ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY
DEPARTMENT OF
ELECTRICAL AND COMPUTER ENGINEERING

Performance Evaluation of TCP over Multi-path Routing Protocol using TCP-DOOR-TS algorithm in Mobile ad hoc Networks

A thesis submitted to the school of graduate studies of Addis Ababa University in partial fulfillment of the requirements for the degree of
Masters of Science in Computer Engineering

By
Henock Mulugeta

Advisor: Prof.Dr. Sayed Nouh

January 2008

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

"Performance Evaluation of TCP over Multi-path Routing Protocol using TCP-DOOR-TS algorithm in Mobile ad hoc Networks"

By

Henock Mulugeta

FACULTY OF TECHNOLOGY

APPROVAL BY BOARD OF EXAMINERS

Chairman Department of Graduate Committee	Signature
Prof.Dr. Sayed Nouh	
Advisor	Signature
Dr.Kumudhal Raymond	
Examiner	Signature
Dr. Manoj V.N.V.	
External Examiner	Signature

Acknowledgement

I would like to thank my advisor Prof.Dr.Sayed Nouh for his guidance and encouragement. He gave me all his knowledge, time, patience and advices.

I would also like to thank my family especially my father Mulugeta and my mother Geni, for their endless support and motivation. They were by my side by giving me extreme support throughout my graduate studies. With out them, it was impossible.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

ACK	Acknowledgment
AODV	Ad Hoc On-Demand Distance Vector
ARQ	Automatic Repeat Request
ATCP	Ad hoc TCP
AWND	Advertised Window
CWND	Congestion Window
DAG	Direct Acyclic Graph
DSDV	Destination-Sequenced Distance-Vector
DUPACKs	Duplicate Acknowledgments
ECN	Explicit Congestion Notification
FIN	Finish
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
MAC	Medium Access Control
MANETs	Mobile Ad Hoc Networks
NS- 2	Network Simulator 2
OOO	Out Of Order
OSI	Open System Interconnection
PSH	Push
QRY	Query
RRN	Route Re-establishment Notification
RST	Reset
RTO	Retransmission Time Out
RTT	Round Trip Time
SACK	Selective Acknowledgment
SMTP	Simple Mail Transfer Protocol
SRTT	Smoothed Round Trip Time
SSTHRESH	Slow Start Threshold

SYN	Synchronous
TCP	Transmission Control Protocol
TCP-DOOR-TS	TCP Detection of Out of Order and Response with Time Stamp
TCP-F	TCP Feed back
TORA	Temporally Ordered Routing Algorithm
UDP	User Datagram Protocol
URG	Urgent
UPD	Update

Abstract

Mobile ad hoc networking is a concept in computer communications, which means that if users want to communicate with each other, they can form a temporary network with out any pre-existing communication infrastructure. Each node participating in the networks acts as host and router in order to transmit, receive and forward packets.

In mobile ad hoc networks [MANETs], the wireless mobile nodes may dynamically enter the network as well as leave the network. Due to the dynamic nature of MANET, nodes are typically distinguished by their high degree of mobility and route failure due to frequent link breakages, which affects the end-to-end transmission of data. In order to increase the reliability of data transmission and to reduce delay due to route re-computation, some routing protocols, such as Temporally Ordered Routing Algorithm (TORA), maintain multiple routes between a sender-receiver pair, and use multi-path route to transmit packets. In such a case, packets coming from different paths may not arrive at the receiver in order. Since TCP receiver is not aware of multi-path route, it would misinterpret such out-of-order packet arrivals as a sign of network congestion. TCP performs poorly in this environment, because TCP was previously designed for reliable wired network in which all packet losses are assumed to be due to network congestion.

In this thesis by implementing TCP Detection of Out-of-Order and Response with Time Stamp (TCP-DOOR-TS) algorithm, the performance degradation of TCP has been overcome, and it is confirmed that out-of-order packet is really delivered due to multi-path routing protocol. The

simulation results have shown that, TCP-DOOR-TS algorithm has a higher throughput and good put than TCP-SACK. For two different simulation scenarios a throughput of 13 % and 7.45 % and a good put of 12.7% and 7.45 % improvements have been achieved.

Keywords: Mobile ad hoc networks, TCP-SACK, TCP-DOOR-TS, out-of-order packet.

CHAPTER ONE

INTRODUCTION

1.1 Motivation

Mobile ad hoc networks (MANETs) [1] can be represented as a complex distributed systems, which comprises of wireless mobile nodes, and the nodes are freely and dynamically move and self-organize within the given topology. With the above scenario they form random, and temporary "ad-hoc" network topologies. By doing so they allow devices to spontaneously internetwork in areas with no pre-existing communication infrastructure.

Now a days Mobile ad hoc networks have become very much important. Because they can be implemented in infractureless connection with out the need of traditional fixed infrastructure networks. Such networks are mainly consisting of highly mobile nodes, which are mobile with in specific covering area and can be connected dynamically in an arbitrary manner. They have provided new challenges, which is the result of the unique characteristics of the wireless medium and the dynamic nature of the network topology.

The followings are some of the characteristics, complexities, and design constraints, which are specific to MANET [1]:

• *Autonomous and infrastructure-less:* MANET does not depend on any established infrastructure or centralized administration.

• *Multi-hop routing:* No default router is available; every node works as a router and forwards each other's packets to provide information sharing between mobile hosts.

• *Dynamically changing network topologies:* In mobile ad hoc networks, nodes can move randomly. Due to the random movement of nodes the network topology, which is typically multi-hop, can change frequently and unpredictably, resulting in: -

1- Route changes
2- Frequent network partitions and
3- Possibly packet losses.

The TCP protocol, which was previously developed for reliable end-to-end delivery of data over unreliable wired networks, by ignoring the properties of wireless Ad Hoc Networks, can lead to TCP implementations with poor performance.

In MANETs, the major problem of TCP lies in performing congestion control in case of losses that are not caused by network congestion. Since bit error rates are very low in wired networks, all TCP versions assume that packet losses are due to network congestion. Consequently, when a packet is detected to be lost, either by timeout or by multiple duplicated ACKs, TCP slows down the sending rate by adjusting its congestion window size. When TCP is implemented in MANETs, since out of order delivery of packets are considered to be loss of packet, without the network becomes congested, TCP reduces the congestion window size, which degrades TCP's performance a lot.

The performance of TCP degrades in Ad hoc networks. This is because TCP faces new challenges due to several reasons, which are specific to MANETs: route failures, network partitions, dynamic network topology, and power constraints [2].

• *Route failures:* In wired networks route failures occur very rarely. However in MANETs most of the time route will be failed. The main causes of route failures are node mobility and link failures. The route reestablishment duration after route failure in Ad hoc networks depends on the underlying routing protocol, mobility pattern of mobile nodes, and traffic characteristics.

• *Network partition:* Figure 1.1 shows network partition scenarios, Ad hoc network can be represented by a simple graph G. Mobile stations are the "vertices". A successful transmission between two stations is an undirected "edge". Network partition happens when G is disconnected. The main reason of this disconnection in MANETs is node mobility. Another factor that can produce network partition is energy-constrained operation of nodes [2].

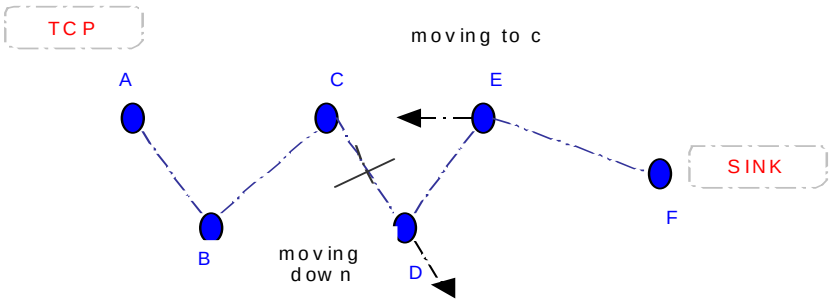


Figure 1.1 Network partition scenario

When D is moving away from C, the network will be partitioned. The network is reconnected when E is moving toward C.

• *Dynamic network topology:* The dynamic nature of the network topology Changes frequently due to movement of nodes.

• *Power constraints:* Because batteries carried by each mobile node have limited power supply, processing power is limited.

All the aforementioned problems contribute to the random nature of packet loss in wireless networks. TCP was designed for conventional wired networks, which doesn't have any packet loss unless the network is congested, so TCP assumes that all packet loss is due to congestion.

But mobile Ad hoc networks produce some challenges to TCP due to the fact that previously it was not designed to operate in a highly dynamic scenario in terms of topology. Indeed, in mobile Ad hoc networks, packet losses are not always due to network congestion, as it is mostly the case in wired networks.

Due to mobility of nodes route failure may occur, hence packets coming from TCP sender may not arrive in order at the TCP receiver, which means routes in MANETs, are short-lived due to frequent link breakages. To reduce delay due to route re-computation and node mobility, some routing protocols such as Temporally Ordered Routing Algorithm (TORA) maintain multiple routes between a sender-receiver pair. In such a case, packets coming from different paths may not arrive at the receiver in order. Being unaware of multi-path routing, TCP receiver would consider such out-of-order packet arrivals as a sign

of congestion. The receiver will thus generate duplicate ACKs that cause the sender to invoke
congestion control algorithms like fast retransmission (upon reception of three duplicate ACKs), which degrades TCP's performance a lot.

In this thesis, we have evaluated the performance of TCP in mobile ad hoc networks by implementing TCP Detection of Out of Order and Response with Time Stamp (TCP-DOOR-TS) algorithm by using multi-path routing protocol TORA and compare the result with that of conventional TCP Selective Acknowledgment "TCP-SACK".

1.2 Objective

The objective of this thesis is to evaluate the performance of TCP over multi-path routing protocol such as Temporally Ordered Routing Algorithm (TORA), using TCP-DOOR-TS algorithm in mobile ad hoc networks.

The specific objective of this thesis is:-

- To evaluate the performance of TCP-SACK over multi-path routing protocol (TORA).

- TCP-SACK is modified into TCP-DOOR-TS, both at the receiver and sender sides.

- Performance evaluation of TCP-DOOR-TS algorithm over TORA is performed.

- TCP-DOOR-TS is evaluated for two different scenarios:-

i- Disabling period T=RTT

ii-Disabling period T=2XRTT

Assumption: - we assume that the network is not congested so as to identify whether out-of-order packet is really delivered or not due to multi-path routing protocol like TORA. If out-of-order packet is delivered, it should be detected and responded accordingly in order to overcome the degradation of the performance of TCP due to the effect presented above.

TCP-DOOR-TS algorithm also deals with packet loss in the network. From TCP version, TCP-SACK has been taken for the entire simulation.

1.3 Contribution

In this thesis, we have proposed TCP-DOOR-TS algorithm. It is an end-to-end approach implemented in TCP-SACK in order to improve the performance of TCP by detecting and responding for out-of-order packet delivery due to multi-path routing protocol like Temporally Ordered Routing Algorithm (TORA). We have used Time Stamp option for detecting mechanism. TCP-DOOR-TS is mainly worked with detection of out-of-order packet and packet loss in the network; it provides mechanism for responding for the detected and lost packets.

TCP-DOOR-TS algorithm is an extension of TCP-SACK. We have used NS-2 to implement the algorithm and evaluate its performance. The implemented algorithm was compared with that of one of TCP's version known as TCP-SACK.

1.4 Outline of the thesis

In chapter 2, we present the background materials that have been used. The Transmission Control protocol (TCP), Selective Acknowledgment (SACK), and Temporally Ordered Routing Algorithm (TORA) have been discussed. Different approaches for improving TCP's performance in mobile ad hoc networks, which were done previously and comparison of the surveyed literatures, are discussed in chapter 3.In chapter 4, we describe the TCP detection of Out-of-Order and Response with Time Stamp algorithm in detail and comprehensively. First detection of out-of-order packets and then the response mechanism will be discussed. Implementation of TCP-DOOR-TS algorithm using network simulator version 2.29 is discussed in chapter 5.In chapter 6, the performance evaluation of TCP-DOOR-TS will be presented, first the simulation scenarios and model will be presented, then the traffic and mobility models, Performance Metrics, Simulation code, parsing the Simulation trace files will be discussed. Finally, Performance evaluation results and analysis will be presented in detail. In chapter 7, conclusion and future work will be presented.

CHAPTER TWO

BACKGROUND

In this chapter, we present the background materials for implementing TCP-DOOR-TS algorithm. TCP-DOOR-TS is implemented both at the sender and receiver side of TCP-SACK. First, TCP Algorithm and Selective Acknowledgement (SACK) are introduced, and then TORA will be discussed.

2.1 The Transmission Control Protocol

Transmission Control Protocol (TCP) is a connection oriented point-to-point protocol. It is a means for building a reliable communications stream on the top of the unreliable Internet Protocol (IP). TCP is the protocol that supports nearly all Internet applications.

2.1.1 TCP in the Internet protocol stack

Figure 2.1 shows the structure of the Internet protocol stack, in which the TCP/IP is composed of the Network (IP) layer and Transport (TCP) layer. Each one of the four layers is responsible for a specific purpose so as to communicate hosts with each other, when we say hosts it may be computers, or processes with in a computer. Internet protocol stack is redesigned from the previously used OSI reference model [4].

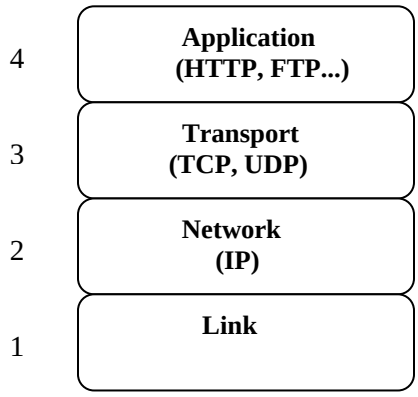


Figure 2.1 Internet Protocol stack

The application layer is responsible for producing and consuming the user's data that pass through each layer stack and is transferred across
the network. In this layer are running the most common application programs such as, Internet Explorer, File Transfer, E-mail, and World Wide Web (w.w.w). Each one of the application programs has a part in it that resides in the application layer; this part is called application layer entity, for example:

- *Internet Explorer:* It uses HTTP protocol of the application layer for exchanging messages.

- *File Transfer:* It uses File Transport Protocol (FTP).

- *E-mail:* It uses Simple Mail Transfer Protocol (SMTP) to send messages.

- *World Wide Web (WWW):* It uses the application protocols mentioned above with some additional components.

The application layer and transport layer communicate with each other by using ports and sockets, which is each application program has its own logical port.

The transport layer is responsible for the end-to-end transmission of the data created by the application layer. The most widely used transport protocols are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). UDP provides a non-reliable data delivery over IP, because it is connectionless transport. Since TCP is connection oriented, it guarantees end-to-end data delivery for the packets routed by the network layer. Generally, every application is associated with a particular port number in the transport protocols. This association allows multiple applications to share the same transport protocol between two hosts.

The network layer is mainly work with routing of packets between sender and receiver hosts. For this purpose, every node in the global Internet is having a unique IP address, which is helpful for uniquely identifying a specific host. The Network layer supports different routing protocol like Dynamic Source Routing Protocol (DSR), Temporally Ordered Routing Algorithm (TORA).

The link layer (also called data-link layer or network interface layer) specifies how the packets of the network layer are transported over the physical medium connecting two nodes. The link layer uses Address Resolution Protocol (ARP) in order to resolve the IP address of the specific host in to hardware (MAC) conversion. This layer deals with all physical transmission details such as frame size, synchronization, frequency, etc.

The main characteristics and mechanisms of TCP are discussed as follows: -

2.1.2 Connection setup

In TCP, the two hosts (sender host and receiver host) that want to communicate with each other for a certain period of time, first they handshake with each other. Handshaking consists of three phases. The sender first sends a special TCP segment (only the TCP header and IP header) to the receiver; at this point there is no need to send data. Receiver acknowledges and sends another special segment. Finally, the sender acknowledges the special segment from the receiver. Figure 2.2 shows the three phases of handshaking. The sender host passes data through sockets, and then TCP directs these data to the send buffer. TCP takes a block of data from the send buffer. The maximum amount of the block of data is limited by the Maximum Segment Size (MSS). TCP encapsulates each block of sender's process data with TCP header and forms a TCP segment. When TCP receives a segment, the segment's data is placed in the receive buffer of the connection. The application reads data from this buffer. A TCP connection consists of buffers, variables and a socket connection to a process in one host and another set of buffers, variables and socket connection to a process in another host. No buffers or variables are allocated to the connection in the network elements (such as routers) between two hosts. If a certain time, called timeout, has passed without acknowledgement, a new connection request is sent [5].

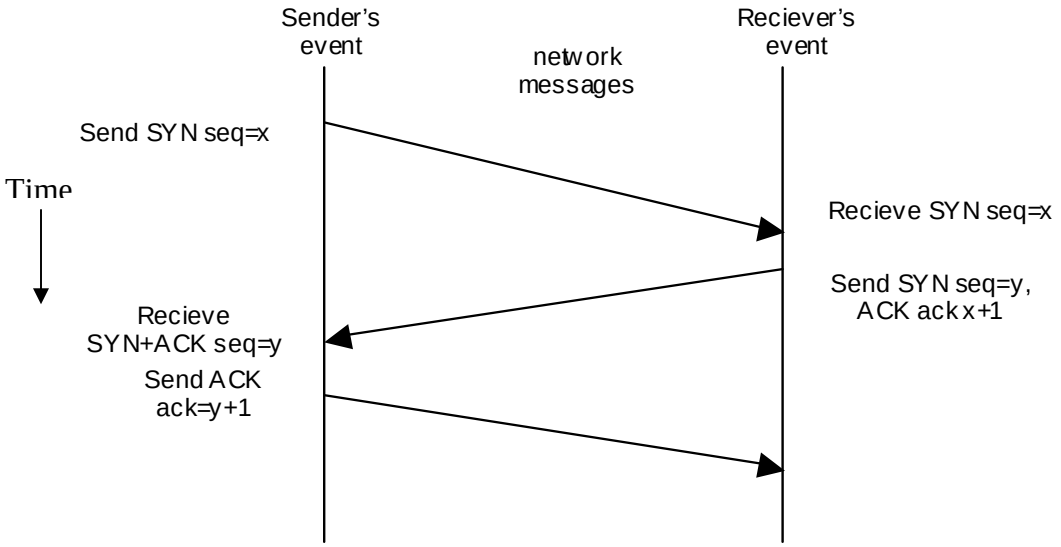


Figure 2.2 Three phases of handshaking between the TCP sender and receiver

TCP views data as ordered stream of bytes. The sequence number of a segment is the byte-stream number of the first byte of the segment. The acknowledgement number that one host puts in its segment is the sequence number of the next byte that host is expecting from another host. TCP only acknowledges bytes up to the first missing byte in the data stream; this is known as cumulative acknowledgement. If a host receives out of order segments, TCP either discards out of order bytes or keeps them and waits for the missing bytes to fill in the gaps.

While TCP uses three segments to initiate the connection, it needs four segments to terminate the connection. When the sender host wants to end connection, it sends a FIN segment to the receiver host. The receiver host has to acknowledge the FIN segment, but can still send data to the sender host. This is known as half-close state of TCP connection. When the receiver host decides to close the connection, it

sends a FIN segment. Finally, when the sender acknowledges the FIN segment the connection is completely closed. Figure 2.3 shows TCP connection termination phases.

