Minimum Congestion Routing for a 17 GHz Wireless Ad Hoc Network

by
Daniël Johannes Van Wyk Kotze

Thesis presented in partial fulfilment of the requirements for for the degree Master of Science in Engineering at Stellenbosch University

Supervisor: Dr. R. Wolhuter
Department of Electrical and Electronic Engineering

March 2011

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2011

Opsomming

Sleutel woorde: Ad hoc radio pakkie netwerk, millimeter golflengte, kluster roete protokol, data verkeersopeenhoping beheer, simulasie, teoretiese modelleering.

Ondersoek word ingestel na ’n geskikte roete protokol vir ’n millimeter golflengte ad hoc radio pakkie netwerk. Daar word gevind dat ’n hiërargiese kluster roete protokol ideaal is vir ’n hoë digtheid van nodusse. As gevolg van die hoë bandwydte, wat moontlik beskikbaar is met millimeter golflengte transmissie, word pakkies gebruik om kommunikasie skakels tussen nodes in stand te hou en data pakkie verkeersopeenhoping te beheer. Kluster leiers word verkies en gebruik teken-pakkies om nodes met ’n groter data pakkie las meer transmissie kanse te gee. Sodoende word die verkeersopeenhoping van data pakkies verminder. Hallo pakkies word gereeld gestuur om die roete inligting vars te hou en gebroke kommunikasie skakels vinnig op te spoor. As ’n gebroke skakel gevind word, word ’n alternatiewe roete vinnig opgestel, binne ’n sekonde. ’n Simulasie word opgestel om die protokol te toets. Veranderinge aan die oorspronklike proaktiewe kluster protokol word aangebring om roete lengte te verklein en oorhoofse roete inligting kommunikasie te verminder. ’n Teoretiese model gebasseer op tou-staan teorie word ontwikkel om die wagtyd van ’n pakkie te bepaal. Alhoewel, insig verkry is deur die protokol te analiseer deur middel van tou-staan teorie, word daar voorgestel, as gevolg van die protokol se kompleksiteit, om eerder ander wiskundige modelleeringstegnieke te gebruik soos ’n Markov toestands model of ’n Petri net.

Abstract

Keywords: Ad hoc radio packet network, millimeter wave, cluster based routing protocol, congestion control, simulation, theoretical modelling.

An investigation is made to find a suitable routing protocol for a millimeter wave ad hoc wireless network. It is discovered that a hierarchical routing protocol is ideal for a high node density. Due to the high bandwidth that is possibly available, with millimeter wave transmission, packets are used to keep links between nodes active and to control data packet congestion. Cluster leaders are elected and use token packets to provide nodes with more queued messages with more transmission chances, assisting the network in congestion control. Hello messages are sent frequently to keep routing information at nodes fresh and to detect broken links quickly. If a broken link is found a new route is readily available, within a second. A simulation is created to test the protocol. Changes are made to the original proactive cluster routing protocol to reduce the route length and lessen routing overhead. A theoretical model is developed to estimate the mean waiting time for a packet. Although insight is gained by modelling the latency with queueing theory it is suggested, due to the protocol's complexity, to use other mathematical modelling techniques such as a Markov state model or a Petri net.

Acknowledgements

I would like thank my supervisor Dr. Riaan Wolhuter for his patience, guidance and wisdom.
My family also has been a source of immense support, thank you.
My parents for their curiousity and support.
Thank you Oswald Jumira for proof reading the thesis.
Konrad Blum for introducing me to Git revision control and various other tips.
Ryno thank you for being a friend, in the true sense of the word.
Microsoft Research for sponsoring a travelgrant to attend the 2008 IFIP Summer School at Meraka.
Also I would like to thank Dr. Thinus Groenewald and Dr. Herman Spies.
I thank God for the breath of life.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

MMW	Millimeter-wave
UWB	Ultra-wideband
HIPERLAN	High Performance Radio Local Area Network
ETSI	European Telecommunications Standards Institute
QoS	Quality of Service
SOHO	Small Office Home Office
VOIP	Voice Over Internet Protocol
CGSR	Cluster-head Gateway Switch Routing
DSDV	Destination Sequenced Distance Vector
AODV	Ad hoc On-demand Distance Vector
Wi-Fi	Wireless Fidelity
MAC	Media Access Control
CSMA	Carrier Sense Multiple Access
CSMA/CD	CSMA with Collision Detection
CSMA/CA	CSMA with Collision Avoidance
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
UDP	User Datagram Protocol
TCP/IP	Transmission Control Protocol and Internet Protocol
ST	Slot Time
DIFS	Distributed Interframe Space
SIFS	Short Interframe Space
EIFS	Extended Interframe Space
Hz	Hertz
s	second(s)
m	milli
k	kilo
G	giga

Symbols

P	Power
G	Antenna gain
λ_W	Wavelength
d	Displacement
C	Data rate
B	Bandwidth
S/N	Signal to noise ratio
t	Time
α	Path loss exponent
S_F	Scaling factor
X	Random variable
λ_A	Customer arrival rate
μ	Customer service rate
ρ	Traffic intensity
N	Number of customers in queue
T	Mean waiting time for a customer in a queue
p	Probability

Chapter 1

Introduction

Ad hoc wireless networks provide an advantage in that no prior infrastructure have to be set up. However, one inherent problem of these networks are their limited data transfer capacity. On the other hand, the data transfer requirements of media applications are increasing by the day. Consumers demand a faster data transfer speed and higher content quality. To cope with the demand for greater bandwidth, we propose to investigate the feasibility of using a millimeter-wave (MMW) ad hoc network with particular interest in suitable routing protocols therefor. A MMW system will allow one to transmit data at higher throughput, but with the cost of strong signal attenuation as predicted by Frii's equation [27].

Ultra-wideband (UWB) [43] is another technology trying to address the bandwidth problem for short range communication. Obstacles for UWB include interference with other systems (2,4 GHz and 5 GHz unlicensed bands) and the lack of international harmony with regard to the operating spectrum [17]. These problems are absent for MMW networks and the technology is currently receiving increasing attention.

1.1 MMW Ad hoc Networks

Due to the nature of MMW propagation, ad hoc networks using MMW technology are better suited for indoor environments. The IEEE 802.15c is a working group focusing on MMW Wireless Personal Area Networks (WPAN) [1]. Various studies have been conducted to investigate MMW transmitters in the $57 - 64$ GHz band, as the Federal Communications Commission (FCC) has allocated this band for unlicensed use [17].

A Japanese study [38] provides an overview of a MMW ad hoc wireless access system, they have developed for indoor conference use.
Applications for wireless data transmission over small distances include:

- high definition video streaming,
- large file transfer and
- wireless ad hoc networks (connectivity between different electronic devices) [17, 39].

A wireless MMW signal can be blocked easily, because of the weak diffractive properties of MMWs [39].

1.2 HIPERLAN Standard

HIPERLAN is an acronym for HIgh PErformance Radio Local Area Network [18]. This standard aims to bring the high data rates of wired Local Area Networks into the wireless realm. The 5 GHz and 17 GHz bands have been identified for potential use. The European Telecommunications Standards Institute (ETSI) has developed the HIPERLAN/2 standard. HIPERLAN focuses on Quality of Service (QoS) support within the Small Office Home Office (SOHO) environment [14].

1.3 Project Outline

In this project we plan to use the 17 GHz band for our application. Investigation of millimeter-wave transmission for possible use in ad hoc networks, is composed of multiple sections. These consist of developing the the physical transmitter hardware and a routing protocol respectively. This thesis constitutes the development of *only* the routing protocol as a first step. Hardware development will be addressed in another project. Figure 1.1 shows the critical path diagram for the total research project.

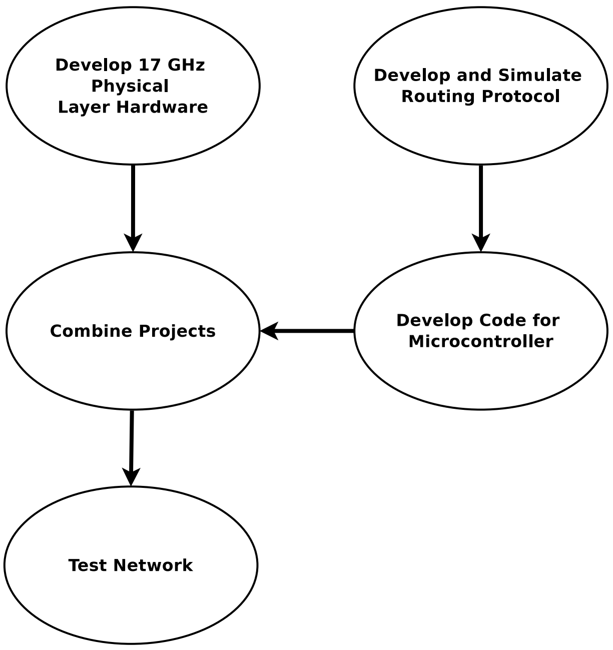

Develop 17 GHz Physical Layer Hardware
Develop and Simulate Routing Protocol
Combine Projects
Develop Code for Microcontroller
Test Network

Figure 1.1: *Research critical path diagram.*

1.4 Objectives for Routing Protocol

The routing protocol was designed to meet requirements that will make it more suitable for the 17 GHz frequency band, than for use in a lower frequency range.

- High densities of nodes must be handled effectively, by reducing redundant routing overhead transmissions.
- In the 17 GHz band, nodes have high bandwidth availability. We utilise this extra bandwidth for Quality of Service support or link sensing.
- Congestion must be avoided or reduced. By improving packet latency, the protocol will be more attractive for Quality of Service applications such as video streaming and Voice Over Internet Protocol (VOIP).
- A new route to a destination node must be readily available should a link be broken. This requirement was included, because electromagnetic waves at higher frequencies can be easily blocked [28] and diffraction of the waves is weaker [39].

1.5 Contributions

- An adaptation of the Cluster-head Gateway Switch Routing (CGSR) protocol for use in a MMW network is presented. This routing protocol combines a number of desirable features:
  - High densities of nodes are handled efficiently by reducing the number of nodes producing full routing overhead.
  - Nodes make use of high available transmission bandwidth for link sensing, by frequently sending small hello packets.
  - Congestion is decreased via a Quality of Service mechanism giving nodes with more queued traffic more transmission chances.
  - The protocol allows a new route to be found within 1 s, if a route link is broken.
- The protocol described was simulated within the OMNeT++ simulator. At the time no known hardware specifications were available for devices transmitting data at a carrier frequency of 17 GHz. It was therefore decided to continue with the simulation analysis at 2,4 GHz as carrier frequency. Although the protocol was only simulated at 2,4 GHz it is anticipated that the protocol will have similar performance at 17 GHz, except for higher fading. Herewith a useful tool for simulation modelling of such topologies, was developed.
- In addition to the simulation tool, a theoretical model was developed, based on queueing theory.

1.6 Work Done

1.6.1 Literature Overview

Research on literature of basic data networks is presented, progressing to the subject of ad hoc wireless network routing protocols. It was found that hierarchical protocols provides a good solution at high node densities, because of decreased routing overhead and faster route convergence.

1.6.2 Design Overview

The design choices of developing the routing protocol are documented. Changes were made to the original Cluster-head Gateway Switch Routing (CGSR) Protocol. The first change constituted decreasing the path length by including information on the next-hop-cluster. The other change reduced the redundant nodes that will produce full routing overhead. Each node is at least one hop away from a node producing full routing overhead. The comparison of the new protocol and the original protocol with regard to routing overhead and path length must still be done, but this is beyond the scope of the study.

It is shown how to scale the pixel distance of the OMNeT++ Graphical User Interface, for physical distances not displaying conveniently. The implementation specifics of a cluster based protocol are shown. Cluster generation, node states, routing tables, routing information distribution and the routing function are documented. A new node state is defined, namely, the *satellite node*. The *satellite node* state was not defined in previous implementations of CGSR.

A channel control method is presented, making use of token passing with prioritisation. Message buffers try to enforce fairness in the transmission queue by splitting messages generated by the node itself and messages requiring forwarding, into different buffers. The content of protocol messages are presented in byte field form. A simulation performance module was created to measure network characteristics in simulation time. Graphs representing the network, indicating node states and forwarder nodes are generated by the performance module.

1.6.3 Latency Model

A latency model was created using queueing theory. The model determines the ratio of the time a cluster is busy with transmission in a neighbouring cluster. It is argued that the time a cluster is busy in a neighbouring cluster increases the cycle time of that cluster. The cycle time is used to calculate the service rate of data packets at a node in the cluster. The waiting time of data at a node is calculated using queueing theory and the waiting times are summed to determine the latency of a route. The model is applied to various scenarios.

1.6.4 Protocol Testing

The protocol was tested extensively with regard to successful cluster setup, network connectivity, routing overhead and route recovery. Cluster setup was tested by generating networks where nodes are distributed randomly within an rectangular area to see how clusters are generated. The time needed to set up the clusters is measured. The aggregate network connectivity was determined and plotted at different points in time, to determine if all nodes have complete routing information. The time required for the routing information to propagate throughout the network is determined.

The routing overhead per node for different network sizes, was ascertained and plotted. First, the routing overhead per node of a small fifteen node network is determined. With each simulation run the number of nodes was increased with five, in the same area. Using this method the routing overhead per node with increasing node density could be determined. It was found that the routing overhead per node decreases with increasing node density. The result can be compared with the Destination Sequenced Distance Vector (DSDV) routing protocol which is expected to grow linearly, $O(N)$, with increasing number of nodes. It can be said that the developed protocol uses routing overhead more efficiently than DSDV at a high node density.

The effect of token prioritising on mean packet latency was investigated by comparing it to a setup where no priority is given to nodes with more queued traffic. The latency is improved in most instances where token prioritising is used. The improvement is more visible with higher hop counts and larger network sizes. Instances where there was no improvement in latency, an increase in throughput was observed.

The way the protocol reacts when a single route link is broken was studied. The protocol finds a new route, within 1 s after a link is broken, along which data packets can be sent. One instance of routing ambiguity occurred, where two routes were used to reach a destination node.

1.6.5 Comparison with Previous Work

CGSR [12, 16] defines a cluster based proactive routing protocol where every node produces full routing overhead, i.e. routing information on clusters and node-cluster associations. In our application we reduce nodes producing redundant routing overhead transmissions, by letting every node be at least one hop away from a node producing full routing overhead. The concept of reducing redundant routing overhead transmissions in cluster based protocols has been researched in Ad hoc On-demand Distance Vector with Passive Clustering (AODV-PC) [15], Cluster Based Routing Protocol (CBRP) [20] and Adaptive Routing using Clusters (ARC) [10]. All these protocols build on the reactive protocol of AODV. Our protocol builds on the original proactive CGSR protocol. CGSR extends the proactive Destination Sequenced Distance Vector (DSDV) protocol.

CGSR and our protocol both use priority token scheduling. CGSR gives preference to nodes transmitting more messages. We however, give preference to nodes in a cluster with more queued messages.

Our protocol provides increased support for the fast detection of broken links by sending small hello messages (only including addresses of critical nodes) frequently every 0,1 s. If 5 consecutive hello messages are missed, the link state is set to *unstable*. The handover of the data stream to the new route occurs within 1 s. CBRP has a local repair mechanism using full two hop topology information to repair the route. Hello messages of the CBRP protocol are sent every 1,5 s or 2 s and contains the whole neighbour table and cluster adjacency table. When a route error occurs with CBRP, a Route Error packet is sent from the broken link and a Route Reply packet is sent from the packet's destination node with the new route. The proactive nature of our protocol negates the need for the Route Error packet and Route Reply packet.

A theoretical latency model is developed to estimate mean packet latency of the cluster based protocol. Queueing theory principles are used to develop the model. To the best of author's knowledge there has been no other attempt to create a latency model for a cluster based protocol.

1.7 Content

The content of the thesis is organised as follows:

- *Chapter 2* discusses basic literature on data networks and specifically ad hoc networks. Important concepts such as the layered OSI model; latency and throughput; and congestion control are explained. Physical properties of the wireless channel is stated and Frii's equation is provided. The hidden node problem is explained. Subsequently, sender and receiver initiated MAC layer protocols are discussed. Next an in depth investigation of routing layer protocols is given. Hierarchical protocols such as Cluster-head Gateway Switch Routing (CGSR) and related cluster protocol research are carefully scrutinised.

- *Chapter 3* contains the design and implementation of the cluster based protocol. Design choices are explained and changes to the existing protocol are stated. The implementation environment of the OMNeT++ simulator is presented. A Wireless Fidelity (Wi-Fi) network transceiver's transmission parameters are used. OMNeT++'s graphical network display uses a pixel to represent 1 m and a section was provided to show how to scale the graphics. After a general overview of the routing protocol development, the cluster generation mechanism is shown. Next, all the different node states are discussed with regard to each state's function. A section on the protocol's routing tables will follow, after which, the routing information distribution mechanism will be explained. A flow diagram of the routing function is presented. The token scheduling method and the message buffers are discussed. Content of the protocol messages are displayed. A performance module is presented for aggregate simulation time measurements.

- *Chapter 4* develops a latency model for a cluster based network. Since the model is based on queueing theory a short introduction on the subject is given. From there the latency model is developed. The rest of the chapter shows how the model can be applied to various topologies. The latency model is also compared to mean latency values obtained through simulation. Reasons are provided why the theoretical model and the measured simulation results disagree sometimes. Other investigated approaches based on queueing theory are summarised. Future work on modelling the latency by means of Petri nets is suggested.

- *Chapter 5* describes tests to prove the functionality of our designed network protocol and show how the objectives set out in the introductory chapter are achieved. The general simulation setup is explained and the sizes of the different playgrounds are shown in tabular form. Our clustering algorithm is tested by measuring the time it needs to set up clusters and the number of clusters generated. The following test determines if all nodes have full routing information on all the other nodes in the network and the time each network needs to reach full connectivity. Further tests determined how effective the protocol uses routing overhead at a high node density. To show that the protocol provides Quality of Service support, we determined the effect that prioritising nodes with more queued traffic has on the mean packet latency. Lastly, to show the protocol can recover from single link breakages, a test was set up to send data packets along a route and after a fixed amount of time a link in the route is broken. The protocol recovers from the link break very quickly, within 1 s.

- *Chapter 6* concludes the thesis and presents a summary of the research findings. The advantages and disadvantages of using cluster based routing are stated. Recommendations on improving the protocol are given. Some possible shortcomings in areas of the study are indicated in an appropriate section. Interesting remarks relating to the classification of the routing protocol type are given. Lastly directions of future research endeavours are shared.

- *Appendix A* contains all the graphs of the networks simulated in chapter 5.

- *Appendix B* documents assorted scripts. The BASH script for invoking the multiple simulation runs, is included.

- *Appendix C* presents an article on the research, published in the proceedings of the Southern Africa Telecommunications and Network Applications Conference 2010.

- *Appendix D* describes the content of an accompanied multimedia disk.

Chapter 2

Literature Study

Data networks can be seen as a subfield of telecommunications, the science of communicating across a distance [37]. The sharing of information in businesses, within society and globally, has become important and Internet connectivity helps us to distribute critical data. Ad hoc wireless networks present us with the opportunity to expand connectivity at a low cost, because no prior communication infrastructure have to be set up. This chapter starts by introducing wireless networks and the various aspects required to understand the method of communication. Next wireless MAC-layer protocols are discussed. Wireless routing protocols are investigated, focussing on hierarchical routing protocols. Lastly, a summary on articles about cluster based protocols is given.

2.1 Ad Hoc Wireless Networks

In 1969 the Department of Defence understood the importance and possibilities of packet switched radio to connect mobile nodes in the battlefield. Defence Advanced Research Projects Agency (DARPA) Packet Radio project started the concept of ad hoc wireless networking in the early 1970's. Ad hoc networks are used to provide networking for areas without wired or cellular infrastructure. These networks are usually set up for a limited period of time and provide a base for specific media applications for example audio or video streaming etc. [25]

A network adheres to the following basic principles:

- All nodes in a network are equal.
- If nodes are not within range of one another, data can be routed through an intermediate node (See Figure 2.1).

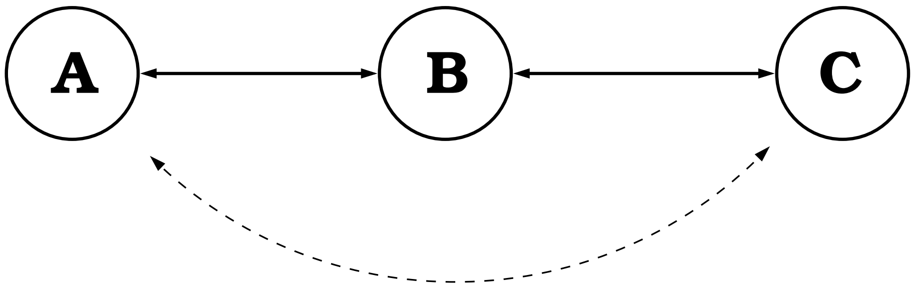


Figure 2.1: *Node B can communicate with nodes A and C directly. If node A wants to communicate with node C then node A can route data through node B.*

2.2 Basic Networking

In this section networking concepts are discussed. First, very basic networking concepts are introduced, followed by the seven layer OSI model. Crucial concepts of throughput and latency, which are needed to analyse a network, are explained. Subsequently congestion control is discussed, that can assist in making design choices.

2.2.1 Basic Concepts

Communications channel: All nodes in a network need a communications channel to communicate with other nodes. This channel can either be a physical connection through a wire or a wireless transmitter.

Packet switched network: In this type of network data is broken into smaller sets of information called packets. These packets are transmitted across the channel.

2.2.2 Seven Layer OSI Model

The Open System Interconnection (OSI) reference model is a layered model that shows how a set of protocols are used to communicate across a network. Each layer has a function and each layer of one node in a network can communicate with its corresponding layer of another node in the network. Not all of the layers in the model need to be used when designing a new set of communication protocols. The important layers are the application, transport, network, data-link and physical layers. Figure 2.2 on page 11 shows the OSI reference model.

Application Layer

The application layer is the topmost layer in the model. This layer is used by applications on a host computer to communicate with applications on another host on the network. This hides the complexity of communication between nodes from the applications.

Transport Layer

The transport layer is the layer responsible for setting up end-to-end connections. This layer is responsible for the reliable transmission of packets from one node to another. If Transmission Control Protocol (TCP) is used, the destination node sends an acknowledgement if a packet is correctly received. User Datagram Protocol (UDP) does not send an acknowledgement for the correct reception of a packet.

Network (Routing) Layer

The network layer is responsible for the routing of data packets. This enables two nodes, not directly connected, to communicate with one another by "hopping" to an intermediate node. This layer also defines the addressing scheme the network uses. Nodes use the addressing scheme to determine which node a packet must be forwarded to.

Data-Link Layer

The data-link layer defines the set of rules a group of two or more nodes use to communicate with one another. This layer also consists out of two sub-layers the Logical Link Control and the Media Access Control. The layer also groups the data to be sent into frames. These frames are transmitted on the physical medium.

Physical Layer

The physical layer represents the method by which data is transmitted from one node to another on the communications channel.

2.2.3 Throughput and Latency

The performance of a network is measured with respect to throughput and latency. The throughput is the end-to-end data capacity of the network. Throughput measures the number of bits that can be transmitted in a given amount of time, in bits per second (bps), from source to destination. Latency is defined as the end-to-end travel time of a message from one node to another. The Round Trip Time (RTT) is the time needed for a message to travel from a source to destination and back to the source again. [33]

2.2.4 Congestion Control

Data networks only have a finite capacity which means if certain links are overused some of the messages may have to be dropped. When a link is overused it is said the link is congested. Congestion has a negative effect on the performance of a data network. There are different approaches to prevent congestion from degrading network performance. One method is to route data in such a way that it avoids congested links. Another method is to prevent congestion from occurring by reserving bandwidth for transmission or controlling

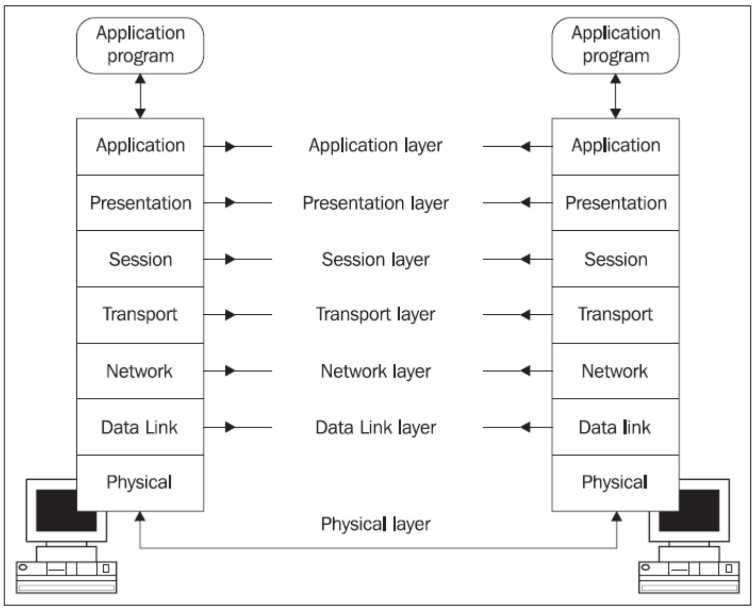


Figure 2.2: *The figure shows the function of the various layers in the OSI protocol stack. (Reproduced from [3]).*

the rate at which a node generates traffic. Routers in the network can allocate bandwidth fairly among nodes. This is called a router-centric approach. Nodes in the network can detect when packets are dropped. The dropped packets can signal that a link is congested and the rate at which traffic generated is reduced. This is called a host-centric approach. [33]

2.3 Wireless Transmission Considerations

The wireless transmission of data holds certain restrictions, namely path loss, fading and interference. Due to loss of the electromagnetic signal strength, there is a limit on the range of wireless transmissions. Frii's equation can be used to calculate path loss.

$$P_r = P_t G_t G_r \left(\frac{\lambda_W}{4\pi d} \right)^2 \tag{2.1}$$

- P_r is the received signal strength.
- P_t is the transmission signal strength.
- G_r is the receiver antenna gain.
- G_r is the transmitter antenna gain.
- d is the displacement from the transmitter to the receiver.

- λ_W is the wavelength of the transmitted electromagnetic wave.

Fading is another feature causing variations in the signal strength at the receiver. Reflection, diffraction and scattering of electromagnetic waves contribute to fading. Interference on wireless channels can corrupt the data of a transmission if another signal also transmits in the same frequency band.

Information transmission has a minimum time constraint. Shannon's theorem defines the maximum transmission data rate. [29]

$$C = B \times \log_2(1 + S/N) \tag{2.2}$$

- C is the maximum data rate.
- B is the bandwidth of the channel.
- S/N is the ratio of signal power to noise power.

2.4 MAC-Layer Protocols for Wireless Ad Hoc Networks

As mentioned earlier, the MAC-layer protocol is a set of rules nodes use to access the physical layer in order for two or more nodes to communicate with one another. First, this section will focus on problems that must be overcome to ensure collision free transmission. Then the different types of MAC-layer protocols, namely, sender-initiated, receiver-initiated, reservation based, and schedule based protocols will be explained.

2.4.1 Hidden and Exposed Node Problem

Sometimes nodes can be in range of a common receiver, but out of range for one another. See figure 2.3 on the following page. Node A and node C are both in range of node B, but node A is out of range for node C. Therefore if node A is busy transmitting, node C will not detect that node B is receiving a message and can start transmitting simultaneously. A collision at node B will occur and the transmission data will be lost. This is called the hidden node problem, because node C is hidden for node A.

