Analysis of Different Congestion Avoidance Algorithms

Dr. Neeraj Bhargava
Associate Professor: Dept. of Computer Science School of Engineering & System Sciences MDS University, Ajmer, India.

Dr. Ritu Bhargava
Lecturer: Dept. of Computer Application, Govt. Women Engineering College, Ajmer, India.

Manish Mathuria
Research Scholar: Dept. of C.E. and I.T., Govt. Engineering College, Ajmer, India.

Shilpi Gupta, Kamal Kumar Jyotiyana
Dept. of Computer Science
School of Engineering & System Sciences
MDS University, Ajmer, India.

***Abstract*—The network is a crucial part of any communication, so the motivational part is that how to maintain the network smoothness. The purpose of this research paper is to analyze and compare the different congestion control & avoidance mechanisms. Some important mechanisms have been proposed for TCP/IP protocols, namely: Tahoe, Reno, New-Reno, TCP Vegas, and SACK. Basically, the core concept of this paper is to compare TCP Tahoe and Reno. Reno has two versions: first, is New-Reno a modified version of TCP, and the second version is TCP SACK. Commonly, TCP has three Performance Parameters, i.e. TCP-Throughput, Average End-to-End Delay, and Packet-Delivery-Fraction (in high & low mobility). TCP Protocols ensure better data transfer, speed, and reliability and congestion control. Mainly, TCP includes various network congestion algorithms like, Additive Increase Multiplicative Decrease (AIMD), Slow Start, Congestion Avoidance, Fast Retransmission, Fast Recovery. This paper benchmarked TCP Congestion Avoidance Algorithms.**

Keywords-TCP, Tahoe, Reno, Vegas, AIMD, SACK, New-Retransmission.

I. INTRODUCTION

TCP (Transmission Control Protocol) is a set of rules (protocol) used along with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. While IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet. TCP is a reliable, connection oriented protocol which specifies ordered delivery of packet [1][2]. TCP is used for transmission of data from an application to the network.

TCP/IP provides end to end connectivity between nodes and specifies how the data should be transmitted, formatted, routed and received at the receiving end. TCP IP has four layers and each layer work individually and hides the method from another layer:

I. Link Layer (contains Ethernet) used in the local communication link for local network.

II. Internet Layer (contains IP) performs in a local area network or in a wide area network.

III. Transport Layer (contains TCP) used in communication between two or more nodes.

IV. Application Layer (contains HTTP) used to transmits applications, messages and more. HTTP also used in communication services such as client to server for providing web services and also much more [3].

II. CONGESTION AVOIDANCE IN TCP

Congestion simply means overfill or overcrowded. Congestion refers to a network state where a node or link carries so much data that quality of service deteriorates resulting in queuing delay, frame or data packet loss and the blocking of new connections. In a congested network, response time slows with reduced network throughput [4]. Congestion occurs when bandwidth is insufficient and network data traffic exceeds capacity.

Computer networks have experienced an explosive growth over the past few years and with that growth have come severe congestion problems. In congestion it is common to see that internet gateways drop 10% of the incoming packets because of buffer overflows.

III. CONGESTION AVOIDANCE ALGORITHMS

There are various algorithms through which we can avoid congestion. These algorithms ensure that our packets are not lost.

A. Additive Increase/Multiplicative Decrease (AIMD)

The additive increase/multiplicative decrease (AIMD) algorithm is a feedback control algorithm best known for its use in TCP Congestion Avoidance. It remains constant at each step. It ensures packet conservation. It reaches steady state quickly. It has been shown that AIMD is a necessary congestion for TCP congestion control to be stable. Because the simple Congestion Control Mechanism involves timeouts that cause retransmissions, it is important that hosts have an

accurate timeout mechanism. Timeouts set as a function of average Round Trip Time (RTT) and standard deviation of RTT. However TCP hosts only sample Round Trip Time once per RTT using coarse grained clock [1].

Formula:

1. MaxWindow=Min(CongestionWindow, AdvertisedWindow).
2. EffectiveWindow=MaxWindow – (LastByteSent – LastByteAcked).
3. Increment=MSS x (MSS/CongestionWindow).
4. CongestionWindow=CongestionWindow + Increment.

Key terms:

- *Congestion Window*: It is a variable held by TCP source for each connection.
- *Advertised window*: It is a field in TCP Header.
- *Max window*: It is based on the result of minimum value of any of these windows (congestion window or advertised window).
- *Effective window*: It is the calculated by (Last byte sent is deducted from Last byte acked).

