MININET TERM PAPER

1. ***Introduction***

Mininet is a network emulator, behaves like a real machine (created using software rather than hardware) and is a collection of end hosts, switches, routers, and links on a single Linux kernel. Clients and servers communicate through mininet using iperf (creates TCP and UDP data streams) in order to measure the throughput.

TCP is highly reliable protocol while UDP is fast transmission protocol (maximum throughput). Here, we will see the behaviour of TCP and UDP by setting up the network topology, flow rules, and do iperf using mininet and then observe.

The goal is to analyse the changes happen in bandwidth being allocated for TCP and UDP at each interval, while traffic is being sent to and from different hosts, and to analyse the behaviour of TCP and UDP, when they share same queue for data transmission and when they uses different queues for data transmission. Observe what happens when we stop UDP traffic, what changes occur in TCP flow.

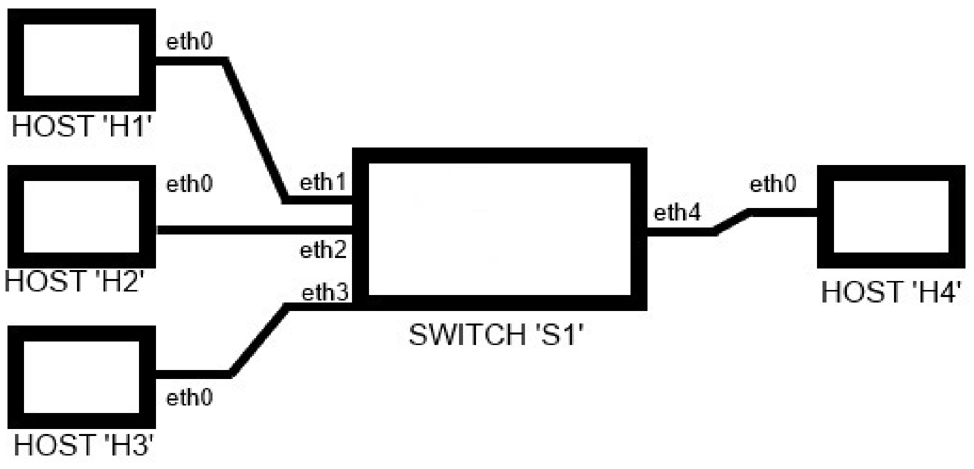
2. ***Experimental Design***

For the given assignment, we created a topology with 4 hosts (H1, H2, H3, H4) connected to a switch (S1). The link bandwidth is 1Gbps. H4 is a server while H1,H2 and H3 are the clients. H1 and H3 are transferring iperf traffic with TCP and UDP simultaneously, while H2 is downloading file from http server (H4).

For first part of the assignment, for each UDP connection bandwidth limit has been limited to 512Mbps. For every connection with H4, traffic is being transferred simultaneously. For H1 and H3, both TCP and UDP traffic is being transferred for 120 seconds for each experimental iteration, while UDP traffic stops after 80 seconds. After that instant, we will observe the changes occurred in TCP traffic flow. All TCP, UDP and other traffic is sharing the same bandwidth link.

For second part of the assignment, we will set two different queues (q1, q2), one for TCP and one for UDP traffic, and each queue is supposed to be limited to use 512 Mbps bandwidth. All the TCP and UDP traffic coming from different hosts will be sent to their respective queues at egress port.

For first part :



NETWORK TOPOLOGY WITH 1-SWITCH, 4-HOSTS

In *terminal 1* we executed the command '*sh topo.sh*'.
the above bash file has the following command.
"*sudo mn --topo=single,4 --link=tc,bw=1000 --controller=remote --mac*" .
This means we are creating a topology with a single switch and 4 hosts, bandwidth of 1Gbps is being used. controller is a remote controller (to specify user defined flow rules) and mac addresses of all hosts and switch.

In *terminal 2* we executed '*sh net.sh*' , this bash file contains the flow rules for the topology.
the commands in the above bash file are as follows

```
:-> "sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,action=output:1"
:-> "sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,action=output:2"
:-> "sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,action=output:3"
:-> "sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:04,action=output:4"
:-> "sudo ovs-ofctl add-flow s1 dl_dst=ff:ff:ff:ff:ff:ff,action=flood"
:-> "sudo ovs-ofctl dump-flows s1"
```

Here we created the flow rules. H1 will be connected to ethernet port 1 of the switch, and similarly H2, H3 and H4 are connected to port 2, 3 and 4 respectively. If mac address is this (00:00:00:00:00:01) send to output interface eth1, and so on respectively. The last one this flow rule matches the every packet and it will flood in the switch.

