ITS 413 Internet Technologies and Applications

Assignment: Phase 3 Report

By: Group04

Mr.Suppachai Suwanwatcharachat (5222781601)

Mr.Sonnatas Chaisorn (5222782096)

Mr.Rungsemund Chunvichit (5222791683)

Date: 28 March 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

To determine how different parameters and scenarios impact on the performance of transport protocols (TCP and UDP) in a wireless LAN and Ethernet link.

Network Diagram

Most of the experiment will use this diagram to test network, if have a special equipment you can see in detail that will show in every experiment

Wired throughput test

Client

WRT-54gl

Server

Equipment Specifications

Computer A

Brand: Compaq
OS: Window 7 - 32bit
Processor: Pentium Dual-Core CPU 2.20GHz
Ram: 1.00 GB

Wired LAN interface: Realtek RTL8102E/TRL8103E Family PCI-E Fast Ethernet NIC

Wireless LAN interface: Broadcom 802.11g Network Adapter#2

Computer B

Brand: Asus
OS: Window 7 - 32bit
Processor: Intel Core 2 Duo CPU 2.4GHz
Ram: 2.00 GB

Wired LAN interface:Atheros AR8151 PCI-E Gigabit Ethernet Controller (NDIS 6.20)

Wireless LAN interface:Atheros AR9002WB-1NG Wireless Network Adapter

Router: LINKSYS

Model: Linksys WRT54GL Ram: 16 MB OS: OpenWRT

Standards IEEE 802.3, IEEE 802.3u, IEEE 802.11g, IEEE 802.11b

Ports Internet: One 10/100 RJ-45 Port

Ethernet: Four 10/100 RJ-45 Switched Ports

One Power Port

Buttons One Reset Button

LEDs Power, DMZ, WLAN, Ethernet (1, 2, 3, 4), Internet

Cabling Type CAT 5

RF Power (EIRP) in dBm 18

UPnP able/cert Able

Security Features Stateful Packet Inspection (SPI) Firewall, Internet Policy

Wireless Security Wi-Fi Protected Access™ 2 (WPA2), WEP, Wireless MAC Filtering

Parameters

Parameter	Value
Channel	6
Protocol Used	TCP
Time to test	10 seconds
WEP	off

Shell Script

We use **shell script** in this experiment to run iperf command. Shell script will create a command that we can run code and loop it until you get result.

```
#!/bin/bash
COUNT=0
MAX=3
while [$COUNT –lt $MAX]; do
        Iperf -c 1.1.1.1 –t 10
        Let COUNT=COUNT+1
done
```

Experiments and Results

Experiment 1 – Impact of window sizes on TCP performance

In this experiment we will test a performance of TCP, how impact when we change some parameter between we send data such as: Maximum Segment Size (MSS), Window, and Length of data

We measured the average throughput of the Wired Ethernet link by decreasing the sending rate every time. The average was taking from 3 different tests. This is our result.

Network Diagram

Most of the experiment will use this diagram to test network, if have a special equipment you can see in detail that will show in every experiment

Wired throughput test

Client WRT-54gl Server

Command

Experiment 1

Command	Description
iperf –M #	**set the MSS**
iperf –w #	**set the TCP buffer size**
iperf –l #	**set the data length**

1.1 Testing on maximum Segment Size

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 1500 to 100 Bytes (reduce 100 Bytes per time)

First, we will test the maximum segment size, the command that we used in this step is:

```
iperf -M
```

And this is our result

Table: Maximum Segment Size result

MSS (Bytes)	AVG (Mbit/sec)
1500	94
1400	93
1300	93
1200	92
1100	91
1000	91
900	90
800	81
700	70
600	61
500	51
400	41
300	31
200	21
100	10

1.2 Testing on Window Size

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 16 to 2 KB (decrease 1 Kb per time)

Next, we will test the window size that will set the buffer size, the command that we used in this step is:

iperf -w

And this is our result

Table: Window size result

Window size (KB)	AVG (Mbit/sec)
16	94
15	75
14	74
13	67
12	67
11	55
10	56
9	60
8	45
7	51
6	51
5	37
4	37
3	5
2	3

1.3 Testing on Datalength

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 300 to 25 Bytes (reduce 25 Bytes per time)

Next, we will on datalength that will set the length of data write or read, command that we used in this step is:

```
iperf -l
```

And this is our result

Table : Datalength result

Datalength (Bytes)	AVG (Mbit/sec)
300	93
275	94
250	94
225	94
200	94
175	94
150	82
125	69
100	55
75	41
50	28
25	14

