Global Journal of Computer Science and Technology: E
Network, Web & Security
Volume 14 Issue 1 Version 1.0 Year 2014
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Modified TCP for Time Critical Applications

By Abhay Kumar
JB Institue of Engineering and Technology, India

Abstract- A network is defined to be a congested network if the load on the network exceeds the capacity of the network. The traditional congestion control technique of slow-start and AIMD was adopted when the aim was more on the stability of the Internet. But as more and more time critical applications such as multimedia applications are being used, we need alternate technique that reduces the drastic fluctuations of window size present in the existing technique. Thispaper proposes a techniquefor fast delivery of packet for a time critical application. It reduces the packet overhead and time compared to existing slow-start and AIMD technique. The proposed technique uses information or intelligence from the unexpected packet received. It is a fine modification of the existing slow-start and AIMD technique by adapting them for time critical applications. We propose modification at both thesender and the receiver hosts without modifying anything in the intermediate hosts of the network. Extensivesimulation show that proposed technique reduces congestion in the network by reducing both packet overhead and time compared to the traditional slow-start and AIMD technique and delivers the packets in timely manner than the existing techniques.

Keywords: *network protocols, TCP, congestion control, slow-start, aimd.*

GJCST-E Classification : *C.2.5*

Strictly as per the compliance and regulations of:

Modified TCP for Time Critical Applications

Abhay Kumar

Abstract- A network is defined to be a congested network if the load on the network exceeds the capacity of the network. The traditional congestion control technique of slow-start and AIMD was adopted when the aim was more on the stability of the Internet. But as more and more time critical applications such as multimedia applications are being used, we need alternate technique that reduces the drastic fluctuations of window size present in the existing technique. Thispaper proposes a techniquefor fast delivery of packet for a time critical application. It reduces the packet overhead and time compared to existing slow-start and AIMD technique. The proposed technique uses information or intelligence from the unexpected packet received. It is a fine modification of the existing slow-start and AIMD technique by adapting them for time critical applications. We propose modification at both thesender and the receiver hosts without modifying anything in the intermediate hosts of the network. Extensivesimulation show that proposed technique reduces congestion in the network by reducing both packet overhead and time compared to the traditional slow-start and AIMD technique and delivers the packets in timely manner than the existing techniques.

Keywords: network protocols, TCP, congestion control, slow-start, aimd.

I. Introduction

The Internet is a global network of interconnected computers which allows individuals and organizations around the world to communicate and share information with each other. This demand has natural fluctuation; therefore, the Internet performance is largely governed by it, leading to possible congestion which occurs when resource demands exceed the capacity of the network. Due to the explosive growth of the Internet and increasing demand for multimedia applications like voice over IP, real-time video streaming, IPTV and financial transactions, the issue of congestion has received tremendous attention from academia and industry. Transmission of real-time multimedia applications typically has large bandwidth, small delay and low-loss requirements. However, the current Internet does not guarantee any quality of service (QoS) as it is based on best-effort service model of IP [1]. A network is said to be congested from the perspective of a user if the service quality noticed by the user decreases because of an increase in network load. The goal of congestion control mechanisms is simply to use the network as efficiently as possible, that is, attain the highest possible throughput while maintaining a low loss ratio and small delay. Congestion must be avoided because it leads to queue growth and queue growth leads to delay and loss [2

As the network grew, it was clear that unrestricted data transfer by many users over a shared resource, i.e., the Internet, could be bad for the end users; excess load on the links leads to packet loss and decreases the effective throughput. This kind of loss was experienced at a significant level in the '80s and was termed congestion collapse [5]. Thus, there was a need for a protocol to control the congestion in the network, i.e., control the overloading of the network resources. It led to the development of a congestion control algorithm for the Internet by Van Jacobson [5]. This congestion control algorithm was implemented within the protocol used by the end hosts for data transfer called the Transmission Control Protocol (TCP).

There are several different flavors of TCP congestion control, each of which operates somewhat differently. But most of the versions of TCP are window-based protocols, wherein the idea is that each user maintains a number called a window size, which is the number of unacknowledged packets that are allowed to be sent into the network. Any packet from the new window can be sent only when an acknowledgment for the last packet in the previous sent window is received by the sender. TCP adapts the window size in response to congestion information. The window size is increased if the sender determines that there is excess capacity present in the route and decreases if the sender determines that the current number of in-flight packets exceeds the capacity of the route. The exact means of determining whether to increase or decrease the window size is what determines the difference between the congestion control mechanisms of different TCP flavors. The most commonly used TCP flavors used for congestion control in the Internet today are Reno and New Reno [12]. Both of them are updates of the TCP-Tahoe, which was introduced in 1988[5]. Although, they vary significantly in many regards, the basic approach to congestion control is similar. The idea is to use successful reception of packets as an indication of available capacity and dropped packets as an indication of congestion. In most cases, eachtime the destination receives a packet, it sends an acknowledgement (also called ACK) asking for the next packet in sequence to be transmitted. When an acknowledgment for a windowis received, the protocol increases its window size. However, on reception of three duplicate acknowledgments or dupacks (i.e., four successive

Author: JB Institue of Engineering and Technology, Hyderabad, India. e-mail: abhay1880@gmail.com

identical acknowledgments) by the sender is taken by it as an indication that packet has been lost due to congestion. In case the source does not receive any acknowledgement for a finite time (RTO [13]), it assumes that all unacknowledged packets have been lost. In both the cases the source aggressively proceeds to cut down the window size and retransmit the lost packets.

TCP Vegas improves upon TCP Reno through three main techniques. The first is a new retransmission mechanism where timeout is checked on receiving the first duplicate acknowledgment, rather than waiting for the third duplicate acknowledgment, and results in a more timely detection of loss. The second technique is a more prudent way to grow the window size during the initial use of slow-start when a connection starts up, and it results in fewer losses. The third technique is a new congestion avoidance mechanism that corrects the oscillatory behavior of Reno. The idea is to have a source estimate the number of its own packets buffered in the path and try to keep this number between α (typically 1) and β (typically 3) by adjusting its window size. The window size is increased or decreased linearly in the next round-trip time according to whether the current estimate is less than α or greater than β. Otherwise the window size is unchanged. The rationale behind this is to maintain a small number of packets in the pipe to take advantage of extra capacity when it becomes available. A source periodically measures the round-trip queuing delay and sets its rate to be proportional to the ratio of its round-trip propagation delay to queuing delay, the proportionality constant being between α and β. Hence, the more congested its path, the higher the queuing delay and the lower the rate. The Vegas source obtains queuing delay by monitoring its round-trip time (the time between sending a packet and receiving its acknowledgment) and subtracting from it the round-trip propagation delay [7].

