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

Investigating the Performance of TCP Variants and Routing Protocols in Mobile Ad Hoc Networks

A Dissertation

By

HENOCK MULUGETA

Submitted to the school of graduate studies of Addis Ababa University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In

Computer Engineering

Advisors:

Dr.Kumudha Raimond Dr.-Ing. Dereje Hailemariam

June, 2014

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

"Investigating the Performance of TCP Variants and Routing Protocols in Mobile Ad Hoc Networks"

By

Henock Mulugeta

ADDIS ABABA INSTITUTE OF TECHNOLOGY

APPROVAL BY BOARD OF EXAMINERS

Prof. Ho Yeol Kwon	
Dean, SECE, AAiT	Signature
Dr. Kumudha Raimond	
Advisor	Signature
Dr.-Ing. Dereje Hailemariam	
Co-Advisor	Signature
Dr. Nune Sreenivas	
Internal Examiner	Signature
Dr. Suresh Kumar	
Internal Examiner	Signature
Prof. Hyoung Joong Kim	
External Examiner	Signature

Dedicated to: Asegedech and Geni

ABSTRACT

Mobile Ad Hoc Network (MANET) is a collection of mobile nodes that can dynamically and randomly move and self organize to form network topology. MANETs have provided new challenges, which are the results of the unique characteristics of the wireless medium, the dynamic nature of the network topology, lacks of well secured boundaries, route failure due to frequent link breakages, wireless channel error, and using multi-path routes, which affects the end-to-end transmission of data. Transmission control protocol (TCP) performs poorly in such networks, since TCP's congestion control mechanism cannot distinguish between congestion and non-congestion related packet losses. TCP was previously developed for wired networks with the assumption that packet loss is an indication of congestion. However, in MANET TCP performs congestion control action for several types of losses that are not related to congestion. Consequently, when a packet loss is detected either by timeout or three duplicated acknowledgments, TCP slows down the sending rate by adjusting its congestion window size (*cwnd*) and unnecessarily retransmit a packet, which leads to lower throughput.

In this dissertation research, we have proposed a method for TCP to distinguish between packet losses due to congestion or route failure due to mobility of nodes. The proposed protocol is called TCP Packet Loss Detection and Response (TCP-PLDR). We have developed an analytical model of throughput of TCP with selective acknowledgment (TCP-SACK) and TCP-PLDR protocols as a function of packet loss probability and round trip time (RTT), in the presence of congestion and route failure losses. The model captures the behavior of TCP's congestion avoidance mechanism and its impact on throughput. Results have shown that TCP-PLDR is TCP friendly while it improves the throughput of TCP-SACK when there is a packet loss

due to route failure/change. Simulation was conducted using network simulator (ns-2) and results have shown that TCP-PLDR improves TCP-SACK's performance in MANET. As an example, simulation experiment for route failure and congestion loss scenario shows that TCP-PLDR improves the throughput of TCP-SACK on average by 39%. Moreover, the proposed protocol (TCP-PLDR) was evaluated in the presence of wireless channel error, and multi-path routing protocol like temporally-ordered routing algorithm (TORA) by making use of throughput, end-to-end delay, and packet delivery ratio (PDR) performance metrics and results showed that TCP-PLDR performed better than TCP-SACK.

As stated above, in MANET TCP is unable to distinguish packet losses due to congestion or route/link failures due to mobility of nodes. Hence the way how routing protocols respond to route failures and route recovery mechanism has an effect on the performance of TCP variants. In the second part of this dissertation paper, the effect of routing protocols; Ad hoc On Demand Distance Vector (AODV), Destination Sequenced Distance Vector (DSDV), Dynamic Source Routing (DSR), and Temporally Ordered Routing Algorithm (TORA), on the performance of TCP variants; TCP-Newreno, TCP-Reno, TCP-SACK, and TCP-Tahoe, under different mobility pattern and node density were studied thoroughly. Simulation results showed that, for all variants of TCP, AODV achieved the highest throughput. From TCP variants, TCP Newreno performed better than the other variants over the stated routing protocols. Besides, the best performing combination of routing protocols and TCP variants were identified. It is also confirmed that the performance of TCP variants are highly dependent on the underlying routing protocols in MANET. This result was taken as a valuable input to design and study the first part of this dissertation research. Moreover, performance of TCP-PLDR is compared with other variants of TCP (TCP-Reno, TCP-Newreno, TCP-Vegas, ATCP, and TCP-Westwood) and routing protocols (AODV, DSDV, DSR, and TORA). From TCP variants, in terms of throughput, TCP-PLDR performed better than the

other variants. Whereas, in terms of delay, TCP-Vegas outperformed the other variants. From routing protocols, TCP-PLDR can be coupled with AODV and it is confirmed that TCP-PLDR is more suitable for reactive routing protocol than proactive in MANET.

In MANET, packet drop attack due to malicious nodes can affect normal operation of routing protocols, performance of TCP, and performance of the network at large. The third part of dissertation study investigates the impact of malicious packet drop attack on the performance of two reactive TCP variants (Newreno and Sack) and one proactive TCP (Vegas) and two reactive routing protocols (AODV and DSR) by making use of throughput, end-to-end delay, and PDR performance metrics. Simulation was conducted by adding different percentages of malicious nodes in the network. Results showed that from TCP variants, Vegas outperformed Newreno and Sack in the absence of malicious nodes (0% malicious nodes). However, as the percentage of malicious nodes added in the network increases from 5% to 50%, Newreno and Sack performed better than Vegas though all of them are affected by malicious nodes. It is also noted that, even though both AODV and DSR protocols are highly affected by malicious nodes, AODV is more robust to malicious packet drop attack than DSR.

Finally, performance analysis of TCP-PLDR and TCP-SACK were evaluated exhaustively under security attack (malicious packet drop attack). Upon completion of exhaustive simulation, it is confirmed that TCP-PLDR is more robust to malicious packet drop attack than TCP-SACK.

Keywords: MANET, TCP-PLDR, TCP-SACK, Tahoe, Reno, Newreno, Vegas, Westwood, AODV, DSDV,DSR,TORA, route failure/change, packet loss, out-of-order packet, congestion, wireless channel error, multi-path route, malicious packet drop attack.

Acknowledgments

Foremost, I would like to thank God (α and Ω), his holy mother (Kidist Mariam), and beloved family, we did it. Without you it was impossible. Dear God I know that you know what is next.

I would like to express my deep gratitude and respect to my advisors Dr.Kumudha Raimond and Dr.-Ing. Dereje Hailemariam for their guidance, support, and encouragement. They gave me all their time, knowledge, patience, and advice. They always encouraged me to publish our research papers in reputable conferences and journals.

I would also like to thank my Ph.D research committee: Dr.-Ing. Getahun Mekuria, Dr.Frehiwot Woldehanna, and Dr.Dejene Ejigu for their valuable questions, comments, and encouragements in all stages of this thesis. In particular, I must acknowledge Dr.Dejene for giving me a chance to take courses with IT doctoral program and providing me an opportunity to publish one paper in ACM MEDES'12.

I am also indebted to Dr.Zhangyu Guan and Professor Tommaso Melodia from State University of New York for paying registration fee for IEEE ICCT'11 conference. They believed in my PhD research work and encouraged me optimistically without ever seen each other.

A very special thanks goes out to my father Mulugeta, my wife Yemisrach, and my angel daughter Mariamawit for their continuous love, endless support, and motivation. They were by my side by giving me extreme support throughout my Ph.D study.

Finally, my mother Geni, we really miss you. We suddenly lost you. We will never forget you. May your Soule rest in peace.

Table of contents

List of Tables

List of figures

List of Abbreviations and Acronyms

ABSE	Adaptive Bandwidth Shared Estimation
ACK	Acknowledgment
ADV	Adaptive Distance Vector
AODV	Ad hoc On-demand Distance Vector
AOMDV	Ad hoc On-demand Multipath Distance Vector
AWND	Advertized Window
ATCP	Ad-hoc TCP
BEAD	Best Effort Acknowledgment Notification
BER	Bit Error Rate
CBR	Constant Bit Rate
CPU	Central Processing Unit
CWND	Congestion Window
DoS	Denial of Service
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
ECN	Explicit Congestion Notification
ELFN	Explicit Link Failure Notification
EPLN	Early Packet Loss Notification
FIFO	First-In First-Out
FTP	File Transfer Protocol
HS-TCP	High Speed TCP
ICMP	Internet Control Message Protocol
IP	Internet Protocol
NS-2	Network Simulator 2
MAC	Medium Access Control
MANET	Mobile Ad hoc NETwork
MATCP	Modified Ad hoc TCP
MSS	Maximum Segment Size
NAM	Network Animator
OLSR	Optimized Link State Routing
OOO	Out-Of-Order
OTCL	Object Oriented Tools Command Language

PDR	Packet Delivery Ratio
QRY	Query
RFC	Request For Comments
RFN	Route Failure Notification
RERR	Route Error
RREP	Route Reply
RREQ	Route Request
RTO	Retransmission Time Out
RTT	Round Trip Time
RTTVAR	Round Trip Time Variation
RRN	Route Re-establishment Notification
SACK	Selective Acknowledgment
SMR	Split Multipath Routing
SSTHRESH	Slow Start Threshold
SRTT	Smoothed Round Trip Time
TCL	Tools Command Language
TCP	Transmission Control Protocol
TCP-AR	TCP Adaptive Retransmission Time Out
TCP-DOOR	TCP Detection of Out-of-Order and Response
TCP-DOOR-TS	TCP Detection of Out-of-Order and Response with Time Stamp
TCP-F	TCP Feedback
TCP-PLDR	TCP Packet Loss Detection and Response
TORA	Temporally Order Routing Algorithm
UDP	User Datagram Protocol
VANET	Vehicular Ad-hoc NETwork
WRP	Wireless Routing Protocol
WSN	Wireless Sensor Network

CHAPTER ONE

INTRODUCTION

MANET consists of wireless mobile nodes that can freely and dynamically to move and self-organize to form temporary network topology. Ad hoc network topologies permit users and devices to interconnect in areas with no pre-existing communication infrastructure [1]. Nowadays, MANET is well known to be used and driving a revolutionary change in our information communication technology (ICT). These days, we are moving from personal computer age (i.e., one computing device per person) to distributed computing age in which users can get information anytime anywhere (ubiquitous or pervasive computing). Several electronic platforms are currently devised so that users can access all the required information whenever and wherever needed. The mobile nature of these devices makes MANET the easiest and preferable solution for their interconnection.

Currently, mobile users can use their cellular phone to check e-mails, browse internet; travelers with portable computers (e.g., laptop, palmtop, etc.) can use internet in rural areas where base stations are not available; users can exchange files and other information by connecting portable computers and mobile devices through ad hoc networks; at home, users can transfer files between laptops and desktops. Generally, potential application of MANET includes situations where infrastructure is difficult to set up and deploy, such as battle field communication (soldiers, tanks, and planes); disaster relief operations (flood, earthquake, etc.); civilian environment such as meeting rooms, sports stadiums; personal area networks such as cell phones, laptop, earphone; in class room, ad hoc network can get established between

student PDA's and laptop of the instructor and many more applications [1, 2].

1.1 MANET: Characteristics, complexities, and design constraints

Some of the characteristics of MANET are: multi-hop routing with no default router (i.e., every node in MANET works as a router to forward each other's packet); infrastructure-less; no centralized administration to coordinated mobile nodes; mobility support for arbitrary movement of nodes; location and service discovery for distributed network topology control. These features make the estimation of retransmission time out (RTO), round trip time (RTT), and available network bandwidth unpredictable. Generally, the followings are some of the characteristics, complexities, and design constraints, which are specific to MANET [1,2,3,4]:

• *Autonomous and infrastructure-less:* MANET does not depend on any established infrastructure or centralized administration such as base stations or access point for their operation and interconnection.

• *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 nodes. In MANET, each node can be able to communicate directly with other nodes that reside within its transmission range. For communicating with nodes that reside out of this range, the node should use intermediate nodes to transmit the messages hop by hop.

• *Dynamically changing network topologies:* In MANET, nodes can move randomly and arbitrarily. Due to the random movement of nodes, the network topology changes frequently and unpredictably, which results in:

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

- *Device discovery*- Nodes can move in and out of the ad hoc networks arbitrarily. Therefore, each node should know and inform the existence of other nodes, which needs dynamic update to provide up-to-date route selection mechanism.

- *Bandwidth optimization*- Wireless links have basically lower capacity than the wired links.

- *Limited resources* -Mobile nodes depend on limited battery power, processor speed, and storage capacity.

- *Scalability* – Mobile network shall be able to provide all the services in the presence of large number of nodes.

- *Infrastructure-less and self operated* – Since there is no fixed infrastructure or base station that coordinates the operation of mobile nodes, each node should participate, cooperate, and acts as a router to manage and forward each other's packet.

- *Poor Transmission Quality* - high bit error rate (BER), which results from signal attenuation, is a typical characteristic of ad hoc networks.

- *Topology maintenance*- Since nodes are moving randomly and arbitrarily, each node should update and maintain their route information periodically to preserve the consistency of the network topology, which may consume much of the scarce power resource.

- *Distributed operation*: The decentralized nature of MANET requires that any routing protocol executes in a distributed manner.

- *Limited physical security* – In MANET, since the topology of the network changes dynamically and nodes can enter and leave the network without any authentication, it is very much vulnerable to different types of security attacks.

1.2 Background of problem

The unique characteristics of MANET presented in section 1.1 possess serious challenges to TCP. TCP is a reliable transport layer protocol, which is responsible for the end-to-end delivery of data packets over unreliable network layer. Many researchers have shown that the performance of current TCP's congestion control mechanism is inadequate in MANET. Generally, TCP's degraded performance in MANET can be attributed to a number of factors [1,2,3,4,5]:

At the MAC layer - wireless medium contention and hidden terminal problem can cause channel capturing, leading to link-layer packet dropping.

At the network layer - mobility often leads to route breakage and re-routing, which in turn disrupts TCP's transmission.

At the TCP layer - exponential RTO back-off amplifies the route breakage effect, and may result in large delay of TCP transmission blackouts.

Specifically, the performance of TCP degrades in MANET. This is because; TCP faces new challenges due to several reasons, which are specific to MANET. Some of the challenges are [6,7,8,9,10]:

TCP suffers from frequent route failures - route failure in MANET is a frequent event. If route fails while transmitting packets, then the underline routing protocol should re-establish the failed route. However, since TCP sender does not have mechanism to know the route re-establishment period, the throughput can degrade because of large idle time to transmit data packets. Also, if the newly established route is longer or shorter (in terms of nodes) than the old route, then TCP may not compute RTT appropriately, which leads to severe fluctuation in RTT estimation. Besides, packets may arrive at the TCP receiver side out-of-order. This is because, it is possible that

the end-to-end delay on the new route is shorter than the old route, so that a later packet using the new route arrives earlier than a packet being forwarded through the old route. This results in an out-of-order packet delivery. In this case, TCP assumes that packet is lost in transit, so it retransmits the lost packet unnecessarily and reduces the amount of packets to be transmitted, which once again affects the performance significantly.

Fig.1.1 shows how route failure due to mobility of nodes affects the performance of TCP. The green and yellow colored boxes are data and acknowledgment packets, respectively. If route fails while transmitting packets, packets can get dropped on old route due to link breakages. Then there is no throughput till route is repaired. Even if route is repaired, TCP's time out can get expired, which will further affects the throughput as there are no packets transmitted. Consequently, window size is reduced, and packets are retransmitted unnecessarily that leads to lower throughput.

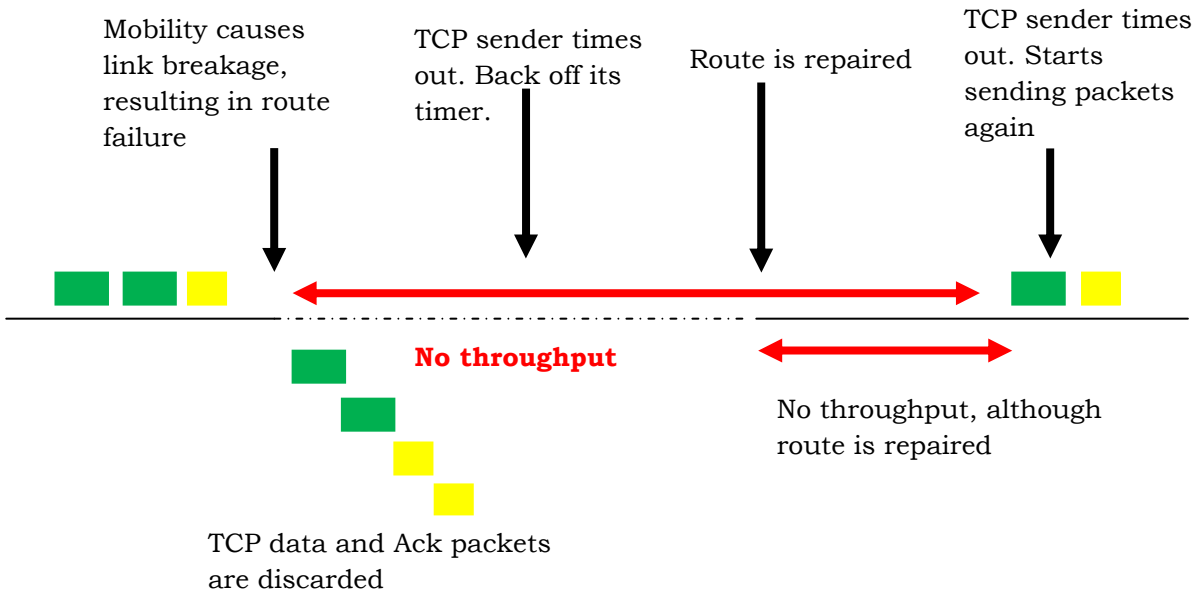


Figure 1: 1 Mobility causes link breakages, which results in lower throughput

TCP is unable to distinguish between losses due to route failures and network congestion - frequent route failures and route changes due to nodes mobility can cause reduction in throughput. Firstly, route failure can cause packet drops at intermediate nodes, which will be misinterpreted as congestion loss. Secondly, route change can introduce frequent out-of-order packet delivery too, which will further confuse the current TCP's congestion control algorithm.

Network Partitioning - It is due to nodes' mobility or energy-constrained operation of nodes. If the sender and receiver of a TCP connection lie in different partitions, all the sender's packets can get dropped. Consequently, the TCP sender invokes congestion control algorithm to reduce its window size inappropriately. Fig.1.2 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 MANET is nodes' mobility. Another factor that can create network partition is energy-constrained operation of nodes [1,2]. When D is moving away from C, the network can get partitioned. The network is reconnected when E is moving towards C.

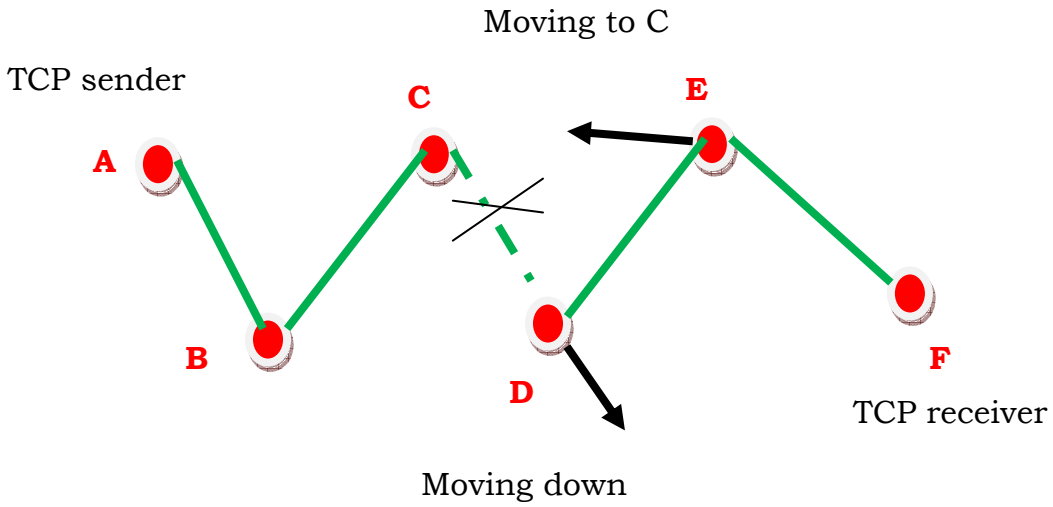


Figure 1: 2 Network partition scenarios

Power constraint - Since batteries carried by each mobile node have limited power supply, the life time of a given node is limited, which may bring in network partitioning. As stated above, if network partitioning happens, then packets can get lost. Once again, TCP sender can call inappropriate congestion control algorithm to reduce sending rate and retransmit packets. Moreover, since each node acts as a router as well as an end system, unnecessary retransmission of TCP packets consume this scarce power resource, which causes inefficient utilization of available power.

Multipath routing - routes in MANET are short-lived due to frequent link breakages. To reduce delay due to route re-computation, multi-path routing protocols such as temporally-ordered routing algorithm (TORA) maintain multiple routes between TCP pairs. In such a case, packets coming from different paths may not arrive at the receiver side in-order. Since TCP receiver is unaware of multi-path routing, it may misinterpret such out-of-order packet arrivals as a sign of network congestion.

Generally, multi-path routes affect TCP by two factors. The first one is inaccuracy of average RTO measurement that leads to more premature time outs than the standard one. The second one is out-of-order packet delivery via different paths. In fig.1.3, TCP sender tries to send three packets to TCP receiver. At some point during the course of transmission, link from sender to node A has failed. Packet 2 is then sent through route *sender-B-receiver*. Consequently, packet 2 reached to TCP receiver before packet 1, which creates out-of-order delivery of packets. However, in normal scenario packet 1 should arrive first followed by packet 2 and then packet 3, respectively. Generally, what out-of-order mean is that, when a packet sent earlier arrives later than subsequent packets.

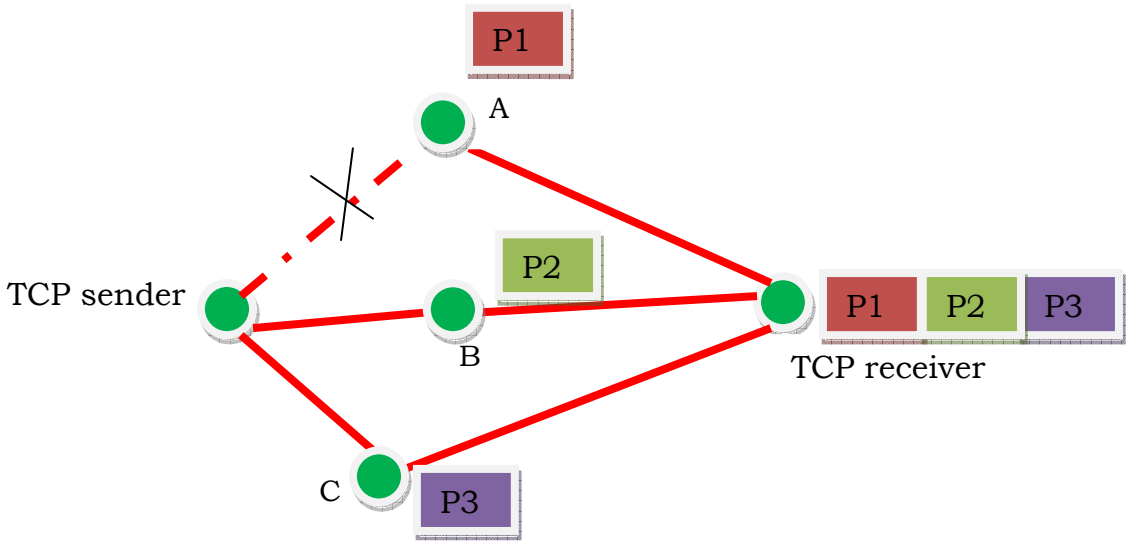


Figure 1: 3 Out-of-order deliveries of packets due to multi-path route

Wireless channel errors - wireless channel is error-prone compared to wired line because of fading, interference, and noise [2]. In MANET, the wireless channel suffers from high BER that cause frequent packet retransmission at MAC layer. In particular, during packet retransmission, if the wireless channel is still in a bad condition, several retransmission failures will be caused at the MAC layer.

MAC layer will react by first discarding the head-of-line frame, which is forwarded to the next hop, and then notifying the network layer about a link failure. Once the routing protocol detects a route failure, it initiates a route re-establishment process. Before a new route is found, no data packet can be sent out. Moreover, if the route re-establishment period is greater than the retransmission timer of TCP, TCP will assume congestion in the network, call exponential back-off algorithm, set its window size to one segment, and finally retransmit the lost packet. This retransmission can lead to wastage of scarce bandwidth and battery power. Most importantly, when a new route is found, the TCP throughput continues to be low for some time due to the fact

that now it is in slow start phase. Thus, false route failure induced by high BER in wireless channel unnecessarily initiates a route re-establishment process, and degrades TCP's performance in MANET.

Packet loss through security attack – it is known as *malicious node packet drop attack*. A malicious node participating in a routing path may purposely drop packets at the network layer in order to collapse network's performance [11,12,13,14]. A *selfish node* in the network can also deliberately drop packets to save its own battery life, CPU usage, and memory. In both cases, first the attacker node put itself politely in the routing path, and then it starts dropping packets.

If the attacker drops the whole packets it received from other node, then the attack is known as *black hole attack*. Another form is *gray hole attack*, in which the attacker drops packets randomly and selectively. In both cases, malicious nodes drops packets forwarded to them. Since TCP assume that all packets drop/loss as an indication of network congestion, it falsely retransmit the lost packet and reduce the packet sending rate, which degrades its performance significantly.

More specifically, while the malicious nodes drop the whole packet, TCP sender keeps retransmitting packets and call exponential back-off algorithm to double the RTO value. Consequently, TCP sender will remain idle for a long time, reduce its throughput, and finally leads to resetting the connection after subsequent timeouts.

All the aforementioned problems contribute to packet losses in ad hoc networks. In MANET, the major problem of TCP is its inability to differentiate among different types of packet losses. TCP was previously developed for wire-line networks where it assumes that all types of packet losses are due to congestion inside the network [8]. Consequently, upon the reception of either

three duplicate acknowledgments or expiration of RTO, which are an indication of packet losses, TCP slows down the sending rate by adjusting its window size and unnecessarily retransmit packets that degrade its performance a lot.

1.3 Objective of this dissertation study

The general objective of this dissertation research is to investigate the performance of TCP variants and routing protocols in MANET. Moreover, to design packet loss/out-of-order packet detection and response mechanism for TCP in MANET.

The specific objective of this dissertation research is:

1. To design a new algorithm for TCP to distinguish between packet losses due to route failure or network congestion in MANET.

    - At TCP receiver side, to detect whether packet loss occurred due to real congestion or route failure due to mobility of nodes.
    - At TCP sender side, to propose appropriate packet loss response mechanism based on the type of packet losses mentioned above.

2. To implement the proposed algorithm on the selected TCP variant (TCP-SACK) and type of routing protocol (AODV).

3. To analytically/theoretically model the proposed protocol and compare with TCP-SACK.

4. To implement and evaluate the proposed protocol in the presence of wireless channel error and multi-path routing protocol.

5. To evaluate and compare the proposed protocol with other TCP variants and routing protocols.

6. To evaluate and compare the proposed protocol with TCP-SACK in the presence of security attacks.

7. To study the effects of routing protocols on the performance of TCP variants in MANET.

8. To evaluate and select the best performing combination of TCP variants and routing protocols that goes with the proposed algorithm. The combination includes variants of routing protocol, variants of TCP, variation in mobility pattern, variation in traffic characteristics, and variation in node density.

9. To investigate the effects of malicious packet drop attack on the performance of TCP variants and routing protocols in MANET.

1.4 Contributions of this dissertation research

The results of this dissertation research show that TCP-PLDR improves the performance of TCP-SACK in MANET for all simulation scenarios. Traditional TCP has been designed to perform well in wired networks under the assumption that all types of packet losses are an indication of congestion. However, in MANET, TCP performs congestion control action for several types of packet losses that are not related to congestion. Consequently, it falsely retransmits the lost packet and reduces its sending rate, which degrades its performance a lot.

In this dissertation research, we have proposed TCP-PLDR protocol where the contributions are summarized as follows:

- TCP-PLDR distinguishes packet losses due to congestion or route failure by using time stamp option, which was not considered in previous works. And it is confirmed through analysis and simulation that timer based packet loss detection scheme (explicit loss detection scheme) is better than acknowledgment based scheme (implicit loss notification scheme).

- Throughput, end-to-end delay, packet delivery ration (PDR), congestion window size (*cwnd*), and RTO performance metrics showed that TCP-PLDR performed better than TCP-SACK protocol in all simulation scenarios.

- TCP-PLDR changes implicit acknowledgment based loss detection scheme into explicit timer based packet loss detection and notification scheme.

- Model and prove analytically/theoretically the steady state throughput of TCP-PLDR as a function of packet loss probability and RTT and compare the result with TCP-SACK. The model captures the behaviour of TCP's congestion avoidance mechanism and its impact on throughput. Results have shown that TCP-PLDR is TCP friendly while it improves the throughput of TCP-SACK when there is packet loss due to route failure, multipath routing protocol, and those due to wireless channel error in MANET.

- Extensive simulation was conducted to evaluate and compare TCP-PLDR with other TCP variants (Reno, Newreno, Vegas, ATCP, and Westwood) and results confirmed that TCP-PLDR showed significant performance improvement, in terms of throughput, than the other

variants, while it shows comparable performance in terms of delay and PDR performance metrics.

- The performance of TCP-PLDR was also evaluated in the presence of wireless channel error and multipath routing protocol and simulation results showed that it performs better than TCP-SACK.

In MANET, TCP is unable to distinguish packet losses due to congestion or route/link failures due to mobility of nodes. Hence the way how routing protocols respond to route failures and route recovery mechanism has an effect on the performance of TCP variants.

The second part of the dissertation research studied thoroughly the effects of three reactive routing protocols (AODV, DSR, and TORA) and one proactive routing protocol (DSDV) on the performance of four TCP reactive variants (Newreno, Reno, Sack, and Tahoe), under different mobility patterns and node densities. A comprehensive simulation studies demonstrates the complex interaction of TCP variants with routing protocols under the conditions of frequent route failure. It is also identified that the performance of TCP variants are highly dependent on the underlying routing protocols. Moreover, the best performing combination of routing protocols and TCP variants under different simulation scenarios were selected.

The above results revealed that every protocol performed differently in different MANET scenarios. Therefore, we have identified that using different combination of TCP variants and routing protocols leads to optimum performance than a single one. Moreover, we have confirmed through performance analysis that TCP is unable to distinguish between packet losses due to route failure or congestion. Hence, routing protocols have their own effect on the performance of TCP variants. This result was taken as a

valuable input to design and study the first part of this dissertation research. Finally, performance of TCP-PLDR is compared with other TCP variants.

The third part of the dissertation research investigated the impact of malicious packet drop attack on the performance of two reactive TCP variants (Newreno and Sack), one proactive TCP variant (Vegas), and two reactive routing protocols (AODV and DSR) by making use of throughput, end-to-end delay, and PDR performance metrics. Simulation was conducted by adding different percentage of malicious nodes in the network.

Finally, performances of TCP-PLDR and TCP-SACK were evaluated exhaustively under security attack (malicious packet drop attack). Up on completion of exhaustive simulation, it is confirmed that TCP-PLDR is more robust to malicious packet drop attack than TCP-SACK.

Parts of this dissertation research have been published in the following journals and conference proceedings [15,16,17,18,19].

1.5 Organization of the dissertation research

The rest of the dissertation research paper is organized as follows. Chapter 2 gives overview of TCP variants and routing protocols in MANET. Since the main part of the dissertation research mainly deals with congestion control algorithms of TCP, this chapter gives detail description of TCP variants and working principles of congestion control algorithm. Besides, descriptions of different routing protocols used in the simulation, which are relevant to the subsequent discussion are presented.

Chapter 3 gives detail description regarding literature review. Related works to improve performance of TCP in MANET are presented comprehensively,

which are classified as TCP with and without feedback approaches. These approaches are compared with their advantages and drawbacks. More specifically, descriptions of acknowledgment loss detection scheme with its drawbacks are presented. Related works for TCP variants, routing protocols, and malicious packet drop attacks are also presented in this chapter.

Chapter 4 gives TCP-PLDR protocol description. First TCP-PLDR algorithm both at the receiver and sender side is presented; analytical model of TCP-PLDR and TCP-SACK are also described comprehensively. Implementation of TCP-PLDR using network simulator (ns-2) is presented in detail. Moreover, the new approach is evaluated with the existing TCP variant and its relative merits are discussed and highlighted.

Chapter 5 discusses the performance evaluation of TCP variants and routing protocols under different mobility patterns and node densities. Moreover, performance of TCP-PLDR is compared with other variants of TCP and routing protocols and results are discussed and highlighted.

Chapter 6 presents simulation results and discussion of TCP variants and routing protocols under security attack. In particular, TCP-PLDR and TCP-SACK are compared by making use of different performance metrics and their results are also shown up when the network is under malicious packet drop attack.

Finally, chapter 7 gives summary and conclusions for this dissertation paper and recommend suggestions for future research directions and works.

CHAPTER TWO

TCP VARIANTS AND ROUTING PROTOCOLS IN MANET

2.1 Overview of TCP

TCP is a reliable transport layer protocol, which is responsible for the end-to-end delivery of data packets over unreliable network. The sending rate of TCP is controlled by congestion window size (*cwnd*), which limits the number of packets in flight, and slow-start-threshold (*ssthresh*), which is the boundary between slow start and congestion avoidance phases. TCP detects packet losses either by three duplicate acknowledgments (3 *dupacks*), which is faster and more efficient, or by expiration of retransmission timer, which is slower and less efficient. TCP is a window based flow and congestion control protocol. It uses sliding window protocol to manage its data transmission. The sending rate of a TCP connection is controlled by two mechanisms; flow control and congestion control [20,21,22,23,24].

Flow control - is used to avoid that a TCP sender doesn't overflow the receiver's buffer size. Therefore, the receiver should advertise in every acknowledgment transmitted a window limit to the sender by a variable known as advertised window (*awnd*). *awnd* is used to prevent the sender from overflowing the receiver's buffer.

Congestion control - is mainly dealt with data traffic inside the network. Its purpose is to prevent congestion inside the network when the TCP sender is faster than the network in transmitting or forwarding data packets (*cwnd*).

cwnd is used to prevent TCP sender from sending data packets beyond the capacity of the network.

Connection control - TCP sender controls and limits the sending rate by making use of both *awnd* and *cwnd*. Sending window (w) size of the sender is taken as the minimum value of *awnd* and *cwnd*.

$$w = \min(awnd, cwnd) \tag{2.1}$$

2.1.1 TCP's Congestion Control mechanisms

Currently, TCP's congestion control algorithm has four phases (mechanisms) [RFC-2581]. These are slow start, congestion avoidance, fast retransmit, and fast recovery [4,20,21,22].

Slow start – when connection begins, *cwnd* is equal to one maximum segment size (*cwnd =1MSS*). In slow start phase, the sender first sends one segment and wait for its acknowledgment. Afterwards, *cwnd* is doubled whenever acknowledgment is received. This means, *cwnd* growth exponentially until first packet loss event. The main drawback of slow start is large amount of time required during start up. Moreover, if the data packets that are supposed to be transmitted are very small, then the bandwidth efficiency will be reduced.

Congestion avoidance - during slow start phase congestion window increases exponentially. At some point during the connection if a bottleneck in the network happens, intermediate nodes will start discarding packets [4]. In normal TCP operation discarding packet is a sign of congestion. Beyond a certain threshold, *cwnd* is increased only by $\frac{1}{cwnd}$ each time an acknowledgment is received. This is an additive increment.

Fast retransmit - In the case for which RTO doesn't expire and TCP sender receives three duplicate acknowledgments, it means that a packet was lost, but other packets have already reached at the TCP receiver side. In this case, TCP sender will retransmit the lost packets without waiting for the expiration of RTO.

Fast Recovery - after fast retransmission, TCP sender performs fast recovery. In fast recovery, *ssthresh* is set to one-half of the current window size. *cwnd* will be set to *ssthresh* plus three (because of three duplicate acknowledgments received).

TCP uses two mechanisms to detect packet losses, which are an indication of congestion.

Retransmission Time Out (RTO) - 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 [RFC-2988]. Until RTT measurement has been made for a packet sent, TCP sender should set RTO to three seconds. When the first RTT measurement R is made, the sender must set:

$$SRTT = R, \quad and \quad RTTVAR = \frac{R}{2} \tag{2.2}$$

$$RTO = SRTT + \max(G, K * RTTVAR), where\ k = 4 \tag{2.3}$$

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

$$RTTVAR = (1 - \beta) * RTTVAR + \beta * |SRTT - R'| \quad (2.4)$$

$$SRTT = (1 - \alpha) * SRTT + \alpha * R' \quad (2.5)$$

α and β are normally set to $1/8$ and $1/4$, respectively. After the computation, the *RTO* must be updated as follows:

$$RTO = SRTT + \max(G, K * RTTVAR) \quad (2.6)$$

If *RTO* gets expired at the sender side, it means that network is severely congested. TCP sender enters into slow start phase and reduces *cwnd* to one segment, restart the retransmission timer, and retransmit the lost packet.

Three Duplicate acknowledgments - In normal TCP operation, if a packet is lost or delivered out-of-order at the receiver side, it would mean that packet is lost in transit due to congestion inside the network (mild congestion). The receiver shouldn't modify the sequence number of the incoming packets. However, it generates and sends duplicate acknowledgments to implicitly inform TCP sender that there is a packet loss. At the sender side, during congestion avoidance phase, reception of four back-to-back identical acknowledgments (referred to as three duplicate acknowledgments) cause the TCP sender to perform fast retransmit and to enter fast recovery algorithm. In fast retransmit, TCP sender does the following: 1) retransmits the lost packet. 2) Sets *ssthresh* to $\frac{cwnd}{2}$ and 3) sets *cwnd* to $ssthresh(new)$ plus 3 packets. On receiving a full acknowledgment, TCP sender sets *cwnd* to $ssthresh$, terminates fast retransmit, and resumes congestion avoidance.

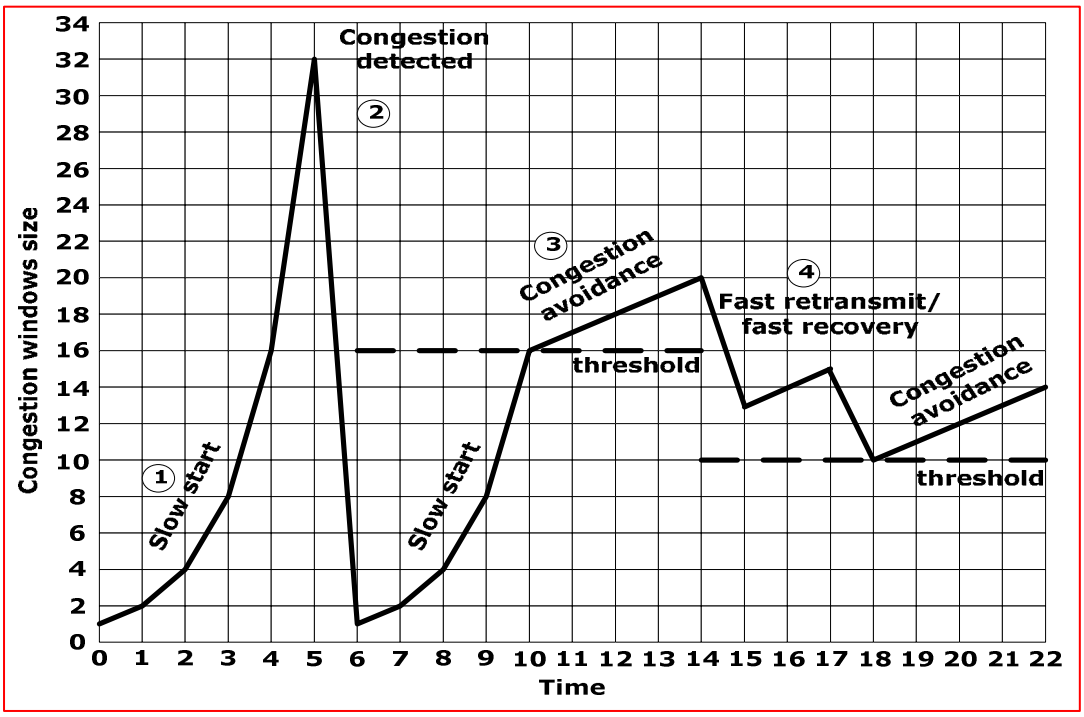


Figure 2:1 TCP congestion control mechanisms

Fig. 2.1 can be enlightened as follows. While TCP sender is in slow start phase, if it receives acknowledgment for previously unacknowledged data packets, then window size increments as follows:

$$cwnd = cwnd + MSS \tag{2.7}$$

However, if $cwnd > sstrhesh$, then it sets its state to congestion avoidance phase. This results in doubling of $cwnd$ for every RTT.

While TCP sender is in congestion avoidance state and if it receives acknowledgment for previously unacknowledged data packet, then TCP sender adjusts it window size as:

$$cwnd = cwnd + MSS * \left(\frac{MSS}{cwnd}\right) \quad (2.8)$$

Eqn.2.8 is an additive increase made by TCP sender, which results in increase of *cwnd* by 1 MSS for every RTT.

While TCP sender is in slow start or congestion avoidance phases and if packet loss is detected by three duplicate acknowledgments, then TCP's state variables are adjusted as follows:

$$sstrhesh = \frac{cwnd}{2}, \ and \ cwnd = sstrhesh + 3 \quad (2.9)$$

It then enters into congestion avoidance state where fast recovery is implemented. This is known as multiplicative decrease. In this case, *cwnd* will not go down below 1 *MSS*.

While TCP sender is in slow start or congestion avoidance phase and if timeout occurs, then it adjusts the state variables as follows:

$$sstrhesh = \frac{cwnd}{2}, and \ cwnd = 1MSS \quad (2.10)$$

It then enters to slow start phase. Finally, while it is in slow start or congestion avoidance phase and if it receives single duplicate acknowledgment, then it increments *duplicate count* for segment being acknowledged. Here *cwnd* and *sstrhesh* are not changed.

2.2 TCP variants

2.2.1 TCP Tahoe

TCP-Tahoe works based on three algorithms to control congestion inside the network namely; slow start, congestion avoidance, and fast retransmit algorithms [25,26,27]. The main drawback of TCP-Tahoe is, whenever packet loss occurs, it should enter into slow start phase where the *cwnd* starts from one, which wastes the scares bandwidth of ad hoc networks.

2.2.2 TCP-Reno

TCP-Reno maintains all the algorithms, which are stated in TCP-Tahoe [27,28,29,RFC-2001]. Moreover, one algorithm known as fast recovery was included in TCP-Reno. In case of fast recovery algorithm, reception of three duplicate acknowledgments from TCP receiver would mean that packet loss occurs in the network. Upon reception of three duplicate acknowledgments, TCP sender set $sstrhesh$ to $\frac{cwnd}{2}$ and $cwnd$ to $sstrhesh + 3$ (for three duplicate acknowledgments) that is defined in eqn. 2.9.

Fast recovery avoids the problem of entering slow start phase in case of packet loss by estimating outstanding packets in the network (packets sent but not yet acknowledged). With this mechanism, it overcomes the problem of inefficient utilization of available bandwidth, which was observed in TCP-Tahoe. However, if multiple packets lost from one window of data, TCP-Reno should wait for the expiration of RTO, then it retransmits the lost packet, and enters fast recovery algorithm. The main problem of TCP-Reno lies here, for each packet loss, Reno enters fast recovery phase, reduces $sstrhesh$, $cwnd$ and exits fast recovery phase up on the reception of partial acknowledgement. After frequent reduction, $cwnd$ becomes so small that there will not be

enough duplicate acknowledgements to initiate fast recovery algorithm. Hence the retransmission timer can get expired.

Generally, the major drawbacks of TCP Reno are: (i) when multiple packets lost from one window of data, TCP-Reno sender enter into slow start phase due to multiple RTO, which degrades the performance of the network a lot. ii) Since only one packet is retransmitted per RTT, it takes for TCP-Reno sender a long time to recover from congestion.

2.2.3 TCP Newreno

TCP-Newreno tried to overcome the problem that was seen in TCP-Reno by modifying fast recovery algorithm [30,31,RFC-3782]. The modification was done to avoid for TCP-Newreno sender that waits for the retransmit timer to expire when multiple data packets lost from one window of data. In case of TCP-Newreno, up on the reception of partial acknowledgment, the sender shouldn't come out of fast recovery algorithm. Rather, it retransmits the lost packet and stays in fast recovery phase till all the unacknowledged packets from one window of data are acknowledged. Therefore, reduction of *sstrhesh* and *cwnd* is totally avoided. Consequently, TCP-Newreno overcomes the problem of multiple reduction of *cwnd* from one window data by avoiding expiration of RTO multiple times, which was observed in TCP-Reno.

However, both Reno and Newreno are able to retransmit only one lost packet within one RTO when the sender is in fast recovery phase. More specifically, the major drawback of Newreno is that it retransmits at most one lost packet per RTT, which results in delay in retransmitting the later dropped packets in the window. Thus the available bandwidth may not be effectively utilized.

2.2.4 TCP with Selective Acknowledgment (SACK)

TCP-Sack modified fast retransmission and fast recovery algorithms, which were implemented in TCP-Reno [23,24,25,RFC-2018]. The problem that was observed in TCP-Reno and Newreno was, they only retransmit one lost packet in one RTO. However, in TCP-Sack multiple packets can be retransmitted in one RTO when multiple data packets lost from one window of data. If three duplicate acknowledgments received form TCP receiver, TCP-Sack invokes fast retransmit and fast recovery algorithms just like TCP Newreno does. It retransmits the lost packet and makes $sstrhesh = \frac{cwnd}{2}$, $cwnd = sstrhesh + 3$ as stated in eqn.2.9. Moreover, TCP-Sack uses a variable known as *pipe* to estimate packets that are sent but not yet acknowledged (outstanding packets) [23,RFC-2018]. If *pipe* is less than *cwnd*, then TCP-Sack can send new packet, it then sets$pipe = pipe + 1$. If lost packet is retransmitted and duplicate acknowledgments are received, then the value of *pipe* is set to $pipe = pipe - 1$. Finally, if partial acknowledgment is received, then $pipe = pipe - 2$.

$$\begin{Bmatrix} pipe = pipe + 1 & if\ pipe < cwnd \\ pipe = pipe - 1 & if\ dupacks \\ pipe = pipe - 2 & if\ partialack \end{Bmatrix} \quad (2.11)$$

2.2.5 TCP-Vegas

It uses proactive congestion avoidance mechanisms to increase and decrease the size of *cwnd* [28,29,32]. Rather than increasing the sending rate until a packet loss occurs, it tries to avoid packet losses by decreasing the sending rate when it senses early congestion even if there is no indication of packet losses. Other TCP variants assume that packet loss as a sign of congestion to adjust their sending rate. Whereas, TCP-Vegas use the difference in the *expected RTT* and the *actual RTT* to adjust the *cwnd* size [33]. Therefore, the

performance of TCP-Vegas is highly depends on the accuracy of RTT estimation.

Packet delay indicates a sign of congestion. In this condition, when a duplicate acknowledgment is received, the timestamp for the acknowledgment is compared to the timeout value. If the timestamp is greater than the timeout value, then it will retransmit rather than wait for three duplicated acknowledgments. It detects congestion at an initial stage based on increasing RTT values of the packets. It adopts a more sophisticated bandwidth estimation scheme based on RTT measurements. To estimate the available bandwidth and adjust *cwnd* accordingly, the differences between expected and actual transmission rates are taken into consideration. The *expected throughput* is computed as:

$$Expected\ throughput = \frac{cwnd}{baseRTT} \tag{2.12}$$

, where $baseRTT$ is the smallest observed RTT measurement for the connection, and *cwnd* is the number of bytes in-flight. The *actual throughput* is computed as follows:

$$Actual\ throughput = \frac{rttLen}{RTT} \tag{2.13}$$

, where RTT is the average RTT of the segments acknowledged during the last RTT, and *rttLen* is the number of bytes transmitted during the last RTT. The difference (*Diff*) between the two measurements is calculated in *baseRTT* segments as follows:

$$Diff = \left(\frac{cwnd}{baseRTT} - \frac{rttLen}{RTT}\right) * baseRTT \tag{2.14}$$

If the difference is under a certain threshold α, then the *cwnd* increases by a full segment size since there is evidence that the expected throughput is achievable and so the sending rate should increase. If the difference is above a (possibly different) threshold β, then this is taken as a sign of early congestion and the *cwnd* decreases by a full segment size. Otherwise, the *cwnd* remains unchanged. The decision process used to adjust the sending rate per RTT is given below:

$$cwnd = \begin{cases} cwnd + 1 & if\ Diff < \alpha \\ cwnd - 1 & if\ Diff > \beta \\ cwnd & if\ \alpha \leq Diff \leq \beta \end{cases} \quad (2.15)$$

In this way, it is able to utilize extra bandwidth without the network congestion and fluctuation in window size. In the original Vegas papers [34], α and β thresholds were set to 1 and 3 respectively.

2.3 Routing protocols in MANET

An immense number of routing protocols have been developed for MANET. These routing protocols are used to find route for transmission of data packets [35]. Nowadays, researches on routing protocols in MANET are in an infant stage. This is because; routing protocols shall work with the limitations of ad hoc networks, which are route failures, high power consumption, low bandwidth, high error rates, and arbitrary movements of mobile nodes [36].

2.4 Classification of routing protocols in MANET

The purpose of routing protocols in MANET is to find or search routes and maintain consistent route information by mobile nodes. Based on the

outlined functions, routing protocols in MANETs are classified as [35, 36,37,38]:

- Proactive, and
- Reactive routing protocols

2.4.1 Proactive routing protocols

These protocols are known as table-driven routing protocols. They try to maintain consistent and up-to-date routing information from each node to every other node in the network. They always have an updated routing table. They are mainly built up on the concept of link state and distance vector algorithms (Bellman-ford and Dijikistra algorithms).

The main drawback of table-driven or proactive routing protocols is whether a given route is needed or not, they should acquire and maintain this route in their routing tables, which may require exchanging of control messages periodically among nodes to update and maintain latest network topology information [35,37]. Moreover, these protocols are not suitable for large networks. This is because; as the dynamic nature of the mobile nodes changes the network topology dynamically and frequently, each and every node should exchange control packets periodically, which leads to more control overheads. Exchanging of more control messages consumes scares resources of MANET like bandwidth and power. Finally, frequent updating of routing table can consume the scarce bandwidth of ad hoc networks and there is an overhead to maintain up-to-date global network topology information, which leads to high energy consumption. In this dissertation paper, one of proactive routing protocol DSDV is used and studied.

2.4.2 Reactive routing protocols

Sometimes they are called source-initiated on-demand routing protocols [37,38,39]. These types of routing protocols obtain and maintain routes on their routing tables based on the request made by the source node. This means, routing paths are searched only when needed. Moreover, routing tables are updated only when they are active.

In reactive routing protocols, r*oute discovery, route maintenance,* and *route erasure* are the basic operations. When a source node requires a route to a destination, it initiates a route discovery procedure [39,40]. This process completes once a route is found or no route is available after searching all possible route combination. Once a route is established, it is maintained by a route maintenance procedure until the route is no longer needed. In MANET, since active routes may be disconnected due to nodes mobility, route maintenance is a critical issue for reactive routing protocols.

In reactive routing protocols, there is no need to create route in the absence of data traffic. Therefore, route creation and maintenance overhead are reduced compared to proactive routing protocols [39]. Thus, reactive routing protocols have better scalability than proactive routing protocols. However, when using reactive routing protocols, source nodes may get suffered from long delays for route searching before they forward data packets. The other drawback is, due to mobility of nodes, route failure is a frequent event. Since reactive protocols do not store routing information in their routing table, there is a delay in route re-computation, which may affect the performance of the network. In this dissertation research, some of reactive routing protocols like DSR, AODV, and TORA are used and studied.

