**P4 PROGRAM EXERCISES**

**TESTING OUTPUT:**

Contents

[Q1) BASIC IPV4 FORWARDING 3](#_heading=h.gjdgxs)

[OUTPUT: 3](#_heading=h.30j0zll)

[Q2) BASIC TUNNELING 5](#_heading=h.1fob9te)

[OUTPUT: 5](#_heading=h.3znysh7)

[Q3) CALCULATOR 8](#_heading=h.2et92p0)

[OUTPUT: 8](#_heading=h.3dy6vkm)

[Q4) ECN (EXPLICIT CONGESTION NOTIFICATION) 9](#_heading=h.1t3h5sf)

[OUTPUT: 9](#_heading=h.4d34og8)

[COMMANDS: 10](#_heading=h.2s8eyo1)

[Q5) FIREWALL 11](#_heading=h.17dp8vu)

[OUTPUT: 11](#_heading=h.3rdcrjn)

[COMMANDS: 11](#_heading=h.26in1rg)

[Q6) LINK MONITOR 12](#_heading=h.lnxbz9)

[OUTPUT: 12](#_heading=h.35nkun2)

[COMMANDS: 13](#_heading=h.1ksv4uv)

[Q7) LOAD BALANCER 14](#_heading=h.44sinio)

[OUTPUT: 14](#_heading=h.2jxsxqh)

[COMMANDS: 16](#_heading=h.z337ya)

[Q8) MRI (Multi-Hop Route Inspection) 17](#_heading=h.3j2qqm3)

[OUTPUT: 17](#_heading=h.1y810tw)

[COMMANDS: 18](#_heading=h.4i7ojhp)

[Q9) MULTICAST 19](#_heading=h.2xcytpi)

[OUTPUT: 19](#_heading=h.1ci93xb)

[COMMANDS: 19](#_heading=h.3whwml4)

[Q10) QOS (QUALITY OF SERVICE) 20](#_heading=h.2bn6wsx)

[OUTPUT: 20](#_heading=h.qsh70q)

[COMMANDS: 22](#_heading=h.3as4poj)

[Q11) REFLECTOR 23](#_heading=h.1pxezwc)

[OUTPUT: 23](#_heading=h.49x2ik5)

[COMMANDS: 24](#_heading=h.2p2csry)

[Q12) REPEATER 25](#_heading=h.147n2zr)

[OUTPUT: 25](#_heading=h.3o7alnk)

[Q13) VERIFYING CHECKSUM 28](#_heading=h.23ckvvd)

[OUTPUT: 28](#_heading=h.ihv636)

[COMMANDS: 29](#_heading=h.32hioqz)

[Q14) METER 31](#_heading=h.1hmsyys)

[OUTPUT: 31](#_heading=h.41mghml)

[COMMANDS: 31](#_heading=h.2grqrue)

[Q15) COUNTER 33](#_heading=h.vx1227)

[Q16) SOURCE ROUTING 34](#_heading=h.3fwokq0)

[OUTPUT: 34](#_heading=h.1v1yuxt)

[COMMANDS: 35](#_heading=h.4f1mdlm)

[Q17) IP FORWARDING 36](#_heading=h.2u6wntf)

[OUTPUT: 36](#_heading=h.19c6y18)

[COMMANDS: 36](#_heading=h.3tbugp1)

[Q18) DIGEST MESSAGES 37](#_heading=h.28h4qwu)

[COMMANDS: 37](#_heading=h.nmf14n)

[Q19) HEAVY HITTER 38](#_heading=h.37m2jsg)

[OUTPUT: 38](#_heading=h.1mrcu09)

[COMMANDS: 39](#_heading=h.46r0co2)

[Q20) IP FORWARDING 2 TABLES 40](#_heading=h.2lwamvv)

[OUTPUT: 40](#_heading=h.111kx3o)

[COMMANDS: 41](#_heading=h.3l18frh)

[Q21) STATEFULL FIREWALL 42](#_heading=h.206ipza)

[OUTPUT: 42](#_heading=h.4k668n3)

[COMMANDS: 42](#_heading=h.2zbgiuw)

[Q22) TERNARY MATCH 43](#_heading=h.1egqt2p)

[OUTPUT: 43](#_heading=h.3ygebqi)

[COMMANDS: 43](#_heading=h.2dlolyb)

[Q23) IP FORWARDING 2 44](#_heading=h.sqyw64)

[OUTPUT: 44](#_heading=h.3cqmetx)

[COMMANDS: 45](#_heading=h.1rvwp1q)

(The command to run all programs is running **make run** in the folder containing the program along with a small makefile which includes a variable storing the target switch of BMv2 and other the path to the JSON file containing information about network topology.)

# Q1) BASIC IPV4 FORWARDING

In this we will simply forward the packet according to the IP Address of the destination.

## OUTPUT:

**CASE WHEN USING GRPC SWITCH TABLE ENTRIES ARE DONE USING JSON FILES:**

**Text

Description automatically generated**

**CASE WHEN TABLE ENTRIES ARE NOT PRESENT:**

**Text

Description automatically generated**

**USING SIMPLE SWITCH:**

**Text

Description automatically generated**

The only difference between the above SIMPLE\_SWITCH AND SIMPLE\_SWITCH\_GRPC is the configuration commands of the switches.

# Q2) BASIC TUNNELING

The main aim of this to represent the tunneling meaning that without seeing the IP address it takes decision based on the egress port provided by the control plane.

## OUTPUT:

**PING TEST:**

**Text

Description automatically generated**

**SENDING PACKETS USING IPV4 FORWARDING:**

**Graphical user interface, text

Description automatically generated with medium confidence**

**SENDING PACKETS USING TUNNEL:**

**Text

Description automatically generated with medium confidence**

Here we are using tunneling because we have externally provided using the argument **–dst-id** followed by port number. Seeing this port number the program will encapsulate the my\_tunnel packet into the ethernet header and instead of ipv4 table the mytunnel table will be looked up to and will forward it to the mentioned egress port and ultimately to the host which is attached to that port of the switch.

