Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EXPERIMENTAL EVALUATION OF TCP CONGESTION CONTORL MECHANISMS IN SHORT AND LONG DISTANCE NETWORKS

Mudassar Ahmad, Md Asri Ngadi, Mohd Murtadha Mohamad
Department of Computer Science, Faculty of Computing,
Universiti Teknologi Malaysia (UTM), 81310 Johor, Malaysia.

E-mail : mudassar.utm@gmail.com , dr.asri@utm.my *(Correspondence Author),* murtadha@utm.my

ABSTRACT

Originally TCP was designed for early, low bandwidth, short distance networks, so Standard TCP did not utilize the maximum bandwidth in today's high bandwidth network environments. Therefore a lot of TCP congestion control mechanisms also known as TCP variants have been developed for today's long distance high bandwidth networks. In this paper the experimental results evaluating the performance of TCP Reno, HighSpeed TCP, BIC TCP, TCP CUBIC and Compound TCP in short and long distance high bandwidth networks are presented. Results show that TCP CUBIC shows the highest performance in goodput whereas TCP Compound shows the highest performance in protocol fairness and TCP friendliness as compared to the other stat of the art congestion control mechanisms.

Keywords: *Congestion Control Mechanism, Protocol Fairness, TCP Friendliness, Goodputs.*

1 INTRODUCTION

TCP is officially adopted as a standard in RFC 793 (Requests for Comments) in 1981 and is designed to deal with message flow control and error correction [1]. TCP works on the top of IP, ensuring reliable communication in today's Internet. TCP is a reliable because of its main component that is congestion control which is responsible for detecting and re acting the overload traffic on the Internet. TCP still requires high performance by preventing congestion collapse. Long distance, high bandwidth networks are spanning in several continents rapidly and TCP has been widely used as a primary transfer protocol in these networks.

TCP performance is one of the main critical issues in long distance networks, because TCP is not utilizing the maximum bandwidth. TCP performance depends upon the three main components: poor loss detection, coefficients of congestion window (*cwnd)* before or after loss and increase in *cwnd* at the beginning of connection. The small additive growth of *cwnd* used in TCP congestion control was blamed for its poor performance on these networks. Many advanced TCP congestion control mechanisms have been proposed, adopting more scalable *cwnd* window growth function for better performance in high bandwidth, long distance high bandwidth networks. Most of these new congestion control mechanisms only modify the protocol behavior during congestion avoidance phase. These advance TCP congestion control mechanisms are briefly discussed in Section 3.

For long distance high bandwidth networks most of existing TCP congestion control mechanisms are not optimized, because when they run in such environments, they fall into some rare states where TCP obtain extremely low performance. The aim of this paper is to compare the performance of competing TCP congestion control mechanisms by using a set of benchmark tests that can probe a series of important aspects inside protocols and can apply to all other TCP proposals. In this paper experimental evaluation and behavior of TCP Reno [2], HighSpeed TCP [3], BIC TCP [4], TCP CUBIC [5] and Compound TCP [6] are discussed. The empirical results highlight a number of deficiencies in the studied protocols and finally to render this weakness in the protocols future direction toward the deployment of the protocol in real network is also suggested.

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2 TCP CONGESTION CONTROL

In order to avoid congestion collapse, TCP congestion control follows a packet conversation principle [7], which confirms the transmitted packet delivery by acknowledgment (ACK). For every packet sent on the network by a source, an ACK is expected to be transmitted back from the destination. The source controls the packet sending rate by using a variable called congestion window *cwnd*, which determines the number of packets that the source is allowed to send.

The destination also advertises to the source the amount of data it is willing to buffer for connection called advertised window (*rwnd)*. By using these two variables, the source can transmit the data up to the maximum amount of congestion window or advertised window. Data transmission between source and destination depends upon comparative minimum values of either *cwnd* or *rwnd*. Figure 1 shows the typical behavior of TCP congestion control.

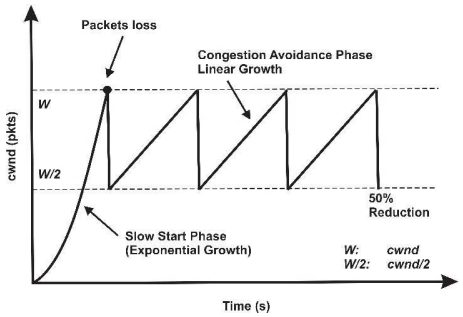


Figure 1: Dynamics of TCP Congestion Control

TCP congestion control has four main components, slow start and congestion avoidance, fast retransmit and fast recovery. Slow start and congestion avoidance algorithms control the transmissions. The slow start threshold (*ssthresh*) is used to determine which algorithm, slow-start or congestion avoidance is being used by TCP to control the data transmission. If the amount of *cwnd* is less than *ssthresh*, then slow start algorithm is used and if *cwnd* is greater than or equal to *ssthresh*, then congestion avoidance algorithm is used as denoted in Eq. 1.

$$\begin{cases} cwnd < ssthresh & SlowStartAlgorithm \\ cwnd \geq ssthresh & CongestionAvoidanceAlgorithm \end{cases} \quad (1)$$

During the initial stage of the connection, the slow start algorithm increases the congestion window exponentially to find the unknown equilibrium state of the network. However, on the other hand, congestion avoidance algorithm controls the growth of congestion window, because the source has already reached the equilibrium state of the network. After receiving three duplicate ACKs, the fast retransmit algorithm re transmits the dropped packet without waiting for a retransmission timer to expire. After fast retransmit algorithm, the fast recovery algorithm continues to work to maintain the same number of packets prior to entering into the fast recovery. When the source receives the ACK of lost data, fast recovery algorithm terminates.

Slow start algorithm is used to probe the time-varying available bandwidth of the current network path. Source increases its *cwnd* by one at each ACK, which doubles it when receiving ACKs for all the packets. Eq. 2 denotes the *cwnd* evolution during slow start upon receiving an ACK. A destination running different ACK scheme affects the ramp-up speed of slow start algorithm on the source. Microsoft Windows and FreeBSD [8] operating system use delayed ACK from the beginning of the connection, while Linux uses quick ACK for initial 16 packets, then delayed ACK, because delayed ACK is mandatory for TCP end systems. When the size of *cwnd* becomes larger than a *ssthresh*, the source exit slow start phase and enters into congestion avoidance phase. *ssthresh* is an estimated conservative measure of available link bandwidth in the network path.

$$ACK : cwnd \leftarrow cwnd + 1 \; if \; (cwnd < ssthresh) \quad (2)$$

Filling the network pipe in slow start phase is a very important concept in performance. In congestion avoidance phase, normally a source increases its congestion window by *(1/cwnd)* for each incoming ACK (for TCP Reno [2], New Reno [9] and SACK [10] only). This makes the source gradually increase its *cwnd* by only one packet per each RTT, because the source has already reached the equilibrium state of the network. Upon loss detection after receiving three duplicate ACKs, the source reduces its *cwnd* by half. Eqs. 3 & 4 show the *cwnd* evolution during congestion avoidance upon receiving an ACK and upon loss detection respectively. The source uses packet loss as an indication of network congestion. In the absence of network congestion, the source increases its *cwnd*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

additively, while in presence of network congestion by receiving three duplicate ACKs, source drops its *cwnd* by half *(1/2)* of its current *cwnd* to reduce the congestion in the network path. This mechanism is called Additive Increase and Multiplicative Decrease (AIMD) [11]. Eq. 5 shows the general Standard TCP with $(\alpha = 1)$ and $(\beta = \frac{1}{2})$.

