Tetcos Engineering

TETCOS

Transmission Control Protocol (TCP)

Comparison of TCP Congestion Control Algorithms using NetSim™

Table of Contents

1. Abstract

In this paper we discuss the different congestion control algorithms of the TCP variants of Old Tahoe, Tahoe, Reno, New Reno, BIC and CUBIC. All the algorithms suggest mechanisms for determining when to retransmit a packet and how it should update the congestion window. In our analysis we create a network scenario in NetSim and compare the congestion window behaviour of all the six variants of TCP.

2. Introduction to TCP

TCP is a transport layer protocol, part of the TCP/IP suite which defines how to establish and maintain a connection in a network. It is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which supports a variety network applications.

TCP's design philosophy has evolved considerably from where the goal was to develop an effective packet switching protocol to a protocol which is fair, robust and reliable. As the internet traffic has increased substantially over the past few decades it was important for TCP to be fair and take into consideration the congestion in a network.

2.1 Problem of Congestion

Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet and it subsequently leads to performance degradation. Congestion in a network may occur when the load on the network (i.e. the number of packets sent to the network) is greater than the capacity of the network.

Congestion Control refers to techniques that can either prevent congestion, before it happens, or remove congestion after it has happened. TCP does this by maintaining a 'congestion window', which indicates the maximum amount of data that can be sent into the network without being acknowledged. Different variants of TCP use a certain combinations of algorithms which update the congestion window by sensing congestion in the network and thereby preventing it.

3. TCP Variants

The six variants of TCP and the congestion control algorithms used by them are mentioned below.

3.1 Old Tahoe

Old Tahoe is one of the earliest variants of TCP. It implements two algorithms called slow start and congestion avoidance to update the congestion window.

3.1.1 Slow Start

At the start of data transmission the size of congestion window is one. This means TCP can send only one packet until it receives an acknowledgement. When the ACK is received by the sender the congestion window increases to two. Now the sender can send two data packets. Upon the arrival of every new ACK the sender increases its congestion window by one. This phase is known as the slow start phase where the congestion window increases exponentially.

So on the arrival of a new ACK, $cwnd += MSS$;

3.1.2 Congestion Avoidance

TCP will continue the slow start phase until it reaches a certain threshold, or if packet loss occurs. Now it enters in to a phase called congestion avoidance. Here the congestion window grows linearly. This means that the congestion window increases from 'n' to 'n+1' only when it has received 'n' new

ACKs. The rate of growth of congestion window slows down because this is the stage where TCP is susceptible to packet loss. The formula used here is $cwnd += (SMSS * SMSS)/cwnd$.

3.1.3 Timer Expiry Event

TCP maintains a timer for every data packet that it has sent into the network. If it doesn't receive an ACK before the timer expires, it assumes that a packet loss has occurred. TCP does the following when the timer expires

- Threshold value is reduced to half of the congestion window or twice the maximum segment size, whichever is the maximum. $ssthresh = max\ (cwnd\ /\ 2, 2 * SMSS)$.
- The packet which is assumed to be lost is retransmitted.

- Congestion window is reduced to one.
- Slow start is resumed.

3.2 Tahoe

TCP Tahoe implements all the above mentioned algorithms used by Old Tahoe. The Fast Retransmit algorithm was included in Tahoe to improve the response time of TCP.

3.2.1 Fast Retransmit

One of the major drawbacks of Old Tahoe is that it depends on the timer to expire before it can retransmit a packet. TCP Tahoe tries to improve upon Old Tahoe by implementing the fast retransmit algorithm. Fast Retransmit takes advantage of the fact that duplicate ACKs can be an indication that a packet loss has occurred. So, whenever it receives 3 duplicate ACKs it assumes that a packet loss has occurred and retransmits the packet.

3.3 Reno

TCP Reno retains the basic principles of Tahoe such as Slow Start, Congestion Avoidance and Fast Retransmit. However it is not as aggressive as Tahoe in the reduction of the congestion window. Reno implements the Fast Recovery algorithm which is described below.

3.3.1 Fast Recovery

The congestion window drop to one on the arrival of a 3 duplicate ACK can be considered as an extreme precaution. Arrival of 3 duplicate ACKs corresponds to light congestion in the network and there is no need for the congestion window to drop down drastically. The Fast Recovery algorithm does the following on the arrival of a third duplicate ACK:

- The threshold value is set to half of the congestion window. $ssthresh = cwnd/2$.
- The congestion window is now set to be threshold plus three times the MSS. $cwnd = ssthresh + 3 * SMSS$.
- On the arrival of another duplicate ACK the congestion window increases by one MSS. This is done because an ACK signifies that a segment if out of the network and the sender can pump in another packet into the network. This is somewhat similar to slow start. $cwnd += SMSS$.

