Throughput Performance Comparison of TCP and MPTCP

Burhan Qayoom, School of Computer and Information Sciences,
University of Hyderabad, Telangana, India.
qayoom.burhan@gmail.com

I. Abstract

Regular TCP is based on the idea of a host having a single connection with the network. Nowadays, we have multi homed machines that have more than one interface to connect to a network. Regular TCP does not incorporate the use of more than one connection for a single host. This leads to wastage of resources if there is a possibility of utilizing more than one network connections for traffic flow. Multipath TCP (MPTCP) is an Internet Engineering Task Force (IETF) standard that leverages a multipath approach to accumulate the bandwidth of multiple connections. MPTCP runs on Transport layer and uses it as the subflow protocol. In this paper, we present a throughput comparison of regular TCP and Multipath TCP and show how MPTCP produces a higher throughput in the transmission of data. We have carried out three experiments to compare the throughput in TCP and MPTCP. The first experiment consists of a server and a host comparing throughput in TCP and MPTCP with two links. The second experiment uses four hosts for the same procedure. The second experiment follows the second experiment but the server with four links.

Keywords: TCP, multi homed, MPTCP, subflow.

II. Introduction

MPTCP is a major extension to TCP that takes advantage of multiple paths available in the network. This not only increases the throughput but also makes the transmission of the data more robust. In addition, no modification is required at the application layer. MPTCP can be used in today's Internet infrastructure with affecting the functioning of the existing system. These things are possible because MPTCP used TCP as it subflow protocol. The benefits of MPTCP regarding throughput may come at some cost of of data delivery delay and jitter. This happens if the delays in the subflows differ considerably.

MPTCP connection between two hosts is established using three-way handshake with TCP. The source sends a MP_CAPABALE option in the SYN packet to indicate the source can perform MPTCP. The destination replies SYN ACK packet which also contains MP_CAPABALE option. Finally, the source replies with ACK. Once the connection has been established any of the hosts can create a subflow if it finds a new path to the destination by sending a MP_JOIN option in a new SYN packet. The destination sends a MP_JOIN along with SYN ACK on finding the new path to the source. The sender acknowledges the reception of SYN ACK with MP_JOIN with an ACK with MP_JOIN. Finally, the destination acknowledges this message with ACK message. Data can be transferred now between the two hosts.

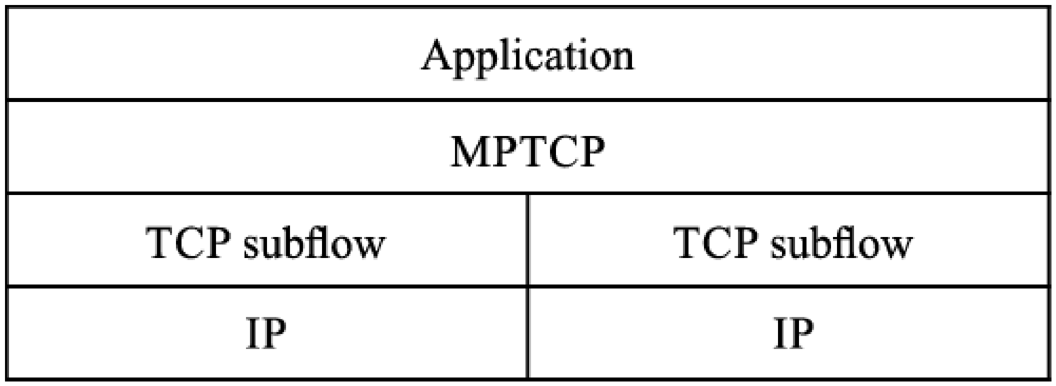


Fig 1: Architecture of MPTCP

The architecture of MPTCP is shown in the figure 1. MPTCP provides an abstraction to the application from the various TCP subflows and for an application all the data seem to be coming from one single connection. In order to coordinate between multiple subflows, MPTCP provides two level of sequence number: Data sequence number (DSN) at the MPTCP session level and a regular sequence number at MPTCP subflow session level. A regular sequence number like in TCP ensures reliability in a TCP subflow and DSN ensures data reliability among the subflows.

A. Mininet

Mininet is a network emulator used for emulating a large network on a single machine. We can create virtual hosts, switches, controllers in the mininet emulator. It runs real Linux network code including the kernel and the network stack software for the operation and it supports OpenFlow protocol for Software defined networking.

B. TCP Performance Issues

In early networking days, very less traffic was sent over the Internet. TCP was doing well for the amount of traffic generated. However, due to technology boom, the amount of traffic generated and sent over the Internet has increased many folds. This has brought about some of the issues:

1. Head-of-line blocking:

It is a performance issue caused by a packet blocking the packets in a queue to wait in line for transmission. This happens when the output port of the packet to be forwarded is busy. TCP allows to reorder the segments and recreate the byte stream which makes sure a in-order delivery and reliability. The TCP connection is susceptible to congestion control leading to fast retransmit and slow start. This can lead to head-of-line blocking. Similarly, the subflows in MPTCP can suffer from this problem. Multipath TCP, like TCP, allows reordering of the packets and in order delivery. A low delay subflow packet may have to wait for the high delay packet before they are ordered. This leads to delay in the packet delivery. In MPTCP, it is the duty of the scheduler to schedule the data packets across the subflows that ensures minimum delay.

2. Receive-Window

A window is maintained by the receiver to reorder the out of delivery reception of the packets. This window size is proves to be small when the reordering in MPTCP is done both at the TCP subflow level as well as the MPTCP level. The receiver window has to accommodate the out of delivery packets at the MPTCP level. Thus, the size of receiver window size is critical for a good performance.