$$ACK : cwnd \leftarrow cwnd + \frac{1}{cwnd} \; if \; (cwnd \geq ssthresh) \quad (3)$$

$$Loss : cwnd \leftarrow \frac{1}{2} \times cwnd \quad (4)$$

$$AIMD : \begin{cases} ACK & = & cwnd \leftarrow cwnd + \frac{\alpha}{cwnd} \\ Loss & = & cwnd \leftarrow (1-\beta) \times cwnd \end{cases} \quad (5)$$

3 EVALUATING HIGH SPEED NETWORK PROTOCOLS

In this section performance of TCP Reno, HighSpeed TCP, BIC TCP, TCP CUBIC and Compound TCP is measured with respect to goodput, protocol fairness and TCP friendliness. These congestion control mechanisms have been the subject of consideration and experimentation in recent years with the implementation in Linux operating system which is now publically available on the Internet. NS-2 [12] with dumbbell network topology is used for simulation as shown in Figure 7. For each simulation two flows of each TCP congestion control mechanisms are run on short and long RTTs network scenarios with different bottleneck and link speed bandwidth. Simulation time for each experiment test is set to 300 seconds and no background traffic is used. For all the simulation experiments, the buffer size of Drop Tail router is set to [0.01, 0.02, 0.05, 0.1, 0.2, 0.4, 0.5, 1.0, 1.5, 2.0] BDP-Q. Complete set of simulation parameters are shown in Table 1. Before proceeding, a comprehensive review of each protocol with respect to its behavior on each acknowledgment and after loss event is discussed in the form of equations in next sub sections.

3.1 TCP Reno

TCP Reno [2] is developed by Van Jacobson and it is an enhanced form of TCP Tahoe [13]. In TCP Reno, congestion is detected via a packet loss through Re-transmit Time Out (RTO) not by three duplicate ACKs. After loss event the value of congestion window (*cwnd)* is set to half *(1/2)* of its previous value as denoted in Eq. 7. If the source is still able to receive the ACKs and after receiving a number of duplicate ACKs, TCP Reno enter in the fast recovery phase and the source re transmits the lost packet, however, unlike TCP Tahoe, it will not fall back into slow start state [14]. In fast recovery mode, when a duplication ACK is received, the *cwnd* size is increased by one segment *(cwnd=cwhd+1)*. However cwnd is restored to *ssthresh (cwnd=ssthresh)* when a non-duplicate acknowledgment corresponding to retransmitted segment is received.

The Main problem in TCP Reno is that fast retransmit mechanism assumes that only one segment is lost, if more than one segment is lost, it leads towards poor performace in the presence of multiple packet losses. TCP New Reno [9] and TCP SACK [10] solved this issue. ACK starvation is another problem in TCP Reno, which occurs due to the ambiguity of duplicate ACKs. Hence, TCP Reno is better than TCP Tahoe only in case of single packet loss, but not much better if multiple packets are lost. Most of the new TCP congestion control mechanisms are based on TCP Reno. Eqs. 6 & 7 represent the value of *cwnd* in slow start and fast recovery phases. Figure 2 shows the typical behavior of congestion window growth of TCP Reno.

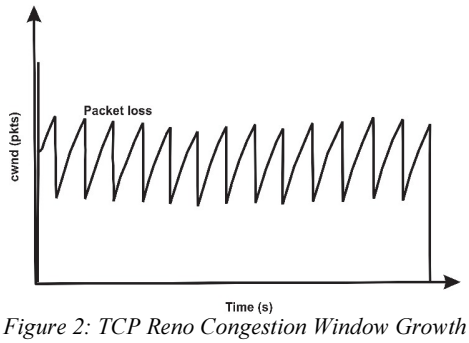


Figure 2: TCP Reno Congestion Window Growth Behavior

$$ACK = \begin{cases} cwnd & \leftarrow & cwnd + 1 \\ ssthresh & \leftarrow & \frac{1}{2} \times cwnd \end{cases} \quad (6)$$

$$Loss = \begin{cases} ssthresh_{n+1} & \leftarrow & \frac{1}{2} \times cwnd_n \\ cwnd_{n+1} & \leftarrow & ssthresh_n \end{cases} \quad (7)$$

3.2 Highspeed TCP

HighSpeed TCP [3] is a modified form of original TCP congestion avoidance algorithm. HighSpeed TCP uses modified AIMD parameters, where the

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

linear increase factor $\frac{f_\alpha(cwnd)}{cwnd}$ and the multiplicative decrease factor $g_\beta(cwnd)$ are adjusted by a convex function for the current *cwnd* size. When the size of *cwnd* is less than or equal to *38*, HighSpeed TCP uses the similar increase and decrease factors as Standard TCP. When the *cwnd* grows beyond the *cutoff* value, the function raises the increase factor and reduces the decrease factor proportional to the *cwnd* size. Based on [3, 15], Eqs. 8 & 9 denote the congestion window value at each ACK and loss events respectively. Figure 3 shows the growth of *cwnd* for this protocol.

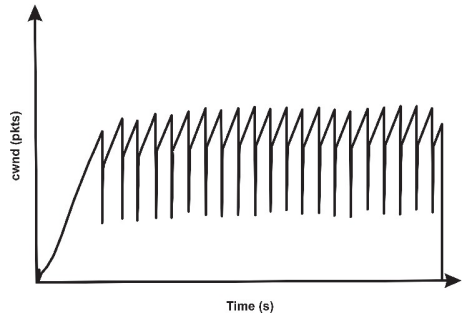


Figure 3: HighSpeed TCP Congestion Window Growth Behavior

$$ACK: cwnd \leftarrow cwnd + \frac{f_\alpha(cwnd)}{cwnd} \quad (8)$$

$$Loss: cwnd \leftarrow g_\beta(cwnd) \times cwnd \quad (9)$$

3.3 BIC TCP

BIC (Binary Increase Congestion Control Algorithm) TCP [4] uses two window size control policies called additive increase and binary search increase to maximize the *cwnd*. For a packet loss, BIC reduces its *cwnd* by a multiplicative decrease factor (β) as shown in Eq. 10. The *cwnd* size prior to reduction is set to (W_{max}) and after reduction is set to (W_{min}). Since packet loss have occurred at (W_{max}), the *cwnd* size that the network can currently handle without loss must be some where between these two numbers, so BIC performs a binary search by using (W_{max}) and (W_{min}) parameters, by jumping to the midpoint between these two parameters. If the distance between the midpoint $\left(\frac{W_{min}+W_{max}}{2}\right)$ and the current minimum (W_{min}) is larger than the maximum increment (S_{max}), BIC increases the current window size by (S_{max}), thus $cwnd = cwnd + S_{max}$ and this is called linear increase. If BIC does not get packet loss at the updated window size, that window size becomes the new (W_{min}) and if it gets a packet loss, the updated window size becomes the new (W_{max}). This process continues until the window increment is less than minimum increment (W_{min}) at which point, the window is set to the current maximum (W_{max}). The window growth function of BIC TCP is explained graphically in Figure 4.