After installing the above flow rules to the switch, we got the following connections:

```
:->mininet>net
        H1      h1-eth0:s1-eth1
        H2      h2-eth0:s1-eth2
        H3      h3-eth0:s1-eth3
        H4      h4-eth0:s1-eth4
```

Then, in *terminal 1*, we executed the command: "*xterm h1 h2 h3 h4*".
By doing this, terminals for each host will be opened.

Now H4 will be the tcp, udp and http server. In "Node: H4" (terminal for H4), run '*sh server.sh*'. This bash file has the following command:
"*iperf -sp 5002 -i 4 > tcp & iperf -sp 5003 -u -i 4 > udp & python -m SimpleHTTPServer 8080*"

First part of command (*iperf -sp 5002 -i 4 > tcp*) creates *tcp server* at port number 5002 with '*-sp*' options; tcp is default type of server. It records the traffic and bandwidth at the interval of every 4 (*-i 4*)seconds and ' *> tcp* ' records the results. Second part of the command (*iperf -sp 5003 -u -i 4 > udp*) creates *udp server* at port 5003. '*-u*' is used to change the server type to UDP. It records the traffic and bandwidth at the interval of every 4 seconds and ' *> udp* ' records the results. Third part of the command creates the *HTTP server* over tcp at port number *8080*.

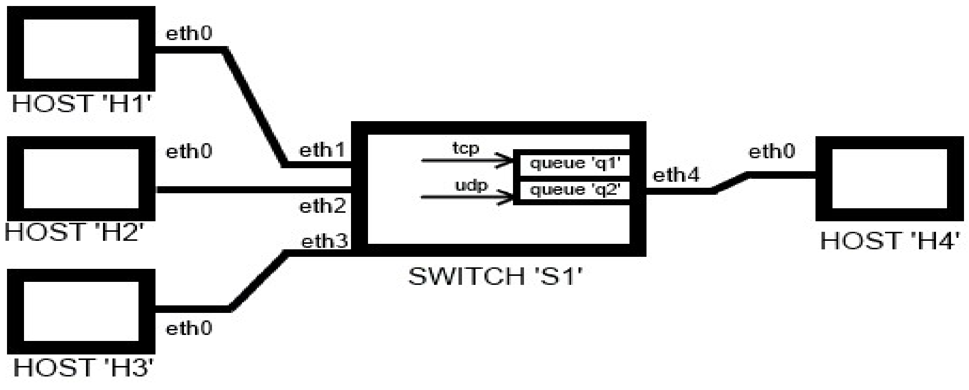
H1 and H3 are the clients, which are sending tcp and udp traffic simultaneously. At the 'xterm' terminals for both H1 and H3, that is
in both "Node: H1" and "Node: H3" , we will set up the following command:

"i*perf -c 10.0.0.4 -p 5002 -i 4 -t 120 & iperf -c 10.0.0.4 -u -p 5003 -i 4 -t 80 -b 512M*". '*-c*' is used for client, *'-t'* is for time limit to transfer traffic in seconds. *'-b'* is used for bandwidth allocated to that particular traffic type.

H2 is the http client, which transfers the traffic over TCP. Command used in xterm h2 is as follows:

"*wget -r 10.0.0.4:8080*". '*wget*' to download and '*-r*' option will create a structure of directories beginning with '*10.0.0.4:8080'* include the downloaded file from the server inside it.
Now we will start the server and start the traffic from all the clients simultaneously and analyse the behaviour of traffic.

For second part:



NETWORK TOPOLOGY WITH 1-SWITCH, 4-HOSTS AND QUEUE AT EGRESS PORT

Here the topology is same as of part 1, but here we will create separate queues at egress port to handle both tcp and udp traffic separately. Each of the two queues will be limited to bandwidth 512Mbps. The third queue (*q0*) is for other traffics (*ICMP, ARP etc.*), so that we can observe TCP and UDP behaviour more accurately.

