Survey of End-to-End TCP Congestion Control Protocols

Mrs Uma R Pujeri
Assistant Professor,
Adithya Institute of Technology,
Coimbatore.
umapujeri@gmail.com

Dr. V. Palanisamy
Principal,
INFO college of Engineering,
Coimbatore
info@infoengg.com

Abstract - **This paper is a survey of TCP congestion control principles and techniques. The TCP protocol is used by the majority of the network applications on the Internet. TCP performance is strongly influenced by its congestion control algorithms that limit the amount of transmitted traffic based on the estimated network capacity and utilization. In this work TCP Tahoe, TCP Reno, TCP NewReno, TCP Vegas are considered. Also discussed how TCP reacts to the congestions and depending on the algorithm how the congestion window is minimized and how the packets are retransmitted. Simulation has been done with WAN type network analysing the throughput and performance of these TCP variants in order to discover which of them has a better performance.**
Index Terms—TCP, Protocol, PSTN, Congestion, CWD, ACK, RTT, SSThresHold.

I. INTRODUCTION

Today, computer networks are the core of modern communication. The scope of communication has increased tremendously and all aspect of public switched telephone network(PSTN) are computer controlled. Due to computer networks communication between networked computers was possible. Computer networks made growth in business, online shopping, online education, online reservation possible. This made computer network to grow tremendously. The number of host started increasing exponentially increasing the traffic on the network. As the traffic on the networks increases the congestion in the network also increases.

Congestion has become an important issue in the packet-switched networks. When the traffic or load in the network increases beyond the capacity of the network will lead to the congestion in the computer network. Congestion will severely affect the throughput. When congestion occurs in a computer network throughput will severely decrease and delay will increase.

II. CONGESTION CONTROL ALGORITHMS

The algorithm for congestion control is the main reason we can use the internet successfully today. Despite largely unpredictable user access of internet, resource bottleneck and limitation, without TCP congestion control algorithm the Internet could have became a history a long time ago. During congestion, the network throughput may drop to zero and the path delay may become very high. A congestion control algorithm helps the network to recover from the congestion state. A congestion avoidance scheme reduces the delay in the network and increases throughput. Such schemes prevent a network from entering the congested state. Congestion avoidance is a prevention mechanism while congestion control is a recovery mechanism. The purpose of this paper is to analyse and compare the different congestion control and avoidance mechanisms which have been proposed for TCP/IP protocols, namely: Tahoe, Reno, New-Reno, and TCP Vegas. TCP is a reliable connection-oriented end-to-end reliable protocol. TCP sends a packet and waits for an acknowledgefrom a receiver. During the transmission the packet may be lost due to two reasons

- Due to network error
- Or due to congestion.

Thus it becomes important for TCP to react to a packet loss and take action to reduce congestion. TCP ensure reliability. It sets timer and waits for acknowledgement from the receiver. If it does not receive the acknowledgement within the specified timer. It retransmits the packet again to the receiver. In this paper we will study different congestion control algorithm and how they react when the congestion occurs and how they differ from each other.

A. *TCP TAHOE*

TCP Tahoe is a congestion control algorithm. TCP is an reliable protocol i.e. TCP sends packets and waits for an acknowledge when the receiver receive the packets it sends back the positive acknowledge .TCP also ensures the equilibrium i.e. number of packets sent is equal to the number of packets received. It also maintains a congestion window CWD to reflect the network capacity. 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.

TCP Tahoe Congestion Control algorithm ensures this equilibrium

- Slow start
- Congestion Avoidance
- Fast Retransmit.

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 suggest 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.

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.

Fast Retransmit: Coarse grained TCP time-outs sometimes lead to long periods wherein a connection goes dead waiting for a timer to expire. Fast Retransmit a heuristic that sometimes "triggers" the retransmission of a packet faster than permissible by the regular time-out. Every time a data packet arrives at a receiver, the receiver ACKs even though the particular sequence number has been acknowledged. Thus, when a packet is received in out of order, resend the ACK sent last time a duplicate ACK.

