IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735. Volume 5, Issue 6 (Mar. - Apr. 2013), PP 74-79
www.iosrjournals.org

Comparison of TCP congestion control mechanisms Tahoe, Newreno and Vegas

Digvijaysinh B Kumpavat[1], Prof. Paras S Gosai[2], Prof. Vyomal N Pandya[3]
[1](Research student, Department of EC engineering, Govt.Engg. College, Surat, Gujarat, India)
[2](Asso. Prof., Department of EC engineering, Govt.Engg. College, Surat, Gujarat, India)
[3](Asst. Prof., Department of EC engineering, CKPCET, Surat, Gujarat, India)

***Abstract*:** *The widely used reliable transport protocol TCP, is an end to end protocol designed for the wireline networks characterized by negligible random packet losses. This paper represents exploratory study of TCP congestion control principles and mechanisms. Modern implementations of TCP contain four intertwined algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery. In addition to the standard algorithms used in common implementations of TCP, this paper also describes some of the more common proposals developed by researchers over the years. We also study, through extensive simulations, the performance characteristics of four representative TCP schemes, namely TCP Tahoe, New Reno and Vegas under the network conditions of bottleneck link capacities for wired network.*

***Keywords* -** *Congestion avoidance, Congestion control mechanisms, Newreno, Tahoe, TCP, Vegas.*

I. Introduction

The standard algorithms in TCP implementations today can be found in RFC 2001[4]. This reference document specifies four standard congestion control algorithms that are now in common use. The four algorithms, Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recoveries are described below.

1.1 Slow start

Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate, otherwise known as sender-based flow control. This is accomplished through the return rate of acknowledgements from the receiver. When a new connection is established, the congestion window is initialized to one segment. Each time an ACK is received, the congestion window is increased by one segment. The sender can transmit up to the minimum of the congestion window and the advertised window. The congestion window is flow control imposed by the sender, while the advertised window is flow control imposed by the receiver. The former is based on the sender's assessment of perceived network congestion; the latter is related to the amount of available buffer space at the receiver for this connection. The sender can transmit the minimum of the congestion window and the advertised window of the receiver, which is simply called the transmission window. The sender starts by transmitting one segment and waiting for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent. When each of those two segments is acknowledged, the congestion window is increased to four. This provides an exponential growth, although it is not exactly exponential because the receiver may delay its ACKs, typically sending one ACK for every two segments that it receives. At some point the capacity of the internet can be reached, and an intermediate router will start discarding packets. This tells the sender that its congestion window has gotten too large.

1.2 Congestion avoidance

Congestion can occur when data arrives on a big pipe (a fast LAN) and gets sent out a smaller pipe (a slower WAN). Congestion can also occur when multiple input streams arrive at a router whose output capacity is less than the sum of the inputs. Congestion avoidance is a way to deal with lost packets. In the Congestion Avoidance algorithm a retransmission timer expiring or the reception of duplicate ACKs can implicitly signal the sender that a network congestion situation is occurring. The sender immediately sets its transmission window to one half of the current window size (the minimum of the congestion window and the receiver's advertised window size), but to at least two segments. If congestion was indicated by a timeout, the congestion window is reset to one segment, which automatically puts the sender into Slow Start mode. If congestion was indicated by duplicate ACKs, the Fast Retransmit and Fast Recovery algorithms are invoked.

1.3 Fast retransmit

When a duplicate ACK is received, the sender does not know if it is because a TCP segment was lost or simply that a segment was delayed and received out of order at the receiver. Typically no more than one or two

duplicate ACKs should be received when simple out of order conditions exist. If however more than two duplicate ACKs are received by the sender, it is a strong indication that at least one segment has been lost. The TCP sender will assume enough time has lapsed for all segments to be properly re-ordered by the fact that the receiver had enough time to send three duplicate ACKs.

When three or more duplicate ACKs are received, the sender does not even wait for a retransmission timer to expire before retransmitting the segment (as indicated by the position of the duplicate ACK in the byte stream). This process is called the Fast Retransmit algorithm and was first defined in [5]. Immediately following Fast Retransmit is the Fast Recovery algorithm.

1.4 Fast recovery

It is an improvement that allows high throughput under moderate congestion, especially for large windows. The receipt of the duplicate ACKs tells TCP more than just a packet has been lost. Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer.

