SURVEY ON END TO END CONGESTION CONTROL TECHNIQUES IN DIFFERENT NETWORK SCENARIOS

[1]USMAN AHMAD, [2]DR. MD ASRI BIN NGADI, [3]DR. ISMAIL FAUZI BIN ISNIN
[1, 2] Department of Computer Science, Faculty of Computing,
Universiti Teknologi Malaysia (UTM), 81310 Johor, Malaysia.

E-mail : usman.ahmad82@yahoo.com, dr.asri@utm.my, ismailfauzi@utm.my

ABSTRACT

Most of the traffic on the Internet is depend upon the Transmission Control Protocol (TCP), so the performance of TCP is directly related to Internet. Many TCP variants are developed and modified according to the environment and communication needs. Most of current TCP variants have set of algorithms which control the congestion in critical situations and maintain the throughput and efficiency of network. Now a day's TCP is facing fast growth of Internet with the demands of faster data communication techniques on high speed links. In last 15 years many computer systems and cellular networks become linked together with protocol stack used in TCP. TCP variants with different congestion control techniques are working in different operating systems but a very small number of techniques are able to minimize the congestion in the network. This paper presents a survey on end-to-end congestion control techniques used in different TCP versions. The main purpose of this study is to review the characteristics and behavior of TCP variants with different techniques to control the congestion in the different network scenarios.

Keyword: *TCP Variants, Congestion Control, Internet, High Speed TCP*

1. INTRODUCTION

Transmission Control Protocol (TCP) [1] is a connection oriented protocol which provides the end-to-end reliable data communication among the different networks. Now almost all traffic of Internet is carried by TCP, so the behavior of TCP is coupled with the overall Internet performance. Many TCP variants are proposed to improve the efficiency of the network. This paper presents the survey of congestion control proposals depending upon it fundamental behavior with its end-to-end principal.

The proposed approaches described the wide range of techniques that allow the source (sender) to detect loss events, changes in router, state of congestion, *RTT* variations and bottleneck buffer sizes. The major responsibility of TCP is its ability to provide reliable data transmission between two hosts on the Internet. TCP standard specifies the sliding window for the flow control of data having different mechanisms inside it. First is, all the data stored (buffered) before sending by assigning a sequence number to each stored byte and then stored data is packetized into TCP packets for transmission. Second, a window (portion) containing data need to transmit towards the receiver using IP protocol.

When sender side receives an acknowledgment of data from receiver (destination) side then sender transmits new portion of data which is also called sliding window. In short it means that sender side holds the responsibility to block the transmission until receiver sends acknowledgment.

Sliding window based on data flow control is a simple mechanism but there are many conflicting concepts inside it, for a example TCP must have maximum sliding window size (congestion window) for the maximum throughput but on the other hand if the congestion window is too large then there are many chances of packet losses because receiver and network have limited resources. Thus need minimum congestion window for minimum packet losses. Therefore, the problem is to find optimal value for the sliding window which provides good throughput and does not overwhelm the receiver and network. Additionally, TCP should be able to detect packet losses timely, which means the interval should be shorter between data transmission and loss detection. However this interval should not be too short that TCP detects loss prematurely and retransmits the packet unnecessarily which cause congestion and wastage of the network resources. So when and how sender side detects loss is a big issue for TCP. Initially TCP is designed to protect the input buffer of receiver side from overflowing. The technique is based on receiver's window concept which is a way for the receiver window to share the information about available buffer size (storage) with sender side. The size of the sender window should not be exceeding than the size of receiver's window. Another case when the receiver side cannot be able to process data as fast as sender generates then receiver sends a signal to sender for reducing the transmission rate or value of window.

To control the congestion in the network is an extensive studied area over the last 20 years and numbers of proposals are proposed aimed to control the congestion with different techniques. Many proposals related to congestion control are studied by Hanbali et al [2] (congestion control in ad hoc networks), Lochert et al [3] (congestion control mobile networks), Widmer et al [4] (congestion control for non-TCP protocols) Balakrishnan et al [5] (congestion control for wireless networks), Leung at al [6] (congestion control for network with high levels of packet reordering), Low et al [7] (current up to 2002 TCP variants and their analysis), Hasegawa and Murata [8] (fairness issues in congestion control).

In this review we tried to describe, classify and analyze the major Congestion Control techniques which optimize the various parameters of TCP. In section II those congestion control techniques are discussed which built a foundation for the end to end congestion control algorithms. This foundation is consisting on probing the network resources, loss and delay base techniques for the detection of congestion and mechanisms for the detection of packet loss quickly. In section III we discussed various congestion control techniques which are proposed to solve poor network utilization in high speed bandwidth networks.

2. CONGESTION COLLAPSES

Congestion occurs in the network when the demand (load of data injected into the network) is increased or closed to capacity of the network. Due to congestion in the network, (i) the throughput of the given path decreases, (ii) packet delay increases in the network and (iii) the rate of packet drop increases. So in these conditions there is no any technique to reduce the congestion and sate of network can be move towards the congestion collapses and all end to end communication stops.

Congestion collapse is introduced and described as a problem for TCP based network by Nagle in 1984 [9]. The first congestion collapse was observed when the rate of throughput decreased to pathetic level between UC Berkely and Lawrene Berkeley Labs [8]. The original TCP algorithm has only flow control technique to prevent the network from over flowing. TCP did not have any technique to control or reduce the load (traffic) in the network, when the network is in the state of congestion. In 1988 V.Jacobson. et al introduced many algorithms based on AIMD (*Additive Increase Multiplicative Decrease*) and packet conservation principle technique to prevent network by congestion collapse [8]. The conservation principle means when the system is running in stable state with full data transmission rate then no new packet will enter until old packet leaves the system [8].

Jacobson used the ACK clocking technique for the estimation of old packet when it leaves network so new packet can be able to enter for transmission and when the packet loss occurs then the source reduces the transmission rate by half of the last sending rate (multiplicative decrease) Chiu a Jain [10] described if all TCP sources use multiple decrease technique then the network will not move towards the congestion collapse moreover the TCP flow will use fair allocation of bandwidth on the given path. Below many TCP variants are discussed to solve the problem of congestion, moreover in

Table 1 the summary of these TCP Variants are discussed.

2.1 TCP Tahoe

The earliest end to end solution for congestion problem in TCP has been proposed by Jacobson [11]. This solution is based on TCP specification [1] having number of algorithms which can be divided into three techniques, slow start, congestion avoidance and fast retransmit. Tahoe algorithm is based on principle of "*conservation of packets*", that means during the packet transmission new packet cannot be injected into the network unless a packet is taken out. TCP Tahoe congestion control technique is proposed by modify the original TCP slow start, congestion avoidance and fast retransmit procedures. Tahoe is also proposed new window mechanism containing data called congestion window (*cwnd*). The sender side of TCP cannot transmit the data more than the size of congestion window and advertised receiver window (*window=min(cwnd, rwnd))* where the *cwnd* is congestion window and *rwnd* is receiver window.

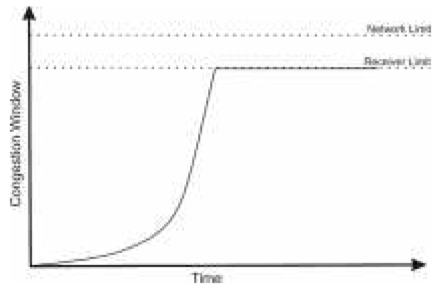

Figure 1: Behavior of congestion window in slow start phase

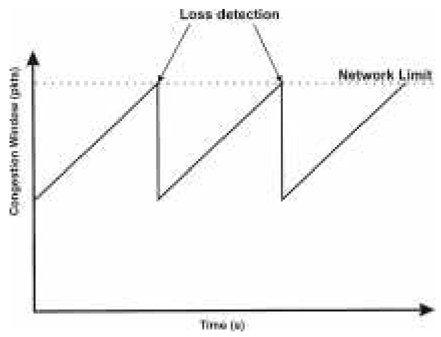

Figure 2: Behavior of congestion window in congestion avoidance phase after packet loss

The size of congestion window is set to one segment after establishing the connection and increased by one segment after receiving each acknowledgment which is exponential increment in the size of congestion window. This procedure is called *Slow Start*, shown in Figure 1 and after packet loss or on reaching to predefined threshold *ssthresh* it enters in congestion avoidance phase as shown in Figure 2. After some time when the capacity of the network reached and an intermediate router will start to drop packets then Tahoe uses two different mechanisms to detect packet loss. First is retransmission time out (RTO) and second is at the arrival of three duplicate acknowledgments. If the packet loss is detected by three duplicate acknowledgments then Tahoe enters in the fast retransmission phase and resend the dropped or lost packet. In TCP Tahoe after indication of congestion the current size of congestion window is halved and saved in a variable called slow start threshold *ssthresh* after that sender side sets the size of congestion window to one segment and starts the slow start mechanism by increasing the congestion window exponentially and when the value of congestion window reaches to predefined *ssthresh* then the congestion window increases approximately by one segment per round trip time (*RTT*) which is also called linear increment in congestion window.

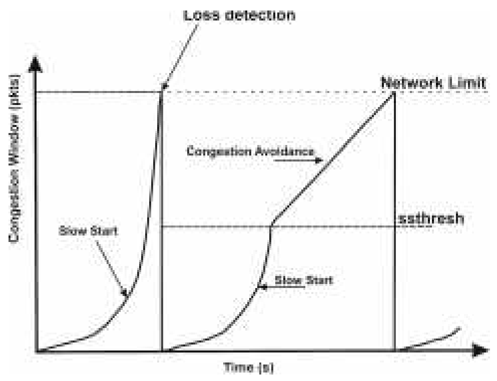

Figure 3: Slow Start and Congestion Avoidance cycle of congestion window

This whole procedure is called congestion avoidance. The process in which the algorithm's phase switches from Slow Start phase to Congestion Avoidance phase is called slow start congestion avoidance cycle which is shown in

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

Figure 3. In congestion avoidance phase the algorithm tries to increase the size of congestion window in a more conservative manner than slow start phase. Tahoe was major breakthrough in congestion control and set the guidelines and principles for the new TCP implementation.

2.1 DUAL TCP

TCP Tahoe gave great service to the Internet community by solving the congestion in the network, but in this solution there are many drawbacks, the behavior of Tahoe algorithm induced significant changes in network buffer utilization, round trip time and packet losses.

