# Experimentation with Linux TCP versions To understand the Congestion Control Mechanism

Shiva Prasad Thagadur Prakash
547903
shiva.thagadurprakash@aalto.fi

Viswanathan Manihatty Bojan
547039
viswanathan.manihattybojan@aalto.fi

## I. INTRODUCTION

TCP (Transmission Control Protocol) is a standard that explains how a network can be established between any two nodes and thereby providing a medium for data transfer. It is a connection oriented protocol and provides a reliable and sequential delivery of the packets between the hosts that employ the TCP suite. The prime feature of the TCP protocol is the way how it handles the congestion and ensures a reliable delivery of the packets over the network. The TCP protocol is designed to read the network conditions and establish a healthy transfer of data by adjusting its window size and Slow Start threshold values.

Since 1988, there has been a lot of TCP versions that had undergone revisions in order to enhance the way they handle the congestion. Linux supports some of the popular TCP versions such as Cubic, Reno, Vegas, Veno, BIC, etc.

As a part of this report, we decided to figure out how the congestion is being handled by the TCP Versions – Cubic, Reno and Vegas. We also tried to understand its congestion handling behavior in a network path with disruptions. This was done by inducing and modifying the network related parameters like delay, bandwidth and loss in the connectivity path using the traffic control (tc) commands.

Finally, the experiments were conducted and the results were recorded using tcp_probe and Iperf. The behavior of the congestion control in different TCP variants were simultaneously plotted using GNUPLOT. And the throughput data was collected separately and displayed in tabular format. The same is then compared and studied with other TCP variants under each scenarios.

## II. EXPERIMENTATION SETUP

Fig.1 shows the experimentation setup. The experimentation setup consists of two end hosts and a Wifi router. One of the hosts acted as a client and the other acted as a server. Both the hosts were running on Ubuntu 14.04 LTS and they were connected to a wireless router. The

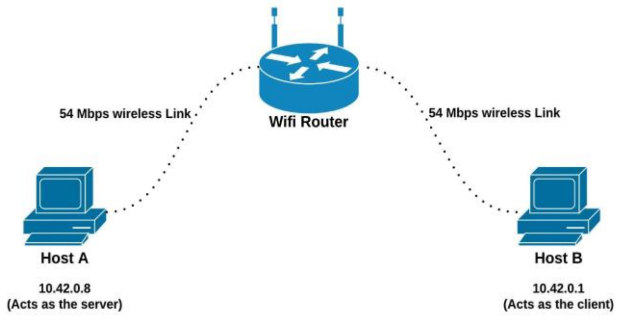

Fig.1- Experimentation Setup

transmission and the reception rates of the wireless router with its end hosts is 54 Mbps. Both the hosts were under the same subnet. The below network diagram depicts the experimental setup that was considered.

By default, the bandwidth between the two end hosts was measured using Iperf and it was 10 Mbps. The delay between the two end hosts came around 5 ms.

## III. PROCEDURE

A. The list of available TCP versions in the Linux distros can be found using the below command
*sysctl net.ipv4.tcp_ available_ congestion_ control*

B. Then choose the TCP version that needs to be used in the current experimentation using the below command
*sysctl -w net.ipv4.tcp_congestion _control=<tcp_version>*

C. Set the network path properties using the tc, netem and the tbf commands.

- To Modify Bandwidth:
*sudo tc qdisc add dev wlan0 root handle 1: tbf rate <desired_bandwidth>mbit buffer 3000*

- To Introduce Delay:
*sudo tc qdisc add dev wlan0 parent 1:1 handle 10: netem delay <desired_delay>ms*

- To Introduce Packet Loss:
  *sudo tc qdisc add dev wlan0 root netem loss <loss_percentage>%*

- To clean up the network properties that were set:
  *sudo tc qdisc del dev wlan0 root*

D. The TCP data was then captured using tcp_probe and Iperf.

```
sudo modprobe tcp_probe port=5001 full=1
sudo cat/proc/net/tcp_probe >  <file_name> &
pid = $!
< Iperf commands to begin the TCP data transfer >
sudo kill $pid
```

E. Iperf Commands To Begin The TCP data transfer:

Server End  : *iperf –s*
Client End  : *iperf –c <server ip address>*

F. Data Plotting using GNUPLOT:

The obtained data from the above commands will then be plotted using GNUPLOT. The GNUPLOT gives a graphical view of the test results obtained which can then be used for analysis purpose.