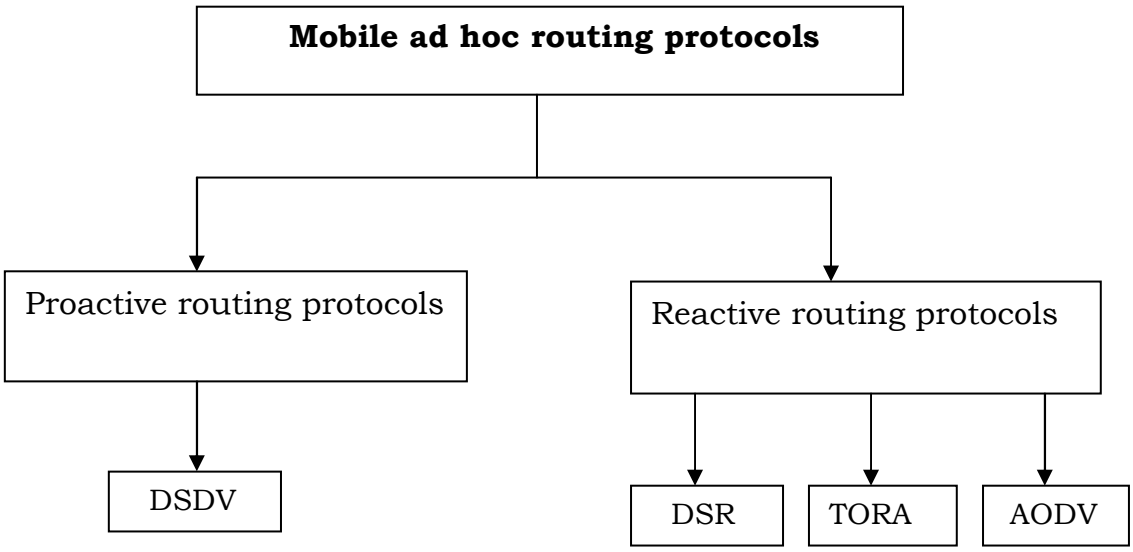


Figure 2:2 Classification of routing protocols in MANET

2.5 Protocols description

In this section, first some of proactive routing protocols then reactive routing protocols are presented.

2.5.1 Destination-Sequence Distance-Vector (DSDV)

DSDV is a table-driven proactive routing protocol, which is designed to operate in MANET. It is derived from the classical Bellman-ford routing algorithm with some modification suitable to adapt to ad hoc networks [39,40]. Main modification was done to make sure that loop-free route is established in routing table by making use of the concept of sequence numbers. Every node in the network maintains a routing table, which contains:

- The next node and
- Number of nodes required to reach to all reachable destinations.

Each routing table entry is labeled with a sequence number assigned by the destination node. Here, the sequence numbers are mainly used to categorize old routes from new ones and to avoid formation of route loops [41]. Usage of sequence numbering in DSDV can be illustrated as follows in fig. 2.3; a route ***R*** is considered more favorable than ***R'*** if ***R*** has a greater sequence number or, if the routes have the same sequence number but ***R*** has lower node-count.

Route loops can occur when incorrect routing information presents in the network after a change in the network topology, e.g., link failure. To overcome this problem, DSDV uses sequence numbers, which adapts to the dynamic network topology such as in an ad-hoc network.

DSDV protocol requires that each mobile node in the network must constantly advertise to each of its neighbors its own routing table. Since, the entries in the table my change very quickly, the advertisement should be done or broadcasted from each node to every other node frequently to make sure that every node can locate its neighbors in the network. This leads to exchanging of more control message, which wastes scares bandwidth of MANET.

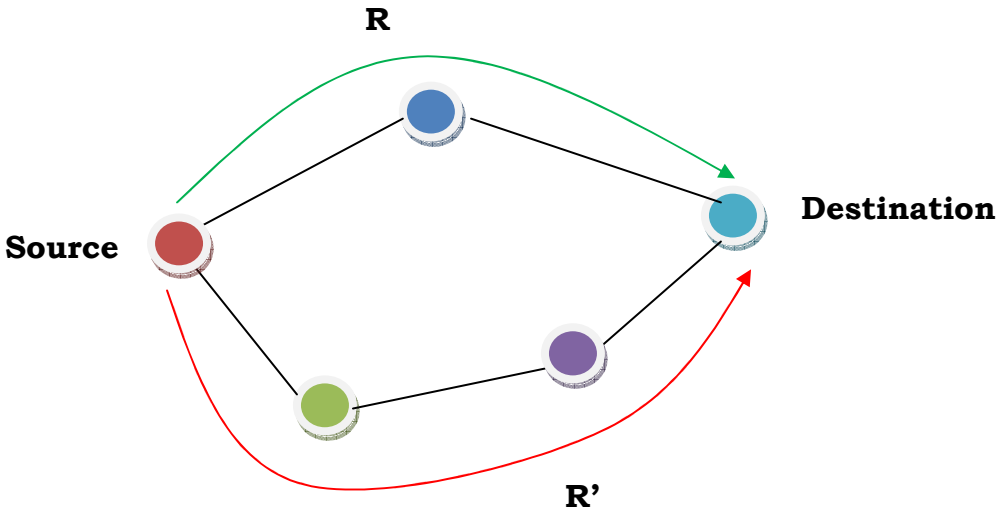


Figure 2:3 Route computations in DSDV

2.5.2 Dynamic Source Routing Protocol (DSR)

DSR is a source-initiated on-demand reactive routing protocol designed for ad hoc network [40,42,43]. Mobile nodes have routes in cache memory. When a source node wants to send data packet to a destination and if it does not have a known route to that destination on its routing table, the source node starts a route discovery procedure by broadcasting a *route request* (RREQ) packets. That is why it is categorized under reactive routing protocols. The protocol is composed of two main operations *route discovery* and *route maintenance*, which work together to allow nodes to discover and maintain routes to a given destinations [40]. When source node **S** has to send a packet to destination node **D** and if it has a route to the destination, it will use this route to send data packet. However, if it doesn't know the route, it initiates a route discovery procedure by broadcasting RREQ packet. RREQ packets propagate until destination or another node that knows the route to **D**. RREQ packet contains: the address of the destination, the source node's address and a unique identification number.

Every node that received the RREQ packet makes sure that whether it has a route to the destination. If it does not, it adds its own address to the *route record* of the packet and then forwards the packet through its outgoing links. Route record in each mobile node is built during route discovery procedure. Fig. 2.4 and 2.5 illustrates route discovery and reply procedures in DSR. When destination node gets the first RREQ, it replies with a *route reply* (RREP) message by following the route stored in RREQ from source to destination in reverse order. When the source node gets RREP, it stores the route in the cache memory. Finally, when source node sends data packets, it sends the whole route in the header (source routing). Intermediate nodes make use of this header to forward packets on the already established route. Here, the header increases with the length of the route.

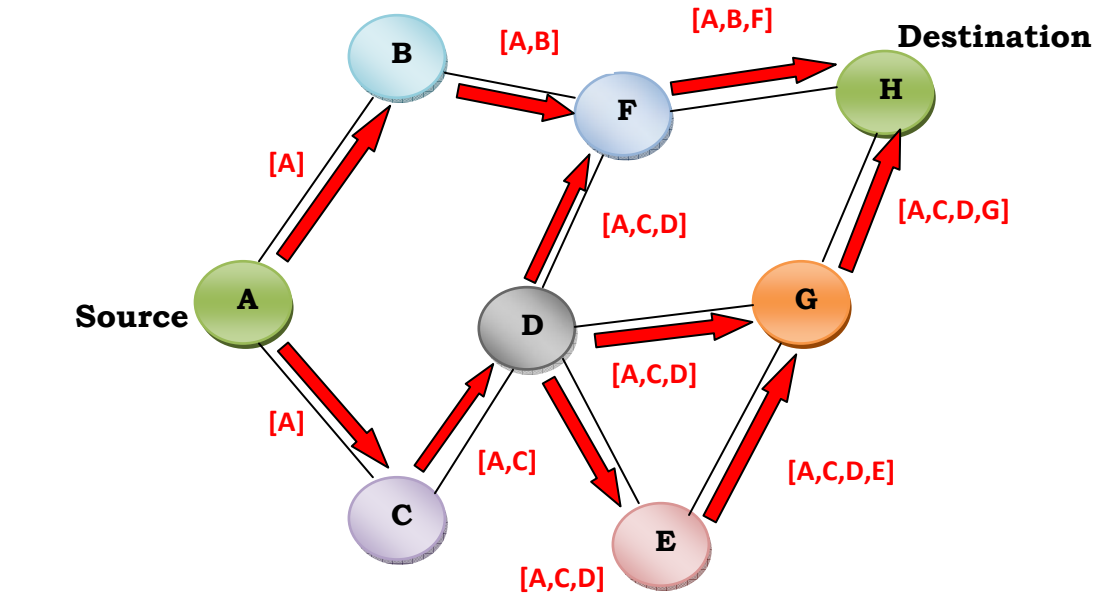


Figure 2:4 DSR - route discovery route record

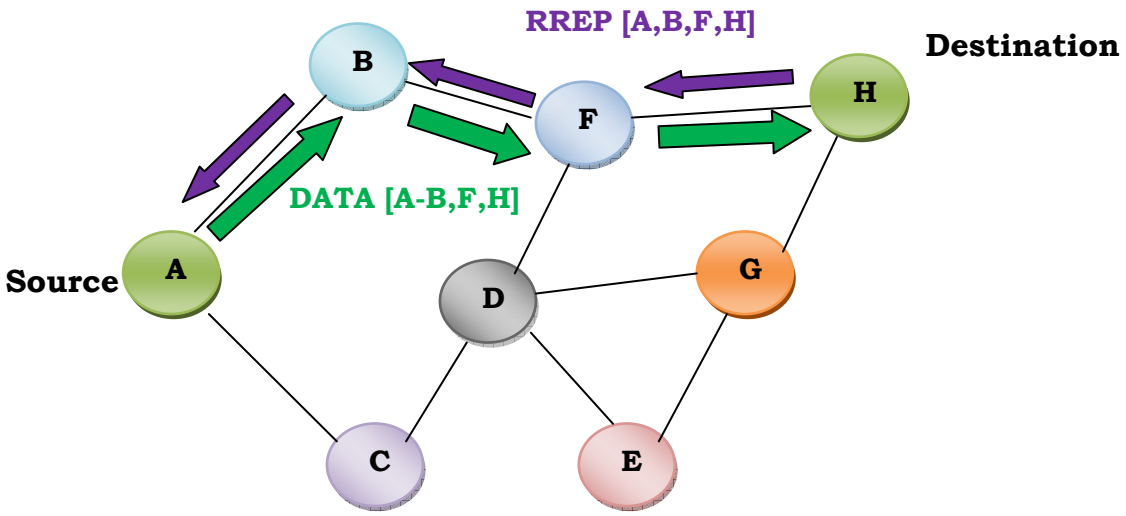


Figure 2:5 DSR - route reply with the route record and data transmission

One advantage of DSR is that unlike other protocols, no periodic routing packets are required, i.e. DSR does not use any periodic routing advertisement, which reduces exchanging of control messages. Being entirely on demand behavior and lack of periodic updating of routing information reduces packets overhead down to zero. Moreover, the use of cache reduces even more route discovering. A single route discovering can be used to learn routes to multiple destinations. However, old cache can adversely affect the performance. With passage of time and nodes mobility, cached routes may get invalid. Consequently, a sender node may try several outdated or stale routes, which are obtained from local cache, or replied from cached by another nodes before finding valid routes. Besides, since DSR discovers routes on-demand, it may have poor performance in terms of control overhead in networks with high mobility and heavy traffic loads. Scalability can be taken as another disadvantage of DSR, because DSR depends on blind broadcasts to discover routes.

2.5.3 Ad Hoc On-Demand Distance Vector (AODV) Routing

AODV is a reactive unicast MANET routing protocol [40,43,44,45]. AODV only maintains routing information about the active links. It takes most of the advantageous concepts from DSR and DSDV algorithms. The on-demand route discovery and route maintenance from DSR and node-by-node routing and usage of node sequence numbers from DSDV makes it suitable for MANET environment [38]. Basically AODV is an improvement on DSDV, since it minimizes the number of required broadcast messages by creating routes on-demand. In DSDV algorithm, a complete list of routes is maintained. Moreover, it tried to improve DSR in the following points:

- It maintains routing tables in order that packets should not contain destination route

- Routes have identification number to detect old ones
- Routes have life time and expire
- Each node has a list of neighbors that is useful to relay data
- When a route expires it is notified to neighbors through route error (RERR) packet
- It uses Hello packets to detect neighbor connectivity. When Hello packets do not arrive, it means connectivity failure.

AODV uses control packets such as RREQ, RREP, and RERR to manage routes between communicating nodes. Fig.2.6 illustrates route discovery in AODV. RREQ packets are broadcasts like DSR. When a source node wants to send packets to the destination but no route is available, it initiates a *route discovery* operation. In the route discovery operation, the source broadcasts RREQ packets. A RREQ message includes:

- Source-address
- Source-sequence no to maintain freshness information about the route to the source.
- Destination-address
- Destination-sequence no to specifies how fresh a route to the destination must be before it is accepted by the source.
- Node-count.

The source node broadcasts a RREQ message to its neighboring nodes, which in turn broadcast the message to their neighbors, and so on. Intermediate nodes store reverse routes back to the source node. Since an intermediate node may have many reverse routes, it always selects the route with the smallest number of nodes. The (source-address, broadcast-id) pair is used to identify the RREQ uniquely. Sequence numbers are important to make sure that loop-free and up-to-date routes are created. To reduce flooding overhead, a node discards RREQs that it has received before.

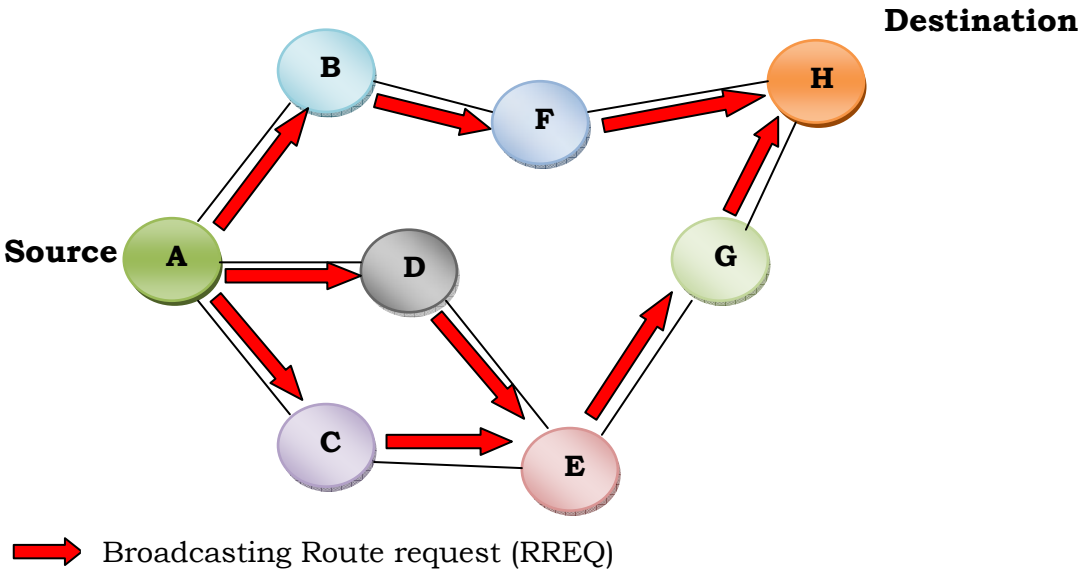


Figure 2:6 route discovery in AODV

Fig.2.7 illustrates RREP procedure in AODV. When a destination node receives one RREQ with fresh route, it replies with a RREP message. This RREP travels along the route established by RREQ but in inverse path to the source node. When the RREP message passes through intermediate nodes, these nodes update their routing tables, so that in the future, messages can be routed through these nodes to the destination.

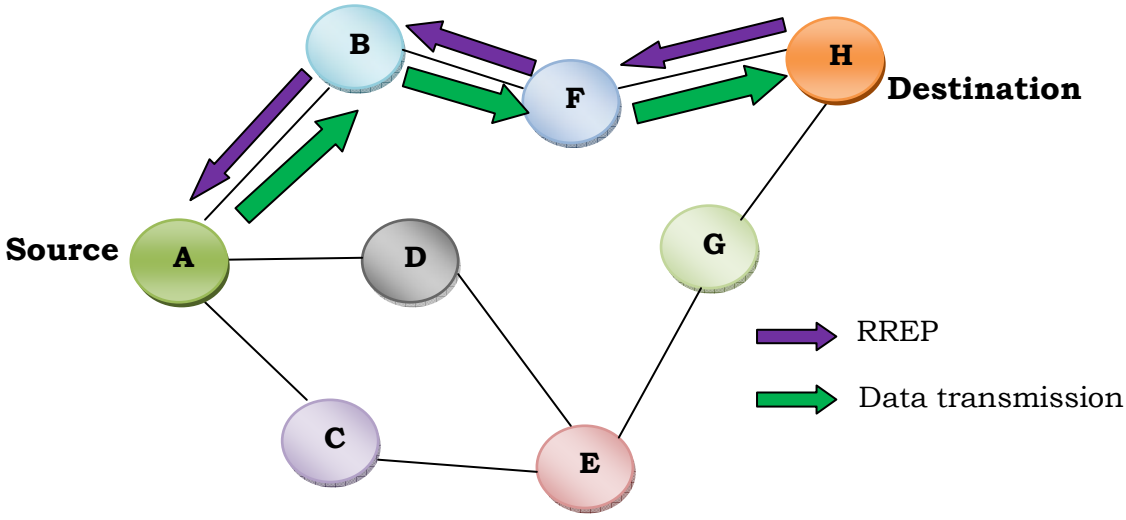


Figure 2:7 route reply path in AODV

2.5.4 Temporally-Ordered Routing Algorithm (TORA)

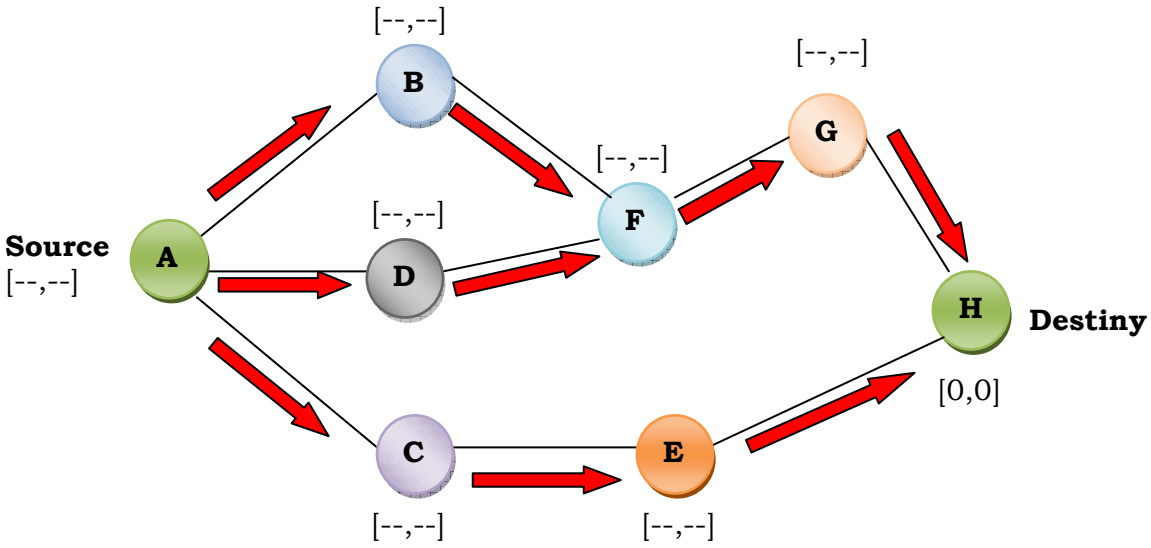
TORA is distributed multi-path routing algorithm based on the concept of link reversal [46,47]. TORA was proposed for highly dynamic mobile, multi-hop wireless networks. It is a source-initiated on-demand routing protocol. This means, the source is responsible for initiating for routing packets and the protocol is called whenever there is a need to route packets between sender-receiver pair.

TORA tries to find multiple routes from a source node to a destination node. The protocol has three basic functions: route creation, route maintenance, and route erasure. Each node has logical time of a link failure, the unique id of the node, which defines the new reference level, a reflection indicator bit; 0 means original level, and 1 means reflected level, a propagation ordering parameter to order nodes relative to reference level, and 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 link failure. The last two values define a *height* with respect to the reference level.

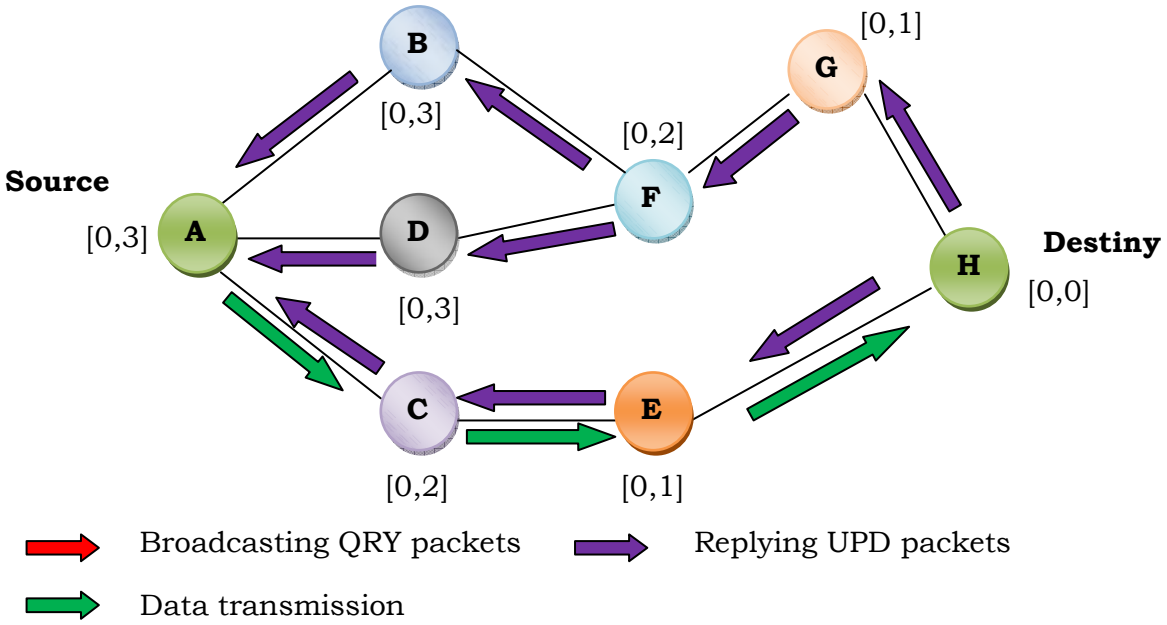
In TORA, route creation is performed using query (QRY) and update (UPD) packets. The route creation algorithm starts with the height (propagation ordering parameter) of destination set to 0 and all other node's height set to NULL (i.e. undefined). The source broadcasts a QRY packet with the destination node's id in it. A node with a NONE-NULL height replies with a UPD packet that has its height in it. A node receiving an 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. Fig. 2.8 illustrates a route creation process in TORA. As shown in fig. 2.8.a, source node **A** first propagates QRY message to its neighbors

towards destination node **H**. Node **F** does not propagate QRY from node **D.** This is because**,** it has already seen and propagated QRY message from node **B**. In figure 2.8.b, the source node **A** may have received a UPD from node **B** or node **C**. However, since node **C** sends a height with lower value than node **B**, node **A** take a route through node **C**.

In MANET, due to mobility of nodes, a route might get broken. In this case, route maintenance is required to re-compute a new route for the same source-destination pair. For example, as shown in fig. 2.9, when a link between node **F** and node **G** is broken, it generates a new reference level (the new reference level will be node **F**). This results in the propagation of that reference level by neighboring nodes. Links are reversed to reflect the change in adapting to the new reference level.



a) Propagation of QRY message



b) Height of each node updated as a result of UPD message

Figure 2:8 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.

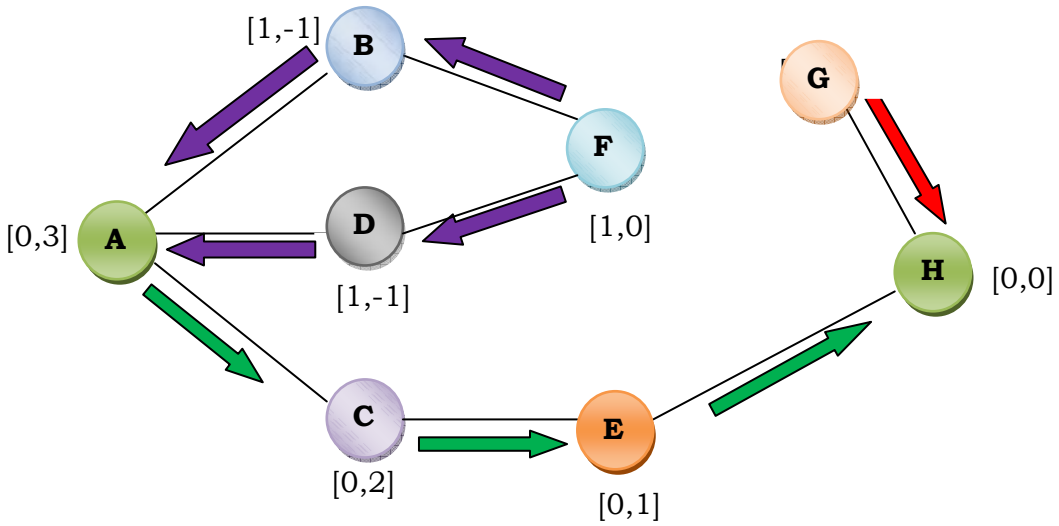


Figure 2:9 Re-computing new route on failure of link 5-7 (here the new reference level is node F)

CHAPTER THREE

RELATED WORKS

Literatures show that various researches have been conducted to improve the performance of TCP in MANET. The researches are classified into two groups [3,4,6,20,48].

3.1 TCP with feedback

Two or more layers cooperate together and exchange information to distinguish non-congestion related causes of packet losses. The feedback approaches basically help TCP to distinguish packet drop/loss due to real network congestion and other types of losses mentioned above such as wireless link errors, link contention, and route failures.

Explicit link failure notification (ELFN) [Holland and Viadya] tried to differentiate congestion from route failure based on feedback from lower layer [49]. The network layer gives feedback or information to the TCP sender about link and route failures so that it can avoid invoking congestion control algorithm. When a TCP sender receives an ELFN message, it disables timers, stop all data transmission, and finally send probing message to detect restored route. After finding a new route, the sender leaves the snooze state, restores its retransmission timers to their previous states and continues as normal. The advantage of this approach is it can avoid execution of congestion control upon link failure. The disadvantage is it uses old TCP states (*cwnd* and RTO) after route restoration. It also requires assistant from

intermediate node makes it complex for protocol design and deployment. It also solely depends on one routing protocol (DSR).

TCP-Feedback (TCP-F) [Chandran et al., 2001] is a feedback-based approach to handle route failures in MANET [50]. It allows TCP sender to distinguish between packet losses due to route failure or network congestion. It uses route failure notification (RFN) and route re-establishment notification (RRN) packets to detect packet loss due to route failure. When a node detects the failure of a route, it explicitly sends RFN message to TCP sender. On receiving the RFN, it goes into a snooze state until it is notified of the restoration of the route through RRN packet. On receiving the RRN, TCP sender will leave the snooze state and will resume transmission based on the previous sender window and timeout values as in the case of ELFN. The merit of this approach is it is able to handle route disruption at any wireless link. The drawback of this approach is there is a potential danger of failure if the RFN or RRN packets are lost. The resumed transmission rate may be set inappropriately. Moreover, it requires assistant from intermediate nodes.

Ad hoc TCP (A-TCP) tried to distinguish the problem of packet loss due to high BER, frequent route changes, network partitions, and packet reordering [51]. The main idea of this approach is to add a layer between TCP and IP at the sender's protocol stack. This ATCP layer controls the current TCP's state and monitors TCP from calling its congestion control mechanisms inappropriately. The TCP sender of ATCP has four states; *normal state*, which does nothing; *congested state* where it take congestion behavior of TCP; *loss state* where it put TCP in the persist state and send unacknowledged packets; *disconnect states* where it places TCP into the persist state and send probes to the destination node, when it leaves the persist state, slow start is invoked. It tries to listen to the network state by explicit congestion notification (ECN) and destination unreachable (ICMP) messages and put TCP sender into appropriate states mentioned above. TCP at the sender is frozen and no

packets are sent until a new route is found. The advantage of this approach is it is able to handle most of the problems related to wireless losses. The drawbacks are it requires assistant from intermediate nodes (costly to perform and respond slowly to non-congestive losses). It is inefficient in using the available bandwidth with frequent route changes and network partition. Moreover, ATCP introduces a thin ATCP layer between transport and network layers, which requires major modifications and cost to the network protocol stack. Also, it doesn't allow a source to send new packet when it is in the loss state [3].

Modified ATCP (MATCP) ATP has lossless links and fixed epoch time as its limitations [52]. In general, wireless links are lossy and bandwidth has impact on epoch time. The authors proposed MATCP protocol, which considers lossy wireless links and adjust epoch timer according to the different wireless bandwidths. Experiments were performed for different network size. Success rate and energy consumption of MATP is proved better than its existing counterpart.

The disadvantage of depending on lower layer (network layer) feedback is, it results in a re-designing of feedback support for each and every new ad hoc routing protocol. Moreover, the unreliable transmission of feedback to TCP sender further complicates the approaches based on network layer information [4].

3.2 TCP without feedback

This approach makes TCP to differentiate different types of packet losses and to take appropriate action without depending on feedback from lower layers, by taking into consideration that feedback mechanism may incur additional complexity and cost in ad hoc networks.

Fixed-RTO is a sender-based technique that does not depend on feedback from the network layer [53]. It distinguishes between packet losses caused by route failures and congestion by using RTO. When two timeout expire in sequence, TCP sender assumes route failure rather than congestion. The unacknowledged packet is retransmitted but RTO is not doubled a second time. This is in contrast with the standard TCP, where an exponential back-off algorithm is used. RTO remains fixed until route is re-established and retransmitted packet is acknowledged. However, as stated by the authors themselves, the assumption that two consecutive time outs are the exclusive results of route failure needs more analysis. Especially, in the case of congestion loss.

TCP Adaptive RTO (TCP-AR) in contrast to Fixed RTO, Haifa et al. proposed a new mechanism called TCP-AR that consists of adapting RTO value to network conditions [54]. They deploy the throughput filter already proposed in the adaptive bandwidth share estimation (ABSE) to distinguish route failure from network congestion. The path congestion level is determined by comparing the estimated rates to the instantaneous sending rate obtain from *cwnd*.

TCP detection of out-of-order and response (TCP-DOOR) [55] detects packet loss due to route failure by using additional sequence numbers (included in TCP header options). To detect out-of-order data packets, TCP sender uses a 2-byte TCP header option called *TCP packet sequence number* to count every data packet including retransmissions. For out-of-order packet detection, TCP receiver uses a *1-byte header option* to record the sequence in which duplicate acknowledgments are generated. Upon detecting out-of-order packets (internally or informed by the receiver), TCP sender responds by either: (1) temporarily disabling congestion control for a given time interval or (2) resetting the state to its value prior value to enter congestion avoidance phases. The drawbacks of this approach are, transmission rate may be set

inappropriately after a route change. It also fails to perform well in a congested network environment with substantial persistent packet reordering. Moreover, the assumption that out-of-order packet events are exclusive results of route failure needs much more analysis. This is because; multi-path routing protocols such as TORA and wireless channel error may produce out-of-order events that are not related to route failures.

Adaptive back-off response approach (ABRA), the authors explored a new adoption technique of TCP to frequent route failures without depending on feedback from the network layer [56]. It is based on adapting the TCP exponential back-off algorithm after the expiration of the retransmission timer. When the sender receives acknowledgment for a retransmitted packet, TCP returns to its state before the expiration of its retransmission timer. Moreover, *cwnd* and *sstrhesh* are divided by 2 and 4 respectively. Their study showed that this approach can improve TCP's performance in MANET.

In TCP-WELCOME [9] the authors proposed a sender based solution to distinguish packet loss due to route failure, wireless error, or congestion in MANET based on RTT. It distinguishes between packet losses causes and then triggers the most appropriate packet loss recovery according to the identified loss cause. It realizes its loss differentiation by observing the history of RTT samples evolution over the connection and the data packet loss triggers (three duplicated acknowledgments and RTO).

Backup path [57] - to improve path availability thereby reducing route failure, the authors studied TCP performance with an on-demand multipath ad hoc routing protocol named split multipath routing (SMR). Backup path routing actually uses only one path at a time and keeps the other paths as backup routes. TCP Reno is used. They implemented both SMR routing and fixed RTO scheme. It adapts the value of RTT value to retransmit the unacknowledged packet as soon as possible after route reestablishment.

So far we have seen various researches, which tried to solve the problem of TCP's inability to distinguish between congestion and non-congestion related packet losses. ELNF, TCP-F, FIXED-RTO, TCP-AR, TCP-DOOR, Backup path, ABPRA, and other proposals tried to deal with the problem of route failure and congestion loss in MANET. ATCP and TCP-WELCOME proposed to solve the problem of packet loss due to congestion, route failure, or wireless channel error. Generally, all the proposals tried to detect and recover from specific packet losses (i.e. route failure loss, or wireless error loss) in addition to congestion loss. TCP with feedback approach are based on explicit notification from lower layers (network layer, Link layer) to detect link/wireless channel failure. But they differ in how to detect these failures. TCP without feedback distinguish between packet losses induced by route failures, wireless channel errors, and congestion by employing end-to-end TCP layer approach. The main advantage of end-to-end solution is they don't require any explicit notification from lower layers. Besides, they don't require any cooperation from intermediate nodes to detect packet losses.

In general, TCP with feedback proposals reported better performance than TCP without feedback approach. However, TCP with feedback approach is more complex to design and deploy. TCP without feedback provide designing protocols independently makes it suitable for deployment. Moreover, those proposals dealt with route failure and congestion; ELFN, TCP-F, ATCP, Fixed RTO, TCP-DOOR, and TCP-AR, when route failure detected, TCP sender freezes its state variables (*cwnd*, RTO). Freezing TCP involves two steps: stop sending packets (new or retransmission), and freezing all timers, the sending window of packets, and the values of state variables. However, if link failures happen frequently, TCP will still suffer significant performance degradation even when the above proposals are applied. This is because; TCP should wait for route re-establishment without sending any new data packets. Moreover, all the proposed schemes uses acknowledgment based packet loss detection

scheme (implicit loss detection schemes), which is based on heuristics. Let's see briefly how loss detection schemes using three duplicate acknowledgments are used with its drawbacks in MANET where a significant packet loss is happened due to route failure, wireless channel error, using multi-path route, etc., than congestion.

3.3 Three duplicate acknowledgment loss detection and response scheme

TCP uses two mechanisms to detect packet losses. The first one is based on RTO. If RTO gets expired at sender side, it means network is severely congested. TCP sender enters into slow start phase and reduces *cwnd* to one segment, restart the retransmission timer, and retransmit the lost packet as stated in eqn.2.10. The second one is using three duplicated acknowledgments. In normal TCP operation, if packet is lost or not delivered in order at the receiver side, it would mean that packet is lost in transit due to congestion inside the network (mild congestion). The receiver shouldn't modify the sequence number of the incoming packets. Rather, it generates and sends duplicate acknowledgment to the sender to implicitly inform the TCP sender that there is a packet loss.

At the sender side, during congestion avoidance phase, receptions of four back-to-back identical acknowledgments (referred to as three duplicated acknowledgments) triggers the TCP sender to perform fast retransmit and to enter fast recovery algorithm. In fast retransmit, TCP sender does the following: 1) retransmit the lost packet. 2) Set $sstrhesh$ to $\frac{cwnd}{2}$ and 3) sets $cwnd$ to $sstrhesh(new)$ plus 3 packets as stated in eqn.2.9. Up on receiving a full acknowledgment, TCP sender sets $cwnd$ to $sstrhesh$, terminates fast retransmit, and resumes congestion avoidance. Fast retransmit/recovery algorithm can improve TCP's performance in the presence of irregular packet

reordering. However, drawbacks of using three duplicated acknowledgments as an indication of packet loss are [58,59]:

- During constant packet reordering, using three duplicated acknowledgments degrades TCP's performance.
- The fast retransmit/recovery mechanism may not detect all types of packet losses.
- The fast retransmit/recovery mechanism may not help if a retransmitted packet is also dropped.
- If there are subsequent out-of-order packets, TCP will call frequently fast retransmit/recovery algorithms, reduce *cwnd* and*sstrhes*. Finally enters to slow start phase. Hence, this leads to lower throughput.
- With implementation of cumulative acknowledgment used by TCP receiver, TCP sender doesn't know exactly which packets are lost.
- It may also generate false positives.
- False retransmission can cause unnecessary network load due to packet reordering.

3.4 TCP's Time Stamp Option Loss Detection Scheme (explicit loss detection scheme)

If this option is used, TCP receiver should explicitly inform TCP sender about lost or out-of-order packet delivery by using time stamp option. This mechanism requires modification both at TCP sender and receiver sides in order to negotiate how and when data packets are retransmitted and *cwnd* is reduced (perform fast retransmit/recovery operations). The implementation of TCP time stamp option is to detect out-of-order packets at TCP receiver side, and at the sender side, on receiving acknowledgment with out-of-order bit set

to one, it postpone triggering of congestion control algorithm for some period of time to identify whether data packet is lost or not based on timer than duplicate acknowledgment. The algorithm is implemented as an extension to TCP-SACK (chapter 4 gives detail description of the proposed algorithm).

Finally, Most of the proposed solutions are solely rely on simulation results (lack of theoretical analysis). Various researches have been proposed to model the throughput of TCP variants in wired networks [60],[61],[62],[63],[64],[65]. In [60] the authors proposed a stochastic model for the steady-state throughput of a TCP-Newreno bulk data transfer as a function of round-trip time and loss behavior. In [61] the authors presented analytic model to estimate the latency and steady-state throughput of TCP Tahoe, Reno, and Sack. They modeled the timeouts, evolution of *cwnd* during slow start, and the delayed acknowledgment timer. In [62] the authors modeled for evaluating utilization of TCP Tahoe using Markov chain model. They developed an algorithm to calculate the utilization and packet drop rate. However, analytical modeling of TCP variants is not well considered in MANET in the presence of mild congestion and route failure losses. In this paper, we proposed TCP-PLDR protocol to deal with packet loss issues in MANET. (Details are discussed and presented in chapter four).

Various researches have been carried out in the literature to study different routing protocols and TCP variants in MANET. The authors in [66] compared the performance of AODV, DSR, and WRP under different traffic scenarios. They used PDR, throughput, end-to-end delay, and routing message overhead performance parameters. The data packets are generated by CBR, TELNET and FTP. However, they have used only one TCP variant and they didn't test for different node density and mobility patterns.

In [67], the authors studied the performance of two TCP variants namely TCP-Reno and TCP-Vegas over AODV and optimized link state routing (OLSR)

protocol. First, they measured the performance of TCP-Reno over AODV and OLSR routing protocols for different node mobility patterns based on throughput. Simulation results showed that, OLSR outperformed AODV. They also measured the throughput of TCP-Vegas over AODV and OLSR routing protocols. Once again, OLSR showed maximum throughput than AODV. Finally, they measured the performance of TCP-Reno and TCP-Vegas over OLSR and AODV. Form the simulation result the authors observed that, TCP-Vegas had a higher throughput than TCP-Reno.

In [68], the authors evaluated routing protocols and performance of TCP variants in static and ad hoc networks. Routing protocols used are; DSDV, OLSR, AODV, and DSR. TCP variants used: TCP-Vegas, TCP Tahoe, TCP-Reno, TCP-Newreno, and TCP-Sack. Performance metrics: Throughput and expected throughput. From the author's simulation result, all TCP variants showed the highest performance for DSR routing protocol. However, they didn't test for different mobility patterns and node densities, besides multi-path routing protocol like TORA is not considered in their simulation. Finally, for detail simulation analysis TCP's state variables like *cwnd*, number of occurrences of RTO, time spent in RTO, number of fast retransmit and recovery calls made, and time spent in fast retransmission/recovery phases by TCP variants were not well considered in their study.

In [12], the authors studied the performance of the following TCP variants; TCP Reno, TCP Adaptive Westwood, Scalable TCP, HS-TCP, BIC – TCP, Fast TCP, under File Transfer Protocol (FTP) for high bandwidth – delay product. Only one routing protocol AODV was implemented in their simulation. The paper was supposed to choose the best TCP variant for a particular application. First, they studied the relationship between the number of packets transmitted and the bandwidth followed by the relationship between the number of packets transmitted and the propagation delay.

An investigation of TCP Reno over AODV, DSR and the Adaptive Distance Vector (ADV) routing protocols has been performed by Dyer et al. [53]. The ADV routing protocol has been shown to maximize TCP throughput, with AODV being the second best performer under various mobility conditions. Further, the throughput penalty of utilizing stale cache entries under moderate mobility has been noted in the case of DSR.

However, in MANET no researches have been conducted on an exhaustive study of the performance of TCP variants over different proactive and reactive routing protocols under different mobility patterns and network node densities. Therefore, the second part of this dissertation paper addresses this issues through extensive simulation comparison and evaluation in order to select the best performing combination of TCP variants and ad hoc routing protocols, which will be taking as a valuable input to the main part of the dissertation study (packet loss detection and response mechanism for TCP in MANET). Moreover, TCP-PLDR is evaluated over different proactive and reactive routing protocols and other well-known TCP variants. (Details are presented in chapter five).

Todd *et al.* [70] presented a survey on the current state of analysis techniques to prove or disprove security properties increases protocol confidence. The authors in [71] studied the impact of different mobility patterns on MANET vulnerability to distributed DoS attacks. Whereas, the authors in [72] studied a new kind of attack known as *rushing attack*, which results in DoS when it is used against all previous on-demand ad hoc network routing protocols like DSR, AODV, and secure protocols based on them such as Ariadne, ARAN, and SAODV. In [73], the authors investigated the impact of flooding attack on DSR protocol messages to network performance.

In [74] DoS attack was introduced and analyzed in AODV routing protocol. The attack spread was modeled by a semi-Markov process. Their analysis pointed out that the impact of DoS attack may be spread by the mobility of malicious nodes and results showed that it is faster in dense networks than in sparse networks. However, there is lack of researches, which investigates the effect of malicious packet drop attack on the performance of reactive and proactive TCP variants and routing protocols that adversely degrades their performance and network's performance at large. Moreover, we investigated performances of TCP-PLDR and TCP-SACK protocols under security attack (malicious packet drop attack) and differentiated which protocol is more robust to security attack. (Details are presented in chapter six).

CHAPTER FOUR

TCP-PLDR: PACKET LOSS DETECTION AND RESPONSE MECHANISM FOR TCP IN MANET

4. TCP-PLDR Protocol Description

Firstly, we have designed TCP-DOOR-TS [18] algorithm to detect out-of-order delivery of packet at the TCP receiver side due to multi-path routing protocol like TORA, and responding to the detected packet at the TCP sender side. The assumption made is that out-of-order packet is most likely caused by multi-path route from TCP sender to receiver. Moreover, we have designed TCP-PLDR algorithm, which is a revised and expanded version of TCP-DOOR-TS algorithm. TCP-DOOR-TS used time stamp option like TCP-PLDR to detect out-of-order packets at the receiver side. However, at TCP sender side, the main difference with TCP-PLDR is that in TCP-DOOR-TS algorithm:

1- On the reception of out-of-order bit set to one, for a time period T (T=1*RTT and T=2*RTT), it will disable some of the TCP's state variables like TCP-PLDR algorithm, which are discussed in algorithm 1 and 2.

2- In addition, on the reception of the first duplicate acknowledgment, once again for a time period T (T=1*RTT and T=2*RTT), TCP sender disable its states variables.

Disabling TCP's state variables two times for both case 1 and 2 mentioned above doesn't improve the performance of TCP as required. If out-of-order packets delivered frequently, which is a normal case in MANET; TCP will still

suffer significant performance degradation. This is because disabling period may exceed the normal RTO value so that with in all this time only one new packet is send, which may not efficiently utilize the scarce bandwidth efficiently in the case of consistent packet reordering. Section 4.3 gives analysis to select disabling period T=RTT. Moreover, unlike TCP-PLDR protocol, TCP-DOOR-TS use both timer and acknowledgment based loss detection schemes.

In TCP-PLDR algorithm, the following points are added, which was not considered in TCP-DOOR-TS algorithm:

- TCP-PLDR is mainly designed to distinguish between packet loss due to route failure and congestion.

- TCP-PLDR changes acknowledgment based loss detection scheme into pure timer based loss detection scheme (it doesn't use any duplicated acknowledgments to detect packet losses; drawbacks were stated in section 3.3).

- In TCP-PLDR, a detail analysis is made to select disabling period T=RTT (section 4.3 gives the detail).

- In TCP-PLDR, its throughput is analytically/theoretically modelled as a function of *RTT* and packet loss probability *p* (available in section 4.5).

- Its throughput is modelled in the presence of congestion loss and route failure loss and compared with TCP-SACK (section 4.5).

- TCP-PLDR is implemented and evaluated in the presence of wireless channel error and multi-path routes.

– TCP-PLDR is also implemented and evaluated in the presence of security attacks (black hole attack).

– TCP-PLDR is compared with different TCP variants (SACK, Reno, Newreno, ATCP, Vegas, and Westwood) and different routing protocols (AODV, DSDV, DSR, and TORA).

TCP-PLDR is an end-to-end packet loss detection and response mechanism proposed to improve the performance of TCP in MANET. It doesn't require any cooperation from intermediate nodes and other layers. Cooperation is made only at TCP sender and receiver sides making it pure end-to-end solution. TCP-PLDR is a modification of TCP-SACK protocol.

4.1 TCP-PLDR: Packet Loss or Out-of-order Packet Detection at the Receiver Side

In TCP-PLDR algorithm, the assumption made is that packet loss or out-of-order packet delivery is most likely caused by route failure/change, wireless channel error, multi-path route, or congestion. In order to detect packet loss or out-of-order packet delivery at TCP-PLDR receiver side, we used TCP's time stamp option [RFC-1323]. TCP-PLDR sender records the exact time whenever packet is sent to TCP-PLDR receiver by using its time stamp option. TCP-PLDR receiver can compare the time stamp in each packet with the previously received one to detect out-of-order packets. We used one variable (*ooo_option_bit_*) for out-of-order packet detection. If data packet is not delivered in-order for instance due to route failure/change, TCP-PLDR receiver will set *ooo_option_bit_* to 1 and will send to TCP-PLDR sender. Otherwise, if in order packet is received, TCP-PLDR receiver can send acknowledgment as TCP-SACK does. The following algorithm (algorithm 1) presents detection of packet loss or out-of-order packet delivery at TCP-PLDR receiver side.

Algorithm 1 Packet loss or out-of-order packet detection at TCP-PLDR receiver side

1. Initialize necessary variables (*present_pkt_ts_, previous_pkt_ts_, ooo_option_bit_*).
2. Retrieve current time stamp (*present_pkt_ts_*) from incoming packet, which is sent by TCP-PLDR sender.
3. Retrieve previously saved packet time stamp (*previous_pkt_ts_*), which is stored at TCP-PLDR receiver side.
4. Compare present packet time stamp with previously received packet time stamp. If present packet time stamp is less than previous packet time stamp i.e.,

   if (*present_pkt_ts_* < *previous_pkt_ts_*) {

   4.1 Set out-of-order option bit (*ooo_option_bit_*) to 1

   4.2 Send *ooo_option_bit_=1* with *ack* packet to be acknowledged to TCP-PLDR sender.

   4.3 Put the value of present packet time stamp (*present_pkt_ts_*) into previous packet time stamp (*previous_pkt_ts_*) for next comparison.

   }
5. If present packet time stamp is greater than previous packet time stamp, it means in sequence packet is received, TCP-PLDR receiver will send acknowledgments as TCP-SACK does.
6. Reset *ooo_option_bit_ = 0*, *present_packet_ts_ = 0* and other necessary variables.

// *ooo_option_bit_* is a variable that is available in TCP header file (tcp.h), which is common to both TCP sender and receiver.

Program 1 presents TCP-PLDR protocol module at the receiver side.

Program 1 TCP-PLDR packet loss detection at the receiver side (TCP-sink.cc)

```
          /*Receive packet function*/

void TcpSink::recv(Packet* pkt, Handler*)   {
                    …
    previous_pkt_ts_(0),present_pkt_ts_(0),
    ooo_option_bit_ (0);
    previous_pkt_ts_ =present_pkt_ts_;
    present_pkt_ts_=th->ts();
   if (present_pkt_ts_ < previous_pkt_ts_) {
        ooo_option_bit_ = 1;
        th->ooo_option() = ooo_option_bit_;
            }
      previous_pkt_ts = present_pkt_ts_;

            /*ACK the packet*/

void TcpSink::ack(Packet* opkt)    {
        packet* npkt = allocpkt();
                …
          if (ooo_option_bit_== 1)
           ntcp->ooo_option()=1;
          else
           ntcp->ooo_option()=0;
                …
           }
          /*Reset variables*/
   ooo_option_bit_  = 0, present_pkt_ts_ = 0;
           previous_pkt_ts_ = 0; }
           th->ooo_option() = 0;
           packet::free(pkt);
               …
              }
```

4.2 TCP-PLDR: Packet Loss or Out-of-order Packet Response Algorithm at the Sender Side

Once out-of-order packet is detected, TCP-PLDR receiver informs TCP-PLDR sender by setting out-of-order bit to 1 (*ooo_option_bit_=1*) as stated in algorithm 1. Up on the reception of *ooo_option_bit_=1*, for a time period T, TCP-PLDR sender will postpone calling of fast retransmission/recovery algorithms, disable some of the state variables (duplicate acknowledgment, timeout), adjust *cwnd*, and send one new packet with the assumption that within this period T, out-of-order packet may get delivered at receiver side. Here out-of-order packet can be delivered in two scenarios:

First scenario, packet can be lost due to route failure and it can be retransmitted by link layer but may deliver out-of-order packets (link layer acknowledgment is used in IEEE 802.11; the analysis is also presented in section 4.3) and during this period T, link layer can retransmit the lost packet with the assumption that packet is not lost due to congestion.

Second scenario, subsequent packets can get delivered out-of-order due to route change (new routes). However, during this period T, if out-of-order packet is not reached at the receiver side, then the assumption that out-of-order packet delivery due to route failure and route change is changed, packet is lost in transit due to congestion; fast retransmit/recovery algorithms will be triggered. Algorithm 2 describes packet loss response mechanism at TCP-PLDR sender side.

Algorithm 2 Packet loss or out-of-order packet delivery response at TCP-PLDR sender side

1. Retrieve acknowledgment packet and check for out-of-order bit (*ooo_option_bit_*)
2. If out-of-order bit is set to 1 (*ooo_option_bit_=1*), then TCP-PLDR sender does the following. For a time period T, TCP-PLDR sender:
   - 2.1 Temporarily disables some of the state variables like duplicate acknowledgment (*dupacks_=0*) and RTO (*timeout_=false*).
   - 2.2 Adjusts window size for *cwnd* to evolve based on the network condition by calling *opencwnd()* algorithm and
   - 2.3 Send one new packet by calling *send_one()* algorithm.
3. During this period T, if out-of-order packet is not acknowledged by the receiver (it means packet is lost in transit due to congestion), then disable period will get expired and call fast retransmission/recovery algorithms.
4. During this period T, if out-of-order packet is acknowledged by receiver, disable period will be expired and resume normal operation.
5. If out-of-order bit is set to 0 (*ooo_option_bit_=0*), then TCP-PLDR sender will send packets as TCP-SACK sender does.

Program 2 presents the module at TCP-PLDR sender side.

