Performance Analysis of TCP Congestion Control Algorithms

Habibullah Jamal, Kiran Sultan

***Abstract*—** The demand for fast transfer of large volumes of data, and the deployment of the network infrastructures is ever increasing. However, the dominant transport protocol of today, TCP, does not meet this demand because it favors reliability over timeliness and fails to fully utilize the network capacity due to limitations of its conservative congestion control algorithm. The slow response of TCP in fast long distance networks leaves sizeable unused bandwidth in such networks. A large variety of TCP variants have been proposed to improve the connection's throughput by adopting more aggressive congestion control algorithms. Some of the flavors of TCP congestion control are loss-based, high-speed TCP congestion control algorithms that uses packet losses as an indication of congestion; delay-based TCP congestion control that emphasizes packet delay rather than packet loss as a signal to determine the rate at which to send packets. Some efforts combine the features of loss-based and delay-based algorithms to achieve fair bandwidth allocation and fairness among flows. A comparative analysis between different flavors of TCP congestion control namely Standard TCP congestion control (TCP Reno), loss-based TCP congestion control (HighSpeed TCP, Scalable TCP, CUBIC TCP), delay-based TCP congestion control (TCP Vegas) and mixed loss-delay based TCP congestion control (Compound TCP) is presented here in terns of congestion window verses elapsed time after the connection is established.

Key words - *Congestion control, High-speed networks, TCP.*

I. INTRODUCTION

Moving bulk data quickly over high-speed data network is a requirement for many applications. These applications require high-bandwidth links between network nodes. To maintain the stability of Internet all applications should be subjected to congestion control. TCP [9] is well-developed, extensively used and widely available Internet transport protocol. TCP is fast, efficient and responsive to network congestion conditions but one objection to using TCP congestion control is that TCP's AIMD congestion back-off algorithm [1] is too abrupt in decreasing the window size, thus it hurts the data rate.

Habibullah Jamal, Professor, Department of Electrical Engineering, University of Engineering and Technology Taxila, Pakistan and also the Vice Chancellor of the university. Phone: 92-51-9047401, Fax: 92-51-9047420; E-mail: drhjamal@uettaxila.edu.pk.
Kiran Sultan, Assistant Professor, Air University, Islamabad, Pakistan. E-mail: kiransultan@mail.au.edu.pk

Standard TCP constraints the congestion window that can be achieved in realistic environments. In the past few years, we have witnessed a surge of TCP variants that address the under-utilization problem most notably due to the slow growth of TCP congestion window that makes TCP unfavorable for high BDP networks. The rest of the paper is organized as follows: Related works, including TCP modifications and new protocols are reviewed in section 2. Standard TCP congestion control algorithm is described in section 3. Three prominent window-based high-speed TCP congestion control algorithms that use packet-loss as an implicit indication of congestion are described in section 4. TCP Vegas, a delay-based TCP congestion control is explained in section 5. Compound TCP approach is described in section 6. The comparative analysis of these algorithms in terms of congestion window growth function verses elapsed time for different scenarios of two network topologies is presented in section 7. Finally, this work is concluded in section 8.

II. BACKGROUND AND RELATED WORK

The standard TCP congestion control algorithm which we refer to as TCP Reno [1] was developed in 1988. [4], [5], [6], [7], [8], [10], and [15] explain several enhancements in TCP Reno. Few modifications addressing the conservative approach of TCP to update its congestion window under congestion condition are:

i. **Loss-based TCP congestion control**: HSTCP [11], BIC-TCP [12], STCP [13], CUBIC-TCP [16], HTCP [17] etc.
ii. **Delay-based congestion control**: TCP-Vegas [22], Fast-TCP [3], TCP-LP [18] etc.
iii. **Mixed loss-delay based TCP congestion control**: Compound TCP [19], TCP Africa [20] etc.
iv. **Explicit congestion Notification**: XCP[21] etc

Most of these protocols deal with modifying the window growth function of TCP in a more scalable fashion. Tomoya *et. al*. [2] proposed a TCP-friendly congestion control that realizes efficient data transmission in high-speed networks, fairness with TCP Reno and fair bandwidth allocation among flows with different RTTs. A scheme that determines the size of congestion window each time a new acknowledgment is received instead of employing slow start/congestion avoidance approach is proposed in [14].

III. TCP Reno

TCP Reno implements the TCP's AIMD mechanism of increasing the congestion window W by one segment per round-trip time for each received ACK and halving the congestion window for each loss event per round-trip time. TCP Reno controls the congestion window as follows:

$$\textit{Increase}: \quad W = W + \frac{1}{W} \tag{1}$$

$$\textit{Decrease}: \quad W = W - \frac{1}{2}W \tag{2}$$

When the link bandwidth does not change, TCP Reno periodically repeats the window increase and decrease. TCP Reno's congestion window in terms of packet loss rate (p) is defined as:

$$W_{reno} = \frac{1.22}{p^{0.5}} \tag{3}$$

As shown in equation (3), TCP Reno places a serious constraint on the congestion window that can be achieved by TCP in realistic environments. For example, for a TCP Reno connection with 1500-byte packets and 100ms RTT, achieving a steady-state throughput of 1Gbps would require an average congestion window of 8300 segments, and an average packet loss rate of 2×10^{-8}. This requirement is unrealistic in current networks. The congestion window takes more than 4000 RTT to recover after a loss event which prevents efficient use of the link bandwidth. TCP requires extremely small packet loss rate to sustain a large window which is not possible in real life networks.

