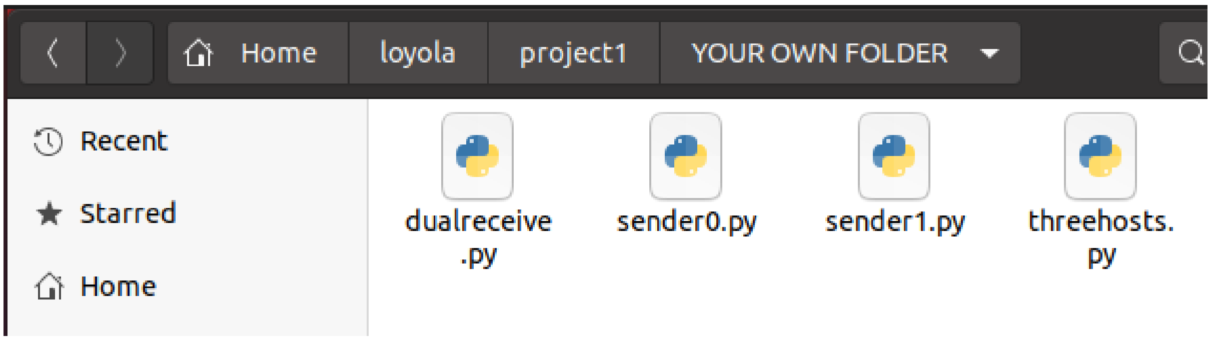
LINUX TRAFFIC CONTROL: TC

This is the first Mininet project to use the Linux tc command and Linux HTB to set bandwidth and delay on various links within a network.

I start this project with the files **threehosts.py, dualreceiver.py, sender0.py, sender1.**py as shown below in the screenshot.

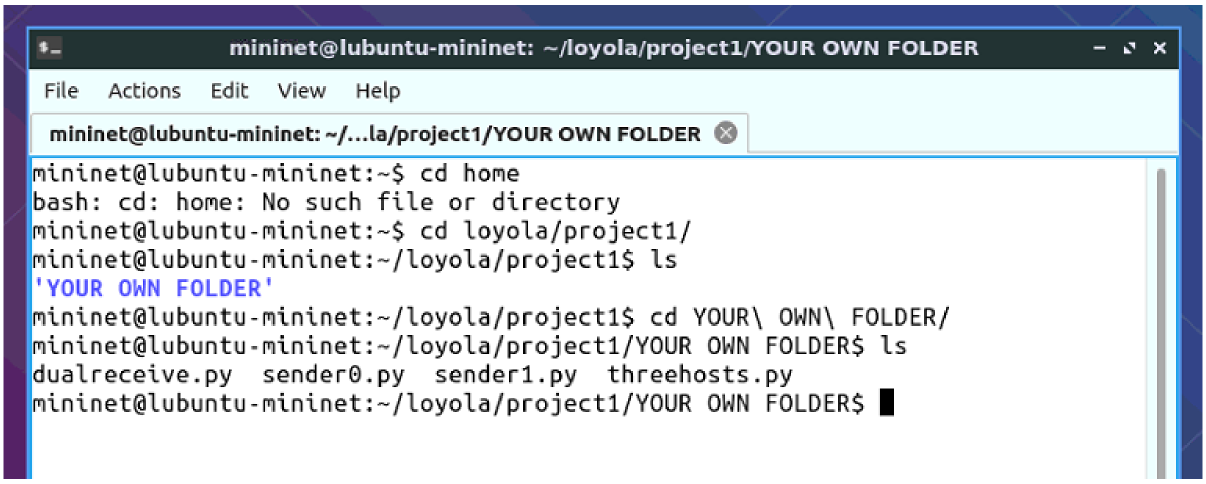


The topology is as follows:

```
h1----+
      |
      r ---- 25 Mbps, 50 ms ---- h4
      |
h2----+
```

Each node is without restriction of bandwidth and time delay except the r--h3 link, which is set to a bandwidth of 25 Mbps and a delay of 40 ms one way, 10 ms the other.

First of all, I create project1 folder inside the Loyola folder the I create YOUR OWN FOLDER inside the project1 folder and I paste the required python scripts as shown in the below screenshot