In *terminal 1*, executing the command '*sh topology.sh*', which contains the following command:

"*sudo mn --topo=single,4 --controller=remote --mac*" .

In another terminal, *terminal 2*, we will execute the bash command as '*sh queh4.sh'*, which is containing the command as follows:

"*sudo ovs-vsctl -- set Port s1-eth4 qos=@newqos -- --id=@newqos create QoS type=linux-htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create Queue other-config:min-rate=1000000000 other-config:max-rate=1000000000 -- --id=@q1 create Queue other-config:min-rate=512000000 other-config:max-rate=512000000 -- --id=@q2 create Queue other-config:min-rate=512000000 other-config:max-rate=512000000*"

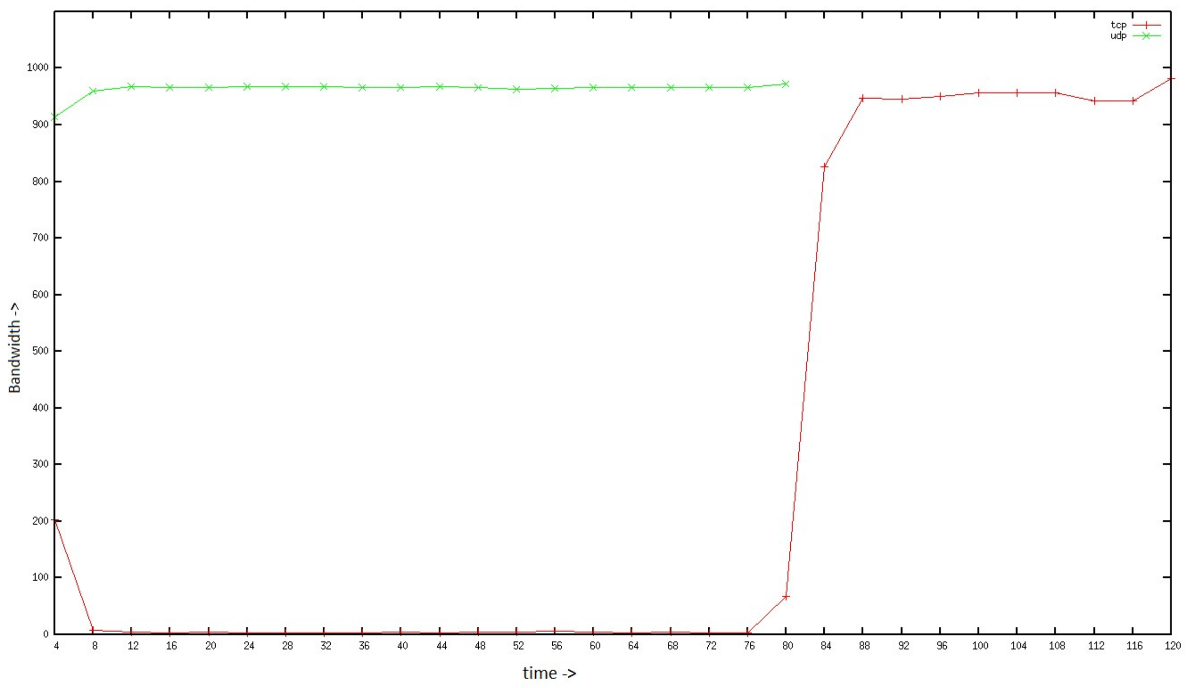
Here *s1-eth4* is the *egress* port. At this port three queues are created, q1 is used for TCP traffic, q2 is used for UDP traffic, and q0 is for other traffics (*ICMP, ARP, etc.*) with their specified flow-rates.

Now, executing the shell file '*sh qflow.sh*', which contains the following command:#!/bin/sh

```
:->”sudo ovs-ofctl add-flow s1
priority=65535,ip,nw_dst=10.0.0.4,tcp,tp_dst=5002,actions=enqueue:4:1”
:->“sudo ovs-ofctl add-flow s1
priority=65535,ip,nw_dst=10.0.0.4,udp,tp_dst=5003,actions=enqueue:4:2”
:-> “sudo ovs-ofctl add-flow s1 priority=1,dl_dst=00:00:00:00:00:04,actions=output:4”
:-> “sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,actions=output:1”
:-> “sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,actions=output:2”
:-> “sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,actions=output:3”
:->“sudo ovs-ofctl add-flow s1 dl_dst=ff:ff:ff:ff:ff:ff,actions=flood”
```