III. Experimental Setup

The mininet emulator is installed on the Linux machine. To get the source code:

```
git clone git://github.com/mininet/mininet
```

The command to install mininet :

```
mininet/util/install.sh -a
```

After installing Mininet emulator, install the MPTCP Linux kernel. The link to download MPTCP v0.94:

```
sudo apt-key adv --keyserver hkp://keys.gnupg.net --recv-keys 379CE192D401AB61
sudo sh -c "echo 'deb https://dl.bintray.com/cpaasch/deb stretch main' >
/etc/apt/sources.list.d/mptcp.list"
sudo apt-get update
sudo apt-get install linux-mptcp
```

The MPTCP should be enabled to run multiple subflows in TCP. This is done by the following command:

```
sysctl -w net.mptcp.mptcp_enabled=1
```

In order to disable MPTCP, we use:

```
sysctl -w net.mptcp.mptcp_enabled=0
```

The server and clients are run in the mininet using xterm terminal emulator. This is done by writing the xterm command followed by the hosts which needs to be run.

Iperf: Iperf is a command line tool for network diagnosis by measuring the throughput between a server and a client. Iperf is used in the experiments to send traffic from a client to a server. The command to make a host a server is:

```
Iperf  -s
```

The command to run a host as a client is:

```
Iperf  -c  [ip address of the server]
```

The scripts to carry out the experiments is written in python and a bandwidth limit of 1Gbps is applied.

A. Experiment 1

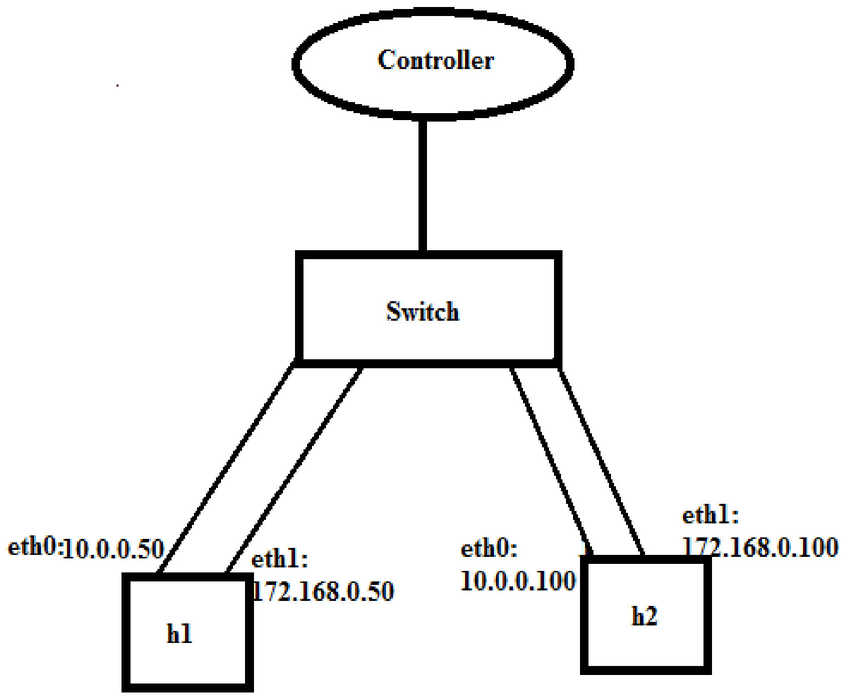


Figure 2: Experiment 1 Topology for MPTCP with 2 interface and TCP

The topology used in experiment 1 is shown in figure 2. Since, the experiment is carried out in mininet, the topology consists of a SDN controller with an open switch using southbound API i.e., OpenFlow. In addition, we have two hosts h1 and h2. H1 is run as server throughout the experiment and h2 as the client. The script after running in the mininet emulator will configure IP addresses and interface tables. The experiment consists of three sub experiments.

In sub experiment 1, h1 and h2 are connected to the switch with two links each having IP addresses as shown in the figure 2. Before sending the traffic from the client h2 to server h1, enable the MPTCP kernel in the machine. The server and client terminal are run using the xterm commands as:

```
xterm h1 h2
```

This is followed by making host h1 as server and h2 as client using iperf command.

For server: `iperf –s`

For client: `iperf –c 10.0.0.50 –t 100 –i 10`

-t defines the total time for which client will send traffic to the server and –i defines the time interval between each throughput calculation. The traffic sent is sent through subflows created by

the MPTCP with each subflow running TCP. The throughput of the traffic is recorded for analysis.
In sub experiment 2 to send traffic using regular TCP, disable the MPTCP kernel in the machine. In this case, one of the link in the topology does not work when traffic is sent from client to the server using iperf. This gives the TCP throughput.

