ENHANCING TCP PERFORMANCE IN MOBILE AD HOC NETWORK USING EXPLICIT LINK FAILURE NOTIFICATION (ELFN)

RAAID N. ALABAEDY

UNIVERSITY UTARA MALAYSIA
2012

ENHANCING TCP PERFORMANCE IN MOBILE AD HOC NETWORK USING EXPLICIT LINK FAILURE NOTIFICATION (ELFN)

A project submitted to Dean of Awang Had Salleh Graduate School in Partial Fulfilment of the requirement for the degree Master of Science of Information Technology University Utara Malaysia

By
RAAID N. ALABAEDY

PERMISSION TO USE

In presenting this project in partial fulfilment of the requirements for a postgraduate degree from the University Utara Malaysia, I agree that the University Library may make it freely available for inspection. I further agree that permission for copying of this project in any manner in whole or in part, for scholarly purposes may be granted by my supervisor(s) or in their absence by the Dean of Postgraduate Studies and Research. It is understood that any copying or publication or use of this project or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to University Utara Malaysia for any scholarly use which may be made of any material from my project.

Requests for permission to copy or to make other use of materials in this project, in whole or in part, should be addressed to

Dean of Awang Had Salleh Graduate School

College of Arts and Sciences

University Utara Malaysia

06010 UUM Sintok

Kedah Darul Aman

Malaysia

ABSTRACT

The dynamics and the unpredictable behaviour of a wireless mobile ad hoc network results in the hindrance of providing adequate reliability to network connections. Frequent route changes in the network relatively introduce incessant link failures which eventually degrade TCP performance considerably. In this research, we are going to study the potential improvement of TCP performance when Explicit Link Failure Notification is implemented as opposed to the standard TCP mechanism. ELFN modifies the 'slow start' mechanism that is used in standard TCP so that the throughput achieved from the network can be maximized.

ACKNOWLEDGMENTS

In the Name of ALLAH, the Most Gracious and the Most Merciful. Peace is upon to Muhammad S.A.W., the messenger sending to guide people for the truth way.

First, all praises and thanks goes to almighty ALLAH for giving me the patience, the health and the guidance in completing this thesis successfully as well as giving me the chance to work in such an environment in Malaysia and in UUM in particular.

Second, the first person I would like to express my deep and sincere gratitude to is my supervisor Dr. Mohammed M. Kadhum, School of Computer Sciences, Universiti Utara Malaysia. His wide knowledge and his logical way of thinking have been of great value to me. His supports, motivations, guidance and incisive advice have inspired me to generate fruitful approaches in achieving the objective in this research. Furthermore, I would like to thank my evaluator Dr. Edy Santoso to evaluate this work and give my very important nodes. Also, would like to thank the School of Computer Sciences (lecturers and staff), Universiti Utara Malaysia for supporting this study and for giving me an opportunity to express my ideas and findings in the form of a dissertation.

Third, this thesis is especially dedicated to my beloved parents for their prey, love and encouragement to see their son succeed. To my brothers and sisters, I love you all. A special thank to my master and PhD friends, thank you for struggler, laughter and share together with me before we finally reach the final steps. , Also, I

would like to record a special word of thanks for the support given by Mr. Adib M. Monzer Habbaland all lecturers; I know them during my study.

Last but not least, I especially wish to express my love for my wife who did not only endure my manifold activities but also provided inspiration and support for my inclination to perfectionism and my kids (AliAldur and Ratij), I love you all.

Regards

Raaid N. Alabaedy

2012

CONTENTS

LIST OF FIGURES

Chapter one

Chapter two

Chapter three

Chapter Five

LIST OF TABLES

APPENDIX

LIST OF ABBREVIATION

ACK	Acknowledgement
AODV	Adhoc On-demand Distance Vector
BDP	Bandwidth-Delay Product
BIC	Binary Increase Congestion
BS	Base Station
cwnd	Congestion Window
DSR	Dynamic Source Routing
ECN	Explicit Congestion Notification
ELFN	Explicit Link Failure Notification
GPPL	General Purpose Programming Languages
GUI	Graphical User Interface
IP	Internet Protocol
IPSEC	Internet Protocol Security
IPv6	Internet Protocol Version 6
LL	Link Layer
MAC	Media Access Control
MANET	Mobile Ad Hoc Network

M-TCP	Mobile TCP
NRL	Normalized Routing Load
NS	Network Simulator
PSL	Plain Simulation Language
RREP	Route Reply
RREQ	Route Request
RTT	Round Trip Time
rwnd	Receiver Window
SACK	Selective Acknowledgment
SP	Simulation Packages
TCL	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol

CHAPTER ONE

INTRODUCTION

Mobile ad hoc network has gained a lot of attention in recent years due to its dynamic characteristics and its self-governing behaviour which does not require a fixed infrastructure (Abduljalil et al., 2006). Numerous ongoing research are focusing on routing protocols for example (Mittal, 2009), (Runcai et al, 2009), (Maan et al., 2011) and (Qian et al., 2009). In this project, we are focusing towards the performance of TCP in mobile ad hoc networks.

It is unavoidable to use TCP in mobile ad hoc network taking into account the applications and services it can provide to the network users. Hence, this project is meant to bring forward a specific issue related to TCP congestion control mechanism which is modified so that it would give a better performance in mobile ad hoc network.

1.1. Introduction

The popularity of wireless network has been growing steadily. Wireless ad hoc networks have been popular because they are very easy to implement without using base stations. The wireless ad hoc networks are complex distributed systems that consist of wireless mobile or static nodes that can freely and dynamically self-organize (Jain, et at., 2002). The ad hoc networks allow nodes to seamlessly communicate in an area with no pre-existing infrastructure. Future advanced technology of ad hoc network will allow the forming of small ad hoc networks on campuses, during conferences and even in homes. Furthermore, there is an increasing need for easily portable ad hoc networks in rescue

mission, especially for accessing rough terrains. However, the quick adaptation and ease of configuration of ad hoc networks come at a price. In wireless ad hoc networks, route changes and network partitions occur frequently due to the unconstrained network topology changes. Moreover, this kind of network inherits the traditional problems of wireless communication, such as unprotected outside signals or interferences, unreliable wireless medium, asymmetric propagation properties of wireless channel, hidden and exposed terminal phenomena, transmission rate limitation and blindly invoking congestion control of transport layer. Although most of these limitations and complexities are due to the lack of fixed backbone or infrastructure, building ad hoc network temporarily is not only simple and easy to implement but also cost-effective and less time-consuming if compared to an infrastructure network that needs to establish a based station and fixed backbone. Among the above mentioned problems and limitations, the impact of transport layer limitations is analyzed across ad hoc routing protocols throughout the network topologies.

Transmission Control Protocol (TCP) (Postel, 1981) is the de facto standard designed to provide reliable end-to-end delivery of data packet in the wired networks. Normally, TCP is an independent protocol that is not related to the underlying network technology. However, some assumptions of TCP, such as consideration of only static node, packet losses due to congestion or buffer overflows are inspired from the features of wired networks. In the wireless network, these assumptions may not be correct all the time due to the rapid network topology changes, node movements and limited battery power. In order to apply TCP to an ad hoc environment, TCP has to overcome many problems, such as packet losses due to congestion, high bit errors, node mobility, longer delay and so on. The following TCP versions, Tahoe (Stevens, 1997), Reno (Allman, 1999), NewReno (Floyd & Fall, 1999), Vegas (Brakno et al., 1994) and Westwood (Gerla et al., 2002), are enhanced

versions of TCP and perform differently depending on how the routing protocols can quickly adapt route changes due to link breaks in an ad hoc network environment.

1.2. IEEE 802.11 Challenges

The unique characteristics of ad-hoc designs impose several challenges in comparison with single hop networks such as cellular networks or WLANs, when they run over 802.11 MAC protocol. The most serious challenge is the RTS/CTS handshaking implemented in 802.11, which is not efficient enough to prevent collisions due to large distribution of mobile nodes and multi-hop function in ad-hoc networks. It has been proved through analytical model and simulation experiments (Fu, et al., 2005) that RTS/CTS cannot function well in topologies more than three hops (3 hop scenario) between sender and receiver. For larger number of hops, the RTS/CTS exchange cannot prevent the existence of famous hidden node problem. Mobile nature of ad-hoc networks, where each node may experience different degree of channel contention and collision, is another problem, (Zhai et al., 2006). The interaction between the MAC and higher layers has a significant effect on the network performance.

1.3. Medium Contention and Spatial Reuse

The Hidden and Exposed terminals are defined based on transmission range and sensing range of the nodes. Transmission range represents the range within which a packet is successfully received if there is no interference from other radios. Sensing range is the range within which a transmitter triggers carrier sense detection to sense an ongoing signal. Spatial reuse facilitates maximum possible non-conflicting simultaneous transmissions in MAC layer. A hidden terminal is the one that can neither sense the transmission of a

transmitter nor correctly receive the reservation packet (i.e. CTS control frame) from its corresponding receiver (Zhai, et al., 2006). In other words, a hidden terminal is a node that is within the transmission range of a receiver but out of the sensing range of an intended transmitter. Therefore, it can interfere with an ongoing transmission at the receiver by transmitting at the same time. Consider the scenario illustrated in Figure 1.1 to see the cause of hidden nodes. Here node D is a hidden terminal while B is transmitting to C because it is out of B's sensing range. Therefore, D's transmission collides with RTS reception in C. After seven attempts, both B and D assume that C is unreachable and the packets will be dropped (Armaghani & Jamuar, 2008), (Zhai, et al., 2006).This leads to bandwidth wastage as both data transmissions are destroyed.

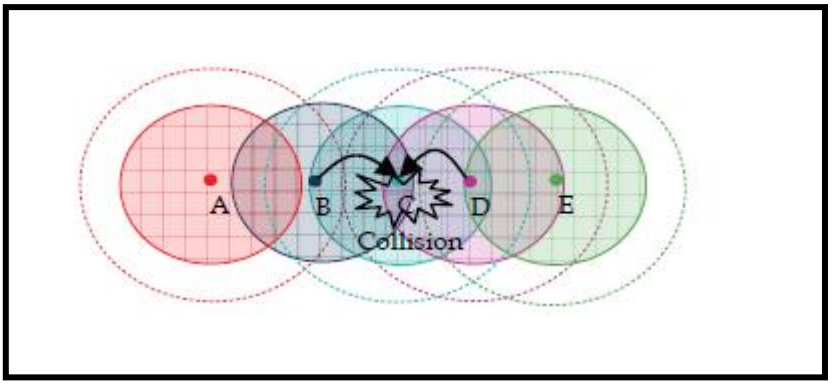

Figure1.1 Hidden Terminal (Armaghani & Jamuar, 2008)

1.4. Problem Statements

The history of TCP begins when reliability in the network has become an important aspect of a network performance and highly demanded. TCP is used mainly for wired network in which frequent congestion is expected and congestion avoidance mechanism belongs to TCP such as slow start and exponential back off is exercised. Relatively, when TCP is being used in wireless environment, it still conceives that congestion is the cause of packet loss, not bit errors (Nehme, Phillips, & Robertson, 2003), (Stangel & Bharghavan, 1998).

This confusion, will then temper the network performance as TCP will apply the congestion control mechanism whenever it is experiencing packet loss in the network. The congestion control mechanism in TCP will slows down the transfer rate unreasonably causing the network to fall into a condition in which it is under utilizing the available bandwidth (Chiasserini & Meo, 2001).

1.5. Research Questions

i. How can we improve the performance of TCP given the dynamics of mobile ad hoc networks?

ii. Will TCP with Explicit Link Failure Notification (ELFN) increase the network throughput?

1.6. Research Objectives

The main objective of this research is to enhance the performance of TCP in mobile ad hoc network by incorporating Explicit Link Failure Notification into TCP's mechanism. In relation to this, other related objectives that worth pointing out are as follows:

i. To implement the Explicit Link Failure Notification (ELFN) mechanism into TCP using NS-2.

ii. To study the impact of using Explicit Link Failure Notification (ELFN) with TCP in mobile ad hoc network.

iii. To compare the performance of TCP in mobile ad hoc network with and without the implementation of Explicit Link Failure Notification (ELFN).

1.7. Research Scope

The scope of this study includes the implementation of Explicit Link Failure Notification into the standard TCP mechanism currently being used in wireless networks. The implementation of this mechanism into TCP will then be evaluated using three significant performance metrics which are listed below:

- Packet Loss
- End to end delay
- Normalized Routing Load

The scope of this research also covers the evaluation of the modified TCP performance under several wireless ad hoc network conditions presented below:

- Nodes density
- Speed of travelling nodes

1.8. Research Significance

The main significance of this research is to improve performance of TCP by adding a mechanism called Explicit Link Failure (ELFN). This mechanism is added into TCP which is then implemented in a wireless ad hoc network.

The standard TCP was built and was optimized on a wired network, hence there are some modification needs to be done, in this case, the congestion control mechanism, in order to enhance the TCP performance in wireless ad hoc network.

1.9. Organization of the Project Report

This project is organized in six chapters as follows:

Chapter 1 presents a brief background and introduction to TCP and its enhanced version that is using Explicit Link Failure Notification (ELFN). The chapter also presents the research problem, scope, objectives, and contributions of this project as well.

Chapter 2 is a literature review that includes a background material on TCP and its related characteristics. This chapter will also present the general framework for this project. The TCP enhancement using Explicit Link Failure Notification (ELFN) will also be presented in this chapter.

Chapter 3 presents the methodology followed in studying the performance of the TCP using Explicit Link Failure Notification (ELFN) when it is being simulated in a mobile ad hoc network. It covers network topology and settings used in the experiments.

Chapter 4 presents the development simulation settings for the performance evaluation of the TCP with Explicit Link Failure Notification (ELFN) in network simulator 2 (ns-2) using Tool Command Language (TCL).

Chapter 5 presents a detailed performance evaluation of TCP with Explicit Link Failure Notification (ELFN), and discussed the results based on the numerical results obtained from simulations.

Chapter 6 presents the conclusions of the research work presented in this project and provide some suggestions for further research.

CHAPTER TWO

LITERATURE REVIEW

Chapter 1 described generally the wireless issues that cause degradation in the performance of the TCP in mobile ad hoc networks. This chapter provides the background on transmission transport protocol (TCP) that forms the general framework of this research. This chapter presents phases of TCP congestion control, namely Slow Start and Congestion Avoidance. It discusses how the link failure phenomenon affects the performance of TCP in MANET. Also, this chapter describes how ELFN can increase the TCP performance.

2.1. Introduction

As mentioned in Chapter one, It is a requirement for every network to have some form of reliable communication where the delivery of the packets to the destination is guaranteed. For wired networks and static wireless networks Transmission Control Protocol (TCP) is the connection oriented transport layer protocol that guarantees this functionality. It assures in-order delivery of the packet and uses flow control and congestion control mechanisms (Holland & Vaidya, 1999), (Caceres & Iftode, 1995). For ad hoc networks however the standard TCP does not give satisfactory performance. In the ad hoc network the nodes are traveling and there are no base stations. In other words the topology of the network is constantly changing. The communication between the sender and receiver nodes occur through other nodes in the network and each of the intermediate nodes is acting as a router for the communication. The connection can have multiple hops. This causes performance

losses due to the high error rate, network congestion and possible connection failure (Xu & Saadawi, 2001).

2.1. Transmission Control Protocol (TCP)

The most well-known common transport protocol is the Transmission Control Protocol (TCP) (Postel, 1981). It lent its name to the title of the entire Internet Protocol Suite, TCP/IP. TCP is the more complex protocol, due to its stateful design incorporating reliable transmission and data stream services.

According to Jon Postel, Transport Control Protocol (TCP) has known to be a reliable transport protocol with congestion control for delivering data traffic. TCP can deliver best-effort service for error-intolerant and delay-tolerant data such as web, email, file transport, etc. All those features of TCP make it suitable for the delivery of important, mission critical, and error-free data which require a reliable data connection. TCP is not well suited for streaming media due to its reliable in-order delivery and congestion control that can cause random long delays (Aditya & Anurag, 2005). Its reliable in-order delivery mechanism has to retransmit the packet if there is a packet loss happen during transmission in the network (Caceres & Iftode, 1995), (Eshak & Baba, 2003). Packet loss in TCP is detected either by time out or three duplicated acknowledgments received by sender from the network. For this reason, if the transmission of streaming media is affected so much by delays, it can affect the quality of service provided to end users. Hence, TCP is only suitable for applications that rely on the reliability and can tolerate the delay like traditional web applications, file transfer and email (Yousefi'zadeh, Habibi, & Furmanski, 2006).

Most real-time traffic ranges from interactive applications such as Voice over Internet Protocol (VoIP) and video conferencing to non-interactive applications like audio and

video streaming commonly use unreliable transport protocol (UDP) as their transport protocol. UDP provides the best transport platform to deliver error-tolerant and delay-intolerant traffic. This is due to some features of UDP such as simpler connectionless implementation, shorter packet header, no congestion control, no acknowledgment, no retransmission, and etc.

Even UDP can serve real-time multimedia traffic very well; there is a friendliness issue with other transport protocol like TCP which delivers reliable best-effort service for error-intolerant and delay-tolerant data like World Wide Web, email, file transport, etc. In competing with TCP traffic when there is bandwidth restriction, UDP traffic consumes more and dominates all the bandwidth, and as a result, TCP traffic will be halted. The same thing can happen when competing with other lower bandwidth applications, such as wireless link, in which real-time traffic utilizing UDP would consume 100% of bandwidth link utilization.

As mentioned in (Chydzinski & Brachman, 2010), nowadays, TCP Newreno is the most commonly implemented algorithm while TCP SACK support is very common and is an extension to TCP Reno and TCP New Reno. On the other hand, TCP CUBIC is a less aggressive and more systematic derivative of TCP BIC, in which the window is a cubic function of time since the last congestion event, with the inflection point set to the window prior to the event. According to Y. Iwanaga et al. (Y. Iwanaga, 2010), most Linux operating systems use TCP CUBIC (Sangtae Ha, 2008) by default, while they natively support many other TCP variants.

TCP header is shown in Figure 2.1. It consists of many fields such as source port, destination port, sequence number, acknowledgment number, and window etc. the value

in the “window” field determines the congestion window in a 16-bit field used to control the congestion in the Internet (Mahbub & Raj, 2004).

<table>
<tr><td colspan="3">Source port (16)</td><td>Destination port (16)</td></tr>
<tr><td colspan="4">Sequences (32)</td></tr>
<tr><td colspan="4">Acknowledgment number (32)</td></tr>
<tr><td>Header length</td><td>Reserved (6)</td><td>Flags (6)</td><td>Receiver window size</td></tr>
<tr><td colspan="3">Checksum (16)</td><td>Urgent pointer data (16)</td></tr>
<tr><td colspan="4">Options</td></tr>
<tr><td colspan="4">Application data (variable length)</td></tr>
</table>

Figure 2.1 TCP Header (Mahbub & Raj, 2004)

TCP has received a lot of attention and fairly large number of researchers has tried to optimize and improve TCP for different environments characterized by heterogeneous sub networks with widely different bandwidths and latencies (for instance TCP over wireless links, satellite links, slow serial links, etc.).