DUAL TCP is proposed by Wang and Croweroft [12]. It tries to refine the congestion avoidance algorithm. DUAL TCP tries to reduce the oscillatory patterns by using a proactive congestion detection technique. Moreover, it proposed a queuing delay parameter for detecting congestion state in the network. Let us assume that there is no change in intermediate router and receiver side acknowledge each packet immediately that means the state of network is congestion free, and RTT_{min} is a minimum *RTT* noted by sender which indicates that network is in congestion free state. If we consider one more assumption that the value of *RTT* is only increase with the increment of buffer utilization and the difference between minimum *RTT* and measured *RTT* value (queuing delay Q = ($RTT - RTT_{min}$) can be used as indicator to measure congestion level on the given path as shows in Figure 4, moreover DUAL TCP algorithm also measure the maximum *RTT* (RTT_{max}) value to quantify the congestion level in the network and the difference between minimum and maximum *RTT* value is indication of maximum congestion level (maximum queuing delay $Q_{max} = (RTT_{max} - RTT_{min})$. Finally the fraction of maximum queuing delay is worked as threshold in DUAL algorithm and when this threshold exceeded, indicates that the state of network is congested. In this proposed algorithm the value of delay threshold is half on the value of maximum queuing delay ($Q_{thresh} = Q_{max}/2$) and the level of congestion estimation is performed per *RTT* on the basis of average *RTT* value ($Q = RTT_{avg} - RTT_{min}$). If the average threshold is exceeded $Q > Q_{thresh}$ then the value of congestion window decrease by 1/8 (multiplicative decrease factor) which is shown in Figure 5.

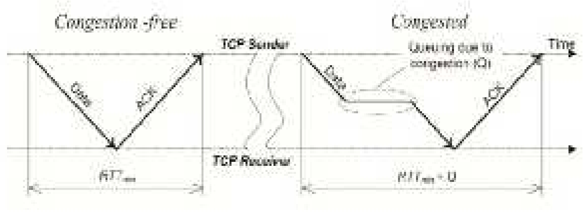

Figure 4: Relation between RTT and congestion

In any TCP the flows cannot be able to utilize available network resources fully, affectively and fairly if the saturation point of the network is estimated incorrectly. As far as TCP DUAL is concerned if the value of threshold is underestimated (Calculation of RTT_{max} is wrong) then the network resources will be unutilized, on the other hand if the threshold is over estimated then it cause unfair distribution among the different TCP flows. For a example, when a new DUAL TCP flow will appear during the transmission of other DUAL flows, in this case the new flow will observe the higher RTT_{min} value and over estimate the value of queuing delay threshold and the flows having lower queuing delay (previous flows) have high chances to Predict the congestion state and trigger the congestion window reduction. During this the other flow (new flow) will continuing increasing its congestion window size without noticing any abnormality in network. The new flow can able to get large share of the network resources as compared to old flows.

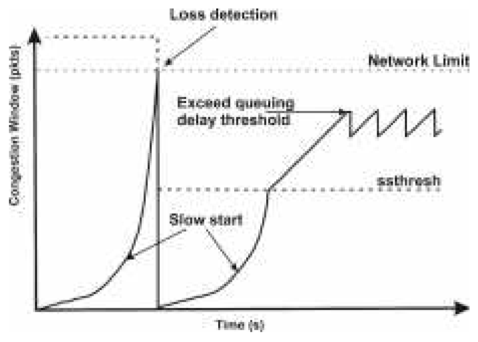

Figure 5: Congestion window behavior of TCP DUAL

2.3 TCP Reno

After detection of packet loss the size of congestion window is reduced up to one packet in TCP Tahoe, this behavior of congestion window leads towards significant degradation of throughput in some cases. For an example 1 percent packet loss during the transmission may cause 75 percent throughput degradation of TCP flows in Tahoe. For the solution of this problem Jacobson revised the slow start and congestion avoidance techniques and proposed a mechanism which diffrencient the major and minor congestion events. Packet loss detection by retransmission time indicates that in certain time interval *(RTO - RTT)* some major congestion event has prevented and sender side should decrease the size of congestion window up to minimal value. Packet loss can also be detected by duplicate acknowledgments. For an example sender side received four ACKs first ACK containing new data and other three ACK are the copies of first one (duplicate ACK), these duplicate ACKs are the indication of some packet loss or does not arrive. In addition the sender side is not only detecting the packet loss and also observing the delivery of data. So packet loss is the indication that the network is congested and the reaction of the loss can be optimistic.

In TCP Reno fast recovery technique is used this optimistic reaction [13] and the typical behavior of its congestion window is shown in Figure 6. In the fast recovery the size of congestion window is halved and stops all increment in congestion window until the recovery of error.

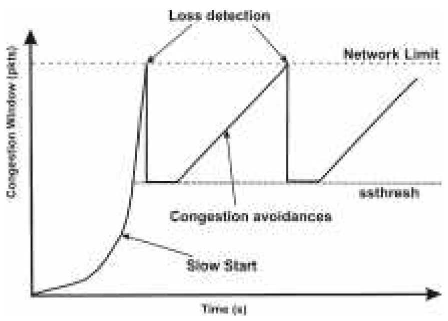

Figure 6: Congestion window behavior of TCP Reno

In other words the sender side stays in fast recovery mode until it receives non duplicate ACK. The stages of algorithm are shown in Figure 7, where the size of congestion window is described in different phases. In Figure 7 the reduction of congestion window by multiplicative decrease factor is shown, which describes the core concept of optimistic network share reduction. After the reduction in size of congestion window *(cwnd to cwnd/2)* the algorithm not only retransmits the unacknowledged data (fast retransmission) but also fills the congestion window with duplicate packets (from state 2 to 3 in Figure 7).

We know that one ACK means at least one packet has delivered, so for the transmission of constant number of packets we have to open a slot in congestion window for transmission of new data (state 4). This slot in congestion window is very important because in the absences of this algorithm TCP cannot be able to send new data before the recovery of the error and the amount of data can be decrease during the transmission. When the sender side receives non duplicate ACK (in state 5) then the algorithm turned towards the termination of congestion avoidance phase by reducing the size of congestion window by half of the original size. This reduction in congestion window is reliable and simple way to ensure the exit from fast recovery.

In short in TCP Reno the fast retransmit and fast recovery techniques are allows the connection to quickly recover the packet losses. But during the multiple packet losses TCP Reno flow may suffer from severe performance degradation because algorithm retransmits the dropped packet per *RTT* and further the size of congestion window may be decrease more than one time. In this case TCP Reno works at very low rate and decrease the throughput significantly. In TCP Reno a single loss is recovered per *RTT*, which improved not only the efficiency of recovering period but also continue the transmission of new data during the recovery.

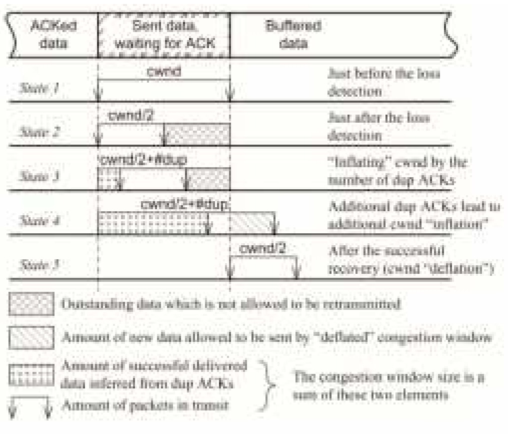
ACKed data
Sent data, waiting for ACK
Buffered data
cwnd
State 1
Just before the loss detection
cwnd/2
State 2
Just after the loss detection
cwnd/2+#dup
State 3
"Inflating" cwnd by the number of dup ACKs
cwnd/2+#dup
State 4
Additional dup ACKs lead to additional cwnd "inflation"
cwnd/2
State 5
After the successful recovery (cwnd "deflation")
Outstanding data which is not allowed to be retransmitted
Amount of new data allowed to be sent by "deflated" congestion window
Amount of successful delivered data inferred from dup ACKs
Amount of packets in transit
The congestion window size is a sum of these two elements

Figure 7: Stages of congestion window of TCP Reno

2.4 TCP New Reno

TCP New Reno was proposed and released in 1996. It is a new version of TCP Reno with some modifications in fast re-transmit and fast-recovery of TCP Reno. These sender side modifications are proposed to overcome the Reno's problems. Partial acknowledgments are used in TCP Reno to exit from fast recovery phase but it creates re-transmission time out (RTO) during the multiple packet losses. Instead of this, in TCP new Reno the sender side does not exit from fast recovery mode after receiving partial ACKs [14] [15]. It assumes that a packet just after an acknowledged is lost and re-transmits this lost packet. So during the multiple packet losses the algorithm recover only one packet per *RTT* in same congestion window by avoiding multiple fast re-transmit of lost packets which creates the re-transmission time out *(RTO).* TCP new Reno continues the fast recovery process until all the loss packets have acknowledged. The implementation of mechanism in TCP New Reno is described below.

2.4.1 Multiple Packet Losses

In TCP New Reno, packets having only high sequence number will be sent during the fast re-transmission process. New Reno uses the same mechanism of re-transmission and fast recovery of TCP Reno. However the difference between New Reno and Reno is when a new ACK is received by sender, new Reno algorithm checks if the ACK have highest sequence number, when the fast re-transmission mode was start then continue the transmission and if the sequence number is not highest then New Reno consider this ACK as Partial ACK that means another packet is lost in the same congestion window so it re-transmit the dropped packet and re set the re-transmission timer with exiting the fast recovery mode. On the other hand new ACK received by sender having the highest sequence number the New Reno algorithm terminates the fast recovery mode and set the size of congestion window to the slow start threshold (*ssthresh*) and perform congestion avoidance process.

2.4.2 False Fast Recovery

New Reno always records the highest sequence number even transmitted packets after re-transmission time out *(RTO).* After receiving the duplicate ACK, New Reno runs a test to determined whether it should enter in fast recovery mode or not. If these ACKs have the same sequence number which saved at the previous time out, then this is a new start of fast recovery and New Reno enters in fast recovery mode and perform related tasks and if the sequence number is not similar as saved before, then New Reno will not enter in fast recovery mode. There is a small change in connection and source in New Reno witch eliminate the waiting time of TCP Reno for the re-transmission time out during the multiple packet losses in the same congestion window. TCP New Reno can avoid the unnecessary congestion window reduction but it may take many *RTTs* for recovery of lost packets.