The fast retransmit and fast recovery algorithms are usually implemented together as follows. [4]

1. When the third duplicate ACK in a row is received, set ssthresh to value:

ssth = min (cwnd/2 , 2 MSS) (1)

Retransmit the missing segment. Set cwnd to ssthresh plus 3 times the segment size. This inflates the congestion window by the number of segments that have left the network and which the other end has cached.

2. Each time another duplicate ACK arrives, increment cwnd by the segment size. This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of cwnd:

cwnd = ssth + no. of dupacks received (2)

3. When the next ACK arrives that acknowledges new data, set cwnd to ssthresh (the value set in step 1). This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

II. Congestion Control Mechanisms

Here we will discuss three congestion control mechanisms.

2.1 TCP Tahoe

TCP Tahoe added the *slow-start*, *congestion avoidance*, and *fast retransmit* algorithms to TCP. TCP Tahoe is briefly described in [8]. With fast retransmit, when a packet is lost, instead of waiting for the retransmission timer to expire, if *tcprexmtthresh* (usually three) duplicate ACKs are received, the sender infers a packet loss and retransmits the lost packet. The sender now sets its *ssthresh* to half the current value of *cwnd* (maintained in bytes) and begins again in the slow-start mode with an initial window of 1. The slow start phase lasts till the *cwnd* reaches *ssthresh* and then congestion avoidance takes over. In this phase, the sender increases its *cwnd* linearly by *cwnd* for every new ACK it receives. Note that with TCP Tahoe the sender might retransmit packets which have been received correctly. Timeouts are used as the means of last resort to recover lost packets. On receiving three *dupacks* Tahoe starts Fast retransmit phase in which it retransmits packet and set *ssth* to half of *cwnd* and then enters in *slow start* phase by setting *cwnd* to one segment.

2.2 TCP Newreno

Standard TCP schemes such as Reno are simple and effective for reliable data transfer in wired networks, where packet loss due to bit errors is rare. TCP newreno [1] have similar slow-start, congestion avoidance and fast retransmit- recovery algorithm as Reno. During the slow-start phase, the sender increases its congestion window *cwnd* by one with each acknowledgment (ACK) received, until the slow-start threshold *ssth* is reached and the congestion avoidance phase takes over. In congestion avoidance, the sender increases its *cwnd* linearly by 1/ *cwnd* with each ACK received. Upon receiving triple duplicate ACKs (TD), the sender infers a packet loss and retransmits the lost packet, i.e. fast retransmit. The sender then sets *ssth* to *cwnd* / 2 , halves its *cwnd* and activates the fast recovery algorithm. In fast recovery, upon receiving a duplicate ACK, by assuming that a packet has left the networks, the sender sends/retransmits a packet to maintain the link to the receiver full. Upon receiving a *partial ACK*, the sender retransmits the first unacknowledged packet. When an ACK acknowledges all the packets transmitted before the fast retransmit triggered, the sender exits fast recovery and sets the congestion window *cwnd* to the slow-start threshold *ssth*. Then the sender enters the congestion avoidance phase.

In the case of multiple packets dropped from a single window of data, the first new information available to the sender comes when the sender receives an acknowledgement for the retransmitted packet (that is

the packet retransmitted when Fast Retransmit was first entered). If there had been a single packet drop, then the acknowledgement for this packet will acknowledge all of the packets transmitted before Fast Retransmit was entered (in the absence of reordering). However, when there were multiple packet drops, then the acknowledgement for the retransmitted packet will acknowledge some but not all of the packets transmitted before the Fast Retransmit. We call this packet a partial acknowledgment. Which is described in [2].

2.3 TCP Vegas

Vegas is an implementation of TCP that achieves between 37 and 71 % better throughput [3]on the Internet, with one-fifth to one-half the losses, as compared to the implementation of TCP in the Reno distribution. There are three techniques that Vegas employs to increase throughput and decrease losses.

2.3.1 New retransmission mechanism

TCP Vegas introduces three changes that affect TCP's (fast) retransmission strategy. First, TCP Vegas measures the RTT for every segment sent. The measurements are based on fine-grained clock values. 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. Second, 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. Third, in case of multiple segment loss and more than one fast retransmission, the congestion window is reduced only for the first fast retransmission.

2.3.2 Congestion avoidance mechanism

TCP Vegas does not continually increase the congestion window during congestion avoidance. It controls *cwnd* by observing changes of RTTs (Round Trip Time) of segments that the connection has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles *cwnd* down. If RTTs become small, on the other hand, TCP Vegas determines that the network is relieved from the congestion, and increases *cwnd*.

