

## Task 1:

wireshark on sw1

**mininet> h1 ping -c 4 h2**

PING 100.0.0.11 (100.0.0.11) 56(84) bytes of data.

64 bytes from 100.0.0.11: icmp\_seq=1 ttl=64 time=70.5 ms

64 bytes from 100.0.0.11: icmp\_seq=2 ttl=64 time=0.512 ms

64 bytes from 100.0.0.11: icmp\_seq=3 ttl=64 time=0.120 ms

64 bytes from 100.0.0.11: icmp\_seq=4 ttl=64 time=0.115 ms

--- 100.0.0.11 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3005ms

rtt min/avg/max/mdev = 0.115/17.834/70.589/30.458 ms

Open flow table of sw1:

**command: sudo ovs-ofctl dump-flows s1**

NXST\_FLOW reply (xid=0x4):

cookie=0x0, duration=5.507s, table=0, n\_packets=1, n\_bytes=42, idle\_timeout=10,

hard\_timeout=30, idle\_age=5,

priority=65535,arp,in\_port=2,vlan\_tci=0x0000,dl\_src=02:a2:6d:44:ae:31,dl\_dst=0a:91:23:38:06:bd,

arp\_spa=100.0.0.11,arp\_tpa=100.0.0.10,arp\_op=1 actions=output:1

cookie=0x0, duration=5.504s, table=0, n\_packets=1, n\_bytes=42, idle\_timeout=10,

hard\_timeout=30, idle\_age=5,

priority=65535,arp,in\_port=1,vlan\_tci=0x0000,dl\_src=0a:91:23:38:06:bd,dl\_dst=02:a2:6d:44:ae:31,

arp\_spa=100.0.0.10,arp\_tpa=100.0.0.11,arp\_op=2 actions=output:2

cookie=0x0, duration=10.56s, table=0, n\_packets=4, n\_bytes=392, idle\_timeout=10,

hard\_timeout=30, idle\_age=7,

priority=65535,icmp,in\_port=1,vlan\_tci=0x0000,dl\_src=0a:91:23:38:06:bd,dl\_dst=02:a2:6d:44:ae:31,nw\_src=100.0.0.10,nw\_dst=100.0.0.11,nw\_tos=0,icmp\_type=8,icmp\_code=0 actions=output:2

cookie=0x0, duration=10.555s, table=0, n\_packets=4, n\_bytes=392, idle\_timeout=10,

hard\_timeout=30, idle\_age=7,

priority=65535,icmp,in\_port=2,vlan\_tci=0x0000,dl\_src=02:a2:6d:44:ae:31,dl\_dst=0a:91:23:38:06:bd,nw\_src=100.0.0.11,nw\_dst=100.0.0.10,nw\_tos=0,icmp\_type=0,icmp\_code=0 actions=output:1

Wireshark:

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help										
Filter: <input type="text"/> Expression... Clear Apply Save										
No.	Time	Destination	Protocol	Length	Sender IP address	Target IP address	Interface	Type	Info	
1	0.000000000	Broadcast	ARP	42	100.0.0.10	100.0.0.11	0		Who has 100.0.0.11? Tell 100.0.0.10	
2	0.058212000	0a:91:23:38:06:bd	ARP	42	100.0.0.11	100.0.0.10	0		100.0.0.11 is at 02:a2:6d:44:ae:31	
3	0.058242000	100.0.0.11	ICMP	98			0	8	Echo (ping) request id=0x0bab, seq=1/256, ttl=64 (req	
4	0.070505000	100.0.0.10	ICMP	98			0	0	Echo (ping) reply id=0x0bab, seq=1/256, ttl=64 (req	
5	1.001615000	100.0.0.11	ICMP	98			0	8	Echo (ping) request id=0x0bab, seq=2/512, ttl=64 (req	
6	1.002072000	100.0.0.10	ICMP	98			0	0	Echo (ping) reply id=0x0bab, seq=2/512, ttl=64 (req	
7	2.003594000	100.0.0.11	ICMP	98			0	8	Echo (ping) request id=0x0bab, seq=3/768, ttl=64 (req	
8	2.003659000	100.0.0.10	ICMP	98			0	0	Echo (ping) reply id=0x0bab, seq=3/768, ttl=64 (req	
9	3.005363000	100.0.0.11	ICMP	98			0	8	Echo (ping) request id=0x0bab, seq=4/1024, ttl=64 (req	
10	3.005426000	100.0.0.10	ICMP	98			0	0	Echo (ping) reply id=0x0bab, seq=4/1024, ttl=64 (re	
11	5.008096000	0a:91:23:38:06:bd	ARP	42	100.0.0.11	100.0.0.10	0		Who has 100.0.0.10? Tell 100.0.0.11	
12	5.008138000	02:a2:6d:44:ae:31	ARP	42	100.0.0.10	100.0.0.11	0		100.0.0.10 is at 0a:91:23:38:06:bd	