So, we summary plot into a table as following

Table 1: Maximum Segment Size result

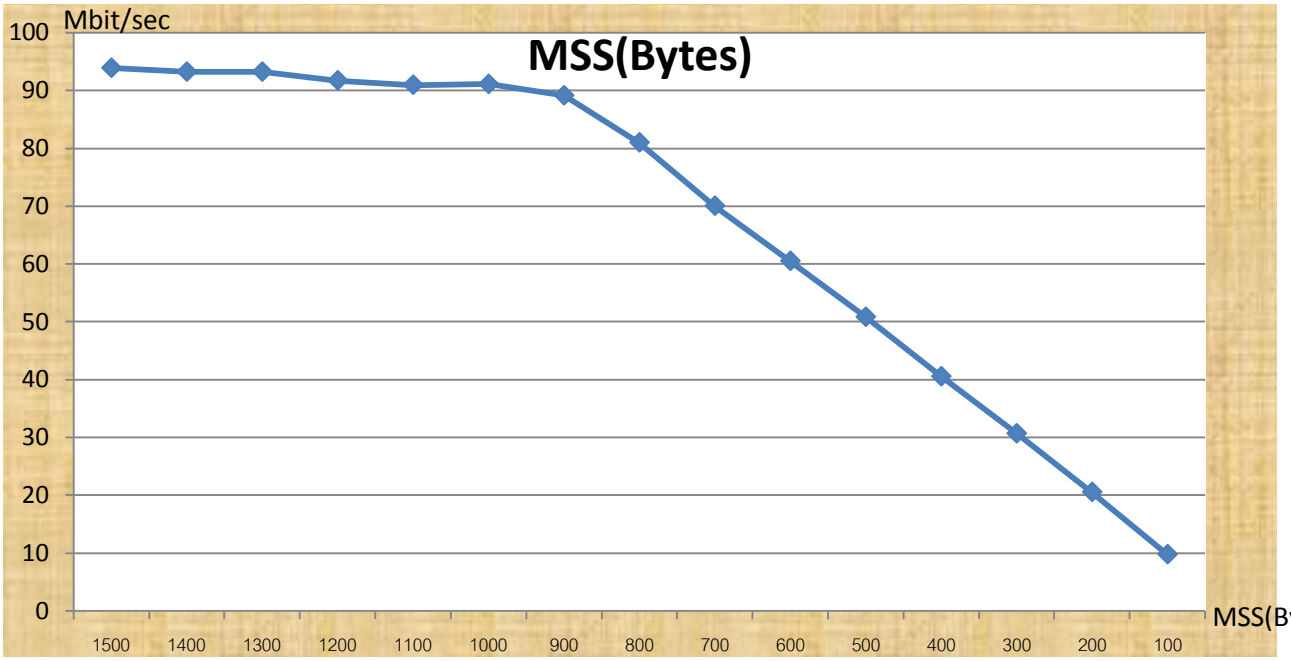


Table 2 : Window Size result

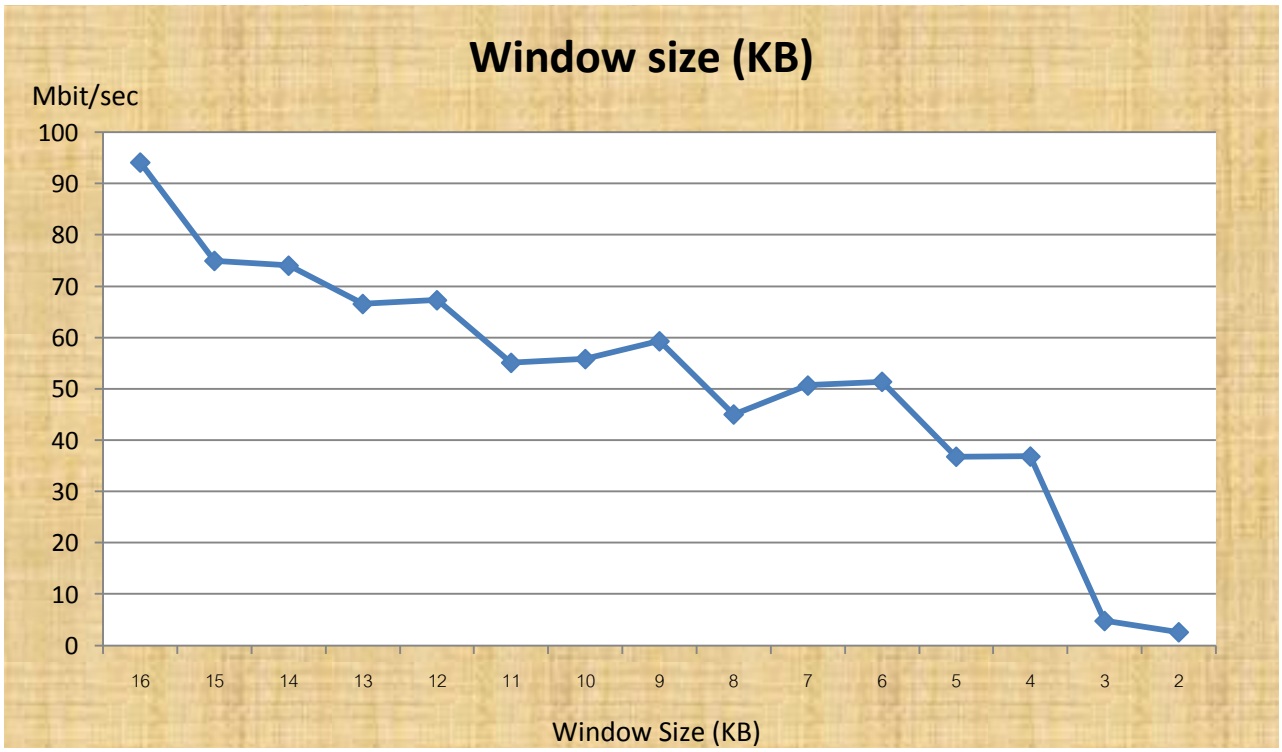
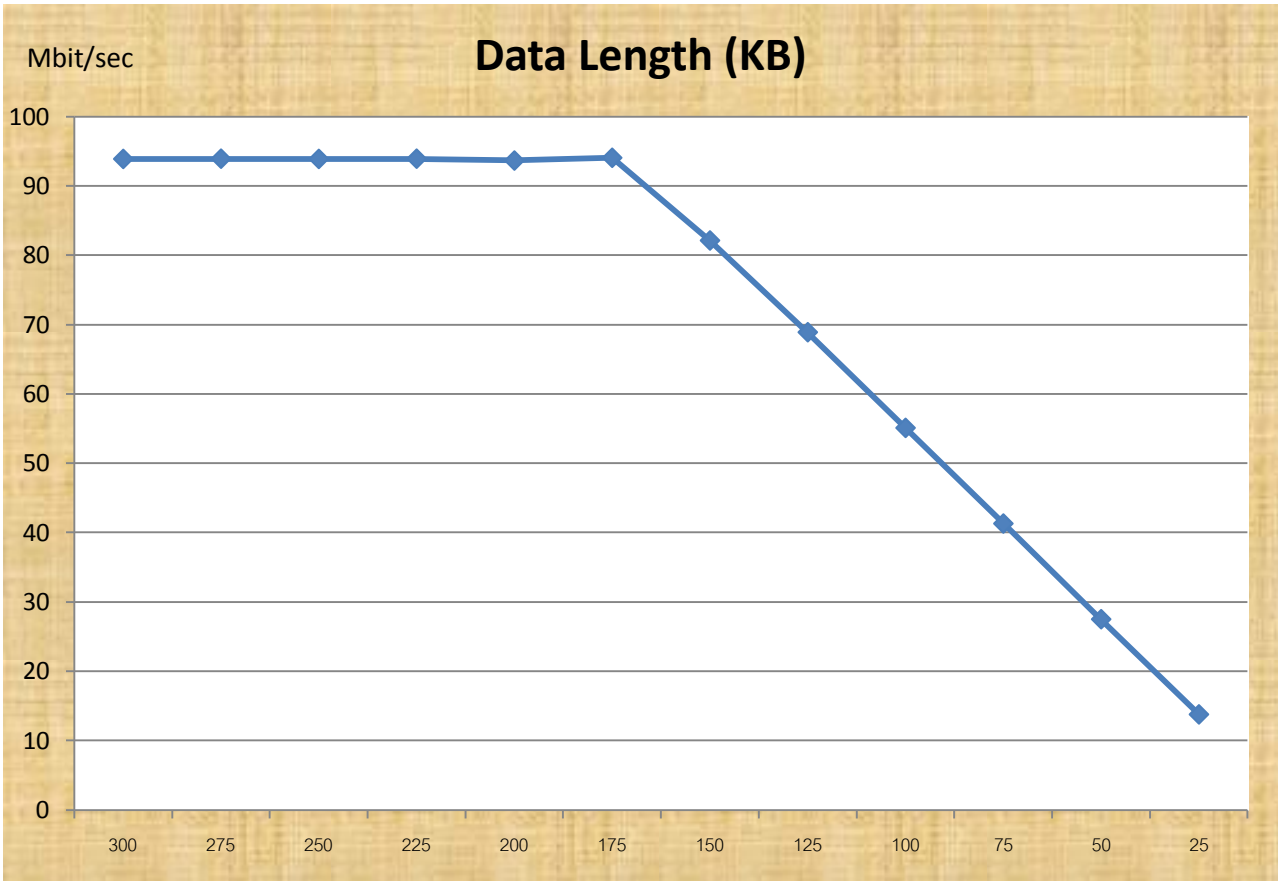


Table 3 : Data length result



Conclusion

All parameters that have been experimented shows an impact with TCP throughput.

Parameter MSS: **TCP throughput linearly increases as Maximum Segment Size increases** until MSS reaches 1000 Bytes, TCP throughput is about 90 Mbps. MSS sizes over 1000 has insignificant effect on TCP throughput.

Parameter Window Size: **TCP throughput linearly increases as Window Size increases** until Window Size reaches 16KB, TCP throughput is maximized around 90 Mbps. Window size over 16 KB has insignificant effect on TCP throughput.

Parameter Data Length: **TCP throughput linearly increases as Data Length increases** until Data Length reaches 175 KB, TCP throughput is maximized around 93 Mbps. Data length over 175 KB has insignificant effect on TCP throughput.
To summarize, increasing MSS, Window Size, or Data Length results in linear increasing throughput.

Experiment 2 – Impact of packet drops on TCP performance(Drop packet with tc and iptables command)

In this experiment we will test a performance of TCP, how impact when we select to drop some packet when we transmit data to destination. In addition, we create a rule by use iptables command to specify how packets coming into your computer and going out of your computer are treated (INPUT for incoming packet, OUTPUT for going out packet and FORWARD for a forward by the router). In particular we will use a probability function to specify how many percentage of packets will drop when transmit a data.

Another way, in this experiment we will use tc command that use to control the transmit queue of your kernel for transmission on the network interface (in normal, the packet are transmit in a first-in-first-out (FIFO)). tc will allow you to change a queuing mechanism, giving priority to specific type of packets, as well as emulate links by delaying and dropping packets. Here we will see the result in our experiment below.

We measured the average throughput of the wired Ethernet link by increasing the sending rate by the step of 1. The average was taking from 3 different tests. This is our result.