Experimental and analytical studies (Altman, et al. 2000) confirm that the current TCP protocols have performance problems in networks with long propagation delay or long delay link and relatively high link error rates, such as satellite networks (Durresi, et al. 2001), (Seungwan & Chulhyoe, 2004). From the view of TCP, the throughput is reciprocal to the RTT of a connection, and it is approximately proportional to the congestion window (cwnd) which represents the amount of unacknowledged data sender can have in transit (Zhang, 1986). As a solution for the use of TCP over long delay link, some approaches have been introduced as solutions, such as TCP-Peach and TCP-Hybla (Saad & Nitin, 2005).

2.2. Phases of TCP Congestion Control

In TCP congestion control, there are two phases, i.e. slow-start and congestion avoidance phases. Each phase has its own purpose. The two subsections below describe briefly about slow-start and congestion avoidance phases.

2.2.1. Slow Start

The standard TCP mechanism uses 'slow start' method that temper the ongoing flow of connections in the network (Rung-Shiang et al., 2005), (So-In, Jain, & Dommety, 2009). Figure 2.2 presents the TCP congestion control mechanism.

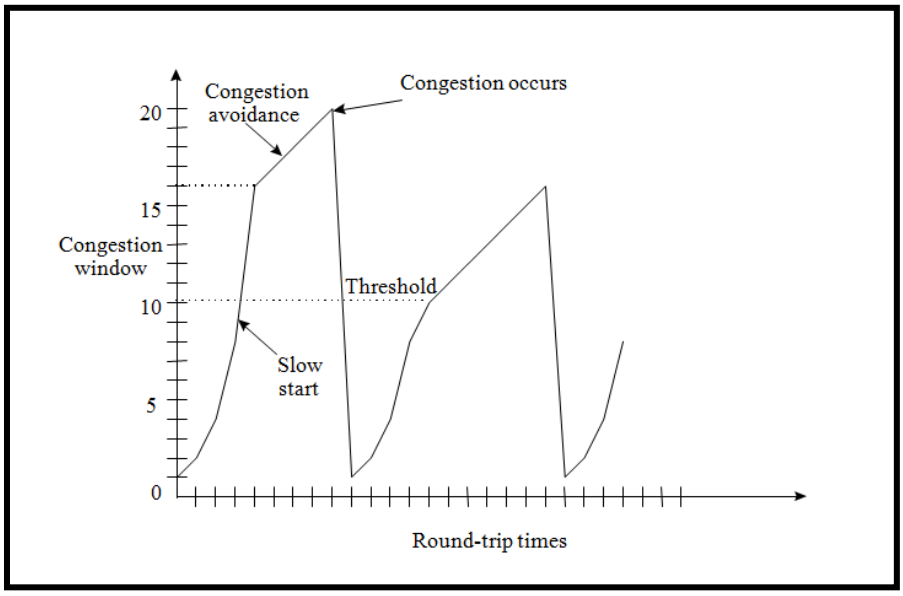

Figure 2.2 TCP Congestion Control Mechanism (So-In, Jain, & Dommety, 2009)

When packets are lost due to network congestion, TCP will enter a 'slow start' phase. In this phase, TCP will sends out one packet and wait for the ACK. If nothing is received and the timer expires, it will send again the same packet. In contrast, if the ACK is received, it will sends out two packets, then it will wait for them to be acknowledged and increase sending out the packet exponentially (Wing-Chung & Law., 2008).

Slow-start is one of the algorithms that TCP uses to control congestion inside the network. It is also known as the exponential growth phase. Slow-start works by increasing the TCP congestion window each time the acknowledgment is received (Minseok & Sonia, 2004). It increases the window size by the number of segments acknowledged. This happens until either an acknowledgment is not received for some segment or a predetermined threshold value is reached. If a loss event occurs, TCP assumes there is a congestion and takes steps to reduce the offered load on the network. Once a loss event has occurred or the threshold has been reached, TCP enters the linear growth in congestion avoidance phase. At this point, the window is increased by one segment for each RTT. This happens until a loss event occurs (Hassan & Raj, 2001).

The algorithm begins in the exponential growth phase initially with a congestion window size of one or two segments and increases it by one segment size for each ACK received. This behavior effectively doubles the window size each round trip of the network. This behavior continues until the congestion window size reaches the size of the receiver's advertised window or until a loss occurs. Over long delay links, the slow-start algorithm consumes too much time (Liang & Matta, 2001). It may take a number of RTTs in slow-start before the connection begins to fully use the available bandwidth of the network. As a solution to reduce the time consumed before achieving the available bandwidth is by using aggressive exponential technique. Aggressive exponential can be used with TCP to request a higher congestion window for long delay link networks (Hassan & Jain, 2004). In using exponential, a host would indicate its desired rate in bytes per second using an Exponential Option in the IP header of a packet. In Exponential, a TCP host, for example host A, would indicate its desired sending rate in bytes per second, using an Exponential Option in the IP header of a TCP packet. Each router along the path could, in turn; either approves the requested rate, reduces the requested rate, or indicates that the Exponential Request is not

approved. In approving an Exponential Request, a router does not give preferential treatment to subsequent packets from that connection; the router is only asserting that it is currently underutilized and believes there is a sufficient available bandwidth to accommodate the sender's requested rate. The Exponential mechanism can determine if there are routers along the path that do not understand the Exponential Option, or have not agreed to the Exponential rate request (Douglas, 2005).

TCP host B communicates the final rate request to TCP host A in a transport- level Exponential Response in an answering TCP packet. If the Exponential Request is approved by all routers along the path, then the TCP host can send at up to the approved rate for a window of data. Subsequent transmissions will be governed by the default TCP congestion control mechanisms of that connection. If the Exponential Request is not approved, then the sender would use the default congestion control mechanisms.

2.2.2. Congestion Avoidance

TCP utilizes the congestion window which can grow or shrink depends on the condition of the network. In normal case, the current congestion window is halved, i.e. it is dropped 50% from the current value during congestion avoidance phase when a congestion event via packet loss is detected. During congestion avoidance phase, packet loss is detected through three duplicate ACKs or ECN marked packets (Archan, 2000). Congestion window is one of the key components in TCP's congestion control, In TCP, congestion window controls the number of packets a TCP flow may have in the network at any time. However, long periods when the sender is idle or application-limited can lead to the invalidation of the congestion window, in that the congestion window no longer reflects current information about the state of the network (Ghazali, 2008).

Basically, congestion window is a parameter in TCP which buffers the packets in the network. In TCP, the congestion detection is done through a mechanism where timeout occurs or three duplicate acknowledgments are received by the sender (Floyd & Fall, 1999).

2.3. TCP Performance over Mobile Ad Ho Networks

The performance of TCP degrades significantly when used over networks that exhibit a large Bandwidth-Delay Product (BDP) or long delay link networks. At any given point in time, TCP's transmission rate cannot exceed the lesser of the congestion window (cwnd) or the receiver's advertised window (rwnd) per Round Trip Time (RTT). As a result, the congestion window size needs to be equal or exceed the BDP before TCP can fully utilize the available bandwidth. However, when a connection is reestablished, TCP does not know what bandwidth is available within the network, hence it does not know what the BDP is and in turn what size the congestion window should be. One or more window growth functions are required to increase the congestion window, ideally to a level that allows all available bandwidth to be utilized, but prevents network resources from becoming oversubscribed. As mentioned before, TCP variants use two algorithms known as slow-start and congestion-avoidance to probe the network or capacity (Postel,1981).

The slow-start algorithm implements exponential window growth, which is designed to start off slowly and probe for network capacity, doubling the congestion window size each one RTT. Once an effective congestion window size has been achieved the congestion-avoidance algorithm continues to probe for additional bandwidth, increasing the congestion window in a linear fashion (Anjum & Jain, 2000). This approach works well when a small congestion window size is required, since slow-start quickly opens the congestion window,

allowing the network capacity to be fully utilized. However, the performance of these algorithms is degraded when the network exhibits a large BDP. Some research in TCP for wireless networks was done to investigate the current transport protocols performance. Current TCP protocols have lower throughput performance in satellite networks, mainly due to the effects of long propagation delay and high link error rates. Both experimental and analytical studies done by Lakshman et al. in (Lakshman, Upamanyu, & Bernhard, 2000) confirmed that the current TCP protocols have performance problems in networks with long propagation delays and relatively high link error rates such as satellite networks. From the view of TCP, the throughput is reciprocal to the RTT of a connection, and is approximately proportional to the congestion window which represents the amount of unacknowledged data the sender can have in transit to the receiver (Jiyong, Daedong, Seongsoo, & Jungkeun, 2011).

In wireless environment, each time the base station (BS) detects a link failure or packet loss, it sends back an ACK to the sender with a zero window size to drive the sender into persist mode, and not drop it's congestion window (Hamrioui & Lalam, 2011). The BS relays ACKs back to the sender only when the receiver has ACKed data in order to maintain end-to-end semantics. This can lead to problems: for instance, assume that the sender has transmitted one widow full of packets and is waiting for ACKs. Suppose the receiver receives them all and ACKs the last transmission (TCP ACKs are cumulative) and then immediately gets disconnected. If the BS relays back the ACK to sender, it will keep transmitting eventually leading to packet loss and congestion window throttling. One could send a duplicate ACK for the last segment, advertising a window size of zero, but such duplicate ACKs may be ignored by the sender (Anjum & Jain, 2000).

Many factors should be taken into account in evaluating any TCP, they are as follows:

- ***Inter-operation with the existing infrastructure***: To realize this goal, ideally, there should not be any change required at intermediate routers or the sender because these are likely to belong to other organizations, making them unavailable formodifications. All approaches that split the connection into two parts require substantial modification and processing at an intermediate node (BS). Some schemes, such as EBSN, also require modification at the sender side. This makes it difficult for these schemes to inter-operate with the existing infrastructure.

- ***Encrypted traffic***: As network security is taken more and more seriously, encryption is likely to be adopted very widely. For instance, IPSEC is becoming an integral part of IPv6, the next generation IP protocol. In such cases the whole IP payload is encrypted, so that the intermediate nodes (be it the base station or another router) may not even know that the traffic being carried in the payload is TCP (Fairhurst, et al., 2001), (Jiwei, et al., 2006). Sometimes data and ACKs can take different paths (for instance, in ADHOC). Schemes based on "intermediary" involvement will have serious problems.

Maintaining true end-to-end semantics: M-TCP does not maintain true end-to-end semantics. But requires a substantial base-station involvement nonetheless. Thus there is a need for true end-to-end signaling without involving any intermediary. If hundreds or thousands of nodes are mobile in the domain of a base station, it could get overwhelmed with the processing of traffic associated with each connection. When a mobile node moves from the domain of one BS to another, the entire "state" of the connection (including any data that was buffered for retransmissions) needs to be handed over to the new base station. This can cause significant amount of overhead and might lead to the loss of some packets and the sender dropping congestion window, which would defeat the original purpose behind the whole endeavor (Ghanem, Elkilani, & Hadhoud, 2009).

2.4. Related Researches

Many Ad hoc routing protocols have been developed and implemented and categorized into different classes.

2.4.1. DSR Routing Protocol

DSR (Johnson, Maltz, & Broch, 2001) was developed by CMU researchers is an on-demand' routing protocol. It employs source routing wherein the source determines the complete sequence of nodes through which a packet is to be routed. The TCP source checks its route cache for a route to the destination, Whenever a TCP source has a packet to transmit, a route request' broadcast is initiated In case no route is found. On receiving this request, each node again broadcasts this request by appending its address to the request packet until this packet reaches the destination. Then the destination sends a route reply to the source containing the route from the source to the destination. When the route reply reaches the source, a connection is established and all subsequent packets contain the complete route in the packet header. DSR will broadcasts 'route error' message when it detects a route failure due to the motion of downstream neighbor. A packet which has no suitable route to its destination temporarily will be reserved in send_buf during the route discovery' period until it is sent out or dropped by DSR for its remaining in send_buf longer than SEND_TIMEOUT.

2.4.2. AODV Routing Protocol

AODV (Kai, Neng, & Ai-fang, 2005) is a popular route protocol for Ad hoc network. This protocol is a reactive protocol. In other words, it uses an on-demand mechanism which means that it discovers routes only when a source node needs them.

It has ability to maintain routes even when the topology of the network is dynamic. Reactive protocol like AODV is suited for MANET as it has low processing and memory overhead and again it minimizes the overall network utilization. Moreover, this protocol uses loop freedom mechanism through the use of sequence number for all routes. The way AODV route request proceed in order to communicate throughout the network or desired destination is as follows: Firstly, the source node initiates route discovery through broadcasting Route Request (RREQ) packet then adjacent nodes will forward RREQ until the packet either is reached at the destination or RREQ arrives at the node that has a new fresh route to the destination. Secondly, a Route Reply (RREP) is sent by receiver to the source (originated route). Once the sender-node receives a RREP, it can begin using this path for data packet transmission. Upon link failure, Route Error (RERR) is sent back to the source node. This important message is generated by the node that the link failure is occurred at. The main difference between DSR-protocol and AODV protocol is that the addresses of the intermediate node are accumulated on the DSR RREQ and RREP control packets. To simplify more the way that DSR network exchanges the information, in the network each node uses the information in the RREQ/RREP packets to know about routes to other nodes in the network and store the routes information in their route caches. One drawback of AODV protocol, and many other Ad hoc network protocols, is the single route concept that requires a source node to establish a new route discovery process whenever a link failure is come across in the current route (Abdule & Hassan, 2010).

2.4.3. TCP-F

TCP-F was proposed nodes to detect the route failures which rely on the network layer at intermediate. In TCP-F, a TCP source can be in two states, active state and snooze state. Transmission is controlled by the standard TCP when TCP source is in active state. The

intermediate node will explicitly send the TCP source a Route Failure Notification (RFN) if it detects a link failure. After receiving the RFN, the TCP source will go into the snooze state by stopping sending any further packet and freezing the value of TCP state variables such as retransmission timer and congestion window size. The TCP source remains in the 'snooze' state until it is notified of the restoration of the route through a Route Re-establishment Notification (RRN) from an intermediate node and then goes back to active state (Chandran, el at, 1998).

2.4.4. ECP-ELFN

According to Monks, et al. (Monks, Sinha, & Bharghavan, 2000) *ECP-ELFN* is used the feedback to notify source route failure too. But the congestion control mechanism of ECP-ELFN is based on hop-by-hop rate control, not base on window control, which make it is top-priority when a transmission require little jitter of sending rate. However, as other rate-based congestion control algorithms, ECP-ELFN is slow-responsive to network congestion, so it has more possibility to cause congestion or worsen the degree of network congestion.

2.5. Explicit Link Failure Notification

Standard TCP misinterpret mobility loss in Ad hoc network as congestion loss, thus, it reduce the TCP performance by invoking unnecessary congestion control action (King, Phang, Ling, & Fong, 2007). Because of the reliable TCP that used in the wired networks performs unsatisfactory and the overall throughput of the ad hoc network is low, so ELFN play important role to increase the throughput. (Romanowicz, 2008). Chandran et al. (Chandran, et al, 2001) propose a TCP-feed-back scheme that uses ELFN from the node recently, Holland and Vaidya (Holland & Vaidya, 1999) revisit the properties of a TCP-

ELFN scheme and have provided valuable insight into how the an explicit link failure notification scheme can improve upon TCP's performance.

ELFN is one mechanism to manage link failures with dynamic cache update scheme to improve the TCP performance considerably (Romanowicz, 2008), that mean ELFN is a way to make TCP better handle cases when there are link failures, which is common in Adhoc networks. Since all nodes acting as routers have the full TCP/IP protocols stack, they have access to the routing protocols of the IP layer. The routing protocol can detect the link failure when the next node in the connection goes out of range, and the packet cannot be delivered. It sends the route error notification (RRER), which is flooded to all of the nodes including the source node. The TCP/IP protocols stack can be altered to use the RRER packet as the link failure notification. After the modifications, when the RRER packet is received, TCP can distinguish this link failure from the congestion. It can enter the "standby" mode by freezing the regular transmission of the packets until the connection is reestablished and then resume the transmission. The routing protocol can be modified to carry additional information in RRER packets, similar to the "host unreachable" ICMP message such as: sender address and port. This can identify at the sender of which connection this message is for (Romanowicz, 2008), (Shinde, Vinayak, Ramesh, & J, 2010). When sender receives the RRER packet and it detects that it is the source of the original message, it can notify the TCP layer about this link failure. TCP will stopping any more transmissions until a new route is computed, thereby preventing those packets from being lost along the broken route; and freezing the state of the TCP connection, thereby preventing it from cutting down its window size to one and entering the slow-start phase (Mahbub & Raj, 2004). When the acknowledgement packet is received TCP can leave the "standby" mode and restore the communication at the state as it was before the link failure (Shinde, Vinayak, Ramesh, & J, 2010).

2.6. Summary

This chapter presented the related background of the topics covered in this project. This chapter began with an overview of transmission control protocol TCP and its congestion control mechanism. It discussed the issues of TCP congestion window control that are related to this research. TCP issues in MANETs were discussed in this chapter as well.

It showed how link failure happens in MANET which degrades the performance of TCP senders as the TCP source has to reinitiate a new global route discovery to all network nodes which causes a significant network-overhead and congestion as well. That is the reason for the motivation to address the main factors that can enhance the performance of TCP in MANET. Furthermore, it covered ELFN technique developed to improve the TCP performance.

In the next chapter, the experimental tool and the methodology used for carrying out this research will be presented.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1. Introduction

Research methodology defines the research steps and procedures. This section will present the general research methodology in network simulation. Simulation has been chosen as our technique to study the network system in this research. The general research methodology for network simulation has been adapted from (Mahbub & Raj, 2004) where the steps of systematic simulation study is presented as shown in Figure 3.1. Hence, the same framework will be used in implementing this project.