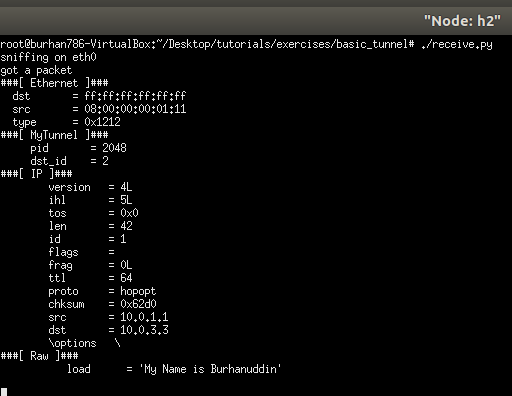
**SENDING PACKETS USING TUNNEL AND NOT IP:**

**SENDING NODE:**

**Text

Description automatically generated**

**RECEIVING NODE:**

****

Here we can see that we have provided ip address of host 3 but external egress port using **–dst-id** as that of host 2 and as we are using tunneling which just looks at the egress port mentioned in the packet. So here despite giving ip address of host 3 the packet reaches host 2.

# Q3) CALCULATOR

(Which performs calculation in the switch and then forwards the packet to the same port to where the receiving host is attached.)

## OUTPUT:

Text

Description automatically generated

Here we can see that we are running a python program which uses scapy library of python which is used to infuse packets into the network which consists of operands and operators which are taken as input from the user, along with appropriate error message.

# Q4) ECN (EXPLICIT CONGESTION NOTIFICATION)

The output is carried out as per the instructions given in the Readme which states that we need to implement ecn where it checks the value of ipv4.ecn field and then then takes action depending on the header filed value as well as the output queue length. If the length is greater than the threshold then the program marks the packet with a ecn value of three denoting congestion in the network and the sender is informed of the congestion and it lowers its sending rate.

## OUTPUT:

Text

Description automatically generated

Here host 1 is sending packets every second for 30 seconds to host2 thus trying to congest the network but ecn comes into action and prevents it from congesting.

### COMMANDS:

1. **make run**

this command fires up the mininet environment with our P4 program installed on the software switch.

1. **On the mininet console run this: xterm h1 h2 h22 h11**

this will open the terminal of the hosts using which we can run programs or configure it.

1. **On h2’s xterm run : ./receive.py**

this starts a server which captures packets coming onto the h2 interface.

1. **On h22’s xterm run: iperf -s -u**

which starts a iperf udp server.

1. **On h1’s xterm run: ./send.py 10.0.2.2 “P4 is cool” 30.**

This will make h1 send a packet every second for 30 seconds addressed to h2

# 

# Q5) FIREWALL

## OUTPUT:

**TCP PACKET SENDING TESTS:**

**Text

Description automatically generated**

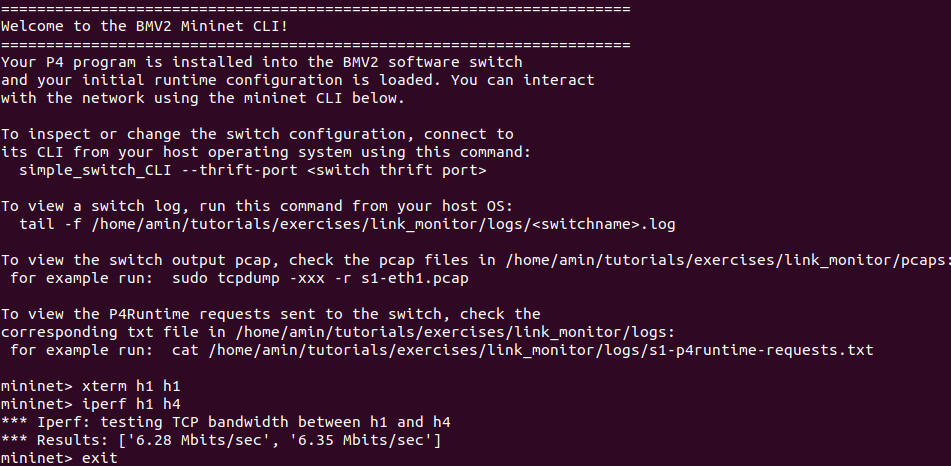
### COMMANDS:

1. **make run:** This will start the mininet topology.
2. **iperf h1 h2:** TCP testing between h1 h2. This is allowed because they are both considered to be in the same internal network.
3. **iperf h1 h3:** TCP testing is successful as seen in the screenshot above because the packet are allowed to leave from host1 and host2 to go to host3 and host4 and not vice versa.
4. **iperf h3 h1:** TCP testing is not successful because the aim of our firewall is to reject the packets coming from hosts 3 and 4 towards hosts 1 and 2 meaning from the external network.

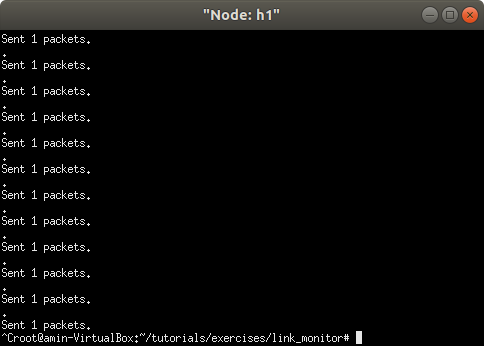
# Q6) LINK MONITOR

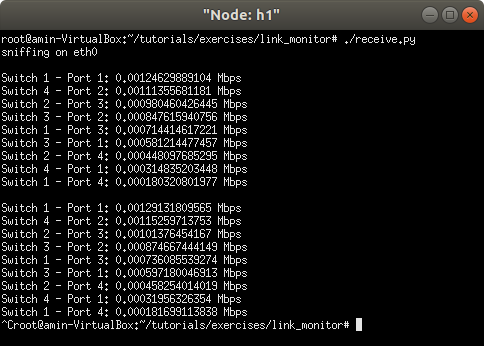
P4 program, link\_monitor.p4, which implements basic IPv4 forwarding, as well as source routing of the probe packets.