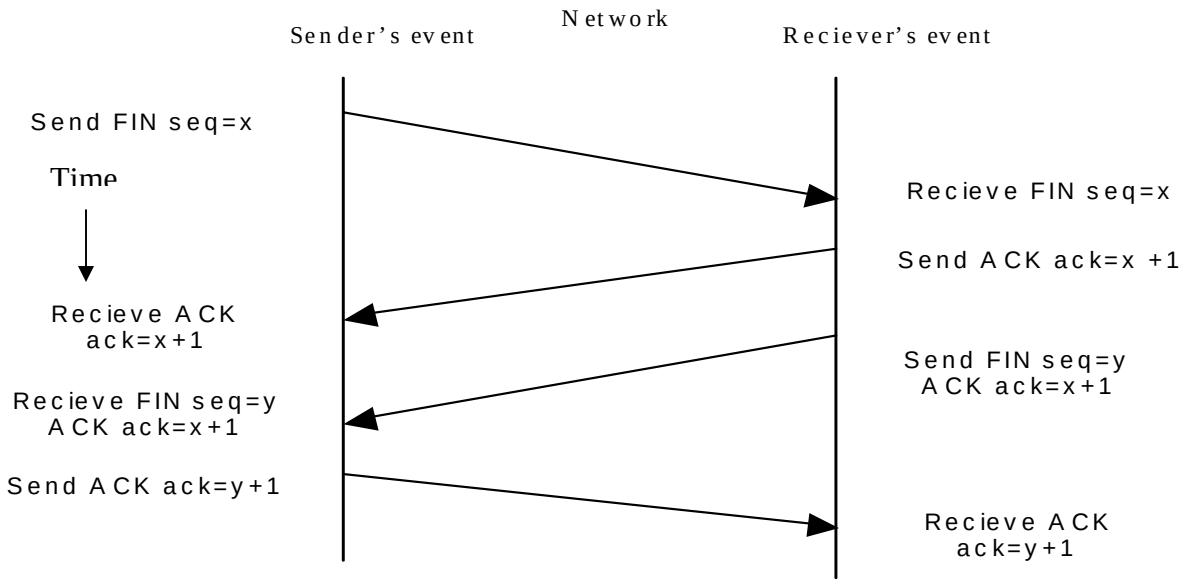


Figure 2.3 TCP connection termination

2.1.3 Flow and congestion control

TCP is a window-based flow and congestion control protocol, which uses a sliding window mechanism to manage its data transmission. It uses the sliding window mechanism, which sends as many segments as the receiving end can handle, before having to wait for acknowledgments. In this mechanism, a window of segments (certain number of segments) is transmitted at once. Each segment has a sequence number. The receiver can acknowledge more than one segment at a time by acknowledging the highest one received, meaning that all the previous segments were successfully transmitted. A field

called the advertised window (AWND), which is controlled by TCP receiver and found in the TCP header, is used to inform the sender of the receiving buffer size. The sending window is limited by the AWND, so that a fast sender does not force a slow receiver. Generally speaking, the purpose of this arrangement is to guarantee that the sender adjusts its transmission rate to meet both sender and receiver needs. Thus, the TCP sender contains a variable denoted window determining the amount of packets it can send into the network before receiving an ACK. This variable changes dynamically over time to properly limit the connection's sending rate.

The sending rate of a TCP connection is regulated by two distinct mechanisms, the *flow control* and the *congestion control*. Although these mechanisms are similar, in the sense that both attempt to prevent the connection from sending at an excessive rate, they have specific purposes.

Flow control is implemented to avoid that a TCP sender don't overflows the receiver's buffer. Thus, the receiver advertises in every ACK transmitted a window limit to the sender. This window is named receiver advertised window (AWND) and changes over time depending on both the traffic conditions and the application speed in reading the receiver's buffer. Therefore, the sender may not increase its window at any time beyond the value specified in AWND.

In contrast to flow control, *congestion control* is concerned with the traffic inside the network. Its purpose is to prevent collapse inside the network when the traffic source (sender) is faster than the network in forwarding data. To this end, the TCP sender also uses a limiting window called congestion window (CWND). Assuming that the receiver

is not limiting the sender, CWND defines the amount of data the sender may send into the network before an ACK is received.

2.1.4 Connection control

If the sender is only limited by the AWND, many packets can be dropped because of a full buffer in an intermediate router. Therefore, not only the receiver buffer should not limit the sending window, but also by the network capacity. The window size resulting from congestion control is called the congestion window (CWND). The sending window size of the sender, W, is taken as being the minimum of the AWND and the CWND.

W = min (CWND, AWND),

Instead of being equal to AWND. The congestion window can be thought of as being a counterpart to advertised window. Whereas AWND is used to prevent the sender from overrunning the resources of the receiver, the purpose of CWND is to prevent the sender from sending more data than the network can accommodate in the current load conditions. The idea of modifying CWND is to reflect the current load of the network. In practice, this is done through detection of lost packets. A packet loss can basically be detected either by a time-out mechanism or by duplicate ACKs.

If a packet is lost, TCP retransmits it (and all the following packets) through its Automatic Repeat Request (ARQ) mechanism. This kind of ARQ is called Go-Back-N [5].

After the occurrence of congestion collapse in the Internet in the late 1980's it was realized that there should be a congestion control

mechanism, which would be required to prevent the TCP senders from overrunning the resources of the network. In 1988, Tahoe TCP was released including three congestion control algorithms operate together: slow start, congestion avoidance and fast retransmit. In 1990 Reno TCP, providing one more algorithm called fast recovery was released.

Currently TCP's connection management has four phases: slow start, congestion avoidance, fast retransmit and fast recovery [6].

Slow start: The way in which TCP data transmission operates during the start of a connection is known as slow start. The slow start algorithm avoids the congestion problem by observing that the rate at which new packets should be transmitted in the network is the same rate at which the acknowledgments are returned by the other end. The sender starts by transmitting one segment and waiting for its ACK. Afterwards, CWND is doubled each time an ACK is received. The main drawback of slow start is the large amount of time that is required during start up. If the data that is being sent is very small, the bandwidth efficiency will be reduced considerably.

Congestion avoidance: During The slow start phase the congestion window increases exponentially. At some point during the connection, a bottleneck in the network might be occurred and will start discarding packets [6]. Therefore, beyond a certain threshold, there is no need to increase the congestion window exponentially, rather linearly. The relation between slow start and congestion avoidance is done through a variable called slow start threshold (SSTHRESH). If CWND is smaller than SSTHRESH, the TCP sender is in slow start phase; otherwise it is in congestion avoidance phase, meaning that CWND is increased only by 1/CWND each time an ACK is received. This is an additive increase, shown in Figure 2.4 [7].

Whether the network is wired or wireless, TCP assumes that almost all packet losses are due to network congestion. Therefore, The TCP sender reduces the amount of packets that are sent to the TCP receiver, which means congestion window size should be reduced, if a packet loss is detected. Retransmission timeout (RTO) or up on the reception of three duplicate ACKs indicates packet loss. When congestion occurs, SSTHRESH is set to half of the current congestion window size.

Figure 2.4 [7] shows as an illustration of how the congestion algorithm works; the maximum segment size here is 1024 bytes. Initially, the congestion window size is 1 KB for transmission 0 here. The congestion window then grows exponentially until it reaches the threshold (16KB). Starting then, it grows linearly. Because it is in congestion avoidance phase. At transmission number 11, a time out occurs. The threshold is set to half the current window (by now 24 KB, so half is 12 KB), and slow start is initiated. When the acknowledgements from transmission 12 start coming in, the first two each double the congestion window, but after that, growth becomes linear again.

TCP responds in different ways when congestion is detected in the network either by: -

1- Retransmission Time Out (RTO).
2- Three duplicate ACKs.

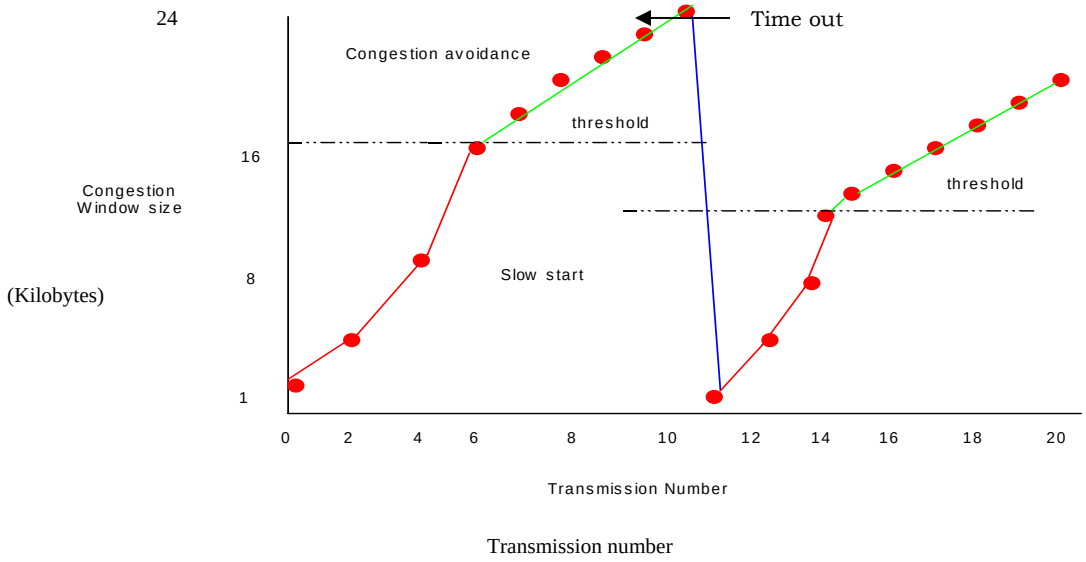


Figure 2.4 TCP connection: slow start, congestion avoidance, and timeout

2.1.5 TCP retransmission timeout (RTO)

Associated with each packet is a timer. When the acknowledgment of the data segments are not received and the RTO expires, TCP retransmits the segments, and enters back to slow start phase, and starts calculating RTO.Expiration of RTO implies that the network is severely congested, so it should start by sending one packet. Since the time between a packet sent and its ACK arrives (known as the Round Trip Time RTT) may vary depending on the network, RTO cannot have a fixed value. A TCP sender maintains two state variables for computing RTO, the smoothed roundtrip time (SRTT) and the round-trip time variation (RTTVAR). Additionally, a clock granularity of G seconds is assumed in the computation. The computation of SRTT, RTTVAR and RTO are as follows [4].

1. Until an RTT measurement has been made for a packet sent between sender and receiver, the sender should set RTO to three seconds.

2. When the first RTT measurement R is made, the sender must set:

$$SRTT = R, \text{ and } RTTVAR = R/2 \quad (2.1)$$

$$RTO = SRTT + \max(G, K \cdot RTTVAR), \text{ where } K = 4 \quad (2.2)$$

3. When a subsequent RTT measurement R' is made, the sender must update the variables as follows:

$$RTTVAR = (1 - \beta) \cdot RTTVAR + \beta \cdot |SRTT - R'| \quad (2.3)$$

$$SRTT = (1 - \alpha) \cdot SRTT + \alpha \cdot R' \quad (2.4)$$

α and β are normally set to 1/8 and 1/4, respectively

After the computation, the RTO must be updated:

$$RTO = SRTT + \max(G, K \cdot RTTVAR) \quad (2.5)$$

When a timeout occurs, the RTO is doubled, with a maximum of 64 seconds.

2.1.6 TCP Duplicate acknowledgment

If a packet has been lost, the receiver keeps sending acknowledgements asking about this lost packet, and does not modify the sequence number field in the ACK packets. When TCP sender receives three duplicate acknowledgments for this lost packet from the TCP receiver, the TCP algorithm at the sender side responds accordingly in the following ways: -

Fast Retransmit: Whenever the RTO event has happened, the implication behind it is that, there is no possibility to transmit new packets through the network due to the fact that the network is severely congested. However, in the case for which RTO doesn't expire and TCP sender receives three duplicate ACKs, it means that a packet was lost, but other packets have already reached at the receiver side. In this case, TCP sender will retransmit the lost packets without waiting for the expiration of RTO [6].

Fast Recovery: After fast retransmission, TCP sender performs fast recovery, which means, after reception of the third duplicate acknowledgment, TCP sender enters into congestion avoidance phase in which the window size is increased linearly instead of entering into slow start phase where the congestion window size increments exponentially from the beginning. SSTHRESH is set to one-half the current window size, but CWND will be set at SSTHRESH plus three (because of the three duplicate ACKs received) [6]. Each time another duplicate ACK arrives, CWND is incremented by one. When a new ACK (acknowledging new data) arrives, TCP enters into congestion avoidance phase.

2.1.7 TCP header

The TCP sender and receiver share some common information (such as, sequence number, acknowledgment number, etc...). This information is sent within a header appended to each TCP segment. The standard size of the TCP header is 20 bytes, but some protocols use 20 more bytes for TCP options as shown in Figure 2.5 [7]. In order to be reliable, the two hosts using TCP must be aware of the connection and should be synchronized with each other.

Source Port and Destination Port: These fields identify the sending and receiving applications. This allows different TCP applications and protocols such as FTP and HTTP to establish parallel connections between two particular hosts. These two port values combined with the source and destination fields in the IP header uniquely identify each connection.

Sequence Number: Each sent segment includes a sequence number, which increases monotonically as a function of the number of bytes transmitted. This permits sequential data delivery, which is needed for managing retransmissions.

Acknowledgment Number: Contains the expected sequence number the sending host is asking for from the receiver host.

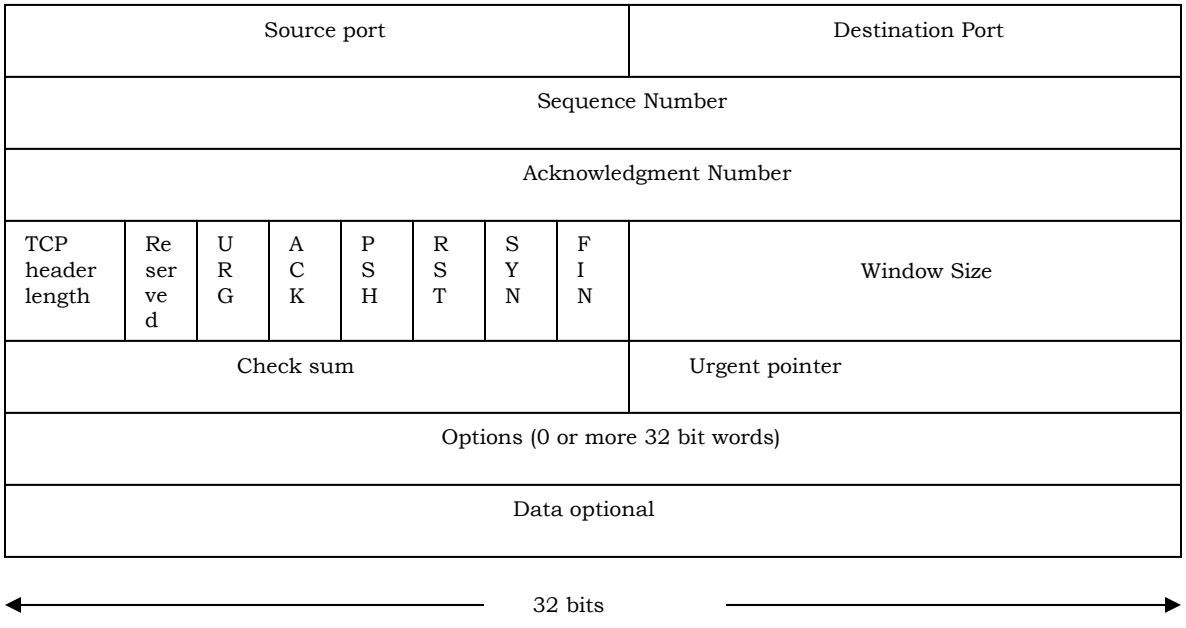


Figure 2.5 TCP header

Header Length: Indicates the length of the header. This is necessary because the length of the Options field is variable. A TCP without any option has a header size of 20 bytes

Reserved: Reserved and always set to zero.

Flags: There are six flags defining the nature of the header, as follows:

URG: This specifies that the Urgent Pointer in this header is valid.

ACK: The acknowledgment number in this header is valid.

PSH: Requires receiver to pass this data to the application as soon as possible.

RST: Resets the connection.

SYN: Synchronizes data and ACK sequence numbers to initiate a connection.

FIN: Indicates that the sender is finished transmitting data.

Window: This field contains the receiver window, which defines the number of bytes the TCP receiver is willing to accept from the sender. This provides a connection flow control governed by the receiver side.

Checksum: It is calculated by the sender considering not only the header but also the data field. The receiver may check the data integrity by checking this field.

Urgent Pointer: It is valid only if the URG flag is set. This field specifies a part of the data that must be sent quickly to the receiver.

Options: This field carries possible options such as the maximum segment size (MSS), timestamps, etc.

2.2 TCP Selective Acknowledgment (TCP-SACK)

TCP-SACK (Selective Acknowledgment) [8] preserves the basic principles of the above TCP functionalities. The characteristics in TCP-SACK lie in its behavior when multiple packets are dropped from one window of data. In SACK, the receiver uses the option fields of TCP header (SACK option) for notifying the sender of up to usually three blocks of non-contiguous set of data received and queued by the receiver. The first block reports the most recent packet received at the receiver, and the next blocks repeat the most recently reported SACK blocks. The sender keeps a data structure known as scoreboard in order to provide information about SACK options (blocks) received so far. In this way, the sender can conclude that whether there are missing packets at the receiver. If so, and its congestion window permits, the sender retransmits the next packet from its list of missing packets. In case there are no such packets at the receiver and the congestion window allows, the sender simply transmits a new packet.

As per the previous discussion, fast retransmission and fast recovery can only handle one packet loss from one window of data with in one transmission time out period; TCP may experience poor performance when multiple packets are lost in one window. To overcome this limitation, recently the Selective Acknowledgement option (SACK) is suggested as an addition to the standard TCP implementation. In the

event of multiple losses within a window, the sender can conclude that which packets have been lost and should be retransmitted using the information provided in the SACK blocks. A SACK-enabled sender can retransmit multiple lost packets in one RTT instead of detecting only one lost packet in each RTT.

The SACK implementation also enters fast recovery upon the receipt of generally three duplicate acknowledgments. Then, its sender retransmits
a packet and reduces the congestion window by half. During fast recovery, SACK controls the estimated number of packets outstanding in the path (transmitted but not yet acknowledged) by maintaining a variable called *pipe*. This variable determines if the sender may send a new packet or retransmit an old one, in that the sender may only transmit if pipe is smaller than the congestion window. At every transmission or retransmission, pipe is incremented by one, and it is decremented by one when the sender receives a duplicate ACK packet containing a SACK option informing it that the receiver has received a new data packet.

The fast recovery is over when the sender receives an ACK acknowledging all data that were outstanding when fast recovery was entered. If the sender receives a partial ACK, i.e., an ACK that acknowledges some but not all outstanding data, it does not exit fast recovery. For partial ACKs, the sender reduces pipe by two packets instead of one, which guarantees that a SACK sender never recovers more slowly than it would do if a slow start had been invoked [4].

Generally speaking, Selective Acknowledgment (SACK) is a strategy that corrects the above TCP behavior in the face of multiple dropped segments. With selective acknowledgments, the data receiver can

inform the sender about all segments that have arrived successfully, so the sender needs to retransmit only the segments that have actually been lost.

Even though the TCP selective acknowledgment mechanism can allow a SACK-enabled sender to retransmit in one RTT multiple lost packets in one transmission window and hence avoid continuous timeouts, this mechanism does not distinguish the reasons for packet losses and still assumes that all losses are caused by congestion. As a result, TCP congestion control procedures are inappropriately called for, which affects the sender's transmission rate.

2.3 Temporally Ordered Routing Algorithm

The Temporally Ordered Routing Algorithm (TORA) is a highly adaptive and efficient distributed multi-path routing algorithm based on the concept of link reversal. TORA is proposed for highly dynamic mobile, multihop wireless networks. It is a source-initiated on-demand routing protocol. This means, the source is responsible for initiating for routing packets and the protocol is called whenever there is a need to route packets between sender-receiver pair.

It finds multiple routes from a source node to a destination node. The main feature of TORA is that the control messages are localized to a very small set of nodes near the occurrence of a topological change. To achieve this, the nodes maintain routing information about adjacent nodes. The protocol has three basic functions: Route creation, Route maintenance, and Route erasure [9].