Network Diagram

Wired throughput test

Client

WRT-54gl

Server

Command

Experiment 2

Command	Description
sudo iptables –L	List the rule of iptables that will contain Rule of INPUT ,OUTPUT ,FORWARD that will specify how packets coming into your computer or going out of your computer (or forward by router).
sudo iptables -A INPUT -m statistic --mode random --probability 0.01 -j DROP	add a rule to the INPUT , by having a chance to drop some incoming packets specify by value that user used on computer B
sudo iptables -D INPUT -m statistic --mode random --probability 0.01 -j DROP sudo iptable –D INPUT #	delete a rule to the INPUT , by use –D mean delete following with command that you use it above or you can write in a short term by use order number(line of that command) instead of whole command
iperf –s	Create a server
Iperf –c	Create a client
$ sudo tc qdisc add dev eth0 root netem loss #%	Tells the Linux kernel to drop on average that user specify value (“#”) of the packets in the transmit queue.
$ sudo tc qdisc show dev eth0	show the current queue discipline
$ sudo tc qdisc del dev eth0 root netem loss #%	delete the queue discipline

2. 1 Dropping packet with iptables

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 1 to 15 (increase 1 per time)

First, we must set the rules to INPUT chain to have a chance to drop incoming packet on computer B. The command that we used in this step is:

```
-A INPUT -m statistic --mode random --probability 0.01 -j DROP
```

Then, we can check the rule is added or not by use this command and output will be:

```
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere            statistic mode random probability 0.010000

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

We demonstrate the packet dropping by run another iperf TCP with this command:

```
$ iperf -c 192.168.1.236 -t 10
```

And this is our result

Table: dropping packet with iptables result

Dropping packets at reciever (using iptables)	Throughput (Mbits/s)
0	94
1	85
2	54
3	41
4	24
5	16
6	11
7	6
8	5
9	4
10	3
11	3
12	2
13	2
14	1
15	1

2. 2 Dropping packet with tc

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 0 to 10 % (increase 1 % per time)

First, we must create a chance to randomly drop packets. The command that we used in this step is:

```
$ sudo tc qdisc add dev eth0 root netem loss 1%
```

Then, if you want to know what value of loss that you used, you can give a command to show the current queue disciplines by use this command and output will be:

```
$ sudo tc qdisc show dev eth0
qdisc netem 8001: root refcnt 2 limit 1000 loss 1%
```

So, let run command to test(for client and server):