- TCP remains in fast recovery phase until it receives a higher ACK from the receiver.
- On receiving a higher ACK the congestion window is set to the threshold value. From now onwards congestion avoidance is followed. $cwnd = ssthresh$.

3.4 New Reno

Reno doesn't improve much upon Tahoe if there are multiple packet losses in the same window. This is because when multiple packet losses occur Reno enters fast recovery multiple times which decreases the congestion window by half every time. In practical scenarios where multiple packet loss in the same window is common, Reno doesn't increase the throughput significantly. A modified version of fast recovery algorithm was suggested to overcome this.

3.4.1 New Reno's Modification to Fast Recovery

In the modified version of Fast Recovery, TCP stores the sequence number of the highest data packet which is sent when the third duplicate ACK arrives in a variable called 'recover'. New Reno doesn't exit fast recovery unless it has received an ACK which is higher than the recover packet.

If a new ACK arrives which has an ACK number lower than the recover packet, it immediately retransmits that packet. This ACK is known as a partial ACK. When a partial ACK arrives the congestion window is reduced by the amount of 'new data' that has been acknowledged by the partial ACK (i.e. the amount of data that has been acknowledged after the retransmitted packet). To this one MSS is added. In mathematical terms. $cwnd = cwnd - new\ data + SMSS$.

TCP stays in fast recovery mode until it receives a full ACK i.e. an ACK with a higher sequence number than recover. Note that if there aren't any multiple packet losses in the same window then New Reno essentially behaves like Reno.

3.5 BIC

In BIC congestion control is viewed as a searching problem in which the system can give yes/no feedback through packet loss as to whether the current sending rate (or window) is larger than the network capacity. The current minimum window can be estimated as the window size at which the flow does not see any packet loss. If the maximum window size is known, we can apply a binary search technique to set the target window size to the midpoint of the maximum and minimum. As increasing to the target, if it gives any packet loss, the current window can be treated as a new maximum and the reduced window size after the packet loss can be the new minimum. The midpoint between these new values becomes a new target. Since the network incurs loss around the new maximum but did not do so around the new minimum, the target window size must be in the middle of the two values. After reaching the target and if it gives no packet loss, then the current window size becomes a new minimum, and a new target is calculated. This process is repeated with the updated minimum and maximum until the difference between the maximum and the minimum falls below a preset threshold, called the minimum increment (S_{min}). This technique is called binary search increase.

3.5.1 Additive Increase

In order to ensure faster convergence and RTT-fairness, binary search increase is combined with an additive increase strategy. When the distance to the midpoint from the current minimum is too large, increasing the window size directly to that midpoint might add too much stress to the network. When the distance from the current window size to the target in binary search increase is larger than a prescribed maximum step, called the maximum increment (S_{max}) instead of increasing window directly to that midpoint in the next RTT, we increase it by S_{max} until the distance becomes less than S_{max}, at which time window increases directly to the target. Thus, after a large window reduction, the strategy initially increases the window linearly, and then increases logarithmically. This combination of binary search increase and additive increase is called as binary increase. Combined with a multiplicative decrease strategy, binary increase becomes close to pure additive increase under large windows. This is because a larger window results in a larger reduction by multiplicative decrease and therefore, a longer additive increase period. When the window size is small, it becomes close to pure binary search increase – a shorter additive increase period.

3.5.2 Slow Start

After the window grows past the current maximum, the maximum is unknown. At this time, binary search sets its maximum to be a default maximum (a large constant) and the current window size to be the minimum. So the target midpoint can be very far. According to binary increase, if the target midpoint is very large, it increases linearly by the maximum increment. Instead, run a “slow start” strategy to probe for a new maximum up to S_{max}. So if cwnd is the current window and the maximum increment is S_{max}, then it increases in each RTT round in steps cwnd+1, cwnd+2, cwnd+4,…, cwnd+S_{max}. The rationale is that since it is likely to be at the saturation point and also the maximum is unknown, it probes for available bandwidth in a “slow start” until it is safe to increase the window by S_{max}. After slow start, it switches to binary increase.

