Network Programming

Comparison of TCP Variants

Project Report

Name	ID Number
Akhil Chilakala	2016A7PS0020H
S.R.Tejaswi	2016A7PS0088H
Bharadhwaj Jakkidi	2016A7PS0039H

Introduction

The TCP algorithms that were used to analyse the behaviour of various TCP algorithms are TCP cubic,TCP reno and TCP vegas.

TCP Cubic

In CUBIC, we try to find the balance between the congestion window size before windows reduction and after windows reduction. Despite this, although the real-time increase of the window enormously enhances the TCP friendliness of the protocol, in short RTT network, CUBIC's window growth is slower than TCP. So, in order to keep the growth-rate the same as TCP, CUBIC uses a new TCP mode to help change this situation. In CUBIC's TCP mode, it uses the same congestion control mechanisms as TCP while the RTT is short.

Advantages of CUBIC

CUBIC is a congestion control protocol for TCP (transmission control protocol) and the current default TCP algorithm in Linux. It is a high-speed variant of standard TCP. The protocol modifies the linear window growth function of existing TCP standards to be a cubic function in order to improve the scalability of TCP overfast and long-distance networks. It also achieves more equitable bandwidth allocations among flows with different RTTs (round trip times) by making the window growth to be independent of RTT – thus those flows grow their congestion window at the same rate.

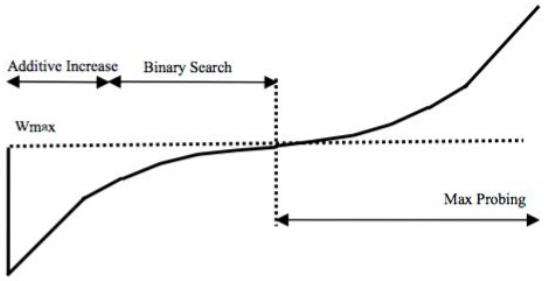


Fig. 1: The Window Growth Function of BIC

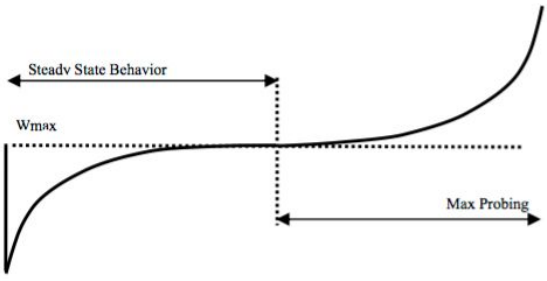


Fig. 2: The Window Growth Function of CUBIC

CUBIC TCP is a nice solution for BDP (Bandwidth Delay Product) network. With the development of Internet, the route trip times usually become very high (around 100 to 200ms). In this case, standard TCP has a low utilize radio. Nowadays many high-speed TCP variants came out and tried to fix this problem. But the difficulties are not only raising the efficiency, but also maintain the RTT fairness and TCP friendly. CUBIC works much better than those previous high-speed TCP variants.