```
$ iperf -c 192.168.1.236 -t 10

$ iperf -s
```

And this is our result

Table: dropping packet with tc result

Dropping packets at sender (using tc)	**Throughput (Mbits/s)**
1	94
2	46
3	25
4	17
5	12
6	8
7	6
8	4
9	3
10	2

So, we plot into a table as following

Table 1: Droping packet with iptables

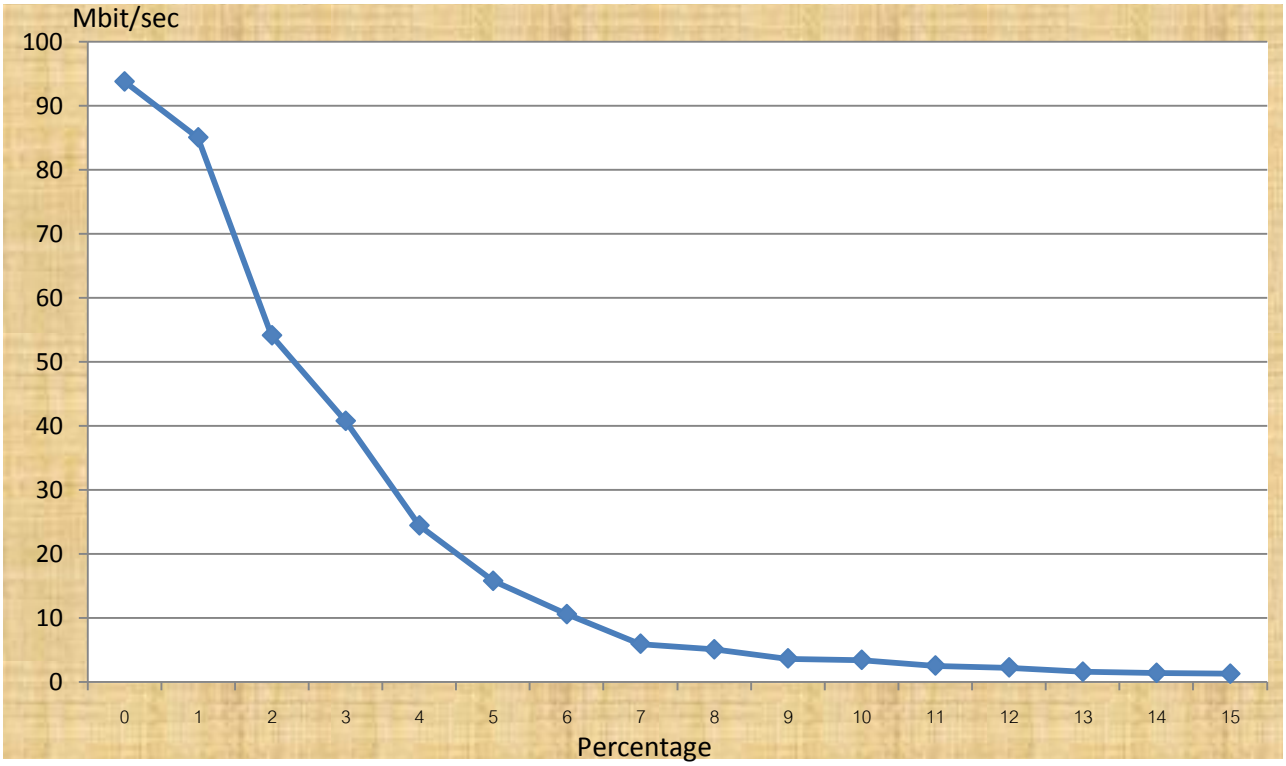


Table 2: Droping packet with tc

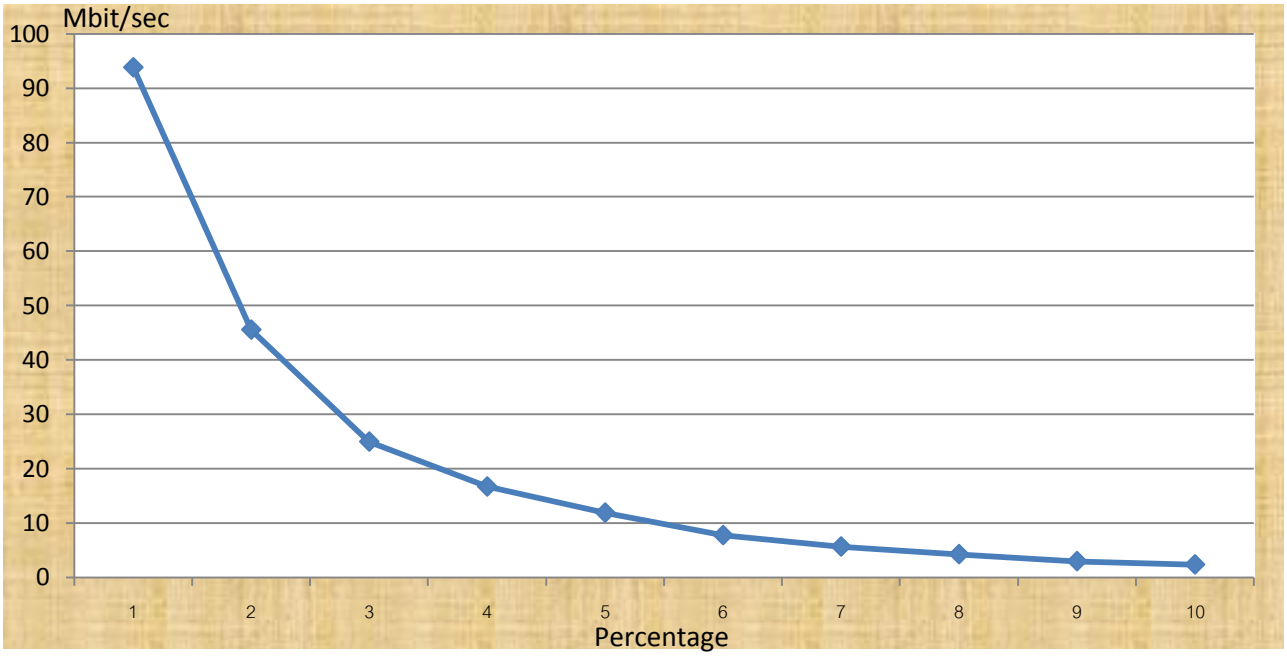
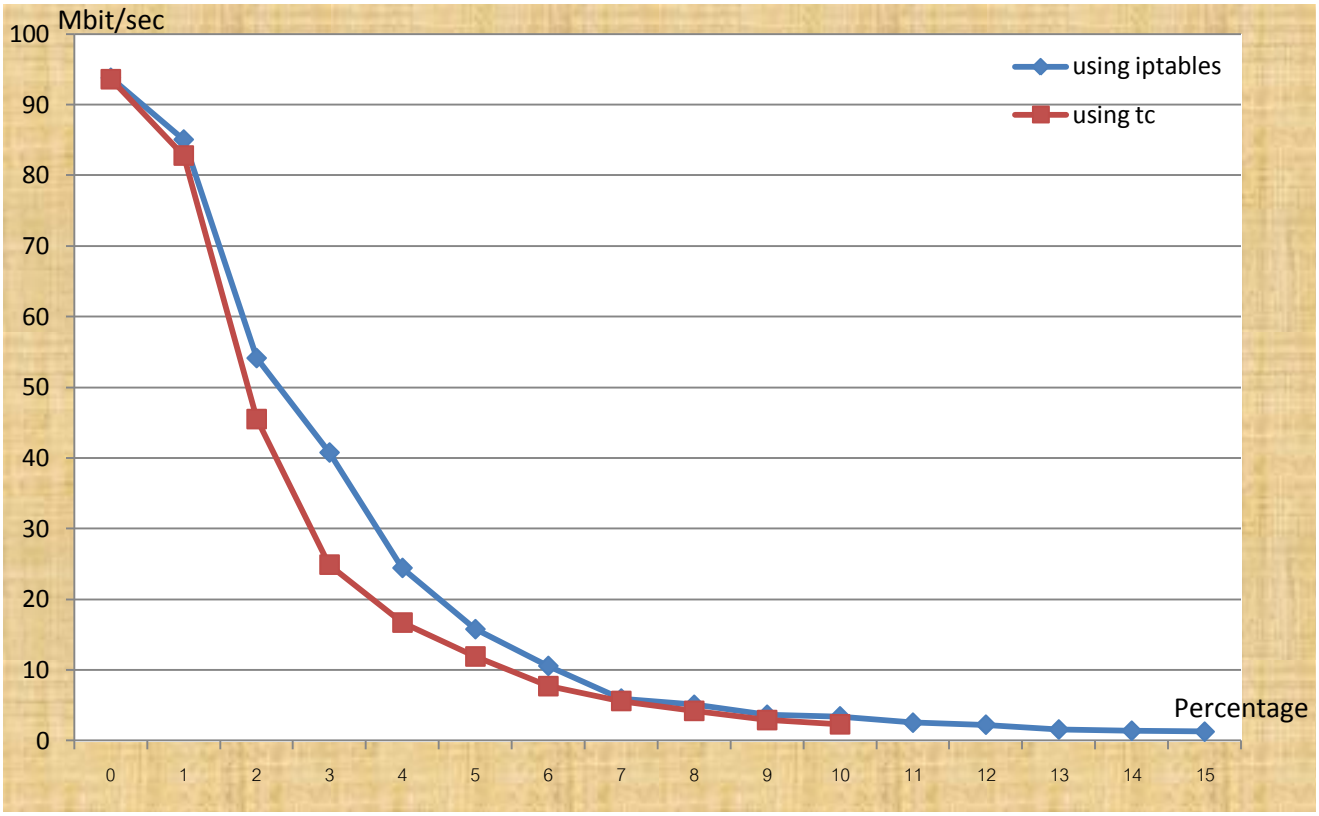


Table 3: Compare between tc and iptables



Conclusion

From the experiment, we can see that packet drop has a drastic impact on TCP performance.

Using iptables**: As packet drop percentage increase, TCP throughput decrease exponentially**, until packet drop percentage reaches 7%, the throughput is nearly zero.

Using tc: **As packet drop percentage increase, TCP throughput decrease exponentially**, until packet drop percentage reaches 7%, the throughput is nearly zero.

The significance drop in performance from packet loss is explained by how TCP deals with lost packets. In this case, the packet loss is due to timeout which simulate a heavy network congestion. To decrease the congestion, TCP halves the congestion windows on each timeout, thus decreases the TCP throughput as we have seen in experiment 1.

Experiment 3 – Impact of different congestion control algorithms on TCP (Using Reno and Cubic algorithm)

In this experiment we will test a performance of TCP, how impact when we select to use some various congestion control algorithm. Before we go that, we should know some topic and parameter that related with this experiment.

Bandwidth Delay Product (BDP) the amount of data that can be in transit in the network. It is the product of the available bandwidth and the latency, or RTT. BDP is a very important concept in a Window based protocol such as TCP. It plays an especially important role in high-speed / high-latency networks, such as most broadband internet connections. It is one of the most important factors of tweaking TCP in order to tune systems to the type of network used.

Way to calculate BDP

BDP (bits) = total_available_bandwidth (bits/sec) x round_trip_time (sec)

or BDP (bytes) = total_available_bandwidth (KBytes/sec) x round_trip_time (ms)

Round-trip time (RTT), also called round-trip delay, is the time required for a signal pulse or packet to travel from a specific source to a specific destination and back again. In this context, the source is the computer initiating the signal and the destination is a remote computer or system that receives the signal and retransmits it.

Selecting TCP Congestion Control Algorithm

In this experiment we have 2 algorithms that we used:

1. Reno algorithms
2. Cubic algorithms

We measured the average throughput of the wired Ethernet link by increasing the sending rate by the step of 4 KB. The average was taking from 3 different tests. This is our process below.

Network Diagram

In this experiment we connect 2 computers directly with fast ethernet 100Mb/s switch ,Then network diagram should be :

Client Server

Command

Experiment 3

Command	Description
$ sudo sysctl net.ipv4.tcp_congestion_control=reno	Set the TCP congestion control algorithm to Reno(or cubic)
$ sudo sysctl net.ipv4.tcp_moderate_rcvbuf=0(or 1)	Turn off (or Turn on) auto-tuning of the TCP receive buffer size. 0= turn off , 1=turn on
$ sudo tc qdisc add dev eth0 root netem delay # ms	Set the delay on the path. On both sender and receiver
$ iperf -s -w “values”	Create a server, measure the TCP throughput for varying BDP and receive buffer sizes on the receiver.
$ iperf -c “IP address”	Create a client (sender)
$ cat /proc/sys/net/ipv4/tcp_congestion_control $ sysctl net.ipv4.tcp_congestion_control	Check what the algorithm use in operating system.
$ sysctl net.ipv4.tcp_available	To see the current set of available algorithms
$ sysctl net.ipv4.tcp_rmem	Set the buffer space available at the receiver for TCP using a kernel configuration parameter.
$ sysctl net.ipv4.tcp_moderate_rcvbuf	Check that it perform auto-tuning BDP or not

Parameters

Parameter	Value
Channel	6
Protocol Used	TCP
Time to test	10 seconds
Bandwidth	100 Mb/s
RTT	10 seconds
BDP	125000 Bytes

We set parameter to be:

1. Bandwidth to 100 Mb/s

2. RTT to 10 seconds (Computer A = 5 seconds, Computer B= 5 seconds)

3. BDP (advertising window size to maximize throughput) to 125000 Bytes

Theorem

When we sets the receive buffer to some value using the -w option, kernel will doubles the buffer size and allocates 75% of the buffer space for receiving segments. Then, when we set buffer to 20KB kernel will double the buffer size to 40KB and use 30KB to receiving segments

In the receive buffer we can calculate to get some optimal throughput. By use BDP * 4/3 and set to kernel, so we try to create some value to test and calculate. Here is a short scenario.

Scenario	Bandwidth [Mb/s]	RTT [ms]	BDP [Bytes]	Receive Buffer [Bytes]
Fast Ethernet, single link	100	10	125000	166667

So we will know that we should set buffer size to be 83334 in the command we will use this command to set it :

```
iperf –s –w 83334
```

So, we can summary a technique to calculate by:

Maximum Advertised Window = receiver buffer size *3/4

Expected throughput = awnt(maximum Advertised Window)/RTT

Here is a result

3. 1 Using Cubic algorithm

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 0 to 7 KB (increase 0.5 per time)

First, we must set the TCP congestion control algorithm to Cubic, command that we used in this step is:

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

Turn off auto-tuning of the TCP receives buffer size:

