Improving end-to-end delay for real-time multimedia traffic in Mobile Ad-hoc Networks

Jacobus Nicolaas Boshoff

Presented in partial fulfilment
of the requirements for the degree

MAGISTER IN COMPUTER AND ELECTRONIC ENGINEERING

Faculty of Engineering
School of Electrical, Electronic and Computer Engineering
North-West University
Potchefstroom Campus

Supervisor: Professor A.S.J. Helberg

16 April, 2008

• Getting it right • Re dira sentle • Ons doen dit reg

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
POTCHEFSTROOM CAMPUS

Abstract

Mobile Ad-hoc Networks (MANETs) are infrastructure-less, self-organizing wireless networks. Nodes in these networks are able to communicate in the absence of access points, via single-hop or multi-hop paths on a peer-to-peer basis. Efficient routing in MANETs remains a big challenge due to the dynamic and uncertain nature of these networks, as well as the limited and continuously altering energy resources and bandwidth of mobile nodes.

MANET routing protocols have to be lightweight due to the limited processing power and memory resources of mobile nodes, and have to be able to adapt quickly to dynamic changes in network topology.

A large number of proposed protocols use the number of hops between a transmitting source node and a receiving destination node as metric to select a suitable route. The route with the least hops is thus used. For real-time traffic, and especially in a congested network, the route with the least number of hops can not be assumed to provide the best Quality of Service (QoS), since queuing time at one node may be orders higher than at another. Furthermore, since depending on the Medium Access Control protocol, the physical environment and the distance between communicating nodes, amongst others, the available bandwidth provided by a specific link can hardly ever be guaranteed.

In this research a QoS extension of the *Ad hoc On-demand Distance Vector* (AODV) routing protocol is proposed, implemented and tested in *OPNET Modeler*. First, the metric for selecting routes is changed to end-to-end delay, and secondly, the protocol is extended to a multi-path protocol.

Both extensions realize much lower end-to-end delay for application packets than AODV, and especially the multi-path extension with which all end-to-end delay are lowered to less than the maximum allowed delay for real-time traffic as recommended by the ITU-T. In most scenarios, the average packet delay variation, packet delivery fraction and routing overhead are improved by the final version of the AODV extension, and the overall protocol performance is also better than that of DSR, OLSR and DYMO.

Opsomming

Mobiele *Ad-hoc* Netwerke (MANETe) is self-organiserende draadlose netwerke en het geen vaste infrastruktuur nie. Nodes in hierdie netwerke is in staat om met mekaar te kommunikeer in die afwesigheid van toegangspunte, deur gebruik te maak van enkel-bons of multi-bons roetes op 'n direkte konneksie basis. Effektiewe roetebepaling vir pakkies in MANETe bly 'n groot uitdaging as gevolg van die dinamiese en onsekere aard van hierdie netwerke, asook die beperkte en kontinu-varieerdende energiebronne en bandwydte van mobiele nodes.

MANET roetebepalingsprotokolle moet liggewigtig wees as gevolg van die beperkte prosseseringskapasiteit en geheue van mobiele nodes, en moet in staat wees om vinnig aan te pas by dinamiese veranderinge in netwerktopologie.

Vele voorgestelde protokolle gebruik die aantal bonse tussen 'n sturende bron node en 'n ontvangende bestemmingsnode as metriek om 'n gepaste roete te kies. Die roete met die minste bonse word dus gebruik. Vir intydse verkeer, veral in versadigte netwerke, kan dit nie aanvaar word dat die roete met die minste bonse die beste kwaliteit van diens gaan bied nie, aangesien die wagtyd by een node ordes hoër kan wees as by 'n ander. Verder, omdat dit so afhang van o.a. die medium-toegangsbeheerprotokol, die fisiese omgewing en die afstand tussen kommunikerende nodes, kan die beskikbare bandwydte wat deur 'n spesifieke skakel gebied word, omtrent nooit gewaarborg word nie.

In hierdie navorsing word 'n kwaliteit-van-diens uitbreiding van die *Ad hoc On-demand Distance Vector* (AODV) roetebepalingsprotokol voorgestel, geïmplementeer en getoets in *OPNET Modeler*. Eerstens word die metriek wat gebruik word om roetes te kies verander na end-tot-end vertraging, en tweedens word die protokol uitgebrei na 'n multi-pad protokol.

Albei uitbreidings realiseer heelwat laer end-tot-end vertraging van datapakkies as AODV, en veral die multi-pad uitbreiding waarvan alle punt-tot-punt vertraging verminder is tot minder as die maksimum toelaatbare vertraging vir intydse verkeer soos voorgestel deur die ITU-T. In meeste gevalle is die pakkie vertraging variasie, die pakkie aflweringsbreukdeel en die roetebepalingoortolligheid verbeter deur die finale weergawe van die AODV uibreiding, en die algehele protokolwerksverrigting is ook beter in vergelyking met DSR, OLSR en DYMO.

Acknowledgements

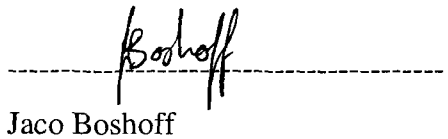
First and above all, I thank God for being my heavenly father, Jesus Christ for being my saviour, and the Holy Spirit for being my guide. Thank you for guiding me through this work, for helping me in hard times, and for the gift of a healthy mind.

Prof. Albert S.J. Helberg: For his guidance in my work, and for always having bright new ideas. Thanks for the encouragement, comments and for believing in my work.

My parents, Nico and Des Boshoff: Thank you for working so hard and by that supporting me in my first two years of study. Thanks for your love.

My beautiful girlfriend, Esté de Villiers: Thank you for being part of my life and for providing me with cheer when it's needed the most. Thanks for carrying God's light into the world and for being such a good representative of His heart and beauty! You mean the world to me.

Telkom Centre of Excellence: Thank you for the financial support, which enabled me to continue with and complete my Master's degree.



Jaco Boshoff

16 April, 2008

Table of Contents

CHAPTER 1 – INTRODUCTION 1
1. INTRODUCTION 1
1.1 THE ROUTING PROBLEM 1
1.2 QUALITY OF SERVICE 2
1.3 MULTI-PATH ROUTING AND LOAD BALANCING 2
2. PROBLEM STATEMENT 2
2.1 MAXIMUM DELAY 2
2.2 MULTI-PATH ROUTING 3
3. IMPORTANCE OF STUDY 3
4. PROJECT SCOPE 4
4.1 DELIMITATIONS 4
4.2 HYPOTHESES 5
5. DOCUMENT OVERVIEW 5

CHAPTER 2 – LITERATURE STUDY 6
1. COMPUTER NETWORKS 6
1.1 WIRELESS NETWORKS 6
1.1.1 Access point networks *6*
1.1.2 Mobile Ad-hoc Networks (MANET) *6*
1.2 LAYERED COMMUNICATION SYSTEM MODELS 7
2. REAL-TIME TRAFFIC 8
2.1 REAL-TIME PROTOCOL 9
3. QUALITY OF SERVICE (QOS) 9
3.1 PERFORMANCE-ORIENTED QUALITY PARAMETERS 10
3.2 NON-PERFORMANCE-ORIENTED QUALITY PARAMETERS 10
3.3 QOS FOR REAL-TIME TRAFFIC 10
4. ROUTING 11
4.1 INTRODUCTION [11] 11
4.2 MULTI-PATH ROUTING 12
4.3 LOAD BALANCING 13
5. MANET ROUTING PROTOCOLS 13
5.1 INTRODUCTION 13
5.2 PROACTIVE VS. REACTIVE PROTOCOLS 14
5.3 THE AD HOC ON-DEMAND DISTANCE VECTOR (AODV) ROUTING PROTOCOL [6] 14
5.3.1 Basic protocol operation [14] *14*
5.3.2 AODV sequence numbers and route updating *15*
5.3.3 Local repair *16*
5.3.4 AODV packet formats *16*
5.4 THE DYNAMIC SOURCE ROUTING (DSR) PROTOCOL [29] 18
6. PROPOSED MANET PROTOCOLS 19
6.1 QUALITY OF SERVICE PROTOCOLS 20
6.1.1 Quality of Service for Ad hoc On-demand Distance Vector Routing [8] *20*
6.2 MULTI-PATH PROTOCOLS 21
6.2.1 Ad hoc On-demand Distance Vector Multipath Routing [9] *21*
6.2.2 Ad hoc On-demand Multipath Distance Vector Routing [14] *22*
6.2.3 Adaptive Multi-path On-demand Routing [13] *22*
6.3 LOAD BALANCING PROTOCOLS 22
6.3.1 Load-Balanced Ad hoc Routing [3] *22*
6.4 THE AODV-MULTIPATH (AODVM) ROUTING PROTOCOL [9] 23
7. CHAPTER CLOSURE 24

CHAPTER 3 – CONCEPTUAL DESIGN **25**
1. DELAY-AWARE AODV (DA-AODV) [5] **25**
1.1 THE CONCEPT 25
1.1.1 Data classification *25*
1.1.2 Route discovery *25*
1.1.3 DA-AODV route updating *26*
1.1.4 Route maintenance *27*
1.1.5 Local repair *28*
1.1.6 Packet queues *28*
1.2 DATA STRUCTURES 28
1.2.1 Packet formats *28*
1.2.2 Table formats *28*
1.3 DA-AODV FLOW DIAGRAMS 29
2. DELAY-AWARE AODV-MULTI-PATH (DA-AODVM) **35**
2.1 THE CONCEPT 35
2.2 DATA STRUCTURES 37
2.2.1 Packet formats *37*
2.2.2 Table formats *37*
2.3 DA-AODVM FLOW DIAGRAMS 37
3. CHAPTER CLOSURE **41**

CHAPTER 4 – IMPLEMENTATION **42**
1. INTRODUCTION **42**
2. IMPORTANT OPNET KERNEL PROCEDURES [28] **44**
2.1 LISTS 44
2.2 HASH TABLES 45
2.3 DATA PACKETS 47
2.4 MEMORY ALLOCATION 47
2.5 OTHER 48
3. EXTENDING THE AODV PROTOCOL **48**
4. THE DA-AODV EXTENSION **50**
4.1 EXTENDED DATA STRUCTURES 50
4.2 EDITED CODE PIECES 50
5. THE DA-AODVM EXTENSION **54**
5.1 EXTENDED DATA STRUCTURES 54
5.2 EDITED CODE PIECES 56
5.3 NEW FUNCTIONS 59
6. CHAPTER CLOSURE **60**

CHAPTER 5 – SIMULATION SETUP **61**
1. PHYSICAL NETWORK LAYOUT **61**
2. NODE CONFIGURATION **61**
2.1 AODV RELATED PARAMETERS 61
2.2 WIRELESS LAN PARAMETERS 63
3. APPLICATIONS CONFIGURATION **63**
3.1 VOICE-OVER-IP 64
3.2 VIDEO CONFERENCING 64
4. NODE MOBILITY CONFIGURATION **65**
5. OBTAINING RESULTS **65**
6. CHAPTER CLOSURE **65**

CHAPTER 6 – SIMULATION RESULTS 66
1. PRESENTED RESULTS 66
1.1 AVERAGE PACKET END-TO-END DELAY 66
1.2 AVERAGE DELAY VARIATION 66
1.3 PACKET DELIVERY FRACTION 67
1.4 ROUTING OVERHEAD 67
2. STATIC SINGLE-HOP NETWORKS 68
2.1 AVERAGE PACKET END-TO-END DELAY 68
2.2 AVERAGE DELAY VARIATION 69
2.3 PACKET DELIVERY FRACTION AND ROUTING OVERHEAD 70
2.4 DISCUSSION 72
3. STATIC MULTI-HOP NETWORKS 73
3.1 AVERAGE PACKET END-TO-END DELAY 73
3.2 AVERAGE DELAY VARIATION 74
3.3 PACKET DELIVERY FRACTION AND ROUTING OVERHEAD 75
3.4 DISCUSSION 77
4. MOBILE SINGLE-HOP NETWORKS 77
4.1 AVERAGE PACKET END-TO-END DELAY 77
4.2 AVERAGE DELAY VARIATION 79
4.3 PACKET DELIVERY FRACTION AND ROUTING OVERHEAD 80
4.4 DISCUSSION 81
5. MOBILE MULTI-HOP NETWORKS 83
5.1 AVERAGE PACKET END-TO-END DELAY 83
5.2 AVERAGE DELAY VARIATION 83
5.3 PACKET DELIVERY FRACTION AND ROUTING OVERHEAD 84
5.4 DISCUSSION 86
6. GENERAL DISCUSSION 86
6.1 VERIFICATION 87
6.2 VALIDATION 88
7. CHAPTER CLOSURE 88

CHAPTER 7 – CONCLUSION AND RECOMMENDATIONS 89
1. CONCLUDING REMARKS 89
2. PROPOSED IMPROVEMENTS 90
2.1 DA-AODV 90
2.1.1 Updating route information 90
2.1.2 Handling application packets 90
2.2 DA-AODVM 91
2.2.1 Management of multiple routes 91
2.2.2 Updating route information 92
2.2.3 Route reply errors 92
2.2.4 RREP message propagation 92
2.2.5 Precursor lists 92
3. FUTURE WORK 93

REFERENCES 94

APPENDIX A 97
1. PROTOCOL COMPARISON FOR 81 NODE NETWORKS 97
2. PROTOCOL COMPARISON FOR 41 NODE NETWORKS 99
APPENDIX B 102
APPENDIX C 119

List of Figures

Figure 2.1 The OSI protocol stack 7
Figure 2.2 RREQ packet format 17
Figure 2.3 RREP packet format 17
Figure 2.4 RERR packet format 18
Figure 2.5 RREP-ACK packet format 18
Figure 2.6 (a) Route Request Table Entry (b) Route Table Entry [9] 23

Figure 3.1 DA-AODV RREQ/RREP packet format 28
Figure 3.2 General packet arrival handle flow 29
Figure 3.3 DA-AODV Application packet arrival handle flow 30
Figure 3.4 DA-AODV RREQ packet arrival handle flow 32
Figure 3.5 DA-AODV RREP packet arrival handle flow 33
Figure 3.6 HELLO packet arrival handle flow 34
Figure 3.7 All packets to destination send flow 35
Figure 3.8 DA-AODVM Application packet arrival handle flow 38
Figure 3.9 DA-AODVM RREQ packet arrival handle flow 39
Figure 3.10 DA-AODVM RREP packet arrival handle flow 40

Figure 4.1 Layered node model 43
Figure 4.2 AODV process model 44
Figure 4.3 Hash table structure example [28] 46
Figure 4.4 Node Attributes 49

Figure 5.1 Physical network layout 62
Figure 5.2 Voice application packet settings 64
Figure 5.3 Video applications packet settings 64

Figure 6.1 Video packet delay (static single-hop network) 68
Figure 6.2 Voice packet delay (static single-hop network) 68
Figure 6.3 Video packet delay variation (static single-hop network) 69
Figure 6.4 Voice packet delay variation (static single-hop network) 70
Figure 6.5 Video packet delivery fraction (static single-hop network) 70
Figure 6.6 Voice packet delivery fraction (static single-hop network) 71
Figure 6.7 Routing overhead (static single-hop network) 71
Figure 6.8 Video packet delay (static multi-hop network) 73
Figure 6.9 Voice packet delay (static multi-hop network) 74
Figure 6.10 Video packet delay variation (static multi-hop network) 74
Figure 6.11 Voice packet delay variation (static multi-hop network) 75
Figure 6.12 Video packet delivery fraction (static multi-hop network) 75
Figure 6.13 Voice packet delivery fraction (static multi-hop network) 76
Figure 6.14 Routing overhead (static multi-hop network) 76
Figure 6.15 Video packet delay (mobile single-hop network) 78
Figure 6.16 Voice packet delay (mobile single-hop network) 78
Figure 6.17 Video packet delay variation (mobile single-hop network) 79
Figure 6.18 Voice packet delay variation (mobile single-hop network) 79

Figure 6.19 Video packet delivery fraction (mobile single-hop network) 80
Figure 6.20 Voice packet delivery fraction (mobile single-hop network) 80
Figure 6.21 Routing overhead (mobile single-hop network) 81
Figure 6.22 Video packet delay (mobile multi-hop network) 82
Figure 6.23 Voice packet delay (mobile multi-hop network) 82
Figure 6.24 Video packet delay variation (mobile multi-hop network) 83
Figure 6.25 Voice packet delay variation (mobile multi-hop network) 84
Figure 6.26 Video packet delivery fraction (mobile multi-hop network) 84
Figure 6.27 Voice packet delivery fraction (mobile multi-hop network) 85
Figure 6.28 Routing Overhead (mobile multi-hop network) 85

