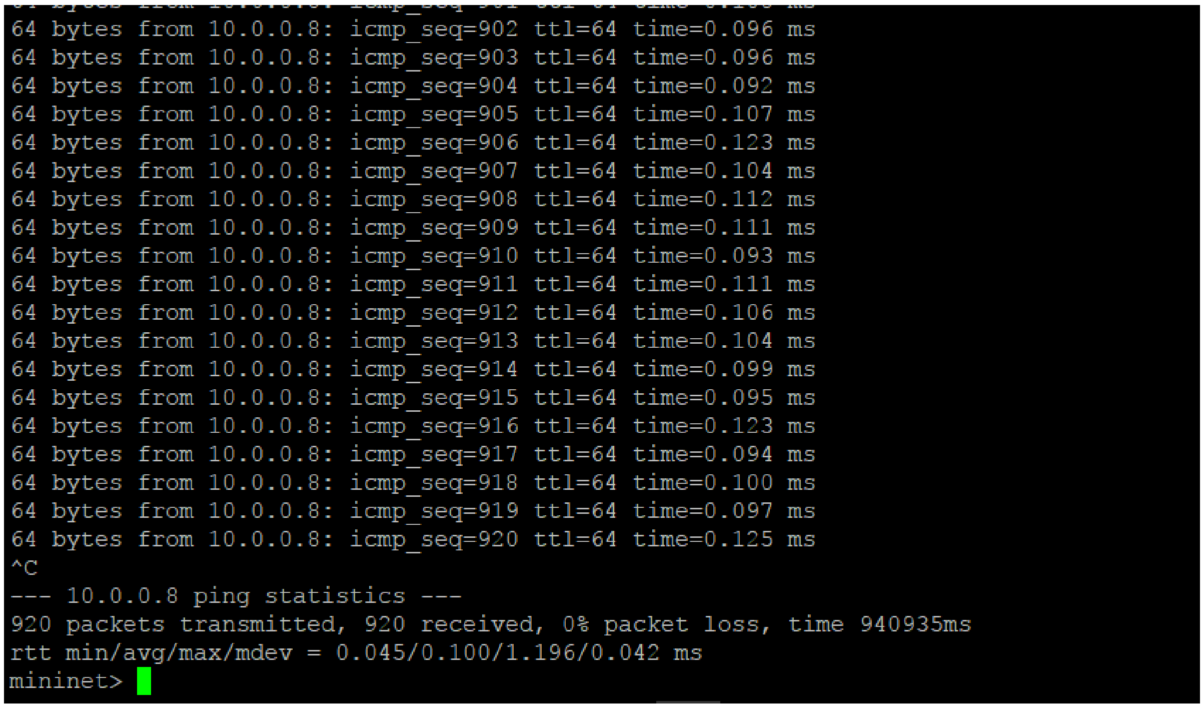
Assignment 2 –
Exploring Software-Defined Networking Techniques

Name : Kunal Shira
Bid : B00637094
e-mail : kshira1@binghamton.edu
October 20, 2017

Instructor
Prof. Hui Lu

Task 1 –

H1 ping h8



```
64 bytes from 10.0.0.8: icmp_seq=902 ttl=64 time=0.096 ms
64 bytes from 10.0.0.8: icmp_seq=903 ttl=64 time=0.096 ms
64 bytes from 10.0.0.8: icmp_seq=904 ttl=64 time=0.092 ms
64 bytes from 10.0.0.8: icmp_seq=905 ttl=64 time=0.107 ms
64 bytes from 10.0.0.8: icmp_seq=906 ttl=64 time=0.123 ms
64 bytes from 10.0.0.8: icmp_seq=907 ttl=64 time=0.104 ms
64 bytes from 10.0.0.8: icmp_seq=908 ttl=64 time=0.112 ms
64 bytes from 10.0.0.8: icmp_seq=909 ttl=64 time=0.111 ms
64 bytes from 10.0.0.8: icmp_seq=910 ttl=64 time=0.093 ms
64 bytes from 10.0.0.8: icmp_seq=911 ttl=64 time=0.111 ms
64 bytes from 10.0.0.8: icmp_seq=912 ttl=64 time=0.106 ms
64 bytes from 10.0.0.8: icmp_seq=913 ttl=64 time=0.104 ms
64 bytes from 10.0.0.8: icmp_seq=914 ttl=64 time=0.099 ms
64 bytes from 10.0.0.8: icmp_seq=915 ttl=64 time=0.095 ms
64 bytes from 10.0.0.8: icmp_seq=916 ttl=64 time=0.123 ms
64 bytes from 10.0.0.8: icmp_seq=917 ttl=64 time=0.094 ms
64 bytes from 10.0.0.8: icmp_seq=918 ttl=64 time=0.100 ms
64 bytes from 10.0.0.8: icmp_seq=919 ttl=64 time=0.097 ms
64 bytes from 10.0.0.8: icmp_seq=920 ttl=64 time=0.125 ms
^C
--- 10.0.0.8 ping statistics ---
920 packets transmitted, 920 received, 0% packet loss, time 940935ms
rtt min/avg/max/mdev = 0.045/0.100/1.196/0.042 ms
mininet>
```