```
$ sudo sysctl net.ipv4.tcp_moderate_rcvbuf=0
```

set the delay on the path. On both sender and receiver with this command:

```
$ sudo tc qdisc add dev eth0 root netem delay 10ms
```

Measure the TCP throughput for varying BDP and receive buffer sizes, with this command:

```
$ iperf -c 1.1.1.1
```

And this is our result

Table:Cubic algorithm result

Using Cubic algorithm With packet loss (%)	Throughput (Mbits/s)
0	93.97
0.5	92.93
1	86.77
1.5	65.83
2	54.17
2.5	42.67
3	33.47
3.5	29.70
4	27.77
4.5	19.10
5	12.17
5.5	12.50
6	9.68
6.5	8.47
7	6.72

3. 2 Using Reno algorithm

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 0.5 to 7 KB (increase 0.5 per time)

First, we must set the TCP congestion control algorithm to Reno, command that we used in this step is:

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

Turn off auto-tuning of the TCP receives buffer size:

```
$ sudo sysctl net.ipv4.tcp_moderate_rcvbuf=0
```

set the delay on the path. On both sender and receiver with this command:

```
$ sudo tc qdisc add dev eth0 root netem delay 10ms
```

Measure the TCP throughput for varying BDP and receive buffer sizes, with this command(server, client):

```
$ iperf -c 1.1.1.1
```

And this is our result

Table:Reno algorithm result

Using Reno algorithm With packet loss (%)	Throughput (Mbits/s)
0	94.60
0.5	93.77
1	88.80
1.5	77.83
2	64.40
2.5	55.30
3	49.17
3.5	33.63
4	25.10
4.5	20.43
5	19.77
5.5	12.30
6	10.17
6.5	8.00
7	7.09

So, we plot into a table as following

Table 1 :Cubic algorithm result

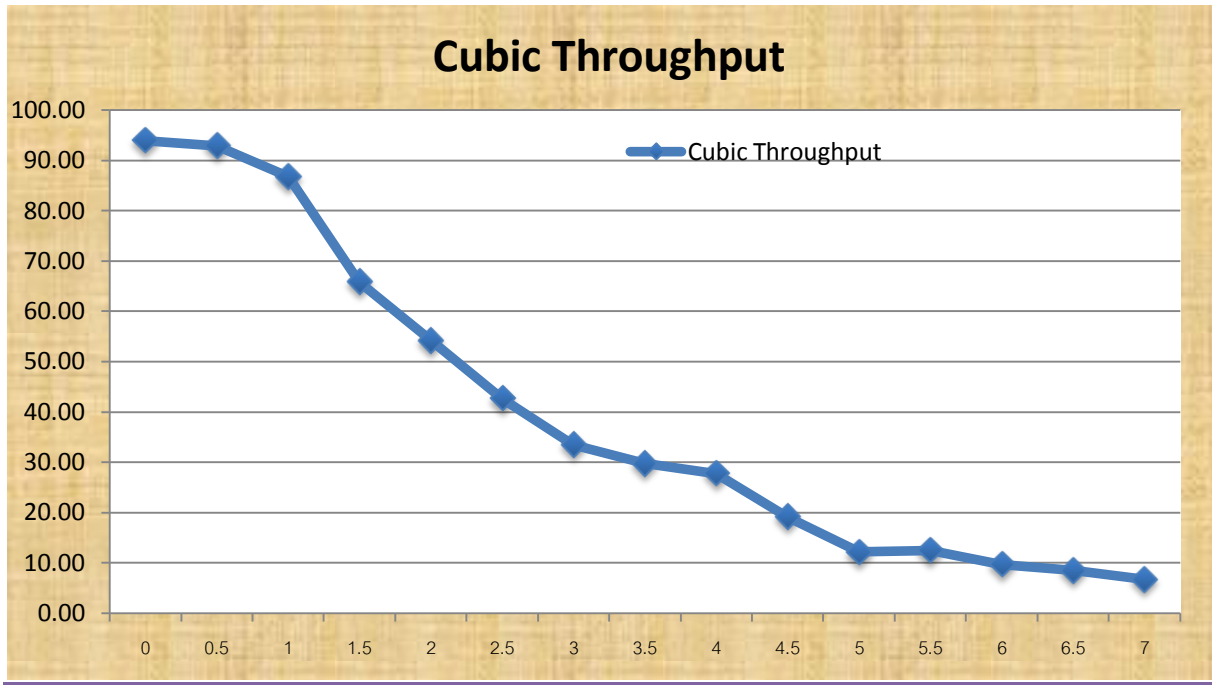


Table 2 :Reno algorithm result

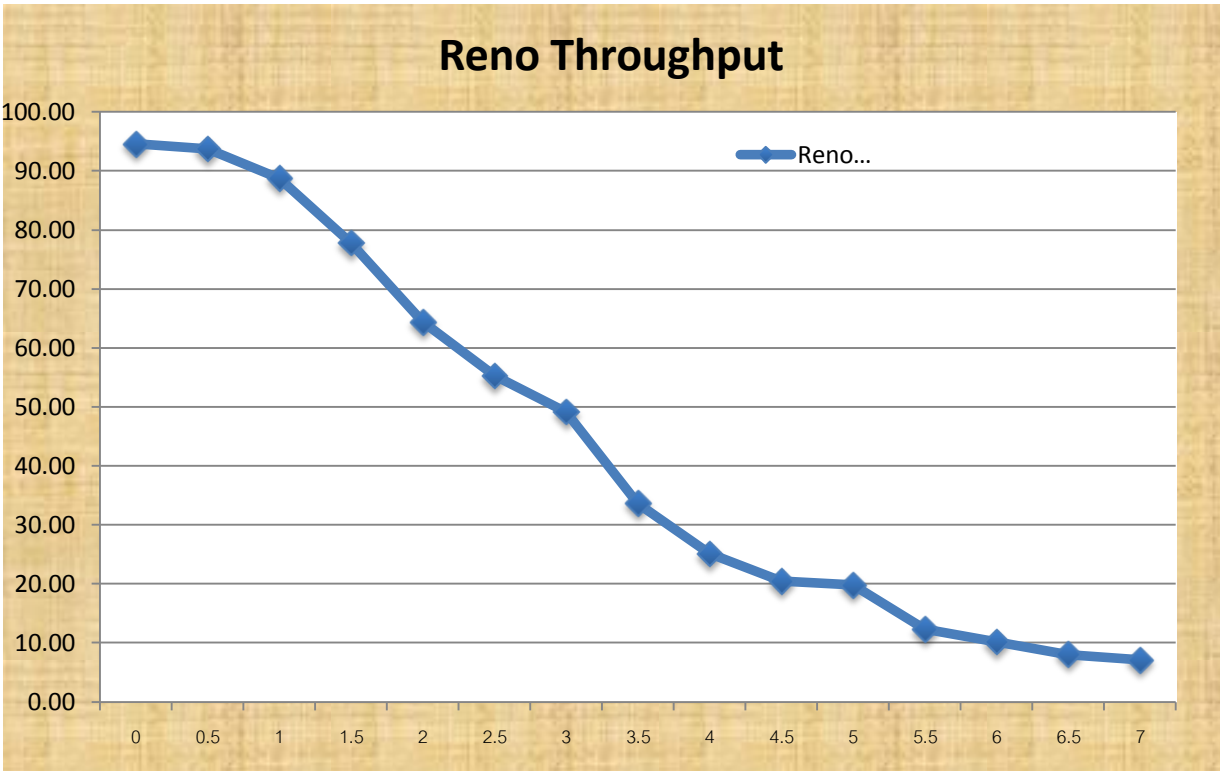
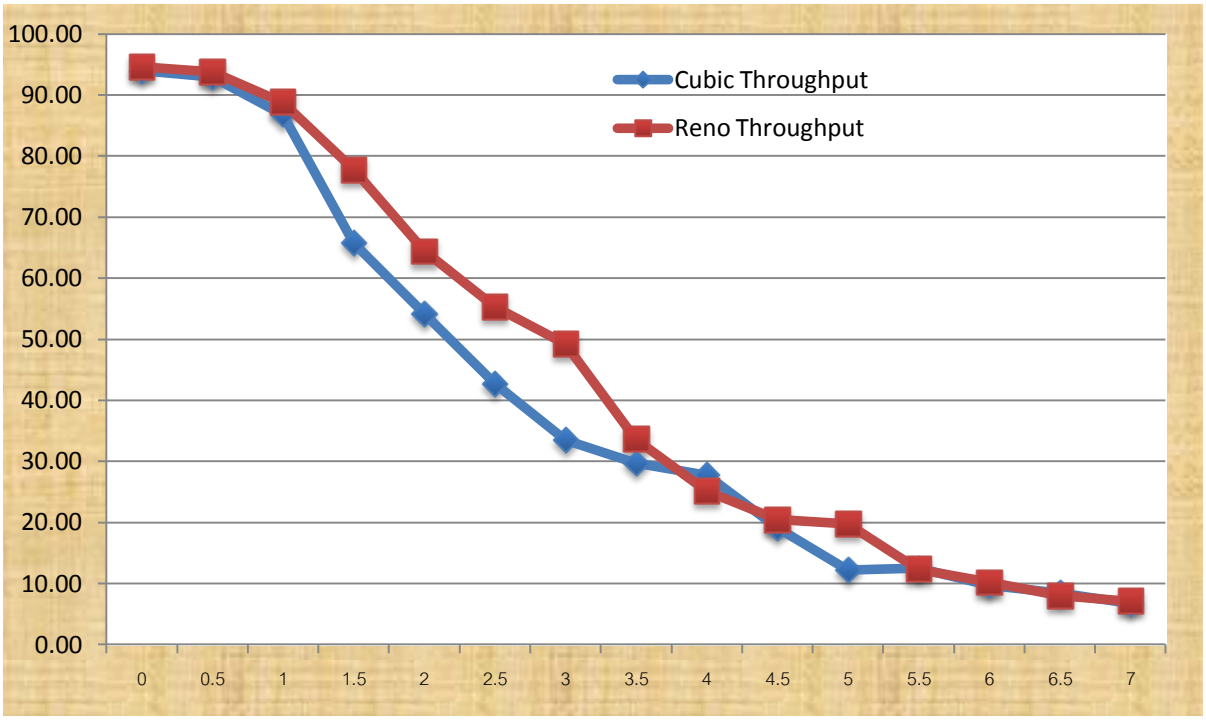


Table 3 :Compare between Cubic and Reno algorithm



Conclusion

Different congestion control algorithm behaves differently to deal with network congestion. In the experiment, we have chosen 2 algorithms; cubic and reno. They both have similar TCP throughput, but cubic has slightly lower throughput at packet loss between 1-4%.

PART 2 : Change window size parameter

1. Using Cubic algorithm

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 4 to 100 KB (increase 4 per time)

First, we must set the TCP congestion control algorithm to Cubic, command that we used in this step is:

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

Turn off auto-tuning of the TCP receives buffer size:

```
$ sudo sysctl net.ipv4.tcp_moderate_rcvbuf=0
```

set the delay on the path. On both sender and receiver with this command:

```
$ sudo tc qdisc add dev eth0 root netem delay 10ms
```

Measure the TCP throughput for varying BDP and receive buffer sizes, with this command(server, client):

```
$ iperf -s -w 50000

$ iperf -c 1.1.1.1
```

And this is our result

Table 1 :Cubic algorithm result

BDP	-w	Buffer Size	Max Adv. Window	Expected Throughput	Measured Throughput 1	Measured Throughput 2	Measured Throughput 3	Average Throughput	Accuracy
[KBytes]	[KBytes]	[KBytes]	[KBytes]	[Mb/s]	[Mb/s]	[Mb/s]	[Mb/s]	[Mb/s]	[%]
125	4	8	6	4.8	2.23	2.23	2.23	2	46.46
125	8	16	12	9.6	6.64	6.66	6.67	7	69.34
125	12	24	18	14.4	11.1	11.1	11.1	11	77.08
125	16	32	24	19.2	15.6	15.6	15.6	16	81.25
125	20	40	30	24	22	22	22	22	91.67
125	24	48	36	28.8	26.5	26.4	26.5	26	91.90
125	28	56	42	33.6	30.9	30.9	30.9	31	91.96
125	32	64	48	38.4	35.4	35.2	35.3	35	91.93
125	36	72	54	43.2	40.2	40.1	40.1	40	92.90
125	40	80	60	48	44.7	44.7	44.7	45	93.13
125	44	88	66	52.8	49.2	49.2	49.2	49	93.18
125	48	96	72	57.6	53.6	53.7	53.6	54	93.11
125	52	104	78	62.4	59.7	59.7	59.7	60	95.67
125	56	112	84	67.2	64.1	64.2	64.1	64	95.44
125	60	120	90	72	68.4	68.4	68.4	68	95.00
125	64	128	96	76.8	73	73	73.1	73	95.10
125	68	136	102	81.6	77.9	77.9	77.8	78	95.42
125	72	144	108	86.4	82.8	82.8	82.8	83	95.83
125	76	152	114	91.2	91.1	91.2	91.1	91	99.93
125	80	160	120	96	93.4	93.1	93.3	93	97.15
125	84	168	126	100	93.3	93.3	93.4	93	93.33
125	88	176	132	100	93.5	93.4	93.5	93	93.47
125	92	184	138	100	93.5	93.3	93.3	93	93.37
125	96	192	144	100	93.5	93.5	93.4	93	93.47
125	100	200	150	100	93.5	93.5	93.5	94	93.50

2. 2 Using Reno algorithm

Protocol used: TCP
Client: Computer A
Server: Computer B
Sending Rate: From 4 to 100 KB (increase 4 per time)

First, we must set the TCP congestion control algorithm to Reno, command that we used in this step is:

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

Turn off auto-tuning of the TCP receives buffer size:

```
$ sudo sysctl net.ipv4.tcp_moderate_rcvbuf=0
```

set the delay on the path. On both sender and receiver with this command:

```
$ sudo tc qdisc add dev eth0 root netem delay 10ms
```

Measure the TCP throughput for varying BDP and receive buffer sizes, with this command(server, client):