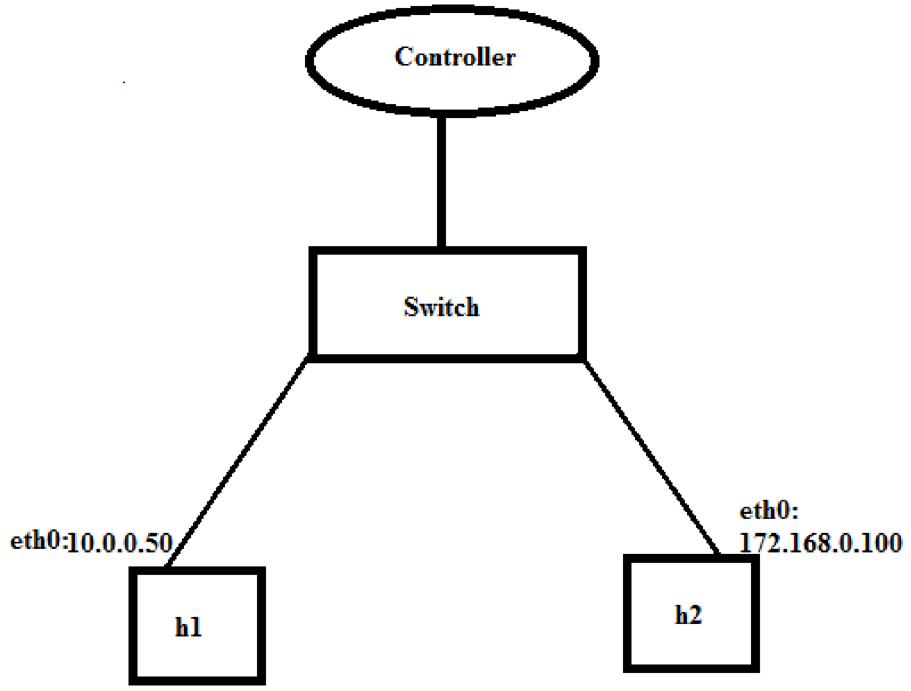


Figure 3: Experiment 1 Topology for MPTCP with 1 interface

The third sub experiment sends traffic from client h2 to server h1 using only one link while MPTCP is enabled. This is done by running a python script with only a single link between h1 and switch with IP address 10.0.0.50 at interface eth0 and only one link between h2 and switch with IP address 10.0.0.100 at interface eth0. The traffic is sent from h2 to h1 and throughput is recorded.

B. Experiment 2

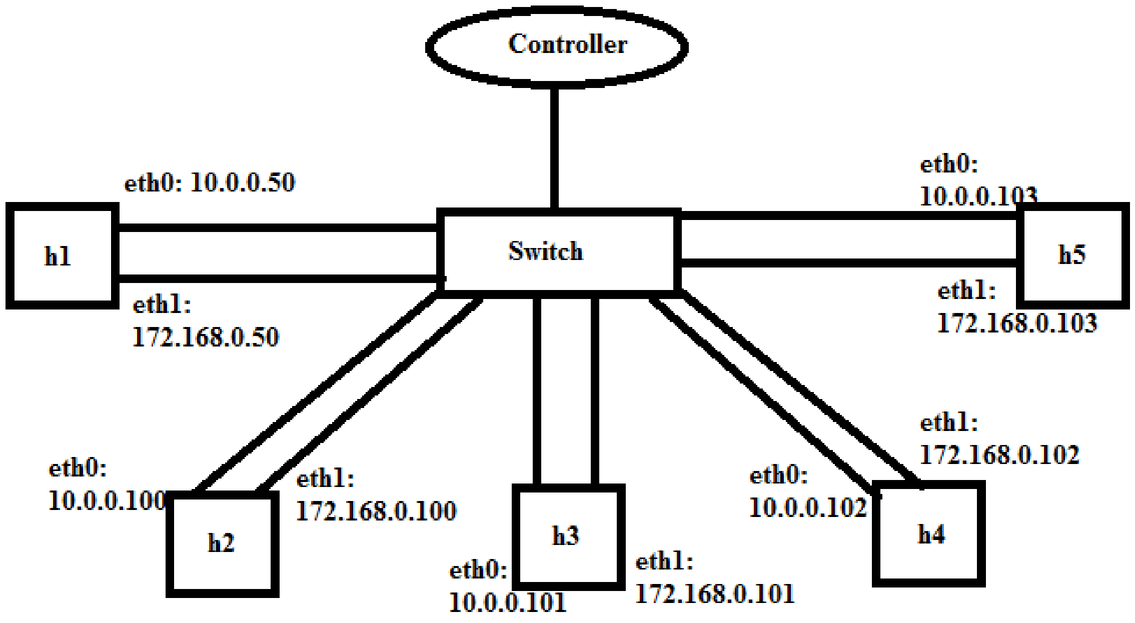


Figure 4: Experiment 2 Topology for MPTCP with 2 interface and TCP

Experiment 2 in contrast to experiment 1 consists of four clients h2, h3, h4, h5 instead of a single client. H1 is used to act as a server. The IP addresses assigned to the interfaces in the hosts is shown in the figure. IP addresses are configured and interface tables are created after the script is run in the mininet emulator. This experiment consists of three sub experiments.

In sub experiment 1, to send traffic to the server, MPTCP is first enabled. The hosts terminals are run using the xterm command

```
Xterm h1 h2 h3 h4 h5
```

H1 is created as a server and rest of the hosts as the clients using

Iperf –s, for server

Iperf –c 10.0.0.50 –t 100 –i 10, for client.

The command for client is run at each client simultaneously. The traffic is sent to the server and the throughput is recorded.

In sub experiment 2, the throughput achieved for the same topology using TCP can be obtained by disabling the MPTCP kernel. The traffic generated now is using only TCP. In this case, only one link is used to send and receive the traffic. The throughput achieved while sending the traffic from all the hosts to the server is recorded.