3.5.3 Fast Convergence

It can be shown that under a completely synchronized loss model, binary search increase combined with multiplicative decrease converges to a fair share. Suppose there are two flows with different window sizes, but with the same RTT. Since the larger window reduces more in multiplicative decrease (with a fixed factor β), the time to reach the target is longer for a larger window. However, its convergence time can be very long. In binary search increase, it takes $log(d) - log(Smin)$ RTT rounds to reach the maximum window after a window reduction of d. Since the window increases in a log step, the larger window and smaller window can reach back to their respective maxima very fast almost at the same time (although the smaller window flow gets to its maximum slightly faster). Thus, the smaller window flow ends up taking away only a small amount of bandwidth from the larger flow before the next window reduction. To remedy this behaviour, binary search increase is modified as follows. After a window reduction, new maximum and minimum are set. Suppose these values are max_win_i and min_win_i for flow i (i =1, 2). If the new maximum is less than the previous, this window is in a downward trend. Then, readjust the new maximum to be the same as the new target window (i.e. $max\_win_i = (max\_win_i - min\_win_i)/2$), and then readjust the target. After that apply the normal binary increase. This strategy is called fast convergence.

3.6 CUBIC

In CUBIC, the window growth function is a cubic function, whose shape is very similar to the growth function of BIC. CUBIC is designed to simplify and enhance the window control of BIC. CUBIC uses a cubic function of the elapsed time from the last congestion event. While most alternative algorithms to Standard TCP uses a convex increase function where after a loss event, the window increment is always increasing, CUBIC uses both the concave and convex profiles of a cubic function for window increase. After a window reduction following a loss event, it registers W_{max} to be the window size where the loss event occurred and performs a multiplicative decrease of congestion window by a factor of β where β is a window decrease constant and the regular fast recovery and retransmit of TCP. After it enters into congestion avoidance from fast recovery, it starts to increase the window using the concave profile of the cubic function. The cubic function is set to have its plateau at W_{max} so the concave growth continues until the window size becomes W_{max}. After that, the cubic function turns into a convex profile and the convex window growth begins. This style of window adjustment (concave and then convex) improves protocol and network stability while maintaining high network utilization. This is because the window size remains almost constant, forming a plateau around W_{max} where network utilization is deemed highest and under steady state, most window size samples of CUBIC are close to W_{max}, thus promoting high network utilization and protocol stability. Note that protocols with convex growth functions tend to have the largest window increment around the saturation point, introducing a large burst of packet losses.

The window growth function of CUBIC uses the following function:

$$W\ (t)\ =\ C(t\ -\ K)3\ +\ Wmax \qquad \ldots\ldots.(1)$$

where C is a CUBIC parameter, t is the elapsed time from the last window reduction, and K is the time period that the above function takes to increase W to W_{max} when there is no further loss event and is calculated by using the following equation:

$$K\ =\ r3\ Wmax\ C\ \beta \qquad \ldots\ldots..(2)$$

Upon receiving an ACK during congestion avoidance, CUBIC computes the window growth rate during the next RTT period using Eq. (1). It sets W (t + RTT) as the candidate target value of congestion window. Suppose that the current window size is cwnd. Depending on the value of cwnd. CUBIC runs in three different modes. First, if cwnd is less than the window size that (standard) TCP would reach at time t after the last loss event, then CUBIC is in the TCP mode. Otherwise, if cwnd is less than W_{max}, then CUBIC is in the concave region, and if cwnd is larger than W_{max}, CUBIC is in the convex region.

3.6.1 TCP-friendly region

When receiving an ACK in congestion avoidance, first check whether the protocol is in the TCP region or not. This can be analysed by the window size of TCP in terms of the elapsed time t. Then find the average window size of additive increase and multiplicative decrease (AIMD) with an additive factor α and a multiplicative factor β to be the following function:

$$\frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{2-\beta}{\beta}\frac{1}{p}} \qquad \ldots\ldots\ldots\ (3)$$

By the same analysis, the average window size of TCP with α = 1 and β = 0.5 is $\frac{1}{RTT}\sqrt{\frac{3}{2}\frac{1}{p}}$. Thus, for Eq. 3 to be the same as that of TCP, α must be equal to 3β/2-β. If TCP increases its window by α per RTT, then window size of TCP in terms of the elapsed time t as follows:

$$Wtcp(t) = Wmax(1 - \beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT} \quad \text{.........} \; (4)$$

If cwnd is less than $W_{tcp(t)}$, then the protocol is in the TCP mode and cwnd is set to $W_{tcp(t)}$ at each reception of ACK.