IV. HIGH-SPEED TCP VARIANTS

Although TCP performs very well in low to middle speed networks (Kbps to several Mbps), it has very poor performance in high (tens of Mbps to Gbps) to very high (Gbps to Tbps) speed networks, as TCP is very inefficient in utilizing the high-speed network bandwidth.

HighSpeed TCP

HighSpeed TCP (HSTCP) [11] is a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows. HighSpeed TCP's modified response function only takes effect with higher congestion windows, it does not modify TCP behavior in environments with heavy congestion, and therefore does not introduce any new dangers of congestion collapse. HSTCP uses three parameters, W_L, W_H, and P_H. To ensure TCP compatibility, HSTCP uses the same response function as TCP Reno when the current congestion window is W_L at most, and uses the HSTCP response function when the current congestion window is greater than W_L. HSTCP ensures that the response function follows a straight line on a log-log scale as does the response function for Standard TCP, for low to moderate congestion. When the value of the average congestion window is greater than W_L, the response function is as follows:

$$W = \left(\frac{p}{P_L}\right)^S W_L \tag{4}$$

where,

$$S = \frac{(\log W_H - \log W_L)}{(\log P_H - \log P_L)} \tag{5}$$

HSTCP keeps average congestion window W_H and W_L, when packet loss rates are P_H and P_L, respectively. Recommended parameters are: $W_L = 38$, $W_H = 83000$ and $P_H = 10^{-7}$. This loss rate sets an achievable target for high-speed environments, while still allowing acceptable fairness for the HighSpeed response function when competing with Standard TCP in environments with packet drop rates of 10^{-4} or 10^{-5}. $P_L = 10^{-3}$ is computed by using equation (3), when $W_L = 38$. Thus, the HSTCP response function is computed as follows:

$$W_{highspeed} = \frac{0.12}{p^{0.835}} \tag{6}$$

It is clear from equation (6) that HSTCP is more aggressive than TCP Reno and a HighSpeed TCP connection would receive ten times the bandwidth of a standard TCP in an environment with packet drop rate of 10^{-6}, which is unfair.

Scalable TCP

Scalable TCP [13] is designed to be incrementally deployable and behaves identically to traditional TCP stacks when small windows are sufficient. Scalable TCP (STCP) and HighSpeed TCP were originally designed for high-speed backbone links, and they appear to be the major candidates for replacing in the next generation Internet the current congestion control mechanism implemented by standard TCP. STCP is a simple sender side modification to TCP congestion control, and it employs Multiplicative Increase Multiplicative Decrease (MIMD) technique. Using Scalable TCP, better utilization of a network link with the high bandwidth-delay product can be achieved. If STCP is mixed with regular TCP then STCP dominates the bandwidth for sufficiently large bandwidth-delay product region. This shows unfriendliness towards standard TCP. Scalable TCP changes the algorithm to update TCP's congestion window to the following:

$$\text{Increase: } W = W + 0.01W \tag{7}$$

$$\text{Decrease: } W = W - 0.125W \tag{8}$$

STCP response function is computed as follows:

$$W_{scalable} = \frac{0.0745}{p} \quad (9)$$

The recovery time after packet loss is 13.42RTT, i.e. proportional to the RTT and independent of congestion window size. An STCP connection can recover even a large congestion window in a short time and so that it makes efficient use of the bandwidth in high-speed networks. Suppose a TCP connection has a 1500 byte MTU and a round trip time of 200ms, then for 10Gbps network, congestion window recovery time after packet loss for STCP is 2.7 sec whereas that for Standard TCP is approximately 4hrs 43mins.

CUBIC TCP

CUBIC TCP[16] is an enhanced version of Binary Increase Congestion Control shortly BIC[12]. It simplifies the BIC window control function and improves its TCP- friendliness and RTT fairness as BIC's growth function is too aggressive for TCP especially under short RTT or low speed networks. As the name of the protocol represents, the window growth function of CUBIC is a cubic function in terms of the elapsed time since the last loss event, whose shape is very similar to the growth function of BIC. CUBIC function provides good scalability and stability. The protocol keeps the window growth rate independent of RTT, which keeps the protocol TCP friendly under short and long RTTs. The congestion epoch period of CUBIC is determined by the packet loss rate alone. As TCP's throughput is defined by the packet loss rate as well as RTT, the throughput of CUBIC is defined only by the packet loss rate. Thus, when the loss rate is high and/or RTT is short, CUBIC can operate in a TCP mode. The congestion window of CUBIC is determined by the following function:

$$W_{cubic} = C\,(t-K)^{3} + W_{max} \quad (10)$$

where, C is the scaling factor, t is the elapsed time from the last window reduction, W_{max} is the window size just before the last window reduction, and $K = (W_{max}\beta/C)^{1/3}$, where β is a constant multiplicative decrease factor applied for window reduction at the time of loss event (i.e., the window reduces to βW_{max} at the time of the last reduction). To achieve reasonable TCP friendliness, fairness, scalability and convergence speed, we set C to 0.4, β to 0.2, and S_{max} (window increment) to 160. The entire window growth function of CUBIC is described by just one function, it dose not need different phases of window control as in BIC, thus simplifying the complexity of BIC.