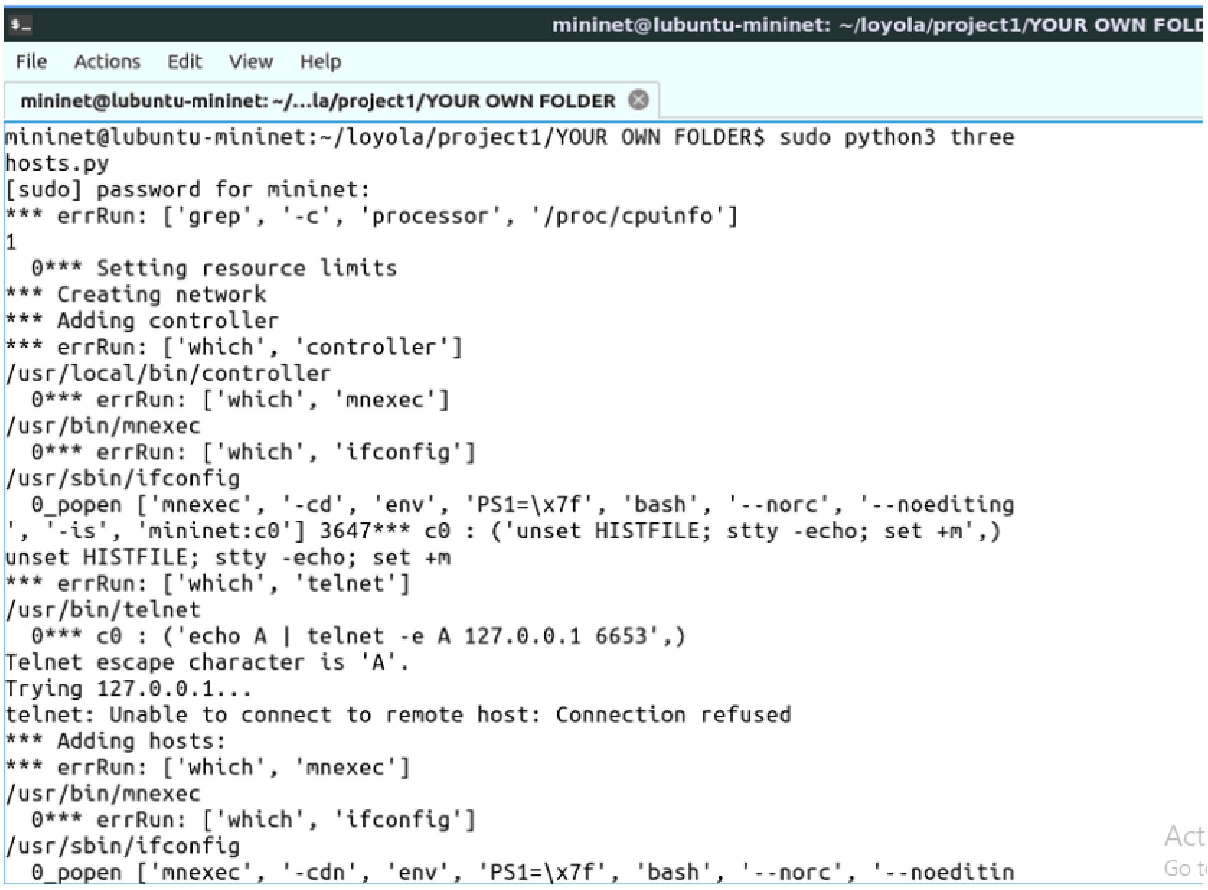

mininet@lubuntu-mininet: ~/loyola/project1/YOUR OWN FOLDER
File Actions Edit View Help
mininet@lubuntu-mininet: ~/...la/project1/YOUR OWN FOLDER

```
mininet@lubuntu-mininet:~$ cd home
bash: cd: home: No such file or directory
mininet@lubuntu-mininet:~$ cd loyola/project1/
mininet@lubuntu-mininet:~/loyola/project1$ ls
'YOUR OWN FOLDER'
mininet@lubuntu-mininet:~/loyola/project1$ cd YOUR\ OWN\ FOLDER/
mininet@lubuntu-mininet:~/loyola/project1/YOUR OWN FOLDER$ ls
dualreceive.py  sender0.py  sender1.py  threehosts.py
mininet@lubuntu-mininet:~/loyola/project1/YOUR OWN FOLDER$ 
```

PART 1

I use Linux HTB to set bandwidth on the h1--r or h2--r links, so that two flows arriving at h2 have two different bandwidth limits. Then I **change** one of the bandwidth limits *while testing is going on*.