## OUTPUT:

****

**two terminals:**





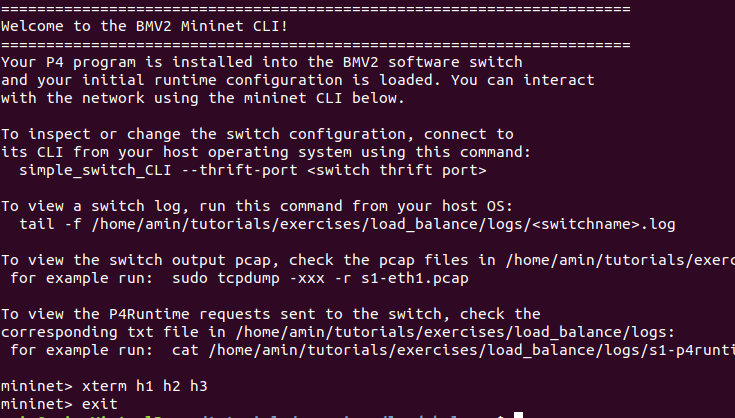
### COMMANDS:

1. In your shell, run:  
   **make run**
2. You should now see a Mininet command prompt. Open two terminals on h1:  
   **mininet> xterm h1 h1**
3. In one of the xterms run the send.py script to start sending probe packets every second. Each of these probe packets takes the path indicated in link-monitor-topo.png.  
   **./send.py**
4. In the other terminal run the receive.py script to start receiving and parsing the probe packets. This allows us to monitor the link utilization within the network.  
   **./receive.py**
5. Run an iperf flow between h1 and h4:  
   mininet> **iperf h1 h4**
6. Type exit to leave each xterm and the Mininet command line. Then, to stop mininet:  
   **make stop**

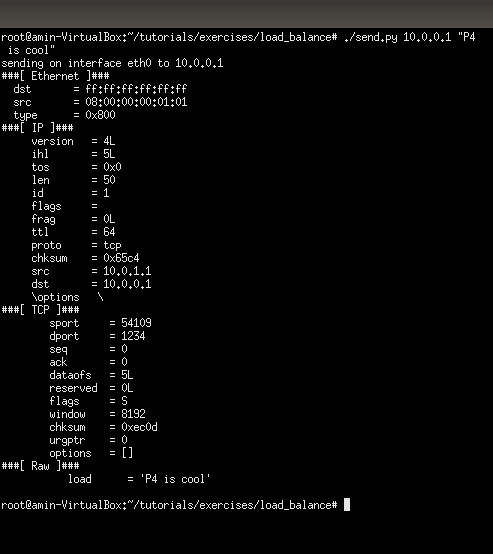
# Q7) LOAD BALANCER

P4 program, load balancer.p4, will properly forward packets.

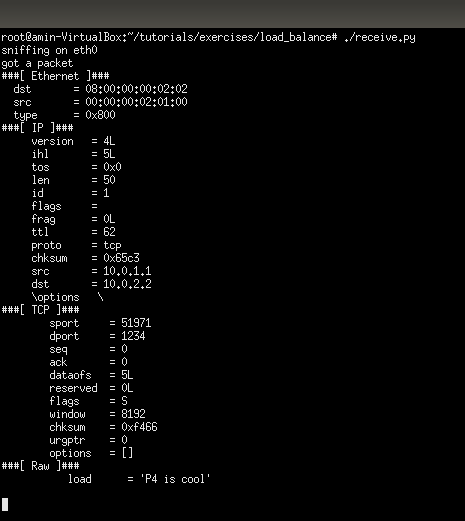
## OUTPUT:

****

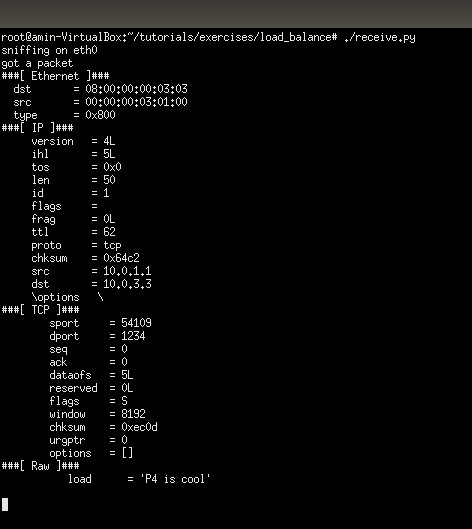
**node h1: (send “P4 is cool”)**

****

**node h2: (text received)**

****

**node h3: (text received)**

****

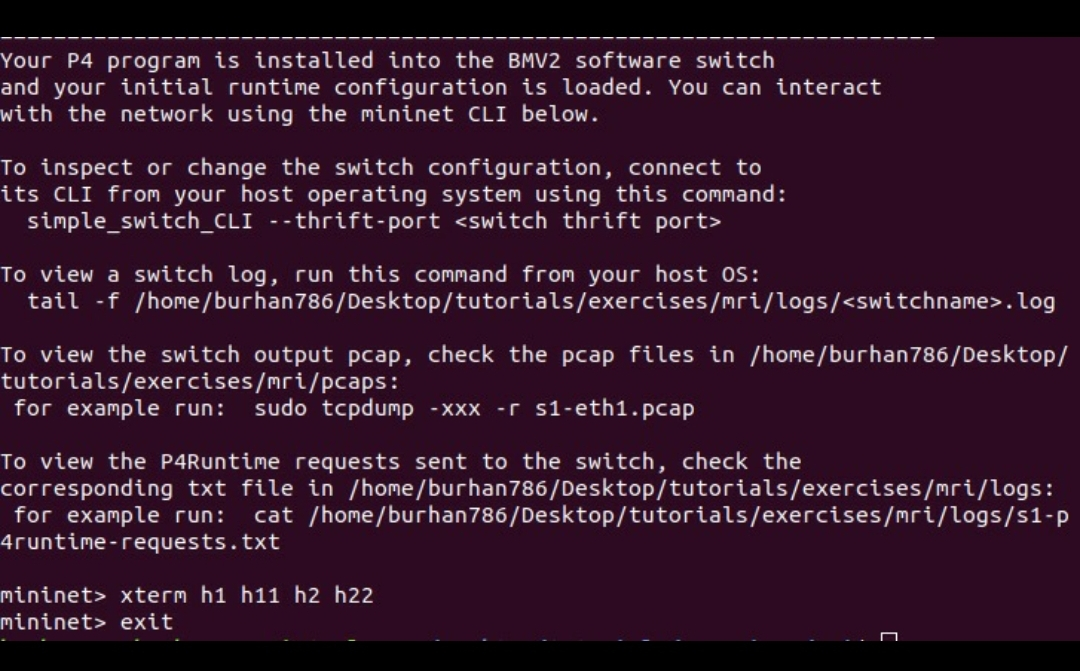
### COMMANDS:

