VARIOUS TCP OPTIONS FOR CONGESTION EVASION

Ravi Kamboj , Gurpreet Singh

ABSTRACT

The Transport control protocol (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. TCP has undergone several transformations. Several proposals have been put forward to change the mechanisms of TCP congestion control to improve its performance. A line of re-search tends to reduce speed in the face of congestion. In this group we have various windows based congestion control algorithms that use the size of the congestion window to determine transmission speed. The aim of this study is to survey the various modifications of window based congestion control. Much work has been done on congestion avoidance hence specific attention is placed on the TCP variants such as Slow Start, TCP Fack, TCP Sack, TCP Asymmetry, TCP Reno, New Reno, and Rate Based Pacing in order to motivate a new direction of research in network utility and maximization. Many Researchers analyzed that, given a specific network topology and flow patterns there exists a TCP window size *at* which TCP achieves best throughput via improved spatial channel reuse. However, TCP does not operate around fixed window size, but typically grows its average window size much larger.

Keywords: Transmission control protocol, TCP variants as TCP Tahoe, TCP Reno, TCP New Reno, Slow Start, Fack, Sack, Rate Based Pacing, TCP Asymmetry.

I. INTRODUCTION

The TCP is a reliable connection-oriented stream protocol in the Internet Protocol suite. A TCP connection is like a virtual circuit between two computers, conceptually very much like a telephone connection. To maintain this virtual circuit, TCP at each end needs to store information on the current status of the connection e. g. The last byte sent.

Ravi Kamboj, *M.Tech Student, Department of Computer Science and Engineering, Yamuna Institute of Engineering and Technology, Gadholi.*
Gurpreet Singh*, Associate Professor, Department of Computer Science and Engineering, Yamuna Institute of Engineering and Technology, Gadholi.*

TCP is called connection-oriented because it Stand with a 3-way handshake and it maintains its state information for each connection.TCP is also called a stream protocol.TCP guarantees that it will deliver data supplied by the application to the other end, although the underlying Layers (IP) offer only unreliable data delivery. i.e, even if the data gets lost, corrupted or duplicated. Sometimes the data transmit from one end to the other or is delivered out of order by the communication system, then TCP is responsible for delivering error-free data in-order to the application at the other end. Hence, it is reliable. This reliability is achieved by assigning a sequence number to each byte of data that is transmitted. It is also required by the receiving TCP to send positive acknowledgments (ACKs) back to the data sender. This acknowledgment mentions the next byte of data expected by the receiver.
TCP sources break messages from higher protocol layers into datagram's that are encapsulated in packets which are then transmitted over the network. These packets are reassembled by the TCP receiver into the original message and passed on to the higher level protocol layers. For every packet sent on the network by a source an acknowledgement (ACK) is expected to be transmitted back from the destination. This ACK is used by the source to determine if the transmitted packet was successfully received at the destination. In this manner packets can be tracked and retransmitted if required.
To facilitate further discussion of TCP [4], the general features of TCP will be highlighted. We shall not describe a specific TCP variant here, and will provide an overview of the features of TCP variants such as TCP Tahoe, TCP Reno, New Reno, Sack and TCP Asymmetry.TCP Tahoe is one of the congestion control algorithms described that adds some new and enhance the earlier TCP implementations, including Slow Start, Congestion Avoidance and Fast Retransmit. It works on the packet conservation policy.

II. CONGESTION CONTROL

The congestion control algorithms have been included in the standards track TCP specified by the IETF [2][17]. Congestion is one of the biggest problems in TCP Network. We have various congestion control algorithms that are used to reduce the traffic on the network. The TCP sender uses a congestion window (cwnd) in regulating its transmission rate based on the feedback it gets from the network. The congestion window is the TCP sender's estimate of how much data can be outstanding in the network packets without being lost. After initializing cwnd to one or two segments, the TCP senders allowed to increase the congestion window either according to a slow start algorithm, that is, by one segment for each incoming acknowledgement (ACK), or according to congestion avoidance, at a rate of one segment in a round-trip time. The slow start threshold (ssthresh) is used to determine whether to use slow start or congestion avoidance algorithm.