2.5 TCP SACK

There is a another way to overcome the multiple packet losses during the transmission is to inform the sender side about all acknowledged packets and TCP SACK uses this mechanism. TCP SACK uses accumulated acknowledgment technique to acknowledge successfully received packets [16].

This technique improves the robustness of acknowledgment but in accumulated acknowledgment mechanism when the packet loss occurs then the source is unable to determine how may packets have been transmitted so it can not be able to recover more than one packet per *RTT* during the transmission. In SACK technique there are many SACK blocks and each block has a group of data. The destination tells the source by using ACK with SACK option that receiver receives the out of order block of data. SACK blocks are used to maintain the image of the receiver buffer after received by sender side, for a example which packet is received and which is lost at the receiver end. A scoreboard is mentioned to record the information of these transmitted and received data with the help of last information in the SACK option. Scoreboard records the sequence number and flag bit of transmitted packet which shows that packet is SACKED or not. Transmitted packet having SACKED bit on, does not need to be retransmitted but the packets with SACKED bit off and also have less sequence number need to be re-transmitted. TCP SACK and TCP Reno are using the same congestion control techniques, but during the multiple packet losses the TCP SACK behaves differently as compared to TCP Reno. The TCP Sack enhances the technique of fast re transmit and fast recovery of TCP Reno and able to recover the multiple packets losses within the one *RTT* in the same congestion window.

However TCP SACK has two major issues. First is in SACK option there are only 3 or 4 SACK blocks are allowed and other problem is when the buffer over flows at the receiver end, it will start to drop the SACKED packets without informing to sender and sender will not be able to re transmit these packets.

2.6 TCP FACK

TCP FACK introduced the FACK (Forward Acknowledgments) Congestion control technique which proposes the recovery mechanisms to control the outstanding packet rate and error recovery [17]. FACK technique uses selective ACKs to indicate packet losses in flow control phase. It re-transmits the lost packet timely as well. Re-transmitted data reported as loss and loss cannot be recovered instantly so the TCP FACK need to store information related to re-transmitted data such as the time of the last re-transmission by using time out mechanism (RTO). The numbers of outstanding packets are calculated in rate control mechanism by using information extracting from SACKs, without considering the congestion window inflation mechanism the TCP FACK uses three variables as shown in Figure 8.

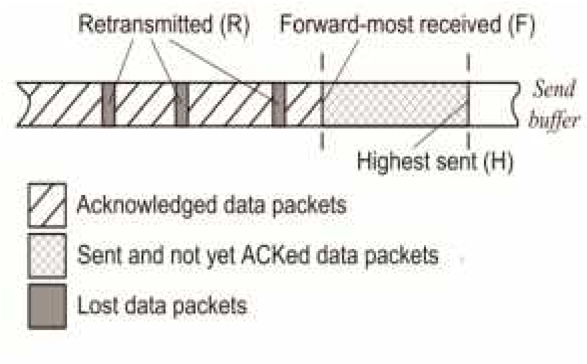
Retransmitted (R)
Forward-most received (F)
Send buffer
Highest sent (H)
Acknowledged data packets
Sent and not yet ACKed data packets
Lost data packets

Figure 8: *Variables of FACK Algorithm*

First is *H* which indicates the highest sequence number of all transmitted data, second is *F* indicates the forward sequence number of all acknowledged packets and third is *R* having total number of re-transmitted packets. A reliable estimation of outstanding data packets on the given path can be calculated by *H-F+R*. Sender side of TCP can use this estimation in decision of sending new data portion more precisely, when the calculated size of total outstanding packet is less than the size of congestion window. The recovery time of packet is improved in TCP FACK as compared to Reno or New Reno shown in simulation results [17]. However the improvements of TCP FACK have been recognized for long time and also been the part of Linux kernel in version 2.1.92

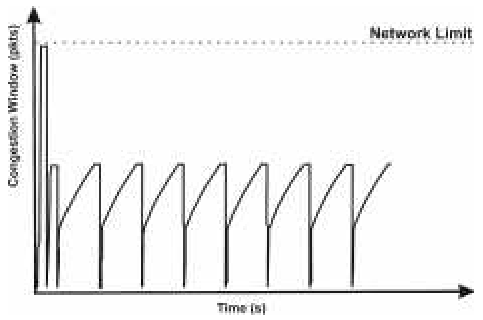

Figure 9: Congestion window behavior of TCP FACK

2.7 TCP Vegas

TCP Vegas is proposed by [18] and introduced the bandwidth estimation technique to avoid the congestion in the network rather react after congestion event. It calculates the total amount of packets that sender side can send by the help of measured *RTT*. TCP Vegas improves three mechanisms as compared to TCP Reno, first is improved slow start mechanism second is modified congestion avoidance mechanism and third is a novel re-transmission mechanism which are discussed below. TCP Vegas congestion window size adjustment technique also has three phases. In Figure 10 the transmission process is shown of TCP Vegas.

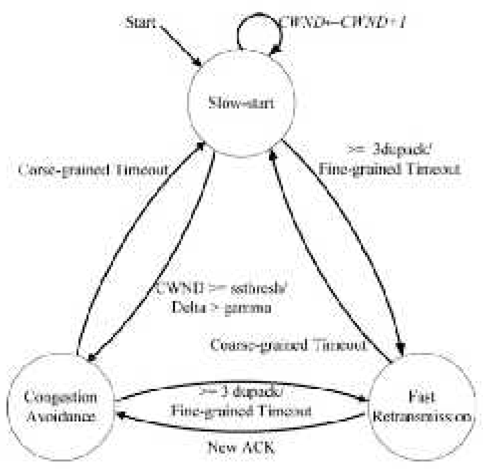

Figure 10: Vega's transition state

2.7.1 Slow Start

During the slow start mode, TCP Vegas tries to grape the bandwidth quickly and increase the size of congestion window exponentially per *RTT* to detect and avoid the congestion during this mode. To avoid the congestion in network Vegas algorithm update the actual and expected sending rate per *RTT*, it calculates the amount of extra data on the given network path and controls the size of congestion window accordingly. Vegas algorithm records the all *RTTs* and assigned the minimum value of *RTT* to base *RTT*. Amount of extra data (Δ) in network pipe can be calculated by equation, $\Delta = (Expected - Actual) \times BaseRTT)$ where the expected sending rate is calculated by the current size of congestion window divided by base *RTT* and actual sending rate is calculated by congestion window divided by recently measured *RTT*. This technique uses during the slow start mode of transmission to decide the switching mode from slow start mode to congestion avoidance mode. If the calculated amount of extra data is greater than the defined threshold γ, the Vegas reduce the size of its congestion window by one-eighth and moves from Slow Start (SS) mode to Congestion Avoidance (CA) mode.

2.7.2 Congestion Avoidance

In congestion avoidance mode TCP Vegas does not increase the size of congestion window continuously. First it calculates the actual and expected sending rate on the given path and then controls the sending rate on the given path by adjusting the size of congestion window. Vegas keeps the size of congestion window same when the calculated amount of extra data (Δ*)* lies between two predefined thresholds α and β .But when the value of (Δ*)* is greater than β which is the indication of congestion, so the size of congestion window is reduced *(cwnd-1)* and when the value of (Δ*)* is less than α, which is the indication of under utilization of network so the size of congestion window increase accordingly *(cwnd+1)* and when the value of α is less than and equal to (Δ*)* and (Δ*)* is less than an equation to β then the size of congestion window remain unchanged ($\alpha \leq \Delta \leq \beta$). In Vegas the size of congestion window is updated after every *RTT*.

2.7.3 Fast Re-transmission and Fast recovery

In TCP Vegas the three duplicate acknowledgments are caused for re- transmission as in TCP Reno too. Vegas enhance the re-transmission mechanism of TCP Reno for the re-transmission of the dropped packets quickly. Vegas calculates the *RTT* of every sent packet based on fine-grained values and with help of these fine-grained *RTT* measurements, Vegas calculated the time out period of each packet. After receiving the duplicate acknowledgments, it checks the status of time out of oldest unacknowledged packets, if the time out period of this packet has expired then the packet is re-transmitted. This modification leads to packet re-transmission after just one or two duplicate acknowledgments on receiving non duplicate acknowledgment after fast re-transmission. The algorithm again checks the expiration of the timer and then may re-transmit another dropped packet. So expired fine-grained timer is the condition for re-transmission of dropped packet after receiving certain acknowledgments. This mechanism improves the loss detection and recovery from multiple dropped packets without restarting the slow start mode. The size of congestion window is reduced to avoid congestion after packet re-transmission by receiving duplicate acknowledgments.

There are two situations for TCP Vegas to set the size of congestion window. In first case if the dropped packet re-transmitted once then the algorithm reduce the size of congestion window up to three fourth of the previous size otherwise Vegas consider that network is under severe congestion and reduce the size up to one half of the current congestion window size.

2.8 TCP Veno

TCP Veno is proposed by [9] and modifies the TCP Reno's congestion control technique for achieving the good throughput utilization. For the early detection of congestion TCP Veno uses the buffer estimation mechanism of TCP Vegas. TCP Veno(Vegas and reNO) introduced the two modifications. First modification is in the presence of excessive buffer utilization ($\Delta > \beta$) estimated by Vegas buffer estimation mechanism; TCP Veno limits the increment in size of congestion window during congestion avoidance mode. In other words when the Vegas estimation technique detects congestion the source starts conservative increment to probe network resources. Second modification is to reduce the congestion window after entering in the fast recovery phase only when the buffer estimation mechanism indicate the congestion. So after detecting the loss and when ($\Delta > \beta$) then the size of congestion window will be halved, on other hand the size of congestion window is reduced up to 80 percent when only packet loss is detected. The dynamics of congestion window of TCP Vegas are shown in Figure 11. In short the TCP Veno congestion control technique slightly improved as compare to TCP Reno. Veno flow stayed longer in congestion avoidance mode having larger congestion window, however there is no effect of these modifications on TCP fairness.