Each node has: –

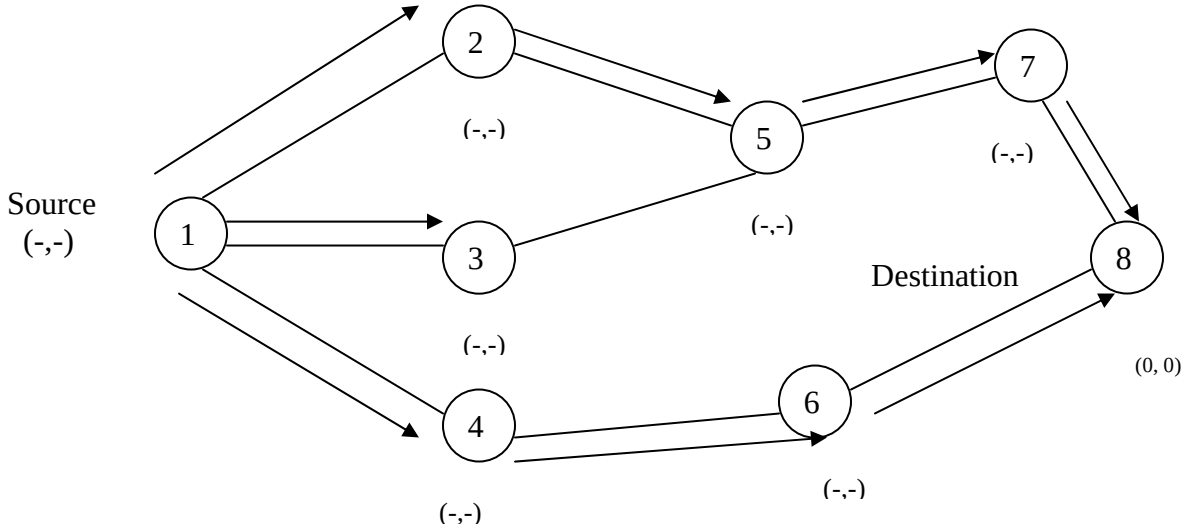
- Logical time of a link failure
- The unique ID of the node that defined the new reference level
- A reflection indicator bit 0= original level, 1=reflected level
- A propagation ordering parameter to order nodes relative to reference level
- The unique ID of the node

The first three elements collectively represent the *reference level*. A new reference level is defined whenever a node loses its last downstream link due to a link failure. The last two values define a *height* with respect to the reference level.

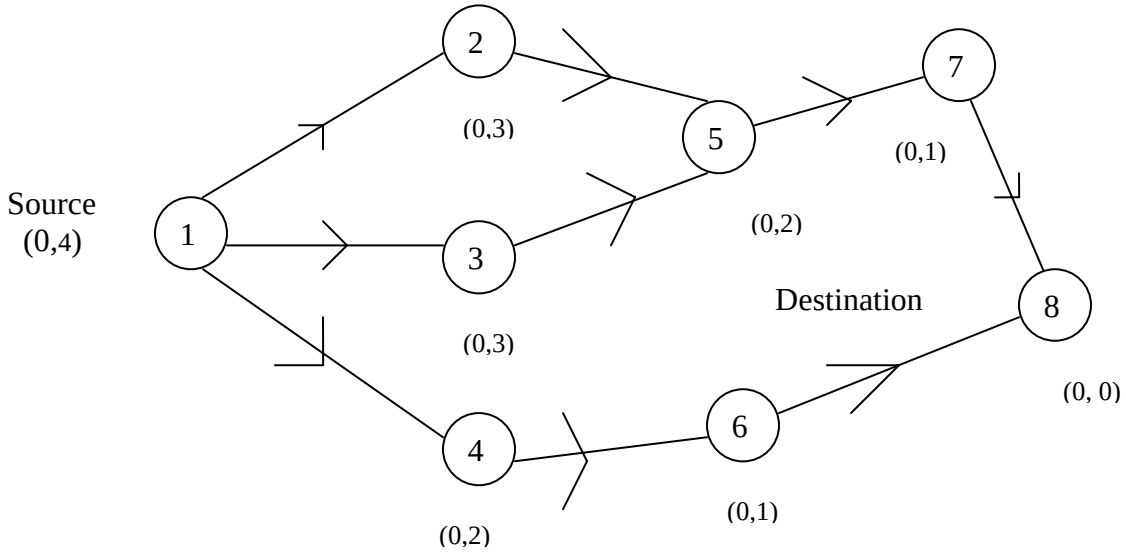
Route Creation is done using Query (QRY) and Update (UPD) packets. The route creation algorithm starts with the height (propagation ordering parameter) of destination set to 0 and all other node's height set to NULL (i.e. undefined). The source broadcasts a QRY packet with the destination node's id in it. A node with a NONE-NULL height responds with a UPD packet that has its height in it. A node receiving a UPD packet sets its height to one more than that of the node that generated the UPD. A node with higher height is considered upstream and nodes with lower height downstream. In this way a directed acyclic graph (DAG) is constructed from source to the destination. Figure 2.6 illustrates a route creation process in TORA [9]. As shown in figure 2.6.a, node 5 does not propagate QRY from node 3 as it has already seen and propagated QRY message from node 2. In figure 2.6.b, the source (i.e. node 1) may have received a UPD each from node 2 or node 3 but since node 4 gives it lesser height, it keeps that height.

When a node moves, the DAG route is broken; route maintenance is needed to reestablish a DAG for the same destination. When the last

downstream link of a node fails, it generates a new reference level. This results in the propagation of that reference level by neighboring nodes as shown in figure 2.7. Links are reversed to reflect the change in adapting to the new reference level. This has the same effect as reversing the direction of one or more links when a node has no downstream links.



(a) Propagation of QRY message through the network



(b) Height of each node updated as a result of UPD messages

Figure 2.6. Route creation in TORA. (Numbers in braces are reference level, Height of each node)

In the route erasure phase, TORA provides a broadcast clear packet (CLR) throughout the network to erase invalid routes.

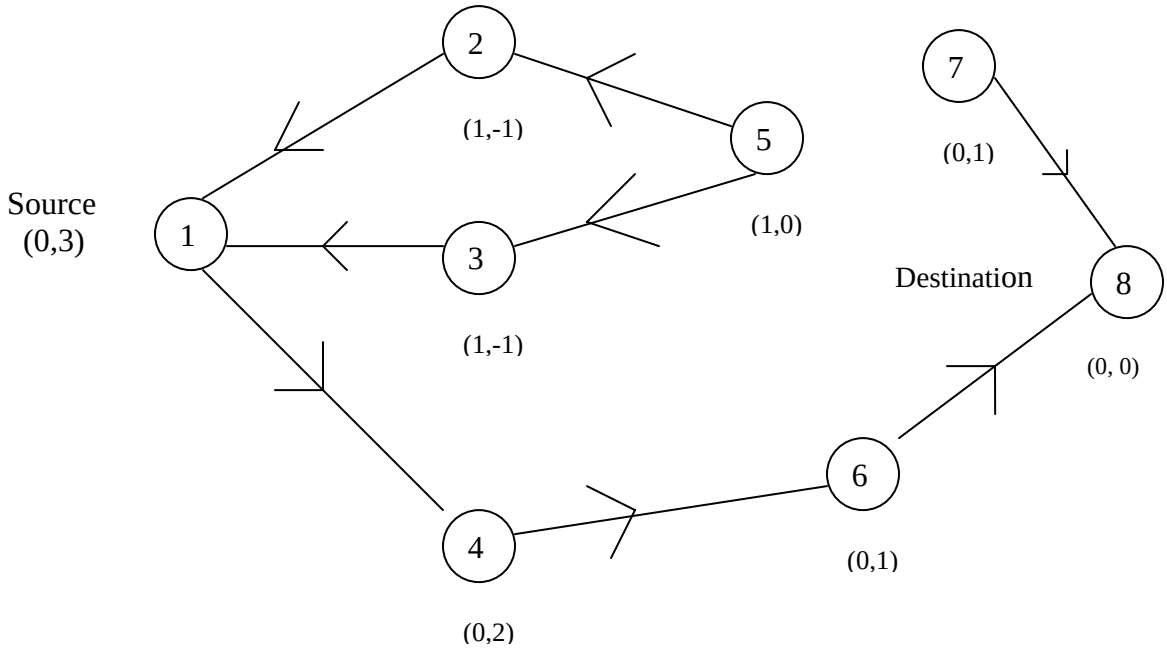


Figure 2.7 Re-establishing route on failure of link 5-7. The new reference level is node5

CHAPTER THREE

Literature Review

In this chapter, different approaches for improving TCP's performance in mobile ad hoc networks, which were done previously have been presented.

3.1 TCP's performance over mobile ad hoc networks

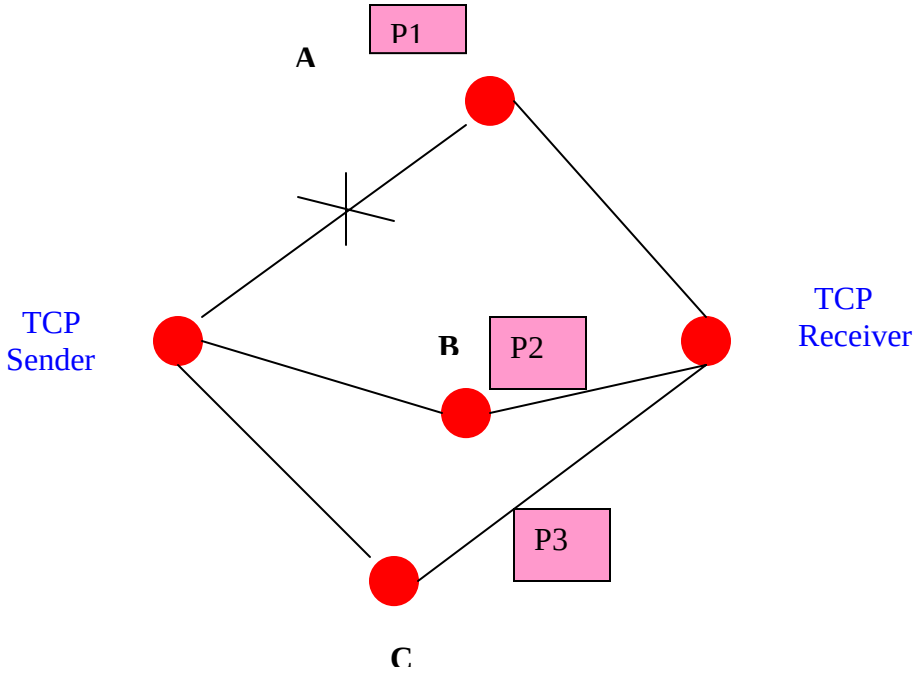
The main causes of TCP's performance degradation in mobile ad hoc networks are [2]: -

I - TCP is unable to distinguish losses due to route failures and network congestion: TCP treats losses caused by route failures as a sign of network congestion, When ever route failure is occurred in mobile ad hoc networks, packets may not arrive in order at the receiver side, as a result the receiver will inform the sender about the delivery of out-of-order packet by sending three duplicate ACKs, up on the reception of three duplicate ACKs, the sender assumes that the network is congested, because reception of three duplicate ACKs means the network is already congested, hence congestion window size and sending rate are reduced drastically, which degrades TCP's performance significantly.

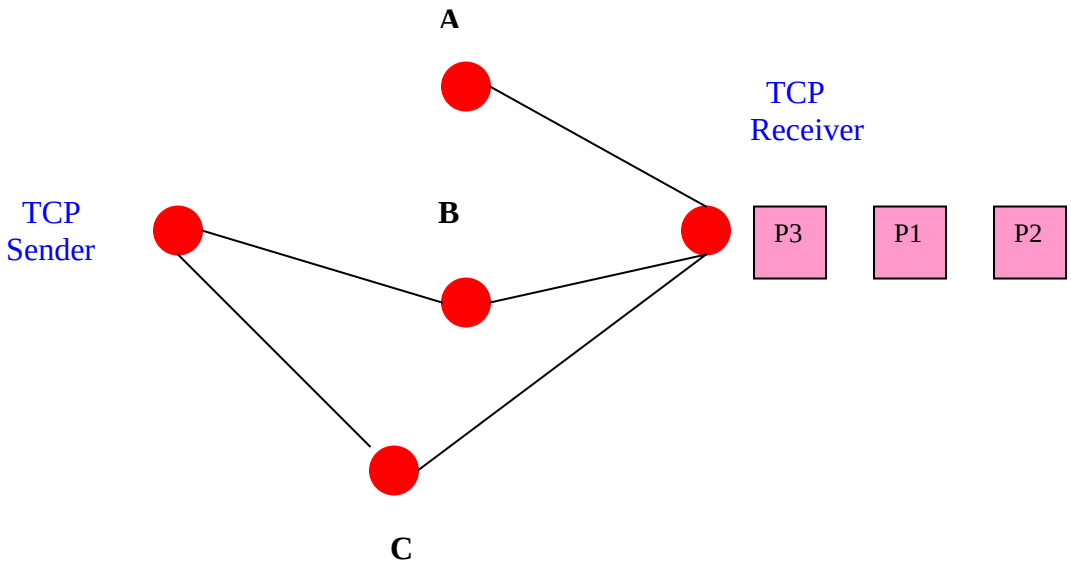
II –TCP suffers from frequent route failures: Due to the mobility of nodes, there is no fixed route in mobile ad hoc networks, hence if sender and receiver are not within their transmission range, route will be failed and

new route will be established depending on the type of routing protocol implemented in MANET.Because of the aforementioned problem, Once again out-of-order packet may be delivered from sender to receiver and degrades TCP's performance a lot.

III – Using Multi-path route in mobile ad hoc networks: When multi-path routing protocol like TORA is used in mobile ad hoc networks, delay due to route reestablishment period will be overcome by maintaining multiple paths from TCP sender to receiver, However due to the failure of the currently used path and the new established route from sender to receiver might be shorter than the previously established path, packet may not be reached at the receiver in order. Figure 3.1 illustrates out-of-order delivery of packet. In figure 3.1.a the TCP sender wants to send three packets to the TCP receiver, at some point, link from sender to node A is failed. Packet 2 is then sent through route sender-B-receiver. As a result that is shown in figure 3.1.b packet2 reaches to TCP receiver before packet1 reaches, which produces out-of-order delivery, but in the real scenario packet1 should reach first followed by packet2 and packet3 respectively. Generally speaking, what out-of-order (OOO) mean is that, when a packet sent earlier arrives later than a subsequent packet.



(a) Propagating packets with multi-path route from sender to receiver



(b) Packet2 reached to the receiver side before packet1 reached

Figure 3.1 Out of order delivery of packet due to multi-path route from sender to receiver.

3.2 Proposals to improve TCP performance in mobile ad hoc networks

Various proposals have been made in the literature to improve the performance of TCP in Mobile Ad hoc networks. These proposals are classified into two groups, according to the first two problems identified in section 3.1 [2]. The proposals are classified as shown in figure 3.2 into: -

1- Cross layer proposals
2- Layered proposals

3.2.1 - Cross layer proposals: -The cross layer proposals mainly deal with the interactions between two layers of the TCP/IP architecture. This proposal was motivated by the fact that providing lower layer information to upper layer should help the upper layer to perform better. Thus, depending on between which two layers there will be information exchange. Cross layer proposals can be further classified into four types: TCP and network, TCP and link, TCP and physical, and network and physical.

From cross layer proposals TCP-F and ATCP are presented.

TCP-F: TCP Feedback [10] is a feedback-based approach to handle route failures in MANETs. This approach allows the TCP sender to distinguish between losses due to route failures and those due to network congestion. Whenever routing agent of a node detects the failure of a route, it explicitly sends a Route Failure Notification (RFN) packet to the source. On receiving the RFN, the source goes into a

snooze state. The TCP sender remains in this snooze state until it is notified of the restoration of the route through Route Re-establishment Notification (RRN) packet. It uses both network and TCP layers.

ATCP: Ad hoc TCP [11] utilizes network layer feedback too. In addition to the route failures, ATCP tries to deal with the problem of high Bit Error Rate (BER). ATCP listens to the network state information provided by ECN (Explicit Congestion Notification) messages and by "Destination Unreachable" message; Upon receiving a "Destination Unreachable" message, the sender enters into the persist state. Upon receipt of an ECN, congestion control is invoked without waiting for a timeout event. If a packet loss happens and the ECN flag is not set, ATCP assumes the loss is due to bit errors and simply retransmits the lost packet. A layer called ATCP is inserted between the TCP and IP layers of the TCP source nodes.

3.2.2 – Layered proposals: - Layered proposals deal with adapting the TCP/IP layers independently of other layers. Thus, depending on which layer is involved, layered proposals can be further classified into three types: TCP layer, network layer, and link layer proposals.

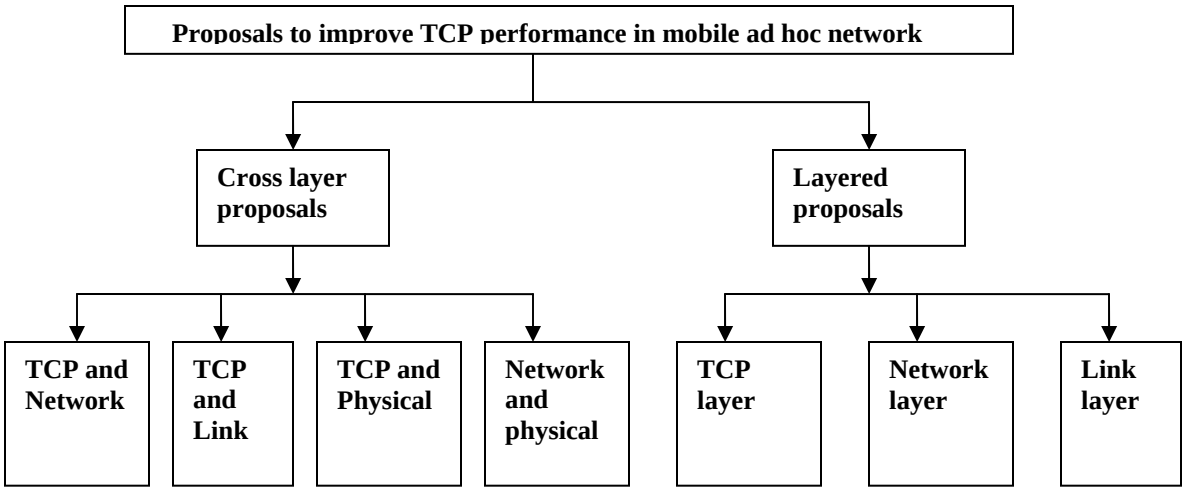


Figure3.2 Classification of proposals to improve TCP performance in Ad hoc networks

From layered proposals, Fixed RTO and TCP-DOOR are presented.

FIXED - RTO: This technique [12] is a sender-based technique that does not depend on feedback from the network. It distinguishes between packet losses due to route failures and congestion by using retransmission time out. Two consecutive timeouts mean a sign of route failures. When two timeout expire in sequence, which corresponds to the situation where the missing ACK is not received before the second RTO expires; the sender concludes that a route failure event has occurred. The unacknowledged packet is retransmitted but the RTO is not doubled in the second time. This is in contrast with the standard TCP, where an "exponential back off" algorithm is used. The RTO remains fixed until the route is re-established and the retransmitted packet is acknowledged.

TCP - DOOR: TCP Detection of Out-of-Order and Response (TCP-DOOR) is an end-to-end approach [3]. It deals with TCP layer only; it doesn't

require any information from the lower layers. This approach does not require the cooperation of intermediate nodes; it is based on out-of-order (OOO) delivery events. OOO delivery of packet is interpreted as an indication of route failure. The detection of OOO packet is accomplished either by means of a sender-based or a receiver-based mechanism. Once the TCP sender knows about an OOO event, it takes the following two response actions: temporarily disabling congestion control and instant recovery during congestion avoidance. In this approach Dynamic Source Routing Protocol (DSR) is used [13], which uses single path route between TCP sender and receiver.