3.6.2 Concave region

When receiving an ACK in congestion avoidance, if the protocol is not in the TCP mode and cwnd is less than W_{max} then the protocol is in the concave region. In this region, cwnd is incremented by $\frac{W(t+RTT) - cwnd}{cwnd}$

3.6.3 Convex region

When the window size of CUBIC is larger than W_{max}, it passes the plateau of the cubic function after which CUBIC follows the convex profile of the cubic function. Since cwnd is larger than the previous saturation point W_{max}, this indicates that the network conditions might have been perturbed since the last loss event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, fluctuations in available bandwidth always exist. The convex profile ensures that the window increases very slowly at the beginning and gradually increases its growth rate. This phase is called as the maximum probing phase since CUBIC is searching for a new W_{max}. Since the window increase function for the convex region is not modified, the window growth function for both regions remains unchanged. To be exact, if the protocol is the convex region outside the TCP mode, cwnd is incremented by $\frac{W(t+RTT) - cwnd}{cwnd}$

3.6.4 Multiplicative decrease

When a packet loss occurs, CUBIC reduces its window size by a factor of β (set β = 0.2). A side effect of setting β to a smaller value than 0.5 is slower convergence. The more adaptive setting of β leads to faster convergence, it will make the analysis of the protocol much harder and also affects the stability of the protocol. This adaptive adjustment of β is a future research issue.

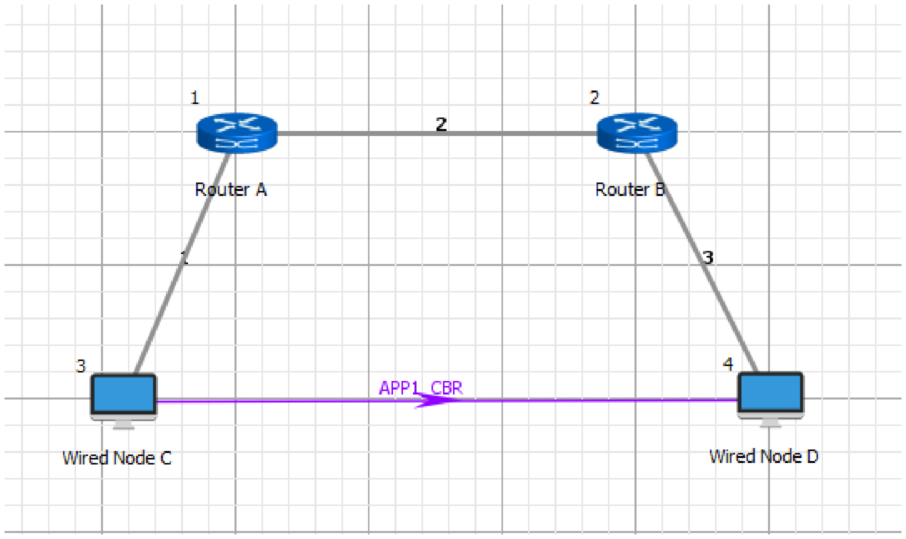
3.6.5 Fast Convergence

When a new flow joins the network, existing flows in the network need to give up their bandwidth shares to allow the new flow some room for growth. To increase this release of bandwidth by existing flows, the following mechanism called fast convergence is added. With fast convergence, when a loss event occurs, before a window reduction of the congestion window, the protocol remembers the last value of W_{max} before it updates W_{max} for the current loss event. Let us call the last value of W_{max} to be $W_{last-max}$. At a loss event, if the current value of W_{max} is less than the last value of it, $W_{last-max}$, this indicates that the saturation point experienced by this flow is getting reduced because of the change in available bandwidth. This allows the flow to release more bandwidth by reducing W_{max} further. This action effectively lengthens the time for this flow to increase its window because the reduced W_{max} forces the flow to have the plateau earlier. This allows more time for the new flow to catch up its window size.

4. Analysis

We have used NetSim to simulate the network scenario. The scenario consists of two nodes connected by two routers. Wired node C is the sender and wired node D is the receiver. Packets of size 1460 bytes are generated by the application layer of the sender every 400 micro seconds of the simulation. This amounts to 27.84 Mega Bits of data generated per second. This is higher than the node to router link which is set to 20 Mega Bits per second. This simulates an infinite backlog of traffic which means that TCP never runs out of data to send.