1. In your shell, run:  
   **make run**
2. You should now see a Mininet command prompt. Open three terminals for h1, h2 and h3, respectively:  
   **mininet> xterm h1 h2 h3**
3. Each host includes a small Python-based messaging client and server. In h2 and h3's XTerms, start the servers:  
   **./receive.py**
4. In h1's XTerm, send a message from the client:  
   **./send.py 10.0.0.1 "P4 is cool"**

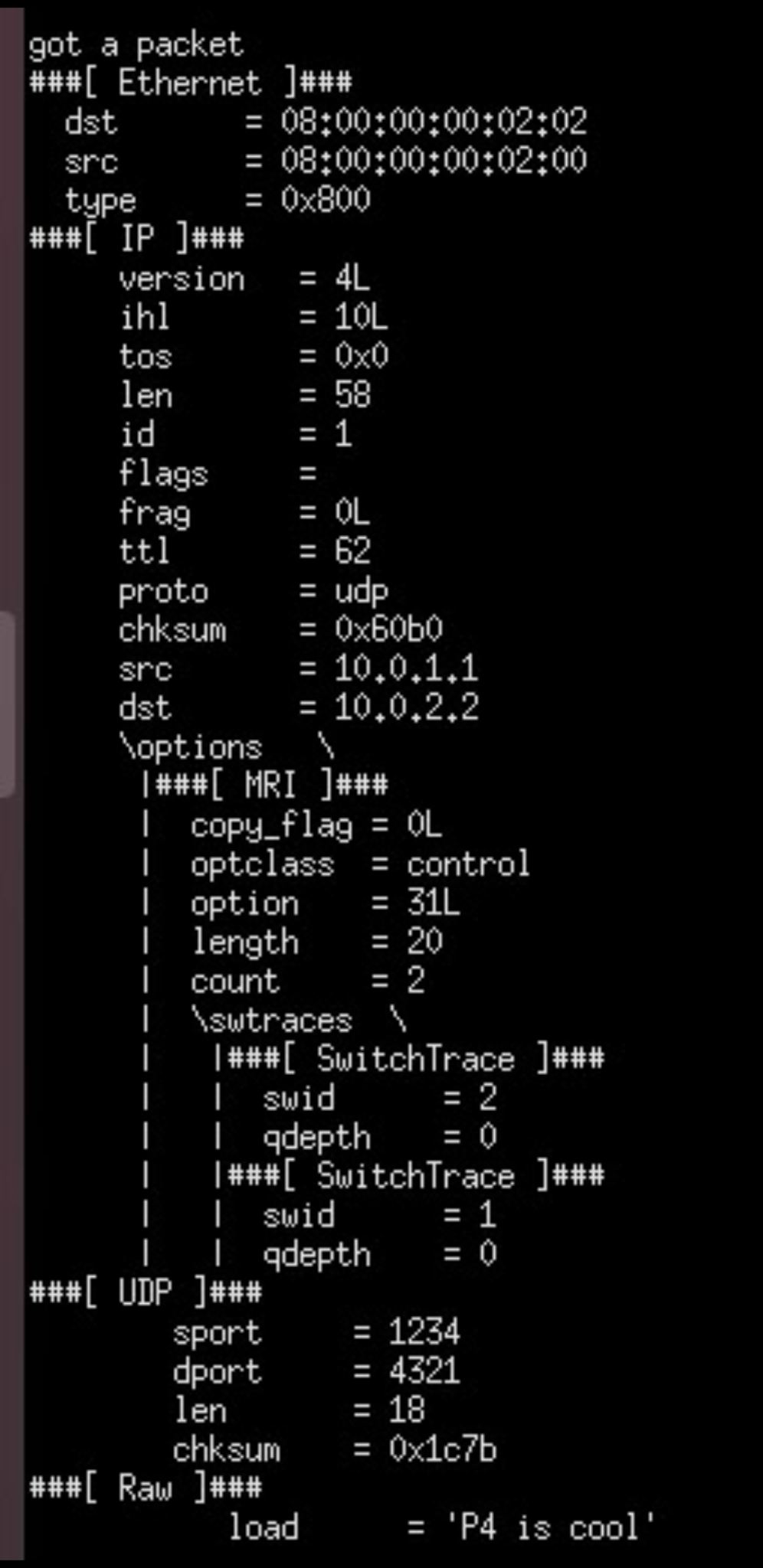
# Q8) MRI (Multi-Hop Route Inspection)

MRI allows users to track the path and the length of queues that every packet travels through. To support this functionality, you will need to write a P4 program that appends an ID and queue length to the header stack of every packet.

## OUTPUT:

****

****

****

### COMMANDS:

1. In your shell, run:

**make**

1. You should now see a Mininet command prompt. Open four terminals for h1, h11, h2, h22, respectively:

**mininet> xterm h1 h11 h2 h22**

1. In h2's xterm, start the server that captures packets:

**./receive.py**

1. in h22's xterm, start the iperf UDP server:

**iperf -s -u**

1. In h1's xterm, send one packet per second to h2 using send.py say for 30 seconds:

**./send.py 10.0.2.2 "P4 is cool" 30**

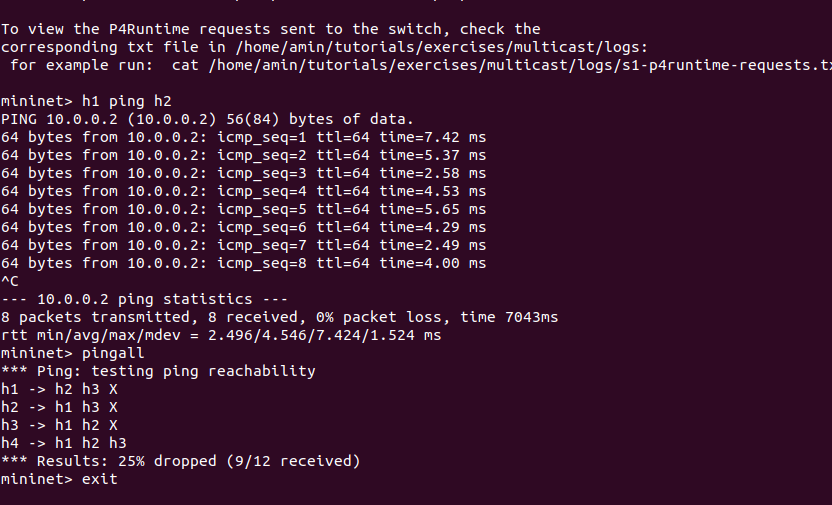
1. The message "P4 is cool" should be received in h2's xterm,
2. In h11's xterm, start iperf client sending for 15 seconds

**iperf -c 10.0.2.22 -t 15 -u**

# Q9) MULTICAST

A P4 program that multicasts packets to a group of ports.

