# ITS 413 Internet Technologies and Applications

*Assignment: Phase 3 Report*

By:

Nabwan Prukpaiboon 5222770471

Smith Gulati 5222791576

Date: March 28, 2012

By submitting this report all members of the group listed above agree that each member has contributed approximately equal amounts to designing and performing experiments, as well as to preparing this report. All members agree that this report accurately reflects the experiments conducted by the group members, and is their own work (not works of other groups).

Sirindhorn International Institute of Technology

Thammasat University

## Aims

1. To determine how different data link and transport control parameters and scenarios impact on the performance (mainly throughput) of the transport protocols in Ethernet link using iperf.
2. To state the hypothetical result of the experiments and proof or disproof the statement.

## Network Diagram

We have used only wired Ethernet topology in our experiments. The reason of using only wired network is that wireless link has many link issues that may impact the harmony and flow of the data across the network. The issues are for instance, channel interference, obstacles, medium access control, distance of nodes from access point etc.

**Wired Ethernet Topology**

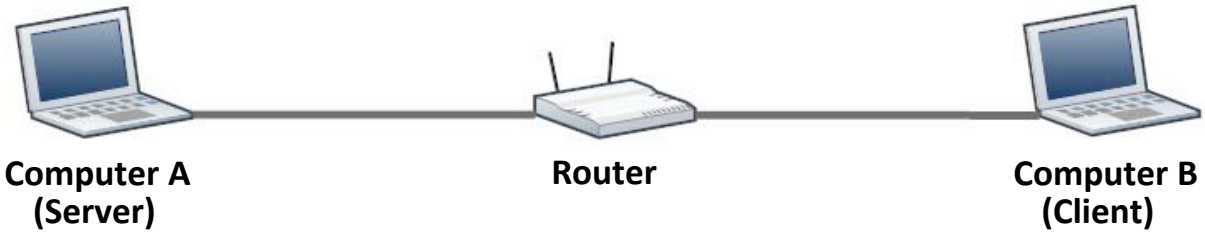


**Figure 1 Wired Ethernet Topology**

As seen in the figure1, Computer A is chosen as the server because it has lower specifications. The reason of doing so is because computer with lower specifications tends to generate data to be sent by iperf slower than the computer with higher specification. Using computer with higher specifications as client can decrease the overhead time in generating these data.

# Equipment Specifications

Following are the devices we used in the course of experiments:

**Computer A (Server)**

| | |
|---|---|
| Model: | Acer Aspire1410 |
| Processor: | Intel Celeron 723 @ 1.20 GHz |
| Memory: | 2 GB DDR2 |
| LAN Interface: | Atheros AR8131 PCI-E Gigabit Ethernet Controller (Up to 1 Gbps) |
| WLAN Interface: | Acer Nplify 802.11b/g/Draft-N |
| Operating System: | Ubuntu 11.10 |

**Computer B (Client)**

| | |
|---|---|
| Model: | Dell Studio 1435 |
| Processor: | Intel Core 2 Dui CPU T6400 @ 2.00 GHz |
| Memory: | 4 GB DDR2 |
| LAN Interface: | Broadcom Netlink (TM) Gigabit Ethernet (Up to 1 Gbps) |
| WLAN Interface: | DELL Wireless 1397 802.11b/g |
| Operating System: | Ubuntu 11.10 |

**Router**

| | |
|---|---|
| Model: | Linksys Wireless – G Broadband Router WRT54GL |
| Ports: | 4 10/100 RJ-45 Switched Ports |
| Firmware: | OpenWrt penWrt Backfire 10.03.1 / LuCI 0.10.0 Release (0.10.0) |
| Kernel Version: | 2.4.37.9 |
| Ethernet Standards: | IEEE 802.3 (Ethernet), IEEE 802.3u (Fast Ethernet) |
| Wireless Standards: | IEEE 802.11g, IEEE 802.11b (Frequency 2.4 GHz) |

**Ethernet Cable**

Unshielded Twisted Pair CAT5 cables 1000BASE-T Ethernet (Gigabit Ethernet)

## Parameters

Following are the parameters that are to be considered in the experiments mentioned in the experiments:

| Parameters | Value |
|---|---|
| **Wireless LAN Data Rate** | 54 Mbps |
| **Ethernet Data Rate** | 100 Mbps |
| **Transport Protocol** | TCP and UDP |
| **Sending Time** | 10 seconds (Default) |
| **Congestion Control Algorithm** | Reno, CUBIC, Vegas, BIC |
| **Packet Drops** | 0% of the data sent |

## Experiments and Results

# Experiment 1

### Hypotheses

1. The increase of percentage of packet drops in a network decreases the average throughput of the network.
2. The change of sending time (-t option in iperf) affects the average throughput of the network.
3. Different TCP congestion avoidance algorithm results in different average throughput of the network.