V. DELAY-BASED CONGESTION CONTROL

Delay-based TCP congestion control algorithms like TCP Vegas attempt to utilize the congestion information contained in packet round-trip time (RTT) samples**.**

TCP Vegas

TCP Vegas is a TCP congestion control algorithm that emphasizes packet delay, rather than packet loss, as a signal to determine the rate at which to send packets. TCP Vegas detects congestion based on increasing Round Trip Time (RTT) values of the packets in the connection unlike TCP Reno which detect congestion only after it has actually happened via packet drops. The algorithm depends heavily on accurate calculation of the Base RTT value. BaseRTT is set to be the minimum of all measured RTTs; it is commonly the RTT of the first segment sent by the connection.

If the connection is not overflown by the traffic, the expected throughput is given by:

$$\text{Expected Throughput} = \frac{\text{WindowSize}}{\text{BaseRTT}} \quad (11)$$

where WindowSize is the size of the current congestion window

Then current actual sending rate is calculated once per round trip time as:

$$\text{Actual throughput} = \frac{W}{RTT} \quad (12)$$

The congestion window is adjusted depending upon the difference between expected and actual sending rates.

$$\text{Difference} = (\text{expected} - \text{actual})\ \text{baseRTT} \quad (13)$$

Also two thresholds a and b are defined such that, $a < b$ and $a > b$ correspond to having too little and too much extra traffic in the network, respectively. When Difference < a, TCP Vegas increases the congestion window linearly during the next RTT, and when Difference > b, TCP Vegas decrease the congestion window linearly during the next RTT. The congestion window is left unchanged when a < Difference < b.

VI. MIXED LOSS-DELAY BASED TCP CONGESTION CONTROL

Loss-based high speed algorithms are aggressive to satisfy bandwidth requirement but this aggressiveness causes TCP unfairness and RTT unfairness. Delay-based approaches provide RTT fairness but it is difficult to meet TCP fairness. Thus there is another approach that is a synergy of delay-based and loss-based approaches that addresses the problems in the two approaches.

Compound TCP

Compound TCP [23] integrates a scalable delay-based component into the standard TCP congestion avoidance algorithm. This scalable delay-based component has a fast window increase function when the network is under-utilised and reduces the sending rate when a congestion event is sensed. To implement Compound TCP maintains the following state variables; cwnd (congestion window), dwnd (delay window), awnd (receiver advertised window).TCP sending window is calculated as follows:

$$W_{in} = \min (\text{cwnd}+\text{dwnd}, \text{awnd}) \quad (14)$$

cwnd is updated in the same way as controlled by standard TCP. Here in this case, on arrival of an ACK, cwnd is modified as:

$$\text{cwnd} = \text{cwnd} + \frac{1}{(\text{cwnd} + \text{dwnd})} \quad (15)$$

Delay-based component is derived from TCP Vegas as explained in the above sub section.

VII. PERFORMANCE EVALUATION

We assessed performance in terms of congestion window verses elapsed time. We used Network Simulator version 2 (ns-2) for simulations of three loss-based, high-speed TCP variants HighSpeed TCP, Scalable TCP, and CUBIC TCP; delay-based TCP congestion control namely TCP Vegas; and mixed loss-delay based TCP Congestion control namely Compound TCP in comparison with TCP Reno using their default parameters for all experiments. The key difference among various high-speed TCP implementations lies in their congestion window growth behavior in response to a congestion event. Experimentation is done on two different topologies. BDP is set to 30,000. All simulations were run long enough to ensure that the system had consistent behavior. Figure 1 and figure 7 show simulation topologies 1 and 2 respectively. The type of data traffic is FTP for both scenarios.