## OUTPUT:

****

### COMMANDS:

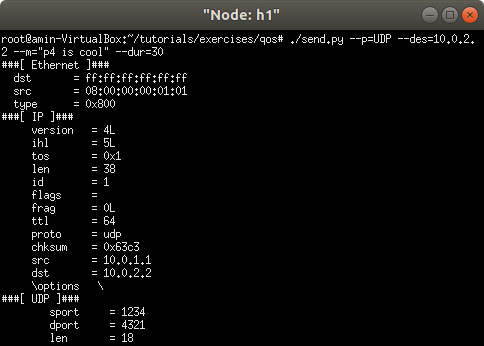
1. In your shell, run:  
   **make run**
2. You should now see a Mininet command prompt.
3. Try to ping between hosts in the topology:  
   **mininet> h1 ping h2**
4. **mininet> pingall**
5. Now after the successful run, close the mininet by typing command- **exit**

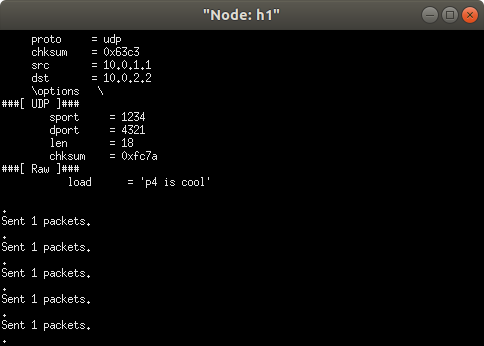
# Q10) QOS (QUALITY OF SERVICE)

It classifies and manages network traffic and provides QOS on modern IP networks.

## OUTPUT:

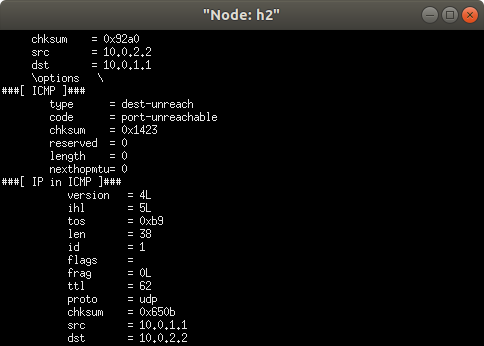
**Node h1: sending UDP packet**

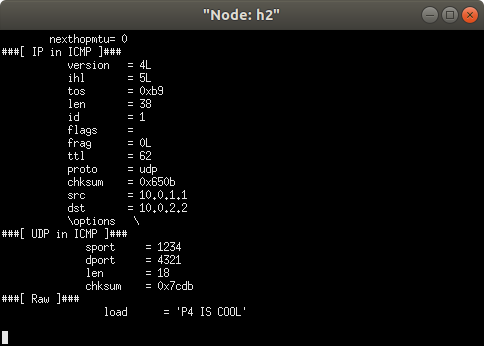
****

****

**Node h2: successfully received packet**

****

****

****

### COMMANDS:

1.In your shell, run:

**make run**

2.You should now see a Mininet command prompt. Open two terminals for h1 and h2, respectively:

**mininet> xterm h1 h2**

3.In h2's XTerm, start the server that captures packets:

**./receive.py**

4.In h1's XTerm, send one packet per second to h2 using send.py say for 30 seconds. To send UDP:

**./send.py --p=UDP --des=10.0.2.2 --m="P4 is cool" --dur=30**

To send TCP:

**./send.py --p=TCP --des=10.0.2.2 --m="P4 is cool" --dur=30**

5.The message "P4 is cool" should be received in h2's xterm.

Then you can close the xterm window.

# Q11) REFLECTOR

The aim of this program is to swap the mac address and redirect it to the sender.

## OUTPUT:

**Text

Description automatically generated**

Here we are trying to send a packet to the host with IP address 10.0.2.2 but it gets redirected to the sender host only.

**Receive.py program running on node1 at eth0 interface catches the redirected packet:**

**Text

Description automatically generated**

### COMMANDS:

1. **make run:** This command will compile the P4 program and instantiate the mininet environment.
2. **Xterm h1 h1:** Opens two xterm terminals which helps us communicate with our client.
3. **./send.py <ip address> <message>:** It runs a python which takes as input the receivers ip address and the message which it wants to send and inserts the packet into the network.
4. **./receive.py:** This python scripts sniffs for any packet which is incoming or outgoing from the interface which it is instructed to sniff.

# Q12) REPEATER

Depending on ingress port we decide which egress port to select

## OUTPUT:

**SENDER:**

**Text

Description automatically generated**

**RECEIVER:**

**Text

Description automatically generated**

Here we are just infusing packets into the network by sending a packet containing hello from host 1 directed towards host 2.

The switch matches the ingress port to a specific number and then decides the egress port according to it. For example, in this question we are using 2 ports port 1 and port 2. So if a packet arrives at port 1 it egresses out on port 2 and vice versa.

**This can be witnessed in the log file of the switch:**

Text

Description automatically generated

# Q13) VERIFYING CHECKSUM

In this exercise we assert that the incoming packets has a valid checksum, if not then the packets are dropped otherwise are forwarded on appropriate egress port.

## OUTPUT:

**RECEIVER RECEIVING PACKETS:**

**Text

Description automatically generated**

**OUTPUT SENDER SENDING PACKETS:**

**Text

Description automatically generated**

### COMMANDS:

1. **make run:** This will start the mininet topology.
2. **Xterm h1 h2:** Opening terminals of hosts to communicate with them.
3. **On host1 run:** python send.py <ds tip address> valid/invalid. The python script will send valid and invalid packets.
4. **On host2 run:** python receive.py. This script will sniff the port for any incoming or outgoing packets.

