
	h8	Internal Call	4
	h9	Camera on ATM	64

The bandwidth in this table represents the assumed the required maximum bandwidth to support this IT service. The design bandwidth for each similar IT service will vary depending on the customer.

2. Task1

2.1 Create Topology

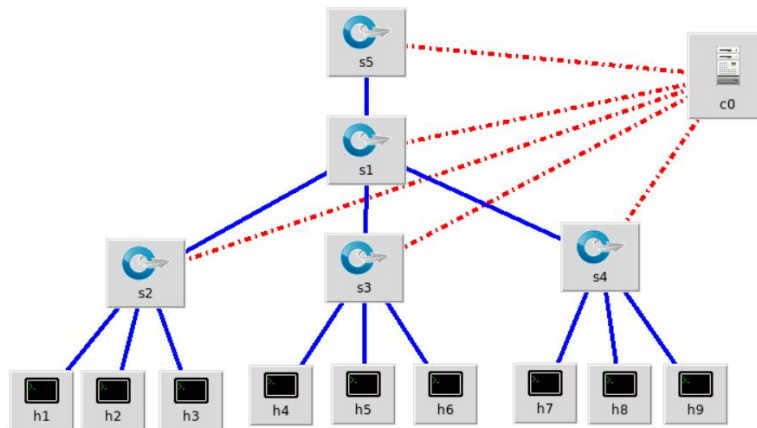


Figure 2.1 Created topology

The Miniedit is used to create a basic tree topology, which is shown in figure 2.1. At the time, each industry can communicate with others. Then the data communication control is implemented.

There are three options to simulate the data communication control: drop the data, delete the links, or transfer the data output port. However, considering the real-world cases, dropping data and turning down the links is unacceptable, therefore, one option is left. To do this, the new switch S5 was added as an individual switch, which does not connect with s2, s3, s4 directly. Then, s1 transfers the data inputted from s2, s3, s4 to s5 by the following commands:

```

pctl add-flow in_port=s1-eth1,actions=output:s1-eth4
pctl add-flow in_port=s1-eth2,actions=output:s1-eth4
pctl add-flow in_port=s1-eth3,actions=output:s1-eth4

```

In this case, s2, s3, s4 will not be allowed to connect with others. The connecting configuration is shown in figure 2.2

```

mininet> pingall
*** Ping: testing ping reachability
h5 -> X X h4 X X h6 X X
h8 -> X X X X X X h9 h7
h2 -> X X X h1 h3 X X X
h4 -> h5 X X X X h6 X X
h1 -> X X h2 X h3 X X X
h3 -> X X h2 X h1 X X X
h6 -> h5 X X h4 X X X X
h9 -> X h8 X X X X X h7
h7 -> X h8 X X X X X h9
*** Results: 75% dropped (18/72 received)
mininet>

```

Figure 2.2 Network connection (add-flow)

To checking the results, after reset the network, the links is deleted by the following commands:

```

links s1 s2 down
links s1 s3 down
links s1 s4 down

```

The result shows that our first method does avoid the data exchange between these industries. The result is shown in figure 2.3.

```

mininet> link s1 s2 down
mininet> link s1 s3 down
mininet> link s1 s4 down
mininet> pingall
*** Ping: testing ping reachability
h5 -> X X h4 X X h6 X X
h8 -> X X X X X X h9 h7
h2 -> X X X h1 h3 X X X
h4 -> h5 X X X X h6 X X
h1 -> X X h2 X h3 X X X
h3 -> X X h2 X h1 X X X
h6 -> h5 X X h4 X X X X
h9 -> X h8 X X X X X h7
h7 -> X h8 X X X X X h9
*** Results: 75% dropped (18/72 received)
mininet>

```

Figure 2.3 Created Topology (link down)

2.2 Use Wireshark to Catch OpenFlow packets

To do this experiment, we first run Wireshark as background, then use the python command line to set h1,h4, h7 as a simple HTTP server. The commands is as following:

```
h1 python -m SimpleHTTPServer 80 &
h4 python -m SimpleHTTPServer 80 &
h7 python -m SimpleHTTPServer 80 &
```

and use h2, h5, h8 to download the data by the following command:

```
h2 wget 10.0.0.1
h5 wget 10.0.0.4
h8 wget 10.0.0.7
```

Then using Wireshark to catch the Openflow packets. The packets' information are shown in the figure 2.4, 2.5 and 2.6.