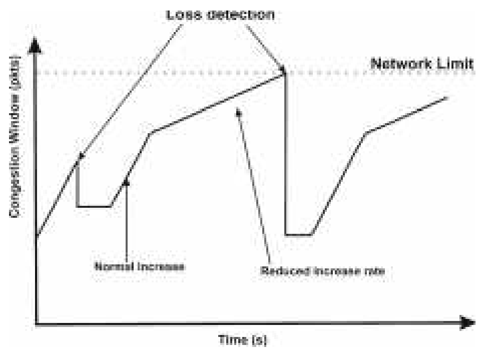

Figure 11: Congestion window behavior of TCP Veno

2.9 TCP Vegas A

Under certain network scenarios and circumstances, TCP Vegas has many internal problems and this all happened due to wrong assumption that *RTT* will always change with the change of buffering. In short when *RTT* will increase due to routing change then the Vegas algorithm will make a wrong decision which leads towards the reduction in flow rate.

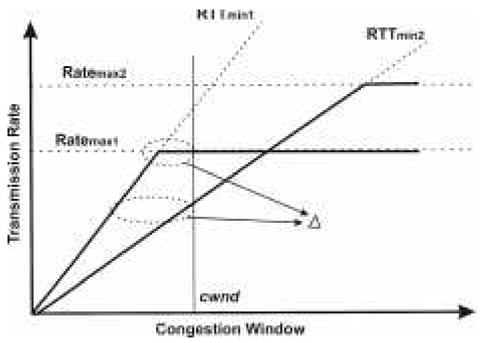

Figure 12: Estimation error when the path has changed

To describe this problem figure 12 is considered with low *RTT* (DSL connection) and high *RTT* (satellite link) paths. When the route change from low *RTT* to high *RTT* having congestion window size equal to *cwnd* then Vegas algorithm may make a wrong decision and can exceed a threshold. Another assumption of calculating the same RTT_{min}

of competing flow on the same path. To illustrate this let's take an example of a network scenario having two TCP Vegas flows. First flow has been transmitting the data for a long time and other flow just started its transmission. Naturally old flow has more chances for observing the minimum *RTT* as compared to new flow, so both flows estimate different congestion state after calculating different minimum *RTTs*. As shown in Figure 13 according to the old flows the network is congested but the estimation of new flow is that the network is in congestion Free State, so there are bright chances for new flow to grape network resources as compared to old one. Vegas A is introduced by [19] to solve these problems of TCP Vegas. Infect Vegas A enhanced the congestion control mechanism of Vegas with all adaptable mechanism. Basically the threshold parameters of TCP Vegas are adjusted considering the steady state of the actual transmission. When Vegas A algorithm detects a increment in the given bandwidth during the stable state of the network *($\alpha \leq \Delta \leq \beta$)* then it assumes that the path is changed and it shifts the control to upper zone *($\alpha = \alpha+1$)* and *($\beta = \beta+1$)*, control is shifted downwards after detecting any abnormality, for a example when estimation is showing the congestion free state *($\Delta < \alpha$)* but actually the sending rate is decreasing moreover the control is always shifted downwards after the estimation of actual congestion. Vegas A introduced additional conditions for congestion window increment mechanism.

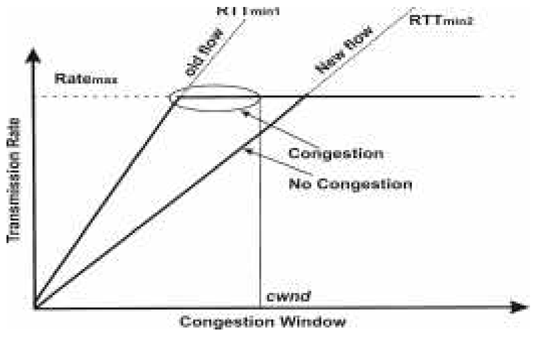

Figure 13 Estimation of error in the presence of high RTTmin

According to these conditions the congestion window is only increase in three situations. In first case the value of lower threshold *α* should be minimum with no congestion estimated on the given path. In second case when the actual sending rate is increased with no congestion estimated *($\Delta < \alpha$)* and third is when the actual sending rate is decrease with flow having steady state *($\alpha < \Delta < \beta$)*. The decrement in the network should be occur when the state of network is congested *($\Delta > \beta$)* or when actual sending rate decrease with congestion free state of the network. Experiments show [19] that Vegas A improve the design of TCP Vegas in various aspects. It does not change Vegas properties of stabilizing throughput in a steady state and slow start does not suffer during change in *RTTs*.

Table: 1 Features of TCP variants for Congestion Control

TCP Variants	**Year**	**Base**	**Addition/Enhancement in Features**	**Status**	**Implementation**			
					Win	**Linux**	**Mac**	**BSD**
TCP Tahoe	1988	RFC793	Slow Start, Congestion Avoidance, Fast Retransmit	Obsolete		1.0		4.3
TCP-DUAL	1992	Tahoe	Queuing Delay as congestion indication for congestion avoidance	Experimental				
TCP Reno	1990	Tahoe	Fast Recovery	Standard	95/NT	1.3.90		4.3 Fr2.2
TCP New Reno	1999	Reno	Fast recovery for multiple packet losses	Standard		2.1.36	10.4.6	Fr 4
TCP SACK	1996	RFC793	Extended information in feedback messages	Standard	98	2.1.90	10.4.6	F2.1

								N1. 1 S2. 6
TCP FACK	1996	Reno, SACK	Packet loss recovery technique based on TCP SACK	Experime ntal		2.1.92		N1. 1
TCP Vegas	1995	Reno	Bottleneck buffer utilization as a primary feedback for the Congestion Avoidance and secondary for the Slow Start	Experime ntal		2.2.10		
TCP Veno	2002	New Reno, Vegas	Increase/decrease parameters based on Reno in congestion avoidance phase.	Experime ntal		2.6.18		
TCP Vegas A	2005	Vegas	Adaptive bottleneck buffer state aware Congestion Avoidance	Experime ntal				

3. HIGH-SPEED NETWORK AND CONGESTION CONTROL

Developed TCP variants (Reno, New Reno, and Sack) became insufficient with the emergence of high speed networks for using network resources effectively. All the congestion control techniques which are in section II aiming to improve the efficiency of network for transferring the data, without discussing the basic principle on which it relay, which was defined in 1988 in TCP Tahoe that in congestion avoidance phase probing of the network resource should be conservative. In TCP this principle is realized with congestion window which is increased by one packet per RTT if there is no congestion detected. This behavior of congestion window works well when the capacity of the network and delay are small but in high speed networks it does not work well. To illustrate this problem (some time called BDP problem) let us take a example of an flow which is trying to grape all network resources. The minimum time required for this is *D ×cwndRTTs*, when there is no packet loss. So a network having 10GiBit/s capacity with 100ms delay and with maximum packet size of 1500 bytes, it would take two hours to obtain all resources [20] [24]. In this section many TCP variants with congestion control techniques are discussed aiming to perform well in high speed networks more over in Table 2 the summary of these TCPs with features are discussed.

3.1 High-Speed TCP

High-Speed TCP (HS-TCP) proposed by [20] [21] after identify the efficiency problem in fast fat networks. HS-TCP is an experimental algorithm for congestion control technique with many objectives, which are link efficiency and fairness. To achieve the goals HS-TCP change the increase factor α of congestion window during the congestion avoidance mode and decrease factor beta after detecting loss with the function *α(w)* and *β(w)* for the size of congestion window. These functions are stated to achieve the above mentioned objectives defined in term of required loss rate and size of congestion window as shown in Figure 14.

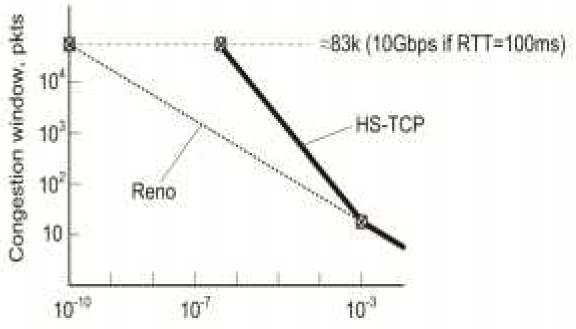

Figure 14: HS-TCP objectives

HS-TCP should use 10Gbps link with loss rate not more than 10^{-7} behave like standard New Reno [22] with loss probability not more than 10^{-3}. So when the value of congestion window is less than or equal to 38 to 70 packets then the value of function *α(w)* and *β(w)* are vary from 1 and 0.5 respectively and then the value is 0.1 when the size of congestion window is more than 84K packets. HS-TCP grapes the network resources more quickly as compared to Reno and behaves conservatively after loss detection. The congestion window behavior of HS-TCP is shown in Figure 15.

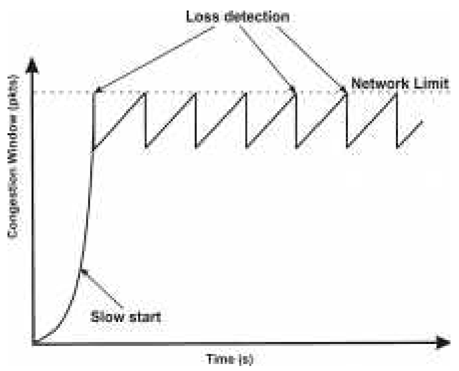

Figure 15: Congestion window behavior of HS-TCP

There is another problem for TCP in high-speed networks which are during the slow start mode the size of congestion window increases exponentially which may leads towards extensive packet losses. For the solution of this problem Floyd [23] introduced a mechanism when the congestion window increase up to 100 packets during the slow start mode (Limited Slow Start), but this limitation has not any significant effect on performance of the network. In the presence of sever congestion HS-TCP behaves like Standard TCP Reno that means it inherits all Reno's characteristics. We are not considering the fairness comparison of HS-TCP and TCP Reno because Reno cannot use the network resources efficiently but Intra fairness of HS-TCP is important. HS-TCP flows achieves good fairness having same *RTT* but unfair in the presence of different *RTTs*. This problem is come from TCP Reno and may be solved by tuning the AIMD algorithm.