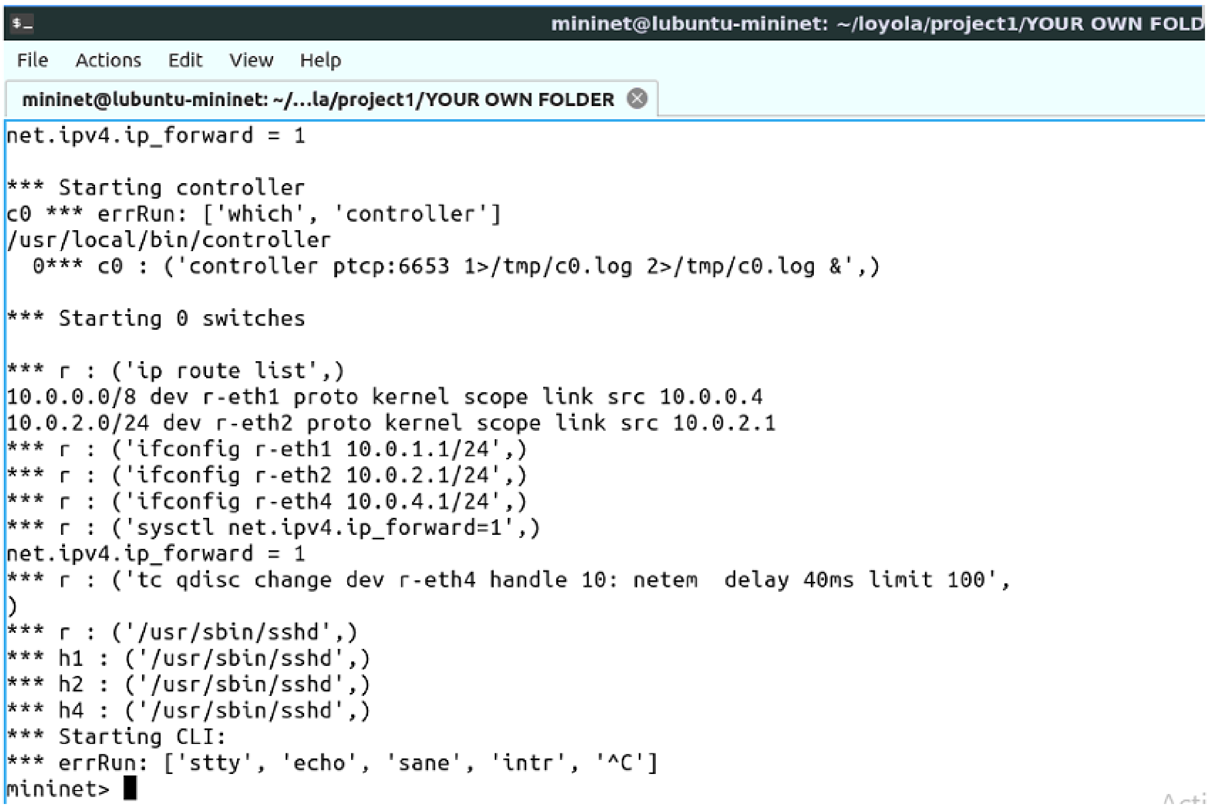
After the above steps I give the below command to run the python script and run the topology as shown in the following screenshot.



mininet@lubuntu-mininet: ~/loyola/project1/YOUR OWN FOLD
File Actions Edit View Help
mininet@lubuntu-mininet: ~/...la/project1/YOUR OWN FOLDER

```
mininet@lubuntu-mininet:~/loyola/project1/YOUR OWN FOLDER$ sudo python3 three
hosts.py
[sudo] password for mininet:
*** errRun: ['grep', '-c', 'processor', '/proc/cpuinfo']
1
  0*** Setting resource limits
*** Creating network
*** Adding controller
*** errRun: ['which', 'controller']
/usr/local/bin/controller
  0*** errRun: ['which', 'mnexec']
/usr/bin/mnexec
  0*** errRun: ['which', 'ifconfig']
/usr/sbin/ifconfig
  0_popen ['mnexec', '-cd', 'env', 'PS1=\x7f', 'bash', '--norc', '--noediting
', '-is', 'mininet:c0'] 3647*** c0 : ('unset HISTFILE; stty -echo; set +m',)
unset HISTFILE; stty -echo; set +m
*** errRun: ['which', 'telnet']
/usr/bin/telnet
  0*** c0 : ('echo A | telnet -e A 127.0.0.1 6653',)
Telnet escape character is 'A'.
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
*** Adding hosts:
*** errRun: ['which', 'mnexec']
/usr/bin/mnexec
  0*** errRun: ['which', 'ifconfig']
/usr/sbin/ifconfig
  0_popen ['mnexec', '-cdn', 'env', 'PS1=\x7f', 'bash', '--norc', '--noeditin
```

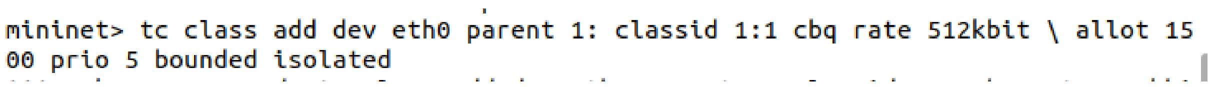
The below screenshot show the nodes with associated Ethernet port i.e. r-eth1 with IP address 10.0.0.0 and r-eth2 with IP address 10.0.2.0 as shown below.