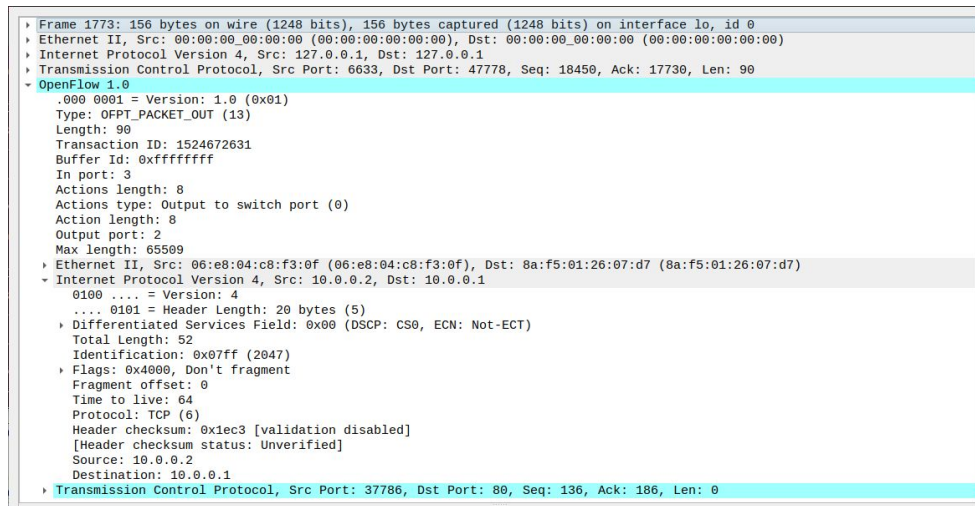


Figure 2.4 h1 to h2

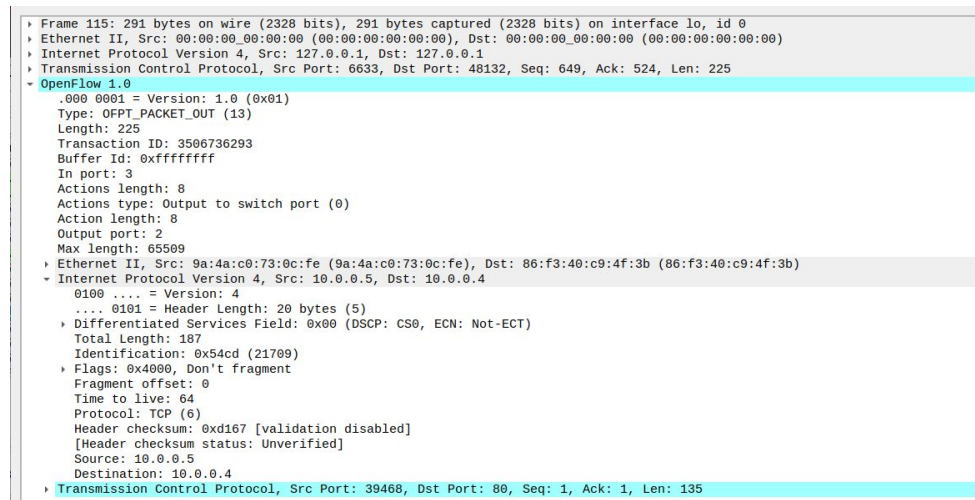


Figure 2.5 h4 to h5

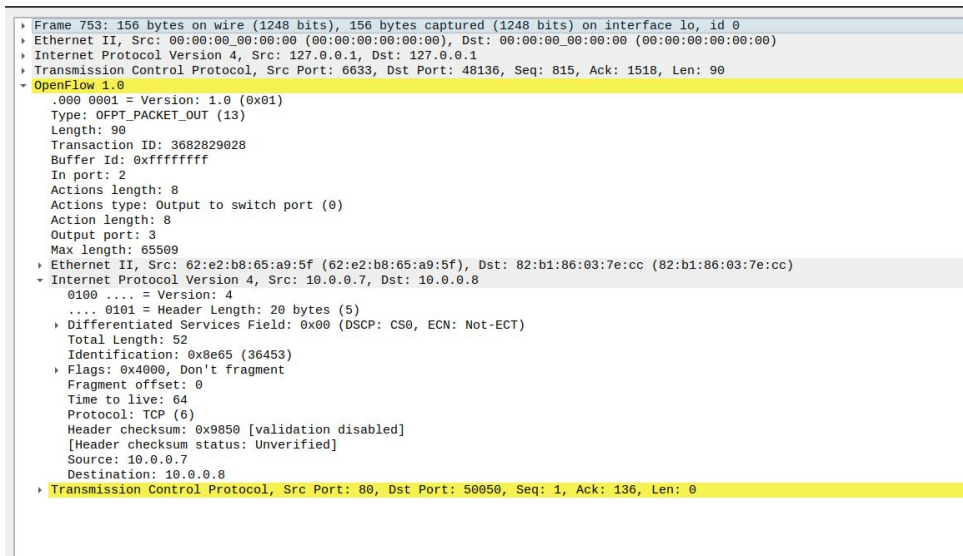


Figure 2.6 h7 to h8

There are no significant differences in these packets. Except the length of the OpenFlow protocol and the length of the internet protocol have some significant differences. For the similarity, the actions of each flow table are both outputting the data to a specific port, since we didn't add the drop or group action to the flow table. And many other parameters are also the same.

2.3 Show and Assign the Bandwidth Paths

As shown in table 1 in section 1, we assign different bandwidth according to the demand of each industry. And the result and topology is shown in figure 2.7.