3.2 Scalable TCP (STCP)

Scalable TCP (STCP) was proposed by [24] which is an alternative solution of HS-TCP to grape the network resources more efficiently. STCP rejects the complex calculation of AIMD parameters (in HS-TCP) and introduce new algorithm for the increment and decrement of the congestion window called Multiplicative increase and Multiplicative decrease *(MIMD)*. In congestion avoidance mode STCP increases its congestion window *w* by a fraction α per *RTT* $(w=w+\alpha\times w)$ where the value of α is 0.01. After packet loss detection it moves in fast-recovery mode and decrease the size of congestion window by a different fraction called *β* $(w = w-\beta\times w)$ where the value of β is 0.125.

The behavior of congestion window of STCP looks similar to HS-TCP, but STCP improves the increase and decrease phases as shown in Figure 16 although the proposed modification solved the target problem by changing the increase and decrease factors but it also create many critical problems for a example in Figure 16 clearly shows the stat of network and can easily move towards congestion in presence of even one STCP flow, which is generally not accepted for many networks. Second, inter fairness behavior of STCP is similar to HS-TCP means there is no improvement in inter-fairness , third the MIMD technique does not provide intra protocol fairness because when the two competing STCP flows are detected loss with same *RTT* then the flow having larger initial share will always took the advantage on having short share. In short we can say that the STCP is extremely unfair protocol.

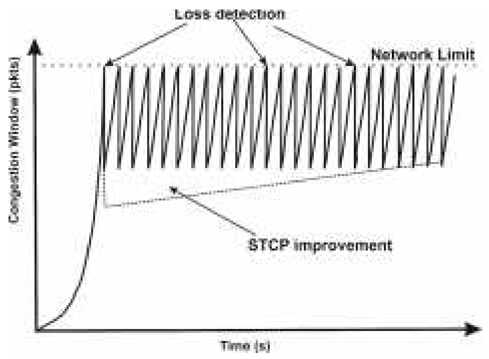

Figure 16: Congestion window behavior of S-TCP

3.3 Hamilton TCP (H-TCP)

Leith and Shorten [25] introduced another congestion control algorithm for TCP aiming to improve the *RTT* fairness (inter and intra). It uses the elapsed time (Δ) since the last congestion event occurs for the estimation of congestion window size in congestion avoidance mode. Due to dependency on function of elapsed time (Δ) the H-TCP flows experiences same network resources and same congestion window increment. In short we can say that H-TCP flows are fair with each other in the same network path as shown. Moreover it shows that after detection of packet loss congestion window decrease by half equally. In H-TCP the increment in congestion window per *RTT* is defined by α (Δ), which is a polynomial function over elapsed time (Δ) when the last congestion event occurred. The increment in congestion window is calculated by equation $\alpha(\Delta) = 1 + 10(\Delta - \Delta_{low}) + 0.5.(\Delta - \Delta_{low})^2$. In this equation whenever (Δ), (Δ_{low}) and the value of *α (Δ)* would be 1 and Δ_{low} is a predefined threshold of H-TCP. We can note here that according to the *α (Δ)* calculation, H-TCP still leads towards the *RTT* unfairness shown in Figure 17, for a example the *α (Δ)* is calculated per *RTT* at time 0, T_1 and T_2 then we can clearly observe that the flow with longer *RTT* always losses as compared to flow having shorter *RTT*. For the solution of this issue H-TCP introduced another technique of tuning the *α (Δ)* with reference RTT(RTT_{ref}) by $\alpha'(\Delta) = \alpha(\Delta) \times RTT/RTT_{ref}$. H-TCP modified the decrease rule during the congestion avoidance phase. After the loss event H-TCP estimates the throughput of the flow *B (k)* and make a comparison with estimation of preceding loss event *B (k-1)*, If the calculated value of *B (k) – B(k-1)/B(k-1)* is smaller than 0.2 then the reduction in size of congestion window is determined by the ratio RTT_{min}/RTT_{max} and when more than 0.2 then 0.5 parameter is used.

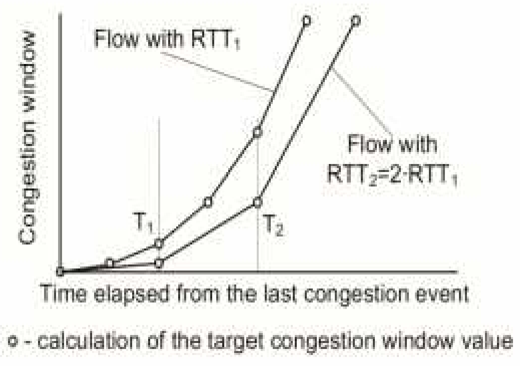

Figure 17: RTT unfairness between two flows of H-TCP

3.4 TCP Hybla

TCP Hybla was proposed by [26] resolve the problem of *RTT* fairness in TCP New Reno, where the flows having shorter *RTT* have always advantage as compared to longer *RTT*. The modifications are made in TCP New Reno 's Slow Start and congestion avoidance phases and make them Sami independed of *RTT*. For the normal increment in the size of congestion window a scaling factor ρ was introduced which is calculated by $\rho = RTT/RTT_{ref}$ where RTT_{ref} is reference of the *RTT* (for example 30ms), so the increment style of congestion window for TCP Hybla is shown in equation 1 for slow start phase and in equation 2 for congestion avoidance phase. The technique is described in Figure 18 where three competing flows with different *RTTs* are shown. Due to high value of *RTT* the ratio of *ρ* become high and in result the size of congestion window is increased more rapidly after receiving each Acknowledgment. Moreover when flows have same time period then the value of congestion window is different for the growth of congestion window.

$$w = w + 2^{\rho} - 1 \qquad (1)$$

$$w = w + \rho^2/w \qquad (2)$$

Anyhow if we calculate the ratio between *RTT* and congestion widow (upper bound), we can observe that all flows are transmitting the data at similar rate. Hyble proposed another two mechanisms for the congestion control. First is pacing technique [27] for the transmission of data and second is packet pair algorithm [28] for the estimation of initial slow start threshold. Pacing technique sets the minimum delay between two consecutive transmitted packets to overcome the burst nature of TCP and packet pair mechanism gives the estimation of capacity of available network path because the advance knowledge of the capacity may help to improve the convergence speed and scalability in high speed bandwidth delay product networks (BDP). However many experiments [26] has proved that the TCP Hyble improved the *RTT* friendliness, but Hybla is developed to turned back to standard TCP mode (Reno congestion control principle) when the *RTT* of flow is less than predefined reference value and this behavior limits the Hybla to deploy in long delay networks (for a example Satellite link).

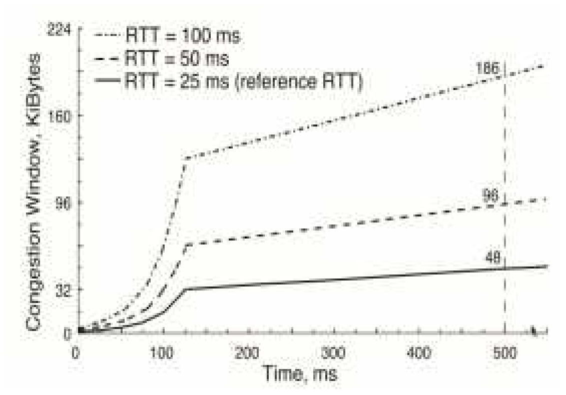

Figure 18: Evolution of congestion window of TCP Hybla

3.5 BIC TCP

Xu et al [29] described that HS-TCP and STCP have severe unfairness problem. To illustrate, when two competing flows detect packet loss simultaneously then according to the HS-TCP algorithm (HS-TCP flow) with *RTT* y will get the larger share of the network and according to STCP algorithm (STCP flow) with small *RTT* can get all network resources but the flow having larger *RTT* cannot be able to get any network resource. Both TCP variants have problem regarding discovery of network resources. To solve this unfairness problem TCP BIC (Binary Increase Congestion Control) was proposed. Basically TCP BIC is an extension of TCP New Reno with a new technique called *Rapid Convergence*. This technique uses the binary search for obtaining the optimal value of congestion window rapidly according to the network resources. The typical behavior of binary search in TCP BIC is shown in Figure 19.

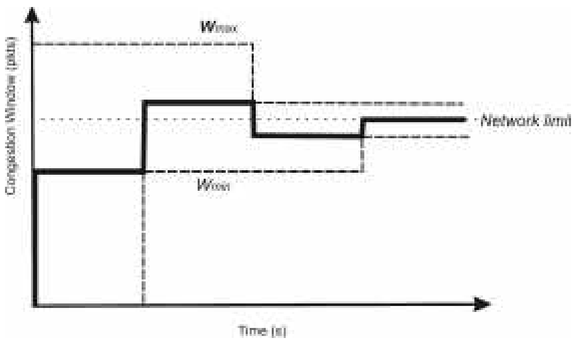

Figure 19: Binary search of TCP BIC

In TCP BIC the size of congestion window is updated between minimum w_{min} and maximum w_{max} after receiving all acknowledgments (at the start the value of w_{min} is one and w_{max} has some high value) After receiving the successful acknowledgment the value of w_{min} increase by the value of previous value of congestion window and after detecting packet loss the BIC algorithm set the value of w_{max} to current value of congestion window and moves towards the fast recovery mode (similar to New Reno). Moreover for the improvement of convergence speed in less loss rate network, BIC algorithm reduce the value of multiplicative decrease parameter β from 0.5 to 0.125 *(w=w-0.125.w)* due to binary search algorithm the convergence time is very high in high BDP networks, which may leads towards the large number of packet losses. The congestion window behavior of TCP BIC is shown in Figure 20.

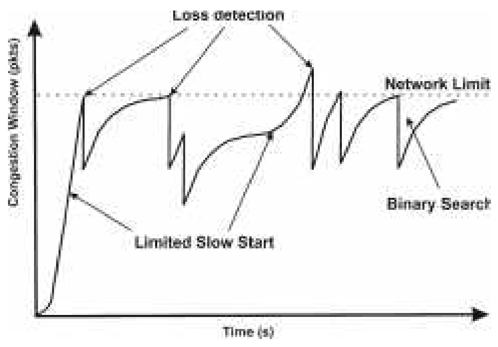

Figure 20: Congestion window growth of TCP BIC

To solve this problem BIC not only uses the Limited Slow Start technique of HS-TCP but it also controls the rapid increment in convergence time when the search range is too big. In short BIC defined a parameter S_{max} which limits the rapid convergence on the other hand when the range is too small then BIC defined a parameter S_{min} for the increment of the congestion window and in last when the value of current congestion window became near or exceed from target value of congestion window then BIC uses Limited Slow Start mode and set a new upper bound and again start the binary search. The TCP BIC technique for optimal value of congestion control is a novel technique in loss based congestion control approaches. According to many experiments [30] [31] in many network scenarios TCP BIC showed low *RTT* fairness and inter fairness (fairness with other TCPs).