The exposed node problem occurs when node B is transmitting a message to node A and node C wants to send a message to another node. Node C will now be prevented from transmitting, because it senses the channel as busy. [29]

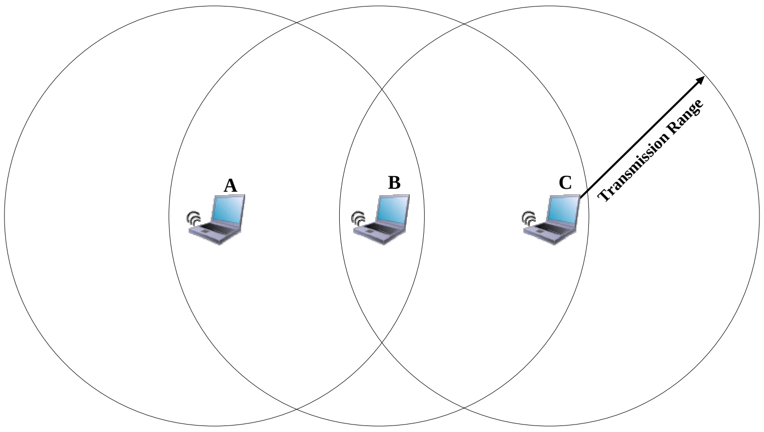


Figure 2.3: *Three nodes with their respective transmission ranges.*

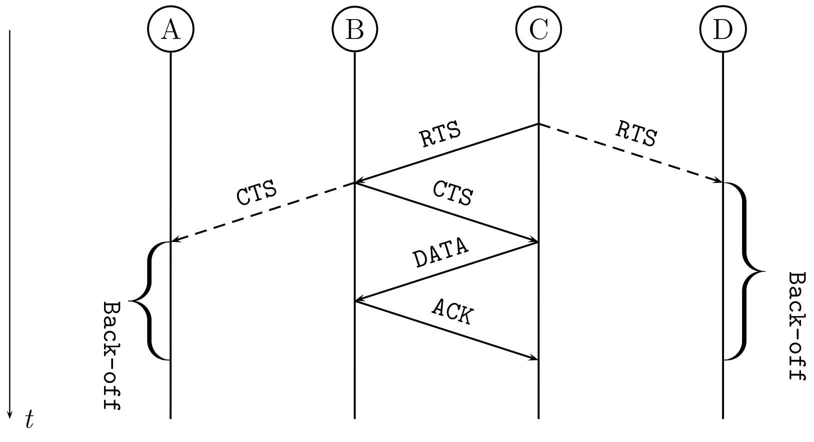


Figure 2.4: *CSMA/CA control packet exchange*

2.4.2 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)

Wireless networks use collision avoidance to solve the hidden node problem. CSMA/CA is a sender-initiated MAC-layer protocol and works as follows. First a sender node senses the channel. If the channel is free for a certain time, Distributed Inter Frame Space (DIFS), it transmits a Request To Send (RTS) packet to a destination receiver node. The receiver node then responds by sending a Clear To Send (CTS) packet to the original sending node. The sender node then sends a packet containing data to the receiver. The receiver node then replies with a positive acknowledgement (ACK) packet if the data was correctly received. The procedure is shown in figure 2.4. The RTS and CTS control packets both contain information on how long the full transmission cycle will be. This time information is used by neighbouring nodes to schedule a moment in the future, when the channel will be sensed again to see if it is available. If collisions occur, time-outs for the CTS or ACK packets expire or after a successful transmission a node goes into a back-off mode. The back-off time is calculated by a binary exponential back-off algorithm. [11]

2.4.3 Multiple Access Collision Avoidance - By Invitation (MACA-BI)

MACA-BI is a receiver-initiated MAC-Layer protocol. The protocol aims to create less control overhead by making the need for a RTS packet obsolete. Instead the MACA-BI protocol uses a RTR (Ready To Receive) packet to indicate to a sender node that it can send a data packet to a receiving node. Like the RTS and CTS control packets the RTR packet also contains information on how long the transmission cycle will be. Neighbours use this information to determine when it is safe to transmit again. One disadvantage of the protocol is that the RTR packet can collide with the data packets in certain situations. [29]

2.4.4 Contention-Based Protocols with Reservation Mechanisms

Protocols with reservation mechanisms assist nodes by reserving bandwidth. These protocols divide the time into a series slots. Each node contends for a slot during a resource allocation phase which is a small time section at the start of each slot. During the rest of the slot time the node winning the contention can send data. Protocols with reservation mechanisms is effective for time sensitive data applications, for instance voice data. [29]

2.4.5 Contention-Based Protocols with Scheduling Mechanisms

Scheduling mechanisms aim to schedule packets at nodes for fair channel access. Scheduling mechanisms can also link priorities to certain nodes, giving a greater priority to nodes with more queued traffic. Battery life can be used to determine priority as well. The greater a node's priority the more channel access it is allowed. [29]

2.5 Routing Layer Protocols for Wireless Ad Hoc Networks

A normal ad hoc network is essentially different from wired networks with respect to bandwidth, topology and resources. Ad hoc networks are limited in bandwidth and consists of mobile nodes causing the topology to change dynamically. Ad hoc networks also rely on batteries that can be depleted of energy [25]. It is reminded that in this study we focus on stationary nodes, without a low bandwidth limitation. It is also assumed that the battery life is not limited. Different examples of routing layer protocols for wireless ad hoc networks will now be discussed.

2.5.1 Proactive Routing Protocols

Nodes using proactive routing protocols set up and maintain routes to all other nodes in the network. The routing information is stored in different tables. Information in the routing tables are periodically transmitted throughout the network [5]. The advantage of a proactive protocol is that accurate information of the network is maintained [29] and therefore present route information is up to date.

2.5.2 Destination Sequenced Distance Vector

Destination Sequenced Distance Vector (DSDV) is based on the Routing Information Protocol (RIP) used in wired networks. DSDV is a proactive routing protocol, improving on the distributed Bellman-Ford algorithm. Every node keeps a table of information that contains for every destination node the:

- address,
- smallest number of hops to the destination,
- node address of the next hop on the shortest route and
- sequence number of the most recent route update.

Routing tables are periodically exchanged between neighbour nodes. The addition DSDV makes to the distributed Bellman-Ford algorithm is an increasing sequence number field, which is added to each route in the table update message. The addition of sequence numbers helps to prevent the count-to-infinity problem [29]. There are mainly two types of updates an "incremental", containing changed routing information, or "full dump", that consists of all routing information. Full dumps are transmitted when there is significant changes in the topology of the network.

When an update packet is received by a node it compares the route information of the packet with its own. Routes of the node are updated if new routes:

- contain a more recent (greater) sequence number or
- have a sequence number equal to the current sequence number, but have a better metric i.e. less hops.

A settling time table is also used by the protocol to prevent changes in the routing table from being advertised too soon. Suppose a route is updated, because a update message is received that has a more recent sequence number for the route. Now, after a little more time, another update message is received, from another node, for the same route with the same sequence number, but with a better metric. The route is updated again. The settling time table effectively prevents the first update message's route information from being advertised,

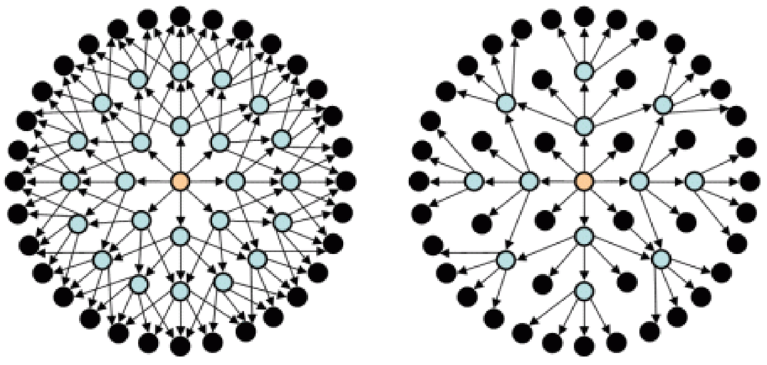


Figure 2.5: *The picture shows on the left the normal flooding mechanism. On the right it is shown how the OLSR flooding mechanism reduces the number of nodes needed to forward a flooding message. Reproduced from [2].*

because a node has to wait the average settling time for a route before it can advertise the route's information.

It is proved in [32] that DSDV provides loop-free paths to all destinations. The routing overhead of DSDV however grows by order of $O(N^2)$ therefore it does not scale well for large networks [5].

2.5.3 Optimized Link State Routing

Optimized Link State Routing (OLSR) is another proactive routing protocol that uses an effective flooding mechanism for its routing updates. OLSR is based on the link state algorithm where each node maintains information on the topology of the network. Unlike a normal flooding protocol where every node broadcasts the flooded message once, OLSR uses only multipoint relay (MPR) nodes to broadcast the flooded update message. Nodes broadcast hello messages that contain a list of all the one hop neighbours. The hello messages are used by a node to determine a subset of neighbour nodes covering its two hop neighbourhood. This subset of nodes are selected as the MPRs. Using MPRs to broadcast a flooded message, the message will reach every node in the network [5, 29]. Figure 2.5 shows how messages are flooded.

2.5.4 Reactive Routing Protocols

Reactive routing protocols or on-demand protocols are designed to decrease the control overhead. This decrease of control overhead is achieved by maintaining only routes of nodes sending data. Routes to destination nodes are determined by flooding the network with Route Request (RREQ) packets. When a node with a route to the destination or the destination node itself is reached, a Route Reply (RREP) packet is sent back to the original node. The RREP packet is sent back by link reversal through bi-directional links, or is flooded through the whole network with the route information included in the packet. Reactive protocols can be grouped into two categories i.e. source routing and hop-by-hop routing. [5]

2.5.5 Dynamic Source Routing

Dynamic Source Routing (DSR) is an on-demand routing protocol that uses source routing. When a sender node wants to send a data packet through the network, it constructs a source route. The source route is included in the packet header and includes every node address of the route the packet must follow, to reach the destination node. Each node has a route cache where learned source routes are stored.

A source route is constructed by route discovery, as follows. First, the network is flooded with a RREQ packet by letting every node broadcast the packet once. Figure 2.6 on the next page shows a RREQ packet propagating through the network. When the route RREQ reaches the destination host or an intermediate host, with a valid route to the destination host in its route cache, a RREP packet is sent back to the original node. The RREQ packet has an id (identification) field generated by the original node to help intermediate nodes determine if they already have broadcast the packet. A route record is also used in the RREQ packet, that records the addresses of the nodes it has visited as it is forwarded through the network. This route record is included in the RREP packet.

The RREP packet is sent back to the original node, if the destination node has a valid route to the original node in its route cache. Otherwise the RREP packet may follow the reverse route in the route record, but this is discouraged. A wireless link may work well in one direction, but be worse in the opposite. Another way the destination node can send the RREP back, is to piggy-back the message on a RREQ packet for the original node.

With DSR, no periodic update messages are sent, so the protocol has to have a mechanism to detect broken routes. Route Error (RERR) messages are used to detect broken links on routes. If the MAC layer reports an error while sending a data message, the routing layer is informed. A RERR message is generated at the node where the error occurred and is sent back to the original sending node. The RERR message contains the addresses of the nodes sharing the broken link. The original node then discards all routes that use the broken link and does another source route construction. [21]

2.5.6 Ad hoc On-demand Distance Vector

Ad hoc On-demand Distance Vector (AODV) is another on-demand protocol combining ideas from DSR and DSDV. Like DSDV, AODV sends out periodic update messages and AODV has the same route discovery mechanism as DSR. AODV does not include the routing information in the packet header like DSR and routes the packets hop by hop. Also the RREP messages in the AODV protocol only include a network address and a sequence number. AODV is useful in networks where nodes are highly mobile. [5]

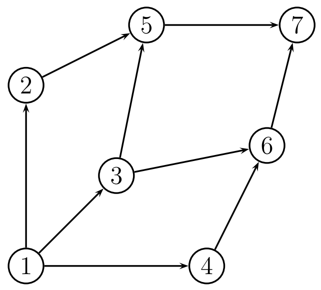


Figure 2.6: *A route request packet from node 1 for a route to node 7 is propagated through the network.*

2.5.7 Hierarchical Routing Protocols

Hierarchical routing protocols group sets of neighbouring nodes into a cell. In a cell, a cluster-head is elected that coordinates nodes within a cluster. Hierarchical routing protocols decreases the size of routing tables and allows for increased scalability. If the node density is high, hierarchical protocols have a much better performance, because it uses less control overhead, routes converge faster and routing paths are shorter on average. [6, 19, 29]

2.5.8 Cluster-head Gateway Switch Routing Protocol

Cluster-head Gateway Switch Routing (CGSR) is a hierarchical protocol, where nodes are grouped into clusters and one node in the cluster is elected as the cluster-head. The cluster-head maintains the cluster and all communications between clusters are routed through the cluster-head. The advantage of the protocol is that nodes only have to maintain a route to their cluster-head. Hence the routing overhead is much lower, because only cluster-heads transmit routing information. A penalty is however paid in control overhead to maintain the cluster, because each node needs to periodically transmit its cluster member table. The cluster member table contains the node-cluster associations of nodes. The cluster based protocol favours networks where nodes are quasi-static. [5, 12, 16]

Clustering Algorithms

Mario Gerla et al. presents two methods of how clusters are formed, namely the lowest-id cluster algorithm and highest-connectivity cluster algorithm [16]. The lowest-id method works as follows. Each node periodically broadcasts a list of all its neighbour nodes. A node is a cluster-head if it only hears nodes with an id (address) higher than its own. Nodes hearing the cluster-head become part of the cluster. A node hearing two or more cluster-heads is called a gateway, otherwise the node is a normal node.

The highest-connectivity cluster works as follows. Again each node broadcasts a list of all its neighbour nodes. A node is elected as a cluster-head if it has the most connections of all of its uncovered neighbours. Uncovered nodes are nodes without an elected cluster-head. If two nodes have a tie for the most connections, the node with the smallest id (address) is elected as cluster-head.

Clusters have two main properties:

- cluster-heads are not directly connected and
- nodes belonging to a cluster are at most two hops away from one another, because the cluster-head is a neighbour of every node in the cluster.

This protocol operates in a mobile environment where nodes move around and cluster-heads can occasionally move into each other's range. In such a case the cluster-heads challenge each other with either the lowest-id or highest connectivity as criteria. The result is that the loser gives up its cluster-head status. Figure 2.7 on the following page shows how nodes are grouped together in a cluster.

MAC-Layer

The MAC-Layer of the protocol has gone through changes. First, Time Division Multiple Access (TDMA) was used within the cluster [16] and this has changed to token scheduling [12]. Among clusters Code Division Multiple Access (CDMA), with different spreading codes, is used to ensure one cluster's transmissions do not interfere with another. Token scheduling is a polling scheme that allocates transmission time among nodes in a cluster.

The procedure for the token scheduling works as follows:

- First the cluster-head receives the permission token and transmits any messages in its transmission queue.
- Then the token is passed on to the next scheduled node in the cluster and this node now has a chance to transmit a message in its transmission queue.
- Lastly, the token is returned to the cluster-head node and the cycle is repeated again.

Gateways are nodes belonging to more than one cluster. These nodes have to switch their spreading code to the code used in a specific cluster to receive messages from that cluster. A situation may arise where a token is sent to a gateway node and the node is "tuned" to the spreading code of the other cluster. This causes a token to be lost and impacts message delivery adversely. If a token is lost, the token is regenerated by the cluster-head after a time-out.

Token scheduling can be used to give greater priority to neighbour nodes from which a packet was recently received. This enables traffic to be sent with less delay. The cluster-head offers more transmission chances to nodes with a higher priority.

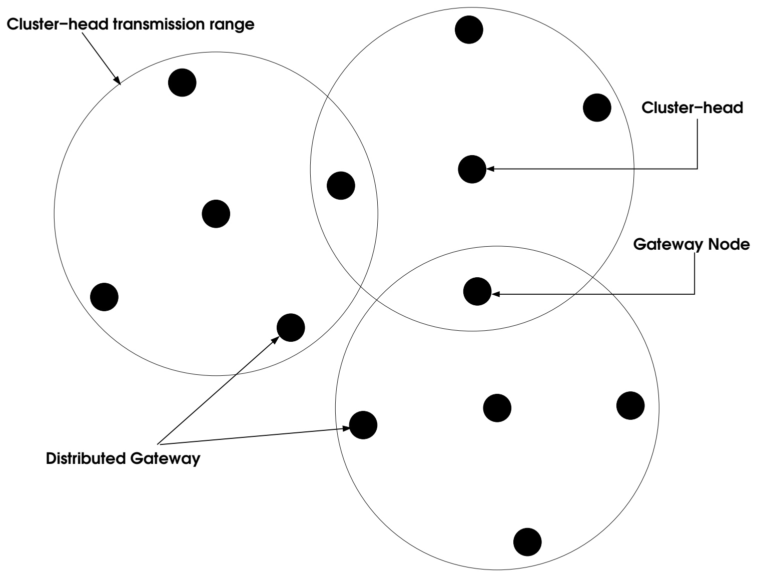


Figure 2.7: *Nodes grouped together in cluster topology.*

Routing

DSDV routing is used as a basis, because it provides loop free routes and prevents nodes from being updated with old routing information by using sequence numbers. DSDV is modified by using the unique property of cluster-heads that nodes in a cluster are one hop away from the cluster. The property can reduce the number of routes, because packets can now be routed to a cluster rather than a node. Each node has a cluster member table which is broadcast periodically. A node updates its cluster member table when new table information is received from its neighbour. The cluster member table maps a node address to an associated cluster-head address. Each node maintains a routing table, used to determine the next node on route to the destination cluster. When a packet is routed from one cluster to another the packet is routed through the common gateway of the clusters [12]. Sometimes clusters will form not sharing a common gateway node. The problem can be solved by creating a distributed gateway. The distributed gateway consists of a node in each of the different clusters that are neighbours [16].

2.5.9 Hierarchical State Routing

As the name implies Hierarchical State Routing (HSR) is another hierarchical routing protocol. Like in CGSR, nodes are grouped into clusters, but elected cluster-head nodes become part of new logical level. Members of the new logical level then elect cluster-heads for themselves and the cluster-heads again become part of the next logical level. This process repeats itself recursively until a single top level cluster is created. The protocol improves scalability by reducing the control overhead further. [31]

2.6 Other Related Work

This section will provide a short summary of related work on cluster based routing. Ben Lee et al. provides an extensive summary on works relating to cluster based routing in [22].

2.6.1 AODV with Passive Clustering

In the PC (Passive Clustering) protocol, AODV is extended to reduce the number of nodes flooding the network during the route acquisition phase. Only cluster-heads and selected gateways flood the network. The advantage of PC is that no explicit cluster setup phase or explicit cluster maintenance overhead is required, instead bits are added to MAC packets to indicate node state. Hello messages are required to maintain up to date cluster and link information, when no other messages was sent for some time. The First Declaration Wins mechanism (FDW) for cluster-head election prevents chain reaction reclustering. In FDW clustering the node that declares itself cluster-head first, becomes the cluster leader. [15]

2.6.2 Genetic Algorithm Applied to Route Optimization

This routing protocol optimises routes by using Genetic Algorithms (GAs). Clusters are set up by electing cluster-heads by assigning weights to certain node traits such as its connectivity, transmission range etc. Genetic algorithms are applied to route packets to improve load distribution. Each route is assigned a chromosome and ranked by a fitness function. After some time a new child chromosome is created with an improved route. [7]

2.6.3 Cluster Based Routing Protocol

Cluster Based Routing Protocol (CBRP) is an on-demand routing protocol using DSR's method of route discovery. Only cluster-heads and gateways forward RREQ and RREP packets. The protocol uses hello messages containing the whole neighbour table and cluster adjacency table and the node's own node state. Clustering allows this protocol to do local route repair between clusters and route shortening is enabled by examining the two hop node neighbourhood. [20]

2.6.4 Adaptive Routing using Clusters

Adaptive Routing using Clusters (ARC) is a hierarchical routing protocol that is combined with AODV. Special features of this protocol are a limited broadcast mechanism to reduce the number of redundant nodes that broadcast a packet and a local route repair mechanism, because more than one gateway links a cluster. Hello messages of the protocol contain the node's state, address and cluster leader table. To increase stability and reduce the ripple effect of cluster-head re-election, a cluster-head gives up its status if comes in range of another cluster-head. The original cluster must be a subset of the cluster it merges with if the cluster-head becomes an ordinary member node. [10]

2.7 Summary

The chapter provides literature relating to data networks. Background is given on wireless MAC-layer protocols comparing sender-initiated, receiver-initiated, reservation mechanism and schedule mechanism protocols.

The different routing disciplines of proactive routing and reactive routing are investigated and examples of each are provided. An in depth summary is provided on DSDV routing (proactive), which uses routing packets with a next hop and sequence number for each route. OLSR only uses selected nodes, called multipoint relays, to transmit routing overhead. Reactive routing protocols such as DSR and AODV use Route Request (RREQ), Route Reply (RREP) and Route Error (RERR) packets for determining routes and detecting route errors.

Hierarchical routing reduces the number of routes in the network by grouping nodes into clusters. Consequently, the nodes only need a route to the cluster. A summary of the CGSR protocol is given, focussing on the cluster generation, the MAC layer operation and the routing mechanism. The chapter ends with a summary on research relating to cluster based routing. An Internet Draft and two articles [15, 20, 10] improve the AODV protocol by using clusters and reducing the number of nodes broadcasting packets.

Chapter 3

Routing Protocol Design and Implementation

The chapter documents the important design choices and implementation environment of the routing protocol. First, the requirements used to make initial design choices are stated and secondly, the implementation environment of the OMNeT++ simulator is discussed. An overview of the development plan for the routing protocol is given. Implementation specifics of the protocol, i.e. the cluster generation, node states, routing tables, routing information distribution and the routing function are examined. The channel management method by the use of tokens is shown. A mechanism to promote the fair queueing of forwarded data messages, is presented. The chapter is concluded by a section on performance measurement.

3.1 Initial Design Choices

As stated in the introduction, the protocol should have the following outcomes:

- high densities of nodes must be handled effectively,
- nodes can make use of a high bandwidth,
- congestion must be avoided or reduced and
- quick rerouting must be possible if communication with a node is lost.

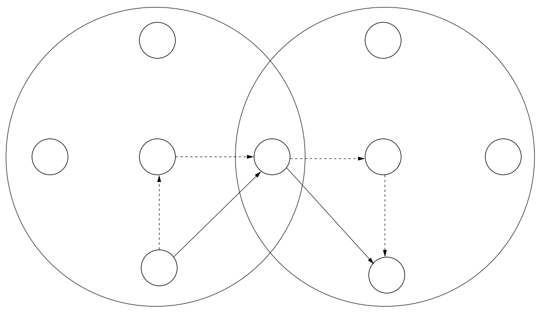
The Cluster-head Gateway Switch Routing (CGSR) protocol was chosen as a base for the following reasons. Cluster based protocols prefer immobile environments, because mobility creates extra overhead for these protocols [5]. The performance of cluster based (hierarchical) routing protocols increases when the node density is high, because node overhead is decreased [6]. CGSR is a proactive routing protocol, therefore routes are maintained and broken routes are repaired automatically. A link-by-link congestion control mechanism is used in CGSR [16].

3.1.1 CGSR Disadvantages and Proposed Improvements

CGSR, however has a few drawbacks. Firstly, routes are longer, because a node always routes a message to its *cluster-head* first and then to the correct *gateway*, instead of routing the message directly to the *gateway* (if the *gateway* is in range of the original node). A precursor to the CGSR, Destination Sequenced Cluster Routing (DSCR), allowed nodes to use the shorter route, but every node generates routing information [12]. The other disadvantage of the protocol is that every node has to broadcast its cluster member table periodically [5]. A cluster member table contains all the nodes in the network and their associated cluster.

Two changes are needed to shorten routes by hopping directly to the correct *gateway*. *Gateways* must share information on their local *cluster-heads* and the next-hop-cluster information of a destination must be available to nodes. The next-hop-cluster is the cluster that must be traversed next to reach the destination cluster.

Control overhead can be reduced by letting only selected nodes transmit the routing information and node-cluster associations. Other nodes only share information on critical nodes within two hops. Figure 3.1 shows the planned changes to the protocol.


(a) The original route chosen by CGSR is shown by the dashed line. The solid line route is the proposed improvement.

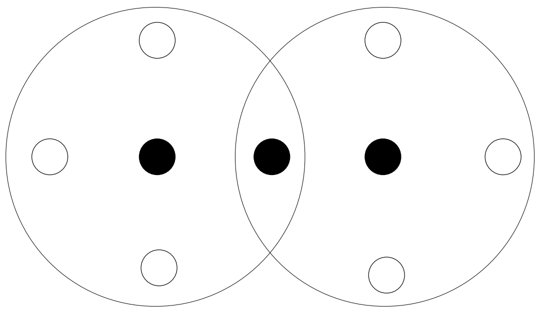

(b) Only selected nodes (filled nodes) generate full routing and node-cluster association information for reduced overhead. Other nodes only share information on critical nodes.

Figure 3.1: *Two improvements made to the protocol are shown.*

3.2 Implementation Environment

Complex systems' internal mechanisms are often studied with the help of simulations [8]. The protocol was implemented in the OMNeT++ discrete event network simulator. The OMNeT++ simulator was chosen for the following reasons:

- it is possible to separate code into discrete modules representing the logical structure of the network model [41],
- a Mobility Framework extension is available for the simulation of wireless networks [24],
- it is easy to extend previous modules due to the object-oriented nature of the simulator [41] and
- previous Masters projects at the University of Stellenbosch have used the simulator [26, 40].

The OMNeT++ simulator has three main components:

- modules implemented in the C++ programming language,
- the NED language specifying the logical module structure, gate connections between modules and input parameters and
- the omnetpp.ini configuration file that specifies simulation options and the values of the input parameters.

The simulator creates a visual representation of the wireless network. This makes testing and debugging the network easier. Figure 3.2 on the following page shows the visual representation of the network, generated by OMNeT++. Each node consists of various layer modules: application, network and Network Interface Card (NIC). The modules of the layers are shown in figure 3.3 on the next page. We focus on the network (routing) layer, the centre module in the previously mentioned figure.