In today's Internet, real-time applications such as VoIP, videoconferencing and on-line gaming mostly use RTP over UDP or UDP alone to transport data. Because these protocols are unresponsive to congestion events, the growing popularity of applications that use them endangers the stability of the Internet. So, to make it possible that real-time applications are widely adopted, common congestion control mechanisms suitable for real time multimedia are expected to be deployed[3] [4].

The existing techniques does not use any information or intelligence from the unexpected packet received, unexpected packets are simply discarded. The proposed techniques tries to retrieve information based on the unexpected packet received and perform the congestion control accordingly.

The remaining paper is organized as follows: Section II explains the system or network model used in this paper. Section III describes our proposed Unexpected Packet based Congestion Control (UPCC) Technique. Section IV presents the simulation results that demonstrate our proposed UPCC technique reduces congestion in the network compared to traditional slow-start and AIMD technique. Finally Section V concludes the paper.

II. System Model

This paper considers a realistic computer network consisting of several sources and destinations connected via multiple routers and links. The source (sender) communicates to the destination (receiver) in form of packets. The series of routers and links that a packet follows from the source to destination is called a route. A pair of sender and receiver may be connected via multiple routes. This network is represented in the Figure 1.

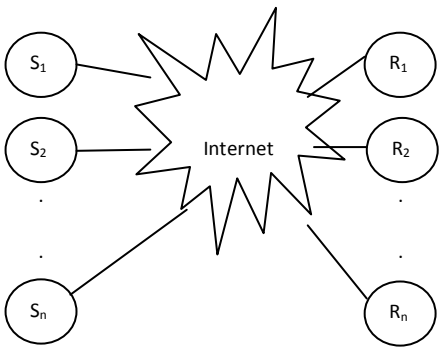

Figure 1 : Network model

For simplicity of the explanation, we consider only a pair of sender (S) and receiver (R) connected via multiple routes, as shown in Figure 2. The sender and the receiver may be running multiple different applications. However, the packets of the application are transmitted using the first come first serve policy. The connection is established using three-way handshake as in case of existing TCP. However, this paper proposes few modifications in this phase also to make the subsequent transmissions congestion aware.

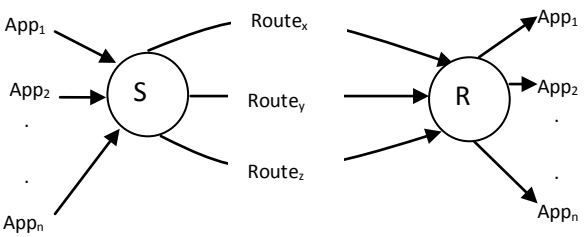

Figure 2 : Simplified network model

TCP operates in two distinct phases. When file transfer begins, the window size is 1, but the source rapidly increases its transmission window size so as to reach the available capacity quickly. Let us denote the window size by W. The algorithm increases the window size by 1 each time an acknowledgement for a packet indicating success is received. This is called the slow-start phase. Since one would receive acknowledgements corresponding to one window's worth of packets in an RTT [13], and we increase the window size by one for each successful packet transmission, this also means that (if all transmissions are successful) the window would double in each RTT, so wehave an exponential increase in rate as time proceeds. Slow-start refers to the fact that the window size is still small in this phase, but the rate at which the window is increased is quite rapid. When the window size either hits a threshold, called the slow-start threshold (ssthresh) or the transmission suffers a loss (immediately leading to a halving of window size), the algorithm shifts to a more conservative approach called the congestion avoidance phase. When in the congestion-avoidance phase, the algorithm increases the window size by 1 every time feedback of a successful packet transmission in the corresponding window is received. When a packet loss is detected by the receipt of three dupacks, the slow-start threshold (ssthresh) is set to half of the current window i. e TCP Reno cuts its window size by half (W ← W/2) and algorithm enters additive increase phase where it start sending segments from current window onwards. Thus, in each RTT, the window increases by one packet i.e., a linear increase in rate. Protocols of this sort where increment is by a constant amount, but the decrement is by a multiplicative factor are called additive increase multiplicative decrease (AIMD) protocols. When packet loss is detected by a timeout, the slow-start threshold (ssthresh) is set to half of the current window and the algorithm enters the slow-start phase i.e., it start sending from 1 packet onwards. Let us call the congestion window at time t as W (t). This means that the number of packets in-flight is W (t). The time taken by each of these packets to reach the destination, and for the corresponding acknowledgement to be received is RTT. The RTT is a combination of propagation delay and queuing delay. A window-based congestion control scheme defines one control rule for window increase, and another rule forwindow decrease. AIMD uses the following control rule [19]:

$$\text{Increase: } W_{t+1} \leftarrow W_t + \alpha, \quad \alpha > 0$$

$$\text{Decrease: } W_t \leftarrow W_t - \beta W_t, \quad 0 < \beta > 1$$

Where α and β refer to the additive increase constant and multiplicative decrease constant β respectively. The standard TCPuses the value of these constants α and β as 1 and 0.5 respectively.

This subsection provides the definition of several terms and the notations that will beused throughout the remainder of this paper.

- *SYN:* To establish a connection, TCP uses a three-way handshake. Synchronize (SYN) [9] packet is the first control packet sent for the three-way handshake by the sender wishing to establish the TCP connection.
- *ACK:* An acknowledgement (ACK) [14] is a control packet used between communicating processes or computers to signify receipt of receiving a data packet, and it is a part of a communication protocol. For example, ACK packets are used in the Transmission Control Protocol (TCP) to acknowledge the receipt of SYN packets while establishing a connection in three-way handshake, and acknowledge the receipt of data packets while a connection is in data transfer phase.
- *SS-AIMD:* In the Slow-Start (SS) [5] [8] and Additive Increase Multiplicative Decrease (AIMD)[5] [14] algorithm, when a TCP connection first starts, the slow-start phase initializes a congestion window to one packet and transmits. After receiving acknowledgement from the receiver, the window increases by one packet for each acknowledgement returned. After successful transmission of these two packets and acknowledgements received, the window is increased to four packets and so on, doubling from there up to a threshold known as slow-start threshold (ssthresh). After slow-start threshold, the algorithm enters into additive increase multiplicative decrease (AIMD) phase where window increases by one packet for successful transmission of all the packets in the window i.e., additive increase. In this phase, the transmission rate slows down to avoid congestion. But whenever a packet is lost, the sender immediately sets its transmission window to one half of the current window size i.e., multiplicative decrease.
- *ssthresh:* Slow-start threshold (ssthresh)[2] is a point where slow-start phase ends and additive increase multiplicative decrease (AIMD) phase starts.
- *dupacks:* When receiver receives a TCP packet with a sequence number higher than the expected one (out of turn packet). The receiver sends an immediate ACK with the Acknowledgement field set to the Sequence number the receiver was expecting. This ACK is a duplicate of an ACK (dupacks) [2] which was sent previously. This is done to update the sender with regards to the missing TCP packets.
- *rwnd:* Receiver advertised window (rwnd)[10] or receiver queue capacity is the most recent advertised window that contains the number of packets a receiver can process. This is one of the

two variables which affect how much unacknowledged data a sender can send; the other variable is congestion window. The receiver advertised window is the value of the window field in a TCP packet header.