List of Tables

Table 6.1 Protocol performance summary for static single-hop networks 72
Table 6.2 Protocol performance summary for static multi-hop networks 77
Table 6.3 Protocol performance summary for mobile single-hop networks 81
Table 6.4 Protocol performance summary for mobile multi-hop networks 86

Table A.1 Protocol comparison for static single-hop networks (Video traffic) 97
Table A.2 Protocol comparison for static single-hop networks (Voice traffic) 97
Table A.3 Protocol comparison for static multi-hop networks (Video traffic) 98
Table A.4 Protocol comparison for static multi-hop networks (Voice traffic) 98
Table A.5 Protocol comparison for mobile single-hop networks (Video traffic) 98
Table A.6 Protocol comparison for mobile single-hop networks (Voice traffic) 98
Table A.7 Protocol comparison for mobile multi-hop networks (Video traffic) 99
Table A.8 Protocol comparison for mobile multi-hop networks (Voice traffic) 99
Table A.9 Protocol comparison for static single-hop networks (Video traffic) 99
Table A.10 Protocol comparison for static single-hop networks (Voice traffic) 99
Table A.11 Protocol comparison for static multi-hop networks (Video traffic) 100
Table A.12 Protocol comparison for static multi-hop networks (Voice traffic) 100
Table A.13 Protocol comparison for mobile single-hop networks (Video traffic) 100
Table A.14 Protocol comparison for mobile single-hop networks (Voice traffic) 100
Table A.15 Protocol comparison for mobile multi-hop networks (Video traffic) 101
Table A.16 Protocol comparison for mobile multi-hop networks (Voice traffic) 101

Important Acronyms

AODV	– Ad hoc On-demand Distance Vector
AP	– Access Point
BE	– Best Effort
CSMA/CA	– Carrier Sense Multiple Access with Collision Avoidance
DSDV	– Destination-Sequenced Distance-Vector
DSR	– Dynamic Source Routing
DSSS	– Direct Sequence Spread Spectrum
ICMP	– Internet Control Message Protocol
IEEE	– Institute of Electrical and Electronic Engineers
IETF	– Internet Engineering Task Force
IP	– Internet Protocol
IPv4	– Internet Protocol version 4
ITU	– International Telecommunications Union
MAC	– Medium Access Control
OSI	– Open System Interconnect
OSLR	– Optimized Link State Routing
PDU	– Protocol Data Unit
QAODV	– QoS-AODV
QoS	– Quality of Service
RERR	– Route Error
RFC	– Request for Comments
RREP	– Route Reply
RREP-ACK	– Route Reply Acknowledgement
RREQ	– Route Request
ToS	– Type of Service
UDP	– User Datagram Protocol
VoIP	– Voice over Internet Protocol
WLAN	– Wireless Local Area Network

Acronyms used in flow diagrams

addr. – address

app. – application

conn. – connectivity

dest. – destination

fwd. – forward

inc. – increment

max. – maximum

msg. – message

orig. – originator

pkt. – packet

req. – request

rev. – reverse

rte. – route

seq. – sequence

src. – source

tbl. – table

Chapter 1 – Introduction

This chapter gives an introduction to the research that is documented in this dissertation, a problem statement and the importance of this study, a scope definition as well as a layout overview of the rest of the document.

1. Introduction

Mobile Ad-hoc Networks (MANETs) are infrastructure-less wireless networks in which nodes are connected via wireless single-hop or multi-hop paths. These networks function without any fixed access points (AP), and nodes are completely free to move around. With no fixed infrastructure, each node in a MANET should be able to perform basic routing functions such as route discovery and maintenance, and packet forwarding. Thus, with no centralized routing control, an efficient routing protocol is a critical factor for the effective operation of a MANET. The growing demand for such infrastructure-less networks brings the question of Quality of Service (QoS) guarantees to the horizon [1].

Real-time service provisioning in wireless networks is a difficult task. This is partially due to the variation of QoS requests of different applications, and the dynamic and distributed nature of mobile networks [2]. Two very limited resources of mobile nodes are bandwidth and battery power [3]. Load balancing, where packets are distributed over multiple paths between a source and destination has proved to eliminate some disadvantages caused by these limitations [l], [3].

1.1 The Routing Problem

Since each node in a MANET has to be able to perform routing functions, the following characteristics make routing in MANETs quite challenging: the dynamic and uncertain topology, limited and varying bandwidth, power limitation of mobile devices, the limited

physical security and lack of centralized control [1]. Various QoS, security or other requirements of modern network applications make the routing problem more intractable than before [4].

1.2 Quality of Service

To assure that the performance requirements of a network application will be met, a network should be able to service various QoS requests of applications [5]. These requests may include minimum bandwidth, maximum loss ratio, maximum delay and delay variation.

High bit error rates and frequent link and node failures in mobile networks make QoS provisioning for real-time applications a very challenging task. Furthermore, applications often demand high bandwidth, flow synchronization and delay sensitivity. [1], [2]

1.3 Multi-path routing and load balancing

In multi-path routing, not only the best path to route packets is stored, but up to n alternative paths, where n is a specified integer, are stored. Potential advantages of multi-path routing are higher reliability, more effective bandwidth utilization and better security. [1]

2. Problem statement

The research presented in this document involves the following: Extending the Ad hoc On-demand Distance Vector (AODV) [6] routing protocol for MANETs to a delay-aware multi-path protocol. The focus area is to improve the QoS in MANETs by creating a protocol, which considers delay requests of real-time multimedia applications (voice and video) in making routing decisions. Extending the abovementioned to a multi-path protocol further extends the solution.

2.1 Maximum Delay

The metric used by AODV in selecting routes is hop count [6]. From [10] and Appendix A, AODV manages unacceptable delay for real-time traffic. In the AODV extensions in this

work, end-to-end delay is used as the metric for selecting routes. This way, an application can be assured that selected routes will provide end-to-end delay levels suitable for its traffic type.

2.2 Multi-path routing

Discovering new routes can be time consuming. If a path, which is used for a communication session fails in the middle of the session, route rediscovery may take up a relatively large amount of time. This could cause an unacceptable increase in packet end-to-end delay, leading to inconsistency and high delay variation. This might be unacceptable for delay sensitive traffic such as real-time traffic.

Independent of how multiple paths are used, a multi-path protocol discovers and stores more than a single route at a time. This may cause initial routing overhead to increase, but if a link, and thus a path fails, and any alternative route to the destination is known and up to date, transmission can immediately continue via such an alternative route. This eliminates potential long waiting periods for route rediscovery, as well as the route rediscovery overhead that is required to establish a new route.

3. Importance of study

Due to the comfort of mobility, combined with low or no installation cost, MANETs are becoming a very attractive alternative to wired, and even AP based wireless networks. As a result of the random nature of MANETs, a large amount of uncertainty exists in these networks, making it very hard to provide QoS guarantees to users. Thus, even though MANETs have some very attractive features, users will expect some level of guarantee that the requested QoS can be provided, before replacing existing and trusted network topologies with MANETs.

Due to the growing popularity of real-time communication over Internet Protocol (IP) networks [15], it is also required of a MANET to be able to service delay sensitive real-time traffic in a way that will cause data to be reliable and of acceptable quality, in order to be a

competitive alternative to wired networks. Current available best effort (BE) routing protocols are not able to satisfy this requirement [10].

4. Project Scope

The scope of this research entails the combination of selective components of two proposed extensions of the AODV protocol, in a new protocol that will meet the outcomes stated in section 2 of this chapter. The first protocol to be used in this work is *QoS for AODV* [8]; a proposal by C.E. Perkins and E. Belding-Royer, of which no results or performance analyses are available or could be found. The second AODV extension is AODVM [9]; a multi-path extension proposed by Z. Ye, S. V. Krishnamurthy and S. K. Tripathi.

This work is a first step in extending a BE routing protocol to a QoS protocol for MANETs. Thus, only selective components from the abovementioned proposals, that will ensure the basic required protocol functionality, are used in this work. Optimizing protocol components such as efficient route maintenance and management are proposed for future work. Such components could significantly improve the protocol's performance, but are not essential for the basic required functionality.

4.1 Delimitations

The following delimitations were set to keep the focus of the work, and to outline attainable research outcomes:

- Implementation is limited to the *OPNET Modeler* simulation environment, thus, no implementation in a physical network or test bed will be done.
- The solution is limited to the MANET routing layer, operating in the network layer (chapter 2, section 1.2) right on top of IP. No cross-layer optimizations are included in the proposed work, and thus, all influencing factors may not be considered.
- All nodes in the simulated networks are set to fully and honestly participate in routing activities, as required of them. Any security threats or potential malicious activities of nodes are assumed to be non-existent, making the networks ideal in this view.

4.2 Hypotheses

This study hypothesizes that:

- Lower packet end-to-end delay can be obtained by changing the main metric of a BE protocol from hop count to route delay. If processing, queuing and propagation delay are exactly the same for all nodes in a route, thus assuming that route delay is uniformly distributed amongst all nodes in a route, hop count could serve as a relative measurement for delay in multi-hop routes. This is rarely the case, especially in congested networks, and the actual measured delay should serve as a more reliable metric.
- Discovering and storing multiple routes simultaneously, and immediately selecting a known alternative route upon the failure of an actively used route can further reduce average packet end-to-end delay.

5. Document Overview

The rest of this document is organized as follows:

Chapter 2 contains the necessary background to understand the foundation of this study. The chapter provides relevant information on computer networks, traffic types, Quality of Service, routing and routing protocols.

Chapter 3 documents the conceptual design of the proposed work. This includes the methodical discussion of the concepts that is unique to this work, data structures and software flows.

Chapter 4 discusses the implementation of the proposed protocols in the *OPNET Modeler* network simulation environment. This includes implementing and registering a new protocol in OPNET, as well as a discussion of the code that was required to make the necessary modifications.

Chapter 5 discusses the simulation setup, in which the proposed implementation is tested and compared.

Chapter 6 presents the processed results of the output data that was obtained from the simulations. The chapter reviews all the important issues regarding these results.

Chapter 7 provides the final concluding remarks to this work. The chapter also proposes potential future study.

Chapter 2 – Literature Study

This chapter gives an overview on all literature and theory that serve as a foundation of this research. It also provides relevant background theory on computer networks, traffic types, Quality of Service, routing and MANET routing protocols.

1. Computer networks

A computer network can be defined as a collection of nodes that are connected to each other via a communication system. This allows nodes to communicate and share resources with each other. The main purpose of computer networks is to realize data communications. From [11], data communication is defined as "the problem of getting information from one place to another reliably while conforming to user requirements".

1.1 Wireless networks

1.1.1 Access point networks

In general, wireless networks are implemented using APs. An AP is a dedicated node, which serves as an intermediate hop between a source and destination pair. All communication happens via the AP [12]. Although end nodes may be mobile, APs are usually fixed and thus provide a degree of structure to a wireless network. Omni-directional or directional antennas can be used with APs, depending on the preferred network coverage area and the desired reception distance.

1.1.2 Mobile Ad-hoc Networks (MANET)

Mobile Ad-hoc Networks (MANETs) are a subset of wireless networks. In these networks, no guarantee exists that any node are fixed to provide a degree of structure to the network. This implies that such a network needs to be self-organized and accommodative to dynamic topology changes. [13]

No APs are used in a MANET, thus nodes communicate directly with each other on a peer-to-peer basis. However, communicating nodes may not always be within each other's transmission range. In such a case, intermediate nodes can be used to form a multi-hop path between a source and a destination. Any node in a MANET should thus be able to perform basic routing tasks such as route discovery and packet forwarding [13].

Three characteristics of MANETs that largely contribute to their dynamic nature are node mobility, and the limited and altering bandwidth and energy resources of mobile nodes [14]. These networks are very useful when no network infrastructure is available, but real-time and reliable communication is crucial. Good examples are military deployments, disaster rescue operations, electronic classrooms and providing network services in rural areas [9], [13].

1.2 Layered communication system models

Communication systems are partitioned into several layers of abstraction, each layer performing a very specific subset of functions or tasks, enabling one system to communicate with another system. A collection of functions in such a layer is called an entity. These entities on different systems communicate with each other by using one or more protocols. Thus, this concept of partitioning a system into abstract layers is referred to as a *protocol stack.* In this protocol stack, each layer is dependent on the layer underneath it to perform a set of lower level function in order to realize communication [11]. This enables designers to modify and improve functions in one layer, without affecting any of the other layers, as long as the interface to the upper and lower layers remains unchanged.

A very common communication model is the Open Systems Interconnection (OSI) model. This model has seven layers and is illustrated in Figure 2.1.

Layer 7	Application
Layer 6	Presentation
Layer 5	Session
Layer 4	Transport
Layer 3	Network
Layer 2	Data Link
Layer 1	Physical

Figure 2.1 The OSI protocol stack

The function of each layer is described below [11]:

Application layer: Provides the user interface to the communications system. It is thus an application through which a user can access information on a network.

Presentation layer: Provides a common format or standard interface to the application layer. This allows hosts that use different representations to communicate with each other.

Session layer: This layer manages interactions between pairs of communicating application processes. It establishes, manages and terminates the connection between two communicating nodes. Services sent from the presentation layer are broken down into basic data flows, which can be sent over the system.

Transport layer: Provides reliable data transfer between users, taking the responsibility of the shoulders of upper layers. It is also responsible for the establishment and final release of a communications channel. For connection-orientated protocols, the transport layer keeps track of the packets and retransmits those that fail.

Network layer: The routing of data from a source node to a destination node takes place on this layer. It "interconnects data link communications paths into a global network that connects all open systems" [11].

Data Link layer: Ensures reliable communication over the physical layer. It detects and possibly corrects error that may have occurred in the physical layer.

Physical layer: The physical layer defines all the mechanical and electrical aspects of the communication medium. This includes the configuration of connectors, voltages and cable specifications.

This model defines services, interfaces and protocols.

2. Real-time traffic

The Internet and IP networks have become a very popular medium for transporting not only data traffic, but also real-time traffic for user services such as voice and video applications [15]. A big challenge in using IPv4 for the transmission of speech or video is the fact that IPv4 was never designed to meet the QoS demands of real-time multimedia traffic, but rather, it handles traffic on a *first-in-first-out* (FIFO) basis, thus providing only BE QoS [16], [17].

2.1 Real-Time Protocol

The *Real-Time Protocol* (RTP) operates right on top of the link layer protocol, which is mainly the User Datagram Protocol (UDP) for real-time traffic. RTP provides a common packet format, which is suitable for carrying real-time multimedia streams. An important feature of this packet format is the sequence number field in the packet header. Multimedia streams are cut up in smaller data pieces for transmission purposes. Therefore, these numbers are used to reconstruct the data streams at the destination, as data packets may arrive out of sequence. Neither RTP nor UDP provide any QoS [11].

3. Quality of Service (QoS)

Different layers of abstractions exist (section 1.2 of this chapter) in a communication system. Each layer performs its own important set of functions. However, most users interface with the system only via an application. Thus, the performance of the underlying layers must be of such quality that satisfactory observation is provided at the application layer. The underlying layers may not even know whether the data to be transmitted is voice, video or other data, but only "see" data packets that have to be delivered according to certain quality criteria.

QoS is a networking concept that has received much attention over the past few years, especially with the rise of real time services over IP networks. Quality of service can be defined as "a set of specific requirements for a particular service provided by the network to the subscribed users" [20]. As mentioned, IPv4 itself does not provide any QoS. Thus, a number of alternative solutions, mainly by the Internet Engineering Task Force (IETF), have been proposed. Some of the most popular and well-studied solutions include *Integrated Services* (IntServe)/*RSVP* [21], [22], *Differentiated Services* (DiffServe) [23], and *Multi-protocol Label Switching* (MPLS) [24], [17].

The International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) has defined two groups of QoS criteria, namely *performance-oriented* and *non-performance-oriented* quality parameters [11].

3.1 Performance-oriented quality parameters

The performance-oriented parameters describe the direct quality of a connection [11]:

Call set-up delay: The time between a call connection request and connection confirmation.

Probability of blocking: The probability that the setup of a requested connection fails.

Throughput: The maximum amount of successful data transfer per time unit on a sustained basis.

Frame delay: The delay between creating and receiving a Protocol Data Unit (PDU).

Frame jitter: The variance between the minimum and maximum frame delay.

Frame Error Rate: The probability of corruption, loss or duplication at the receiver of a PDU.

Probability of dropping: The probability that a connection breaks within a specified time.

Release delay: The time between an end-of-call request and connection release confirmation.

3.2 Non-performance-oriented quality parameters

Non-performance-oriented parameters are not directly related to the communication performance, but refer to related issues [11]:

Level of service: The certainty that an agreed QoS level is delivered.