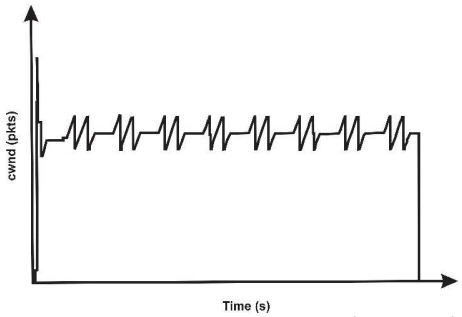


Figure 4: BIC TCP Congestion Window Growth Behavior

$$Loss: cwnd \leftarrow (1-\beta) \times cwnd \quad (10)$$

3.4 TCP Cubic

TCP CUBIC [5] adopted new slow-start algorithm called HyStart [16], which prevents long burst losses by finding a *Safe* exit point during slow-start and thus improves the start-up throughput of TCP CUBIC in long distance, high bandwidth networks. After a window size reduction due to a loss event, TCP CUBIC registers (W_{max}) as the widow size where the loss even occurred. Then it decreases the *cwnd* by a constant decrease factor (β) and enters into congestion avoidance phase and begins to increase the window size by using a concave feature of cubic function, until the window size becomes (W_{max}). The window grows very fast after a window reduction, but as it gets close to (W_{max}), it slows down its growth, around (W_{max}); the window increment becomes

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

almost zero. Eq. 11 show the growth function of TCP CUBIC, where (C) is the TCP CUBIC parameter, (*t*) is the elapsed time from the last window reduction and (K) is the time period that the function requires to increase (W) to (W_{max}) (when there are no further loss events occur). (K) is calculated by using the function given in Eq. 12. TCP CUBIC sets the $(W(t+RTT))$ as the candidate target value of *cwnd*. The *cwnd* growth of this protocol is shown in Figure 5.

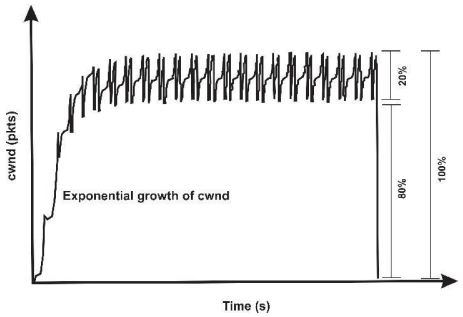


Figure 5: TCP CUBIC Congestion Window Growth Behavior

$$W(t) = C(t-K)^3 + W_{max}$$

(11)

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

(12)

3.5 Compound TCP

Compound TCP (CTCP) [6] developed by Microsoft for the Vista operating system and designed for high bandwidth delay product network uses a scalable delay based component of TCP Vegas [17] into the Standard TCP Reno congestion avoidance algorithm. A new state variable called delay window *cwnd* is introduced in current TCP Control Block (TCB) [18] to control the delay based component in CTCP.

$$cwnd \leftarrow \begin{cases} cwnd + \frac{1}{win} & upon receiving\ an\ ACK \\ \frac{1}{2} \times cwnd & upon\ detecting\ a\ loss \end{cases}$$

(13)

$$win = \begin{cases} cwnd + 0 & Slow\ Start \\ cwnd + dwnd & Congestion\ Avoidance \end{cases}$$

(14)

$$ACK : win_{t+1} \leftarrow win_t + \alpha \times win_t^k$$

(15)

$$Loss : win_{t-1} \leftarrow win_t - \beta \times win_t$$

(16)

$$dwnd_{t+1} \leftarrow \begin{cases} if\ Diff < \gamma \\ dwnd_t + max\left((\alpha(win_t)^k - 1), 0\right) \\ \\ if\ Diff \geq \gamma \\ max\left((dwnd_t - \zeta Diff), 0\right) \\ \\ upon\ detecting\ loss \\ max\left((win_t(1-\beta) - \frac{cwnd}{2}), 0\right) \end{cases}$$

(17)

At the starting of a new connection, this protocol uses the slow-start behavior of the regular TCP by increasing the *cwnd* in a manner similar to TCP New Reno as expressed in Eq. 13 and sets the value of *dwnd* to 0. When the connection switches to congestion avoidance phase, delay-based component is enabled. Thus, CTCP maintains two windows concurrently, a regular *cwnd* based on legacy TCP's AIMD algorithm and a delay window *dwnd* based on delay-based component of TCP Vegas. The sending rate of CTCP is determined by (*win*) by summing these two windows as shown in Eq. 14. Delay window *dwnd* is determined by using a queuing delay mechanism of Vegas. When a new connection is started, this protocol estimated and measured (*baseRTT*) also known as (*minRTT*) and exponentially smoothed round trip time (*sRTT*). It also estimated the number of backlogged packets on the connection as (*Diff*), which is equal to $(\frac{win}{baseRTT} - \frac{win}{sRTT}) \times baseRTT$. It stands for the amount of data that injected into the network in last round but does not pass through the network in this round i.e., the amount of data backlogged in the bottleneck router. An early congestion indication can be detected, if the number of packets in queue (*Diff*) is larger than a threshold (γ). If $(Diff < \gamma)$, the network path is considered as under utilized, otherwise the network path is determined as congested. In the absence of congestion, CTCP window increases as Eq. (15) and if there is a loss, the window is multiplicative decreased as Eq. 16. The overall CTCP follows the behavior defined in theses two Eqs. At the end of each round trip time, (*Diff*) is calculated. Based on the value of this Vegas gap (*Diff*) and a global constant threshold (γ), which is equal to 30 packets in this protocol, (*dwnd*) is calculated as in Eq. 17, here (ζ) is a parameter that defines how rapidly the delay based component should reduce this window when early congestion is detected. If

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the (*Diff*) is small, *dwnd* increases very rapidly to utilize the link maximum. If it is large, this is an indication that the network path is getting congested, the *dwnd* decreases. Figure 6 shows the *cwnd* graph of TCP Compound.

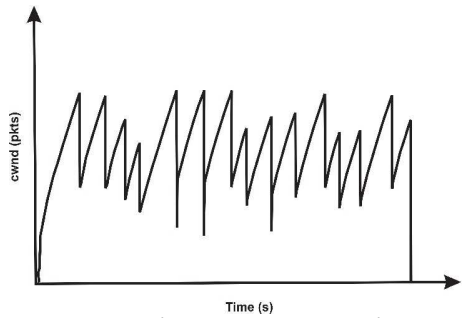


Figure 6: Compound TCP Congestion Window Growth Behavior

4 EXPERIMENTAL SETUP

NS-2 is one of the best commonly used simulation tool for evaluation of TCP protocols. In this paper, NS-2 is used to analyze the goodput, fairness and TCP friendliness behavior of all these TCP variants. Simulation tests are run according to the simulation parameters as defined in Tables 1 & 2. NS-2 version 2.35 on Linux Fedora Core 16 installed on Core i7 HP Elite book 2540P is used. Simulation topology shown in Figure 7 consists of 7 nodes. Two nodes are acting as data sources, two are as data destinations and two are acting as routers among sources and destinations. Hamilton benchmark suite [19] is used in all simulation experiment tests. Tmix traffic [20] is used among source and destination nodes. Awk [21] scripts are used to extract useful information from NS-2 trace files. Finally all empirical data is analyzed in SPSS [22] to get useful results in the form of graphs.