- *cwnd:* Congestion window (cwnd)[12] is a TCP state variable maintained at the sender that limits the amount of data a TCP can transmit without facing congestion through the network. At any given time, a TCP transmit minimum of congestion window and receiver advertised window.
- *TCP:* The Transmission Control Protocol (TCP)[14] is used as a highlyreliable host-to-host protocol between hosts in packet-switched computercommunication networks, and in interconnected systems of such networks.TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of transport layer protocolswhich support multi-network applications. The TCP provides for reliable inter-process communicationbetween pairs of processes in host computers attached to distinct but interconnected computer communication networks.
- *UDP:* The User Datagram Protocol (UDP)[15] is defined as a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [16] is used as the underlying protocol. User Datagram Protocol is unreliable connection-less protocol used at transport layer
- *IP:* The Internet Protocol (IP)[17] is designed for use in interconnected systems ofpacket-switched computer communication networks. The internet protocol provides fortransmitting blocks of data called datagram from sources to destinations. The internet protocol also provides forfragmentation and reassembly of long datagram, if required, fortransmission through "small packet" networks. Internet Protocol is unreliable connection-less protocol used at network layer
- *RTP:* The real-time transport protocol (RTP)[18] provides end-to-end network transport functions suitable forapplications transmitting real-time information, like audio, video ordata, over multicast or unicast network services. RTP does not provide resource reservation and also does not guarantee quality-of-service for real-time services. This transport protocol is also augmented by another real-time control protocol (RTCP) to allow monitoring of the data delivery in amanner scalable to large multicast networks, and to provide minimalcontrol and identification functionality. RTP and RTCP are designedto be independent of the underlying transport and network layers.
- *VoIP:* Voice over Internet Protocol (VoIP) [3] is a mechanism that allows telephone calls to be made over computer networks like the Internet. VoIP converts analog voice signals into digital data packets and supports real-time, two-way transmission of conversations using Internet Protocol.
- *IPTV:* Internet Protocol television (IPTV)[3] is the process of transmitting and broadcasting television programs using the Internet protocol suite over a packet-switched network such as the Internet, instead of being delivered through traditional terrestrial, satellite signal and cable television formats.
- *RTO:* The retransmission timeout (RTO) [13] is aretransmission timer used by the Transmission Control Protocol to ensure data delivery in the absence of anyfeedback from the remote data receiver. The duration of this timeris referred to as RTO. The retransmission timeout timer is used for retransmissions of lost or delayed packet.
- RTT (Δ): Round trip time (RTT)[13] is the length of time it takes for a packet to be sent and the length of time it takes for an acknowledgment of that packet to be received
- *QoS:* Quality of service (QoS) [2] is the ability to provide different priority to different applications, users, or data flows i.e., it guarantees a certain level of performance to a data flow. Quality of service guarantees are important if the network capacity is insufficient, especially for real-time multimedia applications such as voice over IP, online games and IPTV, since these applications often require fixed bit rate and are delay sensitive. A best-effort network like Internet does not support quality of service.

Table 1 : Notations.

X_E	The number of windows and hence number of acknowledgements used in the existing slow-start and AIMD technique
X_P	The number of windows and hence number of acknowledgements used in the proposed Unexpected Packet based Congestion Control technique
T_E	The total time required to transmit an application in the existing slow-start and AIMD technique
T_P	The total time required to transmit an application in the proposed Unexpected Packet based Congestion Control technique
W	Window size
W_s	Window start
W_{end}	Window end
Δ	Round trip time
δ	Time required to transmit consecutive packets in a window

III. Proposed Unexpected Packet Based Congestion Control (UPCC) Technique

The proposed technique is a fine modification of the existing slow-start and AIMD technique by adapting it and making congestion aware. We propose modification at both the sender and receiver hosts without modifying anything in the intermediate hosts of the network. The proposed modification can be described in the form of a dialogue between sender and receiver from initiation to the termination of a connection.

At sender side:

Whenever a sender host wants to communicate it will send a SYN (i) packet to the receiver host expressing its desire to communicate as in existing technique [8] [9]. On sending the SYN(i) packet the sender will start a timer based on RTT within which it should ideally receive an ACK (i+1) packet from the receiver. This can be seen in Figure 3. In case, he does not receive an ACK (i+1) packet, he assumes that there is congestion in the network and therefore it retransmit SYN (i) packet with doubled RTT. This information about congestion is stored in a separate variable 'C' that will be used in data transfer stage, i.e., it set C=1. This communication can be seen in Figure 4.

Algorithm for three-way handshake at sender

start

Send a SYN message and start a timer
Wait for an ACK
If timer expires and no ACK received
C=1
Resend the SYN message with RTT=2RTT

stop

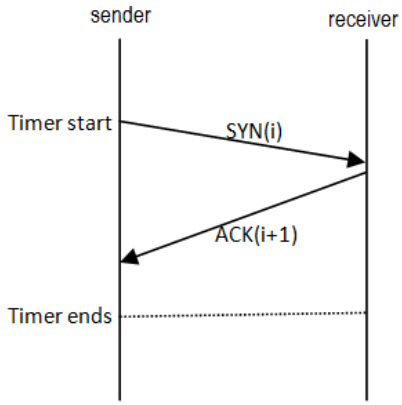

Figure 3 : Three-way handshakein an ideal condition.