Cost: The price paid for maintaining an agreed level of QoS.

Priority: The preference assigned to a specific service.

3.3 QoS for real-time traffic

Delay is an important parameter in real-time data transmission. Speech is very sensitive to delay. The ITU-T standard G.114 recommends that one-way end-to-end delay should be less than 150 ms for high quality Voice-over-IP (VoIP). Delay between 150 ms and 400 ms are tolerable, but not preferred, and above 400 ms is not acceptable at all [25], [26]. It is preferred that heavily delayed packets are rather discarded, since speech signals can tolerate the loss of a limited amount of information. Most network designers try to keep end-to-end delay of real-time packets below 50 ms, but too tight constraints may result in a large amount of packets being dropped, and thus high loss rates [11].

For real-time traffic, three types of delay mainly contribute to the total end-to-end delay. The processing of voice or video samples causes accumulation delay. Processing delay is a result of these samples being packaged. Finally, multiplexing and buffering voice and video packets as they are transmitted over a network causes network delay [27].

Another important factor that has a significant influence on the voice quality is jitter. Jitter is defined as the difference between the arrival times of data packets, or the variation of delay between consecutive packets [11], [27]. Again, as mentioned before, heavily delayed packets should rather be discarded in order to minimize the jitter. Packets can be queued for a short time (total delay should be kept below 400 ms [25]) at the receiver in order to even out the delay [11].

4. Routing

4.1 Introduction [11]

Routing is a network layer (layer 3 of the OSI protocol stack) function and entails setting up paths between a source and a destination, and transferring data packets along these paths. A single fixed path may be set up once and used, forming a virtual circuit. Alternatively, routing decisions may be made on a packet-by-packet basis by the source and/or intermediate nodes.

An ideal routing protocol should be correct, simple, stable, robust, fair and optimal. Although it is not always possible to fully comply with all of these criteria, each criterion should be considered in a protocol design. This is especially true for MANETs with their dynamic nature and limited resources.

Any network is subject to failures, thus, a routing protocol should be able to adapt dynamically to network changes, without the need to restart. Fair service should be provided to all users. Service should not be denied to any user as a result of high demands of other users. Certain requirements can cause contradicting demands of a protocol when requested to be serviced concurrently. An example is low delay and maximum utilization, since higher network utilization directly results in a more congested network, leading to higher delays. It is then important to find an acceptable trade-off.

Centralized routing is a technique where a single node is responsible to collect relevant information, compute routes and distribute the information to all other relevant nodes. Opposed to this is *distributed routing* where nodes rely on information provided by neighbouring nodes to determine routes. This is the more feasible option for MANETs since no centralized infrastructure exists.

4.2 Multi-path routing

Multi-path protocols set up more than one route between a source and destination simultaneously. These multiple routes or paths can then be used for one or more of the following: To improve the reliability of transmitted information by transmitting the same information via multiple paths, load balancing, congestion avoidance, to reduce the frequency of route enquiries and to achieve lower routing overhead [9], [13]. In wireless networks, improving network reliability is one of the more popular focus areas of applying multi-path routing [1], [9], [18], [19].

Node mobility and the random nature of wireless links cause more frequent link breakages, and thus route failures, than in wired networks [13]. If multiple paths between a source and destination are available and known, a transmitting node will always have an alternative route available for transmission when a link or node failure occurs.

Two main types of multiple paths exist. The first is link-disjoint paths where multiple paths will never contain a common link, i.e. a direct link between two nodes. However, nodes are allowed to participate in multiple routes. The second type is node-disjoint paths where a specific intermediate hop may not participate in more than one path [14]. The main advantage of link-disjoint paths over node-disjoint paths is that usually, more paths will exists between a source and a destination, because intermediate hops can be "shared". The disadvantage is that a single node failure may cause multiple paths to fail. But, with node-disjoint paths, a single node failure will always cause only one path to fail. The drawback of node-disjoint paths is that fewer paths are usually available, as each intermediate node can only participate in a single path.

For some applications such as network coding [7], links shared in multiple routes may be essential, making not one of the abovementioned options applicable.

4.3 Load balancing

Load balancing is a routing technique where the traffic load in a network, or even in a single data stream between a source and a destination, is distributed over multiple paths. It has been thoroughly studied and deployed in circuit-switched and packet-switched networks [13]. Load balancing can dramatically increase the throughput in a network, and the energy consumption of mobile nodes in an ad-hoc network can be distributed more evenly.

When the mobility of nodes in a MANET is limited, MANET routing protocols have the tendency to use only a few centrally located nodes in a large number of routes [3]. In such a situation, congestion at the *Medium Access Control* (MAC) layer leads to high packet delays. These centrally located nodes may also suffer from high battery power consumption. Distributing the load evenly over multiple paths in a MANET can minimize these problems.

Higher bandwidth can be obtained by load balanced routing. For this, the traffic load of a single data stream between a source and destination pair can be distributed over n link-disjoint paths, resulting in a total bandwidth of $B = B_1 + B_2 + \ldots + B_{n-1} + B_n$ where B_i is the bandwidth of the i-th path and $i \in [1; n]$.

5. MANET routing protocols

5.1 Introduction

Several MANET routing protocols have been proposed, but only a few have been accepted by the IETF to be considered for standardization. This section discusses two popular and well-studied reactive MANET routing protocols, namely the *Ad hoc On-demand Distance Vector* (AODV) [6] routing protocol and the *Dynamic Source Routing* (DSR) [29] protocol. AODV has been published as an experimental protocol for the Internet community by the IETF (RFC 3561) in 2003 and DSR (RFC 4728) in 2007.

Section 5.4 gives a short overview on DSR. However, AODV is the protocol of interest for this study and therefore section 5.3 explains it in more detail.

5.2 Proactive vs. Reactive protocols

Traditional routing protocols are generally proactive, i.e. each node maintains a list of routes to all other nodes in the network, whether communication with a node exists or not. All nodes are notified in the case of topology changes. When a table driven (proactive) routing protocol is used, the dynamic nature of MANETs results in large amounts of routing overhead and memory utilization [13]. Examples of proactive routing protocols for MANETs are the *Destination Sequenced Distance Vector* (DSDV) [30] and *Optimized Link State Routing* (OLSR) [31] protocols.

Contrary to proactive protocols, the reactive protocols discover, store and maintain a route or multiple routes to a destination only when a node requires communication with the destination node. The route discovery process is thus real-time and reduces routing overhead. However, route discovery latency together with frequent route discovery attempts can degrade the real-time performance of a MANET [13], [14]. Multi-path routing reduces this problem, because route discovery only have to take place when all of the known routes to a destination fail [14]. Examples of reactive MANET routing protocols are AODV and DSR.

5.3 The Ad hoc On-demand Distance Vector (AODV) routing protocol [6]

The AODV routing protocol is a routing protocol for MANETs that provides dynamic, self-starting, multi-hop routing between nodes in such a network. It operates right on top of IP in the network layer. It can adapt quickly to dynamic link conditions. It has low processing and memory overhead and low network utilization. Loop freedom is guaranteed, and using destination sequence numbers eliminates the Bellman-Ford "counting to infinity" problem. When links used in active routes are no longer available, AODV causes nodes using these links to be notified.

5.3.1 Basic protocol operation [14]

AODV uses three message types in order to discover and maintain routes. They are Route Request (RREQ), Route Reply (RREP) and Route Error (RERR) messages. These are UDP packets encapsulated in normal IP packets.

When a node needs to communicate with a new destination, or a destination to which it does not have a valid route, it creates and broadcasts a RREQ message. Either the destination node, or an intermediate node with a valid route to the destination, can reply with a RREP message to this RREQ message (the second is optional). In the second case, it is also optional to notify the intended destination of the reply by sending a *gratuitous* RREP message. Duplicate copies of a specific RREQ message that are received by any intermediate node or destination are discarded.

Upon the reception of a RREQ message, an intermediate node first stores the route back to the originator before forwarding the RREQ message. This is used in the case where a RREP message needs to be sent back to the originator of the RREQ.

Each node is responsible to monitor the link status between itself and all next hops used in any active routes of which the node is part. If a link breaks and causes one or more destinations to become unavailable, it is any node detecting the breakage's responsibility to notify all nodes that are using the link in one or more active routes by sending a RERR message. The precursor lists stored with every route table entry are used to identify these nodes. A precursor list is a list in which a node keeps the necessary information of all the nodes that are likely to use itself as a next hop toward a specific destination.

5.3.2 AODV sequence numbers and route updating

AODV uses the concept of sequence numbers to keep track of route freshness. Each node using AODV maintains a monotonically increasing sequence number. The sequence number is incremented before a new route discovery is initiated, and also before a RREP message is sent, if the node is the intended destination. It is then included in the relevant messages in order to indicate the freshness of the information to all recipients. Each node also keeps track of the highest known sequence number provided by each destination node to which it has a valid route. These are known as *destination sequence numbers* [14].

Any node updates its routing information upon the reception of a RREQ or a RREP message. If a node has no known route to the node offering the information, a route via the *previous hop* from which the new route information was received, is stored. If it has a valid route to

the node offering possibly new information, the route is updated to the new information if the following route update rule is true:

if ($seq_nr^d_i < seq_nr^d_j$) **or**
($(seq_nr^d_i = seq_nr^d_j)$ **and**
$(hop_count^d_i > hop_count^d_j)$)
then
$seq_nr^d_i := seq_nr^d_j$;
$hop_count^d_i := hop_count^d_j + 1$;
$next_hop^d_i := j$;
endif (2-1)

It states that if the new sequence number is higher than the known one, or it is the same, but the new hop count to the node offering the new information is less, the new information is preferred above the known, and thus used. The notation applies for a node i that receives a route advertisement to destination d from a neighbour j. The destination sequence number, hop count and next hop for a destination d at node i is represented by $seq_nr^d_i$, $hop_count^d_i$ and $next_hop^d_i$ respectively.

5.3.3 Local repair

Local repair is a mechanism used by AODV that allows intermediate nodes to try and find an alternative route to a destination node if a broken link in an active route is detected. The node detecting the broken link, will notify all sources that made use of the link in an active route that it broke down. However, the route should not be deleted until further notice. The intermediate node would then initiate a route discovery to try and find an alternative route to the destination. The criterion is that an alternative route, if discovered, should have the same or smaller hop count than the previous known route. Local repair is optional.

5.3.4 AODV packet formats

This section describes the packet formats of the messages used in the AODV protocol. Non-self documenting fields are explained.

Route Request (RREQ) message

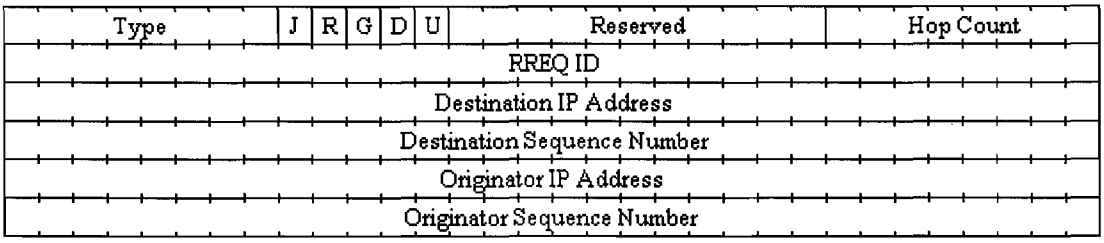


Figure 2.2 RREQ packet format

The first field contains a constant value (1), which identifies the type of AODV message as a RREQ message. The following five bits are used as follows: J, R – Reserved for multicast; G – Gratuitous reply flag; indicates whether a gratuitous RREP should be sent to the destination if an intermediate hop replied to a RREQ message; D – Destination only flag; only the intended destination is allowed to reply to this message; U – Indicates whether or not the destination sequence number is known

The *RREQ ID* is a unique integer number assigned to the specific request by the originator and is used for further reference. The *Destination Sequence Number* field contains the last known sequence number provided by the destination, and the *Originator Sequence Number* the originator's own sequence number, typically incremented just before this packet is created.

Route Reply (RREP) message

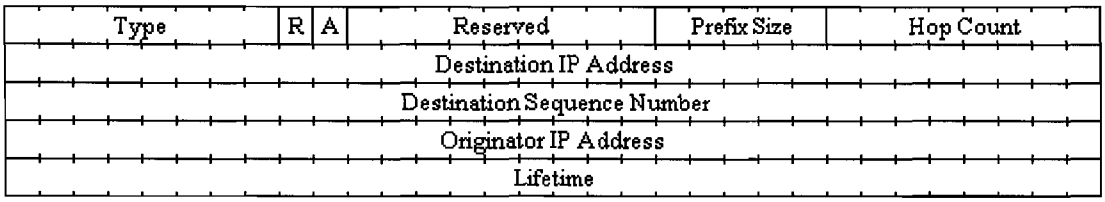


Figure 2.3 RREP packet format

The *Type* field is set to 2 in order to identify this message as a RREP message. Again, *R* is reserved for multicast and *A* indicates that the originator of the message requests acknowledgement. The *Prefix Size* field specifies that the next hop (previous intermediate hop) may be used for any node with the same routing prefix as the requested destination. Other than for a RREQ message, the *Destination IP Address* field specifies the destination to which a route is found, thus, the originator of RREP message. The *Originator IP Address*

field then specifies the intended destination of the RREP massage. *Lifetime* contains the route lifetime and not the packet lifetime.

Route Error (RERR) packet format

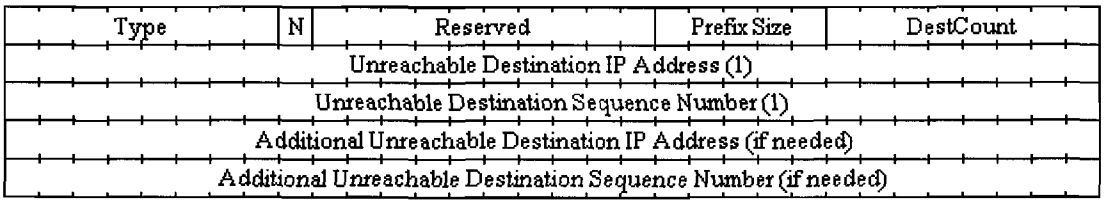


Figure 2.4 RERR packet format

Type is set to 3 for a RERR message. The *N* bit is set whenever the node, which detected the link breakage performs local repair on the link, and nodes should not delete the routes using this link until further notice. The *DestCount* field contains the number of unreachable destinations.

Route Reply Acknowledgement (RREP-ACK) packet format

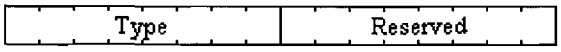


Figure 2.5 RREP-ACK packet format

The *Type* field is set to 4, but this message contains no additional information.

5.4 The Dynamic Source Routing (DSR) protocol [29]

A brief overview on the DSR protocol's operation is given [32]:

DSR is a reactive MANET routing protocol that uses source routing. Therefore, a source node knows the complete hop-by-hop route between itself and every destination it wishes to communicate with, and stores it in its route cache. The source route is also included in the header of every data packet.

As with AODV, whenever a node requires communication with a node to which it doesn't know a route, it dynamically determines such a route by flooding the network with RREQ messages. Every other node receiving a RREQ message rebroadcasts it, unless it is the

intended destination, or an intermediate node with a valid route to the destination. Such a node replies with a RREP message.

As a RREQ message is propagated through a network, it records the hop-by-hop route traversed so far. A RREP message will then use this recorded route in backwards order to find its way back to the originator of the RREQ massage. This route, in its original order, is then stored in the route cache of the originator for future use.

If a link broke down, all source nodes that used the link in one or more routes are notified by using RERR messages. When a source receives a RERR message, it removes all routes that used the broken link from its route cache, and reinitiates the route discovery process if necessary. Furthermore, any intermediate node stores the source route found in the header of a packet it is processing, in its route cache for possible future use.

Some additional optimizations include:
Salvaging: Intermediate nodes are allowed to use an alternative route from its route cache whenever a link over which a packet should be forwarded has failed.
Gratuitous route repair: When a source node receives a RERR message, it piggybacks the RERR message in the next RREQ message to the destination to help notify any other nodes that might be using this link in one or more routes.
Promiscuous listening: If a node overhears a packet which is not intended to be routed via itself, but it can offer a shorter route to the destination, it sends a gratuitous RREP message to the source, advertising this new shorter route. Furthermore, nodes are able to learn about different routes without directly participating in the routing process.

6. Proposed MANET protocols

In this section, an overview is given on a number of proposed MANET routing protocols. This includes QoS protocols, multi-path protocols, load-balancing protocols, as well as a more detailed discussion of AODV-*Multipath*.

