Analysis of Congestion control algirthms in Linux Supported TCP versions

In TCP Cubic, Veno and Reno

Sami Karvonen
Department of Computer Sience
Aalto University School of Science
Espoo, Finland
Email: sami.karvonen@aalto.fi

Siddharth Prakash Rao
Department of Computer Science
Aalto University School of Science
Espoo, Finland
Email: siddharth.rao@aalto.fi

Abstract - In this paper we have experimented and analysed three of the TCP flavours namely TCP Cubic, TCP Reno and TCP Veno. The experiment is based on network properties like "Delay" and "Packet Loss". The evaluation is based on the careful study of the behaviour of each version's plot of Segments (cwnd, ssthresh) against time in seconds, along with the Round Trip Time(RTT).

***Keywords*: Transmission Control Protocol, TCP Cubic, TCP Veno, TCP Reno, Congestion Window**

I. INTRODUCTION

A. *TCP (in general)*

Transmission Control Protocol is a protocol in transport layer, which provides reliable, error-checked communication between two(or more) computers connected with each other via Local Area Network(LAN), Intranet or the Internet[1]. The main feature of TCP is congestion control along with flow control, error detection and reliability in transmission.TCP uses various mechanisms to achieve high performance. TCP adopts to achieve good performance in the available bandwidth by adjusting its window size. Though the basic operations of TCP has not changed, there are various versions of TCP developed, and each differ in the way they handle the window size, transmit the data and hence control the congestion.

B. *Working Principle of TCP:*

In the base version of TCP congestion control, "Additive Increase/ Multiple Decrease" sliding window strategy is used to avoid congestion in the network[2].

Congestion Window (cwnd) is the the variable used in TCP as per the senders estimation of the amount of data that can transmitted through the network without the packets being lost. This variable is set to one or two segments, after which it will be increased depending on either of "Slow start Phase" or "Congestion Avoidance Phase". "Slow Start Threshold"(sstresh) is the variable introduced to determine which of the phase should be used. The TCP algorithm chooses Slow start phase if congestion windows is less than Slow Start Threshold (cwnd < sstresh); and it chooses congestion avoidance phase if congestion windows is greater than or equal to than Slow Start Threshold (cwnd >= sstresh). In the former, the congestion window size is increased exponentially by one segment of incoming acknowledgement(ACK); whereas in the latter, the congestion window size is increased at the rate of one segment per round-trip-time(RTT).

In the case of heavy congestion or out-of-order segments, the sender becomes acknowledged by the receiver on the receival of at least 3 duplicate acknowledgement packets(DUPACK) and the sender retransmits a segment along with setting the sshtresh to half of the the current value. Also, the cwnd is set to sshtresh added by three segments or 1 depending on the case. The retransmission due to incoming duplicate ACKs is called "fast retransmit". Until all the segments are acknowledged by the receiver, the TCP sender follows the fast recovery algorithm. The congestion window is temporarily increased for each incoming duplicate ACK to permit forward transmission of a segment. When the fast recovery phase is over its value is set to the number of segments at the beginning of the fast recovery. This is how the congestion control is done in base version of TCP.

Though the basic working principle of TCP is not different, the way each TCP versions handle the congestion window and the algorithm to control congestion is slightly varies, and this affects the performance. We have considered *TCP Cubic, TCP-Veno and TCP-Reno* for our experiments. We have conducted the experiments for these three TCP versions, analysed and evaluated the same in same environment to get consistent results. The comparison is based on congestion window size and the Round-trip time in various conditions(Delay and Loss).

II. BACKGROUND

A. *TCP Cubic*

TCP Cubic[3] is the default TCP version in Linux kernels 2.6.19 and above.It is optimized for networks with high bandwidth and high latency(e.g. Long Fat Networks). In this, the window size is a cubic function of time. The TCP Cubic version is not dependent on the receipt of acknowledgment by the receiver to increase the window size unlike the base TCP version. Here the TCP window size rely on the last congestion event.In standard TCP, flows with very short RTTs will receive ACKs faster and therefore have their congestion windows

grow faster than other flows with longer RTTs. CUBIC allows for more fairness between flows since the window growth is independent of RTT. In the standard TCP flavors, the acknowledgment will be received faster for short RTTs and hence the congestion window grows faster when compared to other flows with long RTTs. But in TCP-Cubic, there is more fairness between flows as the window growth is not dependant on Round Trip Time.