At receiver side:

On receiving a SYN(i) packet it will send an ACK(i+1) packet containing its available queue capacity 'rwnd' together with its own SYN(j) and set C=0 to inform its readiness for communication and no congestion perceived so far. To complete the three-way handshake of TCP connection it starts its timer waiting for an ACK (j+1) from sender for his SYN (j) as shown in Figure 5. However, if it receives unexpected duplicate SYN (i) message or no ACK (j+1) within its RTT, it indicates that its ACK (i+1) or ACK (j+1) was lost and hence congestion may be present. It responds to this new SYN (i) received or RTT time out by

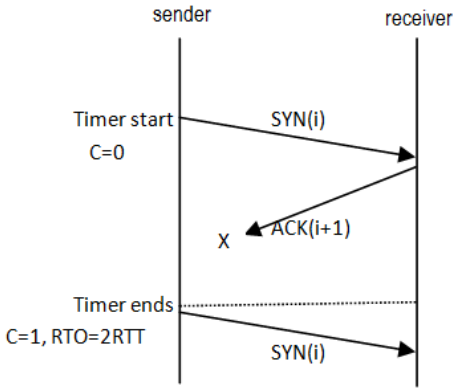

Figure 4 : Three-way handshakewhen ACK from receiver is lost.

retransmitting with a packet containing SYN(j), ACK(i+1), and rwnd. This information about congestion is stored in a separate variable 'C' that will be used in data transfer stage, i.e., it set C=1. This communication can be seen in Figures 6 and 7.

Algorithm for three-way handshake at receiver

start

If (SYN message received)
Send ACK+SYN message and start a timer
If timer expires and no ACK received or duplicate SYN (i) is received
C=1
Resend the ACK+SYN message with RTT=2RTT

stop

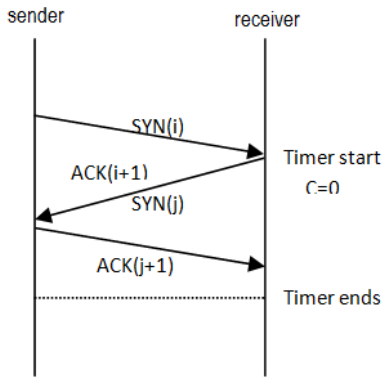

Figure 5 : Three-way handshake in an ideal condition

After the three-way handshake is completed, proposed algorithm enters the data transfer phase. However, during the handshake if no timer expires or no duplicate SYN or ACK packets are received, the

proposed technique presumes network to be congestion free. Thus, it advocates an aggressive start wherein the window size is set to be equalto the receiver queue capacity 'rwnd'. On the other hand, a congestion may be perceived when C=1 at either the sender or receiver side. In such case we follow the same existing

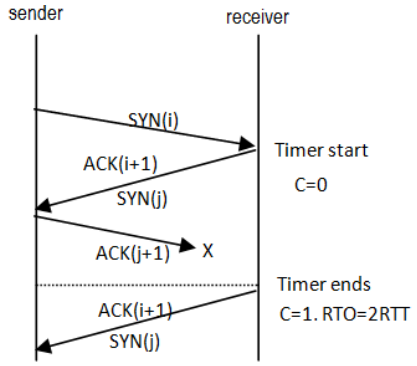

Figure 6 : Three-way handshake when ACK from sender is lost.

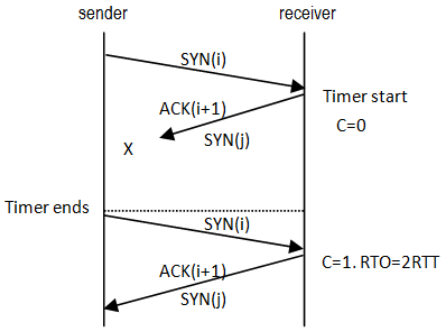

Figure 7 : Three-way handshake when receiver receives duplicate unexpected SYN(i).

slow-start and AIMD technique[5] [8] for selecting the window size. After the selection of window size is made, the data transfer phase is initiated by the sender and the dialogue continues as follows:

Algorithm for window selection

start

If C=0

Window Size = rwnd

// we apply aggressive start i.e., it does not depend on cwnd as per standard TCP [5]

If C=1

Window Size = min (cwnd, rwnd)

// we apply the standard TCP rule i.e., slow-start with AIMD [8]

stop

At sender side:

The sender will start sending the packets up to the window size (W_s, W_{end}) but it doesn't expects any ACK till it completes sending the entire window. In other words, it expects one ACK (w_{end}) per window. In ideal condition it will receive the ACK (w_{end}) and assumes no congestion C=0 and will adjusts the window as per the policy defined above, in the algorithm for window selection.

At receiver side:

On receiving the ACK (j+1) with the window size it will set its window and will wait to receive the data packets. When requisite packets arrive it acknowledges them by sending ACK (w_{end}) for the same. However, at any point of time, if it feels overloaded or underloaded, it will send its updated queue capacity 'rwnd' to the sender piggybacking with ACK(k) where k-1 is the last packet accepted from the sender.

At sender side:

If it receives an unexpected ACK (k) (as it expects only ACK (w_{end}) for any window) then it will simply slides the window such that it starts with the first unacknowledged packet, i.e., packet with sequence number k. Further, it adjusts the window according to the new 'rwnd' suggested by the receiver. Thus, on receiving one unexpected ACK (k) the sender simply slides and adjust the window size and again expects one ACK (w_{end}) within the RTT of the new window. This communication can be seen in Figure 8 where 'k = n+3' and new 'rwnd = 12'.

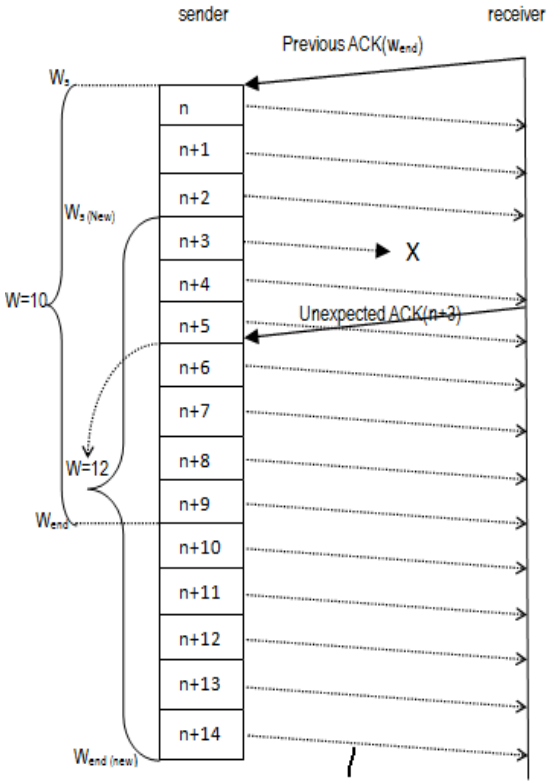

Figure 8 : Data transfer phase when sender receives an unexpected ACK with new rwnd

At receiver side:

The above dialogue presumes that no congestion exists and hence, no packet loss occurs. However, if the receiver finds an out of turn packet it indicates that the intermediate packet/s could be lost. In such case it will send an ACK (k) with current 'rwnd' for the last in order packet i.e k-1 received. It will also slide its window but it does not expect a retransmission of the intermediate packet/s as they may be delayed. However, if it further receives second out of the turn packet it presumes that intermediate packet/s is lost. It sends a duplicate ACK (k) with current 'rwnd' and starts a timer based on RTT within which it should receive the lost packet. In case it does not, it will resend an ACK (k). This communication can be seen in Figure 9.