### Experiment Setup and Background Theory

Transmission Control Protocol (TCP) is a complex transport protocol working alongside Internet Protocol (IP). It is a stream oriented transport protocol providing reliable and ordered delivery of a stream of bytes. One of the main aspects of TCP is congestion control. TCP has mechanisms in controlling and managing the problem of network congestion. These mechanisms are for instance, slow start, congestion avoidance, fast retransmit etc. In addition TCP has many congestion avoidance algorithm variations like Tahoe, Reno, Vegas etc.

The test was started by setting the percentage of packet drops on the server by using the following command in the terminal:

```
$ sudo iptables –A INPUT –m statistic –-mode random –-probability 0.03 –j DROP
```

where, the percentage of packet drops is 3%.

Then the TCP congestion avoidance algorithm is set on the client computer by using the following command in the terminal:

```
$ sudo sysctl net.ipv4.tcp_congestion_control=vegas
```

where, vegas is the congestion avoidance algorithm.

This experiment was performed by using the following parameters:

| | |
|---|---|
| **Sending time** | [10, 20, 50, 120] seconds |
| **Packet Drop** | [0, 1, 3, 5, 10] % |
| **Congestion Avoidance Algorithm** | CUBIC, Reno, Vegas, BIC |
| **Other** | Default |

The reason that we chose the percentage of packet drops as above because beyond the packet drops of 10%, the throughput is about 1.5 to 4 Mbps. If an internet user payed for a 100 Mbps link and gets this amount of throughput then it is already of no use.

The iperf commands used in this experiment is as follow:

On client

```
$ iperf -c 192.168.1.162 -t 120
```

On server

```
$ iperf –s
```

For each reading, we take the average throughput from 3 tests. The reason is to get more comprehensive test result, in addition, by running more tests we can know if any particular test fails.

## Test Result

From the experiment we performed we came out with the following plots:

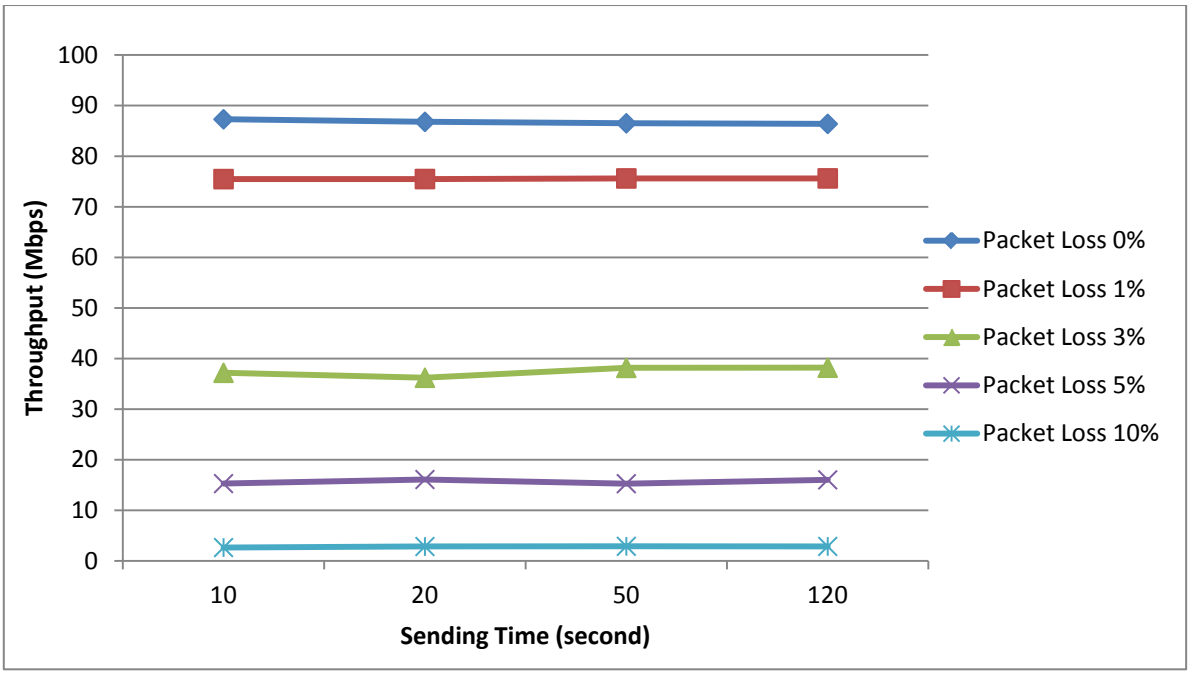


**Figure 2 Average throughputs on different packet loss varying the sending time**

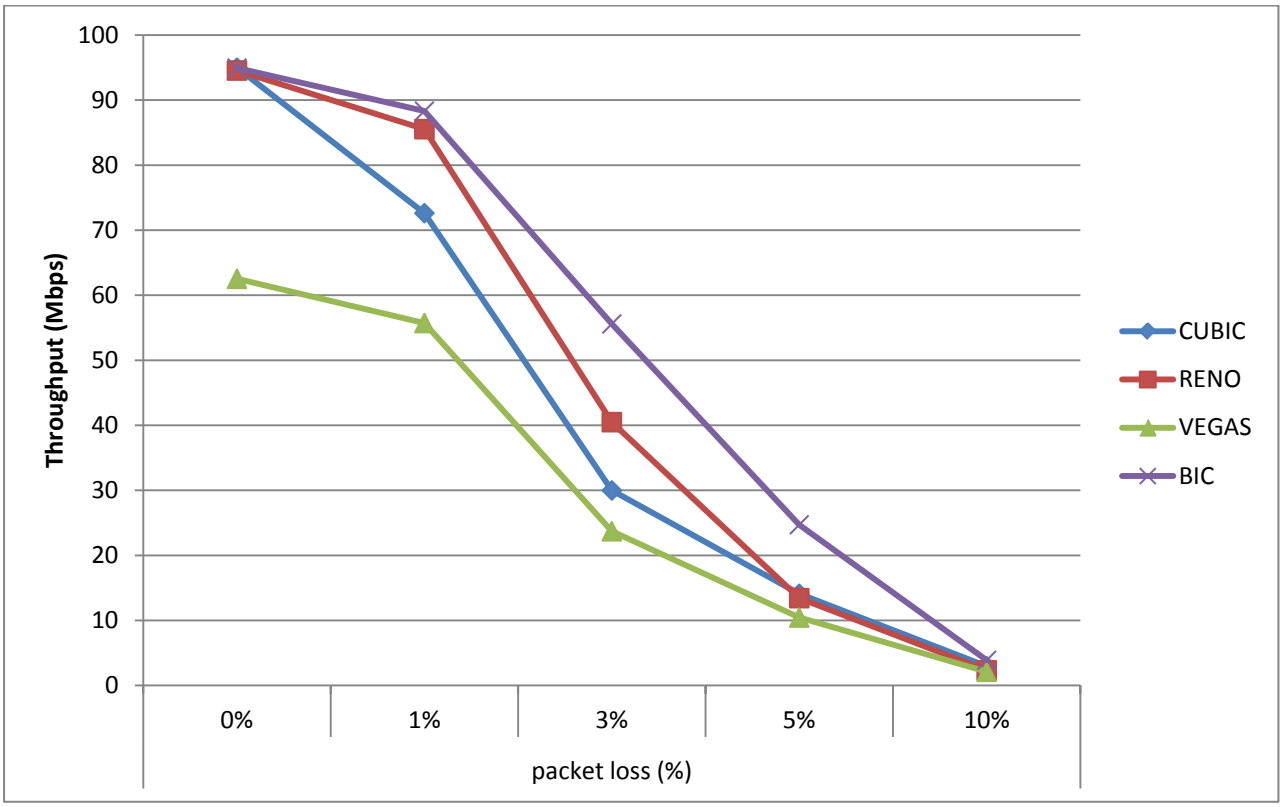


**Figure 3 Average throughputs on different percentage of packet loss on different congestion avoidance algorithms**

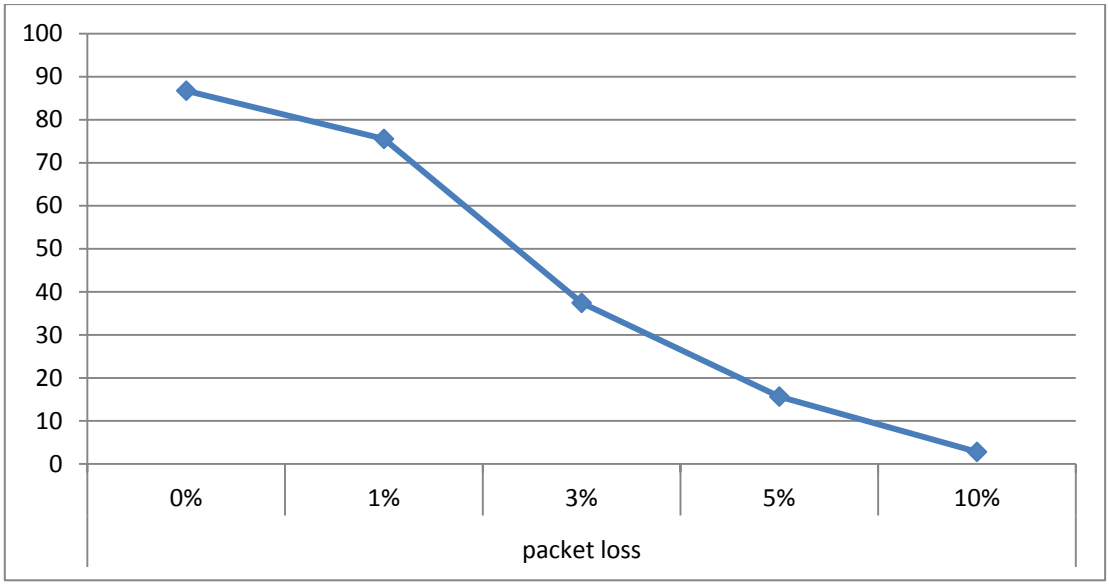


**Figure 4 Average throughputs on different percentage of packet loss ignoring congestion avoidance algorithms**

## Observation and Conclusion

According to Figure 3, the result of experiment has shown that packet loss percentage significantly affect the throughput of the network. Without any packet loss, CUBIC, Reno, and BIC algorithm can achieve almost the same maximum throughput at 95 Mbps approximately while with 1% packet loss the maximum throughput drops to 72, 85 and 88 Mbps respectively. To make it clearer, considering the average throughput among these four algorithms, the average maximum throughput that we can achieve is 86.75 Mbps at 0% packet loss, 75.55 Mbps at 1% loss, 37.44 Mbps at 3% loss, 15.67 Mbps at 5% loss, and 2.81 Mbps at 10% packet loss. According to this scenario, the average maximum throughput is reduced to half with only three percent of packet loss.

For the second hypothesis, the experiment is arranged so that the TCP session is run with different sending time. According to Figure 2, the result has shown that within the same percentage of packet loss, sending time hardly affect the throughput of the network. Considering the experiment with packet loss percentage of 3 percent which has the most vary result, minimum throughput achieved is 36.20 Mbps sending at 20 seconds and maximum throughput is achieved is 38.21 Mbps sending at 120 seconds. Therefore, the difference between maximum and minimum throughput is 2.01 Mbps which is relatively low comparing with average throughput of 37.44 Mbps.

For the third hypothesis, according to Figure 3, the result has shown that each of these four algorithms gives out quite different result but all with the same trend as stated in hypothesis 1. CUBIC and Reno algorithm which are two default algorithms for Ubuntu Linux