3.3 Comparison of the previously done proposals

In general four proposals have been presented. These proposals address the problem of TCP inability to distinguish between losses due to route failures and network congestion. The two TCP and network layer proposals TCP-F, ATCP, are based on an explicit notification from the network layer to detect the route failures. When we compare the cross layer proposals, ATCP emerges as a good proposal, because it supports mechanisms to make the negative impact of high bit error rate (BER) and packets out-of-sequence on TCP performance.

The main advantage of the two TCP layer proposals, FIXED-RTO and TCP-DOOR is that they do not require explicit notification from routing layer or network layer; they don't also require any help from the intermediate nodes.

In general, as a pure end-to-end approach, TCP-DOOR will work well. And it improves TCP performance about 50 %. However, the assumption is that out- of-order events are the exclusive results of route failure and node mobility.

The above method is implemented only in a single path routing protocol like DSR, and the main causes of out-of-order delivery of packets at the TCP receiver side are assumed to be route failure, route change, and node mobility.

Our interesting question is that what if out-of-order delivery of packet is not caused by using single path routing protocol like DSR, which is affected by frequent route changes and node mobility that has a large delay due to route re-computation? This could happen, for example, in multi-path routing protocol. In order to increase the reliability of data transmission and to reduce delay due to route re-computation, some routing protocols, such as Temporally Ordered Routing Algorithm (TORA), maintain multiple routes between a sender-receiver pair and use multi-path route to transmit packets. In such a case packets that come from different paths may reach at the TCP receiver out-of-order. We have proposed TCP-DOOR-TS mechanism to deal with this issue and packet losses.

CHAPTER FOUR

TCP DETECTION OF OUT OF ORDER AND RESPONSE WITH TIME STAMP OPTION (TCP- DOOR-TS) ALGORITHM.

4.1 Overview

TCP-DOOR-TS algorithm is an end-to-end layered proposal solution to improve the performance of TCP in mobile ad hoc networks. It mainly deals with TCP layer in the TCP/IP architecture. In the early stage of TCP implementation that is in wired networks, TCP assumes that packet losses are primarily due to congestion in the network. As a result, when a packet loss is indicated either by the receipt of three DUPACKs or a time-out of the retransmission timer, it enters into congestion control and packet recovery algorithms. This may not be appropriate in a wireless network with maintaining multiple routes between sender-receiver pair, where a significant amount of packet losses in the network could be due to delivery out-of-order packet and packet loss. The TCP-DOOR-TS algorithm modification in TCP-SACK both at the receiver and sender side aims to provide some possible solution by responding for out-of-order detection of packets and changes the time at which congestion control algorithm is invoked.

TCP-DOOR-TS algorithm is aimed to detect out-of-order delivery of packet at the TCP receiver side due to multi-path routing protocol, namely Temporally Ordered Routing Algorithm (TORA), and respond for

the detected packet at the sender side. The algorithm is implemented as an extension to TCP-SACK. Selective Acknowledgment (SACK) is used both at the sender and receiver side. If out-of-order packet is delivered at the TCP receiver (TCP-SINK-SACK), the TCP receiver sends out-of-order bit to the TCP sender (TCP-SACK), which is an indication of out-of-order reception of packets.

We use the time stamp option to detect out-of-order packet at the receiver side. Time stamp option is implemented in all TCP algorithms [14]. Since The TCP sender records the exact time each packet is sent in the TCP packet header. The TCP receiver can compare the time stamp in each packet with the previous one to detect out-of-order packet.

Once out-of-order (OOO) packet is detected, The TCP receiver informs the TCP sender by setting out-of-order bit in the TCP header. Up on the reception of OOO bit, for a time period T the TCP sender: -

1- Temporarily disables some of the state variables like Duplicate Acknowledgment (DUPACK) and RTO.

2- Adjusts the Congestion Window Size (CWND) based on the network condition.

3- Send one new packet.

- Otherwise, if out-of-order packet does not detected by the receiver and the receiver sends three DUPACKs. On the reception of the first DUPACK at the sender side, for a time period T disable triggering of duplicate acknowledgment algorithm and send one new packet.

- If within the disabling period the packet that is out-of-order is not reached, call congestion control algorithm.

Due to the implementation of the above steps at the sender side; the retransmission of packet will not be activated and the congestion window size will not be decreased unnecessarily.

4.2 The proposed algorithm

In this section we present TCP-DOOR-TS algorithm in detail,with implementation using NS-2 of version 2.29.

4.2.1 Out-of-order detection of packet at the TCP receiver side

In TCP-DOOR-TS algorithm the assumption made is that out-of-order packet is most likely caused by using multi-path route from sender to receiver and the network is assumed to be not congested.

In order to detect out-of-order packet at the receiver side, we use the TCP Time Stamp Option. The TCP sender records the exact time whenever packet is sent to TCP receiver. TCP packet header is common for both the sender and receiver. Hence the sender does have the capacity to put time stamp and the receiver can read and respond accordingly. The TCP receiver can compare the time stamp in each packet with the previous one to detect out-of-order packet. Detection of out-of-order-packet algorithm is shown in algorithm 1.

Algorithm 1. Pseudo-code of out-of-order packet detection at the receiver side

1- Retrieve the time stamp (present_pkt_ts_) from the incoming packet, which is sent by TCP sender.

2- Retrieve the previously saved packet time stamp (saved_time_) that is recorded in TCP receiver.

3- Compare the present packet time stamp with the previously received packet time stamp, i.e.:

If (present_pkt_ts_ < saved_time_) {

1- Set out of order option bit (ooo_option_) to 1

2- Send out-of-order option bit (ooo_option_) with the ACK packet to be acknowledged to the TCP sender.

3- Put the value of present packet time stamp (present_pkt_ts_) into saved time (saved_time_) for the next comparison.

4- Then reset ooo_option_ = 0, present_packet_ts_ = 0 and other necessary variables.

// ooo_option_ a variable which should be put in the TCP header (tcp.h)

}

Else {

- In -sequence packet is received
- Send ACK (same as TCPSINKSACK does) }

4.2.2 Responding for out-of-order-packet at the TCP sender side

Upon the reception of out-of-order bit (ooo_option_bit) from TCP receiver, which is an indication of out-of-order delivery of packet at the receiver side, The TCP sender temporarily disables duplicate acknowledgment (DUPACK) and retransmission time out (RTO), and invokes open congestion window (open_cwnd ()) algorithm. This algorithm opens the congestion window, which adjusts the congestion window size in order to continue the CWND to evolve. When in slow-start phase, the algorithm increments cwnd_ by each ACK received. When in congestion avoidance phase the standard configuration increments cwnd_ by its reciprocal. Then send one algorithm will be called in order to send one new packet. The reason for sending one new packet with in the disabling period is that receiving ooo_option_ bit from the TCP receiver would mean that in normal TCP function the TCP sender is accepting one duplicate acknowledgment (DUPACK = 1), and up on the reception of one DUPACK, only one new packet is sent.

Disabling DUPACK and time-out with in the specified period allows out-of-order packets to reach to the TCP receiver, if with in the specified period out-of-order packet is not reached to the TCP receiver, then disabling period will be expired and normal function of TCP will proceed.

Otherwise, if out-of-order packet is not detected at the receiver side and duplicate acknowledgment becomes incremented at the sender side, then on the reception of the first DUPACK at the sender side, Once again for a time period T disable triggering of duplicate acknowledgment algorithm and send one new packet.

But during the above period T, if the cause of out-of-order packet is not reached at the receiver side, then the assumption that the packet loss due to out–of-order is changed, packet is lost in transit, at this point congestion control algorithm will be invoked and the packet that is lost will be retransmitted by using the fast retransmission algorithm of TCP-SACK.

The disabling period T is derived from the NS-2 variables RTT. The disabling period is set to be one and two times RTT. Because the RTO estimate is computed based on the RTT.Theoretically if the disabling period is set to one times RTT (T=RTT), since RTO is calculated as RTT + 4 times the mean deviation of RTT, unnecessary RTO will be avoided, which forces the congestion window size to start from one. However for disabling period T equals to two times RTT (T=2*RTT) have also be simulated in order to compare the result with that of T=RTT, and to have come up with a compromise selection of disabling period T.The algorithm at the sender side is shown in algorithm 2.

Algorithm 2. Pseudo-code of out-of-order packet response at the sender side

1 – Retrieve the received acknowledgment header and check for out-of-order option bit (ooo_option_).

If (ooo_option bit is set to 1) {

For a time period T, The TCP Sender disables its state variables
i.e.
For (Disable period = RTT) {

1 - Set number of duplicate acknowledgment (dupacks_) to zero.

2 – Set the retransmission time out (timeout_) to false.

3 – Adjust the congestion window size (cwnd_) by invoking the Open congestion window algorithm.

4 – Send one new packet.

}
}

Else if (Three duplicate ACK) {

For (Disable period = RTT)
{

- Invoke send one algorithm in order to send one new Packet.

If (disable period expires)
{
Call fast retransmission and fast recovery algorithms.
}
}
}
Else if (standard ACK is received)
{
Do as standard TCP-SACK does
}

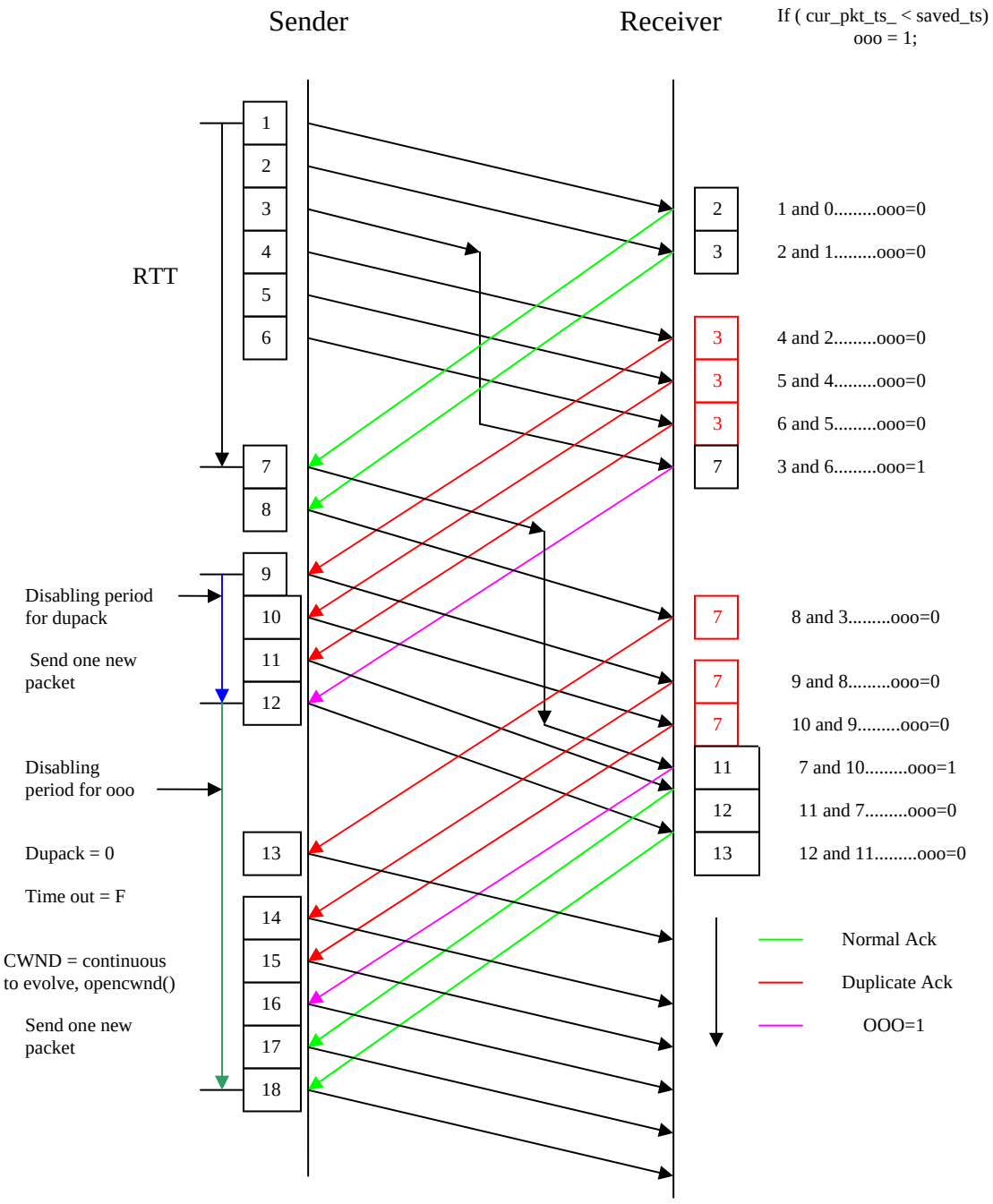


Figure 4.1 TCP-DOOR-TS Out-of-order detection of packet and response mechanism

Figure 4.1 illustrates how TCP-DOOR-TS works. First TCP sender sends six packets; all the packets except packet 3 are reached at the receiver side inorder. Since packet 3 is reached after packet 6 has reached, TCP sender received three duplicate acknowledgments, at the reception of these 3 DUPACKs, disabling period for 3 DUPACKs has started which avoids calling of congestion control algorithm, which should be called in conventional TCP.

At packet number 11 at the sender side, congestion window size is not reduced and packet number 3 is not retransmitted. At packet number 12 acknowledgment for packet number 3 is received, hence disabling period for 3 DUPACKs is expired, however the acknowledgment for packet3 has carried out-of-order bit (OOO =1), once again disabling period for reception of ooo=1 will start, and make DUPACKs=0, time-out = false, call open_cwnd () algorithm so as to make CWND to evolve, and send one new packet. During this disabling period once again due to the reception of out-of-order packet for 7, three DUPACKs have been received by the sender, since disabling period for the reception of ooo=1 is not yet expired, unnecessary reduction of CWND size and retransmission of packet 7 is strictly avoided.

In figure 4.2 once again packet 3 is lost, which is not reached out-of-order at the TCP receiver side. At the TCP sender side disabling period for 3 DUPACKs has started and one new packet is sent, with in this period since packet three is not received, the disabling period is expired. After tolerating for this period, packet 3 is retransmitted and CWND is reduced.

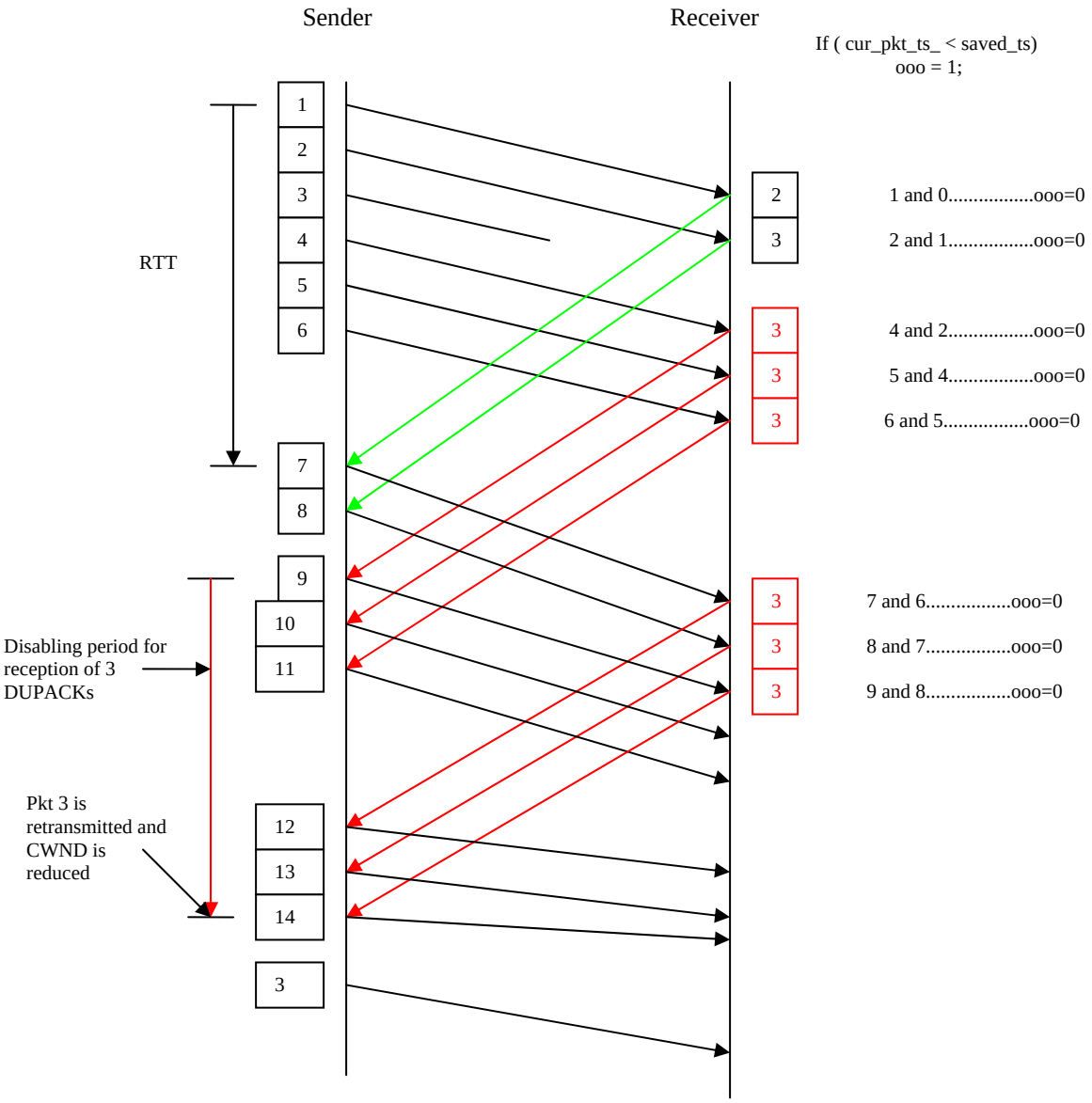


Figure 4.2 TCP-DOOR-TS packet loss detection

CHAPTER 5

IMPLEMENTATION OF TCP-DOOR-TS ALGORITHM USING NS-2 VERSION 2.29.

The simulations were done using NS-2, version 2.29. In this Chapter, first the NS-2 simulation tool and then the implementation details of TCP-DOOR-TS algorithm using NS-2.29 will be presented.

5.1 Introduction to NS-2

NS-2 is an open source discrete event simulator used by the research community for research in networking [15]. It supports for both wired and wireless networks and can simulate several network protocols such as TCP, UDP, multicast routing, etc. More recently, support has been added for simulation of large satellite and ad hoc wireless networks. The NS-2 simulation software was developed at the University of Berkeley. It is constantly under development by an active community of researchers.

NS-2 takes full advantage of the features of object-oriented programming. It is written in C++ and OTcl programming languages. Although it does not guarantee production of a mirror image of the real world, it does try to model most of the protocol behavior accurately and can be used to study various protocols at different levels of the TCP/IP layers. It is focused on modeling network protocols including wired, wireless and satellite networks with transport protocols such as TCP, and UDP with both unicasting and multicasting capabilities. It models Web, Telnet, and FTP applications. It also includes the implementation

of ad-hoc routing and sensor networks. It provides mechanisms for gathering statistics, tracing, and error modeling for the simulations carried out. Apart from the core code of the NS-2, there have been numerous contributions from other researchers.

We have used NS-2 to perform the network simulations. The simulation results were used to evaluate the TCP-DOOR-TS algorithm when multi-path routing protocol TORA is used and compare the result with "TCP-SACK" that is TCP with out using the proposed algorithm. We chose NS-2 for the implementation because it is a freely distributed code and supports many interesting protocols.