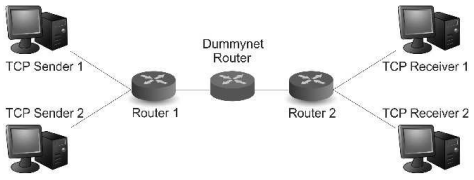


Figure 7: Test Bed.

Table 1: Simulation Parameters1.

Parameter	Values
Protocol	TCP Linux
Variant	CUBIC, BIC TCP, Compound,
	TCP Reno, HighSpeed TCP
Bottleneck Bandwidth	[50, 100, 150, 200, 250] Mbps
Link Bandwidth	[100, 200, 300, 400, 500] Mbps
Flow 1 RTT	160 ms
Flow 2 RTT	[10, 30, 50, ..., 310] ms
BDP Q Size	0.5, 1.0. 1.5, 2.0
Background Traffic	nil
Test duration	600 seconds
Test repetition	3 times

Table 2: Simulation Parameters 2.

Simulation Parameter	Values
Protocol	TCP Linux
TCP Variant	CUBIC, BIC, Compound, Reno, HighSpeed TCP.
Bottleneck Bandwidth	[50, 100, 150, 200, 250] Mbps
Link Speed (Bandwidth)	[100, 200, 300, 400, 500] Mbps
Network Scenarios	Short RTTs, Long RTTs
Flow1 RTTs	
(Short-Diff-RTT)	50 ms
Flow1 RTTs	
(Short-Same RTT)	[2, 4, 6,..,16] ms
Flow1 RTTs	
(Long-Diff-RTT)	100 ms
Flow1 RTTs	
(Long-Same-RTT)	[50, 70, 90,..,190] ms
Flow2 RTTs	
(Short RTT)	[2, 4, 6,..,16] ms
Flow2 RTTs	
(Long RTT)	[50, 70, 90,..,190] ms
BDP-Q Size	0.01, 0.02, 0.05, 0.1, 0.2,
in all networks	0.4, 0.5, 1.0, 1.5, 2.0
Background Traffic	nil
NS-2 Tests Duration	300 Seconds
Tests Repetition	3 Times

5 PERFORMANCE METRICS

To evaluate the performance of TCP variants, many performance metrics measured by simulations are presented in this section. Long term

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and steady state values of the performance metrics are used in analysis. Following is a list of performance metric used in this paper:

• **Goodput:** It is the application layer throughput measured at the data receiving node. It is the number of bits deviled to the application layer of the receiving node by the transport layer per unit time. Goodput per transport flow is normalized with the tight link bandwidth.

$$Goodput = \left(\frac{SentData - RetransmittedData}{TransferTime} \right) \quad (18)$$

• **Protocol Fairness:** This performance metric is defined by Jain [23] to show protocol fairness which is defined as the equality of the bandwidth sharing in a network. Mostly, Fairness is calculated by using Jain's fairness formula which is defined in Eq. 19. For a given set of throughput $(x_1, x_2, x_3, x_4, x_5, ,,,,,,,, x_n,)$, this formula calculates the fairness index. Fairness index is a value between 0 and 1, with 1 showing the most fair or equal allocating or sharing of available bandwidth among competing flows in a network [24]. Here throughput are non-negative, if the entire throughput are same, the fairness index will be 1.

$$f(x_1, x_2, x_3, ...x_n) = \frac{\left(\sum_1^n x_i \right)^2}{n \times \sum_1^n x_i^2} \quad (19)$$

There are two types of fairness: inter protocol fairness and intra protocol fairness. Inter protocol fairness measures the protocol fairness between two flows of a protocol having different RTTs. Intra protocol fairness measures the protocol fairness between two flows of a particular protocol with the same RTT. This performance metric represents a degree of bandwidth share between two flows of the same protocol.

• **TCP Friendliness:** TCP Friendliness relates to how sets of connections running different TCP variants affect the performance of each other. The TCP friendliness doctrine [25] states that a non TCP flow should not consume more available bandwidth than what a confirming TCP flow would consume under the same network conditions (RTT, Loss and Segment Size). Floyd and Mahdavi introduced the TCP friendly equation Eq. 20 in 1997.

$$BandwidthConsumed = \left(\frac{1.22 \times MSS}{RTT \times \sqrt{loss}} \right) \quad (20)$$

6 RESULTS AND DISCUSSION

6.1 Goodput Analysis

In this section goodput of TCP Reno, HighSpeed TCP, BIC TCP, TCP CUBIC and Compound TCP is measured according to Eq. 18. Goodput is measuring on long RTT and short RTT network scenarios having bottleneck bandwidth 50Mbps to 250Mbps and link speed 100Mbps to 500Mbps. In all experiments, different sets of BDP-Q values are used without any background traffic. Figure 8 shows the goodput comparison of TCP Reno, HighSpeed TCP, TCP CUBIC, BIC TCP and Compound TCP variants at bottleneck bandwidth 50Mbps along with link speed among nodes 100Mbps and different sets of bandwidth delay product queue (BDP-Q) sizes. In all scenarios, no any background traffic is introduced while measuring the goodput. There are two major groups of performance evaluation experiments, first group is about short round trip time (RTT) network scenarios having flow1 RTT is 50ms and second group is about long RTT scenarios where flow1 RTT is equal to 100ms. From simulation results it is observed that overall TCP CUBIC shows the highest goodput results whereas TCP Reno shows the lowest goodput results as shown in Figures 8a and 8d. In all experiment when competing flows have either similar or different RTT values, overall higher goodput of all the above TCP variants is noticed in long RTT experiments cases as compared to short RTT scenarios experiments.

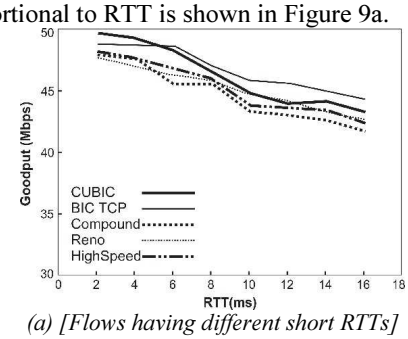
In Figures 8a and 8b, all the above five TCP variants show higher goodput when their individual flows have short RTTs and their RTT values are not similar to each other, whereas in Figures 8c and 8d, all the five TCP variants show higher goodput when their individual flows have long RTTs and their RTT values are similar to each other. Hence, this shows the reverse behavior of Reno, HighSpeed TCP, and BIC. CUBIC and Compound TCP variants in long and short RTTs network scenarios at bottleneck bandwidth 50Mbps along with link speed 100Mbps.

