International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 1, ISSN: 2249 –071X

Improving Packet Delivery Ratio In TCP Using New Reno Scheme

S.Kanthimathi
gayathri.mathi20@gmail.com

Abstract **- In today's Internet, several variants of TCP are deployed. These variants differ with respect to their congestion control and segment loss recovery techniques. There are schemes like TCP Reno, TCP NewReno to overcome the congestion problem. On-demand ad hoc routing protocols respond to network events such as channel noise, mobility, and congestion in the same manner, which, in association with TCP, deteriorates the quality of an existing end-to-end connection. The poor end-to-end connectivity deteriorates TCP's performance in turn. So, to address these problems, two complementary mechanisms are proposed, that is, the TCP fractional window increment (FeW) scheme and the Route-failure nOtification using BUlk-losS Trigger (ROBUST) policy. These two mechanisms result in a significant improvement of TCP throughput without modifying the basic TCP window or the wireless MAC mechanisms.**

Keywords **- Few, On demand Routing, Robust, Throughput**

I. INTRODUCTION

Transmission Control Protocol[2] provides a connection oriented, reliable, byte stream service.

The term connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data. It is a full duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction. [10].

TCP includes a flow-control mechanism for each of these byte streams that allows the receiver to limit how much data the sender can transmit. 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. Therefore congestion results in packet loss.

Congestion avoidance [7],[11],[24] is a way to deal with lost packets. The Transmission Control Protocol (TCP) was proposed and implemented to prevent the future congestion collapses. Then, it has gone through several phases of improvement, and many new features such as fast retransmit and fast recovery have been added[1].

A new version of TCP called TCP Vegas, which achieves higher efficiency and causes much fewer packet retransmissions, and is not biased against the connections with longer round trip times[19]. TCP Vegas is fundamentally different congestion avoidance scheme from that of TCP Reno. TCP Vegas uses a novel congestion control mechanism that attempts to detect congestion in the network before segment loss occurs.

TCP-Vegas achieves 37 to 71 percent higher throughput than TCP Reno and also shown that TCP Vegas does achieve higher efficiency than TCP Reno and causes much less packet retransmissions. Another important issue is the stability of TCP Vegas. Since each TCP Vegas connection attempts to keep a few packets in the network, when their estimation of the propagation delay is off, this could lead the connections to inadvertently keep many more packets in the network, causing a persistent congestion. TCP Vegas[6],[18] however, is not widely deployed on the Internet today.

TCP Reno uses the loss of packets as a signal that there is congestion in the network and has no way of detecting any incipient congestion before packet losses occur. Thus, TCP Reno reacts to congestion rather than attempts to prevent the congestion[4]. TCP Vegas, on the other hand, use the difference between the estimated throughput and the measured throughput as a way of estimating the congestion state of the network. When multiple packets are dropped, Reno has problems.

TCP NewReno introduced an improved fast recovery algorithm that can recover from multiple losses in a single window of data, avoiding many of the retransmission timeout events that Reno experiences. New Reno can deal with multiple lost segments without going to slow start. It uses the concept of Partial ACK when multiple packets are lost. Partial ACK does not take sender out of fast recovery. Partial ACK causes retransmission of the segment following the acknowledged segment.

TCP NewReno (unlike Reno) distinguishes between a "partial" ACK and a "full" ACK. A full ACK acknowledges all segments that were outstanding at the start of fast recovery, while a partial ACK acknowledges some but not all of this outstanding data[3].

Unlike Reno, where a partial ACK terminates fast recovery, NewReno retransmits the segment next in sequence based on the partial ACK, and reduces the congestion window by one less than the number of segments acknowledged by the partial ACK. This window reduction, referred to as *partial window deflation*, allows the sender to transmit new segments in subsequent RTTs of fast recovery. On receiving a full ACK, the sender sets *cwnd* to *ssthresh*, terminates fast recovery[12], and resumes congestion avoidance.