B. TCP Veno

TCP Veno module is congestion control module to improve TCP performance over wireless networks. [4] It is an improvement over TCP Reno congestion control algorithm by using the estimated state of a connection based on TCP Vegas. TCP Veno reduces "blind" reduction of TCP window regardless of the cause of packet loss. This TCP version distinguishes between random loss(non- congestion state) and congestion loss (congestion state). Also depending on this difference it refines the congestion window adjustment. In the wireless environment, the packet loss is because of the noise and link error. Since TCP Veno is intelligence enough to differentiate them, it avoids base RTT to change from time to time and thus the connection can stay in stay for longer in the operating region. In short, Veno refines the Additive Increase, Multiplicative Decrease (AIMD). When the packet loss is more, Veno's performance will be good when compared to other protocols.

C. TCP Reno

TCP Reno performs the tasks in four phases namely slow start, congestion avoidance, Fast Recovery and Fast Retransmit. [5]TCP Reno adjusts the window size based on the phase it performs. When the sender sends a packet, it starts a retransmit timer. When the timer expires and still the acknowledgement is not received, it considers that the packets are lost. It retransmits only those packets that is supposed to be lost and not the subsequent ones without waiting for the retransmission timer to expire by performing the congestion avoidance phase again.

III. *Experimentation Setup*

We used one system as client which has Linux Mint operating system and the other system as server with Ubuntu as the operating system. Both were connected using a router switch which with 100Mbps connection. We made sure that both system has same versions of TCP available for experiments. We ran the iperf in server mode in one machine, and in client mode in another machine. Client is the sender here and the TCP data was collected from the sender's side. The server acts as the bottleneck Emulator and the receiver.

IV. TOOLS

A. Iperf

Iperf is tool which we used to generate the network traffic. The advantage of using iperf is that it allows us to tune various network parameters. Iperf reports bandwidth and jitter loss (In case of TCP transmission). Iperf has to be run at server and client side, also iperf has its own TCP version (it is not dependant on TCP version of the underlying system). This can be selected by -Z flag (e.g. Iperf -c -Z cubic). In our experiment, since to make sure that the system we used is in synch with the experimental set up, we have selected the same TCP versions at system level as well as while initiating iperf.

B. Netem

Netem is a command line network emulator for testing network protocols. We used this emulator in combination with traffic control(tc) tool to set network parameters like delay and loss by choosing the appropriate interface device.

C. TCP_probe

In our experiments we used tcp_probe along with iperf to record the state of TCP connection in response to incoming packet.We analysed the tcp data captured by tcp probe by recording all the tcp conversation in a file which is running in background while the iperf session between client and server is initiated.

D. GNU Plot

GNUPLOT is the command line graphing utility. We used GNUPLOT to graphically represent the data we captured using tcp-probe in graph of time in seconds in x-axis versus segments (cwnd, sshtresh) on y-axis. The visual representation done using gnuplot gave us the better idea of behaviour of the TCP version which we had applied.

V. PROCEDURE (COMMANDS)

A. Applying the TCP version:

The list of TCP versions available in the system is found using the command:

ls /lib/modules/3.8.0-31-generic/kernel/net/ipv4 | grep tcp*

To choose and apply a particular TCP version (say Reno) , the following command is used:

echo "westwood" >/proc/sys/net/ipv4/tcp_congestion_control

B. Capturing TCP data using TCP Probe while using Iperf

- sudo modprobe -r tcp-probe
- sudo modprobe tcp_probe port=5001
- sudo cat /proc/net/tcp_probe > file_name &
- sudo kill <pid_obtained_from_previous_command>

C. Adding Delay and/or loss using netem

To add delay/loss for the first time

sudo tc qdisc add dev eth0 root handle 1:0 netem delay <packet_loss>ms <packet_loss>2%