6.1 Quality of Service protocols

6.1.1 Quality of Service for Ad hoc On-demand Distance Vector Routing [8]

QoS for AODV [8] is a QoS extension of the AODV protocol, originally proposed in 2000 by the authors of AODV. An operational overview on this initial version follows. Extended versions can be found in [33] and [34].

The authors of *QoS for AODV* claim that the proposed extensions can be used to ensure maximum delay and minimum bandwidth along discovered routes [8]. Mobile nodes are now allowed to specify the maximum delay and minimum bandwidth that is required of a route, as part of a RREQ message. Whenever a node along such a QoS route detects that the route, or a single link can no longer provide the requested QoS, the source that requested the route should be notified by sending an *ICMP QOS_LOST* [8] message to the source (refer to [35] for more information on ICMP).

The route table entries are appended with the following fields:

- Maximum delay
- Minimum available bandwidth
- List of sources requesting delay guarantees
- List of sources requesting bandwidth guarantees

A *Delay* field is appended to both RREQ and RREP message formats. In the former, this field contains the maximum allowed transmission time between the forwarding node and the destination. In the latter, it contains the current estimate for the cumulative delay between the forwarding node and the destination.

Before forwarding a RREQ message, a node should compare its own NODE_TRAVERSAL_ TIME value [6] to the remaining allowed delay in the *Delay* field, and only rebroadcast the RREQ message if this value is more than its NODE_TRAVERSAL_TIME value. A node should also, before forwarding a RREQ message, subtract its own NODE_TRAVERSAL_ TIME value from the remaining value in the *Delay* field and store the result in the *Delay* field, before continuing the route discovery process as specified in [6].

Whenever a destination replies with a RREP message, the *Delay* field is initially set to zero. Each forwarding node adds its own NODE_TRAVERSAL_TIME to the value in the *Delay* field and stores this delay value in the corresponding route table entry, before forwarding the RREP message. A forwarding node also records the source IP address found in the RREP message header in the *list of sources requesting delay guarantees* in order to be able to notify them in case a change in delay provided by the route is detected.

As for delay, a *Bandwidth* field is appended to both RREQ and RREP messages, indicating the requested bandwidth in a RREQ message, and the available bandwidth in a RREP message. An intermediate node should only forward a RREQ message if its available link capacity complies with the requested value in the *Bandwidth* field, otherwise, the packet should be discarded.

For a RREP message, the available bandwidth is initially set to infinity. Each intermediate node then compares its own available link capacity with the value in the *Bandwidth* field, and set the field to the minimum of the two values. Its route table is also updated with the bandwidth value before forwarding the RREP message. As for delay, each forwarding node records the source IP address found in the RREP message header in the *list of sources requesting bandwidth guarantees* in order to be able to notify them in case a change in bandwidth provided by the route is detected.

Two main drawbacks of this version are:

- The NODE_TRAVERSAL_TIME value is only a conservative estimation of the average one hop traversal time [6] and may not always be trustworthy since queuing delay may vary dramatically, depending on network congestion, amongst others.
- The ICMP QOS_LOST message only notifies node that delay has increased or bandwidth has decreased [8], but no specific information is provided.

6.2 Multi-path protocols

6.2.1 Ad hoc On-demand Distance Vector Multipath Routing [9]

Ye et al. [9] proposed an extension to the AODV protocol called AODV-*Multipath* or AODVM. The objective of this study was to design a protocol that will find multiple node-disjoint paths in the route discovery phase. The idea is to transmit redundant data over these

multiple paths in order to guarantee reliability. The receiver should be able to reconstruct the data even if more than one node fails simultaneously.

The authors found that the number of node-disjoint paths between a source and its destination is dependent on the node density of the ad-hoc network, as well as the distance between the source and its destination. This limitation forced them to rely of the concept of *reliable nodes*. These nodes would have more sophisticated antenna arrays, processing capabilities, security and robustness.

A drawback of the abovementioned concept is that the total random nature of MANETs is limited by assigning *reliable nodes*. By doing this, some degree of certainty and structure, which is not a typical characteristic of MANETs, is assigned to the network. The random and dynamic nature is one of the main issues, which make routing intricate in MANETs, and can thus not be neglected in the quest for finding proper routing solutions.

6.2.2 Ad hoc On-demand Multipath Distance Vector Routing [14]

In [14], the authors also extended the AODV protocol to a multi-path protocol (AOMDV). Even so, their algorithm determines link-disjoint paths and not node-disjoint paths as in [9]. Their focus was to investigate the fault tolerance property of multi-path routing and not the load balancing property.

6.2.3 Adaptive Multi-path On-demand Routing [13]

The *Adaptive Multi-path On-Demand Routing* (AMOR) is proposed in [13]. This is an extension of the DSR protocol that discovers multiple node-disjoint paths. The metric for path selection is 'transmission reliability' instead of minimum-hop count. This extension to the DSR protocol has the following results (compared to DSR): Additional control overhead during the route discovery phase, better throughput characteristics and smaller average end-to-end delay in high mobility scenarios when path disruptions are frequent.

6.3 Load balancing protocols

6.3.1 Load-Balanced Ad hoc Routing [3]

Hassenhein and Zhou proposed a load balancing routing protocol for MANET called *Load-Balanced Ad hoc Routing* (LBAR) in [3]. In this protocol, a cost function is used to find a

path with the least traffic. The packets can then be transmitted optimally to the destination while balancing the load over the network. The following is used in the cost computation:
Activity – The number of active paths through a node; and
Traffic interference – The sum of all activity of neighbouring nodes on a node.

LBAR does not balance the load of a single session between a source and destination pair over multiple paths. The total network load however, is balanced by first calculating the cost of a path before it is used. Lower end-to-end delay and higher packet delivery fraction than DSR and AODV are achieved by LBAR.

6.4 The AODV-Multipath (AODVM) routing protocol [9]

A large part of this research is based on the AODVM protocol thus; this section gives a detailed description of the operation of AODVM. Unless differently specified, the *source* refers to the node requesting a route to another node, and the *destination* refers to the node to which the route is requested.

Unlike AODV, intermediate nodes are now required to record information found in multiple RREQ messages in a table called the RREQ table (Figure 2.6 (a)). Upon receiving a RREQ message, an intermediate node records the originator of the RREQ message's identity, the destination's identity, the identity of the neighbour from which the intermediate node received the RREQ message, as well as the hop count, in the RREQ table. In AODVM, intermediate nodes receiving a RREQ message, are not allowed to generate and send a RREP message to the originator of the RREQ message. Only the destination is allowed to do so.

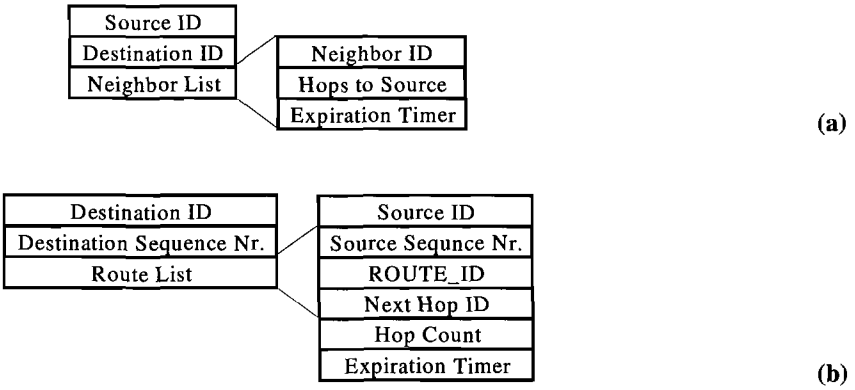


Figure 2.6 (a) Route Request Table Entry (b) Route Table Entry [9]

When the first packet of a specific route request is received by the destination, it updates its sequence number and generates a RREP message. An additional field, called the *ROUTE_ID* field, is added to the RREP message format of AODV. When a destination node receives multiple RREQ messages from the same request, a unique identifying number is assigned to each of these RREQ message and stored in this field of the associated RREP message. The unique RREP message is the forwarded to the originator of the RREQ message, via the neighbour from which the destination node received the RREQ message.

Whenever a RREP message is returned to a requesting node, each intermediate node receiving the RREP message updates its routing table (Figure 2.6 (b)) with the path to the destination node (originator of the RREP message). It then selects the shortest route to the source from the RREQ table, forwards the RREP message along that route, and deletes the RREQ table entry corresponding to the next hop neighbour in that route.

When a node receives a RREP message and has no corresponding entry in the RREQ table left, it generates a Route Discovery Error (RDER) message and sends it to the neighbour that forwarded the RREP message. The neighbour receiving the RDER packet will then attempt to forward the RREP message via an alternative route to the source node. The source should reply to each received RREP message with a Route Reply Confirmation (RRCM) packet, since the destination is unaware of how many of the generated RREP messages will reach the source.

Sequence numbers are used in AODVM as in AODV [6].

7. Chapter closure

The literature set out in this chapter not only serves as the theoretical framework for the work to follow, but also contextually structures previous research and emphasizes the importance of the current study. Furthermore, it forms the corner stone of the proposed work's design, which is presented in chapter 3.

Chapter 3 – Conceptual Design

This chapter explains two new extensions of the AODV protocol. The first extension, considers delay requests that are assigned to application packets, when routing decisions are made. The second extension adds multi-path route discovery to the first extension, allowing nodes to store and use multiple paths to a destination.

1. Delay-aware AODV (DA-AODV) [5]

1.1 The Concept

In our first proposed extension of the AODV protocol, applications are allowed to request a maximum allowed route delay to the destination node it needs to communicate with. Thus, an available route should only be used if the provided end-to-end delay complies with the request of the application.

1.1.1 Data classification

To classify the type of application data encapsulated in an IP packet, the Type of Service (ToS) field in the IP header is used. The default values of 160 for Interactive Multimedia (video conferencing) and 192 for Interactive Voice (Voice-over-IP) are used.

1.1.2 Route discovery

Whenever an application packet is ready for transmission and no valid route is available, the route discovery process is initiated as with AODV [6]. The RREQ and RREP messages are now modified to contain a *Delay* field as proposed in [8]. The difference is that both RREQ and RREP messages use this field to store the cumulative delay up to the node currently processing the packet. Furthermore, nodes do not use the value for NODE_TRAVERSAL_ TIME as proposed in [8], to append to the cumulative delay, but rather use the actual measured delay between the previous hop, and the node itself. In practice, a good estimate for measuring link delay would be to take only the processing and queuing delay at a node since propagation delay is significantly small compared to this, especially in congested networks.

As a result of the abovementioned, intermediate nodes can store routes, with the delay provided by them, during the request phase, as well as the reply phase. Also, since the end goal includes multi-path routing, discovered routes that do not comply with the specific request may still be usable for other traffic types with less strict delay requests. Even though a route, which does not comply with the requested route delay, may be set as the active route, the queued packets that initially requested a route with a specific delay will not be sent until a route, which complies with the request, is discovered. This might be less efficient regarding route discovery time, but by keeping multi-path routing and servicing various traffic types with different delay requests in mind, this will be the more efficient method at the end.

Whenever a new route is discovered and one or more packets are queued for the specific destination (waiting for a route to be discovered), the delay requested by each packet is compared with the delay of the discovered route. A packet is only transmitted if the route delay is the same or less than the requested delay. Once a route is discovered, it is also stored in a common route table on the IP layer. Thus, if packets arrive from the application layer and a route is available from this common IP route table, the route will be used, regardless of the delay provided by it. The route discovery process is thus trusted to provide feasible routes to the IP layer, although it might not always be the case, e.g. when the first discovered route does not comply with a specific delay request, but is still stored in the common IP table (refer to the next section and section 2.1.2 of Chapter 7). To evaluate the request of every packet arriving from the application layer will require cross layer design, which is not included in the scope of this study.

1.1.3 DA-AODV route updating

Once the first route to a destination is discovered, it is stored in the route table, no matter the delay of the route. Nevertheless, queued packets will not be transmitted if the route delay does not comply with their requests. If new routes to the destination are discovered, a modified version of the AODV route update rule described in Chapter 2, determines whether or not the new route is preferred above the existing route. The metrics considered in the AODV route update rule, which is primarily route freshness and secondly minimum hop count if routes are equally fresh, are replaced with minimum delay and route freshness on equal basis. Thus, a new route will not be used if it has a higher delay than the existing route.

The modified route update rule is as follows:

if $((seq_nr^d_j >= seq_nr^d_i)$ **and**
$(route_delay^d_j < route_delay^d_i))$
then
$seq_nr^d_i := seq_nr^d_j;$
$hop_count^d_i := hop_count^d_j + 1;$
$next_hop^d_i := j;$
$route_delay^d_i = route_delay^d_j;$
endif (3-1)

The notation in the DA-AODV route update rule applies as for the AODV route update rule (section 5.3.2 of Chapter 2), with the additional route delay between node i and destination d represented by $route_delay^d_i$.

1.1.4 Route maintenance

As in AODV, each node that is part of at least one active route, broadcasts a *Hello* message locally every HELLO_INTERVAL milliseconds to offer connectivity information [6]. The format of a RREP message is used for *Hello* messages. The *Delay* field in the extended RREP message is also used in *Hello* messages to determine link delay. Upon receiving a *Hello* message, a node compares the link delay between itself and the node that originated the *Hello* message, with a delay threshold that is permitted per link. If the delay exceeds the threshold, the QoS provided by the link is no longer acceptable and the route table entry to the originator of the *Hello* message is set as invalid. Since the link is now marked as invalid, all nodes that used the link in an active route are notified that the route is no longer functional. This is done by the default AODV route error process [6].

A more elegant solution would be to inform all nodes using the specific link in an active route of the precise change in delay, but this would require more extensive coding. It is important to find a compromise between simplicity and efficiency, which may require more complexity. The fact that the traffic types of interest have the same delay requests and the network size is limited, simplifies the matter. Once services with different requests are running on the same

network, assigning a threshold to the delay permitted per link would be less efficient. The second proposed solution (delay change notification) would be preferred in such networks.

1.1.5 Local repair

Chapter 2, section 5.3.3 explains the concept of local repair in AODV. The criterion to repair a route is that the hop count of an alternative route should be the same, or less than that of the broken route. In DA-AODV, hop count is no longer the main metric. The condition is now set that a route discovered during local repair will only be used if the route's delay is lower than a specified value.

1.1.6 Packet queues

A final extension is made to AODV to ensure that packets, queued for route discovery, are dropped if they are queued for longer than a maximum accepted delay period, typically 400 milliseconds for real-time traffic [25].

1.2 Data Structures

1.2.1 Packet formats

In order to be able to record route delay as routes are discovered, an additional *Accumulated Delay* field is appended to the RREQ and RREP messages. A second field, called *Previous Transmit Time*, is also appended to the RREQ and RREP messages. Individual link delays can then be calculated by subtracting this value from the current time, assuming that the time at all nodes is synchronized. Figure 3.1 illustrates the new defined packet formats.

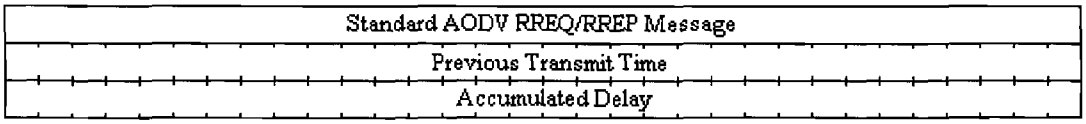


Figure 3.1 DA-AODV RREQ/RREP packet format

1.2.2 Table formats

The standard AODV route table is extended to contain a *route_delay* field in which the total cumulative delay between the node itself, and the destination node for which the route entry exists, is stored. The request table is kept and used as in AODV [6].

1.3 DA-AODV flow diagrams

The flow diagrams in this section represent the protocol flows that are edited in order to enable AODV to consider delay requests of applications. *FIN* represents the entry point of a function or program section, and *FOUT* the exiting point. All modifications to AODV are highlighted in orange.

Figure 3.2 illustrates the routine by which a packet is handled once it is handed over to DA-AODV by the IP layer. This happens whenever a DA-AODV packet arrives at the IP layer or if a non-DA-AODV packet arrives at IP layer and no valid route information for its intended destination exists in the common IP route table.