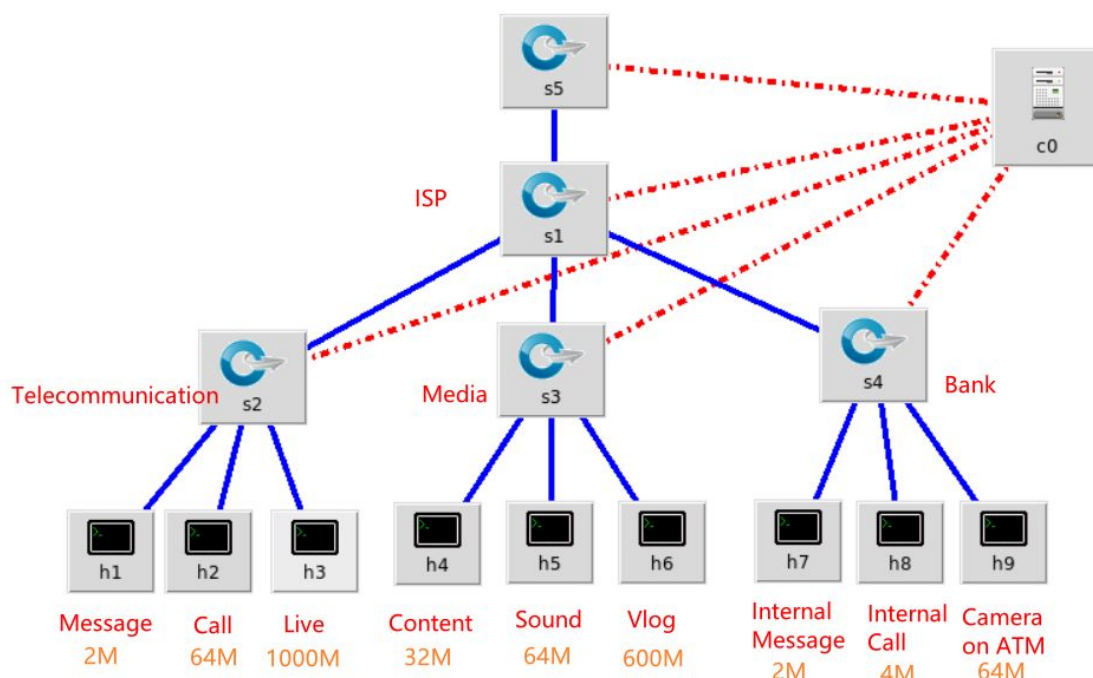


Figure 2.7 h7 to h8

Note that the difference in bandwidth is caused by the number of users that a switch will connect to. For example, In the telecommunication industry, the user number will be larger than other industries, so the bandwidth of the link which is used to transfer the Call and Live is larger. As for the bank, all the functions are only used by a small number of users, so the bandwidth is smaller than the others.

3. Task 2

In the telecommunication network, when the information transmitted between two end devices based on the packet-switched network, the transmission would take a certain amount of time to be finalised. This total time of data packet delivery is called the network delay. Based on the usage of the packet-switched network, there are three types of the delays included, namely the transmission delay, propagation delay and the processing delay. Additionally, the queuing delay should also be considered for the network due to the routers and switches[4].

The delay time of low-latency network connections is smaller, and the delay time of high-latency connections is longer. In addition to propagation delay, delay may also involve transmission delay (attribute of the physical medium) and processing delay (for example, through a proxy server or network hops on the Internet).

In this task, the influence of these delays on the read and write access are required to be discussed. To implement this, the bandwidth paths and corresponding attributes are specified in the Mininet environment by relating the read access permissions to download speed and the write access permissions to upload speed based on the new speed on the network paths. Furthermore, all the parameters, meaning delay, loss and max queue, are set between the s1 and h2.

3.1 Initial Sample

Firstly, the command "iperf h1 h2" is used to test the TCP transmission between s1 and h2, which is set to be the initial sample. When setting the delay as 5ms and the loss as 1%, the upload speed of the transmission data rate is 1.7 Mb/s and the download is 2.23 Mb/s.


```

floodlight@floodlight: ~/Desktop
File Edit View Search Terminal Help
floodlight@floodlight:~/Desktop$ cd Desktop/
bash: cd: Desktop/: No such file or directory
floodlight@floodlight:~/Desktop$ sudo python 5517a2.py
[sudo] password for floodlight:
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(2.00Mbit 5ms delay 1% loss) (2.00Mbit 5ms delay 1% loss) (64.00Mbit) (64.00Mbit)
(1000.00Mbit) (1000.00Mbit) (1.00Mbit) (1.00Mbit) (2.00Mbit) (2.00Mbit) (32.00
Mbit) (32.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (64.00Mbit) (600.00Mbit) (
600.00Mbit) *** Starting network
*** Configuring hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Starting controllers
*** Starting switches
(1.00Mbit) (2.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (600.00Mbit) (2.00Mbit
5ms delay 1% loss) (64.00Mbit) (1000.00Mbit) *** Post configure switches and ho
sts
*** Starting CLI:
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.70 Mbits/sec', '2.23 Mbits/sec']
mininet>