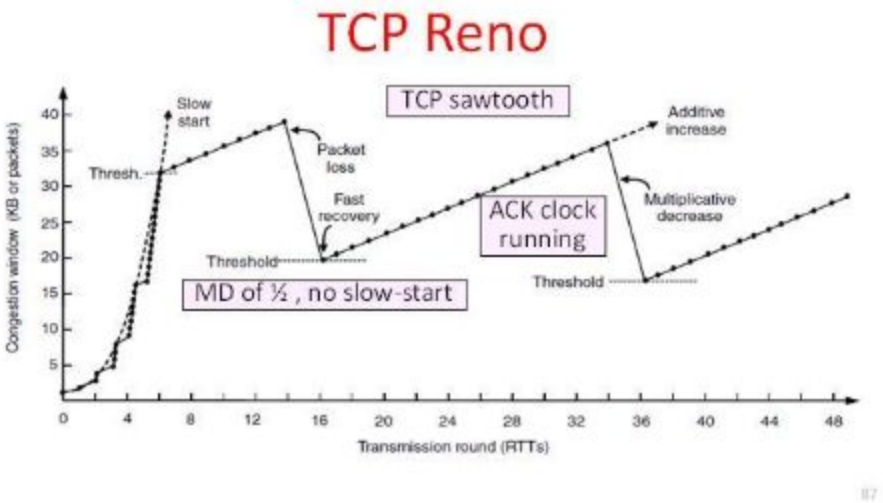
Limitations

Although CUBIC TCP seems a nice solution for high speed transport layer, there are some limitations of it. CUBIC TCP suffers from slow convergence yields poor network responsiveness, prolonged unfairness between flows, increases unfairness between long- and short-lived flows. CUBIC TCP is a good solution for standard Internet environment, but it is not aimed for special usage, such as satellite network or mobility usage.

TCP Reno

Reno retains the basic principle of Tahoe, such as slow starts and the coarse grain retransmit timer. However it adds some intelligence over it so that lost packets are detected earlier and the pipeline is not emptied every time a packet is lost. Reno requires that we receive immediate acknowledgement whenever a segment is received. Reno suggests an algorithm called 'Fast Retransmit'. Whenever we receive 3 duplicate ACK's we take it as a sign that the segment was lost, so we re-transmit the segment without waiting for timeout. Thus we manage to re-transmit the segment with the pipe almost full.

Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to 1. Since this empties the pipe. It enters into a algorithm which we call 'Fast-ReTransmit'.



Limitations

Reno performs very well when the packet losses are small. But when we have multiple packet losses in one window then RENO doesn't perform too well and it's performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet losses. If there is multiple packet drop then the first info about the packet loss comes when we receive the duplicate ACK's. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment reaches the sender after one

RTT. Also it is possible that the CWD is reduced twice for packet losses which occurred in one window.

Another problem is that if the widow is very small when the loss occurs then we would never receive enough duplicate acknowledgements for a fast retransmit and we would have to wait for a coarse grained timeout. Thus is cannot effectively detect multiple packet losses.

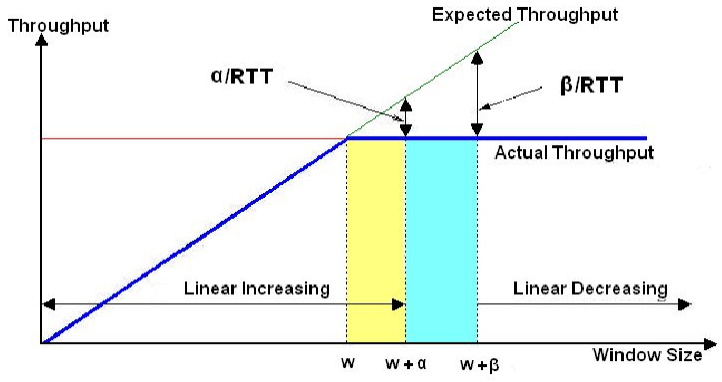
TCP Vegas

TCP Tahoe, Reno and New Reno detect and control congestion, after congestion occurs, but still there is a better way to overcome congestion problem i.e. TCP Vegas. TCP Vegas detects congestion without causing congestion.

Retransmission: TCP Vegas measures the RTT for every segment sent. Using the fine-grained RTT measurements, a timeout period for each segment is computed. When a duplicate acknowledgement (ACK) is received, TCP Vegas checks whether the timeout period has expired. If so, the segment is retransmitted . When a non-duplicate ACK that is the first or second after a fast retransmission is received, TCP Vegas again checks for the expiration of the timer and may retransmit another segment. In case of multiple segment loss and more than one fast retransmission, the congestion window is reduced only for the first fast retransmission

Congestion Avoidance: TCP Vegas does not continually increase the congestion window during congestion avoidance. Instead, it tries to detect incipient congestion by comparing the measured throughput to its notion of expected throughput.The congestion window is increased only if these two values are close, that is, if there is enough network capacity so that the expected throughput can actually be achieved. The congestion window is reduced if the measured throughput is considerably lower than the expected throughput; this condition is taken as a sign for incipient congestion.