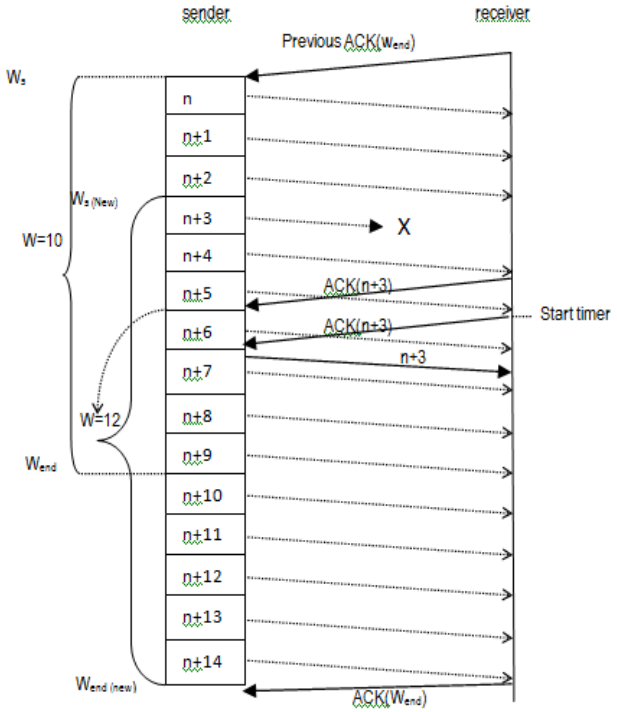

Figure 9 : Data transfer phase when sender receives duplicate unexpected ACK

At sender side:

On receiving the first unexpected ACK (k), the sender simply slides the window as was discussed in Figure 8. But if it receive a duplicate ACK (k), i.e., two ACK (k) it indicates that mild congestion is present in the network. This assumption of mild congestion is based on the understanding between sender and receiver that two duplicate acknowledgements will be send by the receiver only when the receiver receives two out of turn packets. Therefore, it must retransmit only that missing k^{th} packet and continue with sending the packets from first non-transmitted packets in the current window and expect the ACK (w_{end}) for the current entire window. This communication can be seen in Figure 9.

At receiver side:

On receiving the missing packet, it will place it in order and continue receiving till the end of window. If all the packets arrive, the receiver will send the ACK (wend). However, if it misses another packet in the same window, it indicates that the congestion is increasing and it will send the duplicate ACK (j) with 'rwnd'=rwnd/2 as shown in Figure 10.

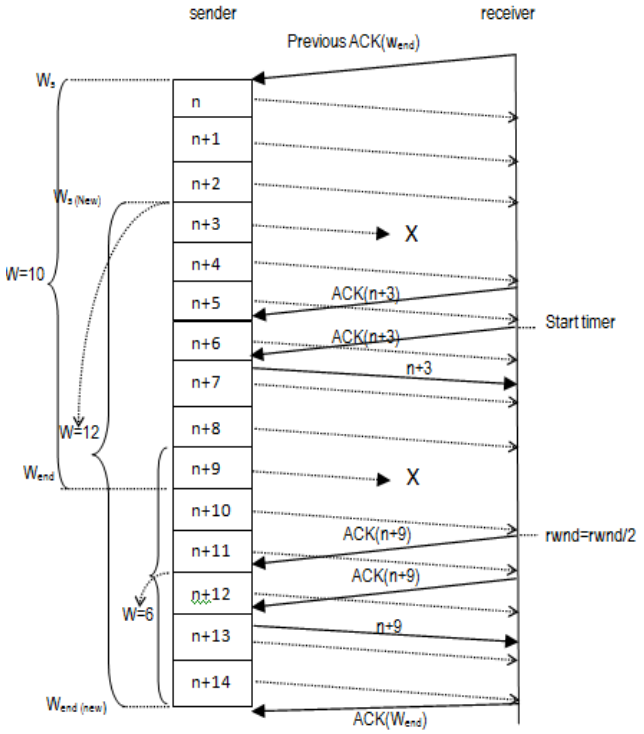

Figure 10 : Data transfer phase when sender receives second pair of unexpected ACK with reduced rwnd.

At sender side:

If sender receives another pair of unexpected ACK (j) in its current window, it indicates that the second packet in the same window has been lost implying that window size is too big. In such scenario the sender will slide the window to the first unacknowledged packet and retransmit the missing packet. It will also reduce its transmission window as indicated by the receiver to half. This communication can be seen in Figure 10.After transmission of the entire window the sender waits for RTT time to receive the acknowledgement ACK (wend). If it receives ACK (wend) within the stipulated time then he assumes that the network is congestion free and continues with the next window. However, if ACK wend) is not received within the RTT the sender presumes high congestion in the network. It retransmits the first packet in the window as shown in Figure 11, and starts the timer with RTT time as perexisting slow-start and AIMD algorithm [5] [8].

At receiver side:

If retransmission of a packet which is not asked by the receiver i.e., unexpected packet is received. The receiver will transmits the ACK (k) where k-1 is the last in order packet received. As demonstrated in the Figure 11, when the sender retransmits the first packet of the last unacknowledged window i.e., Ws=n+9 when it does not receive ACK (n+15) i.e., ACK (W_{end}) within its RTT, the

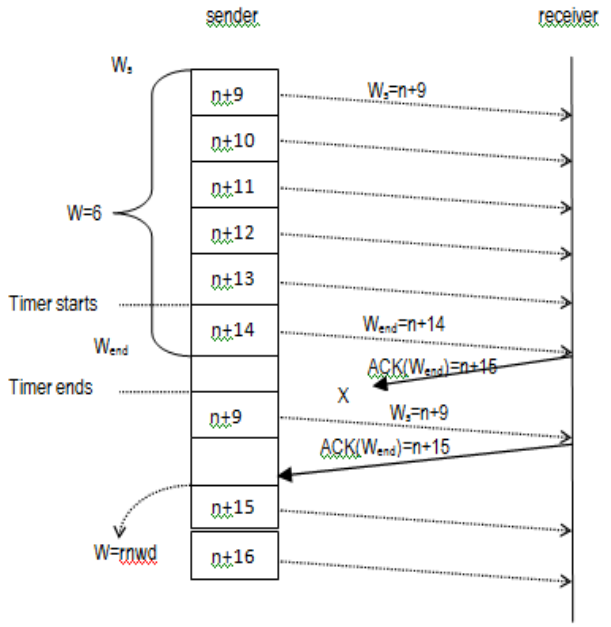

Figure 11 : Data transfer phase when ACK(W_{end}) for complete window is lost

receiver will respond by retransmitting the ACK (n+15) i.e., ACK (W_{end}) indicating the receipt of the complete window n+9 to n+14. By doing this the receiver avoids the retransmission of the remaining packets in the last unacknowledged window i.e., n+10 to n+14. For large window this is substantial reduction in retransmission improving the throughput of the network and reducing congestion.