```

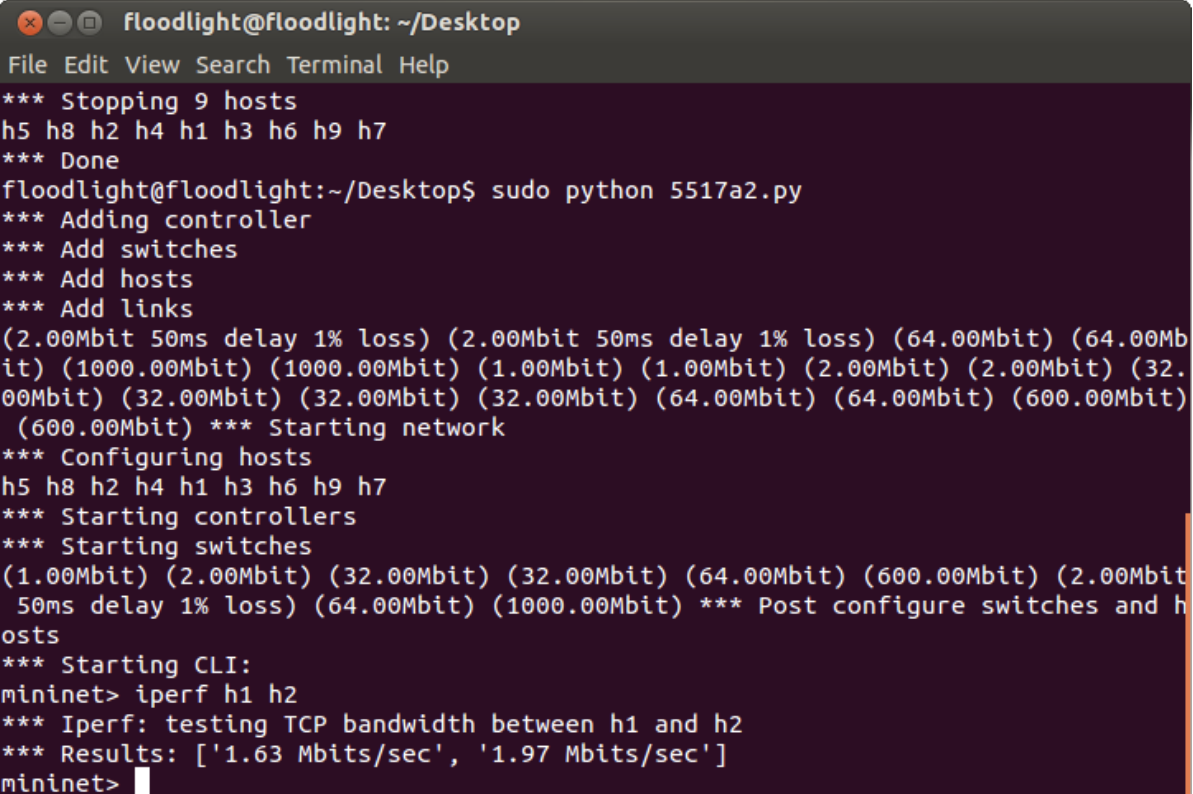
Figure 3.1 The download and upload speed of the initial data

3.2 Network Delays

3.2.1 Transmission Delay

Transmission delay could be described as the time that takes to push the bits of the packet onto the outgoing link, which is decided by the packet size and the link capacity. If the network contains l bits of the packets and c bits per second of the capacity, the transmission delay could be determined as $\frac{l}{c}$. [5]

The transmission delay was simulated in the mininet by changing the delay to 50 ms, and the packet loss rate remains the same. In this case, The transmission data rate of the upload speed is 1.63 Mb/s and the download rate is 1.97 Mb/s. Compared with the initial sample above (1.9 Mb/s for upload and 2.26 Mb/s for download), the speed decreased.



```

floodlight@floodlight: ~/Desktop
File Edit View Search Terminal Help
*** Stopping 9 hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Done
floodlight@floodlight:~/Desktop$ sudo python 5517a2.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(2.00Mbit 50ms delay 1% loss) (2.00Mbit 50ms delay 1% loss) (64.00Mbit) (64.00Mb
it) (1000.00Mbit) (1000.00Mbit) (1.00Mbit) (1.00Mbit) (2.00Mbit) (2.00Mbit) (32.
00Mbit) (32.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (64.00Mbit) (600.00Mbit)
(600.00Mbit) *** Starting network
*** Configuring hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Starting controllers
*** Starting switches
(1.00Mbit) (2.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (600.00Mbit) (2.00Mbit
50ms delay 1% loss) (64.00Mbit) (1000.00Mbit) *** Post configure switches and h
osts
*** Starting CLI:
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.63 Mbits/sec', '1.97 Mbits/sec']
mininet>

```

Figure 3.2 The download and upload speed with transmission delay

3.2.2 Propagation delay

Propagation delay is the time for a bit signal to reach the destination from another end of the link, and what determines the propagation delay is the distance between the Sender and the Receiver. If the speed of the propagation is s , the distance is l , then the propagation delay would be $\frac{l}{s}$. [5]

