Name : S.V.L.Chandrika

Admission No. : 18001A0518

Branch : Computer Science and Engineering

Subject : Computer Networks

College : Jawaharlal Nehru Technological University

Anantapur

**A survey on TCP implementations:**

TCP Implementation, the different mechanism affect the put and efficiency of TCP and how they compare the different congestion control and avoidance mechanism which have been proposed for TCP/IP protocols, namely: TCP Reno,Tahoe, TCP vegas, New-Reno and SACK.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.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.

**TCP Tahoe:**

Tahoe refers to the TCP congestion control algorithm which was suggested by Van Jacobson. TCP is based on a principle of ‘conservation of packets’, i.e. if the connection is running at the available ban dwidth 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.

Slow Start:

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.

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.

**TCP RENO:**

TCP Reno retains the basic principle of Tahoe, such as slow starts and the coarse gain 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.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 thesegment 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’. 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 retransmit a segment, we would have to wait for atleast one rtt before we 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 coTCP VEGAS:

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.Vegas is a TCP implementation which is a modification of Reno.

The three major changes induced by Vegas are:

1)New Re-Transmission Mechanism

2)congestion avoidance

3)Modified slow-start

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 retransmits the segment without waiting for 3 duplicate acknowledgements or a coarse timeout. 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 retransmission, 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. Thus in this way Vegas can detect multiple 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 behaviour 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.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 preventover 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.ntinue with congestion avoidance phase of Tahoe after that.

**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. 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.

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

**SACK::**

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.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 .It 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.Every time it receives an ack it reduces the pipe by 1 and every time it retransmits a segment it increments it by 1. Whenever the pipe goes smaller than the cwd window it checks which segments are un receive and send them. If there are no such segments outstanding then itsends a new packet.Thus more than one lost segment can be sent in one rtt .

**Problems:**

The main and important problem with sack is that currently selective acknowledgements are not provided by the receiver to implement sack we will 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 keepstrack 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)&gt; RTT estimate; if it is then it immediately retransmits the segment without waiting for 3 duplicate acknowledgements or a coarse timeout. 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.

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.

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.

By observing all we can conclude that TCP Vegas is better than the SACK, RENO,

NEW-RENO and TAHOE.