Modified Slow-start: TCP Vegas differs from the other algorithms during it's slow-start phase. The reason for this modification is that when a connection first starts it has no idea of the available bandwidth and it is possible that during exponential increase it over shoots the bandwidth by a big amount and thus induces congestion. To this end Vegas increases exponentially only every other RTT, between that it calculates the actual sending through put to the expected and when the difference goes above a certain threshold it exits slow start and enters the congestion avoidance phase



Project Design

We have used 3 Laptops for the implementation of the project.

Laptop 1- A laptop that VMware and Ubuntu 18.04 LTS installed.

Laptop 2- A laptop with Ubuntu 18.04 LTS.

Laptop 3- A laptop with VMware and Kali 2019-1 installed.

The laptops with VMware have bridged connections with guest WiFi adapter and Network(AC) adapter.

Approach

The usual setup:

Laptop 1 is connected to laptop 2 via ‘x’ port using TCP. Laptop 2 is connected to laptop 3 via ‘y’ port using TCP or using asynchronous server. This is not an ideal setup. But in reality, router is in between the laptops.

Our approach:

A Local WAN (Wide Area Network) is created between the 3 laptops. Laptops 1 and 3 are connected to the WiFi hotspot of Laptop 2.

Test-bed Setup

1) Netem is used on WiFi adapter of laptop 2. As a result, the loss and drop are two times the actual values when both sending and receiving.

2) 'Ping' from laptop 1 to laptop 3 and also from laptop 3 to laptop 1 took approximately 10 milli-seconds on average.

3) Laptop 2 acts as a gateway for laptop 1 and laptop 2

4) Laptop 1 is 10.42.0.158

Laptop 2 is 10.42.0.154

Laptop 3 is 10.42.0.1

Implementation

The files "send.py" and "client.py" are modified for file sharing. 3 images of different sizes are chosen-0.19MB, 1.04 MB, 3.1 MB.

Routing goes from laptop 3 through laptop 2 to laptop 1.

All the proceedings are carried out with "FIREWALL turned OFF".

PART 1:

TCP variants- Initially in linux distributed system cubic is selected as default. But, when we check for the available congestion control algorithms present, we observe the presence of reno too. We chose vegas to be our third type of TCP to be used for analysis.

We used python as our programming language and bash scripting to change and utilise the TCP variants present and netem.

1)To view the TCP variants already present, we need to go to

cd/lib/modules/$(uname-r)/kernel/net/ipv4

*ls tcp_**

2)To select any of them execute

sudo modprobe tcp_vegas(.ko excluded)

3)To view TCP variants available for execution in python,

gedit /proc/sys/net/ipv4/tcp_available_congestion_control

4)To use them in python,

sock.setsockopt(IPPROTO TCP,TCP_CONGESTION,cong)

where 'cong' is binary ascii representation of tcp version used(for example "vegas")

5)This way we can use whichever TCP version we want to use.

6)After connecting laptop 1 and laptop 3 to the hotspot of laptop 2, we get IP address of both the laptops with gateway being the laptop 2.

7)We modified single threaded and client versions of python to send and receive files instead of text messages.

8)Test_bed setup is established.

9) Now the file from laptop 3 to laptop 1 can be sent using different TCP versions by emulating the network in laptop 2's WiFi adapter. i.e. since the route from laptop 3 to laptop 1 goes through laptop 2's WiFi adapter, using netem,we can implement various bandwidth throttling ,delay and drop rates.

10)Using netem:

sudo tc qdisc add dev wlp2s0 root netem loss 10%

This introduces 10% loss of data from both laptop 3 to laptop 2 and laptop 2 to laptop 1. Therefore 20% loss is introduced.

sudo tc qdisc add dev wlp2s0 root netem delay 10ms

This introduces 10 milliseconds loss of data from both laptop 3 to laptop 2 and laptop 2 to laptop Therefore 20 milliseconds loss is introduced.

sudo tc qdisc del dev wlp2s0 root

This deletes all the rules added.