3.6 TCP CUBIC

CUBIC TCP [30] is an enhanced version of TCP BIC proposed to remove the *RTT* unfairness. CUBIC introduced the *RTT* independent congestion window growth function and for this growth CUBIC uses the H-TCP congestion window approach as CUBIC function of elapsed time (Δ) when the last congestion event occurred as shown in equation 3 where the value of *C* is predefined, β is parameter of multiplicative decrease factor in fast recovery process and w_{max} is a congestion window size just before the loss event. CUBIC not only shows good *RTT* fairness as the congestion window is independent to *RTT* but also improves the intra fairness and scalability of TCP BIC. In CUBIC the growth of congestion window is very fast when the value of *w* is far from predefined target value w_{max} and when the value of *w* is near to w_{max} the growth is very conservative.

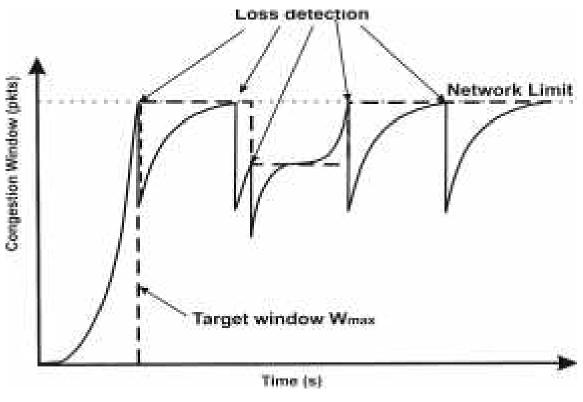

Figure 21: Congestion window behavior of TCP CUBIC

In Figure 21 the behavior of CUBIC congestion window is shown. Where in step one the target window is unknown can be discover by using CUBIC function. For the discovery of target window w_{max} CUBIC starts conventional slow start mechanism. In second step congestion window smoothly touches the target window w_{max} and target window will be updated if loss is detected before touches the w_{max}. If this congestion was temporary then the behavior of congestion window is shows in step 3. Moreover CUBIC TCP provides a technique to ensure the improved performance of congestion window that standard TCP (Reno) because the value of multiplicative decrease factor β is different in CUBIC as compared to Reno β CUBIC not equal to β Reno). The improved performance and fairness of CUBIC is proved by number of experimental studies [30] [32] and by real environment scenarios. Currently CUBIC TCP is a second most used congestion control algorithm of TCP and default TCP for Linux operating system since 2006. Therefore CUBIC has many issues related to resource utilization and packet losses in high-speed bandwidth delay network.

$$w = C\left(\Delta - \sqrt[3]{\beta.w_{max}/C}\right)^3 + w_{max} \qquad (3)$$

3.7 Fast TCP

Fast TCP is introduced by Jin et al [33] [34] and used the idea of queuing delay of Vegas as congestion indication. It introduced the periodic congestion window updating based on delay estimation of the network. However there is a basic difference between Vegas and Fast. In Fast there is a fix value (e.g., each 10 ms) for the updating of the congestion window. Fast TCP calculates the new target congestion window by equation 4 having simple delay estimation *w* shows the current value of congestion window, *Retain* is minimum *RTT* of current *RTT* and α is tune able parameter of the Protocol. According to this equation 4 if the network state is congested $RTT > RTT_{min}$ then FAST will decrease the size of congestion window otherwise increase the congestion window by pre-defined value of α. Value of α create a drastic effect of two parameters of Protocol: Stability and Scalability. For a example if the value of α is high then it can easily scale in any BDP networks but it will face convergence problem (when *w=w×RTT*$_{min}$*/RTT+α)*. On the other hand when the value of α is small then Fast can easily stabilized but it will face the problem of scalability. Therefore the selection of α's value is still an open issue for researchers. According to the authors of FAST the value of α should be constant for the better performance, moreover FAST uses the exponential smoothing mechanism for the calculation of congestion window value. Simulation and Real world tests are showed many serious issues with stability, scalability, *RTT* fairness and intra fairness [34] [35]. FAST technique is totally depend upon the true minimum *RTT* value which is very hard to calculate in some network scenarios moreover the *RTT* is not a good substitute for the queuing delay in the presence of reverse traffic or when there is a change in route. Finally the defined congestion window rule for updating in TCP FAST is not friendly with standard TCP (Reno, New Reno).

$$w = w.\frac{RTT_{min}}{RTT} + \alpha \qquad (4)$$

3.8 TCP Libra

TCP Libra is a congestion control variant proposed by Marfia et al [28] to solve the scalability problems and improve the *RTT* fairness. Libra mechanism is based on New Reno and modifies the congestion avoidance technique for congestion window. In Libra the target value of congestion window is calculated by well known packet pair technique [36]. In Libra's congestion avoidance phase the value of congestion window is increased by equation 5. Where the *RTT* is the current estimated *RTT*, γ and k_1 are predefined parameters (e.g,. 3 and 4 respectively) and the value of *C* is responsible for the scalability of the Libra and also represent the capacity of the link and *P* is the penalized factor which controls the increase rate when the network is facing congestion. Penalized factor *P* can be calculated by equation 6. Where K_2 is a constant value and *Q* and Q_{max} are estimated current and maximum queue delays. In equation 5 $k_1.C$ uses to scale the increment in the available link and factor *P* control the Libra for the reduction of network resources, if the calculated level of buffering ($Q.Q_{max}$) increases then the last part of the equation ($RTT^2/(RTT+\gamma)$) is responsible for the *RTT* fairness of Libra. Moreover Libra also modifies the multiplicative decrease rule in fast recovery mode *w=w-β×w*, where the β scales with equation (θ/RTT+ γ) in this equation the values of θ and γ are constant. This scaling factor is derived analytically [28]. According to the Libra the recommended value for θ and γ is 1 second. In Libra when the current value of estimated *RTT* is large (congestion in the network) then the scaling factor will reduce the value of β further. Number of experiments and simulations show that Libra improves the link utilization and fairness properties of TCP in BDP networks but it does not work well in other congestion control mechanisms.

$$\alpha = k_1.C.P.RTT^2/(RTT + \gamma) \qquad (5)$$

$$P = e^{-k_2 \times Q/Q_{max}} \qquad (6)$$

3.9 TCP New Vegas

Sing and Soh [37] found many advantages on delay based algorithm of congestion control proposed in TCP Vegas, therefore they also found some problems in Vegas. First is it cannot be able to utilize fully high BDP network links, second is on the restart modes (slow start and Fast recovery) its congestion control algorithm generates too much traffic which leads towards the decrement of throughput [13] [38].

TCP New Vegas was proposed to grape the link effectively in BDP networks and to reduce the convergence time. It defined the new mode called *Rapid Window Convergence*. The inside idea is that the algorithm does not stop the slow start phase when the estimated buffer value reaches to its limit or exceeds from per-defined threshold (Δ>α), it continues the slow start phase and congestion window grows exponentially for probing network resources. In detail when the value of estimated buffer reaches to its limit then the New Vegas 's algorithm saves the current value of congestion window in a special veritable called w_r and moves towards *Rapid Window Convergence.* In this mode the congestion window increases *x* packets after each *RTT* as shown in equation 7.

$$x - (w_r)^{-2^{3+n}} \tag{7}$$

Where *n* represents the number of time when early congestion happened and algorithm moves towards the Rapid Window Convergence mode. According to this proposal [37] when the value of *n* becomes more than 3 then it close the *Rapid Window Convergence* and switches to Vegas congestion avoidance mode and after detecting packet loss it behaves like Vegas. For the solution of second problem New Vegas algorithm uses packet pacing technique to control the burst traffic after re-initialization of slow start and fast recovery modes. This algorithm sets the minimum delay between two consecutive packets during transmission [39] [40]. Although according to literature pacing technique has negative impact on TCP Reno's performance but Authors of New Vegas experimentally proved that in the presence of delay base congestion control, packet pacing has positive impact.

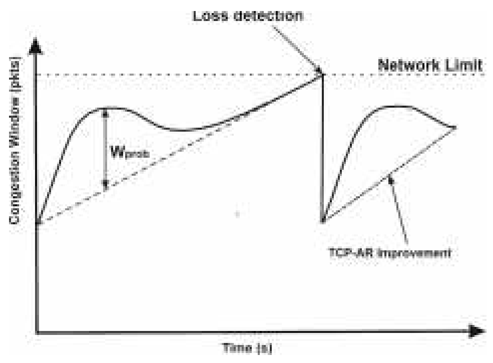

Figure 22: Congestion window behavior of TCP-AR

3.10 TCP- Adaptive Reno

TCP-AR (Adaptive Reno) was proposed by Shimonishi and Murase [41] to improve the performance of TCP and friendliness to standard TCP in high speed networks. It extends the congestion avoidance phase of TCPW-BBW with scalable congestion window. Moreover it defined two components for the increment of congestion window. First is W_{base} which is slow constant increase component (increased by 1 per *RTT*) and second is W_{probe} which is scalable increase component. Scalable component has two important properties. First when the network sate is congestion free then this function gives value near to Westwood-like and second when the network is congested then the value of scalable component is zero. Figure 22 shows the behavior of congestion window of TCP-AR. According to experiments it is proved that TCP-AR improved the network utilization and intra-fairness.

3.11 TCP Fusion

TCP Fusion is proposed by Kaneko et al [42] to improve the performance of TCP. It combines the idea of Vegas (used network buffering estimation), TCP DUAL's queuing delay technique and Westwood's achievable rate mechanism. Fusion algorithm introduced three different linear functions and its switching is depending upon the queuing delay threshold value. If the value of queuing delay is less than the predefined value of threshold (zone A figure 23) then the size of congestion window will increase at fast rate per *RTT* by predefined fraction of Westwood 's rate estimate (scalable increase) and when the value of queuing delay is more than 3 times to predefined threshold (zone C in figure 23) then the congestion window will decrease according to number of packets buffered in the network (for a example Vegas estimate) and when the value of queuing delay lies between 1 and 3 times then the congestion window remains unchanged.