**Script.sh**

```
#! /bin/bash
 gnuplot -persist <<"EOF"
 set style data linespoints
 set xlabel "time (seconds)"
 set ylabel "Segments (cwnd, ssthresh)"
 plot "<file_name>" using 1:7 title "snd_cwnd",\\
"<file_name>" using 1:($8>=2147483647 ? 0 : $8) title
"snd_ssthresh"
EOF
```

G. The graph is then plotted by running the command
*sh script.sh*

The graphs help us to understand the behavior of the TCP variants with respect to the congestion control mechanisms that they were equipped with. Though the results do not exactly replicate the behavior of the TCP variants as understood theoretically, they were able to closely follow on with the congestion control techniques that they employ. The differences in the behavior can be explained based on the network parameters and their behavior in the respective experimental setup.

After setting the network properties, the experiment was made to run for 200 seconds and the same was repeated 3-4 times for collecting the data.

## IV. EXPERIMENT – RESULTS & ANALYSIS

The TCP tests were conducted under four scenarios by varying the network related parameters like bandwidth, delay and packet loss. The different scenarios that were considered are:

A. High Bandwidth + High Delay
B. Low Bandwidth + High Delay
C. High Bandwidth + Low Delay
D. Packet loss of 1% and 5% in the existing environment
E. TCP Veno with a Packet loss of 1%

### *A. Scenario 1 – High Bandwidth + High Delay (Bandwidth - 50 Mbps & Delay - 500 ms)*

By default, the environment has a bandwidth of 10 Mbps and 5ms delay. By taking a bandwidth of 50 Mbps here, we understand that it will try to make use of the complete bandwidth available in the link.

Fig.2 shows the behavior of the Cubic TCP under the current network parameters. We can observe that the congestion window CWND shows a steady progress in the initial stage. This is the slow start phase of the TCP where the congestion window grows by a cubic function. Once after reaching its maximum capacity, it begins to consistently transmit the packets where the CWND size is close to 900 segments for a finite time. During this frame, most of the bandwidth will be used efficiently. After 90 seconds of the run, Cubic detects the congestion because of packet loss. At this point, the size of the congestion window is brought down to the SSTHRESH value. The decrease in the size of the congestion window at the time of an attack is dependent on the value '**β**', which is a constant multiplication decrease factor. Now that the TCP Cubic is RTT independent, the congestion window size once again begins to increase using the cubic function.

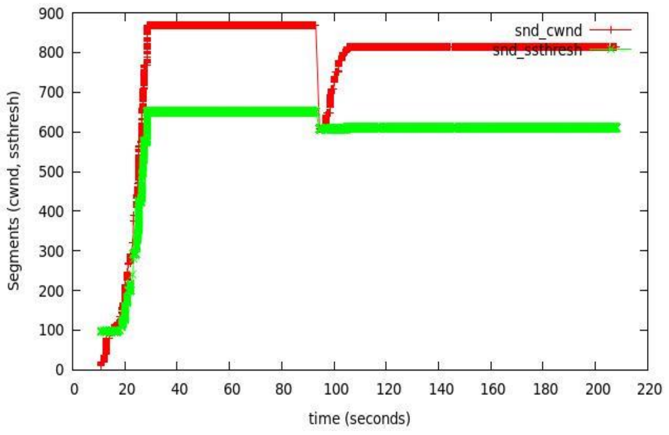

Fig.2- TCP Cubic under Scenario 1

Fig.3 and Fig.4 shows the behavior of the TCP variants Reno and Vegas respectively whenever congestion occurs.