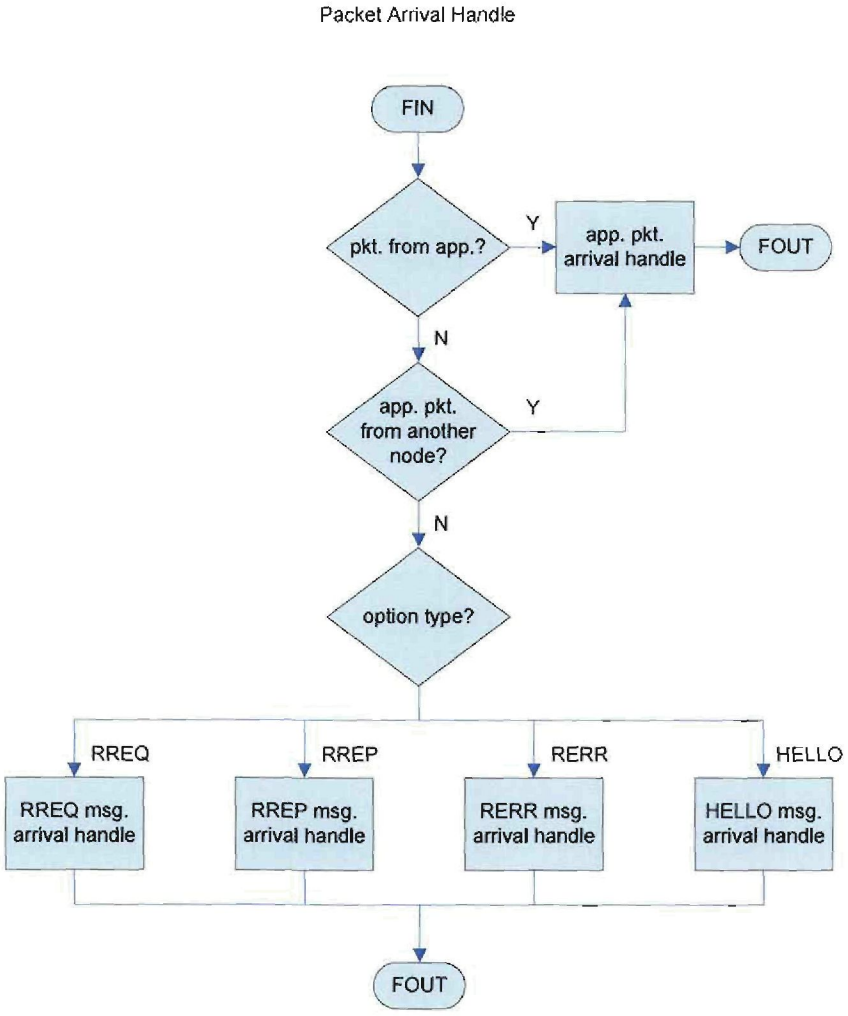


Figure 3.2 General packet arrival handle flow

First, it is determined if the packet contains application data and if so, the packet is handed over to an *application packet arrival handle* (Figure 3.3).

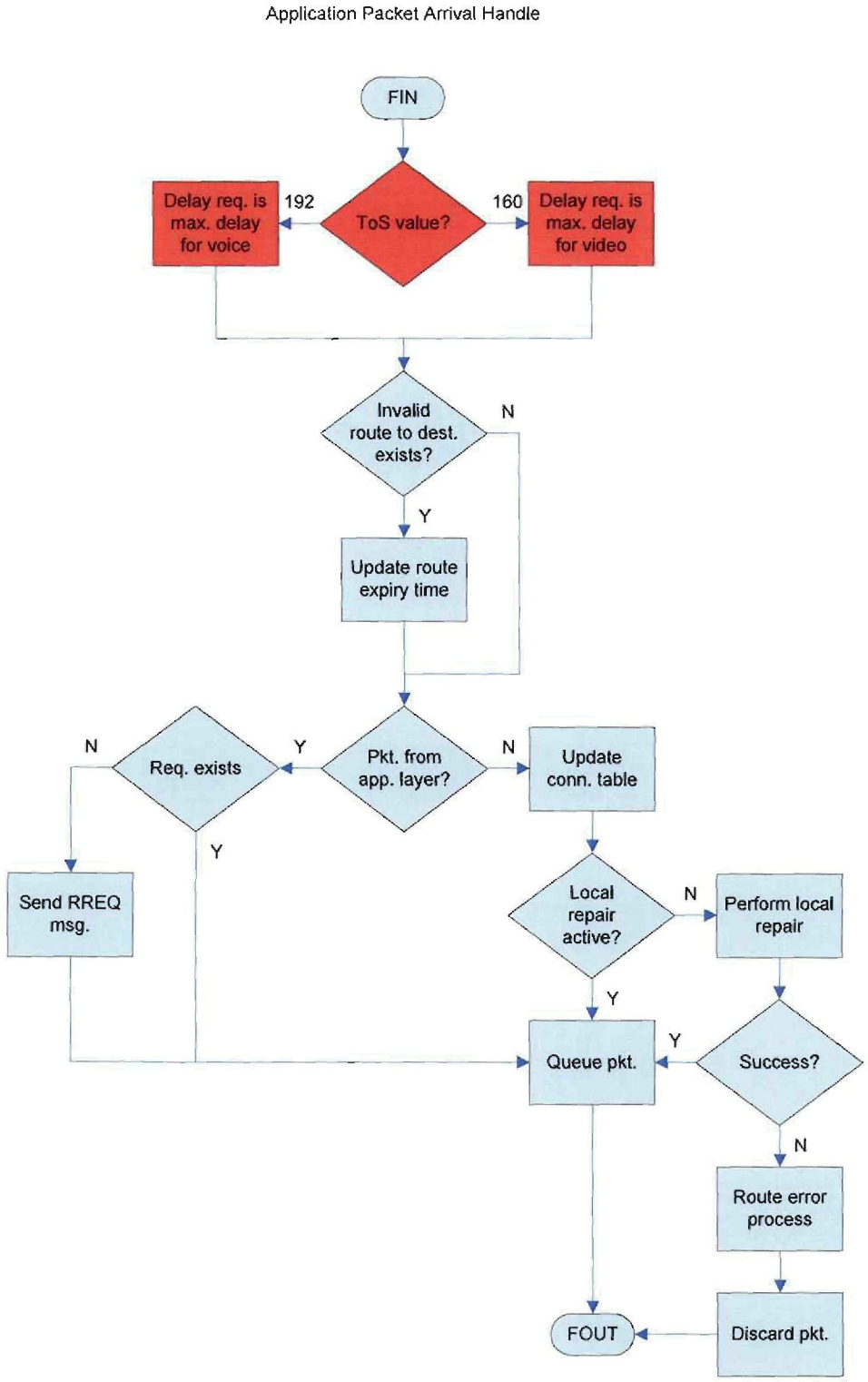


Figure 3.3 DA-AODV Application packet arrival handle flow

Upon the arrival of an application packet, the ToS field in the IP header is read and a delay request is assigned to the packet by means of a lookup table. If the packet arrived from a higher layer, the node is the originator of the packet and a RREQ message is broadcasted if one has not been broadcasted for the specific destination within a specified period of time. The packet is then queued until a route is discovered.

If the packet arrived from a lower layer, connectivity information is updated and local repair is attempted. If local repair is active or can be successfully initiated, the packet is queued, otherwise, the route error process is initiated and the packet is discarded.

If the packet is a DA-AODV message, it is handed over to the applicable packet arrival handle (Figures 3.4 – 3.6).

The first of the DA-AODV message handles is that of a RREQ message (Figure 3.4). The first step when handling a RREQ message is to update the cumulative delay. After updating connectivity information, the node checks if the source address belongs to itself and if it already received this request. This is to help prevent route loops. If the packet should be processed further, an entry is inserted in the forward request table and the hop count is incremented. If the maximum specified hop count is reached, the packet is discarded. Again keeping in mind the servicing of multiple traffic types, less delay sensitive traffic types may benefit from routes that do not necessarily provide low delay guarantees, and thus, the criterion is not changed to delay.

Next, if an entry for the reverse route does not exist in the route table, it is created and all packets queued for the source, are sent (Figure 3.7). If an entry exists with the same *next hop*, the route update rule (section 1.1.3) is evaluated to determine whether the new route's metric is lower. If not, and the next hop on the reverse route differs from the previous hop via which the RREQ message was received, the RREQ message is discarded. This is done to prevent route loops in certain cases [36]. If the new route's metric is lower, the route table entry is updated.

The final step is to determine if the node processing the packet is the intended destination. If so, a RREP message is sent to the source, otherwise, the RREQ message is forwarded.

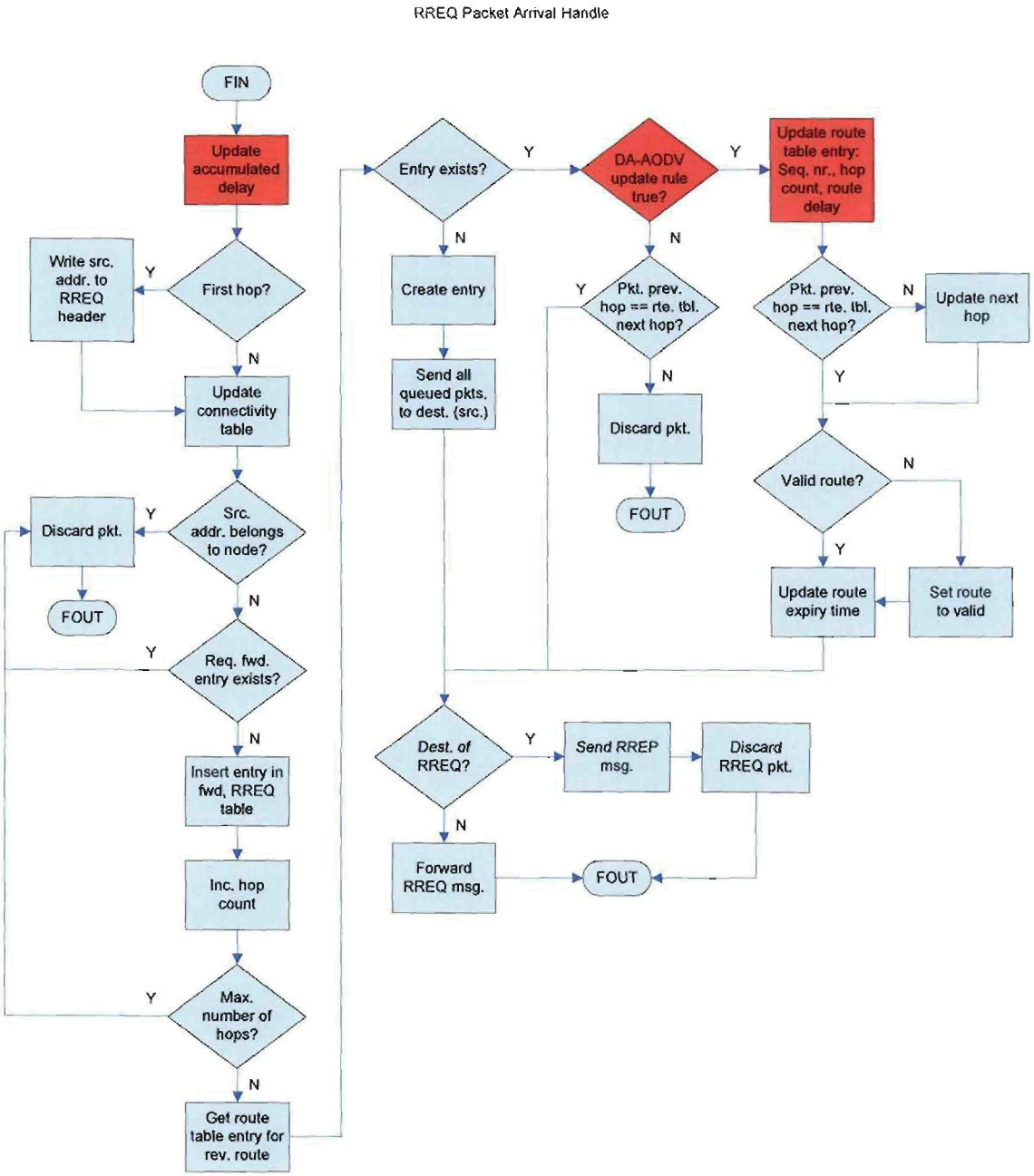


Figure 3.4 DA-AODV RREQ packet arrival handle flow

The first step when handling a RREP message (Figure 3.5) is also to update the cumulative delay. If it is verified that the source address is not the node's own, connectivity information is updated and hop count is incremented. Next, the hop count is verified to be less than the specified allowed maximum. If a valid route does not exists to the destination, it is created, and all packets queued to the destination is sent, as illustrated in Figure 3.7.

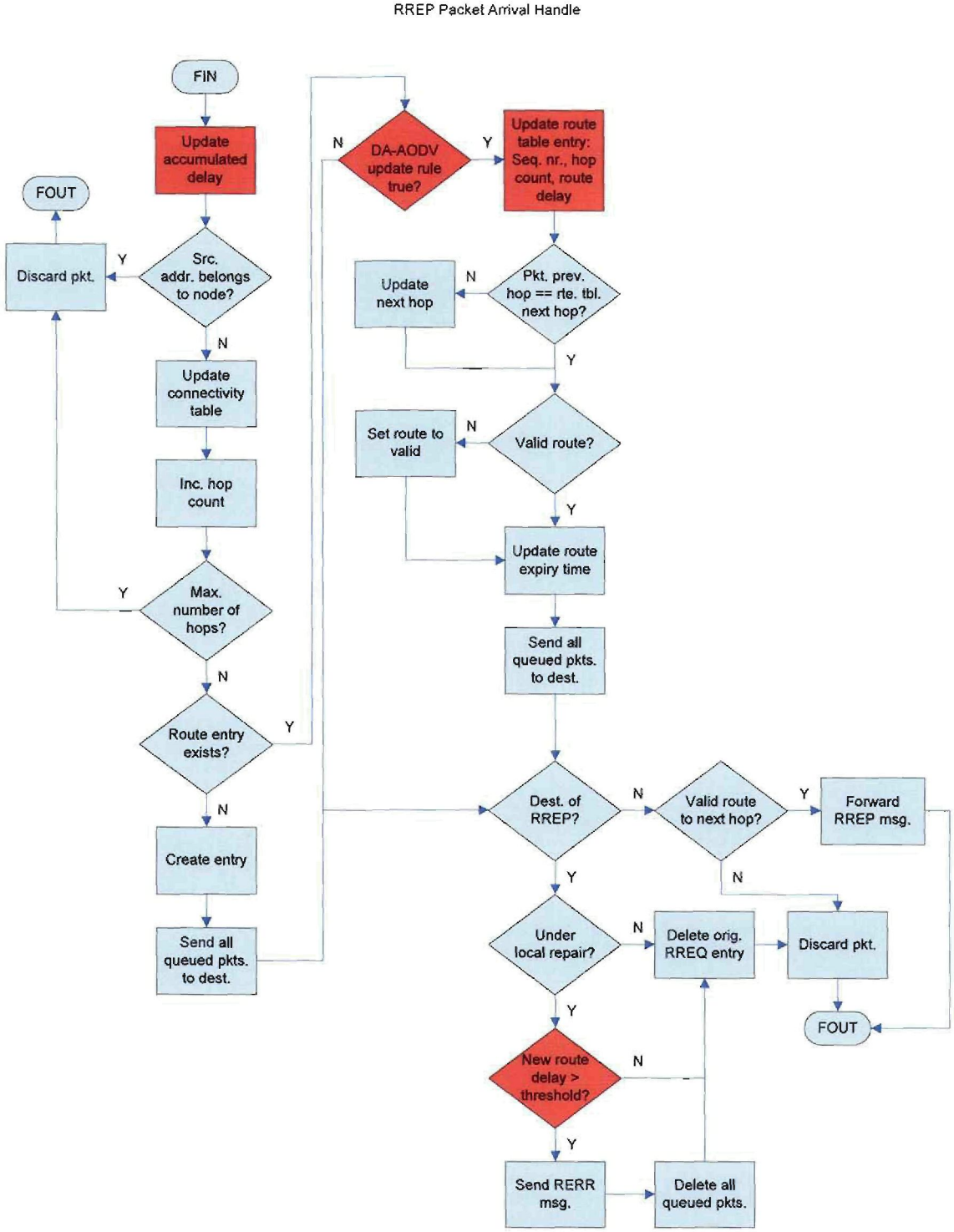


Figure 3.5 DA-AODV RREP packet arrival handle flow

If a valid route exists, the route update rule is evaluated to identify the route with the better metrics between the existing and the new discovered route. If the node is not the intended destination of the RREP message and it has a valid route to the next hop in the reverse route,

it forwards the packet. If the node is the intended destination, the entry in the originator request table is deleted and the packet is discarded. If the route was under local repair, and the new route does not comply with the previous route, a RERR message is sent. A flow diagram for the *RERR packet arrival handle* is not illustrated, as no alterations are required.