When a duplicate ACK is seen by the sender, it infers that the other side must have received a packet out of order. Delays on different paths could be different thus; the missing packets may be delivered. So wait for "some" number of duplicate ACKs before resending data. This number is usually 3.

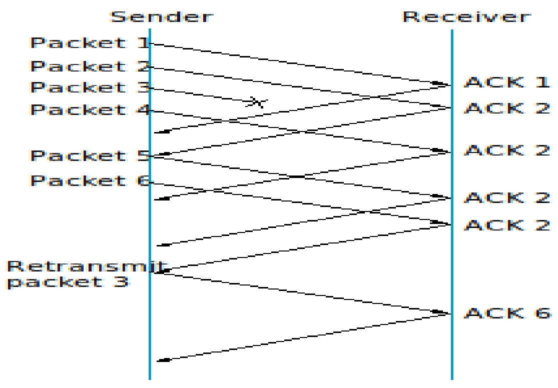

Figure 1:Fast Retransmit

Generally, fast retransmit eliminates about half the coarse-grain timeouts. This yields roughly a 20% improvement in throughput.

Problem : 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, therefore 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 bandwidth delay product links.

Thus Due to automatic set back to slow start mode of operation with initial congestion window of one every time packet loss is detected we see TCP Tahoe does not prevent the communication link from going empty. Hence this may have high cost in high bandwidth product links.

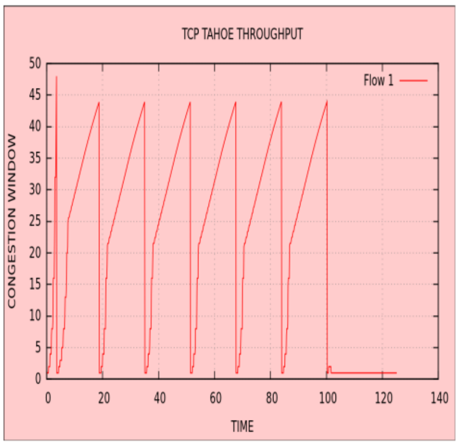

FIGURE 2: TCP TAHOE THROUGHPUT

You can see the operation of TCP Tahoe clearly from the above figure:

1. At approximately time 0, TCP Tahoe starts and it is in the slow start mode: the congestion window size increases exponentially
2. At approximately time 5, packet loss is detected. TCP marks SSThresh = 25 (approximately) and begins another slow start
3. When it reaches CWND = 25 (approximately), the CWND increases linearly - here TCP Tahoe enters the congestion avoidance mode
4. At approximately time 19, TCP Tahoe detects packet loss and begins a slow start.
   SSThresHold is approximately 22.
   TCP begins another slow start and so on.

B. *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 this 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'. 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 at least 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.

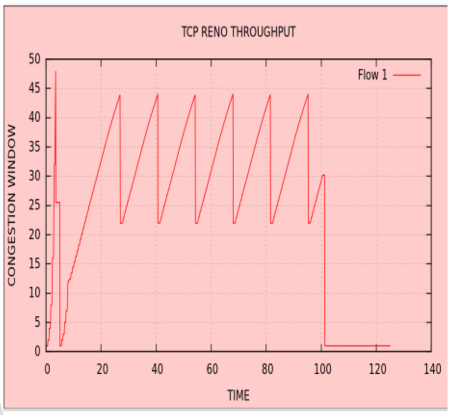

FIGURE 3 TCP RENO THROUGHPUT

Problems: Reno performs 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 its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet loss. 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. 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.

C. *TCP NEW RENO*

TCP NewReno is same as the TCP Reno only difference between the TCP NewReno and TCP Reno is that TCP NewReno can handle multiple packet loss without coming of fast recovery phase.