Figure 9 shows the goodput of Reno, HighSpeed TCP, BIC, CUBIC and Compound TCP variants, in this case goodput of CUBIC is again at high level as compared to other TCP variants whereas TCP Reno is again at lower level in goodput as

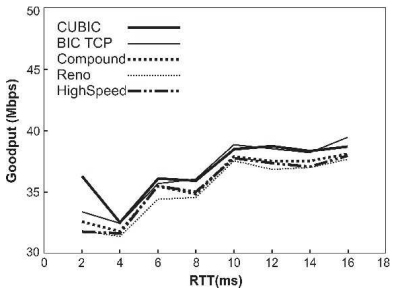
Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

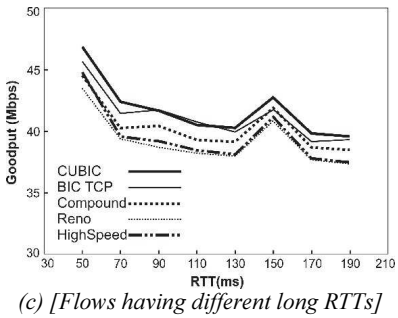
compared to others, this behavior of CUBIC and Reno is same as in Figure 8. It is also observed that goodput of all said TCP variant is higher in long RTT networks scenarios as compared to googput in short RTT case. When both the competing flows of each TCP variant have short RTTs and their RTT values are similar to each other, goodput of all variants increases if the RTT increases or in other words goodput is directly proportional to round trip time. However, when the flows have not similar round trip time values, goodput is inversely proportional to RTT is shown in Figure 9a.



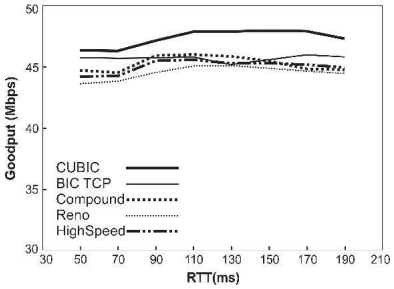
(a) [Flows having different short RTTs]



(b) [Flows having same short RTTs]



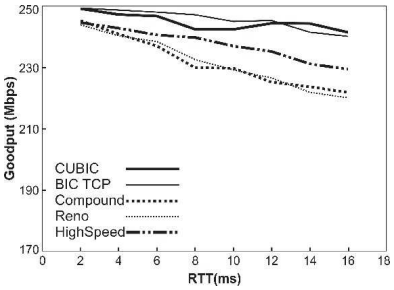
(c) [Flows having different long RTTs]



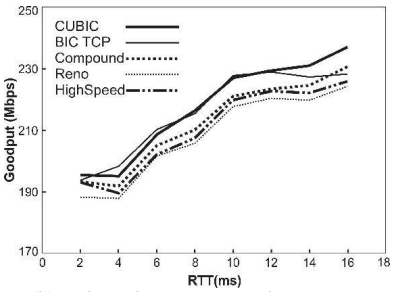
(d) [Flows having same long RTTs]

Figure 8: Goodput comparison of two flows. Setup: Bottleneck bandwidth is 50Mbps, link rate is 100Mbps, 10 sets of BDP buffering are used, 50ms short RTT, 100ms long RTT and No Background Traffic is introduced

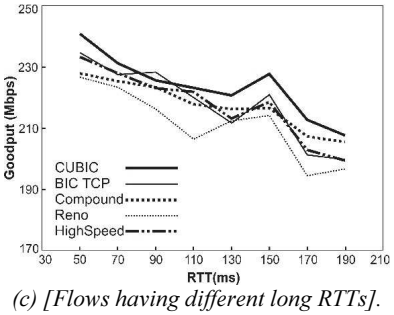
In long RTT network scenarios, when the round trip time of both flows are similar to each other, then CUBIC shows very high goodput while Compound goodput constantly decreases with in increase in RTT as shown Figure 9d.



(a) [Flows having different short RTTs]

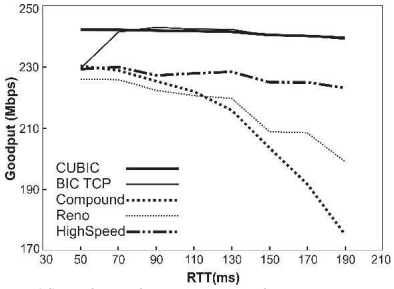


(b) [Flows having same short RTTs]



(c) [Flows having different long RTTs].

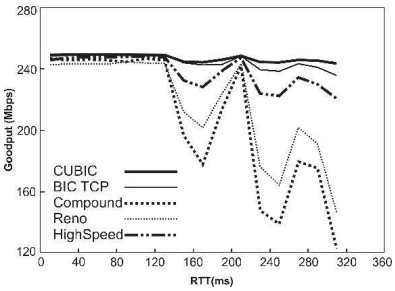
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



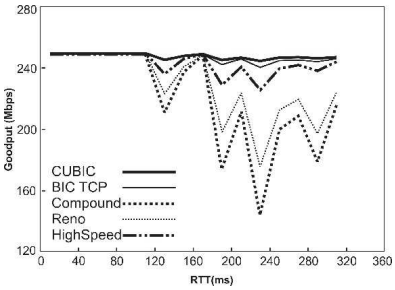
(d) [Flows having same long RTTs]

Figure 9: Goodput comparison of two flows. Setup: Bottleneck bandwidth is 250Mbps, link rate is 500Mbps, 10 sets of BDP buffering are used, 50ms short RTT, 100ms long RTT and No Background Traffic is introduced

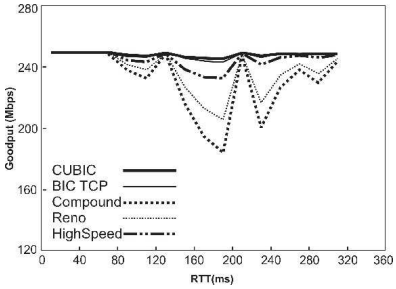
Figures 10 and 11 show the goodput behavior of Reno, HighSpeed TCP, BIC, CUBIC and Compound TCP variants with respect to bandwidth delay product queue (BDP-Q) size. All the experiments are divided into two different network groups having similar and non similar flows RTTs with bottleneck bandwidth 250Mbps and link speed 500Mbs. Simulation parameters of these two network groups are shown in Tables 1 & 2. TCP CUBIC and BIC TCP show similar goodput values



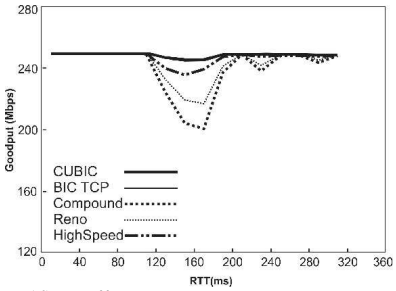
(a) [Buffering queue size is 0.5 BDP]



(b) [Buffering queue size is 1.0 BDP]

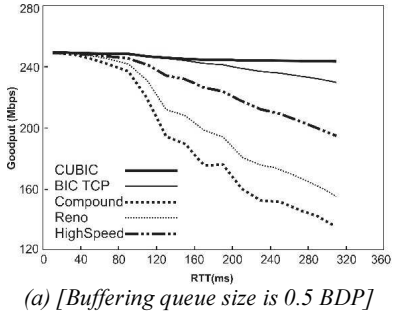


(c) [Buffering queue size is 1.5 BDP]