At sender side

It may receive a unexpected delayed ACK (Wend) in response to Ws retransmitted by it for the previous window. As shown in Figure 11, it receives ACK (n+15) i.e., ACK (W_{end}) in response to its retransmission of packet W_s=n+9. Theexisting techniques [8] will discard this ACK (W_{end}). However, in the proposed technique it is not simply discarded but is used to indicate mild congestion and the previous window packets have been received without any problem. Thus, sender should stop retransmission and recover from slow-start phase by sliding the window up to the first unacknowledged packet and continue with the original window size.

Both sender and receiver utilize the congestion information received for one connection over all other connections made by them leading to recovery from the congestion by the network.

The proposed Unexpected Packet based Congestion Control (UPCC) technique is illustrated with the following motivational example which will illustrate the limitations of existing techniques.

Let an application have 1024 packets and as considered by the authors in [10] [11], the slow-start threshold (ssthresh) as 40 packets and receiver advertised window (rwnd) as 50 packets, we also consider the same. We estimate the packet overhead gain and the time gain for the existing slow-start AIMD technique and the proposed UPCC techniques as follows.

a) Existing SS-AIMD [8] technique in congestion free network

The existing slow-start technique [5] will initially set the window as one packet. When its corresponding ACK arrives, the source sets the corresponding window to two packets. It then transmits two packets. On receiving the two corresponding ACK, it sets the window size to four and so on. Therefore, the slow start technique increases the window size from 1 exponentially up to ssthreshold of 40 packets, forming a geometric progression of 1, 2, 4, 8, 16, and 32. From ssthreshold to rwnd, it will perform additive increase as arithmetic progression of 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 and 50. Beyond rwnd, the current size of the window cannot increase because it has to be minimum of cwnd and rwnd, thus it remains constant at rwnd. The existing technique during the slow start phase will expect an acknowledgement per packet, while in the subsequent phase only one ACK per window will be received. Hence, apart from the 1024 data packet additionally 84 ACK packets will be required.

However, to give the existing technique a fair chance we assume that only one ACK per window is required. Mathematically, the minimum number of ACKs required to transmit the application of 1024 packets in congestion free network is

$$X_E = n_{GP} + n_{AP} + \left\lceil \frac{Number\ of\ packets\ to\ be\ transfered\ -(S_{AP}+S_{GP})}{rwnd} \right\rceil$$

Where

$n_{GP} = \lceil log_2 ssthresh \rceil$

n_{AP} = (rwnd-ssthresh)+1

$S_{AP} = \frac{(rwnd - ssthresh+1)(ssthresh+rwnd)}{2}$

$S_{GP} = 2^{\lceil log_2 ssthresh \rceil}-1$

Thus, the variation in window size will be as 1, 2, 4, 8, 16, 32,40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,50, 50, 50, 50, 50, 50, 50, 50, 50and 16. In other words, additional X_E=27 number of acknowledgements must be sent by the receiver to acknowledge the correct

receipt of each window, causing the packet overhead in the network.

The time required by the slow-start technique to transmit initial packets in the windows of 1, 2, 4, 8, 16, and 32 will be $\Delta + (\Delta+\delta) + (\Delta+3\delta) + (\Delta+7\delta) + (\Delta+15\delta) + (\Delta+31\delta)$ where Δ is RTT and δ is the time to transmit consecutive packet in a window. Similarly, the time required for transmitting packets in the remaining windows can be estimated. Therefore, the total time required by the slow-start and AIMD technique will be $T_E = 27\Delta + 997\delta$.

In the following subsection we discuss packet overhead gain and time gain for the proposed Unexpected Packet based Congestion Control technique in congestion free network.

b) Proposed UPCC technique in congestion free network

In congestion free network, the proposed UPCC technique advocates the use of rwnd (receiver advertised window) as a window size for the transmission. Therefore, the number of ACKs required for transmitting the application of 1024 packets will be

$$X_P = \left\lceil \frac{Number\ of\ packets\ to\ be\ transfered}{rwnd} \right\rceil$$

Hence, X_P=21, implying 21 acknowledgements are required as compared to 27 acknowledgements in the existing technique. Thus, approximately 22% reduction in the number of acknowledgements is achieved through proposed UPCC technique.

Further, the time required to transmit an application of 1024 packets will be $T_P = (\Delta+49\delta) \times 20 + (\Delta + 23\delta) = 21\Delta + 1003\delta$. Thus, the proposed UPCC technique reduces the time by approximately 12% leading to lesser chance of congestion in the network.

In the following subsection we discuss packet overhead gain and time gainin congested network where packet loss may occur while transmitting this application.

c) Existing SS-AIMD [8] technique with single packet loss

Consider that 240th packet is lost while transmitting 11th window where 44 packets can be transmitted without waiting for the acknowledgement. When receiver receives out of turn packets, it sends duplicate acknowledgement. When the sender receives 3 duplicate ACKs, it indicates mild congestion. The existing algorithm updates ssthreshold, cwnd and window size as $ssthreshold = \text{cwnd}/2$, $cwnd = ssthreshold$, and $window\ size = \min(cwnd, rwnd)$ respectively. It retransmits the lost packet and enters in to AIMD phase directly. In AIMD phase, the window starts increasing additively from new calculated ssthreshold to rwnd(receiver advertised window) as arithmetic progression. Thus, the variation in window size will be 1, 2, 4, 8, 16, 32,40, 41, 42, 43, 44(loss occurs),22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43 and 37. Thus, $X_E = 34 + 3$, i.e., 37 acknowledgements which include three duplicate acknowledgements must be sent by the receiver to acknowledge the correct receipt of each window, causing the packet overhead on the network. The total time required for this process will be $T_E = 34\Delta + 991\delta$.