For the next repeats

sudo tc qdisc change dev eth0 root handle 1:0 netem delay <delay>ms <packet_loss>2%

D. TCP traffic generation using iperf

On the server side
iperf -s -Z <tcp_version_name> -i 1
On the Client side:
iperf -c <address_of_the_server> -Z <tcp_version_name> -t 40

E. Plotting the graph using GNUPLOT

The data obtained from the TCP traffic using the tcp-probe was plotted as graph using the GNUPLOT tool to plot with "Time" in seconds on X- axis and "TCP segments(cwnd, ssthresh)" on Y- axis. The code which we used is as follows

plot.sh
#! /bin/bash
gnuplot -persist <<EOF
set data style linespoints
show timestamp
set title "$1"
set xlabel "time (seconds)"
set ylabel "Segments (cwnd, ssthresh)"
plot "$1" using 1:7 title "snd_cwnd", \\
"$1" using 1:(\$8>=2147483647? 0: \$8) title "snd_ssthresh"
EOF
Running the output file with this script
sh plot.sh <tcpprobe_file>

VI. Analysis

A. No delay no loss:

This network is just for testing the setup and confirming that it's working in every case at the same rate. Window size graphs are pretty similar with no loss and no delay as seen in figures 1, 2 and 3.

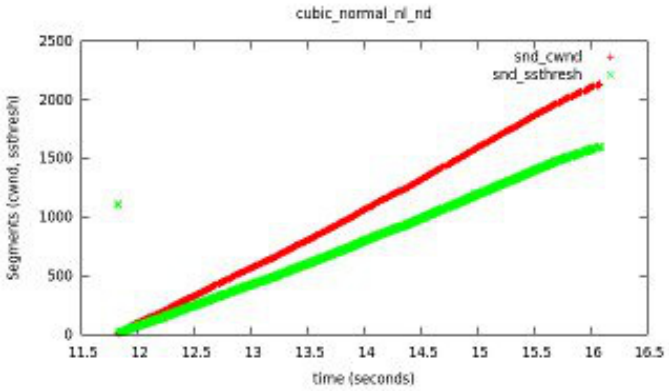

Figure 1, Cubic no loss no delay

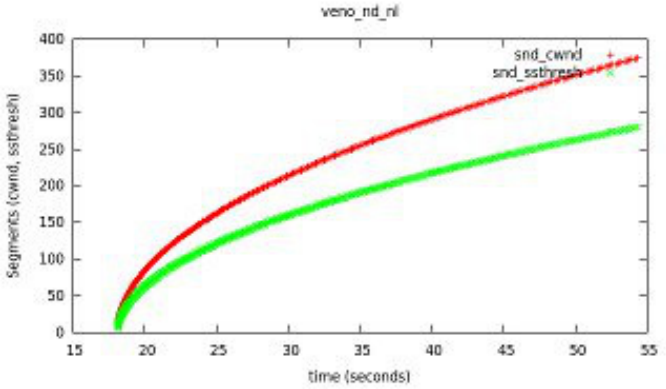

Figure 2, Veno no loss nodelay

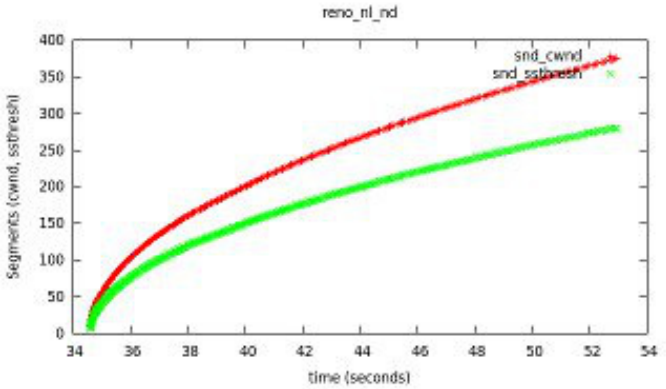

Figure 3, no loss no delay

B. 10ms delay 2% loss:

In this case the network delay is really small and the loss isn't very high either so this would simulate a wireless network at home where the wireless station is very close to the computer.