TCP New Reno improves retransmission during the fast recovery phase of TCP Reno. During fast recovery, for every duplicate ACK that is returned to TCP New Reno, a new unsent packet from the end of the congestion window is sent, to keep the transmit window full. For every ACK that makes partial progress in the sequence space, the sender assumes that the ACK points to a new hole, and the next packet beyond the ACKed sequence number is sent.

A. NewReno fast recovery algorithm

This section presents an overview of New Reno's improved fast recovery algorithm. All other congestion control components of NewReno, namely slow start, congestion avoidance, and fast retransmit, are identical to that of Reno. During congestion avoidance, receipt of four back-to-back identical ACKs (referred to as "triple duplicate ACKs") causes the sender to perform fast retransmit and to enter fast recovery.

In fast retransmit, the sender does the following:

1) Retransmits the lost segment;
2) sets the slow start threshold *ssthresh* to *cwnd* (where *cwnd* is the current congestion window size); and
3) sets *cwnd* to *ssthresh* (new) plus 3 segments.

In fast recovery (FR), the sender continues to increase the congestion window by one segment for each subsequent duplicate ACK received. The intuition behind the fast recovery algorithm is that these duplicate ACKs indicate that some segments are reaching the destination, and thus can be used to trigger new segment transmissions. The sender can transmit new segments if permitted by its congestion window [13].

B. Assumptions

This section outlines our assumptions regarding the application, the sender/receiver, and the network.

1) Application Layer

The model focuses on the steady-state throughput for *TCP bulk transfers* [14],[15] . An application process that has an infinite amount of data to send from a source node to a destination node is considered.

2) TCP Sender and Receiver

The model assumes that the sender is using the TCP NewReno congestion control algorithm. The sender always transmits full-sized (i.e., MSS) segments whenever the congestion window allows it to do so. It is assumed that the sender is constrained only by the congestion window size, and not by the receiver's buffer size or advertised window.

3) Latency Model

The latency of the TCP transfer is measured in terms of "rounds". The first round begins with the start of congestion avoidance; its duration is one RTT[5]. All other rounds begin immediately after the previous round, and also last one RTT. The only exception is the round that terminates fast recovery and switches to congestion avoidance: its duration could be shorter than one RTT.

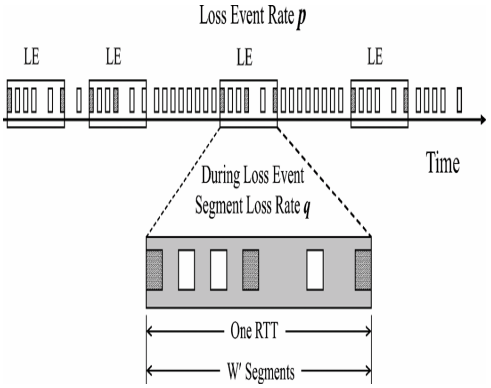

Fig. 1. The two-parameter segment loss Model

4) Loss Model

Fig. 1.explains the work introduces a novel two-parameter segment loss model that captures both the frequency of loss events and the burstiness of segment losses within a loss event. A loss event (LE) begins with the first segment loss in a round that eventually causes TCP to transition from the congestion avoidance phase to either the fast recovery phase or the timeout phase[22]

II. THE ANALYTIC MODEL

This section develops the stochastic throughput model for TCP NewReno bulk data transfer. The model is developed in two steps namely NoTo model and Full model.

A. Model Without Timeout (NoTO)

The evolution of the congestion window can be viewed as a concatenation of statistically identical *cycles*, where each cycle consists of a congestion avoidance period, followed by detection of segment loss and a fast recovery period. Each of these cycles is called a congestion avoidance/fast recovery (CAFR)[16] period. The throughput of the flow can be computed by analyzing one such CAFR cycle. The term *drop window is* referred to the window's worth of segments starting from the first lost segment, the segment transmitted just before the receipt of the first duplicate ACK.

B. Full Model (Full)

This section extends the foregoing model to include timeouts as loss indications. It is referred as the "Full" model. Again the congestion window evolution is viewed as a concatenation of statistically identical cycles. Each cycle consists of several CAFR periods followed by a CATOSS period, where a CATOSS period is the concatenation of congestion avoidance (CA), timeout (TO), and slow start (SS) periods. TCP NewReno may experience a timeout either from the congestion avoidance phase or from the fast recovery phase. The former transition occurs when TCP does not receive enough duplicate ACKs to trigger fast retransmit/fast recovery, while the latter transition occurs when retransmitted segments are lost during the fast recovery phase.