mininet@lubuntu-mininet: ~/loyola/project1/YOUR OWN FOLD
File Actions Edit View Help
mininet@lubuntu-mininet: ~/...la/project1/YOUR OWN FOLDER

```
net.ipv4.ip_forward = 1

*** Starting controller
c0 *** errRun: ['which', 'controller']
/usr/local/bin/controller
  0*** c0 : ('controller ptcp:6653 1>/tmp/c0.log 2>/tmp/c0.log &',)

*** Starting 0 switches

*** r : ('ip route list',)
10.0.0.0/8 dev r-eth1 proto kernel scope link src 10.0.0.4
10.0.2.0/24 dev r-eth2 proto kernel scope link src 10.0.2.1
*** r : ('ifconfig r-eth1 10.0.1.1/24',)
*** r : ('ifconfig r-eth2 10.0.2.1/24',)
*** r : ('ifconfig r-eth4 10.0.4.1/24',)
*** r : ('sysctl net.ipv4.ip_forward=1',)
net.ipv4.ip_forward = 1
*** r : ('tc qdisc change dev r-eth4 handle 10: netem  delay 40ms limit 100',
)
*** r : ('/usr/sbin/sshd',)
*** h1 : ('/usr/sbin/sshd',)
*** h2 : ('/usr/sbin/sshd',)
*** h4 : ('/usr/sbin/sshd',)
*** Starting CLI:
*** errRun: ['stty', 'echo', 'sane', 'intr', '^C']
mininet> 
```

I give the following **tc** commands I used, including the **tc class change** command.

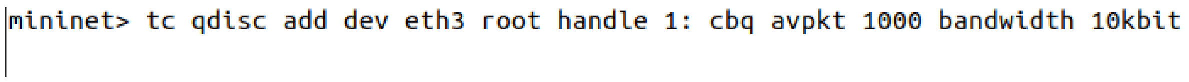
#tc class add dev eth0 parent 1: classid 1:1 cbq rate 500kbit \ allot 1500 prio 5



```
mininet> tc class add dev eth0 parent 1: classid 1:1 cbq rate 512kbit \ allot 15
00 prio 5 bounded isolated
```

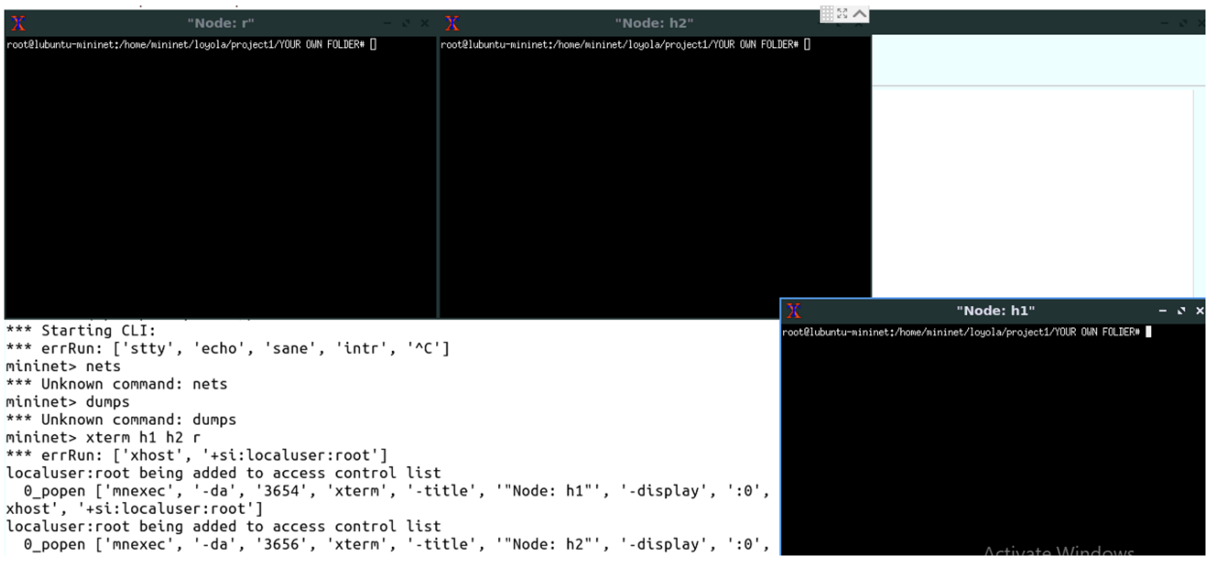
boundedisolated

Tc qdisc add dev eth3 root handle 1: cbq avpkt 1000 bandwidth 10kbit



```
mininet> tc qdisc add dev eth3 root handle 1: cbq avpkt 1000 bandwidth 10kbit
```

By giving the xterm h1 h2 r command the below screenshot is shown and terminals were opened in all the three nodes as shown below.



```
root@lubuntu-mininet:/home/mininet/loyola/project1/YOUR OWN FOLDER#
```