To introduce both delay and loss use,

sudo tc qdisc add dev wlp2s0 root netem delay 10ms loss 10%

To introduce bandwidth throttling use,

sudo tc qdisc add dev wlp2s0 root handle 1:tbf rate 1mbit burst 32kbit latency 400ms

This introduces throttling to average of 1 Mbit per second and only 32 kbit packets are allowed for a single burst. Every packet that waits more than 400 ms is dropped.

11)To add network delay or loss to bandwidth throttling,we need to add

sudo tc qdisc add dev wlp2s0 root handle 1:tbf rate 1mbit burst 32kbit latency 400ms

sudo tc qdisc add dev wlp2s0 parent 1:1 handle 10:netem loss 0.2% delay 20ms

12) In the previously described way, we ran emulations using 3 file sizes(0.19MB, 1.04MB,3,1 MB) , TCP(cubic,reno,vegas) , delay (0,100,200,400ms), drop rates (0,0.2%,0.4%) and bandwidth(no throttling ,throttling to 1Mbit). Using these we made 216 observations.

Key results

HIGH BANDWIDTH

No drop no delay : Reno is the best. This is because when there is no drop and no delay, the more aggressive the tcp the better it is. Since Reno is the most aggressive, it is the best among the 3 algorithms. Vegas takes the highest time because it has very slow start

Drop 0.2 No delay: Since drop is 0.2 which is very small, it doesn't matter much. For small files, no difference is observed. For considerably large files, Vegas gives the worst results because vegas changes its window size even before congestion happens based on the delay of the previous packet. But since drop is very less vegas throttles its window size unnecessarily.

Drop 0.4 no delay: Similar observations made as in ‘drop 0.2 no delay’.

Delay 100 no drop: For large files, Cubic is the best. Vegas is almost similar to cubic in this case. But when files are small, vegas runs the fastest.

Delay 100 drop 0.2: Similar observations as in ‘delay 100 no drop’.

Delay 100 drop 0.4: When there is significant drop and delay, cubic>vegas>reno

Delay 200 drop 0: Similar observations as in ‘delay 100 drop 0.4’ There is significant delay in this case too.

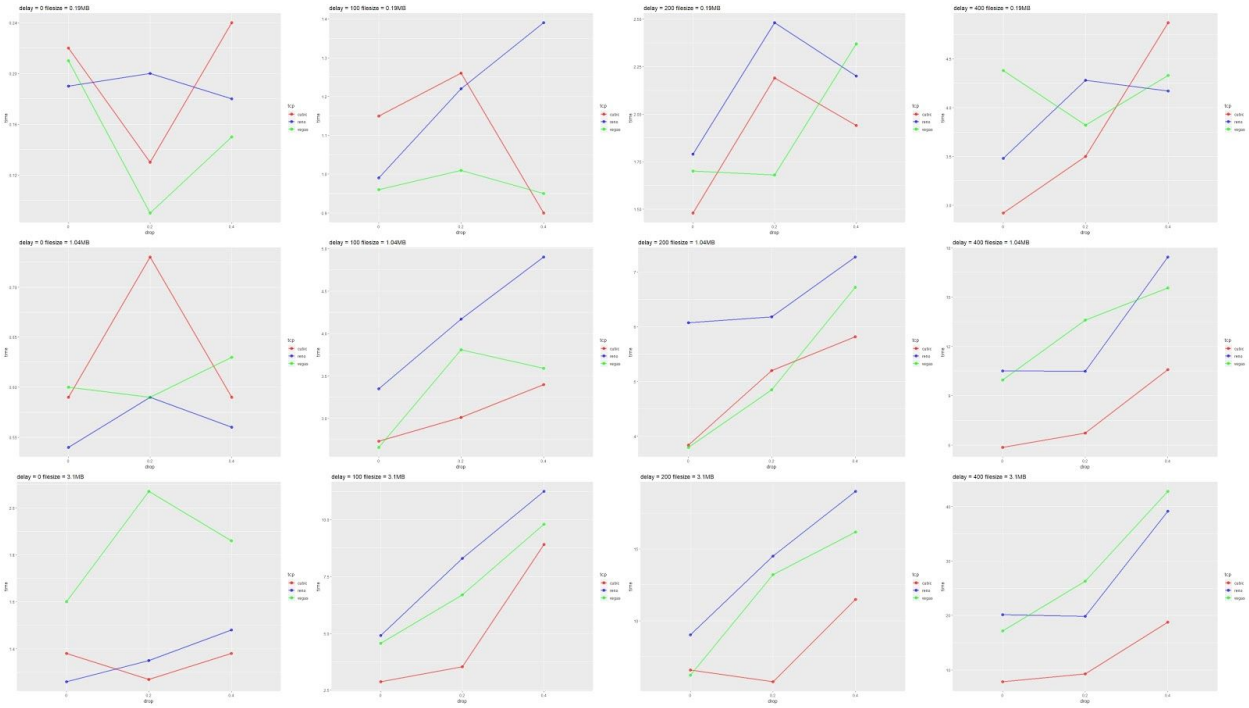
Delay 200 drop 0.2: Reno runs the slowest. Vegas and cubic are almost similar. Vegas outperforms cubic when small files are used.