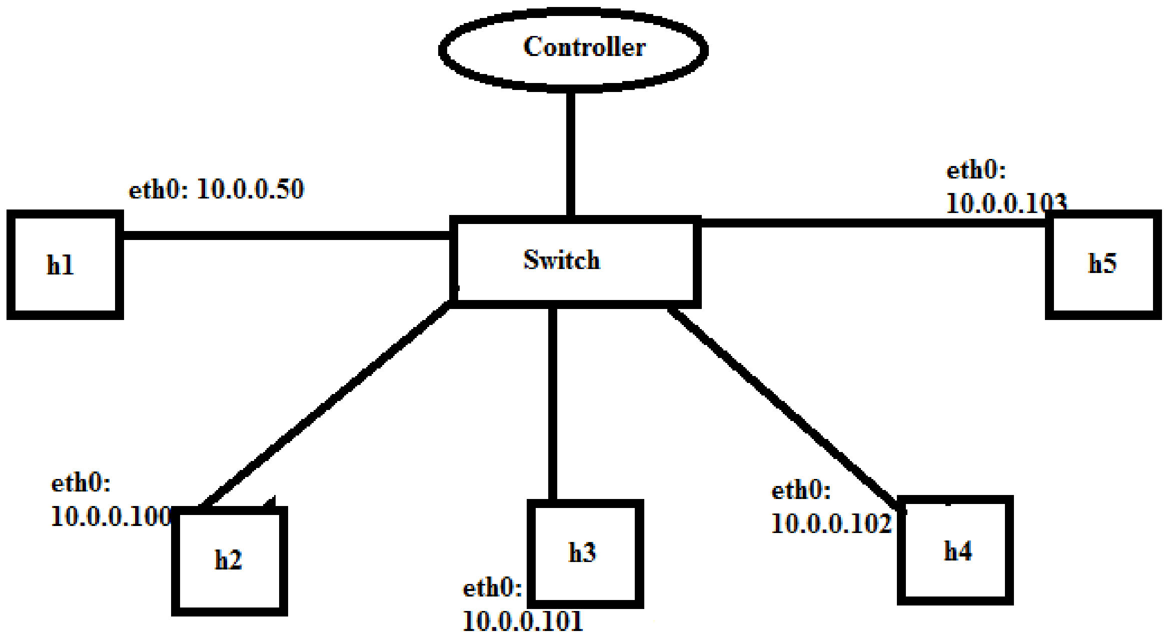


Figure 5: Experiment 2 topology for MPTCP with one interface

The throughput calculation when using MPTCP with a single interface is achieved by running a script that has the topology as shown in figure 5. MPTCP is enabled to send traffic using subflows. The traffic is generated from all the clients to the server using iperf command after running the terminal for each host using xterm command. The throughput is recorded for the experiment.

C. Experiment 3

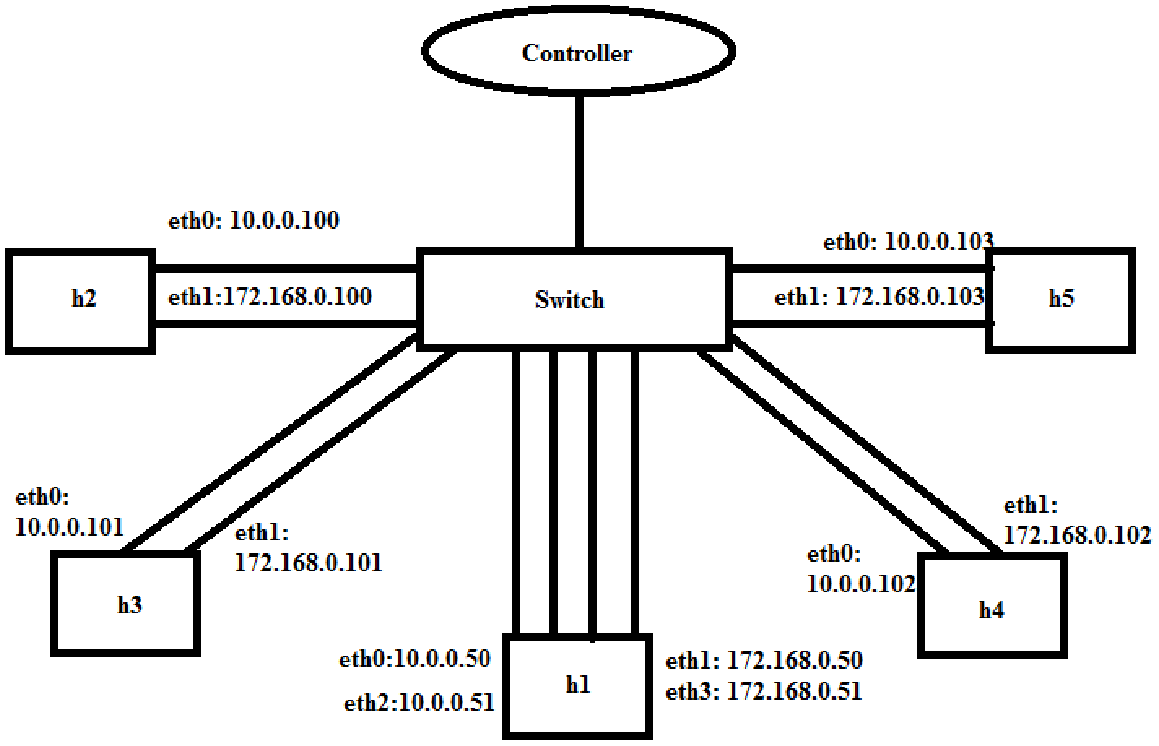


Figure 6: Experiment 3 topology

Experiment 3 is an enhancement of experiment 2 in which the server host has 4 links to receive the data. However, the traffic generated by the clients will only be received by two links in the server. To solve this problem, IP aliasing is used. IP aliasing is a technique in which more than one IP addresses are assigned to a single interface. This results in multiple connections to the network. In the experiment, IP aliasing allows to use more than one IP address for each interface in the client hosts. These IP addresses allow to send the traffic to all the server links including the two links that could not receive without aliasing.

