Performance Evaluation of Routing Metrics for Wireless Mesh Networks

by

Siphamandla Lindelani Nxumalo

(20032463)

(B.Sc. Hons. Computer Science)

Supervisors: Prof M.O. Adigun and Dr. N. Ntlatlapa

This dissertation is submitted to the Department of Computer Science at the University of Zululand's Faculty of Science and Agriculture, in fulfilment of the requirements for the degree of Master of Science in Computer Science

2011

DECLARATION

This dissertation represents the author's original work, conducted at the University of Zululand. It is submitted for the degree of Master of Science in Computer Science in Faculty of Science and Agriculture, at the University of Zululand, KwaDlangezwa. No part of this research has been submitted in the past, or is being submitted, for a degree or examination at any other University. All sources used in this dissertation have been duly acknowledged.

Signature 

NXUMALO SL

DEDICATION

I dedicate this piece of work to my mother, Ms. E K Nxumalo and my daughter,

Aphelele Owethusonke Nxumalo.

ACKNOWLEDGEMENTS

I would like to start by acknowledging members of my family especially my mother, Ms EK Nxumalo, who has always been there for me through thick and thin. Many thanks go to my supervisors, Prof. M.O. Adigun of University of Zululand and Dr. N. Ntlatlapa of CSIR, Meraka institute for guiding me throughout this research work. I would also like to thank department of computer science at the University of Zululand for allowing me to pursue my studies. My appreciation also goes to Thulani Nyandeni for mentoring me in my first year of masters and making sure that I get a solid foundation and introduction to research and wireless networks. Special thanks go to Pragasen Mudali and Bethel Mutanga for supporting and advising me throughout the duration of this project.

I would like to extend my gratitude to my friend and colleague Sakhile Ncanana for his assistance and brotherly advice during hard times. Many thanks to the Wireless Mesh Networks research group and all research colleagues in the department of computer science for making sure I was making progress on the right direction throughout my project. I am humbled by financial support provided by Telkom, Thrip, and Huawei to the department of computer science at the University of Zululand through the CoE program. My gratitude to my sponsor CSIR, Meraka Institute for financial support they provided me with for the duration of this project. Mostly I would like to say that I am grateful to God, my creator for giving me strength and courage to keep going throughout the duration of my study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

An optimal routing metric has a potential to improve performance of a wireless network. A number of routing metrics were designed with mobile ad hoc networks (MANETs) in mind. These routing metrics might be useful in wireless mesh networks (WMNs) but the fact that WMNs and MANETs are two different types of wireless networks with different features might not allow optimal performance of some or all existing routing metrics that were designed for MANETs. These differences in WMNs and MANETs make it important to firstly test if these existing routing metrics can work in WMNs. These existing routing metrics have been compared by different scholars in the literature, but the manner in which they were compared was not consistent, which makes it difficult to draw conclusions as to which routing metric works best for WMNs. The goal of this study was to evaluate the performance of existing routing metrics for Wireless Mesh Networks with a view to designing an optimal one.

The goal of this work was achieved by evaluating the performance of existing routing metrics through NS2 simulation tool installed in Ubuntu Linux version 9.10. Recommendation of design criteria for designing an optimal routing metric for WMNs. Four routing metrics (HOP, ETX, RTT, and EETT) were chosen for evaluation after comparing twenty existing routing metrics. As the first part of the researcher's contribution to knowledge, the four (hop count, expected transmission count, per-hop round trip time,

and exclusive expected transmission) routing metrics were compared under a consinstant manner (using the same routing protocol, performance metric, and WMNs environment).

The overall results of the experiments show that ETX outperformed all the other routing metrics that were simulated. However, the study has shown that ETX is not the best routing metric for WMNs. Thus as the second part of contribution to the body of knowledge, promting the researcher to further recommend four features for designing an optimal routing metric for WMNs. The four features proposed by the researcher were wight path-awareness, efficient wight path algorithm design, quality of service-awareness, and network scalability.

CHAPTER ONE

INTRODUCTION

1.1 Preamble

A wireless multi-hop network is a network of nodes which are connected by wireless communication links, the links are most often implemented with digital packet radios (De Couto, 2003). Nodes must make use of intermediate nodes to forward packets to the intended destination node, because a node can not directly communicate with all the nodes in the network. A node is a communication device that is capable of sending, receiving, and relay packets. Wireless Mesh Networks (WMNs), Mobile Ad hoc Network (MANET), and Wireless Sensor Networks form the family of Ad hoc Multi-hop Wireless Networks (see Figure 1.1).

Different from mobile ad hoc networks and sensor networks, WMNs architecture introduces a gateway which allows for connection to the internet (see Figure 1.1), but not all WMNs are connected to the internet. Wireless mesh networks are dynamically self-organized and self-configured, with the nodes in the network automatically establishing an ad hoc network and maintaining the mesh connectivity (Akyildiz, Wang & Wang, 2005) (see Figure 1.2). WMNs are usually comprized of three types of nodes: mesh clients, mesh routers, and gateway.

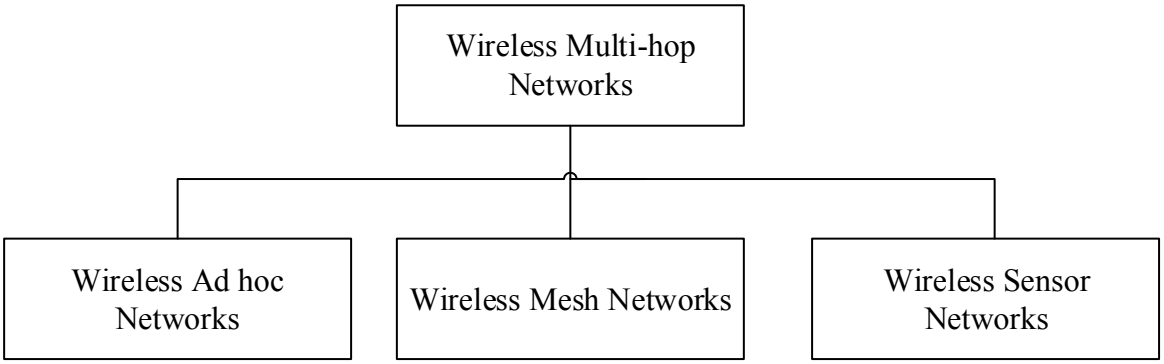

Figure 0.1: Types of Multi-hop Wireless Networks

Nodes do not operate only as hosts but can also operate as mesh routers, mesh clients, or a gateway. In WMNs not all nodes can directly communicate with any other node; relay nodes are used to pass packets across the network (Ramanathan & Redi, 2002). WMNs employ multi-hop communications to forward traffic and route to and from wired Internet entry points.

WMNs are scalable in their deployment. Since the nodes can join or leave at anytime as long as they operate software compatible with other nodes in the network, the area of coverage can be extended by just putting nodes where they can communicate with existing network nodes. WMNs are self-healing because nodes in WMNs dynamically learn their neighbors as well as links to other nodes, there is automatic compensation for the failure or removal of a node (Held, 2005).

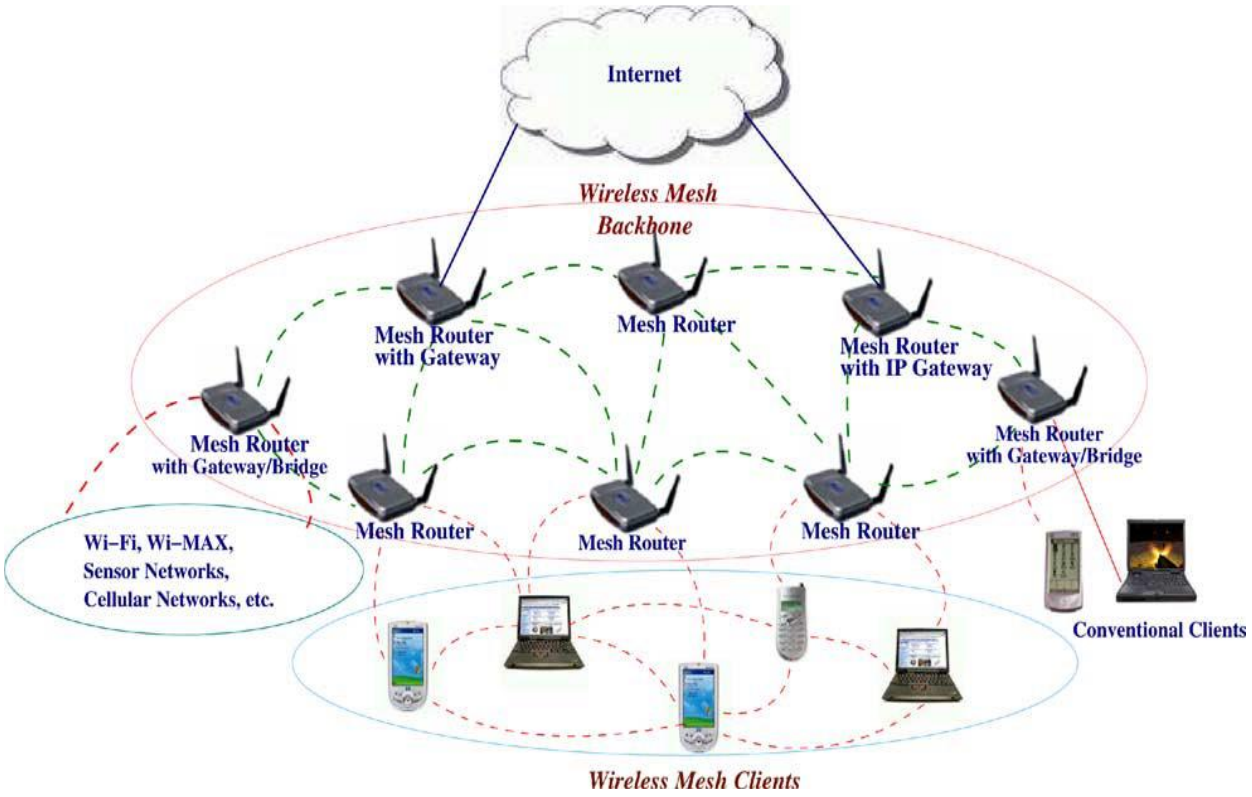

Figure 0.2: A typical example of WMNs (Akyildiz et. al., 2005)

WMNs are also reliable networks, this is achieved through the fact that WMNs nodes function as relay to forward packets to their intended destination node. Since the nodes can join or leave the network at anytime, each node must be able to change its forwarding pattern depending on its neighbors. The failure of a certain route will lead to packets being transmitted through a different route to the destination, in that way reliability is assured in WMNs.

One of the advantages of WMNs is the ease of deployment. Only a few members of the community can be trained and be able to configure and deploy nodes to setup a WMN in a community (Held, 2005). Using WMNs also reduces installation costs. Since the nodes are wirelessly connected to each other, there are no wires required to link the nodes and wires may be quiet expensive. Because WMNs neither require a central administrator nor manual

configuration, they are less costly than centrally controlled networks. WMNs are self-configured; a node is responsible for finding out which nodes is it connected to.

Some of the potential application scenarios of WMNs include broadband home networking, community and neighborhood networking, metropolitan area networks, enterprise networking, health and medical systems, transportation systems, and security surveillance systems. A best performing wireless mesh network should try by all means to provide quality of service (QoS).

An optimal path is the path that satisfies QoS demands such as delay and throughput. QoS is the collective effect of service performance which determines the degree of satisfaction of a user of the service (Bruin, 2006). Routing is the process of selecting optimal path through which a packet is to be sent. Each node in a network needs a routing protocol to select the best path through which a packet should be sent from the source node to the intended destination node. Each intermediate node is capable of relaying packets for its neighbor node through the required route to the destination node.

1.2 Background

This section presents the background about routing protocols types of routing protocols that exist in literature. Subsection 1.2.1 discuses reactive routing protocol, 1.2.2 discuses reactive protocols, while 1.2.3 discuses hybrid routing protocols.

1.2.1 Reactive Routing Protocols

This type of routing protocols was firstly proposed for mobile ad hoc networks. Reactive routing protocols (e.g., dynamic source routing protocol (Johnson & Maltz, 1996), ad hoc on demand vector routing protocol (Perkins, 1997) only establish a path between a source and destination pair when the source node has packets to send to a destination node. A method usually used to discover routes when they are needed is a method known as network-wide flooding. The network-wide flooding method reduces traffic overhead at the cost of increasing latency of finding the route to the destination (Jacquet, Muhlethaler, Clausen, Laouiti, Qayyum & Viennot 2001).

In WMNs, links normally have longer expected lifetimes because of the fact that WMNs nodes are static. Since the frequency of link breaks is much lower than that of flow arrivals in WMNs, flooding-based route discovery method is not only expensive in terms of control message overhead, but also redundant.

1.2. 2 Proactive Routing Protocols

Proactive routing protocols or table driven protocols are types of routing protocols in which every node keeps at least one table that contains routing information to all other nodes in the WMN. Each node keeps updating these tables to maintain consistency of the information in the network. Every time there is a change in the topology of the network, each node sends update messages to other nodes in the network to maintain consistent routing information in the network. Proactive routing protocols differ in the method they use to forward packets along routes. A proactive protocol can either use source routing or hop-by-hop routing.

i. Source routing

Source routing enforces minimal load on the intermediate nodes, because the source node determines the path for a flow. The entire route from the source node to the destination node is stored in the packet headers. Relay nodes are only required to relay packets according to the routes stored in the packet headers. One major disadvantage of source routing method is that putting the entire route in the packet header enforces high message overheard, since packet in wireless network is normally too small to cope with high bit error rate of wireless channels

ii. Hop-by-hop routing

Each node in the network maintains a routing table that has the information of the next hops to each node in the network. A packet only needs to carry the destination address for it to reach its destination. Hop-by-hop is used mostly in wired networks because of its simple forwarding scheme and low message overhead. Owing to these two reasons, hop-by-hop routing is preferred for mesh networks. Although hop-by-hop routing possesses several advantages, it requires careful design of its routing metrics to make sure that there is loop-free forwarding of packets.

1.2.3 Hybrid routing protocol

Hybrid routing combines both proactive and on-demand routing mechanisms to transport a packet from the source node to the destination node. Hybrid routing takes the advantages of both the proactive and the on-demand routing methods. Hybrid Wireless Mesh Protocol (HWMP) is an example of a hybrid routing method. Since HWMP is a hybrid routing protocol, it composed of both reactive and proactive capabilities. HWMP is based on the AODV routing protocol, and it has a configurable extension for proactive routing towards so called mesh portals. The AODV

used by HWMP is called RadioMetric-AODV (RM-AODV) (Aoki et. al., 2005). HWMP uses MAC addresses (layer 2 routing), instead of IP addresses used in layer 3 routing.

1.3 Statement of the Problem

There are a lot of routing metrics that were originally proposed for MANETs and then later adopted for WMNs, but routing in wireless mesh network still needs more attention. There is a need to find out which routing metric works best for WMNs among the existing routing metrics. The existing routing metrics have not been compared using the same routing protocol, performance metrics, and WMNs environment (Draves, Padhye & Zill, 2004a; Yang, Wang & Kravets, 2006; Liu, Huang & 2006). Existing routing metrics need to be compared using the same protocol, performance metrics, and WMNs environment. Such an arrangement will determine if existing routing metrics can be used in WMNs and, if so, which one works best. Therefore, this research formulated a strategy for this investigation in form of research questions:

i. What are the pitfalls in existing WMNs routing metrics?

ii. Which routing metric among existing routing metrics is the best for WMNs?

iii. What features should be considered when designing a routing metric for WMNs?

1.4 Rationale of the Study

There was a need to compare the existing routing metrics in a consistent manner and then recommend the routing metric that works best for WMNs. The analysis shows if there is a need for a new routing metric designed specifically for WMNs. Good performance of the existing

routing metrics in WMNs or coming up with an optimal routing metric is critical for the good performance in WMNs.

Successful completion of this research project will benefit the research community because any scholar who wants to come up with a new routing metric for WMNs will be able to use design criteria that were recommended at the end of Chapter Four of this work, after the evaluation that was conducted in a consistent manner.

1.5 Research Goal and Objectives

Subsection 1.5.1 presents the goal of this research work. The goal is further divided into four research objectives in subsection 1.5.2.

1.5.1 Research Goal

The goal of this research project is to evaluate the performance of existing routing metrics for Wireless Mesh Networks with a view to designing an optimal one.

1.5.2 Research Objectives

The goal of this work can be decomposed into four research objectives that are listed as follows:

i. To survey relevant literature on existing routing metrics
ii. To identify the performance parameters for selecting routing metrics

iii. To simulate, analyze, and evaluate the performance of the existing routing metrics

iv. To recommend design criteria for an optimal routing metric for WMNs

1.6 Overview of Research Methodology

This section overviews the research methods that were used in this research project. The researcher used two research methods to complement each other to fulfill the research objectives listed. Simulation was used as a primary research method, complemented by case-study as the secondary research method.

1.6.1 Primary Research Method: Simulation

Other researchers haved used a testbed to evaluate the performance of the reouting metrics. However, the researcher did not have a bigger and real life testbed to run their experiments on. The researcher had access to a 49-node testbed, howeve this testbed had too few nodes to run the eperiments for this study. As a result, this study has only focused on the simulation. The researcher proposed that there is a need for another study that can further validate the results of this study through the use of a testbed. This is included in the future work section of this document.

Literature survey of routing metrics using a categorization framework that was designed in Chapter Two was firstly conducted before the routing metrics could be simulated. The framework that was used for literature review has the following parameters that were considered: metric name, year, problem addressed by the routing metric, solution approach, advantages,

disadvantages, quality of service-awareness, performance metrics considered, validation method, and routing protocol(s).

Literature survey was conducted so as to study and compare existing routing metrics, and then select routing metrics to be simulated. Literature survey fulfilled objectives one and two. Twenty different routing metrics were reviewed and grouped into four groups in Chapter Three. Routing metrics in each group were compared among themselves, so as to choose one routing metric to represent the group when conducting simulation. There were four routing metrics in total that were chosen for simulation.

The four selected routing metrics were simulated using a network simulation tool called NS2 (Network Simulator version 2) 2.34. NS2 is a discrete event simulator for networking research based on standard C++ and OTcl. The simulation environment that was used to model the test bed is Ubuntu Linux version 9.10. The simulation measured five performance metrics (delay, throughput, packet loss ratio, delay jitter, packet drop ratio). A square grid network topology was used, and AODV routing protocol. The researcher used the simulation area of 1500 x 1500 square units. Transmission range of the nodes was 150m.

While conducting the experiments, the network size was varied. Up to 196 nodes were simulated so as to cater for network scalability. Each experiment was run ten times, and the average of the ten runs was taken. NS2 was used to generate traffic files. The traffic flow was increased as the number of nodes increased. A Tcl script was used to run the simulation and generate trace files.

Trace files were analyzed using trace graph. The analyses from the trace graph were used to generate graphs that were in turn used to interpret the behavior of the routing metrics.

Objective number four was fulfilled by recommending design criteria for an ideal routing metric for Wireless Mesh Networks. The recommendation of design criteria also answered the first and the third research question. The simulated routing metrics' performance was evaluated in Chapter Four. The performance evaluation led to concluding as to which routing metric is the best of all the four routing metrics that were simulated. Design features were proposed to help any scholar that wants to design a new routing metrics from scratch or improve one of the existing routing metrics.

1.6.2 Secondary Research Method: Case-Study

Case-study was the second research method used to complement the primary method. Simulation as a research methodology was unable to fulfill objective three as well as answering the second research question. A case-study was chosen to evaluate the routing metrics and help answer the second research question. The case-study was used to evaluate the performance of the four routing metrics that were simulated. Performance evaluation of routing metrics answered the second research question, while also fulfilling the third research objective. The performance of each routing metric was compared to the performance of the other routing metrics that were also simulated.

1.8 Organization of the Dissertation

The rest of this thesis is organized as follows: Chapter Two of this work reviews literature of existing comparisons of routing metrics. Up to twenty existing routing metrics were reviewed in Chapter Three. A framework for literature review was designed and used to select representative samples. Chapter Three presents both pseudo code and flowcharts specifications for routing metrics that were simulated for our evaluation. Chapter Four starts by describing simulation environment, and it then presents the experimental setup. Chapter Four presents the analysis of the results of the experiments that were conducted before finalizing by recommending design criteria based on our analysis of the results. Chapter Five presents limitations and future works of the study.

CHAPTER TWO

BACKGROUND STUDY

2.1 Introduction

A routing protocol uses a criterion for selecting the best path, this measurement is known as a routing metric. A routing protocol without a routing metric is like a car without an engine, because the routing protocol makes its path selection using the routing metric. A routing metric is the nucleus of a routing protocol. Figure 2.1 shows the relationship between these two components of routing. A node needs a mechanism for selecting a best path through which to transmit a packet. The process of selecting the optimal path through which to send the packet is called routing. Routing is performed in each router so as to find the next hop for the packet.

It was explained in Chapter One that there is a need to compare existing routing metrics in a consistent manner, so as to conclude if there is a routing metric among existing routing metrics that works best for WMNs. Comparison is important because routing metrics need to be compared using the same performance metrics, same environment, and same routing protocol have to be used to make the comparison consistent. The network configurations are also kept the same for all the experiments. This chapter provides the framework that was used for analyzing the routing metrics. It also discusses existing routing metrics, before providing a thorough discussion of the selected routing metrics.

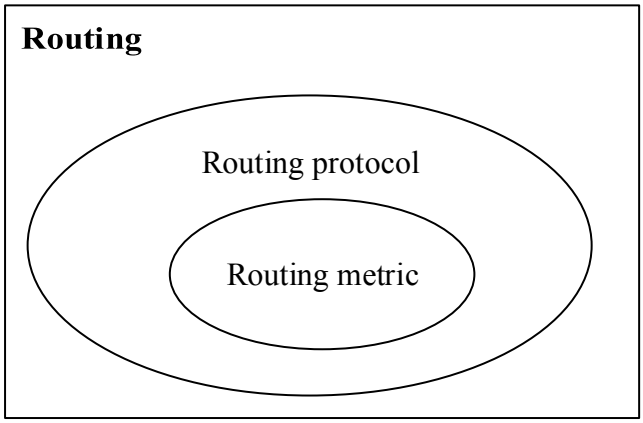

Figure 0.1: Relationship between a routing protocol and a routing metric

Routing in wireless networks has been a problem for a very long time. In an attempt to solve the problem, (Dijkstra, 1959) came up with an algorithm for calculating the shortest path. This routing metric might be simple; the shortest path to the receiver is not always the best one (Draves et al., 2004a) as there are a lot other factors to be considered such as link quality. The approach may not result in the best route as the chosen route may have a high packet loss ratio, or be highly congested (Akyildiz & Wang, 2009; Liu et al., 2008).

This chapter presents what has been done by other scholars in trying to solve the problem of routing in wireless networks. Two works that compare the routing metrics were presented and analyzed in this chapter. The analysis focuses more on the consistency of the comparisons of the routing metrics.

Most of the comparisons done on routing metrics compared routing metrics in an inconsistent manner. An ideal comparison of routing metrics needs to be based on the same routing protocol, same performance metrics, same experimental setup, and same network configuration for all the experiments conducted so as to compare the routing metrics. This work is trying to have routing

metrics compared in a consistent manner. In section 2.2 we discuss the existing comparisons of the existing routing metrics, while 2.3 concludes the chapter.

2.2 Existing Comparisons of Routing Metrics

In this work, we wanted to review specially works that compared routing metrics in a wireless mesh networks setup. We could have reviewed a lot of studies that compared routing metrics, but we obted for the only two that compares routing metrics for WMNs, because this work is specific to WMNs. This work discusses two comparisons of routing metrics in WMNs found in the literature. A lot of efforts have been put into finding solutions for routing in wireless mesh networks (WMNs), and there is still a lot of research going on in this field, resulting in more and more routing metrics being proposed. On the other hand, scholars have carried out research that compares the performance of routing metrics.

2.2.1 Designing Routing Metrics for Mesh Networks (Yang et al., 2006)

The authors of this work discussed five existing routing metrics (hop count, expected transmission count, expected transmission time, weighted cumulative expected transmission time, and metric of interference and channel-switching). Performance evaluation of the routing metrics was done from the results obtained from NS2 simulation. ETX was not considered when the routing metrics were simulated in NS2, and we feel that it should have also been considered for consistent analysis of the five routing metrics. The comparison was only based on four routing metrics out of a possible five as they were discussed early in the same work by these authors. It is not clear which routing protocol was used by authors in this work.

The same performance metrics (maximum channel utilization, network throughput, end-to-end packet delay) were used for all routing metrics. Authors in this work randomly generated 10 networks with 100 nodes and 10 networks with 160 for the experiments, and we feel that they should have varied the network size rather than using only two network sizes (100 and 160). The performance of the routing metrics was compared. MIC outperformed all the other routing metrics that it was simulated with, while HOP was the worst performer. WCETT performed better than ETT while it performed worse than both HOP and MIC. HOP performed better than both ETT and WCETT, while it performed worse when compared to MIC.

2.2.2 A Comprehensive Comparison of Routing Metrics for Wireless Mesh Networks (Liu et al, 2008)

The authors in this work compared 10 different routing metrics designed for diverse quality of service (QoS) requirements in WMNs. Work by (Liu et al, 2008) started off by classifying routing metrics into three groups. This is the approach that our own work followeed (see Table 3.2), work by (Liu etal, 2008) firstly groups routing metrics into three groups: ETX-based, mETX, and other routing metrics less attractive to WMNs routing layer. The authors of this work did not provide any results from the experiments to back their theoretical analysis of the routing metrics.

Table 0.1: Existing routing metrics reviews

Parameters	(Yang et al., 2006)	(Liu et al., 2008)
Routing protocol	Not clear	X
Performance metric	?	X
Network configuration	?	X
Experimental setup	X	X

Our work adopted the approach that is used by (Liu et al, 2008) of firstly grouping the routing metrics. This work evaluates the performance of routing metrics for WMNs. Routing metrics in (Liu et al, 2008) were classified into three groups: ETX-based, mETX-based, and metrics less attractive to WMN's routing layer, and in our study we also grouped the routing metrics into fourgroups. The study that is conducted in our own work used) simulation (NS2) to run experiments. The biggest network size that was simulated is 196 nodes, which can be easily done in NS2 simulation than in a test bed.

The use of test bed requires 196 physical nodes, which is costly than using simulation. The test bed in our wireless mesh network research laboratory (in University of Zululand) has only 14 nodes, which makes it difficult to use a test bed for this work. Our work used simulation for evaluation because of financial constrains, although for future work, running experiments on a test bed is considered.

Table 2.1 summarizes two works by other scholars that were reviewed in this research study. The tick (√) shows that the particular parameter was considered by that particular work while

the cross (x) means that the work did not provide that parameter. This table shows the inconsistency in comparing the existing routing metrics done by other scholars. It was not clear which routing protocol was used in the first work while the second work never provided experimental results; hence none of the parameters was found, therefore both fields were marked with a cross. The first work used same performance metrics for all the routing metrics that were compared which is also the same as what was done in our research study to keep the consistency. Same network configuration was used for all the experiments, but the experimental setup only considered two network sizes.

2.3 Summary

This chapter introduced key concepts such as routing protocol, routing metric, and routing, which are used more often in this work. The chapter reviewed two works that compared routing metrics, so as to justify the inconsistency in previous comparisons of routing metrics. Both these works were found not to have compared routing metrics in a consistent manner, leading to us having a reason to carry out this research project so as to come up with a consistent comparison of routing metrics. The next chapter provides a thorough review of each existing routing metric and later select four routing metrics to simulate and evaluate their performances.

CHAPTER THREE

LITERATURE REVIEW

3.1 Introduction

This chapter conducts a literature survey of available routing metrics for wireless multi-hop networks. A framework was used to analyze the routing metrics in section 3.2 of this chapter. Using literature review framework for the analysis of routing metrics helps us come up with the grouping of the routing metrics. Only one routing metric from a group was evaluated in Chapter Five so that each routing metrics' performance can be compared to the performances of other routing metrics.

