Enhancing TCP Congestion Control for Improved Performance in Wireless Networks

BREESON FRANCIS

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE
IN ELECTRICAL AND COMPUTER ENGINEERING

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada

Abstract

Transmission Control Protocol (TCP) designed to deliver seamless and reliable end-to-end data transfer across unreliable networks works impeccably well in wired environment. In fact, TCP carries the around 90% of Internet traffic, so performance of Internet is largely based on the performance of TCP. However, end-to-end throughput in TCP degrades notably when operated in wireless networks. In wireless networks, due to high bit error rate and changing level of congestion, retransmission timeouts for packets lost in transmission is unavoidable. TCP misinterprets these random packet losses, due to the unpredictable nature of wireless environment, and the subsequent packet reordering as congestion and invokes congestion control by triggering fast retransmission and fast recovery, leading to under-utilization of the network resources and affecting TCP performance critically. This thesis reviews existing approaches, details two proposed systems for better handling in networks with random loss and delay. Evaluation of the proposed systems is conducted using OPNET simulator by comparing against standard TCP variants and with varying number of hops.

Dedicated to my father.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Amiya Nayak, and my co-supervisor, Professor Ivan Stojmenovic, who provided the opportunity, encouragement and support to pursue my master's studies. Especially, I want to thank Prof. Amiya Nayak for pushing me hard to work and for all the discussions which lead me to valuable publication experiences. I thank OPNET Inc. for providing research license to carry out simulations for this thesis.

I am indebted to my father who always wanted me to continue with my studies. Of course, I wish to thank my wife, Rupa, my mother and my son for their patience, endless love and constant encouragements. I would take this opportunity to thank my friend, Kobi Sundar Iyakutti, who was always there to help and encourage me. Without them this work would have never come into existence.

Breeson Francis

Ottawa, August 2012

Table of Contents

Table of Figures

Table of Tables

Acronyms

ACK	Acknowledgement
AODV	Ad hoc On-Demand Distance Vector
AOMDV	Ad hoc On-Demand Multipath Distance Vector
AP	Access Point
ARQ	Automatic Repeat Request
BER	Bit Error Rate
CA	Congestion Avoidance
CI	Contention Indicator
CTS	Clear To Send
CW	Congestion Warning
CWND	Congestion Window
DDLRP	Detecting and Differentiating the Loss of Retransmitted Packets
DIFS	Distributed Inter-Frame Space
DSDV	Destination-Sequenced Distance-Vector
DSR	Dynamic Source Routing
DUPACK	Duplicate Acknowledgement
ECN	Explicit Congestion Notification
EED	End-to-End Delay
FRFR	Fast Retransmit and Fast Recovery

FRL	Fast Retransmission Loss
FTP	File Transfer Protocol
GPRS	General packet radio service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
KB	Kilo Byte
L^2CLAMP	Link Layer Curtailing Large Advertised-windows to Maximize Performance
LACK	Local Acknowledgement
LAN	Local Area Network
LBA	Loss Based Acknowledgement
MAC	Medium Access Control
Mbps	Megabits per second
MSS	Maximum Segment Size
OLSR	Optimized Link State Routing
OOO	Out Of Order
PDR	Path Diversified Retransmission
PEP	Performance Enhancing Proxy
PIFS	Point Coordination Function Interframe Space
PSTN	Public Switched Telephone Network
QAOMDV	Qos-aware Ad hoc On-Demand Multipath Distance Vector

QoS	Quality of Service
RL	Retransmission Loss
RTO	Retransmission Time Out
RTS	Request To Send
RTT	Round Trip Time
SIFS	Short Inter-Frame Space
SS	Slow Start
SSTHRESH	Slow Start Threshold
TCP	Transmission Control Protocol
TCP-AP	Transmission Control Protocol-Adaptive Pacing
T-DLRP	Timestamp based - Detection of Loss of fast Retransmitted Packets
WMN	Wireless Mesh Network

Chapter 1 - Introduction

1.1 Background

Wireless networks (Figure 1) are growing as one of the most popular means to connect devices. One of the prime reasons for this growing popularity is the ease of expansion. No more costly, painstaking and labor intensive excavation or cable laying. Ability to form instant networks and cost effectiveness are the key factors that drive us towards wireless networks. Also, it gives an opportunity to create isolated communities with low or minimal resources.

With the advent of wireless routing protocols like DSDV [33], AODV [34, 35] and DSR [8] automatic route calculation in wireless networks has become a reality. While DSDV is a proactive routing protocol, AODV and DSR only react to topology changes if it affects the traffic. DSDV requires periodic control messages to maintain the routes in sync across the network. These protocols, either proactive or reactive, incline to create broadcast storm due to the inherent broadcast nature of its route discovery. Protocols like AOMDV [27] and OLSR [7] were introduced to mitigate this by using on-demand route discovery or multiple relay points.

TCP [36] is the de facto standard transport protocol used for seamless end-to-end connectivity and reliable data delivery over any underlying technology. TCP makes sure of ordered and guaranteed packet delivery. Almost 90% of the internet application traffic like World Wide Web, email, remote administration, file transfer etc. makes use of TCP for data transmission. Since TCP does not worry about the underlying technology, it can be used in wireless networks as well.

However, wireless networks have some traditional problems, such as interference from outside signals, unreliable nature of wireless medium, asymmetric transmission delay of wireless channels, limited bandwidth, hidden and exposed terminal phenomena and blind invocation of congestion control at transport layer.

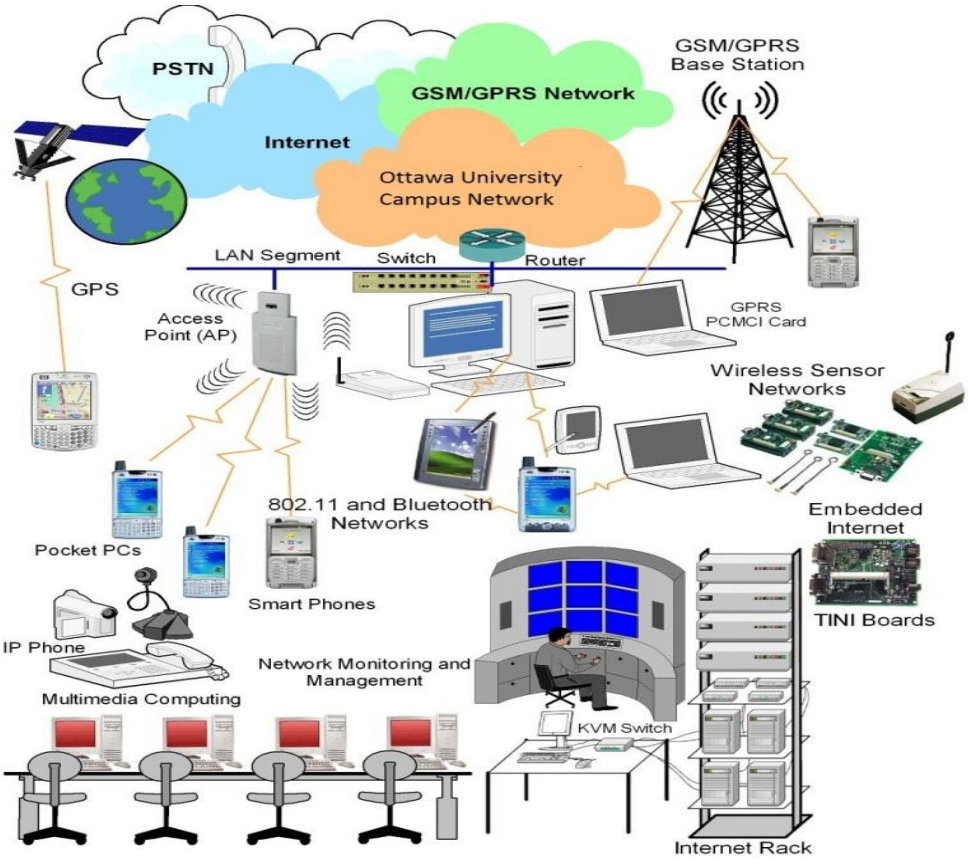

Figure 1: Wireless Network [19]

1.2 Motivation and Objectives

TCP, conceptualized in the early 1980s, is perfectly blended for wired networks providing congestion control mechanisms and retransmission strategy for reliable stream of buffer transfer between end stations. Yet, TCP originally designed for wired network, does not work up to scratch when introduced in wireless networks [17]. Wired TCP assumes packet losses are due to congestion in the path and reacts accordingly. However, as compared to wired networks, in wireless networks a packet loss is not always due to congestion, rather they are caused by the inherent unreliable nature of the wireless transmission medium. Several techniques have been proposed to handle the volatile nature of wireless networks. Congestion window (*cwnd*), which decides the rate of packet transmission, should be handled intelligently while processing packet drops in wireless networks. The usual congestion control mechanism, which reduces the congestion window, will only help in decreasing the TCP throughput rather than mitigating the

congestion. Mechanisms should be used to keep *cwnd* as high as possible, while keeping the congestion under control.

There are several factors that affect packet transmission and path selection in multi hop wireless networks. Some of them are mentioned below:

- *Random loss*: Due to channel fading, high BER, shadowing etc. wireless networks are more prone to transmission losses. Interference from other signals leads to losses due to modification of packet data being transmitted. Link Layer error correction techniques are not adequate to recover the packet loss due to interference and retransmission of the packet is the only way out [48].
- *Larger Round Trip Time (RTTs):* Wireless networks have higher link latencies. This significantly affects the total round trip time and retransmission timeouts become a common place. The subsequent trigger of congestion control mechanisms finally leads to reduced TCP performance. These settings are more common in high delay networks like satellite networks [23].
- *Bandwidth Limitation*: The bandwidth when compared to wired networks is much less in wireless networks leading to buffer overflow at base stations and large round trip time and subsequently lead to lower TCP throughput.
- *Handoffs*: When a mobile device moves from one wireless domain to another one, all the current information has to be transferred to the other domain for seamless connectivity. This again adds up to the delay and reduced TCP throughput [48].
- *Asymmetric Channel Allocation*: Usually the sender gets more channel time/bandwidth compared to the receiver [2]. This results in queuing up of acknowledgements, larger RTTs at sender and traffic becoming bursty in nature.
- *Multipath Routing*: Multiple paths to the same destination lead to significant amount of Out-Of-Order (OOO) packets, which in turn generates duplicate acknowledgements, which then triggers congestion control in TCP [48].
- *Route Instability*: Route instability leads to Out-Of-Order packets, which leads to congestion control trigger in TCP. Figure 2 shows how packet 2 can reach destination before packet 1 and cause OOO packets [48].

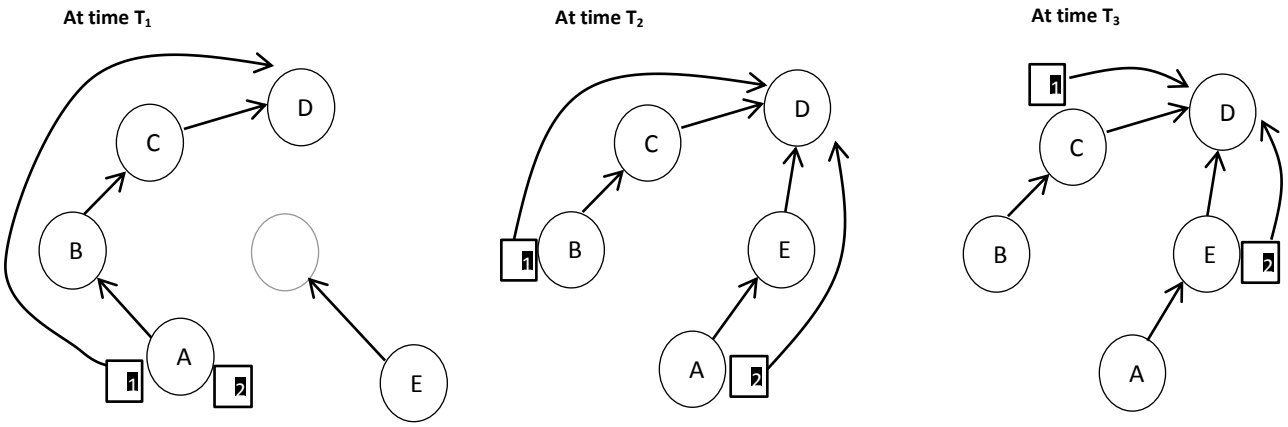

Figure 2: Route Instability

- *Network Partitioning*: Network partitioning happens when, due to node mobility in wireless networks, nodes previously able to communicate are not able to communicate now. TCP backs off its retransmission timer every time a timeout occurs. It is possible that the later nodes fall in place and can communicate but are waiting for the timer to expire before talking to each other, thereby leading to periods of inactivity. Figure 3 shows how network partitioning leads to inactive time periods [48].

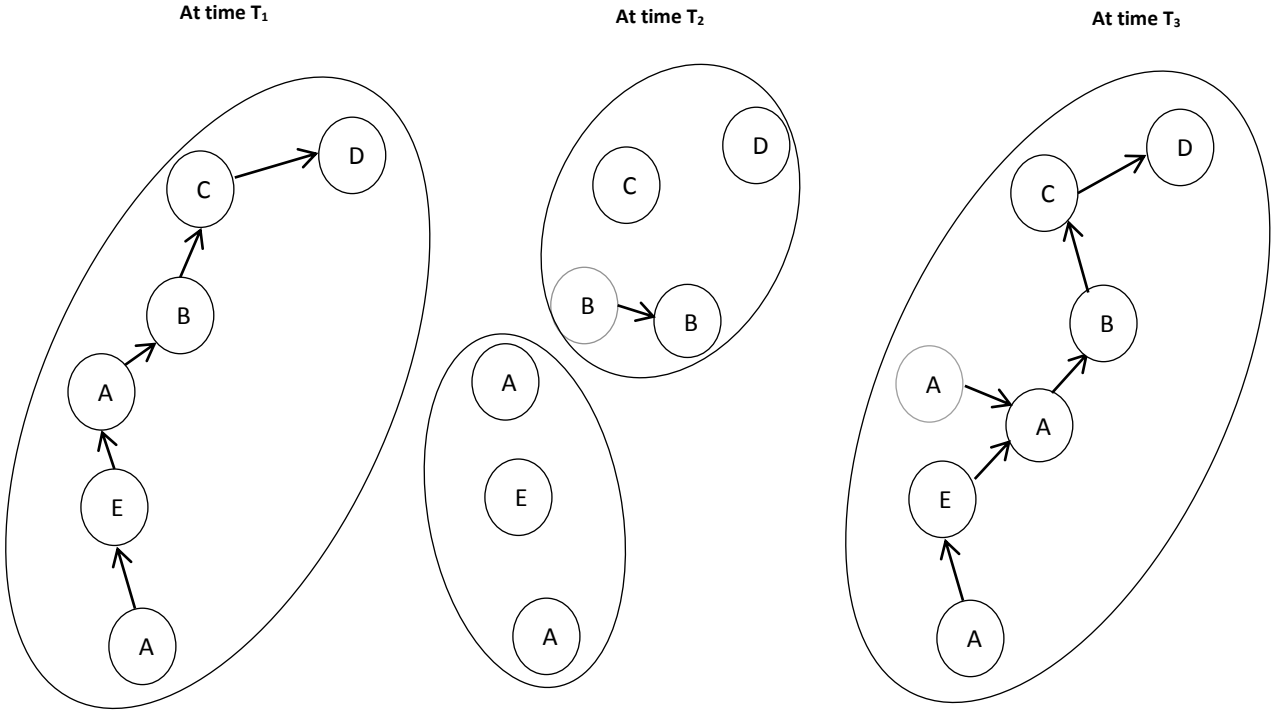

Figure 3: Network Partitioning

The other constraints in wireless networks are memory and processing constraints [24, 28, 41] and QoS constraints [10, 21, 49]. TCP has to be modified to work effectively in wireless networks and treat the problems specific to wireless networks based on their merit rather than blindly triggering congestion control mechanisms. In this thesis we devised TCP sender side mechanisms to handle random losses and retransmission timeouts in high delay networks in such a way as to keep *congestion window* as high as possible, while keeping the congestion under control and keep retransmissions to minimal. The devised mechanisms are assessed against TCP-Reno and TCP New-Reno to see how they fare against more popular TCP variants. OPNET simulator is selected as the simulation tool because of the ease of use of the graphical interface provided and extensive support of TCP. Also the free license availability for research purpose encouraged us to select OPNET simulator.

1.3 Research Contributions

This thesis presents some existing modifications done to TCP for better performance in wireless environments along with some proposed modifications and the same are discussed with simulation results. In the proposed mechanisms foremost importance is given on TCP congestion control. We present approaches towards a TCP sender side modification to improve TCP throughput. The contributions of this thesis in concise are:

i) Due to random loss of packets in wireless networks, TCP is forced to trigger congestion control mechanisms. Modifications are done to TCP, during slow-start and fast retransmit and fast recovery phases, to keep the congestion window at higher levels so as to utilize the available bandwidth efficiently and mitigate the effect caused by random losses. Since congestion window is retained at higher level it provides an enhanced performance in the wireless scenarios and the simulation results show that the modified TCP performance is considerably healthier than standard TCP mechanisms.

ii) TCP when used in networks with considerable delay in transmission, result in frequent retransmission timeouts prompting congestion mechanism to take effect. TCP's retransmission mechanism is altered to take wireless conditions into account

and not to treat all retransmission timeouts in same fashion. Simulation results confirm better performance when compared against TCP variants.

The following two papers [12, 13] have been published from this thesis:

B. Francis, V. Narasimhan, A. Nayak, I. Stojmenovic, “Techniques for Enhancing TCP Performance in Wireless Networks”, in 9th Workshop on Wireless Ad hoc and Sensor Networks (in 32nd International Conference on Distributed Computing Systems), 2012.

B. Francis, V. Narasimhan, A. Nayak, “Enhancing TCP Congestion Control for Improved Performance in Wireless Networks”, in 11th International Conference on Ad Hoc Networks and Wireless, 2012.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

In Chapter 2 the basic operations of TCP and different mechanisms involved for congestion control and flow control are shown. Also, few variants of TCP adaptations used to mitigate the problems which existed in basic TCP are explored.

In Chapter 3 is presented an overview of various existing approaches to improve TCP over wireless networks. Several mechanisms separated based on where the changes are made are detailed in this chapter.

In Chapter 4 a mechanism is proposed to improve TCP performance by modifying existing TCP Reno’s congestion control mechanism.

In Chapter 5 a mechanism to improve TCP bandwidth availability by modifying existing TCP New-Reno’s retransmission mechanism during timeout is proposed.

In Chapter 6 a summary and conclusions are provided along with some suggestions for future research.

Chapter 2 - TCP and Its Operations

Communication networks provide unreliable packet delivery services and packets can be lost or corrupted when intermediate device hardware fails or becomes too heavily loaded to carry all the traffic. Also, networks that dynamically route the packets may deliver them out of order, with considerably delay or even duplicate the packets. At the application level, the programmers may have to build strategies to avoid these problems and make these mechanisms part of each application they develop, which is tedious and repetitive. A general purpose solution would be more suited and TCP is essentially the most fitting one.

TCP forms the backbone for transferring data across the Internet, providing end-to-end congestion control and continuous reliable data delivery services over unreliable communication networks. TCP works as an intermediary layer between the user applications and the Internet Protocol (IP). IP provides data delivery, but does not ensure guaranteed or ordered packet delivery whereas TCP provides guaranteed packet delivery by means of retransmission and ordered packet delivery by using sequence numbering. TCP congestion control mechanism [1] makes sure of end-to-end data delivery without causing congestion en route to destination. TCP makes use of sequence numbering, congestion window and retransmission timer mechanisms to achieve sequential, congestion-less and reliable services. Below are some of the most important features of TCP:

- Connection based protocol
- End-to-end semantics
- Full duplex connectivity
- Detects data duplication
- Provides flow and congestion control

2.1 TCP Connection Establishment and Termination

TCP makes use of three way handshake to setup a connection and the receiver acknowledges every packet sent by the sender. TCP handshaking mechanism is designed in such a way that the communicating parties can agree the parameters, for example the socket, before starting data transfer. TCP 3-way handshake also allows both the participants to open and negotiate separate TCP socket connections at the same time.

Differing from connection establishment, it takes four steps to terminate a TCP connection. This is because of TCP's half close. As TCP is full duplex (discussed later), flow in each direction must be closed independently. Either side can send a FIN packet initiating close of data flow in that direction. A FIN in one direction means that no more data will be flowing in that direction, but in the other direction data flow can still continue. Figure 4 illustrates a basic TCP connection used for HTTP (Internet) traffic.