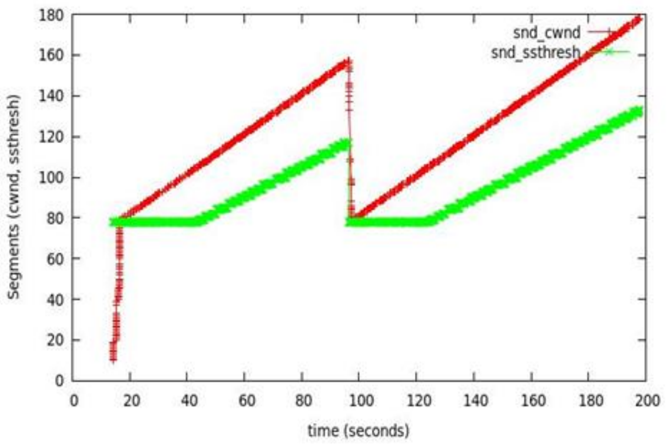

Fig.3- TCP Reno under Scenario 1

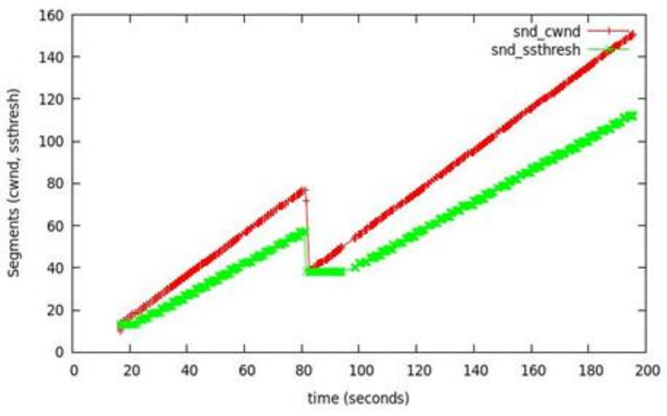

Fig.4- TCP Vegas under Scenario 1

We can observe that both Reno and Vegas were showing similar behavior against the congestion. However, it needs to be noted that the overall throughput of Reno was comparatively better than that of the Vegas.

In the case of Reno, a slow start is observed where the CWND window is increased for every ACK that is received. And this explains the steep rise in the CWND size. After 20 seconds from the run, it reaches the Slow Start threshold. And from now on, the congestion window size will increase additively until a congestion is seen in the network. At the time of such an attack, we can notice that the TCP behavior shows a multiplicative decrease by reducing the size of the congestion window to half and also setting the SSTHRESH value to half of the congestion window. Now, both the CWND and the SSTHRESH will be residing at the same level. And from there, the CWND size increases additively till a point where the Reno TCP can transmit the packets to its full capacity without any packet loss.

In the case of Vegas, we can observe that there is an additive increase in the size of the congestion window right from the start. The SSTHRESH also displays a behavior similar to that of the CWND. Vegas reacts to congestion based on the delay in the acknowledgement received. Here, whenever such a delay is encountered, Vegas considers it as a sign of congestion and brings down the value to the SSTHRESH value. Later, it increases its congestion window linearly and the same can be seen from the 80th second in Fig.4.

Table-I gives the summary of the throughput data exchanged between the two end hosts in each of the TCP variants.

Table I- Throughput data under Scenario 1

| **TCP Variant** | **Messages Exchanged** | **Average Bandwidth Observed During transmission** |
|---|---|---|
| Cubic | 162 MBytes | 6.78 Mbits/sec |
| Reno | 35.1 MBytes | 1.47 Mbits/sec |
| Vegas | 22.0 MBytes | 914 Kbits/sec |

### B. *Scenario 2 – Low Bandwidth + High Delay (Bandwidth - 3 Mbps & Delay - 100 ms)*

The default bandwidth between the two end hosts is around 10 Mbps and the delay is around 5 ms. Hence, in order to perform the experiment under the current scenario, the bandwidth was reduced to 3 Mbps and delay was set to 100 ms

The Fig.5 shows the obtained graph for the Cubic TCP under such conditions. The CWND size begins to grow using the cubical function. We can observe that, there is a congestion that is encountered close to the 80th second. This is contrary to the behavior as seen during the previous scenario where the congestion was noticed at around 90 seconds. This is because, in the previous scenario, with a higher delay (of 500 ms), the buffers available in the network path will have sufficient time to handle the packets and regulate them more comfortably for a longer duration before the existence of the congestion. However, in the present experimental set up with a slightly lower delay, more and more packets are being transferred in the network path in a smaller time duration. This will be an overhead for the routers and they begin to lose the packets much earlier. Once after the packet loss, the CWND size is reduced to SSTHRESH value and it once again begins to grow with respect to a cubical function.