Moreover Fusion algorithm changes the constant value of β in fast recovery phase to the value $\beta=max(0.5,RTT_{min}/RTT)$. Many experimental tests have shown that Fusion improves the link utilization and fairness properties as compare to other algorithms (FAST, C-TCP, BIC, and HS-TCP).

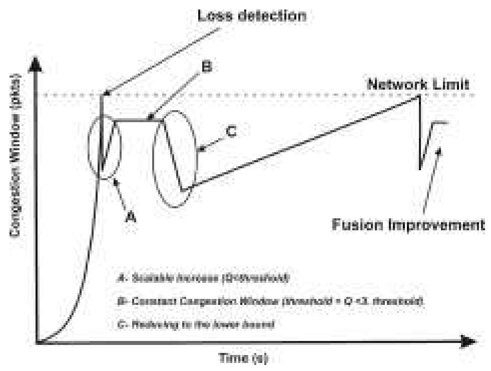
Loss detection
B
Network Limit
C
Congestion Window (pkts)
A
Fusion Improvement
A- Scalable increase (Q<threshold)
B- Constant Congestion Window (threshold < Q <3. threshold)
C- Reducing to the lower bound
Time (s)

Figure 23: Congestion window behavior of TCP Fusion

3.12 TCP Africa

TCP Africa (Adaptive and Fair Rapid Increase Congestion Avoidance) is proposed by King et al [43] to solve many problems related to previous congestion control algorithms like link utilization, *RTT* fairness (inter and Intra) in BDP networks. Africa combines the two mechanisms, first is aggressiveness of HS-TCP when the sate of network is congestion free and conservative behavior of standard New Reno when the network is congested.

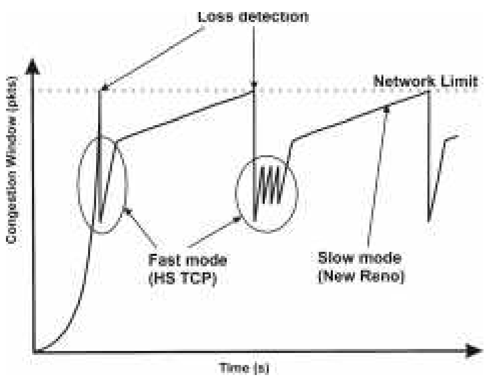

Figure 24: Congestion window behavior of TCP Africa

The idea is borrowed from TCP Vegas to detect state of the network either it is congestion free or congested by comparing the estimate of network buffering *(Δ)* with some predefined constant *α* . More precisely when there is little buffering *(Δ<α)* then TCP Africa moves towards fast mode by applying the HS-TCP's congestion avoidance and Fat Recovery rules. These rules decide the increment and decrement of the congestion window as shown in Figure 24. Otherwise it will move towards the slow start mode of the Reno.(increase by one and decease by half). According to its authors TCP Africa showed good network utilization in high speed networks, low rate of packet losses as compared to HS-TCP and STCP and good fairness (inter and intra) by extensive experimental analysis but unfortunately TCP Africa has not been implemented and tested in real networks.

3.13 Compound TCP (C-TCP*)*

Compound TCP (C-TCP) was presented by Tan et al [44] to improve the efficiency, *RTT* fairness and TCP fairness. Its congestion control mechanism is very similar to TCP Africa. It combines the delay base component for the indication of congestion with conventional Reno congestion control mechanism which is scalable in BDP networks.

C-TCP introduced another scalable component w_{fast} for the calculation of congestion window ($w=w_{Reno} + w_{Fast}$), but congestion window is only updated through this equation when the Vegas estimate (Δ) show little buffering in the network (Δ<α) and α is predefined constant) and when the value of estimate become larger than α then the scalable component reduced the size of congestion window by a value proportional to the estimate itself ($w_{fast} = w_{fast} – ζ.Δ$) where ζ is predefined constant). C-TCP uses the transition between Reno Slow mode and scalable H-TCP for the reduction of congestion window just like slow and fast mode in TCP Africa.

As a result the dynamics of congestion window of both C-TCP and TCP Africa are very similar as shown in Figure 25. Experimental analysis for both simulation and real world has proved that C-TCP showed good link utilization in high BDP network with good *RTT* fairness (inter and intra). That's why C-TCP is default TCP for Microsoft Windows operating system and most developed congestion control algorithm all over the world.

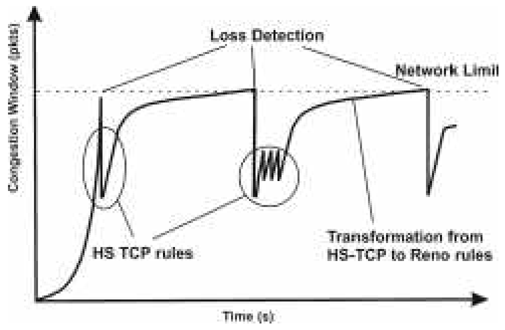

Figure 25: Congestion window dynamics of Compound TCP

3.14 TCP Illinois

Liu et al [45] noted the congestion control algorithms based on delay (Vegas and Fast) which is used as primary indication of congestion can achieve better efficiency as compare to that algorithms which are based on packet loss (Reno HS-TCP, STCP).

Therefore the performance of delay based algorithms may suffer due to noisy delay measurements for an example due to heavy opposite transmission. To solve this problem TCP Illinois was proposed. TCP Illinois is similar to Africa and C-TCP and based on New Reno. Its behavior is very aggressive when the state of the network is congestion free and during congestion events it behaves very gentle. Its algorithms defined both the congestion window w and increase parameter α in congestion avoidance phase ($w = w + \alpha$ per *RTT*) and decrease parameter β in Fast Recovery ($w = w - \beta.w$) after detecting packet loss by 3 duplicate ACKs) as a special functions of queuing delay. The increase parameter α is inversely proportional to queuing delay and decrease factor β is directly proportional as shown in Figure 27. The minimum and maximum values of α, β and queuing delay are defined in algorithm to achieve better performance.

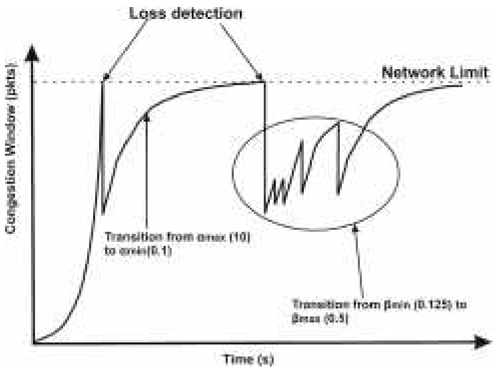

Figure 26: Congestion window behavior of TCP Illinois

In Linux operating system Illinois algorithms defined values of a_{max}=10, a_{min}= 0.3, B_{min}=0.125, b_{max} =0.5, Q_1=0.01, Q_2=0.1. Q_{max} and Q_3 =0.8.Q_{max} where the Q_{max} is maximum queuing delay calculated over the lifetime of the connection. According to this algorithm increase and decrease factors are updated one per *RTT*. However the value of α can be maximum during number of consecutive *RTT* (for a example 6) and the value of queuing delay is less than predefined threshold Q_1 and when the size of congestion window is less than predefined threshold *wt* then algorithm moves towards the compatibility phase (α =1 and β =0.5). This move is similar to STCP and HS-TCP, improves the fairness properties. Figure 26 shows the dynamics of TCP Illinois's congestion window. However the experimental results are showed that TCP Illinois behaved very well in high BDP network along good *RTT* fairness.

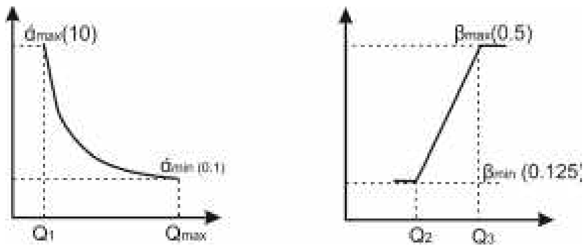

Figure 27: Behavior of additive increase alpha (α) and multiplicative decrease beta (β) with respect to queuing delay Q

3.15 YeAH TCP

Baiocchi et al [46] proposed a congestion control algorithm YeAH TCP (Yet Another High Speed) by combining the packet loss and delay measurement approaches for the indication of congestion. This algorithm uses the technique of Slow New Reno and fast STCP in congestion avoidance and fast recovery phases just like TCP Africa.

In a former way the congestion window increases by one packet per *RTT* and after loss it decreases by half of its current value. YeAH TCP uses two mechanisms for the switching of modes, first is the Vegas estimation of packets buffered in the network and second is TCP DUAL's estimation of congestion level in the network. For the calculation of queuing delay YeAh TCP uses the minimum measured *RTT* instead of average *RTT* and for calculation of congestion level it uses the fraction of minimum *RTT* instead of Maximum *RTT(Q/Q_{max})*. In short when the YeAh TCP calculates low level buffering in network (Δ <

α.Q/Q_{max}) where α is predefined threshold) and queuing delay estimation detects low level of congestion *($Q/RTT_{min} < f_i$)*, where f_i is predefined threshold) then it behaves like as STCP otherwise slow Reno mode activates. According to Experimental analysis YeAh TCP shows high efficiency in BDP networks more over it is confirmed that the combination of loss base and delay component can improve the *RTT* fairness (inter and intra).