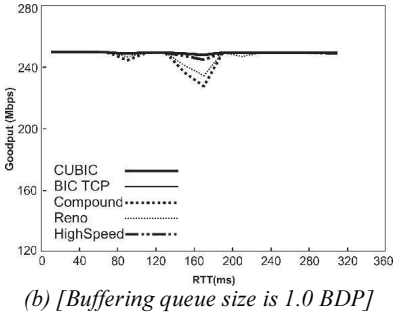


(d) [Buffering queue size is 2.0 BDP]

Figure 10: BDP Q-wise Goodput comparison of two flows having same RTTs. Setup: Bottleneck bandwidth is 250Mbps, link rate is 500Mbps, BDP buffering is configured, 160ms flow 1 RTT and No Background Traffic is introduced

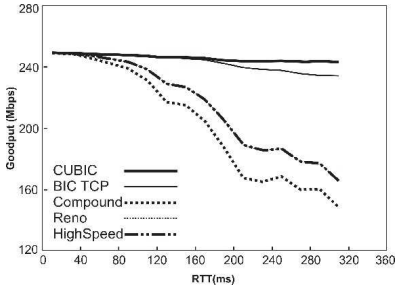


(a) [Buffering queue size is 0.5 BDP]

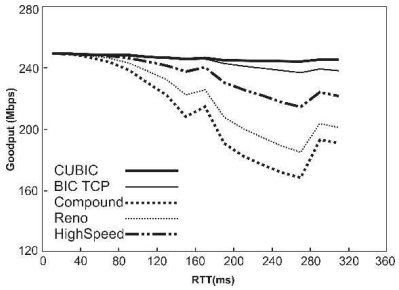


(b) [Buffering queue size is 1.0 BDP]

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



(c) [Buffering queue size is 1.5 BDP]



(d) [Buffering queue size is 2.0 BDP]

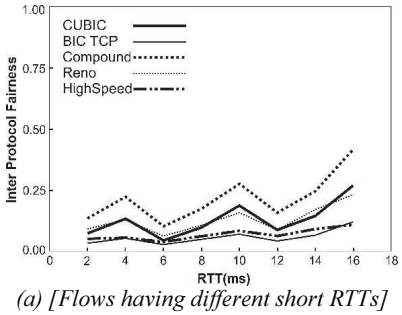
Figure 11: BDP Q-wise Goodput comparison of two flows having different RTTs. Setup: Bottleneck bandwidth is 250Mbps, link rate is 500Mbps, BDP buffering is configured, 160ms flow 1 RTT and No Background Traffic is introduced

at [0.5-2.0] BDP-Q as shown in Figures 10 and 11. Results show that at BDP-Q size 1.0 and when RTTs of both the competing flows are not similar to each other, all the said five TCP versions can achieve similar highest goodput as shown in Figure 11b. It is also observed that goodput of Compound, Reno and HighSpeed TCP is inversely proportional to flows round trip time, specially when both flows have similar RTTs as shown in Figures 11a, 11c and 11d.

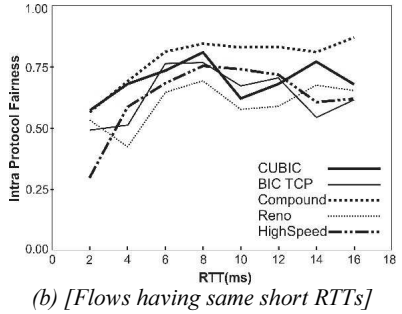
From Figure 10, it is noticed that if the competing flows have [10-120]ms round trip time, then goodput of all five said TCP variants is almost equal showing in one line, whereas in Figure 11, goodput is exactly same at round trip time [10-80]ms. Overall it is also observed that, goodput of each TCP variant is also increase, if the queue size of bandwidth delay product (BDP) increases. It is also observed that goodput of Compound, Reno and HighSpeed TCP decreases, if the round trip time of competing flows increase as shown in Figures 11a, 11c and 11d.

6.2 Protocol Fairness Analysis

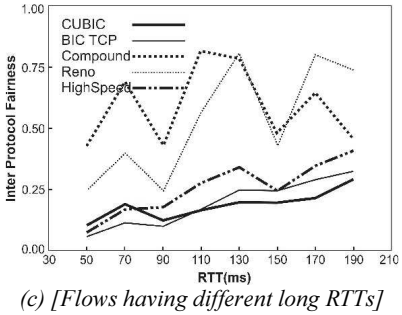
Inter protocol fairness measures the protocol fairness between two flows of a protocol having different RTTs. Intra protocol fairness measures the protocol fairness between two flows of a particular protocol with the same RTT. This performance metric represents a degree of bandwidth share between two flows of the same protocol. Inter and Intra protocol fairness between two individual competing flows of Reno, HighSpeed TCP, CUBIC, BIC and Compound TCP versions are measured in this section. Experiments are performed on short and long RTT network scenarios. For short RTT experiments, flow1 RTT is fixed which is equal to 50ms and flow2 RTT vary from 2ms to 16ms. For long RTT experiments, flow1 RTT is fixed which is equal to100ms instead of 50ms and flow2 RTT starting and ending range is 50ms and 190ms respectively. A major difference between the fairness of long RTTs and short RTTs networks is noticed.



(a) [Flows having different short RTTs]



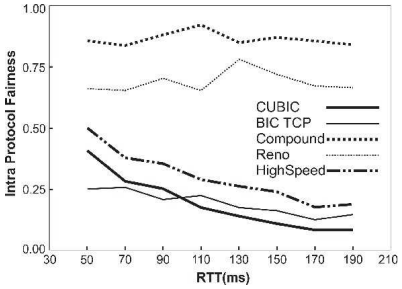
(b) [Flows having same short RTTs]



(c) [Flows having different long RTTs]

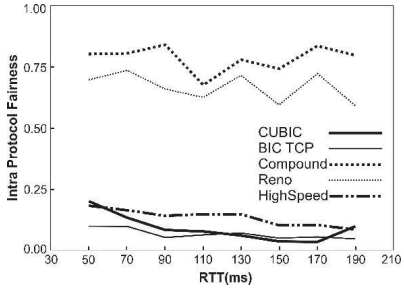
Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



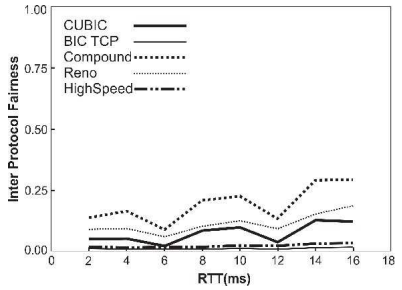
(d) [Flows having same long RTTs]

Figure 12: Fairness comparison of two flows. Setup: Bottleneck bandwidth is 50Mbps, link rate is 100Mbps, 10 sets of BDP buffering are used, 50ms short RTT, 100ms long RTT and No Background Traffic is introduced

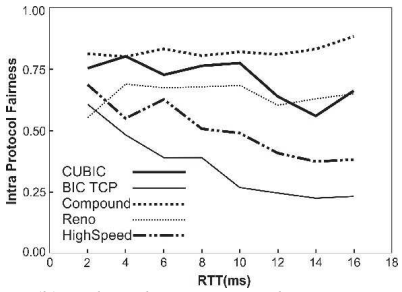


(d) [Flows having same long RTTs]