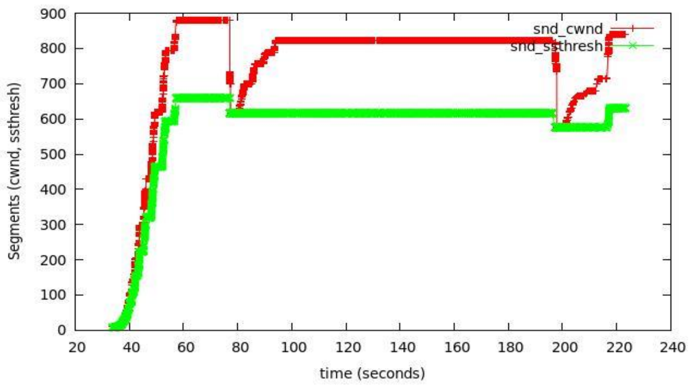

Fig.5- TCP Cubic under Scenario 2

TCP Reno, as seen in Fig.6, showed an aggressive slow start phase in the beginning. Later after the start, it displayed the behavior typical to its characteristic approach of handling the congestion. That is, the CWND and the SSTHRESH values were set to half of the size of the congestion window.

With the current delay of 100 ms, the sender sends the packets and will receive the ACK very soon. As a result of this, the CWND size increases steadily and there is a steep rise in the Slow start phase of the Reno. Once after reaching the SSTHRESH value, the congestion avoidance phase is exhibited with an additive increase. But very soon after 40 seconds of running the experiment, a congestion in the network is faced. This is because of the high traffic in the network with a low latency.

Now, considering the behavior of the Vegas, as seen in Fig.7, we can recognize that, because of the small delay, the response ACK messages are received earlier at a faster rate and the congestion window grows high steadily within the first 40 seconds. Very soon, it meets up with a congestion and both the CWND and the SSTHRESH values reduce and begin to transfer the data at a minimum and consistent rate.

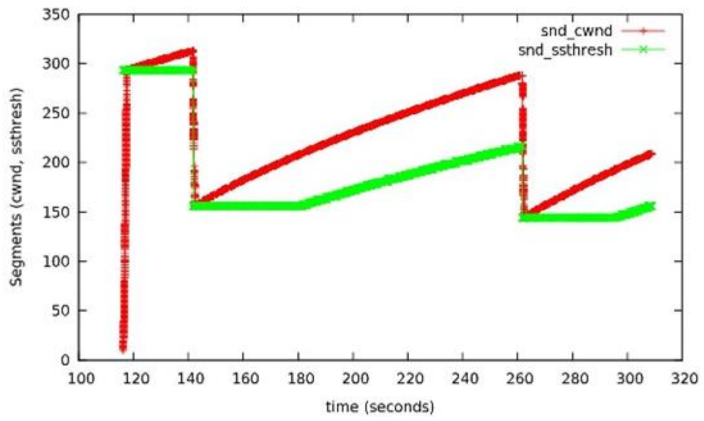

Fig.6- TCP Reno under Scenario 2

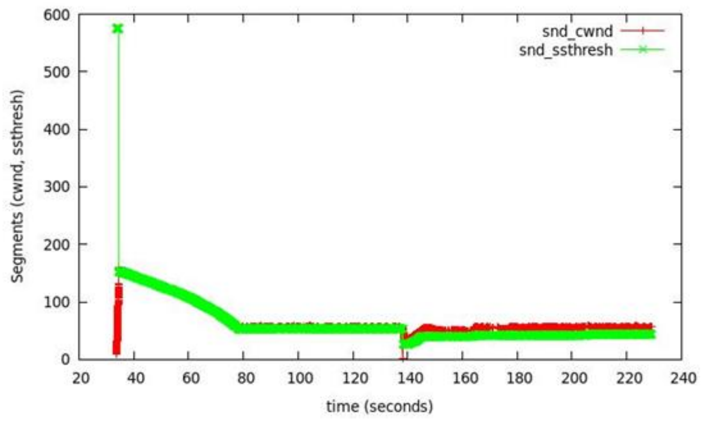

Fig.7- TCP Vegas under Scenario 2

Table II gives the summary of the amount of the messages exchanged and the average bandwidth speed between the two end hosts in each of the TCP variants

Table II- Throughput data under Scenario 2

| TCP Variant | Messages Exchanged | Average Bandwidth Observed During transmission |
|---|---|---|
| Cubic | 68.0 MBytes | 2.84 Mbits/sec |
| Reno | 68.9 MBytes | 2.88 Mbits/sec |
| Vegas | 64.9 MBytes | 2.71 Mbits/sec |

*C. Scenario 3 – High Bandwidth + Low Delay (Bandwidth - 50 Mbps & Delay - 5 ms)*

The network parameters were changed in order to bring an environment where the bandwidth was limited to 50 Mbps and the delay was limited to 5 ms. A bandwidth of 50 Mbps was chosen since we decided to consider a value that lies very close to the limit of data transmission and data reception rates associated with 802.11, which is 54 Mbps. Here we can recognize that, the behavior of all the three TCP variants were very similar to the ones that was studied in the earlier experimental scenario.