Each time a packet sent congestion window is increased by 1. Congestion window is incremented fractionally for each acknowledgement. Congestion window is not allowed below the size of single packet [7].

B. Slow Start

Slow start is one of the algorithms that TCP uses to control congestion inside the network. It is also known as exponential growth phase.

Algorithm:

Step 1. Declaration of a variable for Congestion Window named: **cwnd.** And a variable for storing bandwidth property, named: **ssthresh.**

Step 2. Initialize, cwnd by one (**cwnd=1**).

Step 3. Now, Increment in the value of cwnd as each successful Acknowledgement (ACK) received.

cwnd ← cwnd+1

Step 4. And then, change the value of cwnd exponentially on each Round Trip Time (RTT):

cwnd ← 2x cwnd

Step 5. Finally, enter CA (Congestion Avoidance), when,

Cwnd >= sstresh

During the exponential growth phase, slow start works by increasing the TCP congestion window each time that acknowledgement is received. It increases the window size each time a packet is received [2].

This happens until either the packet is lost or threshold value is reached. Once the threshold value has been reached TCP enters the congestion avoidance phase.

Initially the congestion window size (cwnd) of 1, 2 or 10 segments increases by 1 Segment Size (SS) for each ACK received. Since the receiver sends an ACK for every two segments, this doubles the window size each round trip time of the network. This procedure continues until the congestion window size (cwnd) reaches the size of the receivers advertised window or until a loss occurs. When a loss occurs half of the current cwnd is saved as a Slow Start Threshold (SSThresh) and slow start begins again from its initial cwnd.

C. Fast Retransmission

Fast Retransmission is an enhancement to TCP which reduces the time sender waits before retransmitting a lost segment.TCP may generate an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received. This duplicate ACK should not be delayed. The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected.

Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire.

D. Fast Recovery

After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed. This is the fast recovery algorithm. It is an improvement that allows high throughput under moderate congestion, especially for large windows [1].

The fast retransmit and fast recovery algorithms are usually implemented together as follows:

After receiving three duplicated ACK's in row:

1. Set ssthresh to the current send window.
2. Retransmit the missing segment.
3. Set cwnd = ssthresh+3.
4. Each time the same duplicated ACK arrives, set cwnd++. Transmit a new packet, if allowed by cwnd.
5. If a non-duplicated ACK arrives, then set cwnd = ssthresh and continue with a linear increase of the cwnd (Congestion Avoidance).

IV. VARIANTS OF TCP

A. TCP Tahoe:

TCP Tahoe was suggested by Van Jacobson. TCP Tahoe is one of the simplest one out of five variants. TCP Tahoe is based on the principle of protection of packets when connection is running [1]. It includes Slow Start and Fast Retransmission algorithms. Tahoe suggests that whenever a TCP connection start or re-start after a packet lost it should go through a procedure called "Slow Start" [2][6]. In this algorithm sender set the congestion window to 1 then for each ACK received it increase cwnd to 1. So in the first round trip time (RTT) we send 1 packet in the second we send 2 and in third we send [4][5].

Fast retransmission algorithm reduces the time sender has to wait for receiver's acknowledgement. When triple ACK's is received it will perform fast retransmission and will reduce congestion window to 1 and enters slow start phase [6].

B. TCP Reno:

TCP Reno introduced major improvements over Tahoe by changing the way in which it reacts to detecting a loss through duplicate acknowledgements [1]. TCP Tahoe deals with single packet loss whereas Reno is better with multiple packet loss within window of data. Reno introduces a mechanism called "Fast Recovery" which was activated after fast retransmit [5]. By using fast recovery the sender uses a cwnd that is half the size of the cwnd just before the loss. As such, this forces the TCP Reno to send fewer packets out until it knows that it's okay to send more [2]. Therefore, it has indeed reduced it utilization of the network.

C. TCP New-Reno:

New Reno modifies the Fast Retransmit and Fast Recovery. These modifications are intended to fix the Reno problems above and are wholly implemented in the server side [1].

A modification of Reno lead to New Reno which shows that Reno can be improved without the addition of SACK but still suffers without it. Here, the wait for a retransmit timer is eliminated when multiple packets are lost from a window.

New Reno is the same as Reno but with more intelligence during Fast Recovery. It utilises idea of partial ACK's: when there are multiple packet drops, the ACK's for the retransmitted packet will acknowledge some, but not all the segments send before the Fast Retransmit [5].

In New Reno, a partial ACK is taken as an indication of another lost packet and as such sender retransmits the first unacknowledged packet. Unlike Reno, partial ACK's don't take New Reno out of Fast Recovery [2]. In this way, it retransmits one packet per RTT until all the lost packets are retransmitted and avoids requiring multiple fast retransmits from a single window of data.