Here we have given the priorities (higher the number higher the priority) to TCP and UDP flow-rules, and in other case priority will be low. The server is listening TCP traffic on port 5002 and UDP on port 5003. It is also saying if destination address is 10.0.0.4 and it's a TCP packet send it to queue '*q1*' at '*eth4*', and if destination address is *10.0.0.4* and it's a UDP packet send it to queue '*q2*' at '*eth4*' else send it to output interface '*eth4'*. If mac address is this (*00:00:00:00:00:01*) send to output interface '*eth1'*, and so on respectively. The last one, this flow rule matches the every packet and it will flood in the switch.

3. ***Experimental results and analysis***

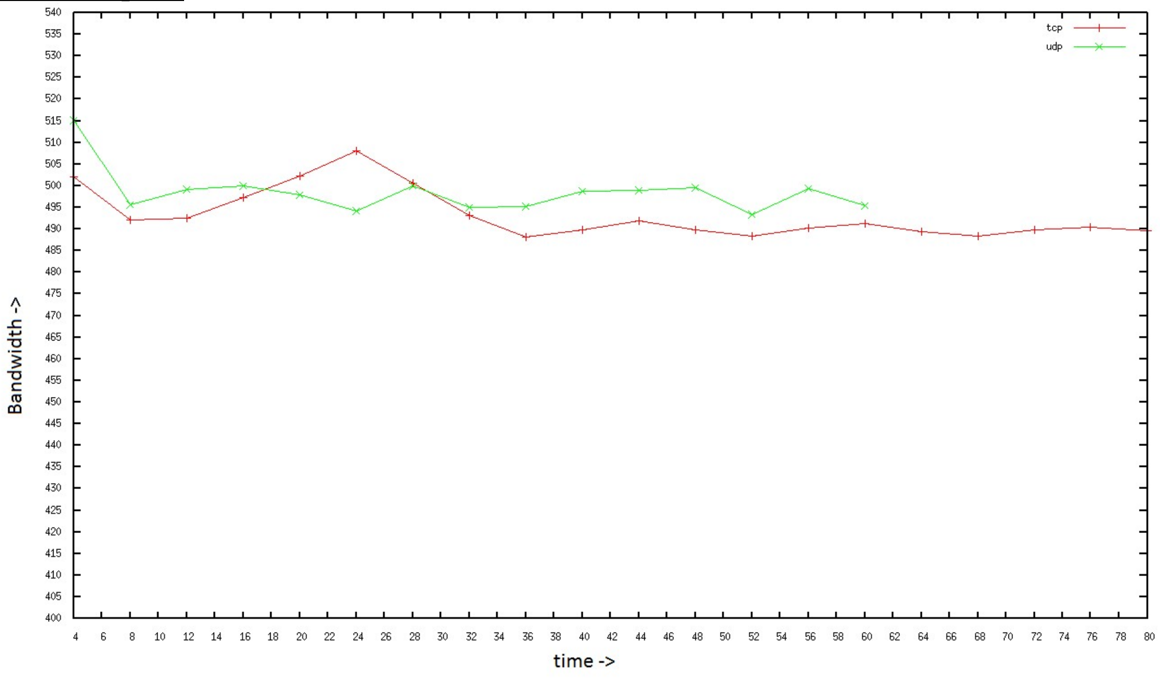


Average of 10 experiments for first part of assignment

For first part, within fraction of seconds, one after other we started TCP and UDP traffic simultaneously from H1 to H4, then HTTP traffic between H2 and H4, then TCP and UDP simultaneously from H3 to H4. Here HTTP works over TCP and as we transferred around 400 MB over HTTP(this big to observe), it took around 3 seconds in transferring HTTP traffic. As we first started traffic from H1, and bandwidth for UDP is limited to 512Mbps. There is some TCP traffic in the queue, and initially UDP is not using the entire bandwidth (i.e. restricted to 512Mbps). Afterwards traffic from H3 comes, and for H3 also, the bandwidth for UDP is limited to 512Mbps. Here intervals of 4 seconds are taken to analyse the traffic flow. At the end of first interval (0-4), we got the cumulative average bandwidth for TCP for h1 and h3 around 200, and for UDP, cumulative average bandwidth is around 900. Now there is a lot of traffic in queue, and queue is unable to handle that much of traffic. Because of 'tail drop' mechanism, some datagrams/packets are dropped. Loss of datagrams doesn't affect UDP, as UDP has no acknowledgement and congestion control policy; but loss of packets causes the TCP source to enter into slow start as it sees there is congestion in the queue. As congestion algorithm has triggered, TCP is deteriorating its performance. At this time UDP sees that there is more space available in queue, it takes advantage and takes over almost entire bandwidth. While TCP multiplicatively decreases its congestion window (cwnd), (at minimum it reaches to 1 MSS). As a result, whenever TCP tries to increase cwnd (additive), it sees congestion and again decreases it, so till the time, UDP data is being transferred from H1 and H3 simultaneously. So cumulative average per second of UDP is more than 950 Mbps and for TCP it is less than 10 Mbps.

At the interval 76-80, cumulative average per second for TCP starts increasing, because H1 started bit earlier, so the UDP stops bit earlier for H1, at this instant, there are 2 TCP traffics are transferred and one UDP traffic is transferred (limited to 512 Mbps). For next fraction of second, until UDP for H3 stops; maximum 512 Mbps bandwidth is being used for UDP in total and remaining TCP can use. So TCP is doing additive increase, so we can see the TCP cwnd increasing in interval 76-80. After 80th second UDP traffic is stopped for both the hosts, now TCP can used the entire bandwidth, so it is additively increasing its cwnd and used this till the end. At HTTP traffic is used for very-very short messages, it doesn't at all affect the bandwidth allocation.