Ethernet II, Src: 0a:91:23:38:06:bd (0a:91:23:38:06:bd), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

0000 ff ff ff ff ff 0a 91 23 38 06 bd 00 00 01 ..... #8.....  
0010 00 00 00 04 00 01 0a 91 23 38 06 bd 64 00 0a ..... #8..d...  
0020 00 00 00 00 00 00 00 00 00 00 .....d. .

File: "/home/click/test1 wk... Packets: 12 ... Profile: Default

Series of events:

1. h1 does ARP req for h2's mac addr.
2. SW1 floods this, while learning h1's mac to port mapping.
3. h2 does ARP reply
4. SW1 delivers this directly to h1 and learns h2's port to mac info
5. h1 sends the ICMP Ping packet which is delivered by the learning switch to h2, and then a PING reply is sent back.

.....

## **Task 2:**

Topology built, every device is a learning switch.

**>pingall**

Succeeded for all hosts.

Now firewalls enabled

**mininet> h1 ping -c 4 h3**

**PING 10.0.0.50 (10.0.0.50) 56(84) bytes of data.**

**--- 10.0.0.50 ping statistics ---**

**4 packets transmitted, 0 received, 100% packet loss, time 3005ms**

h1 unable to reach h3 since its the private zone!

**mininet> h3 ping -c 4 h1**

**PING 100.0.0.10 (100.0.0.10) 56(84) bytes of data.**

**64 bytes from 100.0.0.10: icmp\_seq=1 ttl=64 time=84.2 ms**

**64 bytes from 100.0.0.10: icmp\_seq=2 ttl=64 time=2.07 ms**

**64 bytes from 100.0.0.10: icmp\_seq=3 ttl=64 time=1.16 ms**

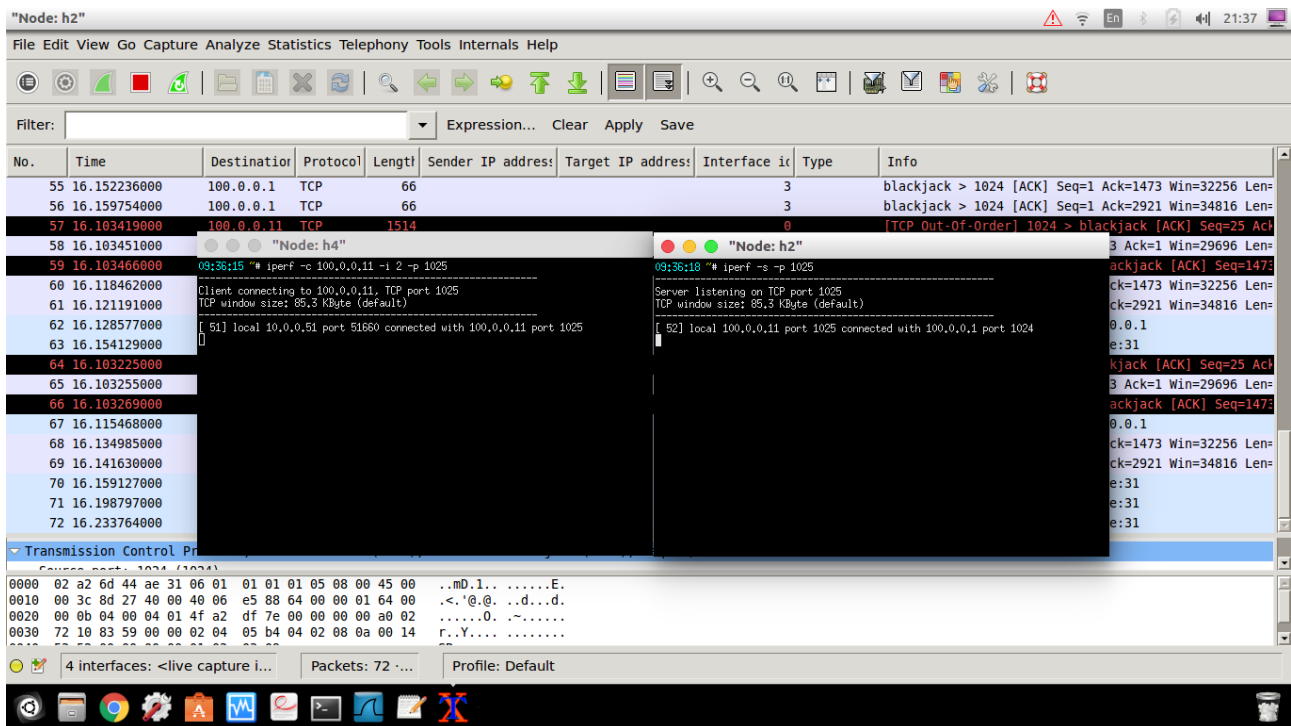
**64 bytes from 100.0.0.10: icmp\_seq=4 ttl=64 time=1.13 ms**