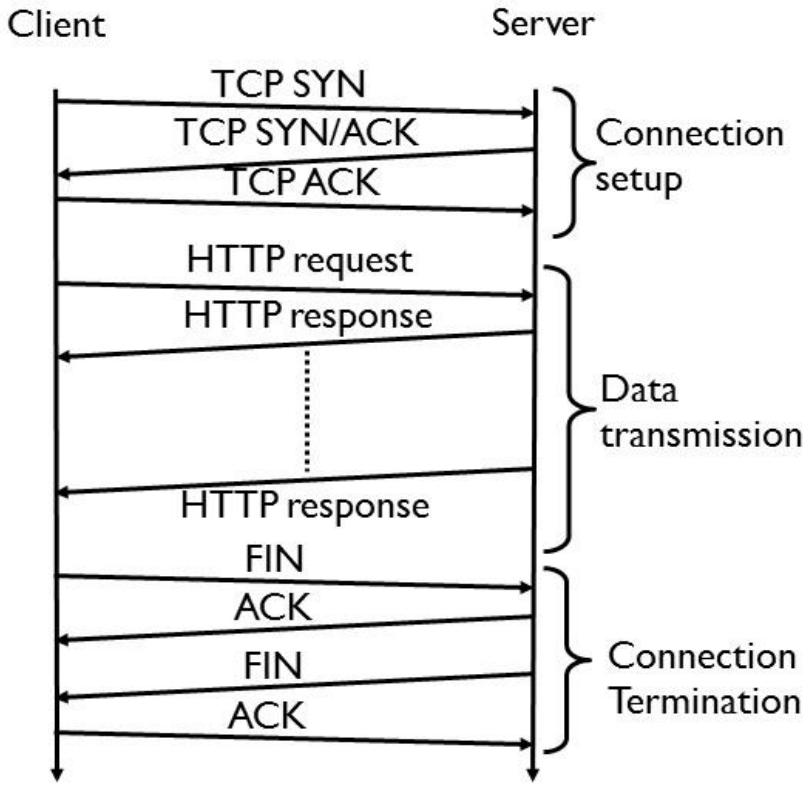

Figure 4: TCP Client/Server Connection

2.2 End-to-End semantics

TCP maintains end-to-end semantics, which means an acknowledgement (ACK) received by the sender confirms that the packet has reached the intended receiver and not any intermediate node. Even though the lower layers provide checks at each transfer, this acts as a double check to ensure that there is no failure at the intermediate machines.

2.3 Full Duplex Connectivity

TCP allows two-way traffic flow i.e. from sender to receiver and receiver to sender concurrently. TCP supports two independent data streams in opposite directions and either stream can be terminated without affecting the other. Also, TCP supports *piggybacking,* where control information for one data stream back to the source is carried as part of data in the opposite direction.

2.4 Data Duplication

TCP detects data duplication by making use of the sequence numbering mechanism. TCP sender assigns sequence number for every packet sent and expects an acknowledgement for the same before proceeding with further data transfer. Every packet sent by the sender is numbered and the receiver when receives packets with the same sequence number more than once distinguishes arrival of a duplicate packet. Also, a TCP receiver expects all the data segments received to be in consecutive order. If detected altered, it sends a duplicate acknowledgement (with the sequence number of the last received in order packet) for each data segment that it received out-of-order. When the TCP sender receives duplicate ACKs beyond a certain threshold, it assumes packet loss and congestion control mechanisms are initiated.

2.5 Flow Control

In standard positive acknowledgement mechanism, the sender sends the packet and waits for the corresponding acknowledgement before sending the next packet all the time sitting idle. This wastes substantial amount of network bandwidth. TCP mitigates this concern by making use of *sliding window protocol*. Sliding window protocol allows the sender to send multiple packets before waiting for an acknowledgement. It allows the sender to send a fixed number of

packets, referred to as *window size*, at any given point in time before receiving an acknowledgment. Figure 5 represents the operation of sliding window protocol.

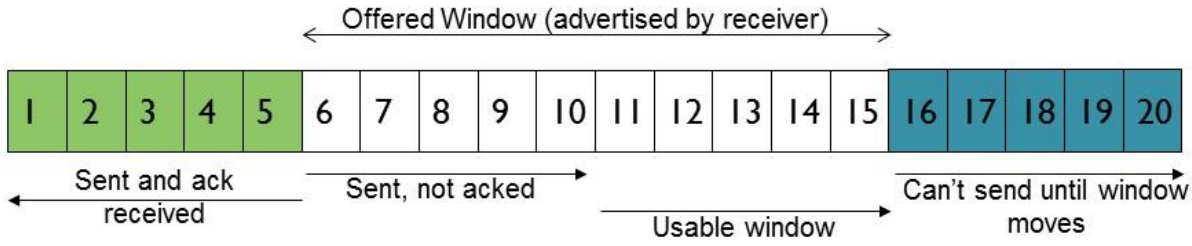

Figure 5: Sliding window protocol

The above figure represents a sliding window mechanism with window size 10. The sender can send up to 10 packets without receiving an acknowledgement. The window slides forward as and when the acknowledgements arrive for the sent packets. Sliding window protocol keeps record of which packets have been acknowledged and maintains separate timers for each unacknowledged packets. If an acknowledgement is not received within a predetermined interval, sliding window considers it as a packet lost and retransmits the packet.

TCP supports variable window size (congestion window) allowing the sender to vary the rate at which it sends packets, essentially realizing *flow control*. Based on the receiver side capability the window size is adjusted to avoid buffer overflow at the receiver, thus the name congestion window. In extreme cases when the receiver is flooded, it advertises a window size of zero intimating the sender to stop sending immediately. Later, when receiver recovers normal state it advertises a non-zero window size to activate data flow again.

2.6 Congestion Control

Congestion is a state of severe delay in data transfer caused by overload of packets at the intermediate nodes. TCP congestion control mechanism makes sure of end-to-end data delivery without causing congestion en route to destination. TCP makes use of sequence numbering, congestion window and retransmission timer mechanisms to achieve sequential, congestion less and reliable services.

TCP employs *slow-start* as one of the mechanism to realize congestion control. During slow-start, TCP starts transmission with congestion window size set to a single segment and waits for the corresponding acknowledgement. When acknowledgment arrives, the congestion window is increased to 2 segments. When acknowledgements for these 2 segments arrive, congestion window is increased to 4. Congestion window is increased exponentially in this manner till it reaches a slow-start threshold, after which it enters *congestion avoidance* phase. During slow-start, if TCP encounters packet loss congestion window is reduced to initial size of 1 segment and slow-start threshold is set to half (or 2 segments size whichever is the minimum) of the current congestion window.

During congestion avoidance, TCP increases the congestion window by (*segment_size * segment_size) / cwnd* where segment_size is the size of the segment acknowledged and *cwnd* is the congestion window size. This is a linear growth of congestion window as compared to the exponential growth of slow-start mechanism. The increase in congestion window is limited to one MSS per round-trip time irrespective of the number of acknowledgements received in that time period. Figure 6 shows the pictorial representation of congestion window growth.

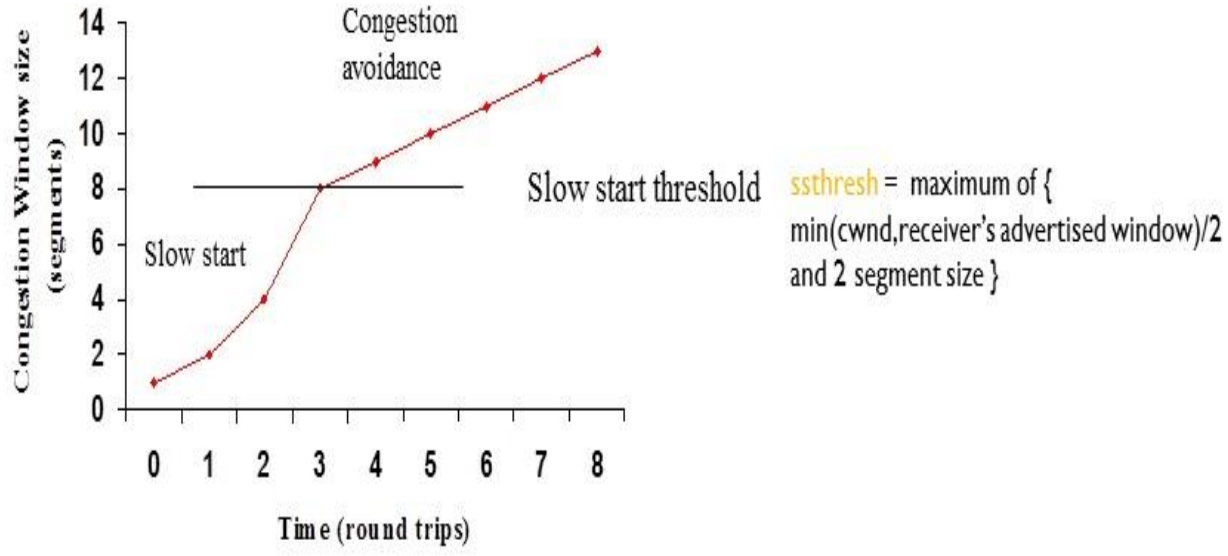

Figure 6: TCP congestion control

When TCP receives a threshold number (usually three) of duplicate acknowledgements, TCP assumes the packet is lost and retransmits the packet without waiting for the retransmission timer to expire. This is called fast retransmission. After fast retransmitting the packet, TCP enters congestion avoidance state rather than slow start state. This is referred to as fast recovery and allows high throughput under mild congestion scenarios. The reason for not performing slow start is due to the fact that a duplicate acknowledgement is generated only when the receiver receives a segment, and which essentially signals that there is still data flowing between the interested parties.

The fast retransmit and fast recovery (FRFR) mechanism are typically implemented together as follows:

1) When the third consecutive duplicate acknowledgement is received, set slow start threshold to half the current congestion window, a minimum of two segments. Retransmit the lost segment and set congestion window to slow start threshold plus 3 times segment size, to take into consideration the segments that left the network and the other side has cached.
2) For each duplicate acknowledgement thereafter, increase the congestion window by segment size and transmit a segment if the new value allows.

2.7 TCP Variants

2.7.1 TCP-Tahoe

Figure 7 shows the working of TCP-Tahoe. In normal non-congested traffic conditions, the congestion window is increased exponentially till slow-start threshold is reached. After the congestion window has reached slow-start threshold, the congestion window is increased linearly until congestion is detected. TCP-Tahoe treats a timeout as indication of congestion. Whenever timeout (non-arrival of acknowledgement before retransmission timeout) occurs, TCP-Tahoe tries to lessen congestion by initiating slow start mechanism by setting the congestion window to one and slow start threshold to half of congestion window. Congestion window is increased additively till slow-start threshold is reached, then increased linearly until a packet loss is en-

countered. One of the major issues with TCP-Tahoe is that it significantly reduces the available bandwidth as the congestion window is reduced to one and has to be rebuilt with every acknowledgement received. Also, it takes considerably long time to detect a packet loss.

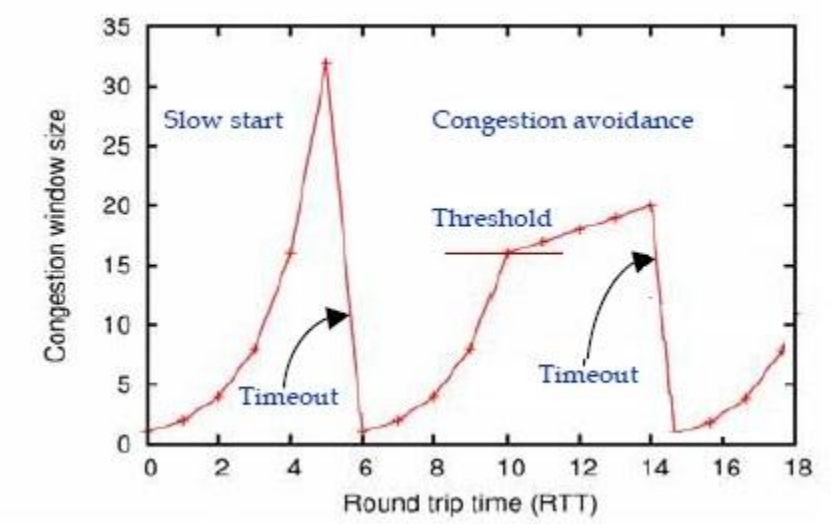

Figure 7: TCP-Tahoe congestion control [52]

2.7.2 TCP-Reno

TCP-Reno tries to solve the issues present in TCP-Tahoe. It uses the same basic principle of TCP-Tahoe but introduces some intelligence to detect packet loss earlier and not to reduce the congestion window drastically. TCP-Reno expects immediate acknowledgements for packets sent. A duplicate packet from the receiver indicates that the next packet in line has reached. If a considerable number of duplicate acknowledgements are received, the sender presumes that the packet has been lost. TCP-Reno uses this logic of *duplicate acknowledgements* (dupacks) to trigger Fast Retransmit. After receiving a predetermined number (usually set to three) dupacks, TCP Reno takes it as a sign of segment lost and retransmits the packet immediately and enters Fast Recovery. In Fast Recovery, slow-start threshold and congestion window is set to half the value of current congestion window. For each subsequent dupack, the congestion window is increased by one and a new segment transmitted if the new value permits it. TCP-Reno remains in fast recovery phase until it receives acknowledgements for all the segments in the transmit window when it entered congestion, after which it enters congestion avoidance. TCP-Reno and

TCP-Tahoe treat a timeout in the same fashion by triggering slow-start. TCP-Reno overcomes the problems of TCP-Tahoe but cannot detect multiple packet loss within the same window and sometimes reduces the congestion window more than once for packet losses occurred within the same transmit window. Figure 8 shows the TCP-Reno congestion control mechanism.

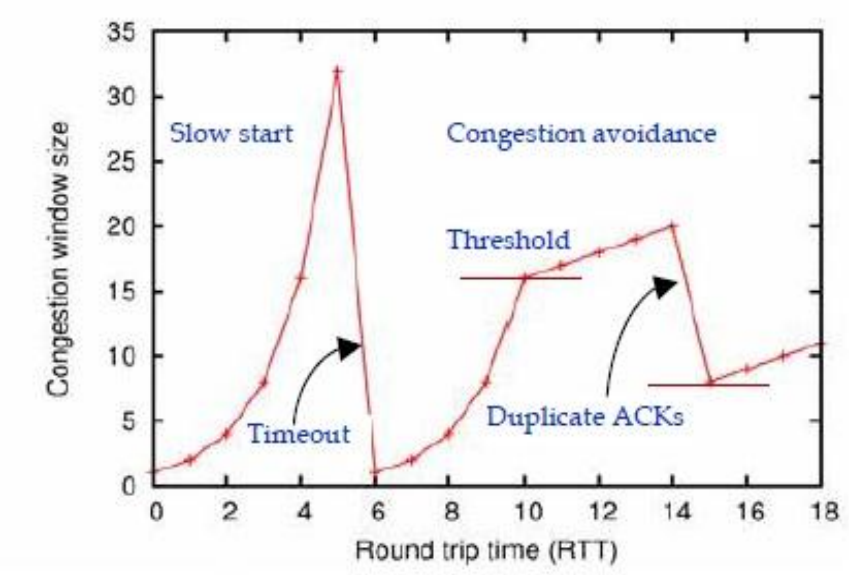

Figure 8: TCP-Reno congestion control [52]

2.7.3 TCP New-Reno

TCP New-Reno [11] tries to overcome the issues present in TCP-Reno by modifying the fast recovery mechanism. Figure 9 shows TCP New-Reno congestion control mechanism. During fast recovery, when a fresh ACK is received TCP New-Reno handles them as below [47]:

- If it ACKs all the segments which were outstanding when TCP New-Reno entered Fast Recovery, then it exits Fast Recovery and sets *cwnd* and *ssthresh* and continues congestion avoidance as in Tahoe.

- If the ACK is partial then it deduces that the next segment in line was lost and retransmits that segment and sets dupacks to 0. It exits Fast Recovery when all the data segments in the window are acknowledged, until then every progress in sequence number generates a redundant packet retransmission which is instantly acknowledged.

TCP New Reno suffers from the fact that it takes one RTT to detect each packet loss. When the ACK for the first retransmitted segment is received, only then can we deduce which other segment was lost.

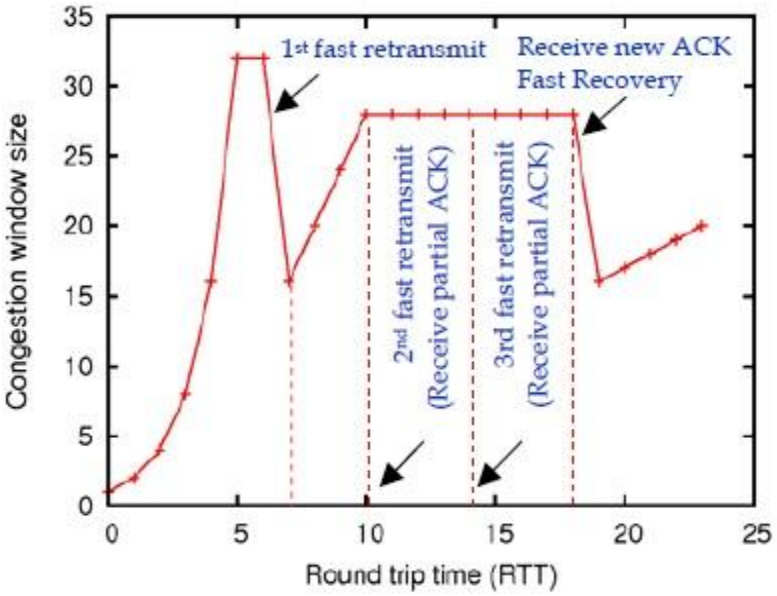

Figure 9: TCP New-Reno congestion control [52]

2.7.4 TCP Vegas

TCP-Vegas, also known as delay-based TCP, unlike other TCP variants like TCP-Tahoe, TCP-Reno and TCP New-Reno, does not wait for packet loss to happen before reducing the sending rate. Instead, it uses delay in packet arrival as congestion indication.

TCP-Vegas uses a refined bandwidth estimation technique to perform congestion control. It calculates the available bandwidth in the network as the difference between actual data flow rate and the expected data flow rate. In situations where there is no congestion, these two rates should be essentially more or less the same. In case of congestion, the actual data flow rate would be less than expected data flow rate. The difference between the data flow rates are calculated using the round trip times as given below:

$$Diff = (Expected \mid Actual)\ BaseRTT$$

where $Expected$ is the expected rate, $Actual$ is the observed rate and $BaseRTT$ is the minimum round trip time.

Based on the calculated $Diff$, TCP-Vegas sender updates the congestion window as follows:

$$cwnd = \begin{cases} cwnd + 1 & if\ Diff < \alpha \\ cwnd - 1 & if\ Diff > \beta \\ cwnd & Otherwise \end{cases}$$

where α and β are predefined thresholds.

Chapter 3 - Literature Review and Related Work

TCP is the most vastly used transport protocol providing end-to-end congestion control and reliable data delivery services. The aim of TCP congestion control is to send packets from source to destination, as quickly as possible, without causing congestion on the intermediate routers. TCP makes use of congestion window, which dictates the number of unacknowledged packets that can be tolerated between the source and the destination, to accomplish this. To achieve reliable services, TCP detects the packet loss and retransmits them. A retransmission timer is started at the onset of each packet to the network, and will timeout if an acknowledgement is not received before the timer expires. If an acknowledgement is not received before the timer expires, TCP retransmits the packets and triggers the congestion control mechanism.

A TCP receiver expects all the data segments received to be in consecutive order. Otherwise, it sends a duplicate acknowledgement for each data segment that it received out-of-order. At the sender side, Fast Retransmit and Fast Recovery are triggered when the number of duplicate ACKs reaches a certain threshold (largely set to three). A retransmission timeout (RTO) or triple duplicate acknowledgements indicates a packet loss or network overload. In this chapter the improvements made on existing TCP to better suit wireless environment are discussed. The changes are predominantly in the way packet losses are differentiated and handled. This review divides the TCP mechanisms into 3 categories based on where the changes are made:

1) Sender based
2) Sender and Receiver based
3) Proxy based

Sender Based: The new changes over the existing TCP are made at the TCP sender side. At the TCP sender side changes are made to the existing congestion control mechanisms (Slow Start, Congestion Avoidance, Fast Retransmission, Fast Recovery) or new mechanisms are implemented which co-exist with the standard TCP.

Sender and Receiver Based: The new changes are made both at the sender side and the receiver side. Receiver side changes are made to respond to the new changes done at the sender side and vice versa.

Proxy Based: The new changes are made at the proxy in such a way that the sender and receiver are oblivious to the changes.

The rest of the chapter is presented according to the classifications made above.

3.1 Sender Based Approaches

3.1.1 Serialized Timer Approach

In wireless networks, the bandwidth is much less compared to wired networks, which lead to considerable delay during packet transmission. There is always a possibility that retransmission timeout occurs before the sender receives ACK for the data packet sent thus triggering congestion control mechanisms.

Lai et al. in [22] propose a novel TCP variant, known as TCP for non-congestive loss (TCP-NCL). TCP-NCL describes a new serialized timer approach in which instead of initiating congestion avoidance mechanism immediately after an RTO, a new timer, congestion decision timer, is started as illustrated in Figure 10.