Simulation Topology 1

For simulation topology 1, equal number of data sending nodes is connected to each main node labeled as 0, 1, 2 and 3. All main nodes have direct connections with one another in our topology. Flow of data between i number of nodes (where i = 1, 2, 3, 4, 5) connected to each main node (0, 1, 2, 3) is as follows:

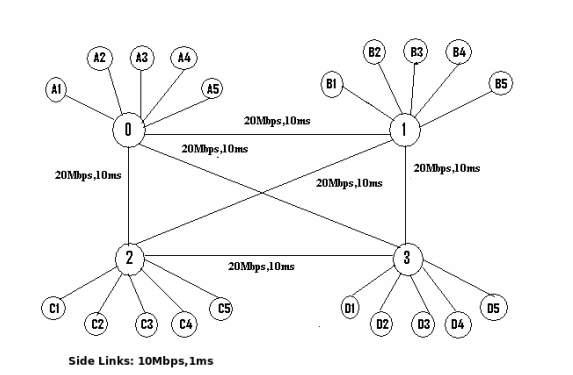


Fig. 1: Simulation Topology 1

Case a:

- A_i is a source node for B_i, C_i, and D_i.
- B_i is a source node for A_i, C_i, and D_i.
- C_i is a source node for A_i, B_i, and D_i.
- D_i is a source node for A_i, Bi, and C_i.

Performance of all the above mentioned algorithms was individually tested for TCP connections explained in case a for figure 1.