D. TCP SACK (Selective Acknowledgment):

Selective Acknowledgment (SACK) is a strategy which corrects the behaviour in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender needs to retransmit only the segments that have actually been lost. If the SACK option is available, the TCP sender always has the information to make intelligent decisions about which packets to retransmit and which packets not to retransmit during fast recovery [3][5].

In TCP SACK, when client request to the server and the server formulates a response broken into four TCP segments (packets). Each time a packet is sent server gets an acknowledgement (ACK). When a packet is sent and an acknowledgement is received this will complete one round trip time (RTT). In second response the packet is dropped somewhere on the network [5]. In TCP SACK client will request to the server until it will get the packet which is lost on the network.

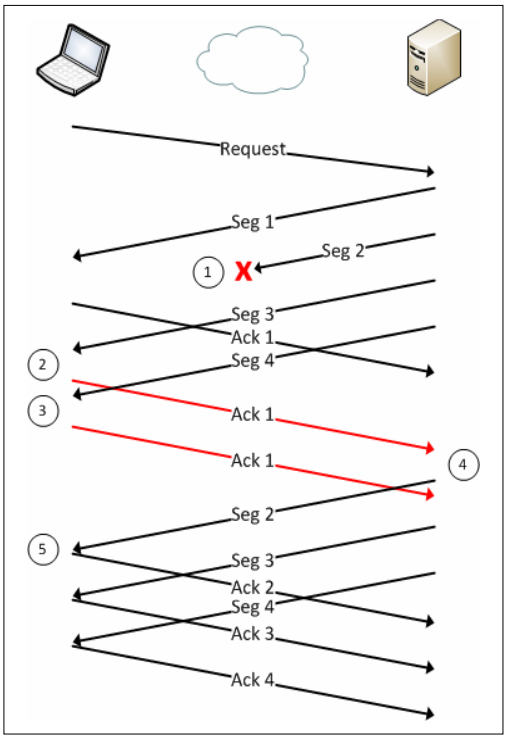

Figure1. Transmission between Two Nodes (Client and Server)

For example: A is a sender and B is a receiver. A sends packet to B from sequence number 101 to 201. After receiving all packets from A, B sends cumulative ACK of 202. Here ACK number 202 means B receives all the packets up to 201. In case if there is no cumulative acknowledgement and all the packets are received by the receiver but one ACK is lost. In that case even if the packet is successfully received by the receiver, the sender retransmits the packets which it didn't receive the ACK.

E. TCP Vegas:

TCP Vegas was introduced because there were some flaws in TCP Reno. It was suggested by Brakmo, O'Malley & Peterson in 1994. Vegas is an alternative implementation that interoperates with any other valid implementation of TCP, and all changes are confined to the sender side [1]. The three major algorithms that were introduced by Vegas are:

a. *Vegas - New retransmission mechanism*

Vegas also extends retransmission mechanism as follows: the system clock is read and recorded each time a segment is sent; when an ACK arrives, the clock is read again and the RTT calculation is done using this time and the timestamp recorded for the relevant segment [1]. Using this more accurate RTT, retransmission is decided as follows:

- When a dupack is received, Vegas checks to see if the new RTT (current time - timestamp recorded) is greater than RTO. If it's, Vegas retransmits the segment without having to wait for the third dupack. In many cases third dupack is never received, and therefore, Reno would have to rely on the coarse-grained timeout to catch the loss.
- When a non-dupack is received, if it is the first or second one after a retransmission, Vegas checks again to see if RTT > RTO; if so, then the segment is retransmitted. This will catch any other segment that may have lost previous to the retransmission without having to wait for a dupack. In other words, Vegas treats the receipt of certain ACKs as a trigger to check if a timeout should happen; but still contains Reno's coarse-grained timeout code in case these mechanism fail to recognize a lost segment [1].

b. *Vegas - New congestion avoidance mechanism*

Reno's congestion detection uses the loss of segments as a signal of congestion. It has no mechanism to detect the incipient stages of congestion - before losses occur - so they can be prevented. Reno is reactive, rather than proactive, in this respect [1]. Reno needs to create losses to find the available bandwidth of the connection.

c. *Vegas - Modified slow-start mechanism*

Reno's slow-start mechanism is very expensive in terms of losses; since it doubles the size of the congestion window every RTT while there are no losses - which is equivalent to doubling the attempted throughput every RTT - when it finally overruns the connection bandwidth, we can expect losses in the order of half the current congestion window, more if we encounter a burst from another connection. Vegas try to find a connection's available bandwidth that does not incur this kind of loss. Toward this end, the congestion detection mechanism is incorporated into slow-start with only minor modifications. To be able to detect and avoid congestion during slow-start, Vegas allows exponential growth only every other RTT. In between, the congestion window stays fixed so a valid comparison of the expected and actual rates can be made. When the actual rate falls down to expected rate by a certain amount. The γ threshold - Vegas changes from slow-start mode to linear increase/decrease mode [1][2].