TCP New Reno Algorithm

The Idea: If the sender remembers the number of the last segment that was sent before entering the Fast Retransmit phase then it can deal with a situation when a "new" ACK (which is not **duplicate ACK**) does not cover the last remembered segment (**"partial ACK"**) This is a situation when more packets were lost before entering the Fast Retransmit. After discovering such situation the sender will retransmit the new lost packet too and will stay at the Fast Recovery stage. The sender will finish the Fast Recovery stage when it will get ACK that covers last segment sent before the Fast Retransmit

Algorithm

[1] Set ssthresh to max (FlightSize / 2, 2*MSS) (FlightSize is number of unacknowledged packet)
[2] Record to "Recovery" variables the highest sequence number transmitted
[3] Retransmit the lost segment and set cwnd to ssthresh + 3*MSS.
[4] The congestion window is increased by the number of segments (three) that were sent and buffered by the receiver
[5] For each additional duplicate ACK received, increment cwnd by MSS.
[6] Thus, the congestion window reflects the additional segment that has left the network.
[7] Transmit a segment, if allowed by the new value of cwnd and the receiver's advertised window.

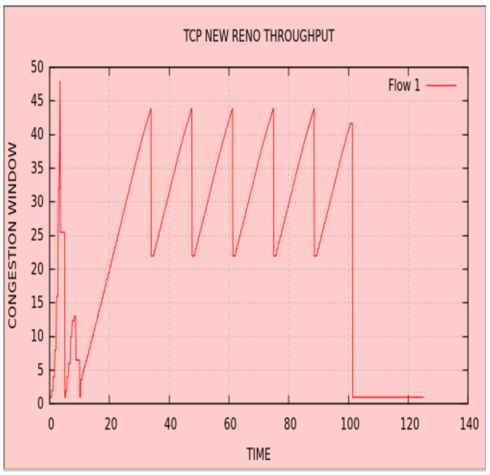

Figure 4 TCP NEW RENO THROUGHPUT

D. TCP VEGAS

TCP Vegas was originally proposed by Brakmo and had several new features in it. One of the most important differences between TCP Vegas and TCP Reno is its congestion avoidance scheme. While TCP Reno (and its variants like NewReno) rely on packet loss detection to detect network congestions, TCP Vegas uses a sophisticated bandwidth estimation scheme to proactively gauge network congestion. TCP Vegas varies it congestion window (cwnd) using the following algorithm

Record sending time of every packet and the time of ACK reception when it receives a duplicate ACK, Vegas checks to see if the RTT is greater than timeout. If it is, then without waiting for the third duplicate ACK, it immediately retransmits the packet.

When a non-duplicate ACK is received, if it is the first or second ACK after a retransmission, Vegas again checks to see if the RTT is greater than timeout. If it is, then Vegas retransmits the packet.

Note: Only decrease congestion window for current window size, not for previous window size.

Congestion Avoidance:

BaseRTT = min RTT ever
Exp Throughput = window size / BaseRTT
Actual Rate = segment size / measured RTT of corresponding RTT
Diff = Exp Throughput – Actual Rate
Diff < α: increase window linearly;
Diff > β: decrease window linearly;
α < Diff < β: keep window size unchanged.
The sender defines three thresholds α, β and γ we set $\alpha = 2$, $\beta = 4$ and $\gamma = 1$

Slow-Start:

Expo growth every other RTT, instead of every RTT as in Reno.

Compare Exp with Actual as in congestion avoidance. When Actual < γ, Vegas changes from slow start to linear increase/decrease.

Result:

Smooth window size, no saw tooth.

40% – 70% throughput improvement.

Problems:

1. Vegas has good performances when the queue sizes at intermediate routers are large; it enables a full utilization of the link while it keeps the sending rate smoother. However, when the available bandwidth is not sufficiently large, Vegas has the same behaviour as Reno and does

not manage to make efficient use of its new mechanism for congestion detection.