If we look at the zoomed pictures of Reno[Figure 4] and Veno[Figure 5] we can see that they are pretty similar but still the differences between the protocols are clear. In Reno the congestion window is increasing really fast which causes it to reach the limit faster and drop down to half the size while in Veno the window is only increased by one every 2 RTT's so it can stay longer in the higher window sizes. This results in Veno not dropping down in window size as often as Reno.

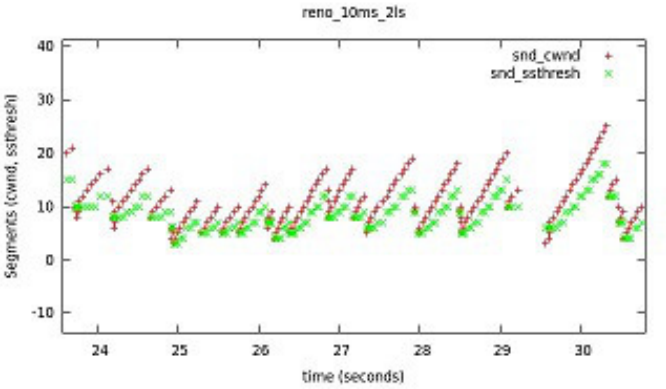

Figure 4, Reno 10ms 2% loss zoomed

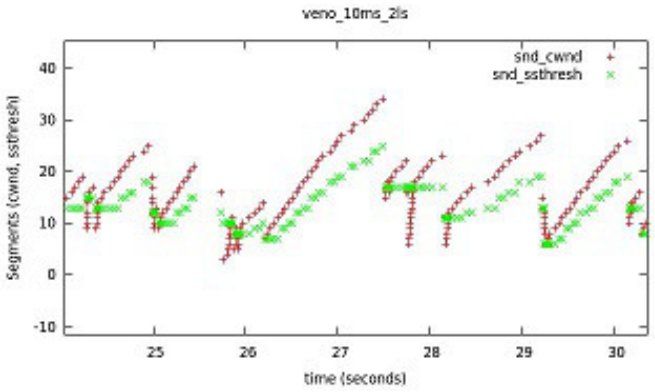

Figure 5, Veno 10ms 2% loss zoomed

In the case of Cubic[Figure 6] the window size is increased using a cubic function which allows it to stay closer to limit of the network longer. Also The window size isn't dropped as much as in the other protocols.

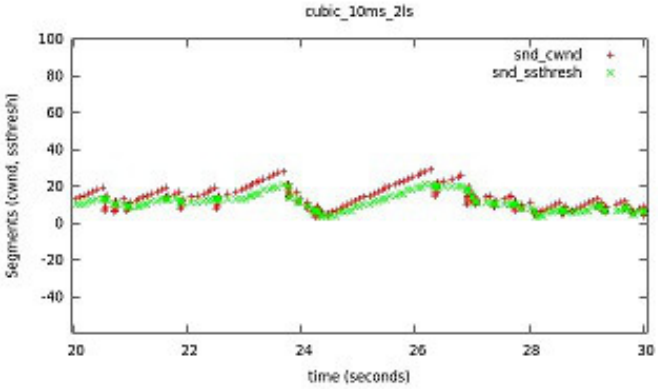

Figure 6, Cubic 10ms 2% loss zoomed

C. 100ms delay 2%loss:

This network has considerably more delay than the last one. This results in Reno and Veno congestion windows growing really slow but we can see same kind of behavior as in the last network when we grow the graph time scope from 5-10 seconds to 40 seconds.

In Cubic the congestion window is grown based on real time instead of RTT. We can see this in the graph as the window size is growing much more rapidly than in the other 2 protocols. Graphs of this network can be seen in [Figures 7, 8 and 9].