From the four groups of routing metrics, four routing metrics were selected, one routing metric from each group. Routing metrics that consider QoS were given first priority. An ideal routing metric for WMNs need to consider the quality of the link before selecting a route through which to send packets, this was made the main selection criterion to choose representative samples. The first group only has one routing metric, leading to automatic selection of hop count routing metric. RTT was selected over the other routing metrics in its group because it considers many aspects of link quality when selecting the best path through which to send a packet. RTT considers loss ratio of the path before selecting the path to use. This routing metric uses probe packets to measure the load of the link, while it also takes interference experienced by the link into consideration. Based on the fact that the ETX is one of the most compared routing metric, the researcher felt that it would also be important for them to compare it to other routng metrics. ETX reduces broadcast and probing overhead. ETX also does not experience interference. One

of the major focuses of ETX is to account for packet loss ratio. EETT is selected to represent the last group. EETT firstly checks the busy degree of the link channel before making any route selection. Up to twenty routing metrics were discussed in this chapter, out of which four routing metrics were selected for evaluation.

3.2 Existing Routing Metrics

3.2.1 Custom Specification for Routing Metrics

We crafted a custom specification for each routing metric using ten parameters (table 3.1). For instance, one paremeter is name; while year as a parameter indicates time origin of the metric. There are eight more parameters, as they can be seen listed and defined in table 3.1. The table serves as the framework used to review and analyze existing routing metrics done in this chapter. This framework is firstly proposed in this work. The same parameters are considered for all the routing metrics being reviewed, but the value of the parameter differs depending on the routing metric in consideration.

3.3 Our Specification of Routing Metrics

As a result of the literature survey of routing metrics for wireless multi-hop networks conducted we have used the custom specification described in the previous section to present each routing metric.

Table 0.1: Parameters of routing metrics

Parameters	**Defination**
Routing metric name	This parmenter provides the name of the particular routing metric that is being reviewed.
Year	Year gives the year the routing metric under review was proposed by a particular scholar. The year helps during the literature survey to identify some trends in the routing metrics.
Problem addressed	The problem that a particular metric is trying to solve, forinstance, how the hop count is concerned with the path length from the sender to the receiver. It can be used to selct routing metrics to compare and simulate. It ould help us to group the routing metrics and pick a certain number of routing metrics in each group.
Solution approach	The way in which the particular routing metric is trying to address a problem.
Advantages	Refer to the strength of the existing routing metric and how it deals with the problems of routing in WMNs.
Disadvantages	Refers to the shortcoming of a particular routing metric and with a view to improve on them.
QoS-awareness	QoS is the collective effect of service performance which determises the degree of satisfaction of the user of the sevice. A routing metric is QoS-aware if it considers QoS parameters.

Performance metrics	Performance metrics such as delay, packet loss, etc considered by a particular routing metric. This parameter helps to identify parameters the routing metric is using.
Validation method	The process of testing routing metrics; it may be done in many ways such as simulation, mathematical proofs, implementation, etc. the validation method that was used to evaluate the performance of the routing metrics is identified.
Routing protocol(s)	A routing protocol uses a routing metric to select an optimal path. This parameter identifies a routing protocol that was used with a particular routing metric.

Table 0.2: Classification of routing metrics (Nxumalo, Ntlatlapa, Mudali, & Adigun , 2009)

Shortest Path	**Packet Loss Rate**	**Delay**	**Interfenrence**
HOP	ETX	PktPair	WCETT
	mETX	RTT	MIC
	ENT	ETT	iAWARE
	iETT	MCR	EETT
	MM	PPTT	ILA
	Routing metric for BATMAN	AETD	
		Airtime link metric	
		WCETT-LB	

The first step was that all the twenty routing metrics considered were partitioned into four different groups depending on what the particular metric is based upon. The groups are based on delay, packet loss rate, shortest path and interference. Table 3.2 groups the routing metrics based on the performance metrics that they use. The grouping concentrated on the performance metrics that the routing metrics gives high priority, so as to make sure that a routing metric falls only inside one group. Hop count is the oldest routing metric and only considers the shortest path from the sender to the receiver. The table shows that many routing metrics consider delay when selecting the optimal path to the receiver. Other routing metrics are more concerned with interference while other metrics are concerned with packet loss rate. The detailed study of all the routing metrics in Table 3.2 is documented in the next chapter.

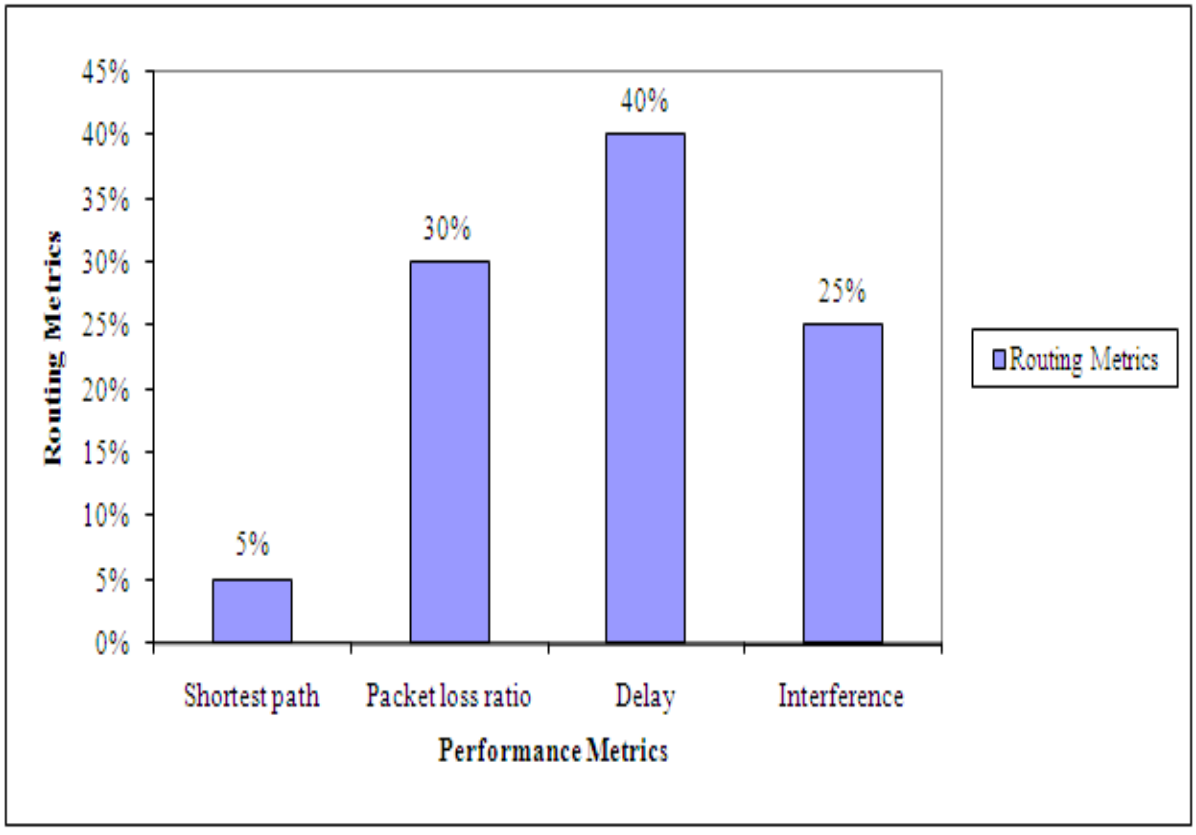

Figure 0.1: Performance metrics and percentages of routing metrics

Table 0.3: Parameters of HOP (Dijkstra, 1959)

Routing metric characteristics	**Value**
Routing metric name	Hop Count (HOP)
Year	1959
Problem addressed	To choose the shortest path
Approach	Selects the path with the least number of hops from the available paths to the detination.
Performance metrics considered	Number of hops
QoS-awareness	No
Advantages	• Easy to measure • Minimal overhead
Disadvantages	Does not consider: • Transmission rate of the link • Reliability/path loss • Load on the link • Interference
Routing protocol(s)	DSR, AODV, DSDV, GSR,and LQSR
Validation tool	Simulation in NS2

Routing has been an active area of research until now; even then different scholars are still trying to find the best routing metric for WMNs. The content of Table 3.2 can also be displayed in a graph as shown in Figure 3.1. Figure 3.1 depicts the percentage of routing metrics considering the particular performance metrics. The graph shows that the high percentage of routing metrics

reviewed were the routing metrics that gives delay higher quality than other performance metrics. Only five percent of the routing metrics (one routing metric) concentrated on shortest path. Thirty percent of the routing metrics put more priority on packet loss ratio, while the remaining twenty five percent gives high priority to interference.

a) The Hop Count (HOP) (Dijkstra, 1959)

Table 3.3 presents information about the hop count routing metric. Hop count routing metric chooses a route with least number of hops among the available routes to the intended destination. If it happens that there is more than one route that has the shortest path, HOP chooses randomly among those routes. It does not consider differences in throughput among those routes, in a process overlooking the fact that a longer route might have higher throughput than a shorter route. Link quality for hop count is a binary concept; it is either the link exists or it does not exist at all. The main advantage of this metric is that it is simple to use. Once the topology of the network is known, it is easy to compute and minimize the hop count between a pair of source and a destination node.

Computing the hop count does not require any additional measurements. The main disadvantage of hop count is that it neither takes packet loss nor bandwidth into consideration. A route that minimizes the hop count does not necessarily help in maximizing the throughput of a flow (De Couto, D. et al., 2003). For example, a path that has four hops and a reliable or fast links can have better performance than a path with two hops and a slow link. But still, the hop count metric, will prefer the two-hop path.

b) Per-hop Packet Pair Delay (PktPair) (Keshav, 1991)

PktPair measures delay between a pair of back-to-back probes to a neighboring node. The analysis of PktPair routing metric is tabulated in Table 3.4.

Table 0.4: Parameters of PktPair (Keshav, 1991)

Routing metric characteristics	**Value**
Routing metric name	Per-Hop Packet Pair Delay (PktPair)
Year	1991
Problem addressed	Try not to choose paths with high delay
Solution approach	Selects the path based on delay, it selects a path with lowest delay.
Performance metrics considered	Delay
QoS-awareness	Yes
Advantages	Measures: • Path loss • Interference in the vicinity • Transmit rate It is not affected by queuing delays at the sending node.
Disadvantages	• Self interference still exists • High overhead
Routing protocol(s)	LQSR
Validation tool	Simulation in NS2

When calculating PktPair, a node sends two packets known as probe packets back-to-back to each neighbor every 2 seconds. The second probe packet is bigger than the first probe packet. The neighbor node calculates the delay between the two packets that were sent back-to-back. It then reports this delay back to the sending node. The sender maintains an exponentially weighted moving average of these delays for each of its neighbors. The main aim of this routing metric is to minimize the sum of these delays. PktPair routing metric also measures many aspects of link quality. The main advantage of the PktPair metric is that this routing metric is not affected by queuing delays at the source node, because the first and the second packets in a pair will be delayed equally. Using probe packets of different sizes makes the routing metric more sensitive to link bandwidth.

PktPair routing metric has many disadvantages though. One of the disadvantages is that it experiences high overheads, because two packets are sent to every neighbor node, and the second packet is larger than the first one. The secondly disadvantage of PktPair is that this routing metric it was discovered that it is that it is not entirely immune to the phenomenon of self interference (Draves et al., 2004a). Table 3.4 depicts information about PktPair routing metric.

c) Expected Transmission Count (ETX) (De Couto, 2003)

Information about ETX is provided in Table 3.5. ETX estimates the number of transmissions (including retransmissions) needed to send unicast packets by measuring the loss rate of broadcast packets between pairs of neighboring nodes. The derivation of ETX starts by measuring the underlying packet loss probability in both the forward and reverse paths.

Table 0.5: Parameters of ETX (De Couto, 2003)

Routing metric characteristics	**Value**
Routing metric name	Expected Transmission Count (ETX)
Year	2003
Problem addressed	EXT minimizes the expected total number of packet transmissions (including retransmissions) required to successfully deliver a packet to the ultimate destination.
Solution approach	Selects path with the lowest number of retransmission.
Performance metrics considered	Packet loss rate, throughput
QoS-awareness	Yes
Advantages	• Broadcast, probing overhead is reduced • No self interference since delay is not measured • Accounts for packet loss rate
Disadvantages	Broadcast probe packet may not experience same loss rate data packets. Does not take into account: Trannsit rate, link load, and interference
Routing protocol(s)	DSDV, DSR, LQSR
Validation tool	Testbed implementaion

To calculate ETX, every node sends a probe packet every second which contains the number of probes received by every neighboring node in the last 10 seconds. A node can calculate loss rate of probe packets on links to and from its neighboring nodes based on the probe packets. These

counts will allow the sending node to estimate the number of times the 802.11 ARQ mechanisms must retransmit a unicast packet, because the 802.11 MAC does not retransmit broadcast packets. ETX is calculated as shown below:

$$ETX = \frac{1}{d_f \times d_r} \tag{3.1}$$

Forward delivery ratio (d_f) is the probability that a data packet was successfully received by the destination node, while reverse delivery ratio is the probability that the acknowledge packet successfully arrives at its original sender. The path metric describes the total of all the ETX values for each link in the path. The routing protocol chooses the path that has minimum ETX among the paths available. ETX improves the throughput, while its drawback is that it fails under variable link conditions. $d_f \times d_r$ is the probability that transmission was successfully received and acknowledged.

d) Per- hop Round Trip Time (RTT) (Adya, Bahl, Padhye, Wolma & Zhou 2004)

RTT routing metric measures the round trip delay seen by unicast probes between neighbor nodes. To calculate RTT, a node sends a probe packet with a time stamp to all its neighbors every 500 milliseconds. Every neighbor node will immediately reply with an acknowledgment probe with a time stamp in it. The time stamp allows the sending node to measure round trip time to every neighbor. The RTT routing metric considers many aspects of link quality (see Table 3.6). Firstly, if either the node or the neighbor node is busy, the probe or the probe acknowledgment packet will experience queuing delay. This delay results in high RTT. Secondly, if other nodes in the neighborhood are busy, the probe or the probe acknowledgment packet will experience delays because of channel contention. That will also result in high RTT.

Table 0.6: Parameters of RTT (Adya, et al., 2004)

Routing metric characteristics	**Value**
Routing metric name	Per-hop Round Trip Time (RTT)
Year	2004
Problem addressed	Tries to ignore paths with high delay.
Solution approach	Selects the path with least delay.
Performance metrics considered	Delay
QoS-awareness	Yes
Advantages	Measures: • Load • Interference • Path loss
Disadvantages	• Can lead to route instability due to queuing delay and self interference • Does not take into account the transmit rate • High overhead • Does not serve as the right measure for the unidirectional transmission
Routing protocol(s)	Multi-radio Unifacation Protocol (MUP) and LQSR
Validation tool	Simulation in NS simulator

Thirdly, if a packet between the nodes is lost, the 802.11 ARQ mechanisms may have to retransmit the probe or the probe acknowledgment packet a number of times to make sure that it is delivered successfully. This leads to increase in RTT along that link. Fourth, if regardless of the ARQ mechanism, a probe or a probe acknowledgment packet is lost, the source node detects the loss, and increases the moving average as described earlier on.

Finally, RTT is affected by self-interference traffic. The RTT routing metric was designed to avoid highly loaded or lossy links. RTT can lead to route instability because it is a load-dependent metric. This metric has its own disadvantages. One disadvantage is that it suffers from the overhead of measuring the round trip time. The other disadvantage is that this measurement method requires that every pair of neighboring nodes send probe packets to each other.

e) Expected Transmission Time (ETT) (Draves et al., 2004a)

Draves et al. (2004a) observed that ETX does not perform best under certain circumstances. One example would be that ETX prefers highly congested links to uncongested links, if the link-layer loss rate of congested links is less than the one for the unloaded links. The authors address this by proposing a new outing metric, which is the expected transmission time (ETT) routing metric. ETX incorporates the throughput into its calculation (Parissodis, Karaliopoulos, Baumann, Spyropoulos & Platter, 2009).

Table 0.7: Parameters of ETT (Draves et al., 2004a)

Routing metric characteristics	**Value**
Routing metric name	Expected Transmission Time (ETT)
Year	2004
Problem addressed	To improve ETX by considering difference in transmission rates.
Solution approach	It estimates the transmission time.
Performance metrics considered	Delay
QoS-awareness	No
Advantages	Reduces interference.
Disadvantages	High overhead
Routing protocol(s)	MR-LQSR
Validation tool	Testbed implementation

To improve ETX, ETT was proposed which considers the differences in link transmission rates. The ETT of a link *l* is defined as the expected MAC layer duration for a successful transmission of a packet at link *l* (Yang et al., 2006). The weight of a path *p* is simply the summation of the ETT*1*s of the links on the path.

ETT is calculated as:

$$ETT_l = ETX_l \times \frac{s}{b_l} \tag{3.2}$$

Where b_l is the transmission rate of link *l* while *s* is the size of the packet. By bring in b_l into the weight of a path, the ETT routing metric captures the impact of link capacity on the performance of the path. Like ETX, ETT is also known to be isotonic. However, one of the disadvantages of ETT is that it still does not fully capture the intra-flow and inter-flow interference in the network.

f) Weighted Cumulative ETT (WCETT) (Draves, Padhye & Zill, 2004b)

To improve upon ETT and ETX, Weighted Cumulative Expected Transmission Time was proposed (Draves et al, 2004b). To minimize intra-flow interference, WCETT was proposed to minimize the number of nodes on the route of a flow that transmit on the same channel. For a path *p*, WCETT is defined as:

$$WCETT = (1-\beta)\times\sum_{i=1}^{n} ETT_i + \beta \times \max_{i\le j<k} x_j \qquad (3.3)$$

where β is a tunable parameter subject to $0 \le \beta \le 1$. x_j is the number of times channel *j* is used along route *p* and captures the intra-flow interference. The j element in the equation 3.3 represents the maximum number of times that the same channel appears along a route. The j element captures the intra-flow interference of a route since it essentially gives smaller weights to routes that have more diversified channel assignments on their links and hence lesser intra-flow interference.

WCETT routing metric has two disadvantages. The first being, that it does not explicitly take into consideration the effects of inter-flow interference, although it does capture intra-flow interference. Therefore, WCETT may send packets through highly congested routes.

Table 0.8: Parameters of WCETT (Draves et al., 2004b)

Routing metric characteristics	**Value**
Routing metric name	Weighted Cumulative Expected Transmission Time (WCETT)
Year	2004
Problem addressed	To reduce intra-flow interference.
Solution approach	Reduce the number of nodes on the path of a flow that transmit on the same channel.
Performance metrics considered	Delay
QoS-awareness	Yes
Advantages	Reduces intra-flow interference.
Disadvantages	High overhead
Routing protocol(s)	MR-LQSR
Validation tool	Testbed implementation

Different scholars have tried to improve on this routing metric. A variant routing metric was proposed. This variant is called WCETT-LB (Weighted Cumulative Expected Transmission Time).

g) Metric of Interference and Channel Switching (MIC) (Yang, Wang & Kravets, 2005)

MIC routing metric was proposed to improve on WCETT by addressing the problem of inter-flow interference. MIC includes inter-flow interference by increasing the ETT of a link by the number of neighbor nodes interfering with the transmission on that link.

Table 0.9: Parameters of MIC (Yang et al., 2005)

Routing metric characteristics	**Value**
Routing metric name	Metric of Interference and Channel Switching (MIC)
Year	2005
Problem addressed	To reduce inter-flow interference
Solution approach	Selects the path based on inter-flow interference.
Performance metrics considered	Interference
QoS-awareness	Yes
Advantages	Reduces inter-flow interference.
Disadvantages	Does not capture the effects of variation in link loss-ratio.
Routing protocol(s)	Load and Interference Balanced Routing Algorithm (LIBRA)
Validation tool	Simulation

MIC uses a routing protocol called Load and Interference Balanced Routing Algorithm (LIBRA) (Yang et al., 2005). LIBRA can create a virtual network from a real network and decompose MIC into isotonic link weight assignments on this virtual network (Yang et al., 2005). By using Bellman–Ford or Dijktra's algorithm on this virtual network. LIBRA can calculate minimum MIC weight path of the real network so that flows can be routed through minimum weight paths and no forwarding loops are created for either distance vector or link-state routing. MIC is calculated as follows:

$$MIC_{(q)} = \frac{1}{N \times \min(ETT)} \sum_{link \ \ l \in p} IRU_l + \sum_{node \ \ i \in p} CSC_i \tag{3.4}$$

$$IRU_l = ETT_l \times N_l \tag{3.5}$$

$$\begin{cases} w_1 & if\ CH(prev(i)) \neq CH(i) \\ w_2 & if\ CH(prev(i)) = CH(i) \end{cases} \tag{3.6}$$

$$0 \leq w_1 < w_2 \tag{3.7}$$

h) Multi-Channel Routing Metric (MCR) (Kyasanur & Vaidya, 2005)

The MRC routing metric is based on WCETT routing metric proposed by (Draves et al., 2004b). MCR extends WCETT in a different way compares to the way MIC does; MCR takes into account the cost of changing channels. MCR is calculated as:

$$MCR = (1-\beta) \times \sum_{i=1}^{n} ETT_i + \beta \times \max_{1 \leq j \leq c} X_j \tag{3.8}$$

where β is a tunable parameter subject to $0 \leq \beta \leq 1$, n is the number of hops on the route, x_j is the total ETT cost on any channel *j*, and the total number of existing channels is *c*. Another component, Switching Cost *j* (SC (*j*)) is defined as follows:

$$p_s(j) = \sum_{\forall i \neq j} InterferenceUsage(i) \tag{3.9}$$

$$SwitchingCost(j) = p_s(j) \times SwitchingDelay \tag{3.10}$$

Where $InterferenceUsage(i)$ is an exponentially weighted average for any channel *j* to calculate what fraction of a second time interval a switchable interface was transmitting on channel *j*.

Table 0.10: Parameters of MCR (Kyasanur and Vaidya, 2005)

Routing metric characteristics	**Value**
Routing metric name	Multi-Channel Roting (MCR)
Year	2005
Problem addressed	Selects channel diverse routes while accounting for interface switching cost.
Solution approach	Manages multiple channels and interfaces.
Performance metrics considered	Delay
QoS-awareness	Yes
Advantages	Introduces channel-switching cost into the routing metric, thereby it can incorporate with the routing protocols like DSR and AODV for multi-channel and channel-switchable wireless network.
Disadvantages	Fails to figure in the inter-flow interference although is assumed all available channels are orthogonal.
Routing protocol(s)	HMCP
Validation tool	Simulation in Qualnet version 3.6.

This value does not figure in the time interval that this interface is tuned to channel *j*, but is idle. The component $p_{s(j)}$ is the probability that the switchable interface will be on a different channel (i, j) when a packet arrives on channel *j*. *SwitchingDelay* is the interface switching latency, which can be computed offline. Hybrid Multi-Channel Protocol (HMCP) (Kyasanur & Vaidya, 2006) is the multi-channel link layer protocol matching with a corresponding multi-channel

routing protocol, including this particular path metric MIC. When a source transmits a packet to a destination, it firstly tunes its switchable interface to the same channel as the destination's fixed channel. So, the probability of using channel *j* over a link only needs to compute that of the source side.

i) Modified ETX (mETX) (Koksal & Balakrishnan, 2006)

Resent research has shown that quality of service-aware (QoS-aware) routing metrics (i.e. ETX), has a potential to improve throughput of WMNs by significant amounts compared to traditional hop count routing metric (Koksal & Balakrishnan, 2006). Modified ETX was developed so as to improve on the weaknesses of ETX which does not cope well with short-term channel variations because it uses the mean loss ratios in coming up with routing decisions. The mETX routing metric captures the time-varying features of a wireless channel in a form that could be directly translated into network and application layer quality constraints (Koksal & Balakrishnan, 2006). Modified ETX is made of two components, which are $\mu\sum$ and $\sigma_{\sum}^{2}$. $\mu\sum$ represents the average of the error probability, while $\sigma_{\sum}^{2}$ represents variability of the error probability. Modified ETX is calculated as:

$$mETX = \exp\left(\mu\sum + \frac{1}{2}\sigma_{\sum}^{2} \right) \tag{3.11}$$

Table 0.11: Parameters of mETX (Koksal & Balakrishnan, 2006)

Routing metric characteristics	**Value**
Routing metric name	Effective Number of Retransmission (mETX)
Year	2006
Problem addressed	To improve up on the shortcoming of ETX of not cope well with short-term channel variations
Solution approach	It uses estimated transmission count.
Performance metrics considered	Delay
QoS-awareness	Yes
Advantages	Works with variable links.
Disadvantages	Complex error estimation method
Routing protocol(s)	DSR, DSDV and AODV
Validation tool	Simulation

j) Effective Number of Retransmission (ENT) (Koksal & Balakrishnan, 2006)

Effective number of retransmission (ENT) was proposed to find paths that satisfy certain higher layer protocol requirement. The main challenge of ENT is to find a route with high throughput while making sure that the end-to-end packet loss rate seen by higher layers does not go beyond a specific threshold. ENT is related to mETX. ENT has the drawback of employing a complex channel condition estimation method.

Table 0.12: Parameters of ENT (Koksal & Balakrishnan, 2006)

Routing metric characteristics	**Value**
Routing metric name	Effecctivve Number of Retransmission (ENT)
Year	2006
Problem addressed	Optimizing aggregate throughtput, while bounding the packet loss rate visible to higher layer protocols such as TCP.
Solution approach	Captures the time-varying characteristics of a wireless channel in a form that could be directly translated into network and application layer quality constraints.
Performance metrics considered	Packet loss rate
QoS-awareness	Provides controlled QoS.
Advantages	It provides controlled quality of service.
Disadvantages	Employs a complex channel condition estimation method.
Routing protocol(s)	DSV, DSDV and AODV
Validation tool	Simulation

k) Interference-Aware Routing Metric (iAWARE) (Subramanian, Buddhikot & Miller, 2006)

Interference-aware routing metric was proposed to capture the effects of variation in link loss-ratio, differences in transmission rate as well as inter-flow and intra-flow interference. When there is no interference in the network, ETT captures the quality of the link very well as links with smaller expected transmission time produce better throughput.0

But when there are more interfering flows in the network, this is not the applicable. There is a need to factor in the varying interference experienced by a link into the routing metric to get routes with good quality. iAWARE is calculated as:

$$iAWARE = \frac{ETT_j}{IR_j} \tag{3.12}$$

When IR_j for the link j is one (meaning that there is no interference), $iAWARE_j$ is just ETT_j which captures the link loss ratio and packet transmission rate of the link j. ETT_j is weighted with IR_j to capture the interference experienced by the link from its neighbors. A link with less *ETT* and high *IR* will have less iAWARE value. The lower the iAWARE of a link the better the link is.

l) Path Predicted Transmission Time (PPTT) (Yin, Xiong, Zhang & Linet, 2006)

Path Predicted Transmission Time is the summation of delay estimation on each link along the routing path, which further consists of packet service time and queuing delay. To calculate PPTT, the source node will calculate PPTT for each route and choose a path with minimal PPTT for Real Time Communication (RTC). PPTT is computed by summing up Link Predicted Transmission Time (LPTT). LPTT is affected by Carrier Sense (*CS*) traffic and Hidden Terminal (*HT*) traffic. CS traffic and HT traffic have two parts which are neighboring traffic and self-traffic. Neighboring traffic can be measured using existing probing techniques.

Table 0.13: Parameters of iAWARE (Subramanian et. al, 2006)

Routing metric characteristics	**Value**
Routing metric name	Interference-Aware Routing Metric (i-AWARE)
Year	2006
Problem addressed	To address the problem of interference aware routing in multi-radio infrastructure mesh networks.
Solution approach	Captures the effects of variation in link loss-ratio, differences in transmission rate as well as inter-flow and intra-flow interference
Performance metrics considered	Packet loss rate, interference
QoS-awareness	Yes
Advantages	Reduces interference
Disadvantages	It does not address the problem of the QoS awareness
Routing protocol(s)	AODV-MR
Validation tool	Wireless testbed

PPTT can be calculated based on the neighboring traffic and self-traffic path rather than the traffic information of the entire network. Thus PPTT is scalable. Experiment results show that PPTT outperforms other non prediction-based routing metrics such as ETX and WCETT in terms of delay and goodput in wireless multi-hop networks (Yin et al., 2006).