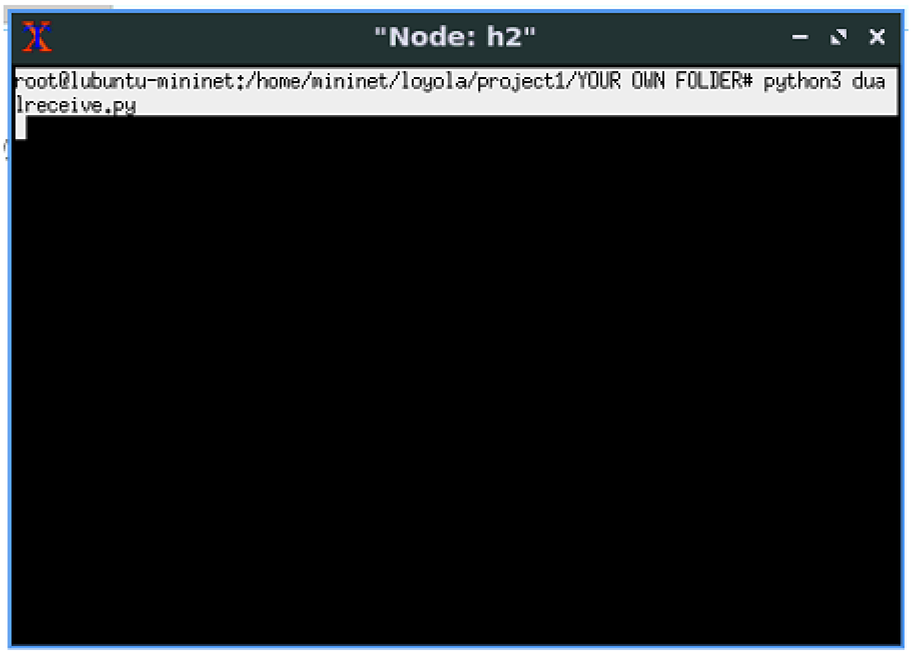
```
root@lubuntu-mininet:/home/mininet/loyola/project1/YOUR OWN FOLDER#
```

```
root@lubuntu-mininet:/home/mininet/loyola/project1/YOUR OWN FOLDER#
```

```
*** Starting CLI:
*** errRun: ['stty', 'echo', 'sane', 'intr', '^C']
mininet> nets
*** Unknown command: nets
mininet> dumps
*** Unknown command: dumps
mininet> xterm h1 h2 r
*** errRun: ['xhost', '+si:localuser:root']
localuser:root being added to access control list
  0_popen ['mnexec', '-da', '3654', 'xterm', '-title', '"Node: h1"', '-display', ':0',
xhost', '+si:localuser:root']
localuser:root being added to access control list
  0_popen ['mnexec', '-da', '3656', 'xterm', '-title', '"Node: h2"', '-display', ':0',
```

To receive data, I use dualreceive.py on h2. To send data, I use sender.py. The full command line for sender.py, sending 2000 blocks to 10.0.3.10 port 5430, is:

```
python3 sender.py 2000 10.0.3.10 5430
```



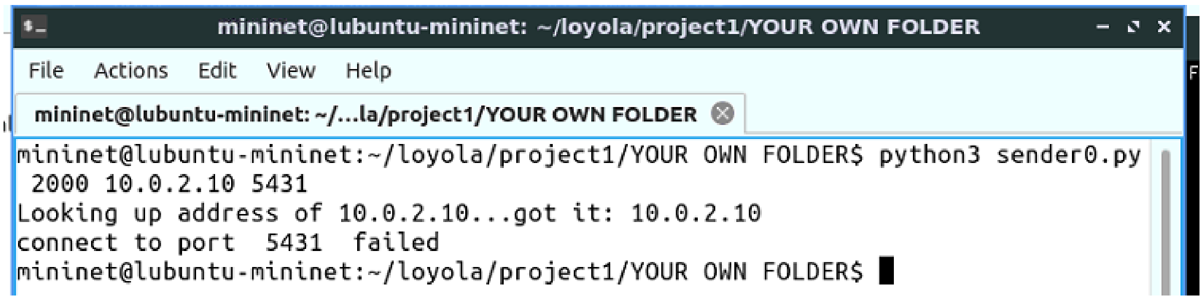
```
root@lubuntu-mininet:/home/mininet/loyola/project1/YOUR OWN FOLDER# python3 dua
lreceive.py
```

I start two flows on h1, from two different ports, or else add a new node, h3, and have one flow start on h1 and the other on h3.

I run htb on r. Because I was trying to control traffic through r, I must apply htb to the *downstream* interface, which for h1->h2 traffic is named **r-eth2**. Here are the commands necessary to limit h1->h2 UDP traffic to 1 mbit/sec, and h1->h2 TCP traffic to 10 mbit/sec. When specifying rates, mbit is megabit/sec and mbps is mega**byte**/sec. Also, these units are written adjacent to the numeric value, with no intervening space!