11.10 gives out almost the same result except that Reno algorithm seems to perform quite better under 1% and 3% packet loss. Since, Vegas algorithm focuses on packet delay than packet loss, gives quite low performance compared to others while BIC algorithm which is the more aggressive version of CUBIC gives the highest throughput at any rate of packet loss. The word aggressive here means BIC reaches peak throughput faster than CUBIC.

Hence, increases in percentage of packet drop in a network decreases the average throughput of the network; change of sending time does not have significant affect on the average throughput of the network and different TCP congestion avoidance algorithm results in different average throughput of the network.

# Experiment 2

## Hypothesis

1. Increasing in number of TCP sessions decrease in fairness among the number of sessions.

## Experiment Setup and Background Theory

Nowadays, TCP composes of around 90% of all traffic in the internet. Many applications use multiple TCP connections at once, for instance, web browsers. These applications, by using many TCP connections get more bandwidth than a application using single connection. Moreover, each TCP connection gets unequal distribution of bandwidth which concerns about the concept of TCP fairness. TCP is fair is all connections using the same link achieve same average throughput. In our report, we will use the definition of ***unfairness*** from *H. Zhou, J. Leis, D. Hoang, P. Nhan, "Throughput and Fairness of Multiple TCP Connections in Wireless Networks"*. The definition is the difference between the maximum and the minimum throughputs divided by the minimum throughputs. The formula is as follow:

$$unfairness = \frac{throughput_{max} - throughput_{min}}{throughput_{min}}$$

The experiment was designed to measure the throughput of several individual TCP sessions running concurrently on the same link and evaluate the fairness among the sessions. These sessions are intended to be started at the same time.