C++ and Tcl are the two languages used in NS-2. Two languages are needed to perform complex programming, coupled with the need for speed when we vary Parameters/configurations to explore a large number of scenarios while studying the various protocols. C++ is fast to

execute, however it is slow to change, making it suitable for the complicated protocol implementation. However, it is very slow when varying parameters and rerunning simulations. Tcl, on the other hand, is much slower but very convenient for varying simulation parameters. Consequently, C++ is used for implementing properties of the protocol while Tcl is used to implement code that needs to be changed often in order to study the protocol behavior.

5.2 Class Hierarchy in NS-2

Figure 5.1 shows the NS-2 architecture. The root of the hierarchy is the class TclObject. It is the superclass of all OTcl library objects such as scheduler, network components, timers, and other objects (NAM). The

simulator has a class hierarchy in C++ (compiled hierarchy) and a corresponding class hierarchy in OTcl (interpreted hierarchy). Both these hierarchies are closely related.

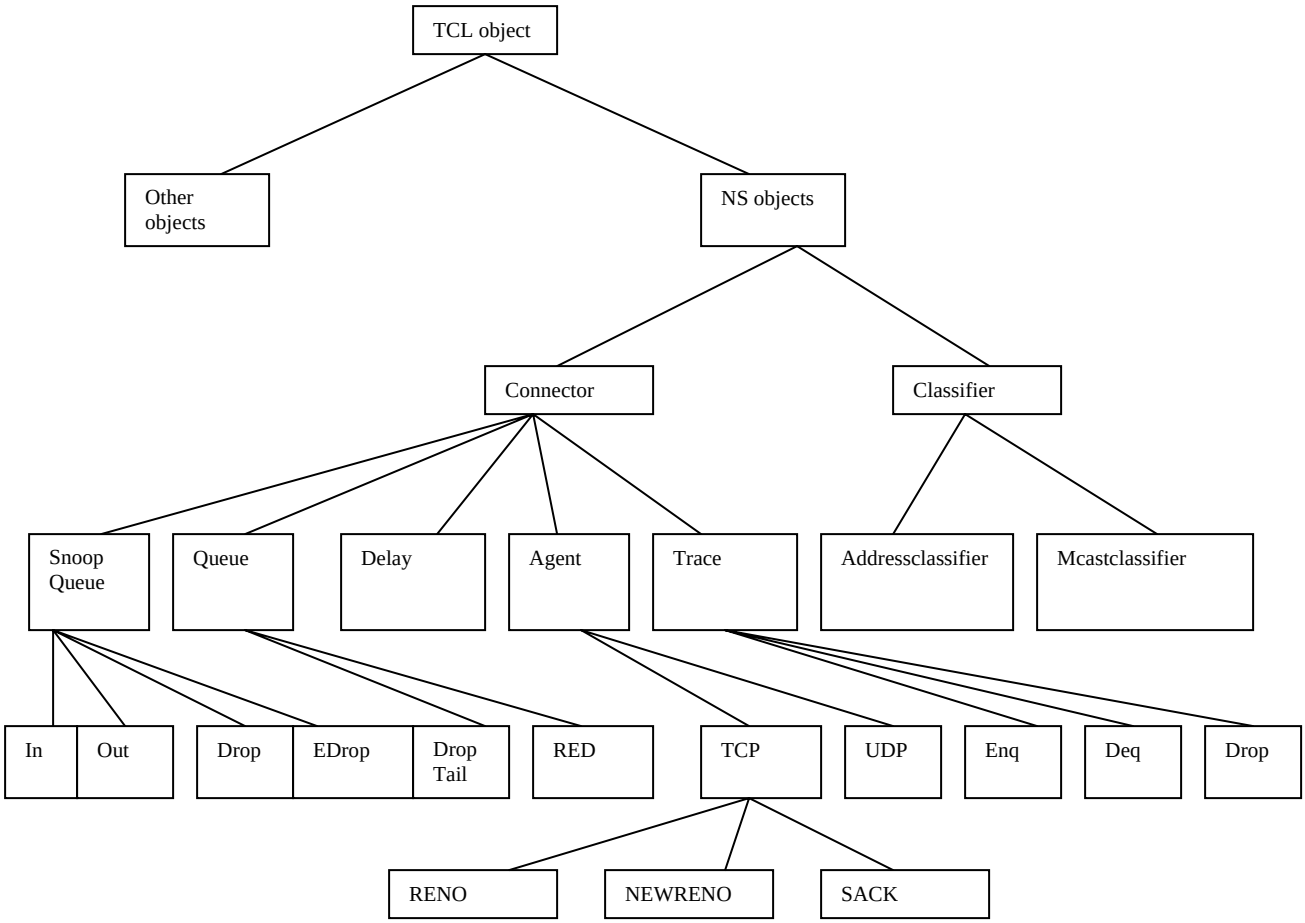


Figure 5.1 NS-2 class hierarchies

5.3 Agents in NS-2

Agents, which are responsible for the end-to-end transmission of stream of bytes in the real network, are also end points in NS-2. The class Agent is the base class, which is partly implemented in OTcl and partly in C++. There are various protocol agents in NS-2, including the basic agents such as TCP and UDP. The agents are usually created

through Tcl during a simulation. In this case, the constructor for the agent in the compiled code is executed. The binding is then performed in the class. The main tasks performed by these agents are processing the requests and responses at the sender and receiver sides. They also implement a timer class if necessary. To create a new agent, we have to first decide on the inheritance structure and create appropriate class definitions, define recv() and timeout() methods, define any necessary timer classes, define the OTcl linkage functions, and write the necessary OTcl code to access the agent. The pre-existing agents of NS-2 provide an excellent base for extending various other complicated protocols.

5.4 Implementation of TCP-DOOR-TS algorithm using NS-2 version 2.29

The implementation of TCP-DOOR-TS algorithm is based on one of the TCP version known as TCP-SACK modules both at the receiver and sender side. At the TCP receiver side the recv () method, which is called whenever new packet is received, and the ack () method, which is processed whenever an acknowledgment is prepared to send to the TCP sender, are mainly used to implement the proposed algorithm. Both the methods are modules of the SACK at the TCP receiver and sender side. Necessary modifications are made in the agent named class TCPSINKSACK and its recv () function, which is the main reception path for packets and provides various other necessary methods. One variable known as ooo_option has been included in the TCP header format so as to inform the sender about the detection of out of order bit. At the sender side the main modification is done on the receive method. This method should check for two variables, namely ooo_option bit and scoreboard. Up on the reception of ooo_option bit it

checks for the value of this bit, if it is one, some of the variables like number of duplicate acknowledgment (numdupack_), and retransmission timeout (timeout_) will be disabled and opencwnd () will be called so as the congestion window to evolve. Lastly, send_one () function will be called, which sends one new packet in order to utilize the network effectively. If ooo_option bit is set to 0 and numdupack_ becomes increments, the TCP sender will check this increment from the variable known as scoreboard, that is it Checks for a duplicate ACK, which Check the SACK block actually acknowledges new data or not and one the increment of numdupack_ = 1, TCP sender will disable triggering of dupack () function, instead it will call sendone() algorithms so as to send one new packet, if with in the disabling period the packet which is lost or out of order is not come, dupack () function will be called, which retransmit the lost packet and reduce the congestion window size cwnd_. Figure 5.2 shows the implementation hierarchy and the important C++ classes for TCP-DOOR-TS implementation are shown with darker shade.

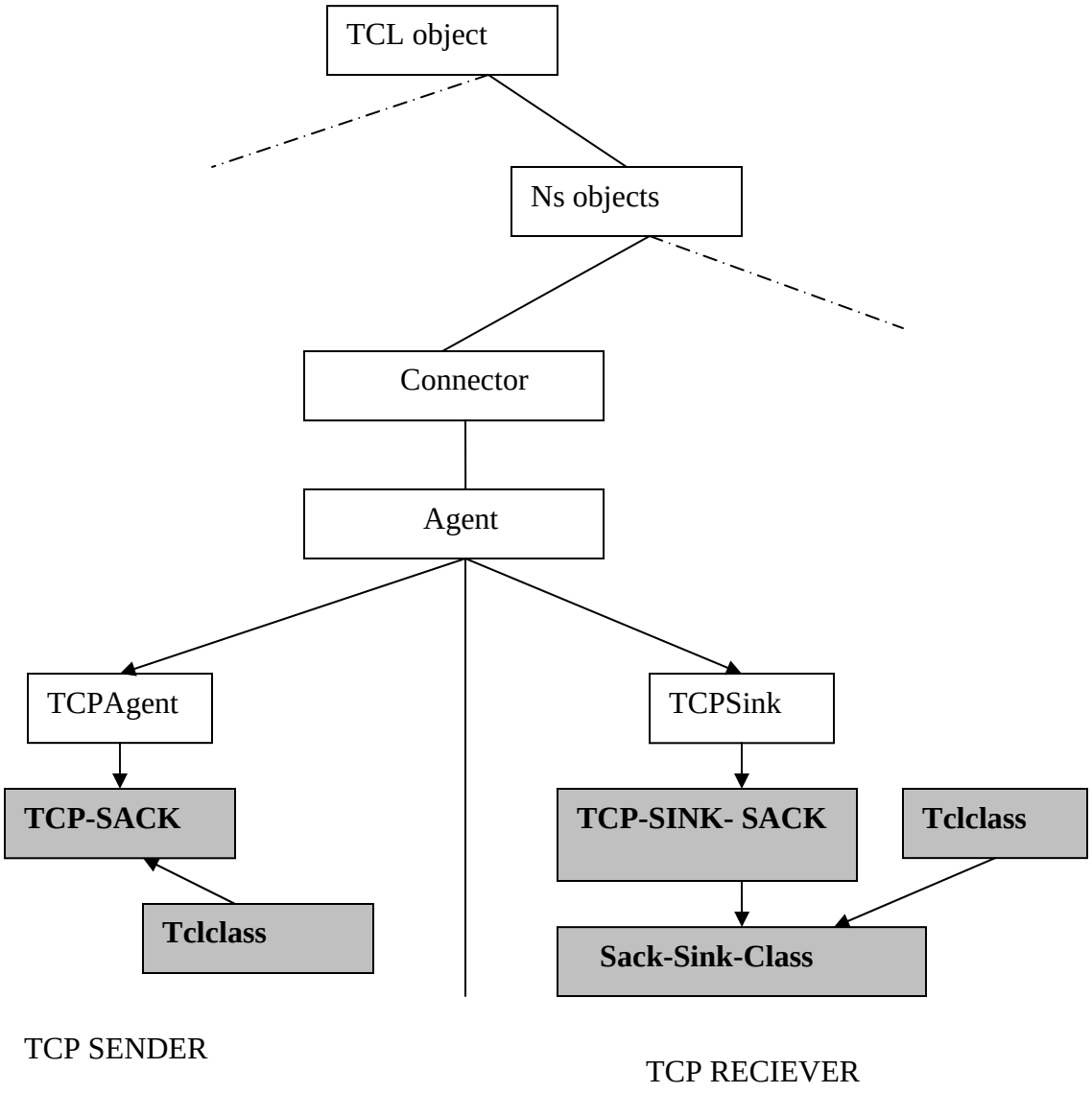


Figure 5.2 TCP-DOOR-TS implementation in network simulator 2 version 2.29

CHAPTER SIX

PERFORMANCE EVALUATION

In this Chapter, we present the performance evaluation of TCP-DOOR-TS
And TCP-SACK Algorithms using NS-2.First simulation scenarios and model will be discussed; the network topology and movement of nodes with traffic models will be discussed next. Then detail simulation results and analysis will be presented.

6.1 Simulation Scenarios and Model

A detail simulation model based on NS-2 has been used in the evaluation, and in order to perfectly evaluate the effect of out-of-order packet while multi-path routing protocol is used and see the effect of implementing TCP-DOOR-TS algorithm in TCP-SACK,different simulation scenarios have been used.

The NS-2 simulator supports for simulating wireless networks consists of different network components including physical, data link, and medium access control (MAC) layer models. From channel type, a wireless channel model with a 250m-transmission range has been chosen. *IEEE 802.11* for wireless networks is used as the MAC layer protocol. All packets (both data and routing) sent by the routing layer are queued at the *interface queue* until the MAC layer can transmit them. The *interface queue* has a maximum size of 50 packets and is worked as a priority queue. There are two priorities each served in First In First Out (FIFO) manner, which means Routing packets have higher

priority than data packets. The routing protocol that has been implemented at the network layer is *Temporally Ordered Routing Protocol (TORA)* which is supposed to produce out-of-order packet as a result of using multi-path route between sender and receiver.

The workload is a single TCP connection between a specific sender S (node 0) and a specific receiver R (node 9). The congestion-disabling period have been varied to be 1 and 2 times the retransmission timer RTT (retransmission timer).

6.2 The traffic and mobility models

For traffic source and application, *File Transport Protocol (FTP)* is used above the agent TCP. The source and destination that have been used through out the simulations were TCP-SACK and TCP-SINK-SACK respectively. The source-destination pairs are spread randomly over the network. The data generator is FTP.1040 byte data packets are used from sender to receiver and 40 byte acknowledgments are used from receiver to sender.

Mobility models were created for the simulations using 20 nodes, and this model was set in such a way that first all the 20 nodes were provided with initial location in the given rectangular topography field. The field configuration used is: 1000 m x 900 m field. Then all the nodes move with in their boundary by setting their final destination and the speed that each node move with. The speed was chosen randomly between 0 and 15m/s and 20 mobile nodes have been used in the simulation. All the simulations are run for 200 simulated seconds. Different mobility and identical traffic scenarios are used across the protocol to collect fair results.

6.3 Performance Metrics

Two important metrics are evaluated over TCP protocol with and with out implementing TCP-DOOR-TS in TCP-SACK: -

6.3.1 Throughput - Number of packets sent by the TCP sender that are generated by File Transfer Protocol, over simulated time and it is presented in packet.

6.3.1 Good put - Number of packets received by the TCP receiver, over simulated time and it is presented in packet. It can also be indicated as the maximum sequence number of packets reached at the destination.

The higher the throughput and good put, the better performance had TCP achieved. For each TCP-DOOR-TS setting, we divided the throughput and good put values by the throughput and good put measured in the base TCP (standard TCP-SACK) case. This ratio measures how much improvement we have achieved with TCP-DOOR-TS under a given ad-hoc scenario, provided that out-of-order packet is really delivered due to the effect of multi-path route.

We use the above metrics in order to confirm that all the packets sent by the TCP sender is really delivered to the TCP receiver, actually there is a mechanism to know about it, TCP algorithm uses a clock based acknowledgment, which means new packet is sent only if the previously sent packet is acknowledged.

Generally speaking, The above two metrics are the most important for evaluating the performance of TCP over multi-path routing protocol

such as Temporally Ordered Routing Protocol using TCP-DOOR-TS algorithm.
In order to analyze the simulated results in detail we use another metrics known as *congestion window size* and *slow start threshold*.

6.4 Simulation code

Besides the implementation of the proposed algorithm in C++ code, there was a need to write the Tcl code in order to set up the wireless simulation components: network components types, parameters like the type of antenna, the radio-propagation model, the type of ad-hoc routing protocol, traffic models and node movement models used by mobile nodes etc. The documented code is available in the Appendix.

6.5 Parsing the Simulation trace files

The simulation was performed for thirty five times for each metrics and scenarios by changing the node positions and speeds. After each simulation, trace files recording the packets that are sent and received by the TCP sender and receiver, packets that are forwarded by the routing protocol implemented, and that are queued and dequeued by Interface queue. These files need to be parsed in order to extract the information needed to measure the proposed performance metrics.

We have also used Network Animator and xgraph in order to analyze the simulation results visually.

6.6 Performance evaluation results and analysis

As per earlier discussion, Out-of-order packet is delivered due to the implementation of Multi-path routing protocol TORA, and TCP-DOOR-TS have detected and responded in order to improve the TCP performance by avoiding unnecessary invoking of congestion control and retransmission algorithms, which reduce the congestion window size that was seen by “TCP-SACK". A detail simulation study is presented below.

6.6.1 Performance evaluation of TCP with TCP-SACK

First we have evaluated the performance of TCP with out implementing TCP-DOOR-TS algorithm that is with TCP-SACK at the sender side and TCP-SINK-SACK at the receiver side. The simulation was done for thirty five times by changing the position and movement of the 20 nodes and average was taken. Figure 6.1 shows the simulation results. Packet sending was started at 20 simulations second, and a maximum of 1551 packets were sent by the TCP sender (TCP-SACK) over 200 simulated times.

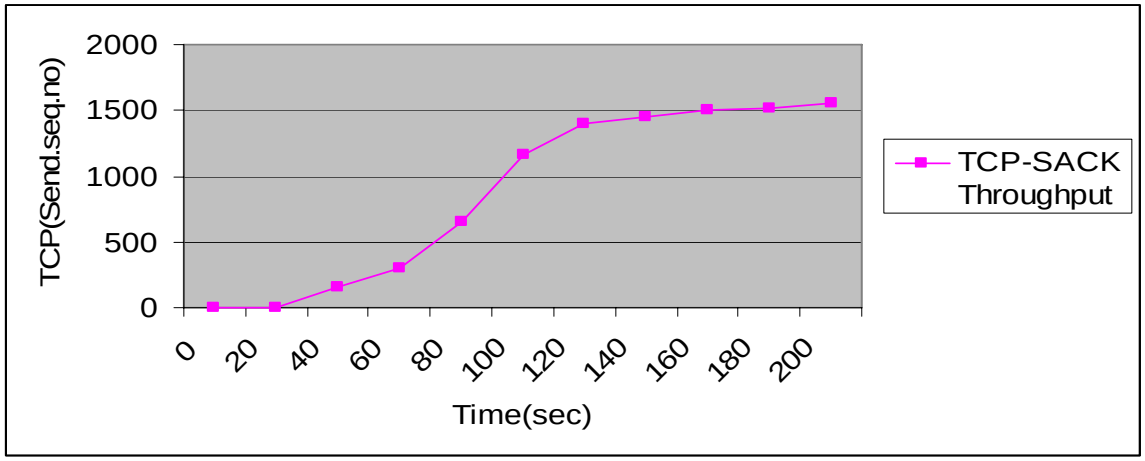


Figure 6.1 The maximum number of packets sent by TCP sender over 200 simulation seconds (throughput)

And as shown in figure 6.2, the TCP receiver received maximum of 1549 packets. Theoretically, since TCP protocol is working on clock-based acknowledgment, new packet is sent only if the previously sent packet is acknowledged. The simulation results in figure 6.1 and 6.2 reveal this fact; the TCP receiver receives almost the entire packet sent by the TCP sender.