OMNeT++ is a discrete-event simulator. Therefore state variables only change at discrete points in time. In the simulator events are scheduled and added to the Future Events List (FEL). All events in the FEL are in chronological order. Suppose events are scheduled at times $t_1, t_2, t_3, ..., t_n$ and $t_1 \leq t_2 \leq t_3 \leq ... \leq t_n$. The event at t_1 is to be executed next. The simulation clock will advance to time t_1 and remove it from the FEL and execute the associated event. When a event is executed, new future events can be scheduled and must be inserted into the FEL at the appropriate position. The simulation clock then proceeds to the next scheduled event. This process is repeated until there are no more scheduled events. Figure 3.4 on page 27 shows a time line with scheduled events. [8]

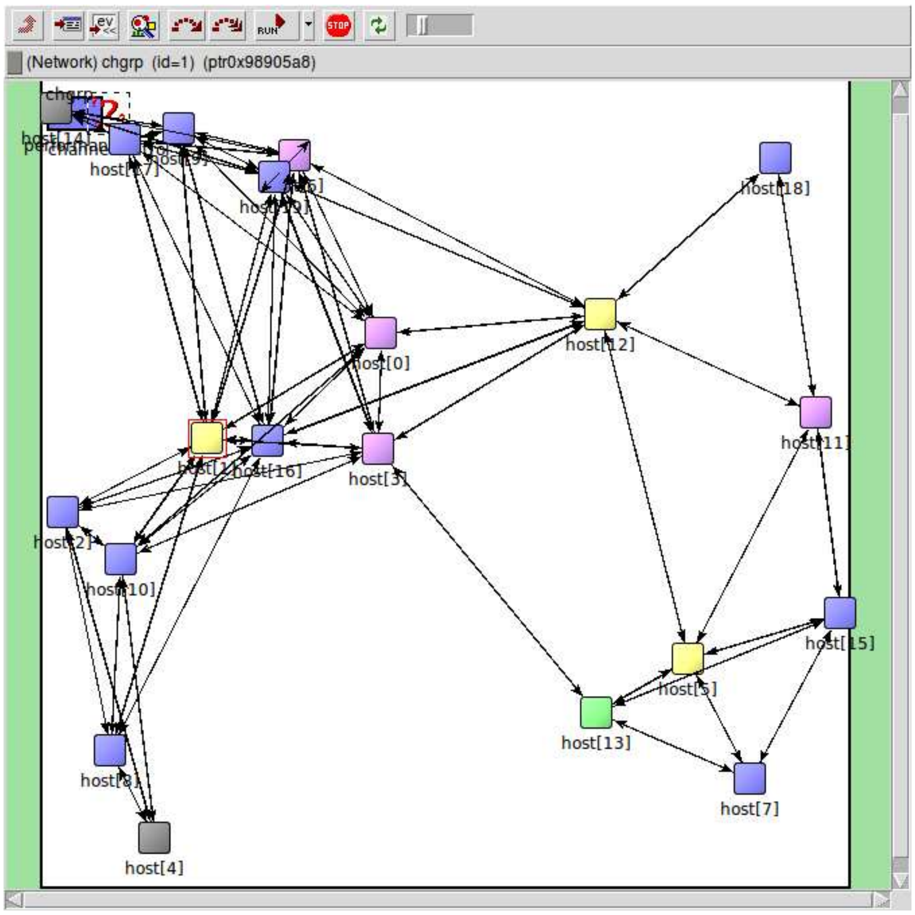


Figure 3.2: *Graphical representation of a network in OMNeT++.*

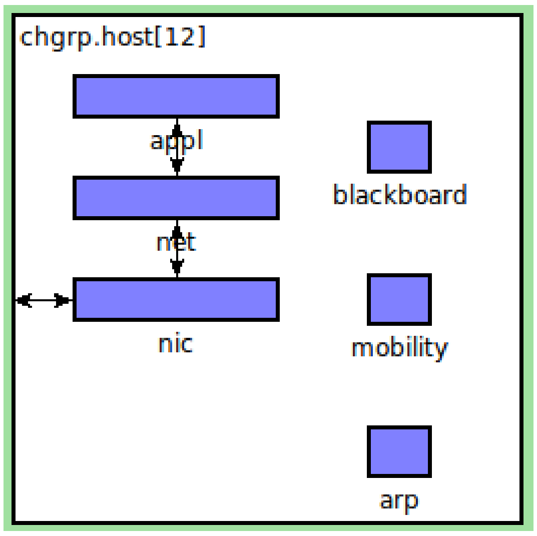


Figure 3.3: *Simple modules in host implementing the various communication layers.*

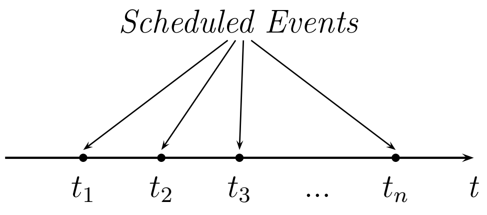


Figure 3.4: *Timeline representing the future events list.*

3.2.1 Physical Layer Transmission Parameters

OMNeT++ allows the transmission parameters to be altered. To make the simulation more realistic the transmission parameters of a IEEE 802.11b Wireless Fidelity (Wi-Fi) transceiver were chosen. Figure 3.5 shows the wireless device. The following values were used:

- Receiving sensitivity: −85 dBm,
- Output power: 15 dBm (31,6 mW) and
- Antenna gain: 2 dBi.

Figure 3.5: *Wireless 802.11 transceiver. Reproduced from [4].*

3.2.2 Scaling the GUI Distance

The parameters chosen in the previous subsection causes the transmission power, P_t, to be insufficient, because 1 pixel in the graphics output (see figure 3.2 on page 26) represents 1 meter of distance. It was decided to scale the distance represented by a pixel, by making it smaller and increase the transmission power accordingly. OMNeT++ uses a variation of Frii's transmission equation with path loss exponent of $\alpha = 4$.

$$P_r = P_t \left(\frac{\lambda_W^2}{16\pi^2 d_1^\alpha} \right) \tag{3.1}$$

Suppose the distance of a pixel is scaled with the following equation.

$$S_F.d_1 = d_2 \tag{3.2}$$

- S_F is a scaling factor.
- d_2 is the new pixel distance.

Now the equation must be rewritten in order for d_2 to replace d_1.

$$P_r = P_t \left(\frac{\lambda_W^2}{16\pi^2 (\frac{d_2}{S_F})^\alpha} \right) \tag{3.3}$$

$$P_r = (P_t.S_F^\alpha) \left(\frac{\lambda_W^2}{16\pi^2 d_2^\alpha} \right) \tag{3.4}$$

The real transmission power can now be replaced by $P_t.S_F^\alpha$ reducing the distance represented by a pixel with a factor of S_F.

3.3 Routing Layer Overview

To develop the routing protocol there are certain logical tasks to be completed in a specific order. Figure 3.6 on the next page shows the development steps for the routing protocol. The first objective of the routing protocol should be to form clusters. The nodes should elect a *cluster-head* among themselves. After clusters have been established, the *cluster-heads* will be known and routing information can be spread through the network. Concepts from the DSDV routing protocol will be used, but only *cluster-heads* and other selected nodes will forward the routing information. Routes will be determined by using the property of the *cluster-head*: nodes in a cluster is one hop away from its *cluster-head* (see section 2.5.8). When the routes to all the nodes are known, data traffic can be generated on the network. Lastly the traffic can be analysed with respect to latency, throughput and control overhead.

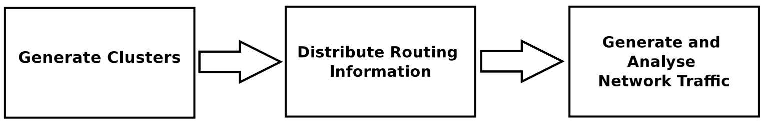

Generate Clusters
Distribute Routing Information
Generate and Analyse Network Traffic

Figure 3.6: *A flow diagram showing the steps for developing the routing protocol.*

3.4 Cluster Generation

In section 2.5.8 two clustering algorithms are mentioned. Since the network being studied is immobile and the chances of two *cluster-heads* being in transmission range of one another is small, the highest connectivity clustering algorithm was chosen. The only time two *cluster-heads* will be in transmission range of one another is when both are elected simultaneously. When such a situation occurs the *cluster-heads* will challenge one another and the *cluster-head* with the least connections will give up its *cluster-head* status [12]. If both *cluster-heads'* number of connections are equal, the *cluster-head* with the highest id (address) will give up its *cluster-head* status.

The *cluster-head* election process will now be discussed. When a network is started every node has the status of *unassigned.* If the node is in the *unassigned* state it broadcasts periodic hello messages. Figure 3.7 shows the nodes transmitting hello messages.

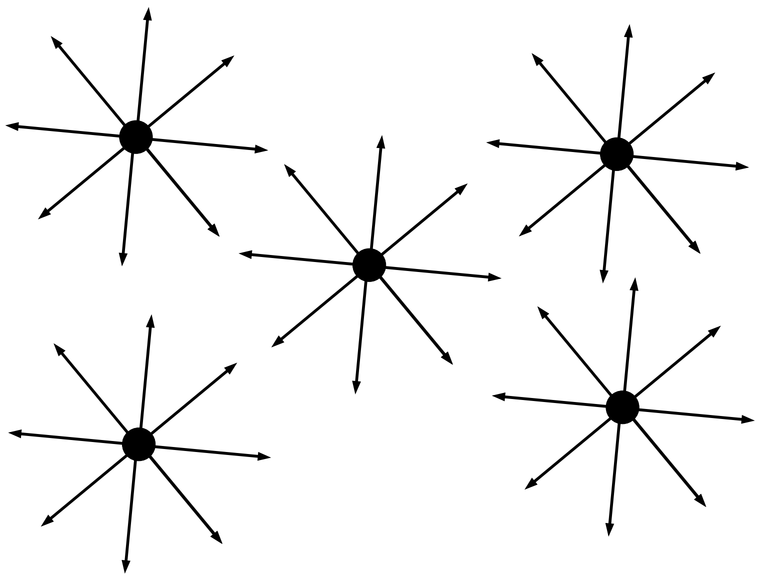


Figure 3.7: *Nodes constantly broadcasting hello messages.*

These hello messages are constantly generated after a short time interval, $t_{short\ hello}$. The short time interval is calculated by:

$$t_{short\ hello} = k \times (1 + 0,2 \times X) \tag{3.5}$$

- X is a random variable with a uniform distribution in the range $[0, 1)$.
- k is a constant value set to 0,04 s.

The hello message contains various data fields, but the following fields are important with respect to the cluster generation algorithm:

- sequence number
- node state
- number of connections
- election address

The sequence number is a unique number and is incremented with each hello message transmission. Sequence numbers help neighbouring nodes to keep track of how many hello messages are lost. A node state field declares the status of the node. The number of connections is the number of neighbours the node can hear (number of entries in the neighbour table, section 3.6.1). The election address contains the id of the neighbouring node with the most connections.

Now, each time a message is received by a node and the election address field is the same as its own address, the *electionCondition* counter is incremented. If *electionCondition* later exceeds a certain election threshold the node becomes a *cluster-head*. The election threshold is calculated by:

$$election\ threshold = k_{threshold} \times connections \tag{3.6}$$

- $k_{threshold}$ is set to 2 .

The threshold prevents nodes from being promoted to *cluster-head* too quickly and situations where more than one *cluster-head* is elected simultaneously within transmission range of one another.

When a node is promoted to *cluster-head*, the node state field advertised in the hello message is updated accordingly. Neighbour nodes hearing a message from the *cluster-head*, update their own status to *assigned*. Figure 3.8 on the next page shows how a cluster is formed, when a *cluster-head* broadcasts a hello message. Nodes not assigned to the new cluster (uncovered nodes), continue to generate hello messages periodically with a short time interval in between. The nodes forming part of the new cluster, now generate hello messages periodically with a longer time interval of 0,1 s. Cluster members' connection with the *cluster-head* and the *cluster-heads'* connection with its cluster members is maintained by the hello messages. The cluster generation process is followed until every node is associated with a cluster. The long setup time of a cluster can be criticised, but it is allowable since no large topology changes are expected from stationary nodes. In the next section node states will be described.

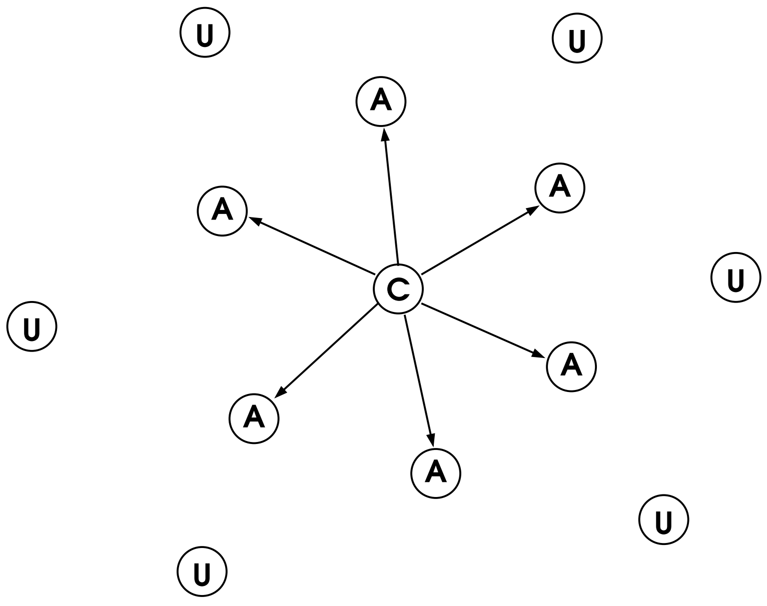


Figure 3.8: *The diagram shows how clusters are formed. "C" is the* cluster-head *node, "A's" are the* assigned *nodes and "U's" are the* unassigned *nodes.*

3.5 Node States

As the clusters are formed, the status of the nodes change and so does their respective functions. Figure 3.2 on page 26 shows the nodes displaying different colours. Each colour represents a different node state. A node can have the following states:

- *offline (red)*
- *unassigned (cyan)*
- *assigned (blue)*
- *cluster-head (yellow)*
- *gateway (purple)*
- *distributed gateway (green)*

- *satellite node (black)*

When a node goes online it starts with *unassigned* status. In the *unassigned* state the node tries to elect a *cluster-head* by broadcasting hello messages, as described in the previous section. A node goes to the *assigned* state if it can hear one *cluster-head.* Nodes can start to send data messages as soon as they receive routing information from the *cluster-head.* A node becomes a *gateway* if it can hear two *cluster-heads.* Data is routed from one cluster to another through a common *gateway* node. Data can also be routed through a *distributed gateway* node. A *distributed gateway* is a node with a neighbour in different cluster, but the neighbour is not a member of the node's own cluster. *Satellite nodes* are hosts without *cluster-head* as a neighbour, but only hear nodes in transmission range of a *cluster-head.* In this way *satellite nodes* also form part of a cluster, but *satellite nodes* are however still uncovered and try to elect a *cluster-head* by broadcasting hello messages. An uncovered node is defined as a node without a *cluster-head* as neighbour.

Figure 3.9 on the next page shows how node states can change. Each time a hello message is received from another node, the *determineNodeState* function is executed by the receiving node. The function uses information in the neighbour table of the node to determine its state. Algorithm 1 on page 34 shows the pseudo code of the *determineNodeState* function.

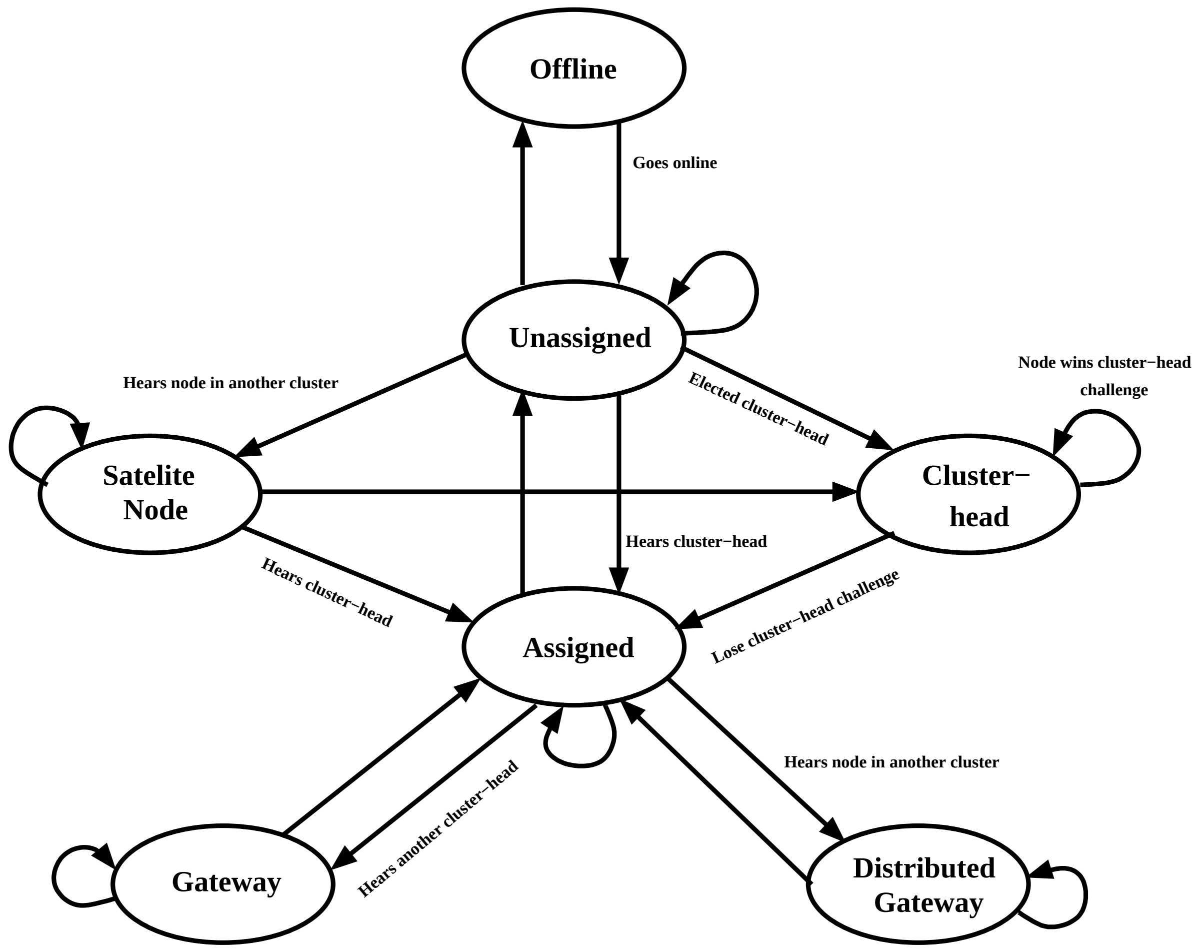
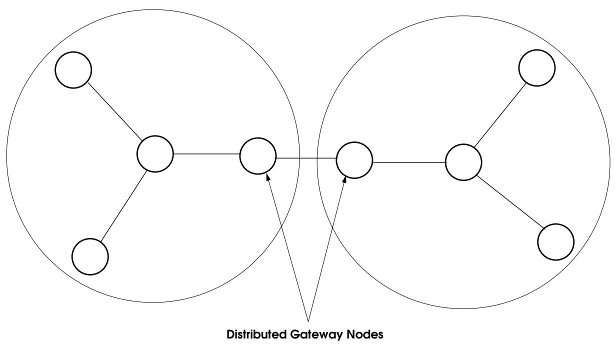

Offline
Goes online
Unassigned
Hears node in another cluster
Elected cluster−head
Node wins cluster−head challenge
Satelite Node
Cluster−head
Hears cluster−head
Hears cluster−head
Lose cluster−head challenge
Assigned
Hears node in another cluster
Hears another cluster−head
Gateway
Distributed Gateway

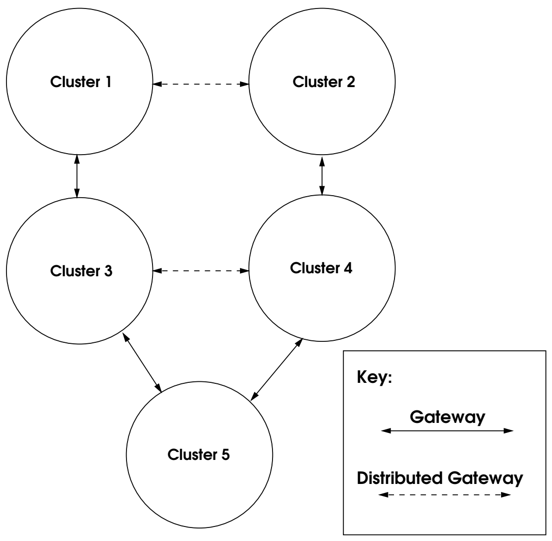
Figure 3.9: *Node state diagram.*

Algorithm 1 Node state algorithm

```
function determineNodeState
    $numClusterHeads \leftarrow determine\ number\ of\ cluster\_heads\ in\ range$
    if $ownNodeState = cluster\_head$ then
        if $numClusterHeads > 0$ then                ▷ Tests for cluster-heads in range
$neighbour \leftarrow get\ cluster\_head\ neighbour\ information$
            if lose challenge to neighbour then
                give up cluster_head status
            else return cluster_head
            end if
        else if $numClusterHeads = 0$ then return cluster_head
        end if
    end if
    if $numClusterHeads > 1$ then
        return gateway
    else if $numClusterHeads = 1$ then
        if test distributed gateway then return distributed gateway
        else return assigned
        end if
    else if $numClusterHeads = 0$ then
        if $neighbours\ part\ of\ cluster$ then return satellite node
        else return unassigned
        end if
    end if
end function
```



(a) Distributed gateway connecting two clusters [16].



(b) Distributed gateway reducing path length. If only gateways were used, the messages routed from cluster 1 to cluster 2, would have to be routed via clusters 3, 5 and 4.

Figure 3.10: *Distributed gateway functions.*

3.5.1 Distributed Gateway

A *distributed gateway* enables communication between two neighbouring clusters with no common *gateway* node. The path length for packets can even be reduced in some instances by using a *distributed gateway.* Figure 3.10 shows how a *distributed gateway* can connect clusters and how path length can be reduced by using a *distributed gateway.*

3.5.2 Satellite Node

The state of *satellite node* was defined to solve a unique problem. Instances rarely occur where there is a single uncovered node at the edge of the network. The rest of the nodes already form part of a cluster. Only uncovered nodes can take part in the election, therefore the single uncovered node cannot elect itself as *cluster-head.* If the node detects other nodes that are part of a cluster, it knows a *cluster-head* is two hops away. The node then sets its status to *satellite node.* Information about the *satellite node* is then relayed, in a hello message, by its neighbours to their *cluster-head.* Figure 3.11 on the next page shows a *satellite node* joining a cluster by relaying its information to a *cluster-head.*

It can be argued that the role of the *satellite node* could be replaced by a *cluster-head,* but this will create an extra entry in the routing table and may increase routing overhead.

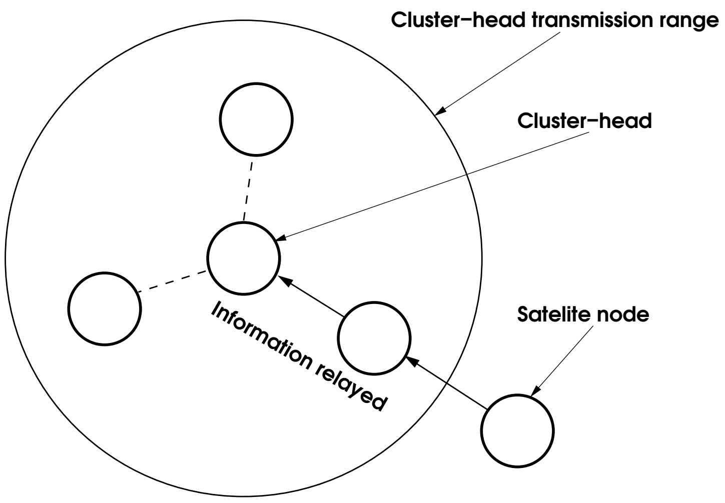


Figure 3.11: *Satellite node information relayed to the cluster-head.*

3.6 Routing Tables

Routing tables enable nodes to determine the next hop to a destination. The main routing tables are the neighbour table and the cluster table set. Information of the node's immediate environment is contained in the neighbour table. The cluster table set contains each node's associated *cluster-head* address and the routing information to the cluster.

Using various routing tables in the protocol may seem redundant, but the tables organise the routing information by different criteria. This simplifies the search for relevant routing information. For example, the gateway table records information on *gateway* nodes. Now *gateways* to other clusters can be found simply by inspecting the gateway table, rather than searching the whole neighbour table for nodes with a *gateway* status and then determining which neighbour cluster they link.

Every time a hello message is received, the neighbour table is updated with the information from the neighbour node. The cluster-head, gateway and satellite node tables are updated from the neighbour table. Only *cluster-head* nodes use the neighbour cluster table. The neighbour cluster table is updated by the gateway table.

Each time a cluster table message is received, the cluster table set is updated. The network node table is updated from the cluster table set. Figure 3.12 on the following page shows the routing table update mechanism.

3.6.1 Neighbour Table

As the name implies, the neighbour table contains information on the nodes within transmission range of the node. For each neighbour the following variables are recorded:

- neighbour address;

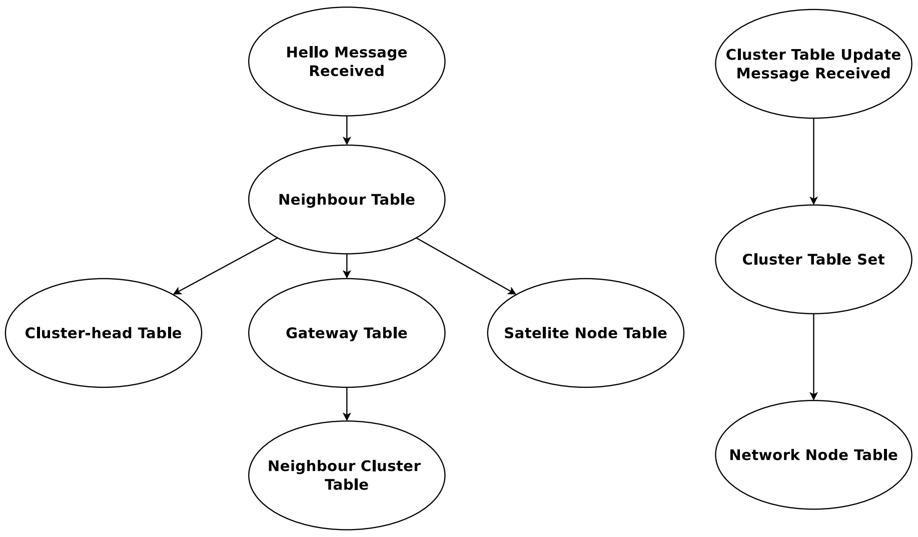


Figure 3.12: *Update mechanism for routing tables.*

- connections, the number of neighbours the node has;
- node state, the current state of the node as described in section 3.5;
- link state, the stability of the link;
- Time To Live (TTL), the time a connection has left before becoming *unstable*;
- hello messages received;
- messages pending, the number of messages in the transmission queue;
- last hello message sequence number;
- time-outs, number of times the link has timed-out;
- number of *cluster-heads*, number of *cluster-heads* nodes within two hops;
- array of *cluster-head* addresses, addresses of *cluster-heads* within two hops;
- array of direct *cluster-head*, boolean field indicating if the *cluster-head* is a direct neighbour or two hops away;
- number of *satellite nodes*, number of *satellite nodes* within two hops;
- array of *satellite node* addresses, addresses of *satellite nodes* within two hops; and
- array of direct *satellite node*, boolean field that indicates if the *satellite node* is a direct neighbour or two hops away.

Link Stability

To ensure that a link is stable, a message must be received from a neighbour node within a specified time. When a hello message is received from a host for the first time, its entry is created in the neighbour table and the Time To Live (TTL) field is set to its maximum value of 5 and the link state is set to *stable*[1]. Then periodically after a constant time, $t_{decrease\ link\ TTL} = 0{,}1\,\mathrm{s}$, has passed, the TTL field is decremented. If the TTL field eventually reaches 0 , the link state associated with the neighbour is set to *unstable*. To keep the link of a neighbour *stable*, a message must be received from the neighbour to restore the TTL to its maximum value. The long hello message generation time and the TTL decrement time are equal.

3.6.2 Cluster-head Table

Information related to the *cluster-heads* within two hops of a node is recorded in the cluster-head table. For each *cluster-head* the following data is maintained:

- *cluster-head* address; and
- direct *cluster-head*, boolean field indicating if the *cluster-head* is a direct neighbour or two hops away.

As stated previously, the cluster-head table is updated from the neighbour table. The cluster-head table update process will now be explained. First, all entries currently in the table are validated, by checking the link with the neighbour providing the entry, is still *stable*. Any entries without a confirmed stable link are removed from the table. Then the cluster-head table is updated with information from neighbours that are *cluster-heads*. Lastly, if neighbours have any direct *cluster-heads*, the table is updated with those addresses and the boolean field is set to indicate the *cluster-head* is two hops away. Algorithm 2 on page 39 gives the method for updating generic tables, like the cluster-head table.

The content of the cluster-head table is included in the hello message of the node.