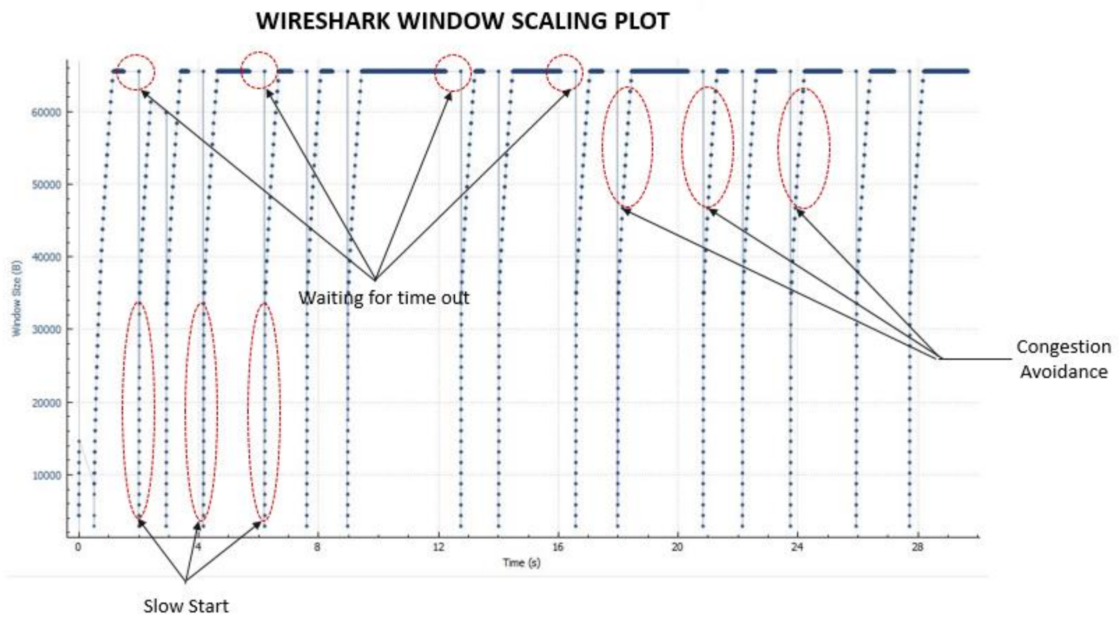
All simulations were run using NetSim version 10.0



Scenario Properties for Old Tahoe and Tahoe

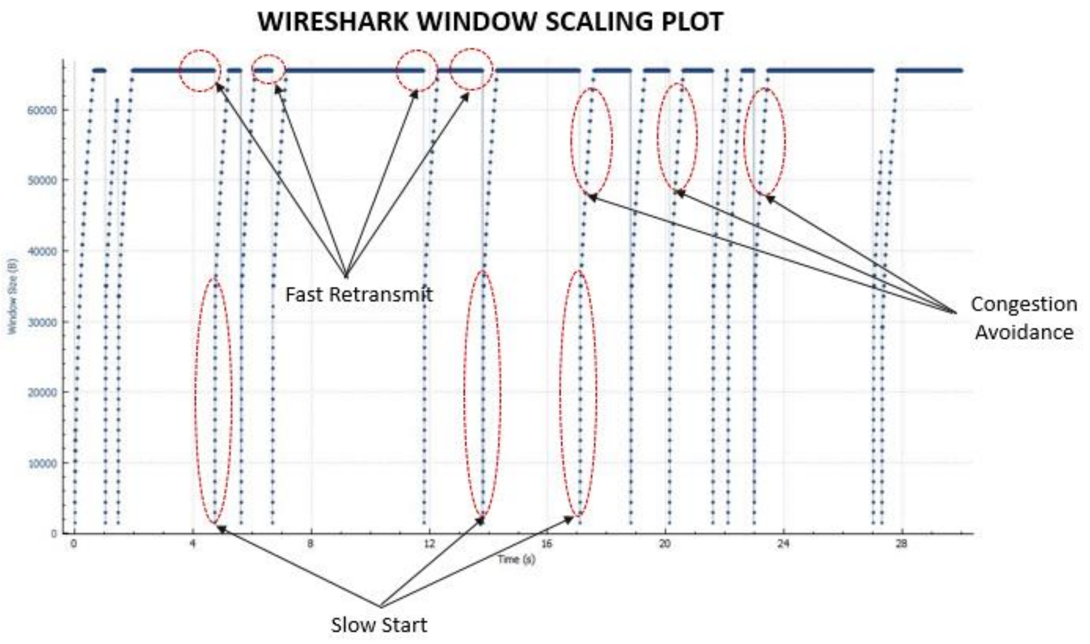
Maximum Segment Size (MSS)	1460 bytes
Node – Router propagation delay	5 µs
Router – Router propagation delay	100 µs
Node – Router link speed (both)	20 Mbps
Router – Router link speed	100 Mbps
Bit Error Rate (All links)	10 e(-8)
Packet generation Rate	Infinite Backlog

4.1 Old Tahoe



The graph shown above is a plot of Congestion window vs. Time of Old Tahoe for the scenario shown above. Each point on the graph represents the congestion window at the time when the packet is sent. You can observe that after every congestion window drop (caused by packet loss or timer expiry) Old Tahoe enters slow start, thereby increasing its window exponentially and then enters congestion avoidance where it increases the congestion window linearly.