Fig.8 shows the recorded graph of Cubic TCP. In the case of TCP Cubic, it makes use of the cubic function in order to increase its congestion window and the same can be seen in the graph where it grows steadily and remains at a point where it can deliver the messages at its highest rate. The SSTHRESH value is also increased in a cubic fashion initially. Because of a very high bandwidth available, it was able to remain in a consistent state from 60th to 160th second. Then, congestion is created and the window size is reduced by '**β**', a constant multiplication decrease factor. At the time of the attack, the congestion window grows once again from the SSTHRESH value.

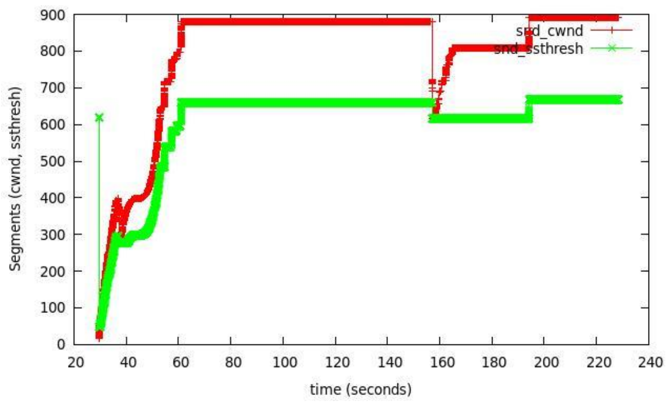

Fig.8- TCP Cubic under Scenario 3

Reno, as seen in Fig.9, displays its commonplace behavior with a very good throughput. It shows an aggressive slow start phase, then moves to congestion avoidance phase by

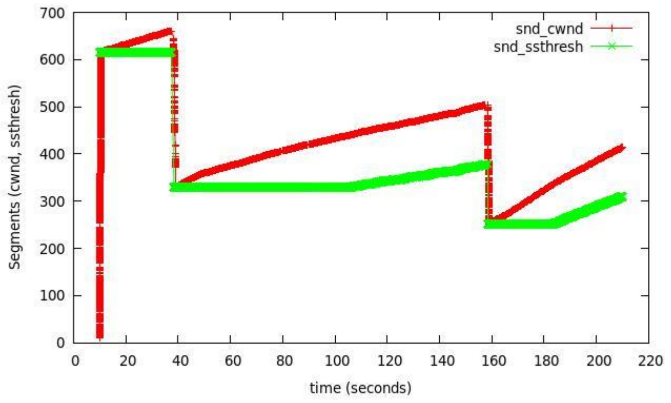

Fig.9- TCP Reno under Scenario 3

increasing the window size additively. At the time of an attack, the window size is reduced to half and it again starts from the SSTHRESH value.

On the other hand in the case of Vegas, as seen in Fig. 10, we are able to observe a steady transmission of data without much disruptions. Though the rate at which the segments are transferred is very low, it maintained a steady path. Additionally, with a very low latency that was set, TCP Vegas was successful in exhibiting a behavior better than that of TCP Reno. When comparing the overall throughput of the data that was exchanged between the two hosts, Vegas was able to transfer higher marginally volume of messages.

Table III explains the Throughput data associated with the TCP variants under this condition.