A Retransmission Decision timer RD_i is started when a packet P_i is sent into the network. If ACK_i is received before RD_i expires, RD_i is reset. Otherwise P_i is retransmitted and congestion response decision timer CD_i is started. If ACK_i is received before CD_i expires, it is considered to be an indication of normal network without congestion and CD_i is reset. Otherwise, on expiration of CD_i congestion control mechanisms are triggered. Therefore, by introducing CD_i TCP sender gets some extra time to determine whether it should go ahead with the congestion control measures or not.

By the introduction of two serialized timers, first one for retransmission and second for congestion response, congestion control is delayed by the expiration time of CD_i and gives us two fold advantages. Firstly, information conveyed after retransmission of packet can be used to decide whether to activate congestion avoidance or not. If a packet is acknowledged imme-

diately, it is either from the initial packet or the retransmission packet. The second case implies that the network is lightly loaded. Secondly, a TCP fast retransmit is better than congestion avoidance.

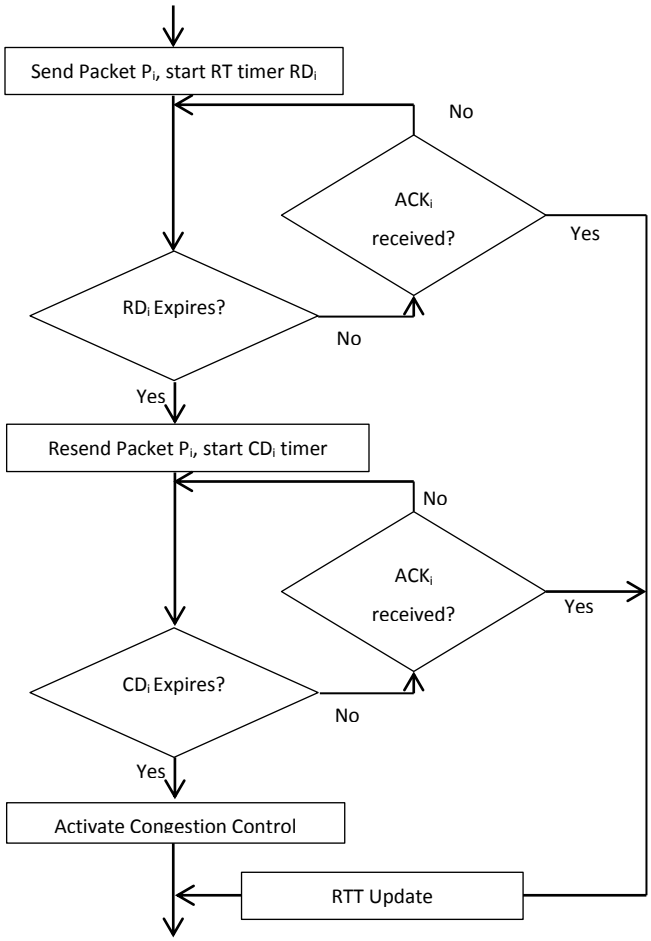

Figure 10: Flowchart for Serialized Timer Model

3.1.2 Modifying Fast Retransmission

During Fast Retransmission, *cwnd* and *ssthresh* are reduced and the lost packet is retransmitted immediately. If the retransmitted packet is also lost, TCP waits for retransmission timeout before sending it again, all the while receiving duplicate ACKs for subsequent packets sent. Fast Retransmission needs to be modified to detect loss of packets retransmitted and send it again immediately.

3.1.2.1 Algorithm to Detect Fast Retransmission Packet Loss

In [37], Prasanthi et al. analyse an efficient algorithm (TCP SAC) in which the slow start threshold (*ssthresh*) is set based on half the difference between the highest data packet sent and the latest acknowledgement packet received. The value of *ssthresh* decides whether to start Slow Start or Congestion Avoidance. Figure 11 shows the state diagram for TCP SAC mechanism.

Suppose sender sends packets P_1, P_2 ….P_{10} and P_3 got lost in transmission. Sender receives *tcpxmthresh*, 3 in this case, duplicate acknowledgements (*t_dupacks*) and detects that a packet is lost. TCP enters Fast Retransmission and Fast Recovery state by reducing the *cwnd* and *sshthresh*. Assuming the retransmitted packet is lost, sender receives additional duplicate acknowledgments (*add_dupacks*), and may send new packets allowed by the *cwnd*. In TCP Reno, without receiving P_3, receiver does not send cumulative acknowledgement and thereby sender goes to Slow Start by reducing the *cwnd* and *ssthresh* values, leading to poor performance.

In TCP SAC when TCP enters Fast Retransmit, it stores the number of outstanding packets. In this example, 10 packets are sent and P_2 has been acknowledged. Therefore, the number of outstanding packets (*t_nps*) is eight. Since the sender received 3 *dupacks*, the receiver has received 3 more packets that the sender had sent. TCP SAC calculates the retransmission loss point (*Rlp*) as 5. Out of the five packets one packet is lost, so total outstanding is 4 packets. If the *add_dupacks* is equal to or more than 4, resend the lost packet immediately without waiting for the retransmission timeout. When sender receives acknowledgement for all the 10 packets, *cwnd* is set to *ssthresh* and TCP SAC exits Fast Recovery. Thus by reducing the transmission timeouts TCP SAC increases the throughput performance.

3.1.2.2 Detecting and Differentiating the Loss of Retransmission

The Authors in [38] propose a mechanism called DDLRP (Detecting and Differentiating the Loss of Retransmitted Packets) which detects and differentiates the loss of retransmitted packets and reacts by retransmitting the packet without waiting for the retransmission timeout. DDLRP consists of two schemes, namely, Retransmission Loss Detection and Retransmission Loss Differentiation.

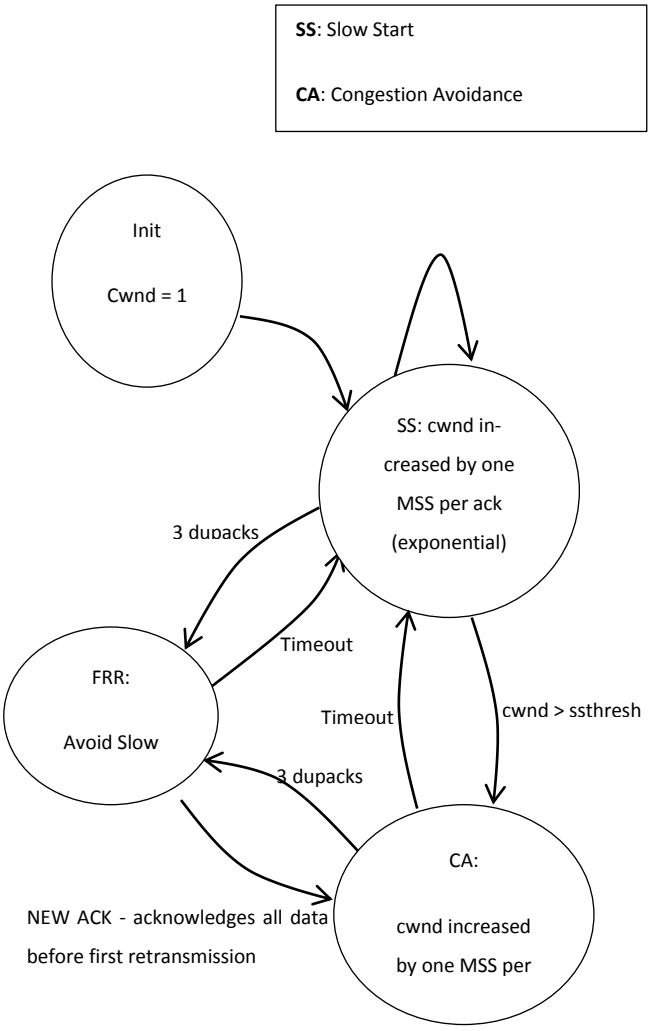

Figure 11: State diagram for TCP SAC

3.1.2.2.1 Retransmission Loss (RL) Detection

RL Detection is a modified version of TCP NewReno. TCP NewReno, which is the most used TCP version, during fast retransmission phase, stores the highest sequence number transmitted in a variable '*Recover*', retransmits the lost packet and sets the *cwnd* value to *ssthresh* plus number of dupacks required to detect loss times *mss*. Then, during fast recovery, for every additional dupack received *cwnd* is increased by one *mss* and a new packet is transmitted if the new *cwnd* value allows for it. When the sender receives an ACK for *Recover,* TCP sets *cwnd* to *ssthresh* and enters congestion avoidance phase. If the received ACK is not for *Recover,* then TCP retransmits the first unacknowledged packet, adds one *mss* to *cwnd* and sends a packet if new *cwnd* value permits it. If the retransmitted packet is again lost, TCP New Reno does not

have a mechanism to find it. RL Detection suggests a mechanism precisely to mitigate this problem.

RL Detection reduces retransmission timeouts when the retransmitted packets are lost. Let *Max_Seq* be the highest sequence number of packets sent by the sender and *Dupack_Lost* be the duplicate ack of the packet that was lost. Then the outstanding number of packets in the network which were not acknowledged can be calculated as below:

Net_Outstand = Max_Seq – Dupack_Lost

The sender retransmits the packet only after receiving *three* (which is the number of duplicate acknowledgements to be received to trigger fast retransmission) duplicate acks. Therefore there are three more packets that have left the network. Loss detection point (*Loss_Detect_Point*) can be calculated as given below:

Loss_Detect_Point = Net_Outstand – Three_Dupacks

During Fast Recovery phase for every additional duplicate acknowledgement (*Add_Dupacks*) cwnd is increased by one mss and packet send if the new value of cwnd allows it. During fast recovery phase, if *Add_Dupacks* is greater than *Loss_Detect_point*, then that indicates that a new packet sent has reached the destination, whereas the retransmitted packet is lost. The TCP sender can retransmit the lost packet without waiting for the retransmission timeout.

3.1.2.2.2 Retransmission Loss (RL) Differentiation

RL Differentiation uses the congestion warning mechanism of TCP New Jersey to differentiate packet loss from congestion loss. Congestion Warning (CW), which was originated from Explicit Congestion Notification (ECN), is a mechanism by which the intermediate routers inform the end systems that the average queue length of one of the intermediate routers have exceeded the threshold level.

In RL Differentiation, when TCP detects a retransmission loss or packet loss through duplicate acknowledgement mechanism, it checks the CW bit. If CW bit is set (CW==1), then TCP sender differentiates it as packet loss due to congestion. If CW is not set (CW==0), then TCP sender differentiates it as packet loss due to transmission error.

In DDLRP, when the sender receives three *dupacks,* it retransmits the lost packet, calculates the Loss_Detection_Point (RL Detection) to detect retransmission loss. TCP sender checks the ACK packets for CW bit (RL differentiation), if it is set then the sender confirms that the packet loss is due to network congestion and reduces cwnd and enters fast recovery phase. If CW is not set, sender confirms that the packet loss is due to transmission loss and retransmits the lost packet without reducing cwnd and enters fast recovery phase.

3.1.3 Modifying Congestion Control Mechanism to Improve TCP Performance

The bandwidth-delay product, which is derived from the rate of acknowledgements received and Round Trip Time, gives a clear picture of the network load [46]. Any congestion control mechanism should take into consideration the bandwidth-delay product observed just before packet loss to decide on how it should proceed on alleviating the congestion. Yongmei et al. in [50] suggest two mechanisms to handle wireless losses modifying the congestion control mechanism of TCP New Reno.

Mechanism 1: This requires modification of the congestion control mechanism at the sender side only. TCP sender continuously computes the BandWidth Estimate (BWE) by monitoring the network for rate of acknowledgements. RTT required to support BWE window is also computed. Based on the packet loss indication *ssthresh* and *cwnd* are set as follows:

- If packet loss is detected by three duplicate ACKs
  then *ssthresh* = (BWE * RTTmin) / segment size

  cwnd = ssthresh if (cwnd > ssthresh)

- If packet loss is detected by timeout then
  cwnd = 1

  ssthresh = max{(BWE*RTTmin) / segment size, 2}

Mechanism 2: This uses the Non-Congestion Packet Loss Detection algorithm to differentiate packet loss due to congestion from packet loss due to link noise. When a packet loss is detected, TCP sender compares the current round trip delay with the calculated delay threshold. If the current value is less than the calculated threshold value, TCP sender assumes that it is a non-

congestion loss and retransmits the packet using fast retransmission, without invoking the congestion avoidance phase. Else, TCP sender considers it as congestion and calculates the new delay threshold and triggers fast retransmission and congestion avoidance.

If the non-congestion packet loss due to wireless link error and data corruption occurs frequently, TCP sender reduces the segment size to reduce the retransmission overhead and corruption probability. Whereas, if the packets are rarely lost due to non-congestion, TCP sender increases the packet size so as to reduce the packet header overhead.

3.1.4 Cross Layer Approach to Improve TCP Performance

Due to the inherent limitations of wireless networks, wireless nodes compete with each other to take control of the wireless channel for packet transmission. These contentions lead to packet drops at the link layer which TCP perceives as congestion, resulting in initiating congestion control mechanisms.

In [15], Hamadani et al. propose a cross layer algorithm, TCP Contention Control, which monitors the effect on achieved throughput by the change in outstanding number of packets in the network and the level of contention experienced by each packet. TCP Contention Control then estimates the amount of traffic that can be sent by the sender to achieve maximum throughput with minimum contention delay. TCP contention control approach introduces a new variable TCP_Contention to achieve this, whose value is changed according to the contention delay observed in the network during various stages as mentioned below.

Fast Probe: TCP enters Fast Probe stage just after the TCP connection is established. TCP_Contention value is increased exponentially during the Fast Probe stage similar to slow start algorithm used to probe the available bandwidth.

Slow Probe: When the throughput and contention delay is reduced compared to the last round trip, TCP Contention Control gradually increases the amount of new data at the rate of one MSS to TCP_Contention for every Round Trip Time (RTT) to use the network efficiently (additive increase).

Light Contention: When the throughput and packet contention delay increases after injecting new data, TCP Contention Control enters Light Contention Stage. During this phase, TCP Contention Control additively decreases TCP_Contention (one MSS per RTT) to avoid reduction in TCP throughput.

Severe Contention: Whenever TCP Contention Control sees that there is an increase in contention delay and a decrease in throughput, which is due to network overload, it enters Severe Contention phase. TCP Contention Control sets TCP_Contention to 2 times *mss,* to force the sender to minimize its transmission rate.

Since TCP Contention Control uses the existing TCP to throttle the network data, it is compatible with all TCP versions without changing TCP congestion control algorithm.

3.1.5 Improving TCP Throughput Stability

In wireless networks, majority of packet drops can be attributed to exposed node problem. When a node, say node 1, sends an RTS to the next node, say node 2, and if the node 2 is in the interference range of other nodes, it simply ignores the RTS. After waiting for 7 consecutive retries, if node 1 does not get a reply, it considers node 2 to be down or has moved out of range and updates the higher layer of link failure.

Omar et al. in [30] introduce a mechanism to increase TCP throughput stability by reducing the effect of exposed node problem. MAC layer reports link failure after it fails to send a Request-to-Send (RTS) packet 7 times. This triggers DSR to calculate alternate route, for which TCP reacts by entering slow start phase leading to zero or near zero TCP throughput. DSR routing algorithm is modified so as to avoid frequent route error processing and alternate route calculation triggered by alarms raised by MAC layer. DSR routing algorithm along with the feedback from TCP mechanism is made to suppress the first link failure alarm and to respond only to the second alarm raised by the MAC Layer.

The algorithm explained above is shown in the form of a flowchart in Figure 12.

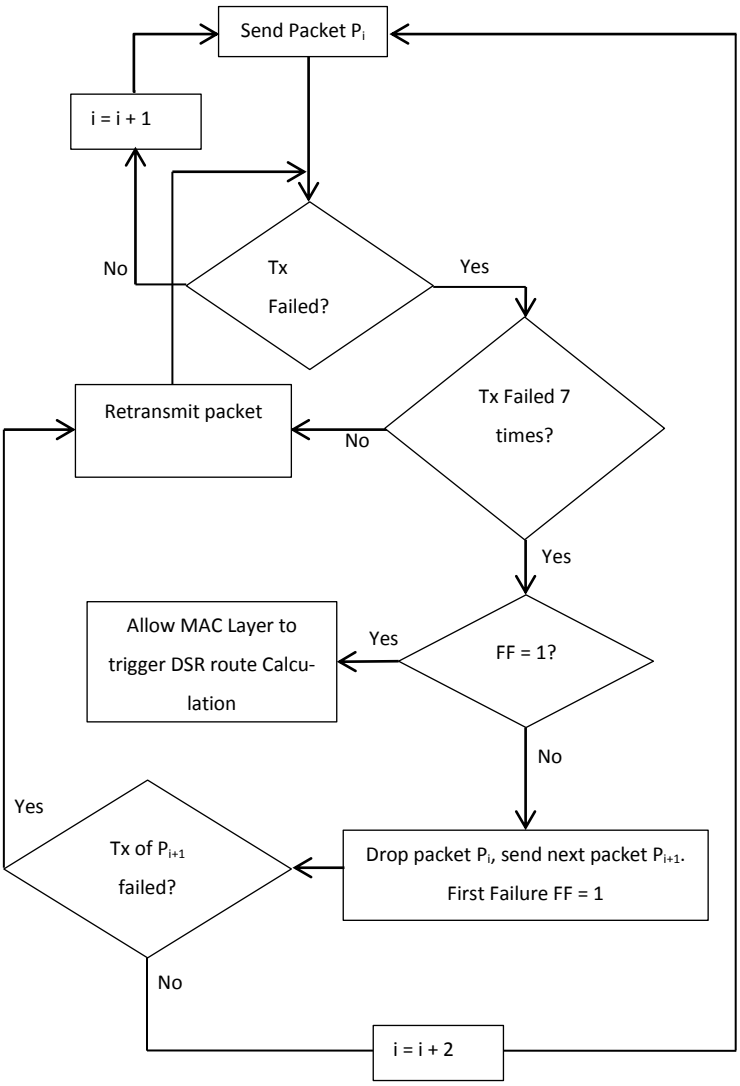

Figure 12: Flowchart for MAC Layer

3.2 Sender and Receiver Based TCP Approaches

3.2.1 Modifying TCP Congestion Control Mechanism

In asymmetric wireless networks, data and acknowledgement packets may take different paths. This leads to different delays, sending rate and packet drop in forward and reverse directions. Since TCP relies totally on acknowledgments received to send new data, a delay in reverse path leads to poor TCP performance. Also, if an ACK is not received within retransmission timeout, congestion control mechanisms will be triggered which in turn reduces TCP throughput considerably. Authors in [6, 43] have devised mechanisms that treat ACKs differentially.

The authors in [39] propose a TCP congestion control protocol (TCP-WAM) which focuses on the ACK losses in asymmetric wireless networks. TCP-WAM is derived by modifying the sending and receiving side of the classic TCP protocol. TCP sender side is modified to reduce the dependency on the reverse channel characteristics and the receiving side is modified to send notification information to the sender indicating whether the previous data packets for which the sender has not received ACK were received by the receiver or not.

TCP-WAM Receiver side modification: TCP WAM receiver first checks whether the packet is marked or not. Marked packets means the sender did not receive ACK for the previously sent data packets and wants to know whether they were delivered or not. Receiver checks from its buffer whether the packets were received, it then replies back by marking the new ACKs with 1 implying that they were received and 0 implying that they were not received. Therefore, the sender knows that the timeout is because of data loss or ACK loss and take appropriate action. TCP-WAM is similar to the classical TCP except that in case of timeout it does not perform the default action of reducing the congestion window.

TCP-WAM sender, unlike classic TCP, does not go to slow start phase on retransmission timeout. Instead, it continues with new data packets as if it received ACKs for the previous packets. However, these data packets are marked in their headers indicating the receiver that it did not receive ACKs for the previously sent data packets. TCP-WAM supposes that all the timeouts are due to ACK packet losses. This action by TCP-WAM takes into account the network asymmetry in wireless networks where the forward and reverse path quality can change intermittently. Therefore it may be possible that the reverse path conditions have improved or changed to other path by the second timeout.

In addition to retaining the congestion window size in case of timeouts, TCP-WAM sender reduces the RTO. This is unlike the classical TCP which backs off at every timeout by doubling the retransmission timeout. TCP-WAM reduces RTO to $\tau = \min \{\alpha * \text{RTO}, 1\}$ seconds where $0<\alpha< 1$.

On receiving marked data packets the receiver responds with marking the ACKs either with 0 or 1 correspondingly. When the received ACK is marked 1, it notifies the TCP sender that the previous data packets were received and continues normal congestion window growth like

classical TCP. Otherwise, the receiver sends duplicate ACKs suggesting network congestion and possible data loss. When no corresponding ACK is received after τ seconds, TCP-WAM assumes network congestion and slow start phase starts.

