A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas

Abstract:

The purpose of this paper is to analyze and compare the different congestion control and avoidance mechanisms which have been proposed for TCP/IP protocols, namely: Tahoe, Reno, New-Reno, TCP Vegas and SACK. TCP's robustness is as a result of it's reactive behavior in the face of congestion, and fact that reliability is ensured by re-transmissions. All the above mentioned implementations suggest mechanisms for determining when a segment should be re-transmitted and how should the sender behave when it encounters congestion and what pattern of transmissions should it follow to avoid congestion. In this paper we shall discuss how the different mechanism affect the through put and efficiency of TCP and how they compare with TCP Vegas in terms of performance.

Introduction:

TCP is a reliable connection oriented end-to-end protocol. It contains within itself, mechanisms for ensuring reliability by requiring the receiver the acknowledge the segments that it receives. The network is not perfect and a small percentage of packets are lost en route, either due to network error or due to the fact that there is congestion in the network and the routers are dropping packets. We shall assume that packet losses due to network loss are minimal and most of the packet losses are due to buffer overflows at the router**[1]**. Thus it becomes increasingly important for TCP to react to a packet loss and take action to reduce congestion.

TCP ensures reliability by starting a timer whenever it sends a segment. If it does not receive an acknowledgement from the receiver within the 'time-out' interval then it retransmits the segment.

We shall start the paper by taking a breif look at each of the congestion avoidance algorithms and noting how they differ from each other. In the end we shall do a head to head comparison to further bring into light the differences.

TCP TAHOE:

Tahoe refers to the TCP congestion control algorithm which was suggested by Van Jacobson in his paper[1]. TCP is based on a principle of 'conservation of packets', i.e. if the connection is running at the available bandwidth capacity then a packet is not injected into the network unless a packet is taken out as well. TCP implements this principle by using the acknowledgements to clock outgoing packets because an acknowledgement means that a packet was taken off the wire by the receiver. It also maintains a congestion window CWD to reflect the network capacity**[1]**. However there are certain issues, which need to be resolved to ensure this equilibrium.

1) Determination of the available bandwidth.
2) Ensuring that equilibrium is maintained.
3) How to react to congestion.

Slow Start:

TCP packet transmissions are clocked by the incoming acknowledgements. However there is a problem when a connection first starts up cause to have acknowledgements you need to have data in the network and to put data in the network you need acknowledgements. To get around this circularity Tahoe suggests that whenever a TCP connection starts or re-starts after a packet loss it should go through a procedure called 'slow-start'. The reason for this procedure is that an initial burst might overwhelm the network and the connection might never get started. Slow starts suggests that the sender set the congestion window to 1 and then for each ACK received it increase the CWD by 1. so in the first round trip time(RTT) we send 1 packet, in the second we send 2 and in the third we send 4. Thus we increase exponentially until we lose a packet which is a sign of congestion. When we encounter congestion we decreases our sending rate and we reduce congestion window to one. And start over again.

The important thing is that Tahoe detects packet losses by timeouts. In usual implementations, repeated interrupts are expensive so we have coarse grain time-outs which occasionally checks for time outs. Thus it might be some time before we notice a packet loss and then re-transmit that packet.

Congestion Avoidance:

For congestion avoidance Tahoe uses 'Additive Increase Multiplicative Decrease'. A packet loss is taken as a sign of congestion and Tahoe saves the half of the current window as a threshold. value. It then set CWD to one and starts slow start until it reaches the threshold value. After that it increments linearly until it encounters a packet loss. Thus it increase it window slowly as it approaches the bandwidth capacity.

Problems:

The problem with Tahoe is that it take a complete timeout interval to detect a packet loss and in fact, in most implementations it takes even longer because of the coarse grain timeout. Also since it doesn't send immediate ACK's, it sends cumulative acknowledgements, there fore it follows a 'go back n ' approach. Thus every time a packet is lost it waits for a timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links.

TCP RENO:

This Reno retains the basic principle of Tahoe, such as slow starts and the coarse grain re-transmit 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. The logic behind this is that whenever we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in

sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements then that means that sufficient time has passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So Reno suggest an algorithm called **'Fast Re-Transmit'.** 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-Re-Transmit'[2].** The basic algorithm is presented as under:

1)Each time we receive 3 duplicate ACK's we take that to mean that the segment was lost and we re-transmit the segment immediately and enter 'Fast-Recovery'

2)Set SSthresh to half the current window size and also set CWD to the same value.