3.6.3 Gateway Table

Neighbouring *gateways* or *distributed gateways* are summarised in the gateway table. The information in the table is relative to the perspective of the node owning the table. The table helps nodes to determine which *gateways* link to which clusters. For a gateway node entry the following data is stored:

- distributed gateway, boolean field indicating if the node is a *gateway* or *distributed gateway*;

[1]Only *stable* links are used in routing tables.

- number of links to clusters, number of different neighbour clusters the *gateway* node links with the current cluster;
- links to clusters array, the array contains the *cluster-head* addresses of the neighbour clusters;
- direct link to cluster array, the boolean array stating if the *cluster-head* of the neighbour clusters is one hop or two hops away from the *gateway* node; and
- number of connections, the number of neighbours seen by the *gateway.*

The gateway table is updated from the neighbour table with algorithm 2. Figure 3.13 displays the routing information represented by the gateway table.

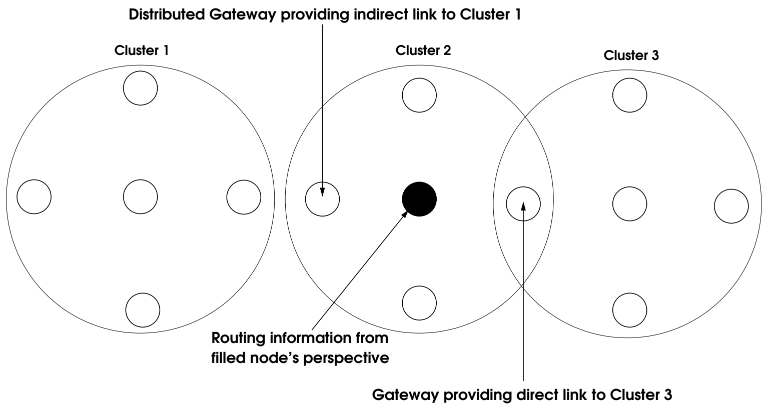

Distributed Gateway providing indirect link to Cluster 1
Cluster 1
Cluster 2
Cluster 3
Routing information from filled node's perspective
Gateway providing direct link to Cluster 3

Figure 3.13: *The diagram presents a visual description of information in the gateway table. The perspective of the filled node is used.*

Algorithm 2 Generic table update algorithm

```
procedure UPDATETABLE
    for each node entry in table do
        if node entry invalid then
            remove node entry from table
        end if
    end for
    for each node in neighbour table do
        if neighbour node entry has certain a attribute then
            add neighbour node to generic table
        end if
    end for
end procedure
```

3.6.4 Satellite Node Table

Satellite nodes are special in the sense that they belong to a cluster, but they are two hops away from the *cluster-head* node. For this reason it was decided to create a satellite node table for easy reference to *satellite node* routes. Each *satellite node* entry contains the following fields:

- direct *satellite node*, boolean field indicating if the satellite is a direct neighbour or two hops away;
- service node address, the address of the preferred intermediate node, that relays information, if the *satellite node* is two hops away;
- number of intermediate nodes, number of intermediate nodes between current node and *satellite node*; and
- intermediate node addresses array, array of maximum 4 intermediate node addresses.

The satellite node table is updated from the neighbour table with algorithm 2. *Satellite nodes* included in the table, must have the attribute of being either one or two hops away from the node owning the satellite node table. Figure 3.14 on the following page shows the intermediate nodes of a *satellite node*.

Satellite service node

The satellite service node is the intermediate node between the *cluster-head* and *satellite node* with the most neighbour connections. A satellite service node forwards the routing information to the *satellite node*. Token passing (discussed in section 3.9) is done implicitly via the satellite service node.

3.6.5 Neighbour Cluster Table

The neighbour cluster table helps to determine a designated gateway for each neighbouring cluster. Only *cluster-heads* have a neighbour cluster table. A designated gateway is chosen by the *cluster-head* and forwards routing information to its corresponding neighbour cluster. The neighbour cluster table is updated from the gateway table. *Gateways* with the most neighbours are selected as the designated gateway. If there is no direct link to a neighbour cluster through a *gateway*, the *distributed gateway* with the most neighbours is selected as the designated gateway.

3.6.6 Cluster Table Set

The cluster table set contains the main routing information of destination nodes. Each cluster in the network has a corresponding cluster table in the set. From now on the *cluster-head*'s address of each cluster will be referred to as the *cluster address*. To damp routing

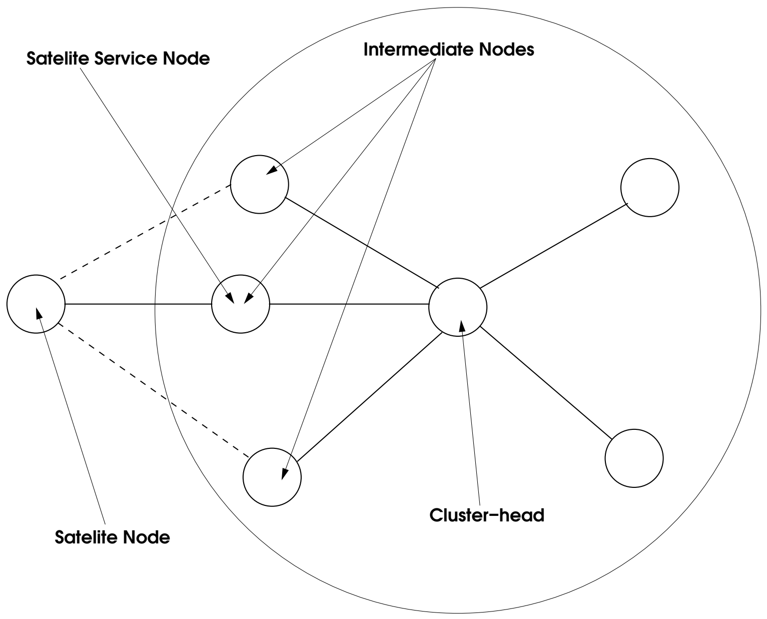


Figure 3.14: *Intermediate nodes of* satellite node.

fluctuations the cluster table keeps pending routing information. The routing fluctuation damping and the *pending* fields will be explained in section 3.6.6. Each cluster table records the following for each cluster:

- *cluster-head* address, the *cluster address*;
- reachable through node address, the address of the next hop to the cluster;
- next-hop-cluster, the next cluster that must be traversed to reach the destination cluster;
- current cluster, the *cluster address* of the node owning the cluster table set;
- hops to destination cluster, the number of hops needed to travel to the destination's *cluster-head*;
- node address array, the addresses of nodes belonging to the cluster;
- cluster table sequence number;
- reachable through node address (pending);
- next-hop-cluster (pending);
- current cluster (pending);
- hops to destination cluster (pending);

- cluster table sequence number (pending); and
- Time To Live (TTL), the time the cluster table is still valid.

Each node updates its own cluster table set with the information in a cluster table packet. A cluster table entry, with sequence number s_L, is updated if the sequence number s_R of the cluster table in the update packet, satisfies the following inequality: $s_R > s_L$. If the sequence numbers are equal, $s_R = s_L$, a cluster table entry can also be updated, but the update message's cluster table must have a smaller hop count. This implementation of sequence numbers agrees with the DSDV routing protocol [32] discussed in section 2.5.2. *Cluster-head* nodes, however create their own entry in the cluster table set and updates the entry with a new sequence number periodically. The entry created by the *cluster-head* consists of all its neighbours and *satellite nodes*.

The TTL field in the cluster table works the same way as the link stability mechanism explained in section 3.6.1. If a cluster table has not been renewed by an update message and times out (when the TTL field reaches zero), the cluster table entry is removed from the cluster table set. Figure 3.15 describes and displays the information of a cluster table entry.

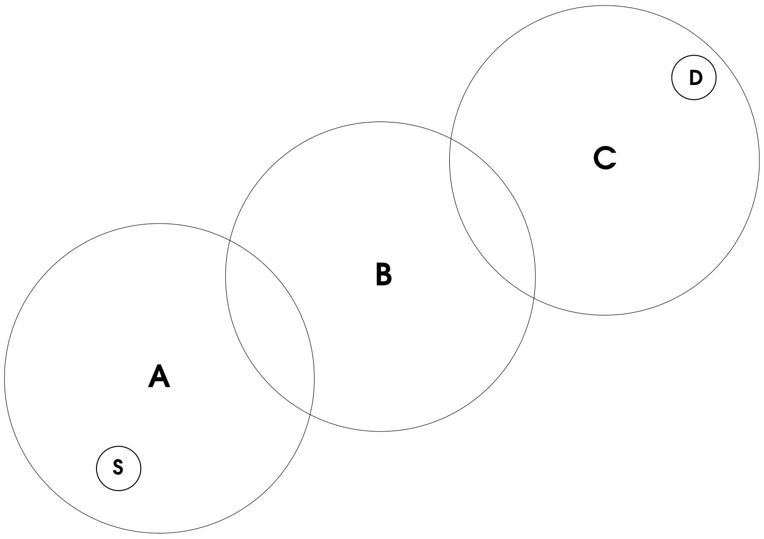


Figure 3.15: *Suppose a node "S" wants to route data to node "D". Then from the perspective of node "S", cluster "A" is the* ***current cluster****, cluster "B" is the* ***next-hop-cluster*** *and cluster "C" is the* ***destination cluster****.*

Damping Routing Fluctuations

To prevent routes from suddenly changing, routing changes are damped. Now if a single update packet is missed from an original node and a more recent update packet is received from another node, the sudden change is avoided. The routing information of the other node is stored in the pending fields. If five consecutive updates are missed from the original node, the main routing fields are replaced by the pending routing fields.

3.6.7 Network Node Table

The network node table contains a list of all the nodes in the network that are currently routable. Searching for routable nodes is simplified by using the network node table, because the table links each node's address to its associated clusters' addresses. Each network node entry contains the following information:

- number of *cluster-heads*, the number of clusters to which the node belongs; and
- *cluster-head* addresses array, the addresses of clusters to which the node belongs.

3.7 Routing Information Distribution

Routing information is distributed in a way that reduces the number of nodes forwarding routing information. In some sense the method is similar to the MPRs of OLSR, as explained in section 2.5.3. Only selected nodes forward the routing information.

First, the *cluster-head* generates its own cluster table, listing its neighbours and *satellite nodes*. All the valid cluster tables in the cluster table set of the *cluster-head* are broadcast. The broadcast message is called the cluster table packet. In the header of the cluster table packet there is a forwarder nodes list. The forwarder nodes are selected nodes that forward the cluster tables to neighbouring clusters and local satellite nodes. Designated gateways and satellite service nodes of a cluster are the selected forwarder nodes for a cluster. If a node receives a cluster table packet and its address is in the forwarder nodes list, the forwarding responsibility of the node is "switched on". The forwarding responsibility of a node is "switched off" if a node receives cluster table packet and its address is no longer in the forwarder nodes list. Figure 3.16 on the following page shows the distribution of routing information.

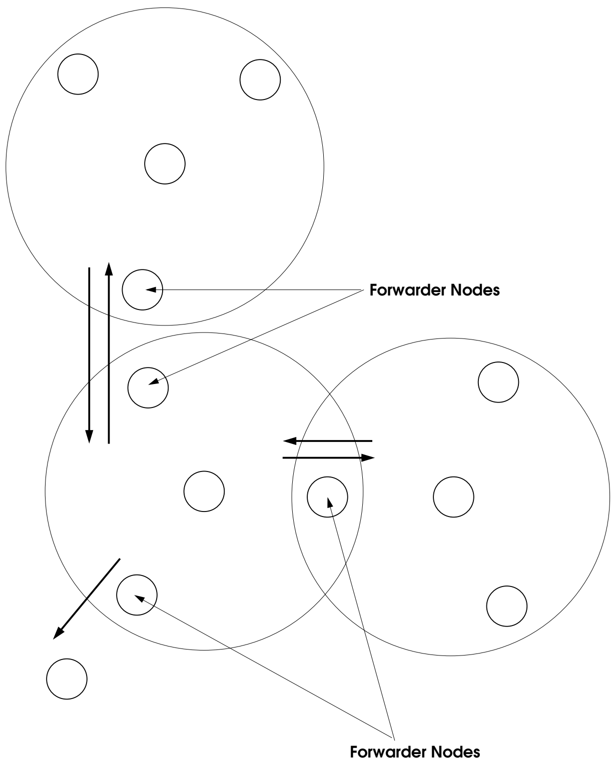


Figure 3.16: *The routing information is distributed to neighbouring clusters and* satellite nodes *through the forwarding nodes.*

3.8 Routing Function

The routing function determines the next hop to a destination node. Information in the various routing tables are used to select the best next hop. The routing function can be divided into two parts. One part determines if the destination is available locally, for example if the destination is a neighbour or *satellite node*. The second part is utilised if the destination node is part of a remote cluster. If the node is part of more than one remote cluster, the nearest cluster is selected as the destination. Figure 3.17 on the next page displays a flow chart indicating how the next hop address is determined.

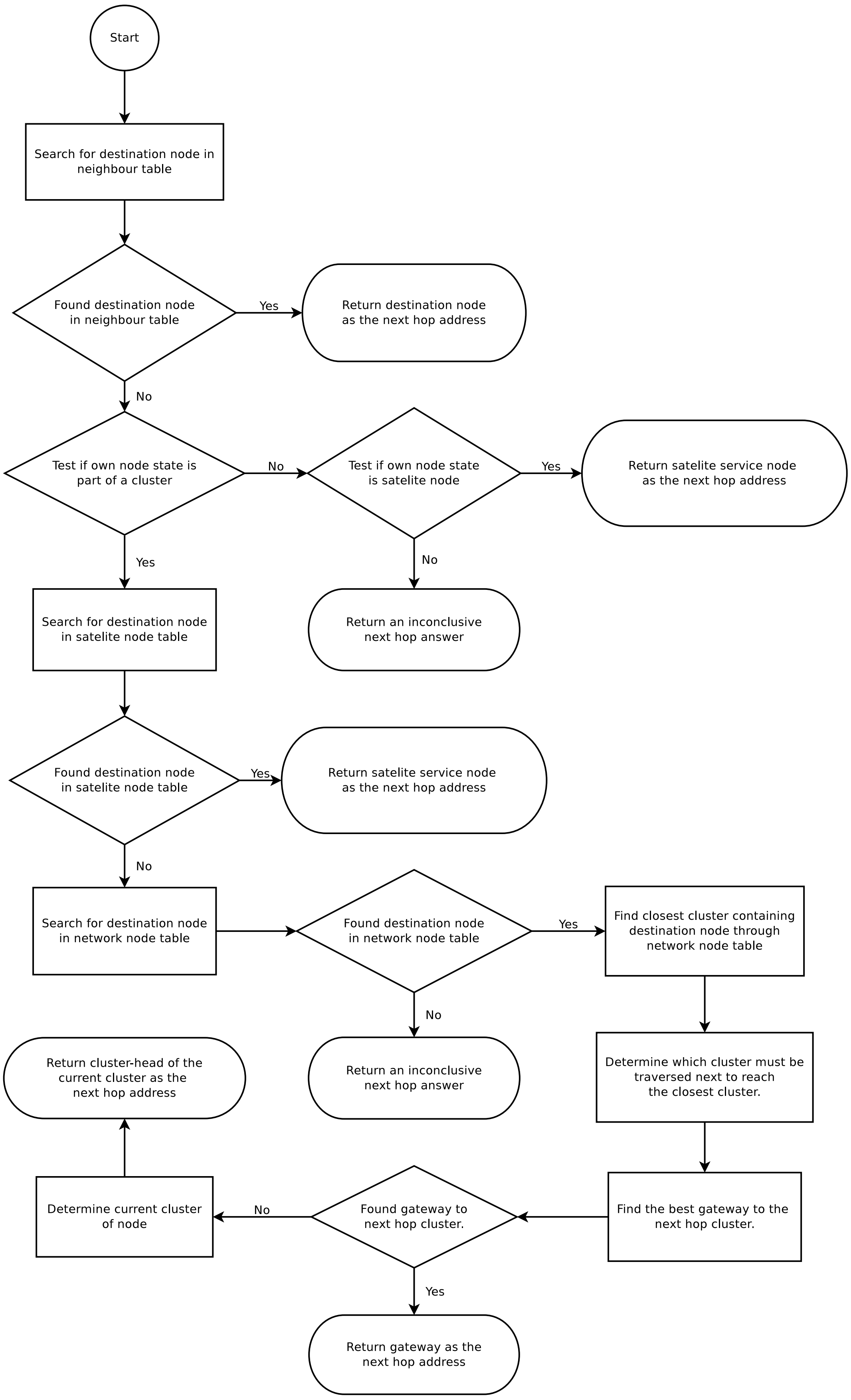

Start
Search for destination node in neighbour table
Found destination node in neighbour table
Yes
Return destination node as the next hop address
No
Test if own node state is part of a cluster
No
Test if own node state is satelite node
Yes
Return satelite service node as the next hop address
Yes
No
Search for destination node in satelite node table
Return an inconclusive next hop answer
Found destination node in satelite node table
Yes
Return satelite service node as the next hop address
No
Search for destination node in network node table
Found destination node in network node table
Yes
Find closest cluster containing destination node through network node table
No
Return cluster-head of the current cluster as the next hop address
Return an inconclusive next hop answer
Determine which cluster must be traversed next to reach the closest cluster.
Determine current cluster of node
No
Found gateway to next hop cluster.
Find the best gateway to the next hop cluster.
Yes
Return gateway as the next hop address

Figure 3.17: *Determine next hop function flow diagram.*

3.9 Token Scheduling

Token scheduling [12] was implemented for a measure of congestion control. Since *cluster-heads* transmit more routing information and can be chosen as a route, if no shorter route can be found, *cluster-heads* are given more transmission chances. The focus of the token scheduling was to give each node a fair chance to transmit its message in the cluster. A token packet is always broadcast and it is noted that different clusters do not use different spreading codes as per the original protocol discussed in section 2.5.8. Although a decrease in performance can be expected with this implementation decision, the implementation is greatly simplified.

Cluster-heads generate a token and pass the token to their neighbours. Before a *cluster-head* transmits a token message it transmits a data message, if it has any. Each time a neighbour receives a token, the node is given a chance to transmit one data message. A data message is sent via the CSMA/CA mechanism to the next hop node. After the neighbour has transmitted the data message, the token is sent back to the *cluster-head* and the process is repeated. The *cluster-head* cycles through its neighbours. At the end of the cycle the token is “passed” to the *cluster-head*, to give the *cluster-head* a fair chance.

In the process of passing tokens, a token can be lost. A lost token can be regenerated by the *cluster-head* after a time-out. To determine the time-out, the total time passing from when the *cluster-head* sends the token message, the neighbour sends a data message and the token is returned to the *cluster-head*, is calculated. The following equation is used:

$$t_{time-out} = 2 \times t_{token\ transmission} + t_{data\ transmission} \tag{3.7}$$

$t_{time-out}$ equates to $0,00726s$, but the final time-out value of $t_{time-out} = 0,0075s$ is chosen. Note that an extra mean backoff time has been added to each packet transmission time, as a time buffer. The details of the equation will be discussed in the next chapter.

If a neighbour has significantly increased traffic, the token can be passed to the neighbour more than once in a cycle. The following equation is used to determine the number of transmission chances.

$$transmission\ chances = \frac{m_i}{\max(1, \min(m_1, m_2, ..., m_n))} \tag{3.8}$$

- m_i is the number of messages in the transmission queue of the neighbour whose transmission chances are considered.
- $m_1, m_2, ..., m_n$ are the number of messages in each cluster member’s transmission queue.

A *cluster-head* determines the number of messages in the transmission queue of a neighbour, by inspecting the *messages pending* field in the token message returned by a neighbour. The number of messages in a transmission queue are the messages in the personal and general message buffer summed. Message buffers will be explained in section 3.10.

3.9.1 Implicit Token Passing to *Satellite Nodes*

Since *satellite nodes* are not directly in range of a *cluster-head*, a solution must be found to include *satellite nodes* in token passing. It was decided to pass the token implicitly to the *satellite node* when the satellite service node returns the token to the *cluster-head*. In essence token passing continues as if there were no *satelite node*. A boolean field inside the token message is set which indicates that the token is being passed implicitly. When a satelite node receives such a token it only returns the message to its satelite service node. Figure 3.18 shows how tokens are passed to neighbour nodes and includes a case where a token is passed implicitly to a *satellite node*.

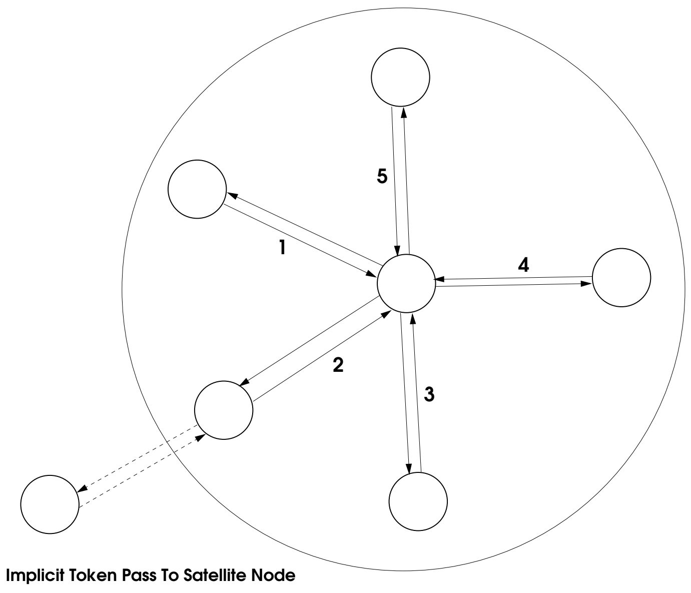


Figure 3.18: *Implicit token passing to* satellite node*. The dashed line shows the implicit token pass.*

3.10 Weighted Fair Queueing with Message Buffers

Message buffering was implemented with the idea to prevent congestion at nodes. Since all nodes in an ad hoc network route one another's messages, it is important that no node abuses its position in the network to route its own messages. If a node generates a large amount of traffic it can congest its own transmission queue with its own messages.

Weighted fair queuing was implemented to solve the problem [33]. Messages arriving at the node and messages generated by the node, are separated into two buffers. Messages generated by the node itself enter the personal message buffer of the node. Other messages received from neighbours that need to be forwarded enter the general message buffer directly. Figure 3.19 on the next page shows the personal and general message buffers. Every time a token is received, a message is transmitted from the general message buffer or if the general buffer is empty, a message is transmitted from the personal buffer.

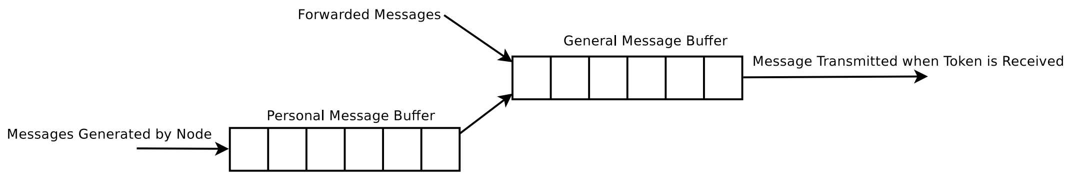

Forwarded Messages
General Message Buffer
Message Transmitted when Token is Received
Personal Message Buffer
Messages Generated by Node

Figure 3.19: *The personal message buffer and the general message buffer.*

Now let $m_{adjacent} = (\textit{number of nodes in adjacent cluster})$. For every $m_{adjacent}$ forwarded messages in the general message buffer, one message of the personal message buffer proceeds to the general one. Figure 3.20 shows a data flow from one cluster to another and which cluster is considered the adjacent cluster.

For each cluster a node is part of, it has a set of a general and personal message buffers. *Gateway* nodes therefore have more than one message buffer set. The different message buffer sets of *gateways* separate the flow of messages to different clusters. *Satellite nodes* with a *cluster-head* two hops away also have a message buffer set. The message buffer set is called the cluster transmission handler.

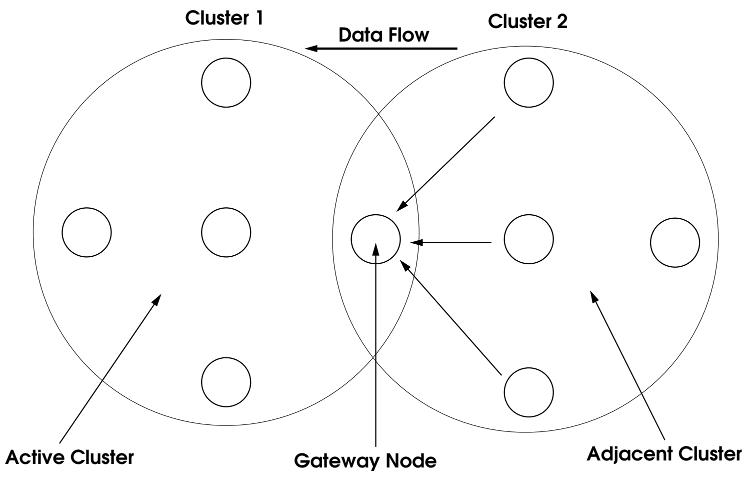


Figure 3.20: *The diagram shows a data flow from cluster 2 to cluster 1. If the gateway node functions in cluster 1, then cluster 2 is considered the adjacent cluster.*

By queueing data messages in the buffer at the routing layer, the message queue of the MAC-layer is kept empty most of time. This ensures that when a routing message (hello message or cluster table message) is generated and passed to the MAC-Layer, it is sent as soon as the channel is available. In other words, routing messages receive preference and do not have to wait for data messages to be transmitted first.

3.10.1 Choosing Between Different Cluster Transmission Handlers

Nodes part of more than one cluster, need to decide which cluster they are going to use to transmit their message. To make the decision the next hop address of the message is taken into account.

If the node and the next hop of the message are in the same cluster, the cluster they share (with the least number of member nodes) is chosen to transmit the message. Otherwise, if the node and the next hop of the message are in different clusters, the cluster of the transmitting node that has the least number of member nodes, is chosen to transmit the message.

3.11 Protocol Messages

The protocol has a number of routing layer messages. This section will describe the different messages, their content and their function.

3.11.1 Hello Message

Hello messages are used to determine a node's connectivity. As mentioned previously, hello messages are used when electing a *cluster-head*. Hello messages determine *stable* links with neighbours. The messages are kept small by sharing only the addresses of critical nodes within a two hop range. *Cluster-heads* and *satellite nodes* are critical nodes and a maximum of four of each can be included in the message. The critical nodes are limited to four to limit the size of the hello packet. The periodical generation time of hello messages is kept short (0,1 s) to ensure broken links are detected faster. Figure 3.21 on the next page shows the contents of a hello message.

3.11.2 Cluster Table Message

The cluster table message contains the routing information of all distant nodes (outside of the one hop range of neighbour nodes and two hop range of critical nodes). Only selected nodes generate cluster table messages periodically and every 0,5 s. The messages consist of a header, containing the forwarder nodes, and multiple cluster table entries. Figure 3.22 on page 51 shows the contents of a cluster table message.

3.11.3 Token Message

Small token messages are used as a measure of congestion control and to schedule nodes within a cluster for transmission. The "IT" bit field is a switch indicating if the token message is an implicit token. Figure 3.23 on page 51 shows the contents of the token message.