3.2.2 Reduced Control Traffic for Improvements in TCP Performance

The hidden/exposed problems in wireless networks affect the TCP performance more than anything else. To solve this, 802.11 implements RTS/CTS mechanism where every time a node wants to send packet, it sends an RTS and waits for a CTS. This signaling mechanism increases the control traffic considerably [16].

In [4], Chen et al. analyze the TCP performance in multi hop wireless networks based on the injected traffic and the control traffic. Injected traffic is mostly dependent on the optimal congestion window size which is proposed to be between n/3 and n/2, where n is the total number of hops. Setting a fixed *cwnd* reduces the applicability of TCP. Therefore the literature focuses more on reducing the control traffic to increase TCP throughput.

Due to the underlying MAC layer, the transmission overhead time for longer data packets and ACK packets are almost of the same order of magnitude. Figure 13 explains the basic 802.11 data transfer in MAC Layer. Assuming a basic rate of 1 Mbps (used for control traffic) and a data rate of 11 Mbps, the overhead transmission rate is $3SIFS+RTS+CTS+ACK_{mac}$ = 414μs. Based on the above calculation time to send a 40 byte ACK_{tcp} and 1040 byte data packets are 443 μs and 1170 μs respectively.

To reduce ACKs, the strategy of using large ACK delay timeout does not work well always. If the congestion window is small, the receiver waits for ACK delay timeout to send the ACK, which leads to underutilization of the network.

The literature proposes TCP sender to put the current value of *cwnd* in the option field of TCP header, thereby informing the receiver about its congestion window size. When the receiver calculates that the received-but-unacknowledged packets equal *cwnd,* it knows that the sender is waiting for an ACK to proceed with sending the remaining packets and sends an ACK immediately. These self-triggered ACKs lessen the need to send ACK for every data segment

and thereby reduce the control traffic, which in turn leads to increased TCP throughput performance.

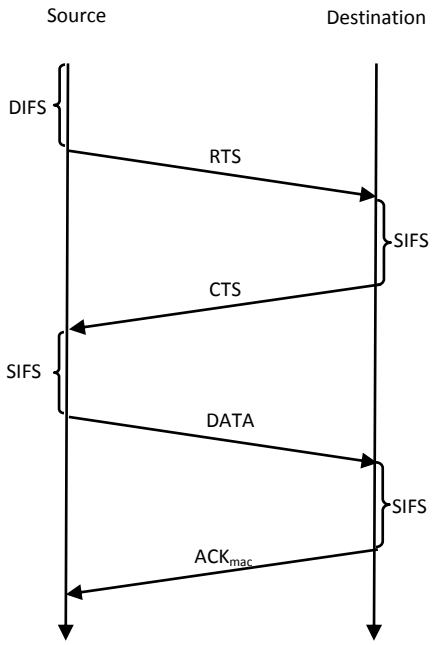

Figure 13: 802.11 MAC Data transfer

3.2.3 Loss Based Acknowledgement Mechanism

In standard TCP, for every packet received TCP receiver replies back with an acknowledgement. This creates plentiful of ACK packets which lead to increased network congestion, takes up a large chunk of the sender's processing time to process it, and reduces the network throughput in half duplex mediums like Ethernet and wireless networks.

The effect of ACK packets on network performance is analyzed by Long et al. in [26]. A novel mechanism of Loss Based ACK (TCP-LBA) is introduced to mitigate the above mentioned issues. In TCP-LBA, the TCP Receiver waits for a period of RTO and then rearranges all the packets received. If any packet is lost during transmission, TCP receiver replies with ACK packets for the packets lost with corresponding number mentioned in the packet. TCP sender responds by retransmitting those packets alone. If there are no packets lost during transmission, TCP receiver sends an ACK packet claiming it has received all the packets.

To achieve TCP-LBA mechanism a few changes are performed on TCP header. Firstly, Sequence number field in the TCP header is changed to total number of pieces. Secondly, the Acknowledgement number field in the TCP header is changed to the number of the current packet. Thirdly, out of 6 bit reserved field in TCP header 2 bits are made use of. Of the four values that can be generated using the 2 bits packet type, two are made use of and the remaining two are reserved. When TCP sender is sending packet it fills the field packet type with value 0x00 indicating that it is sending the packet. When the receiver finds out that there are some packets missing it replies back an ACK packet with packet type set to value 0x01. TCP sender waits till TCP receiver sends an acknowledgement for receiving all the packets.

Thus by reducing the number of ACK packets sent, this mechanism improves the TCP throughput performance in multi hop wireless networks with high packet delay.

3.2.4 Receiver-aided Scheme for Improving TCP Performance

In conventional TCP, sender derives the network condition from the acknowledgement arrivals and packet's Round Trip Time. However, in high bit error rate wireless networks its challenging for the TCP sender to distinguish packet loss due to channel error from congestion losses. TCP receiver on the other hand will be in a better position to determine the real cause of packet loss and should inform the source [3].

In [5], the authors propose a receiver-aided mechanism in which the TCP receiver monitors the contention state of the connection and accordingly informs the TCP sender about it via ACK mechanism. TCP receiver uses end-to-end delay as contention criteria. Let S_i is the sending time and R_i is the receiving time then end-to-end delay (EED), which is $R_i - S_i$, over N hops can be defined as follows:

$$EED = \sum_{h=1}^{N} (T_q + T_{DIFS} + CW_h * T_{slot} + T_{RTS} + T_{CTS} + T_{DATA} + T_{ack} + 3T_{SIFS})$$

where T_q refers to queuing delay, T_{RTS}, T_{CTS}, T_{DATA} are transmission delay of request-to-send, clear-to-send and data packets respectively. CW_h is the contention window and h is the number of hops.

Contention Indicator is defined as:

$$CI_{new} = \frac{7}{8} CI_{old} + \frac{1}{8}(EED_{i+1} - EED_i)$$

With the above indicators, receiver analyzes the network congestion situation and defines slight, middle or heavy congestion. When a packet loss is detected, receiver with the above indicators decides whether the packet loss has occurred due to wireless loss or congestion loss. If a wireless channel error is identified then receiver does not take any action and the sender continues with the current *cwnd.* If the receiver recognizes it as congestion loss or dubious loss, the same is informed to the sender through ACK packet by setting the corresponding option. On ACK arrival, sender checks for the congestion state. If there is no congestion then sender goes on with the normal procedure of increasing the *cwnd* with acknowledged data and continues transmission. If the sender detects congestion, then *cwnd* is reduced. *cwnd* for the above three cases are defined as

Wireless Loss: *cwnd* = *cwnd* + *packet_received*

Dubious Loss: *cwnd* = 0.8(*cwnd + packet_received*)

Congestion Loss: *cwnd* = 0.5(*cwnd + packet_received*)

The TCP Receiver using contention indicator identifies the network situation and informs the sender accordingly. With the aid of receiver TCP sender takes appropriate congestion control to increase TCP throughput.

3.2.5 Cross Layer approach for Improving TCP Performance

In conventional TCP, for every packet received by the TCP receiver it responds with a corresponding TCP ACK. The same mechanism is followed at the MAC level as well i.e. MAC ACK for every packet received. In [42], authors have come up with a novel idea where TCP acknowledgements at the receiver is silently suppressed and regenerated at the sender.

The proposed mechanism modifies the protocol stacks at the sender and receiver with Link Layer CLAMP (L^2CLAMP) agent running between the data link layer and the higher layers. These agents support TCP ACK generation and suppression at TCP sender and receiver respectively.

Figure 14 explains the L^{2}CLAMP mechanism. The basic idea of this approach is to shift the ACK generation from receiver side to sender side and thereby avoiding the need to send TCP ACK over wireless transmission medium.

At the sender side, L^{2}CLAMP agent sniffs the outgoing traffic and whenever a TCP data packet is detected, it stores all the relevant data (sequence number, port number, flags) required for the generation of the TCP ACK. The agent maintains separate queues for each receiver and all the flows corresponding to the same receiver are put in the same queue. When the sender's data link module detects a MAC ACK, it intimates the L^{2}CLAMP module, which in turn regenerates the TCP ACK packet from the information already stored in the buffers.

At the receiver side, L^{2}CLAMP agent sniffs the outgoing traffic and whenever a standalone non-duplicate TCP acknowledgement packet is detected, it silently drops it.

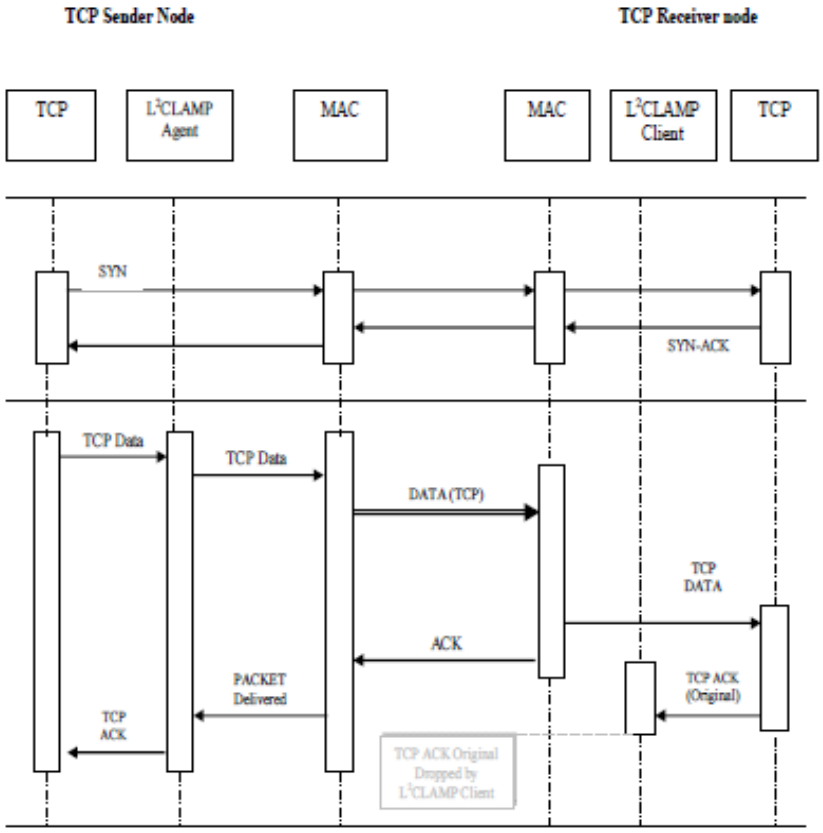

Figure 14: L^{2}CLAMP mechanism [48]

3.3 Proxy Based TCP Approaches

Proxy Based approach, also known as split connection, makes use of the intermediate intelligent devices to mitigate issues caused due to congestion or loss of packets. Proxies end the TCP connection from source or previous proxy and start a new connection towards the destination. In this way, TCP can have independent flow/error control, packet size, retransmission timeouts etc. for individual connections between proxies separately. Any lost packet can be recovered from the source or most recent proxy on the path. It increases parallelism and leads to reduced bandwidth usage on retransmission. In heterogeneous networks, where wired and wireless networks are connected, optimized TCP for wireless networks can be introduced in the wireless part. Figure 15 shows how proxies work in wireless network scenario.

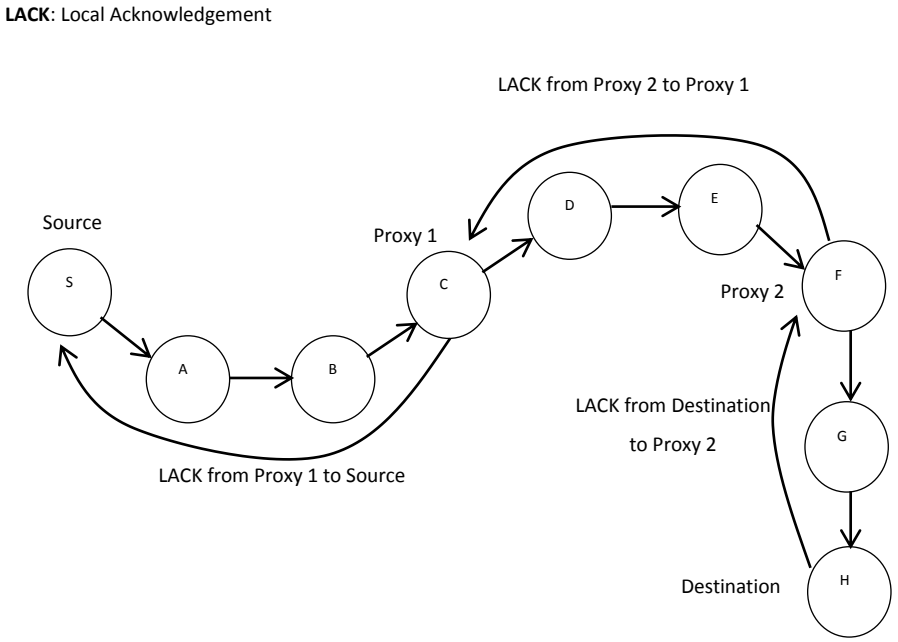

Figure 15: TCP proxy

However, introducing proxies results in loss of end-to-end semantics of TCP and increased delay due to buffering [28]. Buffers at intermediate proxies are prone to get exhausted if the link succeeding it is a slower link. During Handoffs, buffers need to be transferred leading to increased handoff latency. If a proxy fails, the data buffered at the proxy is lost.

3.3.1 Mitigation of Bursty Packets by TCP proxy

In heterogeneous networks, where wired and wireless networks are connected through a proxy, the packets are buffered at the proxy. If a packet is lost, the proxy buffers the subsequent packets till the lost packet is retransmitted. The proxy sends a burst of buffered packets on arrival of the lost packet.

Shikama in [44] proposes to employ a TCP proxy called PEP (Performance Enhancing Proxy) that acts as a traffic controller between the TCP Sender and the Receiver. PEP which is placed in a node that connects the wireless section to the wired section splits the connection and thereby reduces the large round-trip delay.

Packet losses in wireless segment triggers fast retransmit at the TCP sender after receiving 3 duplicate acknowledgements. The receiving side of the PEP accepts out-of-order TCP segments and stores it in a buffer. When the lost segment is retransmitted PEP reassembles the packets and sends it towards the TCP receiver. Since the reassembled data (forward data) can be up to the window size, a large number of segments can be generated at the same time.

PEP which runs as an application program in the intermediate node performs the pacing at the input of TCP. PEP segments the large forward data into chunks of MSS and forwards it over the wired section at a specific rate. Employing PEP has the following advantages:

- Pacing can be done on independent TCP connections
- Pacing can be done in cooperation with the flow control between two TCP connections. PEP stops receiving further data from the wireless section until it sends all the chunks of the forward data to the wired section.

A problem though when introducing PEP pacing is to determine the rate at which the smaller chunks have to be sent to the wired section.

3.3.2 Proxy-based TCP with Adaptive Rate Control and Intentional Flow Control

In TCP proxy based wireless networks, sub sessions on either side of the proxy compete for the same wireless channel. Itoh et al. in [20] introduce two methods for resolving the wireless

channel capture problem. The proposed methods are adaptive rate control in the second sub-session and intentional flow control in the first sub-session.

Adaptive Rate Control: It makes sure that the second sub-session does not continuously keep sending segments from the proxy host by introducing an inter-departure time at the proxy host. Transmission rate which should depend on the arrival rate from the first sub-session is kept a little more than the arrival rate so that the transmission layer buffer does not overflow. Inter-departure time of segment from proxy host is defined as follows:

$$S_n = \beta\{(1-\alpha) \, . \, S_{n-1} + \alpha \, . \, R_n\}$$

where R_n is inter-arrival time between two consecutive segments at the proxy host and S_n is allowable inter-departure departure time of segment sent from the proxy host, which is updated with every arrived segment. The parameters α and β are given values 0.125 and 0.9 respectively. Since β is 0.9 the transmission rate from the proxy host is slightly larger than the arrival rate at the proxy.

With Adaptive Rate Control mechanism, inter-departure time is controlled, which in turn prevents bulk transmission of frames in second sub-session. Therefore, wireless channel capture in second sub-session is lessened leading way to increased TCP throughput performance.

Intentional Flow Control: In this method the proxy host intentionally advertises a small window size to the source, a value of 2 times *mss*. With an advertised window of 2, the source can at most send two consecutive frames. This prevents the source from sending bulk transmission and resolves the wireless capture problem in the first sub-session.

Chapter 4 - Modifying TCP Congestion Control to Handle Random Loss of Packets

4.1 Introduction

TCP uses congestion window to perform congestion control, which keeps track of the number of packets that can be sent by the sender without being acknowledged by the receiving side. Basically, congestion window decides whether TCP sender is allowed to send packets at any particular instance. TCP accomplishes reliable data delivery by deploying retransmission timer mechanism which detects packet loss and retransmits them. If an acknowledgement is not received before the expiry of retransmission timer, TCP retransmits the packet and triggers congestion control.

In networks with interference from external sources, every random packet loss is treated by TCP as congestion. Congestion control mechanisms ensue which shrink the congestion window and as a result the sending rate gets reduced drastically. It is imperative that care be taken not to reduce the congestion window for every random loss of packet. TCP congestion control mechanisms have to be attuned to keep the congestion window as high as possible even in the event of a packet loss. This chapter presents methods to treat random packet loss and the resultant retransmission timer expiry or duplicate ACK arrival keeping the congestion window as high as possible or near to the existing value.

4.2 Related Work

Various schemes [28, 31, 34, 35] have been proposed in wireless networks to improve TCP throughput and to handle congestion indication in such a way that TCP throughput is retained high. Lai et al. in [22] proposed an innovative TCP variant, known as TCP for non-congestive loss (TCP-NCL). TCP-NCL describes a new serialized timer approach in which a new timer, congestion decision timer, is started instead of initiating congestion control mechanisms immediately after a retransmission timeout. If the corresponding acknowledgement is received before congestion

decision timer expires, TCP sender continues with the normal data transfer without invoking congestion control mechanisms.

Yongmei et al. in [50] suggest mechanisms to differentiate packet losses due to congestion from wireless losses. In the case of wireless losses, the authors suggest to modify congestion window based on bandwidth-delay product, which indicates the present network load. Bandwidth-delay product is derived from the rate of acknowledgements received and round trip time. Chen et al. [5] propose a receiver-aided mechanism in which the TCP receiver monitors the contention state of the end-to-end connection and the TCP sender is informed about it via the acknowledgement mechanism. TCP receiver uses end-to-end delay as contention to decide network congestion level and the same is notified to TCP sender, which decides the congestion window size based on it.

The authors in [40] suggest that the entire lifetime of a TCP connection is allocated into a finite number of slots and a constant TCP congestion window is used during these slots where the network scenario is assumed to remain unchanged. The beginning of a new slot triggers window recalculation and the congestion window would be set according to the connection's available share in that slot. Elrakabawy et al. [9] proposed a new TCP congestion control algorithm in wireless networks, called TCP with adaptive pacing (TCP-AP). This algorithm measures the change of round-trip delay and determines the network congestion status based on the delay. TCP-AP estimates the appropriate congestion window size according to varying round-trip delays, and the algorithm also introduces rate control during the congestion window adjusting to avoid outburst flows.

Some study has gone into TCP congestion window overshooting problem. Nahm et al. [29] proposed a modified congestion window adaptation mechanism, in which, instead of increasing additively, TCP congestion window is allowed to increase every round trip time (RTT) by a certain fractional rate. Papanastasious et al. [31] proposed a slow congestion avoidance (SCA) scheme. During the congestion avoidance phase in traditional TCP mechanism, the *cwnd* value linearly increases by one maximum segment size (MSS) for every round trip time. In the SCA scheme, authors have defined a new variable *ca_increase_thresh* indicating the number of ACKs received before TCP starts congestion window adaption. *cwnd* value is kept unchanged till

the number ACKs received is smaller than *ca_increase_thresh*. Once the number reaches *ca_increase_thresh*, the *cwnd* value is increased by one MSS. This way, the *cwnd* value increases slowly mitigating the congestion window overshooting problem.

4.3 Proposed Mechanism

The main notion of the proposed mechanism is to keep the congestion window as high as possible during congestion control. There are mainly two scenarios when congestion window is reduced. One is during a RTO and the other is when the TCP sender receives a threshold number (usually set to three) of duplicate ACKs.

When an RTO occurs, TCP sender enters slow start phase, where *cwnd* is reduced to 1 and *ssthresh* is set to half the value of existing *cwnd*. *cwnd* is increased additively till *ssthresh* is reached. When *cwnd* reaches *ssthresh*, it is further increased linearly till a packet loss is detected. The first idea of the proposed mechanism is to set *ssthresh* and *cwnd* to half the existing *cwnd* when a retransmission timeout occur, with a minimum of 2 * maximum segment size (MSS). This keeps the *cwnd* value considerably high so that a sudden decrease in TCP throughput is not observed. By reducing the *cwnd* to half, we reduce the probability of further congestion occurrence at the same time keep its value high enough for efficient use of the available bandwidth. The algorithm for the above mentioned concept is as given below and corresponding flowchart is presented in Figure 16.