```
$ iperf -s -w 50000

$ iperf -c 1.1.1.1
```

And this is our result

Table 2 :Reno algorithm result

Bandwidth	RTT	BDP	-w	Buffer Size	Max Adv. Window	Expected Throughput	Average Throughput	Accuracy
[Mb/s]	[ms]	[KBytes]	[KBytes]	[KBytes]	[KBytes]	[Mb/s]	[Mb/s]	[%]
100	10	125	4	8	6	4.8	2.21	46.04
100	10	125	8	16	12	9.6	6.64	69.13
100	10	125	12	24	18	14.4	11.10	77.08
100	10	125	16	32	24	19.2	15.53	80.90
100	10	125	20	40	30	24	22.00	91.67
100	10	125	24	48	36	28.8	26.40	91.67
100	10	125	28	56	42	33.6	30.80	91.67
100	10	125	32	64	48	38.4	35.20	91.67
100	10	125	36	72	54	43.2	40.00	92.59
100	10	125	40	80	60	48	44.50	92.71
100	10	125	44	88	66	52.8	49.27	93.31
100	10	125	48	96	72	57.6	53.80	93.40
100	10	125	52	104	78	62.4	59.47	95.30
100	10	125	56	112	84	67.2	64.93	95.14
100	10	125	60	120	90	72	68.37	94.95
100	10	125	64	128	96	76.8	73.77	94.75
100	10	125	68	136	102	81.6	78.03	95.63
100	10	125	72	144	108	86.4	82.63	95.64
100	10	125	76	152	114	91.2	91.47	100.29
100	10	125	80	160	120	96	93.33	97.22
100	10	125	84	168	126	100	93.33	93.33
100	10	125	88	176	132	100	93.40	93.40
100	10	125	92	184	138	100	93.57	93.57
100	10	125	96	192	144	100	93.60	93.60
100	10	125	100	200	150	100	93.57	93.57

So, we plot into a table as following

Table 1: Cubic algorithm (plot from average throughput)

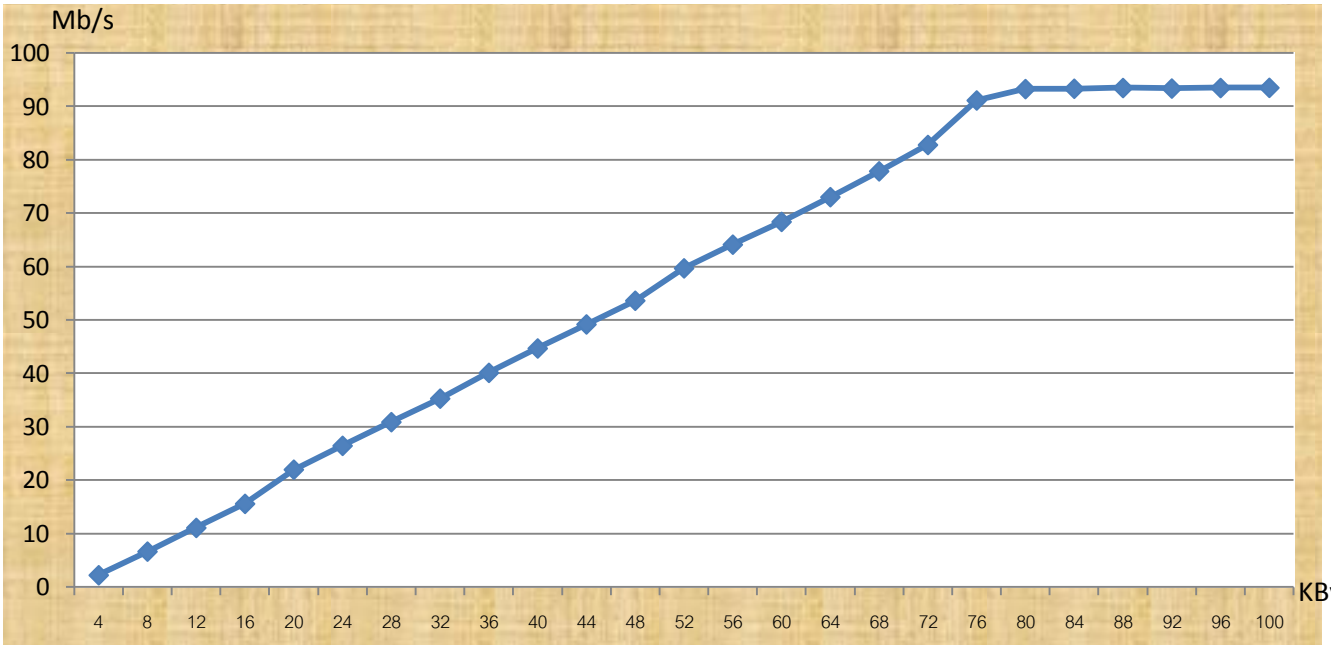
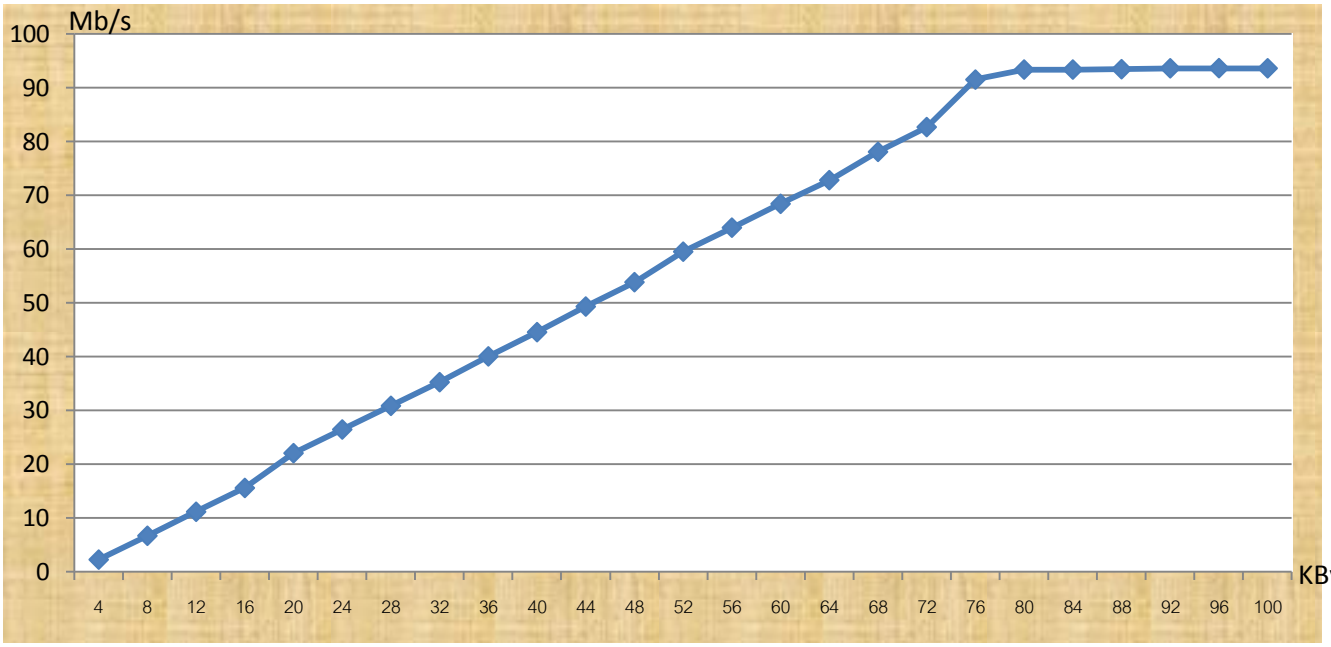


Table 2: Reno algorithm (plot from average throughput)



Conclusion

Throughput increases, as window size increases. But reno and cubic algorithms have no effects on the TCP throughput.

Appendix

Maximum Segment size

MSS (Bytes)	Throughput1 (Mbit/s)	Throughput2 (Mbit/s)	Throughput3 (Mbit/s)	Throughput(average) (Mbit/s)
1500	93.7	94	94	93.9
1400	92.8	93.4	93.5	93.23333333
1300	93.2	93.3	93.2	93.23333333
1200	92	90.8	92.4	91.73333333
1100	91.7	89.5	91.7	90.96666667
1000	91.1	91	91.2	91.1
900	90.1	90.1	87.2	89.13333333
800	81.2	80.4	81.4	81
700	70.2	69.5	70.4	70.03333333
600	60.1	61	60.4	60.5
500	51.4	50.7	50.4	50.83333333
400	41	40.6	40	40.53333333
300	30.7	30.7	30.6	30.66666667
200	20.5	20.5	20.6	20.53333333
100	9.75	9.76	9.78	9.763333333

Window size

Window size (kB)	Throughput1 (Mbit/s)	Throughput2 (Mbit/s)	Throughput3 (Mbit/s)	AVG
16	94.1	94.1	94.2	94.13333333
14.6	75.5	74.9	74.5	74.96666667
13.7	73.8	74.4	73.9	74.03333333
12.7	66.2	66.1	67.4	66.56666667
11.7	66.2	67.7	68	67.3
10.7	55.8	55.7	53.8	55.1
9.77	56.3	55.5	55.7	55.83333333
8.79	58.5	59.8	59.6	59.3
7.81	46.1	43.7	45.3	45.03333333
6.84	49.5	51.9	50.6	50.66666667
5.86	50.2	51.7	52.3	51.4
4.88	36.9	37	36.5	36.8
3.91	37.1	37	36.4	36.83333333
2.93	4.77	4.33	5.25	4.783333333
2	2.34	3.02	2.42	2.593333333

Data length

Data length(KBytes)	Throughput1 (Mbit/s)	Throughput2 (Mbit/s)	Throughput3 (Mbit/s)	AVG
300	93.8	93.9	94	93.9
275	94	93.8	94	93.93333333
250	94	93.8	93.9	93.9
225	93.9	94.1	93.8	93.93333333
200	93.4	93.9	93.7	93.66666667
175	94.1	94.1	94	94.06666667
150	82.7	81.2	82.5	82.13333333
125	68.9	68.9	68.9	68.9
100	55.1	55.1	55.1	55.1
75	41.3	41.3	41.3	41.3
50	27.5	27.5	27.5	27.5
25	13.7	13.8	13.8	13.76666667

Dropping packet (with iptables command)

Dropping packets at reciever	Throughput1 (Mbit/s)	Throughput2 (Mbit/s)	Throughput3 (Mbit/s)	Throughput(average) (Mbit/s)
0	93.8	93.8	93.8	93.80
1	84.3	84.7	86.2	85.07
2	54.5	58.1	49.8	54.13
3	39.3	40.9	42.1	40.77
4	27	21.1	25.2	24.43
5	15	15.3	17	15.77
6	8.69	11.2	11.8	10.56
7	5.12	5.79	6.91	5.94
8	5.99	3.27	5.93	5.06
9	4.2	2.94	3.82	3.65
10	3.85	4.17	2.25	3.42
11	2.51	2.21	2.88	2.53
12	2.15	2.61	1.97	2.24
13	1.97	1.28	1.45	1.57
14	2.04	1.16	0.972	1.39
15	1.31	1.26	1.2	1.26

Dropping packet (with tc command)

Dropping packets at sender	Throughput1 (Mbit/s)	Throughput2 (Mbit/s)	Throughput3 (Mbit/s)	Throughput(average) (Mbit/s)
1	78.5	89.3	78.8	82.20
2	44.6	44	48	45.53
3	22.4	26.3	26.1	24.93
4	14.9	17.7	17.4	16.67
5	10.4	13.3	11.9	11.87
6	7.41	7.53	8.23	7.72
7	5.45	5.34	6.1	5.63
8	4.92	4.31	3.44	4.22
9	2.79	3.21	2.8	2.93
10	2.76	1.75	2.48	2.33

Using Cubic algorithm

cubic	Packet Loss (%)	Throughput 1	Throughput 2	Throughput 3	Cubic Throughput