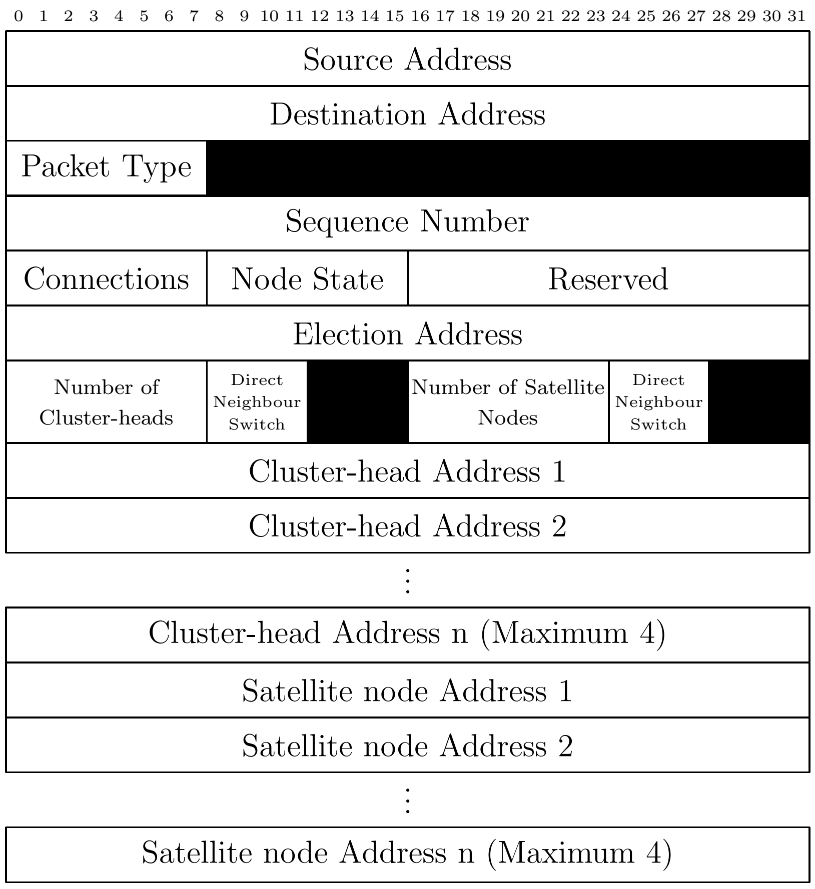


Figure 3.21: *Hello Message Contents.*

3.11.4 Data Messages

Data messages contain any content a node wants to send to a destination in the network. The size of the data packet payload is kept constant at 1 kB. Each data message has a sequence number and a TTL field. The TTL field is set to 20 initially and decremented with each hop. If the TTL field reaches zero the packet is removed from the network. Data messages are sent through the network by the User Datagram Protocol (UDP) transport layer protocol. Although the MAC layer provides hop by hop acknowledgements for data packets, the UDP transport layer does not guarantee delivery at the destination node by an acknowledgement packet.

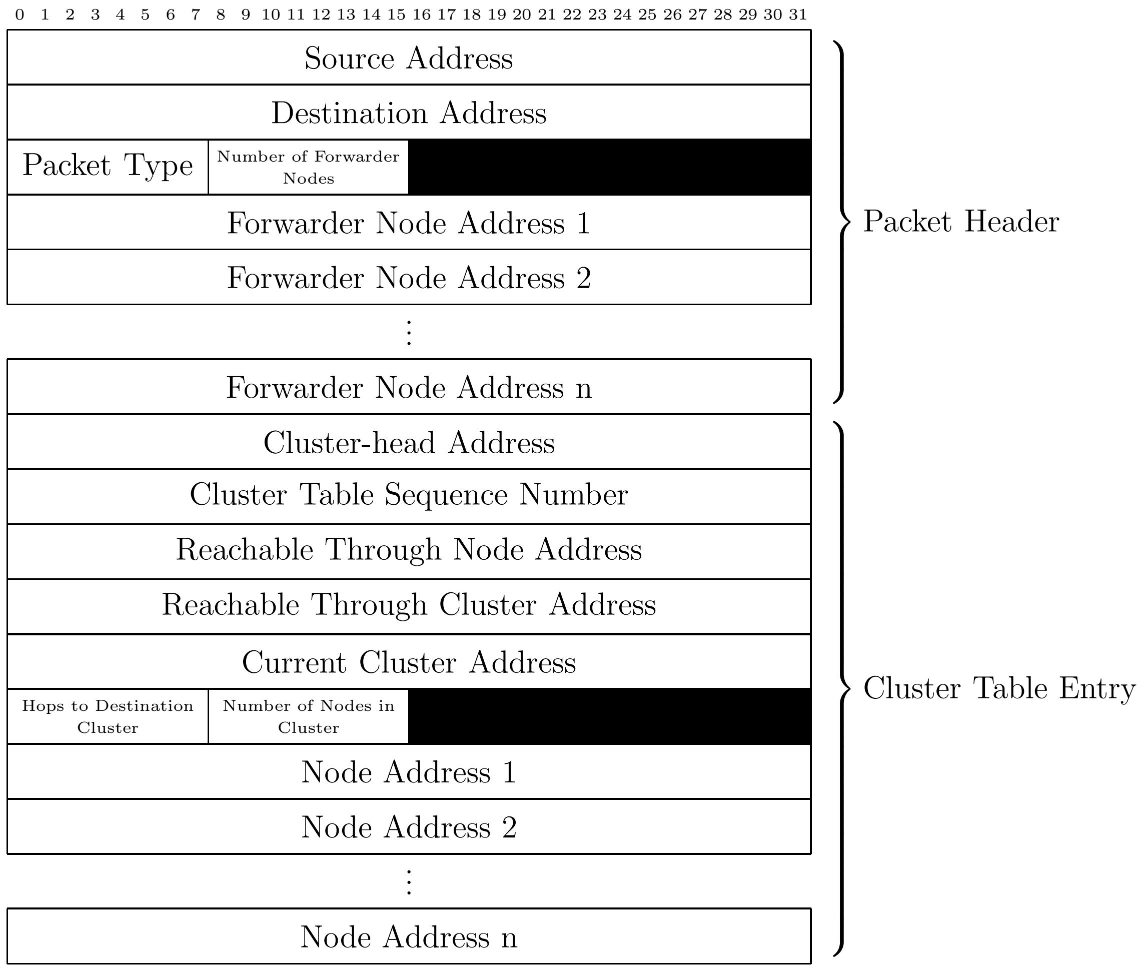


Figure 3.22: *Cluster Table Message Contents.*

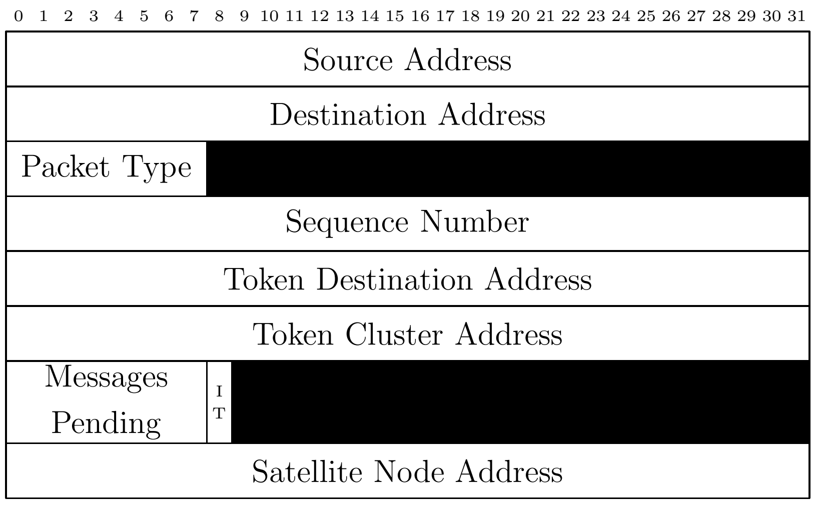


Figure 3.23: *Token Message Contents.*

3.12 Performance Monitoring

The measurement of the performance of the routing protocol is an important feature. It was decided to create an external module, to keep the implementation of the routing protocol itself and performance code separate for greater clarity. This approach also allows characteristics of the network to be determined at different points of simulation time.

Nodes have write access to the performance module memory to record certain statistics for example the total messages sent, received and lost. At the start of each simulation each node registers with the performance module and gives access to the important data structures of the node, such as the node's routing tables etc.

The performance module records the latency of all packets and groups them according to their hop count. In this way the mean latency for a specific hop count can be determined. Statistics on node neighbours are determined: the mean number of neighbours of a node and the minimum and the maximum number of neighbours. The composition of the network is determined, in other words the percentage of nodes that are in specific state. The network is tested to ensure no two *cluster-heads* are neighbours. Throughput of the network is also measured. After each simulation run the data bitrate demanded and the bitrate achieved are recorded.

A graphviz (drawing tools for graphs) source file is generated by the performance module. This source file is then compiled with the *neato* program [30] for a visual display of node relationships. The graph is shown in figure 3.24 on the following page, the different node's states are represented by different shapes.

Output files for debugging and analysis purposes are generated by the performance module. Each node's routing tables and the routes a packet followed to reach a destination node is recorded in a folder for each simulation.

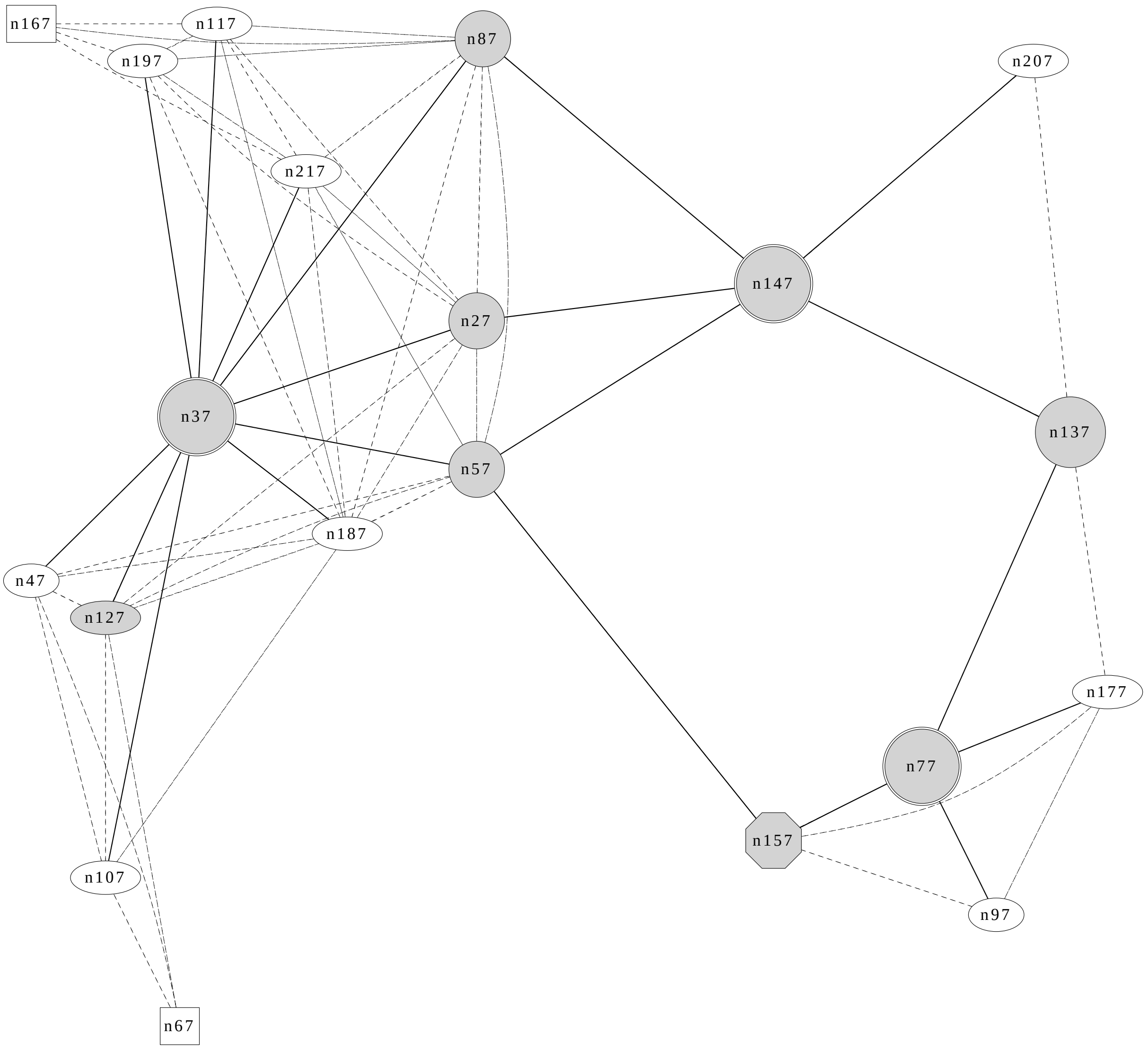


Figure 3.24: *Graph output generated by the neato program. The double circles represent* cluster-heads*, circles are* gateways*, octagons are* distributed gateways*, ellipses are* assigned *nodes and rectangles are* satellite nodes*. Forwarder node's shapes are filled with a grey colour. The solid lines indicate a neighbour link between a* cluster-head *and a cluster member node and the dashed line indicates a normal neighbour link. Links between two distributed gateways are also indicated with a solid line.*

3.13 Summary

The routing protocol design and implementation environment are documented in this chapter. In the beginning of the chapter the initial design choices are stated and the reasons for choosing CGSR as a base for the protocol are shared. Changes to CGSR include the shortening of the path length and the reduction in the number of nodes producing full routing overhead. The OMNeT++ simulator was chosen as the implementation environment and a Wi-Fi tranceiver's transmission parameters were used. It is shown how to scale the graphics output of OMNeT++.

An overview of the routing protocol development is given. The cluster generation mechanism of using hello messages to elect the most highly connected neighbour, is explained. All the different node states are defined and a *determine node state* algorithm is given. The content of the different routing tables are shown and clarified. Distribution of routing information is described. A flow chart of the routing function is presented.

The token scheduling mechanism is shown. An equation to determine the number of transmission chances a node can receive in a cluster transmission cycle, based on the number of queued messages at the node, is given. A message buffer mechanism is shown queueing forwarded traffic and traffic generated by the node in different buffers. Content of the different protocol messages are displayed in byte fields.

The chapter ends with a description of a performance module for measuring parameters in simulation time. Graphs generated by the performance module show the different node states and indicate forwarder nodes with a filled grey shape. The performance module generates output files for debugging and analysis purposes.

Chapter 4

Theoretical Analysis

This chapter investigates theoretical aspects of the routing protocol. A theoretical model is presented to predict the mean latency of data packets and attempt to validate our simulation model. Since the latency model is based on queueing theory, a short introduction will be given on the subject. The latency model is presented as a first attempt to model latencies in a cluster based protocol. Subsequently, application of the latency model on different topologies, will be shown. The results obtained is compared to the simulated latency results. A discussion will follow on why the theoretical latencies are different from the simulated latencies. Other queueing theory approaches were investigated and are summarised in the before last section. Lastly, a summary will be given of the comparison between the theoretical model and the simulation values and a potential direction for future work.

4.1 Introduction to Queuing Theory

A queueing model can be used to study data transfer latencies in networks. The Kendall notation is used to characterize different single waiting line queues [13].

$$A/B/m/k/n \tag{4.1}$$

- A is the arrival process probability distribution.
- B is the service process probability distribution.
- m is the number of servers.
- k is the maximum queue length.
- n is the customer population size.

The queueing discipline describes the way in which customers enter and leave the queue. FIFO (first in first out) is an example of a queueing discipline. The $M/M/1/\infty/\infty$ queuing system models a queue where the arrival process and service process are of an exponential probability distribution (Markovian). If the arrival pattern follow a Poisson distribution it can be proved the inter-arrival times have an exponential distribution [44].

$$A(t) = \lambda_A e^{-\lambda_A t} \text{ obtained from [44].} \tag{4.2}$$

- λ_A is the mean arrival rate.

Traffic intensity ρ is defined by:

$$\rho = \frac{\lambda_A}{\mu} \text{ obtained from [44].} \tag{4.3}$$

- μ is the mean service rate.

A queue becomes unstable when:

$$\lambda_A \geq \mu \tag{4.4}$$

The number of customers N in the queue is given by:

$$N = \frac{\rho}{1-\rho} \text{ obtained from [44].} \tag{4.5}$$

Little's law is given by:

$$N = \lambda T \text{ obtained from [23].} \tag{4.6}$$

- T is the mean waiting time for a customer in the queue.

The waiting time can now be derived:

$$T = \frac{1}{\mu - \lambda_A} \text{ obtained from [44].} \tag{4.7}$$

A Jackson queuing network can be constructed to simplify queueing problems. Poisson streams can be combined to give a new stream with the arrival rate equal to the sum of each original stream. A stream can also be split and each split creates a new stream with a rate proportional to its ratio of the original stream. The waiting times of queues in series can be summed to give the total waiting time. [44]

4.2 Latency Model

An exact theoretical model is difficult to obtain, because the probability of a token packet being lost is an intricate value to determine. The model presented here, only aims to give an estimate of which latencies can be expected. An ideal mathematical model has the properties of simplicity, logical consistency and accuracy. The investigated protocol unfortunately has many dependencies and is difficult to model accurately. To develop the model certain simplifications and assumptions are made. It is assumed that for the data network, we are investigating the arrival process and service process follow an exponential distribution. A route in the network can be seen as a series of queues at each node. The total waiting time for a route is given by:

$$T_{route} = \sum_{i=1}^{n} \frac{1}{\mu_i - \lambda_i} \tag{4.8}$$

- μ_i is the mean service rate at node N_i,
- λ_i is the mean arrival rate at node N_i and
- Nodes $N_1, N_2, N_3, ..., N_n$ are included in the summation where node N_n is the node before the destination.

Now the mean arrival rate and mean service rate must be determined for each node. It will be convenient to express the arrival rate as a bit rate and determine the packet arrival rate. The following equation links the generated bit rate to the generated number of packets:

$$\lambda_1 = \frac{G_{input}}{D_{data\ packet\ size}} \tag{4.9}$$

- G_{input} is the generated bit rate.

It is assumed that packets lost by the MAC-layer is minimal, therefore, the arrival rate at each node remains the same: $\lambda_1 = \lambda_2 = ... = \lambda_n$.

The service rate of a cluster member node is dependent on the cluster cycle time. Although some packets will not wait a full cluster cycle for the token if the queue is empty, we assume all packets must wait at least a full cluster cycle to be serviced. The service rate for node N_i is:

$$\mu_i = \frac{1}{T_i^{service}} \tag{4.10}$$

- $T_i^{service}$ is the time node N_i waits to receive a token correctly.

If a node is a *cluster-head* the service rate is simply:

$$\mu_i = \frac{1}{t_{data\ transmission}} \tag{4.11}$$

Now, we will determine the time a node waits to receive a token correctly. If p_{miss} tokens are missed by a node it needs to wait another cycle. The probability that two consecutive tokens will be missed is $(p_{miss})^2$ and the chance m consecutive tokens will be missed is $(p_{miss})^m$. For each missed token the node has to wait an extra cycle. Hence the mean waiting time for a node is increased by $\sum_{m=1}^{\infty} m.(p_{miss})^m$ [36]. The node service time is determined by the following equation:

$$T_i^{service} = T_j^{cycle}.(1 + \sum_{m=1}^{\infty} m.(p_{miss})^m) \tag{4.12}$$

- T_j^{cycle} is the cluster cycle time of cluster j and
- p_{miss} is the probability that a token will be missed by the node.

p_{miss} can be determined by the following equation:

$$p_{miss} = p_{lost\ range} + p_{busy\ token} + p_{busy\ data} \tag{4.13}$$

- $p_{lost\ range}$ is the probability that a token packet is lost due range.
- $p_{busy\ token}$ is the probability that a token is lost due to a neighbouring cluster's token message interference.
- $p_{busy\ data}$ is the probability that a token is lost due to the channel being busy with the transmission of a data message.

We assume the lost token probability $p_{lost\ range}$ is 0,01 if the distance of the node to the *cluster-head* is near, 0,025 for a medium distance and 0,05 if the distance is far. These values were determined from token loss factors from simulation.

The number of tokens lost is also influenced by the number of nodes shared by a neighbouring cluster. If the node considered is not shared by two or more clusters $p_{busy\ token} = 0$. The more nodes that are shared by a neighbouring cluster the more token misses will occur. The nodes not part of the neighbouring cluster also have a influence on the shared nodes, because nodes that are not shared is less probable to time-out and token packets are broadcast by the *cluster-head*. Therefore, shared nodes hear tokens that are not necessarily addressed to them. $p_{busy\ token}$ is calculated by:

$$p_{busy\ token} = (\frac{b_{jk}}{a_j}).(1 - \frac{b_{jk}}{a_k}) \tag{4.14}$$

- a_j is the number of nodes in cluster j.
- a_k is the number of nodes in neighbour cluster k.
- b_{jk} is the number of nodes shared by the cluster and the neighbour cluster.

Transmission of data messages also prevent tokens from being received correctly. $p_{busy\ data}$ is a measure of how busy the channel is transmitting data messages and is determined by:

$$p_{busy\ data} = h.\frac{D_{DATA}.\lambda_i}{G_{TR}} \tag{4.15}$$

- h is the number of nodes transmitting data messages in range of the considered node.
- D_{DATA} is the data message size.
- G_{TR} is the transmission bit rate.

The mean cluster cycle time can be determined by:

- estimating the time it takes to pass the token to every node in the cluster and
- adding the time delay caused by token time-outs.

Note that to simplify the theoretical model each node is only given one transmission chance. The total cycle time can be calculated by:

$$T_j^{cycle} = T_j^{token} + T_j^{time\text{-}outs} \tag{4.16}$$

- T_j^{token} is the time needed to do the token exchange in the cluster.
- $T_j^{time\text{-}outs}$ is the time used to account for token time-outs.

The time needed to perform the token exchange cycle is calculated by:

$$T_j^{token} = a_j . t_{tsr} \tag{4.17}$$

- a_j is the number of nodes in cluster j that is a direct neighbour of the *cluster-head.*
- t_{tsr} is the time it takes to pass a token to a node and to pass the token back to the *cluster-head.*

Note that *satelite nodes* are not considered to have an influence on the cycle time, because they are not neighbours of *cluster-heads.* The following equation describes the composition of t_{tsr}:

$$t_{tsr} = 2 \times t_{token\ transmission} \tag{4.18}$$

Table 4.1 on page 61 shows the different components of $t_{data\ transmission}$ (t_{dt}), $t_{token\ transmission}$ (t_{tt}) and various other parameters. Table 4.2 on page 62 shows the different components of the data parameters.
The influence of the token time-outs can be calculated by:

$$T_j^{time\text{-}outs} = p_{time\text{-}outs}.(t_{time\text{-}outs} - t_{tt}).a_j \tag{4.19}$$

- $p_{time\text{-}outs}$ is the probability that a sent token times out.
- $t_{time\text{-}outs}$ is the time after which a token times out (t_{tt} the token transmission time is subtracted, because the transmission time of the token is taken into account already in T_j^{token}).

$p_{time\text{-}out}$ is influenced by all the nodes the *cluster-head* sends a token. $p_{time\text{-}out}$ is calculated by:

$$p_{time\text{-}out} = \frac{1}{a_j}.\sum_{n=1}^{a_j}(1 - p^n_{receive\ node}.p^n_{return\ ch}) \tag{4.20}$$

- $p^n_{receive\ node}$ is the probability that the token packet will be correctly received by node n.

- $p^n_{return\ ch}$ is the probability that the token will be successfully returned to the *cluster-head* from node n.

$p^n_{receive\ node}$ is related to p_{miss} and can be determined by:

$$p^n_{receive\ node} = (1 - p_{miss}) \tag{4.21}$$

$p^n_{return\ ch}$ is only influenced by the node's distance from the *cluster-head* and can be determined by:

$$p^n_{return\ ch} = (1 - p_{lost\ range}) \tag{4.22}$$

Table 4.1: *Table showing the composition of time parameters.*

Parameter	Composition
t_{ST}	$t_{ST} = 20\,\mu\text{s}$
t_{SIFS}	$t_{SIFS} = 10\,\mu\text{s}$
t_{DIFS}	$t_{DIFS} = 2 \times t_{ST} + t_{SIFS} = 50\,\mu\text{s}$
t_{BEB}	$t_{BEB} = E[X] \times t_{ST} = 310\,\mu\text{s}$
X (random number)	$X \vert X = \{0, 1, 2, ..., 31\}$
t_{bt} (before transmission)	$t_{bt} = t_{DIFS}$
t_{tt}	$t_{tt} = t_{bt} + t_{td}$
t_{dt}	$t_{dt} = t_{bt} + 3 \times t_{SIFS} + t_{RTS} + t_{CTS} + t_{dd} + t_{ACK}$
G_{PHY} (bit rate)	$G_{PHY} = 1\,\text{Mbps}$
G_{TR} (bit rate)	$G_{TR} = 2\,\text{Mbps}$
$t_{bit\ PHY}$	$t_{bit\ PHY} = \frac{1}{G_{PHY}}$
$t_{bit\ TR}$	$t_{bit\ TR} = \frac{1}{G_{TR}}$
t_{RTS}	$t_{RTS} = D_{PHY\ HEAD}.t_{bit\ PHY} + D_{RTS}.t_{bit\ TR}$
t_{CTS}	$t_{CTS} = D_{PHY\ HEAD}.t_{bit\ PHY} + D_{CTS}.t_{bit\ TR}$
t_{ACK}	$t_{ACK} = D_{PHY\ HEAD}.t_{bit\ PHY} + D_{ACK}.t_{bit\ TR}$
t_{dd}	$t_{dd} = D_{PHY\ HEAD}.t_{bit\ PHY} + (D_{MAC\ HEAD} + D_{NET\ HEAD} + D_{DATA}).t_{bit\ TR}$
t_{td}	$t_{td} = D_{PHY\ HEAD}.t_{bit\ PHY} + (D_{MAC\ HEAD} + D_{NET\ HEAD} + D_{token}).t_{bit\ TR}$
$t_{time\text{-}out}$	$t_{time\text{-}out} = 0{,}0075\,\text{s}$
t_{hello}	$t_{hello} = D_{PHY\ HEAD}.t_{bit\ PHY} + (D_{MAC\ HEAD} + D_{NET\ HEAD} + D_{hello}).t_{bit\ TR}$
$t_{routing}$	$t_{routing} = D_{PHY\ HEAD}.t_{bit\ PHY} + (D_{MAC\ HEAD} + D_{NET\ HEAD} + D_{routing}).t_{bit\ TR}$
T^{cycle}_{hello}	$T^{cycle}_{hello} = 0{,}1\,\text{s}$
$T^{cycle}_{routing}$	$T^{cycle}_{hello} = 0{,}5\,\text{s}$

Table 4.2: *Table showing the composition of data parameters (header and packet sizes obtained from OMNeT++ source code).*

Parameter	Composition
$D_{PHY\ HEAD}$	$D_{PHY\ HEAD} = 192\,\text{bits}$
$D_{MAC\ HEAD}$	$D_{MAC\ HEAD} = 272\,\text{bits}$
$D_{NET\ HEAD}$	$D_{NET\ HEAD} = 128\,\text{bits}$
D_{RTS}	$D_{RTS} = 160\,\text{bits}$
D_{CTS}	$D_{CTS} = 112\,\text{bits}$
D_{ACK}	$D_{ACK} = 112\,\text{bits}$
D_{token}	$D_{token} = 145\,\text{bits}$
D_{DATA}	$D_{DATA} = 8204\,\text{bits}$
D_{hello}	$D_{hello} = 240\,\text{bits}$ (accounted for 4 critical nodes)
$D_{integer}$	$D_{integer} = 32\,\text{bits}$
D_{byte}	$D_{byte} = 8\,\text{bits}$
$D_{routing}$ $(cluster\text{-}head)$	$D_{routing} = \left(6.(clusters) + \left(\sum_{i=1}^{n} C_i\right) + (forwarder\ nodes)\right).D_{integer} + 2.D_{byte}$
$D_{routing}$ (forwarder node)	$D_{routing} = \left(6.(clusters) + \left(\sum_{i=1}^{n} C_i\right)\right).D_{integer} + 2.D_{byte}$
C_i	Number of nodes in cluster i

4.3 Application of Latency Model

A Python [34] script was created to simplify the testing of the theoretical model. A function is created that accepts a data rate as input parameter and calculates the total latency for the whole route. Waiting times for various data rates can now be determined. The theoretical model is compared to the simulated results in the following sections.