4.2 Tahoe

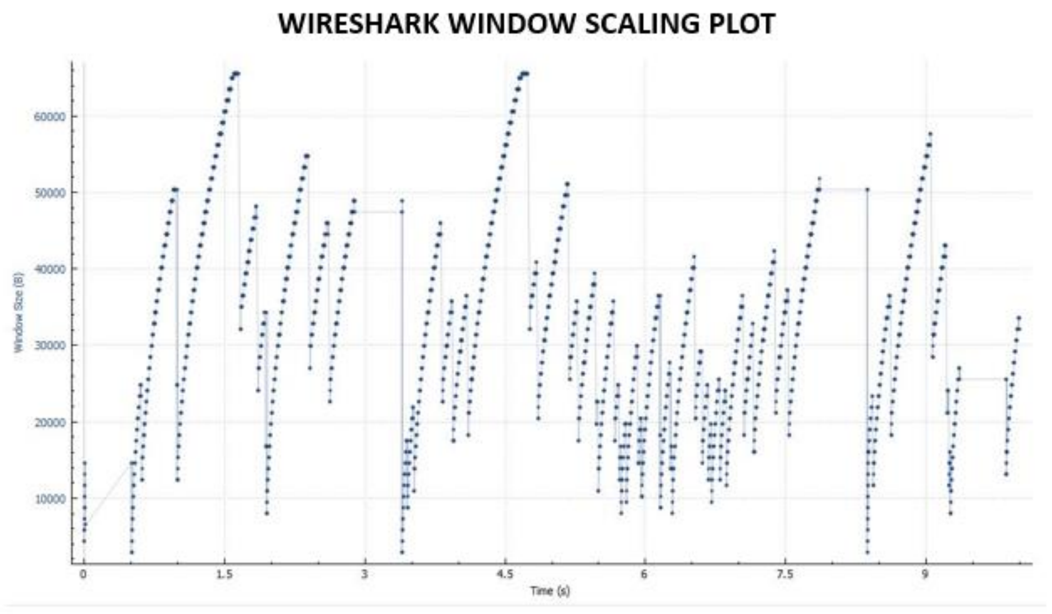


Tahoe uses the fast retransmit algorithm with which it responds to packet errors faster than Old Tahoe. Comparing the graphs of Old Tahoe and Tahoe one can observe that the latter drops the congestion window and retransmits faster than Old Tahoe (when three duplicate ack's are received).

Scenario Properties for Reno and New Reno

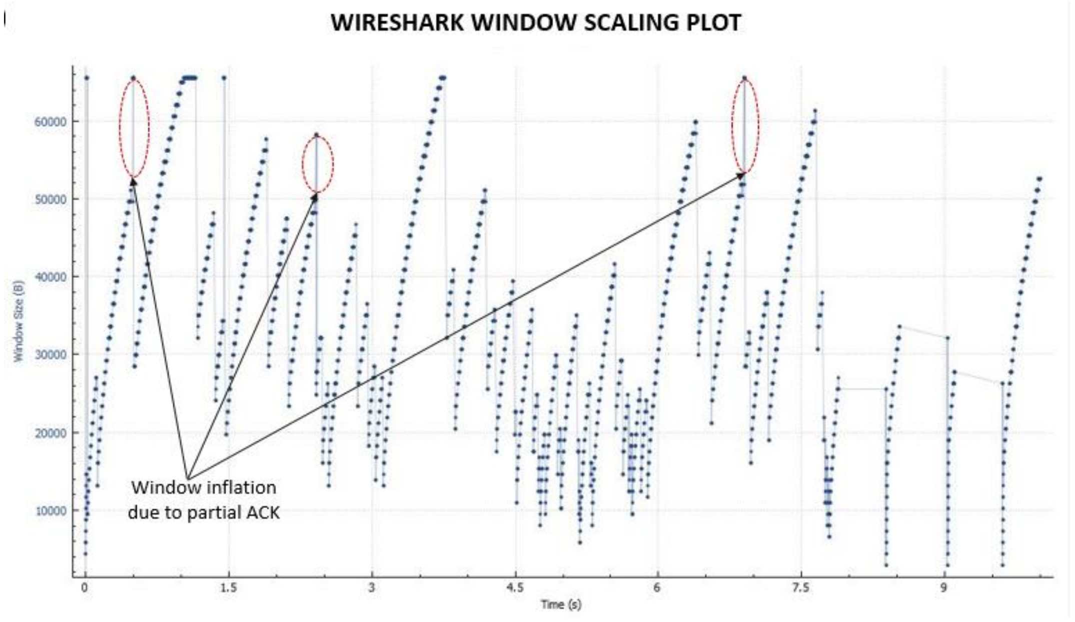
Maximum Segment Size (MSS)	1460 bytes
Node – Router propagation delay	5 µs
Router – Router propagation delay	100 µs
Node – Router link speed (both)	20 Mbps
Router – Router link speed	100 Mbps
Bit Error Rate (All links)	10 e(-7)
Packet generation Rate	Infinite Backlog