In Figure 3.6, showing the flow for handling a *Hello* message, the link delay is calculated and compared with the specified threshold value. If it exceeds this value and an entry for the route exists, it is set to invalid. If the link delay complies with the specified threshold, the route and connectivity tables are updated.

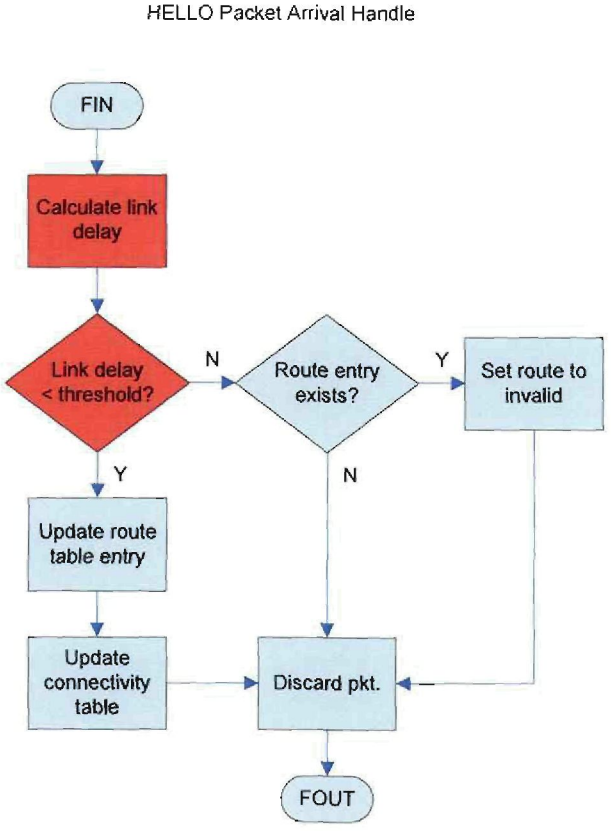


Figure 3.6 Hello packet arrival handle flow

Figure 3.7 illustrates the flow for a routine to send all queued packets to a specific destination. The routine for each queued packet is as follows: The queued time is compared with a threshold value (for delay sensitive packets), and the packet is discarded if the threshold is exceeded. If not, a delay request is assigned according to the ToS value and compared with the route delay to determine if the route is suitable for the packet. If not, the packet is again inserted into the queue.

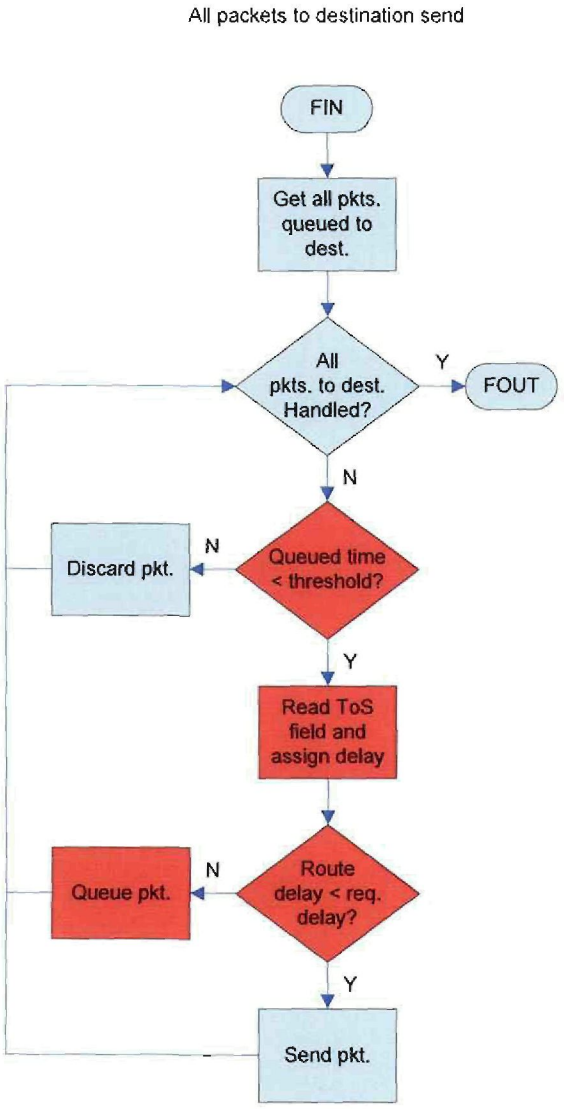


Figure 3.7 All packets to destination send flow

2. Delay-aware AODV-Multi-path (DA-AODVM)

2.1 The Concept

In our second proposed extension, DA-AODV is combined with a simplified implementation of AODVM [9] in a new protocol called Delay Aware AODV-Multi-path or DA-AODVM. As in AODVM, intermediate nodes forwards multiple RREQ messages containing the same source address and request ID. The only exception is where a RREQ message with a specific request ID has already been received from the same previous hop. The RREQ table is managed as in AODVM. Upon receiving a RREP message, an intermediate node searches for the reverse route with the smallest hop count in the RREQ table [9]. Once a next hop is

selected, the entire entry of the specific source-destination pair is deleted from the RREQ table. This prevents the node from forwarding multiple RREP messages, in order to assure that a node does not participate in multiple paths. If no reverse route is available, the RREP message is discarded.

If a RREP message reaches the node that originated the RREQ, the new discovered route is stored in the route table, together with the route ID and route delay. If a route that uses the same next hop address already exists, a refined version of the DA-AODV route update rule is evaluated in order to decide whether or not the new route should be used instead of the existing one. This refined version states that if the new route's sequence number is the same as the existing one, but the new route provides lower delay, or the new route is fresher, and its delay is lower than the requested delay, the new route is selected over the existing route:

if $(((seq_nr^d_j == seq_nr^d_i)$ **and** $(route_delay^d_j < route_delay^d_i))$
or $((seq_nr^d_j > seq_nr^d_i)$ **and** $(route_delay^d_j < delay_request)))$
then
$seq_nr^d_i := seq_nr^d_j;$
$hop_count^d_i := hop_count^d_j + 1;$
$next_hop^d_i := j;$
$route_delay^d_i = route_delay^d_j;$
endif (3-2)

Before the route discovery process is initiated as in AODV and DA-AODV, DA-AODVM first searches the route table for alternative routes, which also complies with the delay request of the application. If an alternative route is found, it is used; otherwise it initiates a new route discovery.

On the multi-path side of the protocol, the following issues still need to be addressed:
The effective updating of routes (currently only active routes are updated) and; in the route reply phase, the previous hop nodes need to be notified if no reverse route is available and alternative reverse routes need to be established.

Chapter 7 addresses these and more issues thoroughly.

2.2 Data Structures

2.2.1 Packet formats

Packets, as defined for DA-AODV, are used. One alteration is made: A *route_id* field, as proposed in [9], is added to the RREP message.

2.2.2 Table formats

For the multi-path extension of DA-AODV, the route table entry is modified to the format illustrated in Figure 2.6 (b). Thus, for each destination (a single route table entry), a list of possible routes to the destination, called the route list, is stored. The *route_delay* field (section 1.2.2) is thus now part of each route list entry, instead of the route table entry as in DA-AODV. An additional field (*list_pos*) used to hold the entry's position in the list, is also included in the route list entry. The same entry/list concept is used in the *forward request table* to store multiple requests' information for a single destination.

2.3 DA-AODVM flow diagrams

The flow diagrams in this section represent the protocol flows in extension to the flows of DA-AODV in section 1.3. The *general packet arrival handle* flow, *Hello packet arrival handle* flow and *all packets to destination send* flow are the same as for DA-AODV. Again, all alterations to AODV are highlighted in orange.

In the *application packet arrival handle*, the first alteration to DA-AODV is just after a delay request is assigned. The route table entry, now holding multiple routes, is searched for an alternative route. If one is found, it is set as the new active route to the destination and the rest of the routine is skipped. If not, but an invalid route to the destination is known, the expiry time is updated. The rest of the routine continues as in DA-AODV.

Updating the request table when a RREQ message arrives differs for DA-AODVM since information of multiple requests needs to be stored (Figure 3.9). If a forward request table entry already exists for the destination, an entry for the neighbour from which the packet is received is added to the neighbour list in the entry. If no entry exists, an entry is created before the neighbour entry is added in the neighbour list. Again, if no reverse route exists, a new route table entry is created, and then the new route is inserted in the route list.

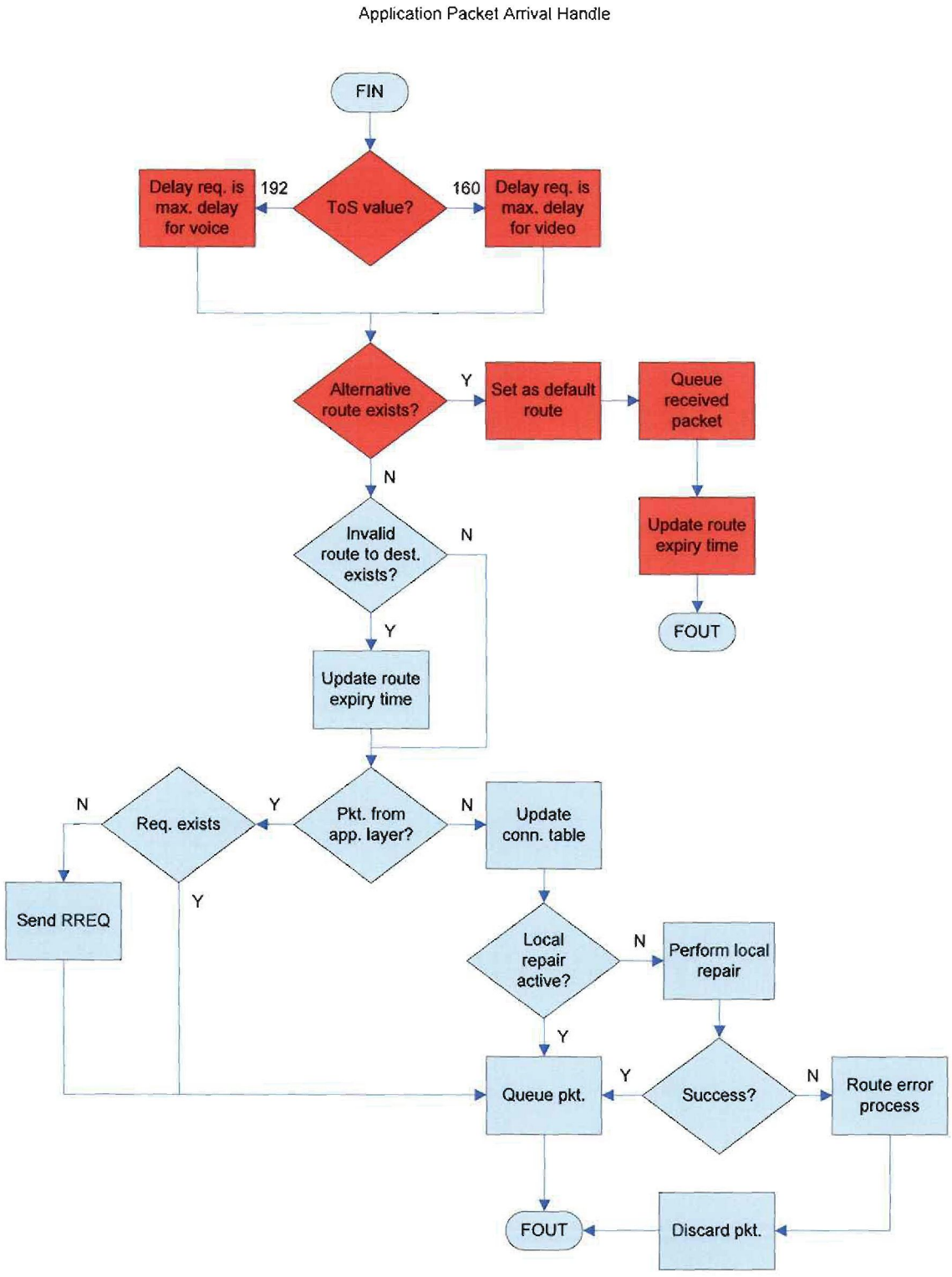


Figure 3.8 DA-AODVM Application packet arrival handle flow

If an entry exists with the same *next hop* than from which the RREQ message was received and the route update rule is true, the existing route is updated. Information from RREQ

messages is not used to store multiple paths to keep intermediate nodes from participating in multiple paths between a source and a destination.