The subsection discusses the performance of the proposed technique under the same scenario.

d) Proposed UPCC technique with single packet loss

When 240th packet is lost and receiver receives the 241st packet after the receipt of 239th packet. It sends an acknowledgement for the correct receipt of 239th and expecting 240th packet, i.e., ACK (240). However, when it receives the 242nd packet it will resend the ACK (240). When sender receives two ACK (240), it retransmits the lost packet and keeps the $window\ size = rwnd$(receiver advertised window) indicating mild congestion. Thus, the proposed UPCC technique in congested network with single packet loss requires,$X_P = 23$, acknowledgements where 2 extra acknowledgements are used for informing loss of 240th packet. Approximately 38% and 24% reduction in the packet overhead and time is achieved respectively.

Further, the performance of the proposed UPCC technique is evaluated with respect to the existing SS-AIMD technique under severe congestion wherein the ACK is not received within the stipulated RTO time, i.e., when RTO timer expires.

e) Existing SS-AIMD technique [5] when RTO timer expires

Whenever sender's RTO timer expires before receiving acknowledgement, it indicates severe congestion. The sender presumes that the entire window is lost and starts retransmission by reducing the window size back to one. In the above example, while transmitting an application consisting of 1024 packets if the RTO timer expireswhen the sender window is 44. The algorithm updates ssthreshold, cwnd and window size as $\text{ssthreshold} = \text{current window}/2$, $\text{cwnd} = 1$ and $\text{window size} = \min(\text{cwnd}, \text{rwnd})$ respectively. It then starts retransmission by entering into slow-start phase where the window size increases exponentially from 1 up to new ssthreshold of 22 packets as geometric progression. After this it enters in AIMD phase, where the window size increases additively from new ssthreshold calculated to rwnd(receiver advertised window) as arithmetic progression. Thus, the variation in window size will be 1, 2, 4, 8, 16, 32, 40, 41, 42, 43, 44(timer expires), 1, 2, 4, 8, 16, 22, 23, 24, 25,…, 44 and finally 5. Hence, $X_E = 40$ number of acknowledgements must be sent by the receiver. The total time required by the existing SS-AIMD technique when timer expires will be $T_E = 44\Delta + 1028\delta$.

The performance of the proposed Unexpected Packet based Congestion Control (UPCC) technique when RTO timer expires is as follows:

f) Proposed UPCC technique when RTO timer expires

Whenever sender's RTO timer expires before receiving an acknowledgement, it indicates severe congestion due to single or multiple packet loss. In the proposed technique the sender starts retransmission of the first unacknowledged packet and waits for RTO time again. In its response the receiver will send the ACK of the last correctly received packet with the rwnd. This ACK will indicate that how many packets are lost. If one packet loss is perceived then the proposed technique will assume that the network had mild congestion and has recovered from it so it will continue with the rwnd received i.e., 50, 50, 50, 50, 50(timer expires), 1, 50, 50,…, 50, and 25. However, if multiple packets are lost then it updates rwnd and window size as $rwnd = rwnd/2$, and $window\ size = rwnd$ respectively. This reduction in window size will continue if repeatedly multiple packet loss occur , however, the window size will be boosted on successful transmission of a complete window as 50, 50, 50, 50, 50(timer expires), 1, 25, 50, 50, 50, …,50 and 48. Therefore, approximately 42% and 40% reduction in packet overhead is received in case of single and multiple packet loss when timer expires respectively. Further, reduction in the transmission time is perceived as approximately 35% and 32% lower for the proposed UPCC technique in the case of single and multiple packet lossrespectively when timer expires.

The above example demonstrated that as more and more packet are lost the performance of the proposed UPCC technique improves both in terms of packet overhead gain and time gain.

IV. Simulation Results

We perform extensive network simulations with the help of ns-2, the widely used open-source network simulator [20]. We compared our proposed Unexpected Packet based Congestion Control (UPCC) technique with traditional slow-start and AIMD technique (NewReno[12]) and found that proposed UPCC technique reduces the packet overhead by 22% to 40% as shown in Figure 12 and also reduces the time to transmit an application by 12% to 32% as depicted in Figure 13. The variations in packet overhead and time depend on the level of congestion present in the network. The simulations were conducted in three different categories as 1) congestion free 2) single packet loss and 3) multiple packet loss. Figures 14 and 15 gives the results for congestion free network that shows that proposed UPCC technique reduces packet overhead and time thus minimizing the chance of congestion in the network.

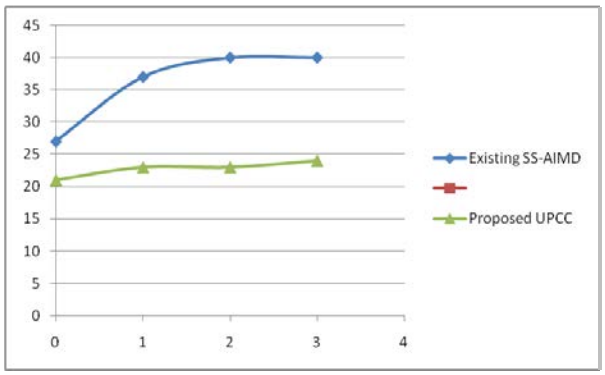

Figure 12 : Packet overhead gain of proposed UPCC vs SS-AIMD

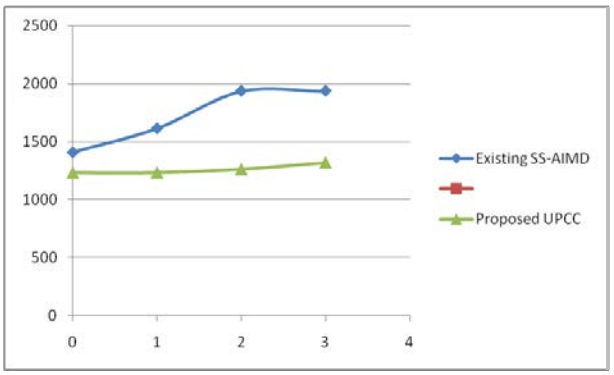

Figure 13 : Time gain of proposed UPCC vs SS-AIMD

Similarly, we conducted simulations for varying application sizes in case of multiple packet loss as shown in Figures 16 and 17 that clearly demonstrate that our proposed UPCC technique reduces packet overhead and time thereby minimizing the chance of congestion in the network.