This experiment was performed by using the following parameters:

| | |
|---|---|
| **Sending time** | 300 seconds |

| | |
|---|---|
| **Number of TCP Sessions** | [1, 2, 3, ..., 50] sessions |
| **Congestion Control Algorithm** | Cubic |
| **Other** | Default |

The reason we chose the sending time to be 300 seconds is because our script runs several instances of client connecting to a single server which were intended to be started at exactly the same time but during the run time the client instance starts at different times. The error rate of the time at which the clients start is ±4 seconds. To minimize and even out this glitch we increase the sending time to 300 seconds so that the glitch will affect less on the average throughput of the test.

The reason of limiting the maximum TCP sessions running concurrently to be 50 is because we already tried more number of sessions and the result is the unfairness although increase but the result has too much fluctuation. To respond to this limitation, we introduced a trend line in the plot, so that we can at least show the prediction of the unfairness value with more number of TCP sessions.

The shell script used in this experiment on the client is as follow:

```
for k in {1..30}
do
      gnome-terminal --command "iperf -c 192.168.1.162 -t 300"
done
```

The above script creates 30 instances of iperf client running in parallel for 300 seconds each.

The iperf command used in this experiment on the server is as follow:

```
$ iperf –s
```

## Test Result

From the experiment we performed we came out with the following plots:

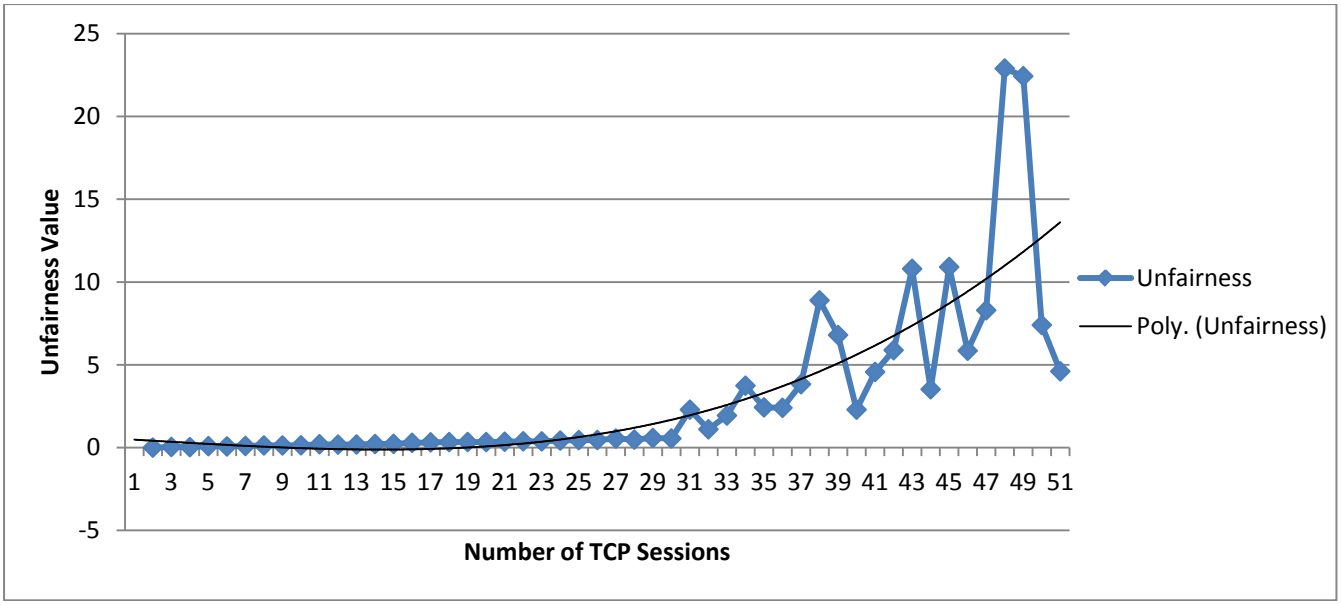


**Figure 5 The unfairness value among the TCP sessions running concurrently with increasing in number of TCP sessions**

## Observation and Conclusion

In this experiment, we have tested running multiple TCP sessions concurrently, measured the throughput and increased number of TCP sessions one by one. According to Figure 5, the result from experiment has shown that with the amount of TCP sessions less than 30, fairness has been achieved. But after the number of TCP sessions exceeds 30, unfairness started to fluctuate and is on an increasing trend. The reason behind the rise of the unfairness seems to come from the number of connection that the client has to maintain exceeds the capability of the client. Third order polynomial trend line is added to display the trend.