In congestion avoidance phase, the *cwnd* is updated as shown in [6]:

$$Cwnd\ (t+t_A) = \begin{cases} cwnd(t)+1, & \text{if} \quad diff < \alpha/base\_rtt \\ cwnd(t), & \text{if} \quad \alpha/base\_rtt < diff < \beta/base\_rtt \\ cwnd(t)-1, & \text{if} \quad \beta/base\_rtt < diff \end{cases} \qquad (3)$$

$$diff = cwnd(t)/base\_rtt - cwnd/rtt$$

where *rtt* is an observed round trip time, *base_rtt* is smallest value of observed RTTs, and α and β are some constant values.

2.3.3 Modified slow-start mechanism

A similar congestion detection mechanism is applied during slow-start to decide when to change to the congestion avoidance phase. To have valid comparisons of the expected and the actual throughput, the congestion window is allowed to grow only every other RTT.

In [3], an additional algorithm is presented, which tries to infer available bandwidth during slow-start from ACK spacing. However, this algorithm was marked experimental, and it was not used in the evaluation of TCP Vegas.

III. Simulation Environment

This section describes the simulation environment used to investigate the influence of the various congestion control mechanism in TCP. Here we used simulator ns-2 for better scheduling event and controlled environment.

3.1 Network topology

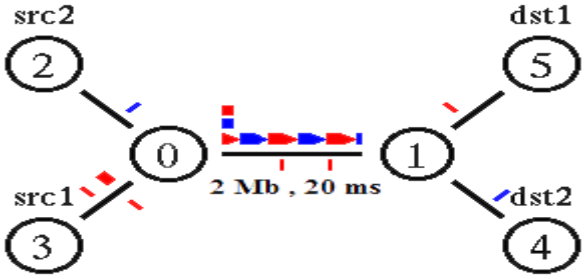

Figure 1 Network topology

The network model that we will use in the analysis is depicted in Fig. 1. The model consists of two sources (srcl, src2), two destinations (dst1, dst2), two intermediate nodes (or routers) (node 0 and node 1), and links interconnecting between the end stations and routers. We consider two connections; Connection 1 from src1 to dst1, which is assigned TCP traffic, and Connection 2 from src2 to dst2, which is assigned UDP traffic. Both connections are established via Node 0 and Node 1, and the link between Node 0 and Node 1 is shared between two connections. The bandwidth of the shared bottleneck link is 2 Mbps. The buffer size of Node 0 is 10 [segments]. The propagation delays between srci and dsti is 40 ms (i=1,2)**.**

IV. Simulation for Congestion window versus time

Congestion window of TCP changes based on change in its basic algorithms for every TCP variant. Simulation result of congestion window describes slow start, congestion avoidance, fast retransmit and fast recovery algorithms in TCP variants.

4.1 TCP Tahoe

Fig. 2 shows *cwnd* vs time for TCP tahoe for network topology defined as above.