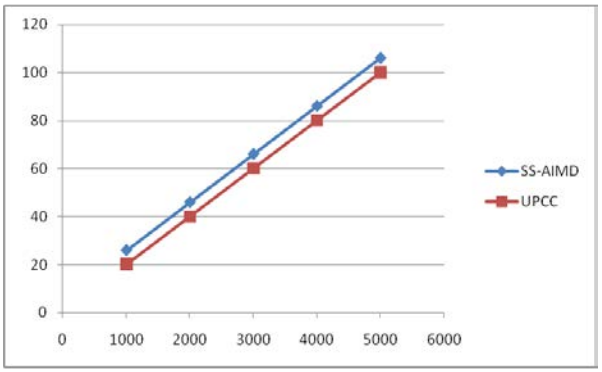

Figure 14 : Packet overhead gain of proposed UPCC vs SS-AIMD in congestion free network

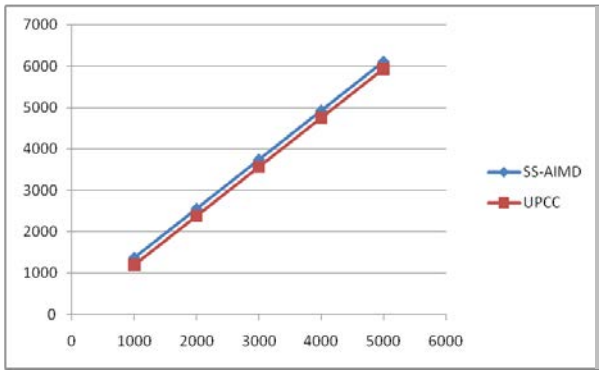

Figure 15 : Time gain of proposed UPCC vs SS-AIMD in congestion free network

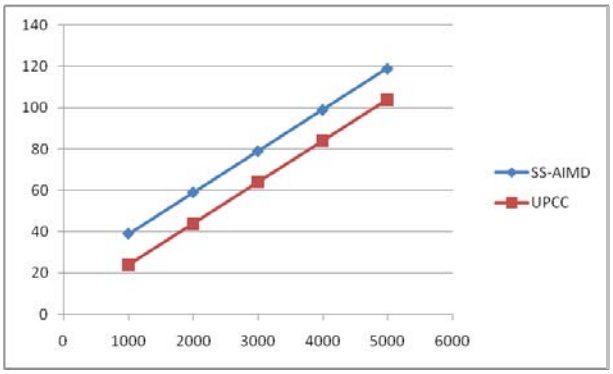

Figure 16 : Packet overhead gain of proposed UPCC vs SS-AIMD in congested network with multiple packet loss

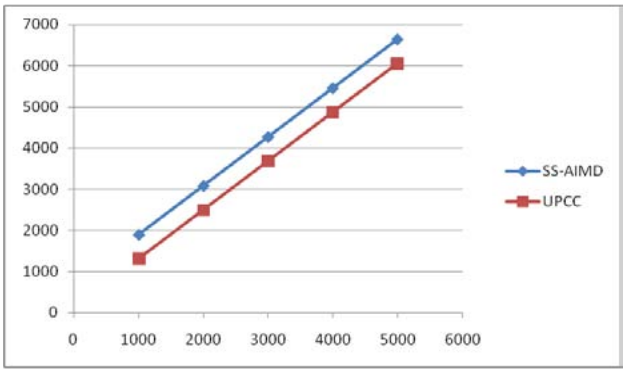

Figure 17 : Time gain of proposed UPCC vs SS-AIMD in congested network with multiple packet loss

V. Conclusions

In this paper we have demonstrated the benefit of using Unexpected Packet based Congestion Control (UPCC) technique. The simulation results shows that UPCC technique reduces the packet overhead and also reduces the time to transmit an application of various sizes as compared to the existing slow-start and AIMD technique.

References Références Referencias

1. R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, B. V. Steeg "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," Cisco Systems, December 10, 2012, Internet Draft, draft-pan-tsvwg-pie-00 (work in progress).
2. Michael Welzl, "Network Congestion Control, Managing Internet Traffic," John Wiley & Sons Ltd, 2005.
3. Filipe Abrantes, and Manuel Ricardo, "On Congestion Control for Interactive Real-time Applications in Dynamic Heterogeneous 4G Networks," March 17, 2005. This work was funded by the Portuguese Science and Technology Foundation.
4. S. Floyd and K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, Networking, IEEE/ACM Transactions on, August 1999.
5. Jacobson, "Congestion avoidance and control," ACM Computer Communication Review, vol. 18, pp. 314–329, August 1988.
6. Srinivas Shakkottai and R. Srikant, "Network Optimization and Control, Foundations and Trendsin Networking" Vol. 2, No. 3 (2007) 271–379, 2008 S. DOI: 10.1561/1300000007.
7. Steven H. Low, Fernando Paganini, and John C. Doyle, Internet Congestion Control, IEEE Control Systems Magazine, February 2002.
8. M. Allman, V. Paxson, E. Blanton "TCP Congestion Control," Purdue University September 2009, Network Working Group, Request for Comments: 5681, Category: Standards Track.
9. Y. Cheng "Seeding RTO with RTT sampled during three-way handshake," Google. IncInternet Draft, draft-ycheng-tcpm-rtosynrtt-00.txt (work in progress), Intended status: Standard Updates: 3390, 2988, Creation date: June 30, 2010, Expiration date: January 2011.
10. B. Constantine, G. Forget, R. Geib, R. Schrage "Framework for TCP Throughput Testing," Internet Engineering Task Force (IETF), Request for Comments: 6349, ISSN: 2070-1721.
11. B. A. Forouzan "TCP/IP Protocol Suite," Tata McGraw-Hill, 3rd edition.
12. S. Floyd, T. Henderson, A. Gurtov "The New Reno Modification to TCP's Fast Recovery Algorithm," April 2004, Network Working Group, Request for Comments: 3782, Category: Standards Track.
13. V. Paxson, M. Allman, J. Chu, M. Sargent "Computing TCP's Retransmission Timer," June 2011, Internet Engineering Task Force (IETF), Request for Comments: 6298, Category: Standards Track, ISSN: 2070-1721.
14. RFC: 793, "TRANSMISSION CONTROL PROTOCOL," Defense Advanced Research Projects Agency, September 1981, University of Southern California.
15. Postel, J., "User Datagram Protocol," RFC 768, ISI, 28 August 1980.

16. Postel, J., "Internet Protocol," RFC 760, USC/Information Sciences Institute, January 1980.
17. RFC: 791, "INTERNET PROTOCOL," Defense Advanced Research Projects Agency, September 1981, University of Southern California.
18. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson "RTP: A Transport Protocol for Real-Time Applications," January 1996, Network Working Group, Request for Comments: 1889, Category: Standards Track.
19. Shudong Jin, Liang Guo, Ibrahim Matta, Azer Bestavros "A Spectrum of TCP-friendly Window-based Congestion Control Algorithms," Boston University, July 2002, This work was supported in part by NSF grants CAREERANI-0096045, ANI-0095988, ANI-9986397, and ITR ANI-0205294, and bygrants from IBM, Sprint Labs, and Motorola Labs.
20. Network Simulator ns-2. http://www.isi.edu/nsnam/ns/.