According to the result, the unfairness of TCP sessions seems to be increasing thus, it can be concluded that increasing in number of TCP sessions decreases fairness among sessions.

# Experiment 3

## Hypothesis

1. When there is one TCP session running on a link, increasing number UDP sessions running concurrently decrease the throughput of the main TCP session.

## Experiment Setup and Background Theory

No matter we are using TCP or UDP, it is obvious that increasing number sessions running concurrently will decrease the average throughput. Consider a scenario when there is a file download using TCP starting at the time 0 and at a certain time, an application started streaming a video using UDP. This experiment intend to perform such experiment in order to find the result which indicates a certain trend in the change in average throughput of the TCP session started a while ago.

This experiment was performed by using the following parameters:

| | |
|---|---|
| **File size for each TCP session** | $2 \times 10^8$ Bytes |
| **File size for each UDP session** | $2 \times 10^7$ Bytes |
| **Sending Rate of UDP sessions** | 100 Mbps |
| **Number of UDP Sessions** | [1, 2, 3, ..., 30, 50, 100, 200] sessions |
| **Other** | Default |

The reason of choosing $2 \times 10^8$ Bytes as the file size to be transferred by a TCP session is because we wanted the session to last long so that it could be enough to run concurrently alongside many UDP sessions and yet finished at the very last. On the other hand, we chose $2 \times 10^7$ Bytes the file size to be sent by a UDP session because we want the UPD sessions to finish before the main TCP sessions. Anyhow, later we found out during the course of experiment that even if we run 200 UDP sessions alongside the main TCP session, all the UDP sessions are completed first and that is also the reason of why after running tests up to 30 UDP sessions, we decided to skip to 50, 100 and 200 UDP sessions. Another reason of why we skip from 30 all the way to the values aforementioned is that as we increased the number of UDP sessions, the throughput of the main TCP session decrease at a very slow rate. So, by skipping the number of UDP sessions we can get a quicker feel of what the trend should be. Though the average throughput of the main TCP session might further decrease after increasing the number of UDP sessions but we could not go on further do to the limitation of hardware (the client computer hanged when trying 300 UDP sessions).

The shell script used in this experiment on the client is as follow:

```
for k in {1..3}
do
	for i in {1..1}
	do
		gnome-terminal --command "iperf -c 192.168.1.162 -n 200000000"
	done

	sleep 3s
```

```
	for j in {1..25}
	do
		gnome-terminal --command "iperf -c 192.168.1.162 -u -b 100M -n
20000000"
	done
done
```

The above script creates one TCP session and after 3 seconds it creates 25 UDP sessions running concurrently. Each test was run 3 times so the reading was taken as the average values of the tests.

Since, we have to run two instances of server on the server computer, the iperf commands used for doing so are as follow:

```
$ iperf –s
$ iperf –s –u
```

## Test Result

From the experiment we performed we came out with the following plots:

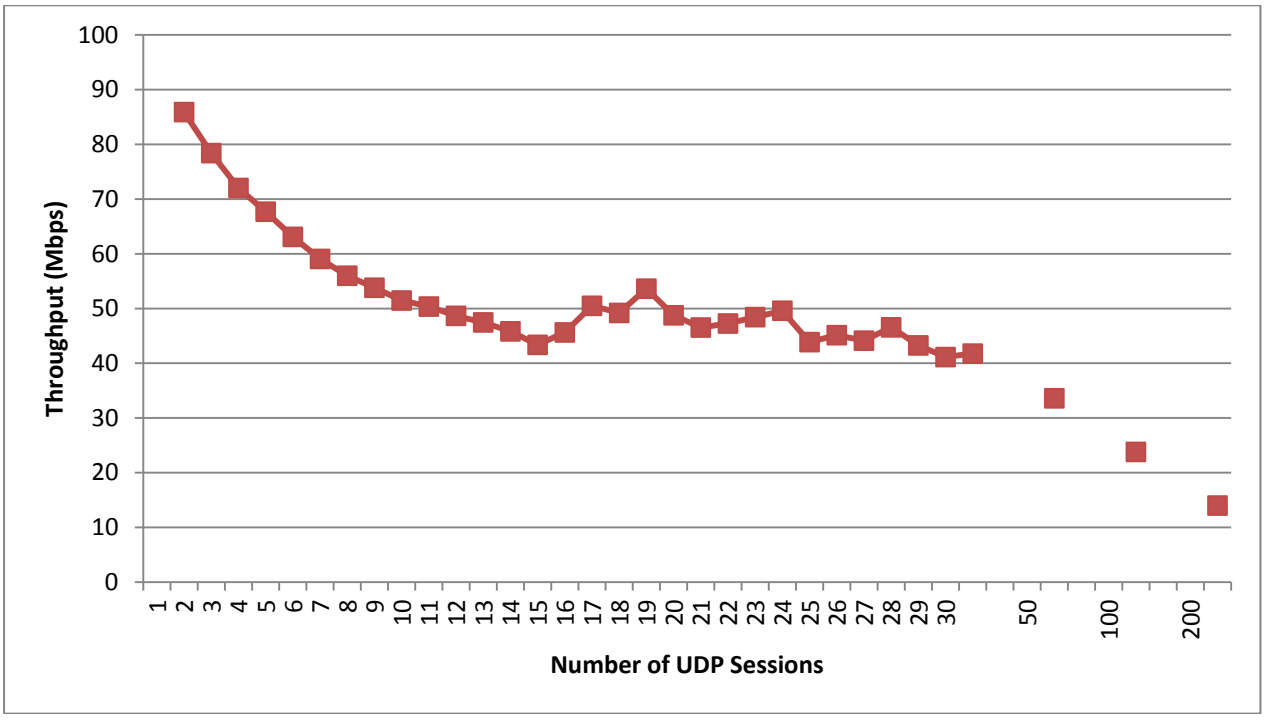


**Figure 6 The average throughput of a main TCP session when the number of UDP sessions increases**

### Observation and Conclusion

In this experiment, we have tested running a TCP session along with various amounts of UDP sessions and measure the throughput of TCP. According to Figure 6, the result form experiment has shown that at first throughput of TCP session seems to decrease very fast with only one or two UDP sessions running. After passing couple of test while the amount of UDP sessions is still less than 15, throughput of TCP session seems to decrease slightly slower. But after the number of TCP sessions exceeds 15, throughput of TCP session started to fluctuate up and down. After running a test with 30 UDP sessions, the result starts to become unreadable so we decided to skip and run test at 50, 100, and 200 UDP sessions instead. As expected, even though the throughput fluctuates, trend of the throughput is still decreasing. The reason that the throughput of TCP sessions decreases when the number of UDP sessions increases is when there is more session sharing the same link, the throughput tends to be divided among all the sessions.

According to the result, throughput of TCP sessions seems to be decreasing quite fast at first then slightly slower over time, thus, it can be concluded that increasing in number of UDP sessions decreases throughput of the main TCP sessions.

## Experiment 4

### Hypotheses

1. The throughput of a UDP connection decreases linearly with the increasing in percentage of packet drops.
2. The throughput of a TCP connection decreases substantially with the increasing in percentage of packet drops.

### Experiment Setup and Background Theory

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are both transport protocol that work on the TCP/IP network model but are not much alike. TCP is connection oriented, sends data in stream of bytes, sends and receives packets in order and has retransmission scheme. On the other hand, UDP is connectionless, sends data individually, can receive packet out of order and does not have any retransmission scheme. The usages of these two transport protocols are very different.

This experiment was performed by using the following parameters:

| | |
|---|---|
| **Packet Drop** | [0, 1, 2, ..., 30] % |
| **Other** | Default |

The reason of stopping to increase the percentage of packet drops at 30% is that we can see the trend already. We have tried percentage of packet loss for more than 30% we still get the same trend i.e. average throughput of TCP approaches 0 Mbps while average throughput of UDP decreases at constant rate.

The iperf command used in this experiment on the client is as follow:

For TCP

```
$ iperf –c 192.168.1.162
```

For UDP

```
$ iperf –c 192.168.1.162 –u -b 100M
```

The iperf command used in this experiment on the server is as follow:

For TCP

```
$ iperf –s
```

For UDP

```
$ iperf –s -u
```

Each test is run 3 times so the reading was taken as the average values of the tests.

## Test Result

From the experiment we performed we came out with the following plots:

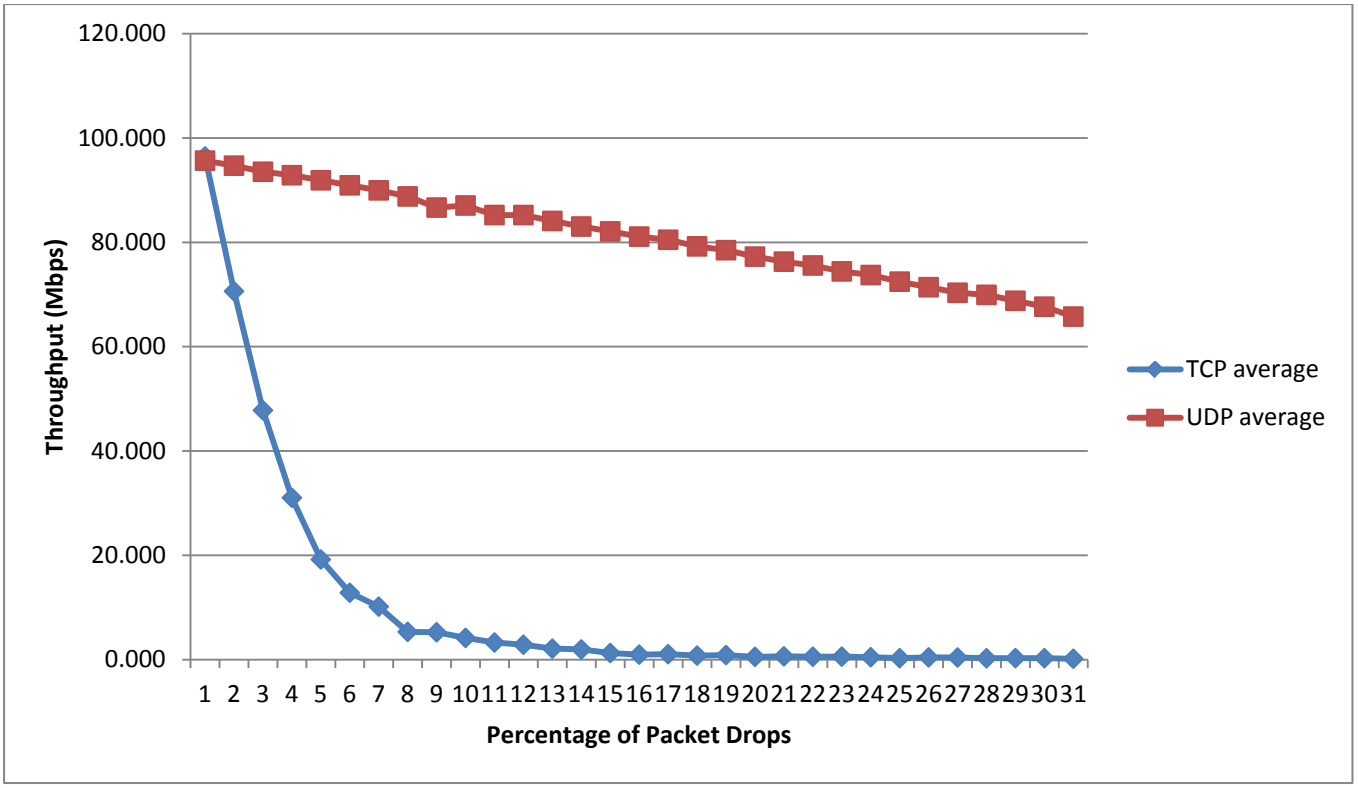


**Figure 7 The average throughput of TCP and UDP as the percentage of packet drops increase**

## Observation and Conclusion

In this experiment, we have tested running a TCP and UDP session over various packet loss percentages, measure and compare the throughput. The result has shown that, over increasing packet loss percentage, both UDP and TCP throughput is decreasing but with very difference pace. According to Figure 7, throughput of UDP slightly decreases at constant rate while throughput of TCP decreases very fast at first then slightly slower over time. Note that at 15 percent packet loss, throughput of TCP decreased so fast that it reached 1 Mbps while UDP remains at 82 Mbps.

Since TCP has retransmission scheme, increasing in packet loss percentage will cause increasing in retransmission which results in tremendous decreases of throughput while UDP does not have retransmission scheme, thus, UDP does not suffer from this drawback. Hence, the throughput of a UDP connection decreases linearly with the increasing in percentage of packet drops and throughput of a TCP connection decreases substantially with the increasing in percentage of packet drops for a period of time then the rate of decreasing decreases approaching zero.

# References


[1] Injong Rhee, and Lisong Xu, *"CUBIC: A New TCP-Friendly High-Speed TCP Variant"*.

[2] Hong Zhou, John Leis, Doan Hoang, and Phuong Nhan, *"Throughput and Fairness of Multiple TCP Connections in Wireless Networks"*.

[3] Sándor Molnár, Balázs Sonkoly and Tuan Anh Trinh, "A Comprehensive TCP Fairness Analysis in High Speed Networks".

[4] Dimitrios Vardalis, *"On the Efficiency and Fairness of TCP over Wired/Wireless Networks"*.