Table: 2 Features of TCP variants for Congestion Control in High Speed delay Networks

TCP Variants	**Year**	**Base**	**Addition/Enhancement in Features**	**Status**	**Implementation**			
					Win	**Linux**	**Mac**	**BSD**
HS-TCP	2003	NewReno Additive	Additive increase and multiplicative decrease factors of congestion window, Limited Slow Start	Experimental		2.6.13		
STCP	2003	NewReno	Multiplicative increase and multiplicative decrease parameters in congestion avoidance phase	Experimental		2.6.13		
H-TCP	2004	NewReno	Increase factor of congestion window with respect to elapsed time, Adaptation of multiplicative decrease factor,	Experimental		2.6.13		
TCP Hybla	2004	NewReno	Scaling the increase factor in Slow Start and Congestion Avoidance phase, Packet Pacing, Estimation of Slow Start threshold	Experimental		2.6.13 2.6.12		
TCP BIC	2004	HS-TCP	Binary search for the congestion window growth, Limited Slow Start	Experimental		2.6.13		
TCP CUBIC	2008	BIC	Cubic growth function for congestion window.	Experimental		2.6.16		
FAST TCP	2003	Vegas	Growth of congestion window at constant rate	Experimental				
TCP Libra	2005	New Reno	Estimation of link capacity by using packet pairs, Control the congestion window increment by using queuing delay.	Experimental				
TCP Vew Vegas	2005	Vegas	Rapid window convergence, Packet pacing, Packet pairing	Experimental				
TCP-AR	2005	Vegas	Congestion window increment by using queuing delay	Experimental				
TCP Fusion	2007	Westwood, Vegas	Congestion window increment by using queuing delay	Experimental				
TCP Africa	2005	HS-TCP, Vegas	Switching between fast and slow phase by using Vegas estimation of network state	Experimental				
Compound TCP	2005	HS-TCP, Vegas	Calculation of slow and scalable components of congestion window	Experimental	Vista, XP	2.6.14, 2.6.25		

TCP Illinois	2006	New Reno, DUAL	Increase and decrease factors as functions of queuing delay.	Experimental		2.6.22		
YeAH TCP	2007	STCP, Vegas	Switching between slow and fast phases by using DUAL and Vegas type estimation.	Experimental		2.6.22		

4. CONCLUSION

In this study we have discussed many types of congestion control approaches and noted that the focus of research has changed with the enhancement in Internet, from the problem to eliminate the congestion collapses to problem to use the network resources efficiently and effectively in different network environments.

In the first section we discussed the introduction of congestion and its problems. In section II we discussed about the congestion collapses and that kind of proposals with techniques which builds the concept of the end to end congestion control principal and solve the problem of congestion collapse. In first proposal, TCP Tahoe proposed the basic method to probe the network resources with using packet loss for the detection of congestion and the limit of the network. Although this technique solves the problem of congestion but it could not able to use network efficiently. As we discussed some solutions for the efficiency problem like, (1) enhance the congestion control principal by making the assumptions about the network limit (TCP Reno, TCP New Reno); or (2) refine the protocol by enhancing the reporting technique of the receiver side and allows sender to estimate the network limit (TCP SACK, TCP FACK) or by developing delay measurement technique for the indication of congestion or for the estimation of network state (TCP DUAL, TCP Vegas, TCP Veno).

In section III we discussed new challenges for the TCP congestion with the advancement of high speed delay network and most recent past proposals. The basic aim of these proposals is to solve the problem of poor utilization of TCP flows in high speed delay networks. For the solution of poor utilization many proposals (HS-TCP, STCP, H TCP) are discussed with aggressive policies to probe the network resources. But unfortunately the aggressiveness of these techniques leads towards inter and intra RTT unfairness. Later proposals (TCP BIC, TCP CUBIC) used the aggressive behavior only when there is no any congestion or congestion indication in the network and after indication of congestion by packet loss, behaved conservatively to probe the available network resources. Other proposals (TCP FAST, TCP Africa, TCP-AR, C-TCP, TCP Libra, TCP Illinois, TCP Fusion, and TCP YeAh) used delay measurements for the indication of congestion in the network. Although these are many disadvantages of these two techniques and also there is not any consensus in research community that which approach is good for the current Internet. Currently C-TCP is used in Microsoft Windows operating systems and TCP CUBIC is used in Linux operating systems.

REFERENCES

[1] J. Postel, "RFC793—transmission control protocol," RFC, 1981.
[2] A. Al Hanbali, E. Altman, and P. Nain, "A survey of TCP over ad hoc networks," IEEE Commun. Surveys Tutorials, vol. 7, no. 3, pp. 22–36, 3rd quarter 2005.
[3] C. Lochert, B. Scheuermann, and M. Mauve, "A survey on congestion control for mobile ad hoc networks," Wireless Communications and Mobile Computing, vol. 7, no. 5, p. 655, 2007.
[4] J. Widmer, R. Denda, and M. Mauve, "A survey on TCP-friendly congestion control," IEEE Network, vol. 15, no. 3, pp. 28–37, May/June 2001.
[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," IEEE/ACM Trans. Netw., vol. 5, no. 6, pp. 756–769, December 1997.

[6] K.-C. Leung, V. Li, and D. Yang, "An overview of packet reordering in transmission control protocol (TCP): problems, solutions, and chal- lenges," IEEE Trans. Parallel Distrib. Syst., vol. 18, no. 4, pp. 522–535, April 2007.
[7] S. Low, F. Paganini, and J. Doyle, "Internet congestion control," IEEE Control Syst. Mag., vol. 22, no. 1, pp. 28–43, February 2002.
[8] G. Hasegawa and M. Murata, "Survey on fairness issues in TCP congestion control mechanisms," IEICE Trans. Commun. (Special Issue on New Developments on QoS Technologies for Information Networks), vol. E84-B, no. 6, pp. 1461–1472, June 2001.
[9] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for trans- mission over wireless access networks," IEEE J. Sel. Areas Commun., vol. 21, no. 2, February 2003.
[10] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," in Proc. SIGCOMM, 1987.
[11] V. Jacobson, "Congestion avoidance and control," ACM SIGCOMM, pp. 314–329, 1988.
[12] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in 4.3– Tahoe BSD TCP congestion control," ACM Computer Communication Review, vol. 22, no. 2, pp. 9–16, 1992.
[13] M. Allman, V. Paxson, and W. Stevens, "RFC2581—TCP congestion control," RFC, 1999.
[14] K. Fall, S. Floyd, Simulation-based comparisons of Tahoe, Reno, and SACK TCP, ACM Computer Communication Review 26 (3) (1996) 5–21.
[15]A. Veres, M. Boda, The chaotic nature of TCP congestion control, in: Proceedings of IEEE INFOCOM, 2000, pp. 1715–1723.
[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanov, "RFC2018—TCP selective acknowledgment options," RFC, 1996.
[17] M. Mathis and J. Mahdavi, "Forward acknowledgement: refining TCP congestion control," in Proc. conference on applications, tech- nologies, architectures, and protocols for computer communications (SIGCOMM), New York, NY, USA, 1996, pp. 281–291.
[18] L. Brakmo and L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," IEEE J. Sel. Areas Commun.,vol.13, no. 8, pp. 1465–1480, October 1995.
[19] K. Srijith, L. Jacob, and A. Ananda, "TCP Vegas-A: Improving the performance of TCP Vegas," Computer Communications, vol. 28, no. 4, pp. 429–440, 2005.
[20] S. Floyd, "RFC3649—HighSpeed TCP for large congestion windows," RFC, 2003.
[21] S. Floyd, HighSpeed TCP and Quick-Start for Fast Long-Distance HighSpeed TCP and Quick-Start for fast longdistance networks (slides), TSVWG, IETF, March 2003.
[22] S. Floyd, T. Henderson, and A. Gurtov, "RFC3782—the NewReno modification to TCP's fast recovery algorithm," RFC, 2004.
[23] S. Floyd, "RFC3742—Limited slow-start for TCP with large conges- tion windows," RFC, 2004.
[24] T. Kelly, "Scalable TCP: improving performance in highspeed wide area networks," Computer Communications Review, vol. 32, no. 2, April 2003.
[25] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long- distance networks," in Proceedings of PFLDnet, 2004.
[26] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," International J. Satellite Communications and Networking, vol. 22, pp. 547–566, 2004.
[27] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the perfor- mance of TCP pacing," in Proc. IEEE INFOCOM, vol. 3, March 2000, pp. 1157–1165.
[28] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "TCP Libra: Exploring RTT-Fairness for TCP," UCLA Computer Science Department, Tech. Rep. UCLA-CSD TR-050037, 2005.
[29] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast, long distance networks," in Proc. IEEE INFOCOM,vol. 4, March 2004, pp. 2514–2524.
[30] I. Rhee and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64– 74, July 2008.
[31] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu, "A step toward realistic performance evaluation of high-speed TCP variants," in Fourth Interna- tional Workshop on Protocols for Fast Long-Distance Networks, Nara, Japan, March 2006.

[32] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet an- other highspeed TCP," in Proc. PFLDnet,ISI,Marina Del Rey (Los Angeles), California, February 2007.

[33] C. Jin, D. Wei, S. Low, G. Buhrmaster, J. Bunn, D. Choe, R. Cottrel, J. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST TCP: from theory to experiments," December 2003.

[34] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motiva- tion, architecture, algorithms, performance," IEEE/ACM Trans. Netw., vol. 14, no. 6, pp. 1246–1259, 2006.

[35]S. Belhaj, "VFAST TCP: an improvement of FAST TCP," in Proc. Tenth International Conference on Computer Modeling and Simulation, 2008.

[36] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "CapProbe: A simple and accurate capacity estimation technique," in Proceedings of SIGCOMM, Portland, Oregon, USA, August/September 2004.

[37] J. Sing and B. Soh, "TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links," in Proc. 4th IEEE International Symposium on Network Computing and Applications (IEEE NCA05), 2005, pp. 73–80.

[38] R. Braden, "RFC1122—Requirements for Internet Hosts - Communi- cation Layers," RFC, 1989.

[39] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," ACM SIGCOMM Computer Communication Review, vol. 21, no. 4, pp. 133– 147, 1991

[40] D. Wei, P. Cao, and S. Low, "TCP Pacing Revisited," in Proceedings of IEEE INFOCOM, 2006.

[41] H. Shimonishi and T. Murase, "Improving efficiency-friendliness trade- offs of TCP congestion control algorithm," in Proc. IEEE GLOBE- COM, 2005.

[42] K. Kaneko, T. Fujikawa, Z. Su, and J. Katto, "TCP-Fusion: a hybrid congestion control algorithm for high-speed networks," in Proc. PFLD- net, ISI, Marina Del Rey (Los Angeles), California, February 2007.

[43] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP," in Proc. IEEE INFOCOM, vol. 3, March 2005, pp. 1838–1848.

[44] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," July 2005.

[45] S. Liu, T. Basar, and R. Srikant, "TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks," in Proc. First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), 2006.

[46] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet an- other highspeed TCP," in Proc. PFLDnet,ISI,Marina Del Rey (Los Angeles), California, February 2007