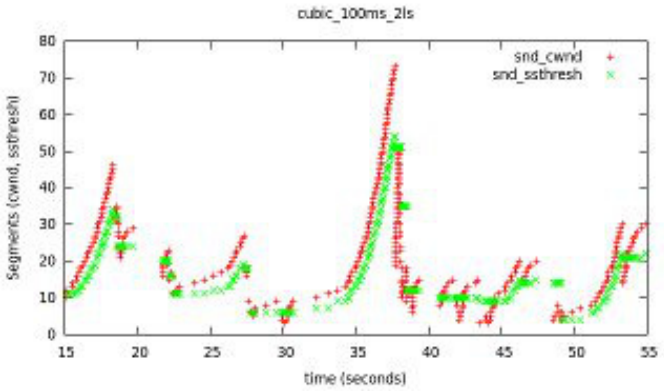

Figure 7, Cubic 100ms 2% loss

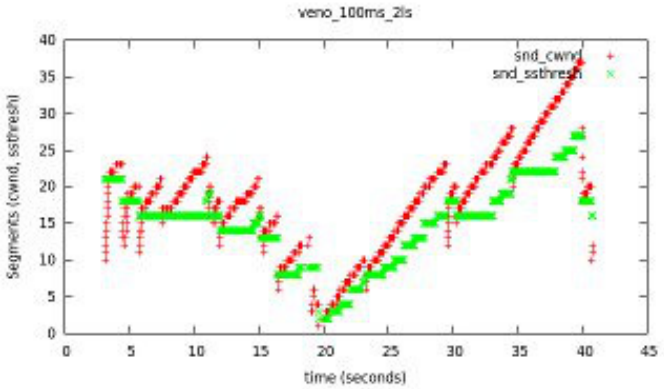

Figure 8, Veno 100ms 2% loss

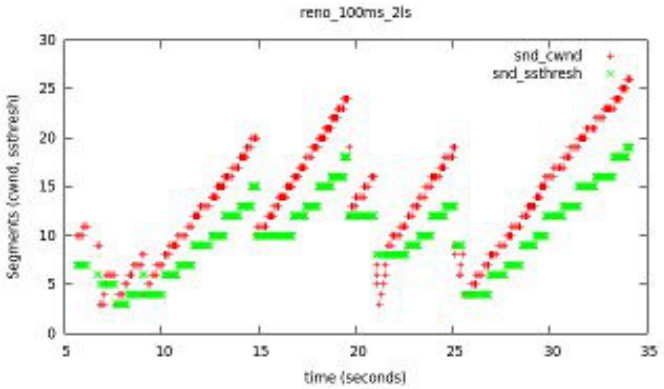

Figure 9, Reno 100ms 2% loss

D. 200ms delay 5% loss:

This network is an extreme case with high delay and high loss. The graphs in this case are really different because the older protocols can't really deal with this high delay and loss. Even cubic [Figure 10] gets barely any data through, but it does find a working threshold and window size very fast. In Renos [Figure 11] case we didn't even get a graph from this situation because tcp_probe didn't record anything so this graph is done with 3% loss. Venos graph can be seen at [Figure 12].