First, I set up the htb "root", and a default class (which we eventually do not use, but the point is that traffic goes through the default class until everything else is set up, so traffic is never blocked).

```
tc qdisc del dev r-eth2 root        # delete any previous htb stuff on r-eth2

tc qdisc add dev r-eth2 root handle 1: htb default 10    # class 10 is the default class
```

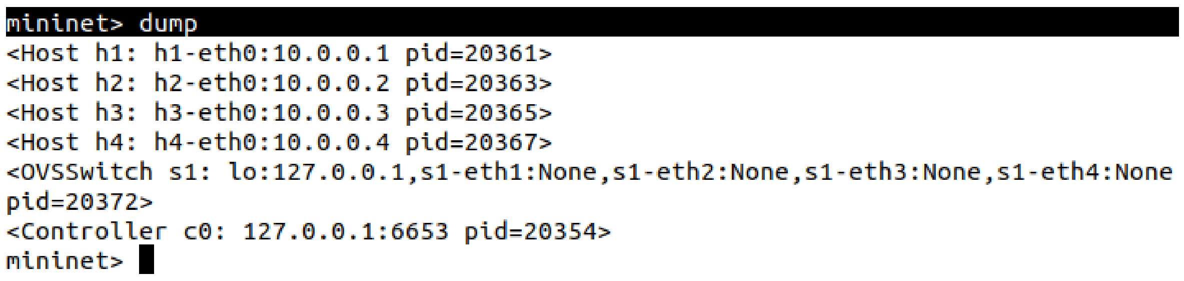

mininet@lubuntu-mininet: ~/loyola/project1/YOUR OWN FOLDER
File Actions Edit View Help
mininet@lubuntu-mininet: ~/...la/project1/YOUR OWN FOLDER

```
mininet@lubuntu-mininet:~/loyola/project1/YOUR OWN FOLDER$ python3 sender0.py
 2000 10.0.2.10 5431
Looking up address of 10.0.2.10...got it: 10.0.2.10
connect to port  5431  failed
mininet@lubuntu-mininet:~/loyola/project1/YOUR OWN FOLDER$
```

Now I add two **classes**, one for UDP (class **1**) and one for TCP (class **2**). At this point I can't tell yet what each class is for; that's in the following paragraph.

Using dump command

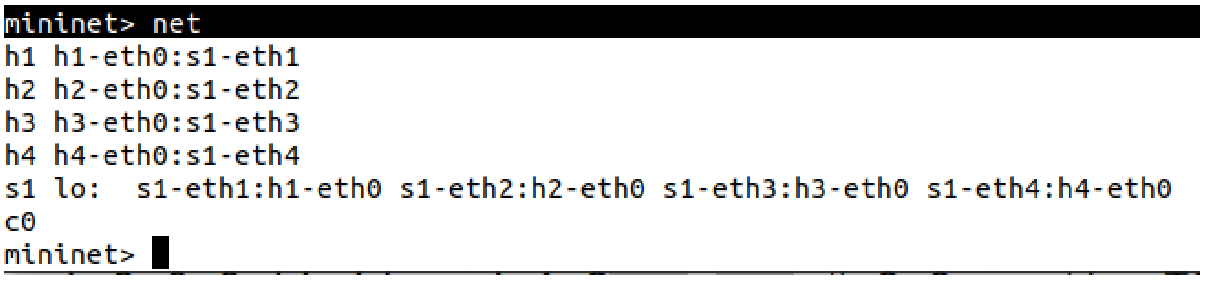
This command show hosts with name, interface, IP address and process ID as shown below.



```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=20361>
<Host h2: h2-eth0:10.0.0.2 pid=20363>
<Host h3: h3-eth0:10.0.0.3 pid=20365>
<Host h4: h4-eth0:10.0.0.4 pid=20367>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None
pid=20372>
<Controller c0: 127.0.0.1:6653 pid=20354>
mininet>
```

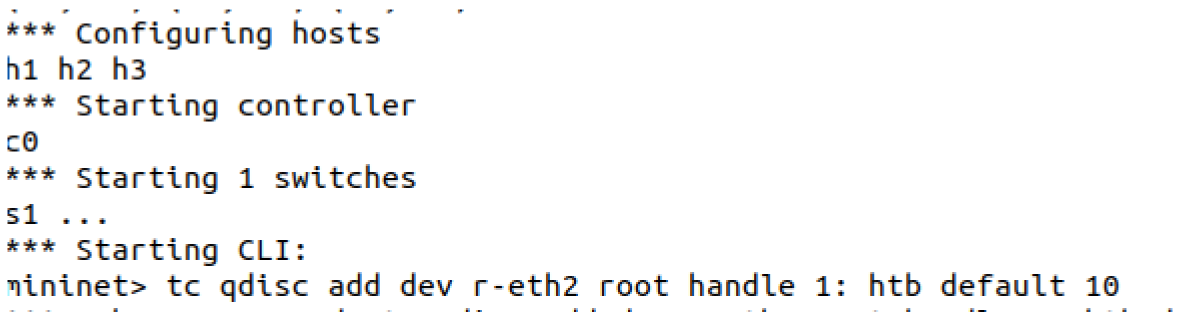
Using net command

This command only shows host names and interfaces.