III. TCP PERFORMANCE

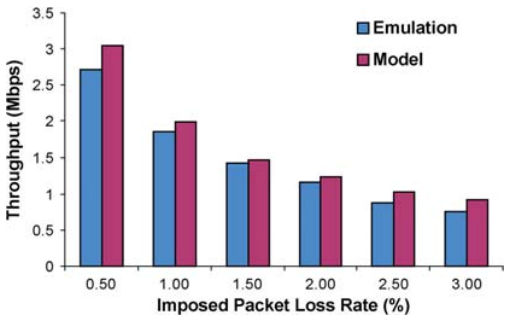

Fig. 2. Model accuracy in TCP

The above figure gives a description about TCP performance with respect to throughput and packet loss. The TCP performance is degraded due to channel noise and low bandwidth delay[20]. So, the following two schemes are defined to improve the throughput performance of TCP New Reno. The TCP FEW scheme is a preventive solution used to reduce the congestion-driven wireless link loss. The ROBUST policy is a corrective solution that enables on-demand routing protocols to suppress overreactions induced by the aggressive TCP behavior[23].

A. TCP: Fractional Window Increment

Figure 3 explains the block diagram of TCP data transmission. Two proper values namely K and α is chosen to control the TCP operation range while preserving the basic TCP window mechanism. K = 0 and α =1 provide the upper bound for the shifted TCP operation range. Thus, the TCP congestion window with factor $0 < \alpha < 1$ has a lower dynamic range with the same loss rate p, or the TCP loss rate with factor $0 < \alpha < 1$ has a lower value for the same window value W.

Based on the above discussion, a new TCP regulation scheme is proposed that allows the TCP congestion window to grow by a fractional rate $\alpha < 1$ (packets) at every roundtrip time. This is equivalent to adding one packet to the window size at every 1/ α round-trip time. Suppose that the current congestion window size is W, the TCP sender sends W packets at every round-trip time and receives W ACKs during one round-trip time from the TCP receiver[19]

B. On Demand Routing

The ROBUST[10] policy in this section is a simple link loss reaction policy for on-demand routing to improve the path robustness against the MAC contention loss driven by congestion. The typical implementation of on-demand ad hoc routing protocols takes a route maintenance/ rediscovery action immediately upon detecting any wireless link loss. However, in IEEE 802.11 multihop networks, the link loss has to be treated differently according to its cause[17].

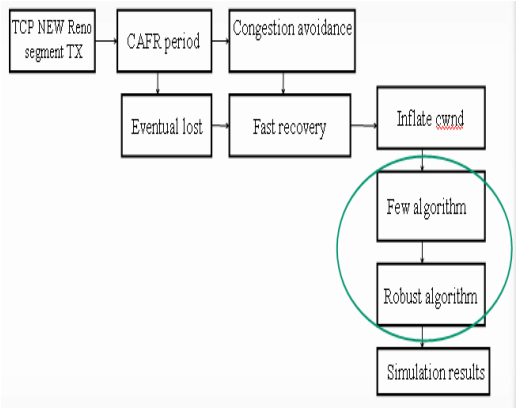

Fig 3: TCP Block Diagram

Node mobility

When a node is no longer available, the current route should be invalidated, and a new route should be established regardless of its cost.

Congestion:

The congestion-driven loss is a sign of network overload, indispensable for a seamless TCP operation. Nevertheless, the existing route should not be affected.

Channel noise:.

Noisy routes are tolerable in the sense that channel noise affects only the quality of the end-to-end connection, not the connectivity itself. A noisy path should be tolerated until a better alternative path is found[8]If needed, a hop-by-hop retransmission can improve the connection quality.