In the task, for measuring the propagation delay, the packet loss rate has been changed to 10% and the value of the propagation delay remained at 5ms. Applying the conditions, Transmission data rates that are received are 615Kb/s for the upload and 628Kb/s for the download. Compared to the initial data, both of the upload and download speeds have dropped significantly.

```

floodlight@floodlight: ~/Desktop
File Edit View Search Terminal Help
floodlight@floodlight:~$ cd Desktop/
floodlight@floodlight:~/Desktop$ sudo python 5517a2.py
[sudo] password for floodlight:
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(2.00Mbit 5ms delay 10% loss) (2.00Mbit 5ms delay 10% loss) (64.00Mbit) (64.00Mbit)
(1000.00Mbit) (1000.00Mbit) (1.00Mbit) (1.00Mbit) (2.00Mbit) (2.00Mbit) (32.00Mbit)
(32.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (64.00Mbit) (600.00Mbit)
(600.00Mbit) *** Starting network
*** Configuring hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Starting controllers
*** Starting switches
(1.00Mbit) (2.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (600.00Mbit) (2.00Mbit
5ms delay 10% loss) (64.00Mbit) (1000.00Mbit) *** Post configure switches and h
osts
*** Starting CLI:
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['615 Kbits/sec', '629 Kbits/sec']
mininet>

```

Figure 3.3 The download and upload speed with propagation delay

3.2.3 Queuing Delay

Queuing delay refers to the time the packet spends in the routing queue waiting for being processed, and it relies on the arrival rate of the transmission data packets, capacity of the link as well as the traffic within the network.[5]

To realize the measurement of the queuing delay, the max queue size has been altered. When the value of the max queue size is 1, the TCP bandwidths obtained show that the upload data rate is 310Kb/s and the download speed is 355Kb/s. But when the value of the max queue size increases to 100, the transmission data rates become 1.86Mb/s for upload and 2.02Mb/s for download respectively.

```

floodlight@floodlight: ~/Desktop
File Edit View Search Terminal Help
*** Stopping 9 hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Done
floodlight@floodlight:~/Desktop$ sudo python 5517a2.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(2.00Mbit 5ms delay 1% loss) (2.00Mbit 5ms delay 1% loss) (64.00Mbit) (64.00Mbit)
(1000.00Mbit) (1000.00Mbit) (1.00Mbit) (1.00Mbit) (2.00Mbit) (2.00Mbit) (32.00
Mbit) (32.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (64.00Mbit) (600.00Mbit) (
600.00Mbit) *** Starting network
*** Configuring hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Starting controllers
*** Starting switches
(1.00Mbit) (2.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (600.00Mbit) (2.00Mbit
5ms delay 1% loss) (64.00Mbit) (1000.00Mbit) *** Post configure switches and ho
sts
*** Starting CLI:
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['310 Kbits/sec', '355 Kbits/sec']
mininet>

```

Figure 3.4 The transmission data rate when the max queue size is 1

```

floodlight@floodlight: ~/Desktop
File Edit View Search Terminal Help
*** Stopping 9 hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Done
floodlight@floodlight:~/Desktop$ sudo python 5517a2.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(2.00Mbit 5ms delay 1% loss) (2.00Mbit 5ms delay 1% loss) (64.00Mbit) (64.00Mbit)
(1000.00Mbit) (1000.00Mbit) (1.00Mbit) (1.00Mbit) (2.00Mbit) (2.00Mbit) (32.00
Mbit) (32.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (64.00Mbit) (600.00Mbit) (
600.00Mbit) *** Starting network
*** Configuring hosts
h5 h8 h2 h4 h1 h3 h6 h9 h7
*** Starting controllers
*** Starting switches
(1.00Mbit) (2.00Mbit) (32.00Mbit) (32.00Mbit) (64.00Mbit) (600.00Mbit) (2.00Mbit
5ms delay 1% loss) (64.00Mbit) (1000.00Mbit) *** Post configure switches and ho
sts
*** Starting CLI:
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.86 Mbits/sec', '2.02 Mbits/sec']
mininet>

```

Figure 3.5 The transmission data rate when the max queue size is 100

3.2.4 Processing Delay

The definition of processing delay is the time required for the router to process the packet header, which is dependent on the speed of the router. With the advanced development of the technology, the processing speed of the router is much higher. Precisely, the processing delays of the route could be microseconds, even less. Therefore, this delay is ignored for the slight influence compared to other forms of the delays.

3.3 Similarity

Three of the parameters are set to check the delay influence on the read and access. To be precise, there are loss, delay and the max queue size, and each of them corresponds to one type of the delay. The delay corresponds to the transmission delay, the loss is related to the propagation delay, and the max queue size is simulated to be the queuing delay.

When the packet loss rate is reduced from the 10%-1% or the max queue size is increased from 1 to 100, both the upload bandwidth and download bandwidth will increase greatly. However, if the delay is reduced from 50ms to 1ms, both upload and download will have a slight impact.

In addition, propagation delay and queue delay will affect upload (write) and download (read) speed. Transmission delay will affect the upload (write) speed and slightly affect the download bandwidth.