Algorithm 1: Retransmission Timeout Handling

```
if (retransmission timeout)
{
  ssthresh = (send_max - send_unacked) / 2;
  if (ssthresh <= 2.0 * send_mss)
    {
      ssthresh = 2.0 * snd_mss;
    }
  cwnd = ssthresh;
}
```

where *send_max* is sequence number of the latest packet sent, *send_unacked* is the sequence number of first unacknowledged segment and *send_mss* is the maximum segment size for outgoing segments.

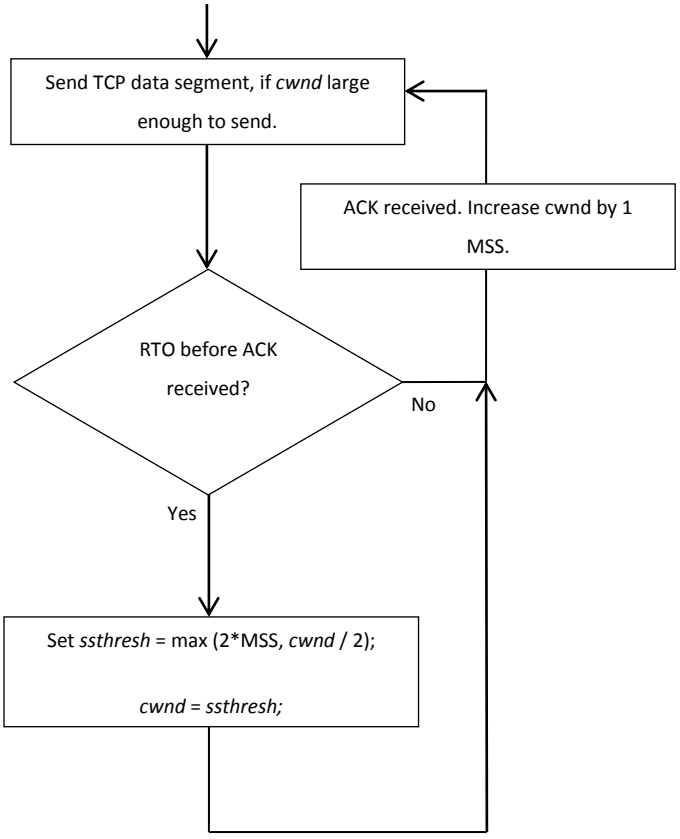

Figure 16: Flowchart for RTO handling

After receiving a threshold number of duplicate acknowledgements (usually set to three), TCP sender enters fast retransmit and fast recovery phase. During Fast Retransmit, the oldest segment in the retransmission buffer for which an acknowledgement is not received will be resent immediately even though no timeout has occurred. After fast retransmission the congestion window is reduced to half of the existing congestion window. Also each duplicate packet is considered an ACK for an already sent packet and congestion window is thus incremented for each duplicate ACK received and a packet is sent if the new congestion window size permits. The second idea of the proposed mechanism is to set *cwnd* to three fourth of the existing *cwnd* instead of half during fast retransmission. This keeps the *cwnd* value in the proximity of where it was before entering fast retransmission and fast recovery phase and avoids sudden decrease

in TCP throughput. The algorithm for the above mentioned concept is as given below and the corresponding flowchart is presented in Figure 17.

Algorithm 2: Threshold Duplicate ACK Handling

```
if (threshold duplicate acks received)
{
    ssthresh = (send_max - send_unacked)* 3 / 4;
    if (ssthresh <= 2.0*snd_mss)
    {
       ssthresh = 2.0*snd_mss;
    }
    cwnd = ssthresh;
}
```

where *send_max* is sequence number of the latest packet sent, *send_unacked* is the sequence number of first unacknowledged segment and *send_mss* is the maximum segment size for outgoing segments.

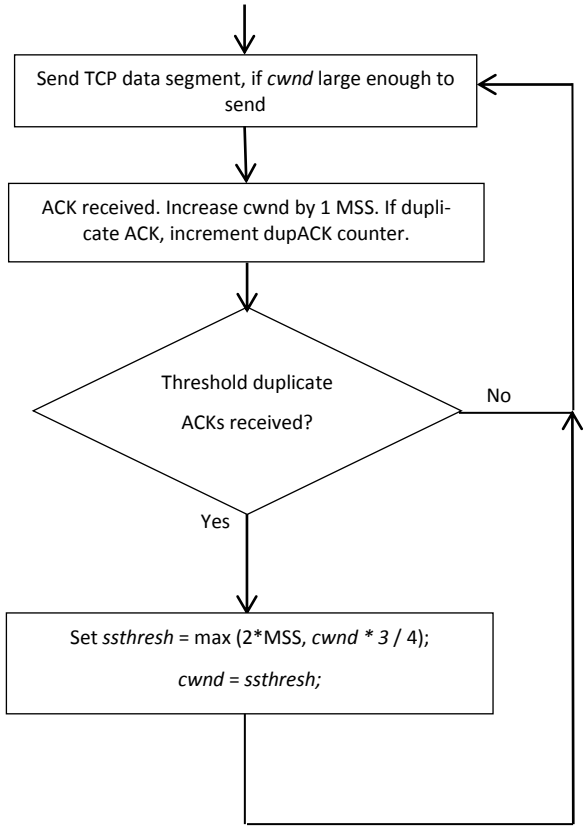

Figure 17: Flowchart for Threshold Duplicate ACK handling

4.4 Simulation and Analysis

In this section, we describe simulation environment and parameter setting done for performance comparison between TCP Reno and modified TCP with the above mentioned changes. Simulation is carried using OPNET simulator. Mobility of nodes is not considered during the simulation. At MAC layer, standard 802.11g with data rate of 24 Mbps with 256 KB buffer size is used for all the simulations. The maximum number of retransmissions at MAC layer is set to 4 for packets larger than 256 bytes (Long_Retry_Limit) and 7 for other packets (Short_ Retry_Limit) as specified in IEEE 802.11 MAC standard. Nodes are using AODV as the routing protocol. At transport layer, TCP Reno with duplicate ACK threshold of 3 and an initial retransmission timeout of 2 seconds is used. File Transfer Protocol (FTP) is used for all the data transfer over TCP. Best effort type of service is used for QoS purpose. Random loss is introduced by randomly dropping packets at the IP level. Table 1 shows the parameters used for OPNET simulator. Different scenarios with multiple hops and single hop wireless connections are simulated and corresponding TCP throughput and congestion window size is compared.

Table 1. OPNET simulator parameters for congested network

Parameter	Value
MAC data rate	24 Mbps
MAC buffer size	256 KB
MAC Long Retry Limit	4
MAC Short Retry Limit	7
Routing Protocol	AODV
TCP Version	Reno
TCP Duplicate ACK Threshold	3
Initial RTO	2 seconds

4.4.1 Single Hop Scenario

In single hop scenario, Figure 18, two nodes are talking to each other over wireless medium directly with congestion introduced at the sender side. FTP traffic is sent from sender to receiver using both the standard TCP Reno and our proposed TCP as transport layer protocol and compared against each other. From the results we observed that the average throughput is increased when the congestion window is kept at values nearer to where it was before congestion occurred. By using the proposed mechanism, congestion window is retained at higher values and thereby higher TCP throughput is achieved.

Figure 19 and Figure 20 show the TCP throughput and congestion window comparison, respectively, between TCP Reno and TCP with proposed changes, in single hop scenario. TCP throughput obtained using our proposed change is considerably healthier and during our simulations we observed that on an average 20 – 25% throughput increase is achieved. Figure 20 shows that when proposed TCP is used the congestion window is retained very close to the values before congestion started and simulation results show that on an average, congestion window size is around 25 – 30% higher than that achieved while using TCP Reno.

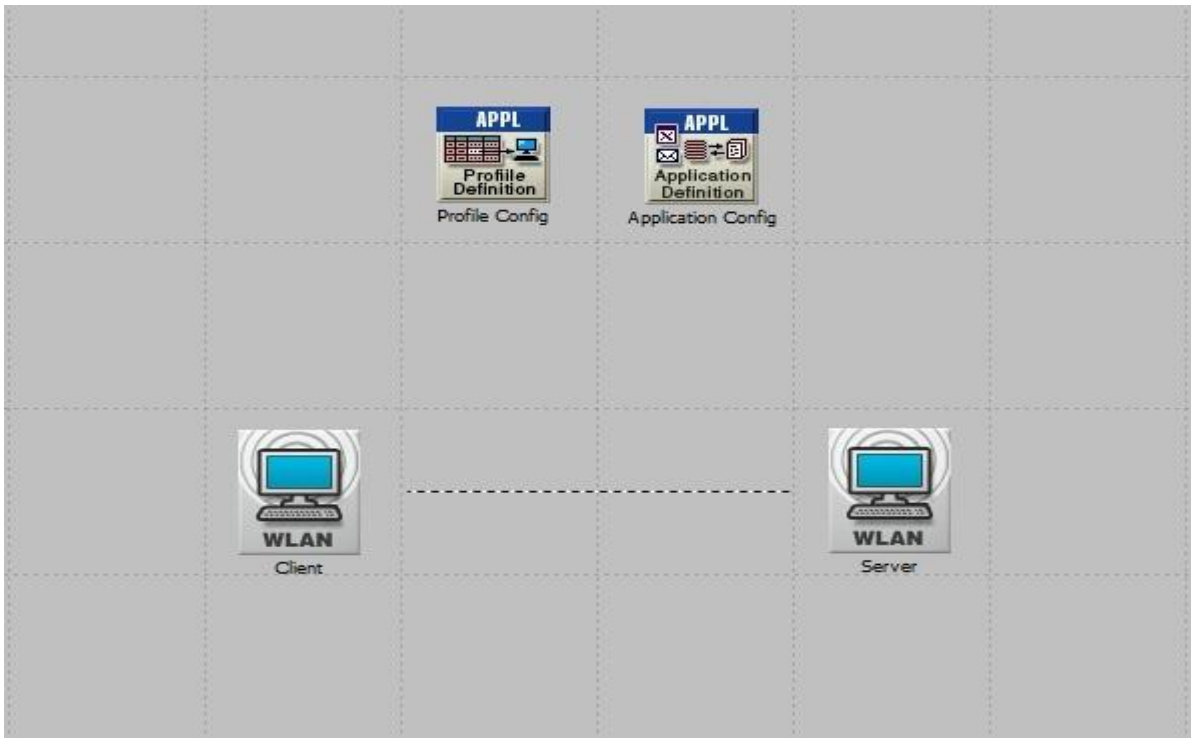

Figure 18: OPNET Simulation Model – Single Hop

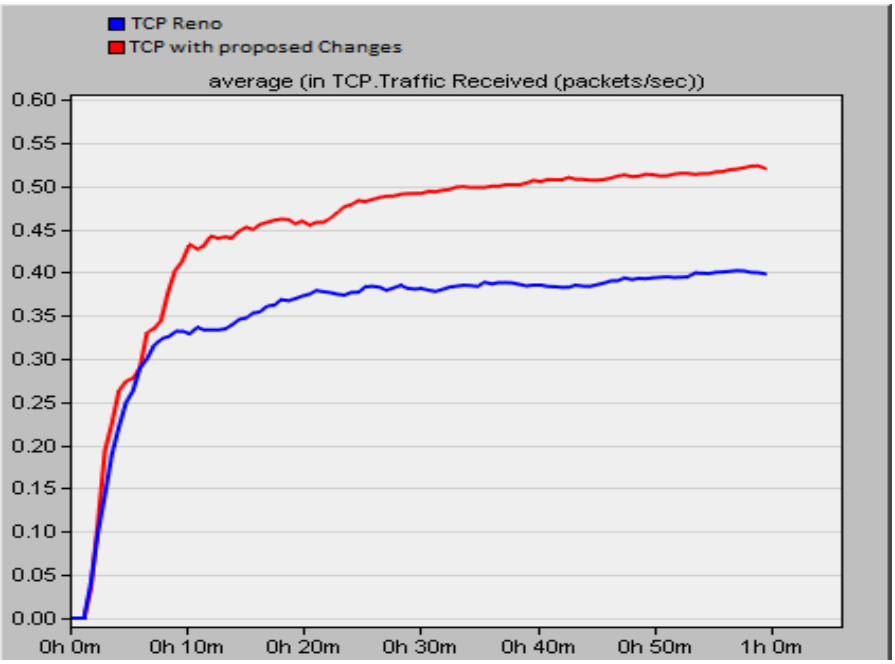

Figure 19: Single hop throughput comparison

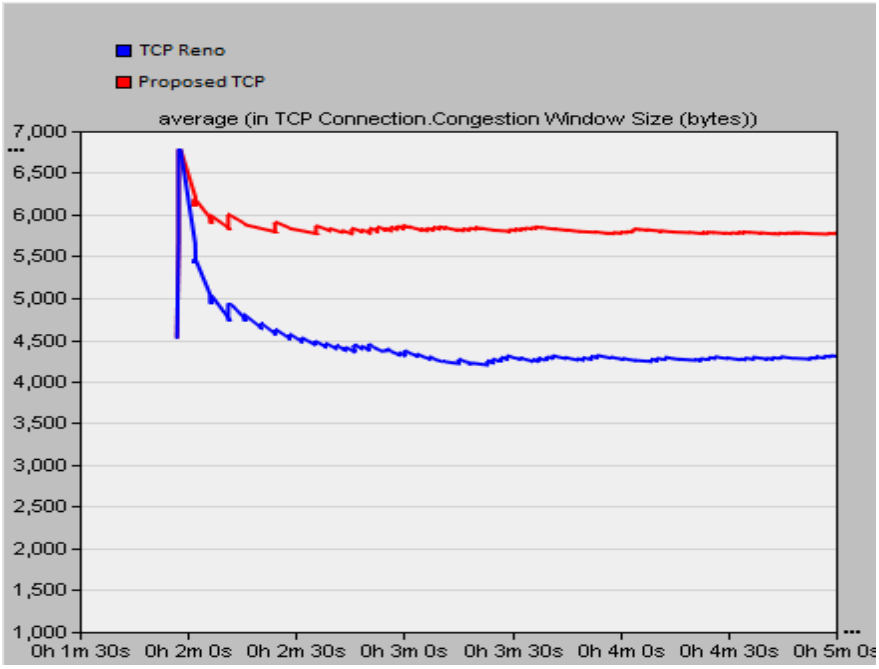

Figure 20: Single hop congestion window comparison

4.4.2 Multi Hop Scenario

We used a multi hop chain topology with four wireless nodes, Figure 21, with congestion introduced at the sender side. Again, FTP is used as the application protocol with TCP Reno and

proposed TCP at the transport layer. All the nodes were using same TCP version during simulation. The proposed mechanism aids in keeping the congestion window at higher values and thereby higher TCP throughput is achieved. From the results we observed that although the average throughput is increased it is considerably less than single hop scenario.

Figures 22 and 23 show the TCP throughput and congestion window comparison, respectively, between TCP Reno and TCP with proposed changes, in multi hop scenario. During our simulations, proposed TCP in multi hop scenario achieved on an average 35 – 40% higher throughput than TCP Reno and congestion window is around 45 – 50% higher than observed with TCP Reno.

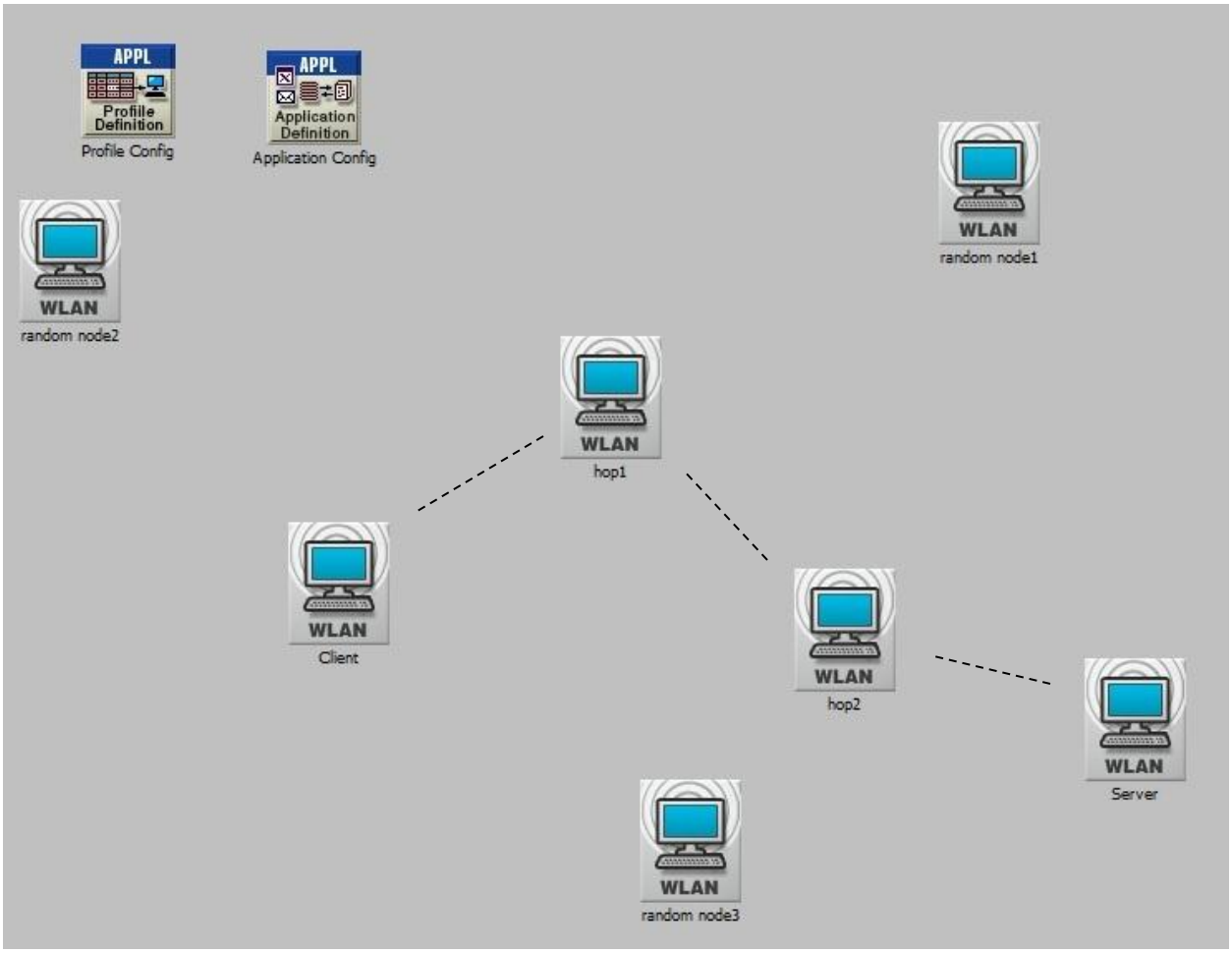

Figure 21: OPNET Simulation Model – Multi Hop (4 Nodes)