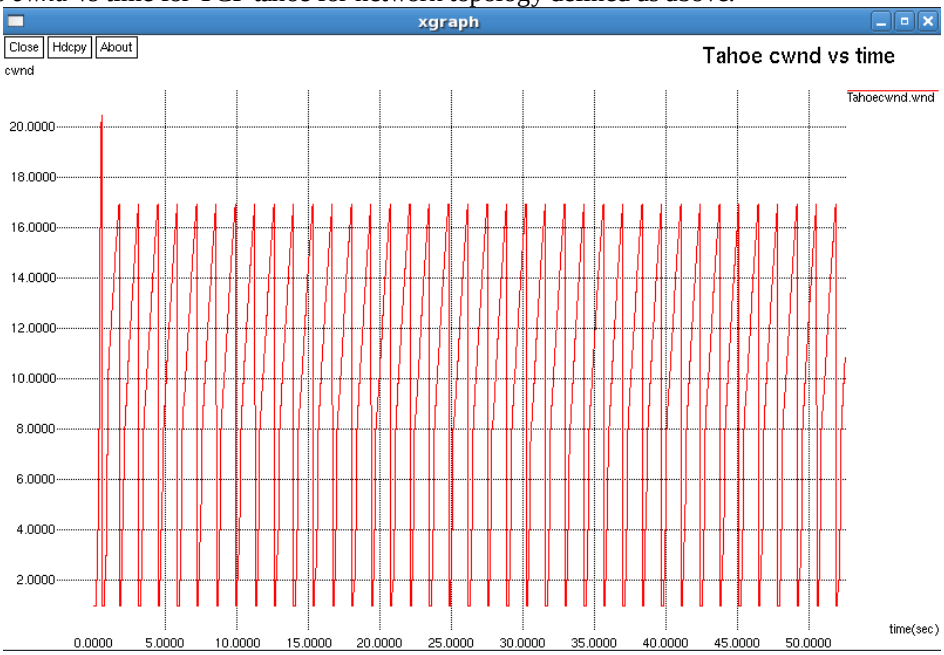

Figure 2 cwnd vs. Time for TCP Tahoe

As shown in Fig. 2 TCP Tahoe *cwnd* is exponentially increased in slow start phase up to *ssth.* In congestion avoidance phase *cwnd* is increased linearly and after receiving three *dupacks* TCP enters in fast retransmit phase. The *ssth* is now set to the half of *cwnd* and congestion window is set to the one segment and TCP enters in slow start phase.

4.2 TCP Newreno:

Fig. 3 shows *cwnd* vs time for TCP Newreno for network topology defined as above.

As shown in Fig. 3 TCP Tahoe *cwnd* is exponentially increased in slow start phase up to *ssth.* In congestion avoidance phase *cwnd* is increased linearly and after receiving three *dupacks* TCP enters in fast retransmit

phase. The *ssth* is now set to the half of *cwnd* and congestion window is set to the value *ssth*, that is, half of the previous *cwnd*. So TCP enters in congestion avoidance phase after retransmission.