Table 0.14: Parameters of PPTT (Yin et al., 2006)

Routing metric characteristics	**Value**
Routing metric name	Path Predicted Transmission Time (PPTT)
Year	2006
Problem addressed	To provide real time communication.
Solution approach	Selects the path based on transmission time.
Performance metrics considered	Delay
QoS-awareness	No
Advantages	Supports real time communication. Reduces self interference
Disadvantages	Complexity
Routing protocol(s)	Distributed Coordination Function (DCF) for MAC layer
Validation tool	Simulation in NS2 and testbed implementation.

m) Adjusted Expected Transmission Delay (AETD) (Zhou, Zhang & Qiao, 2006)

The main aim of AETD is to take into consideration both delay and delay jitter of candidate routes when choosing the best path. AETD was designed to choose paths on which hops operating on the same frequency channel are separated as far as possible. This way, interference and channel contention may be minimized along the chosen paths and the system throughput may be improved. When a series of packets is transmitted from the sender to the receiver, the achieved throughput is determined by the following characteristics of the selected route:

- ETD: the expected end-to-end transfer delay of a single packet;

EDJ: the lower bound of the expected delay jitters between consecutive packet transmissions.

An ideal route shall have both a small ETD as well as a small EDJ. ETD is affected by the following: (1) the hop count of the route; and (2) the bandwidth and link quality of each hop along the route that determine the per-hop transmission rate and transmission time. A shorter path (measured in hops) does not necessarily provide a smaller end-to-end transfer delay. There is a possibility that a smaller hop count implies a longer average hop distance and, there fore, lesser transmission rates and higher overall transfer delay. On the other hand, a route with a higher hop count but shorter average hop distance may instead provide a lesser end-to-end transfer delay.

EDJ is affected by the following: (1) the channel diversity of the path; and (2) the bandwidth and link quality of each hop along the route that determine the per-hop transmission rate and transmission time. A more channel-diverse route experiences less interference as packet transmissions on various channels do not interfere with one another. In the extreme case when the route is perfectly channel-diverse (when packet transmissions on any two hops along the route does not interfere with each other.

Table 0.15: Parameters of AETD (Zhou et al., 2006)

Routing metric characteristics	**Value**
Routing metric name	Adjusted Expected Transmission Delay (AETD)
Year	2006
Problem addressed	To minimize interference and channel contention along the selected route and improve the system throughput.
Solution approach	Multi-radio multi-channel routing
Performance metrics considered	Delay
QoS-awareness	No
Advantages	Minimizes interference and channel contention.
Disadvantages	Does not cater for inter-flow interference in the multi-flow scenarios.
Routing protocol(s)	Distributed Coordination Function (DCF) for MAC layer
Validation tool	Simulation using QualNet simulator.

It is either because they are far apart from each other or because they operate on different frequency channels), packet transmissions on each hop may continue successfully at the same time without encountering any channel conflict and the consequent conflict resolution process. Hence, a very short delay jitter between consecutive packet transmissions is expected under such scenario, which is the same as the maximum single-hop transmission time along the route.

$$ETD_r = \sum_{h_i \in H_r} ETT_{h_i} \tag{3.13}$$

Where $H_r = \{h_1, h_2, \ldots\ldots, h_k\}$ denotes the corresponding hop sequence along a router, and h_i represent the hop between nodes (i-1) and i. r denotes the expected packet transmission time over hop h_i.

$$EDJ_{r(i)} = \begin{cases} ETT_{hk} \quad if \quad i = k = 1, \\ ETT_{hi+1} + EDJ_{r(i+1)} \\ f \exists i+1 < j \le \min\{i+m+1, k\} \\ such \quad that \quad C_{hi+1} = C_{hj} \\ \max\{ETT_{hi+1}, EDJ_{r(i+1)}\} \quad else, \end{cases} \tag{3.14}$$

Where m is the interference distance (and is measured in hops) in the hop-distance-based interference model.

$$AETD = (1-\alpha) \times ETD + \alpha \times EDJ \tag{3.15}$$

n) Airtime Link Metric (IIIE 802.11 WG TGs, 2008)

Airtime link metric considers the transmitting rate, frame delivery ratio, channel access overhead and protocol overhead. It is defined as the default link metric that can be used by a routing protocol to choose an efficient radio-aware path. Airtime cost shows the amount of channel resources used by transmitting the frame over a selected link. The cost metric for each link is calculated as follows:

$$C_a = \left[o_{ca} + o_p + \frac{B_t}{r} \right] \frac{1}{1 - e_{pt}} \tag{3.16}$$

Table 0.16: Characteristics of Airtime cost constants (IIIE 802.11, 2006)

Parameters	Value (802.11a)	Value (802.11b)	Description
O_{ca}	75µs	335 µs	Channel access overhead
O_p	110 µs	364 µs	Protocol overhead
B_t	8224	8224	Number of bits in test frame

Where O_{ca}, O_p, B_t are constants listed in Table 3.16, r and e_{pt} are the bit rate in Mbps and the frame error rate for the test frame size B_t respectively. The rate r stands for the rate at which the mesh point would transmit the test frame under normal conditions and its estimation dependents on local implementation of rate adoption; (e_{pt}) is defined as the probability that a test frame is transmitted at current transmission bit rate (r) links. The main advantage of the airtime link metric is that it considers the quality of various links. By doing that, it is then easy for a routing protocol to choose the route with the best quality. If each node broadcasts the test frames and not unicast them, the overhead of the test frame will be decreased. Airtime link metric has its disadvantages as well.

The main disadvantage is that there is a high overhead of the frame delivery rate measurement especially in unicast test frame situation. If broadcast is used, firstly, even though this overhead is reduced by using a small test frame, there still exists high overhead.

Table 0.17: Parameters of Airtime Link Metric

Routing metric characteristics	**Value**
Routing metric name	Airtime Link Metric
Year	2006
Problem addressed	The airtime link metric addresses the problem of quality of different links.
Solution approach	It considers the quality of different links.
Advantages	It takes into account the quality of different links.
Disadvantages	There is a big overhead of the frame delivery rate measurement especially in unicast test frame solution.
QoS-Awareness	No
Performance Metrics	Packet loss rate
Routing protocol(s)	Simulation in OPNET.
Validation tool	AODV

Secondly, the test frame will be sent at the lowest possible data rate and there is a possibility that they may not experience the same loss rate as the data packets sent at higher rate. An AODV protocol that uses airtime link routing metric is called Radio-Metric Ad-hoc on-demand Distance Vector (RM-AODV).

o) Multi-Metric (MM) (Shen & Fang, 2006)

Multi-metric routing metric was proposed as an improvement of airtime link metric. This routing metric achieves improvement and also addresses load balancing (Shen & Fang, 2006).

Table 0.18: Parameters of MM (Shen & Fang, 2006)

Routing metric characteristics	**Value**
Routing metric name	Multi-metric
Year	2006
Problem addressed	It addresses three problems: • The residual bandwidth should be prior to transmit rate for transmiting capability consideration • It considers frame delivery ratio as it is sensitive to interference • New transmitting paths should go around the hot sport (bottle neck mesh point)
Solution approach	It determines the cost using residual bandwidth, load and frame delivery ratio
Advantages	• The performance of MM-AODV is good for heavy traffic load. • MM-AODV builds new paths around the used paths, so they improve the throughput significantly.
Disadvantages	There is still a need for a better residual bandwidth estimation to replace the “Listen” method used by Multi-metric.
QoS-Awareness	No
Performance metrics considered	Packet loss rate

Routing protocol(s)	Simulation in NS2
Validation tool	AODV

Multi-metric takes into consideration the residual bandwidth and frame delivery ratio (FDR) of the link into consideration when selecting an optimal path. The quality of the path is sensitive to the residual bandwidth, frame delivery ratio and the load of the mesh point (Shen & Fang, 2006). The use of FDR is simply because it is sensitive to interference. The use of FDR helps multi-metric on addressing issues of interference that neglected by airtime link metric. The path's cost is given by linear combination (see equation. 3.14) of minimum residual bandwidth, maximum load and FDR of the link.

$$c_a = \alpha Min_{Bw} - \beta Max_{Load} + \gamma FDR \tag{3.17}$$

Where, Min_{Bw} is the minimal residual bandwidth, and α is its weighted factor. Mac_{Load} is the maximum load of a node in route, and β is its weighted factor. γ is the weighted factor FDR. α , β and γ must satisfy the following constraint of $|\alpha|+|\beta|+|\gamma|=1$ according to (Shen & Fang, 2006). This path selection metric neglects the issues of load balancing. It is very important for a path selection metric to be able to identify potential bottleneck nodes and avoid paths that are made of such nodes. Multi-metric uses AODV as its routing protocol. The AODV routing protocol that uses multi-metric is called MM-AODV.

Table 0.19: Parameters of WCETT-LB (Ma & Denko, 2007)

Routing metric characteristics	**Value**
Routing metric name	Weighted Cumulative Expected Transmission Time with Load Balancing (WCETT-LB)
Year	2007
Problem addressed	To provide load balancing in mesh routers.
Solution approach	It uses estimated transmission count.
Performance metrics considered	Delay
QoS-awareness	No
Advantages	Avoids congestion.
Disadvantages	High overhead
Routing protocol(s)	AODV-Spanning Tree (AODV-ST)
Validation tool	Simulation in NS2.

p) Weighted Cumulative Expected Transmission Time with Load Balancing (WCETT-LB) (Ma & Denko, 2007)

WCETT-LB was proposed to improve on the WCETT routing metric, and it introduces load balancing features at the mesh points and supports global load-aware routing. Integration of a load-balancing scheme can provide improved performance in the entire network. The load-balancing component c is made up of two parts: congestion level and traffic concentration level at each node in a particular route. WCETT-LB is calculated as follows:

$$WCETT - LB(p) = WCETT(p) + L(p) \quad (3.18)$$

$$\text{Where } L(p) = \sum_{node \ \ i \in p} \frac{QL_i}{b_i} + \min(ETT) N_i \tag{3.19}$$

q) Exclusive Expected Transmission Time (EETT) (Jiang, Liu, Zhu & Zhang, 2007)

EETT is used to give a better evaluation of a multi-channel path. EETT of a link *l* is the busy degree of the channel used by that link (link *l*). It is a worst-case estimation of transmission time for passing link *l*. If there are more neighboring links on the same channel with link *l*, link *l* may be forced to wait a bit longer time to do the transmission on that channel. A path with a higher EETT indicates that, it has a more severe interference and requires more time to finish the transmission over all links within the path.

In essence, a better channel distribution over a path results in less intra-flow interference. Hence EETT can accurately reflect the optimality of the channel distribution on a path. The fundamental difference between MIC and EETT is that, MIC takes the impact of link *l* on other links into consideration, while EETT is concerned with the impact of other links on the current link (link *l*). Although the global impact is the same, EETT is supposed to have better performance since it more accurately reflects the impact of the inter-flow interference on a particular path.

Table 0.20: Parameters of EETT (Jiang et al., 2007)

Routing metric characteristics	**Value**
Routing metric name	Exclusive Expected Transmission Time (EETT)
Year	2007
Problem addressed	Selects multi-channel routes with the least interference to maximize the end-to-end throughput.
Solution approach	It uses EETT to check the busy degree of channel used by link 1. If there are more neighboring links on the same channel with link 1, link 1 may have to wait a longer period to do the transmission on that channel.
Performance metrics considered	Interference
QoS-awareness	Yes
Advantages	Gives a better evaluation of a multi-channel path.
Disadvantages	Still needs to addresses the problem of QoS-awareness and cross layer-awareness.
Routing protocol(s)	OLSR
Validation tool	Simulation

r) Interference-Load Aware Routing Metric (ILA) (Shila & Anjali, 2007)

Interference Load Aware (ILA) routing metric was proposed as an improvement from MIC metric. ILA considers the problem of interference-load aware routing in multi-channel wireless mesh networks.

ILA is made of two components: Metric of Traffic Interference (MTI) and Channel Switching Cost (CSC) (Shila & Anjali, 2007). The two components of ILA capture the effect of intra-flow and inter-flow interference, variation in transmission rate, packet loss ratio and congested areas. MTI considers the traffic load of interfering neighbors' contrary to number of interfering neighbors in MIC. The MTI metric is defined as follows:

$$MTI_i(c) = \begin{cases} ETT_{ij}(c) \times AIL_{ij}(c), N_l(c) \neq 0 \\ ETT_{ij}(c), N_l(c) = 0 \end{cases} \quad (3.120)$$

Where AIL_{ij} is the average load of the neighbors that might interfere with the transmission between nodes i and j over channel C . AIL_{ij} Average Interference Load, is calculated as

$$\boldsymbol{AIL}_{ij}(C) = \frac{\sum_{N_l} \boldsymbol{IL}_{ij}(C)}{\boldsymbol{N}_l(C)} \quad (3.21)$$

where

$$N_l(c) = N_i(c) \cup N_j(c) \quad (3.22)$$

Where $IL_{ij}(C)$, Interference Load is the load of the interfering neighbor node. $N_l(C)$ is the set of interfering neighbors of nodes i and node j . ETT_{ij} captures the variation in transmission rate and loss ratio of the links. AIL_{ij} defines the neighboring activity of the nodes. CSC is included to capture the intra-flow interference. The CSC is defined as:

$$CSC_i = \begin{cases} w_1, & CH(prev(i)) \neq CH(i) \\ w_2, & CH(prev(i)) = CH(i) \end{cases} \quad (3.23)$$

Table 0.21: Parameters of ILA (Shila and Anjali, 2007)

Routing metric characteristics	**Value**
Routing metric name	Interference-Load Aware Routing Metric (ILA)
Year	2007
Problem addressed	ILA considers the problem of interference-load aware routing in multi channel wireless mesh networks.
Solution approach	ILA combines the Metric of Traffic Interference.
Performance metrics considered	Interference (MTI) and Channel Switching Cost (CSC) to capture all the characteristic of the mesh network.
QoS-awareness	No
Advantages	• To capture all the characteristics of a mesh network, this metric combines MTI and CSC. • Reduces interference • Delivers high throughput in multi channel networks
Disadvantages	Still needs to consider the issue of QoS and cross layer-awareness.
Routing protocol(s)	AODV
Validation tool	Simulation in NS2

ILA is calculated as follows:

$$ILA(p) = \alpha \times \sum_{link \quad i \in p} MTI_i + \sum_{node \quad i \in p} CSC_i \tag{3.24}$$

s) Improved Expected Transmission Time (iETT) (Biaz & Qi, 2008)

The Improved Expected Transmission Time routing metric was proposed to address the shortcomings of the ETT routing metric. The iETT routing metric is designed to consider (1) the difference of link loss rates within one route and (2) the MAC layer overheads when computing an expected packet transmission time (rather than of simply using packet or bandwidth). By being able to capture the two parameters, iETT chooses a route with better performance. To calculate iETT, an extra medium time delay is added to the expected transmission time that is called Link Interference Delay (LID). LID depends on the difference between highest and lowest loss rates.

LID is expressed as follows:

$$\left(p_k\right)\times\left(a_j x+b_j\right) \tag{3.25}$$

Where $\max\left(p_j\right)$ denotes the highest loss rate and $\min\left(p_k\right)$ stands for the lowest rate on the path.

$$\left(p_k\right)\times\left(a_k x+b_k\right) \tag{3.26}$$

If $j>k, \alpha=1$; otherwise $\alpha=0$

When $j<k$, the position of link *j* with the highest loss rate is before that of link *k* with the lesser loss rate on the route.

Table 0.22: Parameters of iETT

Routing metric characteristics	**Value**
Routing metric name	Improved Expected Transmission Time (iETT)
Year	2008
Problem addressed	To find a path with a higher throughput and lower latency
Solution approach	It takes into account the discrepancy of links loss rates within one path, as well as the MAC layer overheads when computing an expected packet transmission time (instead of simply using packet/bandwidth).
Performance metrics considered	Packet loss rate
QoS-awareness	No
Advantages	• Does not fail when there is a high discrepancy in the quality of the links on a given path. • Capable of finding a path with a higher throughput and lower latency than other routing metrics.
Disadvantages	Still needs to be extended for multi radio scenario
Routing protocol(s)	DSR
Validation tool	Simulation in NS-2

Reversely, if $j>k$, it represents the position of link *j* is after that of link *k* on the route. The final iETT metric can be expressed as follows:

$$iETT = \sum_{i=1}^{n} \left(a_i x + b_j\right) \times \left(ETX_i\right) + LID \tag{3.27}$$

Table 0.23: Parameters of BATMAN (Johnson et al., 2008)

Routing metric characteristics	**Value**
Routing metric name	Routing metric for BATMAN
Year	2008
Problem addressed	It is aimed at maximizing packet delivery.
Solution approach	It considers the quality of different links.
Advantages	It is designed to be able to work in a large network.
Disadvantages	BATMAN has its own routing metric that it was proposed with. It may not be easy to use a different routing protocol with routing metric for BATMAN.
QoS-Awareness	No
Performance metrics considered	Packet loss rate
Validation Method	Wireless testbed
Routing Protocol(s)	BATMAN

t) Routing Metric for BATMAN (Johnson, Aichele & Ntlatlapa, 2008)

This routing metric is a link metric that is used by BATMAN routing protocol (Johnson et al., 2008). The purpose of this routing metric is to maximize the probability of successfully delivering a message. It does not check the quality of the link but rather its existence. The links are compared in terms of originator messages received within the current sliding window.

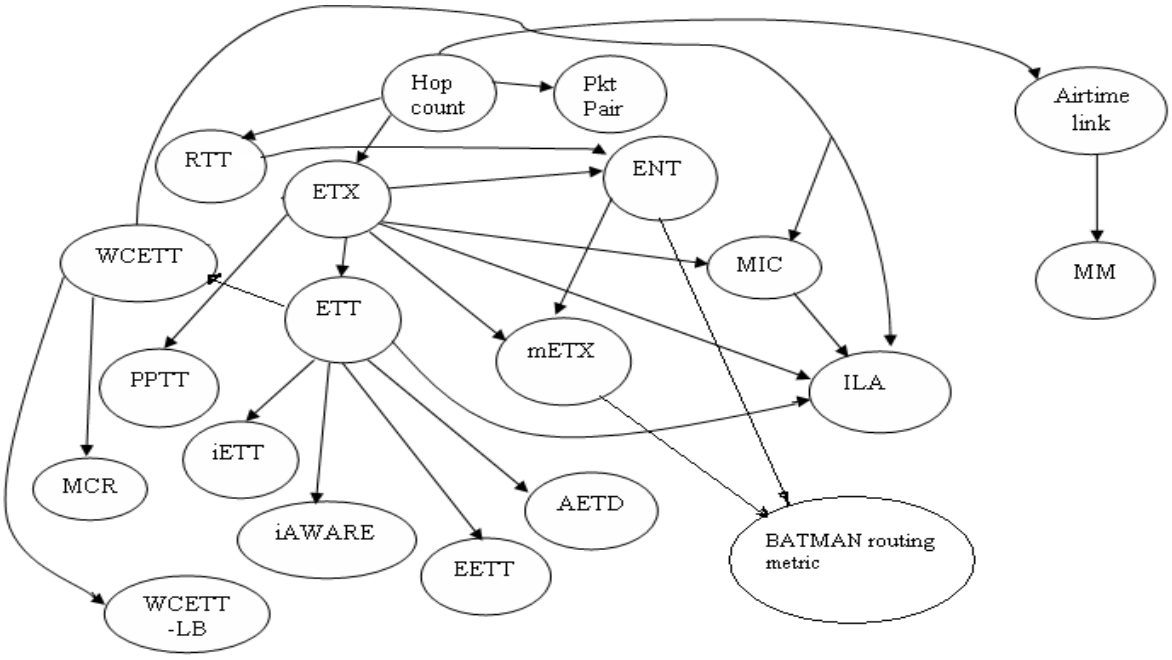

Figure 0.2: Evolution of routing metrics

The algorithm to measure this routing metric is as follows:

a. Consider routing message *m* from source *s* to destination *d* on network *g*. Eliminate all links $(s,i) \forall i \neq K$ to reduce the graph.

b. Associate each link with weight $w_{si,}$ where w_{si} is the number of originator messages received from the destination through neighbor node *i* within the current sliding window.

c. Find the link with largest weight w_{si} in the sub-graph and send m along the link (*s*, *i*).

d. If $i \neq d$, repeat step 1 to 4 for routing message from *i* to *d* in the sub-graph *S*.

After reviewing existing routing metrics, taxonomy of these routing metric was then developed (see Figure 3.2). The figure illustrates the relationship amongst the routing metrics. Hop count is the traditional routing metric; it can be seen from the Figure 3.2 that a lot of routing metrics were proposed so as to improve hop count. A lot of routing metrics were also proposed so as to improve ETX and ETT.

3.4 Summary

This chapter reviewed upto twenty existing routing metrics. The routing metrics were reviewed using a framework that we developed at the beginning of this chapter (see table 3.1). The routing metrics were grouped into four groups, depending on whether they prioritize delay, packet loss ratio, interference, or shortest path. In the next chapter, four routing metrics choosen for simulation after comparing different routing metrics in every group among themselves are discussed. The pseudo code and the flowcharts for these four routing metrics is presented in Chapter Four.

CHAPTER FOUR

SELECTED ROUTING METRICS FOR THE STUDY

4.1 Introduction

After reviewing the literature of the available routing metrics, four routing metrics were selected for evaluation. The routing metrics to be simulated are: hop count (HOP), expected transmission count (ETX), per-hop round trip time (RTT), and exclusive expected transmission time (EETT). There might be more than one routing metric that satisfy this requirement from a group, but for simulation purposes the researcher sticked to only one routing metric per group.

Hop count is the only routing metric from the first group. Although HOP does not take the quality of the link into consideration but it was still simulated for the purpose of making sure that all the groups are represented so that conclusion can be drawn based on all groups. It was important to include the legendary hop count in our evaluation since it has been heavily criticized in the literature, and the evaluation can help justify those claims as the routing metrics were evaluated in a consistant manner.

The other three routing metrics that were selected from the other three groups are ETX, RTT, and EETT. The main objective of ETX is to find paths with high end-to-end throughput (De Couto, 2003).

ETX also accounts for three issues:

i. The wide range of link loss ratios.

ii. The existence of links with asymmetric loss ratios.

iii. The interference between successive hops of multi-hop paths.

RTT also considers quality of the link because it was designed to avoid highly loaded and loosy links. RTT also measures interference. EETT considers other issues that the other routing metrics are not considering, such as channel distribution on long paths. Selected routing metrics for this study are further described in subsection 4.1 through subsection 4.4.

4.2 Hop Count

Figure 4.2 presents the flowchart for the hop count routing metric. This flowchart is derived from the algorithm that is used by the hop count to choose the best path for sending packets. The hop count selects the shortest path to the destination, using random selection if there is more than one route with the shortest path to the destination. In the pseudo code, the statement u: = node in Q with smallest dist [], looks for the vertex *u* in the vertex set *Q* that has the lowest dist[u] value.

That vertex is eliminated from the set Q and returned to the user. dist_between (u, v) calculates the length between the two neighbor-nodes *u* and *v*. *alt* is the length of the route from the root node to the neighbor node *v* if it were to go through *u*. If this route is shorter than the current shortest path recorded for *v*, then that current route is substitute with this *alt* route.

i. Pseudo Code

```
function Dijkstra (Graph, source):
    for each vertex v in Graph:                    // Initializations
    dist[v] := infinity                            // Unknown distance function from source to each
                                                   // node set to infinity
    previous[v]: =undefined                        // Previous node in optimal path from source
    dist[source]: =0                               // Distance from source to source
    Q: =the set of all nodes in Graph              // All nodes in graph are unoptimized, this are in Q
    While Q is not empty:                          // The main loop
        u: =vertex in Q with smallest dist []
        if dist[u]=infinity:
            break                                   // All remaining vertices are inaccessible
        remove u from Q
        for each neighbor u of v:                  // Where neighbour u has not yet been removed
                                                      from Q
            alt: =dist[u]+dist_between (u, v)
            if alt<dist[u]                          // A shorter path of u has been found
                dist[u]: = alt                     // Update distance of u
                previous[v]: = u
    return previous []
```

Figure 4.1:Pseudo code for hop count routing metric

ii. Flowchart

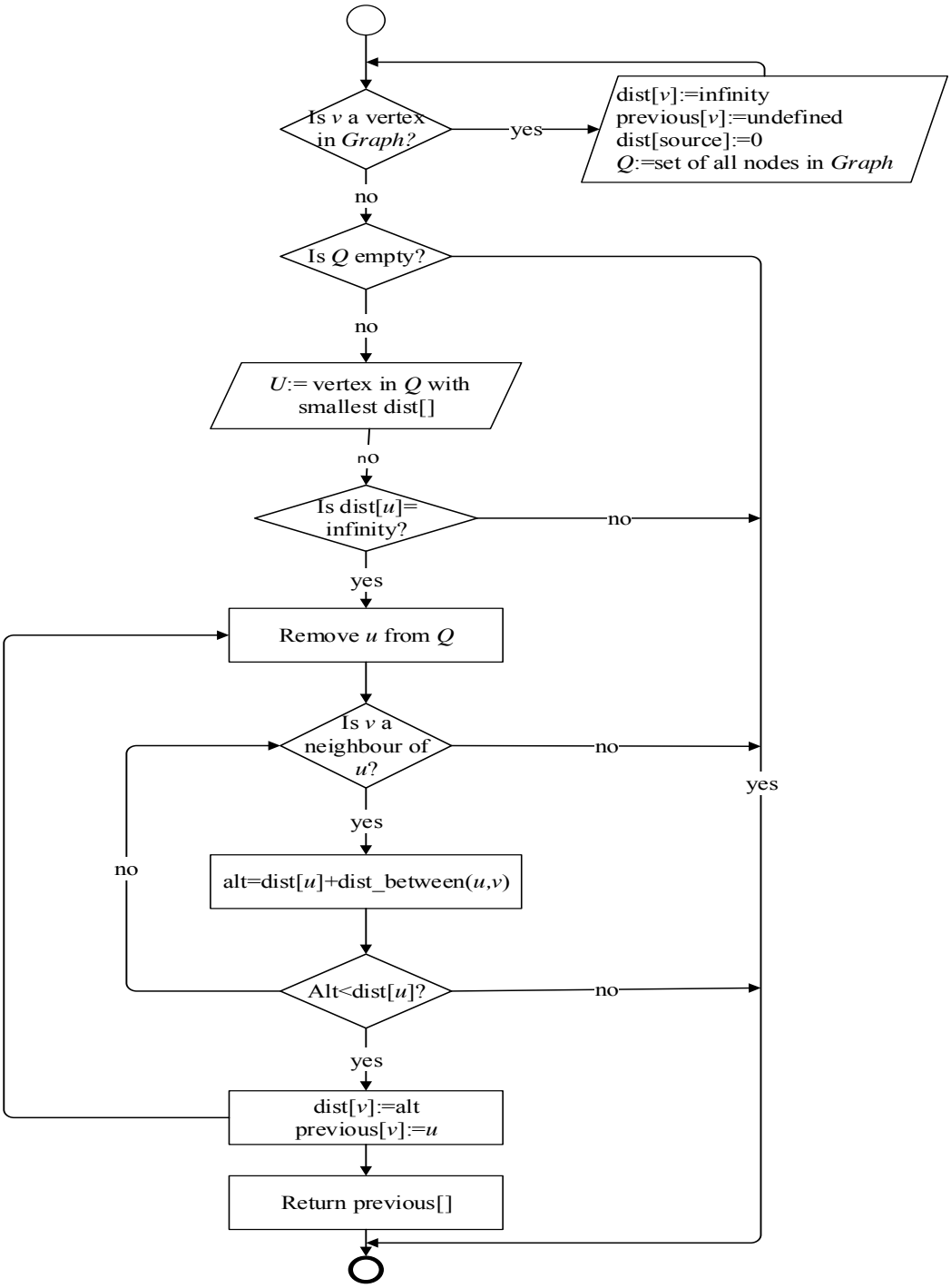

Figure 4.2: Flowchart for the hop count routing metric

4.3 Expected Transmission Count

i. Pseudo Code

```
Broadcast                          // Send packet to all nodes in the network
  Node
Initialize time, count, lossRate   // Initialize variables
  If time = 10 sec
    Send probePacket               // A node broadcaststs a bprobe packet every 10 seconds
    Count = count +1               // Increament count
    If count = 10
  If probePacket is received       // If a probe was successfully received, calculate the new
                                   // ETX
    Calculate lossRate
    Write lossRate
  Else                             // If a packet was lost, apply a 20% penalty
    losRate = lossRate – lossRate * 0.2 // updae the loss rate value
  Choose smallest ETX              // Select he least ETX value
```