Valid and invalid packets being forwarded can be seen in the switch log file.

**Valid packet being accepted and forwarded:**

Text, letter

Description automatically generated

**Invalid packet being dropped:**

**A picture containing scatter chart

Description automatically generated**

Here we can see that our condition of a packet being valid and forwarded is false in the second last line and so the packet is being dropped by the switch.

# Q14) METER

Here we are marking the traffic from a particular traffic source based on the amount of traffic coming from it.

## OUTPUT:

**PYTHON SCRIPT SENDING PACKETS FROM H1 TO H2 AT REGULAR INTERVALS:**

**Text

Description automatically generated**

Here the send and receive are in single script where the receive part is ran as another thread of the send part and hence the alternate send and receive message.

The aim of this program is to drop the packets from the sender whose meter value is more than 0.

### COMMANDS:

1. **make run:** This command will compile the P4 program on the target switch and initialize the mininet environment.
2. **sh ./receive.py 1:** This python script will infuse packets destined to host2 every 1 second hence creating traffic in the network.

**SAME GOES WITH INDIRECT METER:**

**Text

Description automatically generated**

# Q15) COUNTER

**SNIFFING THE PORT OF HOST2 FOR PACKETS:**

**Text

Description automatically generated**

The aim of this program is to count the number of packets that are passing through the switch.

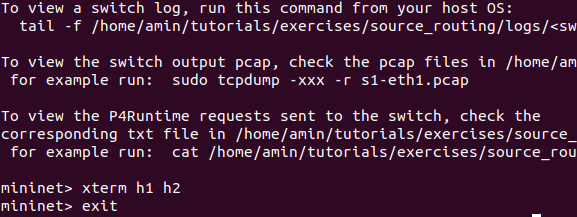
Here I am infusing packets directing towards Host 2 so to count the packets I am running a sniffing program on host2 port and counting the packets that are encountered by this python script on the mentioned port.

# Q16) SOURCE ROUTING

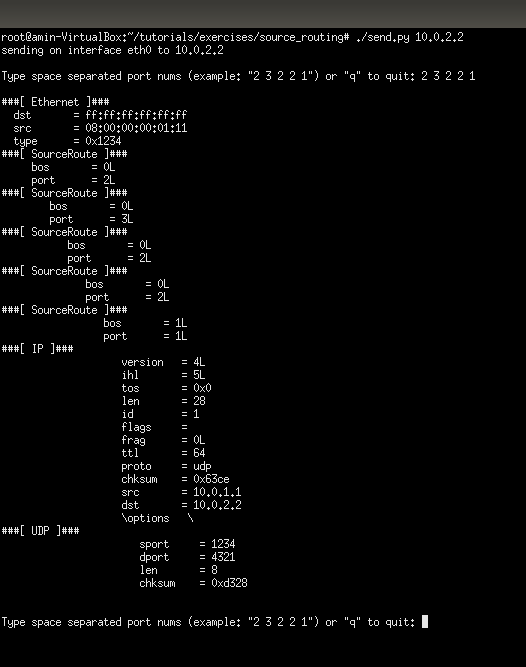
With source routing, the source host guides each switch in the network to send the packet to a specific port.

## OUTPUT:

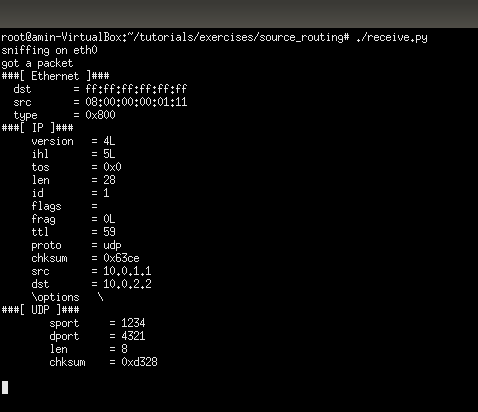
**mininet:**

****

**node h1:**

****

**node h2:**

****

### COMMANDS:

1. In your shell, run:  
   **make run**  
   You should now see a Mininet command prompt. Open two terminals for h1 and h2, respectively:  
   **mininet> xterm h1 h2**
2. Each host includes a small Python-based messaging client and server. In h2's xterm, start the server:  
   **./receive.py**
3. In h1's xterm, send a message from the client:  
   **./send.py 10.0.2.2**

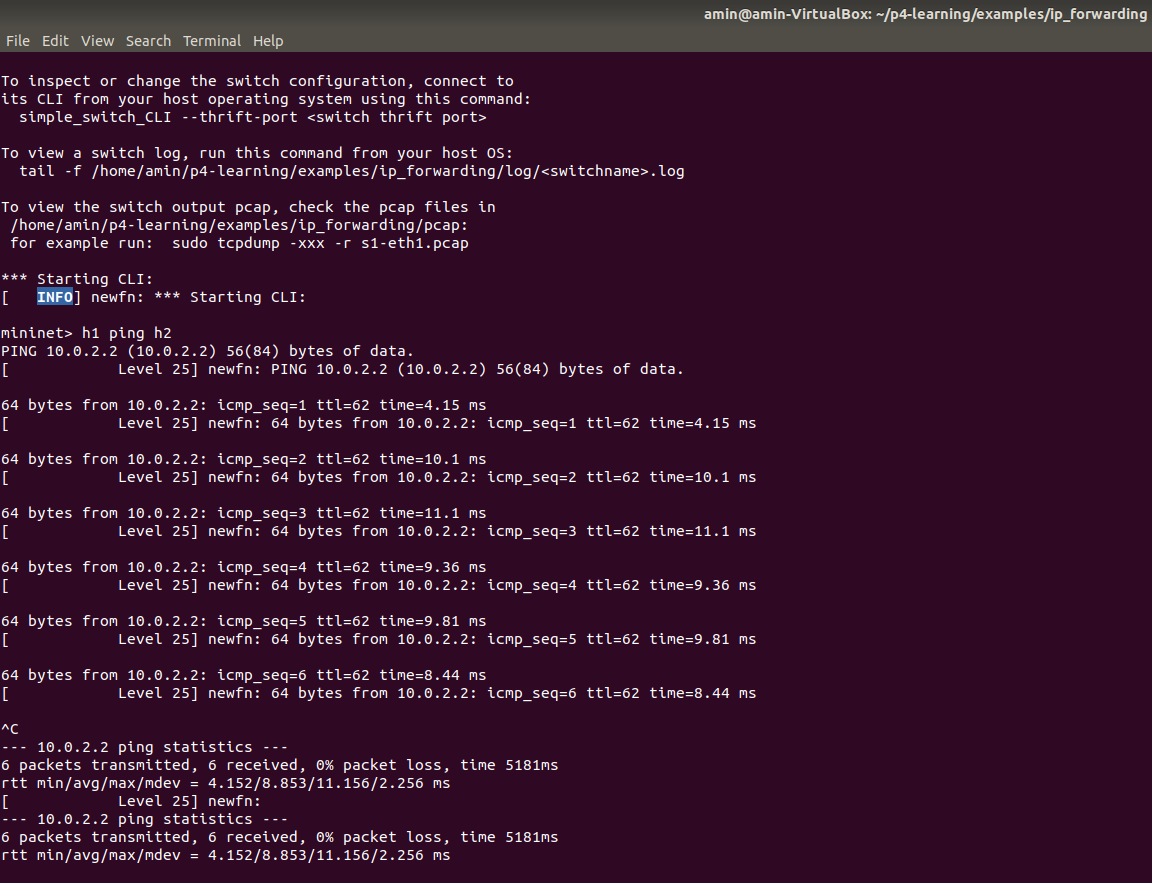
Type a list of port numbers. say 2 3 2 2 1. This should send the packet through h1, s1, s2, s3, s1, s2, and h2.