Table 3.1 Testing result

Delay Type	Value	Upload Bandwidth	Download Bandwidth
Transmission delay (Delay)	1%, 5 ms	1.7 Mbits/sec	2.23 Mbits/sec
	1%, 50 ms	1.63 Mbits/sec ↓	1.97 Mbits/sec ↓
Propagation delay (Loss)	1%, 5 ms	1.7 Mbits/sec	2.23 Mbits/sec
	10%, 5ms	615 Kbits/sec ↓	629 Kbits/sec ↓
Queue delay (Max Queue size)	1	310 Kbits/sec	355 Kbits/sec
	100	1.86 Mbits/sec ↑	2.02 Mbits/sec ↑

In summary, the delay could be measured by the network tools such as ping testing and traceroute, determining the time required for a given network packet to travel from source to destination and back and forth. As the network becomes more popular, the information of multiple sources, such as the mobile, downloads and so on, would be transmitted together. During the progress, the multiple information and data would be mixed and separated over and over. If the delay could not be kept minimum, the loads of the network would be increased as well as the chance of congestion, which would lead to the loss of the information and data packets. Hence, understanding the time spent over the transmission from one end to another as well as where the delay would occur becomes more significant than ever to fit the growing speed and capacity of the network.

4. Task 3

This section investigated the basic functions of the ONOS controller and hopefully demonstrated how the control plane works.

4.1 Implementation the Topology in the ONOS

At this stage, the structure topology setting (part 1) did not change. The ONOS controller was linked to every single switch.

By observing the ip address of controller (there are three by default, in this case the first one is selected)



Figure 4.1 controller address

It is 172.17.0.5. Hence by typing

sudo mn --custom task3.py --topo mytopo --controller=remote,ip= 172.17.0.5

into the terminal prompt, the result shown in below. (The python code will attach to the zip file)

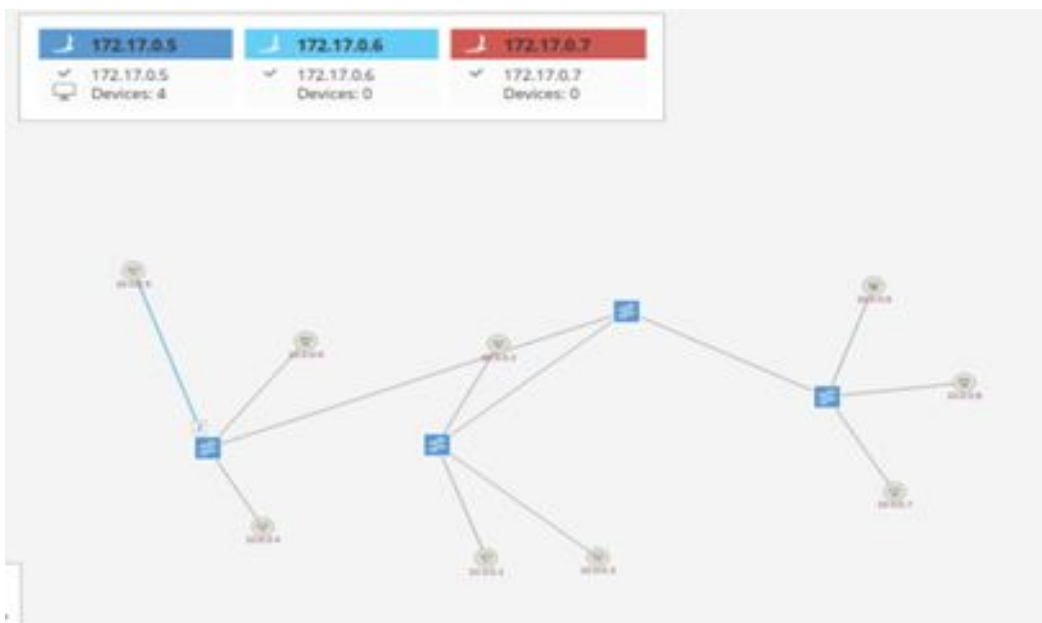


Figure 4.2 Topology on ONOS.

From left to right, the four switches stand for the telecommunication Co. (denoted as s2), the bank Co.(s3) and Internet service provider(s1) and the media (s4). Using pingall to test the network topology.


```

*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9
h2 -> h1 h3 h4 h5 h6 h7 h8 h9
h3 -> h1 h2 h4 h5 h6 h7 h8 h9
h4 -> h1 h2 h3 h5 h6 h7 h8 h9
h5 -> h1 h2 h3 h4 h6 h7 h8 h9
h6 -> h1 h2 h3 h4 h5 h7 h8 h9
h7 -> h1 h2 h3 h4 h5 h6 h8 h9
h8 -> h1 h2 h3 h4 h5 h6 h7 h9
h9 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
mininet>

```

Figure 4.3 Testing result

4.2 Isolating the Media Switch From the Network

For analysis purposes, the media switch has been disconnected from the topology. By observing the device id from the controller

Devices (4 total)