The general format for adding an alias to the IP address is:

Ifconfig host id – interface id : alias id

The reception of packet at all the server links is made sure by adding three alias at each client interface :

Ifconfig host id – interface id : 0
Ifconfig host id – interface id : 1
Ifconfig host id – interface id : 2

The script is run configuring the IP address as well as alias IP address at client's interfaces. In addition, interface tables are created. The window for each host is run using

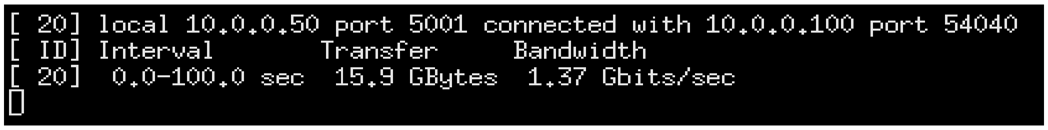
xterm h1 h2 h3 h4 h5

The traffic is sent to the server from all the hosts simultaneously. The data is received at all the four server interfaces. Finally, the throughput is recorded.

IV. Experimental Results and Analysis

A. Experiment 1

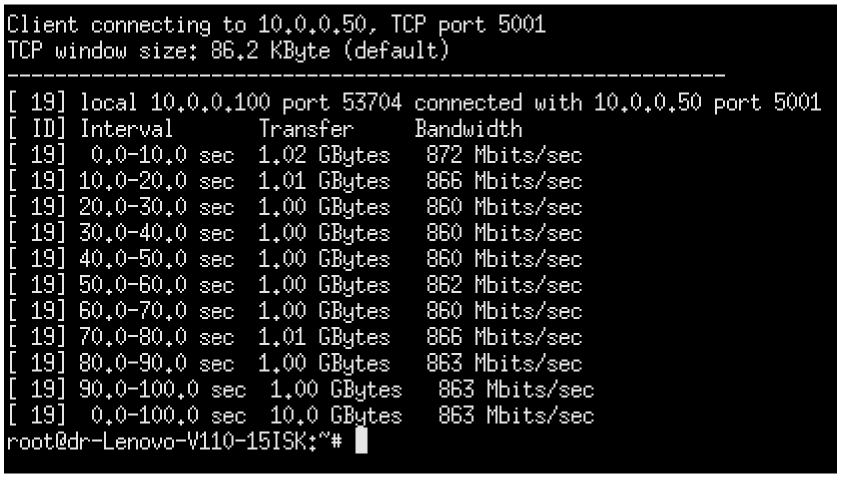
The script for the experiment 1 is run in the mininet emulator and the iperf command is used to send the data to the server and record the throughput.
The throughput received when is using MPTCP with two interfaces is:



```
[ 20] local 10.0.0.50 port 5001 connected with 10.0.0.100 port 54040
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-100.0 sec  15.9 GBytes  1.37 Gbits/sec
```

Although, the bandwidth was restricted to 1Gbps, the throughput achieved is 1.37 Gbps. This is because of the subflows created by the MPTCP for each interface.

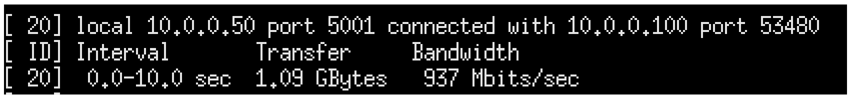
The throughput received when using TCP is:



```
Client connecting to 10.0.0.50, TCP port 5001
TCP window size: 86.2 KByte (default)
------------------------------------------------------------
[ 19] local 10.0.0.100 port 53704 connected with 10.0.0.50 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 19]  0.0-10.0 sec  1.02 GBytes   872 Mbits/sec
[ 19] 10.0-20.0 sec  1.01 GBytes   866 Mbits/sec
[ 19] 20.0-30.0 sec  1.00 GBytes   860 Mbits/sec
[ 19] 30.0-40.0 sec  1.00 GBytes   860 Mbits/sec
[ 19] 40.0-50.0 sec  1.00 GBytes   860 Mbits/sec
[ 19] 50.0-60.0 sec  1.00 GBytes   862 Mbits/sec
[ 19] 60.0-70.0 sec  1.00 GBytes   860 Mbits/sec
[ 19] 70.0-80.0 sec  1.01 GBytes   866 Mbits/sec
[ 19] 80.0-90.0 sec  1.00 GBytes   863 Mbits/sec
[ 19] 90.0-100.0 sec  1.00 GBytes   863 Mbits/sec
[ 19]  0.0-100.0 sec  10.0 GBytes   863 Mbits/sec
root@dr-Lenovo-V110-15ISK:~#
```

In this case, the throughput received is less than the limited bandwidth which justifies the use of TCP having only one connection.

The throughput received when using MPTCP with one interface is:



```
[ 20] local 10.0.0.50 port 5001 connected with 10.0.0.100 port 53480
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-10.0 sec  1.09 GBytes   937 Mbits/sec
```

In this case, MPTCP consists of only one subflow. Hence, the throughput achieved is less than the limiting bandwidth.