The simulation setup will now be described. OMNeT++ accepts XML input files for compound configuration variables. A BASH [35] script was written to generate XML files for different packet generation rates. The OMNeT++ simulation is also invoked multiple times with the help of a BASH script. The simulation writes the latency and throughput results to a file. Next the file is analysed by the previously mentioned Python script. Each data rate is simulated once for 200 simulation seconds.

4.3.1 Network 1 Latency Analysis

The topology of the network and the analysed routes are shown in figure 4.1. Routes are determined by the algorithm used by the routing function.

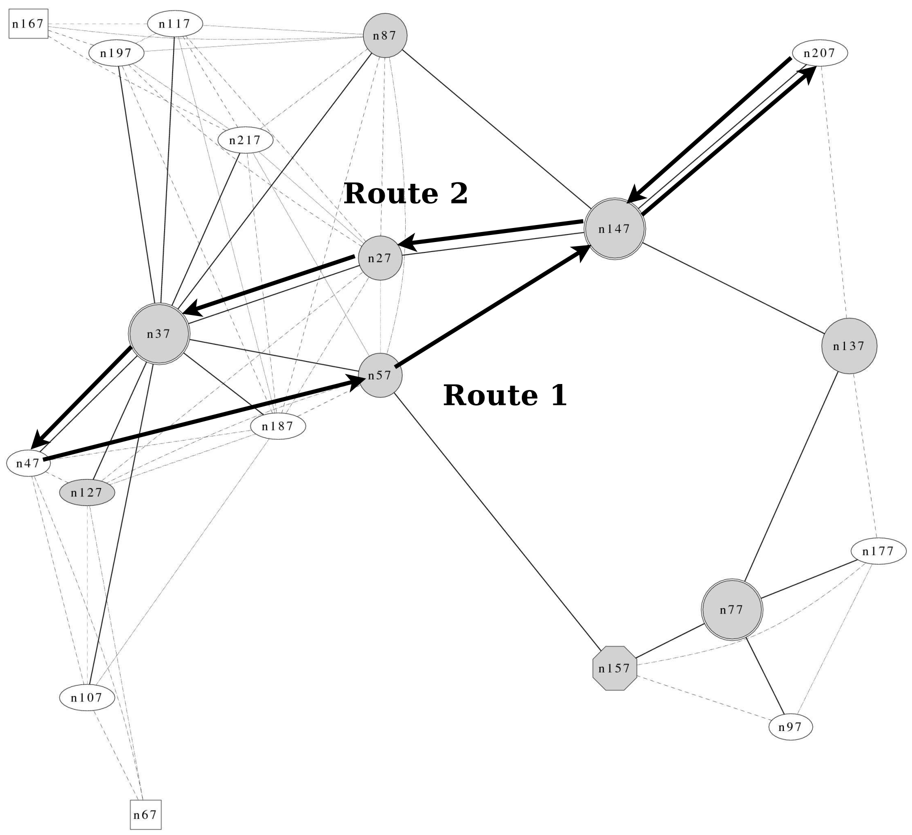


Figure 4.1: *Network 1 topology.*

Route 1 Latency Analysis

The different average cluster cycle times for each cluster, are shown in figure 4.2. Theoretical cluster cycle times are also compared with the measured simulation cycle times. It is interesting to note the cluster cycle time increases with higher data rates, because more tokens are lost due to a higher $p_{busy\ data}$. Although the theoretically predicted cycle time and measured cycle times correspond well, there are some noticable differences. The differences can be attributed to $p^n_{receive\ node}$ and $p^n_{return\ ch}$ (lost token packets) which could not be accurately determined. The theoretically predicted values were compared with debug output of the measured simulated $p^n_{receive\ node}$ and $p^n_{return\ ch}$ values.

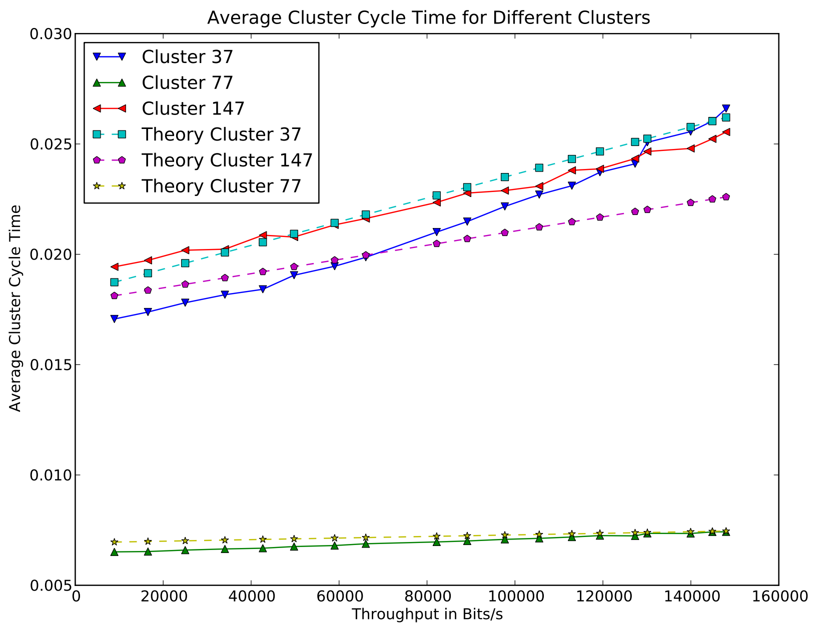


Figure 4.2: *Average cluster cycle time for network 1 route 1.*

Figure 4.3 shows the mean packet latency for route 1 of network 1. The latency predicted is much higher than the measured simulation latency and the highest throughput is much lower than the measured throughput. The reason for this is $p^n_{receive\ node}$ of node 57 which is predicted too low by the theoretical model.

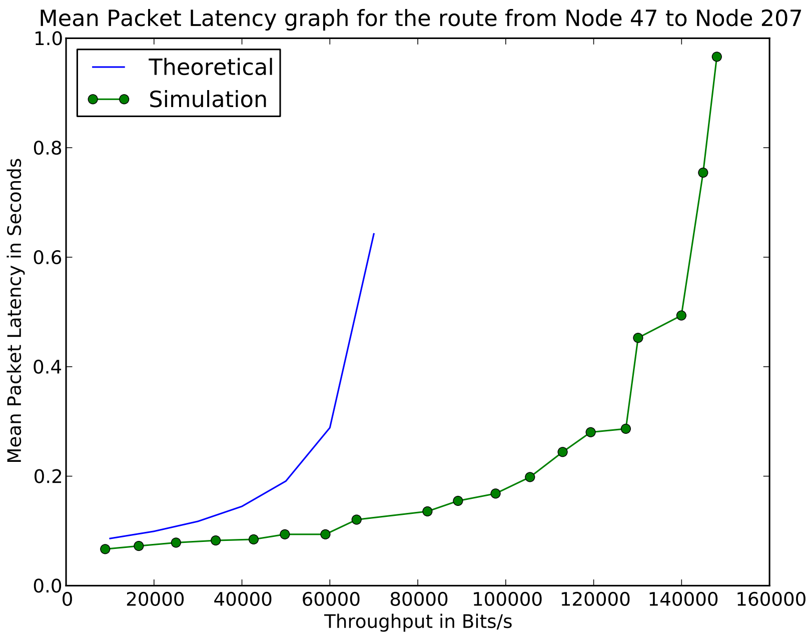


Figure 4.3: *Mean packet latency for network 1 route 1.*

Route 2 Latency Analysis

The mean cluster cycle times of route 2 are shown in figure 4.4. As in route 1 the cycle times correspond well, but is still inaccurate. This can also be attributed to inaccurate predicted values for $p^{n}_{receive\ node}$ and $p^{n}_{return\ ch}$ (lost token packets).

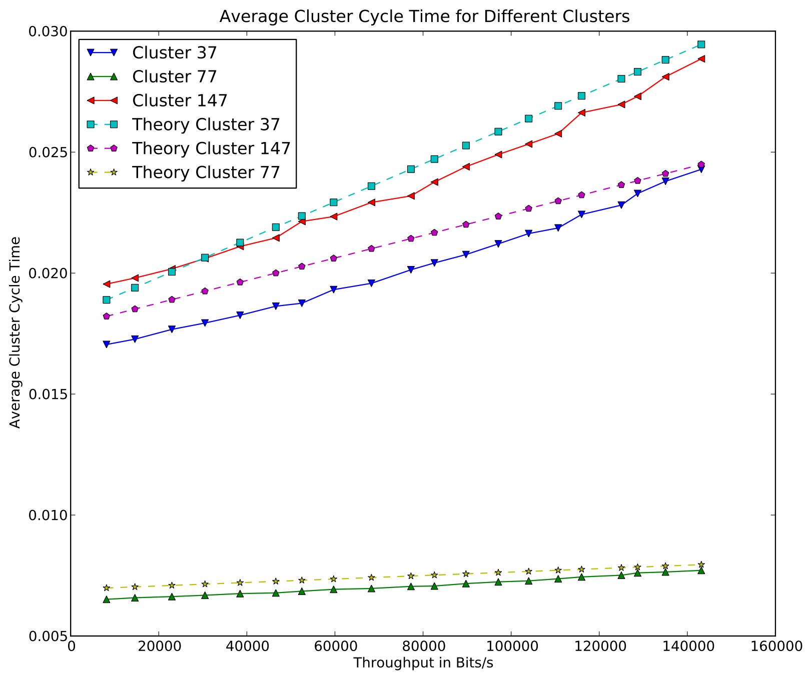


Figure 4.4: *Average cluster cycle time for network 1 route 2.*

Figure 4.5 shows the mean packet latency of route 2. For higher data rates the theoretical model predicts a higher latency. Again this is caused by a too high $p^n_{receive\ node}$ value determined by the model.

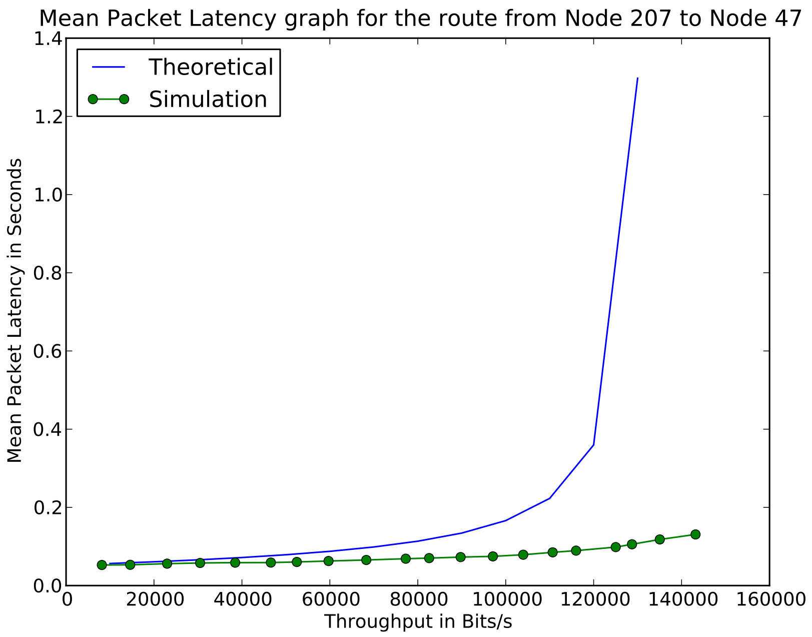


Figure 4.5: *Route latency for network 1 route 2.*

4.3.2 Network 2 Latency Analysis

It was decided to analyse a second network topology to determine how our model compares with simulation results. Figure 4.6 shows the topology of the second network.

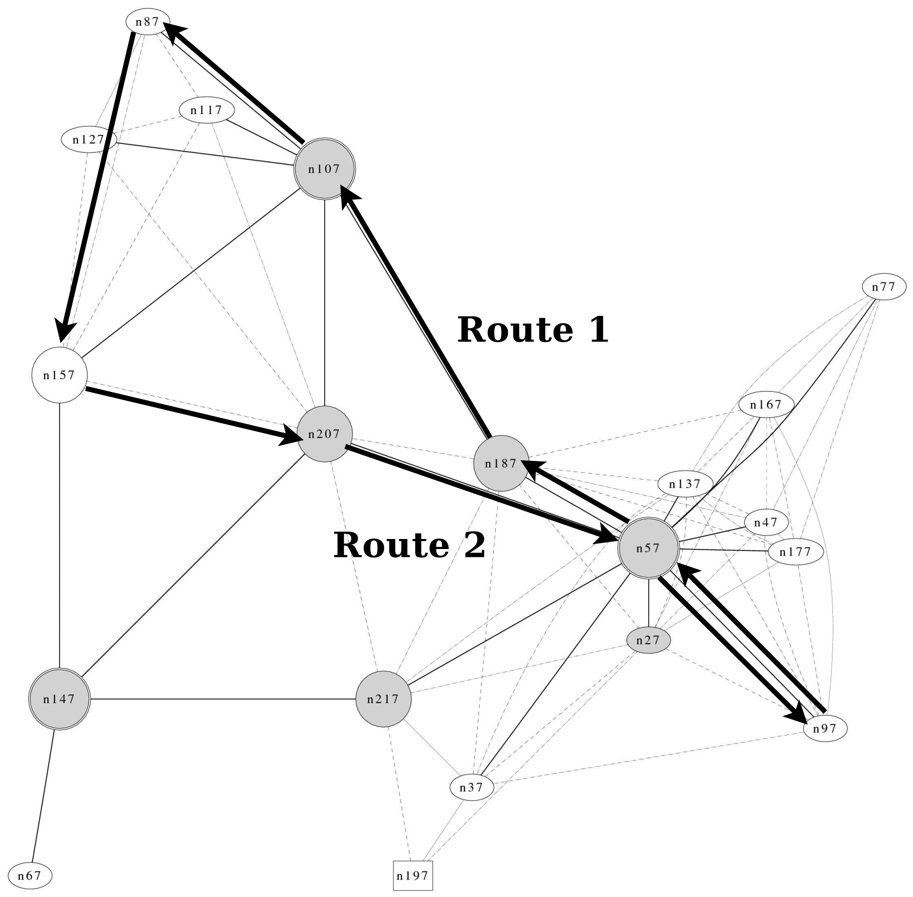


Figure 4.6: *Network 2 topology.*

Route 1 Latency Analysis

Figure 4.7 shows the cluster cycle times for route 1 of the second network. For this network topology the theoretical cluster cycle time is not predicted accurately. Again incorrect values predicted by $p^n_{receive\ node}$ and $p^n_{return\ ch}$ (token packets lost) is suspected to cause the discrepancy.

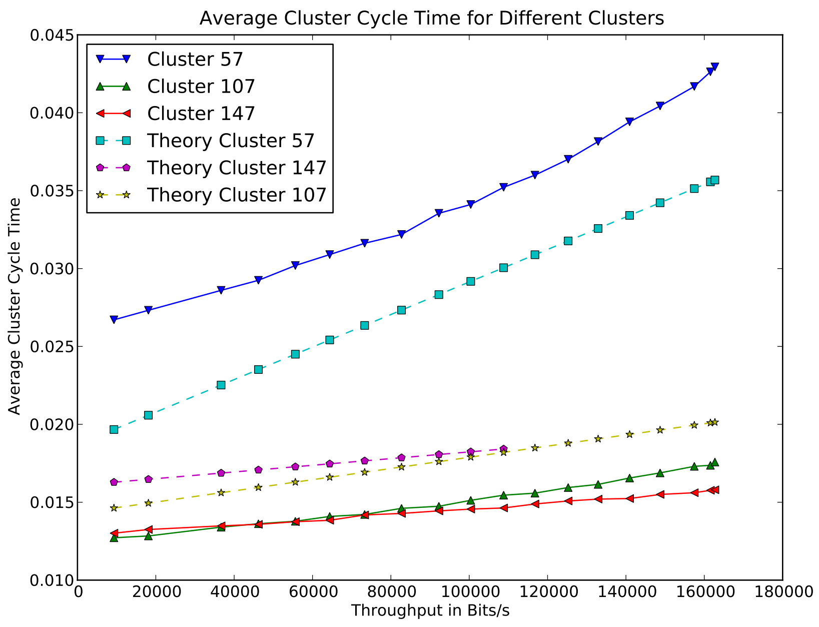


Figure 4.7: *Average cluster cycle time for network 2 route 1.*

The mean packet latency of route 1 is shown in figure 4.8. For lower latencies the theoretical model predicts the latency accurately, but again for higher latencies the model predicts a higher latency. A high $p^{n}_{receive\ node}$ value (tokens lost by the cluster member node) is believed to be the cause of the difference between the simulation measurements and the theoretical model.

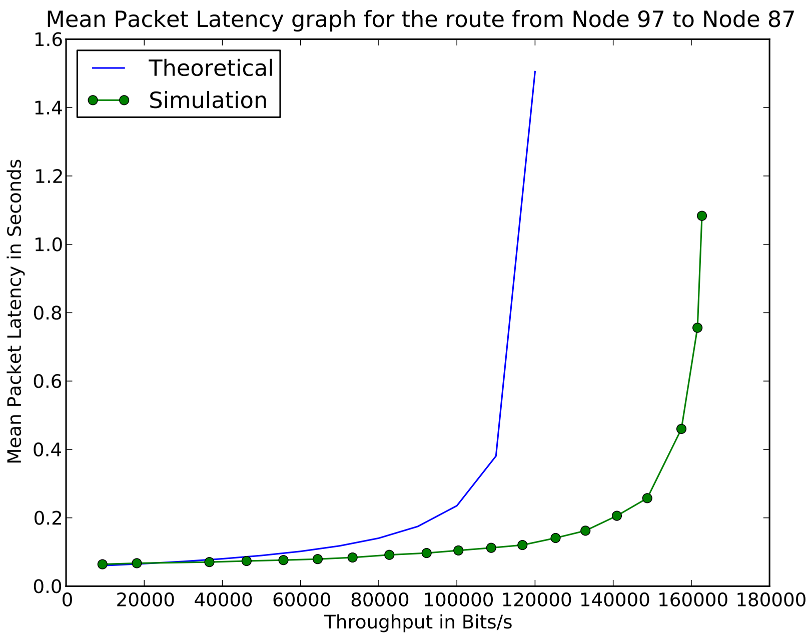


Figure 4.8: *Route latency for network 2 route 1.*

Route 2 Latency Analysis

Figure 4.9 shows the cluster cycle times for route 2 of the second network. It can be seen that the theoretical cluster cycle time and the measured cycle time disagrees. It is assumed that wrong values for $p^n_{receive\ node}$ and $p^n_{return\ ch}$ (token packets lost) is the cause for the wrongly predicted cycle times.

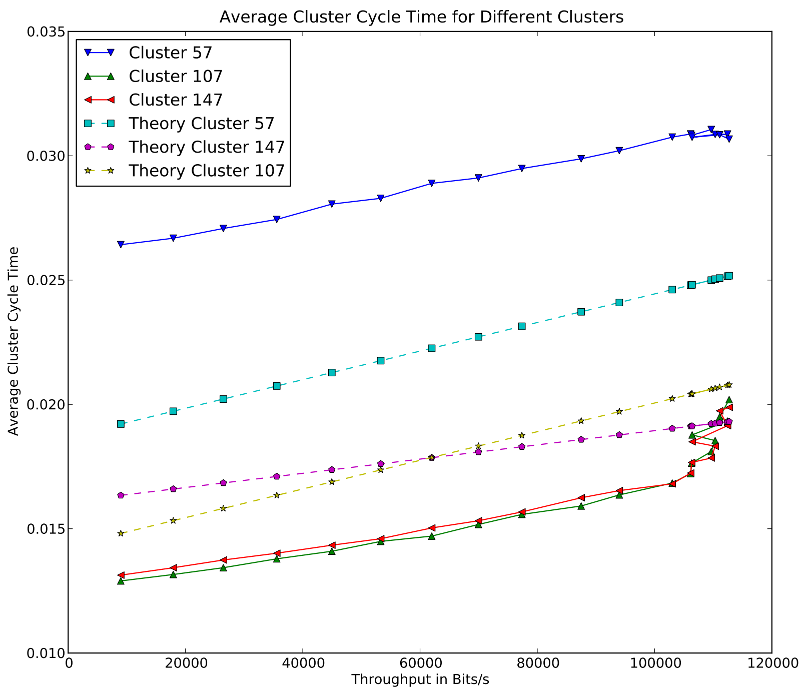


Figure 4.9: *Average cluster cycle time for network 2 route 2.*

The mean packet latency of route 2 is shown in figure 4.10. For this route the measured mean packet latency agrees well with the theoretical model. It is assumed that this is an isolated fortunate incident, because the measured cluster cycle time and theoretical cluster cycle time of the different clusters disagrees.

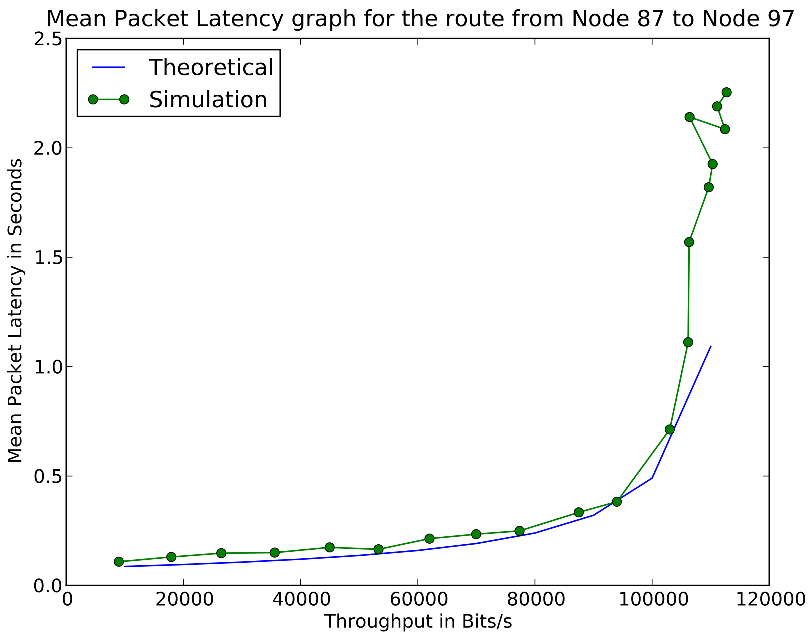


Figure 4.10: *Route latency for network 2 route 2.*

4.4 Reasons for Differences of the Theoretical Model and Measured Simulation Values

First of all the backoff of the CSMA/CA MAC layer was not modelled accurately. Secondly, since nodes share the same wireless channel it is difficult to anticipate all packet collisions. Clusters overlap this makes it difficult to determine the time-out probabilities accurately. Tokens are passed concurrently by each *cluster-head* in each cluster. The distance from a node to its cluster-head, has a substantial influence on the probability if a token packet is lost. The tokens lost due to range are presented by a simple loss probability, $p_{lost\ range}$. Simplifications were also made which contribute to inaccurate modelling.

4.5 Other Investigated Approaches

Two other queueing theory approaches to the theoretical model were investigated. However, the results obtained from the different methods were unsatisfactory. The first attempt to model the cluster cycle time by estimating the influence neighbour clusters have on the cluster cycle time. A ratio of the time the neighbour cluster is busy in another cluster to the cycle time of the neighbour cluster. The ratio of neighbour cluster i on cluster j is given by the equation:

$$r_{ij} = \frac{T_{ij}^{busy}}{T_i^{cycle}} \tag{4.23}$$

The cycle time of cluster j is then increased by a factor $1 + r_{ij}$. This model delivered unsatisfactory results compared to the simulation results, because cluster token time-outs and token misses were not considered.

An model using a Markov state matrix to model the service time of a node in a cluster more accurately was implemented with considering the token time-outs and token misses. The theoretical model results resembled the results of the simulation more closely, but for higher throughput values the model and simulation did not correspond well. One reason the model deviated from the simulation was: packets lost by the MAC-layer were not taken into account. Figure 4.11 on the next page shows the Markov state diagram. Two service rates are used by the model: $\mu_m = \frac{1}{T_{mean\ cycle}}$ (the normal RRP service rate) and $\mu_f = \frac{1}{T_{full\ cycle}}$. The extra state introduced by the model accounts for the situation where no packets arrive after a full cycle time and a single packet from the transmission queue is transmitted.

4.6 Summary

In this chapter a theoretical model based on queueing theory was considered. In some cases the predicted theoretical cluster cycle time compared well with simulated cluster cycle times with both routes considered in network 1 (section 4.3.1). In network 2 the cluster cycle

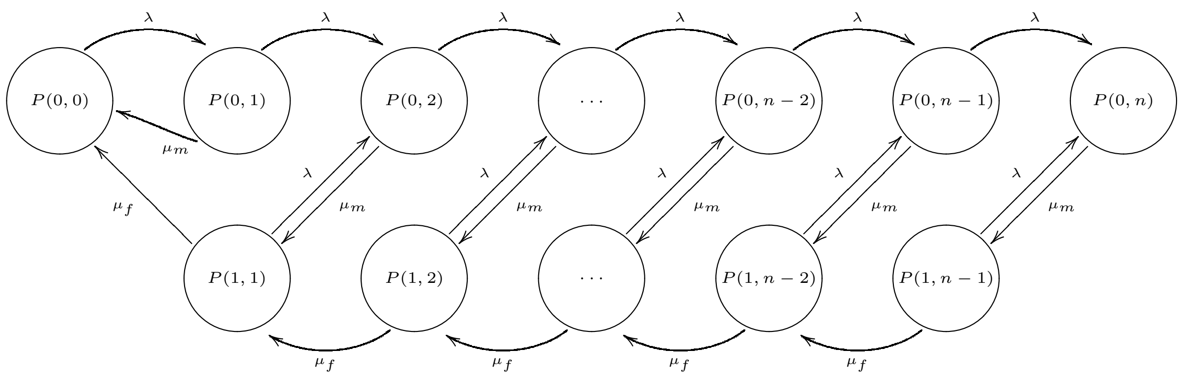


Figure 4.11: *Markov state diagram of queueing system.*

times did not correspond well. Modelling the clusters with queueing theory proved difficult. It is suggested that a different mathematical modelling technique should be used to model the latency such as a Markov state model which defines a state for each node a *cluster-head* passes a token to. Another mathematical modelling approach can be investigated, namely, Petri nets. Petri nets was first used to model discrete manufacturing systems. Concurrent systems can be modelled by using Petri nets [45]. It is suspected that our protocol can be modelled more accurately using Petri nets.

Chapter 5

Tests and Results

Testing of the cluster based protocol and the results obtained will be presented in this chapter. The tests aim to show that our design outcomes stated in section 3.1 have been met.

First of all, the general test setup will be explained. Cluster setup tests will be focused on next. The setup time of clusters will be determined and the number of nodes that transmit full routing information. Aggregate connectivity will be tested on different sizes and random topologies of networks. Simulation time graphs of the aggregate connectivity will be shown. The control overhead per second per node versus the node density will be determined and displayed to determine how effective the protocol operates at high node densities. Next, tests are performed to show the effect prioritising nodes with more traffic has on the mean packet latency. Lastly, tests are performed to show how the protocol repairs a route when a link of an original route is broken.