Check the ttl of the IP header. Each hop decrements ttl. The port sequence 2 3 2 2 1, forces the packet to have a loop, so the **ttl should be 59** at h2.

# Q17) IP FORWARDING

With IPv4 forwarding, the switch must perform the following actions for every packet: (i) update the source and destination MAC addresses, (ii) decrement the time-to-live (TTL) in the IP header, and (iii) forward the packet out the appropriate port.

## OUTPUT:



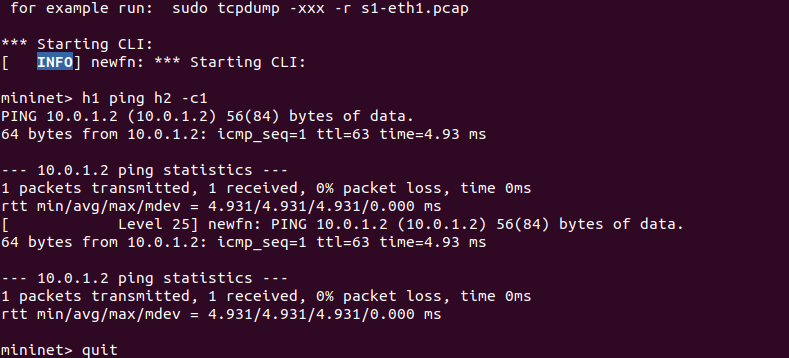
### COMMANDS:

1. **sudo p4run**
2. **mininet> h1 ping h2.**

# Q18) DIGEST MESSAGES

In this example we show how to use the bmv2 digest extern to send information to the control plane using an out of band channel.

**OUTPUT:**

****

### COMMANDS:

1. **sudo p4run**
2. Run the very small controller code that receives packets from the switch:

**sudo python get\_digest.py**

1. Any packet will trigger a digest so we can simply run a ping between h1 and h2, from the CLI

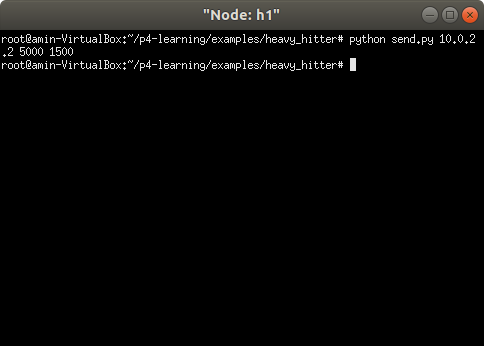
**mininet> h1 ping h2 -c1**

# Q19) HEAVY HITTER

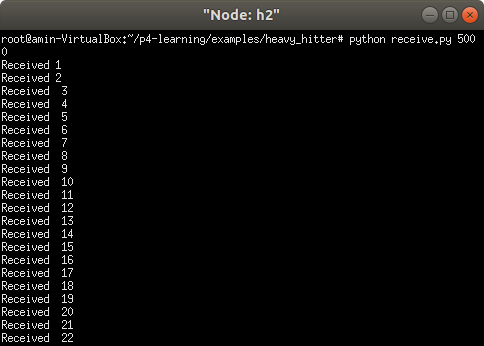
Uses a counting bloom filter to block flows for which the switch has observed more than a certain number of packets (default 1000).