The bar graph comparing the three cases of experiment 1 is:

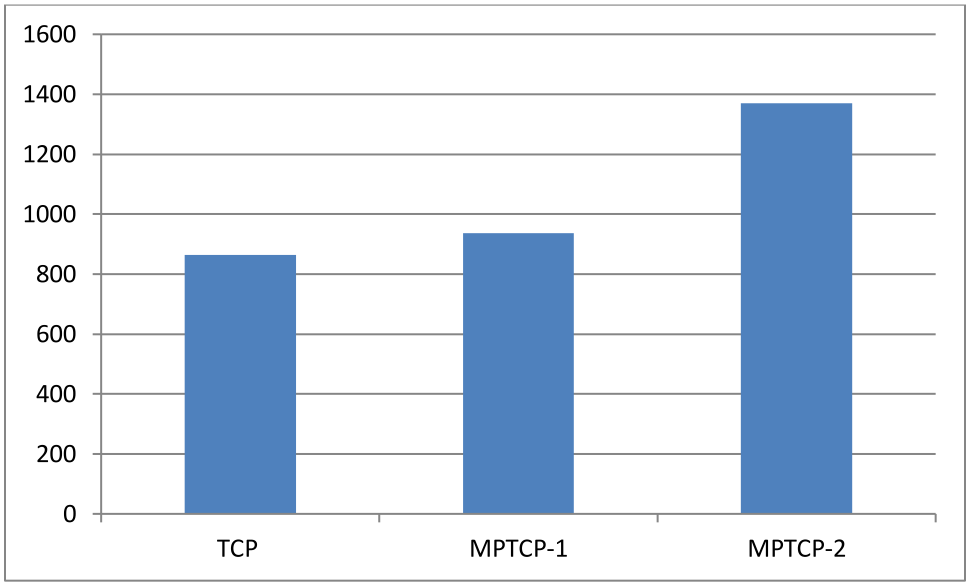
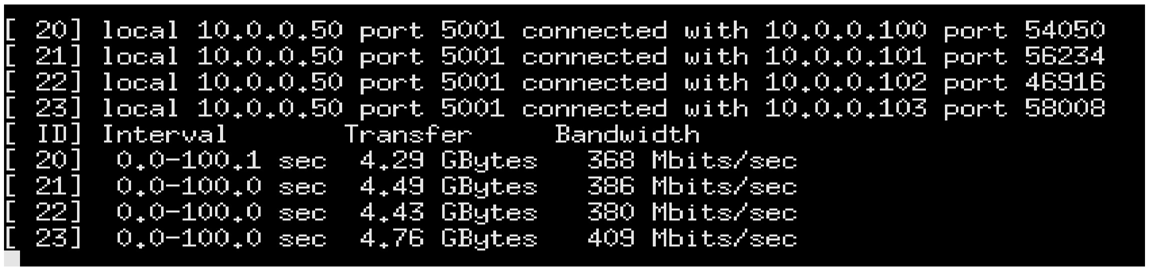


Table 1: Comparing TCP, MPTCP-1, MPTCP-2 for experiment 1

From the analysis, MPTCP with two interfaces outperforms TCP and MPTCP in the bandwidth utilization. Moreover, MPTCP with one interface performs slightly better than the TCP. This is because of the better congestion algorithms employed by the MPTCP.

B. Experiment 2

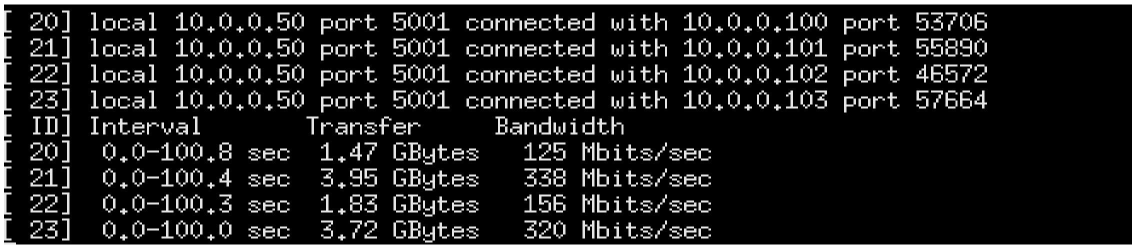
The scripts for running the experiment 2 has the topologies as shown in figure 4 and figure 5. The scripts are run in the mininet emulator and the throughput is obtained using the iperf command. In this experiment, the clients simultaneously send the data to the server. The throughput achieved when using MPTCP with two interfaces is:



```
[ 20] local 10.0.0.50 port 5001 connected with 10.0.0.100 port 54050
[ 21] local 10.0.0.50 port 5001 connected with 10.0.0.101 port 56234
[ 22] local 10.0.0.50 port 5001 connected with 10.0.0.102 port 46916
[ 23] local 10.0.0.50 port 5001 connected with 10.0.0.103 port 58008
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-100.1 sec  4.29 GBytes   368 Mbits/sec
[ 21]  0.0-100.0 sec  4.49 GBytes   386 Mbits/sec
[ 22]  0.0-100.0 sec  4.43 GBytes   380 Mbits/sec
[ 23]  0.0-100.0 sec  4.76 GBytes   409 Mbits/sec
```

The throughput achieved in this case is of the order of 400 Mbps for each client at the server.

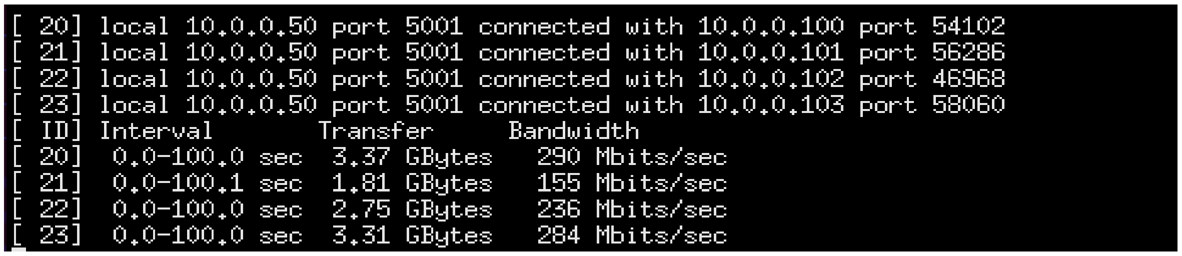
The throughput achieved when using TCP is:



```
[ 20] local 10.0.0.50 port 5001 connected with 10.0.0.100 port 53706
[ 21] local 10.0.0.50 port 5001 connected with 10.0.0.101 port 55890
[ 22] local 10.0.0.50 port 5001 connected with 10.0.0.102 port 46572
[ 23] local 10.0.0.50 port 5001 connected with 10.0.0.103 port 57664
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-100.8 sec  1.47 GBytes   125 Mbits/sec
[ 21]  0.0-100.4 sec  3.95 GBytes   338 Mbits/sec
[ 22]  0.0-100.3 sec  1.83 GBytes   156 Mbits/sec
[ 23]  0.0-100.0 sec  3.72 GBytes   320 Mbits/sec
```