5.1 General Test Setup

Tests were performed on several different random networks with sizes of 20, 50 and 100 nodes. The transmission range of the nodes are approximately 40 m. Table 5.1 on page 75 shows the dimensions of the playground for the different network sizes. A BASH [35] script

Table 5.1: *Playground sizes.*

Network size (number of nodes)	X dimension	Y dimension
20	90 m	90 m
50	149 m	149 m
100	179 m	179 m

was setup to run several simulations consecutively. The graphs of the different test networks can be found in appendix A.

5.2 Cluster Formation

Cluster formation is tested with network setup tests. Different networks are tested with a number of nodes randomly distributed throughout the playground. Table 5.2 on page 76 shows the results obtained of 10 different simulation runs, where 20 nodes are used.

Table 5.2 shows the number of clusters formed, average cluster size, the number of forwarder nodes, the percentage of nodes producing full routing information and the last *cluster-head* election time. In chapter 3 it was shown that the number of nodes producing full routing information can be reduced. Only forwarder nodes produce full routing overhead. In simulation 9 the number of nodes that produce full routing information are reduced to 40 %, which is the highest percentage of forwarder nodes of the different simulations. The average number of clusters is 2,6 and the average last *cluster-head* election time is 0,453 s.

Table 5.2: *Cluster formation results with 20 nodes*

Simulation number	Number of clusters	Average number of nodes in cluster	Number of forwarder nodes	Percentage forwarder nodes	Last *cluster-head* election time
1	2	12,5	4	0,2	0,365 s
2	2	11	4	0,2	0,450 s
3	3	10	6	0,3	0,534 s
4	3	8,333	5	0,25	0,485 s
5	4	7	6	0,3	0,451 s
6	2	12,5	4	0,2	0,455 s
7	2	12	6	0,3	0,490 s
8	3	8	6	0,3	0,480 s
9	3	7,333	8	0,4	0,369 s
10	2	11,5	3	0,15	0,454 s

On page 77, table 5.3 shows 10 network setup tests with 50 nodes used in each simulation run. The number of nodes generating full routing overhead is reduced to at least 40 %. The average number of clusters is 6,8 and the average last *cluster-head* election time is 0,605 s.

Table 5.4 on page 77 shows the results of 10 different simulations with 100 nodes each. The number of nodes that broadcast full routing information is reduced to at least 35 %. The average number of clusters is 11 and the average *cluster-head* election time is 0,736 s.

One interesting observation is that the average last *cluster-head* election time increases with the increase in the number of nodes. The reason for the increase in cluster-head election time is assumed to be more *cluster-head* challenges between neighbours, where one node needs to give up its *cluster-head* status.

Table 5.3: *Cluster formation results with 50 nodes*

Simulation number	Number of clusters	Average number of nodes in cluster	Number of forwarder nodes	Percentage forwarder nodes	Last *cluster-head* election time
1	7	10,143	19	0,38	0,539 s
2	8	10,75	19	0,38	0,609 s
3	5	12,6	13	0,26	0,533 s
4	7	9,429	17	0,34	0,448 s
5	7	8,714	13	0,26	0,680 s
6	7	11,286	20	0,4	0,807 s
7	6	12	15	0,3	0,647 s
8	5	13	18	0,36	0,505 s
9	8	9,375	17	0,34	0,636 s
10	8	8,875	20	0,4	0,648 s

Table 5.4: *Cluster formation results with 100 nodes*

Simulation number	Number of clusters	Average number of nodes in cluster	Number of forwarder nodes	Percentage forwarder nodes	Last *cluster-head* election time
1	13	12,615	32	0,32	0,646 s
2	12	13,167	35	0,35	1,077 s
3	10	15	28	0,28	0,501 s
4	11	13,091	32	0,32	0,705 s
5	10	14,6	34	0,34	0,645 s
6	9	15,667	30	0,3	0,651 s
7	11	13,364	33	0,33	0,656 s
8	13	13	35	0,35	1,080 s
9	12	12,833	33	0,33	0,774 s
10	9	17,111	27	0,27	0,619 s

5.3 Connectivity

One important question to ask is: if fewer nodes transmit full routing information, do all nodes obtain the necessary routing information? Each node is at least one hop away from a forwarder node, so every host should be able to obtain full connectivity. Connectivity is defined as the percentage of nodes in a network, a node can communicate with. If a node has an entry for another destination node in its routing table, the destination is reachable.

The connectivity of a node can be determined with the following equation:

$$Connectivity = \frac{Number\ of\ reachable\ nodes}{Total\ number\ of\ nodes\ in\ network} \tag{5.1}$$

The aggregate network connectivity is defined as the normalised sum of all of the individual nodes' connectivities and can be calculated by the next equation:

$$Aggregate\ Network\ Connectivity = \frac{\sum_{i=1}^{N} C_i}{N} \tag{5.2}$$

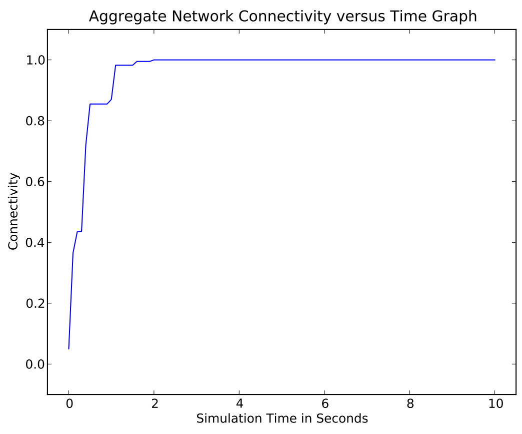
- C_i is the connectivity of node i.
- N is the total number of nodes in the network.

Figure 5.1 on the following page shows the aggregate connectivity for the first 10 s for two different random topologies for network sizes of 20, 50 and 100 nodes, respectively. All of the networks reach an aggregate connectivity of 1 after a few seconds. Therefore, the reduction in the number of nodes transmitting full routing information does not impair the routing knowledge of nodes in the network. The performance module is used to take measurements of the aggregate connectivity every 0,1 s of the simulation.

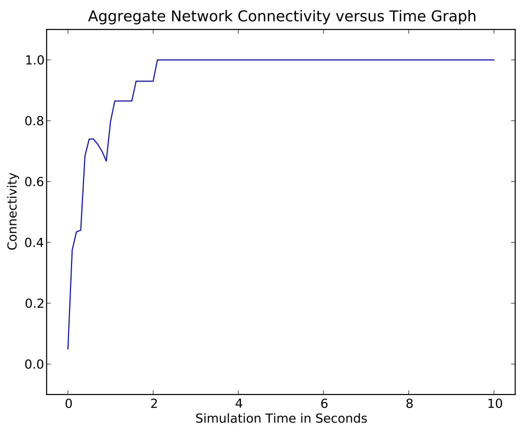
Table 5.5 indicates the time when full connectivity is reached for the different simulation runs. With the increase in network size, the time the routing information needs to proliferate throughout the network increases. The routing information needs to travel an increased number of hops (distance), because the diameter of the network has increased.

Table 5.5: *Full connectivity reached time for different network sizes.*

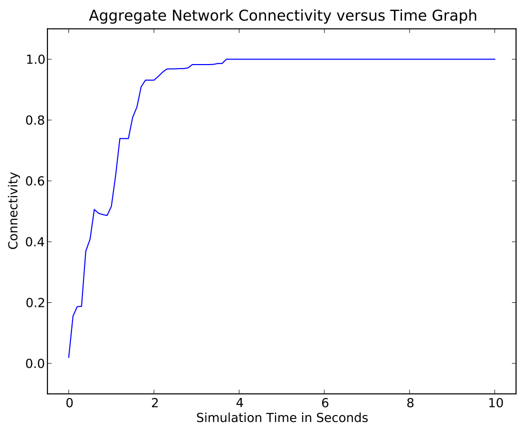
Network size (number of nodes)	Run 1 Full connectivity reached time	Run 2 Full connectivity reached time
20	2 s	2,1 s
50	3,7 s	3,2 s
100	3,9 s	3,4 s (Run 3)



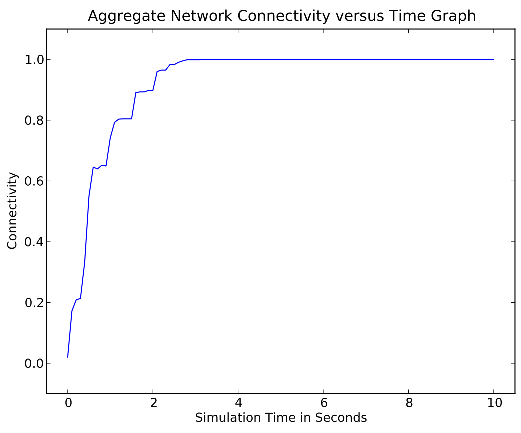
(a) Connectivity 20 Nodes Run 1.



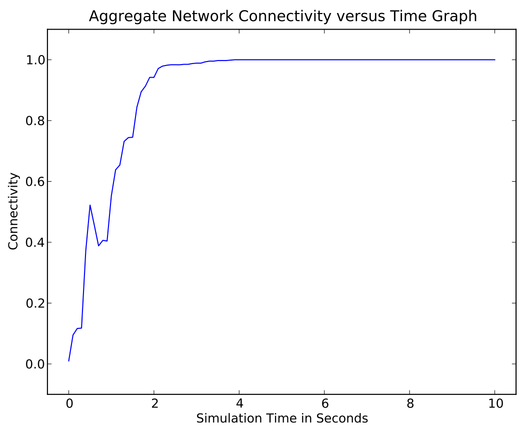
(b) Connectivity 20 Nodes Run 2.



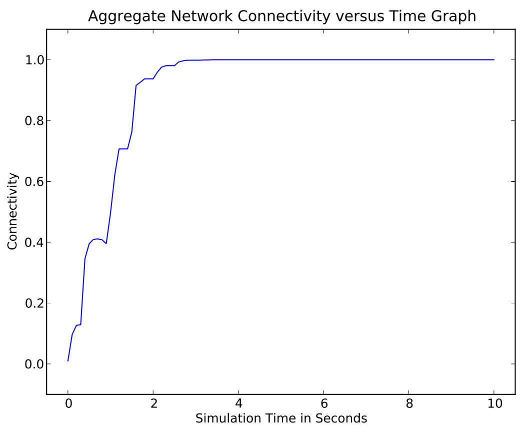
(c) Connectivity 50 Nodes Run 1.



(d) Connectivity 50 Nodes Run 2.



(e) Connectivity 100 Nodes Run 1.



(f) Connectivity 100 Nodes Run 3.

Figure 5.1: *Aggregate connectivity against simulation time.*

5.4 Routing Overhead Performance with Increasing Node Density

A test was performed to show the effectiveness of the protocol at a high node density. For the test, different simulations were performed starting with a network size of 15 nodes and then increasing the network size with 5 nodes in the same area, with each consecutive simulation run. In essence the node density is increased. Each simulation ran for 100 s of simulation time. All nodes are randomly placed.

Figure 5.2 and figure 5.3 on page 81 shows the results obtained from the various simulations. Figure 5.2 shows the routing control overhead per second per node and figure 5.3 the total control overhead per second per node.

The routing control overhead includes all the control overhead without the token passing overhead. Routing control overhead is divided into two parts, namely, the hello message overhead and the cluster table message overhead. As mentioned previously, hello messages maintain the local connections with neighbouring nodes and cluster table messages spread routing information throughout the network.

Hello messages are kept small by only including information on critical nodes within the two hop range. Therefore, the packet size does not grow with the increasing the number of nodes. Hello messages are generated periodically every 0,1 s. The high generation rate of hello messages is costly, as can be seen in figure 5.2, amounting to 80 Bps per node. The high overhead cost is justifiable, because we are designing a protocol intended for a high available bandwidth.

Cluster table messages, however, grow with increasing network size, because more node addresses have to be added to the periodic update messages. Only forwarder nodes generate cluster table messages and all nodes are at least within one hop of a forwarder node. By only letting forwarder nodes generate full routing overhead, redundant routing overhead transmissions are reduced. The cluster table message routing overhead per node, decreases with increasing node density, because the number of nodes not producing full routing information increases with respect to the number of forwarder nodes. In section 2.5.2, it was mentioned that DSDV's routing overhead grows by $O(N^2)$ (where N is the number of nodes in the network). Therefore, the routing overhead per node of DSDV should grow by $O(N)$, in other words show a linear growth. The result shows the new improved cluster protocol is more effective than DSDV at a higher node density, because the routing overhead per node declines with increasing node density.

The total routing control overhead per second per node, in figure 5.2, decreases slightly with increasing node density.

For sake of completeness, the token control overhead was plotted and shown in figure 5.3. The total control overhead consists mostly of token control overhead, because the plot lines in the graph representing the total control overhead and token control overhead, follow one another closely. The token control overhead per node also decreases with increasing node density, but this only indicates that each node in the network receives less transmission chances. There is an increase in control overhead per node when the number of nodes is increased from 30 to 35. The increase can be attributed to the way clusters has formed. More clusters has formed in the simulation run with 35 nodes and the clusters are smaller. A token passing cycle can be completed in a shorter time and therefore more tokens are passed. Thus more control overhead is produced.

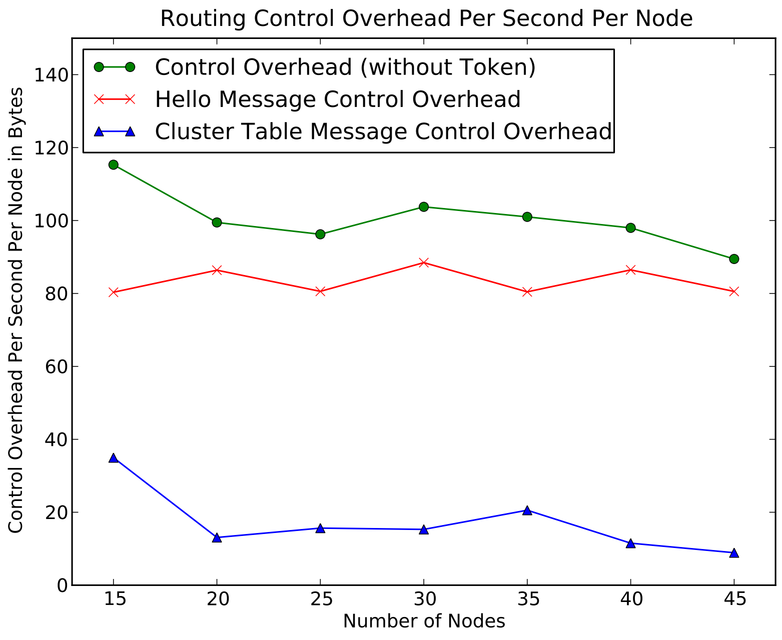


Figure 5.2: *Routing overhead per second per node with increasing node density.*

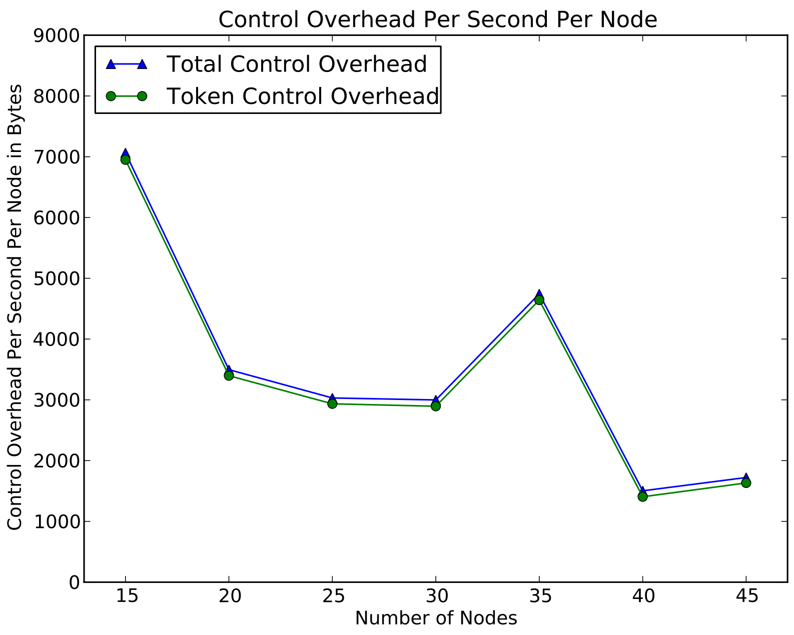


Figure 5.3: *Control overhead per second per node with increasing node density.*

5.5 Effect of Token Prioritising on Latency

To prove that the protocol can handle congestion, a test was set up to determine how the network performs in a highly congested situation. The latency of packets with token prioritising (giving more transmission chances to nodes with more queued traffic) and without prioritising was studied.

With normal CSMA/CA and token passing without prioritising, bandwidth is distributed equally among neighbours. Token passing can increase latency, because nodes now have to wait for a token instead of directly contending for the channel and extra token messages have to be transmitted. With token scheduling it is possible to increase the bandwidth of nodes with more queued traffic and decrease the chance of a node becoming a bottleneck.

A simple UDP transport layer was implemented, that sends a series of packets to a random node in the network. The data rate of the UDP packet stream was set at 250 kbps. Each simulation was run for 300 s of simulation time. From the beginning to the end of the simulation, UDP streams are repeatedly created by every node after a certain offset time. The offset time is calculated by the following equation:

$$t_{offset} = X.t_{slot} \tag{5.3}$$

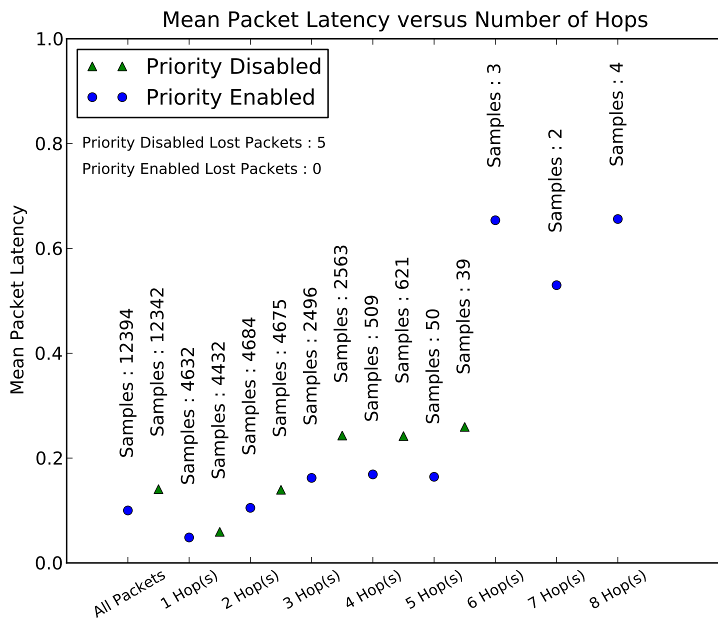
- X is a uniformly distributed integer random number $X \in \mathbf{Z}|2 \leq X \leq 4$.
- t_{slot} is a constant value defining the time of a waiting slot.

0,5 s, 0,7 s and 0,8 s were used as the t_{slot} time for the 20, 50 and 100 node networks respectively. The UDP stream length is a integer random number from 3 to 5 packets. The packet size is kept constant at 1 kB.

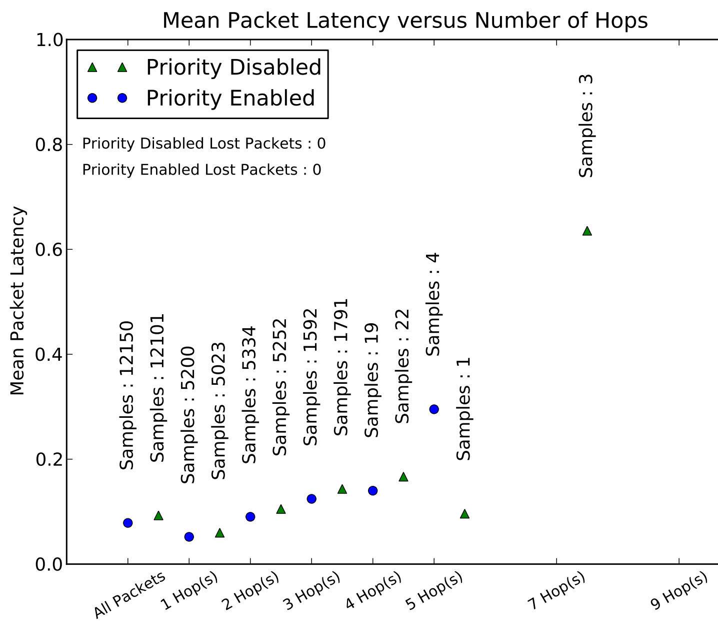
Figures 5.4 to figure 5.6 presents the mean latency results of simulation runs with different network sizes and different topologies, comparing token passing with and without prioritising for the same network. The data points show the mean packet latency results for all packets successfully received and then groups the packets together according to the number of hops it needed to reach the destination. The number of samples are included to see how many packets reached their destination successfully. Additional information is included to show how many packets were lost due to a full transmission queue.

Figure 5.4 on the next page show the results for two different networks with 20 nodes each. The difference in latencies between the simulations using token passing with priority and without priority is small. In subfigure (a) the decrease in latency by using prioritising can be clearly observed with the higher hop counts of 3, 4 and 5. A few odd samples can be seen with higher hop counts, these observations are for packets that followed a different route to a destination before the final route has converged.

Networks with 50 nodes each were tested and their results are given in figure 5.5 on page 85. The improvement in latency by using token prioritising becomes conspicuous with the larger network size. Over a thousand packets were lost due to a full transmission queue in both of the cases, where no prioritising were used.



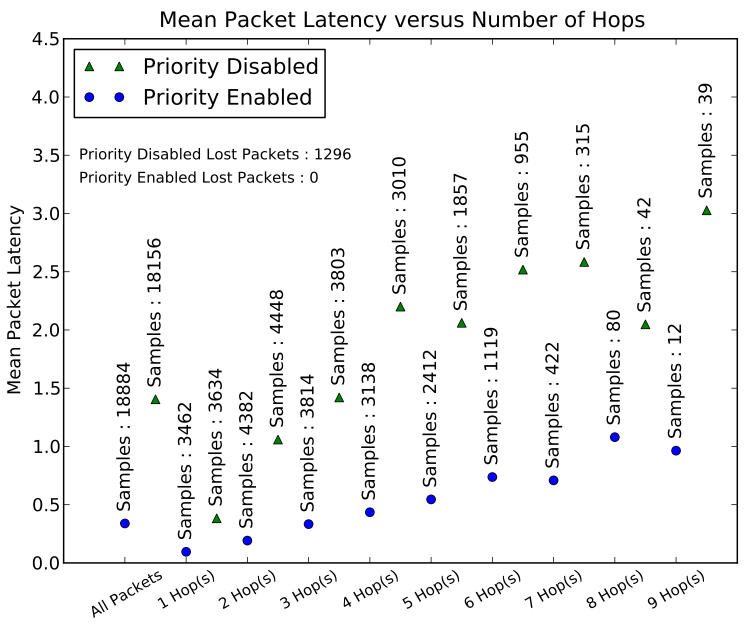
(a) Mean Latency Priority 20 Nodes Run 1.



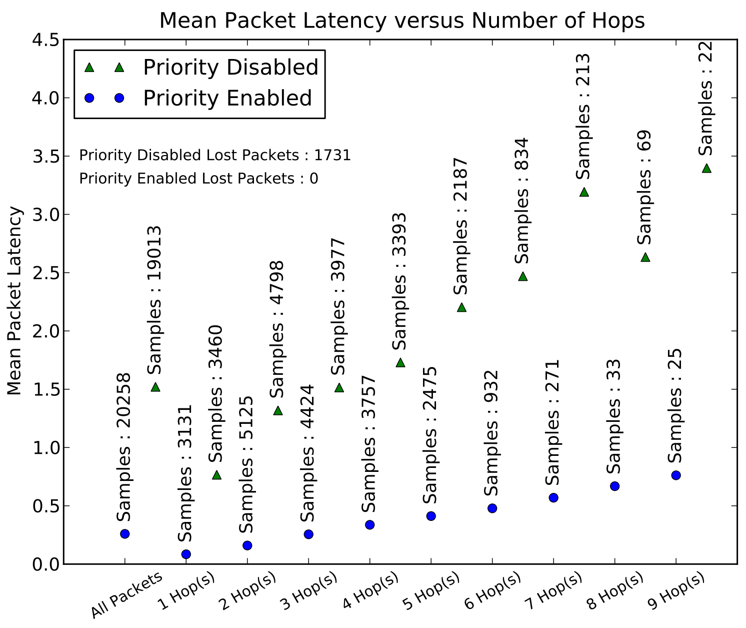
(b) Mean Latency Priority 20 Nodes Run 2.

Figure 5.4: *Effect of prioritising traffic on the mean packet latency for a network size of 20 nodes.*

A hundred nodes were used in both simulation setups that produced the subfigures of figure 5.6 on page 86. In subfigure (a), the latency is clearly improved by prioritisation, but subfigure (b) does not show this improvement. Subfigure (b) does not show the improved latency, because only packets are considered that successfully reach the destination and in



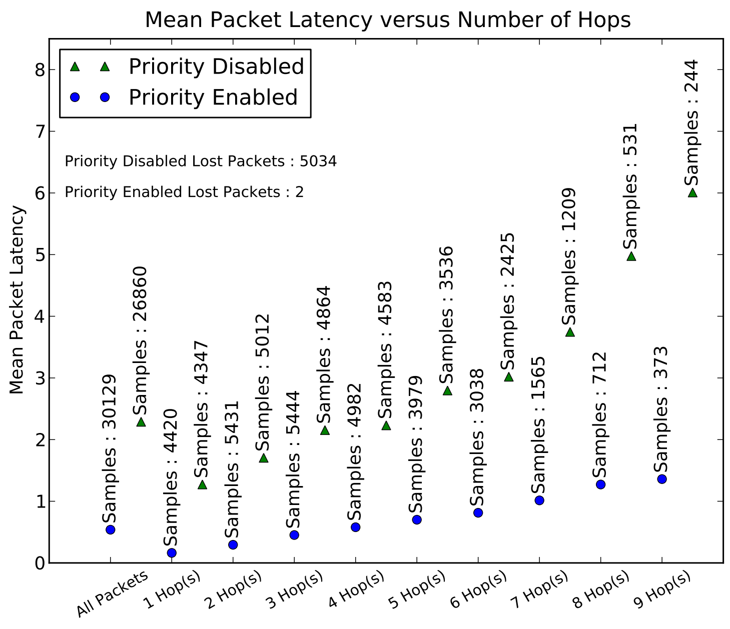
(a) Mean Latency Priority 50 Nodes Run 1.



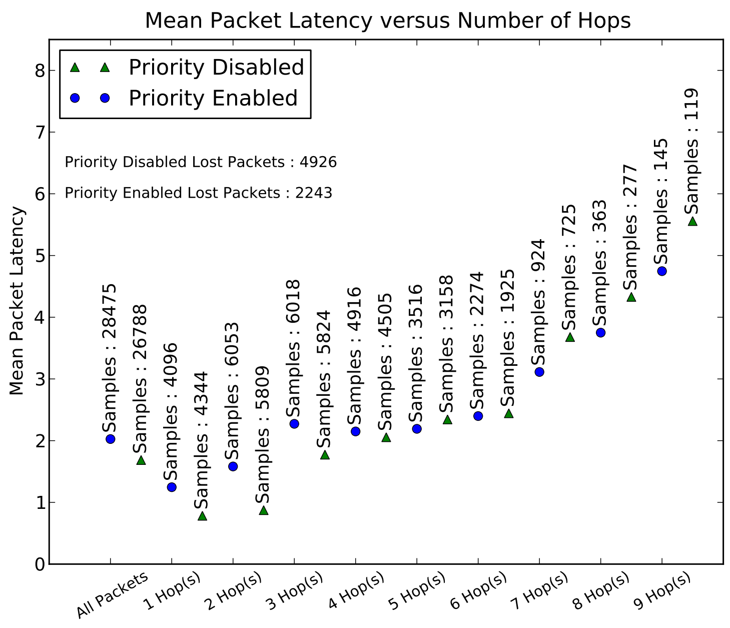
(b) Mean Latency Priority 50 Nodes Run 2.