2. Sensitive to RTT estimation.

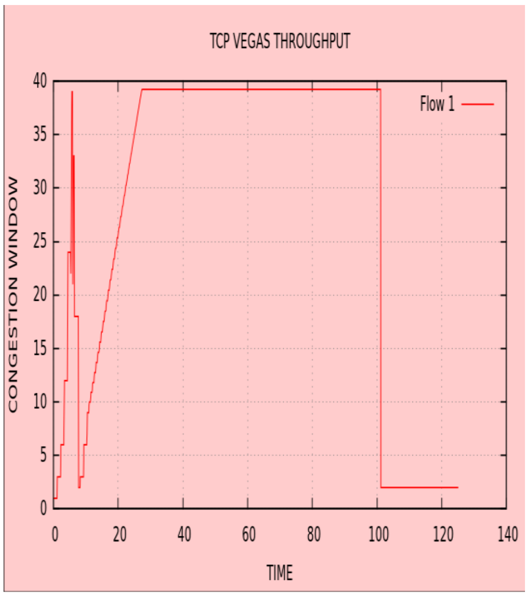

FIGURE 5 TCP VEGAS THROUGHPUT

1) COMPARISION OF TCP CONTROL ALGORITHMS

Table 1: Comparisons of TCP Algorithm

	Algorithm
TCP TAHOE	1. Slow Start 2. Congestion Avoidance Fast Recovery
TCP RENO	1. Slow Start 2. Congestion Avoidance 3. Fast Recovery Fast Retransmission
TCP NEW RENO	1. Slow Start 2. Congestion Avoidance 3. Fast Recovery 4. Fast-Retransmit TCP NewReno can handle multiple packet loss without coming out of fast recovery phase.
TCP VEGAS	1. Slow Start 2. Packet loss detection 3. Detection of available bandwidth

2) COMPARISION OF TCP CONTROL ALGORITHMS WRT PROBLEMS.

Table 2: Comparisons of TCP Algorithm wrt Problems

	Problems
TCP TAHOE	Every time a packet is lost it waits for the timeout and then retransmits the packet. It reduces the size of congestion window to 1 just because of 1 packet loss, this inefficiency cost a lost in high bandwidth delay product links
TCP RENO	TCP Reno is helpful when only 1 packet is lost, in case of multiple packet loss it acts as Tahoe. Then evolved TCP New Reno which is a modification of TCP Reno and deals with multiple packets loss.
TCP NEW RENO	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.
TCP VEGAS	Vegas has good performances when the queue sizes at intermediate routers are large; it enables a full utilization of the link while it keeps the sending rate smoother. However, when the available bandwidth is not sufficiently large, Vegas has the same behaviour as Reno and does not manage to make efficient use of its new mechanism for congestion detection. Sensitive to RTT estimation.

3) COMPARISION OF TCP CONTROL ALGORITHMS WRT NUMBER OF PACKET SENT,RECEIVED,DROPED AND ACKNOWLEDGED.

Table 3: Comparisons of TCP Algorithm WRT PACKET SENT,PACKET DROPED,PACKET RECEIVED AND ACK

	No of Packet Sent	No Of Packet Received	No of Packet Drop	No of ACK
TCP TAHOE	9736	9736	23	43812
TCP RENO	10264	10264	23	46188

TCP NEW RENO	9796	9796	25	44082
TCP VEGAS	11624	11624	3	52308

III. CONCLUSION

In this paper, we have evaluated the performance characteristics of various TCP congestion control schemes under the wired network conditions with bottleneck end-to-end link capacities. We can conclude based on throughput calculation that TCP vegas gives highest performance as it can change its congestion window based on network traffic situation.

REFERENCES

[1] Van Jacobson, Congestion Avoidance and Control. Computer Communications Review, 18 (4), August 1988, 314-329.

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

[3] Paul Meeneghan and Declan Delaney, An Introduction to NS, Nam and Otcl scripting, Department of Computer Science, National University of Ireland, Maynooth, 2004-05.

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

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

[6] B. Sikdar, S. Kalyanaraman and K. S. Vastola, Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno,and SACK, IEEE/ACM Transactions On Networking, 11(6), December 2003, 959-971.

[7] Amit Aggarwal, Stefan Savage, and Thomas Anderson.Understanding the Performance of TCP Pacing, March 30, 2000, IEEE InfoCom 2000.

[8] Harris Interactive. P.C. and Internet Use Continue to Grow at Record Pace. Press Release, February 7, 2000.

[9] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997, RFC 2001.

[10] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in 4.3-Tahoe BSD TCP Congestion Control Algorithm. ACM Computer Communication Review, 22(2):9–16, Apr. 1992