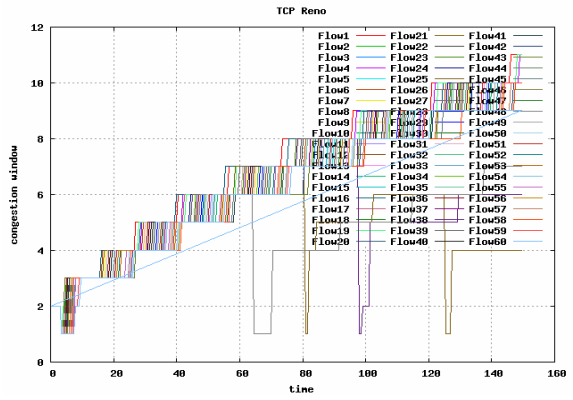


Fig. 2: Congestion window verses elapsed time of TCP Reno

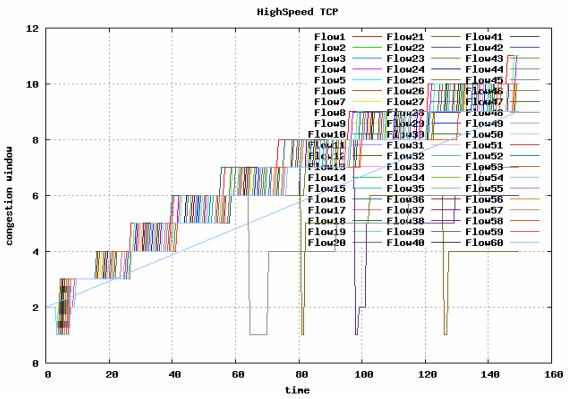


Fig.3: Congestion window verses elapsed time of HighSpeed TCP

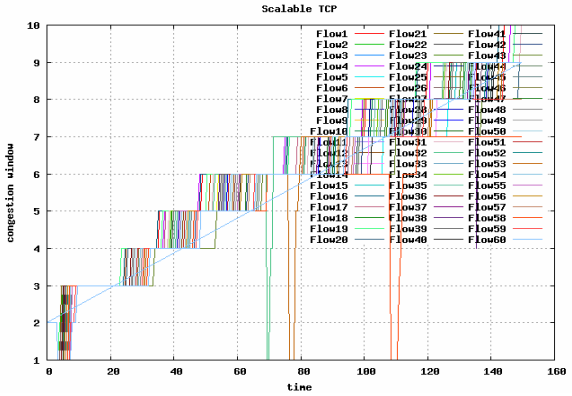


Fig. 4: Congestion window verses elapsed time of Scalable TCP

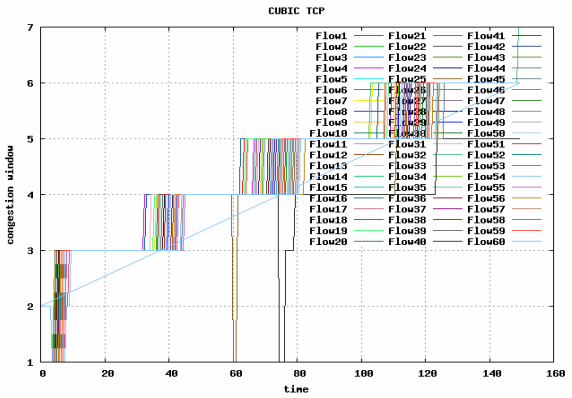


Fig. 5: Congestion window verses elapsed time of CUBIC TCP

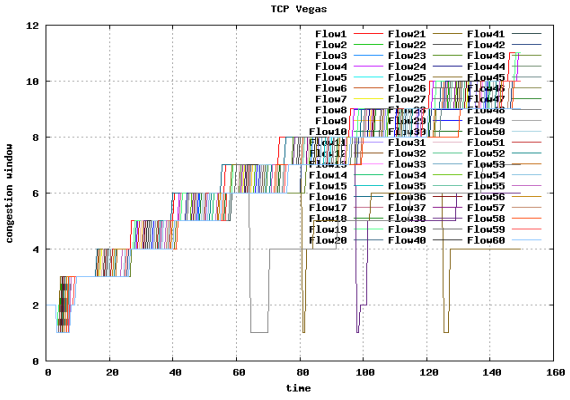


Fig. 6: Congestion window verses elapsed time of TCP Vegas

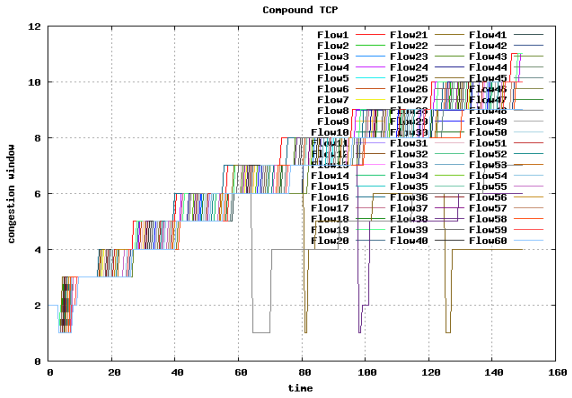


Fig.7: Congestion window versus elapsed time of Compound TCP

Figures 2-7 show the results of 'case a' of simulation topology 1. It can be seen that all flavors of TCP congestion control give same response because equal number of sink nodes are connected to each source node. Thus, no serious congestion event resulting in packet drops occurs at any backbone link even though aggregate side bandwidth of the links with each main node exceeds that of the main links.

Case b:

Case b analyzes simulation topology 1 in four different ways. Four different scenarios with their simulation results is shown below.

Scenario I:

- All TCP connections with i=1 have HighSpeed TCP support.
- All TCP connections with i=2 have TCP Reno support.
- All TCP connections with i=3 have Scalable TCP support.
- All TCP connections with i=4 have CUBIC TCP support.

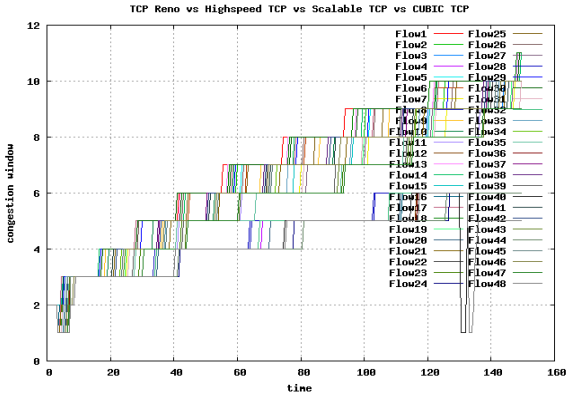


Fig. 8: Combined performance evaluation of TCP Reno, HighSpeed TCP, Scalable TCP and CUBIC TCP

Figure 8 shows there is no serious congestion collapse when each network node receives data traffic from

source nodes with different loss-based algorithms. However, TCP Reno suffers with packet drop events because of aggressive window growth behavior of other three algorithms.

Scenario II:

- All TCP connections with i=1 have HighSpeed TCP support.
- All TCP connections with i=2 have TCP Reno support.
- All TCP connections with i=3 have TCP Vegas support.
- All TCP connections with i=4 have Compound TCP support.

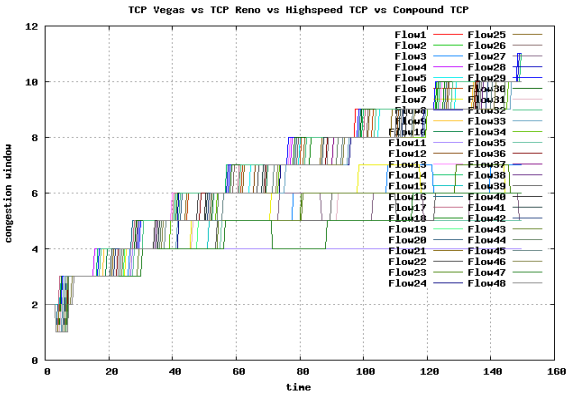


Fig. 9: Combined performance evaluation of TCP Reno, HighSpeed TCP, TCP Vegas and Compound TCP

The simulation results for this network flow show unfair bandwidth allocation to TCP Reno depicted in figure 9 also. HighSpeed TCP connection works in TCP mode in the start of the connection, TCP Vegas calculates BaseRTT in the start of the connection and calculates expected throughput and based on the difference between actual and expected throughput goes on increasing or decreasing the congestion window, so at the start of the connection TCP Vegas also does not take an aggressive start. Compound TCP also waits and increases window as explained in equation 14 and 15. When no congestion event occurs and network under-utilization is noticed, i.e. for HighSpeed TCP, cwnd = W_L; difference < a for TCP Vegas; Compound TCP as explained above in section V, all the algorithms increase their congestion windows and as connection for the backbone links increases, packet drop event occurs resulting in retransmissions forcing the algorithms to reduce their congestion windows.

Scenario III:

- All TCP connections with i=1 have TCP Vegas support.
- All TCP connections with i=2 have TCP Reno support.
- All TCP connections with i=3 have Compound TCP support.
- All TCP connections with i=4 have CUBIC TCP support.