**--- 100.0.0.10 ping statistics ---**

**4 packets transmitted, 4 received, 0% packet loss, time 3025ms**

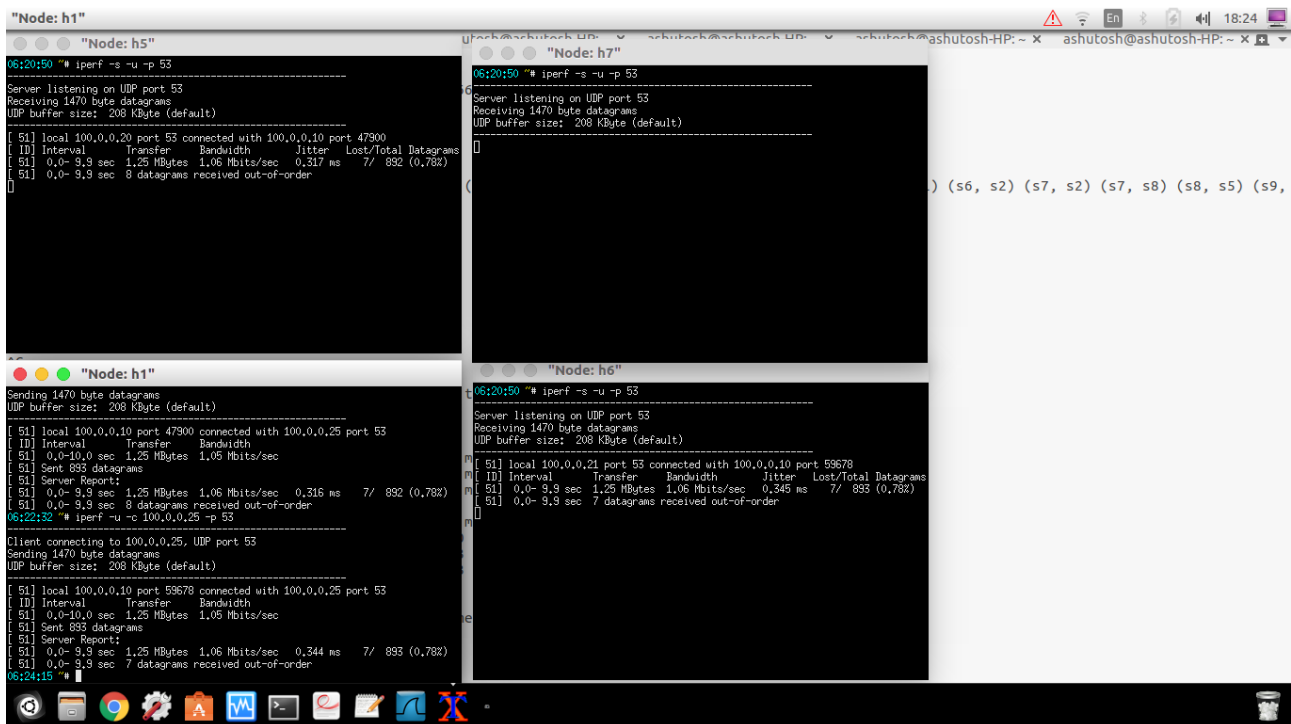
**rtt min/avg/max/mdev = 1.138/22.147/84.211/35.834 ms**