3)For each duplicate ACK receive increase CWD by one. If the increase CWD is greater than the amount of data in the pipe then transmit a new segment else wait. If there are 'w' segments in the window and one is lost, the we will receive (w-1) duplicate ACK's. Since CWD is reduced to W/2, therefore half a window of data is acknowledged before we can send a new segment. Once we re-transmit a segment, we would have to wait for atlease one RTT before we would receive a fresh acknowledgement. Whenever we receive a fresh ACK we reduce the CWND to SSthresh. If we had previously received (w-1) duplicate ACK's then at this point we should have exactly w/2 segments in the pipe which is equal to what we set the CWND to be at the end of fast recovery. Thus we don't empty the pipe, we just reduce the flow. We continue with congestion avoidance phase of Tahoe after that.

Problems:

Reno perform very well over TCP 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 re-transmitted 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. Suppose we send packets 1,2,3,4,5,6,7,8,9 in that order. Suppose packets 1, and 2 are lost. The ACK's generated by 2,4,5 will cause the re-transmission of 1 and the CWD is reduced to 7. Then when we receive ACK for 6,7,8,9 our CWD is sufficiently large to allow to us to send 10,11. When the re-transmitted segment 1 reaches the receiver we get a fresh ACK and we exit fast-recovery and set CWD to 4. Then we get two more ACK's for 2(due to 10,11) so once again we enter fast-retransmit and re-transmit 2 and then enter fast recovery. Thus when we exit fast recovery for the second time our window size is set to 2.

Thus we reduced our window size twice for packets lost 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.

NEW-RENO:

New RENO is a slight modification over TCP-RENO. It is able to detect multiple packet losses and thus is much more efficient that RENO in the event of multiple packet losses.

Like Reno, New-Reno also enters into fast-retransmit when it receives multiple duplicate packets, however it differs from RENO in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast-recovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the CWD multiples times.

The fast-transmit phase is the same as in Reno. The difference in the fast-recovery phase which allows for multiple re-transmissions in new-Reno. Whenever new-Reno enters fast-recovery it notes the maximums segment which is outstanding. The fast-recovery phase proceeds as in Reno, however when a fresh ACK is received then there are two cases:

If it ACK's all the segments which were outstanding when we entered fast-recovery then it exits fast recovery and sets CWD to ssthresh and continues congestion avoidance like Tahoe.

If the ACK is a partial ACK then it deduces that the next segment in line was lost and it re-transmits that segment and sets the number of duplicate ACKS received to zero.

It exits Fast recovery when all the data in the window is acknowledged**[3]**.

Problems:

New-Reno suffers from the fact that its take one RTT to detect each packet loss. When the ACK for the first re-transmitted segment is received only then can we deduce which other segment was lost.

SACK:

TCP with 'Selective Acknowledgments' is an extension of TCP Reno and it works around the problems face by TCP RENO and TCP New-Reno, namely detection of multiple lost packets, and re-transmission of more than one lost packet per RTT.

SACK retains the slow-start and fast-retransmit parts of RENO. It also has the coarse grained timeout of Tahoe to fall back on, incase a packet loss is not detected by the modified algorithm.

SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set CWND to half the current size. Every time it receives an ACK it reduces the pipe by 1 and every time it re-transmits a segment it increments it by 1. Whenever the pipe goes smaller than the CWD window it checks which segments are un received and send them. If there

are no such segments outstanding then it sends a new packet**[5]**. Thus more than one lost segment can be sent in one RTT.

Problems:

The biggest problem with SACK is that currently selective acknowledgements are not provided by the receiver To implement SACK we'll need to implement selective acknowledgment which is not a very easy task.

VEGAS:

Vegas is a TCP implementation which is a modification of Reno. It builds on the fact that proactive measure to encounter congestion are much more efficient than reactive ones. It tried to get around the problem of coarse grain timeouts by suggesting an algorithm which checks for timeouts at a very efficient schedule. Also it overcomes the problem of requiring enough duplicate acknowledgements to detect a packet loss, and it also suggest a modified slow start algorithm which prevent it from congesting the network. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet losses occur. However it still retains the other mechanism of Reno and Tahoe, and a packet loss can still be detected by the coarse grain timeout of the other mechanisms fail.

The three major changes induced by Vegas are:

New Re-Transmission Mechanism:

Vegas extends on the re-transmission mechanism of Reno. It keeps track of when each segment was sent and it also calculates an estimate of the RTT by keeping track of how long it takes for the acknowledgment to get back. Whenever a duplicate acknowledgement is received it checks to see if the (current time-segment transmission time)> RTT estimate; if it is then it immediately re-transmits the segment without waiting for 3 duplicate acknowledgements or a coarse timeout**[6]**. Thus it gets around the problem faced by Reno of not being able to detect lost packets when it had a small window and it didn't receive enough duplicate Ack's.