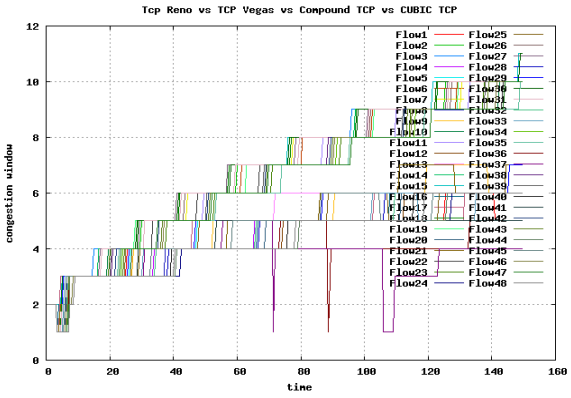


Fig. 10: Combined performance evaluation of TCP Reno, TCP Vegas, Compound TCP and CUBIC TCP

This scenario as shown in figure 10 results in serious congestion events and retransmissions and slow cwnd increase behavior. The cwnd growth is very slow as compared to previous two scenarios because four different TCP variants based on loss-based, delay-based, mixed loss-delay based algorithms are competing for fair allocation of bandwidth.

Scenario IV:

- All TCP connections with i=1 have HighSpeed TCP support.
- All TCP connections with i=2 have TCP Vegas support.
- All TCP connections with i=3 have Scalable TCP support.
- All TCP connections with i=4 have Cubic TCP support.

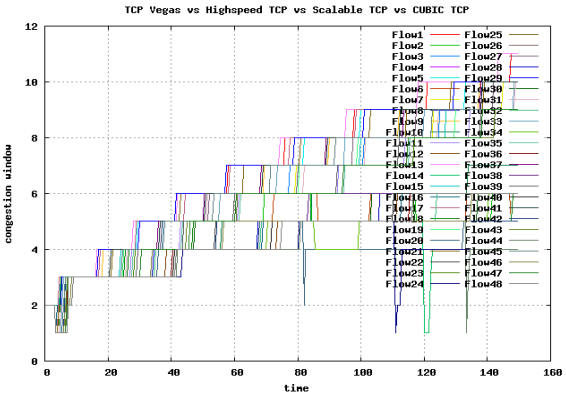


Fig. 11: Combined performance evaluation of TCP Vegas, HighSpeed TCP, Scalable TCP and CUBIC TCP

Figure 11 depicts serious congestion events for scenario IV like scenario III because high-speed loss-based and delay-based TCP algorithms are competing with each other for bandwidth allocation.

Four scenarios of 'case b' show that scenario I has optimum results among all other scenarios. Thus, it is the most favorable that TCP algorithms belonging to similar class either loss-based or delay-based should compete with each other. Serious problems and slow responses result when loss-based and delay-based algorithms compete with each other.

Simulation Topology 2

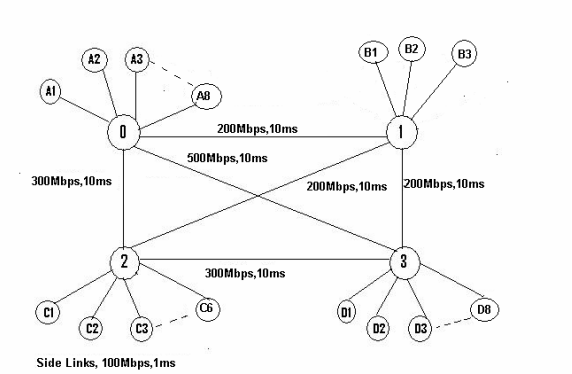


Fig. 12: Simulation Topology 2

Flow of data from sender hosts to receiver hosts is explained for simulation topology 2 in table 1.

Sources	Sinks
A1	B1, C1, D1
A2	B1, B2, C2, D1, D2
A3	B3, C3, D3
A4	C4, D4
A5	C5, D5
A6	C6, D6
A7	D7
A8	D8
B1	A1, A4, A7, C1, C4, D1, D4, D7
B2	A2, A5, A8, C2, C5, D1, D2, D5, D8
B3	A3, A6, C3, C6, D6
C1	B1, D1, D7
C2	B1, B2, D2, D8
C3	B3, D3
C4	D4
C5	D5
C6	D6
D1	C1, B1
D2	B1, B2, C2
D3	B3, C3
D4	C4
D5	C5
D6	C6