Program 2 TCP-PLDR packet loss response at the sender side (TCP-sack1.cc).

```
  /* Receive function to receive ack from receiver*/

void Sack1TcpAgent::recv(Packet *pkt, Handler*)
        {
        hdr_tcp *tcph = hdr_tcp::access(pkt);
                          …
            /* Disabling state variables*/
       if (tcph->ooo_option() == 1)
          {
disableperiod = (int)(((int(t_srtt_) >>
T_SRTT_BITS)*tcp_tick_)/(int(t_rtt_)*tcp_tick_) *
window());
    for (int i = 0; i <= disableperiod; i++)
            {
          dupacks_ = 0;
          timeout_ = false ;
      /*Increase the size of congestion window */
             opencwnd()
                {
          if (cwnd_ < sstrhesh)
              {
               cwnd_+=1;
            }
           else
               {
            increment = increase_num/cwnd;
            cwnd_+=increment;
          }
```

```
          if (maxcwnd && ( cwnd_ > maxcwnd))
                cwnd_=max_cwnd_;
                }
/*Send one new packet with sequence number t_seqno_*/
            send_one()
                {
if (t_seqno_ <= highest_ack_ + wnd_ && t_seqno_ <
curseq_ &&
t_seqno_ <= highest_ack_ + cwnd_ + dupacks_ )
        {
         output(t_seqno_, 0);
          t_seqno_ ++ ;
               }
            }
         }
/*call fast retransmission and recovery algorithms*/
  else
      dupack_action();
             …
          }
```

As in the case of naïve TCP-SACK, TCP-PLDR adjusts *cwnd* dynamically to efficiently utilize the available bandwidth. It uses *opencwnd()* algorithm to increase the size of *cwnd*. The increasing process depends on *cwnd* and *ssthresh* state variables. When network resource is underutilized, TCP-PLDR increases the transmission rate by opening the congestion window. In the slow start phase, where *cwnd* is less than *ssthresh* , TCP-PLDR sender increases *cwnd* by one for every received acknowledgment packet. On the

other hand, if *cwnd* is greater than *ssthresh* , TCP-PLDR sender is in congestion avoidance phase where *cwnd* is increased by $\frac{1}{cwnd}$ for every received acknowledgment packet. *increase_num* is usually set to 1. *cwnd* is bounded within *max_cwnd_*, which is the predefined maximum congestion window size. In TCP-SACK, *opencwnd()* is called whenever new acknowledgment packet is received. However, in TCP-PLDR *opencwnd()* is also called within disabling period with the assumption that during this period, network is not congested but packet is delivered out-of-order due to route failure, wireless channel error, or multi-path route.

The *send_one()* function is designed to send one new packet during fast retransmit phase to make sure that TCP sender should send out a new packet for every received duplicated acknowledgment packet. It prepares sequence number starting at *t_seqno_* and passes to function *output(t_seqno_, 0)* in order to create and transmit one new packet.

In TCP-PLDR algorithm, the reason for sending one new packet within disabling period *T* is that receiving *ooo_option_bit_* from the TCP-PLDR receiver would mean that in normal TCP-SACK operation, TCP sender receives one duplicate acknowledgment (*dupacks_ = 1*), and up on the reception of one duplicated acknowledgment, only one new packet is sent. Basically, it helps to efficiently utilize the available bandwidth.

Disabling state variables (*dupacks_ = 0* and *timeout_ = false*) within the specified period *T* allows out-of-order packets to reach to TCP-PLDR receiver. However during the above period *T*, if packet is not delivered to TCP-PLDR receiver side, then the assumption that packet loss due to route failure is changed, packet is lost in transit due to congestion inside the network. Consequently, congestion control algorithm will be invoked and packet that is lost will be retransmitted by using fast retransmission/recovery algorithm (*dupack_action()*) as TCP-SACK does. Since RTO estimation is computed

based on RTT value, disabling period T is derived from the ns-2 variables RTT. Moreover, disabling period is selected to be one times RTT. Why?

4.3 Analysis of Selecting Disabling Period *T=RTT*

TCP sender calculates $RTO(k)$ for the k_{th} packet based on RTT value given by *T(k)* [RFC-2988]. A TCP sender maintains two state variables to compute *RTO(k)*. The smoothed-round trip time (SRTT) given by $Tavg(k)$ and the round-trip time variation (RTTVAR) given by $\lambda t(k)$. Besides, a timeout granularity of 0.2 seconds is assumed in the computation. The computation of SRTT, RTTVAR and RTO are as follows: Until the first RTT measurement is made for a packet sent from TCP sender to receiver, TCP sender should set RTO to 0.2 seconds. When the first RTT measurement $T(k)$ is made, the sender must calculate SRTT and RTTVAR as:

$$Tavg(k) = T(k), and\ \lambda t(k) = \frac{T(k)}{2} \tag{4.1}$$

$$RTO(k) = min\{ub, max\{lb, \gamma[Tavg(k) + 4\lambda t(k)]\}\} \tag{4.2}$$

When subsequent RTT measurement $T(k+1)$ is made, TCP sender must update the state variables as follows:

$$Tavg(k+1) = \alpha * Tavg(k) + (1-\alpha) * T(k+1),\ \ \alpha\epsilon(0,1) \tag{4.3}$$

$$\lambda t(k+1) = \beta * \lambda t(k) + (1-\beta) * |T(k+1) - Tavg(k+1)| \tag{4.4}$$

α and β are normally set to $7/8$ and $3/4$, respectively. After the computation, $RTO(k+1)$ is calculated as:

$$RTO(k+1) = min\{ub, max\{lb, \gamma[Tavg(k+1) + 4\lambda t(k+1)]\}\} \tag{4.5}$$

,where lb is lower bound and its value is set to 0.2 seconds in ns-2, and ub is an upper bound and its value is set to 10^5 seconds in ns-2. γ is initialized to 1 and it is doubled when timeout gets expired to exponentially back-off the RTO value and it is reset to 1 upon the reception of acknowledgment. Eqn. 4.3 – 4.5 can be re-written as:

$$Tavg(k+1) = \frac{1}{8}\left(7Tavg(k) + T(k+1)\right)$$

$$= \frac{1}{8}(7Tavg(k) + Tavg(k) + T(k+1) - Tavg(k))$$

$$= \frac{1}{8}(8Tavg(k) + \Delta) \quad (4.6)$$

$$\lambda t(k+1) = \frac{1}{4}(3\lambda t(k) + |\Delta|)$$

$$= \frac{1}{4}(3\lambda t(k) - 4\lambda t(k) + 4\lambda t(k) + |\Delta|)$$

$$= \frac{1}{4}(-\lambda t(k) + 4\lambda t(k) + |\Delta|) \quad (4.7)$$

$$RTO(k+1) = \gamma * [T(k+1) + 4\lambda t(k+1)] \quad (4.8)$$

, where Δ is given as$T(k+1) - Tavg(K)$.

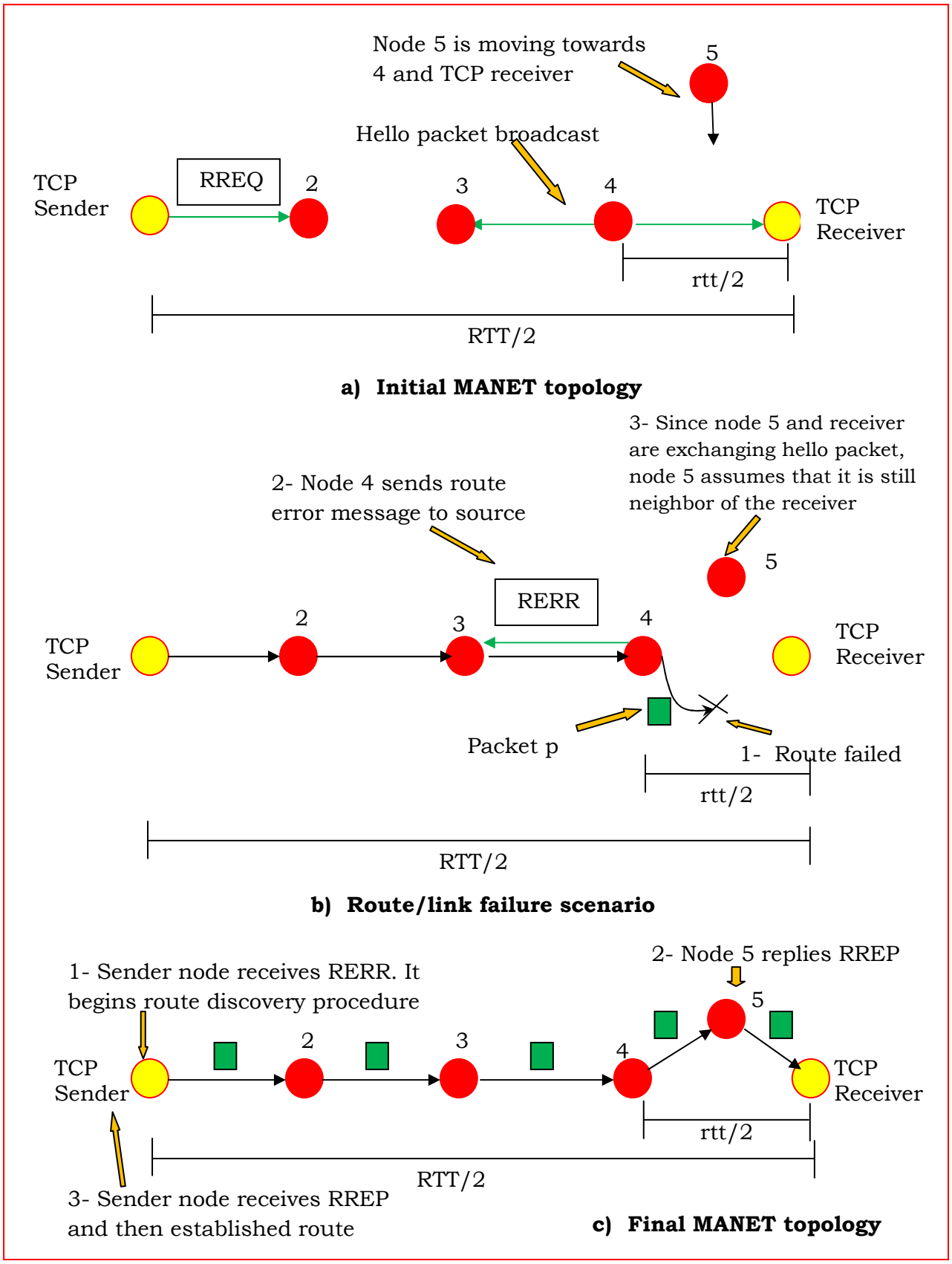


Figure 4:1 Interaction of TCP-SACK and AODV for route failure scenario in MANET

Since AODV is used as a routing protocol, Fig. 4.1 shows the interaction of TCP-SACK and AODV protocols and how a route failure affects the end-to-end transmission of data packets (it is also described in section 2.5.3). As shown in fig.4.1a, first AODV agent at TCP sender broadcasts RREQ message to its neighbors. Subsequently RREQ is broadcasted by the other nodes till it reaches to TCP receiver node (destination node).

Now the receiver node has not forwarded RREQ packet, because, receiver node is the intended target of RREQ. The receiver node responds by sending RREP message towards the source node in reverse path. Finally, a complete path is established in this route *sender__2_3_4_receiver*. Here HELLO packets are exchanged among nodes periodically to make sure that the established routes are valid. A given node should broadcast HEELO message within 4 seconds for neighbors' connectivity, otherwise the route is considered as stale or old route and it is no longer used as valid route.

It is also assumed that node 5 is moving towards and closer to node 4 and receiver node so that it put itself inside the transmission radius/range of both nodes. Whereas, the receiver node is moving away from node 4 and HELLO message is no longer exchanged between node 4 and receiver node as they are out of their transmission range. Consequently, the route becomes invalid. This is the time at which node 4 knows about route failure as shown in fig. 4.1b. Node 4 then sends RERR packet back to the source node so that a new route discovery procedure is initiated by the source node to find or established new route to the receiver as it is shown fig. 4.1c. All the TCP packets along their invalidated route can get dropped. Moreover, due to the newly established route, packets can get delivered out-of-order at the destination. Here TCP sender is not in a position to distinguish the causes of packet losses. These packet losses are misinterpreted as a sign of network congestion. The other problem to TCP is, since there is no explicit notification

about route re-establishment period, TCP's RTO can get affected. Finally the newly created route may be longer or shorter than the old routes (in terms of node) so that TCP may not compute RTT accurately.

By making use of fig. 4.1, let's see how RTT value can be estimated: Let

$Tq(i)$ – Queuing time at node i.

$Tt(i)$ – Transmission time at node i.

Tp – The end-to-end propagation time.

$Ti(t)$ – Packet received by intermediate node i at time t.

$Treceiver(t)$ – Packet received by TCP receiver at time t.

$Tsender(t)$– Acknowledgment received by sender for packet p at time t.

The RTT measurement can be estimated as:

$$RTT = Tq(sender) + Tq(receiver) + Tt(sender) + Tt(receiver) + 2 * \sum_{i=1}^{n} \left(Tq(i) + Tt(i) + Tp(i)\right) \quad (4.9)$$

, where n is the number of intermediate nodes.
Now assume that a packet p sent by TCP sender at time to can get received by intermediate node (in this case node 4) at:

$$Ti(t) = to + \left(\frac{RTT}{2} - \frac{rtt}{2}\right) \quad (4.10)$$

And a packet sent at to can reach to the TCP receiver at:

$$Treceiver(t) = to + \frac{RTT}{2} \quad (4.11)$$

Assume that a packet p is lost due to route failure between node 4 and TCP receiver. Here we have two scenarios. 1- Packet can get dropped due to route failure and link level retransmission of the lost packet but may get delivered out-of-order. 2- Packet can be delivered out-of-order due to route change (new route). For the first scenario, assume that a packet p sent at time to is lost due to route failure. Then, $Ti(t)$ (eqn.4.10) becomes,

$$Ti(t) = to + \frac{RTT}{2} + \frac{rtt}{2} \tag{4.12}$$

At this time, the intermediate node (node 4) may have an indication that a packet p is lost due to route failure. If it immediately retransmits the lost packet (link layer acknowledgments are used in IEEE 802.11), then packet p is recovered at the TCP receiver at,

$$Treceiver(t) = to + \frac{RTT}{2} + rtt \tag{4.13}$$

The TCP sender can receive an acknowledgment for packet p at,

$$\begin{aligned} Tsender(t) &= to + \frac{RTT}{2} + rtt + \frac{RTT}{2} \\ &= to + RTT + rtt \end{aligned} \tag{4.14}$$

Therefore, TCP sender should have to disable calling fast retransmission/recovery algorithms at least for a period $T = RTT$. Since, practically inter-arrival of packets are non-zero and TCP sender may not exactly know the value of rtt, from eqn.4.14, within one *rtt,* link layer can retransmit the lost packet p. Hence, the lower bound for disabling period T can be set to RTT.

For the second scenario; if a packet sent at time *to* is lost due to route failure, and if link layer feedback and packet inter-arrival time are ignored, then subsequent packets; $p+1$, $p+2$, can get delivered at TCP-PLDR receiver side out-of-order due to route changes (new routes). From the above computation (eqn.4.6-4.8), if disabling period is set to one times RTT *(T=RTT)*, since $RTO(k+1)$ is calculated as $T(k+1)+4$ times the mean deviation of *RTT* given by $\lambda t(k+1)$ (eqn.4.8), unnecessary *RTO* can be avoided, which forces the *cwnd* to start from one by entering into slow start phase. Therefore, the upper bound for disabling period T can be taken as *RTT* to avoid unnecessary expiration of *RTO*. To validate this assumption, simulation was conducted for experiments with multi-path routing protocol TORA (T=1*RTT, and T=2*RTT).

4.4 Description of TCP-PLDR Algorithm

Fig. 4.2 and 4.3 describe the difference between TCP-SACK and TCP-PLDR protocols with respect to congestion control algorithm. In fig4.2, one window of data packets (six packets) is sent from TCP-SACK sender to receiver. All the six packets except packet 3 reached to TCP receiver in order. Here we have two scenarios; First scenario, packet 3 is delivered out-of-order (after packet 6 has reached) at the receiver. Second scenario, packet 3 can get lost in transit. In both scenarios TCP receiver shouldn't modify the sequence number of packet 3. Therefore, it sends three duplicated acknowledgments (shown in red line) to implicitly inform the TCP sender that packet 3 is not yet received. Up on the reception of these three duplicated acknowledgments, TCP sender performs fast retransmission of packet 3 and reduces *cwnd* by half. Then after, packet 3 has reached out-of-order at the receiver side. In this scenario, packet 3 is not lost, but the sender assumes that packet loss is an indication of congestion. Consequently, without the network becomes congested, TCP-SACK sender slows down the sending rate by setting $ssthresh = \frac{CWNDold}{2}$ and $CWNDnew = ssthresh + 3$ (for three duplicated

acknowledgments), which leads to lower throughput. For the second scenario; if packet 3 is lost in transit, it seems reasonable for TCP sender to call fast retransmit/recovery algorithms to re-transmit packet 3 and reduce *cwnd* by half. But, in MANET this lost packet is not necessarily due to congestion. However, it may be due to route/link failure due to mobility of nodes. Packet 3 should be retransmitted by calling fast retransmit algorithm. However, reducing window size without the network becomes congested reduces performance of TCP a lot (available bandwidth is not utilized efficiently).

Fig4.3 shows the description of TCP-PLDR protocol. Once again two scenarios are taken as in fig.4.2. First scenario, packet 3 is delivered out-of-order at TCP-PLDR receiver side. Second scenario, packet 3 is lost in transit. In the first scenario; though TCP-PLDR at the receiver side generates three duplicated acknowledgments for packet 3, TCP-PLDR at the sender side shouldn't use these three duplicated acknowledgments as an indication of network congestion as TCP-SACK does. Rather, TCP-PLDR receiver explicitly uses time stamp option to detect out-of-order packet and sets *ooo_option_bit_=1* and sends to TCP-PLDR sender.

Up on the reception of *ooo_option_bit_* set to 1, TCP-PLDR at the sender side, disables some of the state variables like setting *dupacks_=0*, and *timeout_=false*, for a period of *T* time and then calls *opencwnd()* for *cwnd* to increase its current value and *send_one()* algorithm to send one new packet. At packet 11 times, acknowledgment for packet 3 is received, which avoids unnecessary retransmission of packet 3 and reduction of *cwnd*. Consequently, it improves usage of available bandwidth and throughput of TCP's performance. For the second scenario, if packet 3 is lost in transit, disabling period get expired after one RTT. In this case, TCP-PLDR will call fast retransmission/recovery algorithms (*dupack_action()*). Generally, in all scenarios the rationale behind TCP-PLDR protocol is that waiting for three

duplicated acknowledgments is too small and waiting for RTO gets expired is too large. However waiting for one RTT is a compromised selection period in ad hoc networks where a significant amount of packet loss is due to route failure as a result of arbitrary movement of mobile nodes, wireless channel error etc.,.

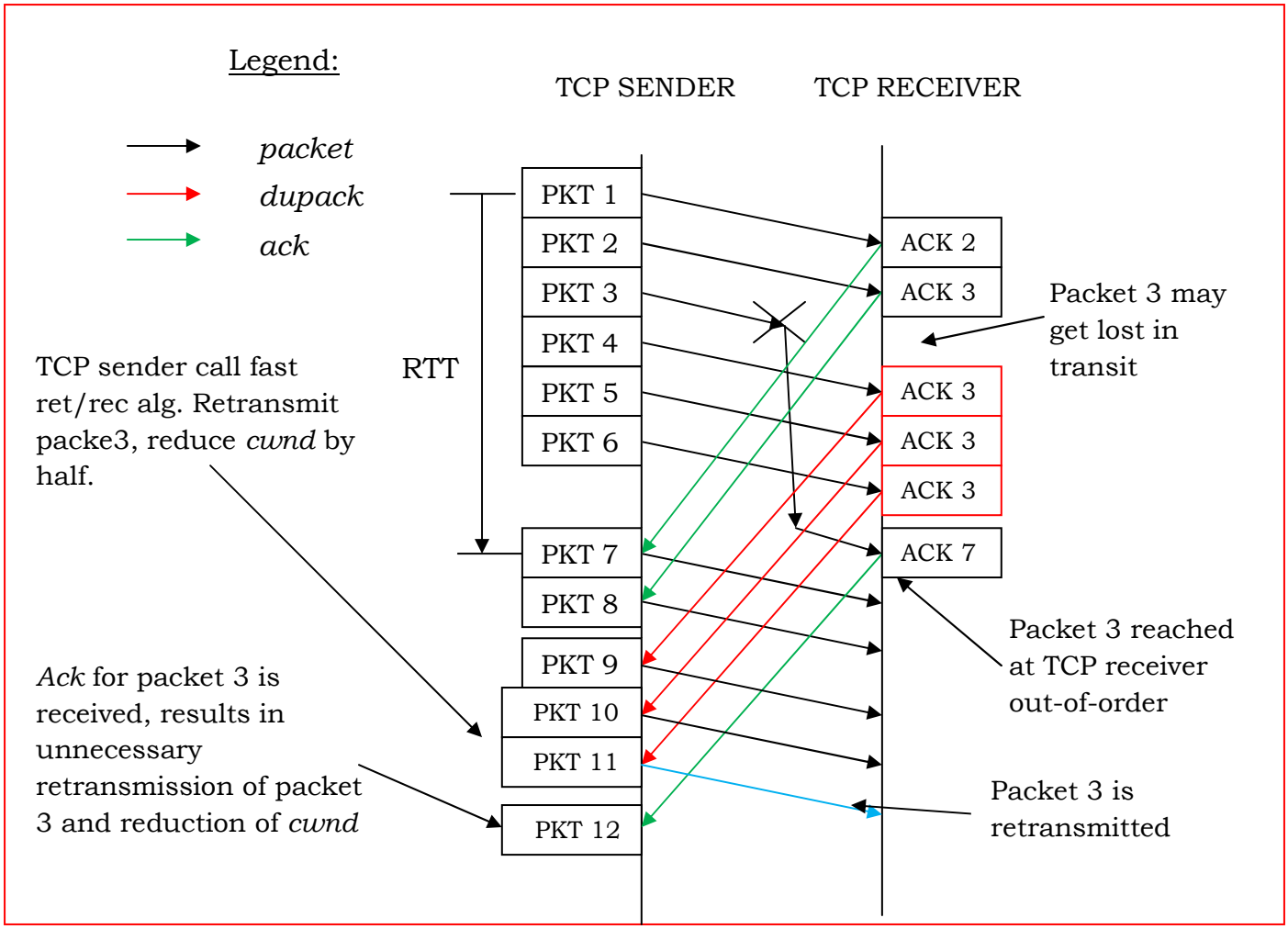


Figure 4: 2 TCP-SACK fast retransmission/recovery algorithms

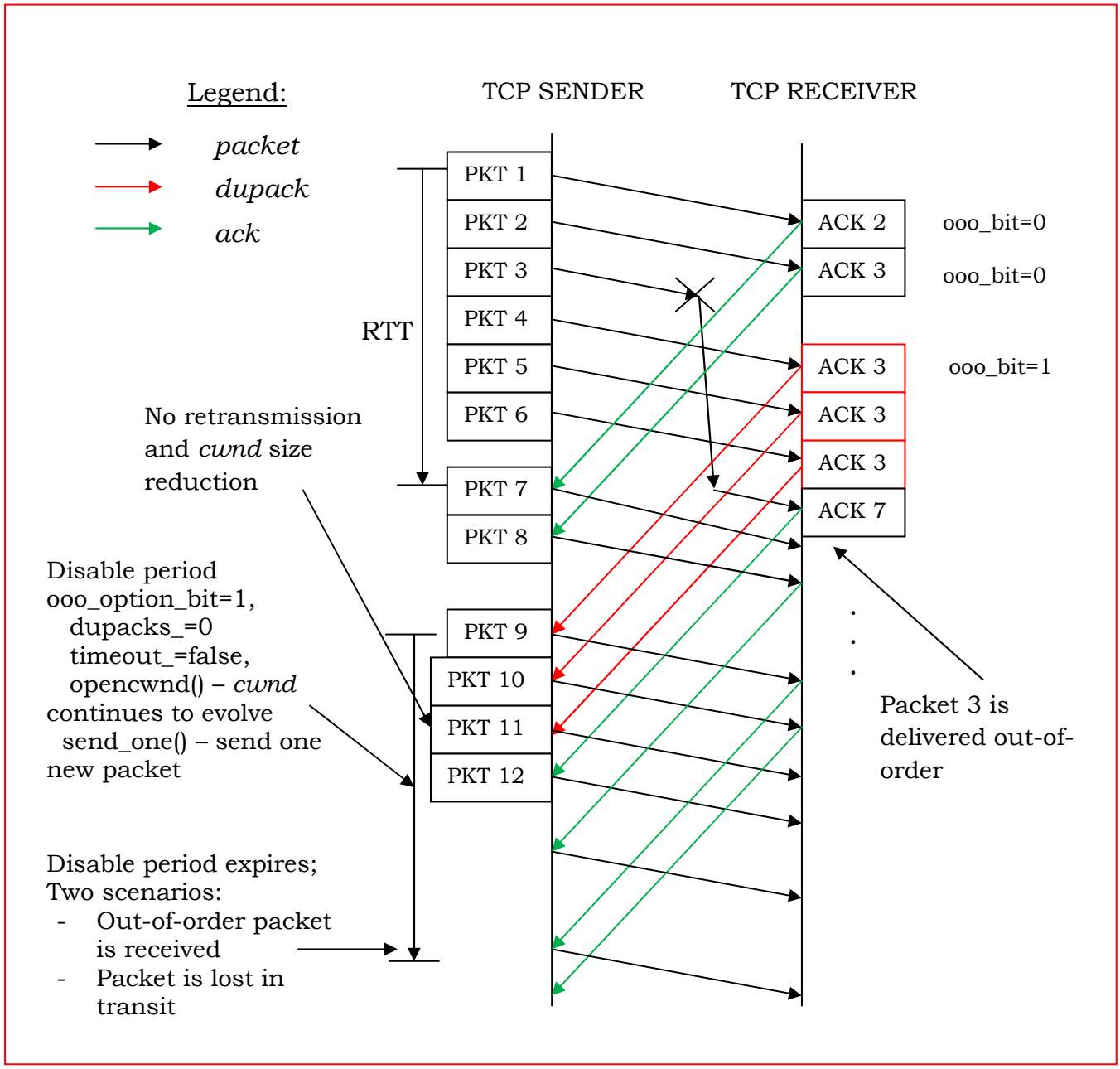


Figure 4:3 Behavior of TCP-PLDR algorithm

4.5 Analytical Model of TCP-SACK and TCP-PLDR Protocols

In this section, the throughput of TCP-SACK and TCP-PLDR protocols are analytically modeled as a function of packet loss probability and RTT when both protocols are in congestion avoidance phase. Random packet loss at constant probability p is assumed. This means, there is one packet loss after successfully transmitted $\frac{1}{p}$ consecutive packets. Multiple packet losses per RTT are taken as one congestion loss. This is because; TCP-SACK assumes that multiple packet loss in one RTT as a single congestion loss [61,64]. Under these assumptions, fig.4.4 shows the *cwnd* with periodic window progress of TCP-SACK under periodic packet losses. It uses additive increase and multiplicative decrease for dealing with mild congestion. $Wmax$ indicates the maximum value of *cwnd* that reached equilibrium. *cwnd* is reduced to $\frac{Wmax}{2}$ after each packet loss, and then it starts a new congestion avoidance phase (see section 2.1.1). The performance metric that is going to be used is throughput, which is expressed as total amount of packet sent per unit time. It is also assumed that the full congestion window is transmitted per *RTT*,

$$T(t) = \frac{W(t)}{RTT} \tag{4.15}$$

, where $T(t)$ is the throughput, $W(t)$ the window size.

The throughput is considered and modeled under moderate congestion so that the network is not severely congested to ignore the slow start phase of TCP-SACK, which is experienced due to expiration of RTO. Besides, the following assumptions are made to analytically model throughput of TCP-SACK and TCP-PLDR protocols. Sender has always data packets to transmit to receiver. Data packets are always a maximum segment size of MSS. TCP sender and receiver have infinite buffer so that *cwnd*<<*awnd*. The network is

at steady state to make the delay constant so that RTT measurement is constant, which is graphically expresses in fig.4.5. Periodic loss events with p percentage of lost segments are assumed in the computation.

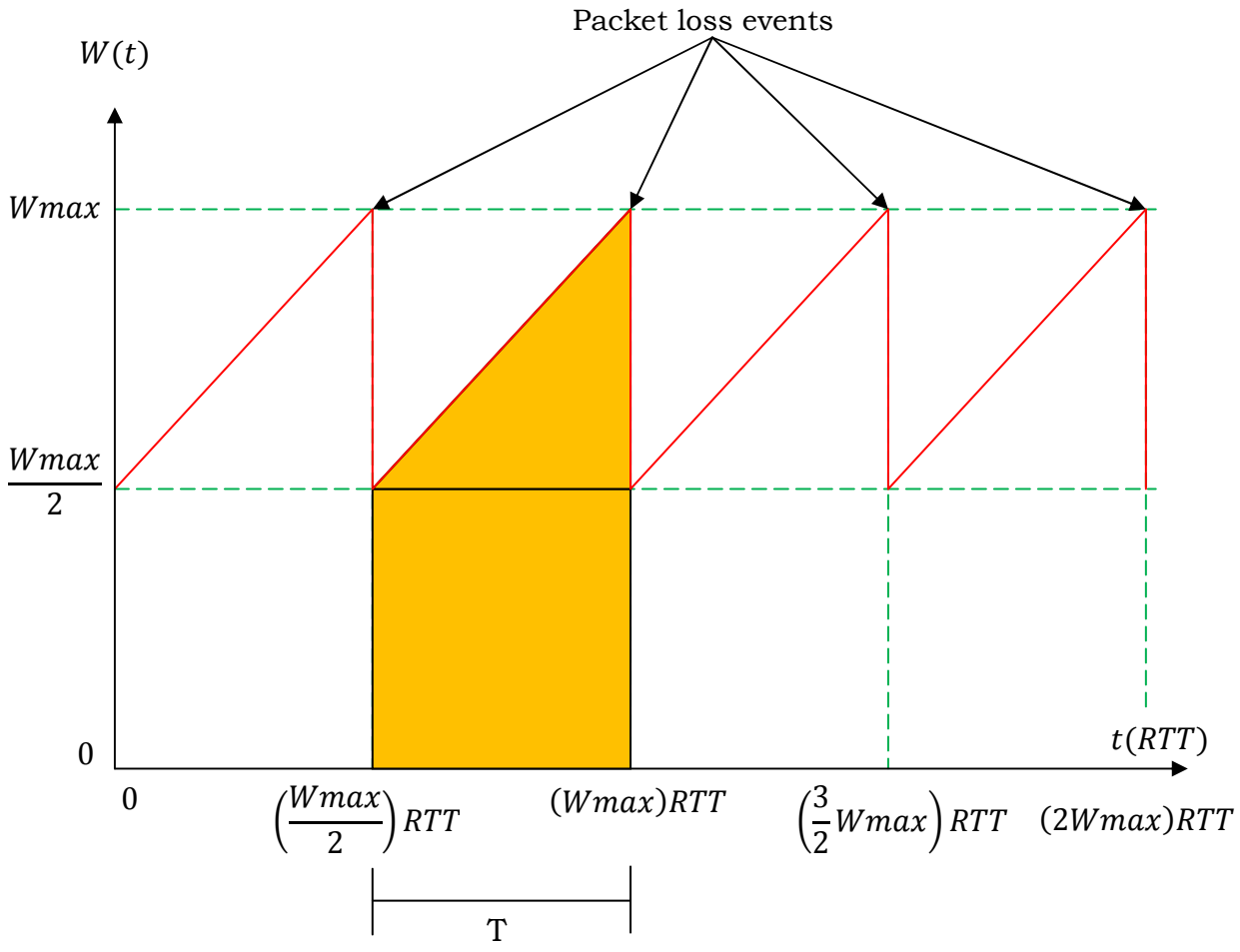


Figure 4:4 TCP-SACK window and throughput model under periodic packet losses

The steady state behavior from $\left(\frac{Wmax}{2}\right)RTT$ to $(Wmax)RTT$, which is equal to one period T shown in fig.4.4 can be described graphically in fig. 4.5. There is an additive increase of *cwnd* followed by a packet loss event so that one period, which is usually called as saw tooth behavior can be achieved. A new period can be started when there is a packet loss in the previous period. After a packet loss event, TCP-SACK sender will use a new window size given as

stated in eqn.2.8. There are i number of packet transmission per constant RTT with no packet loss. However, at the start and end of the period, there are packet losses depicted in red color. This pattern repeats itself again.

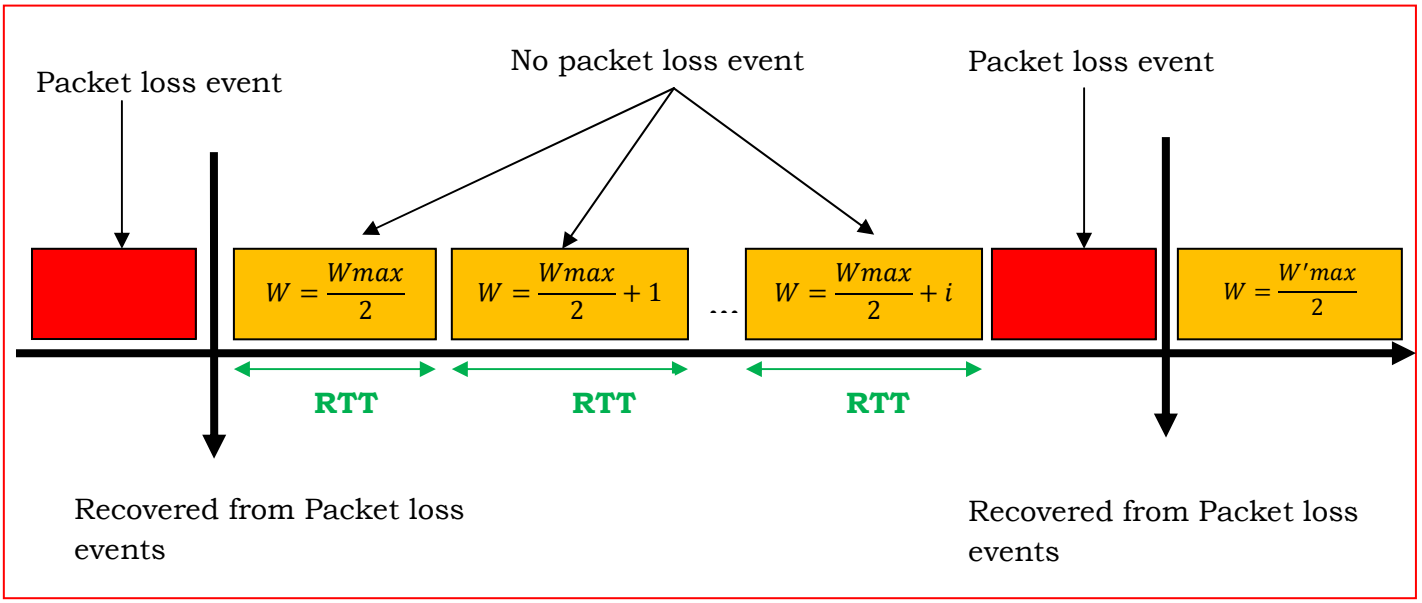


Figure 4:5 one period (saw tooth) behavior of TCP-SACK at steady state

The congestion window behavior can be derived as follow: $W(t) = Wi$ means Wi packets are sent during i^{th} RTT as in fig.4.4 and 4.5. At the end of the i^{th} RTT, if all acknowledgments are successfully received by TCP sender, then

$$W(t) = Wi + 1 < Wmax \qquad (4.16)$$

Otherwise, if there is a packet loss, then $W(t)$ reduces by a period T given as:

$$T = \left(\frac{Wmax}{2}\right) RTT, and$$

$$W(t) = \frac{Wmax}{2} \qquad (4.17)$$

The first assumption that we made from Eqn. 4.16, which says an additive increment of sender's *cwnd* when there is no packet loss can be illustrated graphically as in fig.4.6. This assumption states that *cwnd* of TCP-SACK

will increment by one packet if there is no packet loss. Moreover, fig.4.6 describes how *cwnd* of TCP-SACK sender behaves for every acknowledgment received. Up on reception of new acknowledgment for each packet sent, *cwnd* is increased by a constant value of $\frac{1}{cwnd}$. After receiving all acknowledgments for one window of data, *cwnd* is incremented to $cwnd + 1$. This was clearly explained in chapter two (eqn.2.8).

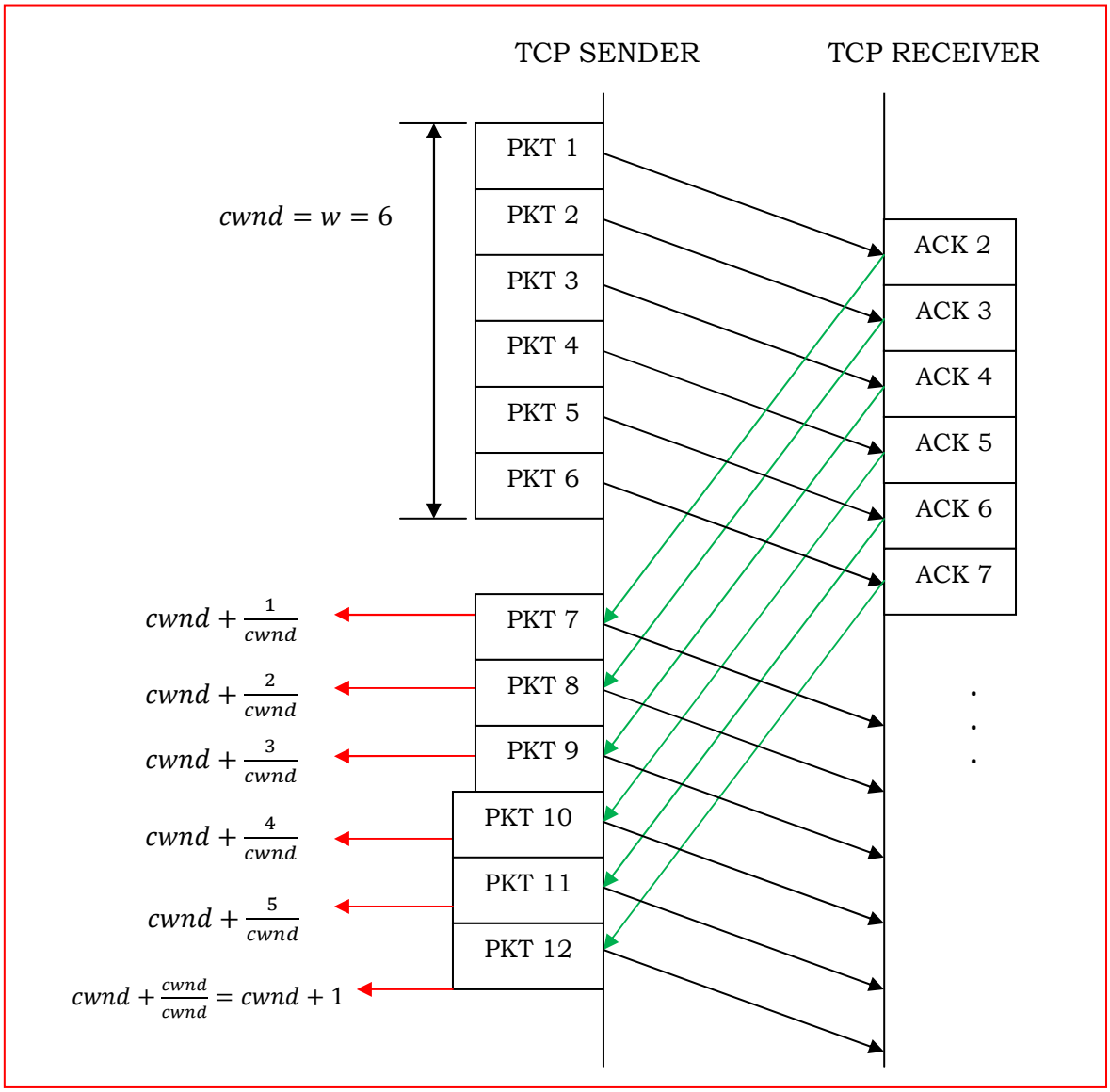


Figure 4:6 Additive increase of TCP-SACK's *cwnd* without a packet loss event

Fig.4.7 shows how *cwnd* of TCP sender behaves when delayed acknowledgment are used. TCP-SACK receiver uses delayed acknowledgment mechanism to reduce the number of acknowledgment packets. In this case, an acknowledgment packet is sent after b *packets* are received (most of the implementation uses $b = 2$). For every two packets transmitted, the receiver sends back acknowledgment thereby reducing the number of acknowledgments by half. *cwnd* is incremented to $cwnd + \frac{1}{2}$ after transmitting packets in one RTT. Generally, if the receiver sends acknowledgments for every b *packets* received in one RTT, then the sender's *cwnd* is incremented to $cwnd + \frac{1}{b}$.

The other assumption we made is random packet loss at constant probability p .This means, one packet loss after successfully transmitted $\frac{1}{p}$ consecutive packets. Moreover, in TCP-SACK multiple packet loss is treated as one loss. This can be described as shown in fig.4.8. If one packet is lost in one RTT (this means in one window of data), then subsequent packets in the same window are also taken as lost. Assume that packet 3 is lost in transit, it means packet 4, 5, and 6 are also lost. This means that the sender shouldn't receive any acknowledgment for this window of data. Therefore, the sender will receive an acknowledgment for the first packet in the next RTT (next window of data, in this case for packet 7).

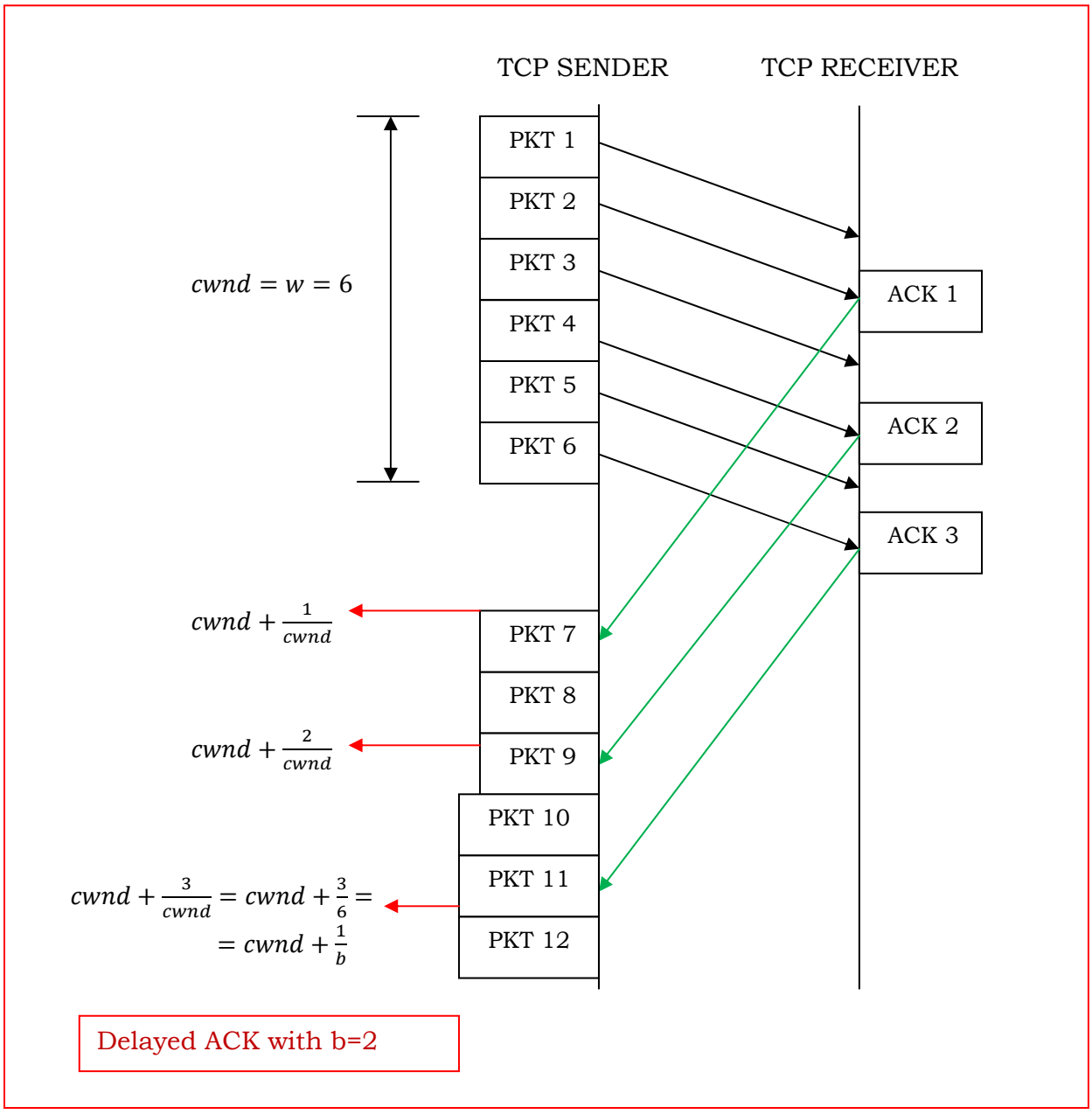


Figure 4:7 Implementation of delayed acknowledgment in behavior of *cwnd*

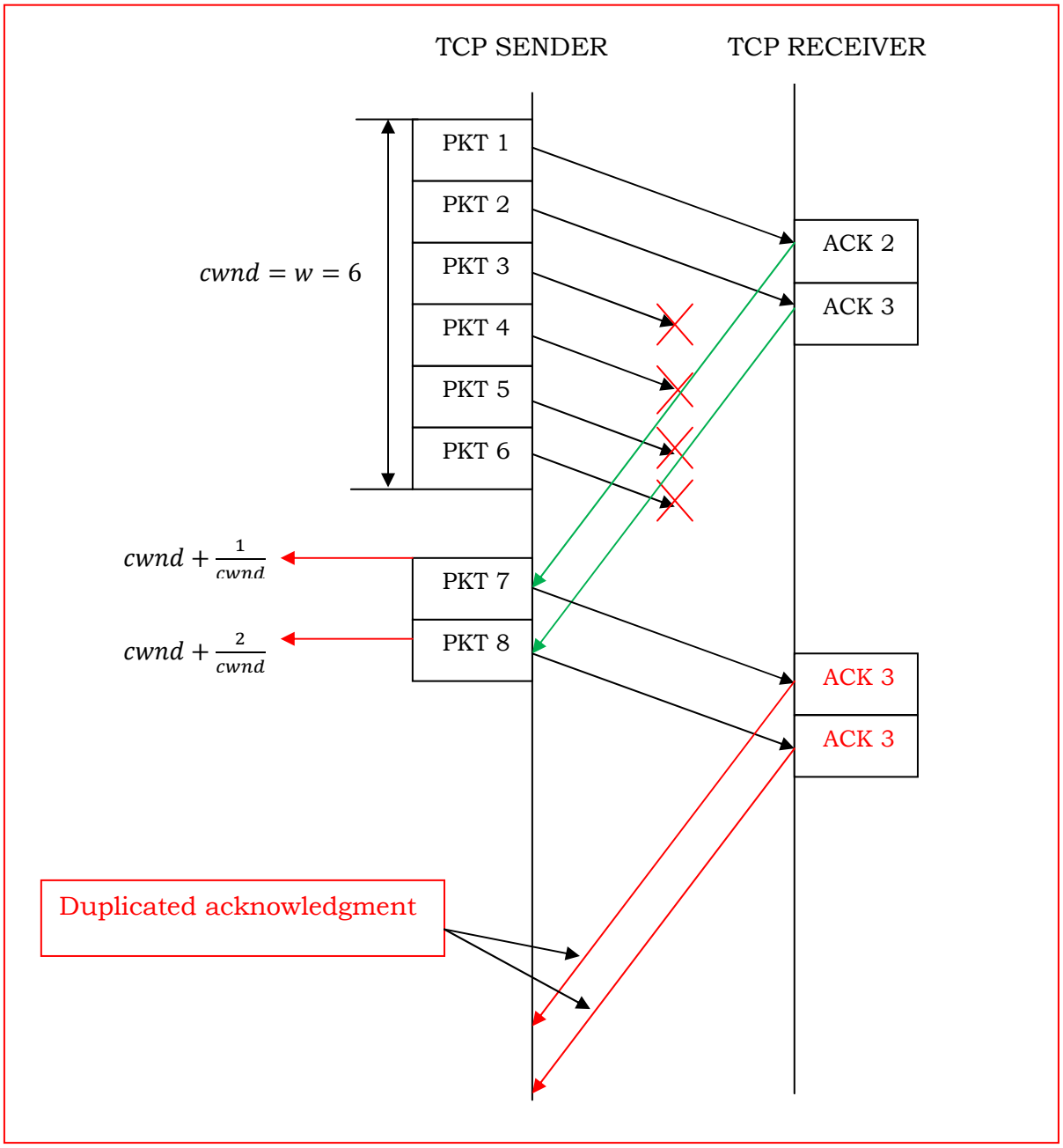


Figure 4:8 packet loss event assumptions.

If p is the percentage of packet loss, then the total number of packets given by $Ptotal$ sent or delivered every period T is given by:

$$Ptotal = \frac{1}{p} \tag{4.18}$$

It is also possible to compute $Ptotal$ by making use of $W(t)$ as:

$$Ptotal = \sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} \left(\frac{Wmax}{2} + i\right) = \left(\frac{Wmax}{2}\right)\left(\frac{Wmax}{2} - 1\right) + \sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} i$$

$$= \frac{(Wmax)^2}{4} - \frac{Wmax}{2} + \sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} i \tag{4.19}$$

$$\sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} i = 0 + 1 + 2 + \cdots + \left(\frac{Wmax}{2}\right) - 1 = \frac{\left(\frac{Wmax}{2} - 1\right)\left(\frac{Wmax}{2}\right)}{2} \tag{4.20}$$

Eqn.4.20 can be proved by using mathematical induction. Eqn.4.20 holds true for initial condition $i = 0$. Substitute 0 on the right and left side of eqn.4.20 gives 0. Now we can proof that if eqn.4.20 holds true for $\left(\frac{Wmax}{2} - 1\right)$, then it holds true for the next number, which is $\frac{Wmax}{2}$. Assume that:

$$\sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} i = \frac{\left(\frac{Wmax}{2} - 1\right)\left(\frac{Wmax}{2}\right)}{2}, \quad required\ to\ prove\ that$$

$$\sum_{i=0}^{\left(\frac{Wmax}{2}\right)} i = \sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} i \; + \left(\frac{Wmax}{2}\right) \tag{4.21}$$

$$= \frac{\left(\frac{Wmax}{2}-1\right)\left(\frac{Wmax}{2}\right)}{2} + \left(\frac{Wmax}{2}\right), by\ inductive\ hypothesis$$

$$= \frac{\frac{(Wmax)^2}{4} - \left(\frac{Wmax}{2}\right)}{2} + \frac{Wmax}{2}$$

$$= \frac{\left(\frac{Wmax}{2}-1\right)\left(\frac{Wmax}{2}\right) + 2\left(\frac{Wmax}{2}\right)}{2}$$

$$= \frac{\frac{(Wmax)^2}{4} + \left(\frac{Wmax}{2}\right)}{2}$$

$$\sum_{i=0}^{\left(\frac{Wmax}{2}\right)} i = \frac{\left(\frac{Wmax}{2}\right)\left(\frac{Wmax}{2}+1\right)}{2} \tag{4.22}$$

Eqn.4.20 is proved by using mathematical induction (eqn.4.21 and 4.22). From eqn. 4.19 and 4.20,

$$Ptotal = \sum_{i=0}^{\left(\frac{Wmax}{2}\right)-1} \left(\frac{Wmax}{2}+i\right) = \left(\frac{Wmax}{2}\right)\left(\frac{Wmax}{2}-1\right) + \frac{\left(\frac{Wmax}{2}-1\right)\left(\frac{Wmax}{2}\right)}{2}$$

$$= \frac{(Wmax)^2}{4} + \frac{(Wmax)^2}{8}$$

$$= \frac{3}{8}(Wmax)^2 \qquad (4.23)$$

The total data packet delivered for each period can also be taken as the area on the shaded region depicted in fig. 4.4, which is calculated as:

$$Ptotal = \left(\frac{Wmax}{2}\right)\left(\frac{Wmax}{2}\right) + \frac{1}{2}\left(\frac{Wmax}{2}\right)\left(\frac{Wmax}{2}\right)$$

$$= \frac{(Wmax)^2}{4} + \frac{(Wmax)^2}{8}$$

$$\frac{3}{8}(Wmax)^2 = \frac{1}{p} \qquad (4.24)$$

Hence the maximum window size $Wmax$ due to packet loss probability p is given by:

$$Wmax = \sqrt{\frac{8}{3p}} \qquad (4.25)$$

Now the average throughput *T(t)*, which is defined as the total amount of packets sent over a given period T can be computed as:

$$T(t) = \frac{Total\ amount\ of\ data\ delivered}{period\ T}\ , then$$

$$T(t) = \frac{\frac{1}{p}MSS}{\left(\frac{Wmax}{2}\right)RTT} = \left(\frac{MSS}{RTT}\right)\frac{2}{p\sqrt{\frac{8}{3p}}}$$

$$= \left(\frac{MSS}{RTT}\right)\sqrt{\frac{3}{2p}} \qquad (4.26)$$

The throughput of TCP-SACK with the following assumptions: random packet losses, and with delayed acknowledgments as it is described in fig.4.7, a cumulative acknowledgment for every b packets is given by:

$$T(t) = \left(\frac{MSS}{RTT}\right)\sqrt{\frac{3}{2bP}} \qquad (4.27)$$

Eqn.4.27 shows that the throughput increases proportional to the packet size and inversely proportional to the *RTT* value and the square root of packet loss probability p. Based on eqn.4.27, any protocol to be TCP-SACK friendly, it is reasonably fair when competing for bandwidth with TCP–SACK flows, but has a much lower variation of throughput $T(t)$ over time compared with TCP-SACK. Generally, a protocol is said to be TCP-SACK friendly if its throughput is *proportional to* $\frac{1}{\sqrt{p}}$.