Figure 4.3: Pseudo code for expected transmission count routing metric

Figure 4.4 depicts the pseudo code for the expected transmission count (ETX) routing metric. This pseudo code presents a how does the ETX goes about in calculating the best route through which to send packets, as it was discussed in earlier in this chapter. To measure ETX, every node broadcasts a probe message every second that contains the number of probes that were received by every neighbouring node in the last 10 seconds. If the probe was successfully received, the new ETX is calculated. If a data packet was lost, there is a 20% penalty for lost data packet.

ii. Flowchart

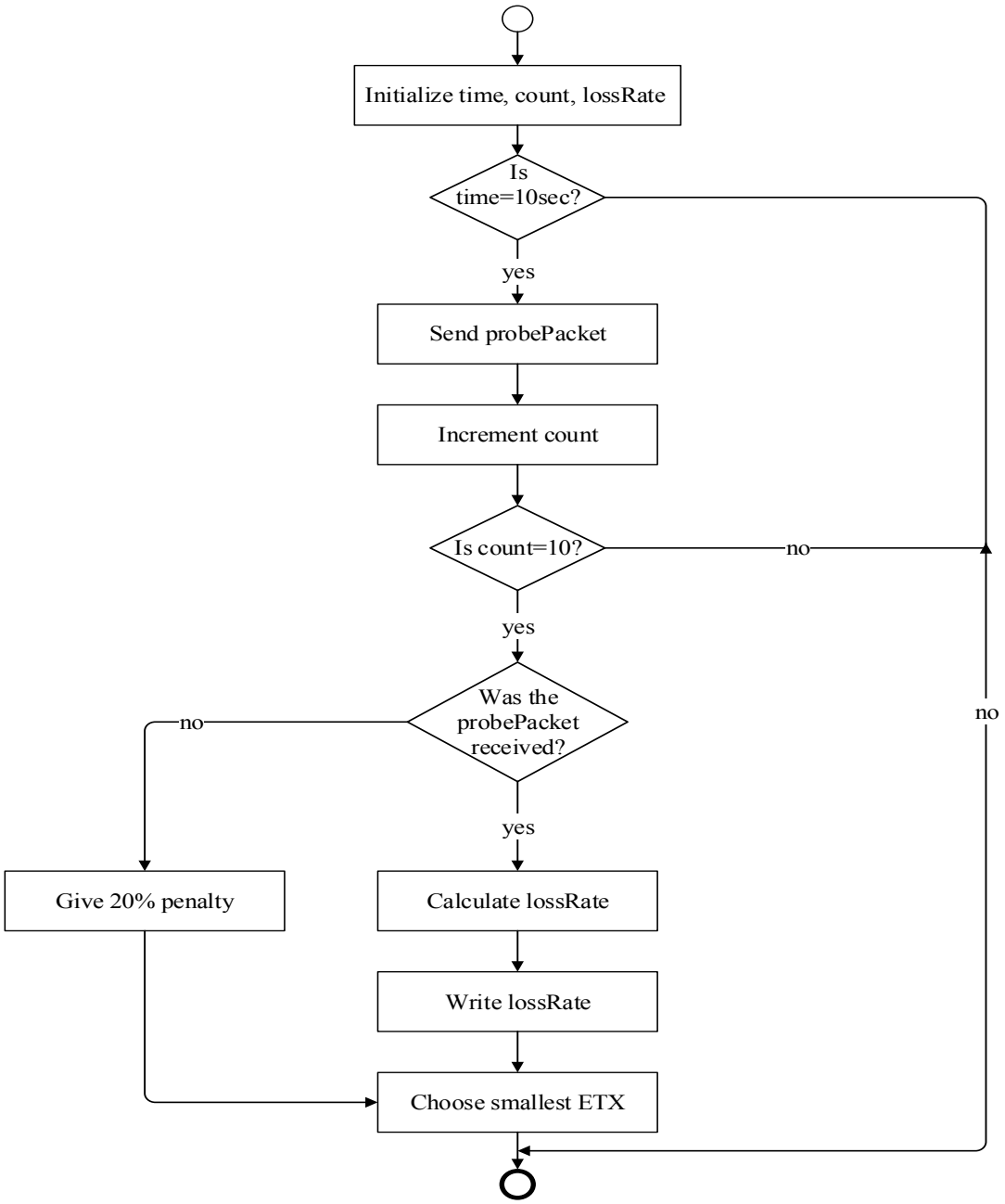

Figure 4.4: Flowchart for the expected transmission count routing metric

Figure 4.5 depicts the flowchart that follows from pseudo code in Figure 4.5. The flowchart shows the steps that are followed by the ETX routing metric when selecting a path to use for sending packets.

4.4 Per-hop Round Trip Time

i. Pseudo Code

```
READ
  Node, neighborNode, avg, time, weight // Read variables
  If time = 500 milliseconds            // Send probe packet every 500 milliseconds
   READ timestamp                       // Get the timestamp on the packet
    SendNeighborNodeState
  NeighborNode                          // Neighbor node responds with ack, echoing time stamp
    Write timestamp
  MeasureRoundTrip                      // Get round trip time
    Keep exponentially weighted average
  If responsePacket is found            // Give 10% to the current sample while calculating
average
    avg = 0.1 * weight                  // Calculate average RTT
  Else
    avg = weight + weigh * 0.2          // Give 20% penalty
  LeastTotalSum
    Get smallest average                // Select the smallest RTT
```

Figure 4.5: Pseudo code for Per-hop round trip time routing metric

Figure 4.6 shows the pseudo code that contains the steps that are followed by per-hop round trip time routing metric when selecting the path to be used for sending a packet.

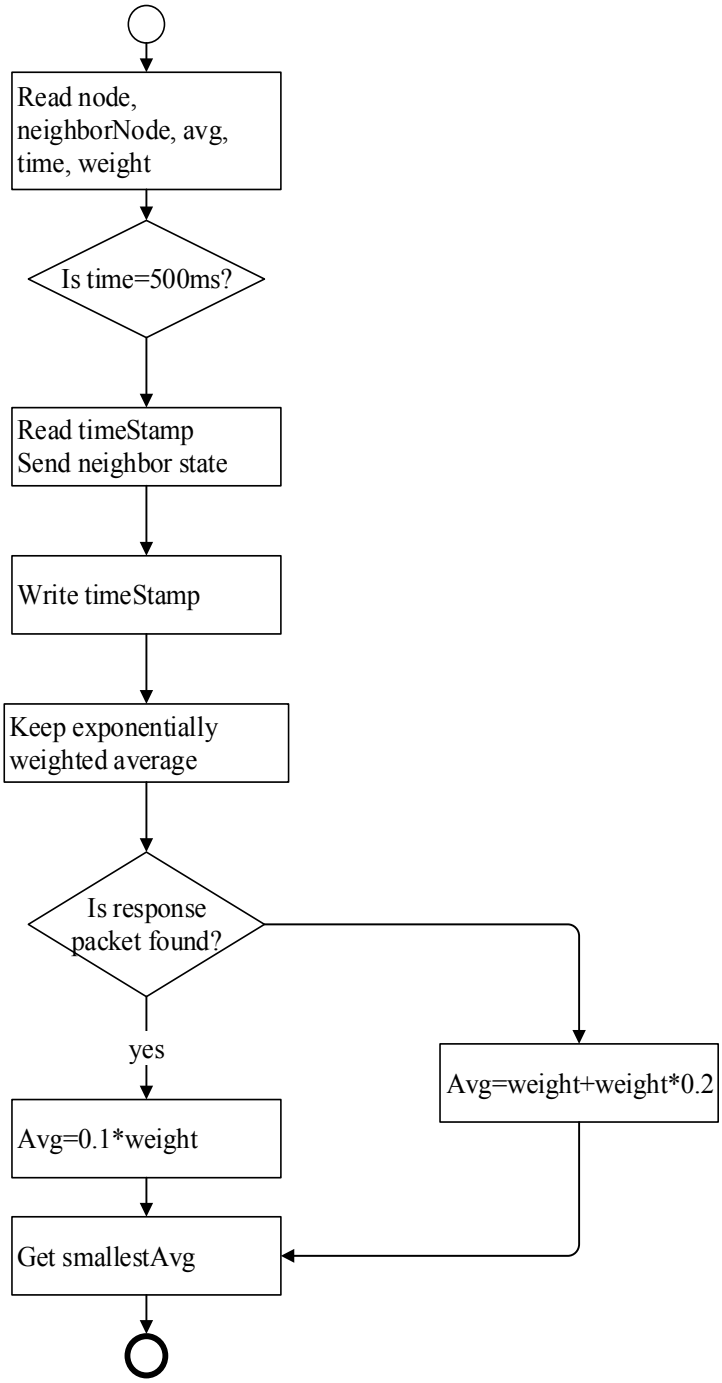

Figure 4.6: Flowchart for the Per-hop round trip time routing metric

The pseudo code shows that to calculate RTT, a node sends a probe packet every 500 milliseconds. The pseudo code in Figure 4.6 can also be represented using a flowchart in Figure 4.7 which depicts the steps followed when measuring the per-hop round trip time.

ii. Flowchart

A node keeps an exponentially weighted moving average of the RTT samples to every neighbor. The average is calculated, and 10% weight is given to the current sample. An average is increased by 20% if either a probe or a probe response packet is lost. The average is also increased by 20% if a packet is lost.

4.5 Exclusive Expected Transmission Time

i. Pseudo Code

```
Initialize numberOfLinks,                                   // Initialize variables
        numberOf Channels,
        numberOfHops,
        interferenceSet
        sumOfEETT
        while numberOfChannels > 0                          // check if the number of channels is
                                                            // greater than zero.
          sumOfEETT = sumOfEETT + interference Set // Add interference to the sum of
                                                            // EETT
          numberOfChannels – 1                              // Decreament the number of channels
        write sumOfEETT                                     // Update the sum of EETT
        choose smallest EETT                                // Select the smallest EETT
```

Figure 4.7: Pseudo code for exclusive expected transmission time

Figure 4.8 shows the pseudo code followed when implementing the exclusive expected transmission time routing metric. This pseudo code shows that the EETT routing metric chooses the route with the smallest EETT. It starts by firstly summing up the EETTs of the all the available paths to the intended destination, before choosing the path with the smallest EETT.

i. Flowchart

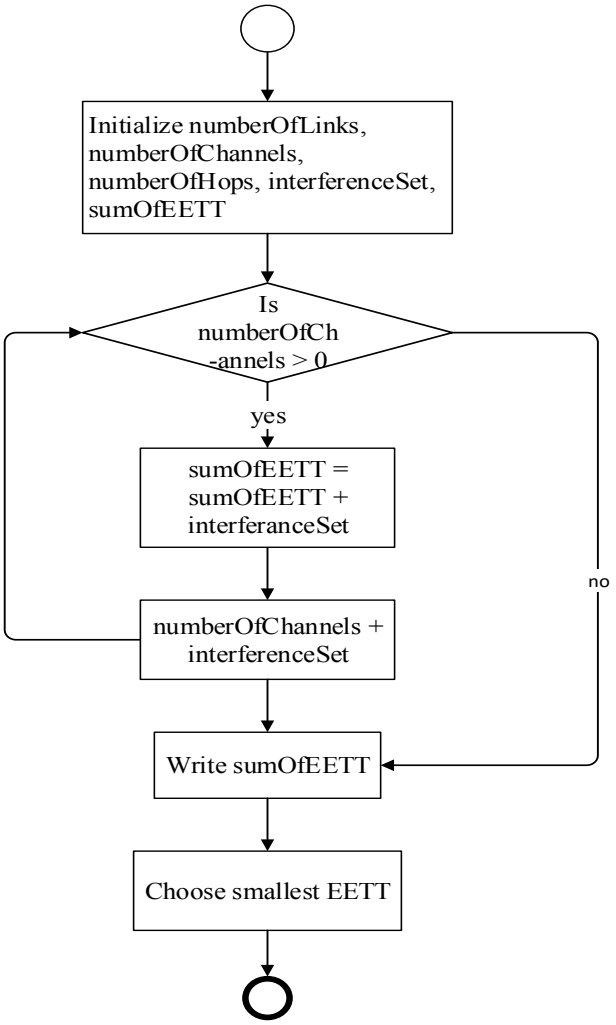

Figure 4.8: Flowchart for the exclusive expected transmission time routing metric

Figure 4.9 depicts the flowchart for the EETT routing metric which is the fourth routing metric that was selected for simulation. The flowcharts show clearly how does exclusive expected transmission time routing metrics goes about in selecting an optimal path for sending packets.

4.6. Summary

This chapter discussed four routing metrics that were selected after conducting a review of different existing routing metrics that was done in Chapter Three. We presented and discussed the pseudo codes and the flowcharts for these four routing metrics. These pseudo codes and flowcharts helps us when hard coding the routing metrics. Chapter Five presents and discusses the simulation results for the four chosen routing metrics. After analysis results, we recommend design criteria for best routing metric for wireless mesh networks.

CHAPTER FIVE

PERFORMANCE EVALUATION OF SELECTED ROUTING METRICS

5.1 Introduction

A number of routing metrics were developed with mobile ad hoc networks (MANETs) in mind. The recent focus on wireless mesh networks (WMNs) now requires routing metrics that are optimal for WMNs. There is a need to start by evaluating existing routing metrics that were designed for MANETs, so as to conclude if there is any that works best for WMNs. These routing metrics have been compared by other authors in literature before, but the manner in which the comparisons have been conducted is not consistent, hence the needs for this research study. A consistent comparison of different routing metrics for WMNs is achieved by keeping the simulation environment and parameters the same for all of our simulations. Simulation environment, evaluation parameters, and WMNs experimental setup are presented and described in this chapter. The core of this chapter is the simulation experiments of the four routing metrics selected for evaluation. WMNs differs form MANETs in a sense that while MANETs have high mobility, WMNs are stationary, and hence we kept our nodes stationary throughout our experiments.

The strategy followed in this investigation is as follows:

i. Twenty routing metrics are reviewed as described in Chapter Three.

ii. The review was based on the framework which formed the basis of grouping the twenty routing metrics into four categories (see Chapter Three).

iii. One and not more than one of the routing metrics in each of the four categories (see Chapter Four) was selected for simulation experiments as described in Chapter Three.

iv. The selected routing metrics are now ready in pseudo code and flowcharts form for simulation experiments in this chapter. AODV has been selected as the routing protocol. AODV has widely been used with a lot of different routing metrics (i.e. HOP, ETX). Hence we use AODV in this work since it is the most used routing protocol in the literature. It is better to compare routing metrics using a routing protocol that is preferred in the literate.

Section 5.2 presents a detailed explanation of the simulation environment used to run the experiments. In this section, the type of simulation tool and version of the simulation tool are briefly described. Evaluation parameters measured are discussed in section 5.3. The experimental setup that was used for simulation based on the square grid topology is discussed in section 5.4. Square grid topology allows us to have same positions for the nodes for all the experiments. Having nodes on the same positions and having the same distance between neighbor nodes helps have consistent comparison of the routing metrics, which this work is trying to achieve.

Experiments that measures simulation parameters discussed in 5.5 are provided in tabular form, and then presented graphically. Detailed discussion of results is presented in section 5.5. Trace files were generated using NS2 and then analyzed using Trace Graph. The traffic volume was increased as the number of the nodes in the network was increased. These graphs are then

interpreted to explain the behavior of each routing metric. Section 5.6 of this chapter recommends design criteria for an optimal routing metric for wireless mesh networks. In section 5.7 we present the conclusion of this chapter.

5.2 Simulation Environment

Four routing metrics are simulated using version 2.34 of the Network Simulator 2 (NS2) tool that was run on Linux 9.04 operating system. The four routing metrics were simulated and all the results were compared to come up with a conclusion at the end of this research project. NS2 is an open-source event-driven simulator tool that was designed particularly for research in computer communication networks. NS2 simulates both wired and wireless networks and is primarily Linux based.

NS2 contains modules for numerous network components such as routing, transport layer protocols, application, etc. NS2 uses two languages, namely: an object oriented simulator (written in C++), and an OTcl (an object oriented extension of Tcl language) interpreter, used to execute user's command scripts. Five performance metrics were used in our experiments. These five metrics are described in the next section (section 5.3).

5.3 Evaluation Parameters

5.3.1 Average delay

Average delay is the time taken to successfully transmit a packet from the sending node to the intended destination node. This time ends when the source node receives the acknowledgement

from the destination node to confirm that the packet was received successfully. The best path must try to minimize average delay. Equation 5.1 was used to calculate average delay:

$$avg\_delay = \frac{\sum_{i=1}^{n} delay_i}{n} \tag{5.1}$$

The number of packets that were successfully received by the destination node is represented by $recvd\_num$. Delay was calculated as follows:

$$delay_i = recv\_time[i] - send\_time[i] \tag{5.2}$$

5.3.2 Throughput

Throughput is the number of packets that were received over a period of time (e.g. seconds, milliseconds). The path with the high packet receiving rate is preferred. Equation 5.3 was used to calculate throughput:

$$throughput = \frac{recvd\_num}{sim\_time} \tag{5.3}$$

We represent the total simulation time by sim_time .

5.3.3 Packet loss ratio

Packet loss ratio is the percentage of sent packets that never reached the intended destination. An optimal route needs to have the lowest packet loss ratio. Packet loss ratio is calculated using the formula below:

$$pkt\_loss\_ratio = \frac{sent\_num - recvd\_num}{sent\_num} \times 100 \quad (5.4)$$

We represent the total number of sent packets by sim_num.

5.3.4 Delay jitter

Delay jitter is the variation in delay over time from point-to-point. Delay jitter is calculated as shown in equation 5.5.

$$delay_{jitter} = jitter + abs(e2eDelay - prev\_e2eDelay) \quad (5.5)$$

5.3.5 Packet drop ratio

Packet drop ratio is the percentage of packet that were successfully received by the intended destination of the nodes, but dropped due to other reasons. Packet drop ratio differs from packet loss ratio in that packet drop ratio looks at packets that were intentionally dropped because of certain reasons, while packet loss ratio looks at packets that never reached the intended destination node at all. One of the reasons could be that the packet was received in error. A formula for calculating packet drop ratio is shown in equation 5.6:

$$PDR = \frac{drop\_num}{recvd\_num} \times 100 \quad (5.6)$$

$drop\_num$ is the number of all dropped packets.

5.4 Experimental Setup

Network sizes of 9, 16, 25…. 196 nodes were simulated using square grid topologies for each experiment conducted. All nodes were placed in an area of 1500m x 1500m. The first two nodes were configured as sources while the last two nodes were configured as destinations. The AODV routing protocol was employed on each node. Each unique network size configuration was simulated 10 times and the results presented are the average of these simulation runs. All the nodes were kept stationary since we were simulating a WMN setup, and in WMNs nodes are stationary opposed to MANETs where the nodes are mobile. This work used five different performance metrics for simulations, keeping the performance metrics (throughput, delay, packet loss ratio, delay jitter, packet drop ratio) the same for each simulation. A summary of the simulation setup employed can be found in Table 5.1.

5.4.1 Routing Protocol

We used AODV routing protocol for all our simulations. AODV was chosen because it has been used with a majority of routing metrics that were to be simulated; AODV was suitable since the focus of this study was on the routing metrics rather than routing protocol. This routing protocol also provides simplicity and the fact that two of the routing metrics (Hop count and Expected Transmission Count) that were evaluated in this work have been used by this routing protocol before. Since the routing protocol uses Expected Transmission Count which uses time, it was not going to take much time and effort to modify it to suite the other two (Per-hop Round Trip Time and Exclusive Expected Transmission Time) routing metrics which also use time and are the descendents of the ETX routing metric.

5.4.2 Packet Buffer Model

Each node in our simulation used a buffer for both data packets and control packets while they are waiting for their turn to be transmitted. The used buffer can store a maximum of 50 packets at a time. A drop-tail queue management algorithm is employed, where the packets are transmitted in a first-come-first served basis. A packet is dropped when the buffer is full.

5.4.3 Physical and Data Link Layer

In our simulations, each node uses an omni-directional antenna. An omni-directional antenna can send data to all the directions with equal transmission strength. One of the advantages of using an omni-directional antenna is so that a node can broadcast packets to any direction at an equal strength, therefore there is no one direction that is better than other directions in the network.

This simulation used a link layer model that is based on the IEEE 802.11 protocol. The 802.11 standard uses a MAC layer known as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). In this method collision of packets never happen because each sender senses the channel before sending a packet, to check if the is no other node that is also trying to send at the same time. If the channel is available (no other node is sending), it sends a packet.

Table 0.1: Simulation parameters

Parameter (s)	**Value (s)**
Number of nodes	9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196
Area	1500m X 1500m
Network topology	Square grid
Simulation time	1000 seconds
Routing protocol	AODV
Routing metrics	HOP, RTT, ETX, and EETT
Recorded parameters	Delay, delay jitter, packet loss rate, packet loss ratio, and throughput

5.5 Simulation Experiments

This section presents results of the experiments that were conducted. Simulation parameters are given in Table 5.1. All results shown in the next subsections are the average results obtained from running each experiment 10 times.

5.5.1 Experiment 1: The effect of network size on delay

The purpose of this experiment was to determine the time that is taken by network layer packets from the sender to the receiver. In this work, the main focus was on network layer packets, because routing is done at the network layer of the ISO layer stack. Some routing metrics may have a high delivery ratio of packets that were sent in the network, but delay also needs to be taken into consideration.

Table 0.2: Effect of network size on delay

Number of Nodes	HOP	RTT	ETX	EETT
9	0.5	0.01	0.2	0.3
16	0.51	0.04	0.23	0.53
25	0.61	0.07	0.26	0.69
36	0.89	0.18	0.43	0.87
49	0.99	0.24	0.49	0.98
64	1.41	0.38	0.57	1.07
81	1.59	0.48	0.69	1.15
100	1.74	0.62	0.76	1.35
121	1.82	0.69	0.87	1.44
144	1.89	0.72	0.96	1.59
169	1.96	0.78	1.04	1.62
196	1.99	0.71	1.19	1.65

High delay decreases the overall performance of the network; therefore, an optimal routing metric for wireless mesh networks must have low delay. The delay that was measured in this work is an end-to-end delay. Delay was measured in seconds (s).

Table 5.2 depicts the simulation results showing the effect of network size on delay and Figure 5.1 graphically depicts the results of this experiment. Figure 5.1 shows that delay for all routing metrics that were simulated increased as the network size was increased. When the number of nodes increases in the network, there are more nodes that want to communicate. In wireless mesh networks, when a node is communicating, its neighbors have to wait for it to finish, this wait results in high delay.

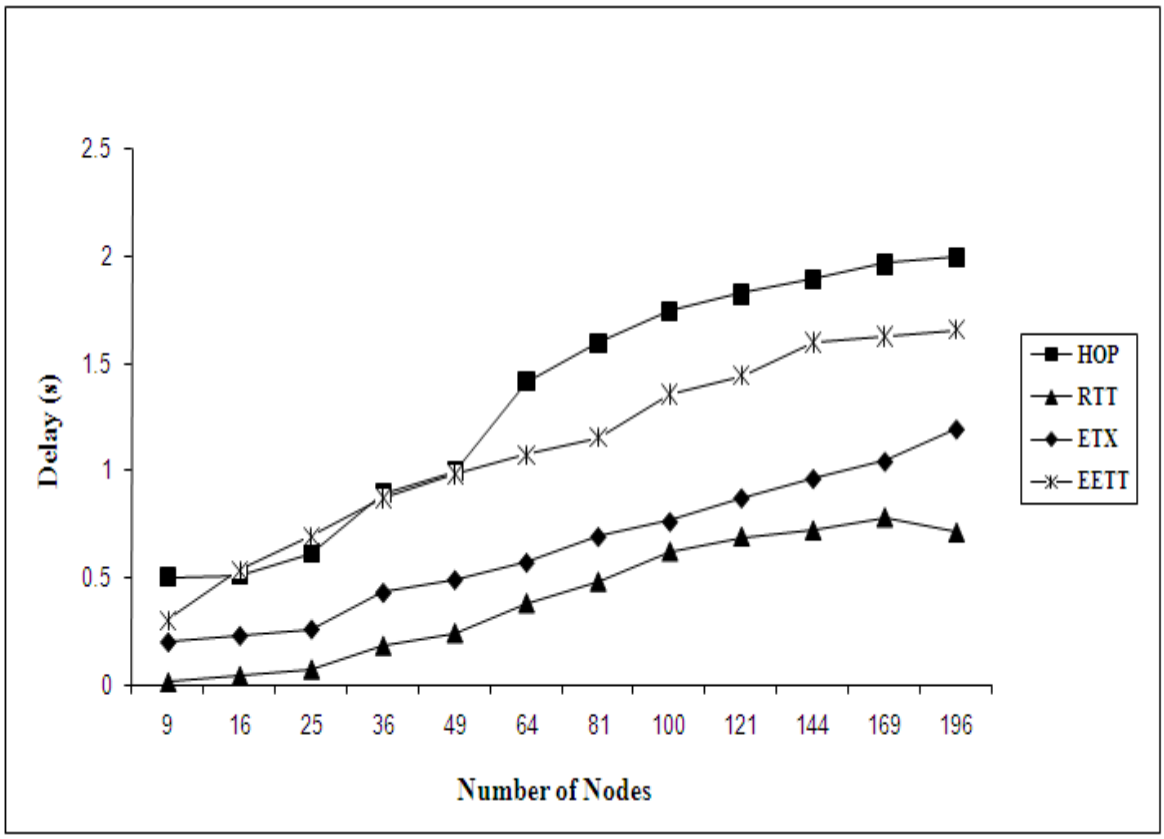

Figure 0.1: Effect of network size on delay

An optimal routing metric for WMNs should try by all means to minimize delay in the network. The sender might end up concluding that a packet has been lost, while it is actually still on its way, it was just experiencing delay. Hop count has the highest delay compared to all other routing metrics, meaning that it performed worst compared to the evaluated routing metrics.

In an ideal wireless mesh network, there is a high possibility that the shortest path will be through the middle of the network, but the middle of the network is usually highly congested. High delay experienced by hop count can be attributed to the fact that it chooses to send packets through a shortest path, which most of the time goes through the middle, especially if it is in a big network.

The longest paths are usually at the sides of a wireless mesh network, they should theoretically experience less congestion and delay compared to the routes through the center of a wireless mesh network. The hop count routing metric avoides using these routes, resulting in high delay and packet loss ratio. Experiment 3 shows that hop count experiences high packet loss ratio (see Figure 5.3).

RTT has the lowest delay of all the routing metrics followed by ETX. EETT performed better than HOP, but poorly compared to both RTT and ETX. Good performance produced by RTT is attributed to the fact that it was designed so that it minimizes delay and packet loss ratio by measuring load of the link and the packet loss experienced by that particul link before sending through it. RTT measures delay seen by unicast probes between neighboring nodes before sending or forwarding packets. ETX uses probe packets to measure packet loss ratio in all possible paths to the destination. Sending probe packets results in high delay, hence ETX had higher delay than RTT.

5.5.2 Experiment 2: The effect of network size on delay jitter

The purpose of this experiment was to determine the average delay jitter that is experienced by the network. Delay jitter is defined as the variation in delay, and it is measured in seconds (s). The variation in delay indicates the time that was taken by a packet to be successfully received by the intended destination, compared to the packet sent before or after it.