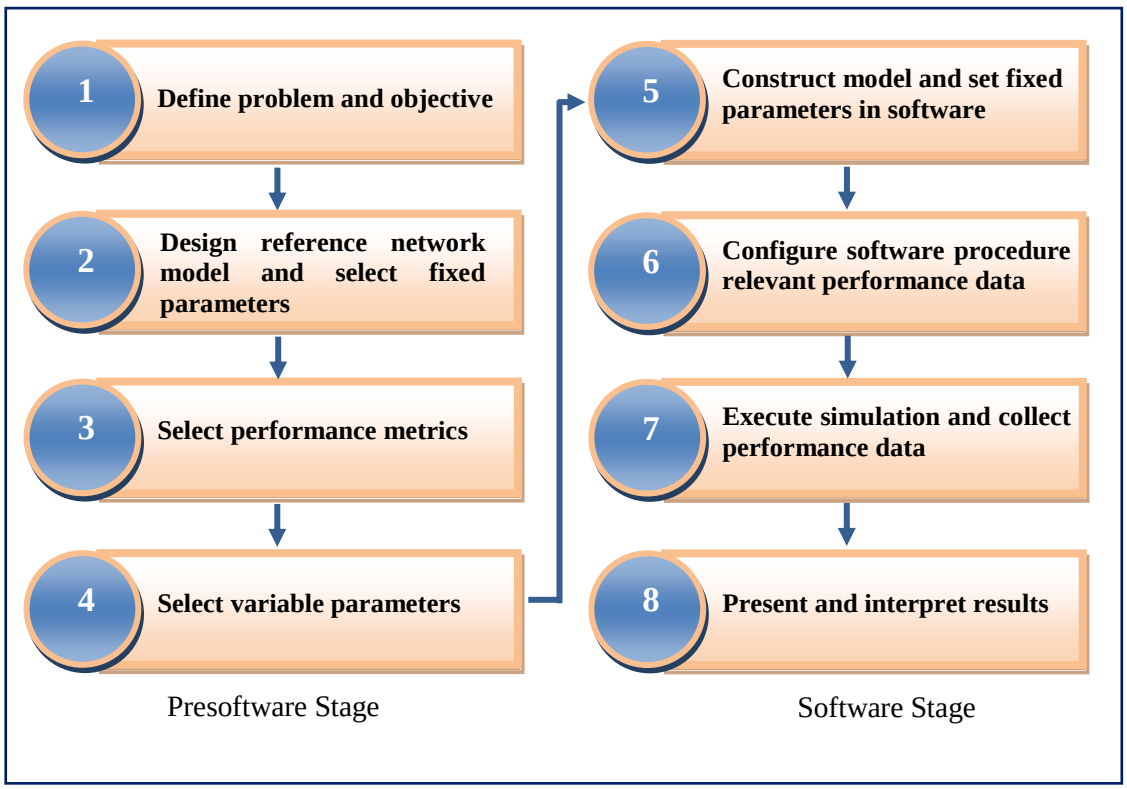

Figure 3.1 Steps of a systematic simulation study (Mahbub & Raj, 2004)

3.2. Network Simulator 2 (NS-2)

Before presenting the steps that will be taken in conducting a systematic simulation study as proposed previously, we have to choose the simulation tools in advance. There are several tools available for simulating TCP/IP networks.

In this section, we will be classify these tools into three main categories namely general purpose programming languages (GPPL), plain simulation language (PSL) and finally simulation packages (SP). Simulation packages are the highest level of simulation tools. Generally, all simulation packages provides built in libraries for TCP/IP networks. Among the simulation packages we have chosen Network Simulation 2 (NS-2) to implement this project.

Network Simulator 2 is a public domain simulation package which is a discrete event network simulator. It has been developed at UC Berkeley. At this point of writing, NS-2 has becoming one of the top choices for researchers especially in computer networks. NS-2 is being known for its capability in simulating advanced TCP/IP algorithms (Issariyakul, 2009). NS-2 is also an object-oriented simulator that is capable to simulate realistic network topologies and characteristics. NS-2 is constructed using C++ language and also object oriented TCL scripts.

3.3. Research Steps and Procedure

In this section the steps that are followed in conducting this research are presented

3.3.1. Defining Problem and Objectives

Defining problem and objectives of a research carried out perhaps is the most important part of this project. We have presented the problem statements and objectives of this

research which can be referred to. The problem statements and objectives define the purpose of the project's implementation.

3.3.2. Design Reference Network Model and Select Fixed Parameters

It is an obligatory step to design the network model required of the simulation that will be performed. The reference network model here refers to the network topology that will be used. After the network topology has been selected, we are required to set fixed parameters for the network that is going to be simulated such as the mobility model, traffic load and other related parameters settings.

3.3.3. Selecting Performance Metrics

This section will present the performance metrics that will be used to evaluate the performance of the networks in this research. These performance metrics are also referred to as output or response variables which are observed at the end of the simulation. In NS-2, in order to get the performance metrics values for example throughput, we have to use other scripts for example in this study we are using AWK script. Other scripting language for example, Perl, can also be used. The performance metrics that will be used in this are packet loss, average end to end delay, and normalize routing load achieved by each simulation settings.

3.3.4. Selecting Variable Parameters

Selecting variable parameters is the next step to setup the simulation environment. In this research, the variable parameters include the number of nodes values and the speed of travelling nodes, representing node densities. Simulations will be repeated using varying

node density, and different speed of travelling nodes in order to evaluate the performance of each simulation settings.

3.3.5. Construct Model and Set Fixed Parameters in Software

In this phase, we develop a constant setting to be used in the simulations to be applied with all the variable parameters mentioned previously in NS-2. For this research, the fixed parameters would be:

- Channel Type
- Propagation Model
- Network Interface Type
- MAC Type
- Interface Queue Type
- Antenna Model
- Maximum Packet in IFQ
- Simulation Time
- The Size of Simulation Area
- Routing Protocol
- Traffic Type

3.3.6. Configure Software to Produce Relevant Performance Data

This phase requires that the configuration and simulation settings done in NS-2 are relevant to the objective of this research. Hence, the parameters specified as mentioned in section 3.5 and section 3.6 should be determined according to the objective of performance evaluation conducted.

3.3.7. Execute Simulation and Collect Performance Data

For this project, execution of the simulation using NS-2 is done repetitively using different values of speed of travelling nodes and number of nodes. After the simulation has been executed, NS-2 will generate a trace file that contains all the information of the simulation which can be extracted later using an AWK or PERL script.

The figure 3.2 shows an example of a trace file generated by NS-2.

```
s -t 0.267662078 -Hs 0 -Hd -1 -Ni 0 -Nx 5.00 -Ny 2.00 -Nz 0.00 -Ne
-1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.255 -Id -1.255 -It
message -Il 32 -If 0 -Ii 0 -Iv 32
s -t 1.511681090 -Hs 1 -Hd -1 -Ni 1 -Nx 390.00 -Ny 385.00 -Nz 0.00 -Ne
-1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 1.255 -Id -1.255 -It
message -Il 32 -If 0 -Ii 1 -Iv 32
```

Figure 3.2 Example of NS-2 Trace File

3.3.8. Present and Interpret Results

Result presentation for this research will be done using a tool called Microsoft Excel. Microsoft Excel proposed as a tool allows to do software reconnaissance visually starting from a set of trace files for different test cases. Microsoft Excel is a graphical tool for data presentation . It has the ability to provide all matrices that are required for the network performance study

3.4. Simulation Setup

The simulation will be carried out using the standard TCP and TCP that is enhanced with Explicit Link Failure Mechanism. The scenarios consist of three different values for node density with TCP and TCP-ELFN respectively, in addition to the three different values for

the speed of travelling nodes for TCP and TCP-ELFN respectively. Hence, in total, there will be 12 scenarios that will be simulated. The organization of the simulated scenarios is presented in the Figure 3.3 below. The performance of the two will be compared and analyzed.

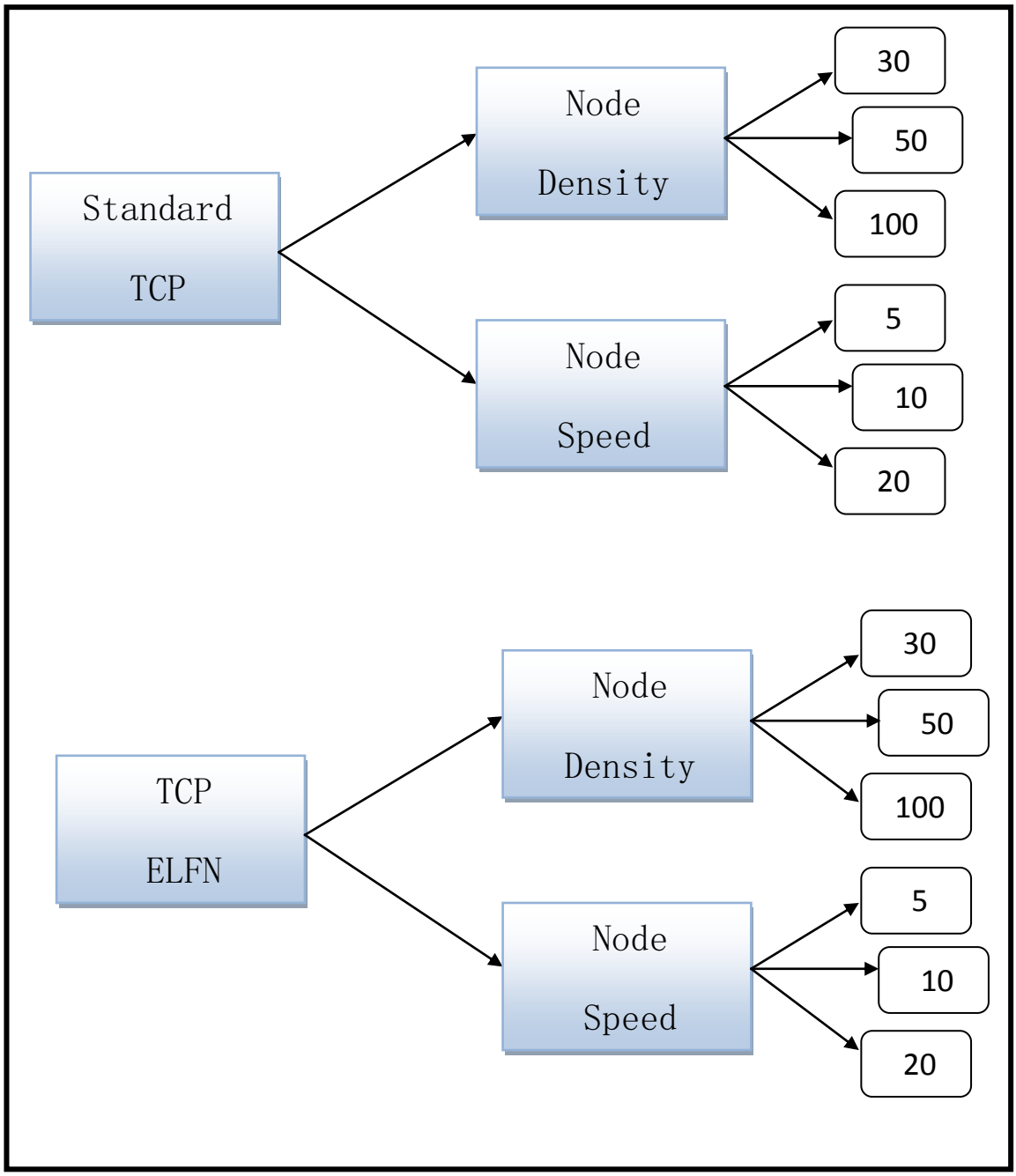

Figure 3.3 Simulated Scenarios

Table 1 and Table 2, presents two different scenarios that will be carried out. The details of the simulation parameters are shown in Table 1 and Table 2 below.

Table 1 Varying Settings for TCP and TCP-ELFN Simulations with Node Density

Parameters	Value
Channel Type	WirelessChannel
Propagation Model	TwoRayGround
Net Interface Type	WirelessPhy
MAC Type	802.11
Interface Queue Type	DropTail/PriQueue
Antenna Model	OmniAntenna
Max Packet in IFQ	50
Transport Protocol	TCP, TCP-ELFN
Simulation Time	100s
Number of Nodes	100
Speed of Nodes	5, 10,20 m/s
Max. Connections	0.5
Simulation Area	500x500m
Mobility Model	Random Waypoint
Routing Protocol	AODV
Packet Size	512 bytes
MAC Type	802.11

Table 2 Settings for TCP and TCP-ELFN Simulations with Varying Speed

Parameters	Value
Channel Type	WirelessChannel
Propagation Model	TwoRayGround
Net Interface Type	WirelessPhy
MAC Type	802.11
Interface Queue Type	DropTail/PriQueue
Antenna Model	OmniAntenna
Max Packet in IFQ	50
Transport Protocol	TCP, TCP-ELFN
Simulation Time	100s
Number of Nodes	30, 50, 100
Max. Connections	0.5
Simulation Area	500x500m
Mobility Model	Random Waypoint
Routing Protocol	AODV
Packet Size	512 bytes
MAC Type	802.11

3.5. Performance Metrics

In this subsection, the performance metrics that is used to evaluate the network performance is presented. This metrics are calculated based on the equations given.

The performance of the standard TCP and TCP-ELFN in mobile ad hoc network will be studied based on three performance metrics which are the packet loss, average end-to-end delay and the normalized routing load. Each metric is calculated using a set of AWK script that is parsed with the simulation trace file in order to get the result. The simulation result will be presented and discussed in the following subsection.

3.5.1. Packet Loss

Measuring the performance of the network using packet loss is crucial in determining how well the network reacts to congestion or in the case mobile ad hoc network, frequent link failure. In order to measure the packet loss is calculated as follows:

$$PktDrop = Pktsent - Pktrecv$$

The AWK script that is used in order to calculate packet loss is as shown below:

```
BEGIN {
        sends=0;
        recvs=0;
        droppedPackets=0;
             }

     {
  time = $3;
```

```
  packet_id = $41;
# CALCULATE PACKET DELIVERY FRACTION
  if (( $1 == "s") &&  ( $35 == "tcp" ) && ( $19=="AGT" ))
      {  sends++;
                }
  if (( $1 == "r") &&  ( $35 == "tcp" ) && ( $19=="AGT" ))
      {  recvs++;
                  }

  # DROPPED dfrp PACKETS
  if (( $1 == "d" ) && ( $35 == "tcp" )  && ( $3 > 0 ))
       {
           droppedPackets=droppedPackets+1;
            }

  #find the number of packets in the simulation
          if (packet_id > highest_packet_id)
             highest_packet_id = packet_id;
         }

  END {
     printf("Packet Sent = %.2f\n",sends);
     printf("Packet Receive = %.2f\n",recvs);
     printf("No. of dropped data (packets) = %d\n",droppedPackets);
      }
```

3.5.2. Normalized Routing Load (NRL)

Normalized routing load is defined by the quantity of routing packets being transmitted per packet sent to the destination. NRL also assumes that each forwarded packet as one

transmission. NRL is immensely associated with the number of path or link changes or disconnection that happened during the simulations.

$$NRL = \frac{Number\ of\ routing\ packets\ sent}{Number\ of\ packets\ received}$$

The Normalized Routing Load (NRL) calculation had been programmed into an AWK script is presented below:

```
BEGIN {
        recvs = 1;
        routing_packets = 1
        }
{
if (($1 =="r") && ($35 =="tcp") && ($19=="AGT"))
recvs++;
if (($1 == "s" || $1 =="f") && ($19 == "RTR") && ($45 == "tcp"))
routing_packets++;
}
}
END{
printf("NRL = %.3f",routing_packets/recvs);
printf("Routing packets = %.3f\n",routing_packets);
```

```
printf("Received packets = %.3f\n",recvs);
}
```

3.5.3. Average End-to-End Delay

When transmission of a packet between two nodes, the average delays between the sending and the receiving is called the average end to end delay. Average end to end delay depicts that, if the value is higher, it means that the network is experiencing congestion hence causing the routing protocols not able to perform efficiently. The average end to end delay is calculated as follows:

$$\boldsymbol{AEED = \sum_{i-0}^{n} \frac{timepacketrcvdi - timepacketsenti}{totalnumberofpacketsreceived}}$$

The end to end delay calculation had been programmed into an AWK script is presented below:

```
BEGIN {
        sends = 0;
        recvs = 0;
        routing_packets = 0.0;
        highest_packet_id = 0;
        sum = 0;
        recvnum = 0;
     }
```

```
    {
  time = $3;
  packet_id = $41;

  # CALCULATE PACKET DELIVERY FRACTION
  if (( $1 == "s") &&  ( $35 == "tcp" ) && ( $19=="AGT" )) {  sends++; }
  if (( $1 == "r") &&  ( $35 == "tcp" ) && ( $19=="AGT" ))   {  recvs++;
}

  # CALCULATE DELAY
  if ( start_time[packet_id] == 0 )  start_time[packet_id] = time;
  if (( $1 == "r") &&  ( $35 == "tcp" ) && ( $19=="AGT" ))
    {  end_time[packet_id] = time;
       }
      else {  end_time[packet_id] = -1;
            }
   #FIND THE NUMBER OF PACKETS IN THE SIMULATION
        if (packet_id > highest_packet_id)
           highest_packet_id = packet_id;
    }
   END {
```

```
   for ( i in end_time )
     {
start = start_time[i];
end = end_time[i];
packet_duration = end - start;
if ( packet_duration > 0 )
   {    sum += packet_duration;
        recvnum++;
      }
}
      delay = sum/recvnum;
   printf("Packet Sent = %.2f\n",sends);
   printf("Packet Receive = %.2f\n",recvs);
   printf("Average end-to-end Delay (ms)= %.2f\n",delay*1000);

}
```

3.6. Summary

In this chapter we have presented the research methodology being use in this project. This chapter has also presented the simulation setup of the experiments that is carried out in details including the settings being used in the NS-2. The following chapter will present the design and implementation of this project.

CHAPTER FOUR

DESIGN AND IMPLEMENTATION OF THE PERFORMANCE EVALUATION MODEL

While Chapter 3 presents the research methodology used for designing and implementing the performance evaluation model designated for evaluating the standard TCP and TCP-ELFN in mobile ad hoc network, this chapter presents the design requirements and the implementation details of the performance evaluation model that are done to satisfy the first objective.

4.1. Introduction

In order to develop a performance evaluation model in computer network, every component in the network simulation must be defined. This includes the number of nodes, the type of connection, type of traffic and mobility model being used.

The following subsection of this chapter presents the performance evaluation model that is used in this project.

4.2. The Model Implementation

In this section, the codes and files required to implement and run the performance evaluation model are presented.

4.2.1. Tool Command Language (TCL) Script

Tool Command Language (TCL) is utilized to implement the performance evaluation model in Network Simulator 2 (ns-2). In the following, the details of TCL codes for developing the performance evaluation model are provided.

- Define the type of the wireless channel

```
set val(chan)           Channel/WirelessChannel
```

- Define the radio-propagation model

```
set val(prop)           Propagation/TwoRayGround
```

- Define the network interface type

```
set val(netif)          Phy/WirelessPhy
```

- Define the MAC type

```
set val(mac)            Mac/802_11
```

- Define the queuing technique. The IF statement specify that, if the routing protocol is DSR, the queuing technique will be CMUPriQueue.

```
if { $par1=="DSR"} {

   set val(ifq)            CMUPriQueue

    } else {
```

```
        set val(ifq)            Queue/DropTail/PriQueue

    }
```

- Define the link layer type

```
set val(ll)             LL
```

- Define the antenna type

```
set val(ant)            Antenna/OmniAntenna
```

- Define the maximum number of packets in the interface queue

```
 set val(ifqlen)          50
```

- Define the routing protocol. $par1 refers to the value entered by user.

```
set val(rp)             $par1
```

- Define topography area by (x) and (y).

```
set val(x)              500
```

```
set val(y)              500
```

- Define the file for the trace file to written in.

```
set val(tr)             n20.tr
```

- Define the number of nodes. Example below shows number of node as 30.

```
set val(nn)             30
```

- Specify the file name that has the traffic pattern of the simulation.

```
set val(cp)             $par2
```

- Specify the file name that has the traffic pattern of the simulation.

```
 set val(sc)             $par3
```

- Specify time for the simulation to end.

```
set val(stop)           100.0
```

- Instantiate the network simulator.

```
set ns_              [new Simulator]
```

- To specify the file for the network simulator to write the trace on.

```
set tracefd     [open $val(tr) w]

$ns_ trace-all $tracefd
```

- Use the new trace file format.

```
$ns_ use-newtrace
```

- Define the topography of the simulation.

```
set topo        [new Topography]

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

- Create and setup the general operation director (god).