The following analysis shows the behavior of TCP-PLDR when the network is moderately congested so that its behavior can be analyzed in congestion avoidance phase (fig.4.9). When TCP-PLDR algorithm is implemented in TCP-SACK protocol, main modification was done in congestion avoidance phase to avoid frequent timeout. Once again, all same assumption that we made on TCP-SACK holds true to study the steady state behavior of TCP-PLDR.

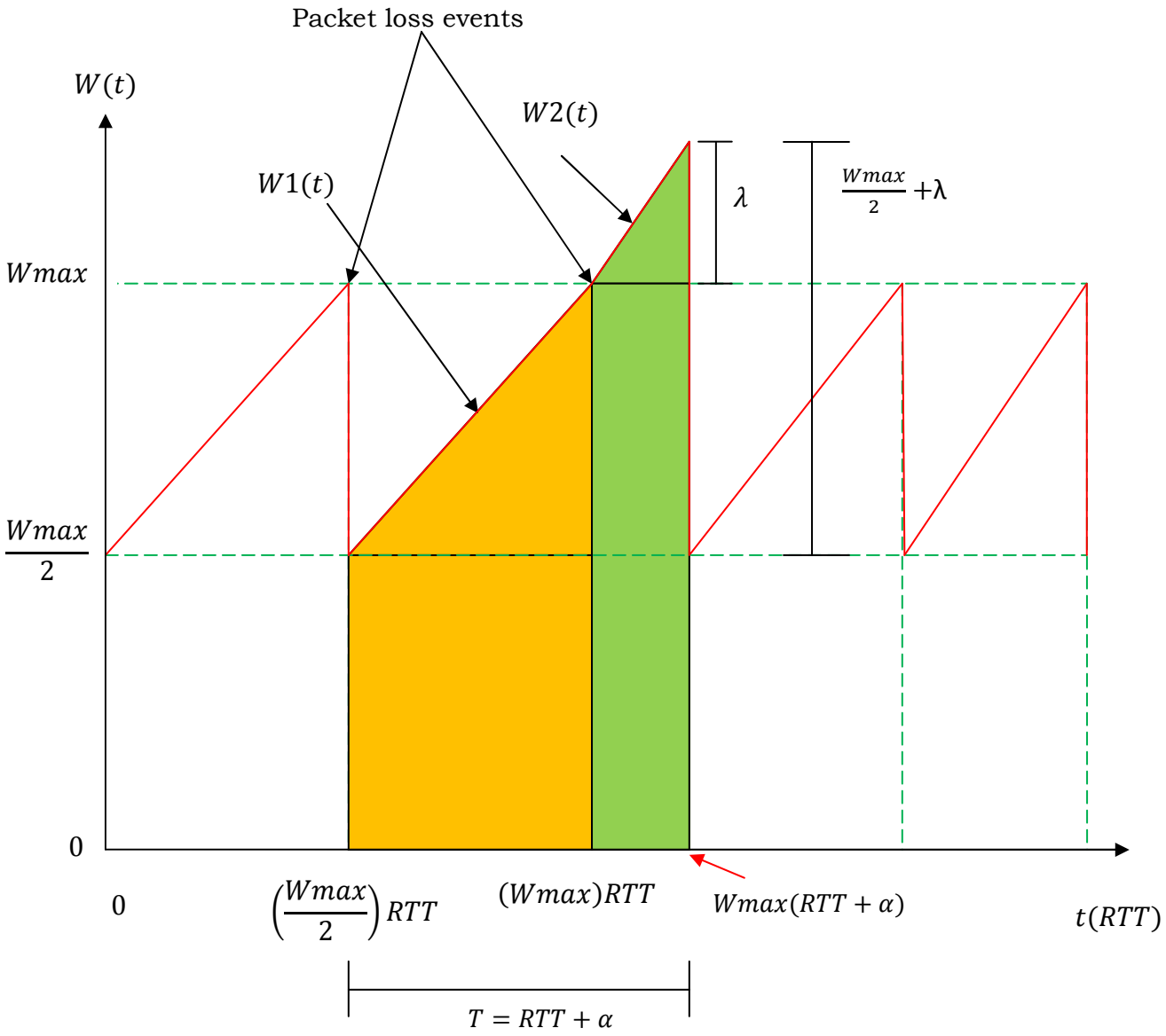


Figure 4:9 Throughput model of TCP-PLDR in congestion avoidance phase

The congestion window of TCP-PLDR can be represented by making use of two congestion window functions; $W1(t)$ that determines the congestion window behavior before packet loss event. $W2(t)$, which determines the congestion window behavior after packet loss event. It is assumed that the disabling period $\alpha = RTT$ is selected in TCP-PLDR. Since we ignored the slow

start phase, the congestion window function $W1(t)$ increases its window size by 1MSS for every RTT in the absence of packet loss events (and thus increases its transmission rate by additive factor) just as TCP-SACK does. The congestion window function $W2(t)$ has two components: for disabling period $\alpha = RTT$ between $(Wmax)$ RTT and $Wmax(RTT + \alpha)$ increases its transmission ranges by additive increase. After the disabling period get expired, at $Wmax(RTT + \alpha + \beta)$, the congestion window is cut in half after loss event (decreases by multiplicative factor). As shown in fig.4.9, the throughput (in packets per second) can be taken as the number of packets that can be sent between two consecutive packet losses ($Ptotal$) over one period of time interval between two successive packet losses (T). From fig.4.9 the period or the time between two successive packet losses T can be given as:

$$T = Wmax(RTT + \alpha) - Wmax(\frac{RTT}{2})$$

$$= \frac{(Wmax)RTT}{2} + (Wmax)\alpha$$

$$= Wmax\left(\frac{RTT}{2} + \alpha\right) \qquad (4.28)$$

Since the disabling period introduces additional λ windows, eqn.4.28 can be given as:

$$T = (Wmax + \lambda)\left(\frac{RTT}{2} + \alpha\right) \qquad (4.29)$$

Now the total packet delivered or sent by TCP-PLDR sender $Ptotal$ can be given as the shaded area (yellow + green) as shown in fig.4.9.

$$Ptotal = \frac{(Wmax)^2}{4} + \frac{\frac{1}{2}(Wmax)^2}{4} + (Wmax)^2 + \frac{1}{2}\lambda(Wmax)$$

$$Ptotal = \frac{11}{8}(Wmax)^2 + \frac{\lambda}{2}(Wmax) \qquad (4.30)$$

Since $Ptotal$ is the total number of packets delivered for one period or between two successive packet losses, the steady state packet loss probability is given as in eqn.4.18 as $Ptotal = \frac{1}{p}$. Therefore,

$$\frac{11}{8}(Wmax)^2 + \frac{\lambda}{2}(Wmax) = \frac{1}{p}$$

$$\frac{11}{8}(Wmax)^2 + \frac{\lambda}{2}(Wmax) - \frac{1}{p} = 0 \qquad (4.31)$$

Solving eqn.4.31 gives the value of $Wmax$ as,

$$Wmax = \frac{-\frac{\lambda}{2} \pm \sqrt{(\frac{\lambda}{2})^2 - \frac{44}{8}(-\frac{1}{p})}}{\frac{22}{8}}$$

$$Wmax = \frac{-2\lambda \pm 4\sqrt{(2p\lambda^2 + 44)/8p}}{11} \qquad (4.32)$$

Finally as per eqn. 4.29, the throughput of TCP-PLDR can be computed as,

$$T(t) = \frac{Total\ amount\ of\ data\ delivered}{period\ T}\ , then$$

$$T(t) = \frac{\frac{1}{p}MSS}{Wmax(\frac{RTT}{2} + \alpha)}$$

$$T(t) = \left(\frac{MSS}{RTT} + 2\alpha\right)\left(\frac{22}{-2\lambda + 4\sqrt{\frac{p(2\lambda^2 + 44)}{8}}}\right) \qquad (4.33)$$

Where α is the disabling period, which is selected to be $\alpha = RTT$, and λ is the new $cwnd$, which is increased by one packet during this disabling period α.

Now, if TCP-SACK experiences congestion losses with Pc (packet loss probability due to congestion) and route failure losses with Pr (packet loss probability due to route failure), its throughput that was given in eqn.4.27 can be approximated as:

$$T(t) = T(TCP - SACK) = \frac{MSS}{RTT}\frac{C}{\sqrt{Pc + Pr}} \tag{4.34}$$

, where C is constant. Eqn.4.34 would mean that since TCP-SACK cannot distinguish packet loss due to congestion or route failure, it halves its window size for both types of losses. However, since TCP-PLDR knows the exact cause of packet losses (congestion or route failure losses), it only reduce its congestion window size when there is a congestion loss. Consequently, its throughput, which is given in eqn.4.33, can be approximated as:

$$T(t) = T(TCP - PLDR) = \frac{MSS}{RTT}\frac{C}{\sqrt{Pc}} \tag{4.35}$$

, where C is constant. Therefore, the ratio of improvement achieved by TCP-PLDR over TCP-SACK can be approximated as:

$$\frac{T(TCP - PLDR)}{T(TCP - SACK)} = \sqrt{1 + \frac{Pr}{Pc}} \tag{4.36}$$

This approximation shows that the throughput improvement increases with Pr and decreases with Pc. Therefore, the ratio $\frac{Pr}{\mathrm{Pc}}$ is the main reason for the

performance improvement we can expect from TCP-PLDR. Moreover, from the analytical result, it is also confirmed that the throughput model of TCP-PLDR is similar to TCP-SACK. Since its throughput is inversely related to the square root of packet loss probability ($T(t)\ \alpha\ \frac{1}{\sqrt{p}}$), it is TCP friendly. However, there is a throughput improvement of the green shaded area shown in fig.4.9. The green shaded triangle would mean that by tolerating for α period of time, λ amount of window size is increased. Besides, additional (green shaded area) amount of data packets can be delivered with packet loss probability p.

4.6 Implementation of TCP-PLDR Using ns-2

The network simulator (ns) is currently one of the most widely used network simulators. Ns-2 was primarily chosen due to the fact that it is proven simulation tool utilized in several previous MANET studies, to mention a few [8,9,22,24,25,27,29,30,32,33,35,36,37,38,43,49,51,53,54,55,56]. First the ns-2 simulation tool is presented. The implementation details of TCP-PLDR using ns-2.29 is described next.

4.6.1 Introduction to ns-2

Ns-2 is an open source discrete event simulator used by the research community for research in networking [75]. We chose ns-2 for the implementation because it is a freely distributed code and supports many interesting protocols. 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 so that modification and reuse of different protocols are possible. It is written in C++ and Otcl (tools command language) programming languages. Even

though 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 (MANET, wireless sensor networks (WSN), vehicular ad hoc networks (VANET), satellite networks, etc,.), transport protocols such as TCP, and user datagram protocol (UDP). It models Web, Telnet, and FTP applications. It also includes the implementation of different routing protocols (AODV, DSDV, DSR, and TORA).

It also 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. C++ and tcl are the two languages used in ns-2. Two languages are needed to perform complex programming, in collaboration 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 re-compiling simulations. Tcl on the other hand, is much slower but very convenient for varying simulation parameters to create different network simulation scenarios. Consequently, C++ is used for implementing properties of the protocol while tcl is used to implement code that needs to be changed frequently in order to study the protocol behavior.

4.6.2 Class hierarchy in ns-2

Fig.4.10 shows the ns-2 class structure. The root of the hierarchy is the class TclObject. It is the super class of all Otcl library objects such as scheduler, network components, timers, and other objects (NAM) [75]. 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.

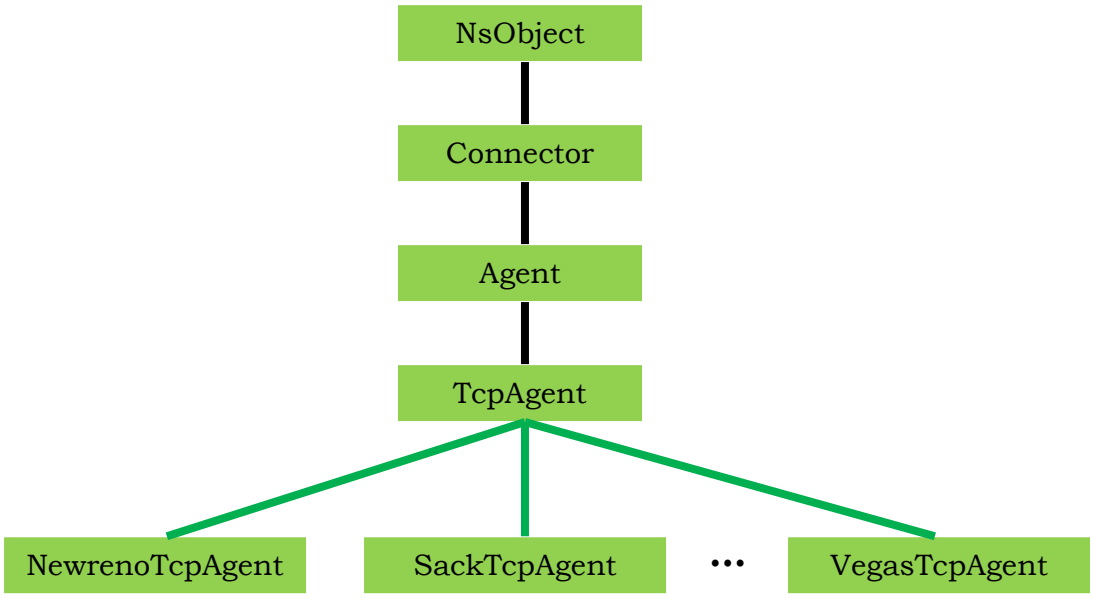


Figure 4:10 Class hierarchy in ns-2

4.6.3 Implementation of TCP-PLDR algorithm using ns-2.29

The implementation of TCP-PLDR algorithm is based on one of the TCP variant known as TCP-SACK modules both at the receiver and sender side. At TCP receiver side *receive()* 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 SACK at TCP receiver and sender side. Necessary modifications are made in the agent named class TCPSinkSack and its *receive()* function, which is the main reception path for packets and provides various other necessary methods. One variable known as *ooo_option_bit_* 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, main modification is done on the receive method. This

method should check for two variables, namely *ooo_option_bit_*. Up on the reception of *ooo_option_bit_*, 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 (details were presented in section 4.2). Fig.4.11 shows the implementation hierarchy and the important C++ classes for TCP-PLDR implementation, they are shown with yellow colored.

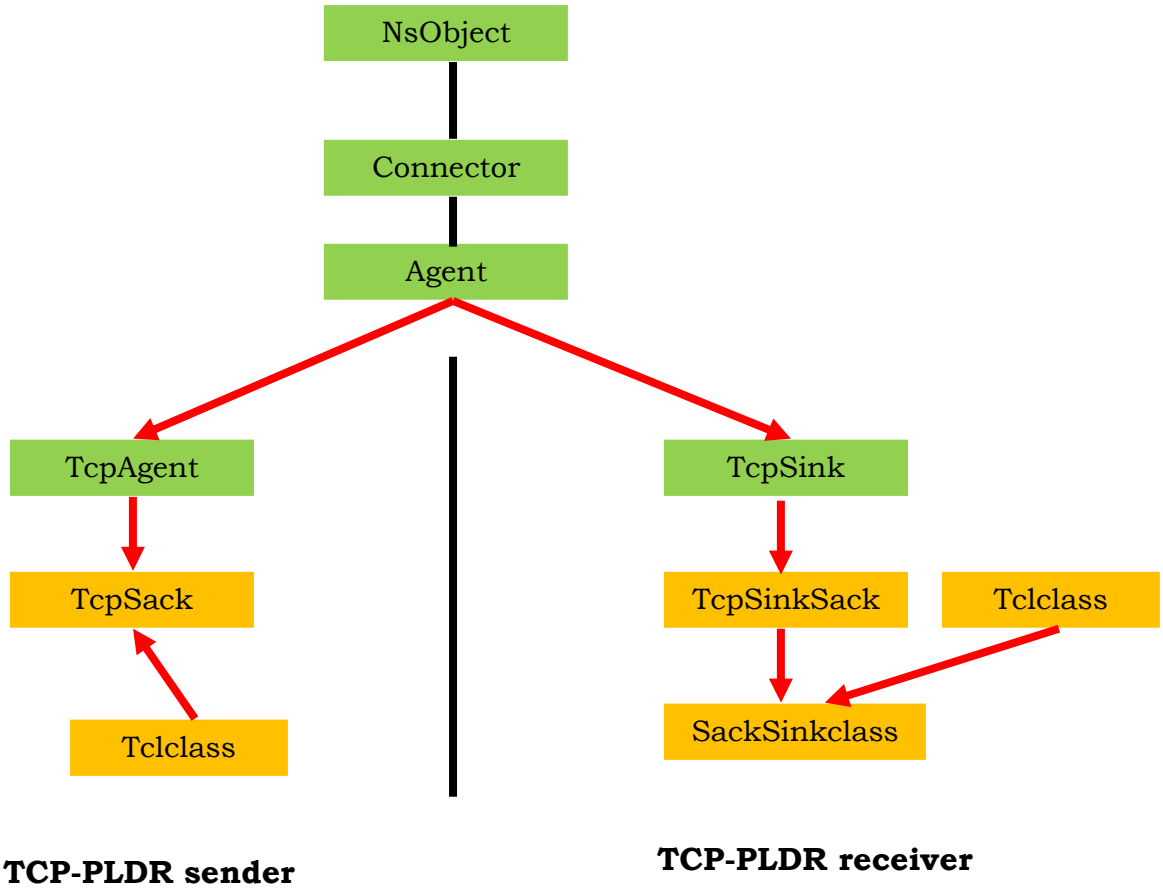


Figure 4:11 TCP-PLDR implementation in ns-2.29

4.7 Performance evaluation

In this section, we present the performance evaluation of TCP-PLDR and TCP-SACK protocols using ns-2. First simulation scenarios and models are discussed. Network topology and movement of nodes with traffic models are presented next. Then, detail simulation results and analysis are presented.

4.7.1 Simulation Scenarios and Model

A detailed simulation model based on ns-2 was prepared to configure MANET topology. The ns-2 simulator supports for simulating wireless networks consists of different network components including physical, data link, and MAC layer models. From channel type, a wireless channel model with a 1000m by 1000m transmission range was selected. IEEE 802.11 for wireless networks was used as a MAC layer protocol. All packets (both data and routing) sent by the routing layer are queued at the interface queue until the MAC layer transmits 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 that routing packets have higher priority than data packets. Routing protocols that were used at the network layer are AODV, which represents single path routing and TORA, which represents multi-path routes.

The workload is a single TCP connection between a specific sender S (node 0) and a specific receiver R (node 9) for one TCP flow. The disabling period was set to one times RTT ($\alpha = RTT$). Simulation parameters and setups are summarized in table 4.1.

Table 4:1

Simulation parameters and setup

Simulation parameters	**Value**
Number of nodes	20
Speed of nodes	1-3m/s, 10-15m/s, 25-30m/s
MAC protocol	IEEE 802.11
Routing protocol	AODV, TORA
TCP	TCP-PLDR, TCP-SACK
Network topology	1000m X 1000m
Application	FTP
Simulation time	200 secs
Antenna type	Omni-antenna
Signal propagation model	Two-ray ground

4.7.2 Traffic and mobility models

For traffic source and application, file transfer protocol (FTP) was used above the agent TCP. The source-destination pairs are spread randomly over the network. 1040 byte data packets are used from TCP sender to receiver and 40 byte acknowledgments are used from TCP receiver to sender. Mobility models were created for the simulations using 20 nodes. The field configuration used was 1000m x 1000m field. The speed was chosen randomly between 2 and 30m/s for different mobility pattern. All the

simulations were run for 200 simulation seconds. Different mobility and identical traffic scenarios are used across the protocols to collect fair results.

4.7.3 Performance Metrics

Eight important metrics were evaluated over TCP-PLDR and TCP-SACK protocols.

Throughput – Is the ratio of total amount of data that reaches to receiver from the sender to the time it takes for the receiver to receive the last packet. It is presented in packet and kilobits per second (kbps). Mathematically, throughput can be given as [TFRC, RFC5-348]:

$$T = \frac{B}{RTT\left(\sqrt{\frac{2p}{3}}\right) + (2RTT)\left(3\sqrt{\frac{3p}{8}}\right)\left(p(1+32p^2)\right)} \qquad (4.37)$$

, where *RTT*=round trip time, *p*=loss rate, *B*=packet size, *T*=throughput

The higher the throughput, the better performance had TCP-PLDR achieved over TCP-SACK. For each TCP-PLDR setting, we divided the throughput values by the throughput measured in TCP-SACK. This ratio measures how much improvement we have achieved with TCP-PLDR under a given ad-hoc scenario.

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

End-to-end Delay - the average time taken by a data packet to reach to the receiver from sender. It also includes the delay caused by route discovery process at the routing layer and the queue in data packet transmission.

$$AvgEtEDelay = \frac{\sum_{i=1}^{n}(TCP_{senttime} - TCP_{receivedtime})i}{\sum_{i=1}^{n}(TCP_{received})i} \quad (4.38)$$

The lower value of end to end delay means the better performance of the protocol.

Packet delivery ratio (PDR) - the ratio of the number of delivered data packet to the receiver from sender. It shows how successful is a protocol in delivering data packets from source to destination.

$$PDR(\%) = \frac{\sum_{i=1}^{n}(TCP_{received})i}{\sum_{i=1}^{m}(TCP_{sent})i} \; X \; 100 \quad (4.39)$$

, where n is number of received packets, and m is number of sent packets. The greater value of PDR means the better performance of the protocol.

Congestion window size (*cwnd*) – is defined as the amount of packets in flight. *cwnd* is used to prevent TCP sender from sending more data than the network can accommodate in the current load condition.

Moreover, in order to analyze the simulation results in detail, we used another performance metrics like number of RTO made by the protocols, time spent in RTO, number of fast retransmission and recovery calls made by the protocols, and time spent in fast retransmission and recovery phases.

4.7.4 Simulation code

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

4.7.5 Parsing the Simulation trace files

The simulation was performed for thirty 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 de-queued by Interface queue are parsed in order to extract the information needed to measure the proposed performance metrics. We have also used Network Animator and X-graph in order to analyze the simulation results visually. In addition, awk file was used to extract the average throughput (kbps), average end-to-end delay (msec), and PDR (%). The documented codes are annexed in appendix II.

4.8 Simulation results and discussion

A. Simulation Scenarios

In order to understand the behavior of TCP-PLDR in the presence of different packet loss models, the results have been compared with TCP-SACK using different simulation scenarios and network configuration

models. More precisely, our study uses the following data packet loss scenarios that are related to MANET.

1. Experiments with route failure scenario:

This scenario helps us to evaluate the effect of route failure and route changes on the performance of TCP-PLDR and TCP-SACK in the absence of congestion. We used different mobility pattern ranging from 1m/s, which represents pedestrian users up to 30m/s that can represent users driving in a high-way, travelling in train, etc,. In this scenario, we performed three different simulations by changing mobility pattern of mobile nodes. Different mobility pattern helps us to see the effect of route failure as the mobility of nodes increases and the behaviour of TCP-PLDR.

2. Experiments with both route failure and congestion loss scenario:

This scenario presents the efficiency of TCP-PLDR to differentiate packet loss due to route failure or due to congestion. The network was congested by setting four TCP flows both at the sender and receiver sides. 20 nodes were used in all simulation scenarios (available in appendix I). Among them, the first TCP flow was set between node 0 and node 9. The second TCP flow was made between node 1 and node 8. The third TCP flow was made between node 2 and node 16. Finally, the fourth TCP flow was set from node 7 to node 15. For all four TCP flows, the traffic pattern was FTP and the source and destination nodes were selected randomly.

3. Experiments with only congestion loss scenario:

This scenario tried to show how TCP-PLDR responds to congested network. It is also important to evaluate how the behavior of TCP-PLDR differs from TCP-SACK in the presence of congestion loss. In order to reduce packet losses due to route failure, the speed of 20 nodes were set to 1-2 m/s. For the first simulation, four TCP flows were used that sent data in parallel (both at the sender and receiver sides) for both TCP-SACK and TCP-PLDR to make the network severely congested. For this scenario, simulations were done for twenty times and average was taken. Then, two TCP flows were used to reduce the congestion level and to further study the performance of TCP-PLDR and TCP-SACK protocols under different congestion levels, respectively. When there is a congestion loss in the network, TCP-PLDR uses its normal congestion control mechanism as TCP-SACK does.

4. Experiments with multi-path routing protocol (TORA):

In experiment 1, 2, and 3, the underlying routing protocol used is single path routing protocol AODV, and the main causes of out-of-order delivery of packets at TCP receiver side are assumed to be route failure, route change, and node mobility. Here we may raise question, what if out-of-order delivery of data packet is not caused by using single path routing protocol like AODV. Single-path routing protocols like AODV are affected by frequent route failures and nodes mobility that has large delay due to route re-computation without sending new data packet. In order to increase the reliability of data transmission, reduce delay due to route re-computation, and send data packets during route re-establishment period, some routing protocols, such as TORA maintain multiple routes between a TCP sender-receiver pair. In such a case, packets that come from different paths may get delivered at TCP receiver side out-of-order. Since TCP receiver is unaware of multi-path

routing, it may misinterpret such out-of-order packet arrivals as a sign of network congestion.

Generally, Multi-path routing protocol affects TCP by two factors. The first one is inaccuracy of the average RTO measurement that leads to more premature time outs than the standard one. The second one is out-of-order packet delivery via different paths. We have conducted extensive simulation to validate this scenario. 20 nodes were used. The speed of nodes was chosen randomly between 5 and 15m/s. Simulation time was set to 200 seconds. All the simulations were run for thirty times by changing the position and movement of 20 nodes and average was taken. Four important performance metrics were evaluated. Goodput, throughput, end-to-end delay, and PDR.

5. Experiments with wireless channel error scenario:

In this scenario, experiment was performed in the presence of wireless channel error. The simulation was done by making the wireless links with burst errors [76] using a two-state Markov chain model (known as the Gilbert-Elliot model), which is widely used to represent error patterns or bursty traffic in packet transmission for wireless networks. As shown in Fig.4.12, Markov chain error model has two states; *good state* (error-free) and a *bad state* (erroneous), thus the chain can be represented with a matrix of 2 X 2 elements. In this simulation, *good state* means a packet received in error free manner (there is no packet loss), whereas, *bad state* indicates a packet received erroneously (packet loss). This model is defined by a transition probability matrix $\prod$ and a steady state error rate ε [77]. The transition probability matrix of this two-state Markov chain error model is given by:

$$\prod = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix} \tag{4.40}$$

The average steady state error rate ε is given by:

$$\varepsilon = \frac{1-p}{2-p-q} \tag{4.41}$$

, where ε is set to be the frame error rate produced by the simulation. The average duration of good state (D_{good}) and the average burst error duration of bad states (D_{bad}) are given by:

$$D_{good} = \frac{1}{1-p}, \quad and \quad D_{bad} = \frac{1}{1-q} \tag{4.42}$$

p is the probability of successfully transmitting a packet given the previous packet was successfully transmitted. Whereas, *1-q* is the probability of successfully transmitting a packet given the previous packet was dropped.

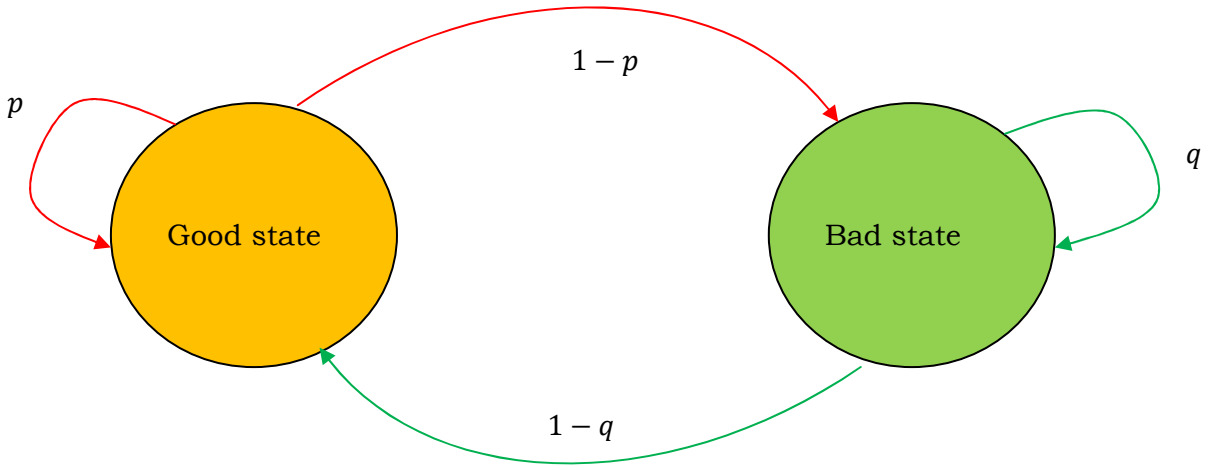


Figure 4:12 Two-state Markov chain wireless error model.

The wireless link is assumed to be in one of the two states. We assume that the wireless link is in the *good state* at the beginning of simulation. The

transitional probabilities p = 0.9913 and q = 0.8509 model the effect of burst errors [78,79]. The error rate ε = 5%. These parameters present a close imitation of burst errors in real wireless and ad hoc networks.

B. Simulation Results

1. Experiments with route failure scenario:

Fig. 4.13 shows the average throughput of TCP-PLDR and TCP-SACK, respectively. The speed of 20 nodes was chosen from 1 - 3m/s for fig.7.a. The average number of packets sent by TCP-PLDR over 200 simulated times was found to be 1932. Whereas, the average number of packets sent by TCP-SACK was found to be 1893. From this simulation result, TCP-PLDR improved the throughput of TCP-SACK by 2%. For this simulation scenario, since the speed of nodes was set to a minimum value, route failure occurs infrequently. However, as stated in eqn.31 and 32, the throughput of TCP-SACK is inversely proportional to the sum of congestion loss rate and route failure loss rate.

Whereas, the throughput of TCP-PLDR is inversely proportional to only congestion loss rate. Therefore, since route failure loss rate is set to a minimum value, both protocols respond for congestion loss rate so that the difference in their average throughput is narrower. However, this slight percentage improvement shows that for this small value of route failure loss rate, TCP-SACK triggers fast retransmit/recovery algorithms inappropriately to reduce its window size by half and retransmit packets, which are not actually lost due to congestion.

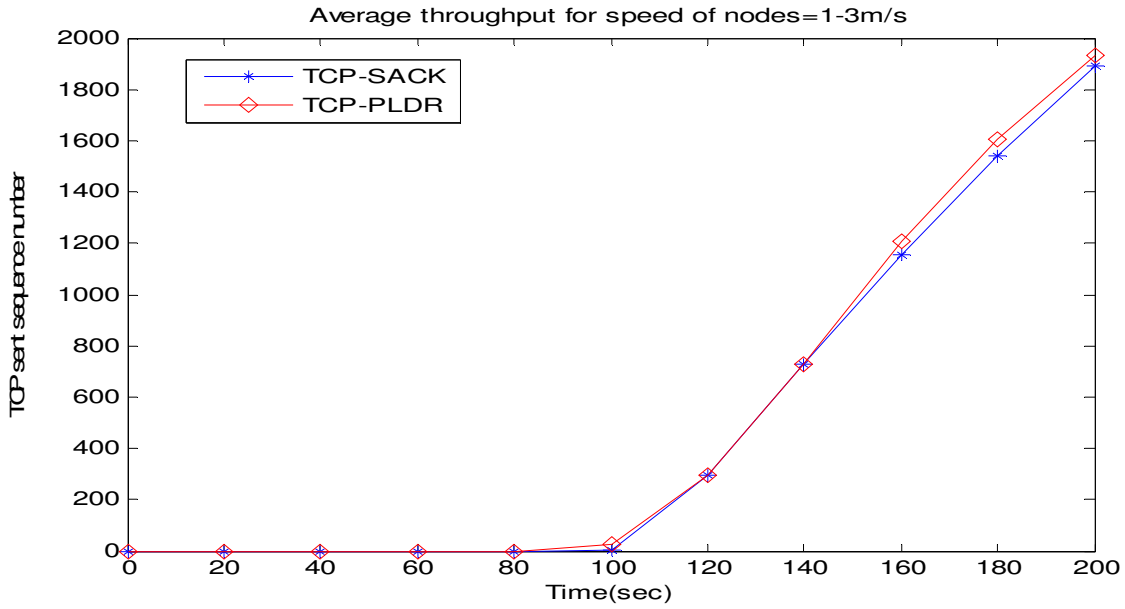


Figure 4:13 Average throughput for route failure scenarios (1 – 3m/s)

Fig.4.14 shows the average throughput achieved by TCP-PLDR and TCP-SACK protocols, respectively. For this simulation scenario, the speed of 20 nodes was selected to be 10-15m/s to clearly show that as the speed of nodes increases, route failure can occur frequently. Simulation results showed this fact. TCP-PLDR sent on average about 2496 data packets over 200 simulation seconds. Whereas, TCP-SACK sent about 2257 packets.

Once again, TCP-PLDR achieved a higher percentage improvement, which is about 10.6% over the naïve TCP-SACK. From this simulation result, it can be concluded that as the speed of nodes increase, route failure and route changes can occur frequently. TCP-SACK assumes that out-of-order packet arrival due to route change and packet loss events due to route failure as a sign of network congestion. Consequently, it invokes congestion control algorithm inappropriately. However, the implementation of TCP-PLDR tried to differentiates the exact cause of packet loss by postponing calling of fast retransmit/recovery algorithms and responds appropriately that leads to better throughput than TCP-SACK.

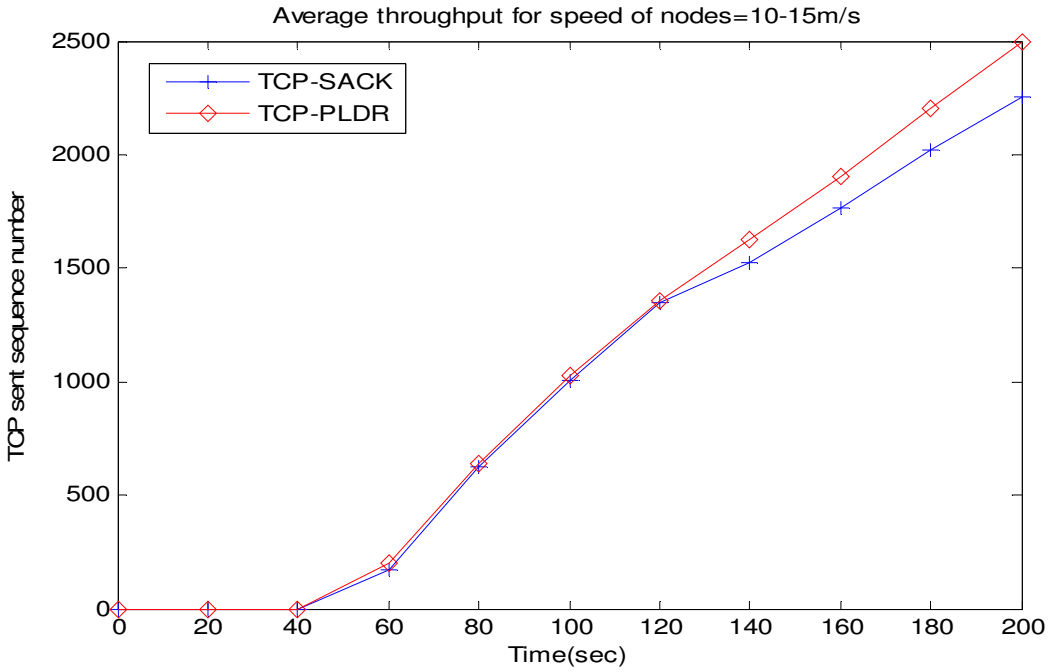


Figure 4:14 Average throughput for route failure scenarios (10 – 15m/s)

In fig.4.15, once again simulation was performed to find the average throughput of TCP-PLDR and TCP-SACK. In this scenario, the speed of nodes was selected randomly between 25m/s and 30m/s. On the average, TCP-PLDR sent 3125 data packets. Whereas, TCP-SACK sent about 2692 packets over 200 simulated time. TCP-PLDR has achieved better performance, which is 16% over TCP-SACK. Generally, TCP-PLDR achieved better throughput than TCP-SACK as the speed of nodes increase in ad hoc networks. Hence, TCP-PLDR is more efficient to detect packet loss where there is frequent route failure and responds appropriately, which was described in eqn.4.34-36.

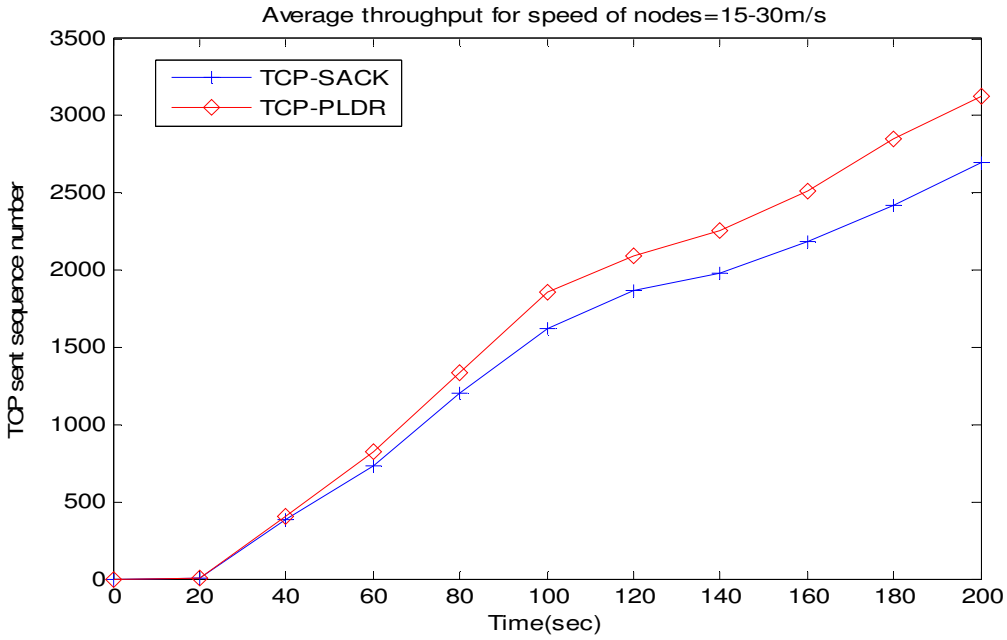


Figure 4:15 Average throughput for route failure scenario (25 – 30m/s)

In fig.4.16, we have compared the performance of TCP-PLDR and TCP-SACK under different nodes' mobility pattern. In this MANET configuration, the speed of 20 nodes was selected to be 2m/s, 15m/s, and 30m/s. On the average, TCP-PLDR had achieved 9.5% improvement over TCP-SACK. From the simulation result, it is confirmed that as the speed of nodes increases from 2 to 15 to 30m/s, the percentage improvement achieved by TCP-PLDR was found to be 2%, 10.6%, and 16.1%, respectively over TCP-SACK. This performance improvement confirmed that, TCP-PLDR is efficient enough to differentiate the type of packet losses as the speed of mobile nodes increases.

In MANET, as the speed of nodes increases, route failure can occur frequently. In this case, multiple packets can get lost or out-of-order packets can be delivered at TCP receiver side. Analytical result (eqn. 4.34-4.36) also stated that throughput of TCP-PLDR increases with Pr and decreases with Pc. Since there is no congestion in the network, the throughput is only managed by receiver's advertised window. Consequently, when there is no congestion loss in the network, TCP-PLDR can efficiently utilize all the available

bandwidth irrespective of increasing the probability of route failure losses. However, in the case of TCP-SACK, the throughput is proportional to $\frac{1}{RTT\sqrt{Pc+Pr}}$. When there is no congestion, only Pr (route failure loss) is taken into consideration. This loss depends on the speed of mobile nodes. Then, for any particular value of Pr, the throughput achieved by TCP-SACK sender is dependent on this route failure loss rates. Since the rate of route failure loss is incremented by increasing speed of mobile nodes, TCP-SACK sender misinterprets this loss as a sign of congestion, which leads to lower throughput.

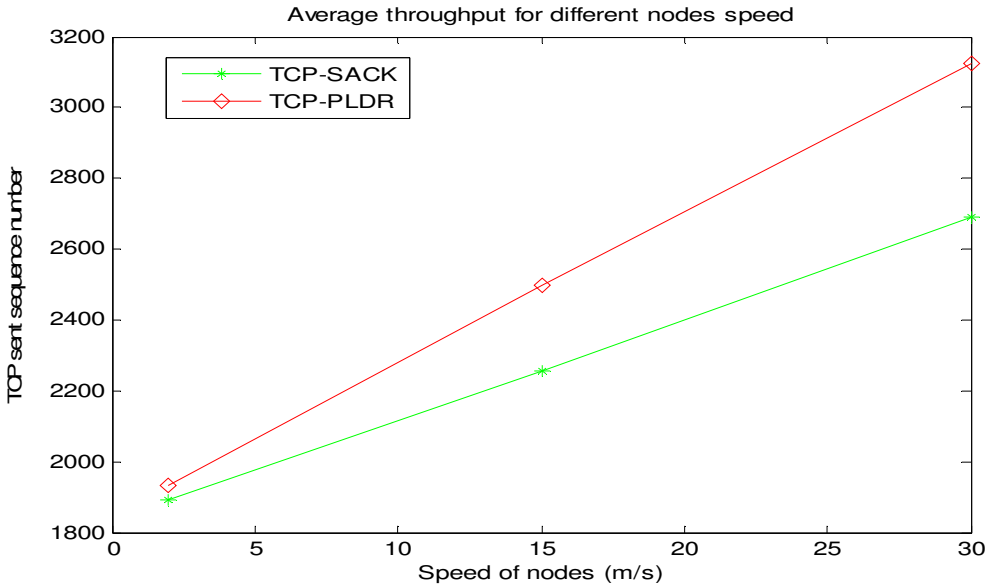


Figure 4:16 Average throughput for route failure scenarios (2, 15, and 30m/s)

2. Experiments with both route failure and congestion loss scenario:

Fig.4.17 shows the average throughput of TCP-PLDR flows in comparison with TCP-SACK flows. As shown in the graph, TCP-PLDR had achieved a 39% improvement than TCP-SACK. This percentage improvement is better than the one which are found in section B.1 (experiments with only route failure scenario). This percentage improvement would mean that TCP-PLDR is more efficient than TCP-SACK to distinguish between congestion and route failure

losses. As stated in eqn.4.34, the throughput achieved by TCP-SACK flows is inversely proportional to the sum of congestion loss rate and route failure loss rate. Whereas as per eqn.35, the throughput achieved by TCP-PLDR is inversely proportional to only congestion loss rate. Therefore, as the congestion loss rate increases, the difference in average throughput of TCP-PLDR and TCP-SACK becomes insignificant. On the other hand, as the route failure loss rate increases, the throughput of TCP-SACK is significantly reduced as there is no mechanism to differentiate whether packet is lost due congestion or route failure. In TCP-PLDR, as route failure loss increases, it shouldn't reduce its sending rate. Consequently, it achieved better throughput than TCP-SACK.

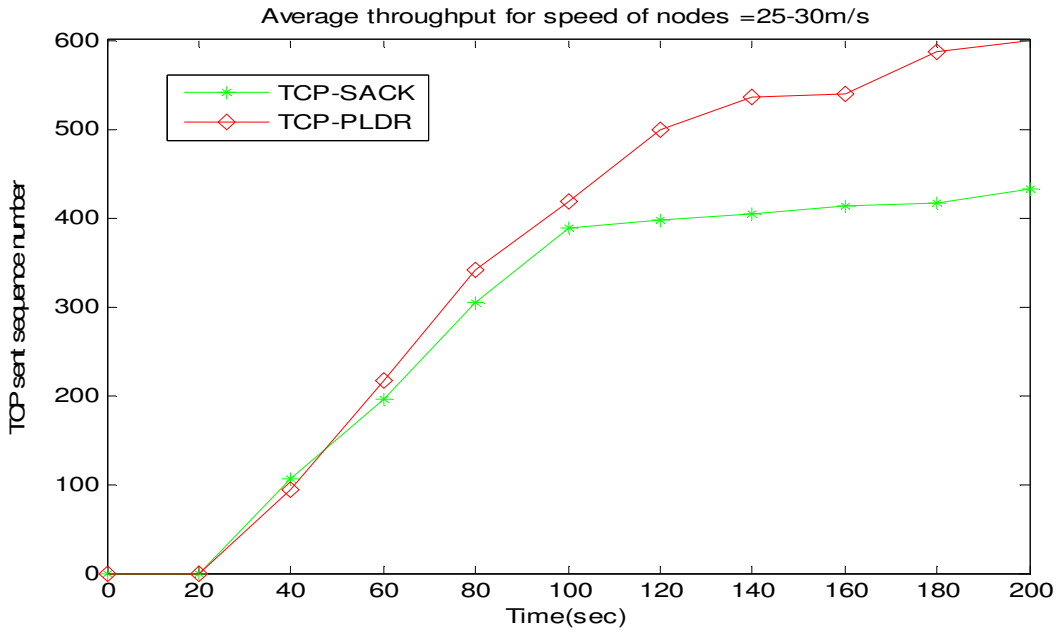


Figure 4:17 Average throughputs for route failure and congestion loss scenarios.

3. Experiments with only congestion loss scenario:

In this simulation scenario (fig.4.18), the network was congested, but movement of nodes was chosen randomly between 1-2m/s. Since the speed

of nodes was set to be a minimum value, it was possible to reduce the frequency of route failure and change. Consequently, it helps us to evaluate the performance of TCP-PLDR with TCP-SACK when there is only congestion loss in the network. TCP-PLDR is modified at TCP-SACK (both sender and receiver side) to make it more robust for route failure/change packet loss events. In the absence of route failure, it should behave like normal TCP-SACK. The simulation result showed this fact. The average throughput of TCP-PLDR and TCP-SACK was found to be almost the same. On the other hand, this result confirms that TCP-PLDR is able to handle correctly congestion loss scenario and triggers appropriately congestion control algorithm to recover from packet loss.

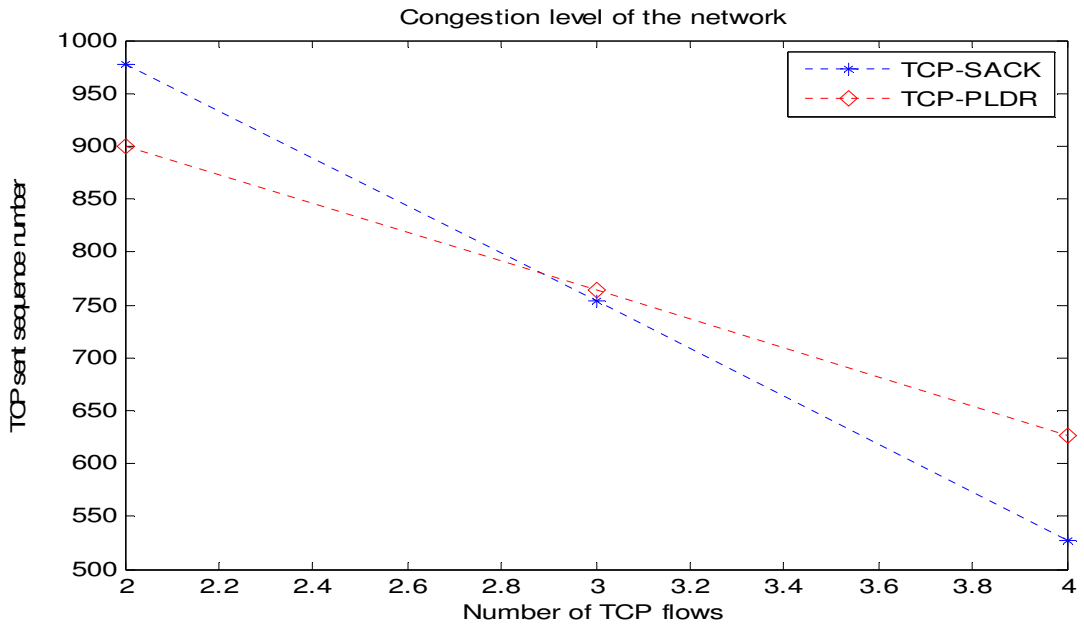


Figure 4:18 Average throughputs for only congestion loss scenario

Fig.4.19 shows the average congestion window size of TCP-PLDR and TCP-SACK over 200 simulated seconds. TCP-PLDR achieved a larger *cwnd* than TCP-SACK, especially at steady state; at 100 simulations second, TCP-PLDR had achieved better *cwnd* than TCP-SACK. Larger *cwnd* means better

throughput.Theoretically, it was also stated that at steady state the throughput of TCP-PLDR is better than that of TCP-SACK.

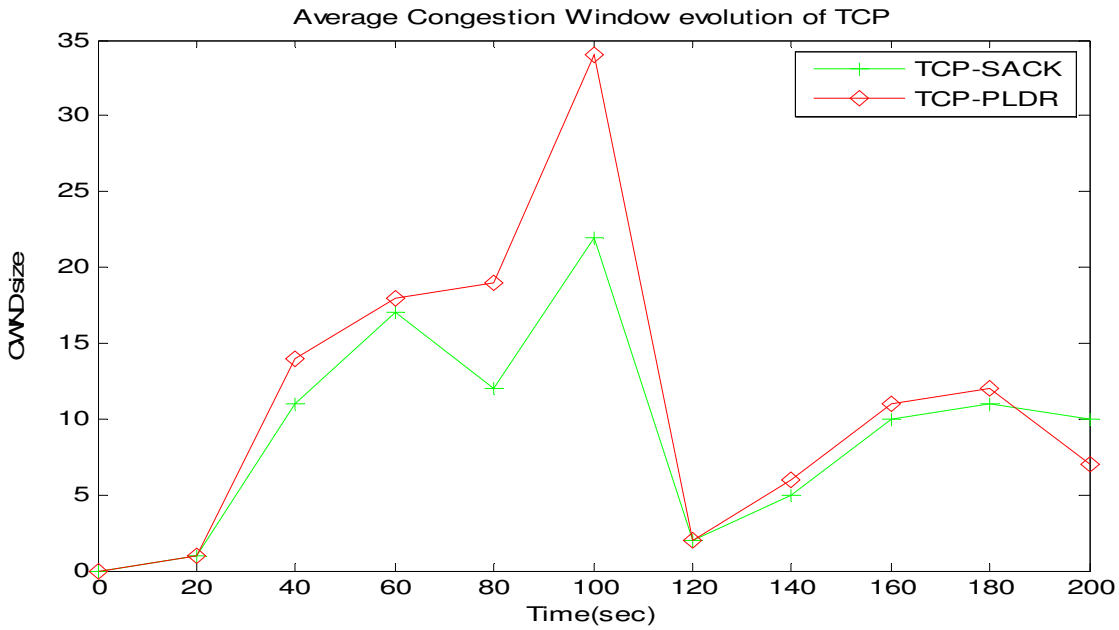


Figure 4:19 Average congestion window size (cwnd) over simulated time

Fig.4.20 shows the average retransmission time out (RTO) calls and the number of time spent in RTO achieved by both TCP-PLDR and TCP-SACK protocols, respectively. If the RTO increases, *cwnd* is not increased. However if the RTO decreases, or remains constant, *cwnd* is increased according to TCP's rule. It can also be stated as, calling RTO frequently would mean that TCP sender should enter slow start phase frequently, which reduces *cwnd* to start from one segment. The main drawback of entering slow start phase is large amount of time required during start up. Generally, RTO forces TCP to reduce data packet flow drastically. As shown in fig.4.20, on the average TCP-PLDR spent little time in time-out (29.538 seconds) than TCP-SACK that spent larger time (44.846 seconds) in RTO. At the same time, the number of time out call achieved by TCP-PLDR (12.4) is significantly better than that of TCP-SACK (18.8). From these simulation results, it is confirmed that:

- TCP-PLDR uses the scares bandwidth of ad hoc networks better than TCP-SACK.
- Packet loss due to route failure/change is detected through time stamp option (explicit loss detection) than using three duplicated acknowledgments (implicit loss notification scheme).
- The disabling period T=RTT value is thus an optimum time to give information regarding route/link failure in ad hoc networks (theoretically, it was also stated in section 4.3).
- In TCP-SACK, after route failure, discovering new routes takes significantly longer time than the RTO interval at the sender. As a result, TCP-SACK sender times-out then retransmits a packet and invokes congestion control algorithm frequently. However, in TCP-PLDR, the disabling period T, which is selected to be one times RTT reduces significantly the problem of frequent time out events. (Theoretically it was also stated in section 4.3 and 4.4).