Table 1: Data flow between TCP source and sink nodes

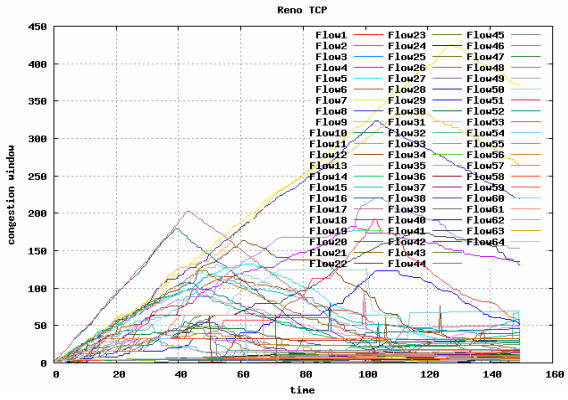


Fig. 13: Congestion window verses elapsed time of TCP Reno

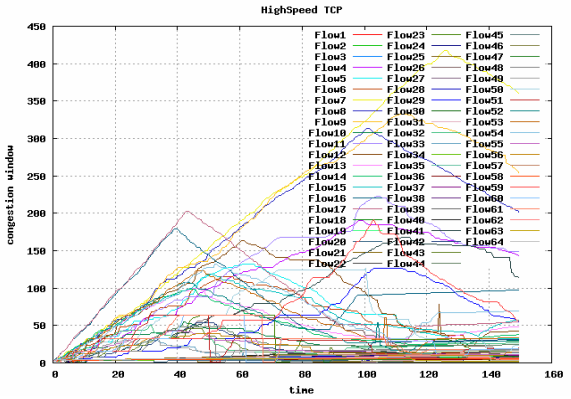


Fig. 14: Congestion window verses elapsed time of HighSpeed TCP

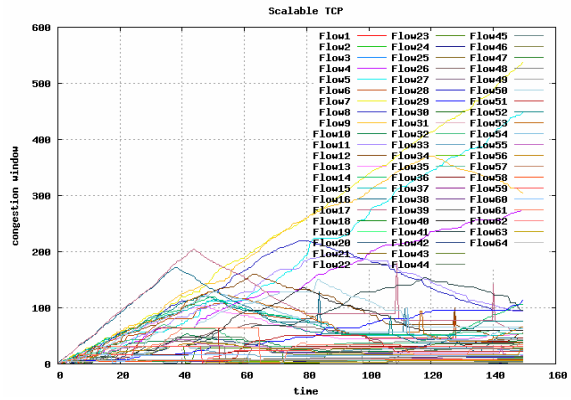


Fig. 15: Congestion window verses elapsed time of Scalable TCP

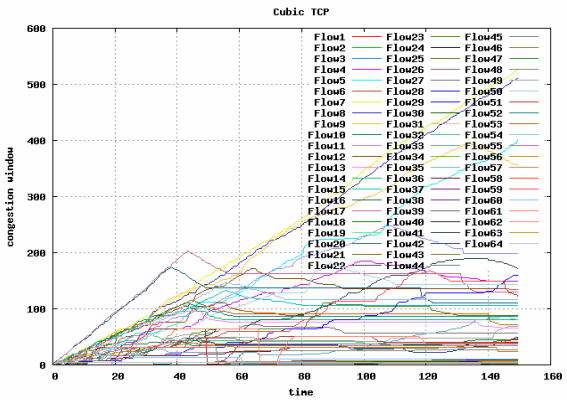


Fig. 16: Congestion window verses elapsed time of CUBIC TCP

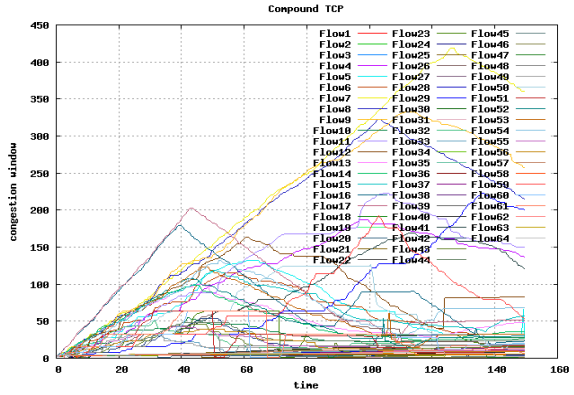


Fig. 17: Congestion window verses elapsed time of Compound TCP

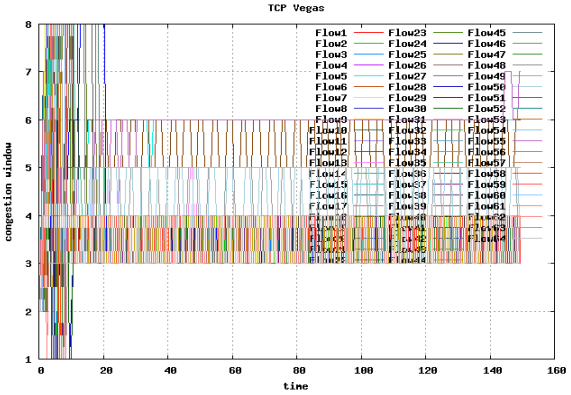


Fig. 18: Congestion window verses elapsed time of TCP Vegas

For all implementations of TCP, node 0 and node 1 experience zero packet drop events. Figure 13 and figure 14 show similar curves for TCP Reno and HighSpeed TCP respectively because High-speed TCP works in two different phases depending upon the network capacity available. For both TCP algorithms congestion windows of the majority of the TCP connections fail to grow to achieve network capacity. Scalable TCP as shown in figure 15, being the most aggressive high-speed TCP variant having throughput ratio of 60.9 with TCP Reno, resulted in maximum packet drop events and congestion window of maximum number of TCP connections in the simulation topology fail to grow above 100 packets. Figure 16 shows that the congestion window growth behavior of CUBIC TCP is much better and smooth as compared to other loss-based TCP variants and there are less abruptly falling congestion windows and congestion window remains constant over the wide range of elapsed time. Compound TCP's behavior is almost similar to HSTCP as shown in figure 17 .Finally, figure 18 shows that TCP Vegas is the smoothest of all TCP congestion control algorithms and majority of TCP connections grow to attain the available capacity. Thus TCP Vegas allocates a fair share of bandwidth to each connection and resulted in no packet drop event. Thus, it is favorable to transmit traffic flows with TCP Vegas support for such network environment.

REFERENCES

[1]. V. Jacobson, "Congestion avoidance and control," the ACM SIGCOMM'88

[2]. Tomoya Hatano, Hiroshi Shigeno and Ken-ichi Okada, "TCP friendly congestion control for highSpeed network", IEEE, 2007.

[3]. David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hedge, "Fast TCP: Motivation, Architecture, Algorithms, Performance", IEEE/ACM transactions on networking, 2006

[4]. W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001, Jan. 1997

[5]. M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", RFC 2581, Apr. 1999

[6]. S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm", RFC 2582, Apr. 1999.

[7]. V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, May 1992

[8]. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, Oct. 1996.