Table 0.3: Effect of network size on delay jitter

Number of Nodes	HOP	RTT	ETX	EETT
9	0.02	0.03	0.02	0.01
16	0.2	0.15	0.03	0.24
25	0.29	0.23	0.05	0.44
36	0.31	0.23	0.08	0.53
49	0.37	0.29	0.1	0.61
64	0.49	0.33	0.19	0.81
81	0.55	0.39	0.27	0.87
100	0.71	0.41	0.36	0.9
121	0.83	0.48	0.41	0.95
144	1.03	0.52	0.47	0.99
169	1.22	0.61	0.55	1.09
196	1.29	0.75	0.61	1.13

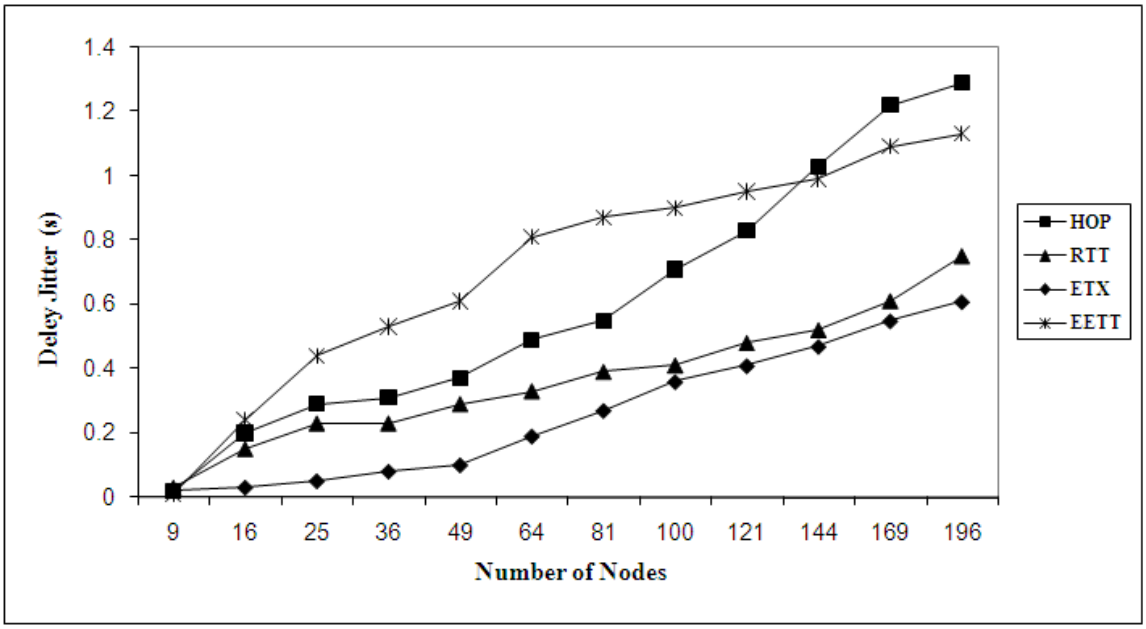

Figure 0.2: The effect of network size on delay jitter

A best routing metric for WMNs must try to keep this variation in delay as slitly as possible. In this work, delay jitter was also measured as one of the performance metric to compare the routing metrics. It is important to measure delay jitter because, for instance if delay of transmission varies too widely in real time applications the quality is greatly degraded. Table 5.3 depicts the

effect of network size on delay jitter and Figure 5.2 depicts the results of the simulation experiment. From the results, it was discovered that delay jitter increased when the network size was increased.

All routing metrics started with very small variation in delay (delay jitter), but ended with high variation in delay when the network size was increased up to 196 nodes (see Figure 5.2). The figure shows that ETX outperformed all the other routing metrics; it shows that ETX took less time to receive the next packet when there were few nodes in the network, but more time when the network contained many nodes. It is to be expected that ETX experiences less delay jitter, since it was seen in experiment 1 (see Figure 5.1) that it experienced less delay (although not the lowest of them all) because of the fact that it uses probe packets to measure delay in different paths before it sends packets. Results of RTT's delay jitter can also relate to those of experiment 1 (figure 5.1).

RTT produced second best (to ETX) delay jitter, which can be attributed to the fact that RTT's main objective is to decrease delay, hence it experiences less delay jitter. EETT was the worst performer of all the routing metrics. Hop count only performed better than EETT, but worse than the other two routing metrics that were simulated. Hop count does not consider the quality of the link resulting in a choice of congested links; hence it has high delay jitter and delay (as seen in Figure 5.1 and Figure 5.2).

Table 0.4: Effect of network size on packet loss ratio

Number of Nodes	HOP	RTT	ETX	EETT
9	29.62	3.03	2.22	4.57
16	33.54	4.06	3.43	9.21
25	48.81	4.87	4.95	16.2
36	50.35	6.56	5.04	17.77
49	56.35	7.13	5.45	30.99
64	61.63	12.13	6.15	38.21
81	67.58	14.13	8.23	42.97
100	79.83	17.13	10.29	49.95
121	83.03	23.03	13.33	53.77
144	89.37	25.76	14.53	53.68
169	91.39	27.05	16.73	57.2
196	92.07	27.78	19.97	64.69

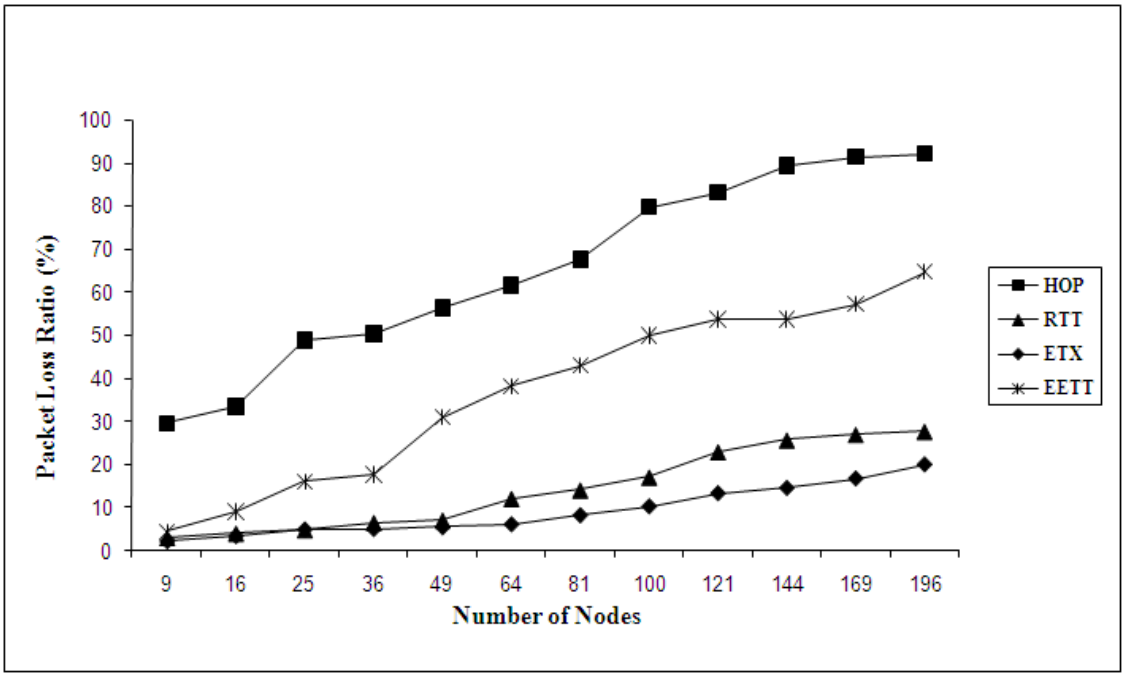

Figure 0.3: Effect of network size on packet loss ratio

5.5.3 Experiment 3: The effect of network size on packet loss ratio

The purpose of this experiment was to determine the effect of network size on the packet loss ratio. Packet loss ratio is measured as the overall percentage of the packets that never arrived to the intended destination node. Packet loss ratio is inversely proportional to packet delivery ratio. An optimal routing metric must minimize packet drop ratio so as to increase the overall data delivery ratio. Table 5.4 shows the effect of network size on packet loss ratio while Figure 5.3 depicts the results of this experiment in a graph. HOP has high packet loss ratio compared to all the other routing metrics, meaning that it performed badly compared to all other routing metrics that were simulated. A high level of packet loss ratio by HOP can be attributed to the fact that HOP may send packets through links highly congested, since it does not consider the quality of the link, but it just chooses the shortest path.

HOP tends to send packets through the paths that go through the middle, since the paths in the middle of the network are usually shorter than those in the outer parts of the network. High level of congestion in the middle of the network causes the increase in packet loss ratio, hence the high packet loss ratio suffered by hop count.

Table 0.5: Effect of network size on packet drop ratio

NN	HOP	RTT	ETX	EETT
9	1.24	1.43	0.41	2.07
16	2.28	1.25	0.38	2.37
25	1.94	1.22	0.86	2.56
36	2.89	2.34	1.98	5.05
49	3.05	5.04	2.32	6.77
64	3.15	8.09	2.74	6.96
81	4.67	10.04	4.45	7.94
100	6.68	10.04	5.37	7.99
121	7.84	11.09	6.04	8.69
144	7.93	12.42	6.04	9.16
169	8.89	12.05	8.27	9.28
196	8.76	10.96	8.25	10.32

ETX is the best performer, followed by RTT when measuring packet loss ratio. ETX measures the loss ratio of broadcast packets between pairs of neighboring nodes by estimating the number of retransmissions needed to successfully deliver a packet to the intended destination node. Packet loss ratio is the major concern of ETX, which is why it has the best performance; this can be seen in Figure 5.3. The good performance by RTT can be attributed to that it measures several facets of link quality, including packet loss ratio. EETT performed better when it is compared to HOP, but is the worst when compared to RTT and ETX.

5.5.4 Experiment 4: Effect of network size on packet drop ratio

The purpose of this experiment was to determine the amount of packets that are dropped at the destination node because they were received because of different resons. Packet drop ratio was measured as the number of routing layer packets that were dropped by the receiver during simulation (see Figure 5.4).

Not all packets that were sent by the source node reach the intended destination, some packets get lost while others are dropped because of different reasons. One of the reasons for dropping packets could be that they were received in error, while the other reason could be that they were received after its time to live (TTL) had expired. The best routing metric for WMNs should be able to identify packet that were received in error so that they can be dropped. A packet should have the same message at the receiving node as it had when it left its sender, otherwise the routing protocol should be able to identify and drop those packets. Table 5.5 depicts the effect of network size on packet drop ratio and Figure 5.4 graphically presents the results.

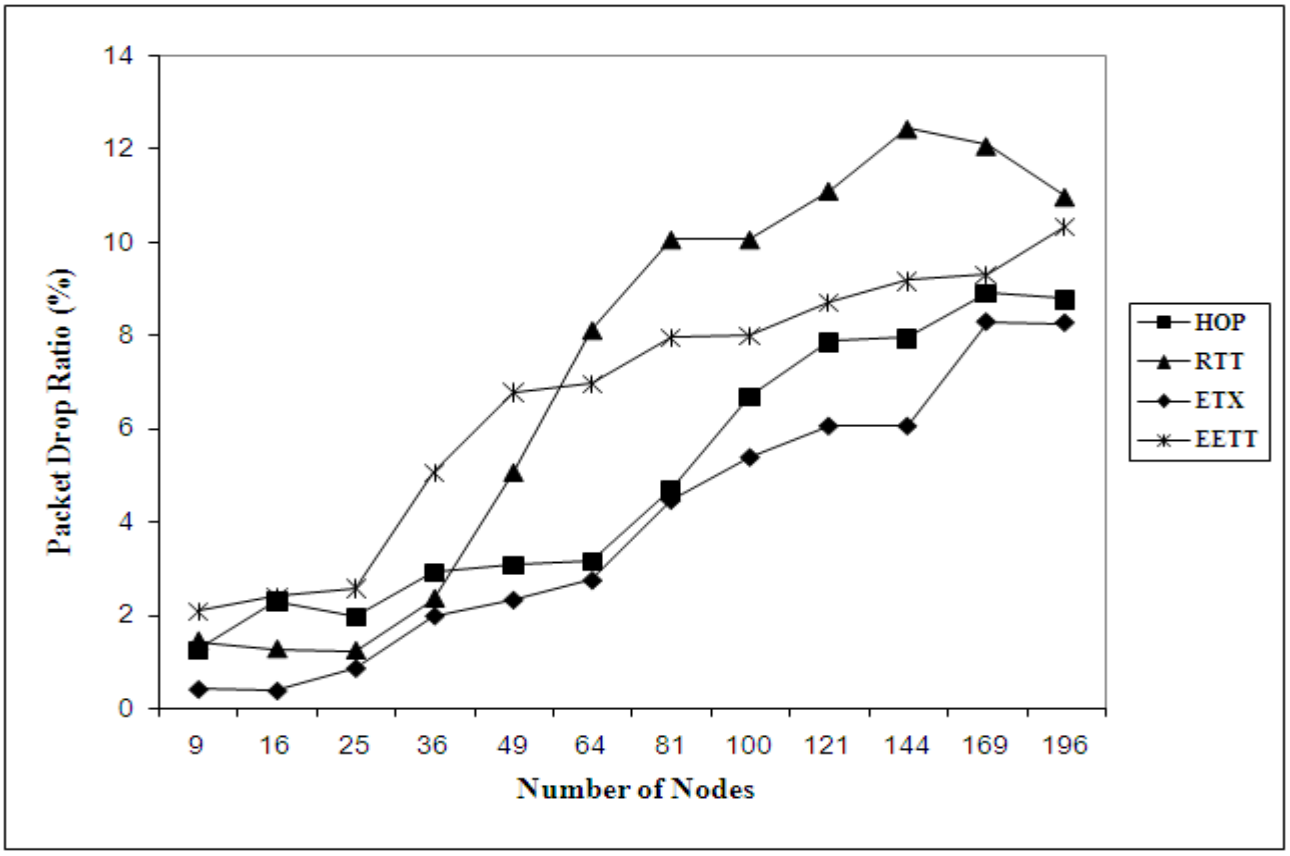

Figure 0.4: Effect of network size on packet drop ratio

Figure 5.4 shows that ETX performed better than all the other routing metrics simulated, meaning that it has a lower number of packets that were dropped. Figure 5.1 showed that ETX suffers from less delay, leading to nodes not waiting too long before forwarding packets. One of the factors that lead to packets being dropped is the high delay that the link is experiencing. ETX's low packet drop ratio can also be attributed to the fact that it suffers less packet loss ratio (see Figure 5.3), since ETX firstly measures loss ratio of the link. HOP performed better than EETT and RTT but poorly when compared to ETX.

EETT performed better than RTT, but worse than both ETX and HOP. RTT was the worst performer; it experienced the highest packet drop ratio. The worse performance by RTT can be attributed to the fact that it experiences high overhead from using probe packets to measure round trip time, to estimate how long does it take to successfully receive and acknowledge a packet.

5.5.5 Experiment 5: Effect of time on throughput

The purpose of this experiment was to determine the throughput of the network over time. Throughput is the number of packets that were received over a period of time (e.g. seconds, milliseconds). Throughput was measured in kilobytes per second (Kbps). Throughput decreases over time because when the simulation starts there will be exchange of packets between nodes, the exchange of packets will increase as the time goes on, leading to congestion on other links, which then negatively affect the throughput of the entire network.

Table 0.6: Effect of time on throughput of sending packets

Time (Seconds)	HOP	RTT	ETX	EETT
50	919.11	730.22	870.34	770.43
100	908.86	726.18	843.23	745.94
150	887.57	701.14	825.89	712.39
200	861.02	681.33	790.76	698.98
250	849.74	623.25	756.65	660.71
300	830.62	506.95	749.2	632.92
350	826.43	475.25	723.54	601.13
400	811.98	487.39	670.5	581.49
450	804.78	484.62	634.51	552.25
500	760.11	462.27	618.25	513.59
550	748.56	447.62	603.71	457.25
600	725.22	388.36	541.46	450.41
650	700.22	316.52	520.23	435.38
700	686.56	309.63	511.53	355.26
750	664.78	277.21	477.34	334.65
800	620.5	265.92	431.5	327.51
850	605.4	238.15	399.32	316.65
900	519.98	218.93	367.5	312.5
950	476.4	201.11	345.39	250.21
1000	430.89	189.17	321.97	251.5

Decrease in throughput over time is reflected by all the experiments that were conducted to measure throughput (see Figure 5.5, 5.6, 5.7). Three different experiments were conducted to measure throughput: throughput of sending packets, throughput of receiving packets, and throughput of forwarding packets.

An ideal routing metric for WMNs has the potential to increase the throughput of the network; hence we thoroughly look at throughput by conducting different experiments to measure throughput. Throughputs for various network sizes were collected and the average throughputs were recorded in each experiment. Throughput is one of the very important metrics to determine the performance of a network. We looked at throughput from different angles by further dividing

the experiments that measures throughput into: throughput of sending, forwarding, receiving, and dropping packets.

i. The effect of time on throughput of sending packets

The purpose of this experiment was to measure throughput at the sender so as to compare the success of different routing metrics when they were sending generated packets. This experiment measured the throughput of the network on the sender's side, while we also measure the throughput at the receiver's side and at the intermediate nodes (shown in Figure 5.6 and 5.7 respectively).

We wanted to have a thourough analysis of throughput, hence we looked at throughput at their different states. Table 5.6 depicts the effect of time on throughput of sending packets in a wireless mesh network and Figure 5.5 graphically depicts the results. Hop count had the best throughput of sending packets compared to all the metrics that were simulated, while RTT performed worse than all the other routing metrics. High throughput of sending packets by hop count can be attributed to the fact that it does not consider the quality of the link when selecting the path through which to send packet.

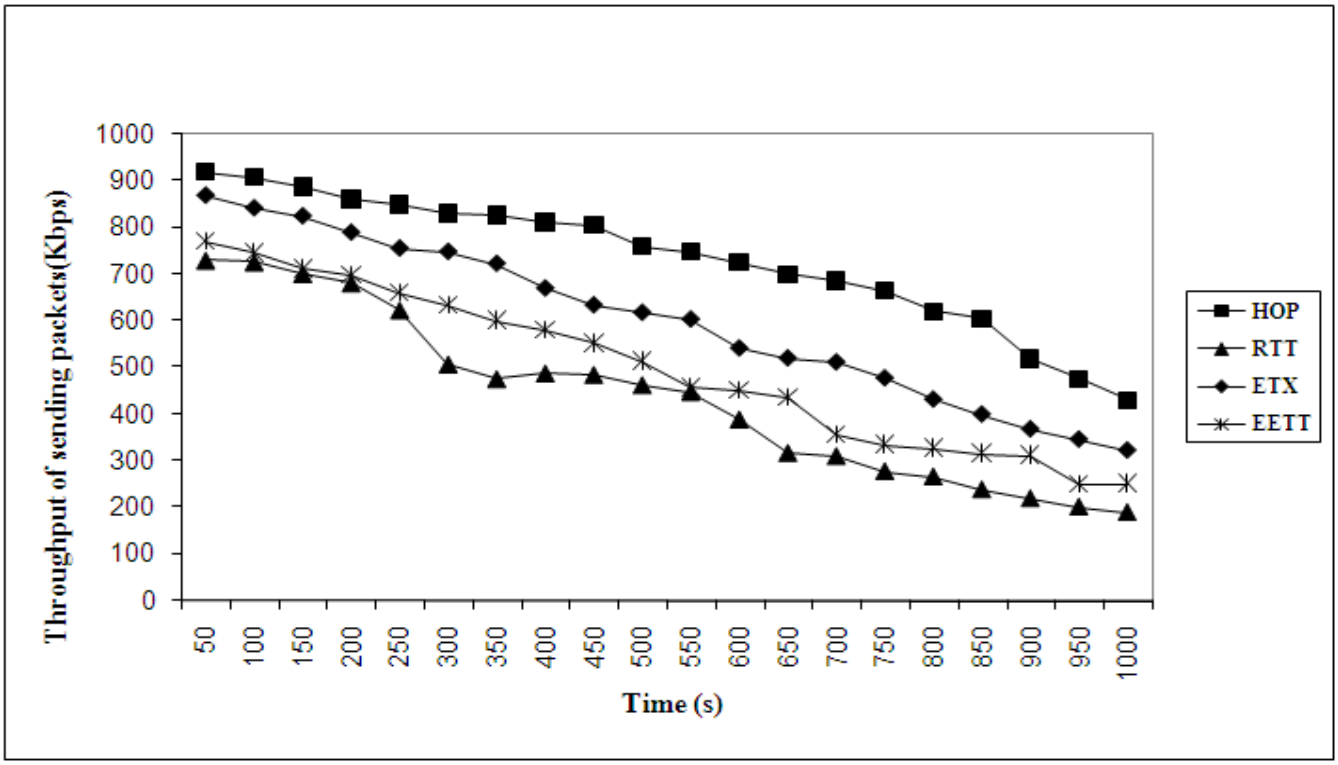

Figure 0.5: Effect of time on throughput of sending packets

Table 0.7: Effect of time on throughput of receiving packets

Time (Seconds)	HOP	RTT	ETX	EETT
50	232.55	278.34	452.71	350.75
100	221.28	263.39	437.49	320.95
150	211.15	246.25	426.61	301.38
200	203.71	234.37	404.63	298.23
250	193.38	218.65	370.85	267.65
300	187.32	208.39	324.78	253.95
350	174.35	194.63	328.12	241.29
400	173.13	187.69	302.32	223.94
450	171.21	156.97	298.27	215.29
500	162.28	132.42	282.62	203.16
550	152.75	124.22	261.26	187.93
600	144.23	109.35	244.25	173.93
650	123.75	106.65	236.25	165.75
700	105.45	103.49	220.95	141.35
750	98.35	102.15	225.41	132.31
800	95.11	97.45	225.61	123.15
850	83.65	95.24	212.13	118.34
900	78.25	87.95	221.92	112.91
950	56.05	82.85	220.29	105.49
1000	42.31	78.95	222.11	93.28

HOP only considers the path with the less number of hops, which makes the process of sending packets in a network using the hop count simple, leading to higher throughput of sending packet as it can be seen in Figure 5.5. ETX performed poorly when it was compared to HOP, but better when compared to the other two routing metrics (EETT and RTT). EETT performed better than RTT. RTT considers many aspects of link quality when selecting the path to use for sending packets, which makes it not simple to send packets using RTT, leading to poor throughput of sending packets.

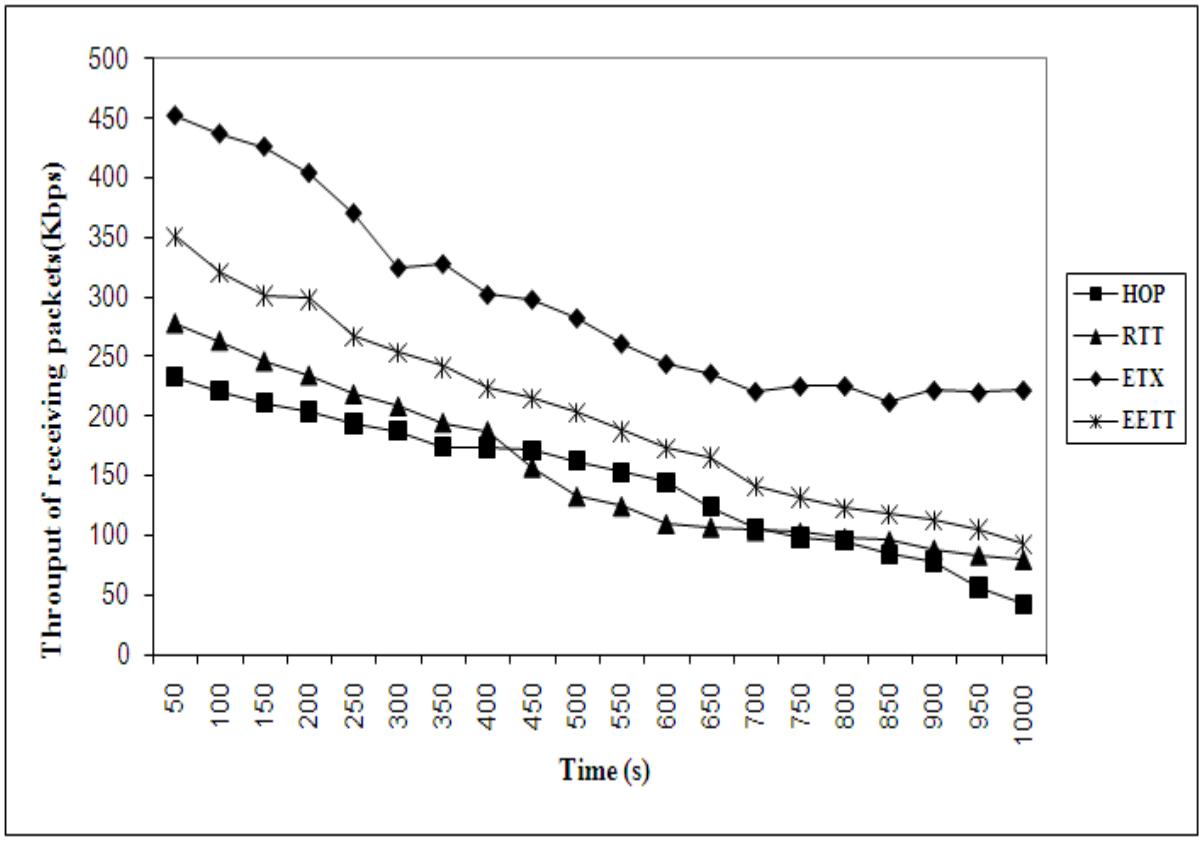

Figure 0.6: Effect of time on throughput of receiving packets

ii. The effect of time on throughput of receiving packets

The purpose of this experiment was to determine throughput of successfully receiving packets by the intended destination node. Table 5.7 shows the simulation results of the effect of time on throughput of receiving packets and Figure 5.6 depicts the results of the experiment in a graph. ETX had the highest throughput of receiving packets over time compared to all the other nodes that were simulated in this work. ETX was designed to improve throughput by measuring packet loss ratio of the links using probe packets before selecting the path to use for sending packets, this is echoed by high throughput of receiving packets in Figure 5.6 and low packet loss ratio in experiment 3 (see Figure 5.3) HOP was the worst performer.

EETT performed badly when compared to ETX, but better than HOP and RTT. RTT performed better when it was compared to HOP. Poor performance by HOP can be attributed to the fact that since it does not consider other aspects of the link except the path length, it may choose a shortest route that has congested links leading to the high loss ratio. Routes in the center are usually the shortest routes, but very congested as well. Hop count will send packets through the middle of the network most of the time since it chooses the shortest path, resulting in many packets getting lost before they reach their intended destination, which causes low throughput at the receiving node. Experiment 3 also confirmed (Figure 5.3) that HOP has the highest packet loss ratio, resulting in low throughput of receiving packets.

iii. The effect of time on throughput of forwarding packets

The purpose of this experiment was to measure the success of relaying packets by intermediate nodes to the next chosen node (if the next node is not the destination), or to the intended destination node.

Table 0.8: Effect of time on the throughput of forwarding packets

Time (Seconds)	HOP	RTT	ETX	EETT
50	68.38	46.58	69.42	56.06
100	68.11	45.43	68.35	54.33
150	65.55	42.88	67.83	54.17
200	64.45	35.34	67.17	51.56
250	61.3	27.46	66.45	51.16
300	60.45	26.43	64.48	49.76
350	60.35	24.94	64.05	49.33
400	58.34	24.02	62.3	46.56
450	57.55	23.14	61.17	44.89
500	55.24	21.17	59.4	36.78
550	54.35	19.86	56.5	30.67
600	52.75	19.17	55.9	29.28
650	51.13	16.8	53.46	26.83
700	47.58	16.4	47.7	24.78
750	43.35	15.9	44.9	24.11
800	41.59	15.7	43.3	24.05
850	38.4	13.3	41.5	23.33
900	32.5	13.8	36.9	23.16
950	23.5	13.3	34.4	22.22
1000	17.5	12.9	31.35	16.3

Nodes in WMNs do not always have direct communication with every other node. Whether a node can directly send packets to another node depends on the transmission range of the two nodes that want to communicate. If the two nodes are within communication range, they can directly communicate, otherwise they require assistance from others along the chosen route to relay packets on behalf of the sending node. In this work not only throughput of receiving packets by the destination node is measured, but throughput of successfully forwarding packets by relay nodes as well as throughput of sending packets by a source node is also measured.