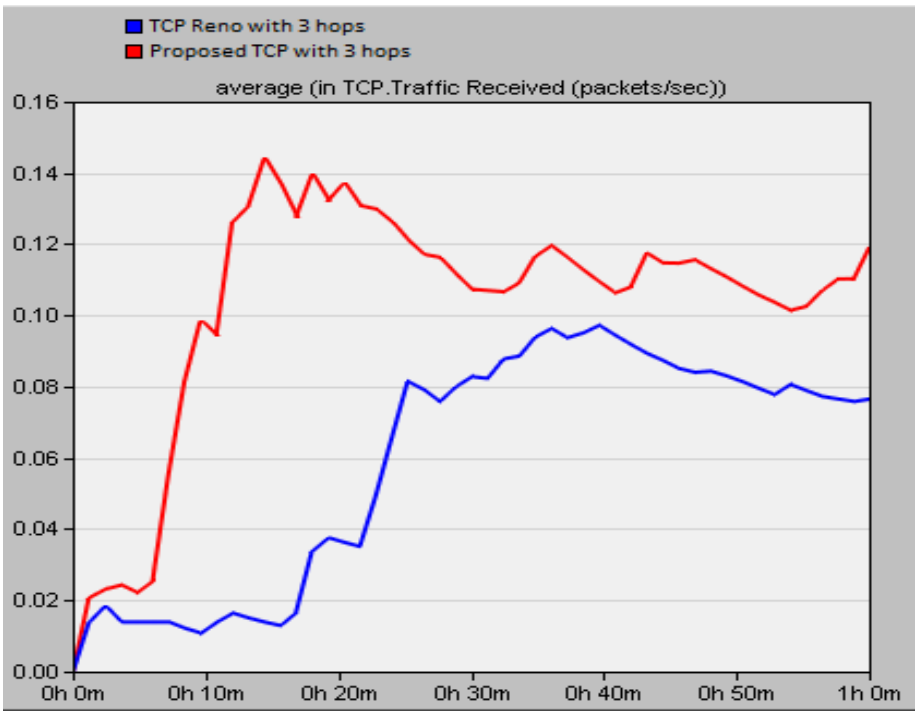

Figure 22: Multi hop throughput comparison

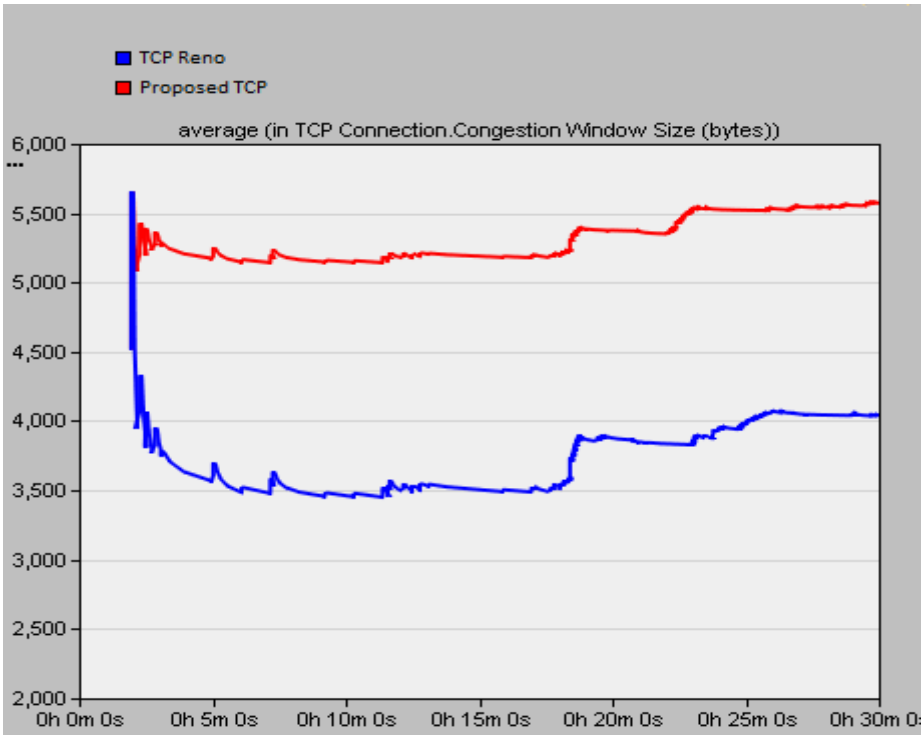

Figure 23: Multi hop congestion window comparison

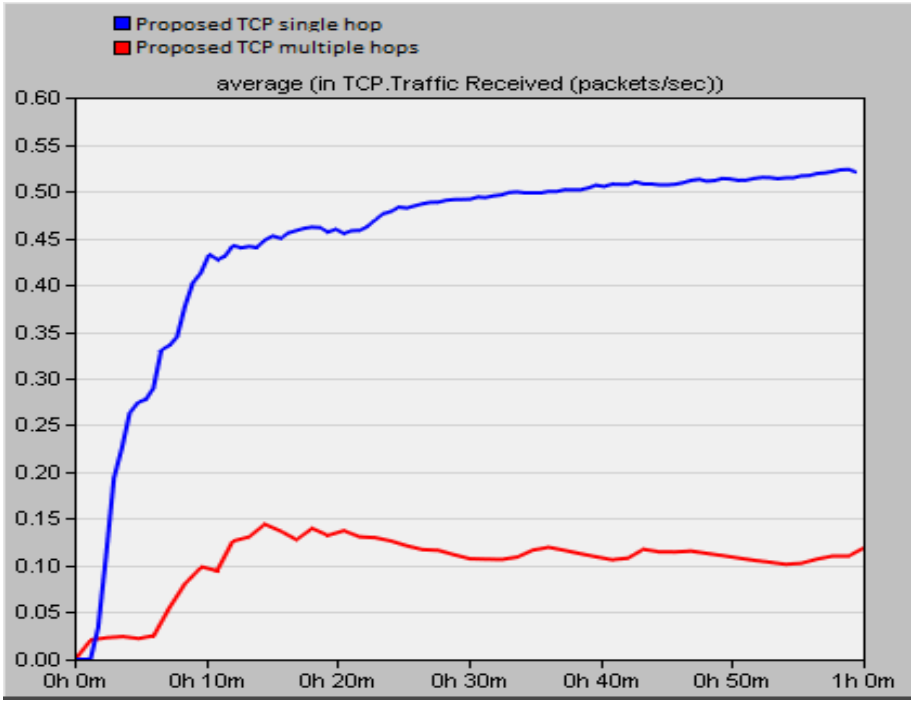

Figure 24: Single hop vs Multi hop throughput comparison

It was observed that as the number of hops increased, the congestion increased due to link layer contention (related timers DIFS, SIFS, PIFS) and increased control traffic (RTS / CTS). As shown in Figure 24, the throughput observed is significantly less in multi hop scenario when compared to single hop scenario.

Figure 25 shows a comparison of multi hop TCP throughput with different number of hops. We have compared proposed TCP throughput against 1, 2, 3, 5 and 7 hops. The results show that as the number of hops increases, link layer contention plays a big part in determining the throughput achieved.

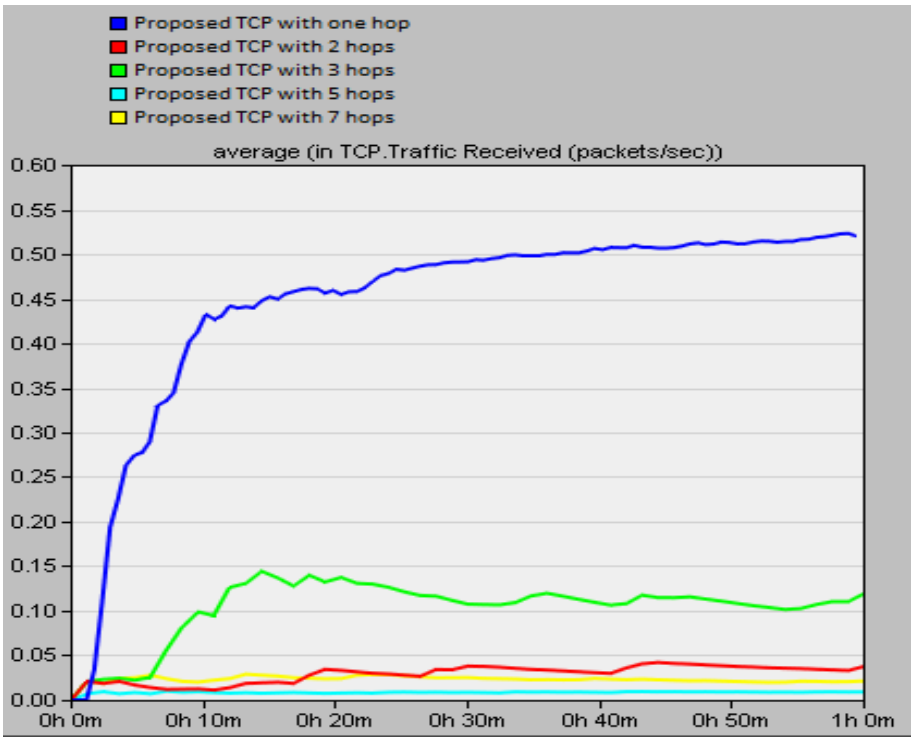

Figure 25: Comparison of throughput with multiple nodes

4.5 Summary

In this chapter, we have proposed sender side TCP modifications to improve TCP performance in wireless networks by incorporating two schemes. We have appraised the performance of the proposed schemes with extensive simulations using OPNET. The simulation results have confirmed that the proposed schemes have resulted in significant performance improvement over TCP Reno. TCP Reno was chosen as the base TCP version due to its widespread use and better handling of duplicate ACKs. TCP Tahoe does not support fast recovery and TCP New Reno wrongly triggers fast recovery when 3 or more packets are reordered. The important feature of the proposed scheme was to retain congestion window as near as possible to the value when congestion occurs. Changes were made to handle congestion window calculation differently during retransmission timeout and duplicate acknowledgement arrival. Also, the proposed changes were limited to sender side only so that it is easy to incorporate in existing wireless networks.

Table 2 shows how the proposed mechanism fares against TCP-Reno in different circumstances. Average throughput and congestion window size in single hop and multi hop scenarios are presented and percentage increase when compared against TCP-Reno is shown.

Table 2. Comparison of throughput and congestion window with proposed changes

Scenario	*Throughput (Packets / Sec)*	*Congestion Window (Bytes)*	*Percent Increase*	
			Throughput	*Congestion Window*
Single Hop	0.46	5500	20 - 25	25 - 30
Multi Hop	0.11	5250	35 - 40	45 - 50

In single hop scenario, a substantial increase of 20 – 25% was observed in TCP throughput and an even better 25 – 30% increase in congestion window size was attained. In multi hop scenario, a huge increase of 35 – 40% and 45 – 50% was attained for TCP throughput and congestion window respectively. Even though the percentage increase in multi hop scenario is much higher than the single hop scenario, due to link level contention the overall TCP throughput in multi hop scenario is much smaller than single hop scenario.

Chapter 5 - TCP Retransmission Modification for Delay Sensitive Wireless Networks

5.1 Introduction

TCP provides various mechanisms for reliable, ordered, non-duplicated, congestion less and flow controlled data transmission between network devices. Hence, TCP is the most trusted transport layer protocol in use. However, when taken into wireless environment where not everything works as predicted, TCP loses a little bit of its shine. Especially when introduced to delay oriented networks, the throughput is considerably reduced because of the timeouts and the corresponding retransmissions and congestion control measures.

One has to take special care with TCP when using in networks where delay is expected, most notably satellite communication networks (Figure 26). Round trip time is considerably high, as expected, in satellite networks and will lead to timeouts. Even if special care is taken to adopt RTT values according to the network conditions, due to the unpredicted nature of wireless networks timeouts are common. TCP retransmission technique has to be modified to take into consideration the volatile nature of wireless networks to succeed in providing higher throughput. This chapter provides a mechanism to tackle this very aspect in a simple and efficient manner.

5.2 Related Work

Numerous schemes have been proposed in wireless networks to handle TCP retransmission in wireless environment in such a way that the overall throughput is not affected greatly. Some authors have combined link layer retransmission mechanisms and TCP mechanisms to achieve better results. Zhizhao et al. in [51] combine link layer selective-reject automatic repeat request (ARQ) mechanism with explicit loss notification mechanism, which then is capable of responding to congestion swiftly while keeping the wireless links more reliable and allowing TCP to react to packet losses aptly depending on the cause.

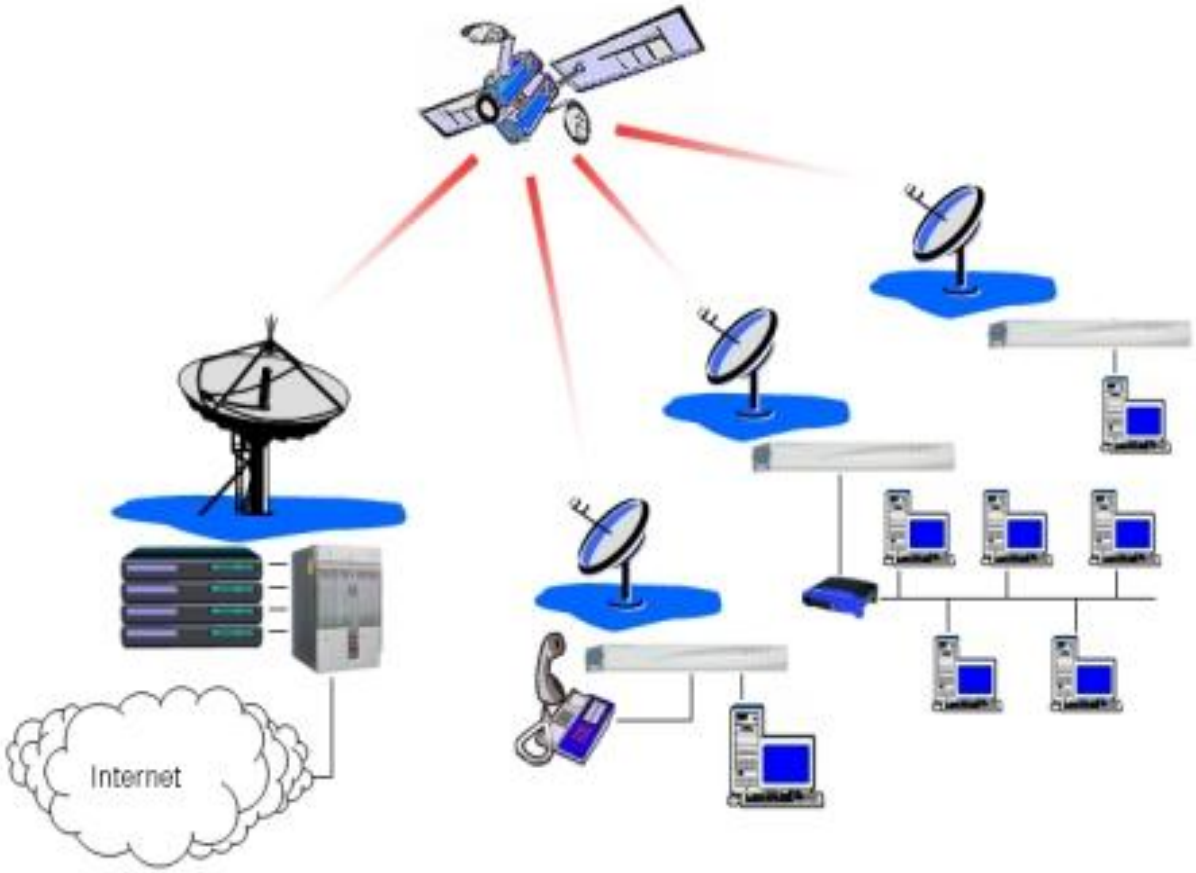

Figure 26: Satellite Network [18]

Guo et al. in [14] propose a novel cross layer design for retransmission in multipath wireless mesh networks (WMN). Path Diversified Retransmission (PDR) makes use of a new QoS aware multipath routing protocol, QAOMDV, to differentiate original packets from retransmitted packets and send them separately in case of RTO. PDR overcomes the problems of RTT estimation and packet reordering in multipath networks and improve the throughput. Authors in [45], suggest a mechanism to detect loss of fast retransmitted packets before timeouts and lessen retransmission timeouts. The new method, Time based – Detection of the Loss of fast Retransmitted Packets (T-DLRP), detects loss of fast retransmitted packets and reacts without waiting for the corresponding retransmission timeout. The mechanism uses three schemes namely, FRL (Fast Retransmission Loss)-Detection, FRL-Differentiation and Immediate Recovery.

Park et al. in [32] suggest an algorithm to detect the cause of retransmission timeout and based on the cause decide whether to reduce congestion window and increase retransmission timeout back-off value. The algorithm estimates the queue usage in the network path during go-back-N retransmissions and determines the cause of RTO, whether it is congestion or not.

5.3 Proposed Mechanism

The main idea of the proposed mechanism is to reduce retransmission timeouts as much as possible and keep TCP running without reducing the congestion window. A timeout occurs when TCP sender does not receive an acknowledgement for a sent packet before the retransmission timer expires. An RTO causes the TCP to trigger slow start which reduces the congestion window to 1 and start rebuilding the congestion window as and when acknowledgements are received. This triggering of slow start mechanism reduces the TCP throughput radically.

Proposed method suggests of keeping the congestion window intact when retransmission timeout occurs by restarting the retransmission timer with newly calculated value. If an RTO occurs for the first time for the TCP connection, retransmission timer is restarted with one and half times the existing value and a flag is set suggesting its running on the extended retransmission timer and TCP connection proceeds as normal. On successful reception of an acknowledgment the flag is reset. If retransmission timer times out for the extended timer also, then slow-start mechanism is triggered. For subsequent RTO on the same TCP connection, retransmission timer is restarted with double the existing value as we know that this connection happens to be in a delayed environment. Care is taken with proper flags, so that situation where retransmission timer never expires does not arise. The algorithm for the above mentioned concept is as given below and corresponding flowchart is presented in Figure 27.

Algorithm: Retransmission Timeout Handling in Delay Networks

```
if (retransmission timeout for first time)
/* delayed_connection == 0 && current_rto == configured_rto */
{
  current_rto = MIN ((1.5 * configured_rto), rto_max);
  delayed_connection = 1;
   timer (current_rto);
} else if (non-first regular timeout on same connection)
 /* delayed_connection == 1 && current_rto == configured_rto */
 {
   current_rto = MIN ((2 * configured_rto), rto_max);
  timer (current_rto);
} else /* after extended timeout - trigger slow-start */
{
    ssthresh = (send_max - send_unacked) / 2;
    if (ssthresh <= 2.0 * send_mss)
    {
     ssthresh = 2.0 * snd_mss;
    }
    cwnd = 1;
 }
```

where *current_rto* is the retransmission timeout value associated with the packet sent, *configured_rto* is the default or configured retransmission timeout value, *send_max* is sequence number of the latest packet sent, *send_unacked* is the sequence number of first unacknowledged segment and *send_mss* is the maximum segment size for outgoing segments, *ssthresh* is the slow-start threshold value and *cwnd* is the congestion window size.

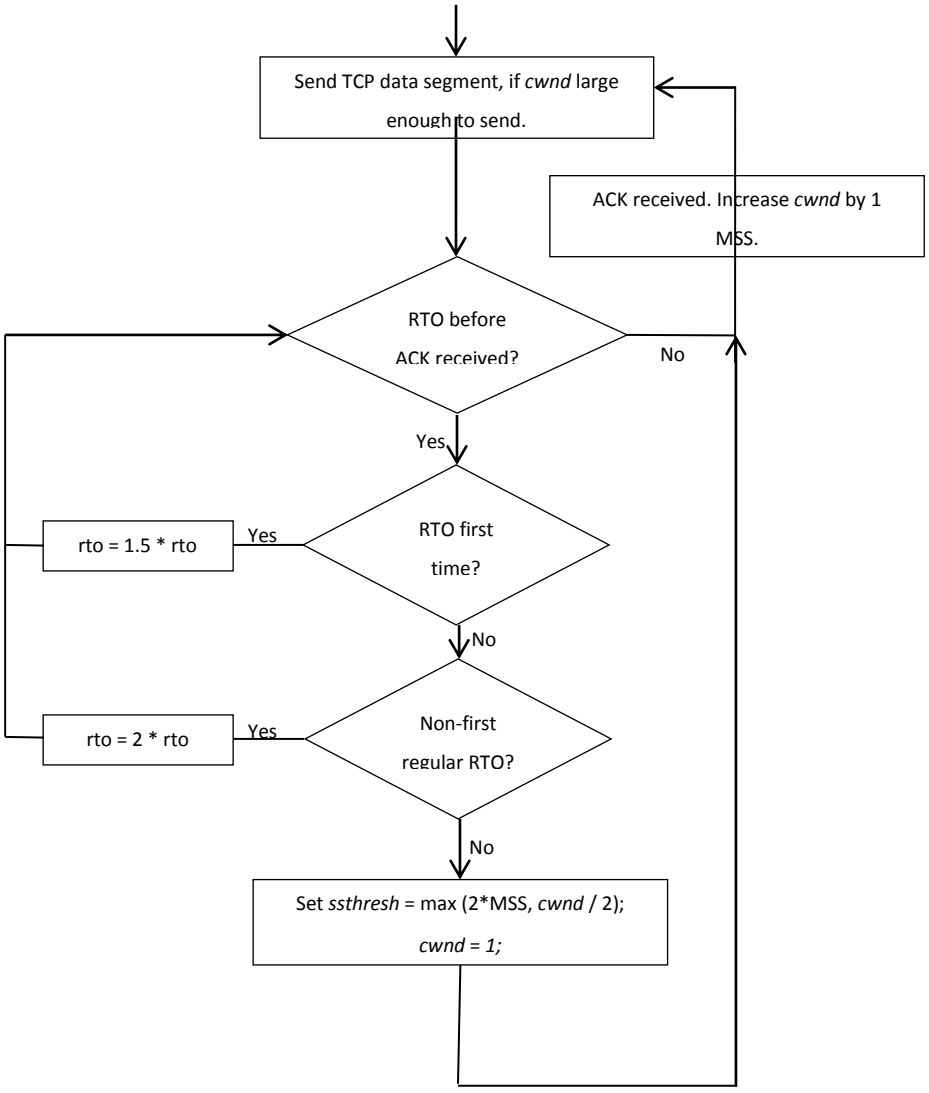

Figure 27: Flowchart for proposed mechanism

5.4 Simulation and Analysis

Simulation is carried using OPNET simulator for performance comparison between TCP New-Reno and modified TCP with the above mentioned changes. We assume nodes to be fixed and symmetric channel allocation at sender and receiver side. Standard 802.11g is used at MAC layer and data rate of 24 Mbps with 256 KB buffer size is used for all the simulations. As per IEEE 802.11 MAC standard specification the maximum number of retransmissions at MAC layer is set to 4 for packets larger than 256 bytes (Long_Retry_Limit) and 7 for other packets (Short_ Retry_Limit). AODV is used as the routing protocol and at transport layer TCP New-Reno with duplicate ACK threshold of 3 and an initial retransmission timeout of 2 seconds is used. File

Transfer Protocol (FTP) is used for all the data transfer over TCP. Best effort type of service is used for QoS purpose. Delay is introduced by reducing the rate of packet handling at the network layer (IP). Table 3 shows the parameters used for OPNET simulator. Simulations are carried out to show the effect of delay on TCP throughput. Also simulations are done with modified TCP in single hop and multiple hops scenarios and corresponding TCP throughput and congestion window size is compared with TCP New-Reno.