Fig.4.21 shows the average time spent and the number of calls of fast retransmission/recovery algorithms made by TCP-PLDR and TCP-SACK protocols, respectively. It is shown that TCP-PLDR spent little time (3 secs) in fast retransmission/recovery phases than TCP-SACK (7 secs). It is also identified that TCP-PLDR is better than TCP-SACK in calling fast retransmission/recovery algorithms. On the average, TCP-PLDR calls fast ret/rec 7 times. Whereas, TCP-SACK calls about 12 times.

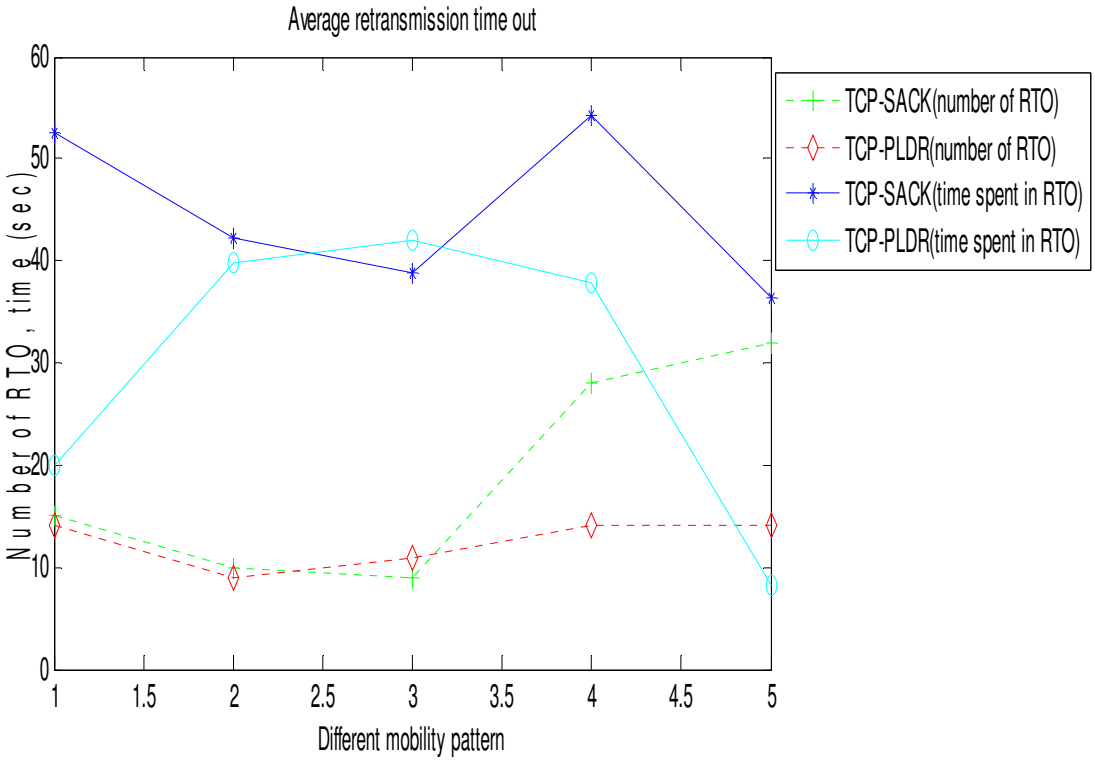


Figure 4:20 Average Retransmission time out (RTO) measurements

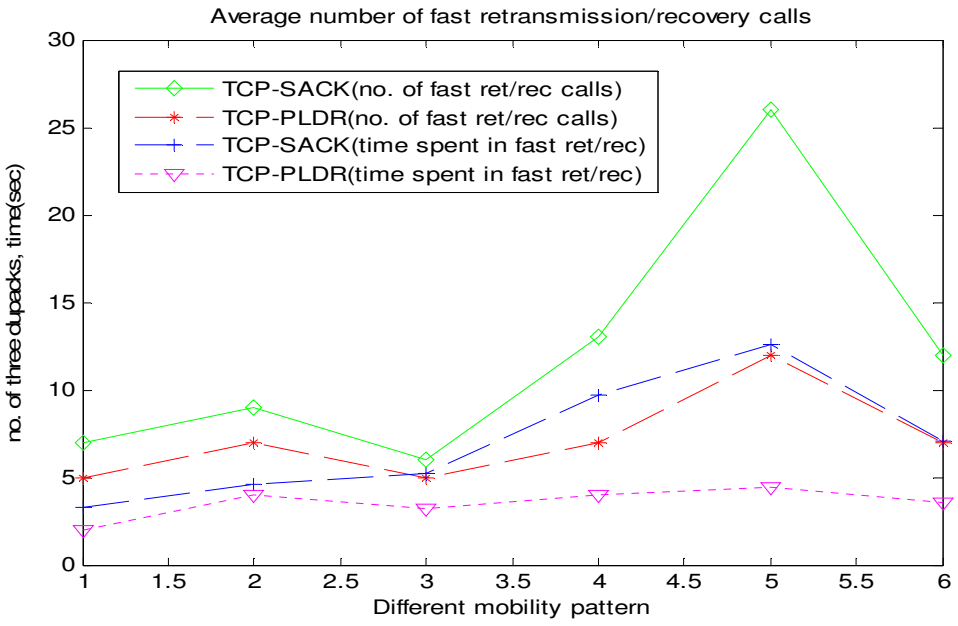


Figure 4:21 Average Fast Retransmission/ Recovery calls measurement

4. Experiments with multi-path routing protocol (TORA):

Fig.4.22 shows the average goodput measurement of TCP-PLDR and TCP-SACK protocols. TCP-PLDR (T=1*RTT) received on average about 1738 packets over 200 simulation seconds. Whereas, TCP-PLDR (T=2*RTT) received about 1665 packets. Finally, TCP-SACK received about 1549 packets. From this simulation results, TCP-PLDR (T=1*RTT) improves the performance of TCP-SACK by 12%. Whereas, TCP-PLDR (2*RTT) improves TCP-SACK's performance by 7.5%. However, compared to disabling period T = 1*RTT, the goodput is decreased by 4.5%. This improvement implies that out-of-order packets are delivered due to multi-path routes in TCP-SACK, which degrades its goodput and implementing TCP-PLDR have improved the effect of out-of-order problem. Moreover, disabling period of one time the round trip time is an optimum time for achieving better result. Theoretically, it is also stated in section 4.3 that whenever the disabling period is set to $T = 2 * RTT$, TCP-PLDR protocol would not respond fairly. Disabling period $T = 1 * RTT$ is therefore an optimum time for the TCP-PLDR to recover from out-of-order packet delivery or packet loss events at the receiver side. Generally, in all scenarios TCP-PLDR performs significantly better than TCP-SACK.

Fig.4.23 shows the average throughput measurements of TCP-PLDR and TCP-SACK over multi-path routing protocol TORA. On average, the throughput of TCP-PLDR was 82kbps, whereas, TCP-SACK's throughput was found to be 73kbps. From these simulation results, TCP-PLDR improves TCP-SACK's performance by about 12%. Better throughput means better utilization of available bandwidth, particularly in MANET where channel bandwidth is scarce.

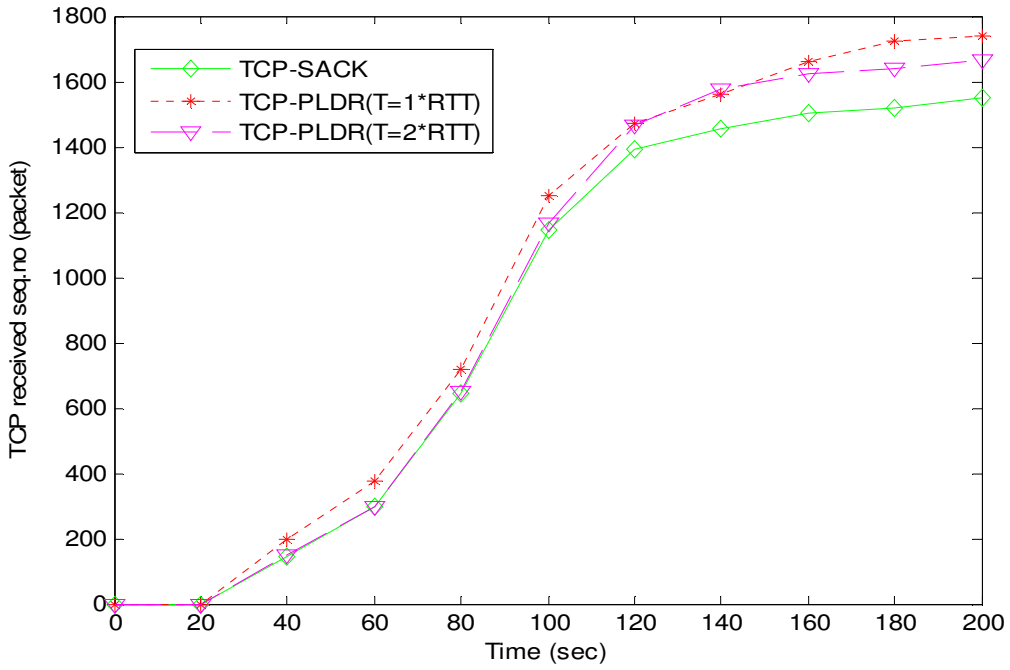


Figure 4:22 Average goodput measurements of TCP-PLDR and TCP-SACK

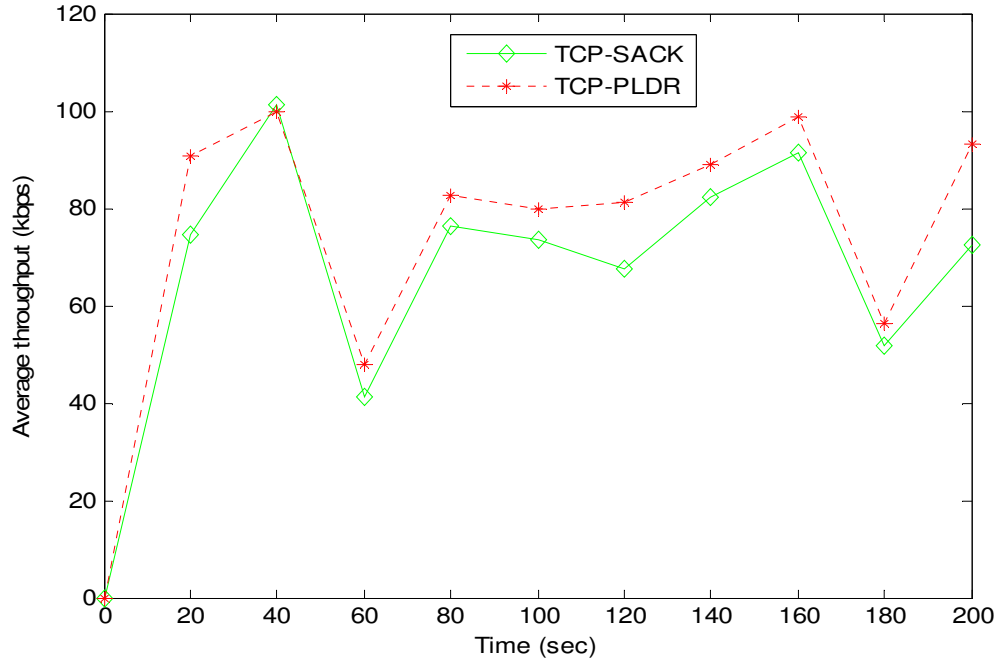


Figure 4:23 Average throughput measurements of TCP-PLDR and TCP-SACK

Regarding end-to-end delay measurements shown in fig.4.24, ten different simulation scenarios were taken by changing the position and movement of mobile nodes to collect fair results. In some simulation scenarios, TCP-PLDR showed lower delay, whereas, in the other scenarios, TCP-SACK performed lower delay. However, average result showed that TCP-PLDR achieved end-to-end delay of about 415.6 msec, and TCP-SACK achieved about 410.4 msec. These numerical results imply that by far SACK is better than PLDR with respect to delay measurement for multi-path routes in MANET.

Results for PDR are shown in fig.4.25. On the average, TCP-PLDR attains about 96.22 %. On the other hand, TCP-SACK achieves 96.1%. Therefore, with respect to PDR, both protocols showed almost the same results.

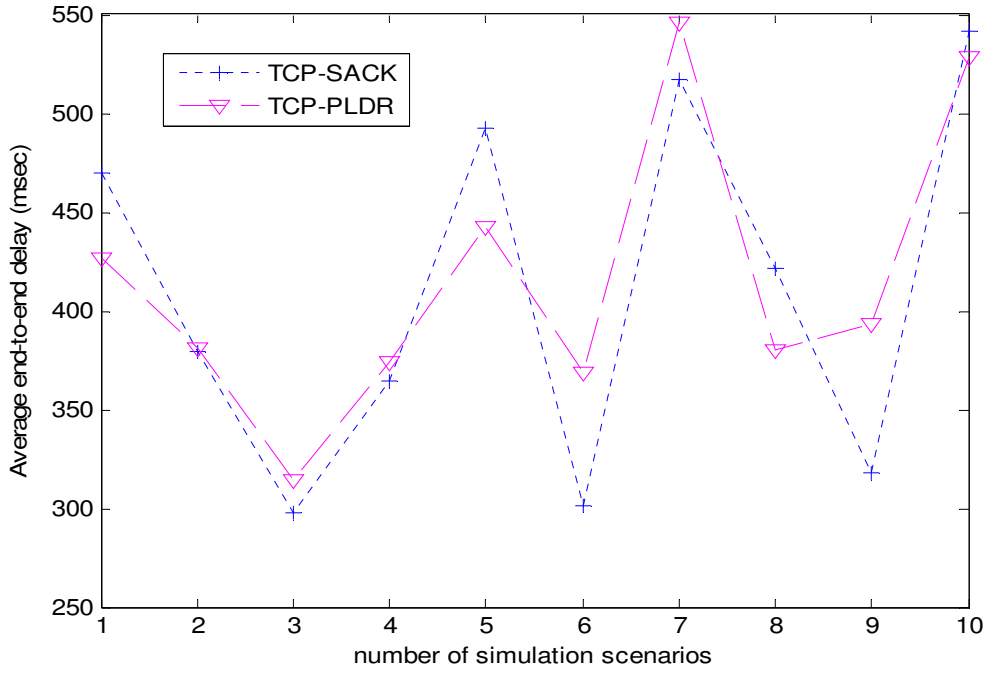


Figure 4:24 Average end-to-end delays of TCP-PLDR and TCP-SACK

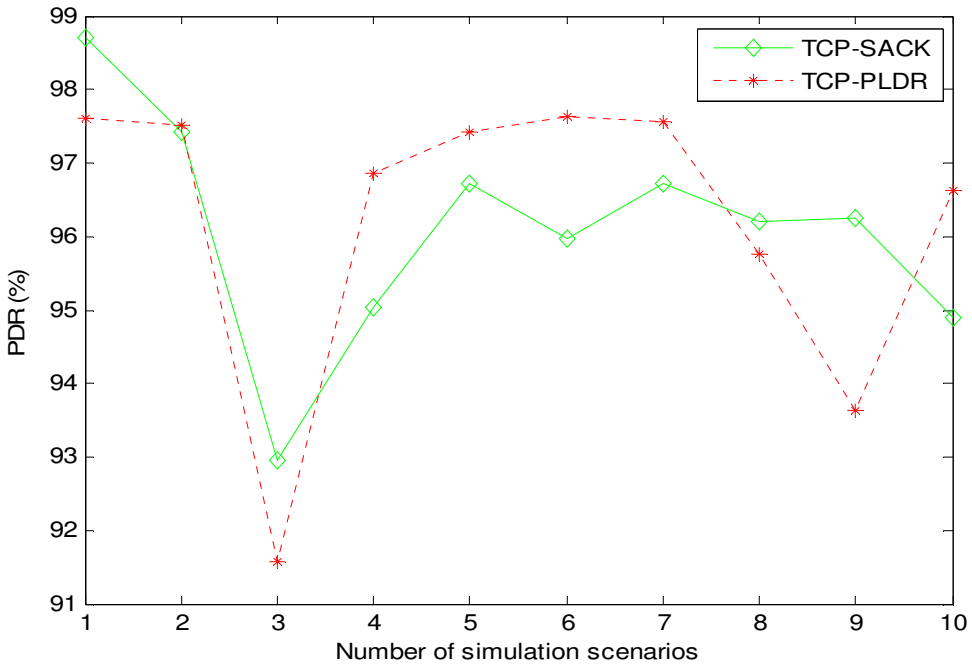


Figure 4:25 Average packet delivery ratios of TCP-PLDR and TCP-SACK

5. Experiments with wireless channel error scenario:

Fig.4.26 shows the average throughput of TCP-SACK and TCP-PLDR protocols when the network is under 5% wireless channel error. Once again, performance metrics used were throughput, end-to-end delay, and PDR. From the simulation results, it is found that TCP-PLDR improves the throughput of TCP-SACK by 6.25%. Wireless channel error makes packets get dropped. In this case, packets can get delivered at the receiver side out-of-order. In TCP-SACK, this is treated as congestion. Consequently, TCP-SACK sender's congestion window size stays at a small value, which reduces the throughput significantly.

Performing appropriate congestion control action may sounds good when a packet loss is due to congestion. However, it can reduce throughput of TCP if a packet loss is due to wireless channel error. In this regard, TCP-PLDR

performs better than TCP-SACK. Since there is no congestion in the network, and the packets are only reached out-of-order at the receiver side, most packets are recovered within disabling period T=RTT, which are mathematically analyzed is section 4.3. Therefore, disabling period T=RTT is an optimum time for reordering packets (link level retransmission is possible in IEEE 802.11) to get delivered at the receiver side of TCP-PLDR protocol. Consequently, TCP-PLDR doesn't reduce its congestion window size within disabling period with the assumption that packets are lost due to channel error so that link level retransmissions helps to deliver packets for TCP, but not lost due to congestion.

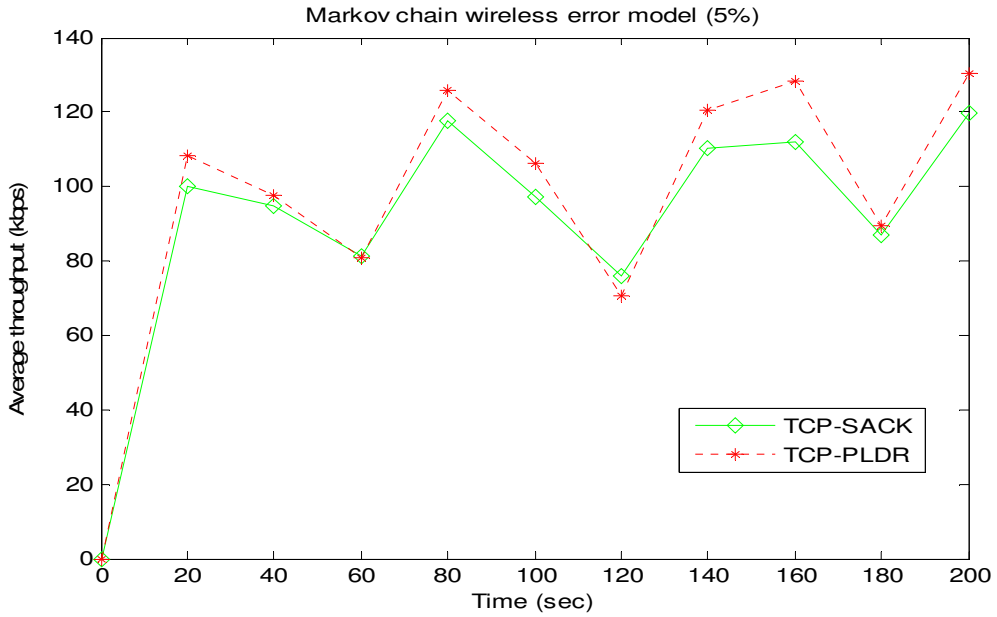


Figure 4:26 Average throughputs with 5% wireless channel error.

Fig.4.27 shows the average end-to-end delay of TCP-SACK and TCP-PLDR protocols in the presence of 5% wireless channel error (Markov chain error model). From the simulation results shown in the figure, it is found that on the average, TCP-SACK achieved about 314.2 msec end-to-delays. Whereas, TCP-PLDR attained about 292.8 msec delays. From this data, the average delay of TCP-PLDR was reduced by 21.4msec. The lower the end-to-end delay

means the better performance of the protocol, which is a significant figure for the improvement we have expected from TCP-PLDR.

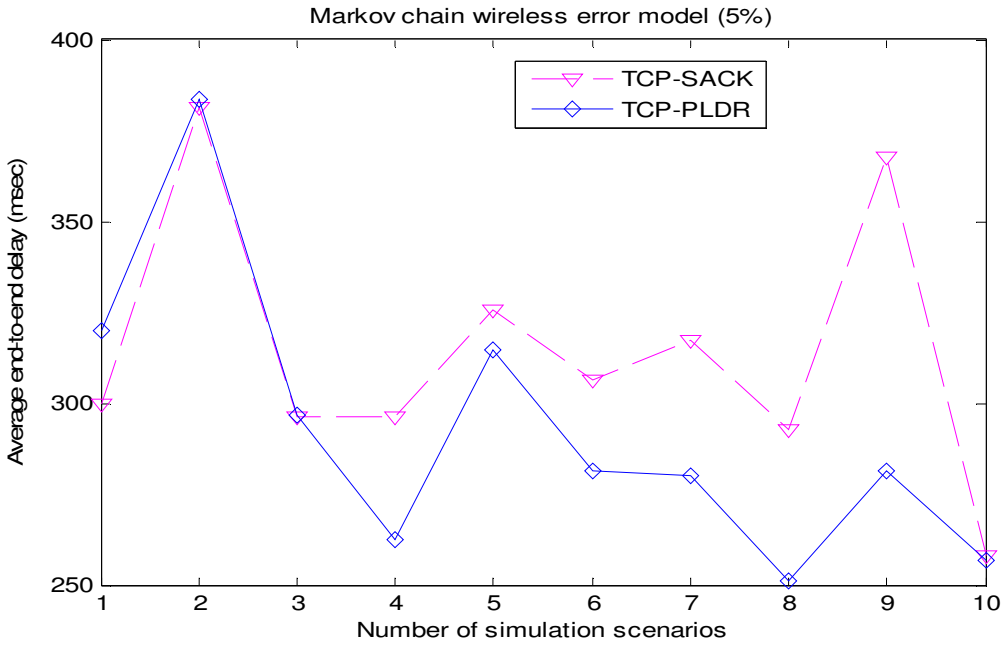


Figure 4: 27 Average end-to-end delays with 5% wireless channel error

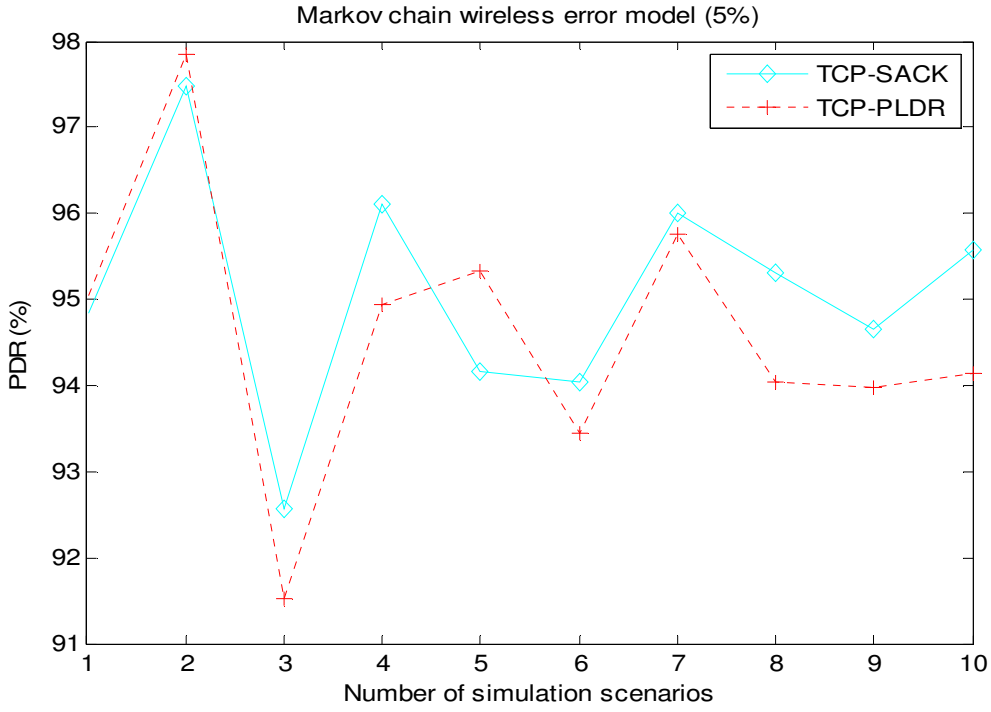


Figure 4:28 Average packet delivery ratios with 5% Wireless channel error

4.9 Summary

This chapter introduced and evaluated TCP-PLDR protocol, which mainly distinguish between packet losses due to congestion in the network and route failure due to mobility of nodes, in MANET. It is an end-to-end solution, which doesn't require any cooperation from intermediate nodes. The proposed algorithm was described with its implementation. And it is confirmed through analysis and simulation that timer based packet loss detection scheme (explicit loss detection scheme) is better than acknowledgment based scheme (implicit loss notification scheme).

Analysis of selecting disabling period T=RTT was presented to make sure that it shouldn't be less than RTT and greater than RTT. To validate and determine the lower and upper bounds of this assumption, experiment was performed for T=1*RTT and T=2*RTT using multi-path routing protocol. Results showed that disabling period T=RTT is an optimum time to recover from different types of packet losses (route failure, wireless channel error, etc.,).

Analytical model of TCP-PLDR and TCP-SACK were investigated in detail, which captures the throughput behavior of both protocols as a function of *RTT* and packet loss probability *p*. The model also captures the behavior of both protocols in the presence of congestion and route failure losses. The result showed that TCP-PLDR is TCP friendly while it improves the throughput of TCP-SACK when there is packet loss due to route failure, multi-path routes, and those due to wireless channel error, where they significantly contribute to out-of-order packet delivery in MANET.

Simulation was conducted using ns-2 for different simulation scenarios and experiments and results showed that TCP-PLDR improves the performance of

TCP-SACK. Moreover, the proposed protocol was evaluated in the presence of wireless channel error with 5% burst error by using Markov chain error model, and multi-path routes. Experiment with route failure scenario has shown that TCP-PLDR improves the performance of TCP-SACK by 2%, 11%, and 16% for speed of nodes 1-3m/s, 10-15m/s, and 25-30m/s, respectively. Experiment with both route failure and congestion loss scenario demonstrated that the improvement achieved by TCP-PLDR over SACK was found to be 39%. Whereas, both PLDR and SACK almost achieved the same result for experiment for only congestion loss scenario.

It is also noted that average *cwnd*, average number of fast retransmission/recovery and RTO calls made by both protocols were experimented and results have shown that TCP-PLDR performed better than TCP-SACK. Experiment for multi-path routes has demonstrated that TCP-PLDR improved the performance of naïve TCP by 12%. However, end-to-end delay measurement has shown that TCP-SACK is better than PLDR. For PDR, both protocols achieve almost the same results. Finally, experiment with wireless channel error scenario showed that in terms of throughput and delay measurements, TCP-PLDR improved TCP-SACK's performance by almost 6.5%. In terms of PDR, both protocols performed the same results.

CHAPTER FIVE

PERFOROMANCE OF TCP VARIANTS OVER PROACTIVE AND REACTIVE ROUTING PROTOCOLS IN MANET

5.1 Introduction

The main function of any network is reliable delivery of data. As stated in introduction section of chapter one, TCP, which was previously developed for reliable end-to-end delivery of data over unreliable wired networks, by ignoring the properties of MANET like mobility of nodes can lead to TCP's implementation with poor performance. One of the major problems of TCP in MANET is unable to distinguish between losses due to congestion and those due to route/link failures. Route failures in MANETs are frequent events, if route fails while transmitting packets, routing protocols should re-establish the failed route. However, since TCP variants do not have mechanism to know the route re-establishment period, the throughput can degrade because of the large delay to transmit data packets. Also, if the new route created is longer or shorter than the old route, then TCP may not be in a position to estimate RTT appropriately. Moreover, packets may reach at the TCP receiver side out-of-order, in this case TCP can assumes that packet is lost in transit, so it retransmits packets unnecessarily and reduces the amount of packets to be transmitted, which affects the throughput of TCP a lot. Therefore, the performance of TCP is largely dependent on the underling routing protocols.

Generally, in MANET the major problem of all TCP variants lies in performing congestion control action in case of losses that are not caused by network

congestion. Since packet loss due to mobility of nodes is very low in wired networks, all TCP variants assume that packet losses are an indication of network congestion. Consequently, when a packet is detected to be lost, either by timeout or by three duplicated acknowledgments, TCP slows down the sending rate, which degrades TCP's throughput a lot. Therefore, the way how routing protocols respond to route failures and route recovery mechanism has an effect on the performance of TCP variants.

In this section, the effect of proactive and reactive routing protocols on the performance of TCP variants is studied. At the end of this study, the best performing combination of TCP variants and routing protocols were selected. The combination includes variants of routing protocols, variants of TCP, variation in mobility pattern, and variation in node density. Moreover, TCP-PLDR is evaluated over different proactive and reactive routing protocols and other well-known TCP variants.

5.2 Simulation Scenarios and Model

To study the effect of routing protocols on the performance of TCP variants, a detailed simulation scenarios and models were prepared. The simulation was performed using ns-2. Ns-2.29 version was used to configure different routing protocols (DSDV, DSR, AODV, and TORA); TCP variants (TCP-Tahoe, TCP-Reno, TCP-Newreno, TCP-Sack, TCP-PLDR) are used. MAC protocol used was 802.11. Table 5.1 summarizes simulation parameters.

Table 5:1

Simulation parameters and setup

Parameters	**Value**
Number of nodes	5 and 20
Speed of nodes	1-3m/s and 15m/s
MAC protocol	IEEE 802.11
Routing protocols	DSDV, DSR, AODV, TORA
TCP variants	TCP-Reno, TCP-Tahoe, TCP-Newreno, TCP-Sack, and TCP-PLDR
Network topology	900m X 1000m
Application	FTP
Simulation time	200 sec
Antenna type	Omni-antenna
Signal propagation model	Two-Ray Ground model

5.2.1 Traffic and mobility model

For traffic source, FTP was used above TCP variants. The network topology was chosen to be 900m X 1000m. Number of nodes used in the evaluation was varied to be 5 and 20. The speed was chosen randomly between minimum of 2 m/s and maximum of 15 m/s. The source-destination pairs are spread randomly over the network. 1040 byte data packets are used from sender to receiver and 40 byte acknowledgments are used from receiver to sender. All the simulations were run for 200 seconds. To make the result fair, all the simulations were performed for more than 700 times by changing the position and movement of nodes and average was taken. The signal propagation model used was the two-ray ground model where signals propagate from sender to receiver in an open environment and

over two mechanisms; one by direct ray and one that is reflected from the ground. This is the standard propagation model used for TCP evaluation in MANET.

5.2.2 Performance metrics

Major performance metrics used is throughput. In order to analyze the simulation results further and in detail, TCP's state variables like *cwnd*, *ssthresh*, duration and number of time-out measurement, duration and number of fast retransmission/recovery measurements made by TCP variants were used.

5.3 Simulation results and discussions

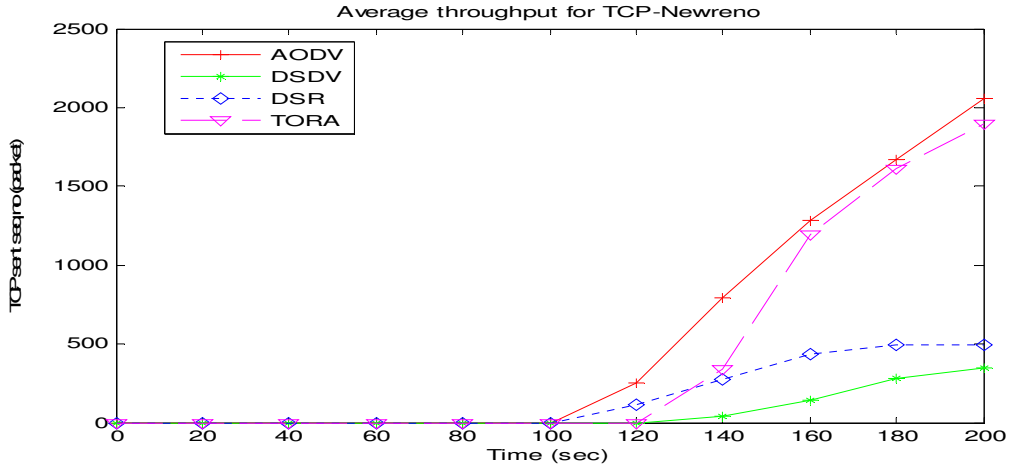
This section discusses the performance investigation and evaluation of the effects of routing protocols namely; DSDV, DSR, AODV, and TORA on the performance of TCP variants; TCP Tahoe, TCP-Reno, TCP-Newreno and TCP-Sack in MANETs. At the end of this evaluation, the best performing combination of TCP variants and routing protocols shall be selected. Finally based on the stated scenarios, suitable routing protocols shall be selected for each TCP variants, which was used as a valuable input for the proposed algorithm, which was discussed in chapter four.

5.4 Performance of routing protocols on TCP variants

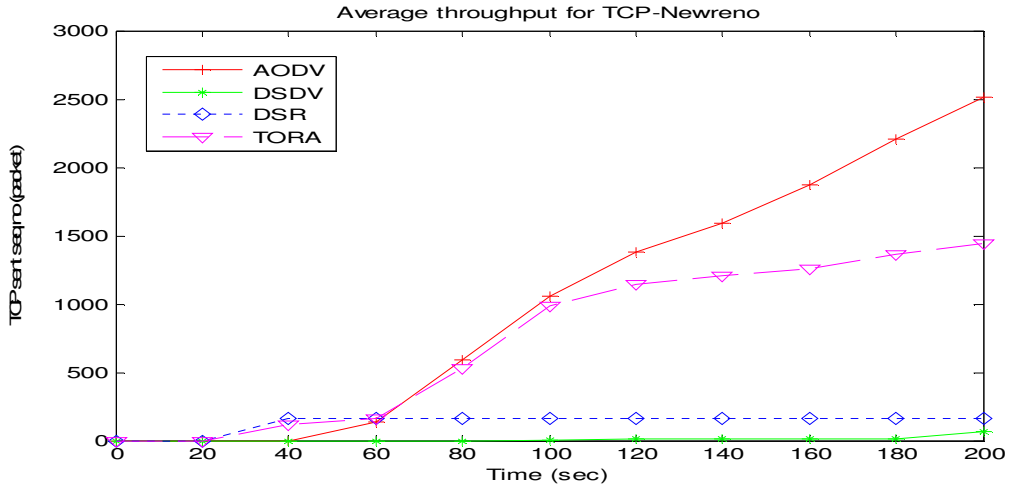
A. For different mobility pattern

First, all the TCP variants were evaluated over the stated four routing protocols. Fig.5.1 shows the average throughput of TCP-Newreno over AODV, DSDV, DSR, and TORA routing protocols. In this simulation, the speed of 20 nodes was selected randomly to a minimum value of 2 m/s, fig.5.1a and maximum of 15 m/s, fig.5.1b. From both figures, AODV

performed better than DSR, DSDV, and TORA for TCP-Newreno. The reason is that, AODV took an advantage of sequence numbering for fresh routing information from DSDV, and reducing control message overhead from DSR. Getting fresh route and reducing control message make AODV the fastest to get route, which has a positive impact on the performance of TCP. In terms of throughput, DSDV achieved the least throughput than the other three routing protocols. From the simulation result, DSR, DSDV, and TORA are more suitable for networks with lower speed of nodes. Whereas, AODV outperforms the other routing protocols for both lower and higher speed of nodes.



a) Speed = 2m/s

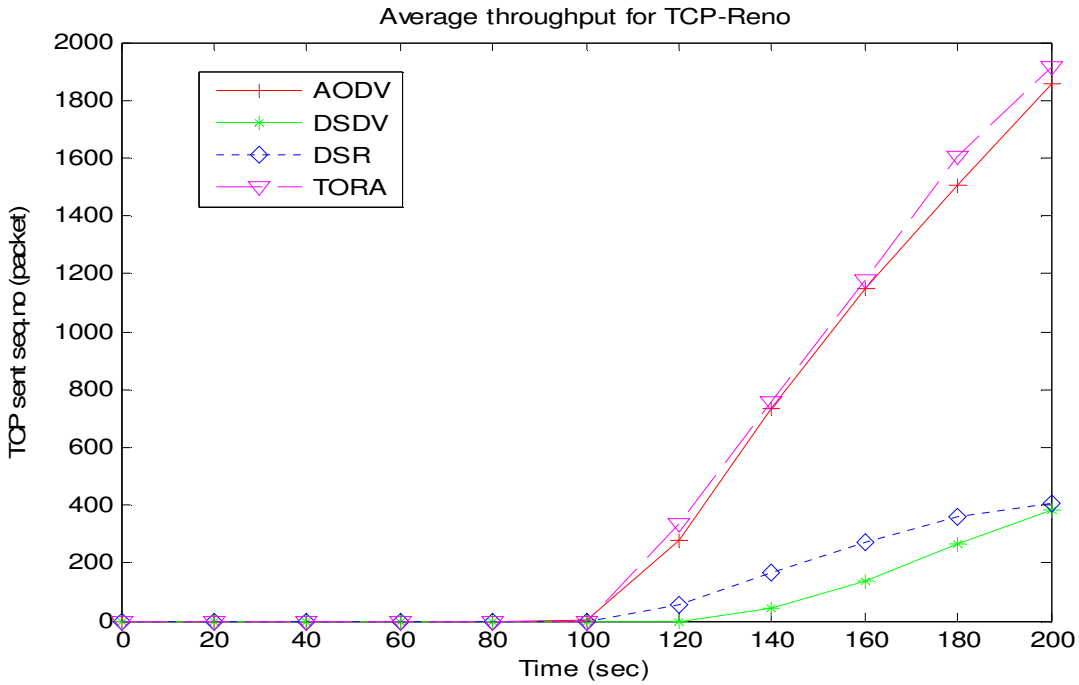


b) Speed = 15 m/s

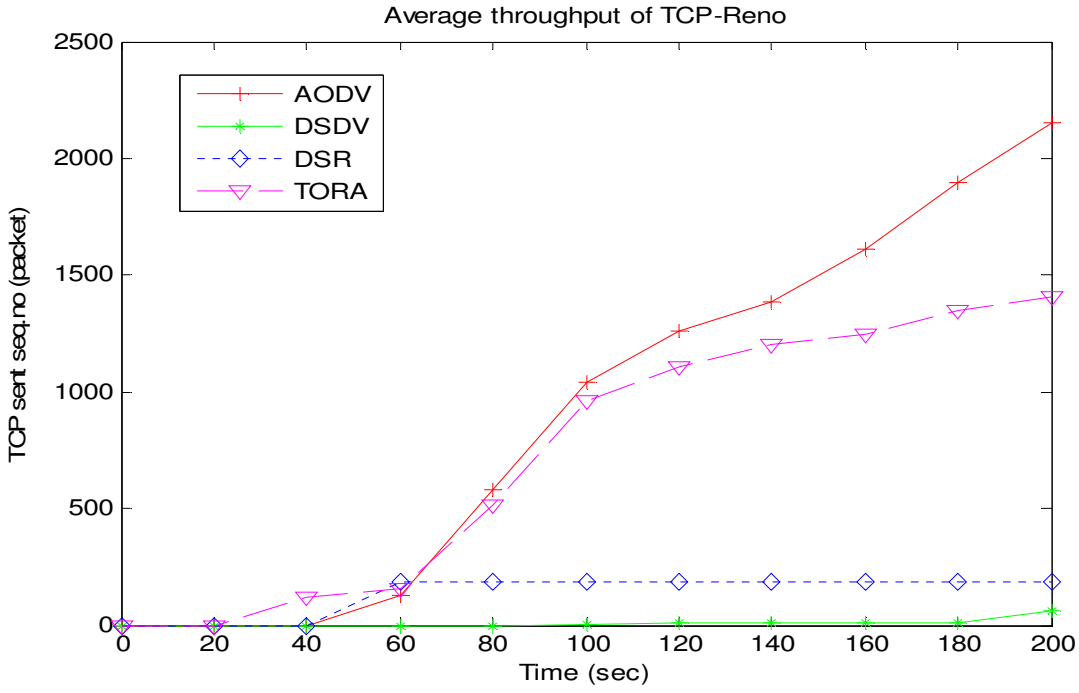
Figure 5:1 Average throughput of TCP-Newreno for speed 2m/s and 15m/s

Fig.5.2 shows the average throughput of TCP-Reno over AODV, DSDV, DSR, and TORA routing protocols for nodes speed 2 and 15 m/s. From the simulation results, AODV and TORA achieved higher throughput than DSR and DSDV for both higher and lower speed of nodes with TCP-Reno. The main problem of DSR with lower throughput is, freshness of route is not known and most importantly, introduces large delay due to route re-computation time in case of route failures due to mobility of nodes. In this case, in the absence of congestion, TCP sender assumes that packet was lost in transit. Hence either TCP sender's RTO can get expired or receives three duplicated acknowledgments, which invokes congestion control algorithm to retransmit packets and reduce *cwnd* that degrades the throughput of TCP a lot.

The least throughput was achieved by DSDV, the reason is that DSDV maintain large route information, which may not be required, and exchange of control messages among nodes to maintain consistent and up-to-date route information that leads to consumption of scares bandwidth resource of MANET. If bandwidth is congested with control messages, then TCP might not send packets to its maximum window size that degrades the performance of TCP.



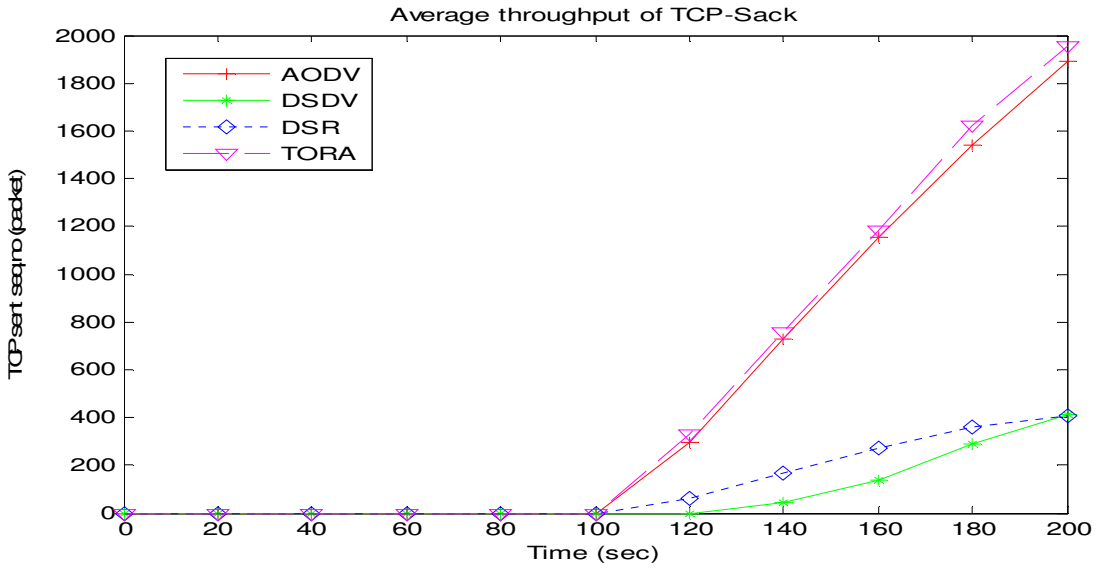
(a) speed=2m/s



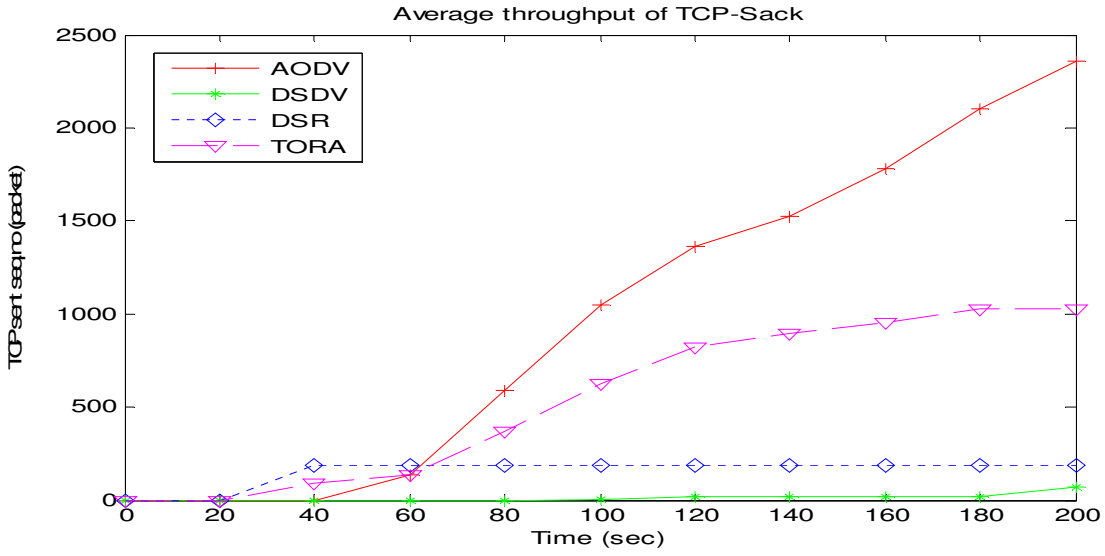
(b) speed=15m/s

Figure 5:2 Average throughput of TCP-Reno for speed 2m/s and 15m/s

Fig.5.3 a and b show the average throughput of TCP-Sack over AODV, DSDV, DSR, and TORA routing protocols for nodes speed of 2 and 15 m/s. Once again, the performance of AODV is the best in terms of throughput. AODV even achieved a higher throughput for higher speed of nodes. DSDV performs the least throughput.



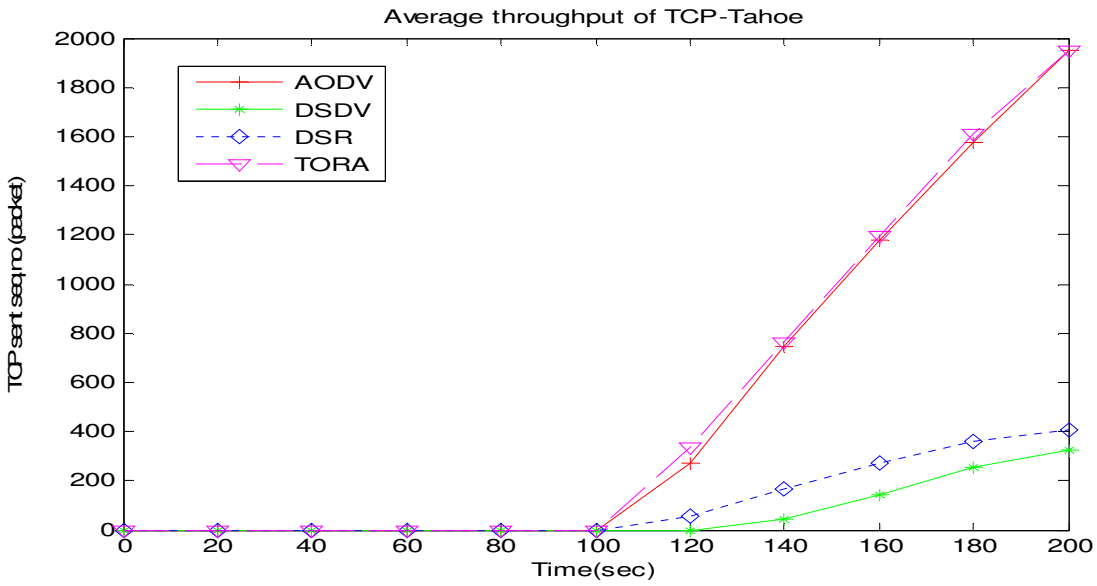
(a) speed=2m/s



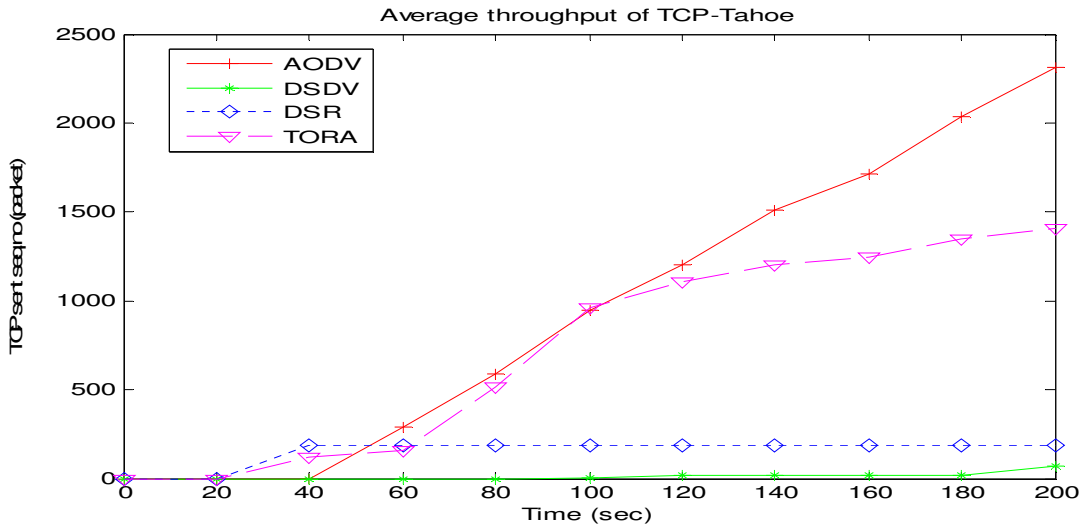
(b) speed=15m/s

Figure 5:3 Average throughput of TCP-Sack for speed 2m/s and 15m/s

Fig5.4 a and b show the average throughput of TCP-Tahoe over AODV, DSDV, DSR, and TORA routing protocols for nodes speed 2 and 15 m/s. Here again, DSDV and DSR have lower throughput than AODV and TORA for both speeds 2 and 15m/s. For speed of nodes with 2m/s, TORA and AODV performed almost the same result.