```
set god_ [create-god $val(nn)]
```

- Redefined the variable for each node configuration.

```
$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) \
                 -channel $chan_1_ \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace OFF
```

- *For* loop for the nodes.

```
for {set i 0} {$i < $val(nn) } {incr i} {
```

```
                    set node_($i) [$ns_ node]

                    $node_($i) random-motion 0

}
```

- Load the connection pattern or traffic file.

```
puts "Loading connection pattern..."

source $val(cp)
```

- Load the scenario file.

```
puts "Loading scenario file..."

source $val(sc)
```

- Define the initial position of the nodes.

```
for {set i 0} {$i < $val(nn) } {incr i}

  {

    $ns_ initial_node_pos $node_($i) 20

      }
```

- Reset all the nodes when simulation ends.

```
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).000000001 "$node_($i) reset";
}
```

- Exit and stop the network simulator when simulation ends.

```
$ns_ at $val(stop).000000001 "puts \"NS EXITING...\"; $ns_ halt"
```

- Display ‘Start Simulation’ for users when the simulation starts.

```
puts "Start Simulation..."
```

- Run the ns-2 simulator according the what is defined in the TCL script

```
$ns_ run
```

4.2.2. Connection Pattern Script

In this project we due to very large number of nodes simulated, we have used the connection pattern script to specify the generation and transferring of traffic in the network. The full script is included in the Appendices section of this report.

To create connection file, run

ns tcpgen.tcl [-type cbr | tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate] > connection_file_name

These command represent:

-type : type of connection CBR or TCP

-nn : number of nodes

-seed : seed value for the random-number generator

-mc : number of connection

-rate : transmission rate

> : save output this file in

4.2.3. Mobility Generation Script

Another script that is used in this simulation is the mobility generation file in which in this script we specifies the initial position of each node, its destination and their related speed. The full script for the mobility generation pattern is included in the Appendices section of this report.

To create mobile node movement file, run

./setdest –n <num_of_nodes> -p <pausetime> -s <maxspeed> -t <simtime> -x <maxx> -y <maxy> > movement_file_name

These command represent:

-n : number of nodes

-p : pause time for node to move again

-s : maximum speed

-t : simulation time

-x : maximum X dimension of the topography

-y : maximum Y dimension of the topography

> : save output this file in

4.3. Building TCL File

This section of coding we are indentified and set all data structure related with this work. The code mentioned below represents an example of the code that should be put in the TCL file.

```
#-----------------------------------------------------------------------
# DEFINITION OF THE PHYSICAL LAYER
#-----------------------------------------------------------------------
set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/802_11
set val(ifq)        Queue/DropTail/PriQueue
set val(ll)         LL
set val(ant)         Antenna/OmniAntenna

#-----------------------------------------------------------------------
# SCENARIO PARAMETERS
#-----------------------------------------------------------------------
```

```
set val(x)              500    ;# X dimension of the topography
set val(y)              500    ;# Y dimension of the topography
set val(ifqlen)         50     ;# max packet in queue
set val(seed)           1.0    ;#random seed
set val(adhocRouting)   ELFN
set val(nn)             30     ;# how many nodes are simulated
set val(cp)             "tcp-30"
set val(sc)             "scen1-30"
set val(stop)            100   ;# simulation time

#-------------------------------------------------------------------------
#  SET UP SIMULATOR OBJECTS
#-------------------------------------------------------------------------
# CREATE SIMULATOR INSTANCE
  set ns_ [new Simulator]
# SETUP TOPOGRAPHY OBJECT
  set topo [new Topography]
# CREATE TRACE OBJECT FOR NS AND NAM
  set tracefd [open ELFN-60tcptraffic.tr w]
  $ns_ use-newtrace         ;# use the new wireless trace file format
  set namtrace  [open ELFN.nam w]
  $ns_ trace-all $tracefd
  $ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
# DEFINE TOPOLOGY
  $topo load_flatgrid $val(x) $val(y)
# CREATE GOD
  set god_ [create-god $val(nn)]
  $ns_ node-config -adhocRouting $val(adhocRouting) \
                   -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 ON \

# ---------------------------------------------------------------------
#  CREATE THE SPECIFIED NUMBER OF NODES [$VAL(NN)] AND "ATTACH" THEM
#  TO THE CHANNEL.
# ---------------------------------------------------------------------
for {set i 0} {$i < $val(nn) } {incr i} {
   set node_($i) [$ns_ node]
   $node_($i) random-motion 0;       #  disable random motion
            }
# DEFINE NODE MOVEMENT MODEL
  puts "Loading connection pattern..."
  source $val(cp)
# DEFINE TRAFFIC MODEL
  puts "Loading scenario file..."
  source $val(sc)

#---------------------------------------------------------------------
# DEFINE NODE INITIAL POSITION IN NAM
# 50 DEFINES THE NODE SIZE IN NAM, MUST ADJUST IT ACCORDING TO YOUR
# SCENARIO THE FUNCTION MUST BE CALLED AFTER MOBILITY MODEL IS DEFINED
```

```
# PUTS "PROCESSING NODE $I"
#-----------------------------------------------------------------------
 for {set i 0} {$i < $val(nn)} {incr i} {
    $ns_ initial_node_pos $node_($i) 50
        }

# ---------------------------------------------------------------------
# TELL NODES WHEN THE SIMULATION ENDS
# ---------------------------------------------------------------------
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
      }
$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