The TCP congestion control [20] specifications artificially increase the congestion window during the fast recovery in order to allow forward transmissions to keep the number of outstanding segments stable. Therefore, the congestion window size does not actually reflect the number of segments allowed to be outstanding during the fast recovery. When fast recovery is over, the congestion window is deflated back to a proper size. Adjusting the congestion window consistently becomes an issue, especially when Sack information can be used by the TCP sender. By using the selective acknowledgements, the sender can derive the number of packets with a better accuracy than by just using the cumulative acknowledgments.

III. TCP VARIENTS

The core objective of all variants is the same, i.e. to maximize the throughput and efficiency of the TCP and difference between them is the way they operate. A number of the TCP variants have been introduced over the years [8]. The need of these variants is originated by inefficiencies and limitations in throughput of TCP protocol. The TCP variants are as:-

- TCP Tahoe
- TCP Reno
- TCP New Reno
- TCP Sack
- TCP Asymmetry
- TCP Rate Based Pacing
- Slow Start

(i)TCP TAHOE

TCP Tahoe involved a few new algorithms in early TCP implementations like Slow-Start, Congestion Avoidance, and Fast Retransmit. Among these the fast retransmit algorithm is of special interest as it has been retained in its basic form in subsequent versions of TCP. In Fast Retransmit, after receiving a small number of duplicate acknowledgments for the same TCP segment (*dup ACKs*), the data sender infers that the packet has been lost and retransmits the packet without waiting for a retransmission timer to expire. Generally, it has been seen that the duplicate acknowledgment threshold is fixed at three. Therefore, on receiving three successive duplicate acknowledgments the sender can infer that receiver has not received the packet, and a retransmission is triggered without waiting for a timeout.

Independence on the retransmission timeout for taking retransmission decisions of the lost packet and subsequent earlier loss recovery leads to higher channel utilization and connection throughput. The protocol returns to a slow start and sets the slow start threshold to one-half of the congestion window. This strategy does not change even if the number of packets dropped is more. Resetting of congestion window to one irrespective of degree of congestion on the network drastically reduces TCP flow has some implication on the network performance and adversely affects it. For connections with a larger congestion window, the delay in slow-start because of the logic of setting the slow start threshold to half the previous congestion window, can have a significant impact on the overall performance of a Tahoe connection. The TCP Tahoe algorithm is given below:

```
*/Tahoe Agorithm*/
if (cwnd<ssthresh)
        cwnd = cwnd + 1            # Slow start algorithm
else (cwnd>= sstresh)
        cwnd = cwnd + 1/cwnd       # Congestion Avoidance
```

(ii) SLOW START

Congestion Avoidance and Slow Start [17] are independent algorithms with different objectives but in practice they are implemented together. A slow - start algorithm is used to control congestion inside the network. It is also known as the exponential growth phase. During the exponential growth phase, slow-start works by increasing the TCP congestion window each time the acknowledgment is received. It increases the window size by the number of segments acknowledged. This happens until either an acknowledgment is not received for some segment or a predetermined threshold value is reached. If a loss event occurs, TCP assumes that it is due to network congestion and takes steps to reduce the offered load on the network. Slow Start probes the network so that the TCP source can get an initial indication of the network bandwidth available. Congestion Avoidance more gently probes the network so that the TCP source can adapt to changing network conditions. A TCP connection will start in Slow Start mode, but switch to Congestion Avoidance mode after *cwnd* reaches the value of *ssthresh*. In addition to these enhancements Tahoe also includes Fast Retransmit, better RTT variance estimation, and an exponential retransmit timer back-off. The slow start algorithm is given below:-

```
*/Slow Start Algorithm*/
If
        cwnd <= ssthresh
then
        /* Slow Start Phase */
        Each time an ACK is received:
        cwnd = cwnd
        + segsize
else /*
        cwnd > ssthresh
        */
        /* Congestion Avoidance Phase */
        Each time an ACK is received:
        cwnd = cwnd + segsize * segsize / cwnd + segsize / 8
endif
        segsize = MSS
```