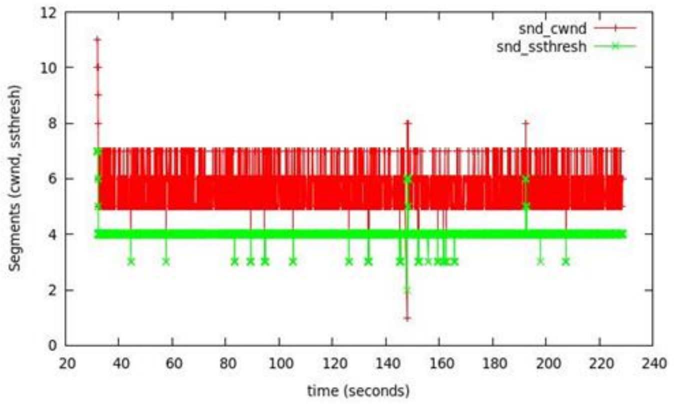

Fig.10- TCP Vegas under Scenario 3

Table III- Throughput data under Scenario 3

| TCP Variant | Messages Exchanged | Average Bandwidth Observed During transmission |
|---|---|---|
| Cubic | 260.0 MBytes | 10.9 Mbits/sec |
| Reno | 218 MBytes | 9.14 Mbits/sec |
| Vegas | 252 MBytes | 10.6 Mbits/sec |

## *D.Scenario 4 – Packet loss of 1% and 5% in the existing environment*

The network bandwidth and the delay were set in a normal state. In the normal state, the bandwidth between the hosts was 10 Mbps and the delay was 5 ms. A loss of 1% was then injected using the tc command. This makes one out of 100 packets to be randomly dropped. The behavior of the TCP variants was then studied for the congestion control. The Fig.11, Fig.12 and Fig.13 shows the observed pattern of Cubic, Reno and Vegas respectively.

This is an independent setup where no external delay that was added and no bandwidth was limited. We can observe that the packet loss of 1% did not impact the final throughput of the TCP variants to a great extent. However, the factors that were affected were the CWND size and the SSTHRESH size, thereby drastically limiting the number of packets that can be exchanged between the two end hosts. Usually, in any network path, a loss of 1% does not have a huge impact. Any loss of close to 5% or higher will begin to impact the network. Adding on, in all the three TCP variants, we can observe that the CWND size did not show any wide deflections from the SSTHRESH value.