h3 was able to easily reach h1!



Here, Starting server at h2 port 1025 and client at h2:  
Connection was possible. However the other way round, communication was not possible.

### Task 3:



Here we can see **load balancer** in action.

H1 is udp client, where as the h5, h6 and h7 are udp servers.

The image displays four terminal windows on a Linux desktop, each showing a different node's activity. The windows are titled "Node: h1", "Node: h10", "Node: h1", and "Node: h9". Each window shows a server listening on TCP port 80 and a client connection from a specific IP address. The connections are timestamped and show details like transfer size and bandwidth.

**Node: h1 (Top Left):**

```
06:53:20 * iperf -s -p 80
-----
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ 52] local 100.0.0.40 port 80 connected with 100.0.0.10 port 47916
[ 10] Interval Transfer Bandwidth
[ 52] 0.0-11.1 sec 3.00 MBytes 2.27 Mbits/sec
```

**Node: h10 (Top Right):**

```
06:53:20 * iperf -s -p 80
-----
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ 52] local 100.0.0.40 port 80 connected with 100.0.0.10 port 47916
[ 10] Interval Transfer Bandwidth
[ 52] 0.0-11.1 sec 3.00 MBytes 2.27 Mbits/sec
```

**Node: h1 (Bottom Left):**

```
06:55:04 * iperf -c 100.0.0.45 -p 80
-----
Client connecting to 100.0.0.45, TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ 51] local 100.0.0.10 port 47916 connected with 100.0.0.45 port 80
[ 10] Interval Transfer Bandwidth
[ 51] 0.0-10.7 sec 3.00 MBytes 2.36 Mbits/sec
06:55:20 * iperf -c 100.0.0.45 -p 80
-----
Client connecting to 100.0.0.45, TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ 51] local 100.0.0.10 port 47917 connected with 100.0.0.45 port 80
[ 10] Interval Transfer Bandwidth
[ 51] 0.0-10.3 sec 2.62 MBytes 2.14 Mbits/sec
06:55:47 ~ [ ]
```

**Node: h9 (Bottom Right):**

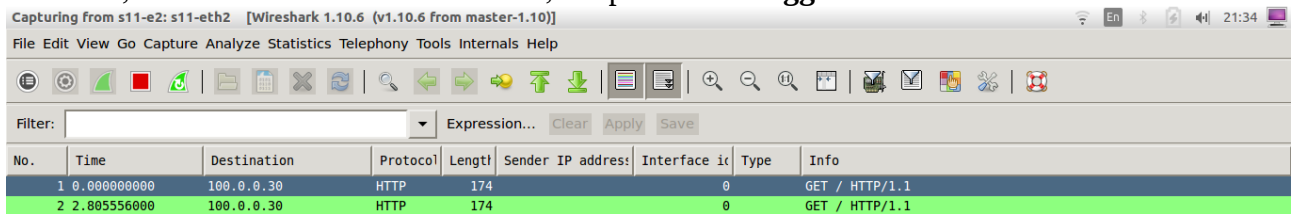
```
06:55:20 * iperf -s -p 80
-----
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ 52] local 100.0.0.41 port 80 connected with 100.0.0.10 port 47917
[ 10] Interval Transfer Bandwidth
[ 52] 0.0-10.5 sec 2.62 MBytes 2.09 Mbits/sec
```

.....