(iii) TCP RENO

This variant of TCP is similar to the Tahoe TCP, except it also includes Fast Recovery. Reno TCP [12] does not return to Slow Start after Fast Recovery (which ends on the receipt of the retransmitted packet), instead it reduces the congestion window to half the current window size. TCP Tahoe and Reno experience poor performance when multiple packets are lost from one window (cwnd) of data. TCP Reno is effective to recover from a single packet loss, but it still suffers from performance problems when multiple packets. With the limited information available from cumulative acknowledgments, a TCP source can only learn about a single lost packet per round trip time. An aggressive source could choose to retransmit packets early, but such retransmitted packets may have already been successfully received.
In Reno [20] the sender' s *usable* window becomes where is the receiver' s advertised window, is the sender' s congestion window, and is maintained at until the number of dump ACKs reaches tcprexmtthresh and thereafter tracks the number of duplicate ACKs. Thus, during Fast Recovery the sender "inflates" its window by the number of dump ACKs it has received according to the observation that each dup ACK indicates some packet has been removed from the network and is now cached at the receiver. After entering Fast Recovery and retransmitting a single packet, the sender effectively waits until half a window of dup ACKs have been received, and then sends a new packet for each additional dup ACK that is received. Upon receipt of an ACK for new data (called a "recovery ACK"). The TCP Reno algorithm is given below:-

```
*/TCP Reno Algorithm */
if
        (cwnd<ssthresh)
        cwnd = cwnd + 1 # slow start
else if
        (cwnd>= ssthresh)
        cwnd = cwnd + 1/cwnd# congestion avoidance
if
        (duplicate ACK)
If
        (duplicate ACK = = (1 || 2) )
        cwnd = ssthresh #packet delayed/ out-of-packet
received
        ssthresh = cwnd/2
else
        (duplicate ACK > 2)
        cwnd = cwnd + Number (ACK) # packet loss due to
congestion
        ssthresh = cwnd/2
```

(iv) TCP NEW RENO

The New-Reno TCP [9][12] in this paper includes a small change to the Reno algorithm at the sender that eliminates Reno's wait for a retransmit timer when multiple packets are lost from a window [Hoe95, CH95]. The change concerns the sender' s behavior during Fast Recovery when a *partial ACK* is received that acknowledges some ,but not all of the packets that were outstanding at the start of that Fast Recovery period. In Reno, partial ACKs take TCP out of Fast Recovery by "deflating" the usable window back to the size of the congestion window. In New-Reno, partial ACKs do not take TCP out of Fast Recovery. Instead, partial ACKs received during Fast Recovery are treated as an indication that the packet immediately following the acknowledged Packet in the sequence space has been lost, and should be retransmitted. Thus, when multiple packets are lost from a single window of data, New-Reno can recover without a retransmission timeout by retransmitting one lost packet per round-trip time until all of the lost packets from that window have been retransmitted.
New-Reno remains in Fast Recovery until all of the data outstanding when Fast Recovery was initiated has been acknowledged. Address this issue by modifying the action taken when receiving new 4Note that cwnd represents the number of packets in flight for the flow until a packet(s) is dropped ACK's. In order to exit the Fast Recovery, the source must receive an ACK for the highest sequence number sent before entering Fast Recovery. Thus, unlike TCP Reno, New "partial ACK's" (those which represent new ACK's but do not represent an ACK of all outstanding data) do not take TCP New Reno out of Fast Recovery. In this way, Reno retransmits one packet per RTT until all lost packets are retransmitted. Although TCP New Reno addresses the issue of multiple drops within a window of data, it does not use all the information on dropped packets available at the receiver. It takes a full round trip for a sender to learn about each lost packet .When multiple segments are lost this inefficiency can reduce throughput substantially. This is of particular importance on high bandwidth-delay product paths as many of

the proposed TCP algorithms for such paths are more aggressive than standard TCP and frequently generate multiple packet losses during normal operation. Improving performance with multiple drops is addressed by the Selective Acknowledgment (SACK) mechanism.