#---------------------------------------------------------------------
# DUMP THE INITIAL SIMULATION INFO TO THE TRACE FILE
# ---------------------------------------------------------------------
puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp
$val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"
puts "Starting Simulation..."
$ns_ run
```

4.4. Summary

This chapter presented the implementation part for the performance evaluation of the standard TCP and TCP-ELFN. The TCL code that is used in this project had been presented in details.

The next chapter will present the performance evaluation of both TCP mechanisms so that their performance can be analyzed based on the numerical result that is obtained from the simulation.

CHAPTER FIVE

EVALUATION AND RESULTS

While the previous chapter presents the design and implementation issues of the performance evaluation setups developed to investigate the different performance between the standard TCP and the TCP that is enhanced with ELFN, this chapter provides the performance of these TCP based on the simulation results. It compares the performance of TCP-ELFN, which includes an additional mechanism called explicit link failure notification, to that of the standard TCP. The performance is compared in terms of the packet loss, average end-to-end delay and normalized routing load.

5.1. Introduction

In this Chapter, we present the performance evaluation of both TCP mechanisms. First, we run the simulations with the standard TCP and with TCP-ELFN to investigate their performance when they are supplied with different number of nodes. This is important to check how the performance of the network using the standard TCP and TCP-ELFN will be when the node density is varied. For this experiment, we have set the speed of the travelling nodes to a constant which is 10m/s. On the other hand, we have set the number of nodes to be 30, 50 and 100 respectively for each simulation

Second, we run the simulations with the standard TCP and TCP-ELFN to investigate their performance when the speed of the travelling nodes in the network is varied. In this project, we have chosen the speed of 5 m/s, 10 m/s and 20 m/s to be studied. For this experiment, we have set the number of nodes to a constant which is 100 nodes. On the

other hand, we have set the speed of the travelling nodes are 5 m/s, 10 m/s and 20 m/s respectively for each simulation.

As mentioned in Chapter 3, the performance evaluation of these TCP and TCP-ELFN will be evaluated using three different performances metric which are packet loss, average end-to-end delay and normalized routing load. The AWK script that is used in calculating these metrics has also been presented in Chapter 3.

The numeric result from TCP and TCP-ELFN simulations will later be compared in order to investigate how both TCP behave when they are supplied with different number of nodes in one case and in another case when they are supplied with varying nodes speed.

5.2. Performance of standard TCP with Varying Number of Nodes

In this part of the experiments, we run the simulations with the standard TCP and investigate its performance when it is supplied with different number of nodes. For the case of using different number of nodes with standard TCP 30, 50, and 100 nodes.

5.2.1. Packet Loss for Standard TCP with Varying Number of Nodes

Packet loss results is a very noticeable performance issues, as mentioned in Chapter 3. Packet drops degrades the performance of the TCP applications significatly when the packet loss is high. For the case of using different number of nodes with standard TCP (30, 50, 100), the packets loss recorded the highest when number of nodes is set to 100.

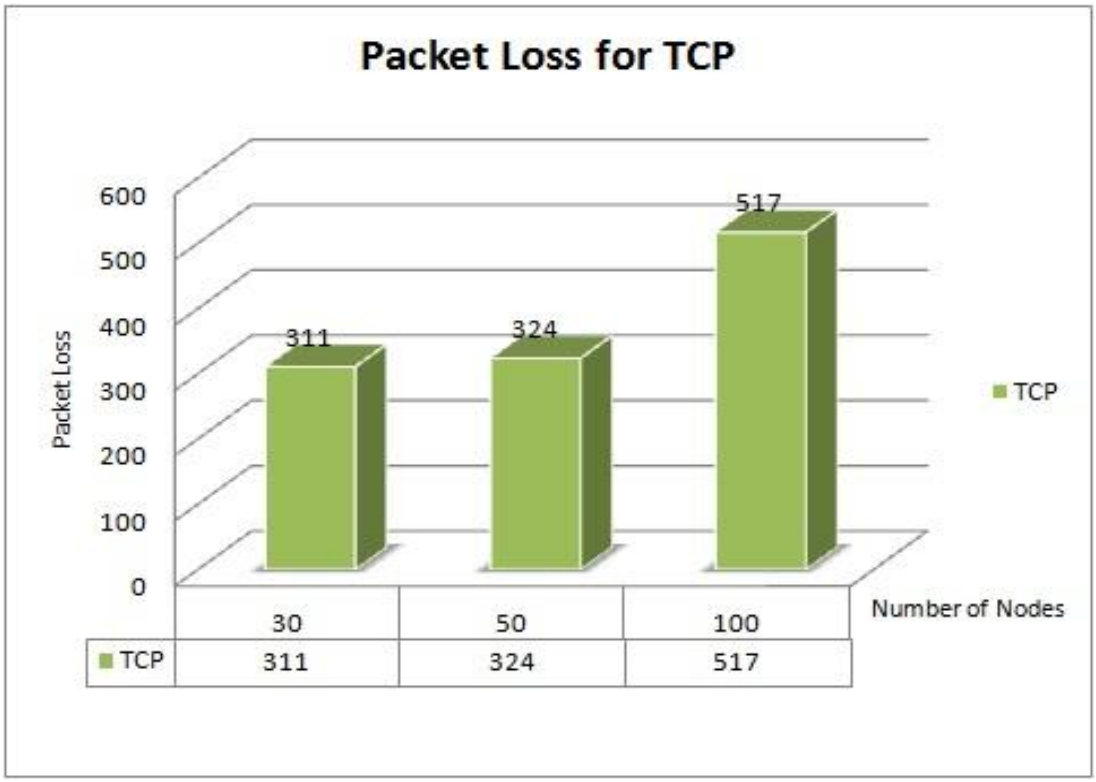

Figure 5. 1 Packet Loss for TCP with Varying Node Density

Relatively, the result in Figure 5.1 shows an increasing pattern of packet loss when the standard TCP is being used. The relevant behind this result is that, when the number of

nodes is increased, the amount of data being transferred into the network will also increase causing congestion and finally leads to packet loss.

5.2.2. Average End-to-End Delay for Standard TCP with Varying Number of Nodes

As defined in Chapter 3, the average end-to-end delay is defined as the average delay between the sending and receiving of packet between two nodes. Figure 5.2 shows that, the average end to end delay increased when the number of nodes is increased. When 30 nodes are supplied into the network, the average end to end delay is 769.17 milliseconds. This value increased to 967.51 milliseconds when the number of nodes is 50, and finally when the node density is 100, the delay increases to 1008.39 milliseconds. So, when using the standard TCP, the average end to end delay will increase with respect to the number of nodes. This is due to the reason that, when there are many nodes in the network, the time it takes to send a packet to a destination will increase respectively because of the busy transmission between nodes and the network.

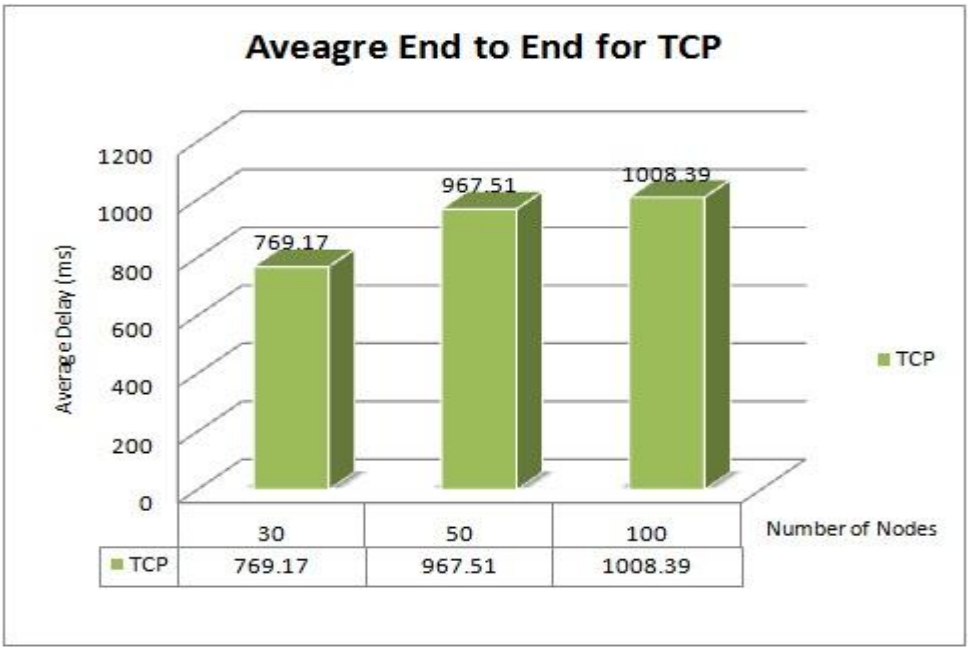

Figure 5. 2 Average end-to-end Delay for TCP with Varying Node Density

5.2.3. Normalized Routing Load for Standard TCP with Varying Number of Nodes

As mentioned in Chapter 3, normalized routing load is defined by the quantity of routing packets being transmitted per packet sent to the destination. NRL also assumes that each forwarded packet as one transmission. NRL is immensely associated with the number of path or link changes or disconnection that happened during the simulations. Figure 5.3 presents the results of standard TCP performance when it is being simulated with different number of nodes. The result shows a drastic decrease of normalized routing load when the number of nodes is increased to 100. In relation to TCP, the performance of the network will be as presented in the Figure 5.6. When the simulation is repeated using TCP with Explicit Link Failure Notification, we are expecting that the result will improve. This will be presented in the later section.

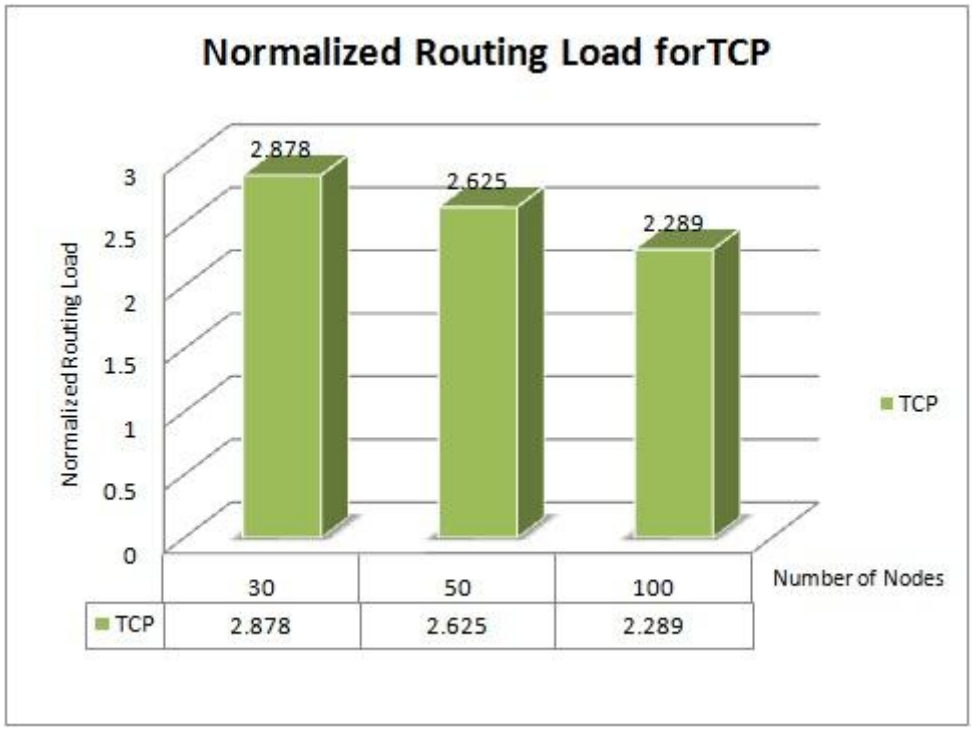

Figure 5. 3 Normalized Routing Load for TCP with Varying Node Density

5.3. Performance of TCP-ELFN with Varying Number of Nodes

In this part of the experiments, we run the simulations with the TCP-ELFN and investigate its performance when it is supplied with different number of nodes. For the case of using different number of nodes with TCP-ELFN are 30, 50, and 100 nodes.

5.3.1. Packet Loss for TCP-ELFN with Varying Number of Nodes

This section will present the results of TCP-ELFN performance when it is supplied with different number of nodes. Figure 5.4 shows that the packet loss of TCP-ELFN is 218 when the number of nodes is set to 30, and increased when nodes is increased to 50. Finally, the amount of packet loss is increase to be 364 when the node is increased to 100.

This result is gained due to the reason that although there are more exchanges of information or packet in the network when the number of nodes is set to 100, however, with the enhancement in TCP-ELFN, the network is capable to handle such cases better.

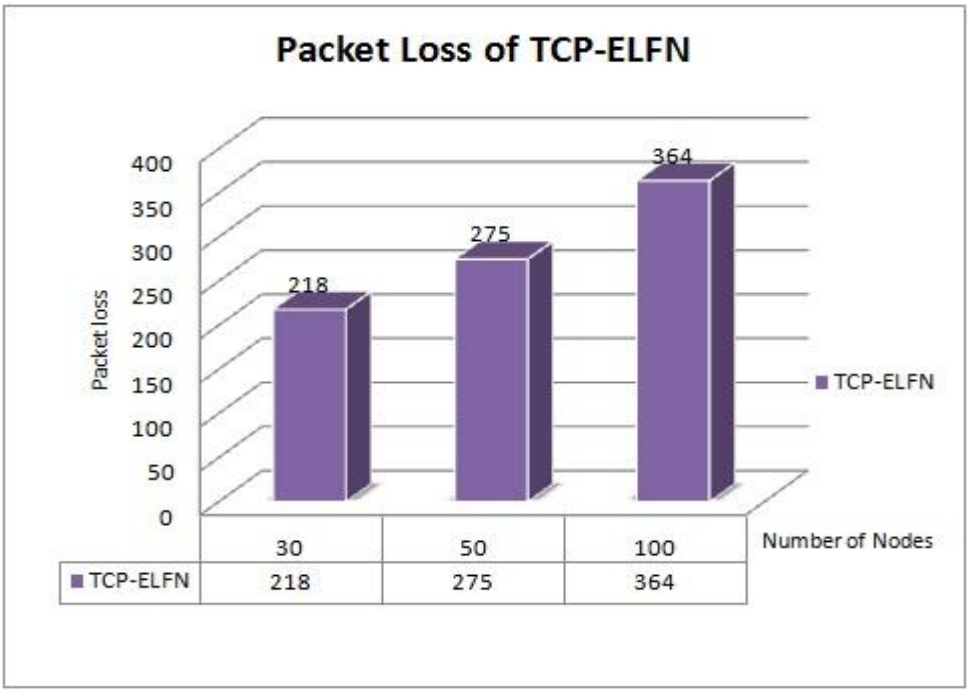

Figure 5. 4 Packet Loss for TCP-ELFN with Varying Node Density

5.3.2. Average End-to-End Delay for TCP-ELFN with Varying Number of Nodes

Figure 5.5 presents the result of the average end-to-end delay for TCP-ELFN with varying node density. From the result we can see that as the number of nodes increased, the average end to end delay also increases. With this result, we can deduced that, although TCP-ELFN may reduce the number of packet loss as the number of node increases, the average end to end delay remains increasing with respect to the number of nodes.

However, we are expecting that, the average end to end delay of TCP-ELFN will be lower than the delay gained when the standard TCP is being used. This will be investigated in the later section.

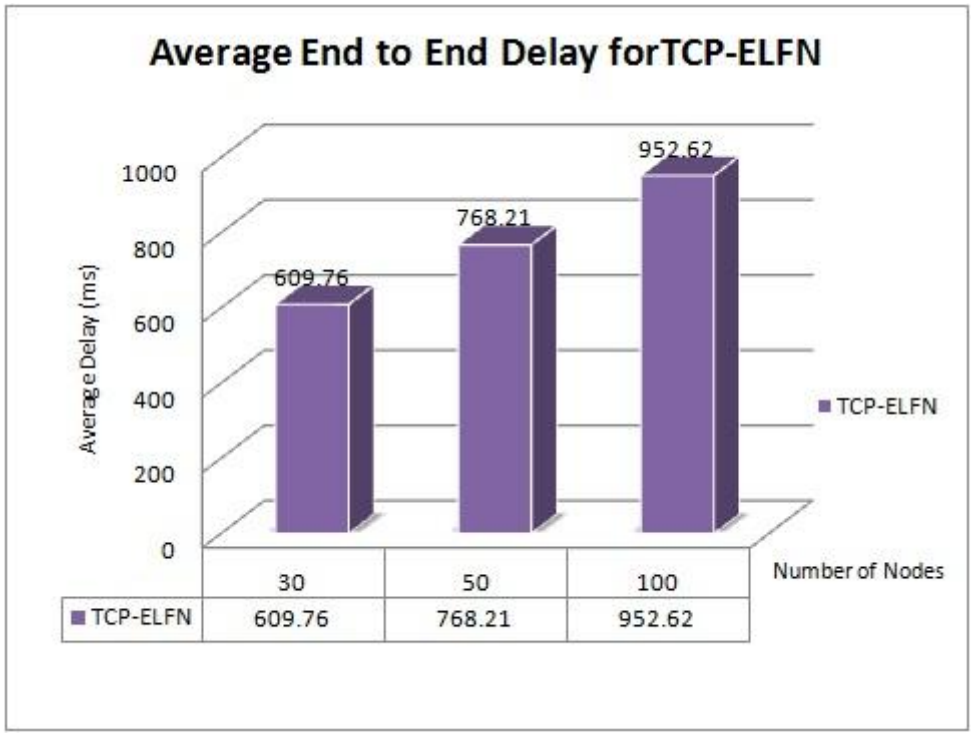

Figure 5. 5 Average End-to-End Delay for TCP-ELFN with Varying Node Density

5.3.3. Normalized Routing Load for TCP-ELFN with Varying Number of Nodes

Figure 5.6 present the result in terms of the normalized routing load for TCP-ELFN with varying node density. Here the simulation is performed similar to those before by increasing number of nodes from 30 to 50 to 100.

In Figure 5.6 we can see that, the normalized routing load behavior is the same as in the standard TCP in which it is increasing with respect to number of nodes.

However, this increasing pattern does not indicate that it has the same performance with the standard TCP. The performance comparison of both TCP mechanisms will be presented in later section.

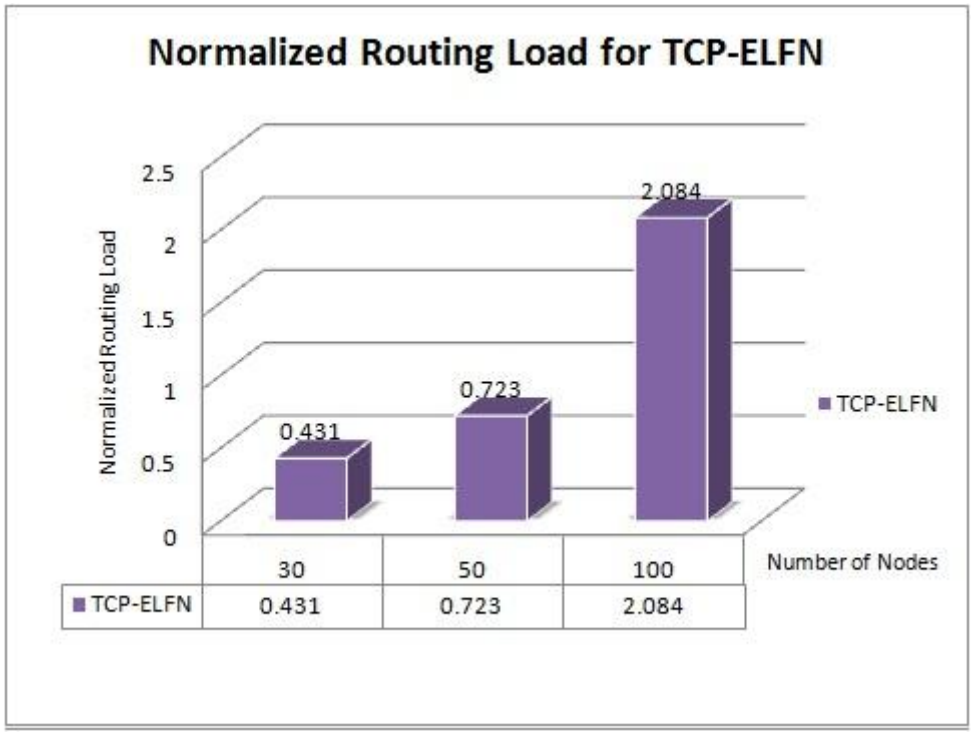

Figure 5. 6 Normalized Routing Load for TCP-ELFN with Varying Node Density

5.4. Performance of standard TCP with Varying Nodes Speed

In this part of the experiments, we run the simulations with the standard TCP and investigate its performance when the speed of the travelling nodes in the network is varied. For case of using speed of the travelling nodes with standard TCP to be 5,10, and 20 m/s.

5.4.1. Packet Loss for Standard TCP with Varying Nodes Speed

Figure 5.7 presented the packet loss of the standard TCP when the speed of the travelling nodes in the network is varied. In this project, we have chosen the speed of 5 m/s, 10 m/s and 20 m/s to be studied. The result does not show a constant pattern, in other hand, the amount of packet loss when the nodes is fixed to travel with the speed of 10 m/s has the highest number of packet loss as compared to the network that the nodes moves at 5 m/s and 20 m/s.

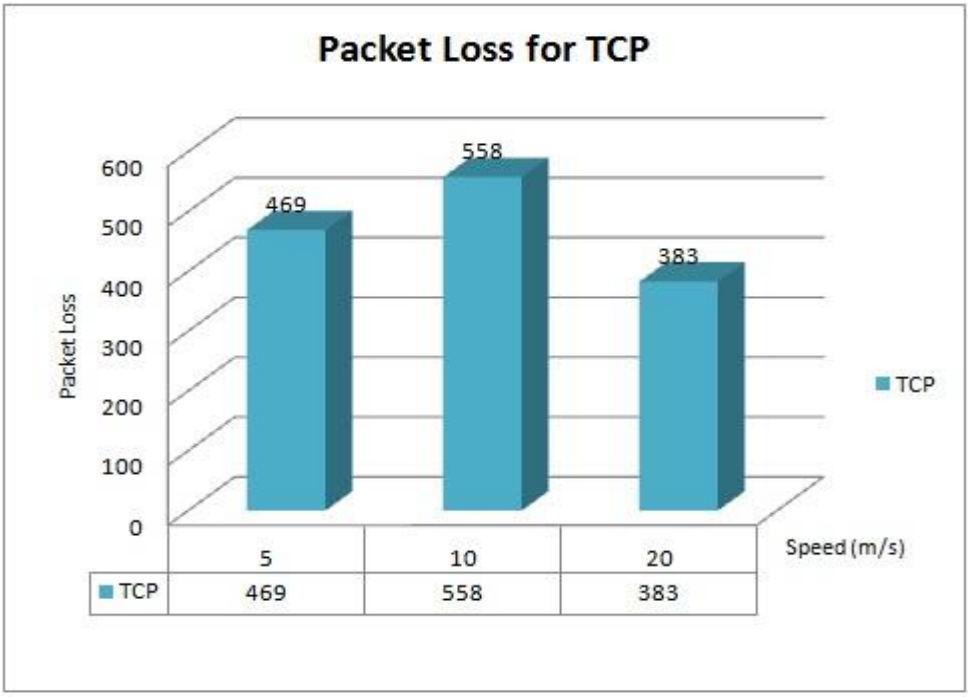

Figure 5. 7 Packet Loss for Standard TCP with Varying Nodes Speed

5.4.2. Average End-to-End Delay for Standard TCP with Varying Nodes Speed

Figure 5.8 presents the average end to end delay for the standard TCP when the node's speed is varied. Similar to the packet loss, the average end to end delay is slightly increased when the node's speed is increased to 10 m/s but is reduced when the speed is set to 20 m/s. The relevant of this result is that, the fast movement of the nodes creates better links for information to be transmitted.

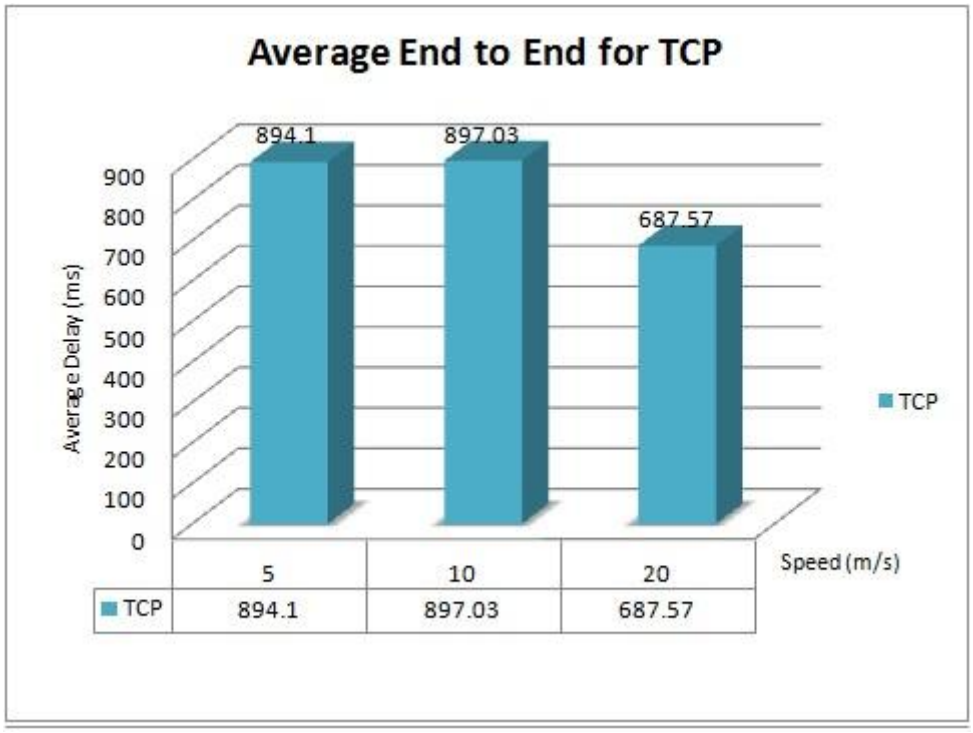

Figure 5. 8 Average End to End Delay for Standard TCP with Varying Nodes Speed

5.4.3. Normalized Routing Load for Standard TCP with Varying Nodes Speed

Figure 5.9 present the result of the standard TCP in terms of normalized routing load when it is supplied with networks of different node's speed. From the figure we can deduced that, as the number of speed increased, the frequency of link breakage is higher hence higher routing load will be in the network. This is due to the reason that, as the node move faster, one link will be disconnected from the other due to that movement, hence causing the need to re-establish the route more frequently.

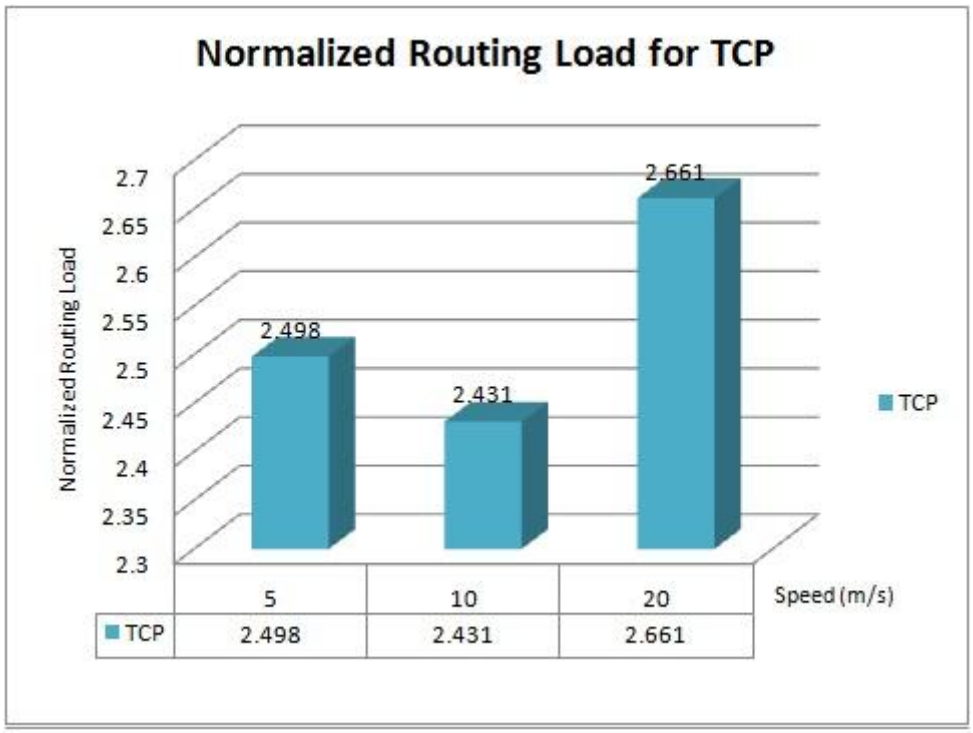

Figure 5. 9 Normalized Routing Load for Standard TCP with Varying Nodes Speed

5.5. Performance of standard TCP-ELFN with Varying Nodes Speed

In this part of the experiments, we run the simulations with the TCP-ELFN and investigate its performance when the speed of the travelling nodes in the network is varied. For case of using speed of the travelling nodes with TCP-ELFN to be 5,10, and 20 m/s.

5.5.1. Packet Loss for TCP-ELFN with Varying Nodes Speed

Figure 5.10 presents the packet loss for TCP-ELFN when the node's speed in the network is varied. We can deduce from the result that the packet loss is decreasing dramatically when the speed of nodes in the network is increased to 20 m/s. This is due to the behaviour of explicit link failure notification mechanism in which it provides the TCP sender with information about link and route failures so that it can avoid responding to the failures as if congestion occurred.

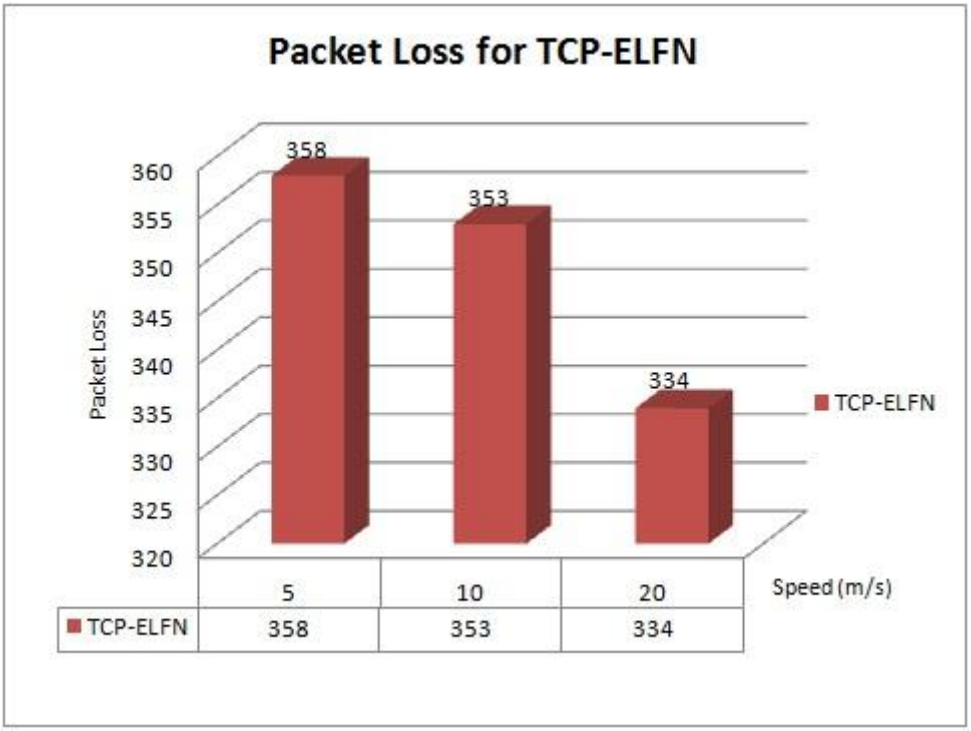

Figure 5. 10 Packet Loss for TCP-ELFN with Varying Nodes Speed

5.5.2. Average End-to-End Delay for TCP-ELFN with Varying Nodes Speed

Figure 5.11 presents the average end to end delay for TCP-ELFN with varying nodes speed. We can see that as the speed is increased to 10 m/s the delay increases dramatically while when it is increased to 20 m/s the end to end delay decreased to 1042.17 milliseconds.

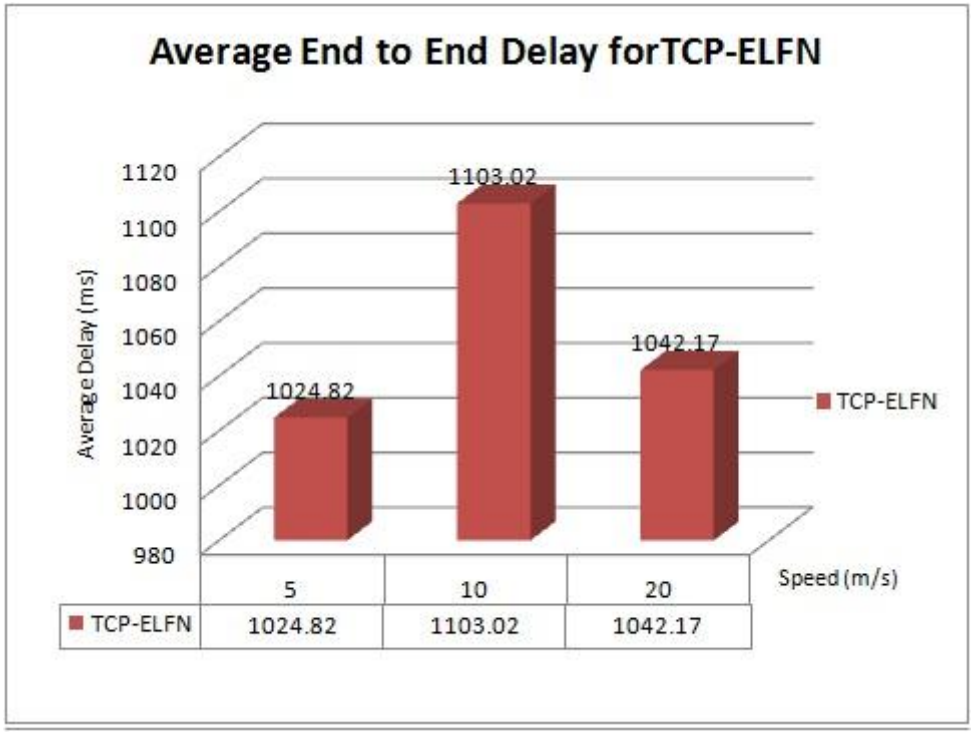

Figure 5. 11 Average End to End Delay for TCP-ELFN with Varying Nodes Speed

5.5.3. Normalized Routing Load for TCP-ELFN with Varying Nodes Speed

Figure 5.12 presents the normalized routing load for TCP-ELFN when the node's speed is varied by 5 m/s, 10 m/s and 20 m/s.

From the result shown in the figure, we can see an increasing pattern of the normalized routing load. However, the difference is subtle.

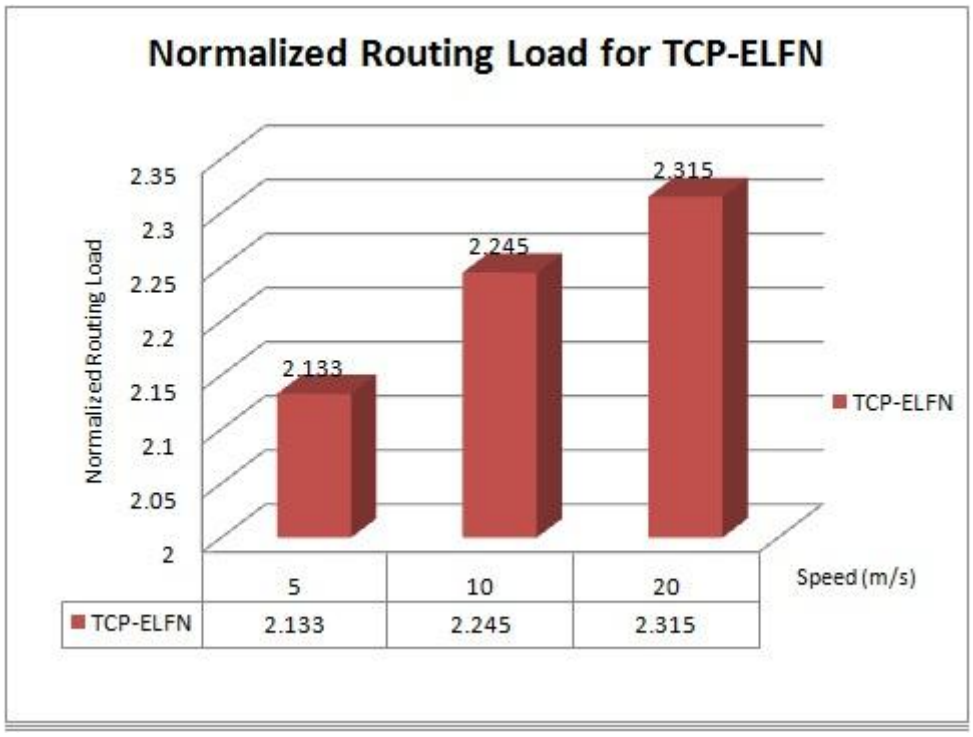

Figure 5. 12 Normalized Routing Load for TCP-ELFN with Varying Nodes Speed

5.6. Comparative between Standard TCP and TCP-ELFN

The third objective of this research is to compare the performance of TCP in mobile ad hoc network with and without the implementation of ELFN. Therefore, this section presents comparison of the standard TCP to TCP with the TCP enhanced with Explicit Link Failure Notification (ELFN).

5.6.1. Comparison of Packet Loss between Standard TCP and TCP-ELFN with Varying Node Density

Figure 5.13 presents the performance comparison of the standard TCP and TCP-ELFN. From all the three settings of node density (10, 30, 50), the result have shown that TCP-ELFN performed better as it produced less packet losses with all the three node density. Even though both TCP mechanisms shows an increasing pattern of packet loss as the number of node increases, the amount of the packet losses at each value of node density shows that TCP-ELFN has lower packet losses which is more preferable in any network.

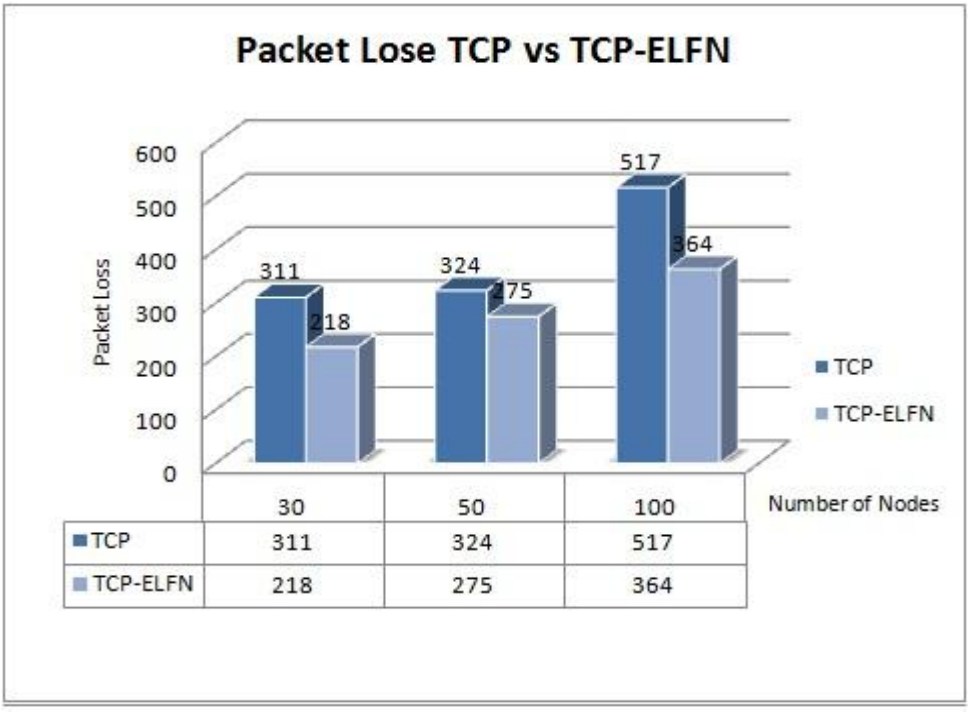

Figure 5. 13 Packet Loss of TCP vs TCP-ELFN with Different Node Density

5.6.2. Comparison of Average End-to-End Delay between Standard TCP and TCP-ELFN with Varying Node Density

Figure 4.14 presents the performance comparison of the standard TCP and TCP-ELFN in terms of average end to end delay.

From the figure we can see that, regardless of the number of nodes being supplied into the network, the average end to end delay of TCP-ELFN is much lower as compared to the standard TCP.

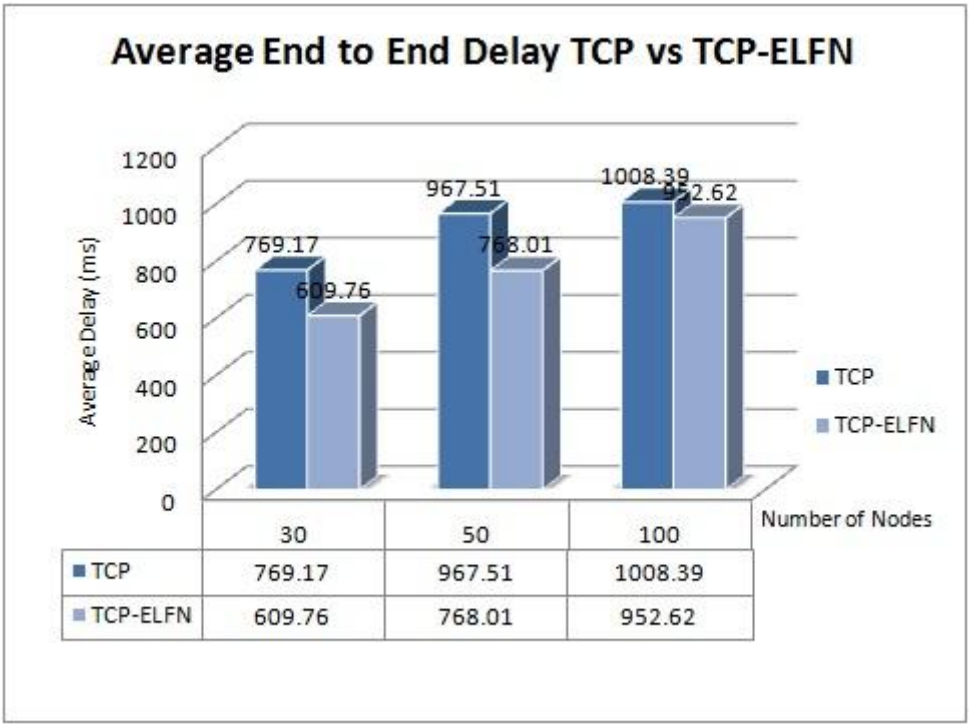

Figure 5. 14 Average End to End Delay of TCP vs TCP-ELFN with Different Node Density

5.6.3. Comparison of Normalized Routing Load between Standard TCP and TCP-ELFN with Varying Node Density

Figure 5.14 below presents the performance comparison of the standard TCP and TCP-ELFN when it is supplied with different number of nodes. The figure shows an interesting pattern of the normalized routing load of both TCP mechanisms.

When the nodes are set to 30 and 50, the normalized routing load for TCP-ELFN is much higher than the standard TCP. However, when the number of nodes is 100, TCP-ELFN exceeds the standard TCP performance by having lesser normalized routing load.

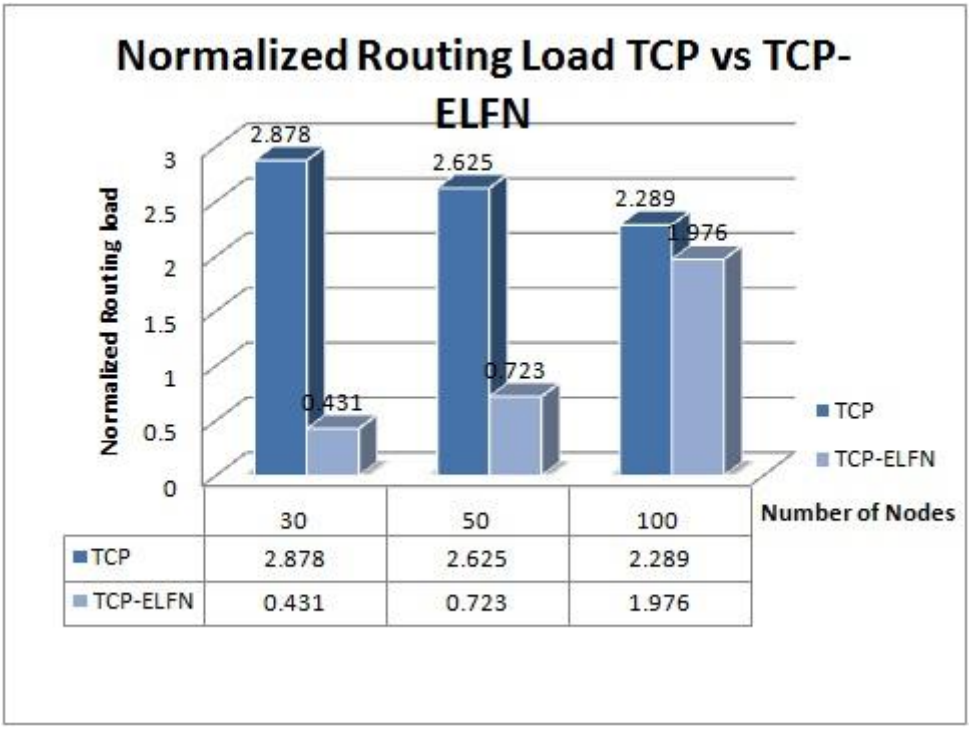

Figure 5. 15 Normalized Routing Load of TCP vs TCP-ELFN with Different Node Density

5.6.4. Comparison of Packet Loss between Standard TCP and TCP-ELFN with Varying Node Speed

Figure 5.16 present the performance comparison of the standard TCP and TCP-ELFN in terms of packet loss when the speed of the travelling nodes is varied. The figure presents a very clear picture that TCP-ELFN performs better than the standard TCP when it comes to packet losses.

The figure also shows that the packet loss decreases dramatically when TCP-ELFN is being used. The reasons as to why such performance is gained will be discussed in the next chapter.

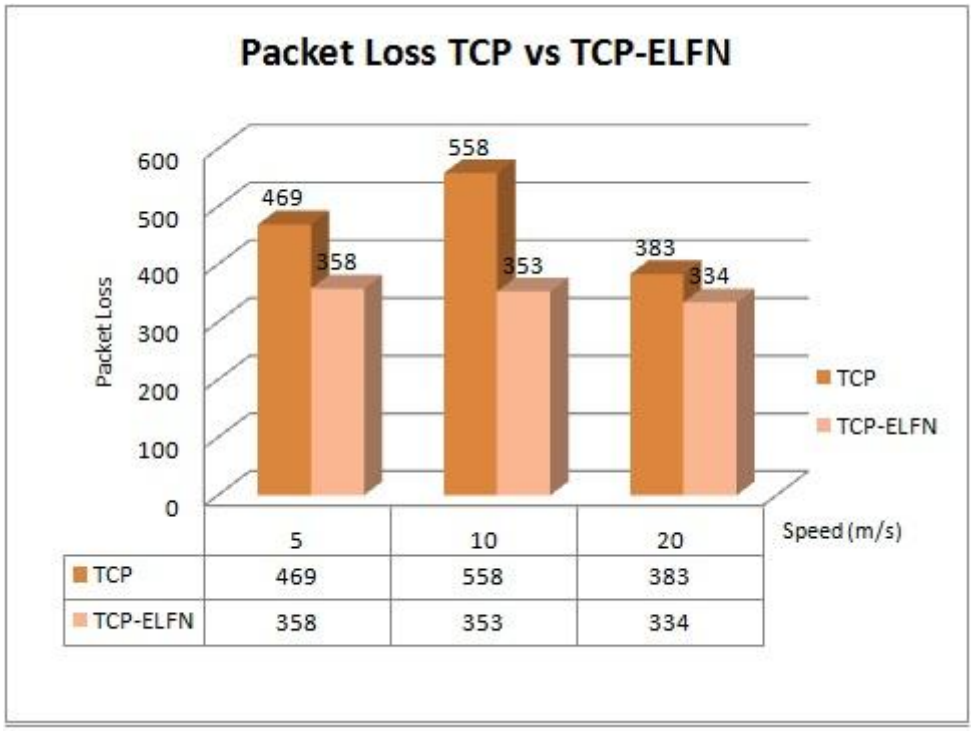

Figure 5. 16 Packet Loss of TCP vs TCP-ELFN with Different Nodes Speed

5.6.5. Comparison of between Average End-to-End Delay between Standard TCP and TCP-ELFN with Varying Node Speed