	0	94	93.9	94.00	93.97
	0.5	93.4	91.6	93.80	92.93
	1	86.7	90.1	83.50	86.77
	1.5	63.5	68.3	65.70	65.83
	2	57.4	50.9	54.20	54.17
	2.5	46.3	40.4	41.30	42.67
	3	35.9	31.4	33.10	33.47
	3.5	28.6	34.5	26.00	29.70
	4	22.8	34.5	26.00	27.77
	4.5	16.5	18.9	21.90	19.10
	5	16.5	3.5	16.50	12.17
	5.5	15	11.5	11.00	12.50
	6	9.33	10.8	8.92	9.68
	6.5	9.1	8.42	7.89	8.47
	7	6.26	7.53	6.37	6.72

Using Reno algorithm

Reno	Packet Loss (%)	Throughput 1	Throughput 2	Throughput 3	Reno Throughput
	0	94.6	94.6	94.60	94.60
	0.5	93.9	93.7	93.70	93.77
	1	89.4	87.5	89.50	88.80
	1.5	76.9	81.1	75.50	77.83
	2	65.8	58	69.40	64.40
	2.5	55.8	60.3	49.80	55.30
	3	46.5	47.8	53.20	49.17
	3.5	39.8	32.1	29.00	33.63
	4	23.7	26.1	25.50	25.10
	4.5	22.1	19.1	20.10	20.43
	5	25.3	14.6	19.40	19.77
	5.5	16.8	10.3	9.81	12.30
	6	11.9	10.1	8.52	10.17
	6.5	9.24	7.21	7.55	8.00
	7	6.83	7.01	7.42	7.09

PART 2 : Change window size

Using Reno algorithm

Bandwidth	RTT	BDP	-w	Buffer Size	Max Adv. Window	Expected Throughput	Measured Throughput 1	Measured Throughput 2	Measured Throughput 3	Average Throughput	Accuracy
[Mb/s]	[ms]	[KBytes]	[KBytes]	[KBytes]	[KBytes]	[Mb/s]	[Mb/s]	[Mb/s]	[Mb/s]	[Mb/s]	[%]
100	10	125	4	8	6	4.8	2.21	2.21	2.21	2.21	46.04
100	10	125	8	16	12	9.6	6.63	6.64	6.64	6.64	69.13
100	10	125	12	24	18	14.4	11.1	11.1	11.1	11.10	77.08
100	10	125	16	32	24	19.2	15.5	15.6	15.5	15.53	80.90
100	10	125	20	40	30	24	22	22	22	22.00	91.67
100	10	125	24	48	36	28.8	26.4	26.4	26.4	26.40	91.67
100	10	125	28	56	42	33.6	30.8	30.8	30.8	30.80	91.67
100	10	125	32	64	48	38.4	35.2	35.2	35.2	35.20	91.67
100	10	125	36	72	54	43.2	40	40	40	40.00	92.59
100	10	125	40	80	60	48	44.5	44.5	44.5	44.50	92.71
100	10	125	44	88	66	52.8	49.2	49.3	49.3	49.27	93.31
100	10	125	48	96	72	57.6	53.8	53.8	53.8	53.80	93.40
100	10	125	52	104	78	62.4	59.6	59.5	59.3	59.47	95.30
100	10	125	56	112	84	67.2	63.9	64	63.9	63.93	95.14
100	10	125	60	120	90	72	68.4	68.4	68.3	68.37	94.95
100	10	125	64	128	96	76.8	72.8	72.7	72.8	72.77	94.75
100	10	125	68	136	102	81.6	78.1	78	78	78.03	95.63
100	10	125	72	144	108	86.4	82.6	82.6	82.7	82.63	95.64
100	10	125	76	152	114	91.2	91.4	91.5	91.5	91.47	100.29
100	10	125	80	160	120	96	93	93.5	93.5	93.33	97.22
100	10	125	84	168	126	100	93	93.5	93.5	93.33	93.33
100	10	125	88	176	132	100	93.5	93.3	93.4	93.40	93.40
100	10	125	92	184	138	100	93.6	93.5	93.6	93.57	93.57
100	10	125	96	192	144	100	93.6	93.6	93.6	93.60	93.60
100	10	125	100	200	150	100	93.6	93.5	93.6	93.57	93.57

Using Cubic algorithm

Bandwidth	RTT	BDP	-w	Buffer Size	Max Adv. Window	Expected Throughput	Measured Throughput 1	Measured Throughput 2	Measured Throughput 3	Average Throughput	Accuracy
[Mb/s]	[ms]	[KBytes]	[KBytes]	[KBytes]	[KBytes]	[Mb/s]	[Mb/s]	[Mb/s]	[Mb/s]	[Mb/s]	[%]
100	10	125	4	8	6	4.8	2.23	2.23	2.23	2.23	46.46
100	10	125	8	16	12	9.6	6.64	6.66	6.67	6.66	69.34
100	10	125	12	24	18	14.4	11.1	11.1	11.1	11.10	77.08
100	10	125	16	32	24	19.2	15.6	15.6	15.6	15.60	81.25
100	10	125	20	40	30	24	22	22	22	22.00	91.67
100	10	125	24	48	36	28.8	26.5	26.4	26.5	26.47	91.90
100	10	125	28	56	42	33.6	30.9	30.9	30.9	30.90	91.96
100	10	125	32	64	48	38.4	35.4	35.2	35.3	35.30	91.93
100	10	125	36	72	54	43.2	40.2	40.1	40.1	40.13	92.90
100	10	125	40	80	60	48	44.7	44.7	44.7	44.70	93.13
100	10	125	44	88	66	52.8	49.2	49.2	49.2	49.20	93.18
100	10	125	48	96	72	57.6	53.6	53.7	53.6	53.63	93.11
100	10	125	52	104	78	62.4	59.7	59.7	59.7	59.70	95.67
100	10	125	56	112	84	67.2	64.1	64.2	64.1	64.13	95.44
100	10	125	60	120	90	72	68.4	68.4	68.4	68.40	95.00
100	10	125	64	128	96	76.8	73	73	73.1	73.03	95.10
100	10	125	68	136	102	81.6	77.9	77.9	77.8	77.87	95.42
100	10	125	72	144	108	86.4	82.8	82.8	82.8	82.80	95.83
100	10	125	76	152	114	91.2	91.1	91.2	91.1	91.13	99.93
100	10	125	80	160	120	96	93.4	93.1	93.3	93.27	97.15
100	10	125	84	168	126	100	93.3	93.3	93.4	93.33	93.33
100	10	125	88	176	132	100	93.5	93.4	93.5	93.47	93.47
100	10	125	92	184	138	100	93.5	93.3	93.3	93.37	93.37
100	10	125	96	192	144	100	93.5	93.5	93.4	93.47	93.47
100	10	125	100	200	150	100	93.5	93.5	93.5	93.50	93.50