(v) TCP SACK

(Selective repeat retransmission policy) Sack is an extension to TCP intended to improve TCPs handling of the multiple packet loss case. When the sender and receiver support Sack, the receiver transmits TCP ACK packets that include a list of up to four ranges of packet sequence numbers. These ranges specify packets that have been successfully received and the gaps between these ranges can be used by the sending machine to infer which packets are likely to have been lost and therefore should be retransmitted. This has the benefit of allowing the source to intelligently retransmit packets and react more efficiently to multiple dropped packets.

In more detail, the Sack data in a TCP ACK consists of the start and stop packet sequence numbers of up to four non-overlapping blocks of packets.

The Sack blocks are not necessarily listed in sequential order as the first Sack block is required to include the sequence number of the most recently received packet. The receiver is not guaranteed to keep the packet that it Sack in previous ACKs and it may do so if facing a memory pressure. The sender can't drop the packets that were Sack since it might need to retransmit them anyway if the receiver did drop the packets eventually.

(vi) TCP FACK

FACK is short name for **Forward Acknowledgment** [12] and is based on TCP Reno with Sack. Fack is using the information provided by Sack to compute a better estimate of the amount of data currently in transit (outstanding data). This information is essential for any congestion control algorithm. To estimate the amount of outstanding data, Fack introduces a new variable, fack, denoting the highest senior known to have been received plus 1. The variables *next* and una represent the first byte of data yet to be sent and the first unacknowledged byte, respectively. The window starts from una and ends with *next*. This means that some blocks from una to fack have been acknowledged, but not all of them. Blocks that have not been acknowledged by SACK are still outstanding, i.e., they are retransmitted but not acknowledged. Thus, the amount of data currently in transit is next − fack + retransmitted. However, Fack might in some cases underestimate the amount of outstanding data. Additionally, Fack addresses another unfortunate side effect experienced by Reno. When halving the window, there is a pause in the sender's transmission until enough data has left the network. This pause is reflected in the ACKs and leads to the transmission of a group of segments at the start of each RTT. This uneven distribution can be avoided by a gradual reduction of the window, something Fack does. This allows congestion to go away and thereby reduces the probability of a double loss.

(vii) RATE BASED PACING

Prior work has suggested that this "slow-start restart" problem is a contributor to poor performance of P-HTTP over TCP .One way of solving the problem is to send segments at a certain pace until we get the ACK clock running again. This pace or rate should be based on a fraction of prior estimates of data transfer rate, since that is the closest estimate of available bandwidth that we have (if we had some magical way of knowing the exact available bandwidth at the end of the idle time, we could have used that). We believe that this modification, called Rate Based Pacing (RBP), will give better performance for the circumstances mentioned in the problem.

RBP IMPLEMENTATION:-

Rate based pacing requires the following changes to TCP:

1. Bandwidth estimation.
2. Calculation of the window that we expect to send in RBP and the timing between segments in that window.
3. A mechanism that clocks the segments sent in RBP.

Idle time detection is done by some TCP implementations. Instead of forcing slow start upon detection of idle time, we modify the behavior to RBP. TCP Vegas gives us a method for bandwidth estimation.

(viii) TCP ASYMTERY

TCP network asymmetry can take several forms. We present a brief classification to describe the structure:

• **Bandwidth asymmetry**: In the network asymmetry typically the downstream bandwidth is 10-1000X the upstream bandwidth. Examples include cable modem, ADSL, and satellite-based networks, especially in configurations that depend on a dialup link for upstream connectivity.

• **Media-access asymmetry:** This manifests itself in several ways. In a cellular wireless network, a centralized base station incurs a lower medium access control (MAC) overhead in transmitting to distribute mobile hosts than the other way around. In a packet radio network, e.g. the Maricom Ricochet™ network), the MAC overhead makes it more expensive to constantly switch the direction of transmission than to transmit steadily in one direction.

• **Loss rate asymmetry:** The network may inherently be more loss in one direction than in the other.