In this case, the throughput received is much less than the MPTCP with two interfaces. This is evident because of no multiple TCP subflows to send and receive data. The average throughput received is 235 Mbps

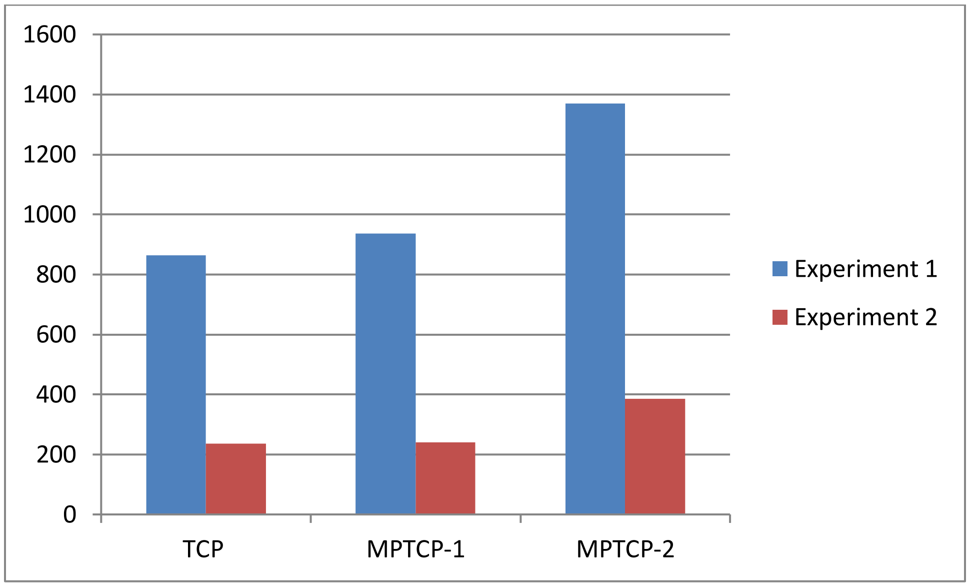
The throughput achieved when using MPTCP with one interface is:



```
[ 20] local 10.0.0.50 port 5001 connected with 10.0.0.100 port 54102
[ 21] local 10.0.0.50 port 5001 connected with 10.0.0.101 port 56286
[ 22] local 10.0.0.50 port 5001 connected with 10.0.0.102 port 46968
[ 23] local 10.0.0.50 port 5001 connected with 10.0.0.103 port 58060
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-100.0 sec  3.37 GBytes   290 Mbits/sec
[ 21]  0.0-100.1 sec  1.81 GBytes   155 Mbits/sec
[ 22]  0.0-100.0 sec  2.75 GBytes   236 Mbits/sec
[ 23]  0.0-100.0 sec  3.31 GBytes   284 Mbits/sec
```

In this case, the average throughput is higher than the average throughput when using TCP. The average throughput achieved is 241 Mbps. This is almost similar to the TCP average throughput. The MPTCP congestion control algorithm does not really enhance the bandwidth is this case.

The comparison of the experiment 1 with experiment 2 in a bar graph is:

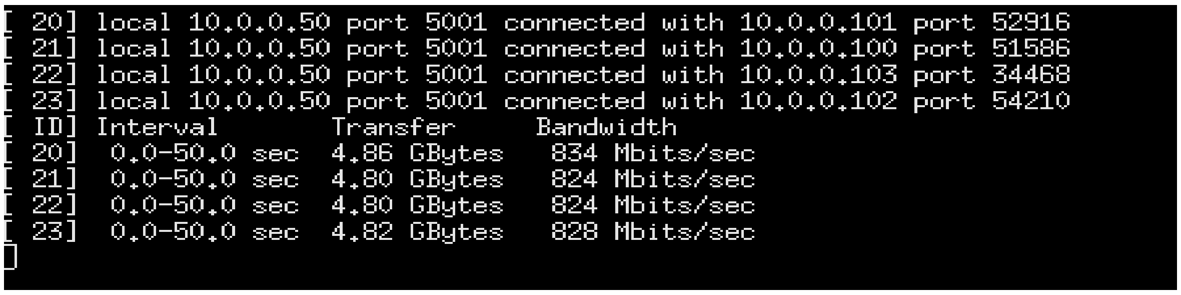


On comparing the experiment 1 with experiment 2, the throughput achieved decreases when the number of clients sending traffic to the server increases at constant number of links. This is because the traffic on the two links of the server increases substantially which hinders the clients to send from sending a lot of data as compared to experiment 1 where there is no hindrance to a single client. The client can send much data as compared to experiment 2. However, the comparison of MPTCP with one interface with TCP shows a variation. In experiment 1, there is a noticeable difference in the higher throughput in MPTCP with one interface and TCP, however in experiment 2, the congestion control algorithm does not do much work to enhance the bandwidth in MPTCP with one interface as compared to TCP.

C. Experiment 3

The experiment 3 topology is shown in figure 6. On running the script, 4 links are added to the server and two IP address at each interface of the client along with 3 alias IP addresses are added at each interface.