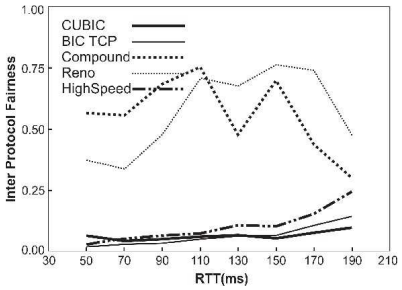
Figure 13: Fairness comparison of two flows. Setup: Bottleneck bandwidth is 250Mbps, link rate is 500Mbps, 10 sets of BDP buffering are used, 50ms short RTT, 100ms long RTT and No Background Traffic is introduced



(a) [Flows having different short RTTs]



(b) [Flows having same short RTTs]



(c) [Flows having different long RTTs]

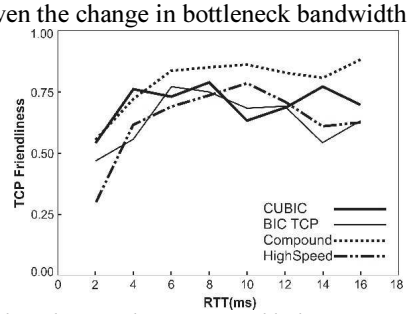
According to Figures 12 and 13, it is observed that TCP Compound which is the default TCP version in Microsoft operating system visa and 7, shows the highest inter and intra protocol fairness in both long and short round trip time network cases. It is also observed that when both the competing flows are configured at long RTTs having bottleneck bandwidth [50-250]Mbps, link speed [100-500]Mbps and without presence of any background traffic, TCP CUBIC which is default TCP protocol in Linux operating system, shows lowest inter and intra protocol fairness as shown in Figures 12c, 12d, 13c and 13d. Reno, HighSpeed TCP, CUBIC, BIC and Compound TCP variants show pathetic low inter protocol fairness as shown in Figures 12a and 13a, when their individual competing flows are configured with short round trip time and their RTT values are not similar to each other, but if both the flows have similar RTT values, all the above said five variants show pathetic high intra protocol fairness as shown in Figures 12b and 13b. In Figures 12d, 13c and 13d, Reno and Compound show very high fairness and a clear gap among the fairness of these two variants with CUBIC, BIC and HighSpeed TCP variants is observed, in this same figure, it is also seen that CUBIC, BIC and HighSpeed TCP variants present almost similar fairness but very low as compared to Reno and TCP Compound as discussed above. Finally it is concluded that all the five TCP protocols show very high fairness when their flows have short RTTs and values of their RTTs are not similar to each other.

6.3 TCP Friendliness

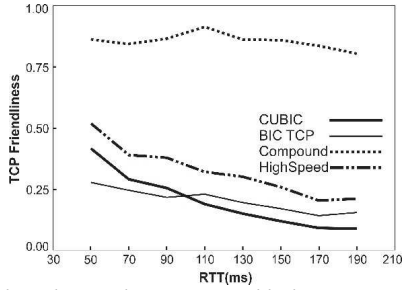
It calculates the bandwidth consumed by a TCP flow confirming with the TCP congestion control algorithms. [26] extended the Eq. 20 in 1998 to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

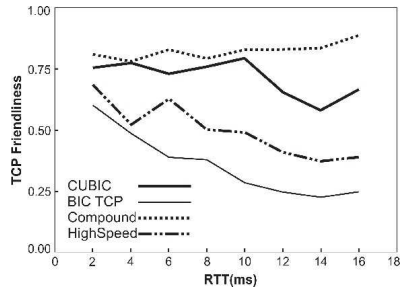
include time out events. In this section friendliness behavior of TCP CUBIC, BIC TCP, TCP Compound and HighSpeed TCP with Standard TCP is measured and from the results, it is found that Compound TCP, which is the default congestion control protocol in Microsoft Vista / 7, show the highest TCP friendliness in all cases of NS-2 experiments of long and short RTT network scenarios having bottleneck bandwidth 50Mbps to 250Mbps and link speed 100Mbps to 500Mbps as shown in Figure 14. All experiments regarding TCP friendliness are run according to simulation parameters defined in Tables 1 & 2. When competing flows of each TCP variant (TCP CUBIC, BIC TCP, Compound TCP and HighSpeed TCP) are short, bottleneck bandwidth is low (50Mbps) and link speed is also low (100Mbps), then all the said four TCP variants behave friendly with Standard TCP as shown in Figure 14a. Results show that TCP CUBIC, BIC TCP and HighSpeed TCP are not friendly with Standard TCP when round trip time of competing flows are very long and even the change in bottleneck bandwidth and link speed cannot improve their TCP friendliness behavior. So it is concluded that these three TCP variants cannot equally share bandwidth with Standard TCP in any network condition when nodes have long RTTs as shown in Figures 14b and 14d.



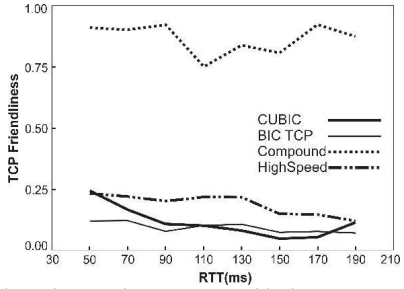
(a) [Flows having short RTTs and link rate is 100Mbps]



(b) [Flows having long RTTs and link rate is 100Mbps]

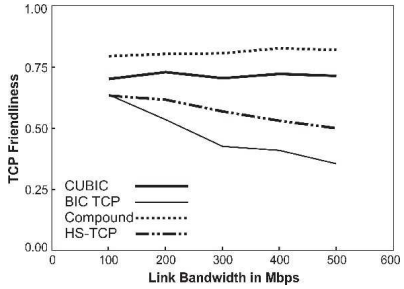


(c) [Flows having short RTTs and link rate is 500Mbps]

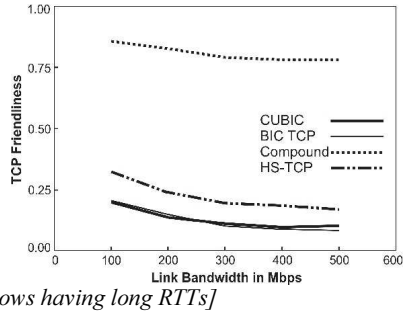


(d) [Flows having long RTTs and link rate is 500Mbps]

Figure 14: Friendliness comparison of two flows. Setup: Bottleneck bandwidth is [50,250]Mbps, link rate is [100,500]Mbps, 10 sets of BDP buffering are used, 50ms short RTT, 100ms long RTT and No Background Traffic is introduced



(a) [Flows having short RTTs]



(b) [Flows having long RTTs]

Figure 15: Link speed wise friendliness comparison of two flows. Setup: Bottleneck bandwidth is [50 to 250]Mbps, link rate is [100 to 500]Mbps, 10 sets of BDP buffering are used, 50ms short RTT, 100ms long RTT and No Background Traffic is introduced

Results show that Compound TCP is the only one TCP variant out of four variants that shows very high TCP friendliness performance even in long RTT network cases as shown in Figures 14b and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

14d. It is also observed that TCP friendliness behavior of BIC TCP is very poor in short RTT network cases when link bandwidth is very high (500Mbps) as shown in Figure 14c**.** Finally it is concluded that all the above four variants show highest TCP friendliness performance at during short RTTs, lower bottleneck bandwidth and lower link speed experiments as shown in Figure 14a.