CONCLUSION

The objective of this paper is to investigate some of the issues arising in the design of TCP congestion control algorithms. In particular, we consider the fairness behavior between competing flows in networks where packet drops are unsynchronized that is, where not every flow sees a packet loss at each network congestion event. We review some of the main proposals for TCP congestion control in high bandwidth-delay product networks. We analyze various problems such as lower throughput, higher delay, large congestion at the network. We governed by the packet conservation principle and the sender's estimate of which packets are still in the network, which are acknowledged, and which are declared lost. Whether to retransmit or transmit new data depends on the markings made in the TCP sender's scoreboard. In most of the cases, none of the requirements given by the IETF are violated, although in marginal scenarios the detailed behavior may be different from what is given in the IETF specifications. However, the TCP essentials, in particular the congestion control principles and conservation of packets, are maintained in all cases.

REFRENCES

[1] Stevens, W., TCP/IP Illustrated, Volume 1: The Procols Addison- Wesley, 1994.

[2] J. Postel, "Transmission Control Protocol", RFC 793, September, 1981.

[3] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[4] Gurpreet Singh, Dinesh Kumar, Amanpreet Kaur, "Simulation based comparison of performance metrics for various TCP extensions using NS-2", Proceedings of IEEE sponsored International Conference on Innovative Technologies (ICIT-09), pages 106-110, 18-19 June, 2009.

[5] M. Mathis, J Mahdan, S. Floyd, A. Roma Now "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[6] V. Jacobson, "Congestion Avoidance and Control", In Proc. ACM SIGCOMM 88, Vol 118, No. 4, pages 157-173,August 1988.

[7] Haining Wang, Hongjie Xin, Reeves D.S, Shin K.G, "A simple refinement of slow-start of TCP congestion control", Computers and Communications, pages. 98-105, 2000.

[8] H. Balakrishnan,V. N. Padmanabhan, and R.H. Katz. "The Effects of Asymmetry on TCP Performance", In Proc. ACM MOBICOM '97, September 1997.

[9] Fall, K. And Floyd, S. 1996. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. ACM SIGCOMM Computer Communication Review, Volume 26, Iissue 3, pages 5-21, July 1996.

[10] T. V. Lakshman, U. Madhow, B. Suter, "Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A Study of TCP/IP Performance", In Proc. IEEE Infocom, Kobe, Japan, Vol 3, Pages 1199-1209, 7-12 April 1997.

[11] D. Borman, R. Braden, and V. Jacobson, "TCP Extensions for High Performance", RFC 1323, May 1992.

[12] Amanpreet Kaur, Gurpreet Singh, Deepti Chauhan, "Radical Analysis of TCP Alternatives: Tahoe, New Reno, Sack, Vegas on the basis of imitation in NS-2", in the proceedings of Journal of Yamuna Journal of Technology and Management, Vol 1, Issue 1, Nov, 2011.

[13] Rayadurgam Srikant, "TCP-Vegas The mathematics of Internet congestion control (Systems and Control: Foundations and Applications)", Book by Springer Science and Business Media, January 09-12-2010.

[14] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm" RFC 2582, Apr. 1999.

[15] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgment (SACK) Option for TCP", RFC 2883, July 2000.

[16] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, Jan. 1984.

[17] Neha Batla, Amanpreet Kaur, Gurpreet Singh, "Congestion Control Techniques in TCP: A Critique", in the proceedings of 3rd National Conference on Advances and Research in Technology (ART-2014), Pages 45.1-45.5, 8-9 March, 2014.

[18] D.D.Clark, "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.

[19] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.

[20] Amanpreet Kaur, Gurpreet Singh, "Weighing The Various Flavors Of Congestion Control Protocol For Tcp-Practical Surveillance In Ns-2", In the proceedings of National Conference on Futuristic Trends in Computing, Communication & Information

System, FTTCCIS and Yamuna Journal of Technology & Business Research - Volume 3. Issue (1-2), pages 2.1-2.6, July 12-13, 2013.

[21] Appenzeller,G, Keslassy, I, and Mckeown N, “Sizing router buffers”, In Proceedings conference on Applications, technologies, architectures, and protocols for computer communications and in newsletter of ACM SIGCOMM Computer Communication Review Volume 34 Issue 4, pages 281-292, October 2004.