Figure 5.17 presents the performance comparison of the standard TCP and TCP-ELFN in terms of average end to end delay when the network is set up to have different values for node's speed.

The figure presented that in all three cases where the node's speed is set to 5 m/s, 10 m/s and 20 m/s respectively, standard TCP perform much better by introducing lesser delay as compared to the TCP-ELFN.

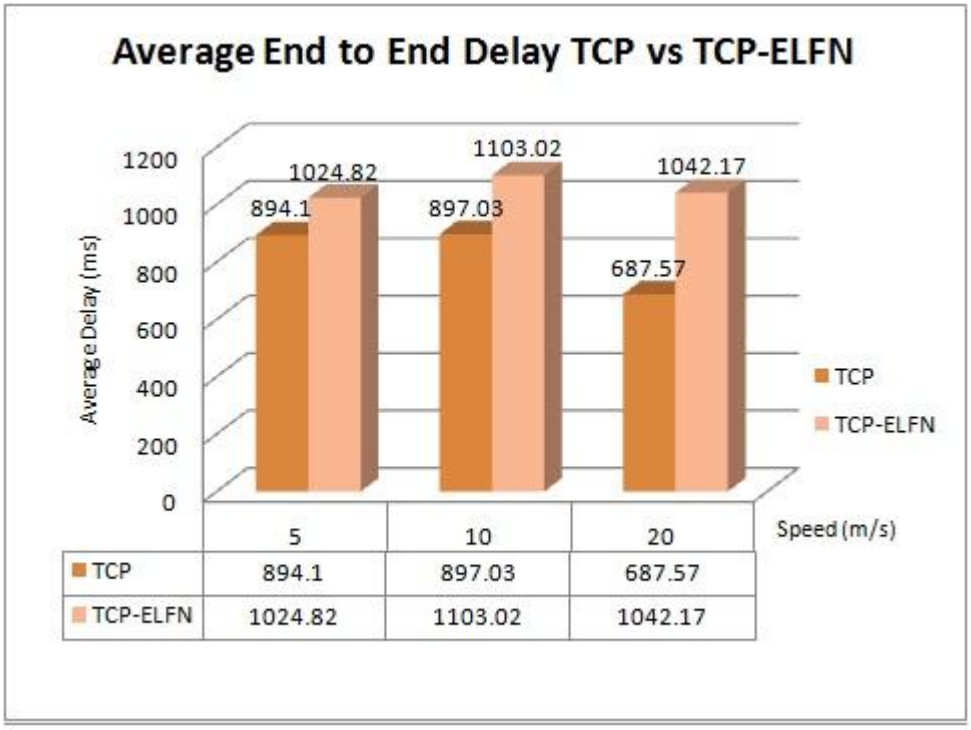

Figure 5. 17 Average End to End Delay of TCP vs TCP-ELFN with Different Nodes Speed

5.6.6. Comparison of Normalized Routing Load between Standard TCP and TCP-ELFN with Varying Node Speed

Figure 5.18 presents the performance of the standard TCP and TCP-ELFN in terms of normalized routing load when the network is supplied with different nodes speed. From the figure we can deduced that TCP-ELFN managed to outperform the standard TCP with its explicit link failure notification mechanism.

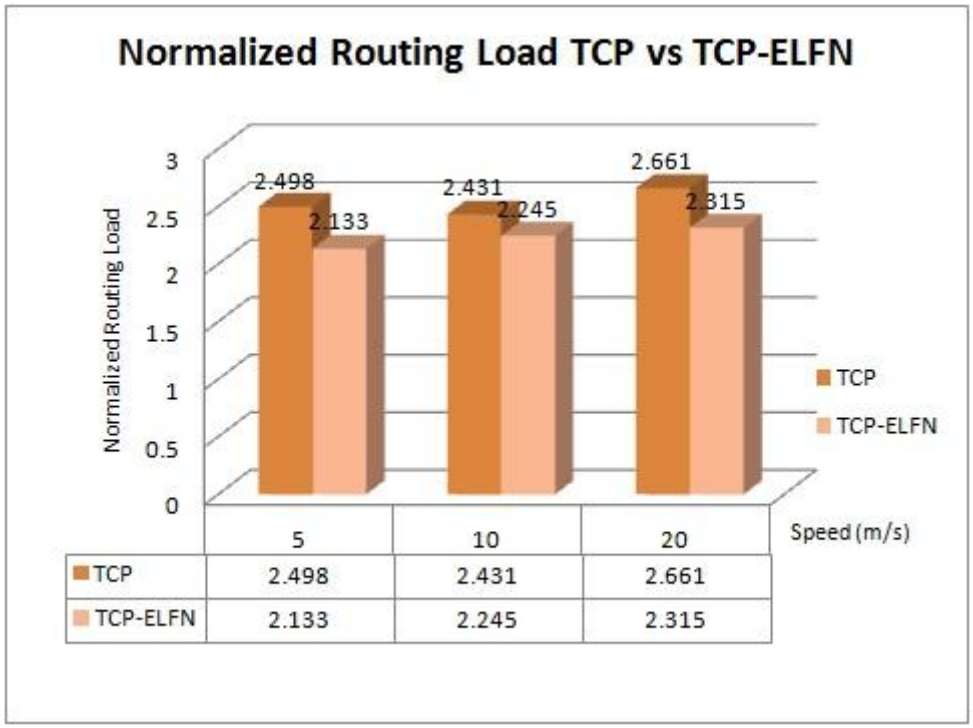

Figure 5. 18 Normalized Routing Load of TCP vs TCP-ELFN with Different Nodes Speed

5.7. Summary

In this chapter, we have presented a thorough performance result for the standard TCP and TCP with explicit link failure notification. We have also presented the performance comparison of these TCP mechanisms. The next chapter will discuss further on the relevance of the performance results that we have gained in this chapter.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

While Chapter 5 dedicated for presenting the performance analysis of the standard TCP and TCP enhanced with Explicit Link Failure Notification in terms of packet loss, average end to end delay, and normalized routing load based on the numerical results obtained from the simulation, this chapter provide the conclusion of the project as a whole. This chapter will also include some suggestions for future work than can be done in the related area.

6.1. Conclusion

In this project, we have studied the effects of mobility and node density on TCP performance in mobile ad hoc network. We have chosen the standard TCP to be compared with TCP-ELFN. Since TCP/IP is a standard network protocol on the Internet, its usage in mobile ad hoc networks is something that is assured. It does not only support a good number of applications but also allow us to seamlessly make use of the Internet. However, many researches that had been carried out have shown that TCP suffers from performance degradation when it is being use in wireless networks due to many factors. Therefore, this project is presented in order to address this issue and to provide its share on this matter.

Additionally, this project was carried out to also address the issue of TCP which is used mainly for wired network in which frequent congestion is expected and congestion avoidance mechanism belongs to TCP such as slow start and exponential back off is exercised. Relatively, when TCP is being used in wireless environment, it still conceives

that congestion is the cause of packet loss, not bit errors. This confusion, will then temper the network performance as the standard TCP will apply the congestion control mechanism whenever it is experiencing packet loss in the network. The congestion control mechanism in standard TCP will slow down the transfer rate unreasonably causing the network to fall into a condition in which it is under utilizing the available bandwidth. TCP-ELFN addresses the issue of the standard TCP in which the standard TCP treats losses induced by route failures as signs of network congestion.

In order to achieve the objectives mentioned in Chapter one, we have presented this project to fulfil these objectives which are to implement and study the impact of comprising TCP-ELFN into mobile ad hoc network simulation, we have also presented the performance comparison of these TCP mechanisms.

From the results presented in Chapter five, we may conclude that the better performance of TCP-ELFN over the standard TCP is due to the reason that, TCP-ELFN provides the TCP sender with information about link and route failures so that it can avoid responding to the failures as if congestion occurred. This means that the TCP performance can be improved by utilizing ELFN in mobile ad hoc. As from the evaluation performance study we have found that TCP with ELFN provides more throughputs compared to standard TCP.

We have seen a significant performance of the simulated mobile ad hoc network when it is supplied with different mobility and node density when TCP-ELFN is in use instead of the standard TCP.

6.2. Suggestions for Future Work

For future work, we would like to suggest the inclusion of other mobile ad hoc routing protocols on the performance of TCP. In addition to this, this project's scalability can be increased by including more number of nodes and more values for nodes speed so that the overall behaviour of the TCP performance can be expanded and studied more thoroughly.

Furthermore, the impact of link layer on the performance of TCP can also be studied. This includes aggregate delay which is caused by local retransmissions over multiple wireless hops. More in depth research are required in order to thoroughly understand the complex interaction between TCP and the lower layer protocols.

Further studies may also include the solutions to these problems in order to provide greater improvement for TCP performance especially in mobile ad hoc networks.

REFERENCES

Abdule, S. M., & Hassan, S. (2010). Divert Failure Route Protocol Based on AODV. *Network Applications Protocols and Services (NETAPPS), 2010 Second International Conference on* , p.p 67 - 71.

Abduljalil, F. M., Bodhe, S. K. (2006). Integrated routing protocol (IRP) for integration of cellular IP and mobile ad hoc networks. *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on* , vol.1, no., pp.4 pp.

Aditya, K., & Anurag, K. (2005). Performance of TCP congestion control with explicit rate feedback. *IEEE/ACM Trans. Netw., 13*(1), 108-120.

Allman, M. (1999). TCP Congestion Control, *Request for comment 2581*.

Altman, E., Barakat, C., Laborde, E., Brown, P., & Collange, D. (2000). *Fairness analysis of TCP/IP*. Paper presented at the Decision and Control, 2000. Proceedings of the 39th IEEE Conference on.

Anjum, F., & Jain, R. (2000). *Performance of TCP over lossy upstream and downstream links with link-level retransmissions.* Paper presented at the Networks, 2000. (ICON 2000). Proceedings. IEEE International Conference on.

Archan, M. (2000). *Dynamics of tcp congestion avoidance with random drop and random marking queues.* University of Maryland at College Park.

Armaghani, F. R., & Jamuar, S. S. (2008). TCP-MAC Interaction in Multi-hop Ad-hoc Networks.

Bakre, A., & Badrinath, B. R. (1995). *I-TCP: indirect TCP for mobile hosts*. Paper presented at the Distributed Computing Systems, 1995. Proceedings of the 15th International Conference on.

Brakno, L. S., O'Malley, S. W., & Peterson, L. L. (1994). TCP Vegas: new techniques for congestion detection and avoidance, *ACM SIGCOMM Computer Communication Review*, Vol. 24, No. 4, pp. 24-35.

Caceres, R., & Iftode, L. (1995). Improving the performance of reliable transport protocols in mobile computing environments. *Selected Areas in Communications, IEEE Journal on, 13*(5), 850-857.

Chandran, K., Raghunathan, S., Venkatesan, S., & Prakash, R. (2001). A Feedback Based Scheme for Improving TCP Performance in Ad-hoc Network. *18th International Conference on Distributed Computing Systems 1998* (pp. 34-39). IEEE 2001.

Chiasserini, C. F., & Meo, M. (2001). Improving TCP over wireless through adaptive link layer setting. *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE* , vol.3, no., pp.1766-1770

Chydzinski, A., & Brachman, A. (2010). *Performance of AQM Routers in the Presence of New TCP Variants.* Paper presented at the Advances in Future Internet (AFIN), 2010 Second International Conference on.

Douglas, E., C. (2005). *Internetworking with TCP/IP (5nd ed.), vol. I*: Prentice-Hall, Inc.

Durresi, A. A., Sridharan, M., Chunlei, L., Goyal, M., & Jain, R. (2001). *Congestion control using multilevel explicit congestion notification in satellite networks.* Paper presented at the Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on.

Eshak, N., & Baba, M. D. (2003). *Improving TCP performance in mobile ad hoc networks.* Paper presented at the Communications, 2003. APCC 2003. The 9th Asia-Pacific Conference on.

Fairhurst, G., Samaraweera, N. K. G., Sooriyabandara, M., Harun, H., Hodson, K., & Donadio, R. (2001). Performance issues in asymmetric TCP service provision using broadband satellite. *Communications, IEE Proceedings-, 148*(2),pp. 95-99.

Floyd, S., & Fall, K. (1999). Promoting the use of end-to-end congestion control in the Internet. *Networking, IEEE/ACM Transactions on, 7*(4),pp. 458-472.

Fu, Z., Luo, H., Zerfos, P., Lu, S., Zhang, L., & Gerla, M. (2005). The impact of multihop wireless channel on TCP performance. *IEEE Transactions on Mobile Computing, 4*(2), pp. 209-221.

Gerla, M., Sanadidi, M. Y., Zanella, R. W., Casetti, A., & Mascolo, S. (2002). TCP Westwood: congestion window control using bandwidth estimation. *IEEE Global Telecommunications Conference*, pp. 1698-1702, 0-7803-7206-9, San Antonio, August 2002, IEEE Computer Society,TX.

Ghanem, T. F., Elkilani, W. S., & Hadhoud, M. M. (2009). *Improving TCP performance over Mobile Ad Hoc Networks using an adaptive backoff response approach.* Paper presented at the Networking and Media Convergence, 2009. ICNM 2009. International Conference on.

Ghazali, O. (2008). *Scaleable and Smooth TCP-friendly receiver-based layered multicast protoco.* Ph.D. Thesis, Universiti Utara Malaysia

Hamrioui, S., & Lalam, M. (2011). *IB-MAC: Improvement of the backoff algorithm for better MAC - TCP protocols interactions in MANET.* Paper presented at the Programming and Systems (ISPS), 2011 10th International Symposium on.

Hassan, M., & Jain, R. (2004). *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*: Pearson Prentice Hall.

Hassan, M., & Raj, J. (2001). TCP performance. *Communications Magazine, IEEE, 39*(4), pp.51-51.

Holland, G., & Vaidya, N. (1999). Analysis of tcp performance over Mobile Ad-Hoc Network. Proceedings of ACM MOBIOM 1999, pp. 257-261 .

Issariyakul, T., Hossain, E. (2009). Introduction to Network Simulator NS2, US:Springer.

Jain, A., Pruthi, A., Thakur, R. C., & Bhatia, M. P., S. (2002). *TCP analysis over wireless mobile ad hoc networks.* Paper presented at the Personal Wireless Communications, 2002 IEEE International Conference on.

Jiwei, C., Yeng Zhong, L., Gerla, M., & Sanadidi, M. Y. (2006). *TCP with Delayed Ack for Wireless Networks.* Paper presented at the Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on.

Jiyong, P., Daedong, P., Seongsoo, H., & Jungkeun, P. (2011). Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing. *Comput. Commun., 34*(6), pp.730-742.

Johanson, D., Maltz, D. A., & Broch, J. (2001). The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (draft-ietf-manet-dsr-06.txt). *Mobile Ad-hoc Network(MANET) Working Group, IETF* .

Kai, Z., Neng, W., & Ai-fang, L. (2005). A new AODV based clustering routing protocol. *Wireless Communications, Networking and Mobile Computing, International Conference on 2005* , vol 2, pp 728 - 731.

Lakshman, T. V., Upamanyu, M., & Bernhard, S. (2000). TCP/IP performance with random loss and bidirectional congestion. *IEEE/ACM Trans. Netw., 8*(5), pp. 541-555.

Liang, G., & Matta, I. (2001). *The war between mice and elephants.* Paper presented at the Network Protocols, 2001. Ninth International Conference on.

Maan, F.; Mazhar, N. (2011). MANET routing protocols vs mobility models: A performance evaluation. *Ubiquitous and Future Networks (ICUFN), 2011 Third International Conference on* , vol., no., pp.179-184, 15-17

Mahbub, H., Raj, J. (2004). High Performance TCP/IP Networking: Concept, Issues and Solution. Pearson Prentice Hall.

Minseok, K., & Sonia, F. (2004). On TCP reaction to Explicit Congestion Notification. *J. High Speed Netw., 13*(2), pp.123-138.

Mittal, S. & Kaur, P. (2009). Performance Comparison of AODV, DSR and ZRP Routing Protocols in MANET'S. *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT '09. International Conference on* , vol., no., pp.165-168.

Monks, J. P., Sinha, P., & Bharghavan, V. (2000). Enhancements and Limitations of TCP-ELFN for Ad Hoc Networks. *Proceedings of The 7th International Workshop on Mobile Multimedia Communications.* MoMuC 2000.

Nehme, A., Phillips, W., & Robertson, W. (2003). The effect of reordering and dropping packets on TCP over a slow wireless link. *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on* , vol.3, no., pp. 1555- 1558.

Postel, J. (1981). Transmission Control Protocol. RFC 0793, Internet Engineering Task

Qian Feng, Zhongmin Cai, Jin Yang, & Xunchao Hu. (2009). A Performance Comparison of the Ad Hoc Network Protocols. *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on* , vol.2, no., pp.293-297.

Romanowicz, E. (2008). TCP with Explicit Link Failure Notification. *Department of Computer Science, York University, Toronto, Canada* ,page 1-9.

Rung-Shiang Cheng, Hui-Tang Lin, Wen-Shyang Hwang, & Ce-Kuen Shieh. (2005). Improving the ramping up behavior of TCP slow start. *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on* , vol.1, no., pp. 807- 812.

Saad, B., & Nitin, H. V. (2005). "De-randomizing" congestion losses to improve TCP performance over wired-wireless networks. *IEEE/ACM Trans. Netw., 13*(3), pp. 596-608.

Sangtae Ha, I. R., & Lisong Xu (2008). CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review - Research and Developments in the Linux Kernel*, pp. 64–74.

Seungwan, R., & Chulhyoe, C. (2004). *PI-PD-Controller for Robust and Adaptive Queue Management for Supporting TCP Congestion Control*. Paper presented at the Proceedings of the 37th annual symposium on Simulation.

Stangel, M., & Bharghavan, V. (1998). *Improving TCP performance in mobile computing environments.* Paper presented at the Communications, 1998. ICC 98. Conference Record.1998 IEEE International Conference on.

Shinde, S. C., Vinayak, J., Ramesh, N., & J, H. (2010). AN EXPLICIT LINK FAILURE NOTIFICATION WITH DYNAMIC CACHE UPDATE SCHEME TO IMPROVE TCP PERFORMANCE USING DSR PROTOCOL IN MANET. *Suneel C Shinde et. al. / International Journal of Engineering Science and Technology Vol. 2(6), 2010,* pp. 2263-2271.

So-In, C., Jain, R., & Dommety, G. (2009). PETS: Persistent TCP using Simple Freeze. *First International Conference on Future Information Networks, October 14-17, Beijing, China* .

Stevens, W. (1997). TCP Slow Start, Congestion Avoidance, Fast Retransmit, *Request for comment 2001*.

Wing-Chung, H., & Law, K.L.E. (2008). Simple slow-start and a fair congestion avoidance for TCP communications. *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on* , vol., no., pp.001771-001774.

Xu, S., & Saadawi, T. (2001). Revealing and solving the TCP instability problem in 802.11 based multi-hop mobile ad hoc networks. *Vehicular Technology Conference, 2001. VTC 2001 Fall. IEEE VTS* (pp. vol. (1), pp 257-261). IEEE.

Y. Iwanaga, K. D. Cavendish, M.Tsuru, and Y. Oie. (2010). *High-speed tcp performance characterization under various operating systems.* Paper presented at the 5th International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2010), Seattle USA.

Yousefi'zadeh, H., Habibi, A., & Furmanski, W. (2006). *A Performance Comparison Study of End-to-End Congestion Control Protocols over MIMO Fading Channels.* Paper presented at the Military Communications Conference, 2006. MILCOM 2006. IEEE.

Zhai, H., Wang, J., Chen, X., & Fang, Y. (2006). Medium access control in mobile ad hoc networks: challenges and solutions. *Wireless Communications and Mobile Computing, 6*(2), 151-170.

Zhang, L. (1986). *Why TCP timers don't work well*. Paper presented at the Proceedings of the ACM SIGCOMM conference on Communications architectures \& protocols.

Zhou, J., Shi, B., Zou, L., & Shen, H. (2003). Improve TCP Performance in Ad Hoc network.

APPENDIX

A- Connection Pattern Script