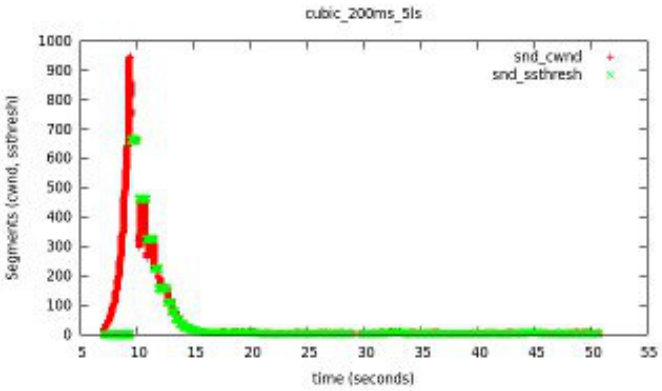

Figure 10, Cubic 200ms 5% loss

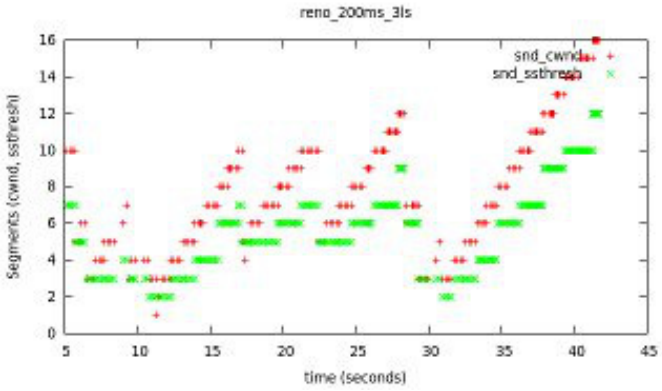

Figure 11, Reno 200ms 3% loss

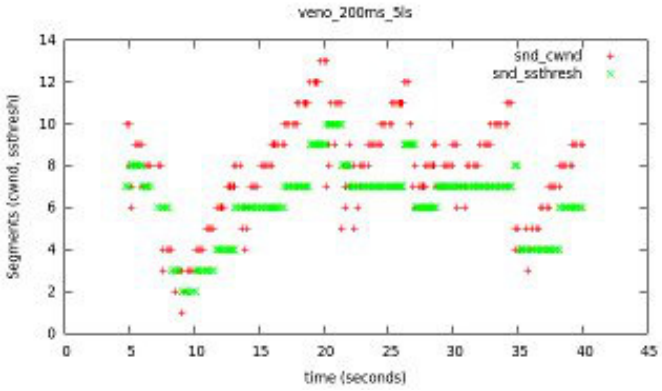

Figure 12, Veno 200ms 5% loss

VII. Conclusion

In the test we transmitted data for 40 seconds in every case and recorded the throughput in the following table[1].

	No loss, no delay	10ms, 2%loss	100ms, 2%loss	200ms, 5%loss
Cubic	94,6	10,9	1,77	0,43
Veno	94,4	14	1,84	0,48
Reno	94,5	9,5	1,21	0,26

Table 1, Throughput values

The values in this table might be quite off, because we used really high delay and loss values compared to the static 100 mbit/s network bandwidth we had. This results in a situation where none of the TCP versions reach even close to the maximum bandwidth of the network and none of the packets are lost because the network is reaching it's limits so the packet loss is just random. Also the buffer sizes won't grow and add delay so predicting congestion by increase in delay will not work for Veno.

With zero delay and zero loss the throughput was about 95 mbits/s in every case so the network was working fine in each case. About the congestion algorithms this network doesn't tell anything because there is no congestion.

In the 10 millisecond 2% loss network Veno got most data through with 14 mbit/s speed while Cubic got 10.9 mbit/s. This is because of the fact that when facing random packet loss Veno decreases the window size to 80% of current size while Cubic decreases it to 50%. As we can see from the transmission speeds Veno can deal quite well with random packet loss when the RTT isn't too long. Reno is the oldest of these versions and wasn't designed to deal with this much packet loss but it still was pretty close to the other two in transmission speed with 9.5 mbits/s.

The 100 millisecond delay and 2% loss network smooths up the difference between Cubic and Veno. This is due to the fact that Veno uses ACK-messages to increase the window size and Cubic uses real time and with 100 millisecond delay the window size of Veno increases really slow. Reno get's the slowest speed again because it can't deal with neither the high delay nor high packet loss.

The last network with 200 millisecond delay and 5% loss is a really extreme conditions especially the loss is really high. In this network all of the congestion algorithms really low speed and for Reno we couldn't even get tcp probe to record anything even though we tried several times. The highest loss value where tcp probe recorded anything for Reno was 3%. From this network we can only conclude that 5% packet loss is way too much for any congestion algorithm to handle.

VIII. References

[1] Wikipedia entry for Transmission Control Protocol(Online) http://en.wikipedia.org/wiki/Transmission_Control_Protocol
[2] V. Jacobson, Congestion avoidance and control, In Proceedings of ACM SIG-COMM '88, pages 314-329, August 1988.
[3] Wikipedia entry for TCP Cubic(Online) http://en.wikipedia.org/wiki/CUBIC_TCP
[4] TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks." C. P. Fu, S. C. Liew, IEEE Journal on Selected Areas in Communication, Feb. 2003
[5] "Analysis of TCP Vegas and TCP Reno" by Omar Ait Hellal, Eitan Altman- Telecommunication Systems 12-2000, Volume 15, Issue 3-4, pp 381-404