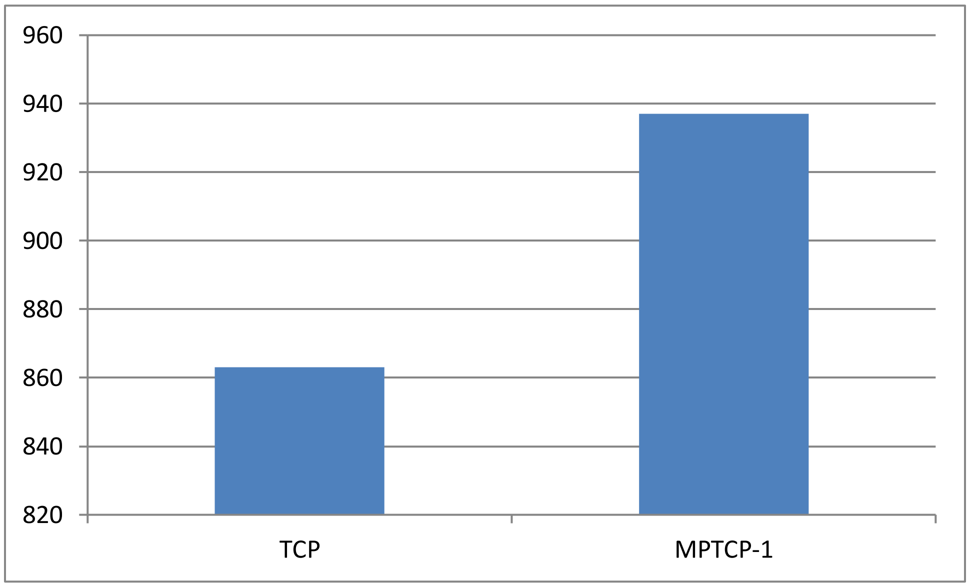
The throughput achieved is:



```
[ 20] local 10.0.0.50 port 5001 connected with 10.0.0.101 port 52916
[ 21] local 10.0.0.50 port 5001 connected with 10.0.0.100 port 51586
[ 22] local 10.0.0.50 port 5001 connected with 10.0.0.103 port 34468
[ 23] local 10.0.0.50 port 5001 connected with 10.0.0.102 port 54210
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-50.0 sec  4.86 GBytes   834 Mbits/sec
[ 21]  0.0-50.0 sec  4.80 GBytes   824 Mbits/sec
[ 22]  0.0-50.0 sec  4.80 GBytes   824 Mbits/sec
[ 23]  0.0-50.0 sec  4.82 GBytes   828 Mbits/sec
[]
```

The average throughput received is this case is 828 Mbps.

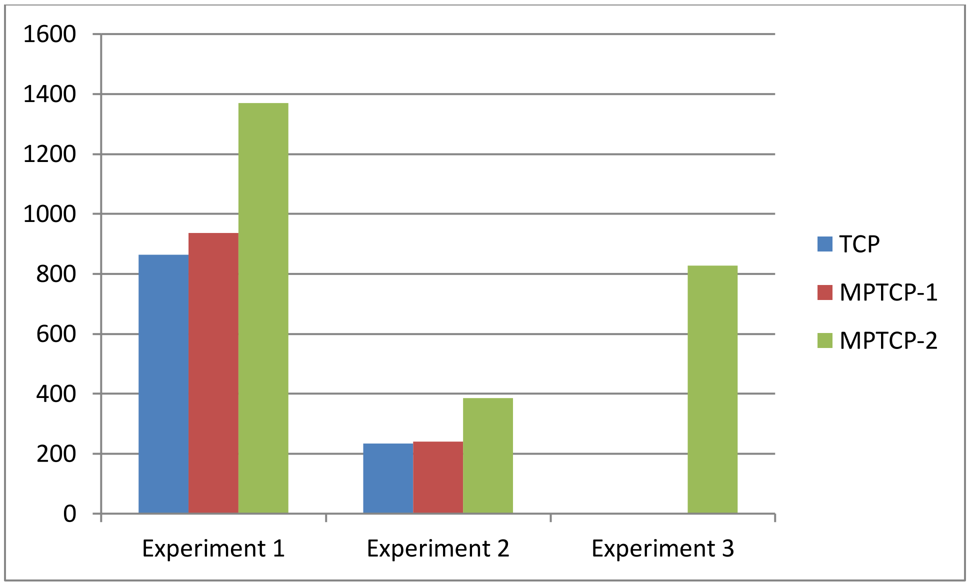
On comparing the experiment 3 result with experiment 2, we get:



It is observed from the bar graph that the throughput achieved in experiment 2 is more than twice the throughput achieved in experiment 2. This is because of the number of links at the server increases by 2 and the MPTCP congestion control algorithm along with scheduler did well when the number of link were more.

V. Conclusion

This paper presents 3 experiments conducted in the mininet emulator to compare the throughput achieved by varying the number of clients and the number of multipath links. In all the experiments, MPTCP outperformed the TCP in achieving the throughput utilization. A comparison between the three experiments can be drawn as:



The conclusion drawn from performing the three experiments for different cases is that as the number of multipath links increases between a client and a host, the throughput achieved is increased. Similarly, the throughput decreases when the number of clients to a server increases for a constant number of links. This is because of the congestion. Although, experiment 3 shows a lower throughput than experiment 1, the addition of two more links only enhances the throughput performance. The less throughput of experiment 3 is because of the 4 clients sending traffic to the server.

VI. Acknowledgements

The experiment work was supported by Kamran Mehboob and Manisha Kumari.