```
#
# nodes: 30, max conn: 0.5, send rate: 0.0, seed: 1
#
#
# 1 connecting to 2 at time 2.5568388786897245
#
set tcp_(0) [$ns_ create-connection  TCP $node_(1) TCPSink $node_(2) 0]
$tcp_(0) set window_ 32
$tcp_(0) set packetSize_ 512
set ftp_(0) [$tcp_(0) attach-source FTP]
$ns_ at 2.5568388786897245 "$ftp_(0) start"
#
# 4 connecting to 5 at time 56.333118917575632
#
set tcp_(1) [$ns_ create-connection  TCP $node_(4) TCPSink $node_(5) 0]
$tcp_(1) set window_ 32
$tcp_(1) set packetSize_ 512
set ftp_(1) [$tcp_(1) attach-source FTP]
```

```
$ns_ at 56.333118917575632 "$ftp_(1) start"

#

# 4 connecting to 6 at time 146.96568928983328

#

set tcp_(2) [$ns_ create-connection  TCP $node_(4) TCPSink $node_(6) 0]

$tcp_(2) set window_ 32

$tcp_(2) set packetSize_ 512

set ftp_(2) [$tcp_(2) attach-source FTP]

$ns_ at 146.96568928983328 "$ftp_(2) start"

#

# 6 connecting to 7 at time 55.634230382570173

#

set tcp_(3) [$ns_ create-connection  TCP $node_(6) TCPSink $node_(7) 0]

$tcp_(3) set window_ 32

$tcp_(3) set packetSize_ 512

set ftp_(3) [$tcp_(3) attach-source FTP]

$ns_ at 55.634230382570173 "$ftp_(3) start"

#

# 7 connecting to 8 at time 29.546173154165118

#

set tcp_(4) [$ns_ create-connection  TCP $node_(7) TCPSink $node_(8) 0]