Table 5.8 depicts simulation results of the effect of time on throughput of forwarding packets and Figure 5.7 depicts results results in a graph. Figure 5.7 shows that ETX had higher throughput of forwarding packets by intermediate nodes compared to all other routing metrics that were simulated in this work. The best performance by ETX can be attributed to the fact that it is more concerned with packet loss ratio. Experiment 3 (see Figure 5.3) shows that ETX has got lower packet loss ratio. ETX measures loss ratio of the possible links that it may use to send packets. Measuring packet loss ratio of the link makes it easy for the intermediate nodes to choose links with low packet loss ratio, leading to successfully forwarding a large number of packets, hence high throughput of forwarding packets opposed to the other routing metrics.

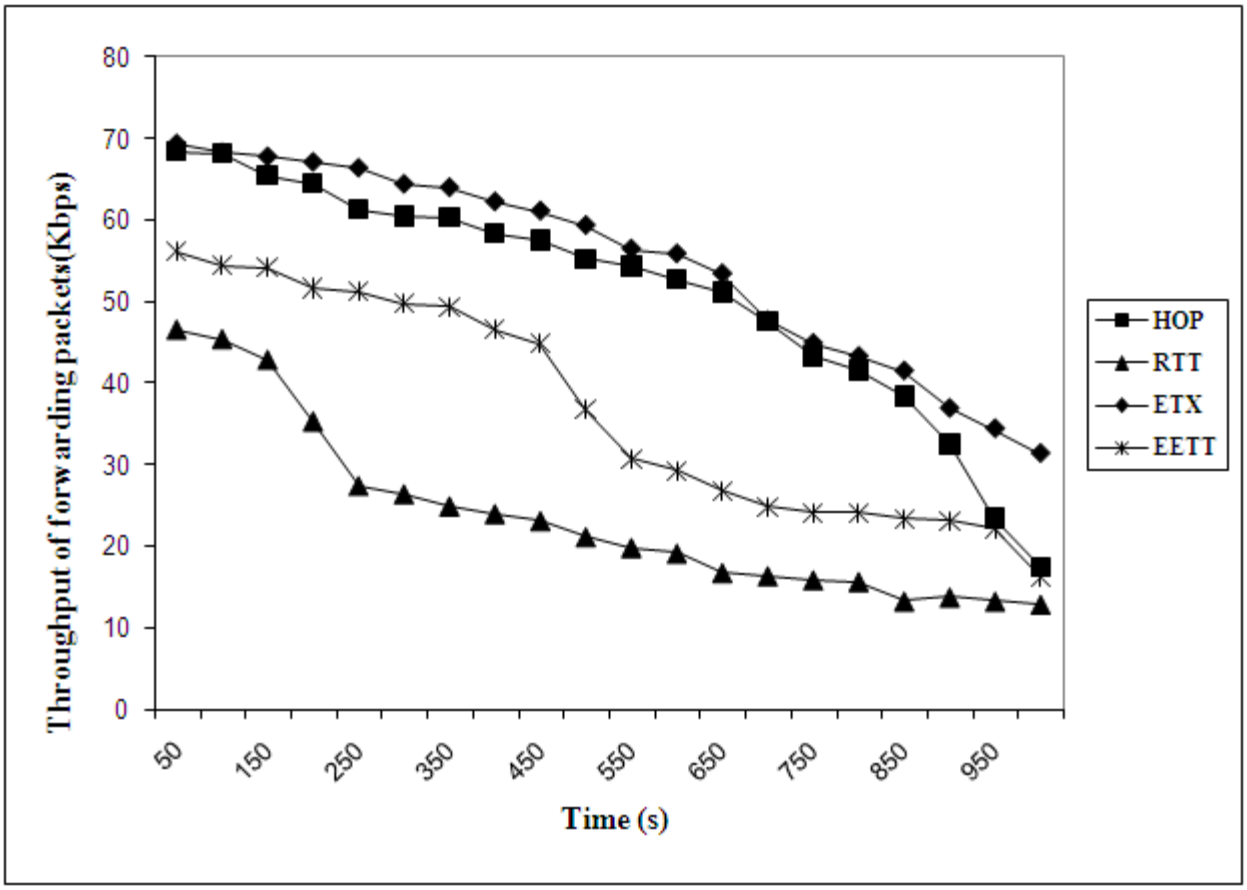

Figure 0.7: Effect of time on the throughput of forwarding packets

Hop count routing metric was the second best performing routing metric (to ETX). Although hop count routing metric had low throughput of receiving packets, it performed better when throughput of forwarding packets by the intermediate nodes was measured. RTT was the worse performer when measuring throughput of forwarding packets by relay nodes. EETT was the second worst performer (to RTT).

5.5.6 Summary of Results

Table 5.9 summarizes the performance of the four routing metrics that were simulated in this chapter. Routing metrics were assigned scores from 1 – 4, with 1 being the worst performance while 4 represents best performance. The last row on Table 5.9 shows the average score of the performance of the routing metrics. It can be seen from Table 5.9 that overall ETX outperformed all the routing metrics, while HOP performed worse than all the routing metrics that were simulated. ETX outperformed all the routing metrics that were simulated when delay jitter in the entire network was measured, it was also the best when measuring packet drop ratio and throughput of receiving packets by the destination nodes. ETX also performed better than all the other routing metrics that were simulated when measuring throughput of forwarding packets by intermediate nodes.

The best performance by ETX can be attributed to the fact that it was designed to improve throughput of the network. ETX measures packet loss ratio of broadcast packets between pairs of neighboring nodes by estimating the number of transmissions that are required to send unicast packets. ETX measures packet loss probability in both the forward and reverse directions. ETX

uses probe packets for measuring loss ratio, which results in high delay, although it is not higher than that of HOP and EETT.

Table 5.9 shows that the hop count's overall performance was the worst compared to all the other routing metrics that were simulated.

Table 0.9: Summary of results

Experiment	HOP	RTT	ETX	EETT
Delay	1	4	3	2
Packet loss ratio	1	3	4	2
Delay jitter	2	3	4	1
Packet drop ratio	3	1	4	2
Throughput of sending packets	4	1	3	2
Throughput of receiving packets	1	2	4	3
Throughput of forwarding packets	1	3	4	2
Overall performance	**1**	**3**	**4**	**2**

Poor performance by hop count can be attributed to the fact that it does not consider the quality of the link. Hop count only considers the route that has the few links. HOP uses random selection to select the route to use among the paths that have the same number of links. In a big network, hop count normally sends packets through the middle of the network, as the shortest routes usually pass through the middle. When the network size increases, the middle becomes more congested, resulting to high delay and packet loss ratio for HOP (as it can be seen on Figure 5.1).

The results from the work by (Yang et. Al., 2006), discussed in Chapter Two showed that MIC performed best compared to all the other routing metrics that were evaluated in that work. The evaluation conducted in our research project has shown that ETX was the best performer

amongst the four routing metrics that were compared. ETT was the worst performing routing metric in the work by (Yang et. Al., 2006), while in our study the evaluation showed hop count as the worst performing routing metric. Our work used the same simulation environment, same routing protocol, and same routing protocol for all the experiments, which could be the reason why the best and the worst routing metrics are different from those in the work by (Yang et al., 2006). The other study by (Liu et al., 2008) just conducted a theoretical analysis of the routing metrics; they did not conduct any experiments.

5.6 Recommendation of Design Criteria

The last objective of this research project is to recommend design criteria for an optimal routing metric for wireless mesh networks (WMNs). This section fulfilled this objective by presenting four design criteria. This section recommended four design criteria after the evaluation of routing metrics was done earlier in this chapter. Recommended design criteria are: weight path-awareness, efficient weight path algorithm design, Quality of Service-awareness, and network scalability

5.6.1 Weight path-awareness

The main goal of every routing protocol is to send packets through minimum weight paths in terms of a particular routing metric that it is using. To make sure that there is an effective utilization of wireless mesh network resources, the minimum weight paths selected by a particular routing protocol must have good performance which involves high throughput, low packet loss and low delay. Figure 5.1 depicts the effect of network size on delay, which shows

high delays for the routing metrics considered in this study. Figure 5.5 through 5.7 shows different throughputs which need to improve for an ideal routing metric for WMN. For this to be achieved, the routing metrics must be able to capture the features of wireless mesh networks that impact the performance of paths. (Yang et al., 2006) also recommended this design criterion in their work. These features are briefly discussed below:

a) Path length

Each link of a path creates extra delay and potentially more packet loss, a longer route normally increases the end-to-end delay while it reduces throughput and packet loss ratio. An optimal routing metric for wireless mesh networks should be able to increase the weight of a route when the route's length increases. Hop count selects the route that has the smallest number of hops without considering the weight of a link. An ideal routing metric for WMNs needs to try and avoid this approach. Other routing metrics such as ETX do take the quality of the link into consideration.

b) Link capacity

In wired networks a link's capacity is independent of the physical distance between the link's points, but in wireless networks the transmission rate between a pair of neighboring nodes (link capacity between two nodes) is directly related to the physical distance between two nodes. In simple terms, channel quality decreases as distance between two nodes increases.

c) Packet loss ratio

A source node might have to transmit a packet several times on a hop with high packet loss ratio, which affects throughput and delay of any flow that goes through that hop. An ideal routing metric for WMNs must be able capture the packet loss ratio to ensure good performance for minimum weight path. There is a need for an improvement on the packet loss ratio for the routing

metrics depicted in Figure 5.3. The figure shows very high loss ratio for HOP and EETT while the loss ratios for both ETX and RTT are a bit lower, but they still loose a lot of packets. A best routing metric for WMNs must decrease the packet loss ratios seen in Figure 5.3.

5.6.2 Efficient weight path algorithm design

Each and every routing protocol uses a certain efficient algorithm to compute minimum weight paths. It can not be guaranteed that routing protocol can have good performance if there is no efficient algorithm to calculate the minimum weight paths based on a particular routing metric. The results of the experiments have shown the hop count routing metric as the worse performer amongst all the simulated routing metrics, this can be attributed to the fact that it uses an algorithm that is not very efficient since it does not consider the quality of a link, but only chooses a path that has less number of hops among the available paths. Table 5.9 shows the summary of the performance of the evaluated routing metrics, and the performance of hop count is significantly worse, leading to the conclusion that there is a need for the design of an effective weight path algorithm to be used by the best routing metric for wireless mesh networks. This design feature of a wireless mesh network routing metric was also recommended by (Yang et al., 2006).

5.6.3 Quality of Service-awareness

Quality of Service has a potential to improve performance of a wireless mesh network. A routing metric like hop count does not consider the quality of the link through which it is intending sending packets. The fact that hop count does not consider link quality degrades network

performance by having high packet loss ratio and low throughput (see Figure 5.2 and 5.6). An optimal routing metric for WMNs need to make QoS one of its priorities. Hop count routing metric does not take the quality of the link into consideration when choosing a route to use for sending packets. Experimental results in section 5.5 showed that hop count performed badly, and its poor performance can be attributed to the fact that it can send packets through congested links as a result of the bad quality of that particular link.

5.6.4 Network scalability

Most of experiments conducted in section 5.5 of this research work have shown that the performance of the network (i.e. throughput, packet delivery ratio) degrades when the network size increases. Figure 5.5, 5.6 and 5.7 show that throughput of the network degrades as the network size is increased for all routing metrics that were simulated. Experiment 3 and 4 also showed that packet loss ratio and packet drop ratio respectively increase when the network size increases, leading to poor performance of the entire network. Delay is also one of the causes of performance degradation; hence it increases as the network size increases, leading to poor throughput. An optimal routing metric for wireless mesh network needs to take network scalability into consideration.

A routing metric should try to maximize network performance despite the number of nodes the network has. Results presented in section 5.5 in this study revealed that the performance of all the routing metrics was decreasing as the network size was increased. Figure 5.1 has shown that network delay increased as more nodes were added in the network, while figure 5.2 has shown that the delay jitter increased as well when more nodes were added. As more nodes were added,

packet loss ratio and packet drop ratio increased as it can be seen in Figure 5.3 and Figure 5.4 respectively. It can be seen in Figure 5.5 through Figure 5.7 that throughput also decreased as the network size was becoming bigger.

5.7 Summary

The simulation experiments were conducted in this chapter and based on the results analysis, the design features for an optimal routing metric for WMNs were recommended in section 5.6. ETX performed best compare to all the other routing metrics that were simulated, but it still has its own disadvantages that do not make it to be an automatic choice as an optimal routing metric for wireless mesh networks. ETX uses probe packets to measure the packet loss ratio of all possible paths to the intended destination. The process of measuring packet loss ratios for all the possible paths results to high network overhead, which results in high delay. An optimal routing metric for wireless mesh network should try by all means to reduce overall delay while it maximizes overall throughput of the network. Four design features for an ideal routing metric for wireless mesh network were proposed in section 5.6 of this chapter. Chapter Six concludes this study and provides future work for enhancing this work.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This study is a successful attempt to investigate existing routing metrics to find out if there is any routing metric among existing routing metrics that can work better in wireless mesh networks, as all the existing routing metrics were designed with mobile ad hoc networks (MANETs) in mind. After comparing different routing metrics, four routing metrics were simulated in NS2 with WMNs simulation environment, and then their performance was evaluated. The main goal of this study was to evaluate the performance of routing metrics for wireless mesh networks. It was important to firstly evaluate routing metrics that already exist, so as to be sure whether there is a routing metric that works for WMNs, although they were designed for MANETs they might work for wireless mesh networks.

This study answered the following research questions: (1) what are the pitfalls in existing WMNs routing metrics? (2) Which routing metric among the existing routing metrics is the best for WMNs? (3) What features should be considered when designing a routing metric for WMNs? The goal of this study was broken down to four objectives that needed to be fulfilled to complete the study. Achieving the set objectives also provided answers to the research questions defined in section 1.3 of Chapter One. This study had the following objectives: (1) to survey relevant literature on existing routing metrics, (2) to use the knowledge gathered from the survey to select representative sample of routing metrics, (3) to evaluate the performance of existing routing

metrics, (4) to recommend design criteria for an optimal routing metric for wireless mesh network.

The first objective was fullfiled by reviewing two existing studies that compared routing metrics for wireless mesh networks and the review of twenty existing routing metrics. This review of the literature answered the first research question. Routing metrics were later grouped into four groups; the routing metrics in each group were then compared among each other to find one routing metric that was going to represent the group during simulation and evaluation. The second research objective was fullfiled by this comparison of routing metrics. The four selected routing metrics were then implemented and evaluated in NS2, by so doing fulfilling objective three. By successfully evaluating the performance of routing metrics, we answered the second research question.

As a result of the evaluation, we discovered that hop count performed worse than all the routing metrics that were compared in this study. EETT performed better than HOP, but worse than both ETX and RTT. RTT performed better than both EETT and HOP, but poor when compared to ETX. Through the analysis of results, we achieved objective number tree. From the foregoing we drew the conclusion that ETX outperformed all other routing metrics that were simulated, but based on the evaluation, it can not be concluded that it is the best routing metric for WMNs. By arriving at this conclusion, we answered the last research question.

The design criteria that informed this study can help guide any scholar that wishes to design a new optimal routing metric for WMNs from scratch or extend already existing routing metric.

By coming up wit these designed criteria; we were able to fulfill the fourth research objective and also anwered the third research question. Recommended design criteria were as follows: (1) weight path-awareness, (2) efficient weight path algorithm design, (3) quality of service-awareness, (4) network scalability. Research questions two and three are also answered by recommending design criteria for an optimal routing metric for wireless mesh networks. The next section (6.2) presents limitations of this work and future enhancements.

6.2 Limitations and Future Work

This section presents the limitations and future enhancements of this study. One of the limitations of this work is that simulation results might not reflect real world results since they do not consider factors like external interference. Conducting the same experiments on a wireless test bed still needs to be considered to further validate results obtained. This study needed a fairly large number of wireless nodes to test network scalability; this would not have been possible with the test bed that is running in the wireless mesh lab at the University of Zululand, because it contains only fourteen nodes. The work required up to 196 wireless nodes. Test bed implementation was not possible because of time and financial constrains. As future work, this study should consider using a test bed which will reflect real world results. The results from the test bed should be compared with simulation results from this work.

One routing protocol (AODV) was used in all the simulation experiments that were conducted in this study, another routing protocol could be used to also run the very same experiments. The use of two routing protocols instead of one should be considered as future enhancement of this work. The use of two routing protocols will help to further validate simulation results, since the

performance of routing metrics will then be judged based on two different routing protocols used. Hybrid wireless mesh protocol is a routing protocol for WMNs, it should be considered as the second routing protocol to use in this study.

In this study, only one routing metric was chosen for simulation from each group. Only one routing metric could be simulated from each group because of time constrains, since the code for the other three routing metrics (RTT, ETX, and EETT) had to be hard coded before they could be used, which took a lot of time to achieve. Selecting more than one routing metric from each group was going to take even more time. As future enhancement to this study, simulating two routing metrics from each group should be considered.

BIBLIOGRAPHY

Adya, A. Bahl, P. Padhye, J. Wolman, A. & Zhou, Z. 2004. A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks. *Proceedings of the First International Conference on Broadband Networks (BROADNETS'04),* Washington, DC, USA, October 2004, pp 344-354, Washington, DC, USA: IEEE Computer Society.

Akyildiz, I.F. Wang, X. & Wang, W. 2005. Wireless Mesh Networks: a survey. *Computer Networks*. 43 (9): 445-487, September 2005.

Akyildiz, I.F. *Wang, X. 2009. Wireless Mesh Networks*. Georgia Institute of technology, USA.

Aoki, H. Abraham, S. Agre, J. Bahr, M. Chari, N. Cheng, R. & Chu, L. Conner, W. Faccin, S. 2005. 802.11 TGs Simple Efficient Extensible Mesh (SEE-Mesh) Proposal. *IEEE P802.11 Wireless LANs*, Document IEEE 802.11-05/0562r0, June 2005.

Bahr, M. 2006. Proposed routing for IEEE 802.11s WLAN mesh networks. *Proceedings of the 2nd Annual international Workshop on Wireless Internet conference*, Boston, MA, United States, 02-05 August 2006, New York, NY, USA: ACM

Biaz, S. & Qi, B. 2008. iETT: A Quality Routing Metric for Multi-Rate Multi-Hop Networks. *IEEE Wireless Communications and Networking Conference*, *WCNC 2008,* Las Vegas, NV, 31 March 2008-03 April 2008, pp. 2729 – 2734, Las Vegas, NV: IEEE.

Corson, M. & Ephremides, A. 1995. A Distributed Routing Algorithm for Mobile Wireless Networks. *Wireless Networks*, 1 (1): 61-81, January 1995.

De Couto, D, S, J. Aguayo, D. Bicket, J & Morris, R. 2005. A High-throughput Path Metric for Multi-Hop Wireless Routing. *Wireless Networks*, 11(4): 419-434, July 2005.

De Couto, D. S. J. 2003. High-Throughput Routing for Multi-Hop Wireless Networks. Massachusetts: Massachusetts Institute of Technology. (M.Eng.).

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (1): 269-271. 1 December 1959.

Draves, R. Padhye, J. & Zill, B. 2004. Routing in Multi-Radio Multi-Hop Wireless Mesh Networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking,* Philadelphia, USA, September 2004, pp. 114 - 128, New York, USA: ACM.

Draves, R. Padhye, J. & Zill, B. 2004. Comparison of Routing Metrics for Static Multi-hop Wireless Networks. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications (SZGCOMM),* Portland, Oregon, USA, August 2004, pp.133 – 144, New York, NY, USA: ACM.

IEEE 802.11 WG TGs, Draft Amendment to Standard IEEE 802.11™: ESS Mesh Networking, P802.11s™ /D1.08, Jan. 2008.

Jacquet, P. Muhlethaler, P. Clausen, T. Laouiti, A. Qayyum, A. & Viennot, L. 2001. Optimized Link State Routing Protocol for Ad hoc Networks. Multi Topic

Conference. *IEEE INMIC 2001, Technology for the 21st Century, In Proceedings, IEEE International Conference,* pp. 62-68. 2001.

Jiang, W. Liu, S. Zhu, Y. & Zhang, Z. 2007. Optimizing Routing Metrics for Large-Scale Multi-Radio Mesh Networks. In *Proceedings of International Conference on Wireless Communications*. *Networking and Mobile Computing*, Shanghai, 21-25 September 2007, pp. 1550 – 1553, Shanghai: IEEE.

Johnson, D.B. Maltz, D.A. & Broch, J. 1996. DSR: The Dynamic Source Routing for Multihop Ad Hoc networks. In *Ad hoc Networking*, Boston, MA, USA, 2001, pp. 139-172, Addison-Wesley Longman Publishing Co., Inc: Boston, MA, USA.

Johnson, D. & Hancke, G. Comparison of two routing metrics in OLSR on a grid based mesh network. From http://wirelessafrica.meraka.org.za/ wiki/ images/ 8/8d/Els evier2008_OLSR_compare.pdf.

Johnson, D.L., Aichele, C. & Ntlatlapa, N. 2008. A Simple pragmatic approach to mesh routing using BATMAN. *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries (WCITD' 2008)*, CSIR, Pretoria, South Africa, 6-7 October 2008, pp. 10, Pretoria, S.A: Scientific Common.

Keshav, S. 1991. A Control-Theoretic Approach to Flow Control. *ACM SINGCOMM Computer Communication Review,* 21 (4): 3-15, September 1991.

Koksal, C.E. & Balakrishnan, H. 2006. Quality-Aware Routing Metrics for Time-Varying Wireless Mesh Networks. *IEEE Journal on selected areas in communications*, 24 (11): 1984-1994, November 2006.

Kyasanur, P. & Vaidya, N, H. 2005. Routing and Link-layer Protocols for Multi-Channel Multi-Interface Ad Hoc Wireless Networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 10(1): 31-43, January 2006.

Liu, H. Huang, W. Zhou, X. & Wang, X.H. 2008. A Comprehensive Comparison of Routing Metrics for Wireless Mesh Networks. *Proceedings of the IEEE International Conference on Networking, Sensing and Control, ICNSC 2008*, Hainan, China, 6-8 April 2008, pp. 955 – 960, Sanya: IEEE.

Ma, L. & Denko, M. 2007. A Routing Metric for Load-Balancing in Wireless Mesh Networks. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07),* 2007, Niagara Falls, Ontario, Canada, 21-23 May 2007, pp. 409-414, Washington, DC, USA: IEEE Computer Society.

Ni, Xian. Lan, Kun-chan. Malaney, Robert. 2008. On the Performance of Expected Transmission Count (ETX) for Wireless Mesh Networks. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, Athens, Greece, Brusses, October 2008, Belgium: ICST.

Nxumalo S.L, Ntlatlapa, N, Mudali, P, & Adigun, M.O. 2009. Performance Evaluation of Routing Metrics for Wireless Mesh Networks. In *Procedings of South African Telecommunication Networks and Applications Conference*, Royal Swazi Spa, Swaziland, 30 August – 2 September 2009.

Parissodis, G. Karaliopoulos, M. Baumann, R. Spyropoulos, T. & Platter, B. 2009. Routing Metrics for Wireless Mesh Networks. (*In* Misra, S. Misra, S.C. Woungang, I. (eds.), *Guide to Wireless Mesh Networks*. London: Springer, pp. 199-230. 2009).

Perkins, C. 1997. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications,* Washington, DC, USA, 1999, pp. 90, Washington, DC, USA: IEEE Computer Society.

Ramanathan, R. & Redi, J. A. 2002. A Brief Overview of Ad hoc Networks: Challenges and Directions. *IEEE Communications Magazine, BBN Technologies,* 40 (5):20-22, May 2002.

Shen, Q. & Fang, X. 2006. A Multi-metric AODV Routing in IEEE 802.11s. In *International Conference on* Communication Technology, 2006, ICCT '06, *ICCT' 06,* Guilin, 27-30 November 2006. Pp1-4, Guilin: IEEE.

Shila, D. M. & Anjali, T. 2007. Load-aware Traffic Engineering for Mesh Networks. In *Proceedings of 16th International Conference, Computer Communications and Networks*. *ICCCN 2007*, Newton, MA, USA, May 2008, pp. 1040-1045, Newton, MA, USA: Butterworth-Heinemann.

Subramanian, A, P. Buddhikot, M, M. & Miller, S. 2006. Interference Aware Routing in Multi-Radio Wireless Mesh Networks. *2nd IEEE Workshop on* Wireless Mesh Networks, 2006, WiMesh 2006, Reston, VA , 25-28 September 2006, pp. 55 - 63, Reston, VA: IEEE.

Yang, Y., Wang, J. & Kravets, R. 2005. Interference-aware Load Balancing for Multi-hop Wireless Networks, Technical Representative. UIUCDCS-R- 2005-2526, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

Yang, Y., Wang, J. & Kravets, R. 2006. Designing Routing Metrics for Mesh Networks. University of Illinois at Urbana-Champaign. 2006.

Yin, S. Xiong, Y. Zhang, Q. & Lin, X. 2006. Prediction-based Routing for Real Time Communications in Wireless Multi-hop Networks. In *Proceedings of the 3rd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks*, Waterloo, Ontario, Canada , 07-09 August 2006, New York, NY, USA: ACM.

Zhang, Q. 2005. Video delivery over wireless multi-hop networks. *ISPACS 2005, In Proceedings of 2005 International Symposium on Intelligent Signal Processing and Communication Systems*, 13-16 December 2005, pp. 793-796.