The image shows a Linux desktop environment with two terminal windows open. The left window, titled "Node: h10", displays the output of the command `iperf -s -p 80`. It shows the server listening on TCP port 80, the TCP window size, and a connection from 100.0.0.10. The right window, titled "Node: h1", displays the output of the command `wget --method=POST 100.0.0.45`. It shows the connection to 100.0.0.45:80 and the sending of an HTTP request. The desktop has a dark theme and a dock with various application icons at the bottom.

In the above screenshot you can see that h1 was able to send a POST method to the server through the ids.

However, when the GET method was sent, the packet was **logged**.

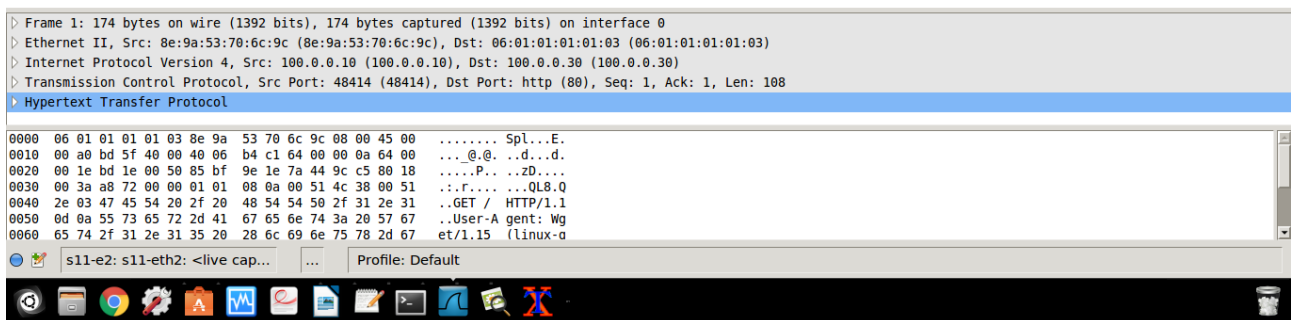


Capturing from s11-e2: s11-eth2 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Destination	Protocol	Length	Sender IP address	Interface	Type	Info
1	0.000000000	100.0.0.30	HTTP	174		0	GET / HTTP/1.1	
2	2.805556000	100.0.0.30	HTTP	174		0	GET / HTTP/1.1	



Frame 1: 174 bytes on wire (1392 bits), 174 bytes captured (1392 bits) on interface 0

Ethernet II, Src: 8e:9a:53:70:6c:9c (8e:9a:53:70:6c:9c), Dst: 06:01:01:01:01:03 (06:01:01:01:01:03)

Internet Protocol Version 4, Src: 100.0.0.10 (100.0.0.10), Dst: 100.0.0.30 (100.0.0.30)

Transmission Control Protocol, Src Port: 48414 (48414), Dst Port: http (80), Seq: 1, Ack: 1, Len: 108

Hypertext Transfer Protocol

0000	06 01 01 01 01 03 8e 9a 53 70 6c 9c 08 00 45 00	..... Spl...E.
0010	00 a0 bd 5f 40 00 40 06 b4 c1 64 00 00 0a 64 00	...@. ...d...
0020	00 1e bd 1e 00 50 85 bf 9e 1e 7a 44 9c c5 80 18	...P... zD...
0030	00 3a a8 72 00 00 01 01 08 0a 00 51 4c 38 00 51	..f.... ..QL8.Q
0040	2e 03 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31	..GET / HTTP/1.1
0050	0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 57 67	..User-A gent: Wg
0060	65 74 2f 31 2e 31 35 20 28 6c 69 6e 75 78 2d 67	et/1.15 (linux-d

s11-e2: s11-eth2: <live cap... Profile: Default

This the wireshark capture at the interface connected to the logging server.

Now, I sent an http **PUT** method packet with **DELETE** written in it.

Again, wireshark capture at the interface connected to the logging server shows that this packet was logged.

*Thus, the above two examples show that IDS is checking the HTTP method type as well as content in the case of PUT.*

#### Task 4:

Report files with counter and rate values is present in the tar ball with names ids\_test.report, napt\_test.report, lb\_test.report