## OUTPUT:

**node h1:**

****

**node h2:**

****

### COMMANDS:

1. Run the topology:

**sudo p4run**

1. Run the receiver and the sending scripts in h2 and h1 respectively:

**xterm h2**

**python receive.py 5000**

1. Send 1500 packets using the same 5-tuple. Only the first 1000 will be received by h2.

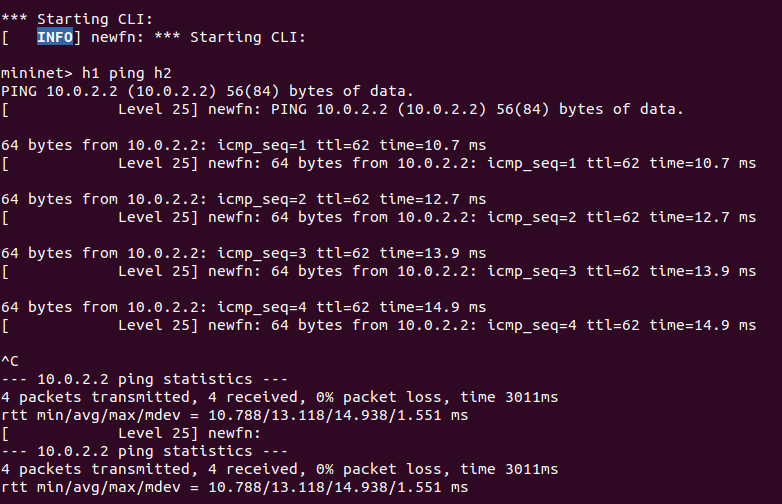
**xterm h1**

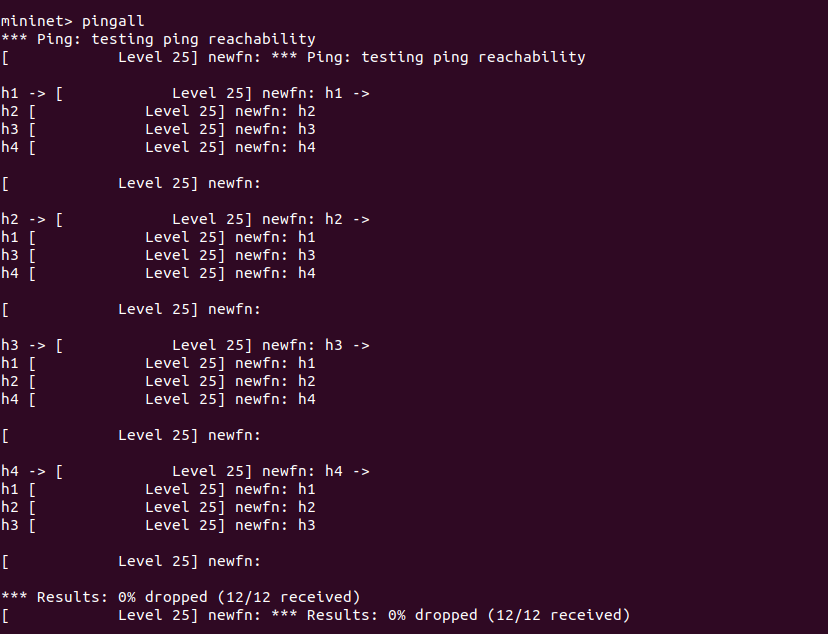
**python send.py 10.0.2.2 5000 1500**

# Q20) IP FORWARDING 2 TABLES

P4 program that implements basic forwarding but using two tables.

## OUTPUT:

****

****

### COMMANDS:

1. Run the topology:

**sudo p4run**

1. Try to ping from one host to another:

**mininet> h1 ping h2**

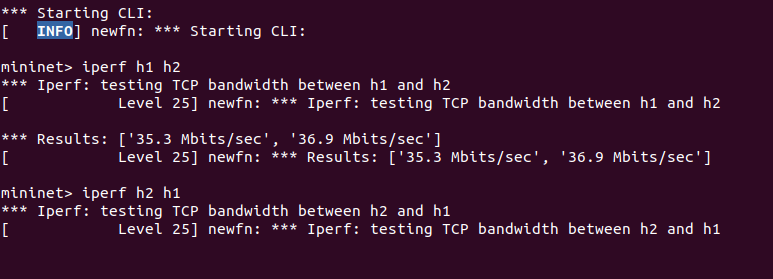
1. Ping from all host pairs to test for connectivity:

**mininet> pingall**

# Q21) STATEFULL FIREWALL

Switch S1 is acting as a stateful firewall, it will only allow connections to be establish from h1. h2 is only able to reply to connections once they have been established from h1.

## OUTPUT:

****

### COMMANDS:

1. Start the topology:

**sudo p4run**

1. Inside the mininet CLI run two different iperfs:
2. From h1 you are able to connect:

\*\*\* Starting CLI:

**mininet> iperf h1 h2**

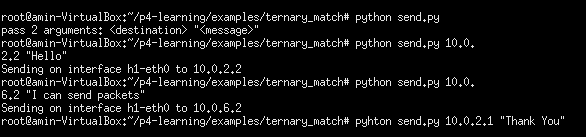
1. From h2 you are not able:

**mininet> iperf h2 h1**

# Q22) TERNARY MATCH

Very simple forwarding program that uses a ternary match.

### OUTPUT:

****

### COMMANDS:

1. To start the topology with the P4 switches:

**sudo p4run**

1. You can send packets and set different destination ip addresses to play with the program.

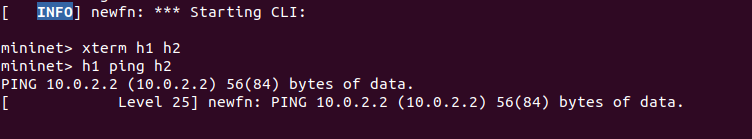
**xterm h1**

**python send.py**

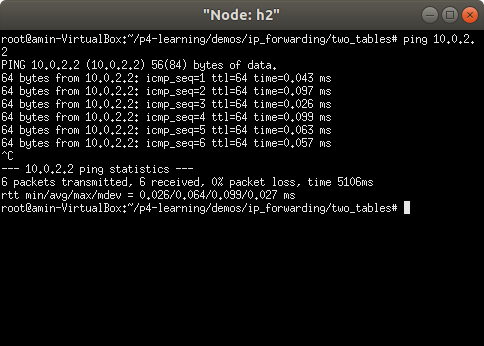
# Q23) IP FORWARDING 2

Hierarchical Forwarding

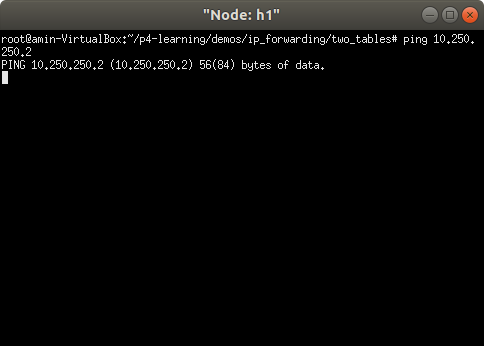
## OUTPUT:

****

**node h2:**

****

**node h1:**

****

### COMMANDS:

1. Run the topology:

**sudo p4run**

1. Runs controller and populates 50k entries:

sudo python controller populate 50000

1. Try to ping from one host to another:

**mininet> h1 ping h2**

1. Simultaneously ping

**xterm h2**

**ping 10.0.2.2**

**xterm h1**

**ping 10.250.250.2**

1. To fail the link between s1 and s2

**mininet> link s1 s2 down**

**END**