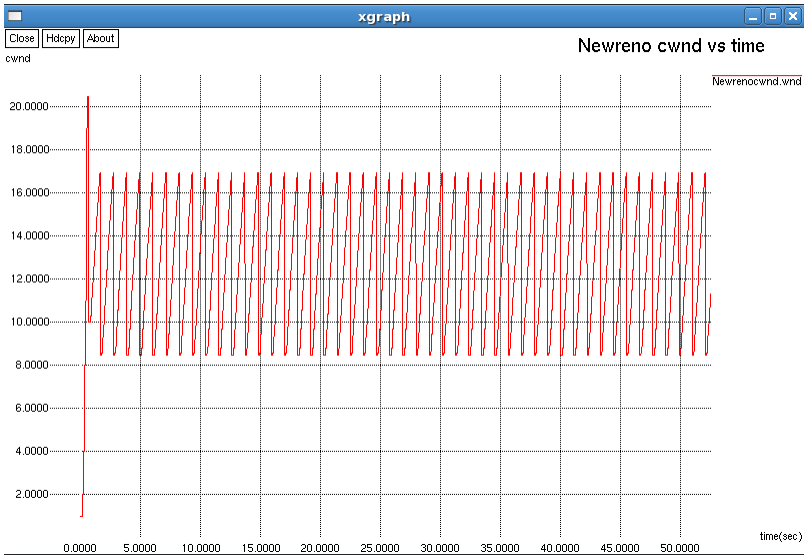

Figure 3 cwnd vs. Time for TCP Newreno

4.3 TCP Vegas:

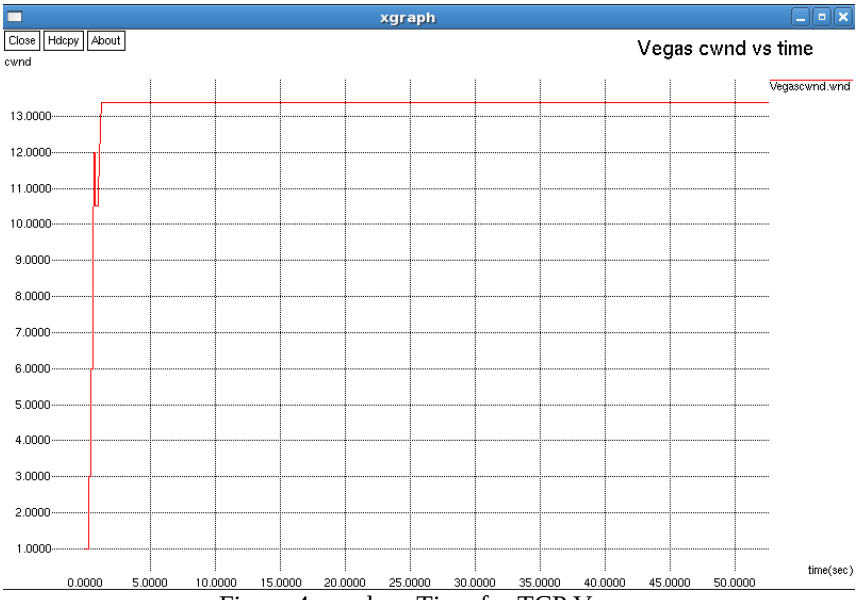

Figure 4 cwnd vs. Time for TCP Vegas

Fig. 4 shows *cwnd* vs time for TCP Vegas for network topology defined as above. As shown in Fig.4 *cwnd* of TCP vegas increase by rate half than TCP Tahoe and Newreno in slow start phase. In congestion avoidance phase *cwnd* is set to constant value as *cwnd* is controlled according to network traffic prediction based on observed RTT values.

V. Throughput vs time

Fig. 5 shows throughput vs time graph of TCP Tahoe, Newreno and Vegas. As we can see that throughput value increases abruptly initially, but then throughput is constant with time which indicates that packets delivery per RTT epoch is constant i.e same number of packets are delivered by network in certain amount of cyclic period of time.

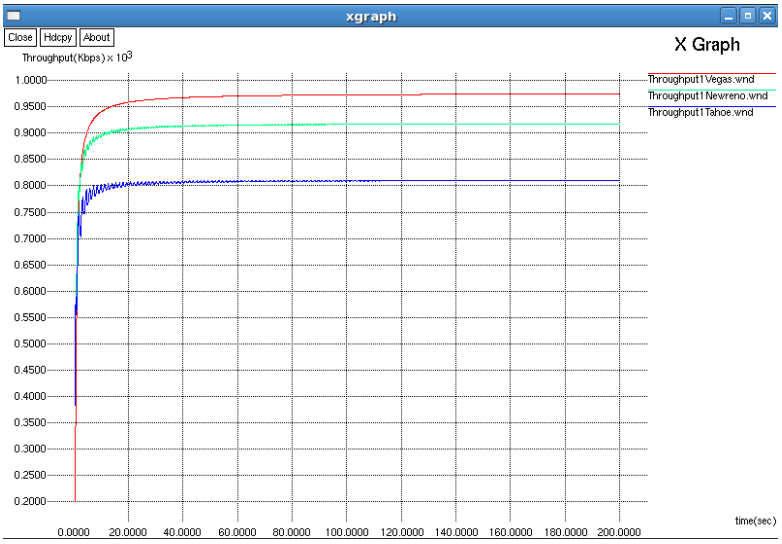

Figure 5 Comparison of TCP Tahoe, Newreno and Vegas throughput vs time

As shown in Fig. 5 TCP Vegas (indicated by RED line) has highest value of throughput, which is 974.803 Kbps for given topology. This due to its wise changes in slow start, congestion avoidance and retransmission algorithms. TCP Newreno (indicated by GREEN line) is next to TCP Vegas. Throughput of Newreno is 918.249 Kbps, which is degraded as compared to Vegas due to throttling of congestion window. TCP Tahoe (indicated by BLUE line) has lowest value of throughput, which is 810.187 Kbps for given topology, as it starts from slow start phase every time after retransmission.

VI. 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, and both type of traffic TCP and UDP. 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. However in wireless network the whole scenario may differ. This is because any packet loss over the wireline links is mainly on account of congestion unlike wireless links where packet losses can result both due to congestion and random losses. Since, TCP does not distinguish between congestion losses and random losses, the throughput of a TCP connection over a wireless link may suffers.

References

[1] Kai Xu, Ye Tian, Nirwan Ansari, Improving TCP performance in integrated wireless communications networks, Elsevier Computer Networks 47, 2005, 219-237

[2] T. Henderson, S. Floyd and A. Gurtov, The Newreno Modification to TCP's Fast Recovery Algorithm, RFC 6582, April 2012, 1-16

[3] Lawrence S. Brakmo and Larry L. Peterson, TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE Journal on Selected Areas In Communication ,13 (8), October 1995, 1465-1480

[4] W. Stevens, TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, January 1997, 1-7

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

[6] G. Hasegawa, M. Murata, and H. Miyahara, Fairness and stability of the congestion control mechanism of TCP, Proceedings of IEEE INFOCOM'99, March 1999, 1329–1336.

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

[8] 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.