Zhou, W. Zhang, D. & Qiao, D. 2006. Comparative study of routing metrics for multi-radio multi-channel wireless networks. *Procedings of Wireless Communications and Networking Conference 2006*. *WCNC 2006*. *IEEE*, 3-6 April 2006, Las Vegas, NV. pp.270-275,

APPENDIX A: AODV Header File

```
#ifndef __aodv_rtable_h__
#define __aodv_rtable_h__
#include <assert.h>
#include <sys/types.h>
#include <config.h>
#include <lib/bsd-list.h>
#include <scheduler.h>
#define CURRENT_TIME Scheduler::instance().clock()
#define INFINITY2 0xff
/*
AODV Neighbor Cache Entry
*/
class AODV_Neighbor {
friend class AODV;
friend class aodv_rt_entry;
public:
AODV_Neighbor(u_int32_t a) { nb_addr = a; }
protected:
LIST_ENTRY(AODV_Neighbor) nb_link;
nsaddr_t nb_addr;
double nb_expire; // ALLOWED_HELLO_LOSS * HELLO_INTERVAL
};
LIST_HEAD(aodv_ncache, AODV_Neighbor);
/*
AODV Precursor list data structure
*/
```

```
class AODV_Precursor {
friend class AODV;
friend class aodv_rt_entry;
public:
AODV_Precursor(u_int32_t a) { pc_addr = a; }
protected:
LIST_ENTRY(AODV_Precursor) pc_link;
nsaddr_t pc_addr; // precursor address
};
LIST_HEAD(aodv_precursors, AODV_Precursor);
/*
Route Table Entry
*/
class aodv_rt_entry {
friend class aodv_rtable;
friend class AODV;
friend class LocalRepairTimer;
public:
aodv_rt_entry();
~aodv_rt_entry();
void nb_insert(nsaddr_t id);
AODV_Neighbor* nb_lookup(nsaddr_t id);
void pc_insert(nsaddr_t id);
AODV_Precursor* pc_lookup(nsaddr_t id);
void pc_delete(nsaddr_t id);
void pc_delete(void);
bool pc_empty(void);
double rt_req_timeout; // when I can send another req
```

```
u_int8_t rt_req_cnt; // number of route requests
protected:
LIST_ENTRY(aodv_rt_entry) rt_link;
nsaddr_t rt_dst;
u_int32_t rt_seqno;
/* u_int8_t t_interface; */
u_int16_t rt_hops; // hop count
int rt_last_hop_count; // last valid hop count
nsaddr_t rt_nexthop; // next hop IP address
/* list of precursors */
aodv_precursors rt_pclist;
double rt_expire; // when entry expires
u_int8_t rt_flags;
#define RTF_DOWN 0
#define RTF_UP 1
#define RTF_IN_REPAIR 2
/*
* Must receive 4 errors within 3 seconds in order to mark
* the route down.
u_int8_t rt_errors; // error count
double rt_error_time;
#define MAX_RT_ERROR 4 // errors
#define MAX_RT_ERROR_TIME 3 // seconds
*/
#define MAX_HISTORY 3
double rt_disc_latency[MAX_HISTORY];
char hist_indx;
int rt_req_last_ttl; // last ttl value used
```

```
/*
* a list of neighbors that are using this route.
*/
aodv_ncache rt_nblist;
};
/*
The Routing Table
*/
class aodv_rtable {
public:
aodv_rtable() { LIST_INIT(&rthead); }
aodv_rt_entry* head() { return rthead.lh_first; }
aodv_rt_entry* rt_add(nsaddr_t id);
void rt_delete(nsaddr_t id);
aodv_rt_entry* rt_lookup(nsaddr_t id);
private:
LIST_HEAD(aodv_rthead, aodv_rt_entry) rthead;
};
#endif /* _aodv__rtable_h__ */
```

APPENDIX B: ETX Source Code

```
#ifndef __aodv_rtable_h__
#define __aodv_rtable_h__
#include <assert.h>
#include <sys/types.h>
#include <config.h>
```

```
#include <lib/bsd-list.h>
#include <scheduler.h>
#define CURRENT_TIME Scheduler::instance().clock()
#define INFINITY2 0xff
/*
AODV Neighbor Cache Entry
*/
class AODV_Neighbor {
friend class AODV;
friend class aodv_rt_entry;
public:
AODV_Neighbor(u_int32_t a) { nb_addr = a; }
protected:
LIST_ENTRY(AODV_Neighbor) nb_link;
nsaddr_t nb_addr;
double nb_expire; // ALLOWED_HELLO_LOSS * HELLO_INTERVAL
};
LIST_HEAD(aodv_ncache, AODV_Neighbor);
/*
AODV Precursor list data structure
*/
class AODV_Precursor {
friend class AODV;
friend class aodv_rt_entry;
public:
AODV_Precursor(u_int32_t a) { pc_addr = a; }
protected:
LIST_ENTRY(AODV_Precursor) pc_link;
```

```
nsaddr_t pc_addr; // precursor address
};
LIST_HEAD(aodv_precursors, AODV_Precursor);
/*
Route Table Entry
*/
class aodv_rt_entry {
friend class aodv_rtable;
friend class AODV;
friend class LocalRepairTimer;
public:
aodv_rt_entry();
~aodv_rt_entry();
void nb_insert(nsaddr_t id);
AODV_Neighbor* nb_lookup(nsaddr_t id);
void pc_insert(nsaddr_t id);
AODV_Precursor* pc_lookup(nsaddr_t id);
void pc_delete(nsaddr_t id);
void pc_delete(void);
bool pc_empty(void);
double rt_req_timeout; // when I can send another req
u_int8_t rt_req_cnt; // number of route requests
protected:
LIST_ENTRY(aodv_rt_entry) rt_link;
nsaddr_t rt_dst;
u_int32_t rt_seqno;
/* u_int8_t t_interface; */
u_int16_t rt_hops; // hop count
```

```
int rt_last_hop_count; // last valid hop count
nsaddr_t rt_nexthop; // next hop IP address
/* list of precursors */
aodv_precursors rt_pclist;
double rt_expire; // when entry expires
u_int8_t rt_flags;
#define RTF_DOWN 0
#define RTF_UP 1
#define RTF_IN_REPAIR 2
/*
* Must receive 4 errors within 3 seconds in order to mark
* the route down.
u_int8_t rt_errors; // error count
double rt_error_time;
#define MAX_RT_ERROR 4 // errors
#define MAX_RT_ERROR_TIME 3 // seconds
*/
#define MAX_HISTORY 3
double rt_disc_latency[MAX_HISTORY];
char hist_indx;
int rt_req_last_ttl; // last ttl value used
/*
* a list of neighbors that are using this route.
*/
aodv_ncache rt_nblist;
};
/*
The Routing Table
```

```
*/
class aodv_rtable {
public:
aodv_rtable() { LIST_INIT(&rthead); }
aodv_rt_entry* head() { return rthead.lh_first; }
aodv_rt_entry* rt_add(nsaddr_t id);
void rt_delete(nsaddr_t id);
aodv_rt_entry* rt_lookup(nsaddr_t id);
private:
LIST_HEAD(aodv_rthead, aodv_rt_entry) rthead;
};
#endif /* _aodv__rtable_h__ */
#include <random.h>
#include <cmu-trace.h>
#include "aodvetx.h"
#include "aodvetx_packet.h"
#define max(a,b) ( (a) > (b) ? (a) : (b) )
#define CURRENT_TIME Scheduler::instance().clock()
#ifdef DEBUG
static int extra_route_reply = 0;
static int limit_route_request = 0;
static int route_request = 0;
#endif
/*
TCL Hooks
*/
int hdr_aodvetx::offset_;
static class AODVETXHeaderClass: public PacketHeaderClass {
```

```
public:
AODVETXHeaderClass() :
PacketHeaderClass("PacketHeader/AODVETX", sizeof(hdr_all_aodvetx)) {
bind_offset(&hdr_aodvetx::offset_);
bind(); // required for dynamic loading
}
} class_rtProtoAODVETX_hdr;
static class AODVETXclass : public TclClass {
public:
AODVETXclass() : TclClass("Agent/AODVETX") {}
TclObject* create(int argc, const char*const* argv) {
assert(argc == 5);
return (new AODVETX((nsaddr_t) Address::instance().str2addr(argv[4])));
}
} class_rtProtoAODVETX;
int
AODVETX::command(int argc, const char*const* argv) {
if(argc == 2) {
Tcl& tcl = Tcl::instance();
if(strncasecmp(argv[1], "id", 2) == 0) {
tcl.resultf("%d", index);
return TCL_OK;
}
if(strncasecmp(argv[1], "start", 2) == 0) {
btimer.handle((Event*) 0);
#ifndef AODVETX_LINK_LAYER_DETECTION
htimer.handle((Event*) 0);
ntimer.handle((Event*) 0);
```

```
#endif // LINK LAYER DETECTION
rtimer.handle((Event*) 0);
return TCL_OK;
}
}
else if(argc == 3) {
if(strcmp(argv[1], "index") == 0) {
index = atoi(argv[2]);
return TCL_OK;
}
else if(strcmp(argv[1], "log-target") == 0 || strcmp(argv[1], "tracetarget") == 0) {
logtarget = (Trace*) TclObject::lookup(argv[2]);
if(logtarget == 0)
return TCL_ERROR;
return TCL_OK;
}
else if(strcmp(argv[1], "drop-target") == 0) {
int stat = rqueue.command(argc,argv);
if (stat != TCL_OK) return stat;
return Agent::command(argc, argv);
}
else if(strcmp(argv[1], "if-queue") == 0) {
ifqueue = (PriQueue*) TclObject::lookup(argv[2]);
if(ifqueue == 0)
return TCL_ERROR;
return TCL_OK;
}
else if (strcmp(argv[1], "port-dmux") == 0) {
```

```
dmux_ = (PortClassifier *)TclObject::lookup(argv[2]);
if (dmux_ == 0) {
fprintf (stderr, "%s: %s lookup of %s failed\n", __FILE__,
argv[1], argv[2]);
return TCL_ERROR;
}
return TCL_OK;
}
}
return Agent::command(argc, argv);
}
/*
Constructor
*/
AODVETX::AODVETX(nsaddr_t id) :
Agent(PT_AODVETX), btimer(this), htimer(this), ptimer(this), wtimer(this),
mtimer(this), ntimer(this), rtimer(this), lrtimer(this), rqueue() {
index = id;
seqno = 2;
bid = 1;
LIST_INIT(&nbhead);
LIST_INIT(&bihead);
logtarget = 0;
ifqueue = 0;
}
/*
Timers
*/
```

```
void
BroadcastTimer::handle(Event*) {
agent->id_purge();
Scheduler::instance().schedule(this, &intr, BCAST_ID_SAVE);
}
void
HelloTimer::handle(Event*) {
agent->sendHello();
double interval = MinHelloInterval +
((MaxHelloInterval - MinHelloInterval) * Random::uniform());
assert(interval >= 0);
Scheduler::instance().schedule(this, &intr, interval);
}
void ETXProbeTimer::handle(Event* event) {
agent->sendETXProbe();
double interval = MIN_PROBE_INTERVAL + ((MAX_PROBE_INTERVAL
- MIN_PROBE_INTERVAL) * Random::uniform());
assert(interval >= 0);
Scheduler::instance().schedule(this, &intr, interval);
}
void ETXWindowTimer::handle(Event*) {
agent->handleProbeWindowTimer();
Scheduler::instance().schedule(this, &intr, PROBE_WINDOW);
}
void ETXManagementTimer::handle(Event*) {
agent->manageETXProbes();
double interval = MIN_PROBE_INTERVAL + ((MAX_PROBE_INTERVAL
- MIN_PROBE_INTERVAL) * Random::uniform());
```

```
Scheduler::instance().schedule(this, &intr, interval);
}
void
NeighborTimer::handle(Event*) {
agent->nb_purge();
Scheduler::instance().schedule(this, &intr, HELLO_INTERVAL);
}
void
RouteCacheTimer::handle(Event*) {
agent->rt_purge();
#define FREQUENCY 0.5 // sec
Scheduler::instance().schedule(this, &intr, FREQUENCY);
}
void
LocalRepairTimer::handle(Event* p) { // SRD: 5/4/99
aodvetx_rt_entry *rt;
struct hdr_ip *ih = HDR_IP( (Packet *)p);
/* you get here after the timeout in a local repair attempt */
/* fprintf(stderr, "%s\n", __FUNCTION__); */
rt = agent->rtable.rt_lookup(ih->daddr());
if (rt && rt->rt_flags != RTF_UP) {
// route is yet to be repaired
// I will be conservative and bring down the route
// and send route errors upstream.
/* The following assert fails, not sure why */
/* assert (rt->rt_flags == RTF_IN_REPAIR); */
//rt->rt_seqno++;
agent->rt_down(rt);
```

```
// send RERR
#ifdef DEBUG
fprintf(stderr,"Node %d: Dst - %d, failed local repair\n", agent->index, rt->rt_dst);
#endif
}
Packet::free((Packet *)p);
}
/*
Broadcast ID Management Functions
*/
void
AODVETX::id_insert(nsaddr_t id, u_int32_t bid) {
BroadcastID *b = new BroadcastID(id, bid);
assert(b);
b->expire = CURRENT_TIME + BCAST_ID_SAVE;
LIST_INSERT_HEAD(&bihead, b, link);
}
/* SRD */
bool
AODVETX::id_lookup(nsaddr_t id, u_int32_t bid) {
BroadcastID *b = bihead.lh_first;
// Search the list for a match of source and bid
for( ; b; b = b->link.le_next) {
if ((b->src == id) && (b->id == bid))
return true;
}
return false;
}
```

```
void
AODVETX::id_purge() {
BroadcastID *b = bihead.lh_first;
BroadcastID *bn;
double now = CURRENT_TIME;
for(; b; b = bn) {
bn = b->link.le_next;
if(b->expire <= now) {
LIST_REMOVE(b,link);
delete b;
}
}
}
/*
Helper Functions
*/
double
AODVETX::PerHopTime(aodvetx_rt_entry *rt) {
int num_non_zero = 0, i;
double total_latency = 0.0;
if (!rt)
return ((double) NODE_TRAVERSAL_TIME );
for (i=0; i < MAX_HISTORY; i++) {
if (rt->rt_disc_latency[i] > 0.0) {
num_non_zero++;
total_latency += rt->rt_disc_latency[i];
}
}
```

```
if (num_non_zero > 0)
return(total_latency / (double) num_non_zero);
else
return((double) NODE_TRAVERSAL_TIME);
}
/*
Link Failure Management Functions
*/
static void
aodvetx_rt_failed_callback(Packet *p, void *arg) {
((AODVETX*) arg)->rt_ll_failed(p);
}
/*
* This routine is invoked when the link-layer reports a route failed.
*/
void
AODVETX::rt_ll_failed(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
aodvetx_rt_entry *rt;
nsaddr_t broken_nbr = ch->next_hop_;
#ifndef AODVETX_LINK_LAYER_DETECTION
drop(p, DROP_RTR_MAC_CALLBACK);
#else
/*
* Non-data packets and Broadcast Packets can be dropped.
*/
if(! DATA_PACKET(ch->ptype()) ||
```

```
(u_int32_t) ih->daddr() == IP_BROADCAST) {
drop(p, DROP_RTR_MAC_CALLBACK);
return;
}
log_link_broke(p);
if((rt = rtable.rt_lookup(ih->daddr())) == 0) {
drop(p, DROP_RTR_MAC_CALLBACK);
return;
}
log_link_del(ch->next_hop_);
#ifdef AODVETX_LOCAL_REPAIR
/* if the broken link is closer to the dest than source,
attempt a local repair. Otherwise, bring down the route. */
if (ch->num_forwards() > rt->rt_hops) {
local_rt_repair(rt, p); // local repair
// retrieve all the packets in the ifq using this link,
// queue the packets for which local repair is done,
return;
}
else
#endif // LOCAL REPAIR
{
drop(p, DROP_RTR_MAC_CALLBACK);
// Do the same thing for other packets in the interface queue using the
// broken link -Mahesh
while((p = ifqueue->filter(broken_nbr))) {
drop(p, DROP_RTR_MAC_CALLBACK);
}
```

```
nb_delete(broken_nbr);
}
#endif // LINK LAYER DETECTION
}
void
AODVETX::handle_link_failure(nsaddr_t id) {
aodvetx_rt_entry *rt, *rtn;
Packet *rerr = Packet::alloc();
struct hdr_aodvetx_error *re = HDR_AODVETX_ERROR(rerr);
re->DestCount = 0;
for(rt = rtable.head(); rt; rt = rtn) { // for each rt entry
rtn = rt->rt_link.le_next;
if ((rt->rt_hops != INFINITY2) && (rt->rt_nexthop == id) ) {
assert (rt->rt_flags == RTF_UP);
assert((rt->rt_seqno%2) == 0);
rt->rt_seqno++;
re->unreachable_dst[re->DestCount] = rt->rt_dst;
re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
#ifdef DEBUG
fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\n", __FUNCTION__, CURRENT_TIME,
index, re->unreachable_dst[re->DestCount],
re->unreachable_dst_seqno[re->DestCount], rt->rt_nexthop);
#endif // DEBUG
re->DestCount += 1;
rt_down(rt);
}
// remove the lost neighbor from all the precursor lists
rt->pc_delete(id);
```

```
}
if (re->DestCount > 0) {
#ifdef DEBUG
fprintf(stderr, "%s(%f): %d\tsending RERR...\n", __FUNCTION__, CURRENT_TIME, index);
#endif // DEBUG
sendError(rerr, false);
}
else {
Packet::free(rerr);
}
}
void
AODVETX::local_rt_repair(aodvetx_rt_entry *rt, Packet *p) {
#ifdef DEBUG
fprintf(stderr,"%s: Dst - %d\n", __FUNCTION__, rt->rt_dst);
#endif
// Buffer the packet
rqueue.enque(p);
// mark the route as under repair
rt->rt_flags = RTF_IN_REPAIR;
sendRequest(rt->rt_dst);
// set up a timer interrupt
Scheduler::instance().schedule(&lrtimer, p->copy(), rt->rt_req_timeout);
}
void AODVETX::rt_update(aodvetx_rt_entry *rt, u_int32_t seqnum, u_int16_t metric,
nsaddr_t nexthop, double expire_time) {
rt->rt_seqno = seqnum;
rt->rt_hops = metric;
```

```
rt->rt_flags = RTF_UP;
rt->rt_nexthop = nexthop;
rt->rt_expire = expire_time;
}
void AODVETX::rt_update(aodvetx_rt_entry *rt, u_int32_t seqnum,
u_int16_t hop_count, double metric, nsaddr_t nexthop,
double expire_time) {
rt->rt_seqno = seqnum;
rt->rt_hops = hop_count;
rt->rt_etx = metric;
rt->rt_flags = RTF_UP;
rt->rt_nexthop = nexthop;
rt->rt_expire = expire_time;
}
void AODVETX::rt_down(aodvetx_rt_entry *rt) {
/*
* Make sure that you don't "down" a route more than once.
*/
if (rt->rt_flags == RTF_DOWN) {
return;
}
// assert (rt->rt_seqno%2); // is the seqno odd?
rt->rt_last_hop_count = rt->rt_hops;
rt->rt_last_etx = rt->rt_etx;
rt->rt_hops = INFINITY2;
rt->rt_etx = INFINITY2;
rt->rt_flags = RTF_DOWN;
rt->rt_nexthop = 0;
```

```
rt->rt_expire = 0;
}
/*
Route Handling Functions
*/
void
AODVETX::rt_resolve(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
aodvetx_rt_entry *rt;
/*
* Set the transmit failure callback. That
* won't change.
*/
ch->xmit_failure_ = aodvetx_rt_failed_callback;
ch->xmit_failure_data_ = (void*) this;
rt = rtable.rt_lookup(ih->daddr());
if(rt == 0) {
rt = rtable.rt_add(ih->daddr());
}
/*
* If the route is up, forward the packet
*/
if(rt->rt_flags == RTF_UP) {
assert(rt->rt_hops != INFINITY2);
forward(rt, p, NO_DELAY);
}
/*
```

```
* if I am the source of the packet, then do a Route Request.
*/
else if(ih->saddr() == index) {
rqueue.enque(p);
sendRequest(rt->rt_dst);
}
/*
* A local repair is in progress. Buffer the packet.
*/
else if (rt->rt_flags == RTF_IN_REPAIR) {
rqueue.enque(p);
}
/*
* I am trying to forward a packet for someone else to which
* I don't have a route.
*/
else {
Packet *rerr = Packet::alloc();
struct hdr_aodvetx_error *re = HDR_AODVETX_ERROR(rerr);
/*
* For now, drop the packet and send error upstream.
* Now the route errors are broadcast to upstream
* neighbors - Mahesh 09/11/99
*/
assert (rt->rt_flags == RTF_DOWN);
re->DestCount = 0;
re->unreachable_dst[re->DestCount] = rt->rt_dst;
re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
```

```
re->DestCount += 1;
#ifdef DEBUG
fprintf(stderr, "%s: sending RERR...\n", __FUNCTION__);
#endif
sendError(rerr, false);
drop(p, DROP_RTR_NO_ROUTE);
}
}
void
AODVETX::rt_purge() {
aodvetx_rt_entry *rt, *rtn;
double now = CURRENT_TIME;
double delay = 0.0;
Packet *p;
for(rt = rtable.head(); rt; rt = rtn) { // for each rt entry
rtn = rt->rt_link.le_next;
if ((rt->rt_flags == RTF_UP) && (rt->rt_expire < now)) {
// if a valid route has expired, purge all packets from
// send buffer and invalidate the route.
assert(rt->rt_hops != INFINITY2);
while((p = rqueue.deque(rt->rt_dst))) {
#ifdef DEBUG
fprintf(stderr, "%s: calling drop()\n",
__FUNCTION__);
#endif // DEBUG
drop(p, DROP_RTR_NO_ROUTE);
}
rt->rt_seqno++;
```

```
assert (rt->rt_seqno%2);
rt_down(rt);
}
else if (rt->rt_flags == RTF_UP) {
// If the route is not expired,
// and there are packets in the sendbuffer waiting,
// forward them. This should not be needed, but this extra
// check does no harm.
assert(rt->rt_hops != INFINITY2);
while((p = rqueue.deque(rt->rt_dst))) {
forward (rt, p, delay);
delay += ARP_DELAY;
}
}
else if (rqueue.find(rt->rt_dst))
// If the route is down and if there is a packet for this destination waiting in the sendbuffer, then
// sendout route request. sendRequest will check whether it is time to really send out request
// or not. This may not be crucial to do it here, as each generated packet will do a sendRequest
anyway.
sendRequest(rt->rt_dst);
}
}
/*
Packet Reception Routines
*/
void
AODVETX::recv(Packet *p, Handler*) {
struct hdr_cmn *ch = HDR_CMN(p);
```

```
struct hdr_ip *ih = HDR_IP(p);
assert(initialized());
direction_ in hdr_cmn is used instead. see packet.h for details.
if(ch->ptype() == PT_AODVETX) {
ih->ttl_ -= 1;
recvAODVETX(p);
return;
}
/*
* Must be a packet I'm originating...
*/
if((ih->saddr() == index) && (ch->num_forwards() == 0)) {
/*
* Add the IP Header.
* TCP adds the IP header too, so to avoid setting it twice, we check if
* this packet is not a TCP or ACK segment.
*/
if (ch->ptype() != PT_TCP && ch->ptype() != PT_ACK) {
ch->size() += IP_HDR_LEN;
}
// Added by Parag Dadhania && John Novatnack to handle broadcasting
if ( (u_int32_t)ih->daddr() != IP_BROADCAST) {
ih->ttl_ = NETWORK_DIAMETER;
}
}
/*
* I received a packet that I sent. Probably
* a routing loop.
```

```
*/
else if(ih->saddr() == index) {
drop(p, DROP_RTR_ROUTE_LOOP);
return;
}
/*
* Packet I'm forwarding...
*/
else {
/*
* Check the TTL. If it is zero, then discard.
*/
if(--ih->ttl_ == 0) {
drop(p, DROP_RTR_TTL);
return;
}
}
// Added by Parag Dadhania && John Novatnack to handle broadcasting
if ( (u_int32_t)ih->daddr() != IP_BROADCAST)
rt_resolve(p);
else
forward((aodvetx_rt_entry*) 0, p, NO_DELAY);
}
void
AODVETX::recvAODVETX(Packet *p) {
struct hdr_aodvetx *ah = HDR_AODVETX(p);
assert(HDR_IP (p)->sport() == RT_PORT);
assert(HDR_IP (p)->dport() == RT_PORT);
```

```
/*
* Incoming Packets.
*/
switch(ah->ah_type) {
case AODVETXTYPE_RREQ:
recvRequest(p);
break;
case AODVETXTYPE_RREP:
recvReply(p);
break;
case AODVETXTYPE_RERR:
recvError(p);
break;
case AODVETXTYPE_HELLO:
recvHello(p);
break;
case AODVETXTYPE_PROBE:
receiveETXProbe(p);
break;
default:
fprintf(stderr, "Invalid AODVETX type (%x)\n", ah->ah_type);
exit(1);
}
}
void AODVETX::recvRequest(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_request *rq = HDR_AODVETX_REQUEST(p);
```

```
aodvetx_rt_entry *rt;
/*
* Drop if:
* - I'm the source
* - I recently heard this request.
*/
if (rq->rq_src == index) {
#ifdef DEBUG
fprintf(stderr, "%s: got my own REQUEST\n", __FUNCTION__);
#endif // DEBUG
Packet::free(p);
return;
}
if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {
#ifdef DEBUG
fprintf(stderr, "%s: discarding request\n", __FUNCTION__);
#endif // DEBUG
Packet::free(p);
return;
}
/*
* Cache the broadcast ID
*/
id_insert(rq->rq_src, rq->rq_bcast_id);
/*
* We are either going to forward the REQUEST or generate a
* REPLY. Before we do anything, we make sure that the REVERSE
* route is in the route table.
```

```
*/
aodvetx_rt_entry *rt0; // rt0 is the reverse route
rt0 = rtable.rt_lookup(rq->rq_src);
if (rt0 == 0) { /* if not in the route table */
// create an entry for the reverse route.
rt0 = rtable.rt_add(rq->rq_src);
rt0->rt_etx += calculateETX(ch->prev_hop_);
}
rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE));
if ((rq->rq_src_seqno > rt0->rt_seqno) II ((rq->rq_src_seqno
// == rt0->rt_seqno) && (rq->rq_hop_count < rt0->rt_hops))) {
== rt0->rt_seqno) && ((rq->rq_etx + calculateETX(ch->prev_hop_))
< rt0->rt_etx))) {
// If we have a fresher seq no. or lesser #hops for the
// same seq no., update the rt entry. Else don't bother.
rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, rq->rq_etx
+ calculateETX(ch->prev_hop_), ih->saddr(),
max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE)));
if (rt0->rt_req_timeout > 0.0) {
// Reset the soft state and
// Set expiry time to CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT
// This is because route is used in the forward direction,
// but only sources get benefited by this change
rt0->rt_req_cnt = 0;
rt0->rt_req_timeout = 0.0;
rt0->rt_req_last_ttl = rq->rq_hop_count;
rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
}
```

```
/* Find out whether any buffered packet can benefit from the
* reverse route.
* May need some change in the following code - Mahesh 09/11/99
*/
assert (rt0->rt_flags == RTF_UP);
Packet *buffered_pkt;
while ((buffered_pkt = rqueue.deque(rt0->rt_dst))) {
if (rt0 && (rt0->rt_flags == RTF_UP)) {
assert(rt0->rt_hops != INFINITY2);
forward(rt0, buffered_pkt, NO_DELAY);
}
}
}
// End for putting reverse route in rt table
/*
* We have taken care of the reverse route stuff.
* Now see whether we can send a route reply.
*/
rt = rtable.rt_lookup(rq->rq_dst);
// First check if I am the destination ..
if (rq->rq_dst == index) {
#ifdef DEBUG
fprintf(stderr, "%d - %s: destination sending reply\n", index,
__FUNCTION__);
#endif // DEBUG
// Just to be safe, I use the max. Somebody may have incremented the dst seqno.
seqno = max(seqno, rq->rq_dst_seqno) + 1;
if (seqno % 2)
```

```
seqno++;
sendReply(rq->rq_src, // IP Destination
1, // Hop Count
0, // ETX FIXME
index, // Dest IP Address
seqno, // Dest Sequence Num
MY_ROUTE_TIMEOUT, // Lifetime
rq->rq_timestamp); // timestamp
Packet::free(p);
}
// I am not the destination, but I may have a fresh enough route.
else if (rt && (rt->rt_hops != INFINITY2) && (rt->rt_seqno
>= rq->rq_dst_seqno)) {
//assert (rt->rt_flags == RTF_UP);
assert(rq->rq_dst == rt->rt_dst);
//assert ((rt->rt_seqno%2) == 0); // is the seqno even?
sendReply(rq->rq_src, // IP Destination
rt->rt_hops + 1, // Hop Count
rt->rt_etx, // ETX FIXME
rq->rq_dst, // Destination IP
rt->rt_seqno, // Destination Sequence Number
(u_int32_t) (rt->rt_expire - CURRENT_TIME), // Lifetime
rq->rq_timestamp); // Time stamp
// Insert nexthops to RREQ source and RREQ destination in the
// precursor lists of destination and source respectively
rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
rt0->pc_insert(rt->rt_nexthop); // nexthop to RREQ destination
#ifdef RREQ_GRAT_RREP
```

```
sendReply(rq->rq_dst,
rq->rq_hop_count, // Hop Count
INFINITY2, // ETX
rq->rq_src,
rq->rq_src_seqno,
(u_int32_t) (rt->rt_expire - CURRENT_TIME),
// rt->rt_expire - CURRENT_TIME,
rq->rq_timestamp);
#endif
// TODO: send grat RREP to dst if G flag set in RREQ using rq->rq_src_seqno, rq-
>rq_hop_counT
// DONE: Included gratuitous replies to be sent as per IETF aodvetx draft specification. As of now,
G
flag has not been dynamically used and is always set or reset in aodvetx-packet.h --- Anant
Utgikar, 09/16/02.
Packet::free(p);
}
/*
* Can't reply. So forward the Route Request
*/
else {
ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
rq->rq_hop_count += 1;
rq->rq_etx += calculateETX(ch->prev_hop_);
// Maximum sequence number seen en route
if (rt) {
rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);
```

```
}
forward((aodvetx_rt_entry*) 0, p, DELAY);
}
}
void AODVETX::recvReply(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_reply *rp = HDR_AODVETX_REPLY(p);
aodvetx_rt_entry *rt;
char suppress_reply = 0;
double delay = 0.0;
#ifdef DEBUG
fprintf(stderr, "%d - %s: received a REPLY\n", index, __FUNCTION__);
#endif // DEBUG
/*
* Got a reply. So reset the "soft state" maintained for
* route requests in the request table. We don't really have
* have a separate request table. It is just a part of the
* routing table itself.
*/
// Note that rp_dst is the dest of the data packets, not the
// the dest of the reply, which is the src of the data packets.
rt = rtable.rt_lookup(rp->rp_dst);
/*
* If I don't have a rt entry to this host... adding
*/
if (rt == 0) {
rt = rtable.rt_add(rp->rp_dst);
```

```
}
/*
* Add a forward route table entry... here I am following
* Perkins-Royer AODVETX paper almost literally - SRD 5/99
*/
if ((rt->rt_seqno < rp->rp_dst_seqno) || // newer route
// ((rt->rt_seqno == rp->rp_dst_seqno) && (rt->rt_hops
// > rp->rp_hop_count))) { // shorter or better route
((rt->rt_seqno == rp->rp_dst_seqno) && (rt->rt_etx > rp->rp_etx))) { // Better
route
// Update the rt entry
rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count, rp->rp_etx,
rp->rp_src, CURRENT_TIME + rp->rp_lifetime);
// reset the soft state
rt->rt_req_cnt = 0;
rt->rt_req_timeout = 0.0;
rt->rt_req_last_ttl = rp->rp_hop_count;
if (ih->daddr() == index) { // If I am the original source
// Update the route discovery latency statistics
// rp->rp_timestamp is the time of request origination
rt->rt_disc_latency[(unsigned char) rt->hist_indx] = (CURRENT_TIME
- rp->rp_timestamp) / (double) rp->rp_hop_count;
// increment indx for next time
rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
}
/*
* Send all packets queued in the sendbuffer destined for
* this destination.
```

```
* XXX - observe the "second" use of p.
*/
Packet *buf_pkt;
while ((buf_pkt = rqueue.deque(rt->rt_dst))) {
if (rt->rt_hops != INFINITY2) {
assert (rt->rt_flags == RTF_UP);
// Delay them a little to help ARP. Otherwise AR may drop packets. -SRD 5/23/99
forward(rt, buf_pkt, delay);
delay += ARP_DELAY;
}
}
} else {
suppress_reply = 1;
}
/*
* If reply is for me, discard it.
*/
if (ih->daddr() == index || suppress_reply) {
Packet::free(p);
}
/*
* Otherwise, forward the Route Reply.
*/
else {
// Find the rt entry
aodvetx_rt_entry *rt0 = rtable.rt_lookup(ih->daddr());
// If the rt is up, forward
if (rt0 && (rt0->rt_hops != INFINITY2)) {
```

```
assert (rt0->rt_flags == RTF_UP);
rp->rp_hop_count += 1;
rp->rp_etx += calculateETX(ch->prev_hop_);
rp->rp_src = index;
forward(rt0, p, NO_DELAY);
// Insert the nexthop towards the RREQ source to the precursor list of the RREQ destination
rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
} else {
// I don't know how to forward .. drop the reply.
#ifdef DEBUG
fprintf(stderr, "%s: dropping Route Reply\n", __FUNCTION__);
#endif // DEBUG
drop(p, DROP_RTR_NO_ROUTE);
}
}
}
void
AODVETX::recvError(Packet *p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_error *re = HDR_AODVETX_ERROR(p);
aodvetx_rt_entry *rt;
u_int8_t i;
Packet *rerr = Packet::alloc();
struct hdr_aodvetx_error *nre = HDR_AODVETX_ERROR(rerr);
nre->DestCount = 0;
for (i=0; i<re->DestCount; i++) {
// For each unreachable destination
rt = rtable.rt_lookup(re->unreachable_dst[i]);
```

```
if ( rt && (rt->rt_hops != INFINITY2) &&
(rt->rt_nexthop == ih->saddr()) &&
(rt->rt_seqno <= re->unreachable_dst_seqno[i]) ) {
assert(rt->rt_flags == RTF_UP);
assert((rt->rt_seqno%2) == 0); // is the seqno even?
#ifdef DEBUG
fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\t(%d\t%u\t%d)\n", __FUNCTION__,
CURRENT_TIME,
index, rt->rt_dst, rt->rt_seqno, rt->rt_nexthop,
re->unreachable_dst[i],re->unreachable_dst_seqno[i],
ih->saddr());
#endif // DEBUG
rt->rt_seqno = re->unreachable_dst_seqno[i];
rt_down(rt);
// Not sure whether this is the right thing to do
Packet *pkt;
while((pkt = ifqueue->filter(ih->saddr()))) {
drop(pkt, DROP_RTR_MAC_CALLBACK);
}
// if precursor list non-empty add to RERR and delete the precursor list
if (!rt->pc_empty()) {
nre->unreachable_dst[nre->DestCount] = rt->rt_dst;
nre->unreachable_dst_seqno[nre->DestCount] = rt->rt_seqno;
nre->DestCount += 1;
rt->pc_delete();
}
}
}
```

```
if (nre->DestCount > 0) {
#ifdef DEBUG
fprintf(stderr, "%s(%f): %d\t sending RERR...\n", __FUNCTION__, CURRENT_TIME, index);
#endif // DEBUG
sendError(rerr);
}
else {
Packet::free(rerr);
}
Packet::free(p);
}
/*
Packet Transmission Routines
*/
void
AODVETX::forward(aodvetx_rt_entry *rt, Packet *p, double delay) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
if(ih->ttl_ == 0) {
#ifdef DEBUG
fprintf(stderr, "%s: calling drop()\n", __PRETTY_FUNCTION__);
#endif // DEBUG
drop(p, DROP_RTR_TTL);
return;
}
if (ch->ptype() != PT_AODVETX && ch->direction() == hdr_cmn::UP &&
((u_int32_t)ih->daddr() == IP_BROADCAST)
|| (ih->daddr() == here_.addr_)) {
```

```
dmux_->recv(p,0);
return;
}
if (rt) {
assert(rt->rt_flags == RTF_UP);
rt->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
ch->next_hop_ = rt->rt_nexthop;
ch->addr_type() = NS_AF_INET;
ch->direction() = hdr_cmn::DOWN; //important: change the packet's direction
}
else { // if it is a broadcast packet
// assert(ch->ptype() == PT_AODVETX); // maybe a diff pkt type like gaf
assert(ih->daddr() == (nsaddr_t) IP_BROADCAST);
ch->addr_type() = NS_AF_NONE;
ch->direction() = hdr_cmn::DOWN; //important: change the packet's direction
}
if (ih->daddr() == (nsaddr_t) IP_BROADCAST) {
// If it is a broadcast packet
assert(rt == 0);
if (ch->ptype() == PT_AODVETX) {
/*
* Jitter the sending of AODVETX broadcast packets by 10ms
*/
Scheduler::instance().schedule(target_, p,
0.01 * Random::uniform());
} else {
Scheduler::instance().schedule(target_, p, 0.); // No jitter
}
```

```
}
else { // Not a broadcast packet
if(delay > 0.0) {
Scheduler::instance().schedule(target_, p, delay);
}
else {
// Not a broadcast packet, no delay, send immediately
Scheduler::instance().schedule(target_, p, 0.);
}
}
}
void AODVETX::sendRequest(nsaddr_t dst) {
// Allocate a RREQ packet
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_request *rq = HDR_AODVETX_REQUEST(p);
aodvetx_rt_entry *rt = rtable.rt_lookup(dst);
assert(rt);
/*
* Rate limit sending of Route Requests. We are very conservative
* about sending out route requests.
*/
if (rt->rt_flags == RTF_UP) {
assert(rt->rt_hops != INFINITY2);
Packet::free((Packet *) p);
return;
}
```

```
if (rt->rt_req_timeout > CURRENT_TIME) {
Packet::free((Packet *) p);
return;
}
// rt_req_cnt is the no. of times we did network-wide broadcast RREQ_RETRIES is the maximum
//number we will allow before going to a long timeout.
if (rt->rt_req_cnt > RREQ_RETRIES) {
rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
rt->rt_req_cnt = 0;
Packet *buf_pkt;
while ((buf_pkt = rqueue.deque(rt->rt_dst))) {
drop(buf_pkt, DROP_RTR_NO_ROUTE);
}
Packet::free((Packet *) p);
return;
}
#ifdef DEBUG
fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d\n",
++route_request, index, rt->rt_dst);
#endif // DEBUG
// Determine the TTL to be used this time.
// Dynamic TTL evaluation - SRD
rt->rt_req_last_ttl = max(rt->rt_req_last_ttl, rt->rt_last_hop_count);
if (0 == rt->rt_req_last_ttl) {
// first time query broadcast
ih->ttl_ = TTL_START;
} else {
// Expanding ring search.
```

```
if (rt->rt_req_last_ttl < TTL_THRESHOLD)
ih->ttl_ = rt->rt_req_last_ttl + TTL_INCREMENT;
else {
// network-wide broadcast
ih->ttl_ = NETWORK_DIAMETER;
rt->rt_req_cnt += 1;
}
}
// remember the TTL used for the next time
rt->rt_req_last_ttl = ih->ttl_;
// PerHopTime is the roundtrip time per hop for route requests.
// The factor 2.0 is just to be safe .. SRD 5/22/99
// Also note that we are making timeouts to be larger if we have
// done network wide broadcast before.
rt->rt_req_timeout = 2.0 * (double) ih->ttl_ * PerHopTime(rt);
if (rt->rt_req_cnt > 0)
rt->rt_req_timeout *= rt->rt_req_cnt;
rt->rt_req_timeout += CURRENT_TIME;
// Don't let the timeout to be too large, however .. SRD 6/8/99
if (rt->rt_req_timeout > CURRENT_TIME + MAX_RREQ_TIMEOUT)
rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
rt->rt_expire = 0;
#ifdef DEBUG
fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d, tout %f ms\n",
++route_request, index, rt->rt_dst, rt->rt_req_timeout
- CURRENT_TIME);
#endif // DEBUG
// Fill out the RREQ packet
```

```
// ch->uid() = 0;
ch->ptype() = PT_AODVETX;
ch->size() = IP_HDR_LEN + rq->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_NONE;
ch->prev_hop_ = index; // AODVETX hack
ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
// Fill up some more fields.
rq->rq_type = AODVETXTYPE_RREQ;
rq->rq_hop_count = 1;
rq->rq_etx = 0;
rq->rq_bcast_id = bid++;
rq->rq_dst = dst;
rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
rq->rq_src = index;
seqno += 2;
assert ((seqno%2) == 0);
rq->rq_src_seqno = seqno;
rq->rq_timestamp = CURRENT_TIME;
Scheduler::instance().schedule(target_, p, 0.);
}
void
AODVETX::sendReply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst,
u_int32_t rpseq, u_int32_t lifetime, double timestamp) {
```

```
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_reply *rp = HDR_AODVETX_REPLY(p);
aodvetx_rt_entry *rt = rtable.rt_lookup(ipdst);
#ifdef DEBUG
fprintf(stderr, "sending Reply from %d at %.2f\n", index, Scheduler::instance().clock());
#endif // DEBUG
assert(rt);
rp->rp_type = AODVETXTYPE_RREP;
//rp->rp_flags = 0x00;
rp->rp_hop_count = hop_count;
rp->rp_dst = rpdst;
rp->rp_dst_seqno = rpseq;
rp->rp_src = index;
rp->rp_lifetime = lifetime;
rp->rp_timestamp = timestamp;
// ch->uid() = 0;
ch->ptype() = PT_AODVETX;
ch->size() = IP_HDR_LEN + rp->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_INET;
ch->next_hop_ = rt->rt_nexthop;
ch->prev_hop_ = index; // AODVETX hack
ch->direction() = hdr_cmn::DOWN;
ih->saddr() = index;
ih->daddr() = ipdst;
```

```
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
ih->ttl_ = NETWORK_DIAMETER;
Scheduler::instance().schedule(target_, p, 0.);
}
void AODVETX::sendReply(nsaddr_t ipdst, u_int32_t hop_count, double etx,
nsaddr_t rpdst, u_int32_t rpseq, u_int32_t lifetime, double timestamp) {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_reply *rp = HDR_AODVETX_REPLY(p);
aodvetx_rt_entry *rt = rtable.rt_lookup(ipdst);
#ifdef DEBUG
fprintf(stderr, "sending Reply from %d at %.2f\n", index,
Scheduler::instance().clock());
#endif // DEBUG
assert(rt);
rp->rp_type = AODVETXTYPE_RREP;
//rp->rp_flags = 0x00;
rp->rp_hop_count = hop_count;
rp->rp_etx = etx;
rp->rp_dst = rpdst;
rp->rp_dst_seqno = rpseq;
rp->rp_src = index;
rp->rp_lifetime = lifetime;
rp->rp_timestamp = timestamp;
// ch->uid() = 0;
ch->ptype() = PT_AODVETX;
```

```
ch->size() = IP_HDR_LEN + rp->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_INET;
ch->next_hop_ = rt->rt_nexthop;
ch->prev_hop_ = index; // AODVETX hack
ch->direction() = hdr_cmn::DOWN;
ih->saddr() = index;
ih->daddr() = ipdst;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
ih->ttl_ = NETWORK_DIAMETER;
Scheduler::instance().schedule(target_, p, 0.);
}
void
AODVETX::sendError(Packet *p, bool jitter) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_error *re = HDR_AODVETX_ERROR(p);
#ifdef ERROR
fprintf(stderr, "sending Error from %d at %.2f\n", index, Scheduler::instance().clock());
#endif // DEBUG
re->re_type = AODVETXTYPE_RERR;
//re->reserved[0] = 0x00; re->reserved[1] = 0x00;
// DestCount and list of unreachable destinations are already filled
// ch->uid() = 0;
ch->ptype() = PT_AODVETX;
ch->size() = IP_HDR_LEN + re->size();
```

```
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_NONE;
ch->next_hop_ = 0;
ch->prev_hop_ = index; // AODVETX hack
ch->direction() = hdr_cmn::DOWN; //important: change the packet's direction
ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
ih->ttl_ = 1;
// Do we need any jitter? Yes
if (jitter)
Scheduler::instance().schedule(target_, p, 0.01*Random::uniform());
else
Scheduler::instance().schedule(target_, p, 0.0);
}
/*
Neighbor Management Functions
*/
void
AODVETX::sendHello() {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_reply *rh = HDR_AODVETX_REPLY(p);
#ifdef DEBUG
fprintf(stderr, "sending Hello from %d at %.2f\n", index, Scheduler::instance().clock());
```

```
#endif // DEBUG
rh->rp_type = AODVETXTYPE_HELLO;
//rh->rp_flags = 0x00;
rh->rp_hop_count = 1;
rh->rp_dst = index;
rh->rp_dst_seqno = seqno;
rh->rp_lifetime = (1 + ALLOWED_HELLO_LOSS) * HELLO_INTERVAL;
// ch->uid() = 0;
ch->ptype() = PT_AODVETX;
ch->size() = IP_HDR_LEN + rh->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_NONE;
ch->prev_hop_ = index; // AODVETX hack
ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
ih->ttl_ = 1;
Scheduler::instance().schedule(target_, p, 0.0);
}
void
AODVETX::recvHello(Packet *p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_reply *rp = HDR_AODVETX_REPLY(p);
AODVETX_Neighbor *nb;
nb = nb_lookup(rp->rp_dst);
if(nb == 0) {
```

```
nb_insert(rp->rp_dst);
}
else {
nb->nb_expire = CURRENT_TIME +
(1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
}
Packet::free(p);
}
void
AODVETX::nb_insert(nsaddr_t id) {
AODVETX_Neighbor *nb = new AODVETX_Neighbor(id);
assert(nb);
nb->nb_expire = CURRENT_TIME +
(1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
LIST_INSERT_HEAD(&nbhead, nb, nb_link);
seqno += 2; // set of neighbors changed
assert ((seqno%2) == 0);
}
AODVETX_Neighbor*
AODVETX::nb_lookup(nsaddr_t id) {
AODVETX_Neighbor *nb = nbhead.lh_first;
for(; nb; nb = nb->nb_link.le_next) {
if(nb->nb_addr == id) break;
}
return nb;
}
/*
* Called when we receive *explicit* notification that a Neighbor
```

```
* is no longer reachable.
*/
void
AODVETX::nb_delete(nsaddr_t id) {
AODVETX_Neighbor *nb = nbhead.lh_first;
log_link_del(id);
seqno += 2; // Set of neighbors changed
assert ((seqno%2) == 0);
for(; nb; nb = nb->nb_link.le_next) {
if(nb->nb_addr == id) {
LIST_REMOVE(nb,nb_link);
delete nb;
break;
}
}
handle_link_failure(id);
}
/*
* Purges all timed-out Neighbor Entries - runs every
* HELLO_INTERVAL * 1.5 seconds.
*/
void
AODVETX::nb_purge() {
AODVETX_Neighbor *nb = nbhead.lh_first;
AODVETX_Neighbor *nbn;
double now = CURRENT_TIME;
for(; nb; nb = nbn) {
nbn = nb->nb_link.le_next;
```

```
if(nb->nb_expire <= now) {
nb_delete(nb->nb_addr);
}
}
}
// ETX Functions
void AODVETX::sendETXProbe() {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_probe *rb = HDR_AODVETX_PROBE(p);
#ifdef DEBUG
fprintf(stderr, "[%d] sending ETXProbe at %.2f\n", index, CURRENT_TIME);
#endif // DEBUG
rb->rb_type = AODVETXTYPE_PROBE;
rb->rb_src = index;
rb->rb_neighbour_count = probeNeighbours_.size();
// copy neighbour list
int i = 0;
for (map<nsaddr_t, u_int32_t>::const_iterator
iter(probeNeighbours_.begin()); iter != probeNeighbours_.end(); ++iter, i++) {
rb->rb_neighbours[i] = iter->first;
rb->rb_probes[i] = iter->second;
#ifdef DEBUG
fprintf(stderr, " %d (%d)\n", iter->first, iter->second);
#endif // DEBUG
}
rb->rb_timestamp = CURRENT_TIME;
```

```
ch->ptype() = PT_AODVETX;
ch->size() = IP_HDR_LEN + rb->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_NONE;
ch->prev_hop_ = index; // AODVETX hack
ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
ih->ttl_ = 1;
Scheduler::instance().schedule(target_, p, 0.0);
#ifdef DEBUG
fprintf(stderr, "[%d] done sending ETXProbe\n", index);
#endif // DEBUG
}
void AODVETX::receiveETXProbe(Packet* p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodvetx_probe *rb = HDR_AODVETX_PROBE(p);
double now = CURRENT_TIME;
#ifdef DEBUG
fprintf(stderr, "[%d]: receiving Probe from %d at %.2f\n", index,
ih->src_.addr_, CURRENT_TIME);
#endif // DEBUG
if (probeNeighbours_.count(rb->rb_src) > 0) {
probeNeighbours_[rb->rb_src] = probeNeighbours_[rb->rb_src] + 1;
} else {
probeNeighbours_[rb->rb_src] = 1;
```

```
}
probePackets_[rb->rb_src].push_back(now);
for (int i = 0; i < rb->rb_neighbour_count; i++) {
if (rb->rb_neighbours[i] == index) {
updateForwardDeliveryRatio(rb->rb_src, rb->rb_probes[i]);
break;
}
}
Packet::free(p);
#ifdef DEBUG
fprintf(stderr, "[%d]: done receiving Probe from %d\n", index, ih->src_.addr_);
#endif // DEBUG
}
void AODVETX::handleProbeWindowTimer() {
#ifdef DEBUG
fprintf(stderr, "[%d] handling EXTProbe window\n", index);
#endif // DEBUG
// Calculate delivery ratio
for (map<nsaddr_t, u_int32_t>::const_iterator
iter(probeNeighbours_.begin()); iter != probeNeighbours_.end(); ++iter) {
updateReverseDeliveryRatio(iter->first);
}
#ifdef DEBUG
fprintf(stderr, "[%d] done handling EXTProbe window\n", index);
#endif // DEBUG
}
void AODVETX::manageETXProbes() {
#ifdef DEBUG
```

```
fprintf(stderr, "[%d] managing EXTProbes\n", index);
#endif // DEBUG
// Remove old probes packets
removeOldProbes();
#ifdef DEBUG
fprintf(stderr, "[%d] done managing EXTProbes\n", index);
#endif // DEBUG
}
void AODVETX::removeOldProbes() {
// Clear old probe packet counts
double now = CURRENT_TIME;
for (map<nsaddr_t, list<double> >::iterator iter(probePackets_.begin()); iter
!= probePackets_.end(); ++iter) {
for (list<double>::iterator plt(iter->second.begin()); plt
!= iter->second.end(); plt++) {
if ((now - *plt) >= PROBE_WINDOW) {
#ifdef DEBUG
fprintf(stderr, "[%d] dropping old EXTProbe %.2f at %.2f\n",
index, *plt, now);
#endif // DEBUG
iter->second.erase(plt, iter->second.end());
probeNeighbours_[iter->first] = iter->second.size();
break;
}
}
}
}
```

```
void AODVETX::updateReverseDeliveryRatio(nsaddr_t neighbour) {
double reverseDeliveryRatio = probeNeighbours_[neighbour] / PROBE_WINDOW;
reverseDeliveryRatios_[neighbour] = reverseDeliveryRatio;
}
void AODVETX::updateForwardDeliveryRatio(nsaddr_t neighbour,
u_int32_t probes_count) {
double forwardDeliveryRatio = probes_count / PROBE_WINDOW;
forwardDeliveryRatios_[neighbour] = forwardDeliveryRatio;
}
double AODVETX::calculateETX(nsaddr_t destination) {
double forwardDeliveryRatio = 0;
double reverseDeliveryRatio = 0;
if (forwardDeliveryRatios_.count(destination) > 0) {
forwardDeliveryRatio = forwardDeliveryRatios_[destination];
}
if (reverseDeliveryRatios_.count(destination) > 0) {
reverseDeliveryRatio = reverseDeliveryRatios_[destination];
}
#ifdef DEBUG
fprintf(stderr, "Calculating delivery ratio\n");
#endif // DEBUG
if (forwardDeliveryRatio > 0 && reverseDeliveryRatio > 0) {
return 1 / (forwardDeliveryRatio * reverseDeliveryRatio);
}
return INFINITY2;
}
```