Delay 200 drop 0.4: When significant drop and delay is present, cubic>vegas>reno. But for small files,we observed that reno is slightly better than vegas.

Delay 400 drop 0: When significant drop and delay is present, cubic>vegas>reno. But we observed that reno is slightly better than vegas for small files due to significant delay

Delay 400 drop 0.2: Cubic is the best. Vegas is better when files are small. Reno outperforms vegas when file size is high because of its aggressive start..

Delay 400 drop 0.4: For large files, cubic is the best. Vegas and reno are close to each other. When files are small, both reno and vegas outperform cubic



LOW BANDWIDTH

No delay no drop: Cubic takes the longest time while Vegas runs the fastest. This could be because cubic is made for high bandwidth uses mainly. While reno and vegas are used for low bandwidth cases. Reno and vegas act almost similarly.

Delay 0 drop 0.2 : Similar observations as in ‘no delay and no drop’.

Delay 0 drop 0.4 : Same as in ‘Delay 0 drop 0.2’ case. Although bandwidth is low, cubic is almost closer to other two, when files are large enough.

Delay 100 drop 0: Cubic takes the longest time while Vegas runs the fastest. This could be because cubic is made for high bandwidth uses mainly. While reno and vegas are used for low bandwidth cases. Reno and vegas act almost similarly

Delay 100 drop 0.2: Vegas is the best when small files are used. When file size increases, there is not much difference among the three algorithms.

Delay 100 drop 0.4: Similar observations as in ‘Delay 100 drop 0.2’