	FRIENDLY NAME	DEVICE ID	MASTER	PORTS	VENDOR	H/W VERSION	S/W VERSION	PROTOCOL
✓	of:0000000000000001	of:0000000000000001	172.17.0.5					
✓	of:0000000000000002	of:0000000000000002	172.17.0.5					
✓	of:0000000000000003	of:0000000000000003	172.17.0.5					
✓	of:0000000000000004	of:0000000000000004	172.17.0.5					

of:0000000000000004	
URI : of:0000000000000004	H/W Version : Open vSwitch
Type : SWITCH	S/W Version : 2.5.5
Master ID : 172.17.0.5	Protocol : OF_13
Chassis ID : 4	Serial # : None
Vendor : Nicira, Inc.	Pipeconf : none

Ports

Enabled	ID	Speed	Type	Egress Links	Name
false	Local	0	Copper		s4
true	1	10000	Copper	02:AA:DD:A9:8C:8D/None	s4-eth1
true	2	10000	Copper	DE:97:3F:C6:01:EC/None	s4-eth2
true	3	10000	Copper	26:0E:7D:BA:03:B7/None	s4-eth3
true	4	10000	Copper	of:0000000000000001/3	s4-eth4

Figure 4.4 IP address of bank switch

In this case, the device id for the switch is:0000000000000004. Hence by typing flows in ONOS CLI command prompt.



Figure 4.5 ONOS CLI

Checking deviceId= of:0000000000000004

```

deviceId=of:0000000000000004, flowRuleCount=4
  id=1000011326ce6, state=ADDED, bytes=0, packets=0, duration=195, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000261e0911, state=ADDED, bytes=4992, packets=64, duration=195, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH_TYPE:lldp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000687421fe, state=ADDED, bytes=4992, packets=64, duration=195, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000810531f2, state=ADDED, bytes=0, packets=0, duration=82, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
0005> ^C0005>

```

Figure 4.6 ONOS flow table

Observing the last flow on this device has lowest priority and the Ethernet type is ipv4. If this flow gets deleted from the flow table, the switch will not function normally since the network layer for this device would not recognize the IP address anymore.

Open the ONOS REST end API. By typing flow id (in this case is 0x10000810531f2 in hexadecimal and 281477141311086 in decimal) and device id (in this case is of:0000000000000004) into the FLOW end API – DELETE option:

POST /flows

Creates new flow rules

DELETE /flows/{deviceId}/{flowId}

Removes flow rule

Implementation Notes

Removes the specified flow rule.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
deviceId	of:0000000000000004	device identifier	path	string
flowId	281477141311086	flow rule identifier	path	double

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

Try it out!

Hide Response

Curl

Figure 4.7 flow end API/delete

The ipv4 is deleted. To confirm this, type flow again in the ONOS CLI command prompt.


```

deviceId=of:000000000000000004, flowRuleCount=3
  id=1000011326ce6, state=ADDED, bytes=0, packets=0, duration=395, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000261e0911, state=ADDED, bytes=9984, packets=128, duration=395, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH TYPE:lldp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000687421fe, state=ADDED, bytes=9984, packets=128, duration=395, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, payload=null, selector=[ETH TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}

```

Figure 4.8 updated flow table

Since there are three hosts connected to deceive of:000000000000000004

10.0.0.7	02:53:90:AF:CC:5A/None	02:53:90:AF:CC:5A	None	false	10.0.0.7	of:000000000000000004/1
10.0.0.9	7A:FB:69:CE:D8:5D/None	7A:FB:69:CE:D8:5D	None	false	10.0.0.9	of:000000000000000004/3
10.0.0.4	86:A5:46:B7:63:16/None	86:A5:46:B7:63:16	None	false	10.0.0.4	of:000000000000000004/3/1
10.0.0.8	C6:8A:2B:D6:33:F5/None	C6:8A:2B:D6:33:F5	None	false	10.0.0.8	of:000000000000000004/2

Figure 4.9 three hosts (the first one, the second one and the last one)

By plugging the MAC address of every pair of hosts,

POST /intents Submits a new intent

Implementation Notes
Creates and submits intent from the JSON request.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
stream	<pre> { "type": "HostToHostIntent", "appId": "org.onosproject.ovsdb", "priority": 55, "one": "02:53:90:AF:CC:5A/-1", "two": "7A:FB:69:CE:D8:5D/-1" } </pre>	input JSON	body	Model Example Value

Parameter content type: application/json

```

{
  "type": "HostToHostIntent",
  "appId": "org.onosproject.ovsdb",
  "priority": 55,
  "one": "46:E4:3C:A4:17:C8/-1",
  "two": "08:00:27:56:8a:15/-1"
}

```

Figure 4.10 host end API/post intents

the communication between each two hosts is established.

```

*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 X X X
h2 -> h1 h3 h4 h5 h6 X X X
h3 -> h1 h2 h4 h5 h6 X X X
h4 -> h1 h2 h3 h5 h6 X X X
h5 -> h1 h2 h3 h4 h6 X X X
h6 -> h1 h2 h3 h4 h5 X X X
h7 -> X X X X X X h8 h9
h8 -> X X X X X X h7 h9
h9 -> X X X X X X h7 h8
*** Results: 50% dropped (36/72 received)
mininet>