To catch any other segments that may have been lost prior to the re-transmission, when a non duplicate acknowledgment is received, if it is the first or second one after a fresh acknowledgement then it again checks the timeout values and if the segment time since it was sent exceeds the timeout value then it re-transmits the segment without waiting for a duplicate acknowledgment**[6]**. Thus in this way Vegas can detect multipple packet losses.

Also it only reduces its window if the re-transmitted segment was sent after the last decrease. Thus it also overcome Reno's shortcoming of reducing the congestion window multiple time when multiple packets are lost.

Congestion avoidance:

TCP Vegas is different from all the other implementation in its behavior during congestion avoidance. It does not use the loss of segment to signal that there is congestion. It determines congestion by a decrease in sending rate as compared to the expected rate, as result of large queues building up in the routers. It uses a variation of Wang and Crowcroft;s Tri-S scheme. The details can found in **[6]**. Thus whenever the calculated rate is too far away from the

expected rate it increases transmissions to make use of the available bandwidth, whenever the calculated rate comes too close to the expected value it decreases its transmission to prevent over saturating the bandwidth. Thus Vegas combats congestion quite effectively and doesn't waste bandwidth by transmitting at too high a data rate and creating congestion and then cutting back, which the other algorithms do.

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.

Conclusion:

Thus it is clear that TCP Vegas is definitely better than

1)Tahoe:

- Cause it is much more robust in the face of lost packets. It can detect and retransmit lost packet much sooner than timeouts in Tahoe.
- It also has fewer re-transmissions since it doesn't empty the whole pipe whenever it loses packets.
- It is better at congestion avoidance and its modified congestion avoidance and slow start algorithms measure incipient congestion and very accurately measure the available bandwidth available and therefore use network resources efficiently and don't contribute to congestion.

2)Reno:

- More than half of the coarse-grained timeouts of Reno are prevented by Vegas as it detects and re-transmits more than one lost packet before timeout occurs.
- It doesn't have to always wait for 3 duplicate packets so it can re-transmit sooner.
- It doesn't reduce the congestion window too much prematurely.
- The advantages that it has in congestion avoidance and bandwidth utilization over Tahoe exist here as well.

3)New-Reno:

- It prevents many of the coarse grained timeouts of New-Reno as it doesn't need to wait for 3duplicate ACK's before it retransmits a lost packet.
- Its congestion avoidance mechanisms to detect 'incipient' congestion are very efficient and utilize nework resources much more efficiently.
- Because of its modified congestion avoidance and slow start algorithm there are fewer re-transmits.

4)SACK:

TCP Vegas doesn't have a clear cut advantage over SACK TCP. The only

fields where it apprears to outperform SACK is:

- In its estimation of incipient congestion, and its efficient estimation of congestion by measuring change in throughput rather than packet loss. This would result in a better utilization of bandwidth and lesser congestion.

- Also it appears more stable than SACK. The reason for this being that SACK uses packet losses to denote congestion. So that the sender continually increase sending rate until there is congestion and then they cur back. This cycle continues and the system keeps on oscillating.. TCP Vegas flattens out its sending rate at the optimal bandwidth utilization point thus inducing stability.

- Another advantage of TCP Vegas or rather the disadvantage of SACK is that it is not very easy to incorporate SACK in the current TCP. We need fields to acknowledge the selective segments and this requires changes at the receiver as well, whereas all the other mentioned algorithms only require changes at the sender side.

References:

[1]V.Jacobson. **"Congestion Avoidance and Control".**SIGCOMM Symposium no Cummunication Architecture and protocols.

[2]V.Jacobson **"Modified TCp Congestion Control and Avoidance Alogrithms".**Technical Report 30,Apr 1990.

[3]S.Floyd, T.Henderson **"The New-Reno Modification to TCP's Fast Recovery Algorithm"** RFC 2582, Apr 1999.

[4]O. Ait-Hellal, E.Altman **"Analysis of TCP Reno and TCP Vegas".**

[5]K.Fall, S.Floyd **"Simulation Based Comparison of Tahoe, Reno and SACK TCP" .**

[6]L.S.Brakmo, L.L. Peterson, **"TCP Vegas: End to End Congestion Avoidance on a Global Internet"**, IEEE Journal on Selected Areas in Communication, vol. 13[1995],(1465-1490).