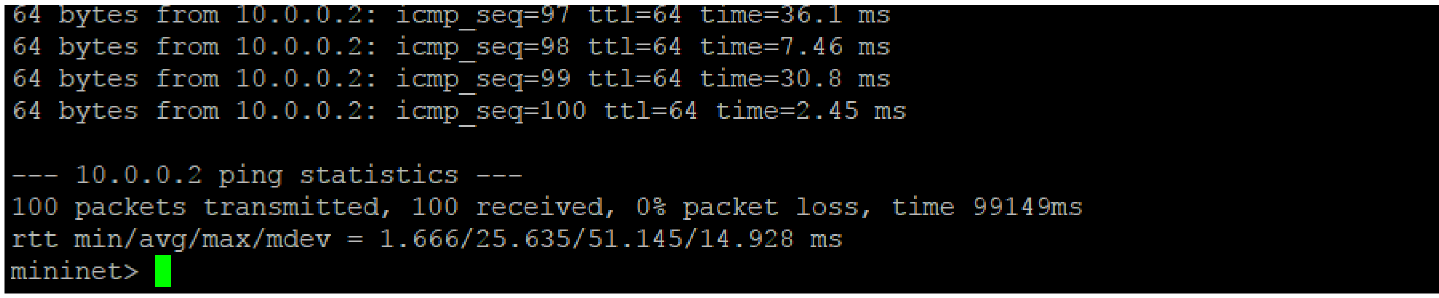
Task 2 –

Q1.

Sequence	Methos
1	_handle_PacketIn (self, event)
2	act_like_hub (self, packet, packet_in) or act_like_switch (self, packet, packet_in)
3	resend_packet (self, packet_in, out_port)

Q2.

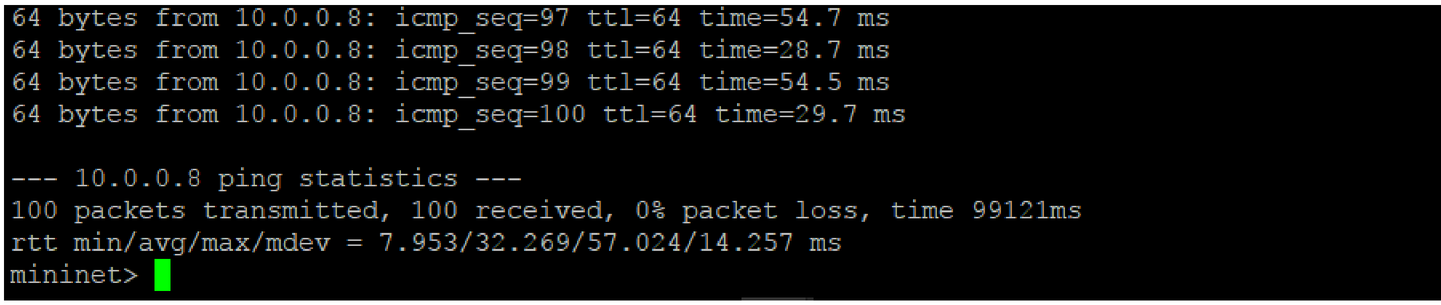
h1 ping -c100 h2 Average = 25.635



```
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=36.1 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=7.46 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=30.8 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=2.45 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99149ms
rtt min/avg/max/mdev = 1.666/25.635/51.145/14.928 ms
mininet> 
```

h1 ping -c100 h8 Average = 32.269



```
64 bytes from 10.0.0.8: icmp_seq=97 ttl=64 time=54.7 ms
64 bytes from 10.0.0.8: icmp_seq=98 ttl=64 time=28.7 ms
64 bytes from 10.0.0.8: icmp_seq=99 ttl=64 time=54.5 ms
64 bytes from 10.0.0.8: icmp_seq=100 ttl=64 time=29.7 ms

--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99121ms
rtt min/avg/max/mdev = 7.953/32.269/57.024/14.257 ms
mininet> 
```

As when h1 pins to h2 the is only one switch s1 lies between then whereas when h1 pings to h8 it has to go through 5 switches hence, time taken to pings from h1-h8 is significantly more.