[9]. Jon Postel, "Transmission Control Protocol," September 1981, RFC 793

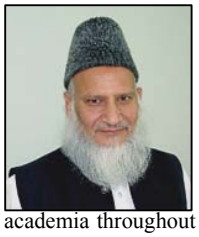
[10]. Allman, M., Balakrishnan, H. and S. Floyd, "Enhancing TCP's Loss Recovery using Limited Transmit:, RFC 3042, January 2001

[11]. S. Floyd: "HighSpeed TCP for Large Congestion Windows", RFC 3649, December 2003

[12]. Lisong Xu, Khaled Harfoush, and Injong Rhee: "Binary Increase Congestion Control for Fast, Long Distance Networks", 2003

[13]. Tom Kelly: "Scalable TCP: Improving performance in high-speed wide area networks"; ACM SIGCOMM

Computer Communication review, April 2003.
[14]. Hamed Vahdet Nejad, Mohammad Hossien Yaghmaee, Hamid Tabatabaee, "Fuzzy TCP: Optimizing TCP Congestion Control", IEEE, 2006
[15]. M. Allman, S. Floyd, C.Partridge, "Increasing TCP's initial window", September 1998
[16]. Injong Rhee, and Lisong Xu: "CUBIC: A New TCP-Friendly High-Speed TCP Variant", Sangtae Ha, Injong Rhee, Lisong Xu.
[17]. D. Leith, and R. Shorten: "H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths", June 20, 2005
[18]. Aleksandar Kuzmananovic and Edward W. Knightly, "TCP-LP: a distributed Algorithm for low priority data transfer", IEEE 2003
[19]. Kun Tan Jingmin Song, Qian zhang, Murari Sridharan, "A Compound TCP Approach For High-Speed and Long Distance Networks".
[20]. R. King, R. Baraniuk and R. riedi, "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP", *In Proc Infocom 2005*
[21]. Dina Kitabi, M. Handley, and C.Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks". ACM SIGCOMM 2002, August, 2002
[22]. Brakmo LS, Peterson LL, "TCP Vegas: end to end congestion avoidance on a global internet". IEEE Journal on Selected Areas in Communications 1995
[23]. Kun Tan, Jingmin Song, Qian Zhang, Murari Sridharan, "A Compound TCP Approach for High-speed and Long Distance Netowrks"



Habibullah Jamal did his B.Sc. Electrical Engineering from University of Engineering and Technology, Lahore, Pakistan in 1974. He earned his MSc and PhD degrees both in Electrical Engineering from University of Toronto, Canada, in 1979 and 1982 respectively. Dr Jamal has served academia throughout his professional career. He is recipient of Performance Excellence in Engineering Award from the Institution of Engineers, Pakistan on the 'Engineers Day' May 29, 2007, 9th Pakistan Education Forum, National Education Award – 2003 and National Book Council of Pakistan Award 1991. His research interests include digital Design, Analog and Digital Signal Processing, and Communications. He is a Fellow/Senior Member of many professional bodies including IEEE.

Kiran Sultan earned her B.Sc. and MSc. degrees both in Electrical Engineering from University of Engineering and Technology, Taxila, Pakistan in 2003 and 2008 respectively. Currently she is Assistant Professor, Department of Electrical Engineering, Air University, Islamabad, Pakistan. Her research interest includes Computer Communication Networks.