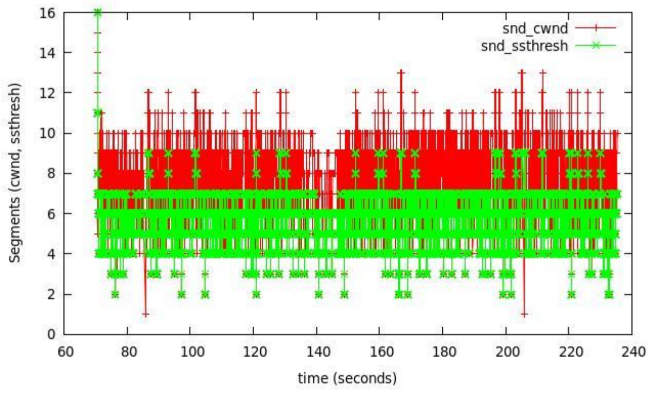

Fig.11- TCP Cubic with 1% packet loss

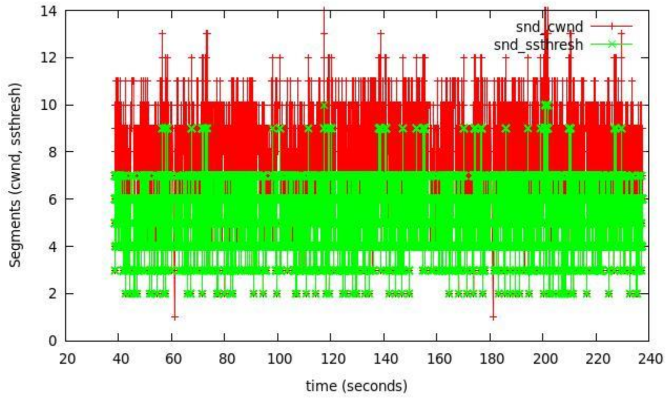

Fig.12- TCP Reno with 1% packet loss

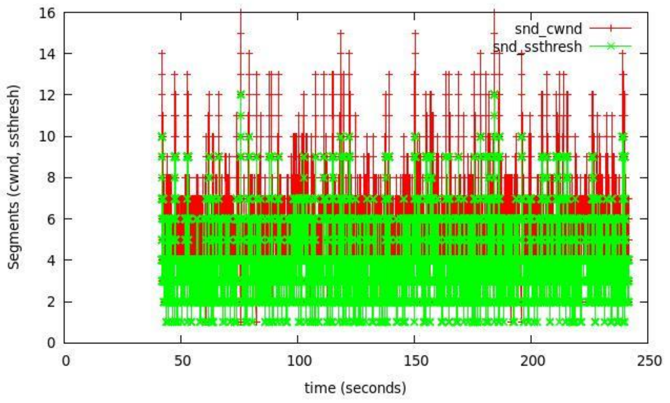

Fig.13- TCP Vegas with 1% packet loss

We can now understand that the induced delay was sufficient enough to stop all the three TCP variants from exhibiting their maximum potential to use the full bandwidth. And in terms of the overall throughput that was recorded, we can come to know from Table IV that Reno was able to perform insignificantly better compared to its other two counterparts by exchanging more data even though all of them maintained an average bandwidth of 11 Mbits/sec during transmission.

Table IV- Throughput data under Scenario 4 for 1% loss

| **TCP Variant** | **Messages Exchanged** | **Average Bandwidth Observed During transmission** |
|---|---|---|
| Cubic | 264 MBytes | 11 Mbits/sec |
| Reno | 273 MBytes | 11.4 Mbits/sec |
| Vegas | 256 MBytes | 10.7 Mbits/sec |

A delay of 5% was then injected in the environment and the throughput was recorded. We were able to observe a considerable decrease in the average bandwidth. Fig.14, Fig.15 and Fig.16 displays the graph obtained at the end of the experiments and Table V contains the throughput results.

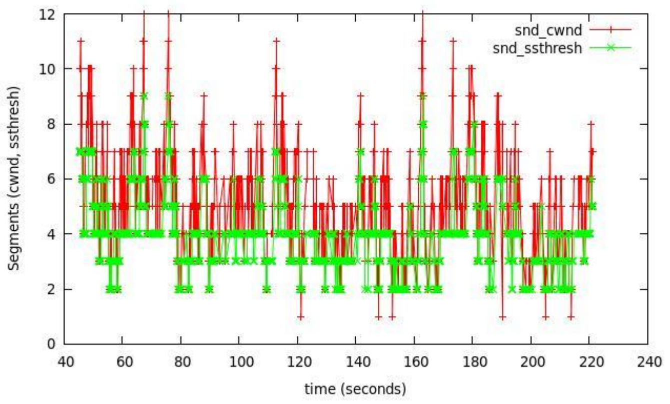

Fig.14- TCP Cubic with 5% packet loss

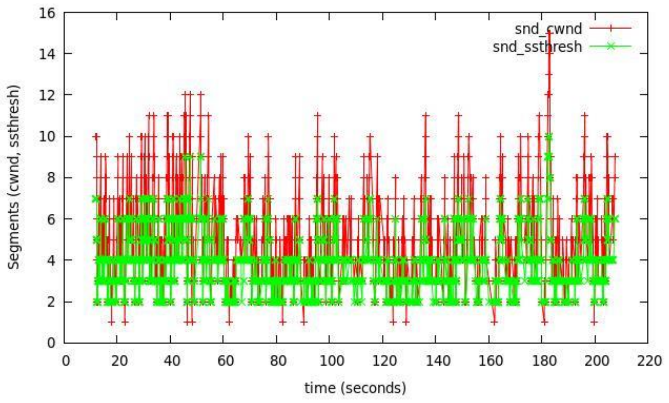

Fig.15- TCP Reno with 5% packet loss

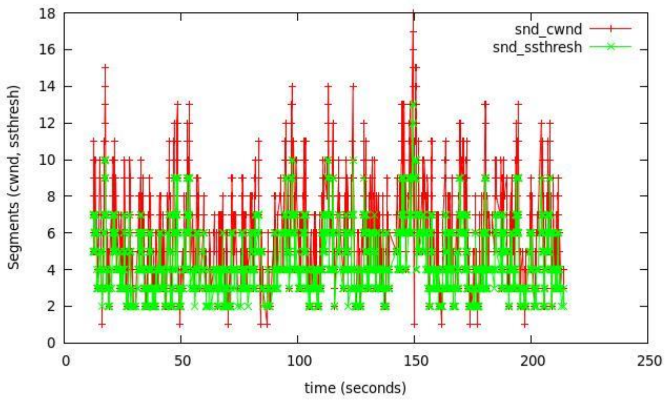

Fig.16- TCP Vegas with 5% packet loss

Table V- Throughput data under Scenario 4 for 5% loss

| **TCP Variant** | **Messages Exchanged** | **Average Bandwidth Observed During transmission** |
|---|---|---|
| Cubic | 12.1 MBytes | 503 Kbits/sec |
| Reno | 11.9 MBytes | 488 Kbits/sec |
| Vegas | 13.5 MBytes | 558 Kbits/sec |