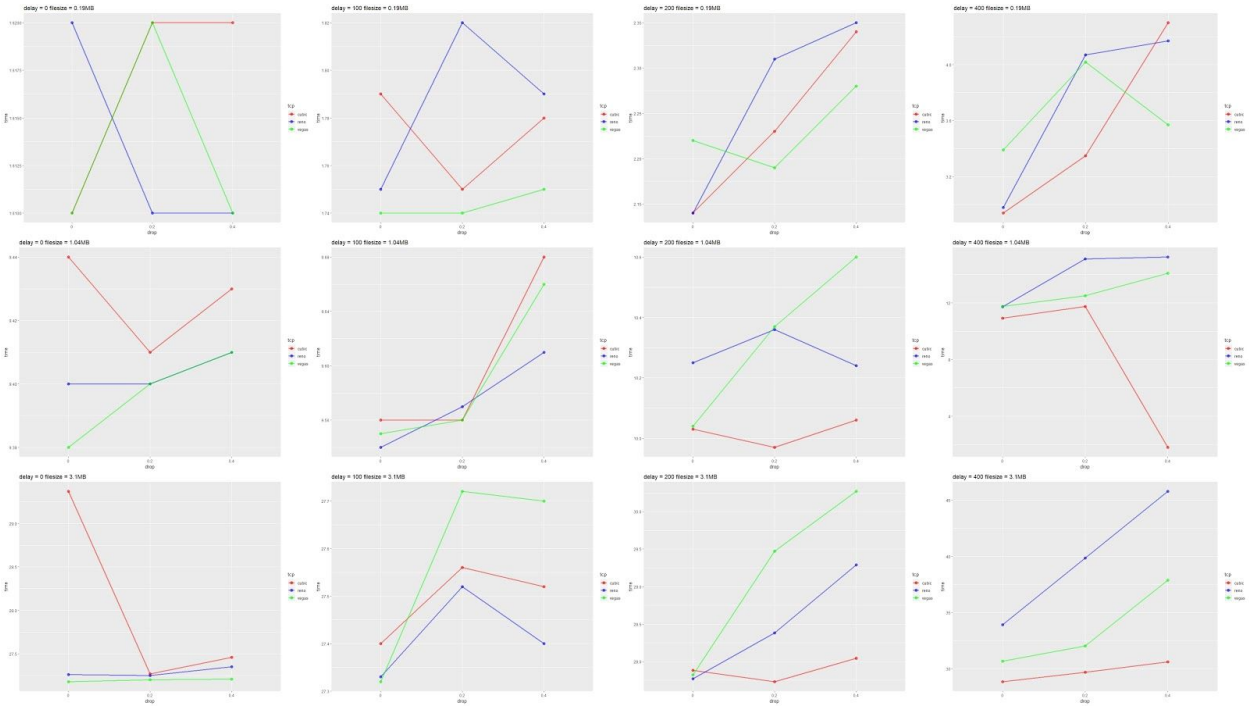
Delay 200 drop 0: Cubic is the best. Vegas is very close to cubic when file sizes are large. From this scenario onwards, the congestion occurs because of either very significant delay or very significant drop. So the better the congestion algorithm it follows,the less time it takes. From now onwards cubic outperforms the other two because, although less bandwidth is used here, the congestion created is very significant.

Delay 200 drop 0.2: Cubic runs the fastest. Reno is almost close to cubic when file sizes are large.

Delay 200 drop 0.4: Vegas is the best when small files are used. But the other two are also very close. For large files, we observe cubic>reno>vegas. The reason behind reno>vegas is

aggressiveness of reno. Since reno is aggressive, it is able to send more packets than vegas before congestion happens. But once congestion happens, reno and vegas almost start being linear wrt congestion in this case.

Delay 400 for all drops: Since this is a very significant delay and bandwidth is low, we observe significant congestion. So in this situation, the algorithm that has better congestion handling, takes the least time. Therefore cubic>vegas>reno. Except when file is very small and drop is 0.4% vegas outperforms the other two, because when files are small and significant congestion is available the other two algorithms expect the congestion to take place before they start handling. Vegas,on the other hand, starts changing its window size before congestion happens. hence when congestion is too much for small file sizes, vegas is better.



CONCLUSION

High Bandwidth:

Similar patterns are observed when drop is decreased by 100% and delay is increased by 100%. When bandwidth is unlimited , cubic is better in almost all cases. However vegas is handy when file sizes are small since vegas handles its window size even before congestion happens and since when the files are small the congestion doesn't occur many times and vegas adjusts itself for those less number of cases.

Low bandwidth:

For lower bandwidth, cubic is not used when there is no congestion, since it is less aggressive in nature. The other two might outperform cubic when congestion is less.

But when congestion is significant and files are pretty large, then, only the congestion control handling of the algorithm used matters. In those cases, we observe cubic>vegas>reno.

Final inferences:

Whenever there is a significant congestion, we observe cubic $>^{[1]}$ vegas $>^{[2]}$ reno.

References:[1]“The performance comparison between TCP Reno and TCP Vegas, Yuan-Cheng lai;Chang Li Yao,IEEE”,

[2]”Performance Analysis of TCP Congestion Control Algorithms ,Habibullah Jamal, Kiran Sultan”

When bandwidth is low and congestion is less, cubic is inefficient. (because cubic is made for high bandwidth situations)

When files are small, vegas can be trusted in any case.(but vegas however starts changing its window size before congestion happens. hence when congestion is too much for small file sizes, vegas is better.)