$tcp_(4) set window_ 32
```

```
$tcp_(4) set packetSize_ 512

set ftp_(4) [$tcp_(4) attach-source FTP]

$ns_ at 29.5461731541651l8 "$ftp_(4) start"

#

# 7 connecting to 9 at time 7.7030203154790309

#

set tcp_(5) [$ns_ create-connection  TCP $node_(7) TCPSink $node_(9) 0]

$tcp_(5) set window_ 32

$tcp_(5) set packetSize_ 512

set ftp_(5) [$tcp_(5) attach-source FTP]

$ns_ at 7.7030203154790309 "$ftp_(5) start"

#

# 8 connecting to 9 at time 20.48548468411224

#

set tcp_(6) [$ns_ create-connection  TCP $node_(8) TCPSink $node_(9) 0]

$tcp_(6) set window_ 32

$tcp_(6) set packetSize_ 512

set ftp_(6) [$tcp_(6) attach-source FTP]

$ns_ at 20.48548468411224 "$ftp_(6) start"

#

# 9 connecting to 10 at time 76.258212521792487

#
```

```
set tcp_(7) [$ns_ create-connection  TCP $node_(9) TCPSink $node_(10) 0]
$tcp_(7) set window_ 32
$tcp_(7) set packetSize_ 512
set ftp_(7) [$tcp_(7) attach-source FTP]
$ns_ at 76.258212521792487 "$ftp_(7) start"
#
# 9 connecting to 11 at time 31.464945688594575
#
set tcp_(8) [$ns_ create-connection  TCP $node_(9) TCPSink $node_(11) 0]
$tcp_(8) set window_ 32
$tcp_(8) set packetSize_ 512
set ftp_(8) [$tcp_(8) attach-source FTP]
$ns_ at 31.464945688594575 "$ftp_(8) start"
#
# 11 connecting to 12 at time 62.77338456491632
#
set tcp_(9) [$ns_ create-connection  TCP $node_(11) TCPSink $node_(12) 0]
$tcp_(9) set window_ 32
$tcp_(9) set packetSize_ 512
set ftp_(9) [$tcp_(9) attach-source FTP]
$ns_ at 62.77338456491632 "$ftp_(9) start"
#
```

```
# 11 connecting to 13 at time 46.455830739092008
#
set tcp_(10) [$ns_ create-connection  TCP $node_(11) TCPSink $node_(13) 0]
$tcp_(10) set window_ 32
$tcp_(10) set packetSize_ 512
set ftp_(10) [$tcp_(10) attach-source FTP]
$ns_ at 46.455830739092008 "$ftp_(10) start"
#
# 13 connecting to 14 at time 83.900868549896813
#
set tcp_(11) [$ns_ create-connection  TCP $node_(13) TCPSink $node_(14) 0]
$tcp_(11) set window_ 32
$tcp_(11) set packetSize_ 512
set ftp_(11) [$tcp_(11) attach-source FTP]
$ns_ at 83.900868549896813 "$ftp_(11) start"
#
# 14 connecting to 15 at time 155.17211061677529
#
set tcp_(12) [$ns_ create-connection  TCP $node_(14) TCPSink $node_(15) 0]
$tcp_(12) set window_ 32
$tcp_(12) set packetSize_ 512
set ftp_(12) [$tcp_(12) attach-source FTP]
```

```
$ns_ at 155.17211061677529 "$ftp_(12) start"

#

# 15 connecting to 16 at time 39.088702704333095

#

set tcp_(13) [$ns_ create-connection  TCP $node_(15) TCPSink $node_(16) 0]

$tcp_(13) set window_ 32

$tcp_(13) set packetSize_ 512

set ftp_(13) [$tcp_(13) attach-source FTP]

$ns_ at 39.088702704333095 "$ftp_(13) start"

#

# 15 connecting to 17 at time 43.420613009212822

#

set tcp_(14) [$ns_ create-connection  TCP $node_(15) TCPSink $node_(17) 0]

$tcp_(14) set window_ 32

$tcp_(14) set packetSize_ 512

set ftp_(14) [$tcp_(14) attach-source FTP]

$ns_ at 43.420613009212822 "$ftp_(14) start"

#

# 16 connecting to 17 at time 121.92280978985261

#

set tcp_(15) [$ns_ create-connection  TCP $node_(16) TCPSink $node_(17) 0]

$tcp_(15) set window_ 32
```

```
$tcp_(15) set packetSize_ 512

set ftp_(15) [$tcp_(15) attach-source FTP]

$ns_ at 121.92280978985261 "$ftp_(15) start"

#

# 16 connecting to 18 at time 137.20174070317378

#

set tcp_(16) [$ns_ create-connection  TCP $node_(16) TCPSink $node_(18) 0]

$tcp_(16) set window_ 32

$tcp_(16) set packetSize_ 512

set ftp_(16) [$tcp_(16) attach-source FTP]

$ns_ at 137.20174070317378 "$ftp_(16) start"

#

# 17 connecting to 18 at time 72.99343390995331

#

set tcp_(17) [$ns_ create-connection  TCP $node_(17) TCPSink $node_(18) 0]

$tcp_(17) set window_ 32

$tcp_(17) set packetSize_ 512

set ftp_(17) [$tcp_(17) attach-source FTP]

$ns_ at 72.99343390995331 "$ftp_(17) start"

#

# 17 connecting to 19 at time 19.655724884781858

#
```

```
set tcp_(18) [$ns_ create-connection  TCP $node_(17) TCPSink $node_(19) 0]
$tcp_(18) set window_ 32
$tcp_(18) set packetSize_ 512
set ftp_(18) [$tcp_(18) attach-source FTP]
$ns_ at 19.655724884781858 "$ftp_(18) start"
#
# 20 connecting to 21 at time 170.32769159894795
#
set tcp_(19) [$ns_ create-connection  TCP $node_(20) TCPSink $node_(21) 0]
$tcp_(19) set window_ 32
$tcp_(19) set packetSize_ 512
set ftp_(19) [$tcp_(19) attach-source FTP]
$ns_ at 170.32769159894795 "$ftp_(19) start"
#
# 20 connecting to 22 at time 160.44260791523504
#
set tcp_(20) [$ns_ create-connection  TCP $node_(20) TCPSink $node_(22) 0]
$tcp_(20) set window_ 32
$tcp_(20) set packetSize_ 512
set ftp_(20) [$tcp_(20) attach-source FTP]
$ns_ at 160.44260791523504 "$ftp_(20) start"
#
```

```
# 24 connecting to 25 at time 60.419296464146719
#
set tcp_(21) [$ns_ create-connection  TCP $node_(24) TCPSink $node_(25) 0]
$tcp_(21) set window_ 32
$tcp_(21) set packetSize_ 512
set ftp_(21) [$tcp_(21) attach-source FTP]
$ns_ at 60.419296464146719 "$ftp_(21) start"
#
# 26 connecting to 27 at time 46.258873029732555
#
set tcp_(22) [$ns_ create-connection  TCP $node_(26) TCPSink $node_(27) 0]
$tcp_(22) set window_ 32
$tcp_(22) set packetSize_ 512
set ftp_(22) [$tcp_(22) attach-source FTP]
$ns_ at 46.258873029732555 "$ftp_(22) start"
#
# 27 connecting to 28 at time 98.067954088592884
#
set tcp_(23) [$ns_ create-connection  TCP $node_(27) TCPSink $node_(28) 0]
$tcp_(23) set window_ 32
$tcp_(23) set packetSize_ 512
set ftp_(23) [$tcp_(23) attach-source FTP]
```

```
$ns_ at 98.067954088592884 "$ftp_(23) start"

#

# 28 connecting to 29 at time 47.128346453946243

#

set tcp_(24) [$ns_ create-connection  TCP $node_(28) TCPSink $node_(29) 0]

$tcp_(24) set window_ 32

$tcp_(24) set packetSize_ 512

set ftp_(24) [$tcp_(24) attach-source FTP]

$ns_ at 47.128346453946243 "$ftp_(24) start"

#

# 28 connecting to 30 at time 99.87114126788039

#

set tcp_(25) [$ns_ create-connection  TCP $node_(28) TCPSink $node_(29) 0]

$tcp_(25) set window_ 32

$tcp_(25) set packetSize_ 512

set ftp_(25) [$tcp_(25) attach-source FTP]

$ns_ at 99.87114126788039 "$ftp_(25) start"

#

#Total sources/connections: 17/26

#
```

B- Scenario Generation Script

```
#

# nodes: 30, pause: 2.00, max speed: 10.00, max x: 500.00, max y: 500.00

#

$node_(0) set X_ 31.926988307576

$node_(0) set Y_ 197.246781440296

$node_(0) set Z_ 0.000000000000

$node_(1) set X_ 447.115668212077

$node_(1) set Y_ 428.545421254227

$node_(1) set Z_ 0.000000000000

$node_(2) set X_ 44.838136603668

$node_(2) set Y_ 201.169450847256

$node_(2) set Z_ 0.000000000000

$node_(3) set X_ 427.665302868556

$node_(3) set Y_ 272.534532516073

$node_(3) set Z_ 0.000000000000

$node_(4) set X_ 232.332119095391

$node_(4) set Y_ 119.906267020969

$node_(4) set Z_ 0.000000000000

$node_(5) set X_ 258.144871144121

$node_(5) set Y_ 232.136260784311

$node_(5) set Z_ 0.000000000000
```

```
$node_(6) set X_ 335.750015647151

$node_(6) set Y_ 240.990244114299

$node_(6) set Z_ 0.000000000000

$node_(7) set X_ 476.802206638935

$node_(7) set Y_ 155.680072582666

$node_(7) set Z_ 0.000000000000

$node_(8) set X_ 46.485622103561

$node_(8) set Y_ 81.300723229197

$node_(8) set Z_ 0.000000000000

$node_(9) set X_ 62.080091776480

$node_(9) set Y_ 199.247276164459

$node_(9) set Z_ 0.000000000000

$node_(10) set X_ 166.759728963027

$node_(10) set Y_ 391.548965462061

$node_(10) set Z_ 0.000000000000

$node_(11) set X_ 354.876316458516

$node_(11) set Y_ 214.607784091127

$node_(11) set Z_ 0.000000000000

$node_(12) set X_ 30.113759512003

$node_(12) set Y_ 73.631237101578

$node_(12) set Z_ 0.000000000000

$node_(13) set X_ 436.468665438118
```

```
$node_(13) set Y_ 157.647610826116
$node_(13) set Z_ 0.000000000000
$node_(14) set X_ 63.061016871755
$node_(14) set Y_ 425.920973902466
$node_(14) set Z_ 0.000000000000
$node_(15) set X_ 268.712966910633
$node_(15) set Y_ 289.879060069761
$node_(15) set Z_ 0.000000000000
$node_(16) set X_ 98.006801187390
$node_(16) set Y_ 51.180205691020
$node_(16) set Z_ 0.000000000000
$node_(17) set X_ 438.912066466573
$node_(17) set Y_ 239.156324240499
$node_(17) set Z_ 0.000000000000
$node_(18) set X_ 162.176087436412
$node_(18) set Y_ 482.499850066371
$node_(18) set Z_ 0.000000000000
$node_(19) set X_ 294.132893364793
$node_(19) set Y_ 132.562380533892
$node_(19) set Z_ 0.000000000000
$node_(20) set X_ 228.285449902381
$node_(20) set Y_ 30.243940719110
```

```
$node_(20) set Z_ 0.000000000000
$node_(21) set X_ 469.269940771197
$node_(21) set Y_ 79.467547947832
$node_(21) set Z_ 0.000000000000
$node_(22) set X_ 353.722327988186
$node_(22) set Y_ 97.452030929276
$node_(22) set Z_ 0.000000000000
$node_(23) set X_ 9.404743196486
$node_(23) set Y_ 88.328530982215
$node_(23) set Z_ 0.000000000000
$node_(24) set X_ 11.547222244978
$node_(24) set Y_ 178.605442785176
$node_(24) set Z_ 0.000000000000
$node_(25) set X_ 432.686769636080
$node_(25) set Y_ 452.719993811510
$node_(25) set Z_ 0.000000000000
$node_(26) set X_ 194.909069580279
$node_(26) set Y_ 396.282178449140
$node_(26) set Z_ 0.000000000000
$node_(27) set X_ 8.836704152179
$node_(27) set Y_ 222.408457370698
$node_(27) set Z_ 0.000000000000
```

```
$node_(28) set X_ 461.428069038558
$node_(28) set Y_ 417.129034936158
$node_(28) set Z_ 0.000000000000
$node_(29) set X_ 210.255688227057
$node_(29) set Y_ 113.155918786403
$node_(29) set Z_ 0.000000000000
$god_ set-dist 0 1 3
$god_ set-dist 0 2 1
$god_ set-dist 0 3 2
$god_ set-dist 0 4 1
$god_ set-dist 0 5 1
$god_ set-dist 0 6 2
$god_ set-dist 0 7 2
$god_ set-dist 0 8 1
$god_ set-dist 0 9 1
$god_ set-dist 0 10 1
$god_ set-dist 0 11 2
$god_ set-dist 0 12 1
$god_ set-dist 0 13 2
$god_ set-dist 0 14 1
$god_ set-dist 0 15 2
$god_ set-dist 0 16 1
```

```
$god_ set-dist 0 17 2

$god_ set-dist 0 18 2

$god_ set-dist 0 19 2

$god_ set-dist 0 20 2

$god_ set-dist 0 21 2

$god_ set-dist 0 22 2

$god_ set-dist 0 23 1

$god_ set-dist 0 24 1

$god_ set-dist 0 25 3

$god_ set-dist 0 26 2

$god_ set-dist 0 27 1

$god_ set-dist 0 28 3

$god_ set-dist 0 29 1

$god_ set-dist 1 2 2

$god_ set-dist 1 3 1

$god_ set-dist 1 4 2

$god_ set-dist 1 5 2

$god_ set-dist 1 6 1

$god_ set-dist 1 7 2

$god_ set-dist 1 8 3

$god_ set-dist 1 9 2

$god_ set-dist 1 10 2
```

```
$god_ set-dist 1 11 1

$god_ set-dist 18 25 2

$god_ set-dist 18 26 1

$god_ set-dist 18 27 2

$god_ set-dist 18 28 2

$god_ set-dist 18 29 2

$god_ set-dist 19 20 1

$god_ set-dist 19 21 1

$god_ set-dist 19 22 1

$god_ set-dist 19 23 2

$god_ set-dist 19 24 2

$god_ set-dist 19 25 2

$god_ set-dist 19 26 2

$god_ set-dist 19 27 2

$god_ set-dist 19 28 2

$god_ set-dist 19 29 1

$god_ set-dist 20 21 1

$god_ set-dist 20 22 1

$god_ set-dist 20 23 1

$god_ set-dist 20 24 2

$god_ set-dist 20 25 2

$god_ set-dist 20 26 2
```

```
$god_ set-dist 20 27 2

$god_ set-dist 20 28 2

$god_ set-dist 20 29 1

$god_ set-dist 21 22 1

$god_ set-dist 21 23 2

$god_ set-dist 21 24 2

$god_ set-dist 21 25 2

$ns_ at 92.436597949187 "$god_ set-dist 17 18 2"

$ns_ at 92.548596411188 "$node_(24) setdest 181.087107669124 329.677617194351
2.486345206192"

$ns_ at 92.644727459786 "$god_ set-dist 15 26 1"

$ns_ at 92.795371705417 "$god_ set-dist 5 19 2"

$ns_ at 93.293096386952 "$god_ set-dist 8 12 1"

$ns_ at 93.321191711099 "$god_ set-dist 3 23 2"

$ns_ at 93.440183836333 "$god_ set-dist 3 25 2"

$ns_ at 93.629023985012 "$god_ set-dist 20 25 2"

$ns_ at 93.842779672485 "$god_ set-dist 4 20 1"

$ns_ at 94.164961121710 "$god_ set-dist 2 5 2"

$ns_ at 94.409117043416 "$god_ set-dist 15 17 2"

$ns_ at 94.511008271171 "$god_ set-dist 4 22 1"

$ns_ at 94.537273915403 "$god_ set-dist 2 28 1"

$ns_ at 94.774415123399 "$god_ set-dist 9 21 1"

$ns_ at 94.848365470711 "$god_ set-dist 2 17 2"
```

$ns_ at 95.050234868142 "$god_ set-dist 5 10 2"

$ns_ at 95.771008437172 "$god_ set-dist 14 15 1"

$ns_ at 96.113474325032 "$god_ set-dist 4 6 1"

$ns_ at 96.150110209805 "$god_ set-dist 1 25 2"

$ns_ at 96.192538705210 "$god_ set-dist 17 23 1"

$ns_ at 96.219304902332 "$god_ set-dist 18 20 1"

$ns_ at 96.286366807687 "$god_ set-dist 1 15 1"

$ns_ at 96.547254162272 "$god_ set-dist 17 27 2"

$ns_ at 96.727313992721 "$god_ set-dist 9 17 2"

$ns_ at 96.944558814991 "$node_(25) setdest 53.014235344473 136.257788245943 0.000000000000"

$ns_ at 97.088345890550 "$god_ set-dist 2 24 1"

$ns_ at 97.089805804361 "$god_ set-dist 21 22 1"

$ns_ at 97.166464566684 "$node_(17) setdest 390.643896361331 194.337828998051 0.000000000000"

$ns_ at 98.532815572419 "$node_(7) setdest 226.043426086435 288.327608716707 0.000000000000"

$ns_ at 98.560974809807 "$god_ set-dist 10 26 2"

$ns_ at 98.944558814991 "$node_(25) setdest 254.459907866004 305.639650922513 9.229589642434"

$ns_ at 99.166464566684 "$node_(17) setdest 483.473752872679 269.346610532164 8.301287413730"

$ns_ at 99.571635006289 "$node_(26) setdest 218.162163481519 311.048963968219 0.000000000000"

```
$ns_ at 99.786028756682 "$god_ set-dist 17 23 2"

$ns_ at 99.878211841276 "$node_(9) setdest 216.340270009538 394.061022429888
0.000000000000"

$ns_ at 99.911538226694 "$god_ set-dist 2 10 2"

#

# Destination Unreachables: 0

#

# Route Changes: 578

#

# Link Changes: 483

#

# Node | Route Changes | Link Changes

#    0 |            33 |           20

#    1 |            44 |           37

#    2 |            33 |           31

#    3 |            30 |           30

#    4 |            31 |           25

#    5 |            23 |           23

#    6 |            25 |           25

#    7 |            36 |           32

#    8 |            40 |           30

#    9 |            42 |           42

#   10 |            38 |           38
```

```
#    11 |          53 |          33
#    12 |          46 |          40
#    13 |          46 |          38
#    14 |          41 |          24
#    15 |          53 |          33
#    16 |          28 |          18
#    17 |          40 |          38
#    18 |          35 |          26
#    19 |          25 |          25
#    20 |          26 |          23
#    21 |          48 |          38
#    22 |          41 |          39
#    23 |          47 |          43
#    24 |          52 |          48
#    25 |          55 |          42
#    26 |          40 |          40
#    27 |          33 |          30
#    28 |          41 |          24
#    29 |          31 |          31
#
```