(a) speed=2m/s



(b) speed=15m/s

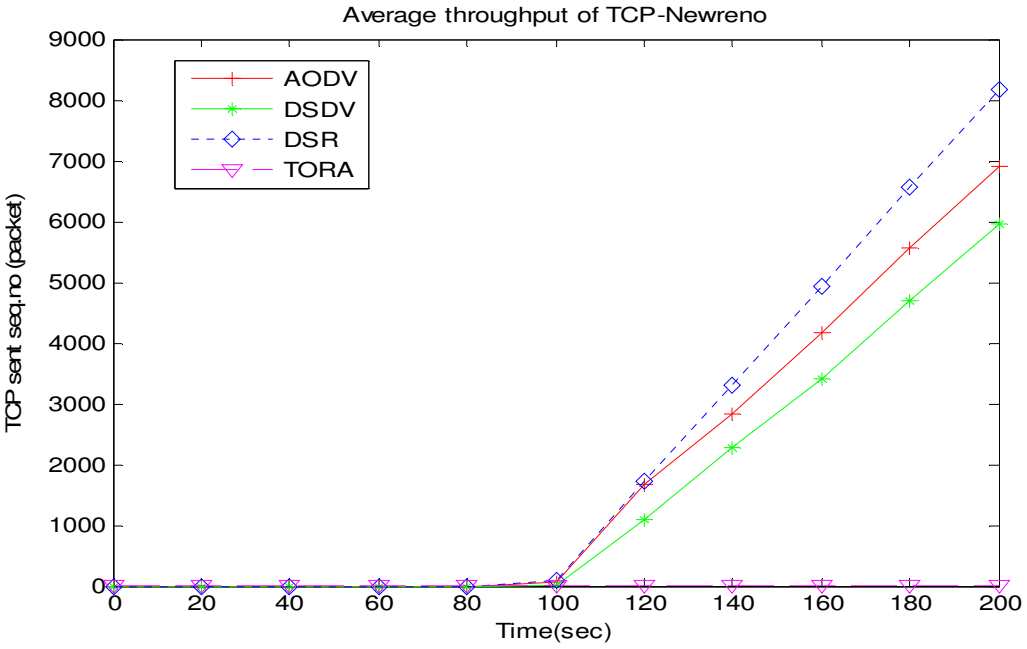
Figure 5:4 Average throughput of TCP-Tahoe for speed 2m/s and 15m/s

Generally, for all variants of TCP; TCP-Newreno, TCP-Tahoe, TCP-Reno, and TCP-Sack, AODV achieved the highest throughput than the other three routing protocols; DSR, DSDV, and TORA, for different mobility pattern. The least throughput was achieved by DSDV for all TCP variants. DSDV has a negative impact on the performance of TCP variants, which has frequent route failure and changes due to the dynamic movement of mobile nodes. Whenever route fails, DSDV should maintain the whole information about the routes before sending data packets that has large delay for TCP to send packets, which degrades the performance of TCP variants. This theoretical analysis is clearly seen in the simulation results. From the above simulation results, for different mobility patterns, reactive routing protocols performed better than proactive routing protocols for all TCP variants.

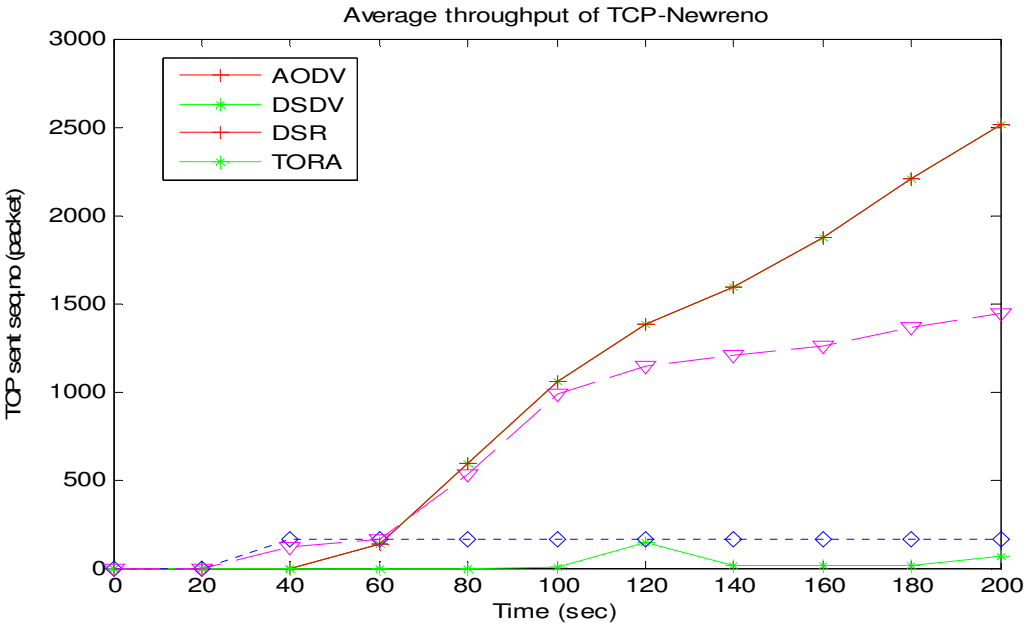
B. For different node density

Once again, TCP variants were evaluated and compared over the stated routing protocols for two node scenarios; 5 nodes and 20 nodes.

Fig.5.5 shows the simulation results of TCP-Newreno over four routing protocols. For small number of nodes (in this case 5 nodes), DSR achieved a higher throughput than AODV, DSDV, and TORA. The least throughput was achieved by TORA. For higher number of nodes (20 nodes), AODV and TORA performed better than DSR and DSDV. In both simulation scenarios, AODV outperforms the other routing protocols. From the simulation results, it can be concluded that TORA is not suitable for a network with small number of nodes (lower node densities). However, as the number of nodes grows, it performs well with availability of multiple routes.



(a) 5 nodes

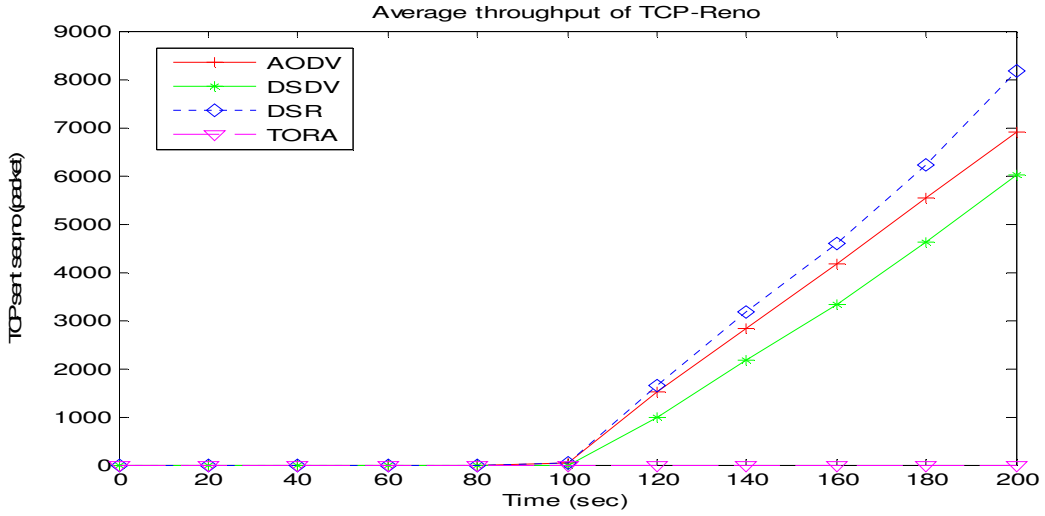


(b) 20 nodes

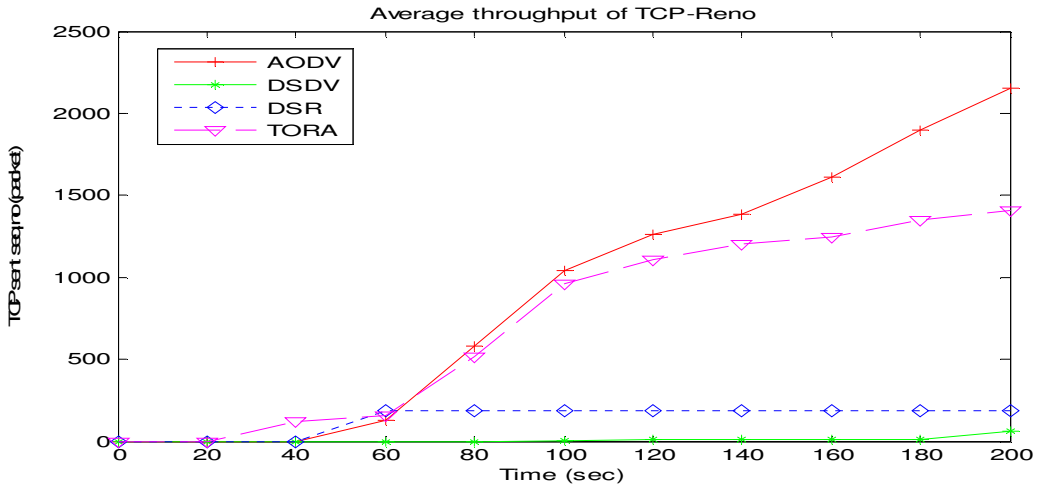
Figure 5:5 Average throughput of TCP-Newreno for 5 and 20 nodes

Fig.5.6 shows the simulation results of TCP-Reno for 5 and 20 nodes, respectively. The graphs show that TCP-Reno and Newreno performed almost the same throughput over the stated routing protocols. TORA

achieved the least throughput, whereas, DSR performed the highest throughput for small number of nodes. As the number of nodes increases, DSR achieved lower throughput. This is because, as the number of nodes increases, network topology can change unpredictably due to the dynamic movement of nodes. In this case, the cache of DSR may contain stale or old routes that resulted in packet losses. Consequently, TCP's performance can be reduced by expiring RTO and unnecessary retransmission of packets. AODV had the highest throughput for both node densities.



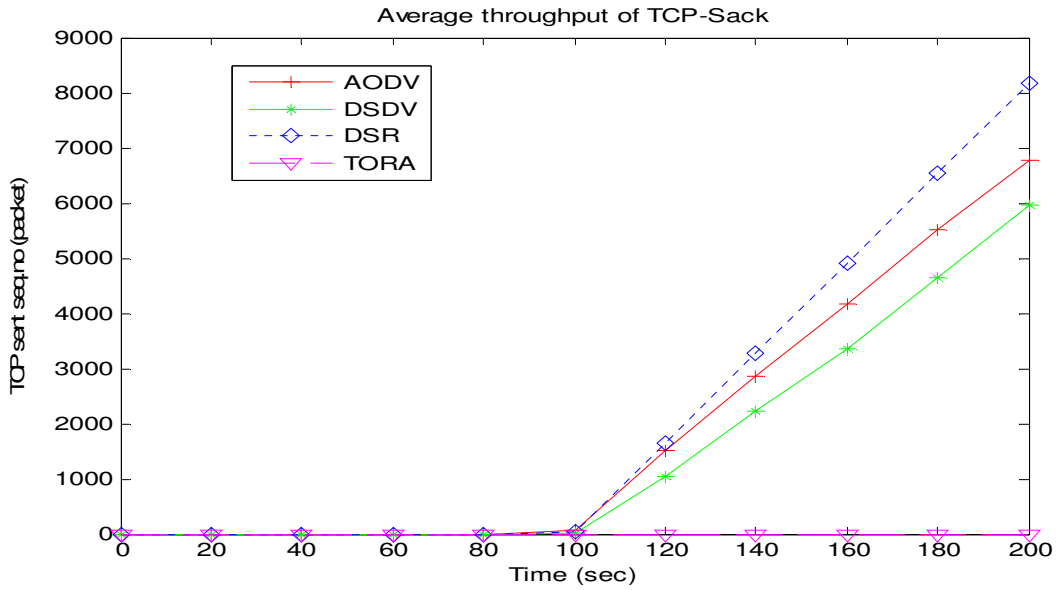
(a) 5 nodes



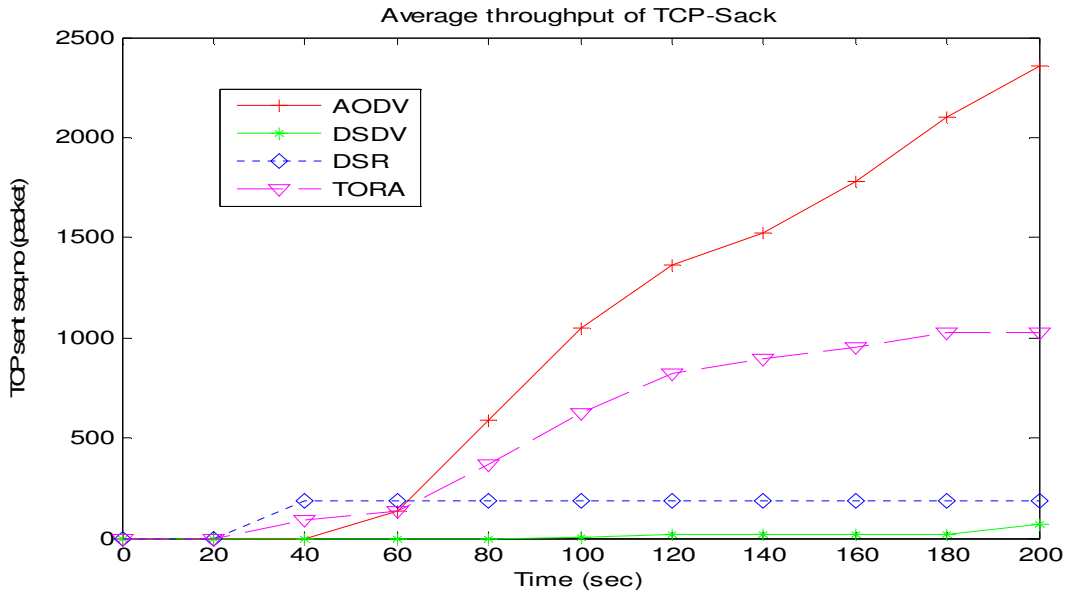
(b) 20 nodes

Figure 5:6 Average throughput of TCP-Reno for 5 and 20 nodes

Fig.5.7 and 5.8 show the simulation results of TCP-Sack and TCP-Tahoe for 5 and 20 nodes respectively, over four routing protocols. For lower number of nodes, DSR and AODV achieved the highest throughput. For large number of nodes, AODV performs better than the other three routing protocols.



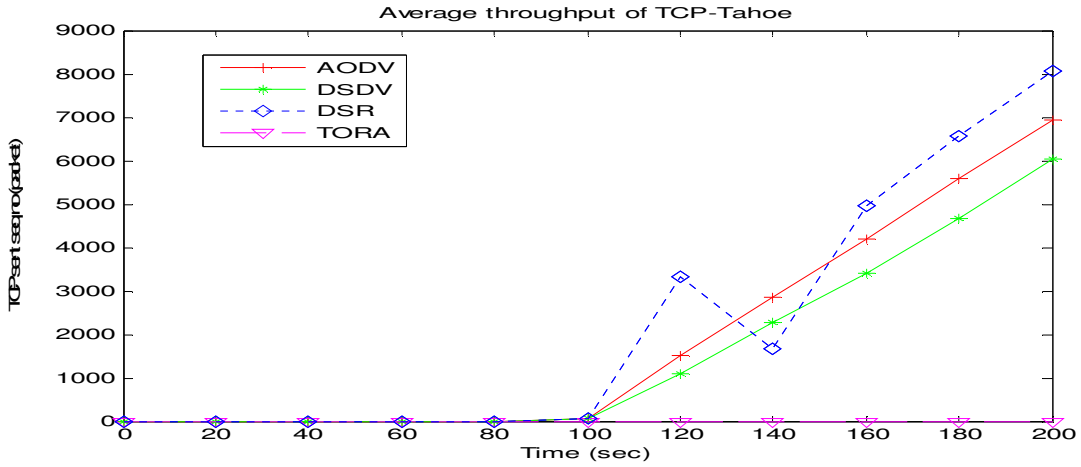
a) 5 nodes



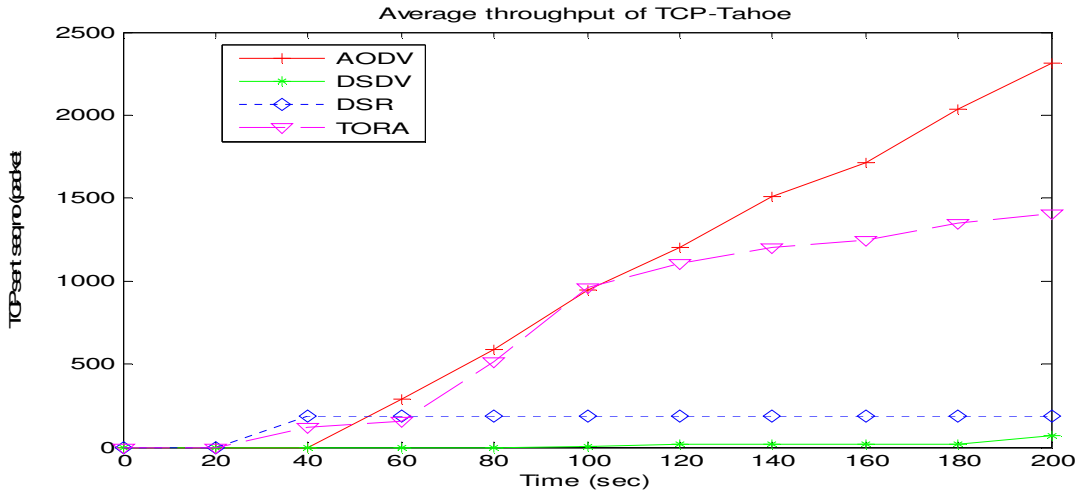
b) 20 nodes

Figure 5:7 Average throughput of TCP-Sack for 5 and 20 nodes

Figure 5.8 shows the simulation results of TCP-Tahoe for 5 and 20 nodes respectively. For lower number of nodes, DSR had a higher throughput. However as the number of nodes increases, DSR should store and maintain large route information in cache and when the source wants to send data, it should send the whole route in the packet header (source routing), which degrades TCP's throughput. AODV performed good result both in large and small number of nodes. Moreover, DSDV is more suitable for small networks where there is no frequent updating of network topology. TORA is suitable for larger network with highly dynamic network topology.



a) 5 nodes



b) 20 nodes

Figure 5:8 Average throughput of TCP-Tahoe for 5 and 20 nodes.

Generally, from the above simulation results, with respect to routing protocols, it can be concluded that reactive routing protocols achieved a higher throughput than proactive routing protocols. For both simulation scenarios:

a- *For different mobility pattern* - speed of nodes 2m/s and 15m/s and
b- *For different node density* - number of nodes 5 and 20.

DSDV, which is proactive routing protocol, achieved the least throughput. Whereas, AODV, DSR, and TORA, which are reactive routing protocols achieved a higher throughput. Here are some rationales behind these results: Proactive routing protocols require periodic updating of route information to maintain routes up-to-date and consistent, which results in exchanging of control message. In addition, nodes maintain a lot of routes in their routing tables, which might not be used, which adds extra routing overhead that consumes scarce bandwidth and limited supply of power in ad hoc networks. Due to the above reasons, proactive routing protocols degrade the performance of TCP variants. In reactive routing protocols, since routes are searched only on-demand, they minimize control overhead and power consumption that leads to better efficiency of TCP variants.

5.5 Observation and discussion of TCP variants over routing protocols

In this section, performance of TCP variants over routing protocols shall be discussed. The simulation scenario comprises of different mobility pattern and node densities. From TCP variants, TCP-Tahoe is taken as a reference for comparison purpose. As it is mentioned in the TCP variants

section, it is the base TCP. The other three TCP variants are modified from Tahoe.

5.5.1 Performance of TCP variants over AODV

This section discusses a detail simulation results of performance of TCP variants over AODV routing protocols by making use of different performance parameters like; throughput, *cwnd*, RTO, and duplicate acknowledgments.

Fig 5.9 shows the average throughput of TCP variants over AODV routing protocol under different network scenarios. For low mobility pattern (speed 1-3m/s), TCP-Newreno achieved about 6% better than Tahoe. Whereas, it shows 10% and 9% improvement over TCP-Reno and TCP-Sack, respectively.

For higher mobility of nodes (up to 15m/s), the performance variation of Newreno is 9% better than Tahoe, and 16% and 7% better than Reno and Sack, respectively. This simulation result confirmed that under different mobility pattern of mobile nodes, TCP-Newreno and TCP-Sack outperformed the other variants. Whereas, TCP-Reno and TCP-Tahoe are significantly the worst performer. This is because; as the speed of nodes increases due to high mobility of nodes, route failure occurs frequently. In this case multiple packets can get lost from one window of data. Since TCP-Reno and Tahoe can't handle multiple packet loss from one window of data, their performance degrades.

However, Compared to TCP-Reno, TCP-Tahoe performed good result. In case of TCP-Tahoe, when packet loss occurs, it retransmits the lost packet and automatically enters into slow start phase to increase *cwnd* exponentially till it reaches to *ssthresh*, which uses the bandwidth

efficiently. However in case of TCP-Reno, for multiple packet loss it should wait for the expiration of RTO by entering and leaving fast recovery algorithm to retransmit the lost packet as well as to enter into slow start phase.

Table5:2

Sent sequence no. of TCP variants over routing protocols

	Newreno	**Reno**	**Sack**	**Tahoe**
AODV	3504	3269	3350	3376
DSDV	1614	1630	1628	1625
DSR	2250	2238	2237	2212
TORA	1197	1183	1004	1192

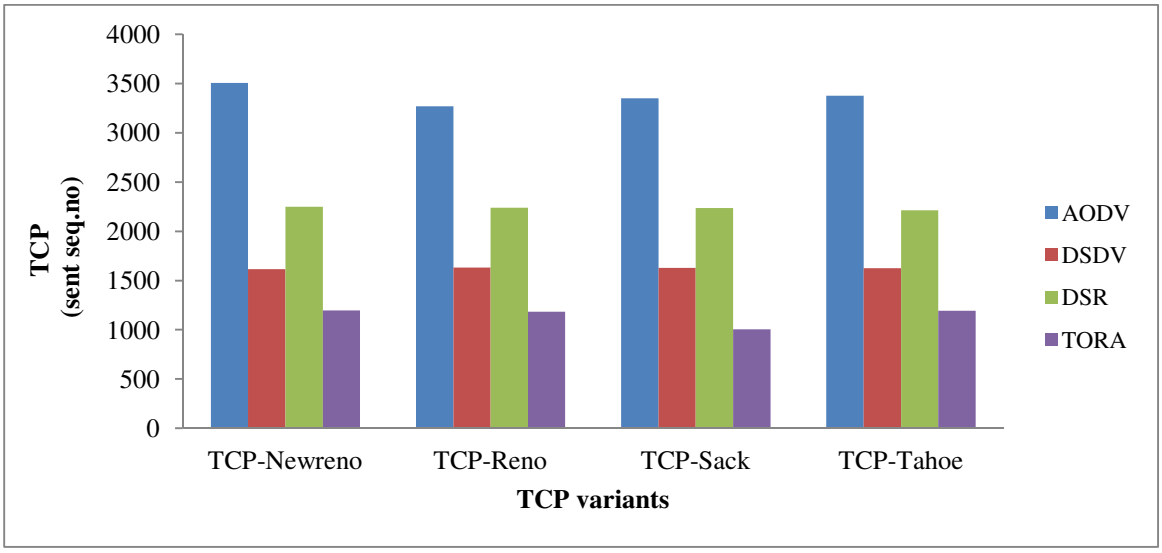


Figure 5:9 Average throughput of TCP variants over AODV

The throughput results of TCP variants for different node densities are also shown in fig.5.9. For small number of nodes (5 nodes), TCP-Tahoe reported better performance than the other variants. The percentage variation is 0.2% increment over TCP-Newreno and 0.1% and 2% over TCP-Reno and TCP-Sack, respectively. For large number of nodes (20 nodes), TCP-Newreno showed significant performance achievement. The variations are 9% over Tahoe and 16% and 7% over Reno and Sack, respectively. Hence, once again TCP-Reno is the least performer.

Fig.5.10 shows the average *cwnd* of all TCP variants. TCP-Newreno has a larger *cwnd* than the others, then Sack, Reno, and Tahoe, respectively. Larger *cwnd* means better throughput as they are directly related with each other.

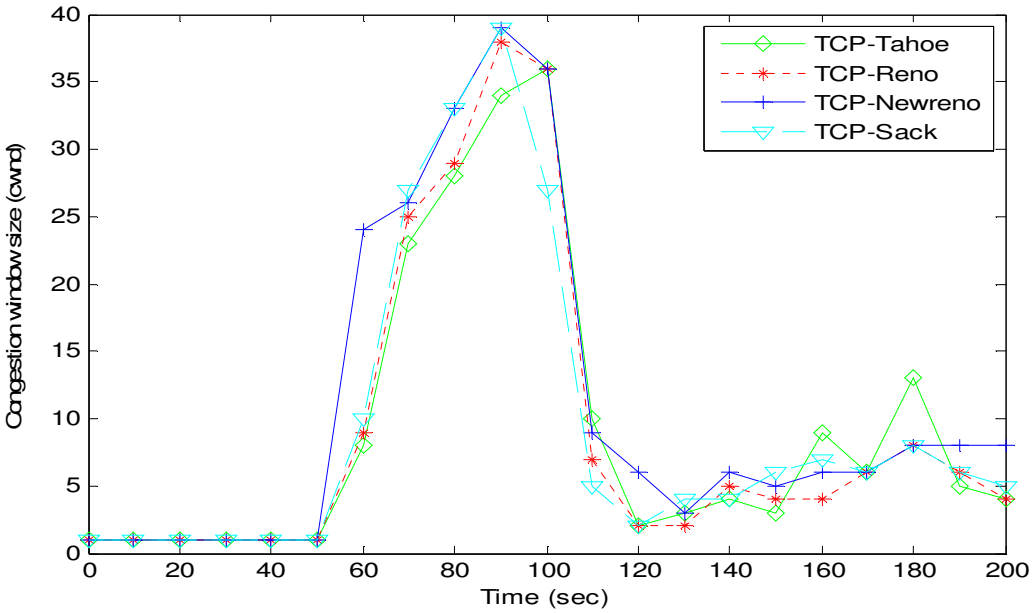
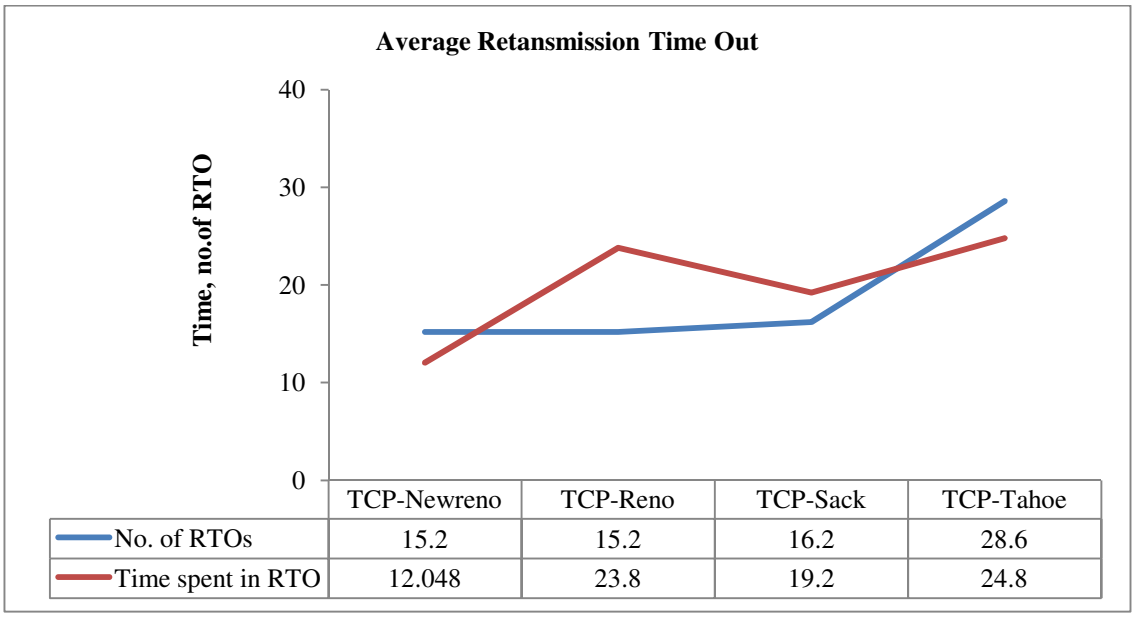


Figure 5:10 Average congestion window size of TCP variants over AODV

Fig.5.11 shows the number of RTO made and time spent in RTO of all TCP variants. If RTO increases, *cwnd* is not increased. However, if RTO decreases or remains constant, *cwnd* decreases, this in turn affects the throughput of TCP variants. Newreno and Sack spent little time in RTO. Whereas, on average Tahoe and Reno spent more time in RTO. Since Newreno and Sack have mechanism to recover in case of multiple packet losses from the same congestion window, they avoid multiple consecutive time outs than Reno and Tahoe. In addition, Reno and Tahoe do not have mechanism to estimate outstanding packets (packets that are sent but not yet acknowledged). Therefore, they retransmit packets which are sent but not yet acknowledged after the expiration of RTO, which results in an increase in number of triggering RTOs that leads to lower throughput.

Fig.5.12 shows time spent in fast retransmission/recovery and the number of occurrences of three duplicated acknowledgments of the stated TCP variants. Since Tahoe doesn't implement fast recovery algorithm, its value remains zero. TCP-Sack experienced a higher value of duplicate acknowledgments and more time in fast retransmit/recovery phases than the other variants. This is due to the need of extra place to store the sack block information. However, it is not the worst performer due to better retransmission and recovery techniques implemented to it.



	TCP-Newreno	TCP-Reno	TCP-Sack	TCP-Tahoe
No. of RTOs	15.2	15.2	16.2	28.6
Time spent in RTO	12.048	23.8	19.2	24.8

Figure 5:11 Average retransmission time out of TCP variants over AODV

When DSDV is used under TCP variants, TCP-Reno performed better than the other variants. Over DSR routing protocol, TCP-Newreno achieved the highest throughput. Over multi-path routing protocol TORA, once again TCP-Newreno had a higher throughput than the other TCP variants.

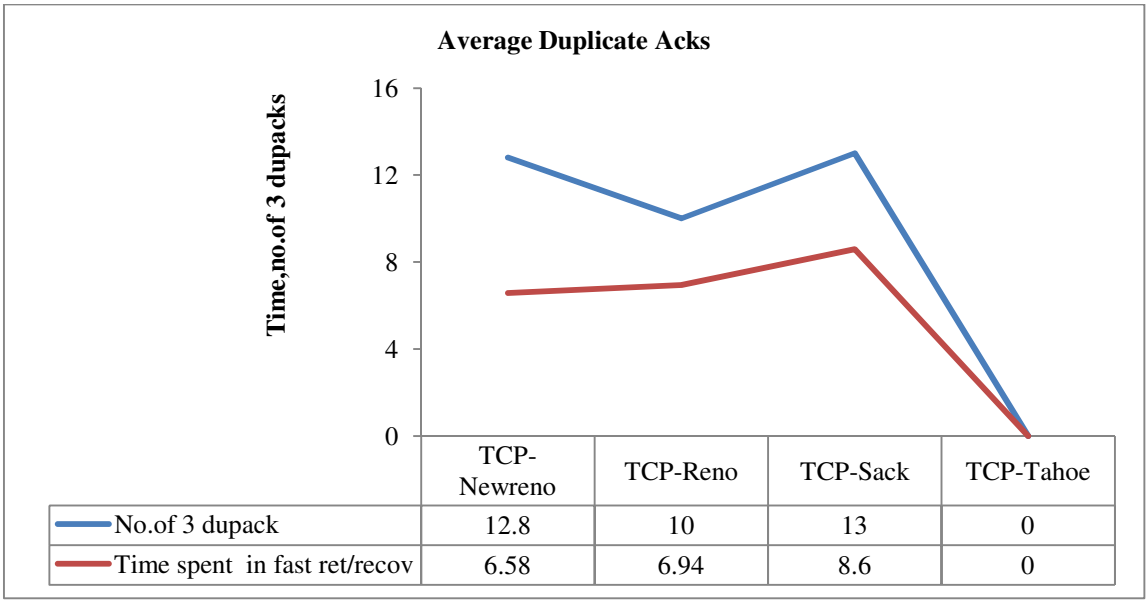


Figure 5:12 Average fast retransmission /recovery calls made by TCP variants over AODV

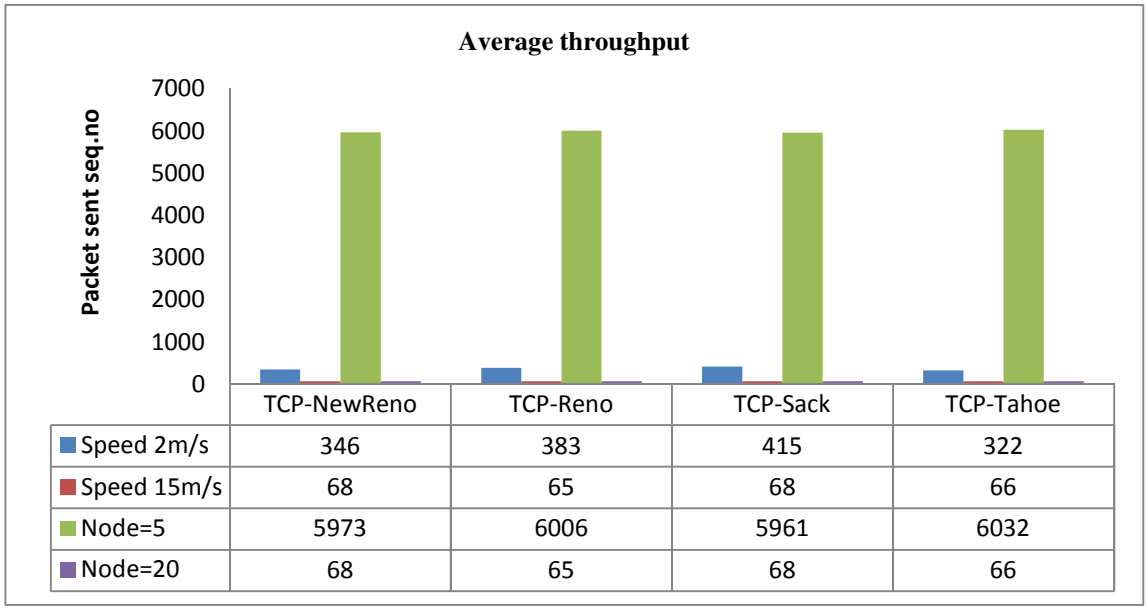
Generally, as the speed of nodes increases, route failure occurred frequently due to high mobility of nodes. In this case, all TCP variants assumed that packet loss due to link/route failure as a sign of congestion. Consequently, they reduced *cwnd* and retransmits packets unnecessarily, which degraded their throughput significantly. It is also confirmed that, as the number of nodes increase, all TCP variants showed worst throughput over AODV, DSDV, and DSR routing protocols.

On the average, TORA achieved the least throughput for all TCP variants. This is due to the fact that, in MANET multi-path routing protocols like TORA affects TCP by two factors; the first one is inaccuracy of average RTT measurement, which may lead to premature time out than the standard one. The second is out-of-order packet delivery through different paths at the TCP receiver side, which will be forced to send three duplicate acknowledgments. Up on the reception of these three duplicate acknowledgments; TCP sender assumes that out-of-order delivery of packets as a sign of network congestion (packet loss). As a result, TCP

variants (sender side) retransmit packets and reduce *cwnd* unnecessarily, which reduces the overall throughput of TCP variants.

5.5.2 Performance of TCP variants over DSDV

Fig.5.13 shows the simulation results of TCP variants over DSDV routing protocols under different network scenarios. For lower network mobility (1-3m/s), TCP-Sack showed performance advantage of 29%/22%/10% over Tahoe/Newreno/Reno, respectively. For higher mobility of nodes (up to 15m/s), TCP-Sack and Newreno achieved 3% performance improvement over TCP-Tahoe, and 5% improvement over Reno.



	TCP-NewReno	TCP-Reno	TCP-Sack	TCP-Tahoe
Speed 2m/s	346	383	415	322
Speed 15m/s	68	65	68	66
Node=5	5973	6006	5961	6032
Node=20	68	65	68	66

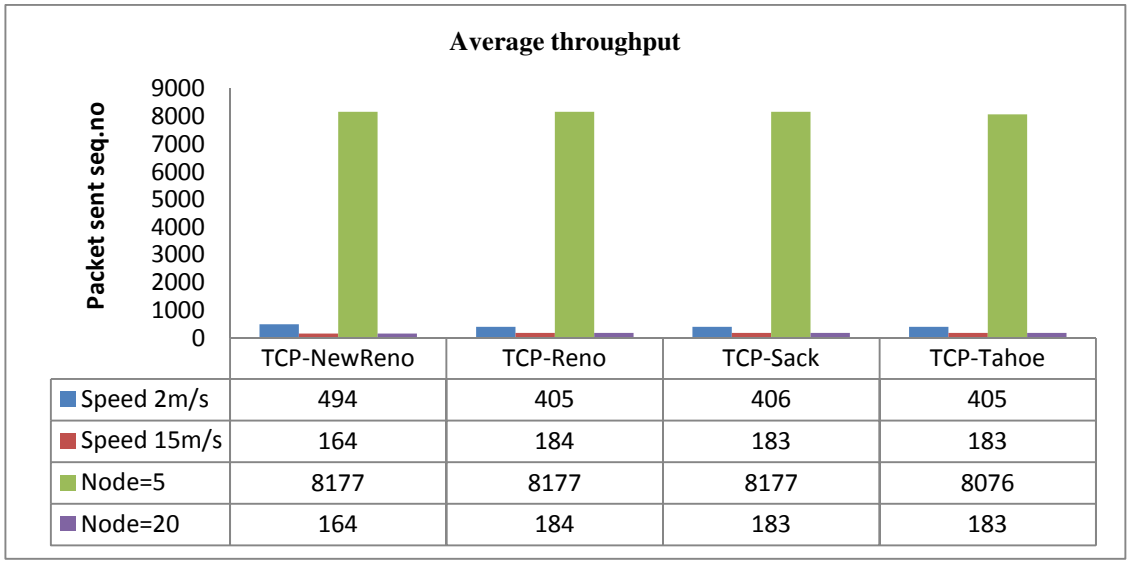
Figure 5:13 Average throughputs of TCP variants over DSDV

For lower network (only 5 nodes), all variants of TCP achieved a higher throughput with almost the same percentage value. It is obvious that for small number of nodes, the underlying routing protocols (in this case DSDV) can simply find routes quickly even in the presence of route failure. As the number of nodes increases, the throughput degrades considerably.

TCP-Newreno and Sack have the same performance improvement 3% over Tahoe and 5% over Reno.

5.5.3 Performance of TCP variants over DSR

The simulation results depicted in fig.5.14 shows the throughput measurement of TCP variants over DSR routing protocol. All the TCP variants except TCP-Newreno performed the same result. Newreno showed a significant throughput improvement, which is 22% over Tahoe for lower node mobility, whereas, it showed 10%/10% performance degradation over Tahoe for high mobility of nodes and large number of nodes, respectively. The other way round, TCP-Tahoe, Sack, and Reno have no significant difference for all simulation scenarios. DSR is more suitable for small network.



	TCP-NewReno	TCP-Reno	TCP-Sack	TCP-Tahoe
Speed 2m/s	494	405	406	405
Speed 15m/s	164	184	183	183
Node=5	8177	8177	8177	8076
Node=20	164	184	183	183

Figure 5:14 Average throughputs of TCP variants over DSR

5.5.4 Performance of TCP variants over TORA

Fig.5.15 shows simulation results of TCP variants over multi-path routing protocol TORA. For lower network speeds, the performance variation of TCP-Tahoe over Newreno and Reno is 3% and 2% respectively. However, as the speed of nodes increases from 2m/s to 15m/s, the probability of route failure is more due to high mobility of nodes, which degrades the performance of all TCP variants.

For small number of nodes, all TCP variants achieved the least throughput, which is given by 0 packet sent. However, as the number of nodes increases (from 5 to 20 nodes), all TCP variants achieved better throughput over TORA. As shown in figure 5.15 and 5.16, on the average, TORA performed the least throughput in all TCP variants. This is due to the fact that, in MANET multi-path routing protocols like TORA affects TCP with out-of-order packet delivery through different paths at the TCP receiver side. In this case, TCP receiver assumes that out-of-order delivery of packets as a sign of packet loss. It then sends three duplicate acknowledgments.

Up on the reception of these three duplicate acknowledgments, TCP variants (sender side) retransmits packets and reduces *cwnd* unnecessarily, which reduces the overall throughput of TCP. In particular, TORA is suitable for highly dynamic network topology and high node densities due to availability of multi-path routes from source to destination.

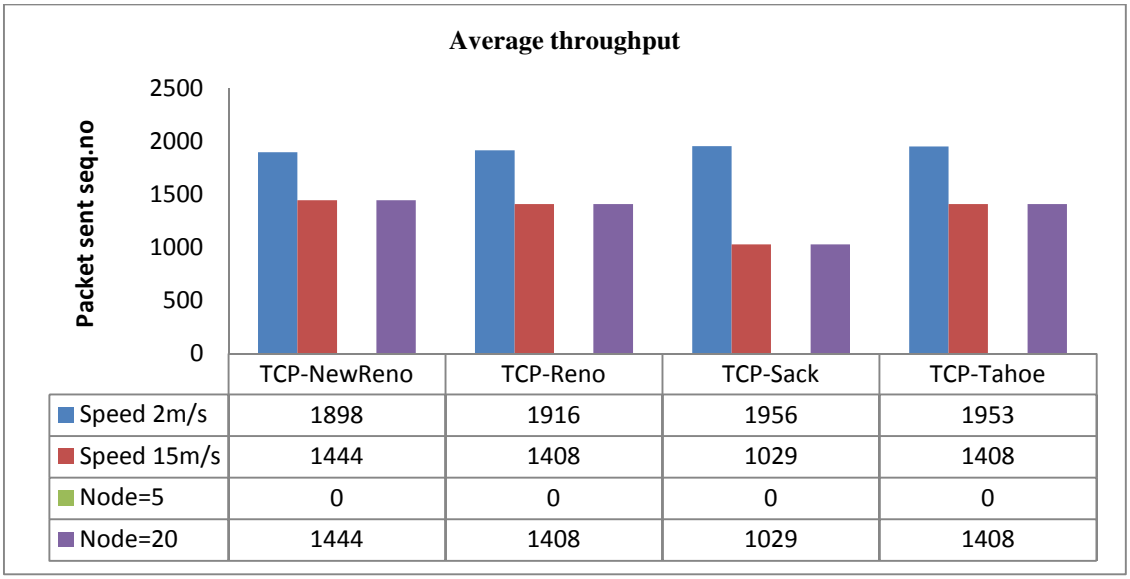


Figure 5:15 Average throughputs of TCP variants over TORA

Generally, for all simulation scenarios; for different mobility pattern and node density, the highest throughput was achieved by all TCP variants over AODV. Table 5.3 and fig.5.16 summarizes the average throughput achieved by all TCP variants over four routing protocols. TCP-Newreno achieved better performance over AODV. When DSDV is used under TCP variants, TCP-Reno performed better than the other variants. Over DSR routing protocol, TCP-Newreno achieved the highest throughput. Over multi-path routing protocol TORA, once again TCP-Newreno has a higher throughput than the other TCP variants.

Table 5:3

Packet sent sequence number of TCP variants over routing protocols

	TCP-Newreno	**TCP-Reno**	**TCP-Sack**	**TCP-Tahoe**
AODV	3504	3269	3350	3336
DSDV	1614	1630	1628	1625
DSR	2250	2238	2237	2212
TORA	1197	1183	1004	1192

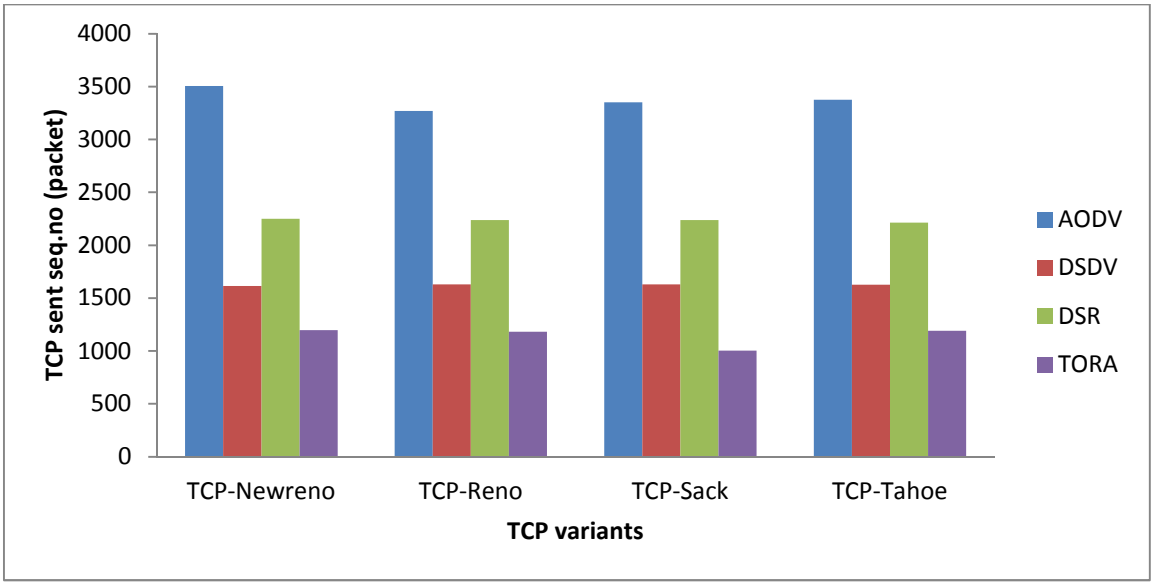


Figure 5:16 Average throughputs of TCP variants over four routing protocols

As shown in fig.5.16, TCP-Newreno achieved the best performance over all routing protocols compared to the other variants. This is because; TCP-Newreno shouldn't wait for the retransmit timer to expire when multiple data packets lost from one window of data. Therefore, TCP-Newreno and AODV are the best performing combination of TCP variants and routing protocols in terms of the stated performance metrics and for different MANET configuration scenarios; mobility patterns and different mobility of nodes.

5.6 Performance of TCP-PLDR over routing protocols

Performance of TCP-PLDR was simulated and evaluated for different routing protocols. Simulation scenarios were prepared in such a way. Routing protocols used were AODV, DSDV, DSR, and TORA. TCP agent used was TCP-PLDR. Speed of nodes was configured from 10m/s to 15m/s. simulation time was set to 200 seconds. The simulation was run for ten times by changing the position and speed of mobile nodes and

average was taken to make the simulation fair and acceptable. Performance metrics used were throughput, end-to-end delay, and PDR.

As shown in fig.5.17, the throughput of TCP-PLDR varies significantly depending on the routing protocol used. It is clearly shown that AODV achieved the highest throughput, which is on the average its sending rate was found to be about 107 kbps. Whereas, proactive routing protocol DSDV showed the least performer (3kbps). On the other hand, DSR and TORA showed almost the same throughput, which are 54 and 53 kbps, respectively. As stated in section 5.4, DSR is suitable for lower speed of nodes, whereas, TORA is suitable for highly dynamic network topology. Consequently, since the average speed of nodes is ranging between 10 and 15m/s, they showed the same results. As a result, TCP-PLDR can be coupled with AODV. Moreover, TCP-PLDR is suitable for reactive routing protocol than proactive in MANET. This result was also found in section 5 for other TCP variants so that TCP-PLDR is a TCP friendly protocol.

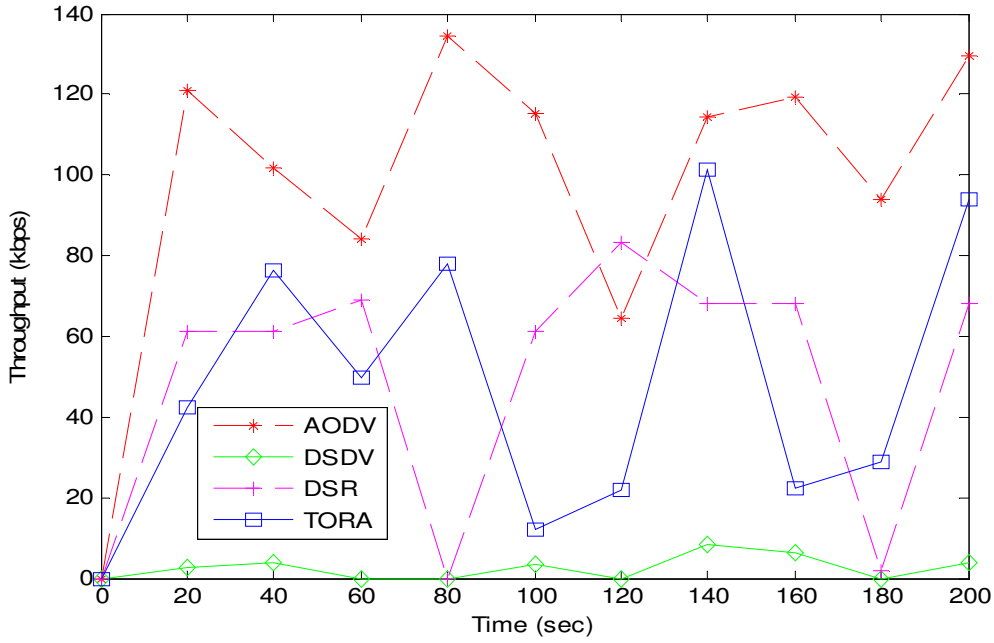


Figure 5:17 Average throughputs of TCP-PLDR over routing protocols

Fig. 5.18 shows the average end-to-end delay of TCP-PLDR over one proactive and three reactive routing protocols. When TCP-PLDR is coupled with AODV, its average end-to-end delay was found to be 310 msec. Whereas, when TCP-PLDR runs over DSR and TORA, delays are measured as 383 and 411 msec, respectively. Finally, TCP-PLDR and DSDV on the average shows 158msec. From these results, DSDV is the least performer. This is because, in some simulation scenarios, it sends a small number of packets, which can minimize the delays. Whereas TORA is the highest performer.

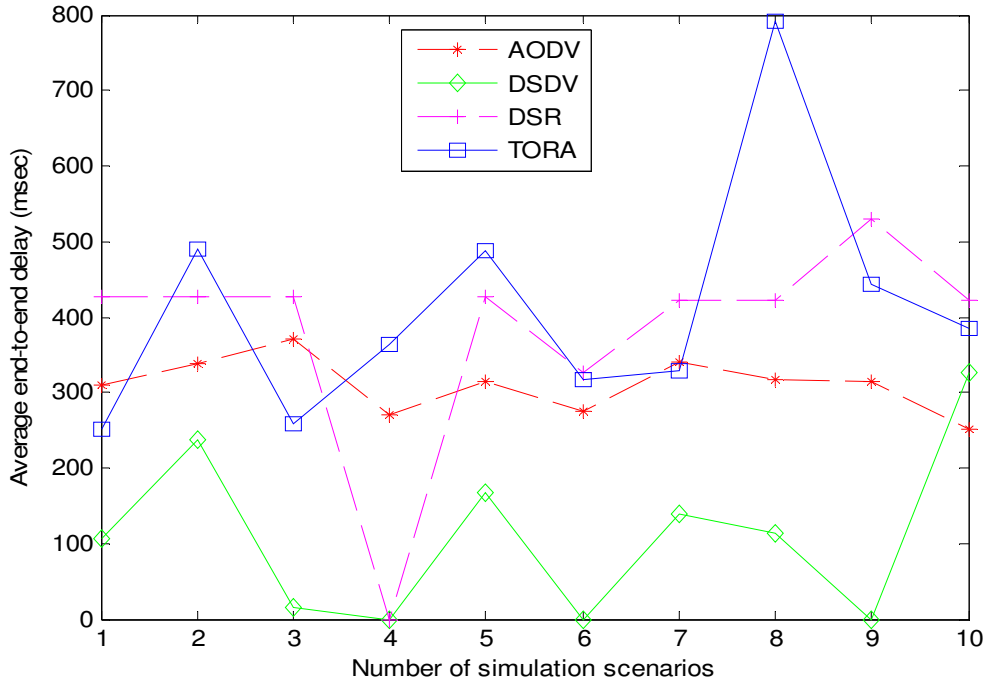


Figure 5:18 Average delays of TCP-PLDR over routing protocols

Fig. 5.19 illustrates the average PDR value achieved by TCP-PLDR over routing protocols. In all simulation scenarios, AODV and TORA showed a consistent PDR values. Whereas, PDR of DSR and most importantly DSDV fluctuates between the maximum and minimum values. On the average, TCP-PLDR achieved a PDR value of 96.4% over AODV, 95% over TORA, 86% over DSR, and 58% over DSDV routing protocols. Generally, from the simulation

results, TCP-PLDR shows the best performance in terms of throughput and data delivery ratio even end-to-end delay over AODV. The next best performer is TORA followed by DSR and finally DSDV. TCP-PLDR is therefore better suited for reactive routing protocols.