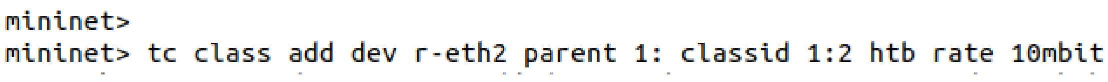
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
mininet>
```

tc class add dev r-eth2 parent 1: **classid** 1:**1** htb rate 1mbit # '1mbit' has no space! The classes are 1:1 and (next line) 1:2



```
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> tc qdisc add dev r-eth2 root handle 1: htb default 10
```

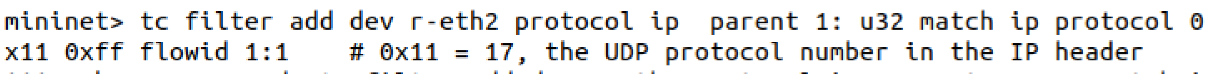
tc class add dev r-eth2 parent 1: **classid** 1:**2** htb rate 10mbit



```
mininet>
mininet> tc class add dev r-eth2 parent 1: classid 1:2 htb rate 10mbit
```

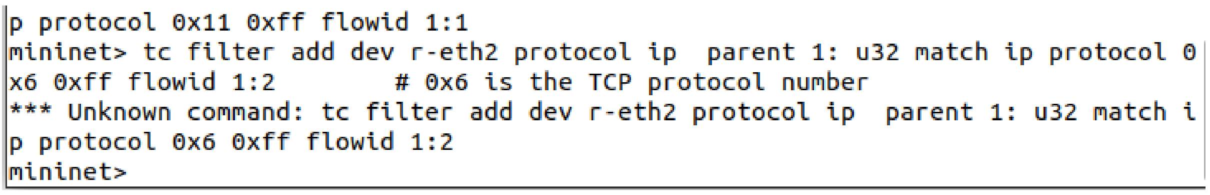
Now I have to create *filters* that assign traffic to one class or another. The flowid of the filter matches the classid above. I will assign UDP traffic to classid 1:1, and TCP traffic to classid 1:2, although the tc filter command calls these **flowids**. The "parent 1:" identifies the root above with handle 1:. The "u32" refers to the so-called u32 classifier; the name comes from unsigned 32-bit. The 0xff is an 8-bit mask.

tc filter add dev r-eth2 protocol ip parent 1: u32 match ip protocol 0x11 0xff flowid 1:1 # 0x11 = 17, the UDP protocol number in the IP header



```
mininet> tc filter add dev r-eth2 protocol ip  parent 1: u32 match ip protocol 0
x11 0xff flowid 1:1    # 0x11 = 17, the UDP protocol number in the IP header
```

tc filter add dev r-eth2 protocol ip parent 1: u32 match ip protocol 0x6 0xff flowid 1:2 # 0x6 is the TCP protocol number

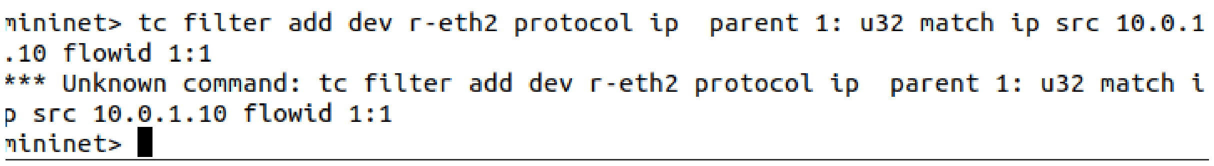