Q3.

iperf: general command-line utility for testing the speed of a single TCP connection.

In this case iperf is Transfer Control Protocol test between two (optionally specified) hosts.

Running an iperf gives parsed the bandwidth achieved between two hosts in terms of Mbits/sec.

```
iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['6.94 Mbits/sec', '8.68 Mbits/sec']
```

```
 iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.95 Mbits/sec', '4.97 Mbits/sec']
```

Q4.

All the switches s1,s2,s3,s4,s5,s6,s7 face same traffic.

In _handle_PacketIn incrementing value of a variable if switch id present else setting starting value event.dpid

Task 3 –

Q1.

New packet is compared with pair of mapping of mac and port no. If no match found then establishes that packet with the port no. Packet destination is again matched if match found then send that packet directly to packet destination on it’s port no.

Q2.

Ping from h1 to h2 - Average = 24.677

```
100 packets transmitted, 100 received, 0% packet loss, time 99149ms

rtt min/avg/max/mdev = 1.626/24.677/63.000/14.630 ms
```

Ping from h1 to h8 - Average = 32.95

```
100 packets transmitted, 100 received, 0% packet loss, time 99144ms

rtt min/avg/max/mdev = 6.922/32.095/55.329/15.105 ms
```

Instead of flooding due to entry in the table sent on the paired port no. Due to which performance improved by about 3 percent as compared to previous act like hub

Q3.

```
mininet> iperf h1 h2

*** Iperf: testing TCP bandwidth between h1 and h2

*** Results: ['6.30 Mbits/sec', '7.62 Mbits/sec']
```

```
mininet> iperf h1 h8

*** Iperf: testing TCP bandwidth between h1 and h8

*** Results: ['3.25 Mbits/sec', '3.95 Mbits/sec']
```

Instead of flooding due to entry in the table sent on the paired port no. Due to which performance improved by about 3 percent as compared to previous case act like hub

Task 4 –

Q1.

Ping from h1 to h2 - Average = 0.071

```
100 packets transmitted, 100 received, 0% packet loss, time 110149ms
rtt min/avg/max/mdev = 0.061/0.071/0.326/0.027 ms
mininet>
```

Ping from h1 to h8 - Average = 0.061

```
100 packets transmitted, 100 received, 0% packet loss, time 110174ms
rtt min/avg/max/mdev = 0.057/0.061/0.358/0.031 ms
```

Performance improved by about 500 times as compared to previous case act like switch i.e. task 3

Q2.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['26.6 Gbits/sec', '26.6 Gbits/sec']
```

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['19.8 Gbits/sec', '19.8 Gbits/sec']
```

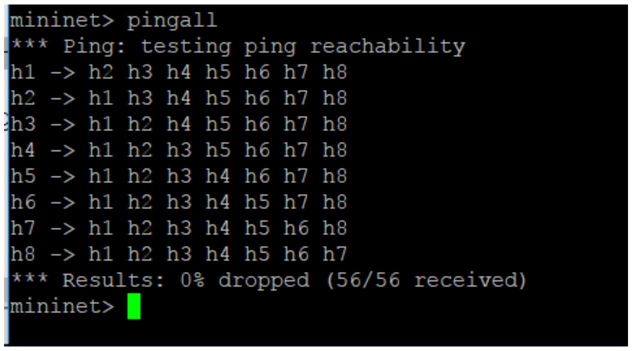
Performance improved extremely as compared to previous case act like switch i.e. task 3

Q3.

Whenever first packet is sent from source to destination it is received by switch and first time forwarded to controller and controller makes entry in flow table. After that step every other packet checked in the flow table instead of going to controller sent directly to the destination.

Q4.

Running pingall to verify connectivity and dump the output.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> 
```

Q5.

Lets say we are dumping output for switch s1

ovs-ofctl dump-flows s1

There are about 8 entries for each switch as there are 8 host present in the network. Entry shows the flow from source to destination. For first packet only the flow entry is made in flow table by controller. Then next packet comes does not go to controller it gets through switch only that's why it's fast.

Meaning of Flow entry:

Duration	-	Flow time
Table	-	id of Open switch table used
N_packets	-	No. of packets flowing
N_bytes	-	No. of bytes flowing
Idle_age	-	How long flow has not matched any packets
Dl_dst	-	MAC address of destination
Actions	-	refers to port where packet will flow