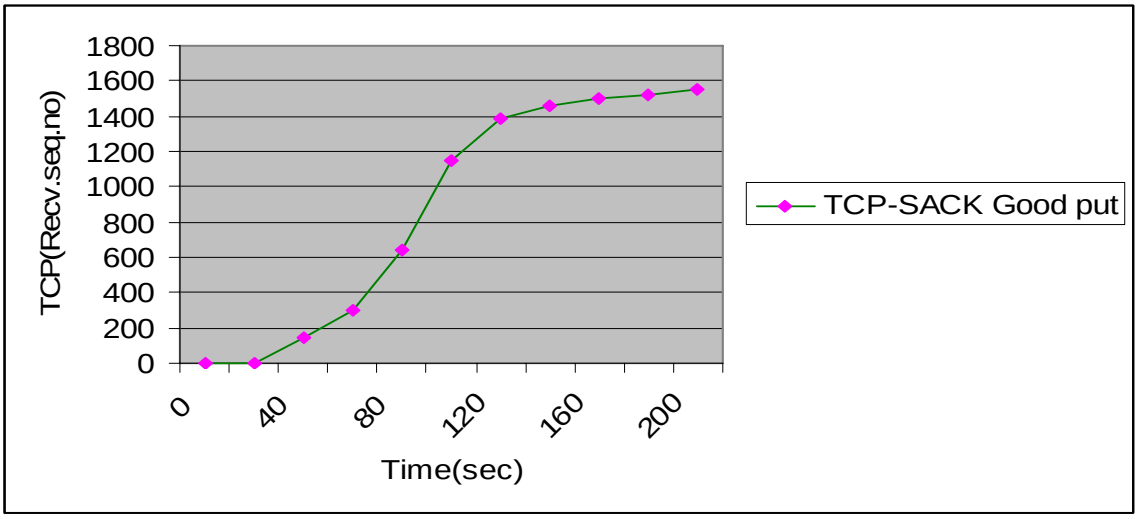


Figure 6.2 The maximum number of packets received by TCP receiver over 200 simulation seconds (good put)

Figure 6.3 and 6.4 tell about the congestion window size [CWND] with the 200 simulated times, as per the previous discussion, CWND means the maximum number of window size that can be sent by the TCP sender at a time. In figure 6.3.a between 0 and 50 simulation seconds the sender sent a maximum of CWND 12. A maximum of 30 CWND was achieved at a simulation time of 100. Whenever the CWND reduces drastically, it means packets are reached out-of-order in which the TCP receiver produces either three duplicate acknowledgment or the expected acknowledgment is not come with in the retransmission time out period, If time out is occurred, slow start threshold is reduced to

half of the current congestion window size, the CWND is reduced below SSTRHESHOLD that is the TCP sender enters in to slow start and start sending packet from one and continuous sending exponentially until it reaches to slow start threshold (SSTRHESHOLD). If the CWND is reduced but not below SSTRHESHOLD, then the TCP sender has received three DUPACKs, so it will start from congestion avoidance phase. From the simulation result, about the time of 85sec the TCP sender receives three DUPACKs, and start sending from congestion avoidance phase because as shown in figure 6.4.a at this 85 second the SSTHRESHOLD was found to be 8, which is equal to half of the CWND.

At about 30,110 and between 100 and 150, and 160 seconds, retransmission timer has expired, and The TCP sender enters into slow start phase in which it start sending form one packet. At the above simulation times as shown in figure 6.4.a the slow start threshold was found to be 2.

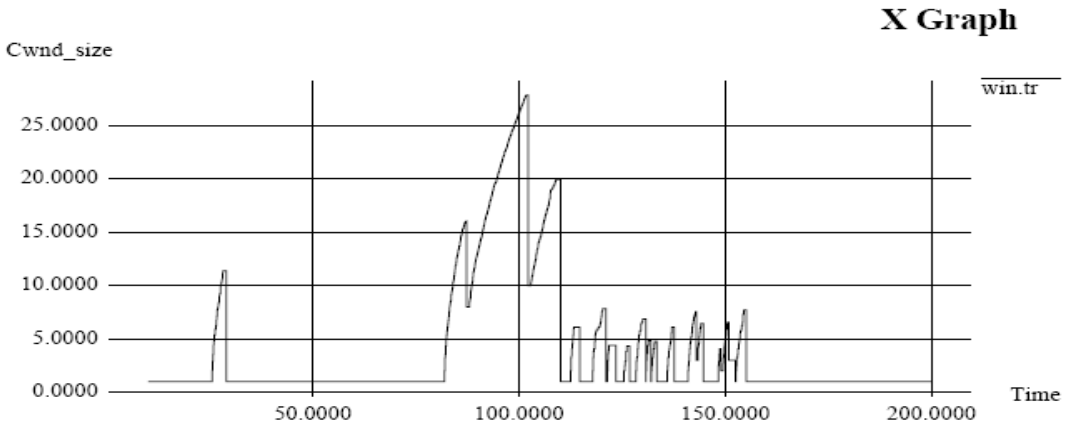


Figure 6.3. a. Congestion window Vs simulated time with TCP-SACK

In Figure 6.3.b, between 100 and 200 simulation seconds, the congestion window size reduced a lot, which is the effect of out-of-order delivery of packets. This reduces the TCP's sender CWND that affects the maximum number of packets that can be sent by the TCP sender (throughput).

The slow start threshold of "TCP-SACK" has been shown in figure 6.4.a and 6.4.b. It shows the boundary between the slow start phase and congestion avoidance phase in order to maintain the consistency of TCP sender – receiver pair.

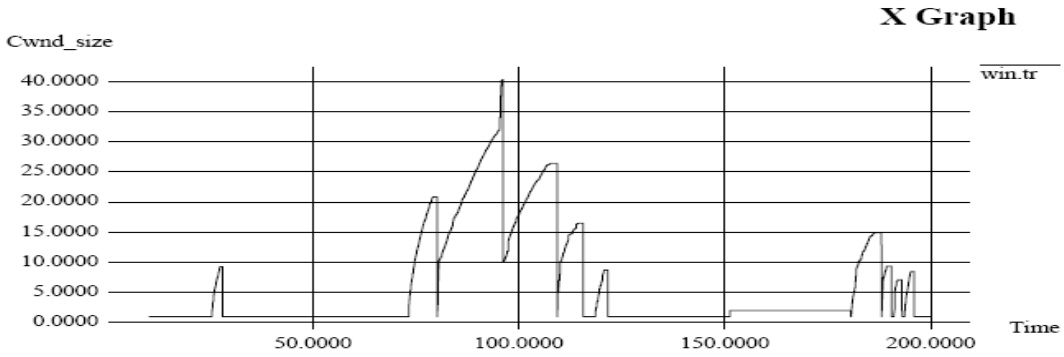


Figure 6.3.b Congestion window Vs simulated time with TCP-SACK

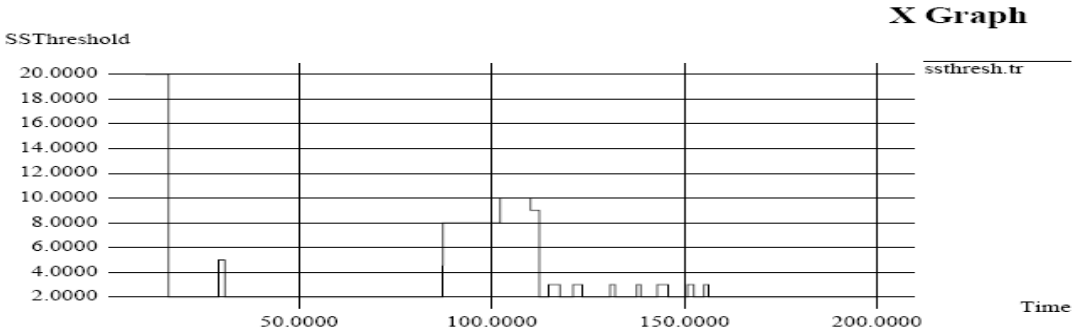


Figure 6.4.a Slow start threshold over 200 simulated times with TCP-SACK.

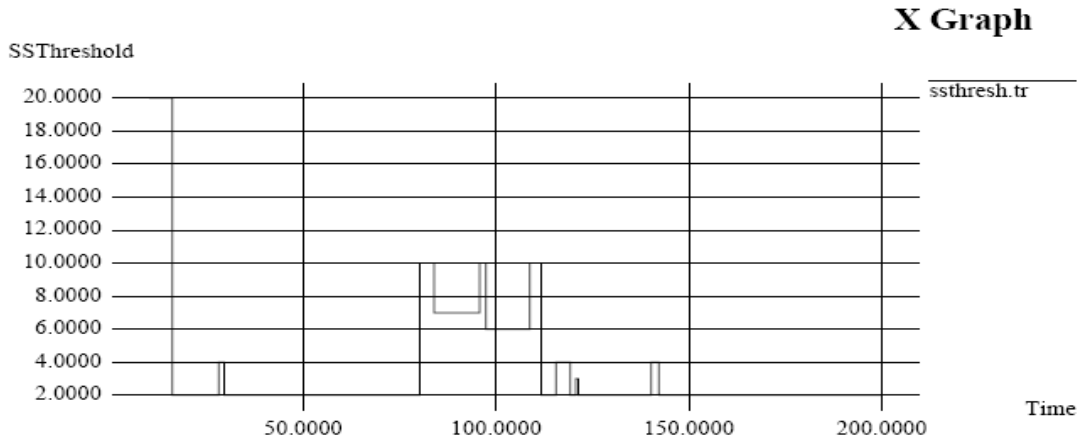


Figure 6.4.b Slow start threshold over 200 simulated times with TCP-SACK.

6.6.2 Performance evaluation of TCP by implementing TCP-DOOR-TS algorithm

In this evaluation we have varied the disabling period T to be 1 and 2 times RTT (retransmission timer).

6.6.2.1 Disabling period T = RTT

Once again the simulation was done for thirty five times by changing the position and movement of the 20 nodes and their speeds and average was taken. When the disabling period T is equal to one time the retransmission timer, as shown in figure 6.5, the maximum number of packets sent by TCP sender over 200 simulated time were found to be 1754.With the above result an average improvement of 13 % was found by implementing TCP-DOOR-TS algorithm. In figure 6.6 the good put effect has been shown and the maximum number of packets received by the TCP receiver were about 1738, which improves the TCP good put

by 12.7%. The above percentages improvement implies that out-of-order packets are really delivered due to multi-path route in case of "TCP-SACK" which degrades the performance of TCP, and implementing TCP-DOOR-TS algorithm have improved the effect of out-of-order problem.

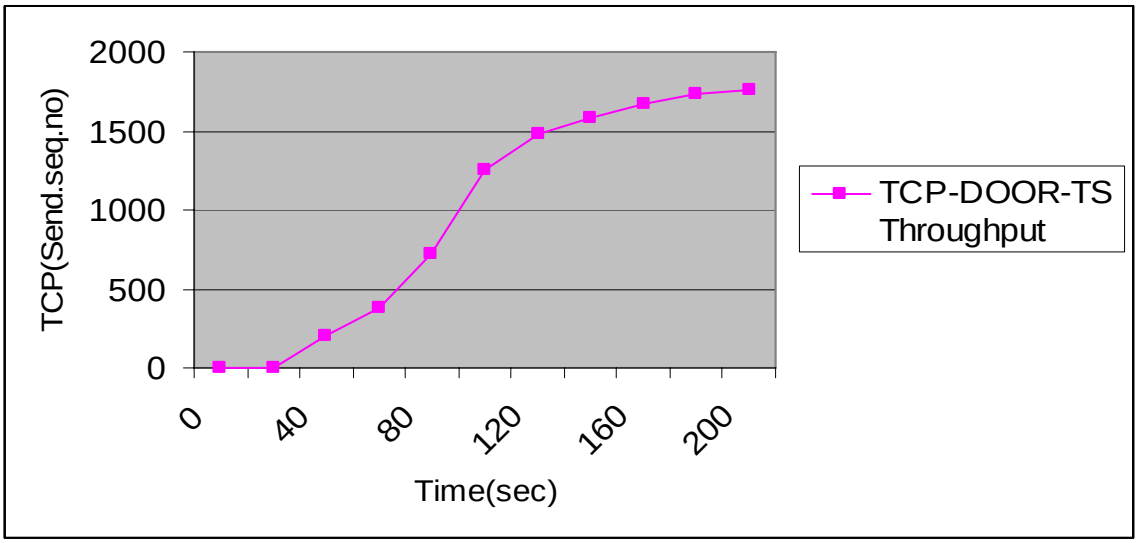


Figure 6.5 The maximum number of packets sent by TCP sender over 200 simulated seconds (throughput), with TCP- DOOR-TS (T = RTT)

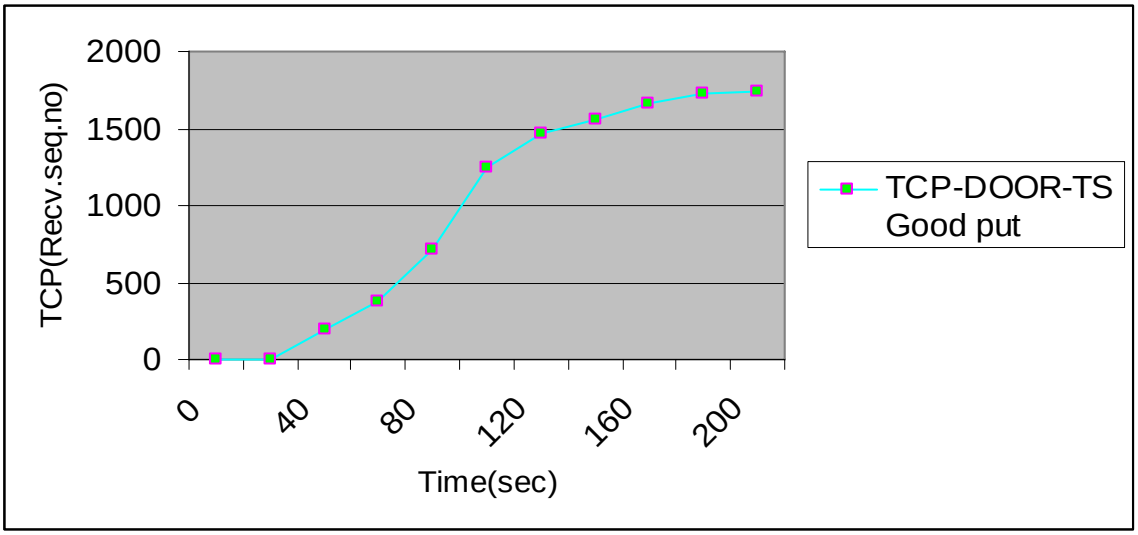


Figure 6.6 The maximum number of packets received by TCP sender over 200 simulation seconds (good put), with TCP-DOOR-TS (T = RTT)

In figure 6.7.a and 6.7.b TCP-DOOR-TS shows larger congestion window size than "TCP-SACK", this indicates better utilization of available bandwidth. The slow start threshold is shown in figure 6.8.a and 6.8.b.

As shown in figure 6.7.b, between 100 and 150 simulation seconds, the congestion window size (CWND) is better than that of shown in figure 6.3.b, which is the effect of detecting out-of-order packet and responded accordingly. In figure 6.3.b between 150 and 200 simulation seconds only one packet has been sent at a time. But as shown in figure 6.7.b between 150 and 200 seconds the congestion size is increased up to a maximum of 15 and many packets have been sent which is the effect of avoiding unnecessary calling of congestion control algorithm.

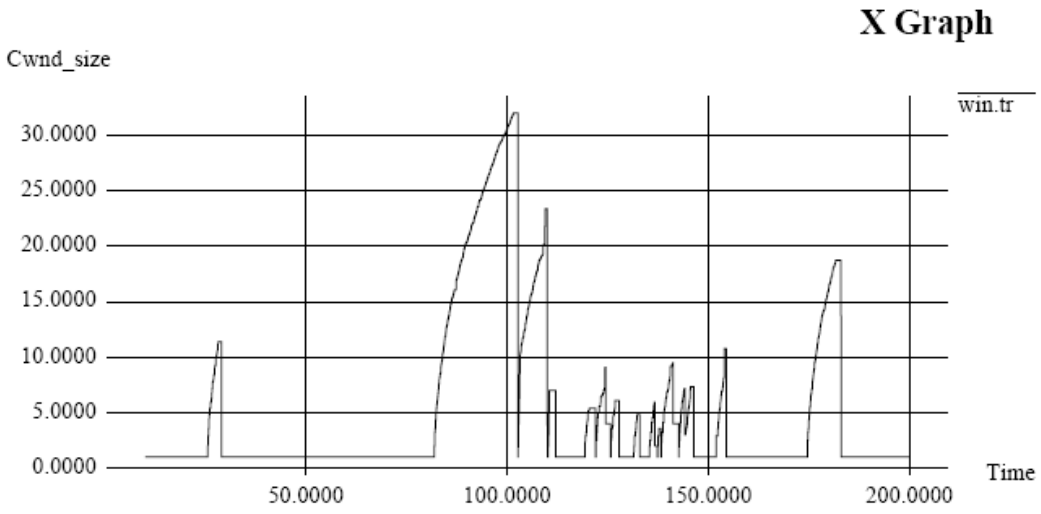


Figure 6.7.a. Congestion window Vs simulated time with TCP-DOOR- TS algorithm (T=RTT)

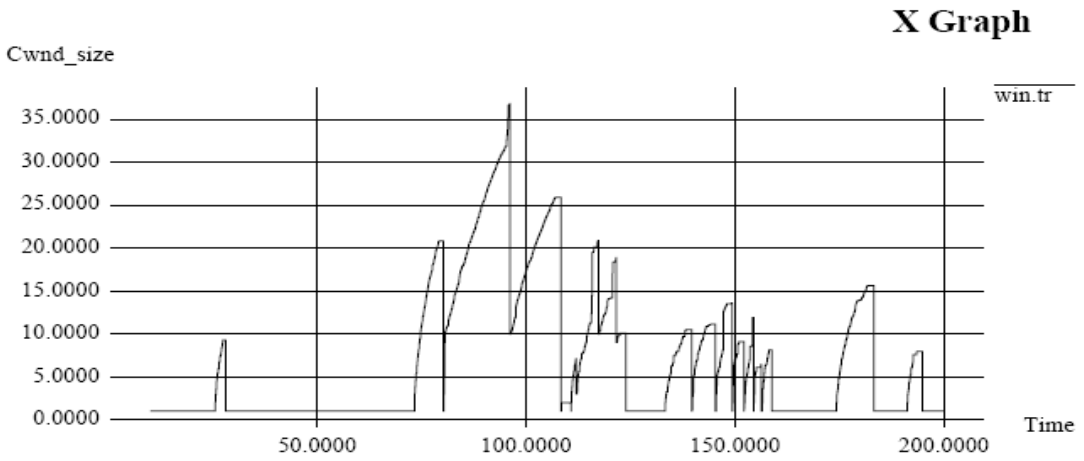


Figure 6.7.b Congestion window Vs simulated time with TCP-DOOR-TS algorithm (T=RTT)

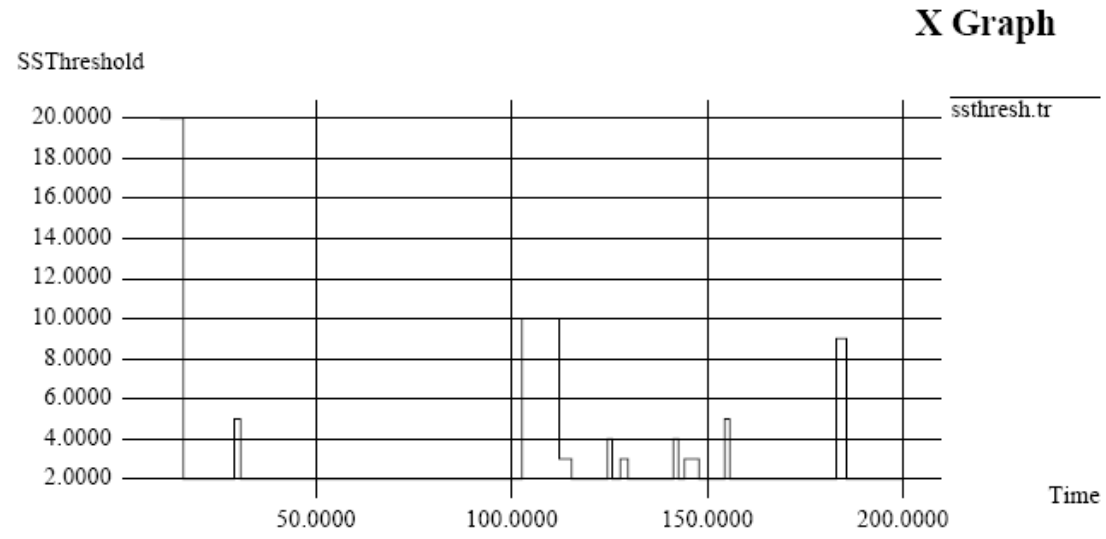


Figure 6.8.a. Slow start threshold over 200 simulated time with TCP-DOOR-TS (t=rtt)

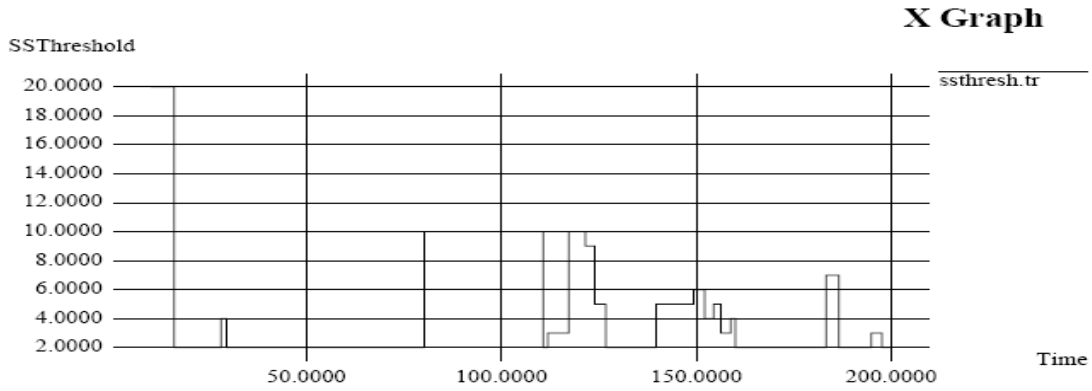


Figure 6.8.b Slow start threshold over 200 simulated time with TCP-DOOR-TS (t=rtt)

6.6.2.2 Disabling period T = 2 x RTT

For disabling period T = 2 x RTT, as shown in figure 6.9, a maximum of 1668 packets have been sent by the TCP sender over 200 simulated times (throughput), which is compared to the base TCP-SACK, it improves the throughput by 7.54% but compared to disabling period of T = RTT, the throughput is decreased by 5.46%.