<u>For second part,</u>



<u>Average of 10 experiments for second part of assignment</u>

There are separate queues for both TCP and UDP traffic, at egress port (s1-eth4). We are starting the traffic in the same way as we did for first part. As we can see both traffic is of always same speed as there are different queues for different traffic. Here as each of the two queues can use maximum of 512 Mbps. In the interval 4 – 6, UDP tries to send datagrams greater than the bandwidth allocated for UDP, resulting loss of datagrams. For TCP also, as some data more than cwnd tries to enter the queues. Using tail drop, that packets are being dropped and TCP is using, Multiplicative Decrease, Additive Increase to its cwnd throughout the transfer time. After 60 seconds, Even when UDP traffic stops, TCP cannot take over the entire bandwidth, as bandwidth for q1 (TCP queue) is limited to 512 Mbps. If we will set different queues for different protocols, we will see the performance in that way only. One is not affecting other because of the differentiation in queue because there is no overlap.

4. <u>Conclusion</u>

As we have observed that if there is only one queue to handle both TCP and UDP traffic, and if there are multiple clients which are transferring traffics simultaneously, then even if we limit bandwidth for each UDP link, still cumulatively UDP will take over the entire bandwidth, and there will be negligible bandwidth remaining for TCP. On the other hand if we set up different queues for both TCP and UDP, none of these queue will overlap the other queue or affect other queue. So the lesson we can derive is we should always use separate queues for TCP and UDP. So that when there is UDP traffic is being transferred with multiple hosts, the TCP traffic does not throttle.

5. ***Additional Content/Appendix***

To get separate columns for x and y from the tcp and udp result

```
:->cat <file name> | grep sec | head -15 | tr - " " | awk '{print $4,$8}' <output file>
```

//server side is getting two entries one from h1 and one from h3
Sum the values of corresponding rows

```
awk 'NR%2{printf "%s ",$0;next;}1' <src_file> > <int_file>; awk '{print $1, $2+$4}' <int_file> >
<res_file>
```

//to inorder to take the average

```
awk ‘NR==FNR{a[NR]=$2;next}{print $1, $2+a[FNR]}’ <file1> <file2> > <res_file>
```

For graph plotting:

```
****install gnuplot****
sudo apt-get install gnuplot-nox
sudo apt-get install gnuplot-x11
***********************
```

//file should have two separate columns in x and y.

```
:->gnuplot
gnuplot>
        plot "<file name> title "<title>" with linespoints
        set xrange [1:15]
        set xtics 1,1,15
        set yrange [0:1000]    //or [900:1000] or [ymin:ymax]
        set ytics 0,2,1000
        replot
```