APPENDIX C: NS2 Simulation Script for Routing Metrics

```
dynlibload aodveett ../src/.libs


set val(chan)Channel/WirelessChannel; # Channel Type
set val(prop)Propagation/TwoRayGround; # radio-propagation model
set val(netif)Phy/WirelessPhy; # network interface type
set val(mac)Mac/802_1; # MAC type
set val(ifq)Queue/DropTail/PriQueue; # interface queue type
set val(ll)LL; # link
layer type set val(ant)Antenna/OmniAntenna; # antenna model
set val(ifqlen)50; # max packet in ifq
set val(rp)AODVEETT # routing protocol
set val(rtAgentFunction)create-aodveettagent
set val(distance)200; # distance between nodes (m)
set val(stop)1000; set val(tracefile)"wireless-sim-aodv-grid.tr"
```

```
set val(namfile)"wireless-sim-aodv-grid.nam"
set opt(n)0; # number ofnodes
set opt(x)0; set opt(y)0; set op(d)0
set opt(seed)0.0 set opt(stop)0.0; # simulation time set opt(tr)""
set opt(tfc)"traf196d"
Agent/AODVEETT
set PacketSize_ 1024; # 1 kB Agent/AODVEETT
set Bandwidth_ 1375000; # 11 Mb/s
//============================================================
proc stop {} { global ns_ tracefd namtrace
$ns_ flush-trace close $tracefd close
$namtrace }
proc usage { argv0 } { puts "Usage: $argv0" puts "\tmandatory arguments:" puts
"\t\t\[-n NODES PER ROW\] \[-x MAXX\] \[-y MAXY\]"
puts "\toptional arguments:" puts "\t\t\[-d distance\] \[-seed seed\] \[-stop sec\]
\[-tr tracefile\]\n" }
proc getopt {argc argv} { global opt lappend optlist seed sc stop tr x y for {set i
0} {$i < $argc} {incr i} { set arg [lindex $argv $i] if {[string range $arg 0 0]
!= "-"} continue set name [string range $arg 1 end] set opt($name) [
lindex $argv [expr $i+1]] } }
proc recordStats {} { global val ns_ sink1 sink2 # How many bytes have been received
by the traffic sinks? set sinkBytes [$sink2 set bytes_] # Get the current time set
now [$ns_ now] # Calculate the bandwidth (in MBit/s) and write it to the files set
bandwidth [format "%.5f" [expr $sinkBytes/4.0*8]] puts "$now BANDWIDTH
$bandwidth" # Reset the bytes_ values on the traffic sinks $sink2 set bytes_ 0
# Reschedule
the procedure $ns_ at [expr $now + 4.0] "recordStats" }
#
```

```
# ==================================================
# Main Program #
# ==================================================
getopt $argc $argv if { $opt(n) == 0 || $opt(x) == 0 || $opt(y) == 0 } { usage
$argv0 exit 1 }
set val(nn) [expr $opt(n) *
$opt(n)]
if {$opt(d) > 0} { puts "Setting distance between nodes to $opt(d)\n" set $val(distance) $opt(d) } if
{$opt(seed) > 0} { puts "Seeding
Random number generator with $opt(seed)\n" ns-random $opt(seed) }
if {$opt(stop) > 0} { puts "Setting simulation duration to $opt(seed) seconds\n" $val(
stop) = $opt(stop) }
 if {$opt(tr) != ""} { puts "Setting tracefile name to opt(tr)\n" $val(tr) = $opt(tr) }
# Initialize Global Variables set ns_ [new Simulator] set tracefd [open $val(tracefile) w]
$ns_ use-newtrace $ns_ trace-all $tracefd set namtrace [open $val(namfile) w] $ns_
namtrace-all-wireless $namtrace $opt(x) $opt(y) # set up topography objecte set topo [new
Topography] $topo load_flatgrid $opt(x) $opt(y)
# Create God create-god $val(nn)
# Create channel set chan_ [new $val(chan)]
# Create node(0) "attached" to channel #1 # configure node, please note the change below.
$ns_ node-config -rtAgentFunction $val(rtAgentFunction) \ -adhocRouting $val(rp)
\ -llType $val(ll) \ -macType $val(mac) \ -ifqType $val(ifq)
\ -ifqLen $val(ifqlen) \ -antType $val(ant) \ -propType $val(prop)
\ -phyType $val(netif) \ -topoInstance $topo \ -agentTrace ON
\ -routerTrace ON \ -macTrace ON \ -movementTrace OFF
\ -channel $chan_ for {set i 0} {$i < $opt(n)} {incr i} { for {set j 0} {$j <
$opt(n)} {incr j} { set id [expr $i * $opt(n) + $j] set node_($id) [$ns_ node]
$node_($id) random-motion 0
```

```
$node_($id) set X_ [expr $val(distance) * $j]
$node_($id) set Y_ [expr $val(distance) * $i]
$node_($id) set Z_ 0.00 } }
for {set i 0} {$i < $val(nn)} {incr i} { $ns_ initial_node_pos $node_($i) 20 }
# Setup traffic flow between nodes # TCP connections between node_(0) and node_(1) puts
"Loading traffic file..." source $opt(tfc)
# Tell nodes when the simulation ends #$ns_ at 6.0 "recordStats" for {set i 0} {$i <
$val(nn)} {incr i} { $ns_ at $val(stop) "$node_($i) reset"; }
$ns_ at $val(stop) "stop" $ns_ at $val(stop).01 "puts \"NS EXITING...\" ; $ns_ halt" puts
"Starting Simulation..." $ns_ run
```