When we see the good put effect, as shown in figure 6.10 for T = 2 x RTT, about 1668 packets have been received by the TCP receiver, which indicates that all the packets sent by the sender have been received by the receiver. The good put result compared with TCP-SACK is about 7.54%, which shows improvement compared to TCP-SACK and it decreases by 5.46% compared to T = RTT of TCP-DOOR-TS.

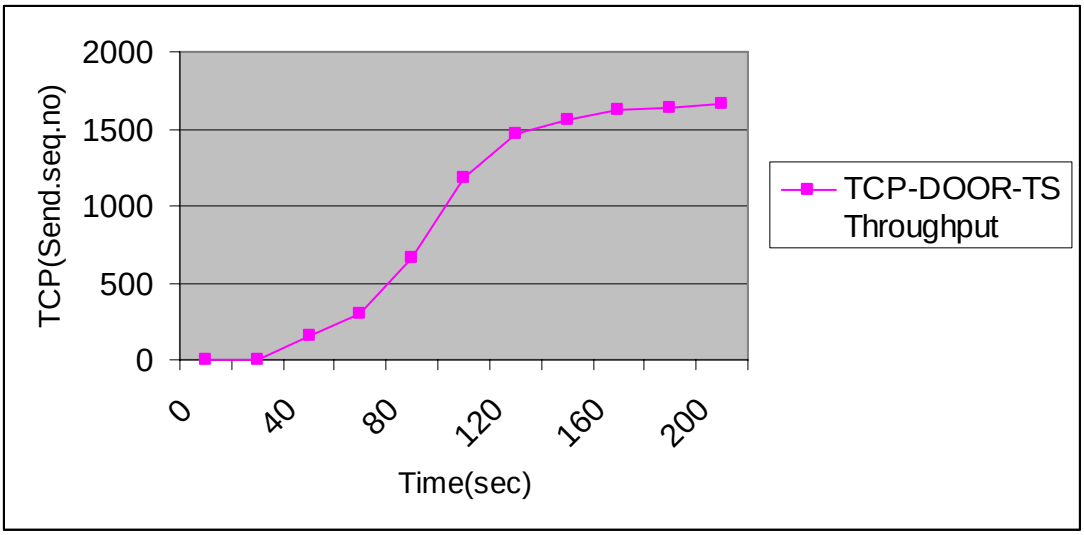


Figure 6.9 TCP throughput measured for TCP-DOOR-TS for T =2 x RTT

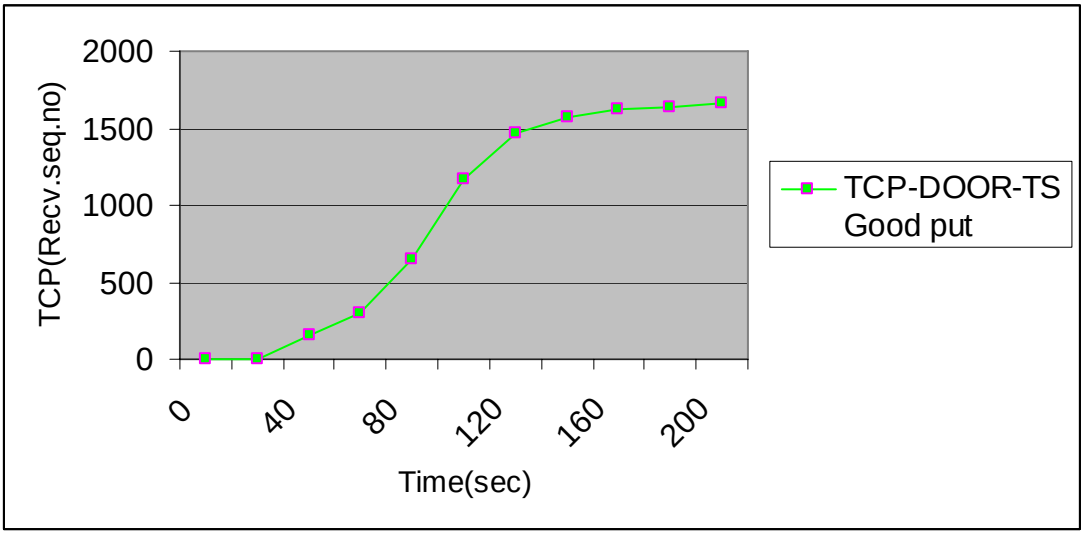


Figure 6.10 Good put result found by implementing TCP-DOOR-TS with t=2xrtt

Figure 6.11 and 6.12 show the congestion window and slow start threshold found by making the disabling period T = 2xRTT. The size of congestion window with in simulated time is some how good relative to base TCP-SACK, and drops to some extent compared to T = RTT.

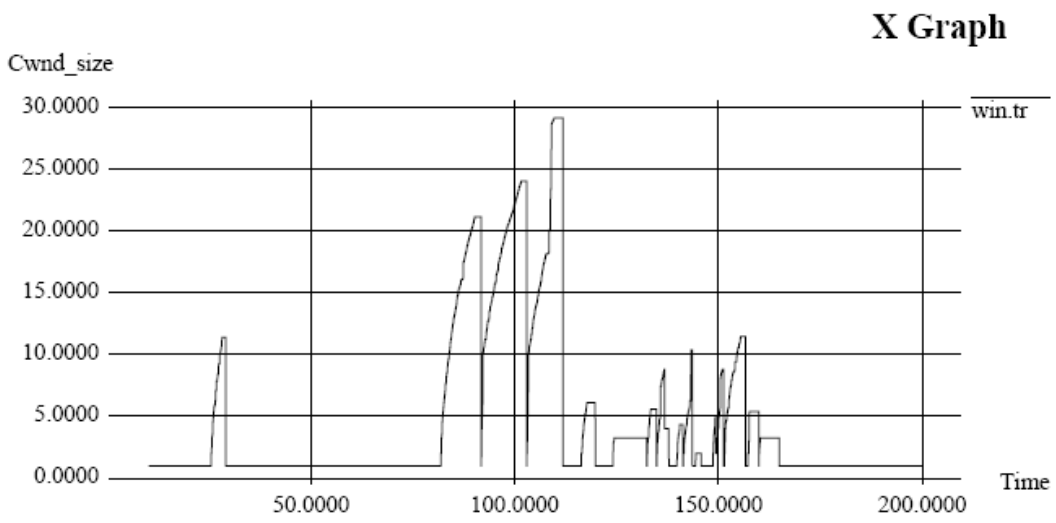


Figure 6.11.a congestion window Vs simulated time for t = 2* rtt

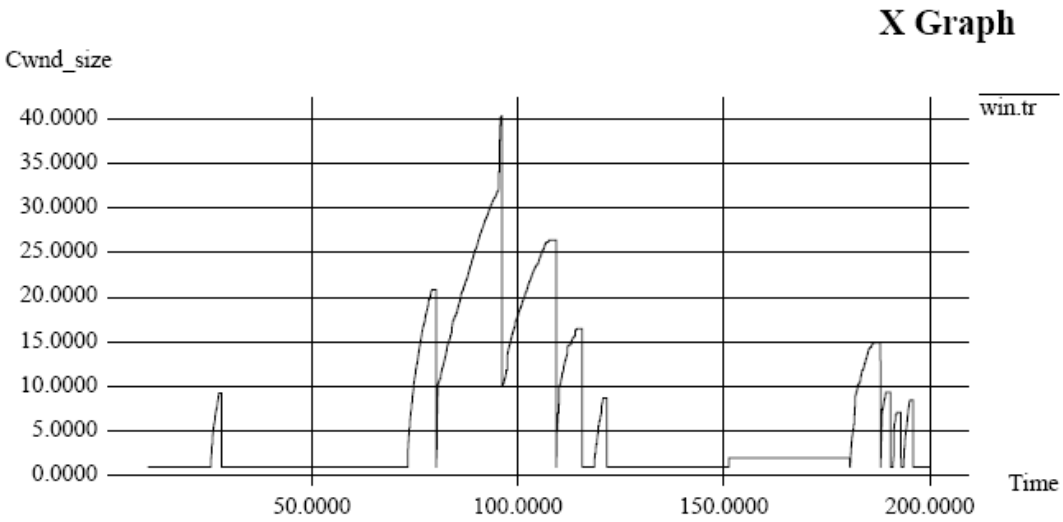


Figure 6.11.b congestion window Vs simulated time for t = 2xrtt

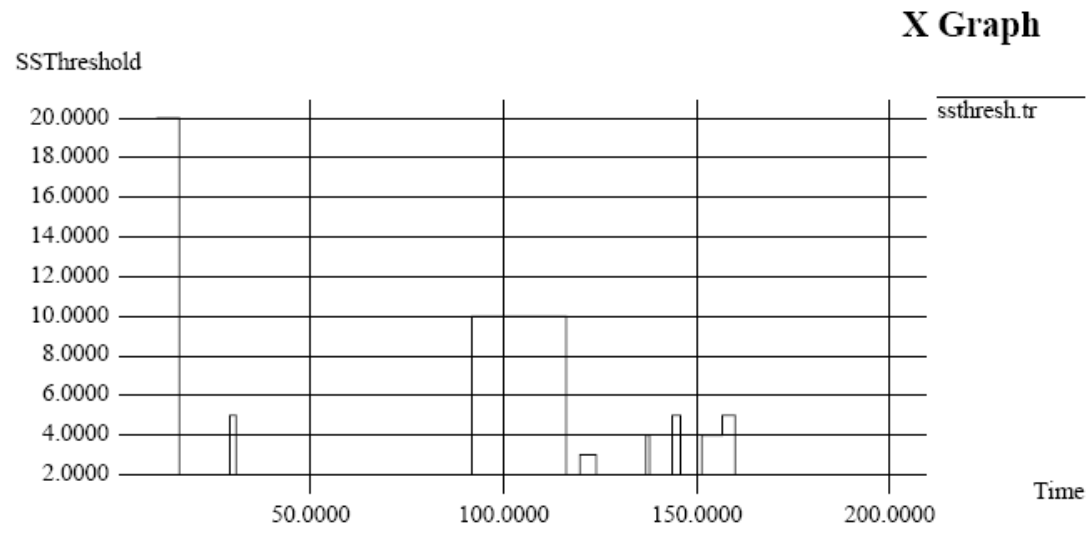


Figure 6.12.a Slow start threshold Vs simulated time for t = 2xrtt

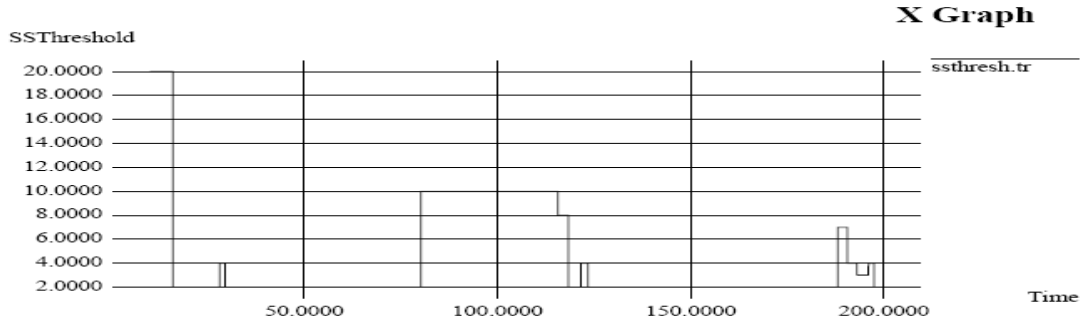


Figure 6.12.b. Slow start threshold Vs simulated time for t = 2xrtt

Figure 6.13 and figure 6.14 show comparison of throughput and good put measurement for both TCP-SACK, and TCP-DOOR-TS with disabling period T=RTT and T=2XRTT. All measures taken by the TCP-DOOR-TS improve the throughput and good compared to TCP-SACK. For disabling period T = RTT, a throughput and good put improvement

ratio of 13% and 12.7% have been achieved, respectively. And for a disabling period of T = 2xRTT, for both throughput and good put improvement ratio of 7.54% has been found.

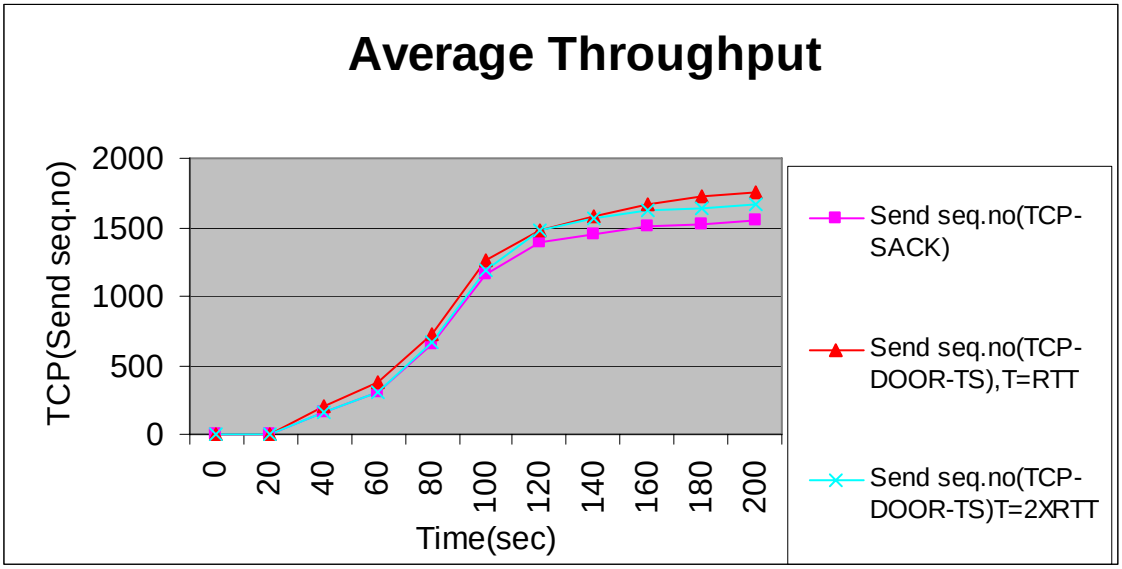


Figure 6.13 Comparison of throughput, TCP-SACK and TCP-DOOR-TS

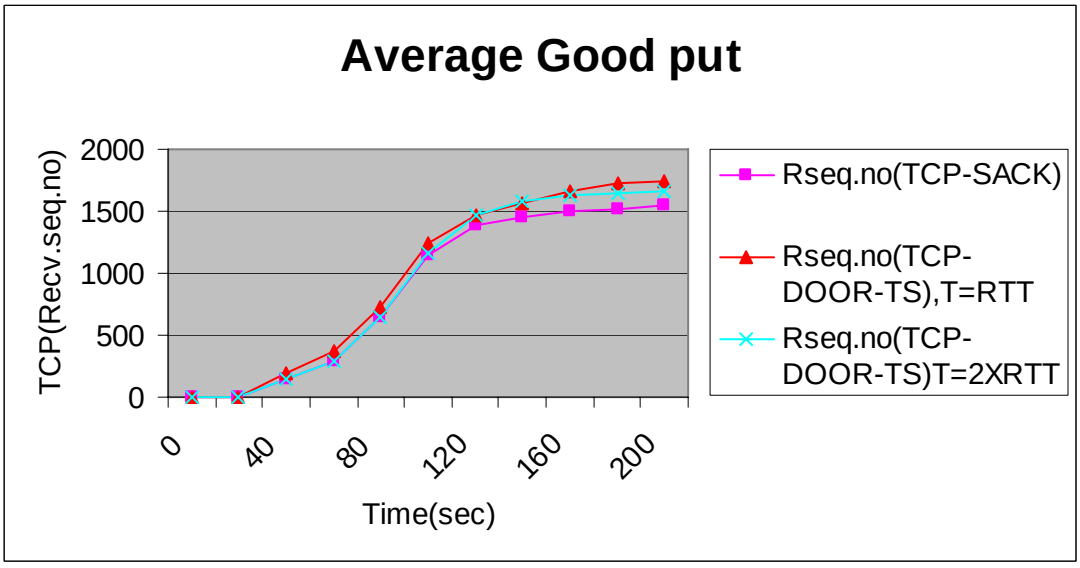


Figure 6.14 Comparison of good put, TCP-SACK and TCP-DOOR-TS

CHAPTER SEVEN

CONCLUSION

The modification of TCP Detection of Out-of-Order and Response with Time Stamp (TCP-DOOR-TS) algorithm in TCP-SACK, when multi-path routing protocol (TORA), is used in mobile ad hoc networks has been evaluated. First it is confirmed that out-of-order packet is really delivered due to multi-path route between TCP sender and receiver, which has significant performance degradation effect. TCP-DOOR-TS algorithm improves the performance of TCP by detecting and responding for out-of-order packet delivery. Unnecessary invoking of congestion control algorithm and retransmission of packets were protected. All measures taken by the TCP-DOOR-TS algorithm improve the bandwidth utilization and increase the throughput and good put up to 13% and 12.7% respectively, compared to TCP-SACK.

The above percentage improvements implies that out-of-order packet is occurred in case of TCP-SACK which degrades the performance of TCP, and implementing TCP-DOOR-TS algorithm have improved the effect of out-of-order problem.

For disabling period T = RTT, a throughput and good put improvement ratio of 13% and 12.7% have been achieved, respectively. And for a disabling period of T = 2 x RTT, for both throughput and good put improvement ratio of 7.54% has been found which implies that disabling period of one times the round trip time is optimum for attaining good result. Theoretically it is also stated in chapter four

(4.2.2) that whenever the disabling period T = 2 x RTT, the TCP protocol would not respond fairly. Disabling period T = RTT is optimum time for the TCP to recover from out-of-order packet at the receiver side.

The congestion window size and slow start threshold of TCP-SACK, and TCP-DOOR-TS with T = RTT and T = 2 x RTT have been evaluated, increments of congestion window size for TCP-DOOR-TS than TCP-SACK has been observed, which implies that unnecessary calling of congestion response algorithm and retransmission of packet have been avoided, which reduces the congestion window size and sending rate a lot.

FUTURE WORK

As a future work TCP-DOOR-TS may be evaluated when the network is congested. Furthermore, performance of TCP-DOOR-TS with other TCP's version like TCP-Reno, TCP-New Reno could be investigated.