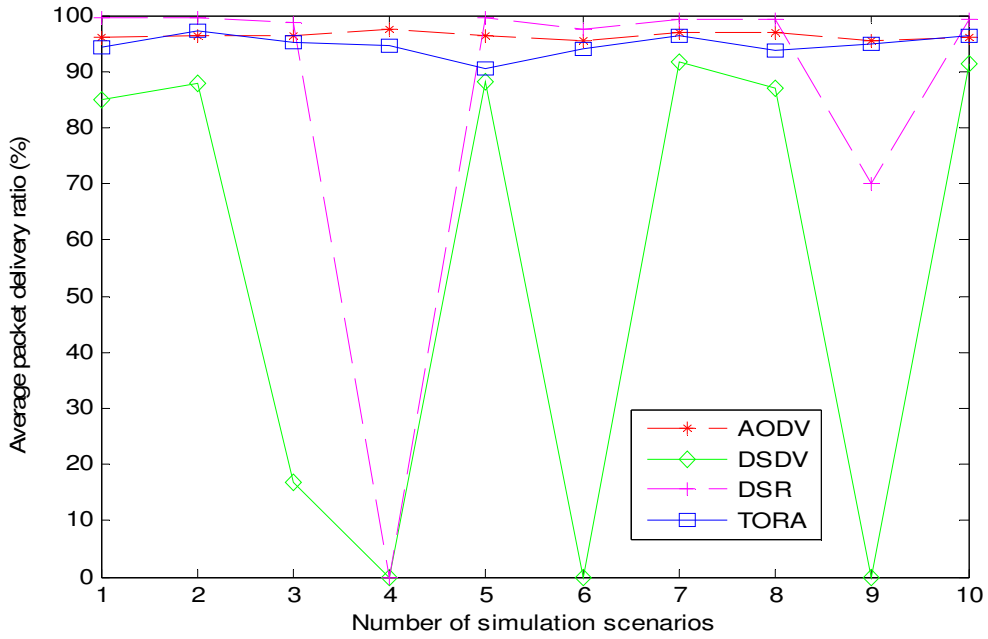


Figure 5:19 Average delays of TCP-PLDR over routing protocols

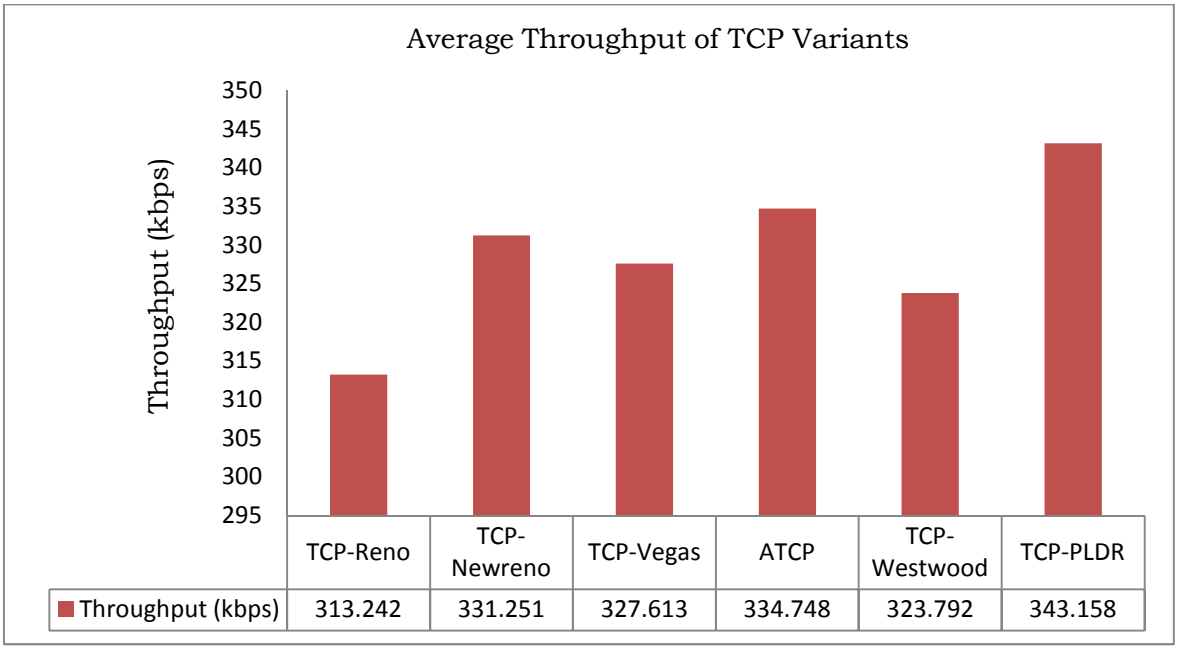
5.7 Performance of TCP-PLDR with other TCP variants

Extensive simulation was conducted to evaluate the performance of TCP-PLDR with TCP-Reno, TCP-Newreno, TCP-Vegas, TCP-Westwood [86], and ATCP. Simulation scenario for this experiment is the same as mentioned in section 4.8.A.2.

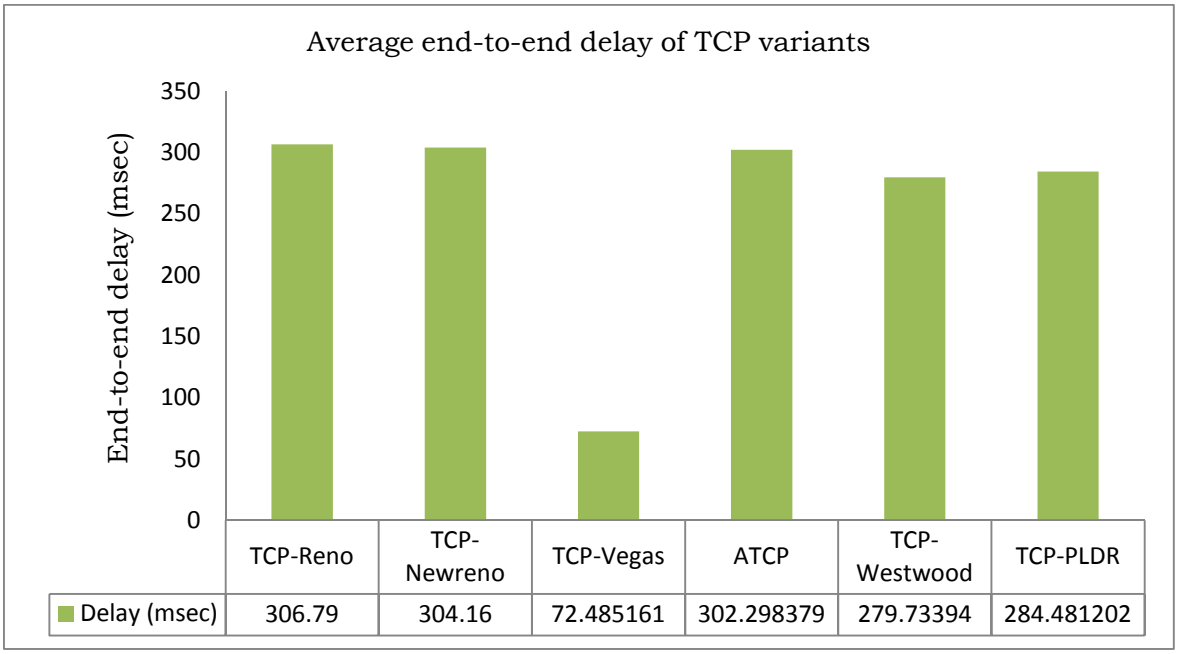
Fig.5.20 shows average throughput, delay, and PDR results of TCP-PLDR, Vegas, Westwood, Reno, Newreno, and ATCP variants. As shown in fig.5.20a, the average throughput of TCP-PLDR improved significantly compared to other TCP variants. Reno and Westwood are the least performers. Since they

assume that packet loss is an indication of network congestion, they falsely retransmit the lost packet and reduce the packet sending rate, which degrades their performance significantly. More specifically, while packets get dropped due to route failures or delivered out-of-order due to route change, they keep retransmitting the packets and call exponential back-off algorithm to double the RTO value. Consequently, TCP sender will remain idle for a long time without sending new packet thereby reduce their throughput. TCP-Newreno and ATCP showed almost comparable throughput. TCP-PLDR outperformed the other variants.

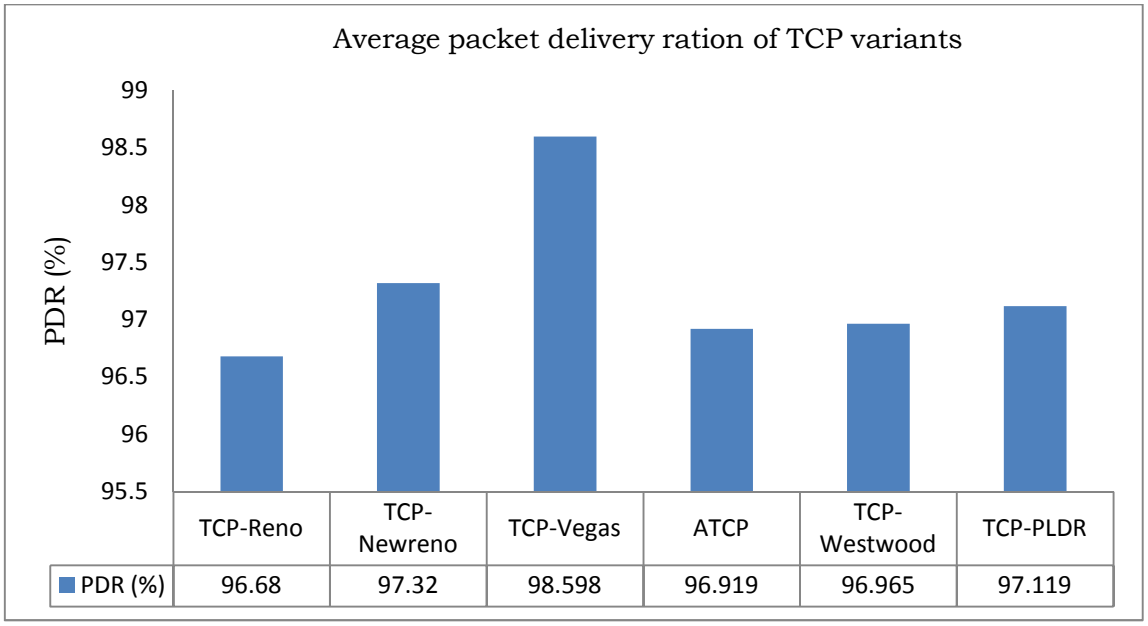
As shown in fig5.20b, Vegas achieved the least delay compared to other variants in effect. Reno and Newreno showed the same delay. ATCP is the worst performer. TCP-Vegas adjust its sending rate before packet loss due to congestion happens. This behavior of Vegas results in correct sending rate without depending on packet loss indication, which results in lower delay. Moreover, TCP-Vegas uses RTT to estimate delay to adjust *cwnd*, which is an important parameter to find an accurate estimation in MANET. However, ATCP suffered to frequent retransmission and exhaustive processing of retransmitted packets at the destination node due to route failure. On the other hand, TCP-PLDR and Westwood (as they are extension of SACK and Newreno, respectively) showed almost the same delay value. In terms of PDR value shown in fig.5.20.c, Vegas, Newreno, and PLDR outperformed the other variants. Reno, ATCP, and Westwood are the least performers, respectively.



a) Throughput (kbps)



b) End-to-end delay (msec)



c) Packet delivery ratio (%)

Figure 5:20 Performance of TCP-PLDR with other TCP variants

5.8 Summary

This chapter has presented the effects of routing protocols (proactive and reactive) on the performance different TCP variants, under different mobility pattern and node density. Performance analysis was performed by making use of four different simulation scenarios. 1- Performance of routing protocols on TCP variants, 2- Observation and discussion of TCP variants over routing protocols, 3- performance of TCP-PLDR over different routing protocols, and 4- performance of TCP-PLDR with other TCP variants.

A comprehensive simulation studies have demonstrated the complex interaction of TCP variants with routing protocols under the conditions of frequent route failure in MANET. It is also confirmed that the performance of TCP variants are highly dependent on the underlying routing protocols. Moreover, the best performing combination of routing protocols and TCP

variants under different simulation scenarios (mobility pattern and node density) were selected.

The above results revealed that every protocol performed differently in different MANET scenarios. Therefore, using different combination of TCP variants and routing protocols leads to optimum performance than a single one. Moreover, TCP is unable to distinguish between packet losses due to route failure and those due to congestion. Hence, routing protocols have their own effect on the performance of TCP variants. This result was taken as a valuable input to design and study the first part of this dissertation research.

CHAPTER SIX

INVESTIGATING THE EFFECTS OF SECURITY ATTACKS ON THE PERFORMANCE OF TCP VARIANTS AND ROUTING PROTOCOLS IN MANET

6.1 Introduction to security in MANET

As stated in the introduction section, MANET is a collection of wireless mobile nodes that form a temporary network without any centralized administration or infrastructure. Such network mainly consists of mobile nodes and can be connected dynamically in an arbitrary manner. The dynamic nature of the network and lack of well secured boundaries (as every node is free to move around) make MANET very much vulnerable to different types of security attacks [80,81]. The attacks include eavesdropping, impersonation, tempering, replay, and denial of service (DoS).

Specifically, in MANET as no default router is available, routing misbehavior that leads to DoS is the main target for security attack. In this type of attack, an adversary first shows itself as an honest node during the route discovery process, and then silently drops some or all of the packets sent to it. Consequently, since TCP has been designed to perform well in wired networks under the assumption that all types of packet drops are an indication of congestion, it falsely retransmits the lost packet and reduces its sending rate, which degrades its performance a lot. The malicious node could also disturb normal route discovery process of MANET routing protocols. Generally, all attacks against the routing process and data forwarding affect overall MANET availability [82,83,84,85].

In this chapter, we have investigated the effect of malicious packet drop attack on the performance of routing protocols and TCP variants in MANET. Moreover, the performance of TCP-PLDR and TCP-Sack was compared in the presence of malicious packet drop attack by adding different percentage (ranges from 0% to 50%) of malicious nodes in the network.

6.2 Routing misbehaviour and attack

MANETs are very much vulnerable to different types security attacks than the wired networks, which are assumed to be trustworthy [80,82,83]. The existing MANET routing protocols do not equipped well with any security mechanisms. In MANET, malicious node can make routing services a target because it is an important service. For MANET routing protocols to operate properly, nodes should trust with each other to establish a routing path. However, if a malicious node put itself in the routing path with the assumption that nodes are trustworthy and cooperative, then normal routing protocols operation is dependent on the intention of malicious node. There are two types of routing attacks.

Routing misbehavior - in which the nodes are unable to behave and perform in accordance with a routing protocol.

Packet forwarding misbehavior – in which the nodes fail to forward data packets in accordance with a data packet transfer protocol. This kind of attacks on packet forwarding/delivery attack is not easy to detect and prevent as there is no detection mechanism in protocol stack (link layer, network layer, and transport layer) [80,81]. Neither end-to-end nodes nor intermediate nodes have knowledge about where and why packets are being dropped. There are two main attack mechanisms in this type. One is *selfishness*, in which the malicious node selectively drops packets that are supposed to be transmitted in order to save its own battery life, CPU, and

memory usages. The other is *DoS* known as *packet drop attack*, which is mainly concerned with dropping packets to make the network congested there by affecting the performance of the network at different layers [84,85]. In both cases, first the attacker node put itself politely in the routing path, and then it silently starts dropping packets. If the attacker drops the whole packets that received from other nodes, then this kind of attack is known as *black hole attack*. Another form is *gray hole attack*, in which the attacker drops packets randomly and selectively.

In this dissertation research, we focused on *packet drop attack* due to malicious nodes and the nodes drops the whole data packets received from other nodes (*black hole attack*). In this type of attack, a malicious node first sends false routing message to tell the honest nodes that it has an optimum and shortest path route and make the honest nodes to route packets towards it.

As an example, in AODV routing protocol malicious nodes can send false RREP (which includes a false destination sequence number that is modified to be equal or higher than the one included in the RREQ) to the source node telling that it has a fresh route towards the destination node. Consequently, source node will establish route through malicious nodes to destination node. This way the malicious node can successfully creates packet drop attack in the network. The same is true in case of DSR. Necessary modification was done on AODV and DSR routing protocols for malicious nodes to purposely drop data packets, which are forwarded to them.

In Fig.6.1, a source node **S** wants to send data packet to destination node **D** but doesn't know a route to **D**. It then broadcasts RREQ packets. Here we have two scenarios. First scenario, no malicious nodes is available (normal operation). Second scenario, when there is malicious node (assume node 4 is malicious). For the first scenario, when node **D** receives RREQ, it replies by

sending RREP to the reverse path set-up when RREQ is forwarded <**D-5-2-S**>. In the second scenario, assume that node 4 is selected to be malicious; when it receives RREQ from **S**, instead of re-broadcasting RREQ to node 6, it sends false RREP to source node **S** declaring that it has fresher route towards **D** than other nodes (node 4 advertized higher sequence number than the other nodes). Consequently, **S** thinks that route discovery is completed and by ignoring the other subsequent RREP messages, it falsely chooses route towards to **D** through malicious node 4. Once the malicious node put itself in the routing path, it is able to do anything with the packets forwards to it. In our case, it chooses to drop the whole packets forwarded to it to create DoS attack.

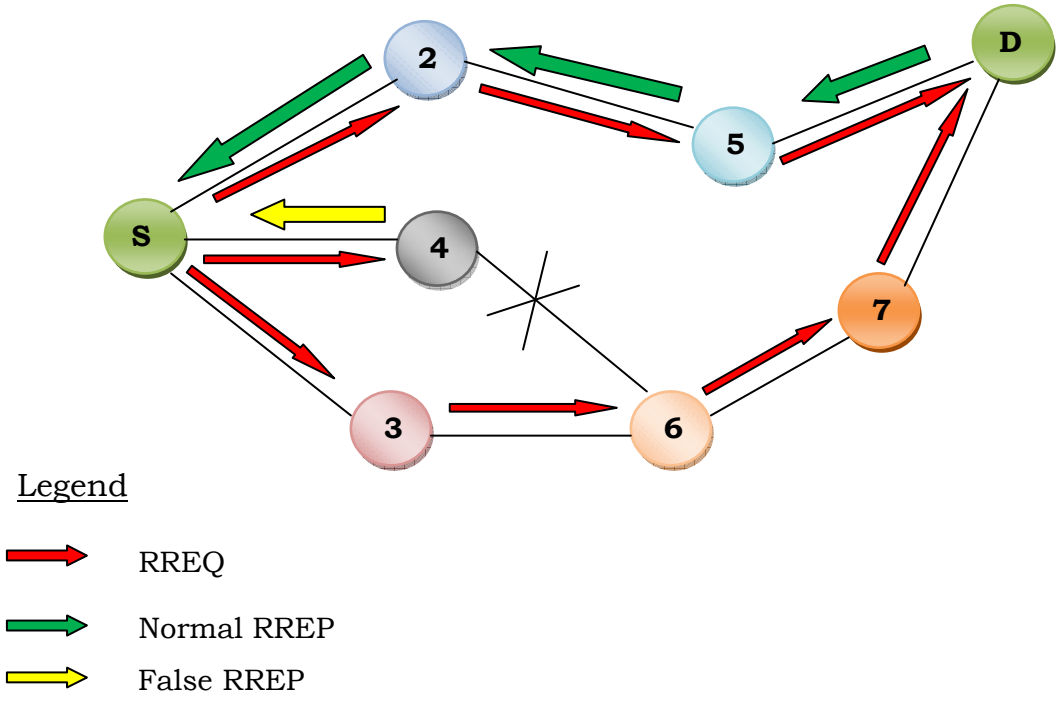


Figure 6:1 malicious packet drop attack in AODV routing protocol

6.3 Implementing malicious nodes in DSR and AODV routing protocols

In this section, the implementation of malicious node in DSR and AODV routing protocols are discussed. Main modification was done to dsragent.cc and dsragent.h files for DSR and aodv.cc and aodv.h files for AODV (see appendix III for AODV implementation) in ns-2.

In dsragent.h a variable *malicious_node* is defined in routing agent of DSR as shown in program 6.1. This variable is useful to define whether a given node is malicious or not latter in dsragent.cc, which described as in program 6.2.

Program 6.1: Adding a variable *malicious_node* of type boolean (dsragent.h)

```
/* DSR Routing Agent */
class DSRAgent : public Tap, public Agent {
public:
…
/* flow state handle functions */
…
bool ignoreRouteRequestp(SRPacket& p);
…
bool malicious_node;
…
```

In program 6.2, a variable *malicious_node* is required to be initialized to false to make sure that all nodes are initially not malicious.

Program 6.2: Initialization of variable *malicious_node* to false. (dsragent.cc)

```
/* DSR Agent Constructor */

DSRAgent::DSRAgent(): Agent(PT_DSR), request_table(128),
route_cache(NULL),
send_buf_timer(this), flow_table(), ars_table()
{
…
ll = 0;
ifq = 0;
mac_ = 0;
…
malicious_node=false;
…
}
```

Program 6.3 is implemented dsragent.cc and it is used to know that which node is set as malicious node.

Program 6.3: code to know which node is set as *malicious_node (dsragent.cc)*

```
int
DSRAgent::command(int argc, const char*const* argv)
{
TclObject *obj;

if (argc == 2)
{
if (strcasecmp(argv[1], "testinit") == 0)
{
testinit();
return TCL_OK;
}
…

if (strcasecmp(argv[1], "hacker") == 0)
{
Malicious_node=true;
return TCL_OK;
}
…
```

Program 6.4 is a tcl code, which is used to set any selected nodes as malicious nodes (in this case for instance, node 1,3, and 4 are selected to be malicious nodes). The full tcl code is available in appendix I.

Program 6.4: tcl code, which setting node 1,3,and 4 as *malicious nodes*

```
# disable random motion
for {set i 0} {$i < $val(nn) } {incr i} {
set node_($i) [$ns node]
$node_($i) random-motion 0
}
…
#
$ns at 10.0 "[$node_(1) set ragent_] hacker"
$ns at 20.0 "[$node_(3) set ragent_] hacker"
$ns at 40.0 "[$node_(4) set ragent_] hacker"
…
# Provide initial location of mobile nodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0
…
```

Program 6.5 is used to tell the malicious nodes what to do. To perform this, we used *handleForwarding(SRPacket &p)* function, which is used to forward packet on to the next node in source route when routing data packets and snooping as appropriate. Therefore, in our case we tell the malicious node to drop the whole data packet, which we call it as malicious packet drop attack when it receives data packets from other nodes.

Program 6.5: code fragment to tell the malicious node to drop data packets (dsragent.cc)

```
/* Packet forwarding and handling Functions */
  void
  DSRAgent::handleForwarding(SRPacket &p) {
        hdr_sr *srh =  hdr_sr::access(p.pkt);
         hdr_ip *iph = hdr_ip::access(p.pkt);
                        …

              if(malicious_node==true) {
           drop(p.pkt, DROP_RTR_ROUTE_LOOP);
                        p.pkt = 0;
                             }
                        …
```

6.4 Performance evaluation

In this section, the effect of malicious packet drop attack on the performance of routing protocols and TCP variants are investigated and presented.

A. Simulation scenarios and model

To explore the effect of malicious packet drop attack in ad-hoc network, a detailed simulation scenarios and model was prepared using ns-2. From TCP variants, TCP-Newreno, TCP-Sack, and TCP-Vegas were selected for the study. Routing protocols used were AODV and DSR. IEEE 802.11 for wireless networks was used as a MAC layer protocol. From channel type, a wireless channel model with a 1000m by 1000m transmission range was selected. The workload is a single TCP connection between a specific sender S (node 0) and a specific receiver R (node 9) for one TCP flow.

B. Traffic and mobility model

For traffic source, FTP was used above TCP variants. Mobility models were created for the simulations using 20 nodes. The speed was chosen randomly between 1 and 3m/s for different mobility pattern to reduce the effect of node mobility for packet loss events. All the simulations were run for 200 seconds. Different mobility and identical traffic scenarios were used across the protocols to collect fair results. Simulation parameters and setups are summarized in table 6.1.

Table 6:1

Simulation parameters and setup

Simulation parameters	**values**
Number of nodes	20
Speed of nodes	1-3m/s
MAC protocol	802.11
Routing protocols	AODV, DSR
TCP variants	TCP-Newreno,TCP-Sack, TCP-Vegas
Malicious nodes (%)	0 - 50%
Network topology	1000m X 1000m
Traffic	FTP
Simulation time	200 secs
Antenna type	Omni-antenna
Signal propagation model	Two-ray ground

C. Performance metrics

Four important performance metrics were used throughout the simulation. Throughput, end-to-end delay, PDR, and *cwnd*.

6.5 Simulation results and discussions

A. Performance of AODV and DSR routing protocols under malicious packet drop attack

Fig.6.2 shows the average throughput of AODV and DSR over TCP-Newreno under different level of security attacks. When there is no malicious node in the network, AODV achieved better throughput than DSR. This is because; DSR uses routes on its cache and it lacks any mechanism to know whether this route is stale or new. In MANET, due to mobility of nodes route change can occur frequently. Without being aware of most recent route changes, DSR may continue to send data packets through stale routes so that data packets can get dropped. However, as the percentage of malicious nodes increases from 5% to 20%, AODV showed constant performance. In this range, DSR suffered a lot. When the percentage of malicious nodes increases from 30% to 50%, both AODV and DSR constantly degraded their throughput and finally goes down to zero. The reason is, they assume that nodes are trustworthy and cooperative. However, as the number of malicious nodes increases, the probability of dropping packets increases as the nodes do not co-operate well, which affects the performance of the network.

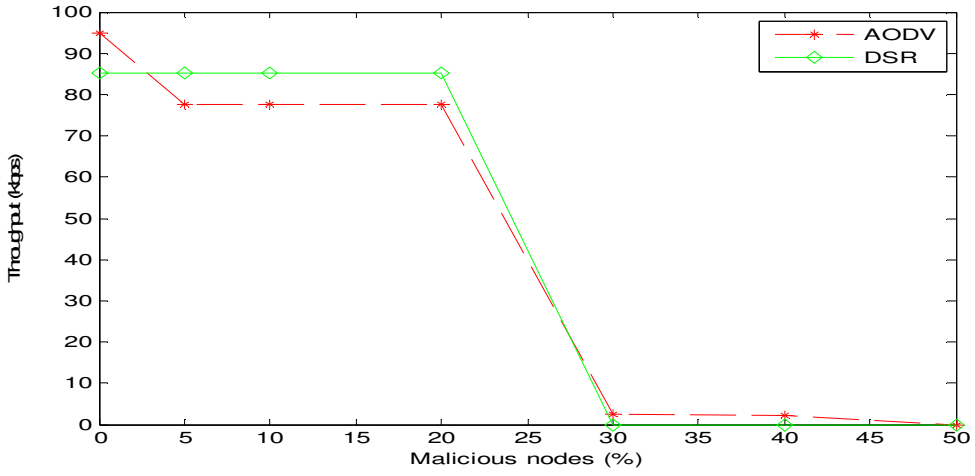


Figure 6:2 Average throughput of TCP-Newreno

Fig.6.3 shows the average end-to-end delay measurement of AODV and DSR routing protocols over TCP-Newreno. Once again, malicious nodes at different percentage were introduced in the network (from 0% to 50%). When there is no malicious node, AODV had a lower average packet delay than DSR. Besides, for both AODV and DSR routing protocols, the end-to-end delay decreased constantly when the percentage of malicious nodes increases in the network. The reason is that, when there is malicious packet drop attack, the malicious nodes can send RREQ and RREP messages before the destination node sends these control messages, which lowers delay in exchanging control messages. Malicious nodes decrease processing time to establish routing paths from source to destination.

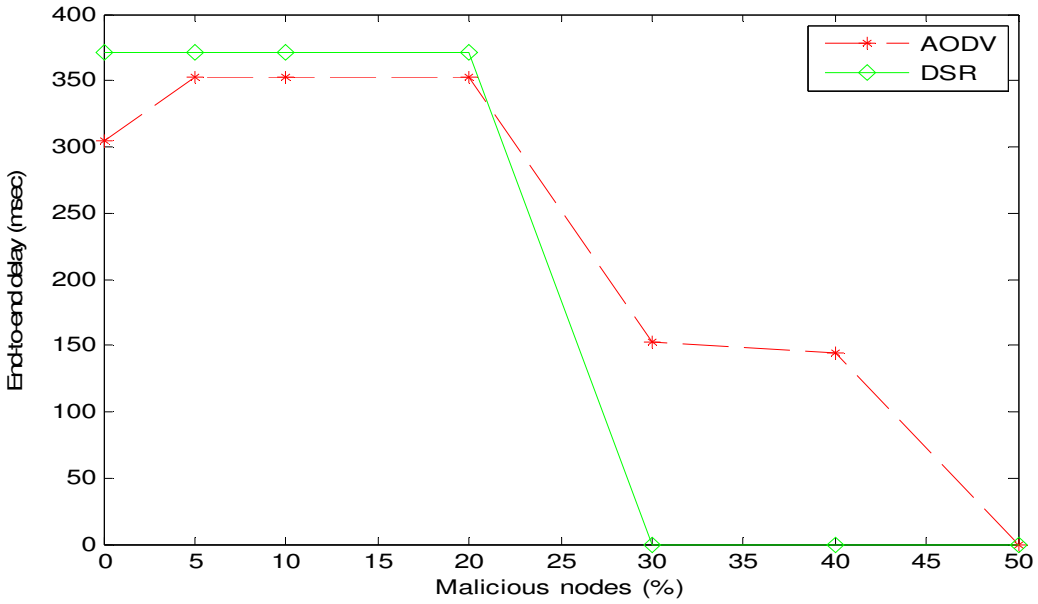


Figure 6: 0:3 Average end-to-end delay of TCP-Newreno

Fig.6.4 compared average PDR of AODV and DSR. In the absence of malicious nodes, both AODV and DSR achieved the same PDR rate (98.6%). However, in the presence of malicious nodes, which range from 5% to 50% of the total nodes, AODV had a higher and constant PDR than DSR. AODV uses

only one route to a destination. A route will get expired, if it is not recently used after a pre-determined elapsed time.

Generally, for the three performance parameters used and different level of malicious nodes, AODV achieved better performance than DSR in the presence of malicious nodes. Moreover, AODV showed almost constant performance when the percentage of malicious nodes ranges between 0% to 30%. When malicious nodes increase from 30% to 50% of the total nodes, its performance declines. Whereas, the performance of DSR fluctuates ups and down when the percentage of malicious nodes increases from 5% to 50%. Particularly, from 30% onwards, it is totally unable to send even a single packet. Therefore, for AODV the effect of increasing percentage of malicious nodes on the stated performance metrics is less than DSR.

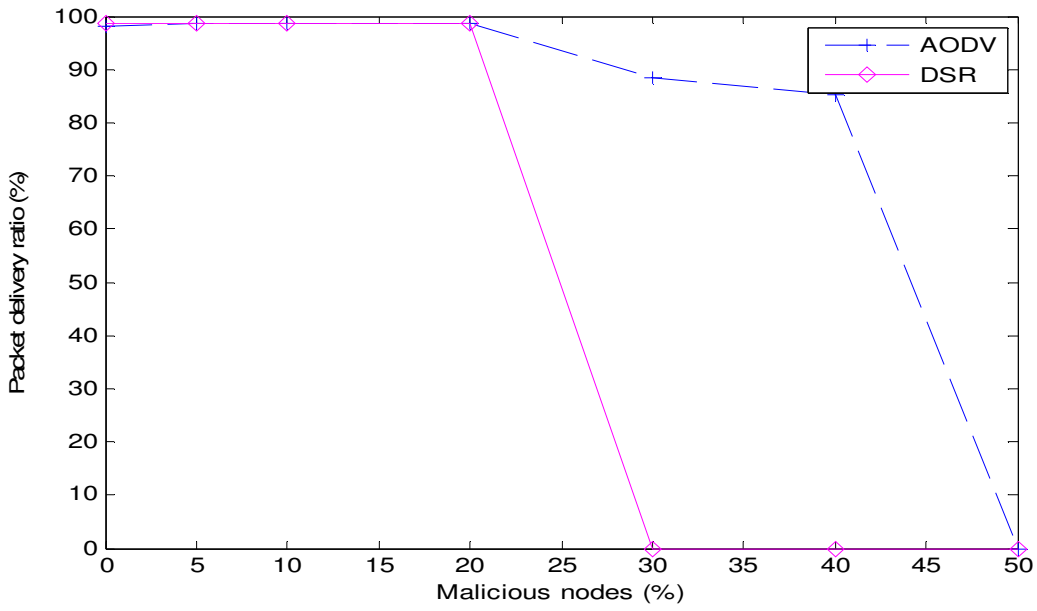


Figure 6: 0:4 Average packet delivery ratio of TCP-Newreno

Figure 6.5, 6.6, and 6.7 show the throughput, end-to-end delay, and PDR of AODV and DSR routing protocols over TCP-Sack, under different percentage of malicious node attacks. For both TCP-Newreno and TCP-Sack, DSR

performed the same result when malicious node introduced in the network ranges from 0% to 50%. However, AODV showed better throughput, which is on the average 48.61kbps for TCP-Sack and 47.41kbps for TCP-Newreno. For average delay, AODV achieved 232.93msec for TCP-Sack and 236.97msec for TCP-Newreno. For average PDR, once again AODV performed 81.62% for Sack and 81.13% for Newreno. Therefore, it is better to use AODV than DSR for both TCP-Sack than Newreno variants.

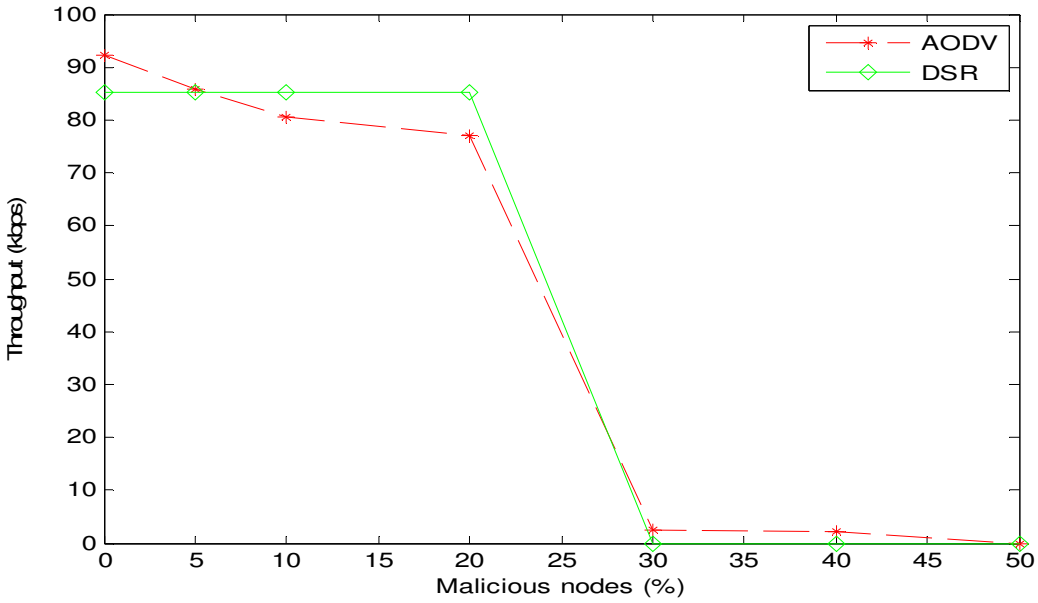


Figure 6:5 Average throughput of TCP-Sack

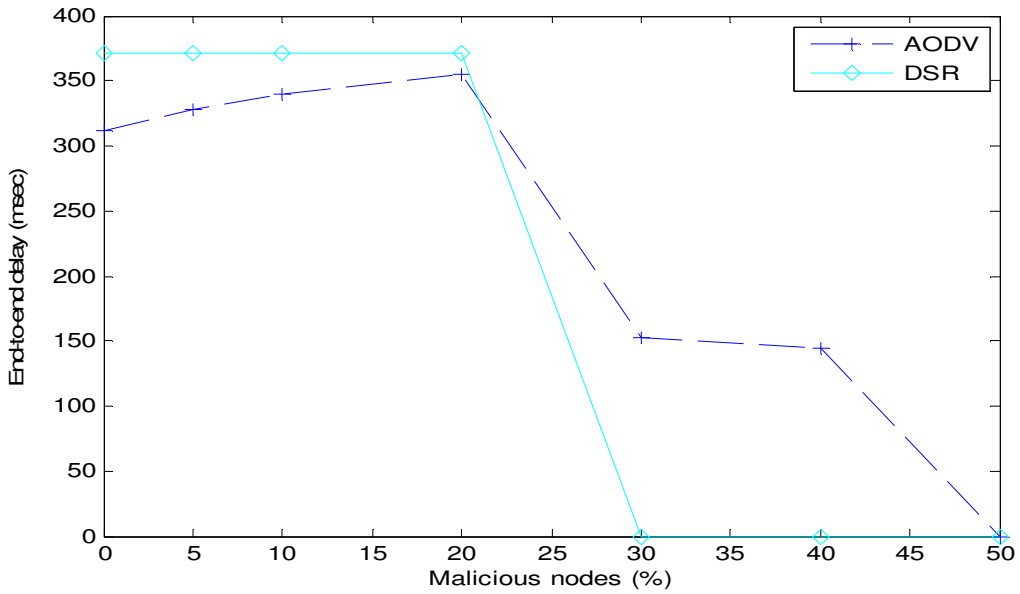


Figure 6:6 Average end-to-end delay of TCP-Sack

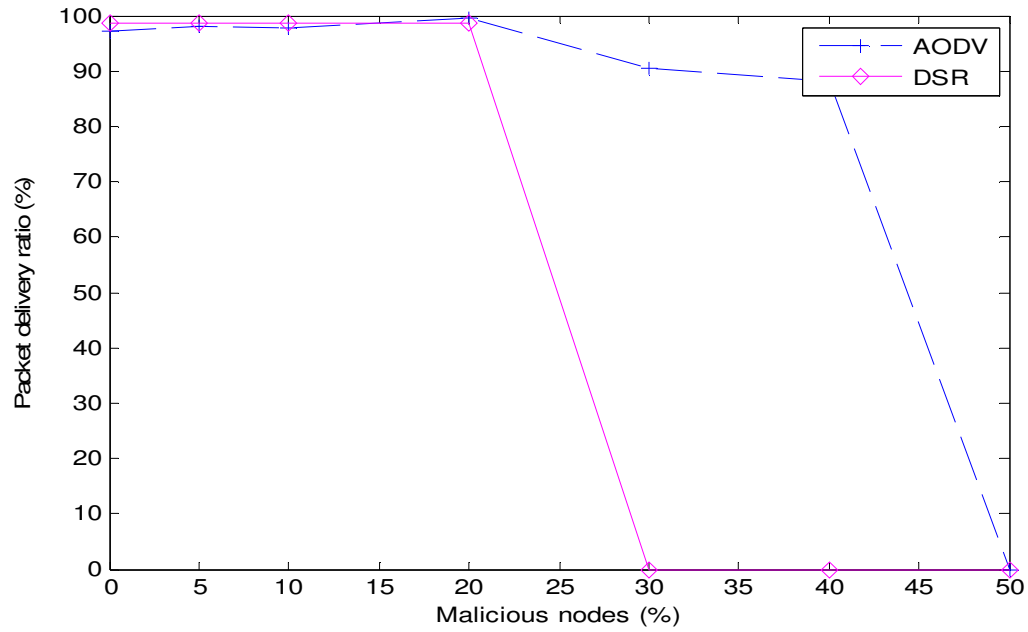


Figure 6:7 Average packet delivery ratio of TCP-Sack

Fig.6.8, 6.9, and 6.10 show average throughput, end-to-end delay, and PDR of AODV and DSR under TCP-Vegas. As shown in fig.6.8, the throughput of AODV decreased as the percentage of malicious nodes increased from 0 to 50%. When the malicious nodes introduced in the network reaches to 30%

and above, its performance degraded drastically. DSR didn't send any data packets in the presence and absence of malicious nodes. Therefore, DSR is not suitable for TCP-Vegas. End-to-end delay and PDR of AODV is by far better than DSR. Therefore, TCP-Vegas can be coupled with AODV in MANET whether malicious nodes are presented or not.

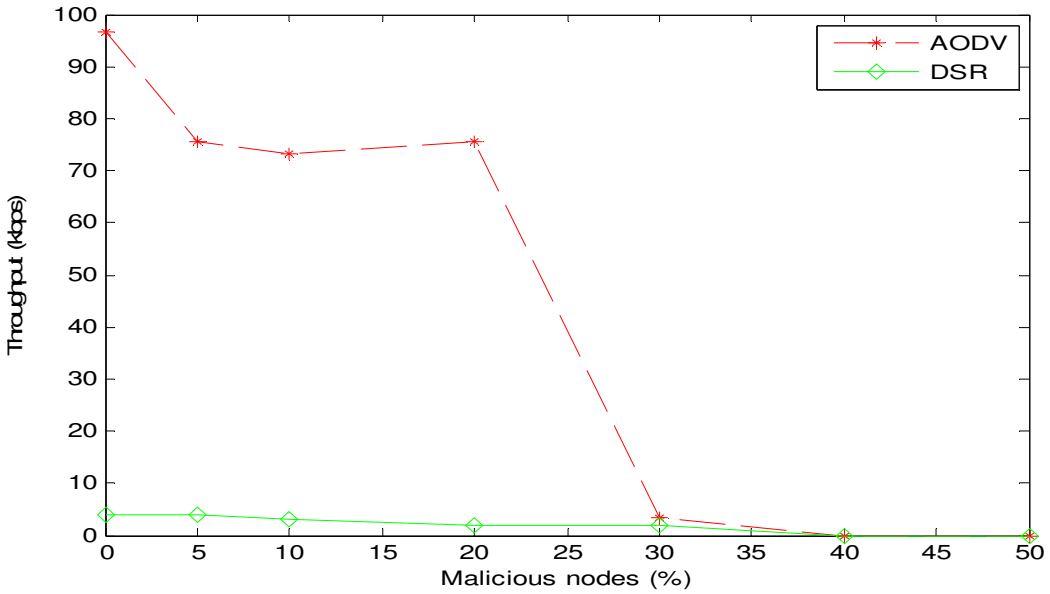


Figure 6:8 Average throughputs of TCP-Vegas

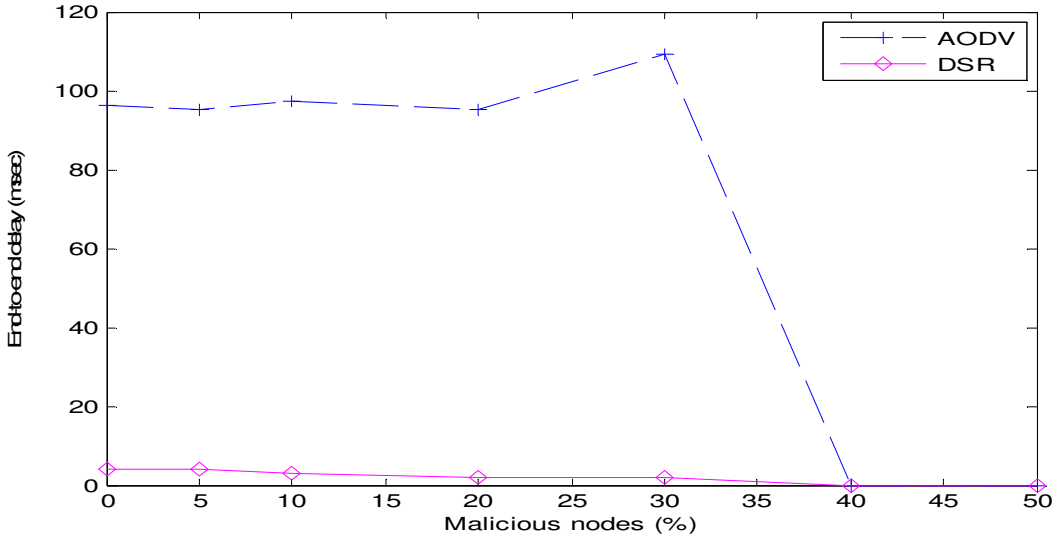


Figure 6:9 Average end-to-end delay of TCP-Vegas

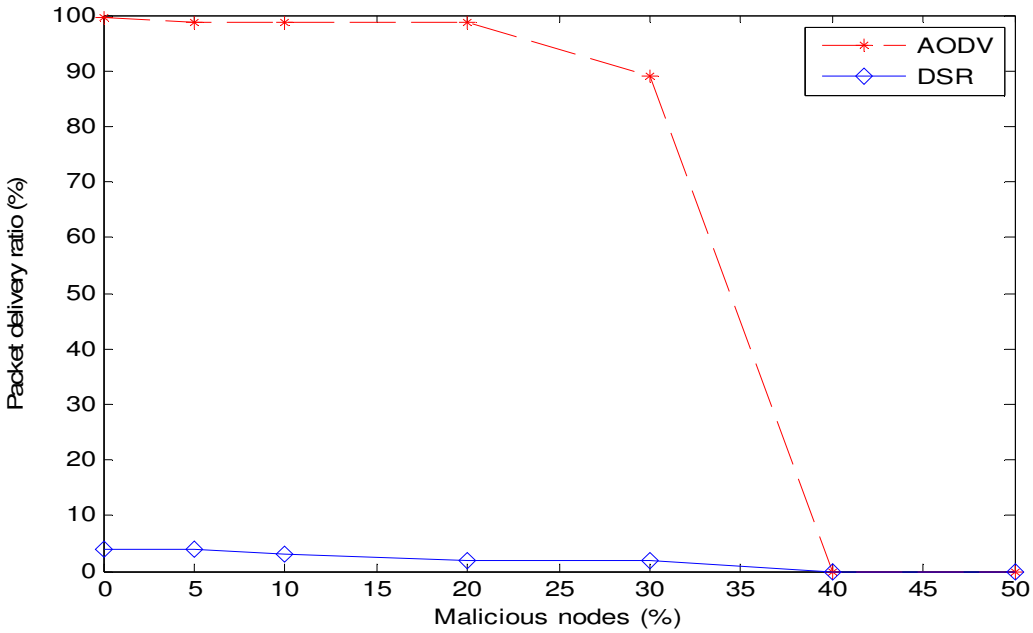


Figure 6:10 Average packet delivery ratio of TCP-Vegas

B. Performance of TCP variants under malicious packet drop attack

Fig. 6.11 shows the average throughput of TCP variants over AODV routing protocol under different percentage of malicious nodes. In the absence of malicious nodes (0%), the throughput of TCP-Vegas, Newreno, and Sack were found to be 96.51, 94.76, and 92.22 kbps, respectively. Hence, Vegas outperformed Newreno and Sack with 0% malicious nodes. However, as the percentage of malicious nodes added in the network increased from 5% to 20%, on the average Sack and Newreno performed better than Vegas though all of them are affected by malicious packet drop attack. This is because; Sack and Newreno can survive from multiple packet losses from one RTT without expiring their RTO. Up to 20% malicious nodes, the three variants are robust to DoS attack. However, above 20% the throughput of all TCP variants degraded drastically and finally goes down to zero. From the simulation result as shown in fig.6.10, malicious nodes dropped all packets forwarded to them. Since TCP variants assume that all packets drop/loss as

an indication of network congestion, they falsely retransmit the lost packet and reduce the packet sending rate, which degrades their performance significantly.

More specifically, while malicious nodes drop the whole packet forwarded to them, all TCP variants keep retransmitting the packet and call exponential back-off algorithm to double RTO. Consequently, TCP sender will remain idle for a long time, reduce its throughput, and finally leads to resetting the connection after subsequent timeouts.

Generally, the major problem of malicious packet drop attack is to make the TCP variants to call exponential back-off algorithm repeatedly, starve the TCP connection, and finally leads to connection re-establishment. This is known as DoS attack. From fig.6.11, it can be concluded that in the range of 20% to 50% malicious nodes, TCP variants are vulnerable to malicious packet drop (DoS) attack. During this period, TCP variants are in slow-start phase and have small *cwnd* such that a smaller number of packet losses are needed to force them to trigger RTO.

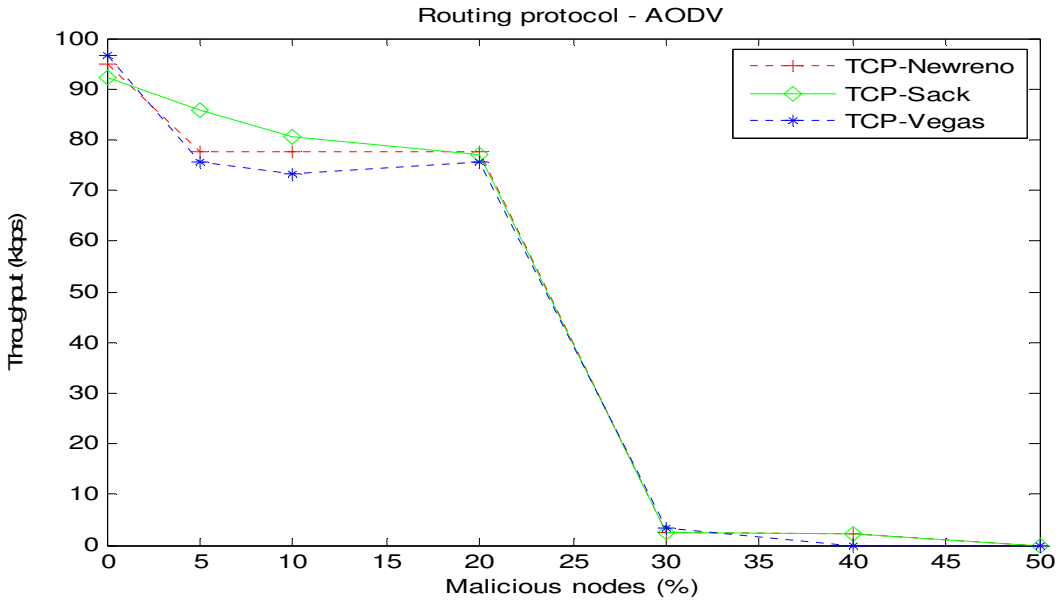


Figure 6:11 Average throughputs of TCP variants over AODV

Fig.6.12 shows the average end-to-end delay of TCP variants over AODV routing protocol. Vegas achieved the least delay compared to Sack and Newreno. Newreno is the worst performer. TCP-Vegas adjust its sending rate before packet loss due to malicious nodes happened. This behavior of Vegas results in correct sending rate without depending on packet loss indication, which in turn results in lower delay. Moreover, TCP-Vegas use RTT to estimate delay in order to adjust *cwnd*, which is an important parameter to find an accurate estimation in MANET. However, Sack and Newreno suffered to frequent retransmission and exhaustive processing of retransmitted packets at the destination node due to malicious packet drop attack. As shown in fig.6.13, in terms of PDR, Newreno and Sack performed better than Vegas when percentage of malicious nodes is above 30%. However, up to 30%, all the three TCP variants almost showed the same result.

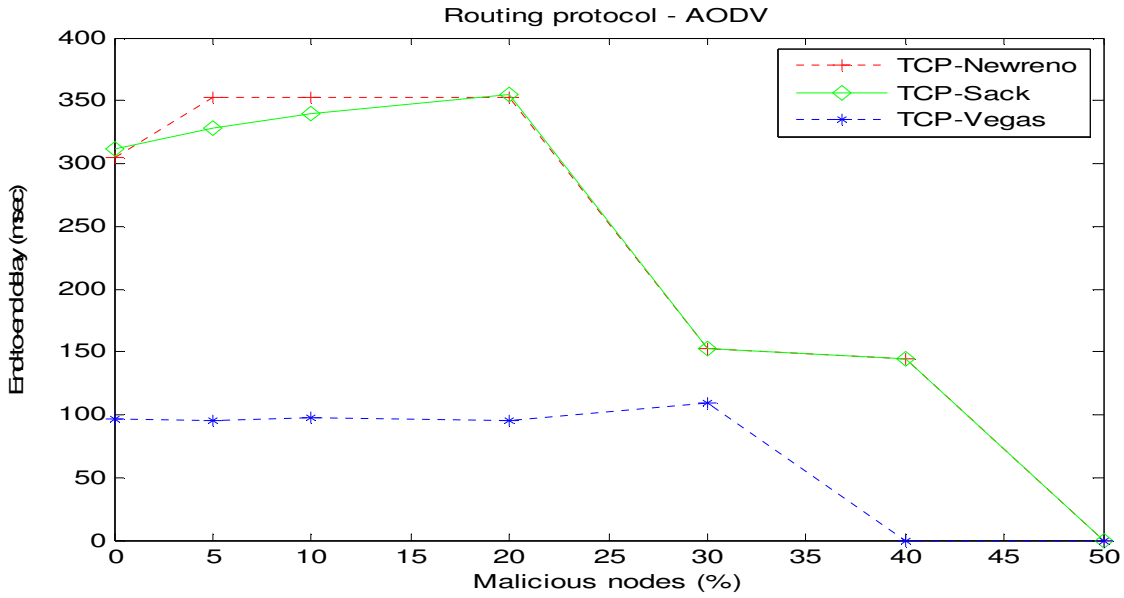


Figure 6:12 Average end-to-end delays of TCP variants over AODV.