Table 3. OPNET simulator parameters for delayed network

Parameter	Value
MAC data rate	24 Mbps
MAC buffer size	256 KB
MAC Long Retry Limit	4
MAC Short Retry Limit	7
Routing Protocol	AODV
TCP Version	New-Reno
TCP Duplicate ACK Threshold	3
Initial RTO	2 seconds

A network with delay is simulated by introducing delay in the wireless medium connecting the devices. FTP traffic is sent from sender to receiver in network without delay and with delay. The TCP throughput and congestion window is compared, and Figure 28 and 29 show the results.

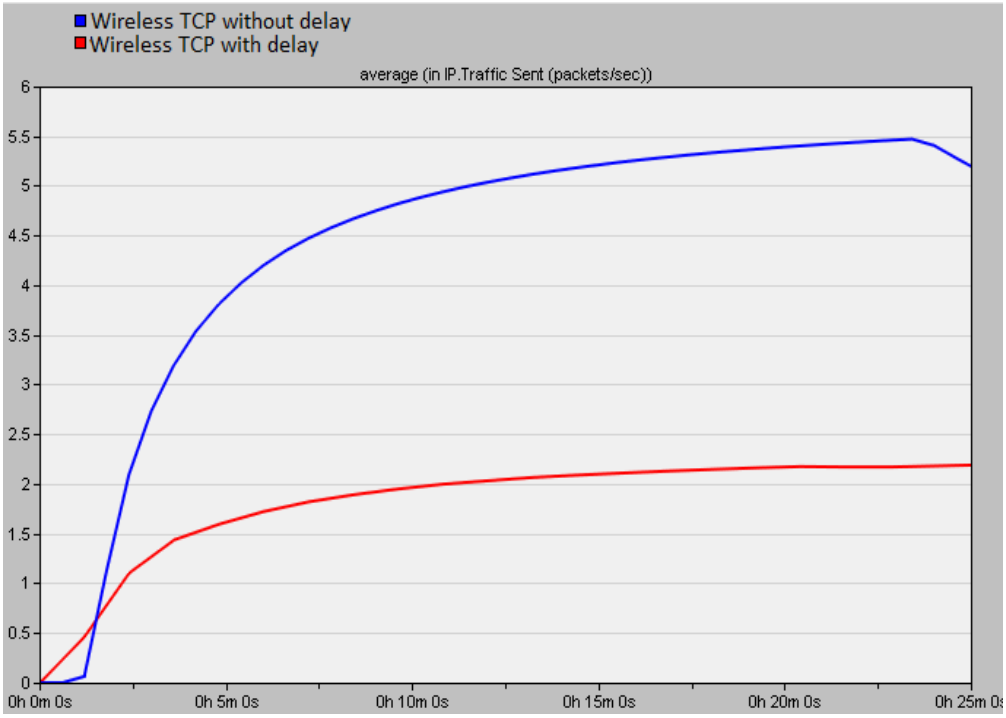

Figure 28: Throughput comparison with and without delay

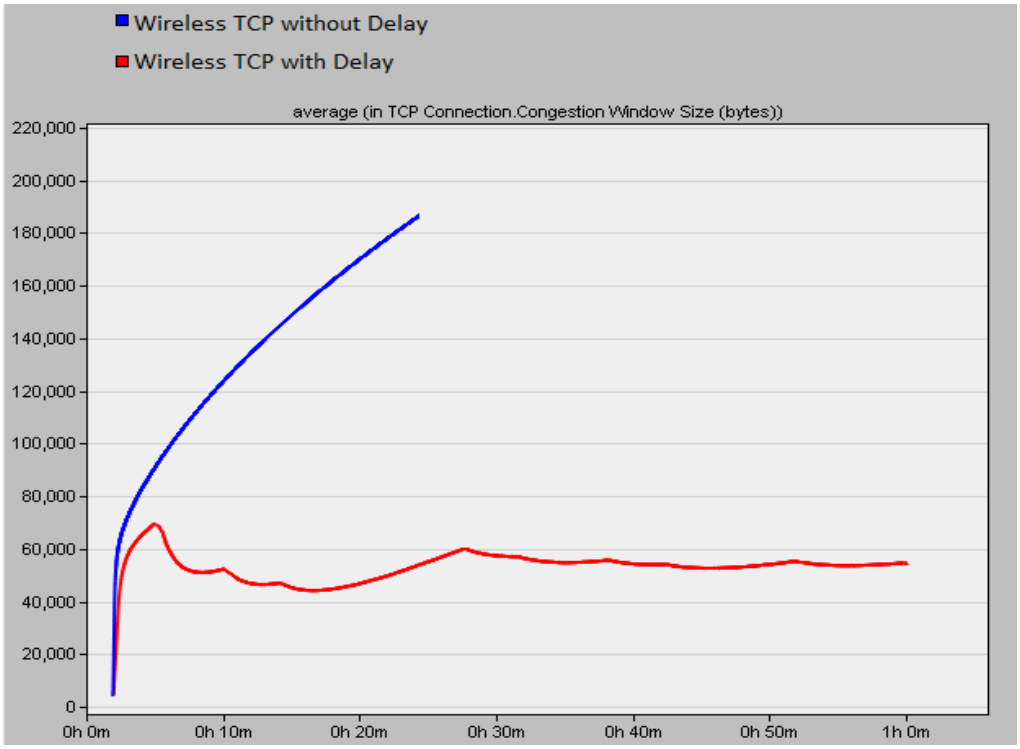

Figure 29: Congestion window comparison with and without delay

Above results show that delay in networks reduce the throughput radically. Figure 29 show that congestion window is reduced frequently thereby reducing the rate at which the sender can send traffic to the receiver. It can be construed that due to the delay in the network there are frequent timeouts which lead to periodic slow-starts which eventually leads to lowered bandwidth. This is where previously mentioned modifications can come handy. Simulations were conducted with single hop and multi-hop scenarios.

5.4.1 Single Hop Scenario

In single hop scenario, two nodes are talking to each other over wireless medium directly with delay introduced at the wireless medium. FTP traffic is sent from sender to receiver using both the standard TCP New-Reno and our proposed TCP as transport layer protocol and the compared against each other. Results second that when the number of timeouts are lessened, the throughput increases correspondingly. By using the proposed mechanism, in delayed networks timeouts are avoided if the acknowledgements are received in proximate time period.

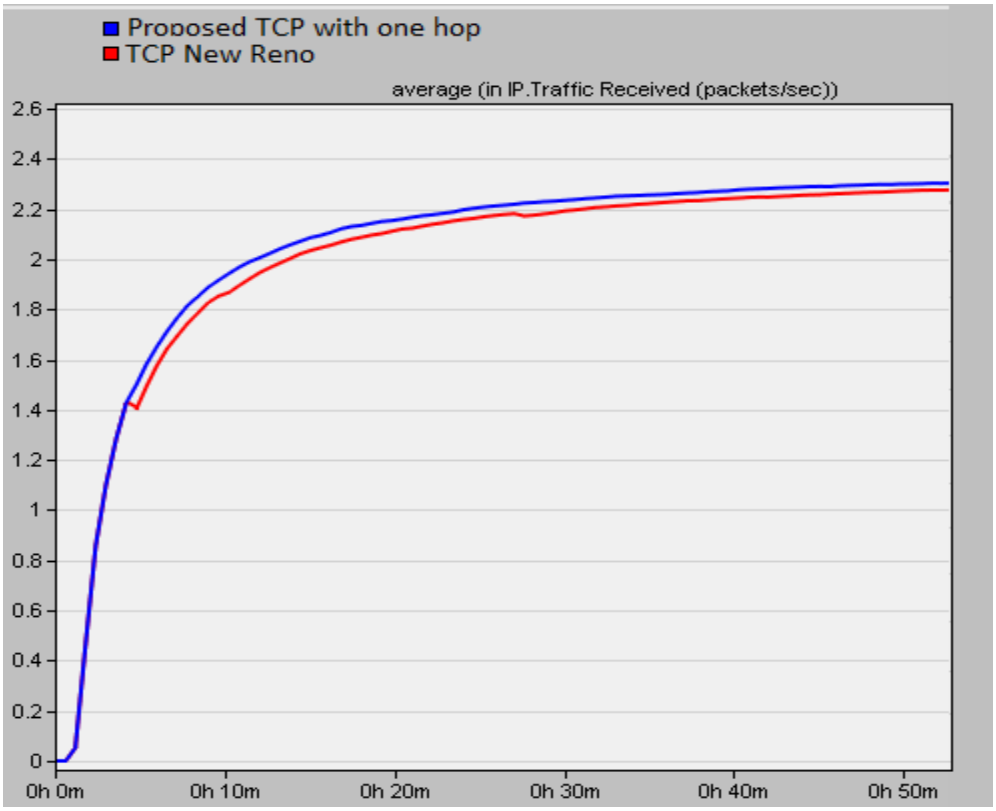

Figure 30: Single Hop throughput comparison in delayed environment

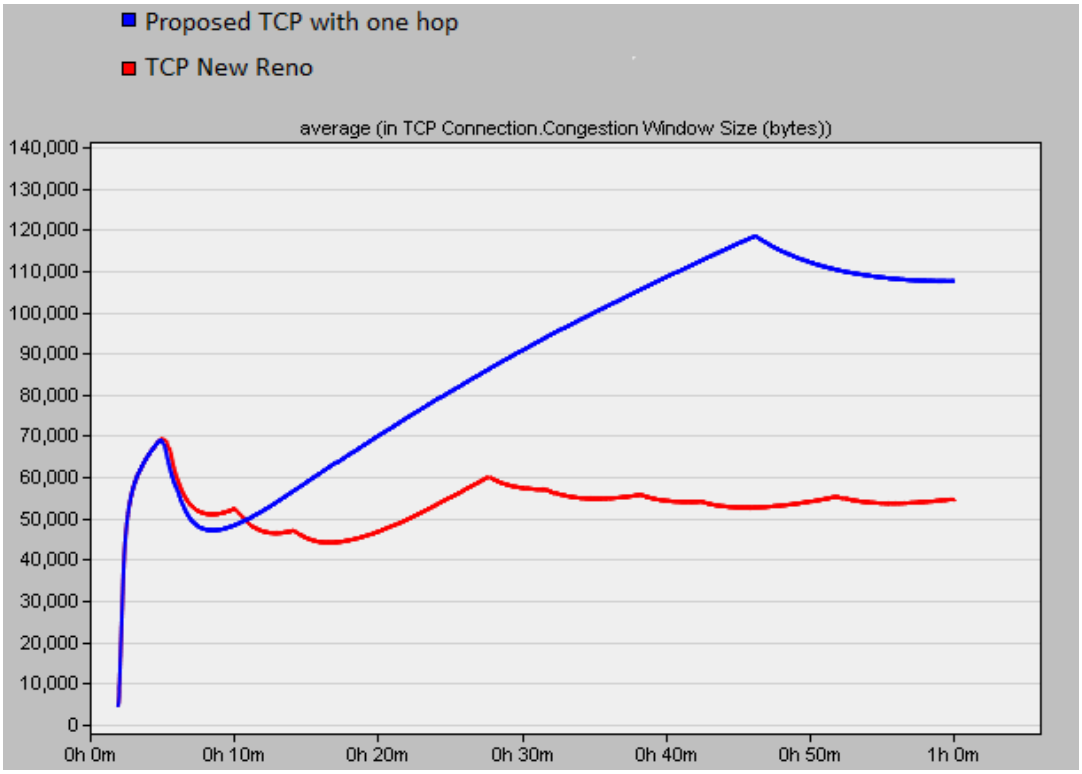

Figure 31: Single Hop throughput comparison in delayed environment

Figure 30 and Figure 31 show the TCP throughput and congestion window comparison, respectively, between TCP New-Reno and TCP with proposed changes, in single hop scenario. TCP throughput obtained using our proposed change shows some improvement over TCP New-Reno and during our simulations we observed that on an average 3 – 5% throughput increase is achieved. However, congestion window shows considerable improvement as reduction of congestion window is avoided by allowing its retransmission timeout to run little longer for TCP connections where delay is expected and during our simulations we observed that congestion window is maintained on an average 35 – 40% higher than TCP New-Reno.

5.4.2 Multi Hop Scenario

We used a multi hop chain topology with four and seven wireless nodes (Figure 32), with delay introduced at wireless medium at each intermediate node. Again, FTP is used as the application protocol with TCP New-Reno and proposed TCP at the transport layer. All the nodes

were using same TCP version during simulation. The proposed mechanism delivers some improvement in throughput aspect and considerable betterment in congestion window front.

Figures 33 and 34 show the TCP throughput and congestion window comparison, respectively, between TCP New-Reno and TCP with proposed changes, with 4 nodes multi hop scenario. During our simulations, proposed TCP in 4 node multi hop scenario achieved on an average a 5 – 10% higher throughput than TCP New-Reno and congestion window is around 45 – 50% higher than what is observed with TCP Reno.

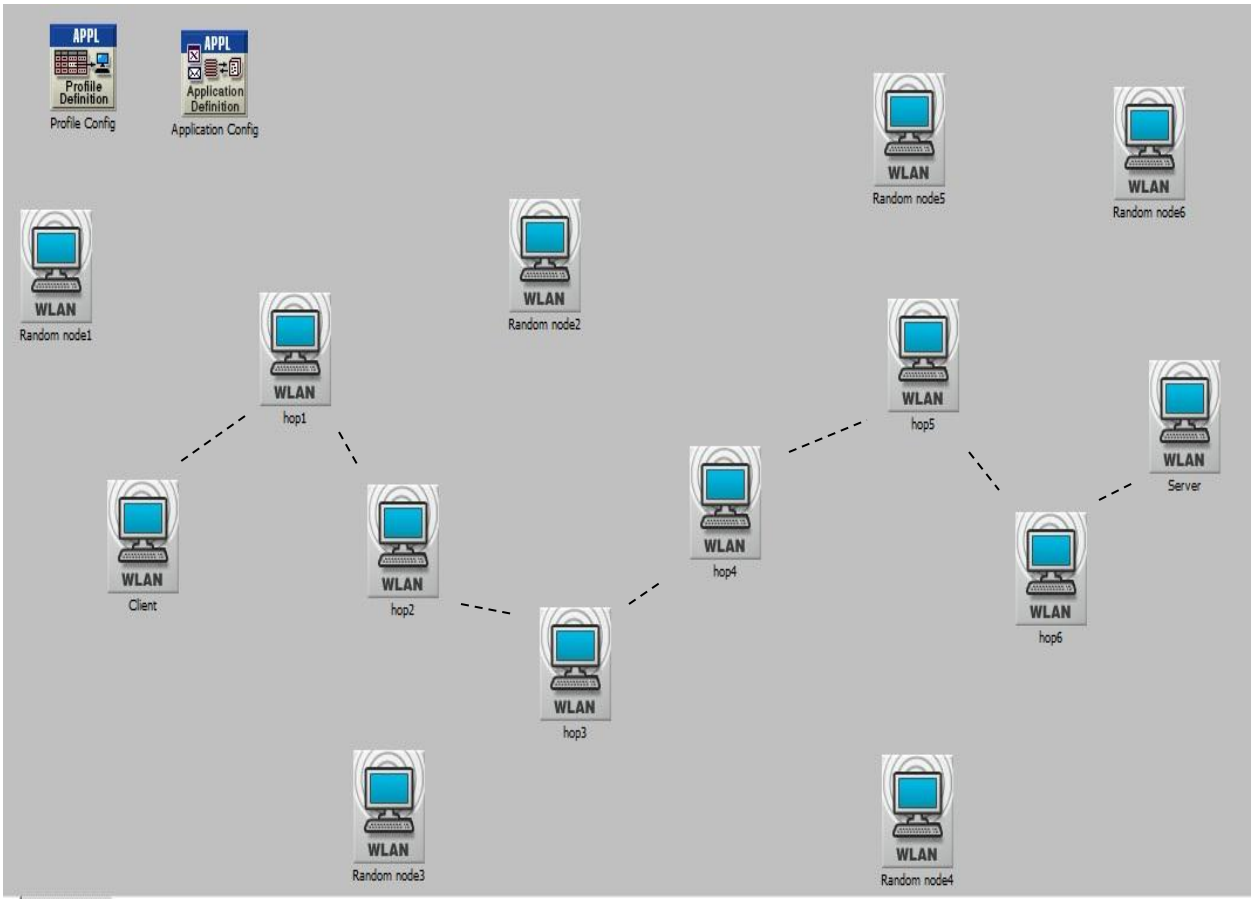

Figure 32: OPNET Simulation Model – Multi hop (7 Nodes)

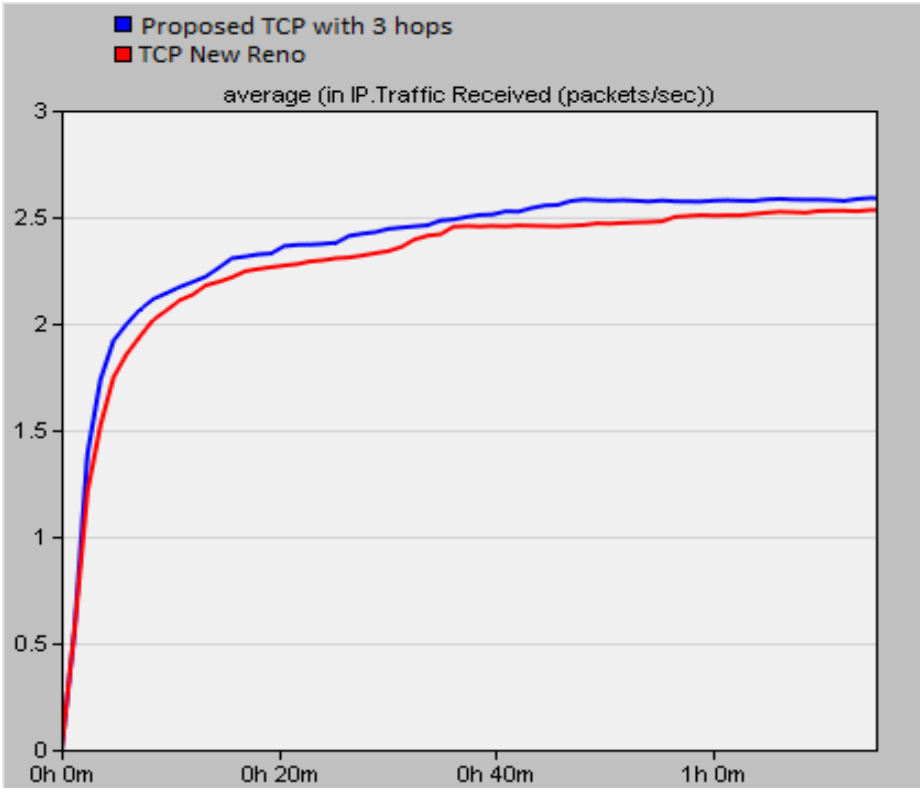

Figure 33: Multi hop (4 nodes) throughput comparison in delayed environment

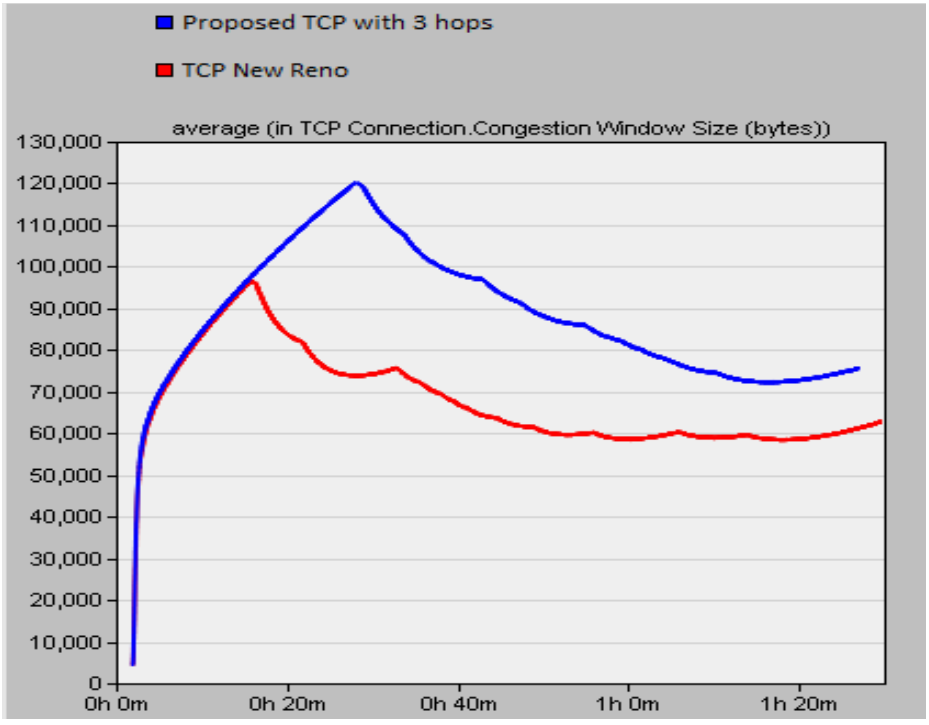

Figure 34: Multi hop (4 nodes) congestion window comparison in delayed environment