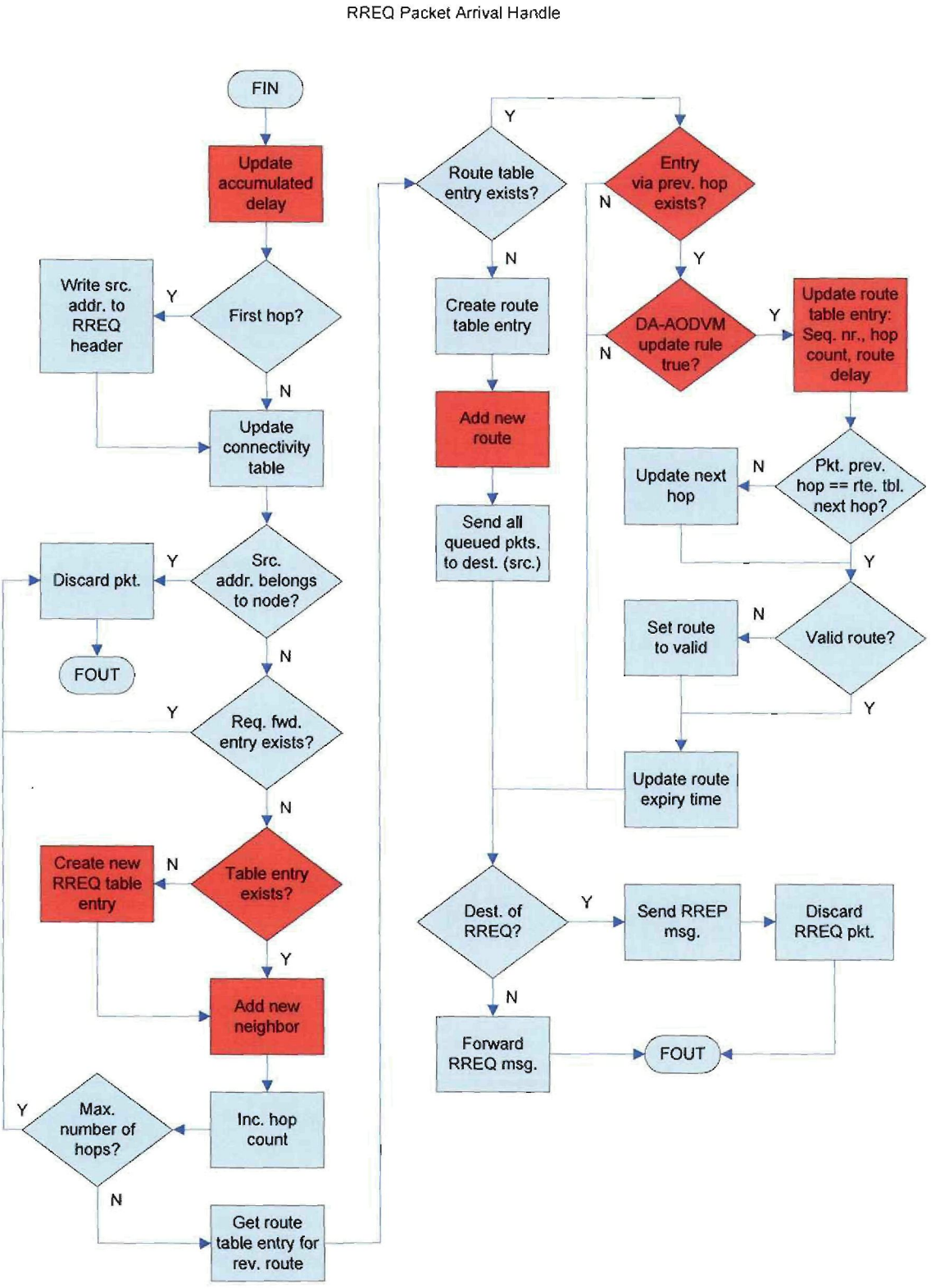


Figure 3.9 DA-AODVM RREQ packet arrival handle flow

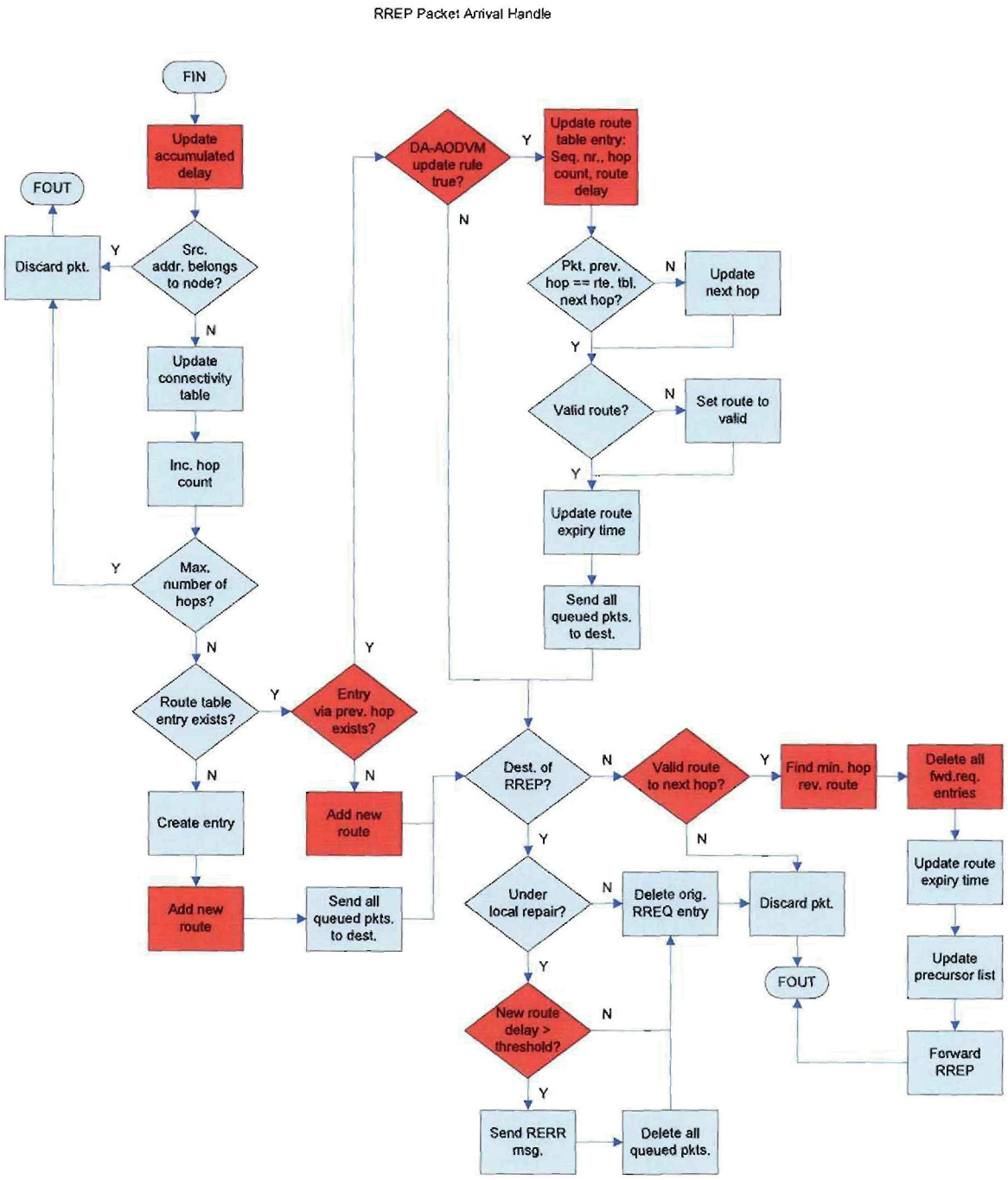


Figure 3.10 DA-AODVM RREP packet arrival handle flow

For a RREP message (Figure 3.10), inserting a new route table entry and updating an existing one is the same as for a RREQ message. If the new route is via a *next hop* other than any of the existing routes, the new route is inserted in the route list.

If the processing node is not the source, the forward request table and not the route table as in AODV and DA-AODV, is consulted to find the route with the smallest hop count back to the

source [9]. All entries for the specific request are then deleted from the table to prevent a node from forwarding multiple RREP messages for the specific request. Finally, the node's precursor list is updated and the packet is forwarded. If the node is the destination of the RREP message, the routine continues as for DA-AODV.

3. Chapter closure

In this chapter, the conceptual designs of the proposed extensions to AODV were discussed. Primarily, only the modifications to the AODV protocol were considered and anything not discussed is in accordance with the AODV protocol specification [6]. This includes protocol operation and data structures. The design is implemented in *OPNET Modeler* simulation environment, and the implementation will be discussed in the next chapter.

Chapter 4 – Implementation

This chapter shows the implementation of the proposed protocol extensions. It introduces OPNET Modeler and explains relevant OPNET kernel procedures. It also discusses the method to create a new protocol in OPNET, and the implementation of the extensions to AODV.

1. Introduction

OPNET Modeler [28] uses a very structured approach in modelling nodes, networks and network processes. The operation of a wireless node for example is divided in multiple layers. This approach makes it easy to edit components in a layer of the protocol stack, without affecting the other. The graphical interface of this layered approach is illustrated in Figure 4.1. Each block in the node model is called a *processor* [28], since it has a certain process running underneath. The red and blue arrows between the processors are called *packet streams*.

Figure 4.2 illustrates the typical graphical interface to a process model. The specific process model is that of the AODV protocol. It is classified as a *child process*, since it is called by the *manet_rte_mgr* process (the *parent process*) in the *wlan_wkstn_adv* wireless node model.

OPNET uses Proto-C; a C/C++-based language. A wide variety of kernel procedures are available, which helps making communication system oriented programming easier. This includes procedures for creating and managing hash tables and lists, allocating and managing physical memory, creating and managing data packets.

The process models in OPNET are state flow diagrams, containing code blocks that are executed as data pieces flow through different states. Code can be written to be executed at the entry and exiting of each state. Any state can be in a *wait* phase where it remains until it is "woken up" by an interrupt.

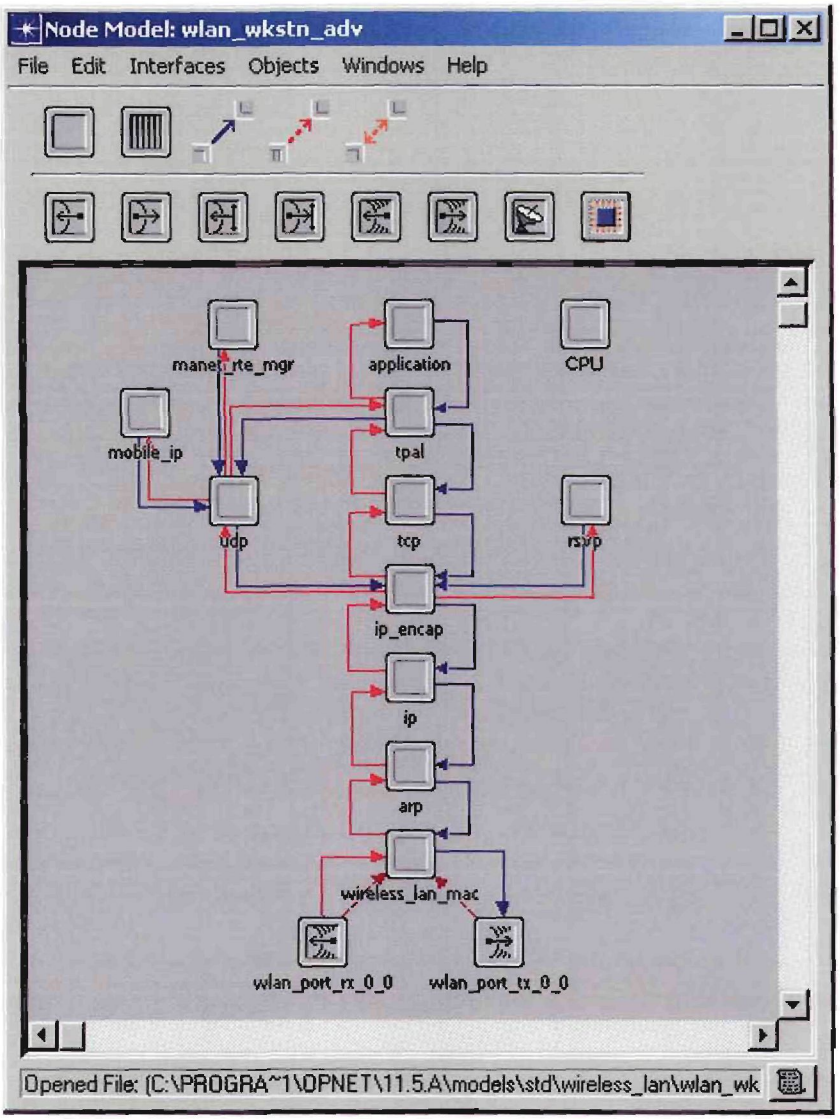


Figure 4.1 Layered node model

The main code is written and stored in the *Function Block* (FB). Code is mainly functional and internally callable by other functions of the same process. Defining a function as *extern* enables any external function declared in the project, to call the function.

Furthermore, any function in an external code file can be called from the process model functions, as long as the external file is declared in process model as an external file.

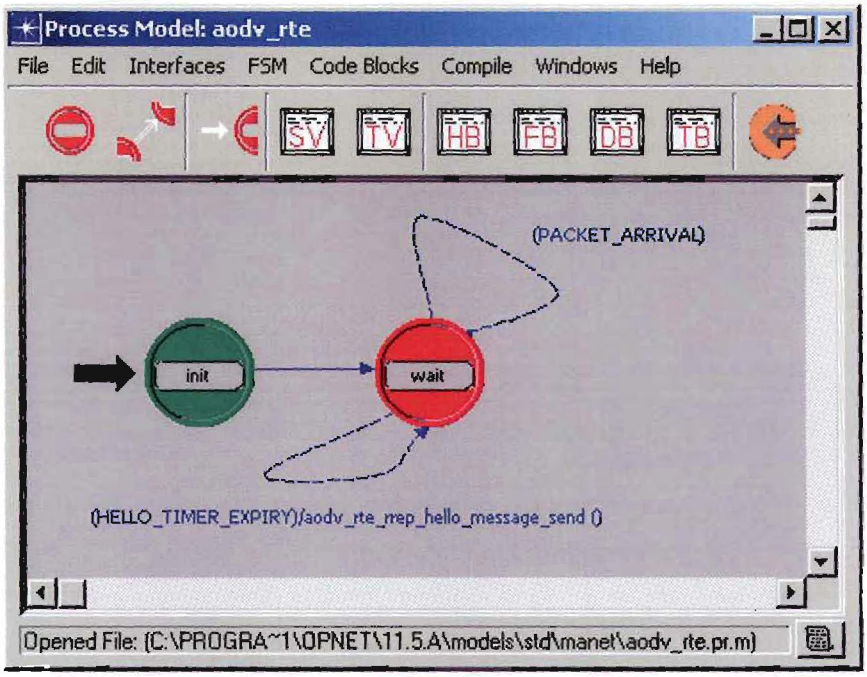


Figure 4.2 AODV process model

2. Important OPNET kernel procedures [28]

This section explains the kernel procedures that were used to implement the proposed protocol extensions. It is assumed that the reader is familiar with standard C language, thus the discussion includes only function and procedures unique to Proto-C.

2.1 Lists

A *List* is an OPNET defined data type, containing multiple values of a single data type. OPNET provide kernel procedures for creating, editing and deleting lists. List entries are pointers to data in memory. Thus, simple data types, as well as complex data structures can be inserted in a list.

op_prg_list_create ()

Creates an empty list, allocated by the simulation kernel, and returns a pointer (*List**) to the new created list.

op_prg_list_insert (list_ptr, element_ptr, pos_index): Inserts a pointer to a data element in the specified list, at the specified position. *list_ptr* is a pointer to the list-of-interest, *element_ptr* is a pointer to the element to be inserted, and *pos_index* the position in the list at which the new element is to be inserted. OPNET constants, *OPC_LISTPOS_HEAD* and *OPC_LISTPOS_TAIL*, can be used as *pos_index*, to insert elements at the head and tail of the list, respectively.

op_prg_list_access (list_ptr, pos_index): Returns a pointer to the element at position *pos_index* in the specified list (*list_ptr*). Again, *OPC_LISTPOS_HEAD* and *OPC_LISTPOS_TAIL* can be used. The return value type is *Vartype**.

op_prg_list_remove (list_ptr, pos_index): Removes the data element at the specified position, from the specified list. The return value is a pointer (*Vartype**) to the removed element.

op_prg_list_size (list_ptr): Returns the size of the specified list (*list_ptr*) as an integer (*int*).

op_prg_list_free (list_ptr): Deletes the specified list and releases associated memory resources.

2.2 Hash tables

OPNET Modeler provides sub-packages of kernel procedures that allow users to easily manage hash tables with binary or string keys. A hash table consists of a data store and a hashing function part (Figure 4.3).

The hash table cells represent the data store and are indexed by a number between 0 and n - 1. A mathematical operation (the hashing function) is performed on the key to return the index [0; n - 1]. When a value is inserted in a hash table, or retrieved from it, the hashing function is used to determine the cell in which the value must be stored, or is stored.

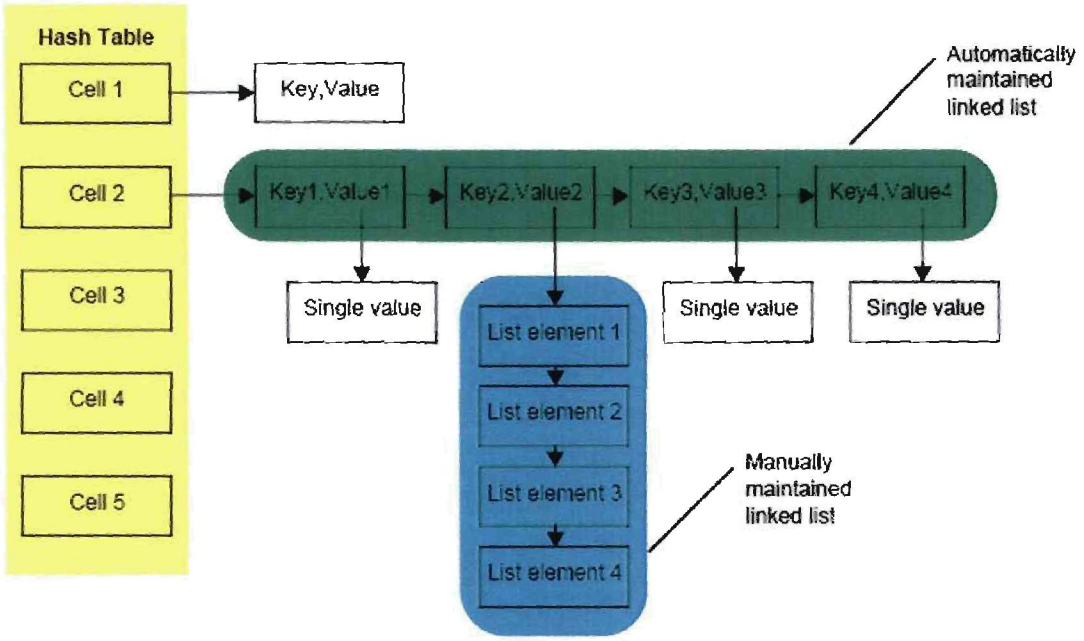


Figure 4.3 Hash table structure example [28]

The following kernel procedures are used to create and manage hash tables in OPNET:

prg_string_hash_table_create (table_size, max_chars): Creates a hash table containing *table_size* cells (*int*) and a maximum key length of *max_chars* (*int*). The function returns a pointer (*PrgT_String_Hash_Table**) to the new hash table.

prg_string_hash_table_item_insert (hash_table_pointer, key_string, contents_ptr, old_contents_pptr): Inserts a new value into the hash table pointed to by *hash_table_ptr*. *key_string* is the string key of the new entry, and *contents_ptr* is a pointer to the value to be inserted. A pointer to the previous value indexed by *key_string*, if any exists, is assigned to *old_contents_pptr*.

prg_string_hash_table_item_get (hash_table_ptr, key_str): Returns the value (*void**) associated with the given key from the hash table pointed to by *hash_table_ptr*.

prg_string_hash_table_item_remove (hash_table_ptr, key_str): Removes the value (*void**) associated with the given key from the hash table pointed to by *hash_table_ptr*. The value, if the key appears in the hash table, is returned (*void**) by the function.

prg_string_hash_table_keys_get (hash_table_ptr): Returns a pointer to a list (*PrgT_List**) of all the keys in the hash table pointed to by *hash_table_ptr*.

prg_string_hash_table_size_get (hash_table_ptr): Returns the number of cells (*int*) in the hash table pointed to by *hash_table_ptr*.

prg_string_hash_table_free (hash_table_ptr): Destroys the hash table pointed to by *hash_table_ptr*, but does not destroy the values in the hash table.