```
p protocol 0x11 0xff flowid 1:1
mininet> tc filter add dev r-eth2 protocol ip  parent 1: u32 match ip protocol 0
x6 0xff flowid 1:2        # 0x6 is the TCP protocol number
*** Unknown command: tc filter add dev r-eth2 protocol ip  parent 1: u32 match i
p protocol 0x6 0xff flowid 1:2
mininet>
```

Now if we start the senders (you'd have to come up with your own UDP sender), we should see UDP traffic getting 1mbit and TCP traffic getting 10mbit. We can also just drop the 1:2 class (and its filter), and have that traffic go to the default. In this case, UDP is limited to 1mbit and TCP is not limited at all.

We can also apply filters to traffic from selected ports or hosts, so different traffic from the same host is treated differently. Here are a couple examples; the first filters by IP *source* address and the second by TCP *destination* port number (16 bits, so we need a 16-bit mask 0xffff):

tc filter add dev r-eth2 protocol ip parent 1: u32 match ip src 10.0.1.10 flowid 1:1



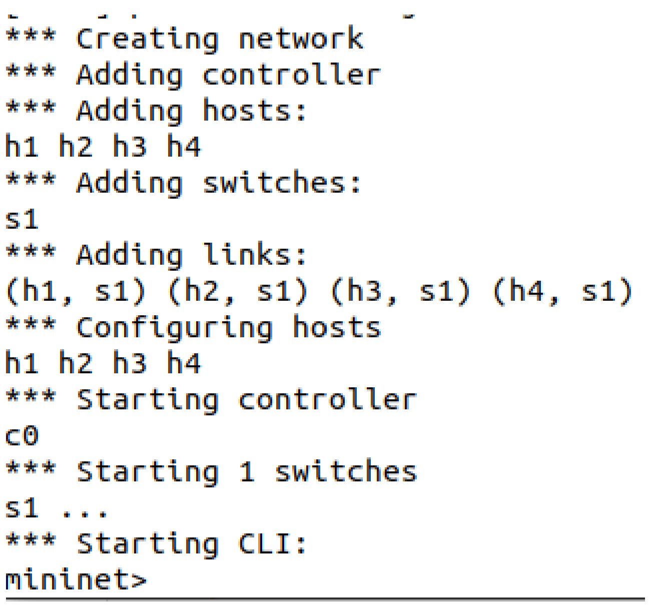
```
mininet> tc filter add dev r-eth2 protocol ip  parent 1: u32 match ip src 10.0.1
.10 flowid 1:1
*** Unknown command: tc filter add dev r-eth2 protocol ip  parent 1: u32 match i
p src 10.0.1.10 flowid 1:1
mininet>
```

PART 2

Basically I do the same, but now I use h4 as the destination, and I also set the delay and the queue capacity. *Set a smaller bandwidth on the r--h4 link, and a higher bandwidth on the h1--r link.*

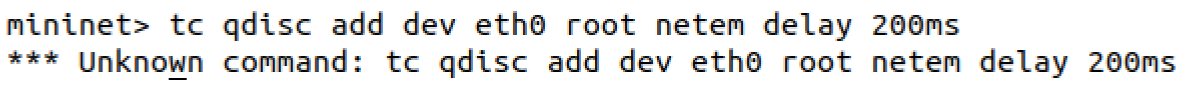
The tricky part here is that there already *is* a queuing discipline attached to r-eth4, so I will have to *change* it rather than *add* it. And I'll have to get the classid m: n numbers correct. I use "tc qdisc show" and "tc class show dev r-eth4" to figure this out. But, basically, here's the hierarchy:

For this part I create a network with 4 nodes as shown below



```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

5:	the root HTB qdisc
5:1	the HTB class below the root, specifying rate and ceil and burst
10:	the **net**em (**net**work **em**ulator) class below HTB that specifies the delay and limit

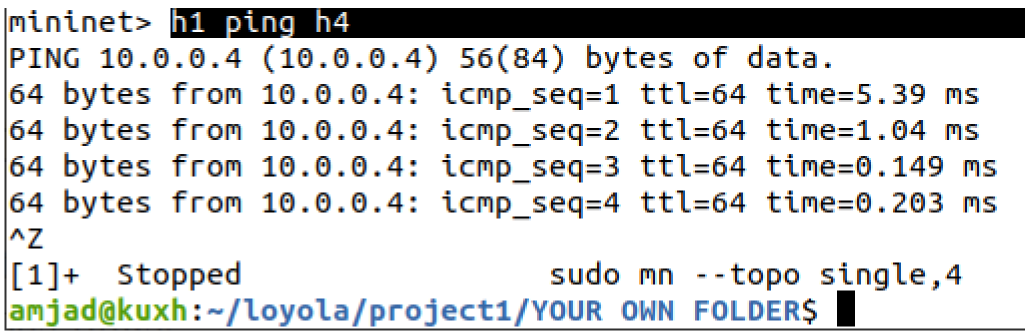


```
mininet> tc qdisc add dev eth0 root netem delay 200ms
*** Unknown command: tc qdisc add dev eth0 root netem delay 200ms
```

The queue capacity won't matter for this assignment, but it *will* matter for the next one (TCP Reno vs Cubic).

I verify my delay with the ping command. I note that netem is "classless", so I can't directly set different delays for different traffic classes.

Using ping command



```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=5.39 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=1.04 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.149 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.203 ms
^Z
[1]+  Stopped                 sudo mn --topo single,4
amjad@kuxh:~/loyola/project1/YOUR OWN FOLDER$ 
```