To account for the link failure tolerability, a generalized link failure sensitivity parameter, ß is introduced, for on demand routing protocols to control connection stability. The number of successive link failures L is counted, and a routing protocol takes action for routing maintenance only when L exceeds a certain threshold, ß, which indicates the limit of tolerable successive link failures.

1. If the transmission fails and L < ß, the current packet is dropped, and the transmission is resumed for the next packet. The current route is kept intact, and the value of L is increased by 1.
2. If the transmission fails and L = ß, the routing agent responds to the link failure and sets L = 0. The routing agent is free to establish a new route and perform necessary routing maintenance operations.
3. Whenever a transmission is completed successfully, the routing agent resets
   L = 0.

IV. CONCLUSION

Improving TCP performance is a complicated cross-layer problem. In future, TCP FEW performance can be varied depending on the network condition (for example, whether the routing is robust or not). TCP FEW can be improved by tuning factor α properly to meet the performance requirement for a given network condition.

REFERENCES

[1] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," in *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 231–242.

[2] M. Arlitt and C. Williamson, "Internet web servers: Workload characterization and performance implications," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[3] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms,"in *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001, pp. 631–640.

[4] L. Brakmo, S. O'Malley, and L. Peterson, "TCPVegas: Newtechniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, New York, Aug. 1994, pp. 24–35.

[5] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000, pp. 1742–1751.

[6] K. Fall and S. Floyd, "Simulation-based comparisons of vegas,Tahoe, Reno, and Sack TCP," *ACM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, Jul. 1996.

[7] V. Firoiu and M. Borden, "A study of active queue management for congestion control," in *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000, vol. 3, pp. 1435–1444.

[8] S. Floyd, "Connections with multiple congested gateways in packet switched networks," *ACM Comput. Comm. Rev.*, vol. 21, no. 5, pp. 30–47, 1997.

[9] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACMTrans. Netw.*, vol. 7, no. 4, pp. 458–472, Aug. 1999.

[10] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithms for increasing the robustness of RED's active queue management," Tech. Rep., Aug. 2001.

[11] S. Floyd, M. Handley, J. Padhye, and J.Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 43–56.

[12] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," RFC 3782, Apr. 2004.

[13] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetw.: Res. Exper.*, vol. 3, no. 3, pp. 115–156, Sep. 1992.

[14] S. Floyd and E. Kohler, "Internet research needs better models," *ACM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 29–34, Jan. 2003.

[15] M. Goyal, R. Guerin, and R. Rajan, "Predicting TCP throughput from non-invasive network sampling," in *Proc. IEEE INFOCOM*, Hiroshima, Japan, Mar. 2002, vol. 1, pp. 180–189.

[16] Q. He, C. Dovrolis, and M. Ammar, "On the predictability of large transfer TCP throughput," in *Proc. ACMSIGCOMM*, Philadelphia, PA, Aug. 2005, pp. 145–156.

[17] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1988, pp. 314–329.

[18] V. Jacobson, "Berkeley TCP evolution from 4.3-Tahoe to 4.3 Reno," in *Proc. 18th IETF*, Vancouver, Canada, Aug. 1990.

[19] H. Jiang and C. Dovorolis, "Passive estimation of TCP round-trip times," *ACM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 75–88, Jul. 2002.

[20] D.Kosti, A. Rodriguez, J. Albrecht, and A.Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proc. ACMSOSP*, Bolton Landing, NY, Oct. 2003, pp. 282–297.

[21] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link," *IEEE/ACM Trans. Netw.*, vol. 6, no. 4, pp. 485–498, Aug. 1998.

[22] T. Lakshman and U. Madhow, "The performance of TCP/IP fornetworks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. Netw.*, vol. 5, no. 3, pp. 336–350, Jun. 1997.

[23] A. Mahanti, D. Eager, and M.Vernon, "Improving multirate congestion control using a TCP Vegas throughput model," *Comput. Netw.*, vol. 48, no. 2, pp. 113–136, Jun. 2005.

[24] M. Mathis, J. Semke, , J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997.

AUTHOR'S PROFILE

S.Kanthimathi
gayathri.mathi20@gmail.com

Copyright © 2012 IJECCE, All right reserved