TABLE I. DIFFERENT CONGESTION AVOIDANCE METHODS

Variants of TCP	**Name of Algorithm**
TCP Tahoe	Slow Start + Fast Retransmission
TCP Reno	Fast Retransmission+ Fast Recovery (In case of single packet loss)
TCP New Reno	Fast Retransmission+ Fast Recovery (In case of multiple packets lost)
TCP SACK	Fast Retransmission + Fast Recovery (In case of re-transmission of more than one lost packet)
TCP Vegas	New Re-transmission Mechanisms + Modified Slow Start + New Congestion Avoidance Mechanisms

V. CONCLUSION

In this paper, comparison of all TCP Congestion Avoidance algorithms is done. Basically, Congestion Avoidance is directly proportional to the performance of any network. After the study of previous literatures, the possibility of congestion avoidance is identified over TCP. As well as, this paper describes the working of Additive Increase or Multiplicative Decrease, Slow Start, Fast Retransmit, and Fast Recovery. The overall advantage of this research is to examine working principle of these algorithms.

REFERENCES:

[1] Yuvaraju B. N, Niranjan N Chiplunkar - "Scenario Based Performance Analysis of Variants of TCP using NS2-Simulator" International Journal of Advancements in Technology, ISSN 0976-4860 Vol 1, No 2(October 2010).

[2] "A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas".

[3] Dr. Neeraj Bhargava and Dr. Ritu Bhargava - "Implementation of TCP_Reno Algorithm in the ns2, International Journal of Engineering and Innovative Technology(IJEIT), ISSN: 2277-3754, ISO 9001:2008 Certified, Volume 2, Issue 2, August 2012.

[4] Minseok Kwon and Sonia Fahmy - "TCP Increase/Decrease Behavior with Explicit Congestion Notification (ECN)".

[5] Kevin Fall and Sally Floyd - "Simulation-Based Comparisons of Tahoe, Reno and SACK TCP.

[6] Jitender Sharma and Amit Kumar Garg - "Analysis of Tahoe: A TCP Variant", International Journal of Engineering and Advanced Technology(IJEAT), ISSN: 2249-8958, Volume-1, Issue-2, December 2011.

[7] Steven Carvellas and Jeril Jose - "TCP Tahoe with More Realistic Time Simulation and Packet Reordering" e-thesis The State University of New Jersey Rutgers (December 2010).

[8] Habibullah Jamal and Kiran Sultan - "Performance Analysis of TCP Congestion Control Algorithms" International Journal of Computer and Communications, Issue 1, Volume 2, 2008.

[9] Van Jacobson and Michael J. Karels - "Congestion Avoidance and Control" (November 1988).

[10] Thomas R. Henderson, Emile Sahouria, Steven McCanne, Randy H. Katz - "On Improving the Fairness of TCP Congestion Avoidance", Clobal Telecommunications Conference, 1998 IEEE.

[11] Deepak Bansal and Hari Balakrishnan - "TCP-Friendly Congestion Control for Real-time Streaming Applications", MIT Technical Report, MIT-LCS-TR-806, May 2000.

AUTHORS PROFILE

Dr. Neeraj Bhargava : Presently working as Associate Professor & Head, Department of Computer Science, School of Engineering & System Sciences, MDS University, Ajmer. He has more than 23 yr of experience for teaching in the University. His areas of interest are Spatial Database, Image Processing and Ad hoc Network.

Dr. Ritu Bhargava: Presently working as lecturer, Department of MCA, Govt. Women's Engineering College, Ajmer. She has more than 9 yr of experience for teaching MCA. Her areas of research are Web GIS, Wireless Network.

Manish Mathuria: Pursuing M.Tech (Information Technology) from Govt. Engineering College Ajmer, Rajasthan, India. His research areas are Digital Image Security, Fingerprint Recognition and Wireless Network.

Shilpi Gupta: Pursuing MCA and working with wireless network group. Her research areas are network protocols and sensors network.

Kamal Kumar Jyotiyana: Pursuing MCA and working with Ad hoc network group. His research areas are wireless sensor network.