Figures 15a and 15b represent the TCP friendliness behavior of TCO CUBIC, BIC TCP, Compound TCP and HighSpeed TCP variants with respect to link speed in Mbps. Figure 15a represents the variants friendliness in short RTTs network configurations whereas Figure 15b shows the friendliness of variants when their individual flows have long RTTs. TCP Compound is at highest point in performance regarding to TCP friendliness in both short and long RTT cases. TCP CUBIC shows very good TCP friendliness in short RTTs case whereas it shows very poor friendliness performance in long RTTs network scenarios. TCP CUBIC and BIC TCP behave exactly similar friendliness behavior with Standard TCP in long RTTs networks as shown in Figure 15b. Finally it is concluded that all four TCP variants show very high TCP friendliness in short RTTs networks as compared to long RTTs network scenarios.

7 CONCLUSION

Results show that, TCP CUBIC shows the highest while TCP Reno shows the lowest goodput results in all experiments. TCP Reno, HighSpeed TCP, TCP CUBIC, BIC TCP and TCP Compound show reverse goodput behavior in long and short RTTs networks with respect to round trip time. Goodput of all above five variants is higher in long RTT networks as compared to short RTT scenarios. When both the competing flows of each TCP variant have short RTTs and their RTT values are similar to each other, goodput of all variants increases if the RTT increases. In other words goodput is directly proportional to round trip time. However, when the flows have not similar round trip time values, goodput is inversely proportional to RTT. TCP CUBIC and TCP BIC show similar goodput results at [0.5 to 2.0] BDP -Q size. Goodput of TCP Compound, TCP Reno and HighSpeed TCP is inversely proportional to flows round trip time, specially when both flows have similar RTTs. Goodput of each TCP variant is also increase, if the queue size of bandwidth delay product (BDP) increases. TCP Compound shows the highest while TCP CUBIC shows the lowest inter and intra protocol fairness in both long and short distance BDP networks. TCP Reno, HighSpeed TCP, TCP CUBIC, BIC TCP and TCP Compound show pathetic low inter protocol fairness when their individual competing flows are configured with short round trip time and their RTT values are not similar to each other, however, if both the flows have similar RTT values, all the above said five variants show very high intra protocol fairness. Finally it is concluded that all the five TCP protocols show very high protocol fairness when their flows have short RTTs and values of their RTTs are not similar to each other. TCP Compound shows the highest TCP friendliness in all cases of short or long distance BDP network scenarios, whereas TCP CUBIC, BIC TCP and HighSpeed TCP variants are not friendly with Standard TCP. Finally it is concluded that all the above four variants show highest TCP friendliness performance at during short RTTs, lower bottleneck bandwidth and lower link speed experiments. TCP CUBIC shows very good TCP friendliness in short RTTs case whereas it shows very poor friendliness performance in long RTTs network scenarios. TCP CUBIC and BIC TCP behave exactly same friendliness behavior with Standard TCP in long RTTs networks. Finally it is concluded that all four TCP variants show very high TCP friendliness in short RTTs networks as compared to long RTTs network scenarios. In future, we will enhance the performance of TCP CUBIC regarding protocol fairness and TCP friendliness and goodput.

ACKNOWLEDGMENTS

This work is supported by Ministry of Higher Education (MOHE) and UTM/RUG/04H11 Universiti Teknologi Malaysia. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFRENCES:

[1] Allman, M., Paxson, V., Stevens, W. et al. (1999). TCP congestion control.

[2] V. Jacobson, Modified TCP congestion avoidance algorithm, email to the end2end list, April 1990.

[3] Floyd, S. (2003). HighSpeed TCP for large congestion windows.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] Xu, L., Harfoush, K. and Rhee, I. (2004). Binary increase congestion control (BIC) for fast long-distance networks. In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4. IEEE, 2514-2524.

[5] Ha, S., Rhee, I. and Xu, L. (2008). CUBIC: A new TCP-friendly high-speed TCP variant. ACM SIGOPS Operating Systems Review. 42(5), 64-74.

[6] Song, K., Zhang, Q. and Sridharan, M. (2006). Compound TCP: A scalable and TCPfriendly congestion control for high-speed networks. Proceedings of PFLDnet 2006.

[7] Jacobson, V. (1988). Congestion avoidance and control. In ACM SIGCOMM Computer Communication Review, vol. 18. ACM, 314-329.

[8] McKusick, Marshall Kirk and Neville-Neil, The design and implementation of the FreeBSD operating system, George V, 2004, Addison-Wesley Professional

[9] Floyd, S.; Henderson, T. & Gurtov, A. The NewReno modification to TCP's fast recovery algorithm RFC 2582, April, 1999

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanov, RFC2018TCP selective acknowledgment options RFC, 1996.

[11] Floyd, S.; Handley, M. & Padhye, J. A comparison of equation-based and AIMD congestion control Citeseer, 2000

[12] The Network Simulator 2 (NS-2) Web page. http://www.isi.edu/nsnam/ns/.

[13] V. Jacobson, Congestion avoidance and control, ACM SIGCOMM, pp. 314-329, 1988.

[14] Kerkar, S. (2004). Performance analysis of TCP/IP over high bandwidth delay product networks.

[15] Li, Y.-T., Leith, D. and Shorten, R. N. (2007). Experimental evaluation of TCP protocols for high-speed networks. Networking, IEEE/ACM Transactions on. 15(5), 1109-1122.

[16] Ha, S. and Rhee, I. (2011). Taming the elephants: New TCP slow start. Computer Networks.

[17] Brakmo, L. and Peterson, L. (1995). TCP Vegas: End to end congestion avoidance on a global Internet. Selected Areas in Communications, IEEE Journal on. 13(8), 1465-1480.

[18] Touch, Joe, TCP control block interdependence, 1997

[19] Hamilton Institute TCP benchmark suite. http://www.hamilton.ie/net/tcptesting.zip.

[20] Weigle, Michele C and Adurthi, Prashanth. A tool for generating realistic TCP application workloads in ns-2, ACM SIGCOMM Computer Communication Review, year 2006, volume 36, pages 65–76, number 3, ACM.

[21] Wood, Lloyd, Awk script to get end-to-end delays from NS2 tracefiles.

[22] Bryman, Alan and Cramer, Duncan. Quantitative data analysis with IBM SPSS 17, 18 & 19: A guide for social scientists, Routledge, 2011.

[23] Jain, Raj, The art of computer systems performance analysis, John Wiley & Sons Chichester, 1991, Vol-182

[24] Aydin, Ilknur and Iyengar, Janardhan and Conrad, Phillip and Shen, Chien-Chung and Amer, Paul, Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, journal = Computer Networks, year = 2012, volume = 56, pages = 1876-1892, number 7, Elsevier

[25] S. Floyd, K. Fall, Promoting the use of end-to-end congestion control in the internet, IEEE/ACM Transactions on Networking 7 (1999) 458-472.

[26] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP Throughput: A Simple Model and its Empirical Validation, ACM SIGCOMM (1998) 303-314.