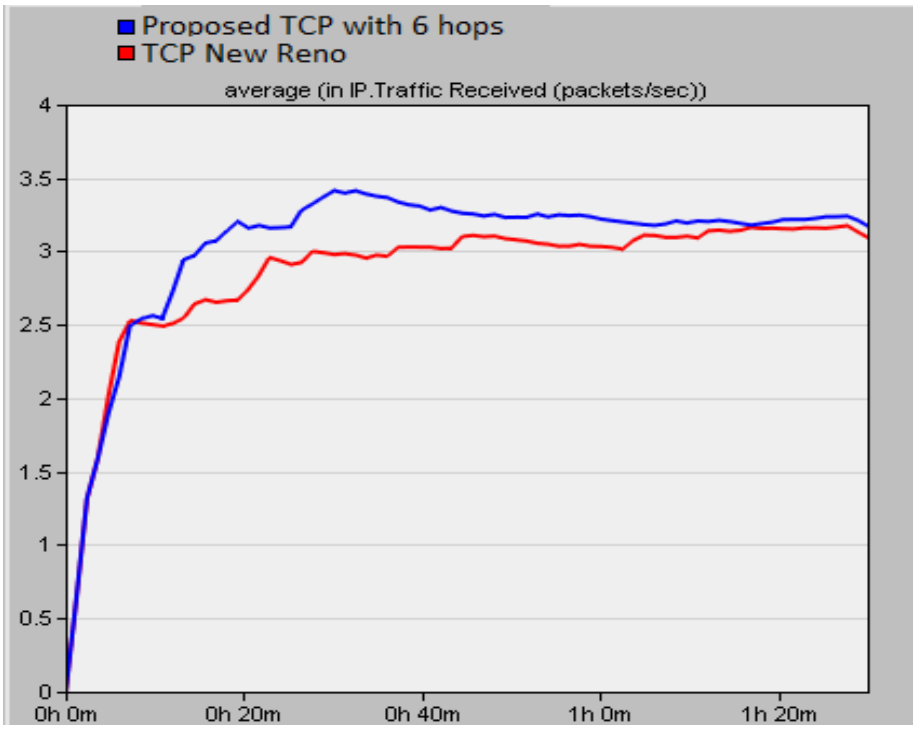

Figure 35: Multi hop (7 nodes) throughput comparison in delayed environment

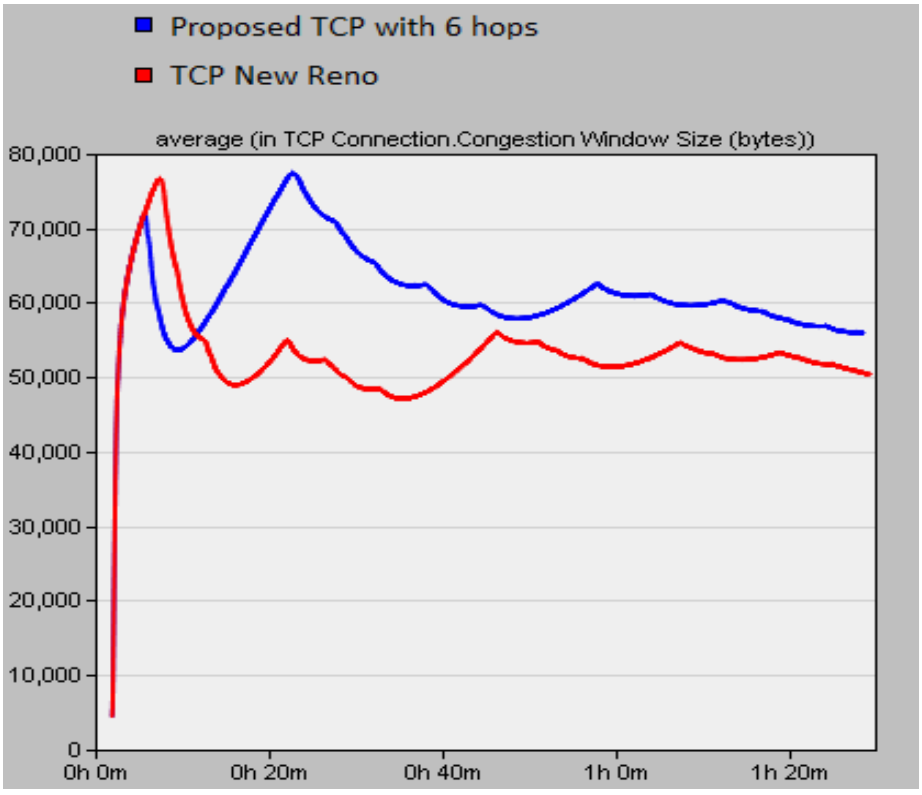

Figure 36: Multi hop (7 nodes) congestion window comparison in delayed environment

Figures 35 and 36 show the TCP throughput and congestion window comparison, respectively, between TCP New-Reno and TCP with proposed changes, with 7 nodes multi hop scenario. During our simulations, proposed TCP in 4 node multi hop scenario achieved on an average 15 – 20% higher throughput than TCP New-Reno and congestion window is around 35 – 40% higher than what is observed with TCP Reno.

5.5 Summary

In this chapter, we have proposed a sender side TCP modification to improve TCP performance in delay oriented wireless networks by changing the retransmission strategy. We have evaluated the performance of the proposed scheme with extensive simulations using OPNET simulator with delay introduced at the network level. The simulation results have confirmed that the proposed schemes have resulted in notable performance improvement over TCP New-Reno. The central aspect of the proposed scheme is to minimize the number of retransmission timeouts. Modifications are made to handle retransmission timeouts in such a way that it does not trigger congestion control each time the retransmission timer expires if the network is known to have delay in packet traversal. Also, the proposed changes were limited to sender side only so that it is easy to incorporate in existing wireless networks.

Table 4 shows how the proposed mechanism fares against TCP New-Reno in different circumstances. Average throughput and congestion window size in single hop and multi hop scenarios are presented and percentage increase when compared against TCP New-Reno is shown.

Table 4. Comparison of throughput and congestion window with proposed changes

Scenario	*Throughput*	*Congestion Window*	*Percent Increase*	
	(Packets / Sec)	*(Bytes)*	*Throughput*	*Congestion Window*
Single Hop	0.46	5500	3 - 5	35 - 40
Multi Hop	0.11	5250	5 - 10	45 - 50

In single hop scenario, an increase of 3 – 5% was observed in TCP throughput and 35 – 40% increase in congestion window size was achieved. In multi hop scenario, 5 – 10% and 45 – 50% increase was accomplished for TCP throughput and congestion window respectively. Even though the percentage increase in multi hop scenario is much higher than the single hop scenario, due to link level contention the overall TCP throughput in multi hop scenario is much smaller than single hop scenario.

Chapter 6 - Conclusions and Future Work

6.1 Summary of Results

In this thesis, we have presented a synopsis of recent techniques that have been adapted to increase TCP throughput in wireless networks. Also, we presented two novel methods to increase TCP performance in networks with random loss and delay. We have adopted changes only at sender side as changing at a single location is simple, cost effective, more feasible and realistic. Also changing at a single location gives the luxury to incorporate the changes without worrying the other side's TCP version.

In Chapter 3, we have presented an extensive survey on latest techniques used for improving TCP in wireless networks. We did not find any previously published article which discussed about recent techniques used for improving TCP performance and thought it is essential to prepare such an article [12] for future references. We have tried to incorporate techniques which make changes at sender side, sender and receiver and proxy based changes. We have given particular care to describe each technique precisely and compact without losing the fundamental idea.

In Chapter 4, we have proposed sender side TCP modifications [13] to improve TCP performance in wireless networks by incorporating two schemes. We have appraised the performance of the proposed schemes with extensive simulations using OPNET. The simulation results have confirmed that the proposed schemes have resulted in significant performance improvement over TCP Reno. TCP Reno was chosen as the base TCP version due to its widespread use and better handling of duplicate ACKs. TCP Tahoe does not support fast recovery and TCP New Reno wrongly triggers fast recovery when 3 or more packets are reordered. The important feature of the proposed scheme was to retain congestion window as near as possible to the value when congestion occurs. Changes were made to handle congestion window calculation differently during retransmission timeout and duplicate acknowledgement arrival. Also, the proposed changes were limited to sender side only so that it is easy to incorporate in existing

wireless networks. The proposed mechanism works particularly well in scenarios where packet drops in wireless medium is significantly high.

In Chapter 5, we have proposed sender side TCP modifications to improve TCP performance in wireless networks with considerable delay. Again, we have used OPNET to conduct simulations with the new changes incorporated and compared it against TCP New-Reno. TCP New-Reno was chosen in this case, since it is the most widely used TCP version. Changes were made to handle retransmission timeouts, which are common in delay centric networks, to increase TCP throughput and simulation results second this.

6.2 Future Work

Although developments in lower layers have substantially increased the reliability of wireless networks, TCP performance in wireless networks is much lower than its wired counterpart. This is mostly due to the control traffic, both at MAC layer and TCP Layer. Reducing control traffic at TCP is one way ahead to increase TCP throughput. Also, adapting the changes to Wi-Fi networks.

This thesis discusses about TCP sender side modifications only. Although it requires coordination from both sides, another way forward would be to incorporate changes at both sender side and receiver side to handle packet loss or delay in accordance with the situation rather than blind congestion control. Modifications to intermediate proxies would also be a good way ahead as it requires minimal changes to sender or receiver.

During our simulation it was also observed that as the number of nodes involved increased, the throughput reduced significantly due to link layer contention. Our future study will also be directed to improve TCP throughput in accordance with link layer contention.

References

[1] M. Allman, V. Paxson, W. Stevens, “TCP Congestion Control, RFC2581”, April 1999.

[2] H. Balakrishnan, V. N. Padmanabhan, R. H. Katz, “The Effects of Asymmetry on TCP Performance”, in Proc. of ACM/IEEE Mobicom, pp. 77–89, 1997.

[3] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, "A Feedback-based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks", in Proc. of 18th Int. Conference on Distributed Computing Systems, pp. 472 479, 1998.

[4] B. Chen, I. Marsic, R. Miller, “Issues and Improvements in TCP Performance over Multihop Wireless Networks”, IEEE Sarnoff Symposium, 2008.

[5] C. Chen, H. Wang, XinWang, M. Li, A.O. Lim, “A Novel Receiver-aided Scheme for Improving TCP Performance in Multihop Wireless Networks”, in Proc. of Int. Conference on Communications and Mobile Computing, pp. 272–277, 2009.

[6] C.F. Chiasserini, M. Garetto, M. Meo, “Improving TCP over wireless by selectively protecting packet transmissions”, in Proc. of 4th Int. Workshop on Mobile and Wireless Communications Network, pp. 316–319, 2002.

[7] T. Clausen, P. Jacquet, "Optimized Link State Routing Protocol", RFC 3626.

[8] DSR, "Dynamic Source Routing Protocol", IETF MANET Working Group Inter Draft.

[9] S. M. Elrakabawy, A. Klemm, and C. Lindemann, “TCP with adaptive pacing for multihop wireless networks,” in Proc. of 6th ACM Int. Symp. Mobile Ad Hoc Networking & Computing, pp. 288–299, 2005.

[10] M.O. Farooq, G.A. Shah, “A reactive QoS routing protocol for MANET”, Ad-Hoc and Sensor Wireless Networks, vol. 13, pp. 13-38, 2011.

[11] S. Floyd, T. Henderson, A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782.

[12] B. Francis, V. Narasimhan, A. Nayak, I. Stojmenovic, “Techniques for Enhancing TCP Performance in Wireless Networks”, in Proc. of 9th Workshop on Wireless Ad hoc and Sensor Networks (in 32nd Int. Conference on Distributed Computing Systems), 2012.

[13] B. Francis, V. Narasimhan, A. Nayak, “Enhancing TCP Congestion Control for Improved Performance in Wireless Networks”, in Proc. of 11th Int. Conference on Ad Hoc Networks and Wireless, pp. 472-483, 2012.

[14] X. Guo, J. Liu, "Path diversified retransmission for TCP over wireless mesh networks", in Proc. of 18th Int. Workshop on Quality of Service (IWQoS), pp. 1-9, 2010.

[15] E. Hamadani, V. Rakocevic, “TCP Contention Control: A Cross Layer Approach to Improve TCP Performance in Multihop Ad hoc Networks”, Wired/Wireless Internet Communications, 2007.

[16] M.M.Hassani, M. Mohammadi, S. Pashforoush, “A Novel Recursive Method for Analysis Performance of TCP Flow in Wireless LAN”, in Proc. of 2nd Int. Conference on Communication Software and Networks, pp. 387–391, 2010.

[17] S. Henna, “A Throughput Analysis of TCP Variants in Mobile Wireless Networks”, in Proc. of 3rd International Conference on Next Generation Mobile Applications, Services and Technologies, pp. 279–284, 2009.

[18] http://www.atrexx.com/satellite-networks/IP-based-Satellite-Networks/, Accessed in July 2012.

[19] http://www.gumuskaya.com/images/WirelessLab.jpg, Accessed in July 2012.

[20] N. Itoh, M. Yamamoto, “Proxy-based TCP with Adaptive Rate Control and Intentional Flow Control in Ad Hoc Networks”, IEEE Global Telecommunications Conference, 2008.

[21] J. Jun, M.L. Sichitiu, "Fairness and QoS in Multihop Wireless Networks", IEEE 58th Vehicular Technology Conference, pp. 2936–2940, 2003.

[22] C. Lai, K. Leung, and V.O.K. Li, “Enhancing Wireless TCP: A Serialized Timer Approach”, IEEE INFOCOM, 2010.

[23] T. V. Lakshman, U. Madhow, “The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss”, ACM/IEEE Trans. on Networking, pp. 336–50, 1997.

[24] K. Li, “Design and performance evaluation of communication algorithms in multihop wireless networks with multiple channels”, Int. Journal of Parallel, Emergent and Distributed Systems, vol. 25, issue 6, pp. 465–488, 2010.

[25] N. Liu, C. Li, S. Hsieh, C. Tsai, M. Shao, “Long-term audio observation by wireless sensor networks with filtering strategies”, Ad-Hoc and Sensor Wireless Networks, vol. 12, pp. 151–167, 2011.

[26] W. Long , W. Zhenkai , “Performance Analysis of Improved TCP over Wireless Networks”, in Proc. of 2nd Int. Conference on Computer Modeling and Simulation, pp. 239–242, 2010.

[27] M. Marina, S. Das, “Ad-hoc On-demand Multipath Distance Vector Routing”, ACM SIGMOBILE Mobile Computing and Communications Review Special Feature on the First AODV Next Generation Workshop, 2002.

[28] D. Murray, T. Koziniec, M. Dixon, "D-Proxy - Reliability in Wireless Networks", in Proc. of 16th Asia Pacific Conference on Communications, pp. 129–134, 2010.

[29] K. Nahm, A. Helmy, and C. J. Kuo, “Cross-layer interaction of TCP and ad hoc routing protocols in multihop IEEE 802.11 networks,” IEEE Trans. on Mobile Computing, vol. 7, pp. 458–469, 2008.

[30] Y.M.Omar, H.I. Sigiuk, H.B. Salem, T.T. Hamdan, "Improving TCP Throughput in Wireless Multi-hop ad hoc networks", in Proc. of 3rd Int. Conference on Signals, Circuits and Systems, 2009.

[31] S. Papanastasious and M. Ould-Khaoua, “TCP congestion window evolution and spatial reuse in MANETs”, Wireless Communications and Mobile Computing, vol. 4, pp. 669–682, 2004.

[32] M. Park, S. Chung, "Distinguishing the Cause of TCP Retransmission Timeouts in Multi-hop Wireless Networks", in Proc. of 12th IEEE International Conference on High Performance Computing and Communications (HPCC), pp. 329-336, 2010.

[33] C.E. Perkins, P. Bhagwat, “Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers”, ACM SIGCOMM Conference on Communications architectures, protocols and applications, pp. 234-244, 1994.

[34] C.E. Perkins, E.M. Royer, “Ad-hoc On-demand Distance Vector Routing, in Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pp. 90-100, 1999.

[35] C.E. Perkins, E.M. Royer, S. Das, "Ad-hoc On-demand Distance Vector Routing", RFC 3561.

[36] J. Postel, "Transmission Control Protocol", RFC 793.

[37] S. Prasanthi, S. Chung, "An Efficient Algorithm for the Performance of TCP over Multi-hop Wireless Mesh Networks", in Proc. of 7th International Conference on Information Technology, pp. 816–821, 2010.

[38] S. Prasanthi, S. Chung, C. Ahn, "An Enhanced TCP Mechanism for Detecting and Differentiating the Loss of Retransmissions over Wireless Networks", in Proc. of Int. Conference on Advanced Information Networking and Applications, pp. 54–61, 2011.

[39] I.A. Rai, T. Hellen, "On improving the TCP performance in asymmetric wireless mesh networks", in Proc. of Int. Conference on Communications, Computing and Control Applications, 2011.

[40] R. Roy, S. Das, A. Ghosh, and A. Mukherjee, “Modified TCP Congestion Control Algorithm for Throughput Enhancement in Wired-cum-Wireless Networks,” in Proc. of 4th Swedish National Computer Networking Workshop, 2006.

[41] V. Raman, I. Gupta, “Performance trade-offs among percolation-based broadcast protocols in wireless sensor networks”, Int. Journal of Parallel, Emergent and Distributed Systems, vol. 25, issue 6, pp. 509–530, 2010.

[42] S. Sathya Priya, K. Murugan, “Improving TCP Performance over Wireless Networks Using Cross Layer”, Recent trends in Networks and Communications, 2010.

[43] N. Sengottaiyan, R. Somasundaram, S. Arumugam, "An Modified approach for measuring TCP Performance in Wireless Adhoc Network", in Proc. of Int. Conference on Advances in Recent Technologies in Communication and Computing, pp. 267–270, 2010.

[44] T. Shikama, "Mitigation of Bursty Packets by a TCP Proxy improving TCP Performance in a Wired and Wireless Network", IEEE Globecom Workshop on Complex Communication Networks, pp. 425–429, 2010.

[45] P. Sreekumari, S. Chung, W. Kim, "A Timestamp based Detection of fast retransmission loss for improving the performance of TCP NewReno over wireless networks", in Proc. of 7th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 60-67, 2011.

[46] K. Sundaresan, V. Anantharaman, H.Y. Hsieh, R. Sivakumar, "ATP: A Reliable Transport Protocol for Ad-hoc Networks", IEEE Trans. on Mobile Computing, vol. 4, issue 6, pp. 588–603, 2005.

[47] TCP Congestion avoidance algorithm, http://en.wikipedia.org/wiki/TCP-congestion-avoidance-algorithm, Accessed in July, 2012.

[48] Y. Tian, K. Xu, N. AnsariI, “TCP in Wireless Environments: Problems and Solutions”, http://comnet.kaist.ac.kr/yhlee/CN_2008_Spring/readings/Tian05-wirelessTCP.pdf, Accessed in July, 2012.

[49] N. Xu, W. Jia, “Joint Packet Scheduling and Radio Resource Assignment for WiMAX Networks”, Ad-Hoc and Sensor Wireless Networks, vol. 13, pp. 193–208, 2011.

[50] L. Yongmei, J. Zhigang, Z. Ximan, “A New Protocol to Improve Wireless TCP Performance and Its Implementation”, in Proc. of 5th Int. Conference on Wireless Communications, Networking and Mobile Computing, 2009.

[51] Z. Zhizhao, H. Zhengzhi, "Improving TCP performance over wireless links using link-layer retransmission and explicit loss notification", Journal of Systems Engineering and Electronics, pp. 17-23, 2002.

[52] M. Zin-Oo, M. Othman, “The Effect of Packet Losses and Delay on TCP Traffic over Wireless Ad Hoc Networks”, http://www.intechopen.com/books/mobile-ad-hoc-networks-applications/the-effect-of-packet-losses-and-delay-on-tcp-traffic-over-wireless-ad-hoc-networks, Accessed in July 2012.