4.3 Reno



TCP Reno upon receiving the third duplicate ACK sets its congestion window according to the formula *cwnd = cwnd/2 + 3*MSS*. On further arrival of duplicate ACKs it increases its congestion window by one MSS. If it receives a higher ACK then it drops the congestion window to the new threshold value. This mechanism is known as fast recovery.

4.4 New Reno

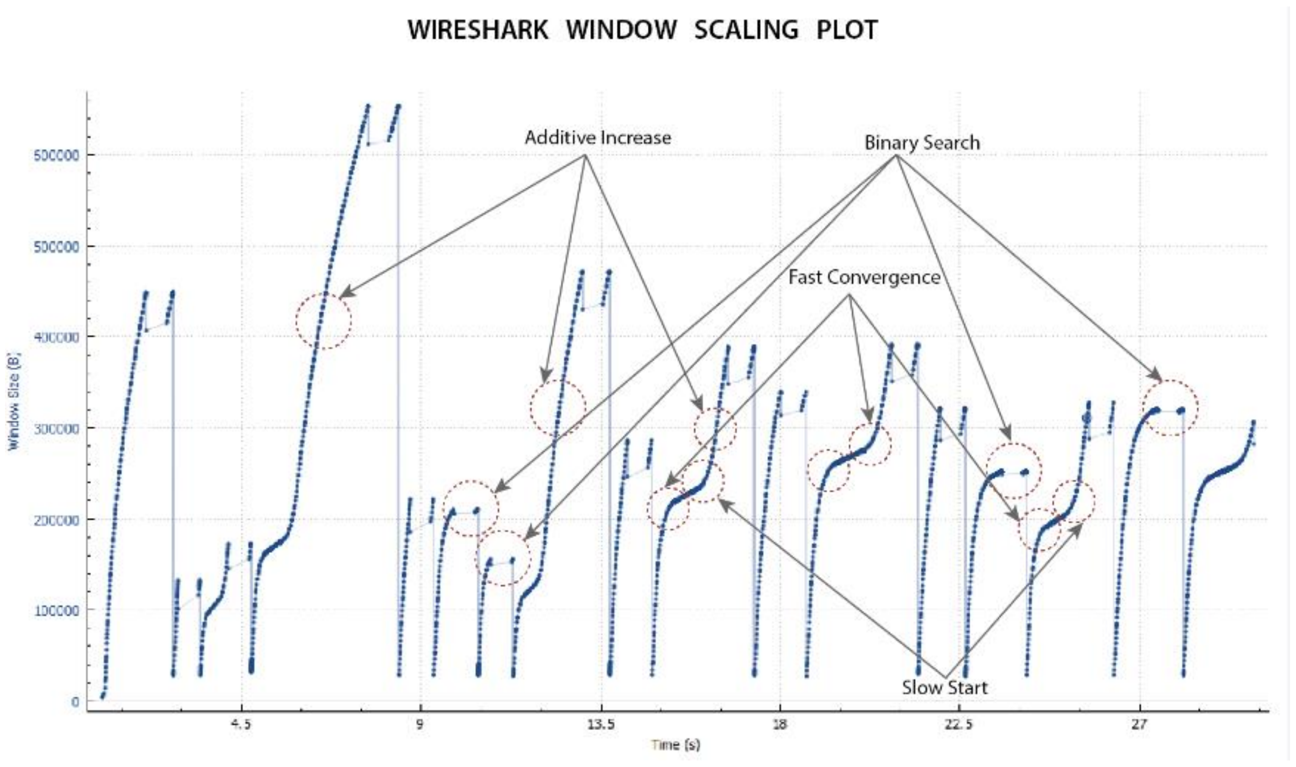


In the New Reno's modification to fast recovery, TCP doesn't exit the fast recovery phase until it receives a full ACK (the ACK for the segment which was sent just before the arrival of the third duplicate ACK). If it receives a duplicate ACK in the fast recovery phase it retransmits the segment corresponding to the duplicate ACK. It then increases the congestion window by one MSS. Hence we see a sharper rise in the congestion window size as New Reno doesn't exit the fast recovery phase until it receives the full ACK.

Scenario Properties for BIC and CUBIC

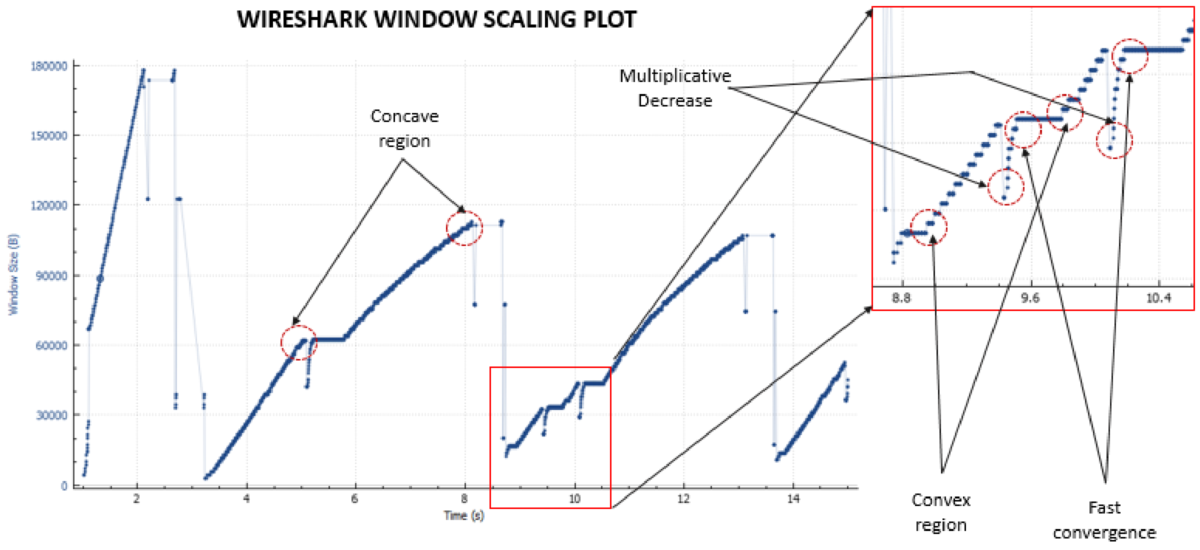
Maximum Segment Size (MSS)	1460 bytes
Node – Router propagation delay	5 µs
Router – Router propagation delay	10000 µs
Node – Router link speed (both)	20 Mbps
Router – Router link speed	100 Mbps
Bit Error Rate (All links)	10 e(-8)
Packet generation Rate	Infinite Backlog

4.5 BIC



The graph shown above is a plot of Congestion window vs. Time of BIC for the scenario shown above. Each point on the graph represents the congestion window at the time when the packet is sent. You can observe Binary Search, Additive Increase, Fast Convergence, Slow Start phases in the above graph.

4.6 CUBIC



The graph shown above is a plot of Congestion window vs. Time of CUBIC for the scenario shown above. You can observe Multiplicative decrease, Fast Convergence, Concave and Convex regions in the above graph.

5. References

1. Information Sciences Institute, “Transmission Control Protocol” RFC 793, September 1981
2. W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,” RFC2001, January 1997.
3. M. Allman, V. Paxson, W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999.
4. S. Floyd, T. Henderson “The New Reno modification to TCP’s Fast Recovery Algorithm”, RFC 2582, April 1999.
5. Lisong Xu, Khaled Harfoush, and Injong Rhee, “Binary Increase Congestion Control for Fast, Long Distance Networks” IEEE INFOCOM. 2004, Vol 4: pp.2514-2524
6. Sangtae Ha, Injong Rhee, Lisong Xu , “CUBIC: A New TCP-Friendly High-Speed TCP Variant”