## *E. Additional Scenario – TCP Veno with a Packet loss of 1%*

Apart from the above set of scenarios, we performed the experiment again with TCP Veno which is a dedicated TCP version in any wireless environment. The existing bandwidth of 10 Mbps and delay of 5ms was taken. Along with this, a packet loss of 1% was injected. Fig.17 depicts the graph that was obtained from the TCP data. TCP Veno displayed a better throughput even when there was a fair amount of packet loss.
1% packet loss in any network is an acceptable loss where the network throughput is not completely hampered.

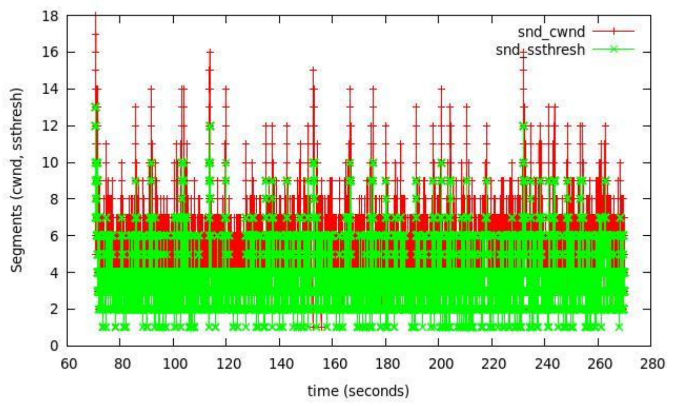

Fig.17- TCP Veno with 1% packet loss

The total size of the messages that were exchanged between the end hosts was 279 MBytes at an average rate of 11.7 Mbits/sec. Throughput of TCP Veno is comparatively better than all of the TCP versions that is considered here for our experiment.

## V.CONCLUSIONS

We can clearly predict from the results of the experiments that, in a setup with high bandwidth and a high delay, TCP Cubic always has a upper hand in the overall throughput and the performance after a congestion is experienced. And this is because, TCP cubic is RTT independent and the growth of its congestion window is defined in real-time. TCP Reno and Vegas are RTT dependent. And since that the delay is high, their congestion window size will grow at a slower rate.

In the case of the low bandwidth and high delay, all the three TCP variants considered here in the experiment were able to showcase a similar behavior both in the number of messages that were exchanged and the average rate at which the messages were exchanged. Each of the TCP variants showcased only a marginal difference in their final throughput. Since that the bandwidth is reduced here, TCP Cubic could not display an exemplary behavior even when there is a scope for the real time congestion window increase. TCP Reno and Vegas, however were able to make the most of their potential to give an increased throughput.

In the case of high bandwidth and low delay, we can notice that all the three variants were able to transfer a large amount of message and with a very good transmission rate. This is because, in an environment with low delay, the RTT will be very quick. And this helps both the TCP Reno and Vegas to increase their congestion window at a much quicker rate and thereby making way for additional messages to be transferred. TCP Cubic, on the other hand, is RTT independent and increases its congestion window in real time. And hence, we could see a very good throughput in all of the TCP variants.

A packet loss of 1% is an acceptable one which might not affect the network in most of the cases. The existing environment was not modified with bandwidth and delay. We can observe that, even in the presence of the loss, a very good throughput is delivered. However, the congestion window growth is affected. All the three variants were able to increase their window size only to an average segment size of 12. Without the loss, the environment would have produced a throughput far better. Apart from this, experiments were also conducted with a 5% loss. And we can observe that it was a severe case and the message transmission rates were affected heavily.

## REFERENCES

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

[2] TCP Vegas: End to End Congestion Avoidance on a Global Internet, by Lawrence S.Brakmo, Student Member, IEEE, and Larry L.Peterson

[3] Wikipedia page for TCP Cubic https://en.wikipedia.org/wiki/CUBIC_TCP

[4] Wikipedia page for TCP Vegas https://en.wikipedia.org/wiki/TCP_Vegas

[5] Wikipedia page for TCP Reno https://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm#TCP_Tahoe_and_Reno