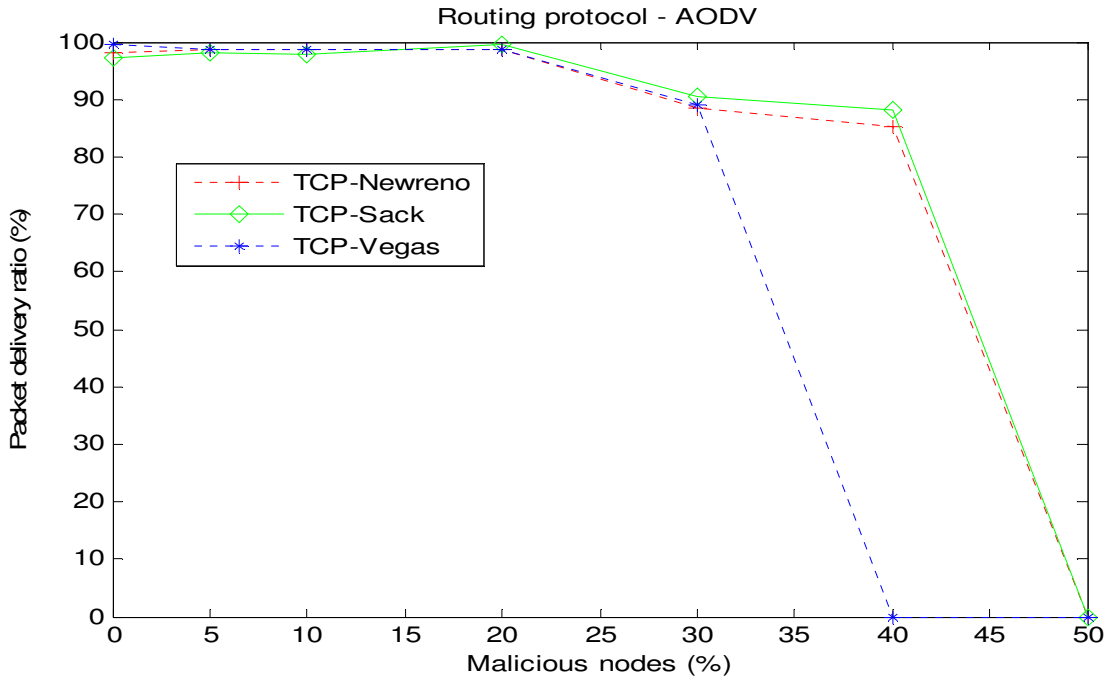
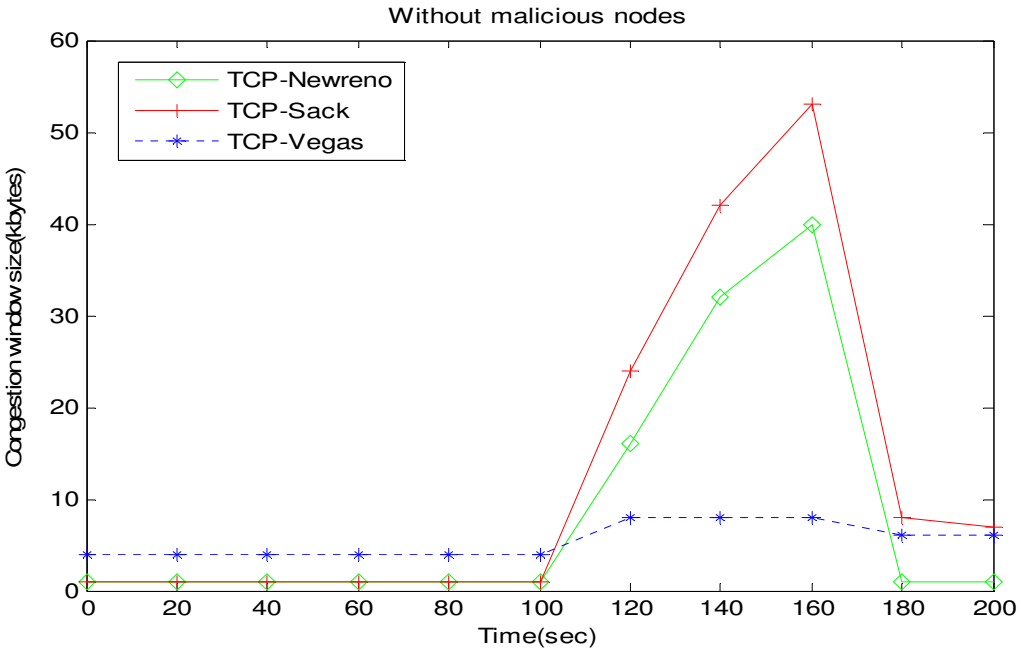


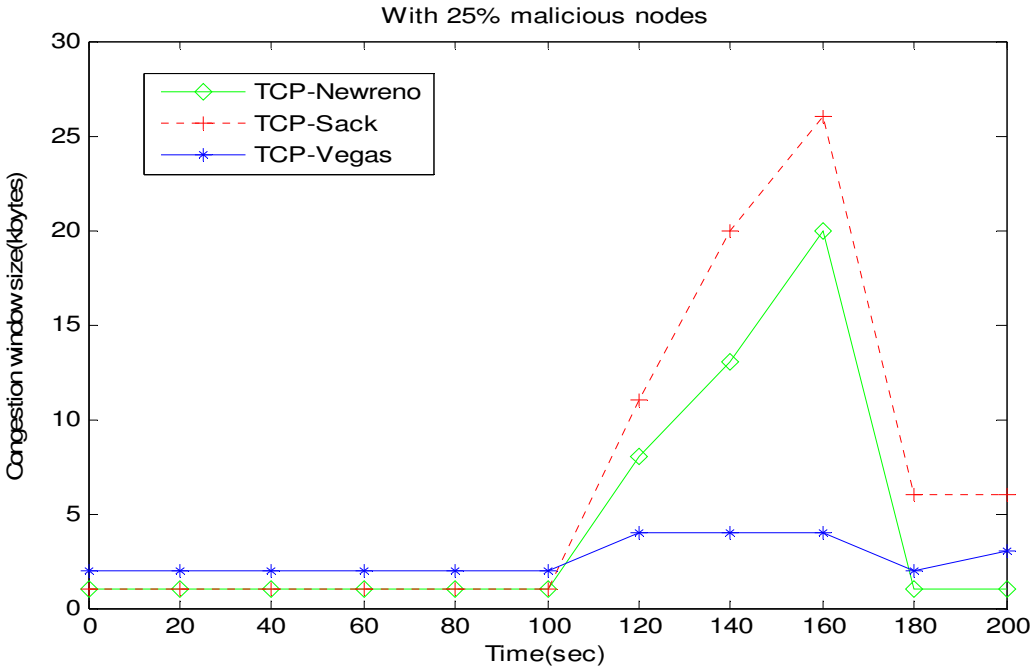
Figure 6:13 Average packet delivery ratio of TCP variants over AODV

Fig.6.14 shows the average window size of Newreno, Sack, and Vegas TCP variants over AODV. Experiment was conducted using two simulation scenarios: 1) when the network is free from malicious nodes (0% malicious nodes). 2) When there are 25% malicious nodes. From the simulation results, it is observed that for the first scenario (no malicious nodes), on the average Sack and Newreno achieved similar *cwnd* size evolution. Sack of course achieved better than Newreno. Larger *cwnd* means better throughput as they are directly related with each other. TCP-Vegas showed almost a constant *cwnd* evolution with minimum value throughout the simulation. Vegas adjusts its sending rate before packet loss happens, which results in consistent and correct sending rate without depending on loss indication. For the second scenario, when 25% percent malicious nodes introduced in the network, the *cwnd* size evolution of Newreno, Sack, and Vegas dropped significantly.

Generally, for both simulation scenarios (in the presence and absence of malicious nodes), Newreno and Sack, which are Reno based TCP variants are more aggressive in their sending rate than Vegas. They usually retransmits packets that are outstanding (packets which are sent but not yet acknowledged) after the RTO gets expired, which may lead to more number of RTO's. However, their ability to recover from multiple packet losses in one window of data helped them to avoid consecutive time outs. On the other hand, TCP-Vegas exhibited little time in RTO. This is because; the constant window size evolution of Vegas was driven by its proactive and conservative congestion avoidance mechanism, which transmits fewer packets along the path than the reactive variants.



a) No malicious node



b) With 25% malicious nodes

Figure 6:14 Average *cwnd* size of TCP variants over AODV with 0% and 25% malicious nodes.

6.6 Performance of TCP-PLDR and TCP-SACK under security attack

TCP-PLDR and TCP-SACK were evaluated in the presence of malicious packet drop attack over AODV routing protocol. As shown in fig. 6.15, up to 15% malicious nodes, TCP-PLDR performed better than TCP-SACK. However, from 15% to 30%, TCP-SACK reported better throughput than TCP-PLDR. On average the sending rate of TCP-PLDR was 60.5 kbps, whereas the rate of TCP-SACK was about 56.2 kbps. Therefore, on average TCP-PLDR is more robust to malicious packet drop attack than TCP-SACK. Fig.6.16 shows end-to-end delay of both protocols. Up to 20% malicious nodes, both PLDR and Sack achieved almost the same delay. However, from 20% to 50% malicious nodes in the network TCP-PLDR reported lower delay than SACK. On the

average delay measurement of TCP-PLDR was found to be 212.7 msec. Whereas, TCP-SACK reported 228.7 msec. From this result, TCP-PLDR achieved lower delay than Sack. The lower end-to-end delay means the better performance of the protocols.

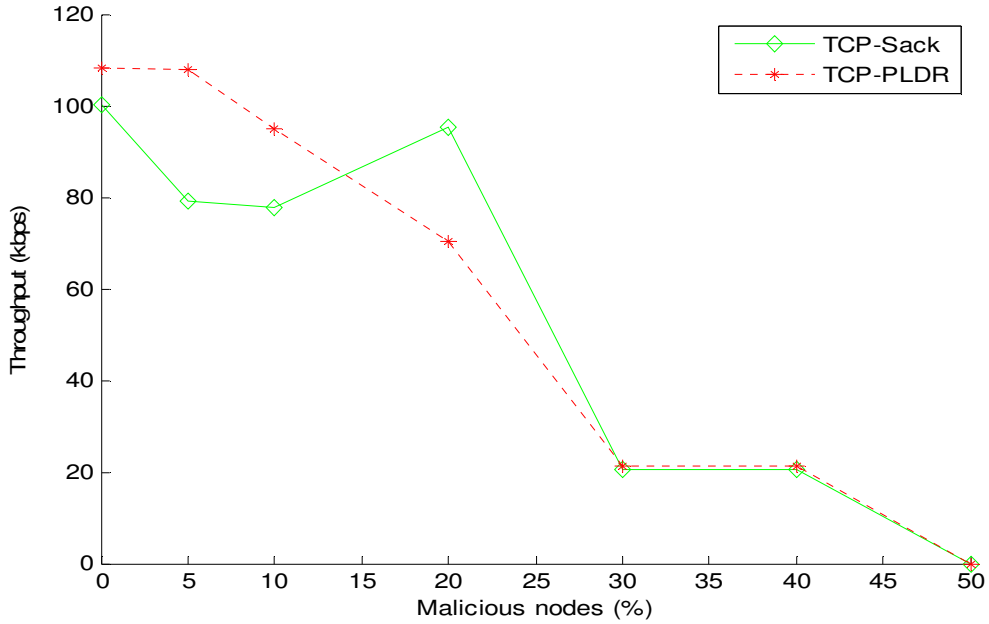


Figure 6:15 Average throughput of TCP-PLDR and SACK over AODV

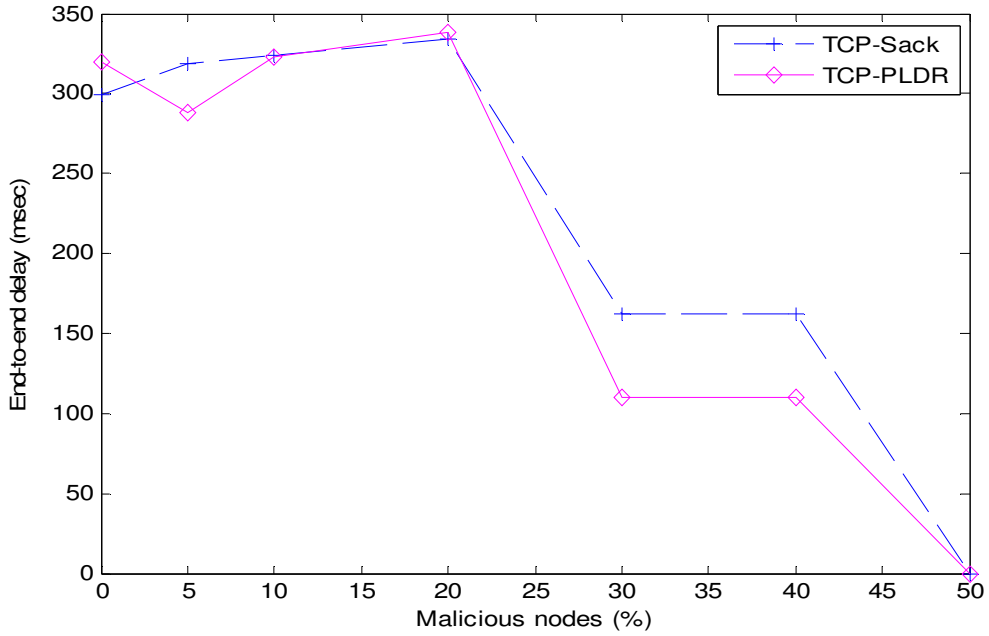


Figure 6:16 Average end-to-end delay of TCP-PLDR and SACK over AODV

As shown in fig.6.17, up to 30% malicious nodes, both PLDR and Sack reported almost the same PDR. However, from 30% onwards TCP-PLDR recorded better packet delivery ration than Sack. On average, the PDR value of TCP-Sack is 70.6%. Whereas, PDR value of TCP-PLDR was found as 80.1%. Therefore, TCP-PLDR reported better PDR than Sack so that the former is more robust to resist malicious packet drop attack than the latter.

Fig.6.18 is goodput value of PLDR and Sack. Up to 20%, TCP-PLDR performed better than Sack. From 20% to 30% both protocols performed almost the same result. From 30% to 50% malicious nodes introduced in the network Sack showed better goodput than PLDR.

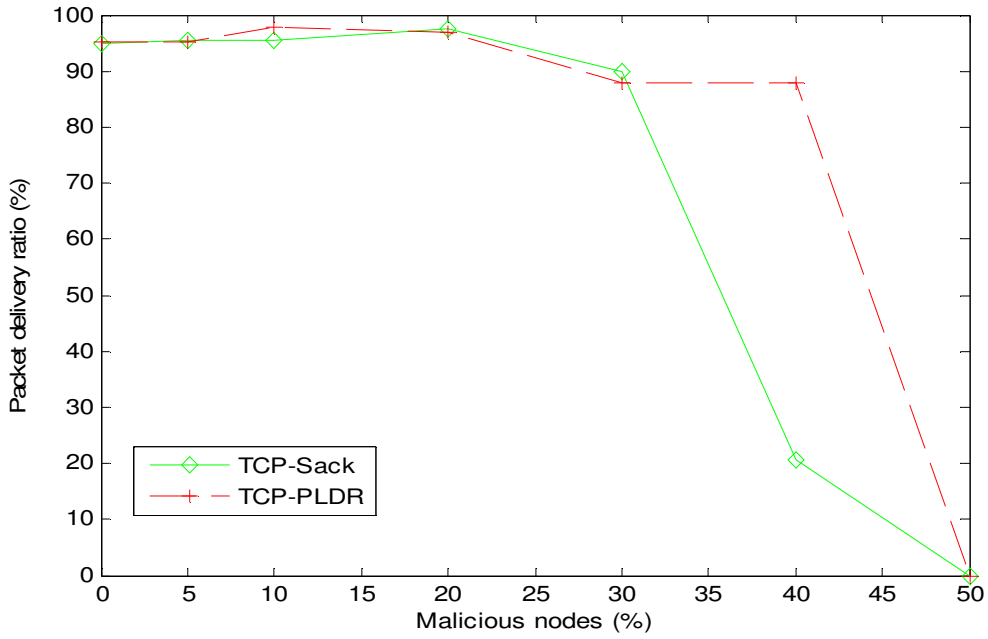


Figure 6:17 Average PDR of TCP-PLDR and SACK over AODV

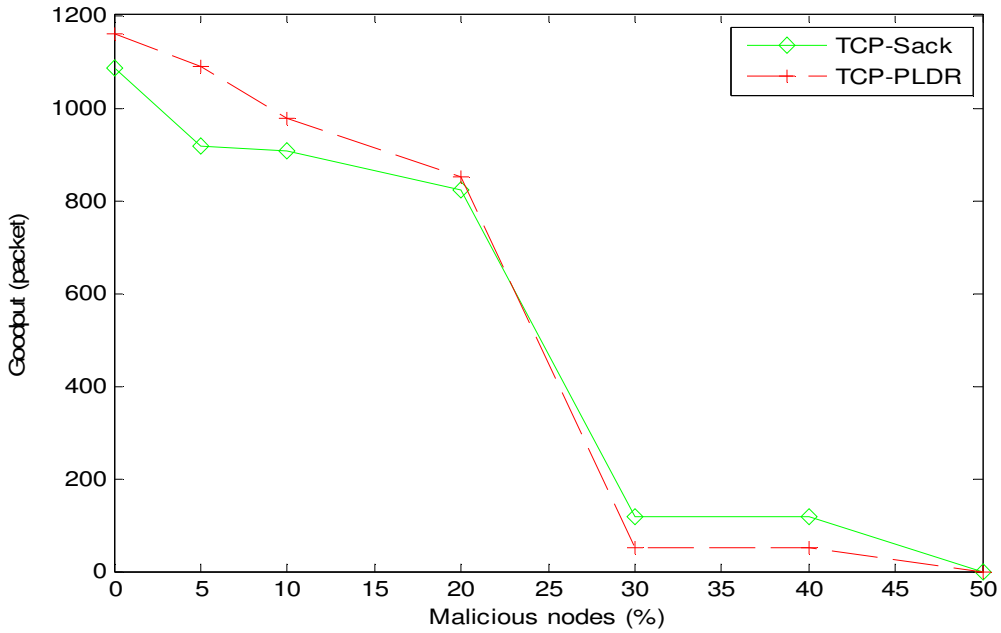


Figure 6:18 Average goodput of TCP-PLDR and SACK over AODV

6.7 Summary

This chapter investigated the impact of malicious packet drop attack on the performance of two reactive TCP variants (Newreno and Sack), one proactive TCP variant (Vegas), and two reactive routing protocols (AODV and DSR) by making use of throughput, end-to-end delay, and PDR performance metrics.

Modification was made in DSR and AODV routing protocols for nodes purposely drop data packets, which are forwarded to them.

Three simulation scenarios were taken into consideration. 1- Performance of routing protocols under malicious packet drop attack, 2- Performance of TCP variants under malicious packet drop attack, and 3- Performance of TCP-PLDR and TCP-SACK under security attack.

Simulation was conducted by adding different percentage of malicious nodes in the network. Results showed that from TCP variants, Vegas outperformed Newreno and Sack in the absence of malicious nodes (0% malicious nodes). However, as the percentage of malicious nodes added in the network increases from 5% to 50%, Newreno and Sack performed better than Vegas though all of them are affected by malicious nodes.

The effect of this attack was magnified with the increase of attack duration and the number of attackers. Moreover, it is confirmed that malicious packet drop attack affects both rate-based and window-based TCP variants. It is also noted that, even though both AODV and DSR protocols are highly affected by malicious nodes, in terms of throughput, delay, and PDR performance metrics and different level of malicious nodes, AODV achieved better performance than DSR. Therefore, AODV is more robust to malicious packet drop attack than DSR.

Finally, performances of TCP-PLDR and TCP-SACK were evaluated exhaustively under security attack (malicious packet drop attack). Up on completion of exhaustive simulation, it is confirmed that TCP-PLDR is more robust to malicious packet drop attack than TCP-SACK. In terms of throughput TCP-PLDR improved the performance of TCP-SACK by 7.5%. In terms of end-to-end delay, TCP-PLDR has shown 7% improvement. Whereas, with respect to PDR, once again TCP-PLDR achieved better performance improvement, which is about 13.5% than SACK.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

In this dissertation research, we have proposed TCP-PLDR protocol, which is a modification of one of TCP variants, TCP-SACK. It uses packet loss detection and response mechanism for TCP to differentiate packet loss due to route failure or network congestion. Simulation and analytical results showed that the performance of TCP-PLDR is better than TCP-SACK. We used different simulation scenarios to study the behavior of both protocols. These are, route failure/route change scenario, both route failure and congestion loss scenario, only congestion loss scenario, multi-path route, and wireless channel error. Moreover, TCP-PLDR was compared with other TCP variants and results showed that it shows significant performance improvement in terms of throughput while it shoed comparable results for end-to-end delay and PDR performance metrics. TCP-PLDR was also compared with four routing protocols (AODV, DSR, DSDV, and TORA).

TCP-PLDR uses timer based packet loss detection scheme, which is explicit loss detection scheme than acknowledgment based that is implicit loss notification scheme. It is an end-to-end solution; modification was done only at the TCP-SACK receiver and sender sides, which makes it easy for deployment.

The performance of TCP variants over different routing protocols were compared and evaluated. We used different mobility patterns and node densities. The best performing combination of routing protocols and TCP variants that goes with the proposed algorithm (the main part of the dissertation study) under different network scenarios were selected. From

routing protocols, AODV performed the highest throughput for all TCP variants. From TCP variants on the average, TCP-Newreno and TCP-SACK achieved a higher throughput than the other variants over all routing protocols. Hence, we have mainly selected TCP-SACK from TCP variants and AODV from routing protocols to study packet loss detection and response mechanism for TCP in MANET.

From the simulation result, it can be concluded that every protocol performed differently in different MANET scenarios. Therefore, using different combination of TCP variants and routing protocols leads to optimum performance than a single one. Besides, TCP is unable to distinguish between packet losses due to route failure and those due to congestion. Hence, routing protocols have their own effect on the performance of TCP variants.

In this paper, we explored the effect of malicious packet drop attack on the performance of three TCP variants and two reactive routing protocols. Simulation was conducted using ns-2 and results show that as the percentage of malicious nodes increases in the network, the performance of TCP variants and routing protocols degraded. The effect of this attack is magnified with the increase of attack duration and the number of attackers. In addition, malicious packet drop attack affects both rate-based and window-based TCP variants.

Now days, TCP is called upon to provide reliable and efficient data transfer in ad-hoc networks. As a consequence, solutions are required to extend the domain of effective TCP operability to ad hoc networks. This is because; TCP is highly affected by malicious packet drop attack in this type of network. Besides, normal operation of routing protocols is affected by malicious nodes.

As a future work, other types of packet losses not related to congestion loss; packet loss due to packet drop attack shall be incorporated in TCP-PLDR

protocol. Other routing protocols shall also be tested with other variants of TCP. Simulation shall also be conducted under different traffic patterns like CBR, TELNET. It is also interesting that routing protocols shall be modified so that honest nodes can distinguish malicious nodes.

Publications (Journals and conference proceedings)

- Henock Mulugeta, Dereje H/mariam, K. Raimond "TCP-PLDR: A Novel Packet Loss Detection and Response Mechanism for TCP in Mobile Ad Hoc Networks." *Ad Hoc & Sensor Wireless Networks, An International Journal (AHSWN).(Philadelphia, USA)*
- Henock M., Dereje H.M, and Kumudha R. (2014/15) "Investigating the Effects of Security Attacks on the Performance of TCP Variants and Routing Protocols in MANET", *Int. J. Computer Applications in Technology. (Switzerland, United Kingdom).*
- Henock Mulugeta, and Kumudha Raimond. "Performance of TCP variants over proactive and reactive routing protocols in MANET." *In Proceedings of the International Conference on Management of Emergent Digital EcoSystems (ACM MEDES'2012), pp. 123-130. ACM, 2012.(A.A, Ethiopia).*
- Henock Mulugeta, and Kumudha Raimond. "Performance improvement of TCP using TCP-DOOR-TS algorithm in mobile ad hoc networks." *13th IEEE International Conference on Communication Technology (IEEE ICCT '11).pp. 642-646. pp. 642-646. 2011.(China)*
- Henock Mulugeta, and Kumudha Raimond. "Performance of TCP variants over proactive and reactive routing protocols in MANET." *Journal of the Ethiopian Society of Electrical Engineers. 6th Scientific Conference on Electrical Engineering (CEE -2012).* 5-6, October 2012. (A.A, Ethiopia)

Appendix I

(MANET simulation tcl file)

```
# 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 mobile
nodes
set val(rp)             AODV                           ;# 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 end
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 channeles
set chan_0_ [new $val(chan)]
set chan_1_ [new $val(chan)]
```

…

```
set chan_19_ [new $val(chan)]

# configure mobile nodes 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
  }

  # Set the following nodes as malicious
  $ns at 10.0 "[$node_(5) set ragent_] hacker"
  $ns at 10.0 "[$node_(8) set ragent_] hacker"
  $ns at 50.0 "[$node_(12) set ragent_] hacker"
  $ns at 100.0 "[$node_(15) set ragent_] hacker"
  $ns at 150.0 "[$node_(3) set ragent_] hacker"
  $ns at 250.0 "[$node_(16) set ragent_] hacker"
  $ns at 200.0 "[$node_(7) set ragent_] hacker"
  $ns at 100.0 "[$node_(14) set ragent_] hacker"

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

# Generation of mobile node movements
$ns at 10.0 "$node_(0) setdest 100.0  50.0  30.0"
$ns at 30.0 "$node_(0) setdest 5.0  50.0  25.0"
$ns at 50.0 "$node_(0) setdest 50.0  50.0  25.0"
                        ...

$ns at 70.0 "$node_(16) setdest 700.0  500.0  28.0"
$ns at 45.0 "$node_(16) setdest 50.0  390.0  25.0"
$ns at 143.0 "$node_(16) setdest 300.0  500.0  30.0"
$ns at 180.0 "$node_(16) setdest 400.0  651.0  30.0"
                        ...

$ns at 110.0 "$node_(19) setdest 450.0   630.0  26.0"
$ns at 130.0 "$node_(19) setdest 500.0   800.0  26.0"
$ns at 50.0 "$node_(19) setdest 250.0   500.0  30.0"
$ns at 70.0 "$node_(19) setdest 500.0   800.0  30.0"
$ns at 30.0 "$node_(19) setdest 200.0   800.0  26.0"
$ns at 40.0 "$node_(19) setdest 368.0   789.0  27.0"
$ns at 180.0 "$node_(19) setdest 600.0   800.0  28.0"

#A two state Markov chain error model
proc TwoStateErr {} {
 set tmp [new ErrorModel/Uniform 0 pkt]
 set tmp1 [new ErrorModel/Uniform 1 pkt]

 # Array of states (error models)
 set m_states [list $tmp $tmp1]
# Durations for each of the states, tmp and tmp1
set m_periods [list 114.94 6.71]

# Transition state model matrix
set m_transmx { {0.9913 0.0087} {0.1491  0.8509} }
set m_trunit pkt

 # Use time-based transition
  set m_sttype time
  set m_nstates 2
  set m_nstart [lindex $m_states 0]
  set em [new ErrorModel/MultiState $m_states $m_periods $m_transmx
$m_trunit $m_sttype $m_nstates $m_nstart]
return $em
}
```

```
$ns node-config -IncomingErrProc TwoStateErr -OutgoingErrProc TwoStateErr

# Set a TCP connection between node_(0) and node_(9)
set tcp1 [new Agent/TCP/Sack1]
$tcp1 set class_ 2
set sink1 [new Agent/TCPSink/Sack1]
$ns attach-agent $node_(0) $tcp1
$ns attach-agent $node_(9) $sink1
$ns connect $tcp1 $sink1
set ftp1 [new Application/FTP]
#$cbr set packetSize_ 500
#$cbr set interval_ 0.005

$ftp1 attach-agent $tcp1
$ns at 10.0 "$ftp1 start"

# Set a TCP connection between node_(1) and node_(8)

set tcp2 [new Agent/TCP/Sack1]
$tcp2 set class_ 2
set sink2 [new Agent/TCPSink/Sack1]
$ns attach-agent $node_(1) $tcp2
$ns attach-agent $node_(8) $sink2
$ns connect $tcp2 $sink2
set ftp2 [new Application/FTP]
#$cbr set packetSize_ 500
#$cbr set interval_ 0.005
$ftp2 attach-agent $tcp2
$ns at 10.0 "$ftp2 start"

# Set a TCP connection between node_(2) and node_(16)
set tcp3 [new Agent/TCP/Sack1]
$tcp2 set class_ 2
set sink3 [new Agent/TCPSink/Sack1]
$ns attach-agent $node_(2) $tcp3
$ns attach-agent $node_(16) $sink3
$ns connect $tcp3 $sink3
set ftp3 [new Application/FTP]
#$cbr set packetSize_ 500
#$cbr set interval_ 0.005

$ftp3 attach-agent $tcp3
$ns at 10.0 "$ftp3 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 $tcp1 $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 $tcp1 $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 $tcp1 $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
```

Appendix II

(awk file to calculate throughput, delay, and PDR)

```
BEGIN {
  idHighestPacket = 0 ;
  idLowestPacket = 10000 ;
  rStartTime = 1000.0 ;
     rEndTime = 0.0 ;
  nSentPackets = 0 ;
       nReceivedPackets = 0 ;
  nReceivedBytes = 0 ;
          rTotalDelay = 0.0;
        }
{
  strEvent = $1 ;
        rTime = $2 ;
  strAgt = $4 ;
        idPacket = $6 ;
  strType = $7 ;
        nBytes = $8 ;

 if ( strAgt == "AGT"    &&    strType == "tcp" ) {
           if ( idPacket > idHighestPacket )
                idHighestPacket = idPacket ;
           if ( idPacket < idLowestPacket )
                   idLowestPacket = idPacket ;
           if ( rTime > rEndTime )
                   rEndTime = rTime ;
           if ( rTime < rStartTime )
```

```
                rStartTime = rTime ;

        if ( strEvent == "s" ) {
                nSentPackets += 1 ;
                rSentTime[ idPacket ] = rTime ;
        }
        if ( strEvent == "r"   &&   idPacket >= idLowestPacket ) {
                nReceivedPackets += 1 ;
                   nReceivedBytes += nBytes ;
                rReceivedTime[ idPacket ] = rTime ;
                rDelay[ idPacket ] = rReceivedTime[ idPacket ] - rSentTime[
idPacket ] ;
        }
  }
}

END {
  rTime = rEndTime - rStartTime ;
  rThroughput = (nReceivedBytes*8 / ( rEndTime - rStartTime ))/1000 ;
  rPacketDeliveryRatio = nReceivedPackets / nSentPackets * 100 ;
  for ( i=idLowestPacket; ( i<idHighestPacket ); i+=1 )
        rTotalDelay += rDelay[ i ] ;
  if ( nReceivedPackets != 0 )
        rAverageDelay = (rTotalDelay / nReceivedPackets) *1000 ;

  printf( "AverageDelay in ms: %15.5f Throughput in kbps: %15.2f
PacketDeliveryRatio in %: %10.2f\n",        rAverageDelay,  rThroughput,
rPacketDeliveryRatio ) ;

}
```

Appendix III

(AODV modification in ns-2)

In this section, the implementation of malicious node in AODV routing protocols are discussed. Main modification was done to aodv.cc and aodv.h files for AODV routing protocol in ns-2. In aodv.h a variable *malicious_node* is defined in routing agent of AODV as shown in program 1. This variable is useful to define whether a given node is malicious or not latter in aodv.cc, which described as in program 2.

Program 1: Adding a variable *malicious_node* of type boolean

```
/* AODV  Routing Agent */
class AODV: public Agent {
...
/*
* History management
*/
double PerHopTime(aodv_rt_entry *rt);
...
bool malicious_node;
...
```

In program 2, a variable malicious is required to be initialized to false to make sure that all nodes are initially not malicious.

Program 2: Initialize variable *malicious_node* to false.

```
/* AODV Constructor */
AODV::AODV(nsaddr_t id) : Agent(PT_AODV),btimer(this),
htimer(this), ntimer(this), rtimer(this), lrtimer(this), rqueue() {
index = id;
seqno = 2;
bid = 1;
...
Malicious_node=false;
...
```

Program 3 is implemented aodv.cc and it is used to know that which node is set as malicious node.

Program 3: code to know which node is set as *malicious_node*

```
int
AODV::command(int argc, const char*const* argv) {
    if(argc == 2) {
    Tcl& tcl = Tcl::instance();
    if(strncasecmp(argv[1], "id", 2) == 0) {
    tcl.resultf("%d", index);
    return TCL_OK;
    }
    …
    if(strcmp(argv[1], "hacker") == 0) {
    malicious_node = true;
    return TCL_OK;
    }
    …
    }
```

Program 4 is a tcl code, which is used to set any selected nodes as malicious nodes (in this case for instance, node 1,3, and 4 are selected to be malicious nodes).

Program 4: tcl code, which setting node 1,3,and 4 as malicious nodes

```
# disable random motion
for {set i 0} {$i < $val(nn) } {incr i} {
    set node_($i) [$ns node]
    $node_($i) random-motion 0 }
    …
    #
$ns at 10.0 "[$node_(1) set ragent_] hacker"
$ns at 20.0 "[$node_(3) set ragent_] hacker"
$ns at 40.0 "[$node_(4) set ragent_] hacker"
    …
# Provide initial location of mobile nodes
    $node_(1) set X_ 35.0
    $node_(1) set Y_ 440.0
    $node_(1) set Z_ 100.0
```

```
$node_(2) set X_ 50-.0
$node_(2) set Y_ 200.0
$node_(2) set Z_ 150.0
…
```

Program 5 is used to tell the malicious nodes what to do. To perform this, we used *rt_resolve(Packet *p)* function, which is used to select the next node when routing data packets. Therefore, in our case we tell the malicious node to drop the whole data packet, which we call it as malicious packet drop attack when it receives data packets from other nodes.

Program 5: code fragment to tell the malicious node to drop data packets

```
/* AODV Route Handling Functions */
 void
     AODV::rt_resolve(Packet *p) {
 struct hdr_cmn *ch = HDR_CMN(p);
     struct hdr_ip *ih = HDR_IP(p);
          aodv_rt_entry *rt;
          …
     // if there is malicious node
     if (malicious_node == true ) {
 drop(p, DROP_RTR_ROUTE_LOOP);
          …
          }
```

REFERENCES

[1] Chlamtac, Imrich, Marco Conti, and Jennifer J-N. Liu. "Mobile ad hoc networking: imperatives and challenges." *Ad Hoc Networks* 1, no. 1 (2003): 13-64.

[2] Xu, Wei-Qiang, and Tie-Jun Wu. "TCP issues in mobile ad hoc networks: Challenges and solutions." *Journal of Computer Science and Technology* 21, no. 1 (2006): 72-81.

[3] Al Hanbali, Ahmad, Eitan Altman, and Philippe Nain. "A survey of TCP over ad hoc networks." IEEE Communications Surveys & Tutorials 7, no. 3 (2005): 22-36.

[4] Wang, Feng, and Y. O. N. G. G. U. A. N. G. Zhang. "A Survey on TCP over Mobile Ad-Hoc Networks." *Nova Science Publishers* (2005): 267-281.

[5] Haniza N., Zulkiflee M., Abdul S."Congestive Loss in Wireless Ad hoc Network" World of Computer Science and Information Technology Journal (WCSIT) ISSN: 2221-0741 Vol. 1, No. 6.pp 269-273, 2011.

[6] Aleksandar Milenkoski and Biljana Stojcevska. "Loss Differentiation Algorithms Vs Congestion Control Schemes: Dynamics and Performance." International Journal of Distributed and Parallel systems (IJDPS) Vol.1, No.1, September 2010.

[7] Mohamed Tekala and Robert Szabo. "Modeling Scalable TCP friendliness to NewReno TCP." IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.3, March 2007.

[8] Ghanem, Tamer F., Wail S. Elkilani, and Mohiy M. Hadhoud. "Improving TCP performance over Mobile Ad Hoc Networks using an adaptive backoff response approach." In *Networking and Media Convergence, 2009. ICNM 2009. International Conference on*, pp. 16-21. IEEE, 2009.

[9] Seddik-Ghaleb, Alaa, Yacine Ghamri-Doudane, and S-M. Senouci. "TCP WELCOME TCP variant for Wireless Environment, Link losses, and COngestion packet loss ModEls." In *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*, pp. 1-8. IEEE, 2009.

[10] Habbal, Adib M. Monzer, and Suhaidi Hassan. "Loss Detection and Recovery Techniques for TCP in Mobile Ad Hoc Network." In *Network Applications Protocols and Services (NETAPPS), 2010 Second International Conference on*, pp. 48-54. IEEE, 2010.

[11] Yang, Hao, James Shu, Xiaoqiao Meng, and Songwu Lu. "SCAN: self-organized network-layer security in mobile ad hoc networks." *Selected Areas in Communications, IEEE Journal on* 24, no. 2 (2006): 261-273.

[12] Soufiene Djahel, Farid Na¨ıt-abdesselam, and Zonghua Zhang. “Mitigating Packet Dropping Problem in Mobile Ad Hoc Networks: Proposals and Challenges.” Communication surveys and tutorials, IEEE 2010. pp 1-5.

[13] Djahel, Soufiene, Farid Nait-abdesselam, and Zonghua Zhang. "Mitigating packet dropping problem in mobile ad hoc networks: Proposals and challenges." *Communications Surveys & Tutorials, IEEE* 13, no. 4 (2011): 658-672.

[14] Mzrak, Alper T., Stefan Savage, and Keith Marzullo. "Detecting malicious packet losses." *Parallel and Distributed Systems, IEEE Transactions on* 20, no. 2 (2009): 191-206.

[15] Henock Mulugeta, Dereje H/mariam, K. Raimond “TCP-PLDR: A Novel Packet Loss Detection and Response Mechanism for TCP in Mobile Ad Hoc Networks.” *Ad Hoc & Sensor Wireless Networks, An International Journal (AHSWN).*

[16] Henock M., Dereje H.M, and Kumudha R. (2014/15) “Investigating the Effects of Security Attacks on the Performance of TCP Variants and Routing Protocols in MANET”, *Int. J. Computer Applications in Technology*.

[17]Henock Mulugeta, and Kumudha Raimond. "Performance of TCP variants over proactive and reactive routing protocols in MANET." *In Proceedings of the International Conference on Management of Emergent Digital EcoSystems (ACM MEDES’2012)*, pp. 123-130. ACM, 2012.

[18] Henock Mulugeta, and Kumudha Raimond. "Performance improvement of TCP using TCP-DOOR-TS algorithm in mobile ad hoc networks." *13th IEEE International Conference on Communication Technology (IEEE ICCT '11).pp. 642-646. pp. 642-646. 2011.(China)*

[19] Henock Mulugeta, and Kumudha Raimond. "Performance of TCP variants over proactive and reactive routing protocols in MANET." *Journal of the Ethiopian Society of Electrical Engineers. 6th Scientific Conference on Electrical Engineering (CEE -2012)*. 5-6, October 2012.

[20] Lochert, Christian, Björn Scheuermann, and Martin Mauve. "A survey on congestion control for mobile ad hoc networks." *Wireless Communications and Mobile Computing* 7, no. 5 (2007): 655-676.

[21] Sengottaiyan, N.; Somasundaram, R.; Arumugam, S. “A Modified approach for measuring TCP Performance in Wireless Ad hoc Network” International Conference on Advances in Recent Technologies in Communication and Computing. IEEE 2010. pp. 267-270.

[22]Dhananjay Bisen, Sanjeev Sharma. “Improved Performance of TCP-Newreno over Mobile Ad hoc Networks using ABPRA.” International Journal of Wireless & Mobile Networks (IJWMN) Vol. 3, No. 2, April 2011.

[23] Haniza N., Zulkiflee M., Abdul S.” Congestive Loss in Wireless Ad hoc Network” World of Computer Science and Information Technology Journal (WCSIT) ISSN: 2221-0741 Vol. 1, No. 6.pp 269-273, 2011.

[24] Di Felice, Marco, Kaushik Roy Chowdhury, and Luciano Bononi. "Modeling and performance evaluation of transmission control protocol over cognitive radio ad hoc networks." In *Proceedings of the 12th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, pp. 4-12. ACM, 2009.

[25] Lakshman, T. V., and Upamanyu Madhow. "The performance of TCP/IP for networks with high bandwidth-delay products and random loss." *Networking, IEEE/ACM Transactions on* 5, no. 3 (1997): 336-350.

[26] Jacobson, Van. "Berkeley tcp evolution from 4.3-tahoe to 4.3-reno." In *Proceedings of the 18th Internet Engineering Task Force, University of British Colombia, Vancouver, BC*, p. 365. 1990.

[27] Sikdar, Biplab, Shivkumar Kalyanaraman, and Kenneth S. Vastola. "Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno, and SACK." *Networking, IEEE/ACM Transactions on* 11, no. 6 (2003): 959-971.

[28] Kim, Dongkyun, Hanseok Bae, Jeomki Song, and J-C. Cano. "Analysis of the interaction between TCP variants and routing protocols in MANETs." In *Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on*, pp. 380-386. IEEE, 2005.

[29] Wierman, Adam, and Takayuki Osogami. "A unified framework for modeling TCP-Vegas, TCP-SACK, and TCP-Reno." In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pp. 269-278. IEEE, 2003

[30] Floyd, Sally, Tom Henderson, and Andrei Gurtov. *The NewReno modification to TCP's fast recovery algorithm*. RFC 2582, April, 1999.

[31] Parvez, Nadim, Anirban Mahanti, and Carey Williamson. "An analytic throughput model for TCP NewReno." *Networking, IEEE/ACM Transactions on* 18, no. 2 (2010): 448-461.

[32] Othman, Mazliza, and May Zin Oo. "Analysis of TCP-Reno and TCP-Vegas over AOMDV routing protocol for mobile ad hoc network." In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, vol. 2, pp. 1104-1108. IEEE, 2010.

[33] Papanastasiou, Stylianos, and Mohamed Ould-Khaoua. "Exploring the performance of TCP Vegas in Mobile Ad hoc Networks." *International Journal of Communication Systems* 17, no. 2 (2004): 163-177.

[34] Brakmo, Lawrence S., and Larry L. Peterson. "TCP Vegas: End to end congestion avoidance on a global Internet." *Selected Areas in Communications, IEEE Journal on* 13, no. 8 (1995): 1465-1480.

[35] Shrestha, Ashish, and Firat Tekiner. "On MANET routing protocols for mobility and scalability." In *Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on*, pp. 451-456. IEEE, 2009.

[36] Barakovic, Sabina, and Jasmina Barakovic. "Comparative performance evaluation of Mobile Ad Hoc routing protocols." In *MIPRO, 2010 Proceedings of the 33rd International Convention*, pp. 518-523. IEEE, 2010.

[37] Fengying, Xu, Liu Zhimin, and Xu Yongchun. "A Comparative Study on Mobile Ad Hoc Wireless Network Routing Protocols Simulation." In *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, pp. 1-4. IEEE, 2010.

[38] Mbarushimana, C., and A. Shahrabi. "Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks." In *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 2, pp. 679-684. IEEE, 2007.

[39] Jayakumar, Geetha, and G. Gopinath. "Ad hoc mobile wireless networks routing protocols–a review." *Journal of Computer science* 3, no. 8 (2007): 574-582.

[40] Royer, Elizabeth M., and Chai-Keong Toh. "A review of current routing protocols for ad hoc mobile wireless networks." *Personal Communications, IEEE* 6, no. 2 (1999): 46-55.

[41] Perkins, Charles E., and Pravin Bhagwat. "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers." *ACM SIGCOMM Computer Communication Review* 24, no. 4 (1994): 234-244.

[42] Johnson, David B., David A. Maltz, and Josh Broch. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks." *Ad hoc networking* 5 (2001): 139-172.

[43] Das, Samir R., Charles E. Perkins, and Elizabeth M. Royer. "Performance comparison of two on-demand routing protocols for ad hoc networks." In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 3-12. IEEE, 2000.

[44] Marina, Mahesh K., and Samir R. Das. "On-demand multipath distance vector routing in ad hoc networks." In *Network Protocols, 2001. Ninth International Conference on*, pp. 14-23. IEEE, 2001.

[45] C. E. Perkins, E. M. Belding-Royer, and S. R. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*. Request For Comments, http://www.ietf.org/rfc/rfc3561. txt , July 2003. Experimental RFC.

[46] Park, Vincent, and M. Scott Corson. *Temporally-ordered routing algorithm (TORA) version 1 functional specification*. Internet-Draft, draft-ietf-manet-tora-spec-00. txt, 1997.

[47] Park, Vincent Douglas, and M. Scott Corson. "A highly adaptive distributed routing algorithm for mobile wireless networks." In *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 1405-1413. IEEE, 1997.

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

[49] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Proceedings of ACM MobiCom'99*, Seattle, Washington, Aug. 1999.

[50] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A feedback-based scheme for improving TCP performance in ad hoc wireless networks," IEEE Pers. Commun., vol. 8, no. 1, Feb. 2001, pp. 34–39.

[51] Karthikeyan Sundaresan,y Vaidyanathan Anantharaman,x HungYun Hsieh,y and Raghupathy Sivakumary ," ATP: A Reliable Transport Protocol for Adhoc Networks" in Proceedings of the 4th ACM international symposium on Mobile Ad Hoc networking & computing 2003, Annapolis, Maryland, USA June-2003 pp.64-75.

[52] Buddha Singh, Adwitiya Sinha, and Silky Makker. "Modified ATP for Lossy Links in Mobile Adhoc Network". IEEE Region 8 SIBIRCON-2010. pp. 339-344.

[53] T. Dyer and R. Boppana, "A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks," Proc. ACM MOBIHOC, Long Beach, CA, USA, 2005, pp.56-66.

[54]H. Touati, I. Lengliz, and F. Kamoun, "Performance of TCP Adaptive RTO in ad-hoc networks based on different routing protocols," in Mobile Wireless

Communications Networks, 2007 9th IFIP International Conference on, 2007, pp. 176-180.

[55] F. Wang and Y. Zhang, “Improving TCP performance over mobile ad hoc networks with out-of-order detection and response,” Jun. 2002, pp. 217–225.

[56]Tamer F. Ghanem, Wail S. Elkilani, Mohiy M. Hadhoud. “Improving TCP Performance over Mobile Ad Hoc Networks Using an Adaptive Backoff Response Approach”. 2009 IEEE. pp. 16-21.

[57] Haejung , Kaixin Xu, Mario Gerla, “ TCP performance over Multi path routing in mobile ad hoc Networks” Samsung Electronics,Seoul, Korea, Computer Science Department, UCLA Los Angeles, CA 90095, USA. 2003 IEEE.

[58]Song Cen, Pamela C. Cosman, and Geoffrey M. Voelker.” End-to-end differentiation of congestion and wireless losses.” ACM/IEEE Transactions On Networking.

[59] Mohammad Amin Kheirandish Fard1Sasan Karamizadeh. “Enhancing Congestion Control to Address Link Failure Loss over Mobile Ad hoc Networks.” International Journal of Computer Networks & Communications (IJCNC) Vol.3, No.5, Sep 2011.

[60] Nadim Parvez, Anirban Mahanti. “An Analytic Throughput Model for TCP NewReno. IEEE/ACM Transactions on Networking, vol. 18, no. 2, April 2010.

[61] B. Sikdar, S. Kalyanaraman, and K. Vastola, “Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno and SACK,” IEEE/ACM Trans. Network., vol. 11, no. 6, pp. 959–971, Dec. 2003.

[62] M. M. Hassani. “An analytical model for evaluating utilization of TCP TAHOE using markovian model”. In IEEE International Conference on Networking Architecture and Storage, 2007.

[63] T-L. Sheu and L-W.Wu.”An analytical model of fast retransmission and recovery in TCP-SACK. Performance Evaluation.” 64:524–546, July 2007.

[64] Mathew Mathis, Jeffery Semeky, Jamshid Mahdevi. “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm.”ACM Computer Communication Review. Volume 27, no3, July 1997.

[65] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. “Modeling TCP Throughput: A Simple Model and its Empirical Validation.” In ACM SIGCOMM’98, pages 303–314, 1998.

[66] Arora, Vasudha, and C. Rama Krishna. "Performance evaluation of routing protocols for MANETs under different traffic conditions." In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 6, pp. V6-79. IEEE, 2010.

[67] Kim, Dongkyun, J-C. Cano, P. Manzoni, and C-K. Toh. "A comparison of the performance of TCP-Reno and TCP-Vegas over MANETs." In *Wireless Communication Systems, 2006. ISWCS'06. 3rd International Symposium on*, pp. 495-499. IEEE, 2006.

[68] Tahiliani, Mohit P., K. C. Shet, and T. G. Basavaraju. "Performance evaluation of TCP variants over routing protocols in multi-hop wireless networks." In *Computer and Communication Technology (ICCCT), 2010 International Conference on*, pp. 387-392. IEEE, 2010.

[69] Bhanumathi, V., and R. Dhanasekaran. "TCP variants-A comparative analysis for high bandwidth-delay product in mobile adhoc network." In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 2, pp. 600-604. IEEE, 2010.

[70] Andel, Todd R., and Alec Yasinsac. "Surveying security analysis techniques in MANET routing protocols." *IEEE Communications Surveys & Tutorials* 9, no. 4 (2007): 70-84.

[71] M. Stojanovic, V. Acimovic-Raspopovic. “The Impact of Mobility Patterns on MANET Vulnerability to DDoS Attacks.” Telecommunication engineering. IEEE. 2012. No.3 (119).

[72] Hu, Yih-Chun, Adrian Perrig, and David B. Johnson. "Rushing attacks and defense in wireless ad hoc network routing protocols." In *Proceedings of the 2nd ACM workshop on Wireless security*, pp. 30-40. ACM, 2003.

[73] Yi, Ping, Yong-kai Zhou, Yue Wu, and Ning Liu. "Effects of denial of service attack in mobile ad hoc networks." *Journal of Shanghai Jiaotong University (Science)* 14, no. 5 (2009): 580.

[74] Xing, Fei, and Wenye Wang. "Understanding dynamic denial of service attacks in mobile ad hoc networks." In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pp. 1-7. IEEE, 2006.

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

[76] Gurtov, Andrei, and Sally Floyd. "Modeling wireless links for transport protocols." *ACM SIGCOMM Computer Communication Review* 34, no. 2 (2004): 85-96.

[77] McDougall, Jeff, and Scott Miller. "Sensitivity of wireless network simulations to a two-state Markov model channel approximation." In *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, vol. 2, pp. 697-701. IEEE, 2003.

[78] Konrad, Almudena, Ben Y. Zhao, Anthony D. Joseph, and Reiner Ludwig. "A Markov-based channel model algorithm for wireless networks." *Wireless Networks* 9, no. 3 (2003): 189-199.

[79] Haßlinger, Gerhard, and Oliver Hohlfeld. "Analysis of random and burst error codes in 2-state Markov channels." In *Telecommunications and*

Signal Processing (TSP), 2011 34th International Conference on, pp. 178-184. IEEE, 2011.

[80] Todd R. Andel, Alec Yasinsac."Surveying Security Analysis Techniques in MANET Routing Protocols".IEEE Communications, Surveys. 4th quarter 2007, Volume 9, No.4.

[81] M. Stojanovic, V. Acimovic-Raspopovic. "The Impact of Mobility Patterns on MANET Vulnerability to DDoS Attacks." Telecommunication engineering. IEEE. 2012. No.3 (119).

[82] Ashish Shrestha, Firat Tekiner. "On MANET Routing Protocols for Mobility and Scalability." Parallel and distributed computing applications and technologies. 2009 IEEE. pp(S) 451-456.

[83] Sabina Barakovi, Jasmina Barakovic. "Comparative Performance Evaluation of Mobile Ad Hoc Routing Protocols." MIPRO 2010, proceeding of the 33rd international convention.IEEE 2010. Pp(s) 518- 523.

[84] Imad Aad, eanPierre Hubaux." Denial of Service Resilience in Ad Hoc Networks." MobiCom'04, Sept. 26Oct. 1, 2004, Philadelphia, Pennsylvania, USA. ACM digital library. 2004.

[85] Aleksandar Kuzmanovic, Edward W. Knightly." LowRate TCP Targeted Denial of Service Attacks". SIGCOMM'03, August 25– 29, 2003, Karlsruhe, Germany. ACM digital library. 2003.

[86] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links ," *Proceedings of ACM MOBICOM*, 2001.