2.3 Data packets

Creating and destroying data packets are made easy by the following kernel procedures:

op_pk_create (bulk_size): Creates an unformatted packet (no fields set) with the specified bulk size. The bulk data size is specified in bits. The return value is a pointer to the new created packet.

op_pk_destroy (pkptr): Destroys (deallocates) the packet indicated by *pkptr* (pointer to the packet-of-interest), releasing associated memory resources.

2.4 Memory allocation

The following kernel procedures simplify the allocation and freeing of memory in OPNET:

op_prg_mem_alloc (size): Allocates a block of memory. The parameter to the function specifies the size in bytes to be allocated. Typically, the pseudo-function *sizeof* () can be used whenever memory needs to be allocated for data structures. The function returns a pointer to the allocated block of memory.

op_prg_mem_free (ptr): Deallocates the block of memory at the address indicated by the parameter *ptr*.

2.5 Other

op_sim_time (): Returns the value of the current simulation time in seconds.

3. Extending the AODV protocol

The first step in extending an existing protocol in OPNET is to create a copy of the protocol. This includes the main protocol process model, packet definitions, as well as all the external code and header files that are declared and used in the process model.

For AODV, files to be copied include: (*.pr.m indicates a process model, *.c a source file, and *.h a header file)

aodv_rte_mgr.pr.m/aodv_rte_mgr.pr.c – Process model and its code file;

aodv_packet_queue.ex.c – External code file containing functions to create and manage packet queues;

aodv_pkt_support.ex.c – External code file containing functions to create and manage AODV related packets;

aodv_request_table.ex.c – External code file containing functions to create and manage request tables;

aodv_route_table.ex.c – External code file containing functions to create and manage route tables;

aodv_support.ex.c – External code file containing text output functions and functions to handle statistics;

aodv.h – Main AODV header file in which constant values, as well as table structures and local statistics handles are declared;

aodv_pkt_support.h – Header file in which packet related constants, as well as AODV related packet structures are defined;

aodv_ptypes.h – Header containing necessary prototypes for external code files.

Furthermore, AODV, when selected as MANET routing protocol, is registered within the IP layer/processor (ip_dispatch.pr.m) as the higher layer protocol. In some cases, decisions made in this processor, as well as in the MANET manager process (*manet_mgr*), are based on the fact that a specific MANET routing protocol (AODV in this case) is used.

Non-AODV files in which are referred to the AODV protocol includes: ip_dispatch.pr.m, manet_mgr.pr.m, ip_cmn_rte_table.ex.c, ip_cmn_rte_table.h, ip_notif_log_support.ex.c, ip_rte_support.ex.c, ip_rte_support.h, ip_higher_layer_proto_reg_sup.h, ip_rte_v4.h.

In these files, for every reference to or consideration of the AODV protocol, the same is implemented for the copy of AODV (called QAODV for all programming related issues) for whenever QAODV is set and registered as the higher layer protocol.

A final change required to use the new copy of the protocol is to register it as an attribute in the *wlan_wkstn_adv* wireless node model's *Node Interfaces*. This enables QAODV to be selected as an *AD-HOC Routing Protocol* (Figure 4.4) in the node's attributes. Also, this enables routing parameters to be set in the node's attributes. In order for QAODV to appear as an *AD-HOC Routing Protocol*, it is registered in the *Model Attributes* of the *manet_mgr*

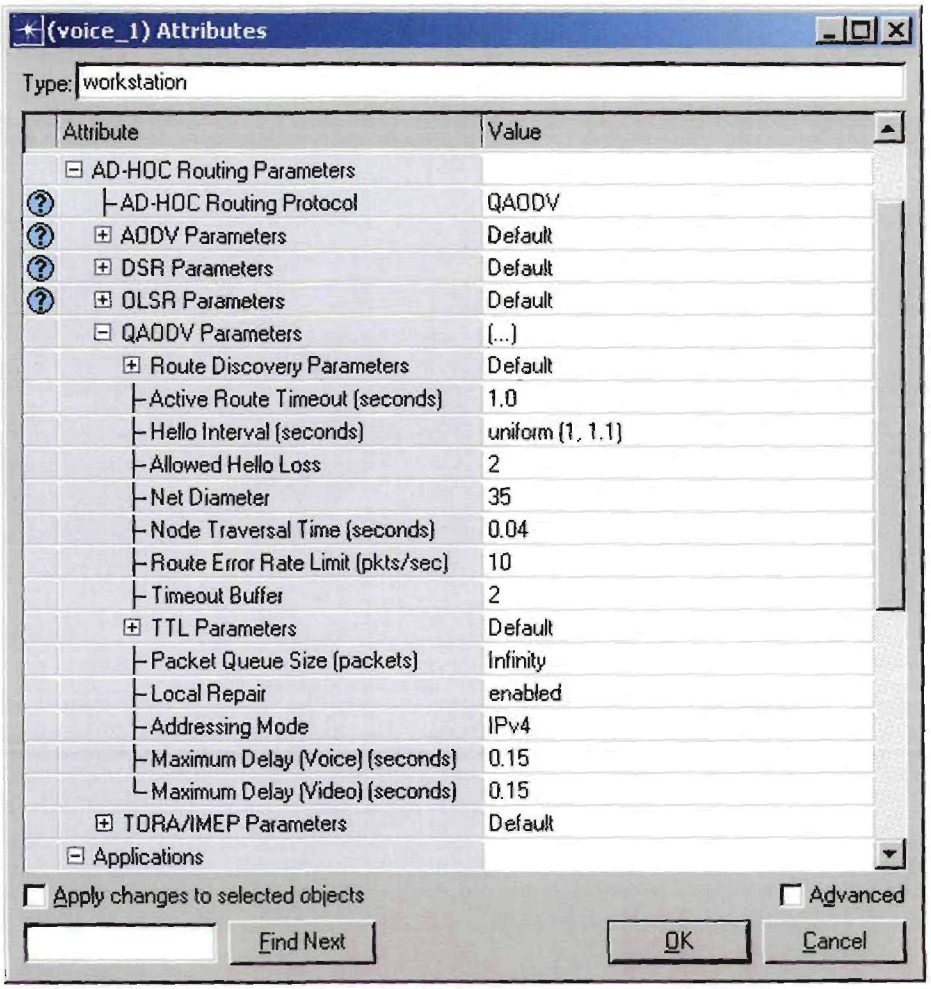


Figure 4.4 Node Attributes

process model, by setting the *Group* field to *AD-HOC Routing Protocol.* This is also where the default values for the routing parameters are set.

Two routing parameters are added: *Maximum Delay (Voice)* and *Maximum Delay (Video).* These parameters are assigned to state variables in the main process model code, which is further used to refer to these values.

4. The DA-AODV Extension

4.1 Extended data structures

Extended data structures, as discussed in section 1.2 of chapter 3 are implemented as follows:

Two fields of type *double* are appended to the RREQ and RREP message structures, defined in **qaodv_pkt_support.h**. These fields are *current_delay/route_delay* and *prev_tx_time* and hold the values of the cumulative route delay, and the time the packet was processed at the previous hop, respectively.

A single change is made to the route table structure, defined in the **qaodv.h** header file. A field (*route_delay*) of type *double* is added to the route table structure to hold the value of a route's delay once it is stored in the route table.

4.2 Edited code pieces

New and modified code pieces are discussed in the sequence it appears in the source files. The code of a single function is discussed at a time. Refer to section 3 of this chapter for the description of each code file.

Code file: qaodv_rte.pr.c

qaodv_rte_app_pkt_arrival_handle (Figure 3.3): (application packet arrival handle)

The first added code piece is to determine the type of data contained in an application packet when it is received, and to assign a delay request accordingly. This is done by reading the ToS field from the IP header. A pointer to the header fields of the received packet is received as parameter (*ip_dgram_fd_ptr*), and the code is as follows:

```
if (ip_dgram_fd_ptr->tos == 160)
        {
        req_delay = max_delay_video;
        }
else if (ip_dgram_fd_ptr->tos == 192)
        {
        req_delay = max_delay_voice;
        }
```

qaodv_rte_rreq_pkt_arrival_handle (Figure 3.4): (RREQ packet arrival handle)

The first added code to this function is to calculate the link delay between the node itself and the previous hop of the RREQ message. This value is stored in the cumulative delay field (*current_delay*) in the RREQ message header, and *op_sim_time ()* is stored as the new *prev_tx_time* value. *rreq_option_ptr* is a pointer to the option fields in the RREQ message header:

```
rreq_option_ptr->current_delay += (op_sim_time () - rreq_option_ptr->prev_tx_time);
rreq_option_ptr->prev_tx_time = op_sim_time ();
```

If a new route entry is inserted in the route table, the cumulative delay is added as parameter to the function creating this new entry. Thus, *rreq_option_ptr->current_delay* is used as an additional parameter to *qaodv_route_table_entry_create* ().

If a route already exists, a new route is evaluated by the following code (rule 3-1):

```
else if ((rreq_option_ptr->src_seq_num >= route_entry_ptr->dest_seq_num) &&
                    (rreq_option_ptr->current_delay < route_entry_ptr->route_delay))
```

qaodv_rte_rrep_pkt_arrival_handle (Figure 3.5): (RREP packet arrival handle)

The cumulative delay is calculated and stored as in *qaodv_rte_rreq_pkt_arrival_handle* except that the *rrep_option_ptr* is used. Also, creating a new route table entry, and evaluating new discovered routes, is handled as in the above, but with the *rrep_option_ptr*.

As explained in section 1.1.5 of Chapter 3, the criterion for using an alternative route discovered during local repair is that the delay should comply with a certain value. This value is set to be the delay request of real-time application packets and if an alternative route's delay is higher than this value, a RERR message is sent and all queued packets are discarded:

```
if (rrep_option_ptr->route_delay > max_delay_voice)
        {
        qaodv_rte_route_error_process (rrep_option_ptr->dest_addr, OPC_NIL,
                                                qAodvC_Data_Packet_No_Route, OPC_TRUE);
        qaodv_packet_queue_all_pkts_to_dest_delete (pkt_queue_ptr,
                                                                rrep_option_ptr->dest_addr);
        }
```

qaodv_rte_rrep_hello_pkt_arrival_handle (Figure 3.6): (Hello packet arrival handle)

The arrival of a *Hello* packet is used to determine if the delay of a single link is still at an acceptable level. First, the link delay is calculated and stored. Then, if the link delay is below the specified threshold, the link is accepted as still valid, and the route and connectivity tables are updated. If not, the route is set to invalid. This will initiate the *route error process*, which will inform all source nodes that used this link that the link is no longer available:

```
rrep_option_ptr->route_delay = op_sim_time () - rrep_option_ptr->prev_tx_time;

qaodv_route_table_entry_param_get (route_table_ptr, ip_dgram_fd_ptr->src_addr,
                                        QAODVC_ROUTE_ENTRY_ROUTE_DELAY, &table_delay);

if (rrep_option_ptr->route_delay < (max_delay_voice/1.5))
        {
        /* update route table entry      */
        qaodv_rte_route_table_entry_update (ip_dgram_fd_ptr,
                                                intf_ici_fdstruct_ptr, tlv_options_ptr);

        /* update connectivity table     */
        qaodv_rte_neighbor_connectivity_table_update (ip_dgram_fd_ptr->src_addr,
                                                                        OPC_TRUE);
        }
else if (route_entry_ptr != OPC_NIL)
        {
        /* set route state to invalid    */
        qaodv_route_table_route_state_set (route_table_ptr,
                                ip_dgram_fd_ptr->src_addr, qAodvC_Invalid_Route, 0);
        }
```

qaodv_rte_all_pkts_to_dest_send (Figure 3.7): (send all queued packets to destination)

For each packet that is queued to the specified destination, it is first determined if the packet has been queued for more than the maximum allowed time period for the specific type of traffic. If it has been, the packet is discarded:

```
if ((op_sim_time () - pk_entry_ptr->insert_time) < 0.4)
```

A delay request is assigned to each packet according to the ToS field in the IP header. Only if the delay value in the route table is less than that of the request, the packet is transmitted; otherwise the packet is queued (since it hasn't been queued for more than the maximum allowed period):

```
if (ip_dgram_fd_ptr->tos == 160)
       delay = max_delay_video;
else if (ip_dgram_fd_ptr->tos == 192)
       delay = max_delay_voice;


if (route_entry_ptr->route_delay < delay)
       {
       // transmit packet
       }
else
       {
       /* queue packet     */
       qaodv_packet_queue_insert (pkt_queue_ptr, ip_pkptr,
                              ip_dgram_fd_ptr->dest_addr, pk_entry_ptr->insert_time);
       }
```

Note that if a packet is re-inserted into the queue, its initial insert time is used as the new insert time to ensure that it will be queued for longer than the allowed time.

Code file: qaodv_route_table.ex.c

qaodv_route_table_entry_create (): (creates a new route table entry)

The delay of a route entry to be created is received as a parameter to this function (see *qaodv_rte_rreq_pkt_arrival_handle*). The value is stored in the *route_delay* field of the route table data structure. This field is accessed via the *route_entry_ptr* pointer:

```
route_entry_ptr = qaodv_route_table_entry_mem_alloc ();
...
route_entry_ptr->route_delay = delay;
```

qaodv_route_table_entry_param_get (): (reads a specific parameter from an entry)

This function returns the value of a specified field in the route table data structure. It is modified to be able to return the value of the *route_delay* field by adding the following code to the switch statement:

```
switch (param)
	{
	...
	case (QAODVC_ROUTE_ENTRY_ROUTE_DELAY):
		{
		dbl_value_ptr = (double*) value_ptr;
		*dbl_value_ptr = route_entry_ptr->route_delay;
		break;
		}
	}
```

dbl_value_ptr is a local variable of type *double** and *value_ptr* a pointer of type *void**, declared as a parameter in which the value is to be returned. The parameter to the switch statement for this specific case is a constant (*QAODVC_ROUTE_ENTRY_ROUTE_DELAY*) declared in **qaodv.h** and received as parameter to the function.

qaodv_route_table_entry_param_set (): (sets a specific parameter to a specified value)

In this function, a field in the route table structure, as specified in the parameter list, is set to a value, also received as parameter. Adding the following *case* modifies the *switch* statement selecting the correct field to be changed:

```
switch (param)
	{
	...
	case (QAODVC_ROUTE_ENTRY_ROUTE_DELAY):
		{
		route_entry_ptr->route_delay = va_arg (arg_list, double);
		break;
		}
	}
```

5. The DA-AODVM Extension

This section discusses the extensions to the code of DA-AODV. Anything that is not explicitly mentioned remains as discussed in section 4.

5.1 Extended data structures

This extension required more complex data structures than DA-AODV. As discussed in section 2.2.2 of chapter 3, each entry in the route and request tables now has to hold a list of possible options of routes and route requests, respectively.

The only alteration made to packet structures is a *route_id* field of type *int,* appended to the RREP and RREQ message structures. All nodes storing the reverse route back to the originator also store the route ID assigned to a RREQ message. The route table entry stnicture is modified to adapt the form of Figure 2.6 (b). Only common information is part of the now shorter *qAodvT_RouteJintry* structure. For simplicity, a single precursor list is defined and used per route entry (refer to Chapter 7).

A new pointer *(route__lptr)* of type *List** is declared in *qAodvT_Route_Entry.* This is used to point to a list of which the entry type is *qAodvT_Route_List_Entry:*

```
typedef struct
        {
        IpT_Dest_Prefix             dest_prefix;
        ■int                        dest_seq_num;
        IpT_Port_Info               next_hop_port_info;
        List*                       precursor_Jptr;
        List*                       route_lptr;
        } qAodvT_Route_Entry;
```

qAodvT_Route_List_Entry is a new defined complex data structure to hold all information that is unique to each of the possible routes to a destination:

```
typedef struct
        {
        InetT_Address               src_addr;
        int                         src_seq_nr;
        int                         route_id;
        Boolean                     valid_dest_sequence_number_flag;
        qAodvC_Route_£ntry_State    route_entry_state;
        inetT_Address               next„hop_addr;
        IpT._Port_I.nfo             next_hop_port_info;
        int                         hop_count;
        double                      route_expiry_time;
        Evhandle                    route_expiry_evhandle;
        int                         list_pos;
        double                      route_delay;
        } qAodvT_Route..List_Entry
```

Since some events, e.g. route expiry, have to occur for route list entries and not for route table entries, a second new complex data structure is defined to contain information regarding the