```

Figure 4.11 testing result

4.3 Get the statistical data

By plugging the device ID into statistics API



Figure 4.12 statistics API/flow table

The JSON file is generated. The result will be attached with this report.
Here is the crucial test results:

```

[{"tableId": "0", "deviceId": "of:0000000000000001", "activeEntries": 4, "packetsLookedUp": 212, "packetsMatched": 172},
{"tableId": "0", "deviceId": "of:0000000000000002", "activeEntries": 4, "packetsLookedUp": 225, "packetsMatched": 204},
{"tableId": "0", "deviceId": "of:0000000000000003", "activeEntries": 4, "packetsLookedUp": 209, "packetsMatched": 192},
{"tableId": "0", "deviceId": "of:0000000000000004", "activeEntries": 9, "packetsLookedUp": 118, "packetsMatched": 57},

```

Figure 4.13 Crucial test results

4.4 Add a New Host on S2

Using REST API- HOST end API – Post,

POST /hosts Creates a new host based on JSON input and adds it to the current host inventory

Parameter	Value	Description	Parameter Type	Data Type
stream	<pre>{ "vlan": "-1", "ipAddresses": ["127.0.0.1"], "locations": [{ "elementId": "of:0000000000000002", "port": "a" }] }</pre>	input JSON	body	Model

Parameter content type:

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

[Try it out!](#) [Hide Response](#)

Figure 4.14 Host end API/ post

Since the mac address is randomly generated,

stream

```
{
  "mac": "46:E4:3C:A4:17:C8",
  "vlan": "-1",
  "ipAddresses": [
    "127.0.0.1"
  ],
  "locations": [
    {
      "elementId": "of:0000000000000002",
      "port": "a"
    }
  ]
}
```

Parameter content type:

Figure 4.15 JSON file

the new host is able to add on s2

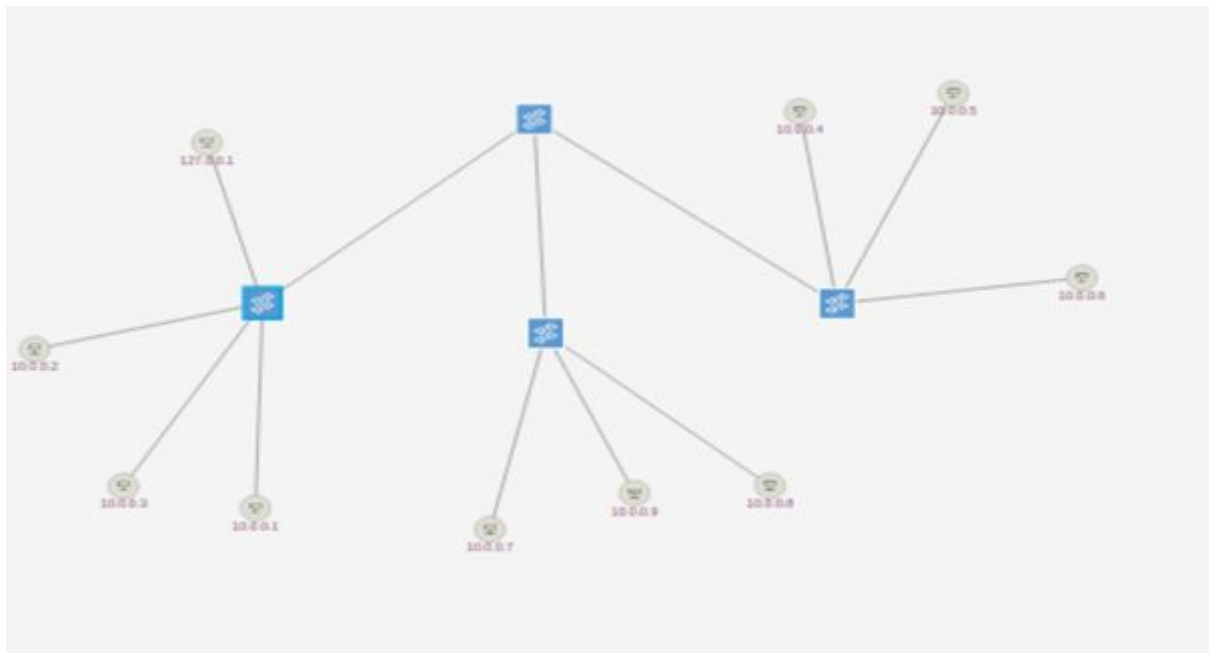


Figure 4.16 testing result

Reference

- [1] Joseph, V., & Chapman, B. (2009). Deploying QoS for Cisco IP and next generation networks: the definitive guide. Morgan Kaufmann. (Chapter - 3)
- [2] Medhi, D., & Ramasamy, K. (2017). Network routing: algorithms, protocols, and architectures. Morgan Kaufmann. (Chapter - 26)
- [3] <https://www.sciencedirect.com/topics/computer-science/bandwidth-requirement>
- [4] <https://www.wisegeek.com/what-is-a-network-delay.htm>
- [5] <https://www.educative.io/edpresso/what-are-the-different-types-of-network-delay>