As shown in the performance evaluation, disabling period T = RTT performs better than T = 2 x RTT, what will be the effect for T = 4 x RTT and T = 8 x RTT, it should be analyzed in the future.

In this thesis, we have evaluated the performance of the proposed algorithm with multi-path routing protocol. And out-order-packet is mainly caused by maintaining multi-path route between sender and receiver; what if single path routing protocols like Destination Sequence Distance Vector (DSDV) and Adhoc On-demand Distance Vector (AODV) are using, it would be an interesting subject to investigate.

REFERNCE LIST

[1] Imrich Chlamtac a, Marco Conti b, Jennifer J-N.Liu c. "Mobile ad hoc networking: imperatives and challenges." Istituto IIT,Consiglio Nazionale delle Ricerche, Pisa, Italy 2003.

[2] Ahmad Al Hanbali, Eitan Altman, Philippe Nain. "A survey of TCP over ad hoc networks". B.P. 93, 06902 Sophia Antipolis Cedex, France. June, 2005.

[3] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad hoc networks with out-of-order detection and response," in Proc. of ACM MOBIHOC, Lausanne, Switzerland, Jun. 2002, pp. 217–225.

[4] Vorgelegt von, Ruy de Oliveira, von Brasilien."Addressing the Challenges for TCP over Multihop Wireless Networks." Bern university, June 2005.

[5] W. R. Stevens, TCP/IP Illustrated, Volume 1: "The protocols." New York: Addison-Wesley, 1994.

[6] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control" IETF RFC 2581, Apr. 1999.

[7] Andrew S.Tanenbaum. "Computer Networks "Fourth Edition, Prentice Hall, March 17, 2003.

[8] M. Mathi, J. Mahdavi, and S. F. A. Romanow.” TCP Selective Acknowledgement Options.” RFC 2018, IETF Network Working Group, October 1996.

[9] [Park97] VD Park and MS Corson "A highly adaptive distributed routing algorithm for mobile Wireless networks", Proc. INFOCOM'97, Apr.1997.

[10] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, “A feedback based scheme for improving TCP performance in Ad-Hoc wireless networks,” in Proc. of the International Conference on Distributed Computing Systems (ICDCS’98), Amsterdam, Netherlands, May 1998.

[11] J. Liu and S. Singh, “ATCP: TCP for mobile ad hoc networks,” IEEE JSAC, vol. 19, no. 7, pp. 1300–1315, Jul. 2001.

[12] X. Yu, “Improving TCP performance over mobile ad hoc networks by exploiting cross-layer information awareness,” *Proceedings of ACM.*

[13] T. Dyer and R. Boppana, “A comparison of TCP performance over three routing protocols for mobile ad hoc networks,” in Proc. of ACM MOBIHOC, Long Beach, CA, USA, 2001, pp. 56–66. *Mobicom*, October 2004.

[14] V. Jacobson, R. Braden, and D. Borman. “TCP extensions for high Performance,” IETF RFC 1323, May 1992.

[15] ns-2 [Online]. Available: http://www.isi.edu/nsnam/ns.

[16] vorgelegt von, Ruy de Oliveira, von Brasilien.”Addressing the Challenges for TCP over Multihop Wireless Networks.” Bern university, June 2005.

[17] Sakuna Charoenpanyasak, Beatrice Paillassa, Farid Jaddi. “Experimental study on TCP enhancement interest in ad hoc networks.” IRIT lab-ENSEEIHT, Toulouse-France.

18] Xiang Chen, Hongqiang Zhai, Jianfeng Wang, and Yuguang Fan.”TCP Performance over Mobile Ad Hoc Networks.”

[19] Stephen Mueller1, Rose P. Tsang2, and Dipak Ghosal1.”Multi path routing in Mobile Ad Hoc Networks: Issues and Challenges.”1 Department of Computer Science, University of California, Davis, CA 95616 2 Sandia National Laboratories, Livermore, CA 94551.

[20] Rajashree Paul. “Selective TCP for wired/wireless networks” Bachelor of Technology, University of Kalyani, 2002

[21] Ruy de Oliveira,Torsten Braun.”TCP in Wireless Mobile Ad Hoc Networks.” IAM-02-003 July 2002.

[22] Zhenqiang Ye, Srikanth V. Krishnamurthy, Satish K. Tripathi.”Effects of Multipath Routing on TCP Performance in Ad Hoc Networks.” Department of Electrical Engineering University of California, Riverside.

[23] Md. Abdullah-Al-Mamun, M. Mahbubur Rahman, Hwee-Pink Tan.”Performance evaluation of TCP over routing protocols for

mobile ad hoc networks."Department of Information and Communication Engineering Islamic University Kushtia 7003, Bangladesh.

[24] Yu He, Cauligi S. Raghavendra, Steven Berson, Robert Braden." TCP Performance with Active Dynamic Source Routing for Ad Hoc Networks." USC Information Sciences Institute, Marina del Rey, CA 90292, USA.

APPENDIX

Simulation code

```
#Title: Performance evaluation of TCP over multi-path routing
#protocol (TORA) using TCP-DOOR-TS algorithm in mobile ad hoc
#networks. A 20-node simulation of mobile ad-hoc network with
#Temporally  Ordered Routing Protocol (TORA).

# Define options
set val(chan)           Channel/WirelessChannel     ;# channel type
set val(prop)     Propagation/TwoRayGround   ;# radio-propagation
model
set val(netif)          Phy/WirelessPhy                ;# network interface
type
set val(mac)            Mac/802_11                     ;# MAC type
set val(ifq)           Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)            LL                              ;# link layer type
set val(ant)           Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                             ;# max packet in ifq
set val(nn)             20                             ;# number of nodes
set val(rp)            TORA                            ;# routing protocol
set val(x)             1000                    ;# X dimension of topography
set val(y)             900                             ;# Y dimension of
topography
set val(stop)           200                            ;# time of  simulation

set ns              [new Simulator]
$ns color 2 Red
```

```
set tracefd       [open simplek1.tr w]
set windowVsTime2 [open win1.tr w]
set namtrace      [open simwrls.nam w]
set sno           [open seqno.tr w]
set ssth          [open ssthresh.tr w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo      [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

#
#  Create nn mobilenodes [$val(nn)] and attach them to the channel.
#
# Create channels
set chan_0_ [new $val(chan)]
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]
set chan_3_ [new $val(chan)]
set chan_4_ [new $val(chan)]
set chan_5_ [new $val(chan)]
set chan_6_ [new $val(chan)]
set chan_7_ [new $val(chan)]
set chan_8_ [new $val(chan)]
set chan_9_ [new $val(chan)]
set chan_10_ [new $val(chan)]
```

```
set chan_11_ [new $val(chan)]
set chan_12_ [new $val(chan)]
set chan_13_ [new $val(chan)]
set chan_14_ [new $val(chan)]
set chan_15_ [new $val(chan)]
set chan_16_ [new $val(chan)]
set chan_17_ [new $val(chan)]
set chan_18_ [new $val(chan)]
set chan_19_ [new $val(chan)]

# configure the nodes
        $ns node-config -adhocRouting $val(rp) \
                        -llType $val(ll) \
                        -macType $val(mac) \
                        -ifqType $val(ifq) \
                        -ifqLen $val(ifqlen) \
                        -antType $val(ant) \
                        -propType $val(prop) \
                        -phyType $val(netif) \
                        #-channelType $val(chan) \
                        -topoInstance $topo \
                        -agentTrace ON \
                        -routerTrace ON \
                        -macTrace OFF \
                        -movementTrace OFF \
                        -channel $chan_0_

  for {set i 0} {$i < $val(nn) } {incr i} {
            set node_($i) [$ns node]
            $node_($i) random-motion 0    ;# disable random motion
  }
```

```
# Provide initial location of mobile nodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 100.0
$node_(1) set Y_ 70.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 150.0
$node_(2) set Y_ 100.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 250.0
$node_(3) set Y_ 126.0
$node_(3) set Z_ 0.0

$node_(4) set X_ 500.0
$node_(4) set Y_ 463.0
$node_(4) set Z_ 0.0

$node_(5) set X_ 600.0
$node_(5) set Y_ 400.0
$node_(5) set Z_ 0.0

$node_(6) set X_ 350.0
$node_(6) set Y_ 463.0
$node_(6) set Z_ 0.0

$node_(7) set X_ 600.0
$node_(7) set Y_ 700.0
```

```
$node_(7) set Z_ 0.0

$node_(8) set X_ 600.0
$node_(8) set Y_ 463.0
$node_(8) set Z_ 0.0

$node_(9) set X_ 400.0
$node_(9) set Y_ 600.0
$node_(9) set Z_ 0.0

$node_(10) set X_ 605.0
$node_(10) set Y_ 500.0
$node_(10) set Z_ 0.0

$node_(11) set X_ 750.0
$node_(11) set Y_ 700.0
$node_(11) set Z_ 0.0

$node_(12) set X_ 900.0
$node_(12) set Y_ 100.0
$node_(12) set Z_ 0.0

$node_(13) set X_ 250.0
$node_(13) set Y_ 900.0
$node_(13) set Z_ 0.0

$node_(14) set X_ 850.0
$node_(14) set Y_ 463.0
$node_(14) set Z_ 0.0

$node_(15) set X_ 600.0
```

```
$node_(15) set Y_ 960.0
$node_(15) set Z_ 0.0

$node_(16) set X_ 350.0
$node_(16) set Y_ 700.0
$node_(16) set Z_ 0.0

$node_(17) set X_ 314.0
$node_(17) set Y_ 700.0
$node_(17) set Z_ 0.0

$node_(18) set X_ 600.0
$node_(18) set Y_ 800.0
$node_(18) set Z_ 0.0

$node_(19) set X_ 700.0
$node_(19) set Y_ 600.0
$node_(19) set Z_ 0.0

# Generation of node movements
$ns at 10.0 "$node_(0) setdest 100.0  50.0  5.0"
$ns at 30.0 "$node_(0) setdest 5.0  50.0  5.0"
$ns at 50.0 "$node_(0) setdest 50.0  50.0  5.0"
$ns at 180.0 "$node_(0) setdest 250.0  250.0  5.0"
$ns at 130.0 "$node_(0) setdest 455.0  150.0  5.0"
$ns at 150.0 "$node_(0) setdest 150.0  150.0  10.0"

$ns at 10.0 "$node_(1) setdest 200.0    160.0  10.0"
$ns at 15.0 "$node_(1) setdest 300.0    160.0  10.0"
$ns at 20.0 "$node_(1) setdest 100.0    160.0  10.0"
$ns at 160.0 "$node_(1) setdest 430.0   600.0  5.0"
```

```
$ns at 170.0 "$node_(1) setdest 600.0    160.0  5.0"
$ns at 190.0 "$node_(1) setdest 220.0    160.0  10.0"
$ns at 70.0 "$node_(0) setdest 100.0  250.0  5.0"
$ns at 80.0 "$node_(0) setdest 155.0  300.0  5.0"
$ns at 100.0 "$node_(0) setdest 250.0  50.0  5.0"

$ns at 30.0 "$node_(2) setdest 600.0  506.0 5.0"
$ns at 35.0 "$node_(2) setdest 300.0  300.0 10.0"
$ns at 30.0 "$node_(2) setdest 200.0  200.0 5.0"
$ns at 160.0 "$node_(2) setdest 600.0  506.0 10.0"
$ns at 175.0 "$node_(2) setdest 300.0  600.0 10.0"
$ns at 195.0 "$node_(2) setdest 600.0  200.0 5.0"

$ns at 20.0 "$node_(3) setdest 300.0   250.0  10.0"
$ns at 40.0 "$node_(3) setdest 160.0   300.0  10.0"
$ns at 80.0 "$node_(3) setdest 250.0   200.0  10.0"
$ns at 90.0 "$node_(3) setdest 150.0   400.0  10.0"
$ns at 100.0 "$node_(3) setdest 300.0   300.0  10.0"

$ns at 70.0 "$node_(4) setdest 600.0  500.0  5.0"
$ns at 45.0 "$node_(4) setdest 700.0  500.0  5.0"
$ns at 70.0 "$node_(4) setdest 300.0  500.0  5.0"
$ns at 70.0 "$node_(4) setdest 150.0  500.0  5.0"
$ns at 80.0 "$node_(4) setdest 300.0  200.0  5.0"
$ns at 120.0 "$node_(5) setdest 300  380  10.0"
$ns at 130.0 "$node_(5) setdest 400.0 470 10.0"
$ns at 60.0 "$node_(5) setdest 600.0  300.0  5.0"
$ns at 70.0 "$node_(6) setdest 460.0  500.0  5.0"
$ns at 45.0 "$node_(6) setdest 50.0  500.0  5.0"
$ns at 143.0 "$node_(6) setdest 800.0  500.0  5.0"
$ns at 180.0 "$node_(6) setdest 550.0  520.0  5.0"
```

```
$ns at 20.0 "$node_(7) setdest 300.0  400.0  5.0"
$ns at 80.0 "$node_(7) setdest 700.0  380.0  10.0"
$ns at 30.0 "$node_(7) setdest 300.0  250.0  5.0"
$ns at 80.0 "$node_(7) setdest 500.0  850.0  10.0"
$ns at 130.0 "$node_(7) setdest 970.0  450.0  5.0"
$ns at 140.0 "$node_(7) setdest 450.0  320.0  10.0"
$ns at 150.0 "$node_(8) setdest 100.0 250 10.0"
$ns at 180.0 "$node_(8) setdest 620.0  400.0  10.0"

$ns at 40.0 "$node_(8) setdest 300.0 250 10.0"
$ns at 100.0 "$node_(8) setdest 620.0  300.0  5.0"
$ns at 110.0 "$node_(9) setdest 750.0   800.0  15.0"
$ns at 130.0 "$node_(9) setdest 500.0   600.0  5.0"
$ns at 50.0 "$node_(9) setdest 350.0   500.0  15.0"
$ns at 70.0 "$node_(9) setdest 500.0   400.0  15.0"
$ns at 30.0 "$node_(9) setdest 200.0   400.0  15.0"
$ns at 40.0 "$node_(9) setdest 250.0   500.0  15.0"
$ns at 20.0 "$node_(9) setdest 250.0   300.0  15.0"

$ns at 100.0 "$node_(9) setdest 800.0   800.0  15.0"
$ns at 125.0 "$node_(9) setdest 600.0   716.0  5.0"
$ns at 136.0 "$node_(9) setdest 469.0   800.0  10.0"

$ns at 10.0 "$node_(10) setdest 221.0  150.0  15.0"
$ns at 30.0 "$node_(10) setdest 50.0  550.0  15.0"
$ns at 50.0 "$node_(10) setdest 50.0  150.0  15.0"
$ns at 180.0 "$node_(10) setdest 440.0  350.0  5.0"
$ns at 130.0 "$node_(10) setdest 224.0  700.0  5.0"
$ns at 150.0 "$node_(10) setdest 150.0  650.0  10.0"
```

```
$ns at 10.0 "$node_(11) setdest 100.0    100.0  10.0"
$ns at 15.0 "$node_(11) setdest 300.0    100.0  10.0"
$ns at 20.0 "$node_(11) setdest 100.0    400.0  10.0"
$ns at 160.0 "$node_(11) setdest 430.0   500.0  5.0"
$ns at 170.0 "$node_(11) setdest 300.0    160.0  5.0"
$ns at 190.0 "$node_(11) setdest 420.0    160.0  10.0"

$ns at 30.0 "$node_(12) setdest 300.0  506.0 15.0"
$ns at 35.0 "$node_(12) setdest 300.0  476.0 5.0"
$ns at 30.0 "$node_(12) setdest 200.0  600.0 15.0"
$ns at 160.0 "$node_(12) setdest 600.0  700.0 10.0"
$ns at 175.0 "$node_(12) setdest 200.0  600.0 10.0"
$ns at 195.0 "$node_(12) setdest 900.0  200.0 5.0"

$ns at 20.0 "$node_(13) setdest 300.0   800.0  10.0"
$ns at 40.0 "$node_(13) setdest 160.0   300.0  10.0"
$ns at 80.0 "$node_(13) setdest 250.0   600.0  10.0"
$ns at 90.0 "$node_(13) setdest 700.0   400.0  10.0"
$ns at 100.0 "$node_(13) setdest 500.0   300.0  10.0"

$ns at 70.0 "$node_(14) setdest 600.0  810.0  5.0"
$ns at 45.0 "$node_(14) setdest 700.0  500.0  5.0"
$ns at 70.0 "$node_(14) setdest 500.0  400.0  15.0"
$ns at 70.0 "$node_(14) setdest 150.0  100.0  10.0"
$ns at 80.0 "$node_(14) setdest 20.0  200.0  5.0"
$ns at 120.0 "$node_(15) setdest 200  380  10.0"
$ns at 130.0 "$node_(15) setdest 400.0 600 5.0"
$ns at 60.0 "$node_(15) setdest 410.0  200.0  15.0"
$ns at 70.0 "$node_(16) setdest 700.0  500.0  5.0"
$ns at 45.0 "$node_(16) setdest 50.0  390.0  15.0"
$ns at 143.0 "$node_(16) setdest 300.0  500.0  10.0"
```

```
$ns at 180.0 "$node_(16) setdest 400.0  651.0  5.0"

$ns at 20.0 "$node_(17) setdest 100.0  400.0  15.0"
$ns at 80.0 "$node_(17) setdest 500.0  210.0  10.0"
$ns at 30.0 "$node_(17) setdest 236.0  250.0  15.0"
$ns at 80.0 "$node_(17) setdest 500.0  431.0  5.0"
$ns at 130.0 "$node_(17) setdest 168.0  350.0  5.0"
$ns at 140.0 "$node_(17) setdest 278.0  472.0  10.0"
$ns at 150.0 "$node_(18) setdest 256.0 250 5.0"
$ns at 180.0 "$node_(18) setdest 236.0  789.0  10.0"

$ns at 40.0 "$node_(18) setdest 300.0 250 10.0"
$ns at 100.0 "$node_(18) setdest 620.0  300.0  5.0"
$ns at 110.0 "$node_(19) setdest 450.0   630.0  15.0"
$ns at 130.0 "$node_(19) setdest 500.0   800.0  5.0"
$ns at 50.0 "$node_(19) setdest 250.0   500.0  15.0"
$ns at 70.0 "$node_(19) setdest 500.0   800.0  5.0"
$ns at 30.0 "$node_(19) setdest 200.0   800.0  15.0"
$ns at 40.0 "$node_(19) setdest 368.0   789.0  15.0"
$ns at 180.0 "$node_(19) setdest 600.0   800.0  5.0"

# Set a TCP connection between node_(0) and node_(9)
set tcp [new Agent/TCP/Sack1]
$tcp set class_ 2
set sink [new Agent/TCPSink/Sack1]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(9) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]

$ftp attach-agent $tcp
```

```
$ns at 10.0 "$ftp start"

# Printing the window size
proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"

# Printing the ssthresh
proc plotssthresh {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set thresh [$tcpSource set ssthresh_]
puts $file "$now $thresh"
$ns at [expr $now+$time] "plotssthresh $tcpSource $file" }
$ns at 10.1 "plotssthresh $tcp $ssth"

# Printing sequence number
proc plotsequencenumber {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set snumber [$tcpSource set t_seqno_]
puts $file "$now $snumber"
$ns at [expr $now+$time] "plotsequencenumber $tcpSource $file" }
$ns at 10.1 "plotsequencenumber $tcp $sno"
```

```
# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 60
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 200.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace windowVsTime2 sno ssth
    $ns flush-trace
    close $tracefd
    close $namtrace
    close $windowVsTime2
    close $sno
    close $ssth
exec xgraph  -x Time -y Cwnd_size win1.tr  -geometry 800x400  &
exec xgraph  -x Time -y Seqno seqno.tr -geometry 800x400   &
exec xgraph  -x Time -y SSThreshold ssthresh.tr  -geometry 800x400  &
exit 0
}

$ns run
```