Figure 5.5: *Effect of prioritising traffic on the mean packet latency for a network size of 50 nodes.*

the case where no token prioritisation is used, more packets are discarded due to a full buffer. These lost packets do not adversely affect the mean latency value. Another interesting observation is that with token prioritisation, at least 1000 more packets were delivered successfully in both cases.



(a) Mean Latency Priority 100 Nodes Run 1.



(b) Mean Latency Priority 100 Nodes Run 2.

Figure 5.6: *Effect of prioritising traffic on the mean packet latency for a network size of 100 nodes.*

Overall, the time it takes to deliver a packet is decreased when token prioritisation is used. With a larger network and higher hop counts, the effect on latency is clearer. In one case where the latency was not improved, the throughput was greater with token prioritisation.

5.6 Fast Rerouting

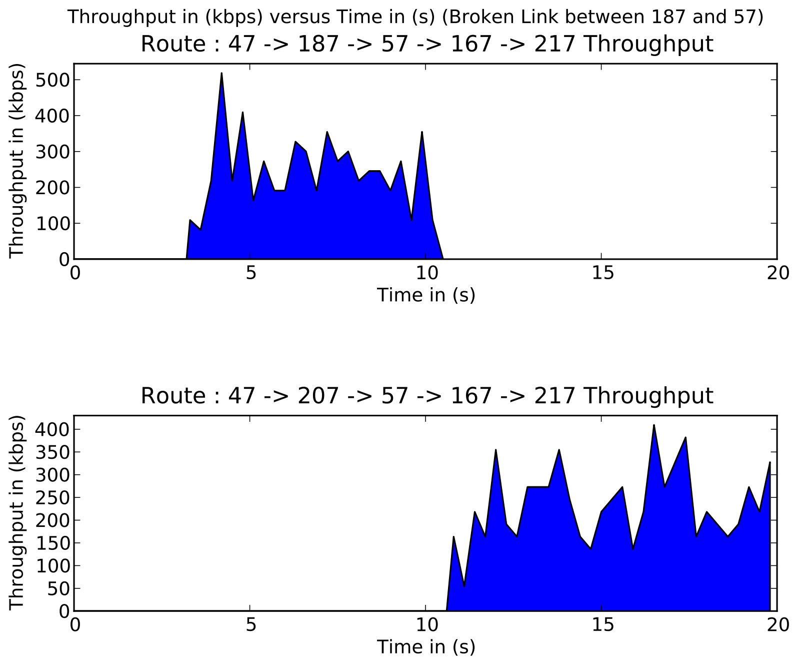
Since links between nodes can be easily be blocked at high carrier frequencies, an alternate route needs to be found quickly. Tests were performed to see how the routing protocol reacts when a single link in a route is broken.

In these tests, two nodes were selected in the network, that are multiple hops away from each other. The simulation is started and the clusters were set up. After 3 s into the simulation, the source node started to send messages to the destination node. A short time passes before the destination node begins to receive packets and a route is created. The transmission of packets along the route continued until 10 s into the simulation. At 10 s the link between the nodes in the middle of the route was broken (the nodes cannot detect any messages from one another). The maximum TTL field for a link with a neighbour node is 5 and the field is decremented every 0,1 s. Therefore, the link between neighbours should time-out after 0,5 s. After the time-out, a new route to the destination is established. The simulation ended at 20 s. Figure 5.7 and figure 5.8 shows the results of different simulation setups. The throughput was measured every 0,3 s.

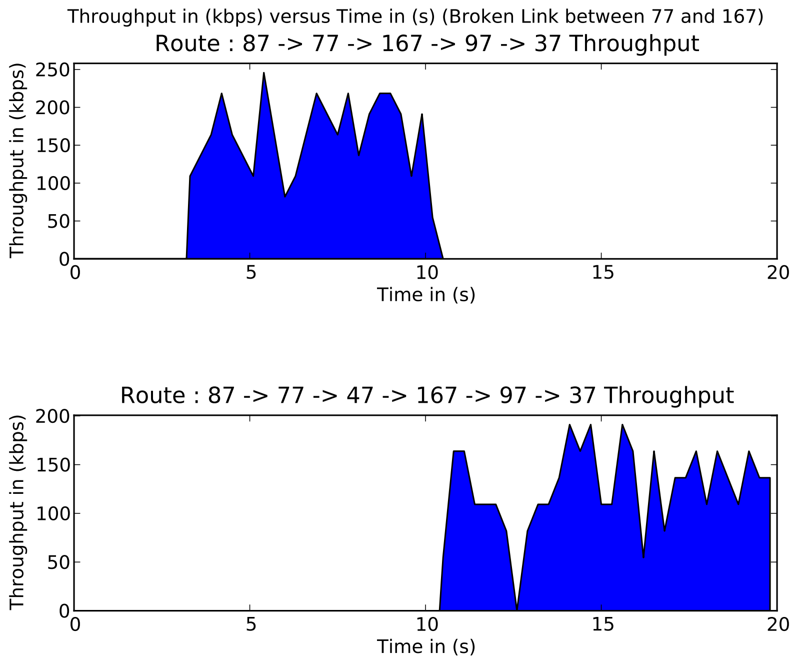
Two simulations with 20 nodes (figure 5.7) and two simulations with 50 nodes (figure 5.8) were executed. In the subfigures (a) and (b) of figure 5.7 on the following page, it can be seen that the throughput of the top route increases at 3 s and decreases at 10 s. At 10 s after a short delay, the throughput along the alternate route (bottom graph) increases. In subfigure (a), the last time packets were received via the original route is approximately 10,2 s. From 10,8 s onwards, packets are received along the reconstructed route. For subfigure (b), packets stop using the original route at 10,2 s and the new route is utilised at 10,5 s.

Subfigures (a) and (b) of figure 5.8 on page 89 each use 50 nodes. In subfigure (a) three different routes were used to reach the destination. An alternate route is used at approximately 7 s in subfigure (a) before the link between the nodes are broken. The reason for using the alternate route is assumed to be a time-out. After the link is broken at 10 s there are no more packets being routed along the original route, but the packets now follow two routes to the destination. The packets are following two routes, because firstly the metric (hop count) is the same for both routes and secondly, the topological cluster structure of both routes are the same (the destination node is a *satellite node* of both clusters). In subfigure (b) only two routes were used. The original route is used until 10,2 s and the adapted route is used from 10,8 s onwards.

Four cases of fast rerouting were investigated. In three of the cases the route was reconstructed within 1 s. The other case used more than two routes to the destination due to routing ambiguity, but also maintained communication throughout the 20 s of simulation.

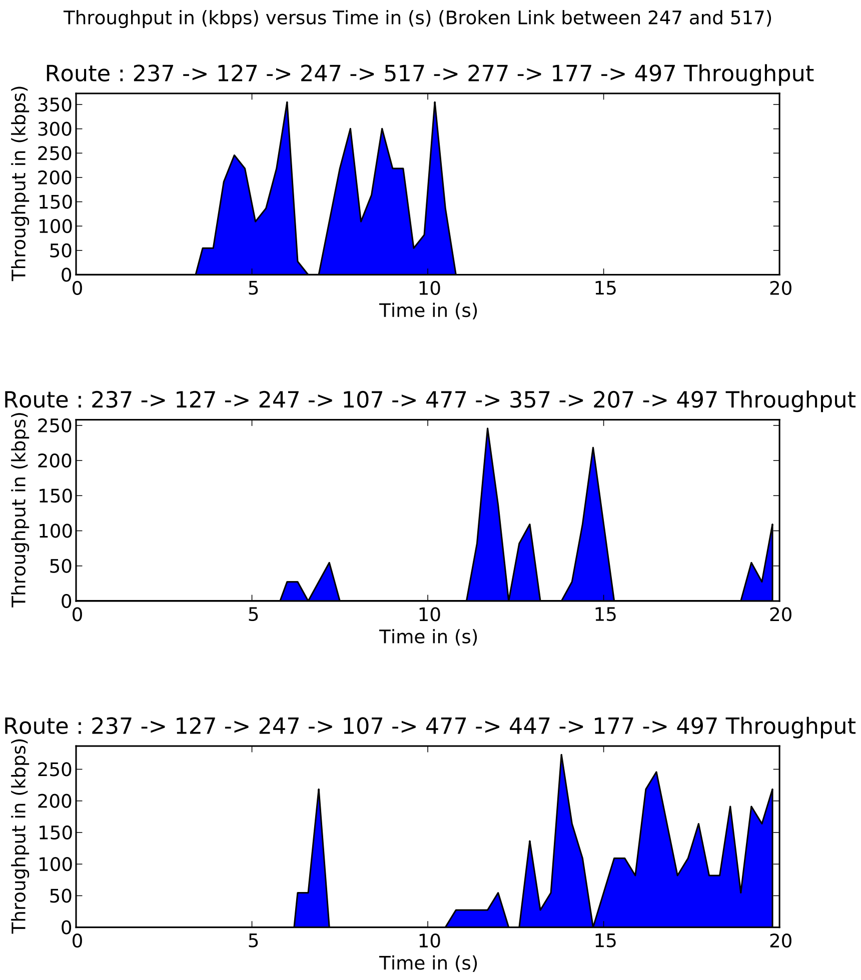


(a) Quick reroute 20 Nodes Run 1.

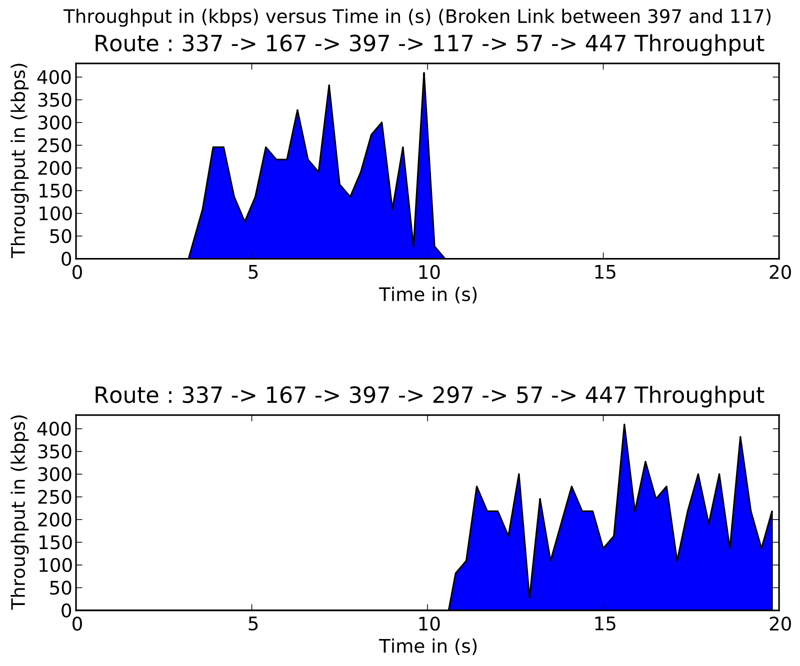


(b) Quick reroute 20 Nodes Run 2.

Figure 5.7: *Quick reroute (handover) of packets with broken link for a network size of 20 nodes.*



(a) Quick reroute 50 Nodes Run 1.



(b) Quick reroute 50 Nodes Run 2.

Figure 5.8: *Quick reroute (handover) of packets with broken link for a network size of 50 nodes.*

5.7 Summary

In this chapter various tests were performed to show the workings of the routing protocol. The general test setup was explained briefly.

The cluster setup phase of the protocol was tested by running simulations on nodes configured in random topologies and different network sizes. The tables present various results, but the important results were about the number of nodes that produce full routing information and the last *cluster-head* election time. With the original CGSR protocol all nodes produce node-cluster association and routing information overhead, but the tests show the nodes that producing full routing information can be reduced to at least 40 %. The tests also showed the setup time for clusters (last *cluster-head* election time) increases with the network size.

Next, it was shown that full aggregate connectivity can be achieved after a few seconds. The time needed to reach full connectivity also increases with the increasing diameter of the network.

A test was performed to show the effectiveness of the protocol at high node densities. The routing overhead per second per node was shown to decrease with increasing node density. This result can be compared with routing overhead per node of the DSDV protocol which is expected to increase linearly with an increase in the number of nodes.

Another test was performed to compare the latency when token prioritisation is used against the latency when token prioritisation is not used. When nodes are given more transmission chances due to more queued traffic, latencies are reduced. The reduction in time a packet needs to travel from source to destination becomes clearer at higher hop counts and larger network sizes. Where the latency of packets were not reduced, greater throughput was observed.

Lastly, a test was performed to show how a route is repaired when a link used by the route is broken. In most setups the a new route was established within 1 s.

In conclusion, all of the design outcomes of section 3.1 have been met.

Chapter 6

Conclusion

This thesis presents a cluster based routing protocol adapted for use in a MMW application. MMW ad hoc networks are relatively new and few applications exist. Although the routing protocol was only tested at 2,4 GHz in a simulated environment, changes were made for operation at 17 GHz.

At higher frequencies the diffraction of electromagnetic waves reduces [39] and the ability of waves to travel through solid objects become impaired [28]. To compensate for the light-like waves that can be easily blocked, the routing protocol was changed to recover with ease from single link breakages.

The protocol utilises routing overhead efficiently at high node densities, which makes it suitable for use in an indoor environment with a high number of devices. A conference setup with many, but almost stationary people could be an ideal application. Congestion is improved by giving nodes with more queued messages increased bandwidth. Nodes make use of a large available bandwidth by sending hello messages frequently, thus keeping local routing information fresh.

This chapter will summarise the findings of our research: stating the contributions to the research area; the advantages and disadvantages of using a cluster based protocol; and possible improvements of the protocol. Some perceived shortcomings of the study and possibilities for future work, will also be covered.

6.1 Summary of Research Findings and Contributions

6.1.1 General Findings

- Basic data communication network concepts were introduced and various routing protocols investigated. Cluster based protocols was found to provide a good routing protocol solution at high node densities, as routes converge faster and the number of routes can be reduced to the number of clusters. The reduction in the number of routes is advantageous, but comes at a price: nodes now need to know the node-cluster associations of the other nodes.
- Advantages of cluster based routing:
  - Instead of having a route entry for every node, each node now only needs a route entry for every cluster.
  - A *cluster-head* creates a landmark which nodes can use to orientate themselves and determine a forwarding direction by identifying neighbours of the *cluster-head*.
- Disadvantages of cluster based routing:
  - Cluster based routing creates increased logic complexity with regard to the node states. Each node state has a different function and behaviour.
  - Another drawback is that nodes become interdependent, because nodes need a *cluster-head* to have a route.

6.1.2 Findings Related to Enhanced CGSR

- Upon careful study of the CGSR protocol it was found that certain improvements can be made to the protocol. First of all, routes can be shortened by including the knowledge of the next-hop-cluster. The number of nodes producing full routing information can be reduced, by only letting selected nodes transmit full routing overhead and node-cluster association information.

- A performance module was implemented that allowed for logical separation of performance code from routing layer code. The performance module enabled measurement of aggregate parameters at simulation time.

- It is difficult to find an accurate theoretical model using queuing theory. Although the theoretical model's predicted cluster cycle time corresponds to the measured cluster cycle time in some cases, it differs greatly in other comparisons of the simulation and theoretical model. Only one case predicts the route latency with accuracy. It is suggested that different mathematical modelling techniques should be investigated such as a Markov state model or Petri net. Petri nets can be used to model concurrent systems [45].

- Results show that our cluster generation algorithm can form clusters successfully for network sizes of 20, 50 and 100 nodes. The last *cluster-head* election time (cluster setup time) increases with increasing network size. Although fewer nodes are used to forward routing information, results prove that full connectivity can be achieved for network sizes of 20, 50 and 100 nodes.

- Routing overhead was also tested with increasing node density to determine routing overhead efficiency. It was found that the routing overhead per node decreases with increasing node density (increasing the number of nodes within the same area). It can be compared with DSDV's routing overhead per node, which is expected to grow linearly ($O(N)$) with increasing number of nodes. The cost in overhead, of sending hello messages at a fast rate, is high, but the large available bandwidth makes this allowable.

- The effect of token prioritising on mean packet latency was also studied to check if congestion of the network can be improved with a high data load. The token prioritisation mechanism gives nodes with more queued traffic more transmission chances. It was found that the latency improved with prioritisation. At higher hop counts and larger network sizes, the improvement on packet latency is accentuated. In situations where the travel time of a packet was not decreased, higher aggregate data throughput was noted and decreased packet loss due to a full transmission queue. Token prioritising provides a platform for better Quality of Service support in the network.

- Another test was performed to show the network protocol's resilience to single link breakages. It was found that by sending hello messages at a high rate and letting the TTL of a link with a neighbour time-out quickly, the protocol is able to recover within 1 s. By using a proactive protocol rather than reactive protocol, the need for using a route repair packet is negated. A route repair packet may impose a delay in repairing the route.

6.2 Recommendations

The following section will state improvements that can be made to the protocol. If the information of a cluster changes or is lost, the information of the specific cluster can be flooded throughout the network by forwarder nodes. First Declaration Wins (FDW) clustering [15] can be used for faster cluster setup. The MAC-layer can be altered for improved token passing support.

Currently the protocol does not incorporate link quality as part of its routing metric, the number of neighbours is used as the routing metric. It is suggested that a one-way version of the Expected Transmission Count (ETX) metric [9] is used to provide a measure of link quality sensing.

The periodic generation time of hello messages can be optimised. Too many hello messages can congest the channel and less hello messages can impair the protocol's ability to sense *unstable* links.

6.3 Shortcomings

Perceived shortcomings of the implemented protocol will be mentioned here. The protocol was only simulated at 2,4 GHz. It is expected that the channel will have increased fading at a carrier frequency of 17 GHz.

Unidirectional links are not well supported. Hello messages only include the addresses of critical nodes and not all neighbours. Therefore, it is not possible to detect unidirectional links between a node and a neighbour where neither is a critical node. It is suggested that if the MAC-layer repeatedly detects an error sending a data packet to a node with a *stable* link, the address of the node is recorded. The sending node should then refrain from sending data to the recorded node for a certain backoff time.

6.4 Interesting Remarks

It was only necessary to include data of critical nodes in the hello message, for correct protocol operation. In a sense, hello messages provide information on the link-state of the critical nodes within the two hop range. Routing information on nodes further away than two hops are distributed throughout the network, with a distance vector method. The protocol can be classified as a proactive hybrid protocol using distance vector globally and link-state locally.

6.5 Future Research

Our work developed a routing protocol for a MMW radio network and simulated it within the OMNET++ network simulator, at a carrier frequency of 2,4 GHz. A physical layer model must be developed for simulating the protocol at higher frequencies. The model will enable us to determine the effects of greater fading and a possibly worse error rate on the network. The desired carrier frequency for the network is to be at 17 GHz or 60 GHz.

Hardware implementation of the network at 2,4 GHz can be useful, as a first check on possible practical implementation. The hardware for the 17 GHz must then be developed and tested. An attempt could also be made to design hardware for an application at 60 GHz, because of the unlicensed spectrum in the $57 - 64$ GHz band [17].

Cluster based protocols is another area of research that can be pursued, due to a lack of a fixed standard implementation. There has been a attempt to promote an on-demand Cluster Based Routing Protocol (CBRP) by a submitted Internet Draft [20], but this was not approved as a Request For Comment yet. The following articles: [22, 7, 20, 10, 15], provide interesting research information on the cluster based protocol.

With the token prioritisation a feedback mechanism was introduced into the system. Future research can focus on modelling feedback systems in these types of data networks. Network conditions could be used as input parameters with the system reacting accordingly.

Modelling routing protocols with graphing theory [42] may provide greater insight into problems.

Bibliography

[1] "IEEE 802.15c Working Group Website." http://www.ieee802.org/15/pub/TG3c.html. February 2011.

[2] "Multipoint Relay Picture." http://wiki.uni.lu/secan-lab/graphics/olsr02.gif. February 2011.

[3] "Seven Layer OSI Picture." http://www.codeguru.com/cpp/sample_chapter/article.php/c12219/. February 2011.

[4] "Wireless 802.11b Transceiver." http://goods.us.marketgid.com/goods/19387/. February 2011.

[5] ABOLHASAN, M., WYSOCKI, T., and DUTKIEWICZ, E., "A review of routing protocols for mobile ad hoc networks." *Ad Hoc Networks*, June 2004, Vol. 1, No. 22.

[6] AKYILDIZ, I. F. and WANG, X., "A Survey on Wireless Mesh Networks." *IEEE Radio Communications*, September 2005.

[7] AL-GHAZAL, M., EL-SAYED, A., and KELASH, H., "Routing Optimization using Genetic Algorithm in Ad Hoc Networks." *IEEE International Symposium on Signal Processing and Information Technology, Cairo, Egypt*, 2007.

[8] BANKS, J., John S. Carson, I., NELSON, B. L., and NICOL, D. M., *Discrete-Event System Simulation*. Prentice Hall, 2001.

[9] BAUMANN, R., HEIMLICHER, S., STRASSER, M., and WEIBEL, A., "A Survey on Routing Metrics." *TIK Report 262*, February 2007.

[10] BELDING-ROYER, E. M., "Hierarchical Routing In Ad Hoc Mobile Networks." *Wireless Communications and Mobile Computing*, 2002, pp. 515–532.

[11] BRENNER, P., *A Technical Tutorial on the IEEE 802.11 Protocol*. BreezeCOM, 1996.

[12] CHIANG, C.-C., WU, H.-K., LIU, W., and GERLA, M., "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel." 1997.

[13] DATTATREYA, G. R., *Performance Analysis of Queuing and Computer Networks*. Chapman & Hall / CRC Computer and Information Science Series. CRC Press, 2008.

[14] DOUFEXI, A., ARMOUR, S., BUTLER, M., NIX, A., BULL, D., and MCGEEHAN, J., "A Comparison of the HIPERLAN/2 and IEEE 802.11a Wireless LAN Standards." *IEEE Communications Magazine*, May 2002.

[15] GERLA, M., KWON, T. J., and PEI, G., "On Demand Routing in Large Ad Hoc Wireless Networks with Passive Clustering." in *Proceedings of IEEE WCNC 2000, Los Angeles*, 2000.

[16] GERLA, M. and TSAI, J. T.-C., "Multicluster, mobile, multimedia radio network." *Wireless Networks*, 1995, Vol. 1, pp. 255–265.

[17] GUO, N., QIU, R. C., MO, S. S., and TAKASHI, K., "60-GHz Millimeter-Wave Radio: Principle, Technology and New Results." *EURASIP Journal on Wireless Communication and Networking*, 2007.

[18] HALLS, G. A., "HIPERLAN: the high performance radio local area network standard." *Electronics & Communication Engineering Journal*, December 1994.

[19] IWATA, A., CHIANG, C.-C., PEI, G., GERLA, M., and CHEN, T.-W., "Scalable Routing Strategies for Ad Hoc Wireless Networks." *IEEE Journal on Selected Areas in Communications*, August 1999, Vol. 17, No. 8, pp. 1369–1379.

[20] JIANG, M., LI, J., and TAY, Y. C., "Internet Draft: Cluster Based Routing Protocol (Work In Progress)."
http://www.math.nus.edu.sg/ mattyc/cbrp.txt. August 1999.

[21] JOHNSON, D. B. and MALTZ, D. A., "Dynamic source routing in ad hoc wireless networks." in *Mobile Computing*, pp. 153–181, Kluwer Academic Publishers, 1996.

[22] LEE, B., YU, C., and MOH, S., "Issues in Scalable in Clustered Network Architecture for Mobile Ad Hoc Networks." *Handbook of Mobile Computing*, 2004.

[23] LITTLE, J. D. C., "A proof for the queueing formula." *Operations Research*, November 1960, Vol. 9, No. 3.

[24] LÖBBERS, M. and WILLKOMM, D., *A Mobility Framework for OMNeT++ User Manual Version 1.0a4*. Omnet++ Community.

[25] MOHAPATRA, P. and KRISHNAMURTHY, S. V. (Eds), *Ad Hoc Networks, Technologies and Protocols*, Ch. 1, p. 2. Springer, 2004.

[26] MORRISON, D. W., "Using ad hoc wireless networks to enable intelligent transport systems: The design and analysis of the TH(O)RP routing protocol." Master's thesis, University of Stellenbosch, December 2006.

[27] MURTHY, C. S. R. and MANOJ, B. S., *Ad Hoc Wireless Networks, Architectures and Protocols*, Ch. 1, p. 7. Prentice Hall, 2004.

[28] MURTHY, C. S. R. and MANOJ, B. S., *Ad Hoc Wireless Networks, Architectures and Protocols*, Ch. 1, p. 4. Prentice Hall, 2004.

[29] MURTHY, C. S. R. and MANOJ, B. S., *Ad Hoc Wireless Networks, Architectures and Protocols*. 2004.

[30] NORTH, S. C., *Neato Users Manual*, April 2004.

[31] PEI, G., GERLA, M., HONG, X., and CHIANG, C.-C., "A Wireless Hierarchical Routing Protocol with Group Mobility." *IEEE*, 1998.

[32] PERKINS, C. E. and BHAGWAT, P., "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers." *SIGCOMM*, 1994.

[33] PETERSON, L. L. and DAVIE, B. S., *Computer Networks*. 3 edition. Morgan Kaufmann, 2003.

[34] PYTHON SOFTWARE FOUNDATION. *Python Documentation version 2.6.4*, November 2009.

[35] RAMEY, C. and FOX, B., *Bash Reference Manual*, February 2009.

[36] ROSSOUW, C. M., "The Design of a Low Cost Ad-hoc Network for Short Distance Data Acquisition." Master's thesis, Stellenbosch University, December 2008.

[37] SHEPHARD, S., *Telecom Crash Course*, Ch. 1, p. 2. McGraw-Hill, 2005.

[38] SHOJI, Y. *et al.*, "Millimeter-Wave Ad-hoc Wireless Access System - (1) System Overview -." *IEEE Topical Conference on Wireless Communication Technology*, 2003.

[39] SMULDERS, P., "Exploiting the 60 GHz Band for Local Wireless Multimedia Access: Prospects and Future Directions." 2002.

[40] VAN ELLEWEE, S., "Determining a Least-cost Routing and MAC Strategy for a Rural Communications Ad hoc Network." Master's thesis, University of Stellenbosch, December 2006.

[41] VARGA, A., *OMNeT++ User Manual Version 3.2*. Omnet++ Community.

[42] WILSON, R. J. and WATKINS, J. J., *Graphs An Introductory Approach*. John Wiley & Sons, Inc., 1990.

[43] WIN, M. Z. and SCHOLTZ, R. A., "Impulse radio: how it works." *IEEE Communications Letters*, 1998, Vol. 2, pp. 36–38.

[44] WOLHUTER, R. and VAN ROOYEN, G.-J., "Elements of telecommunications systems design and teletraffic analysis."

[45] ZHOU, M. C. and VENKATESH, K., *Modeling, Simulation and Control of Flexible Manufacturing Systems, A Petri Net Approach*, Ch. 1, p. 6. World Scientific, 1999.

Appendix A

Network Graphs

The appendix contains all the graphs of the different simulations. Networks used in the network setup tests are shown in the first section (A.1). Note that the networks used in the fast route reconstruction test is the same as the first two networks as the network setup tests in subsection A.1.1 and subsection A.1.2. The second section (A.2) presents the networks used in the test measuring the routing control overhead with increasing node density. Finally, networks used in the token prioritising tests are displayed in the last section (A.3).

A.1 Network Setup Graphs

A.1.1 20 Nodes

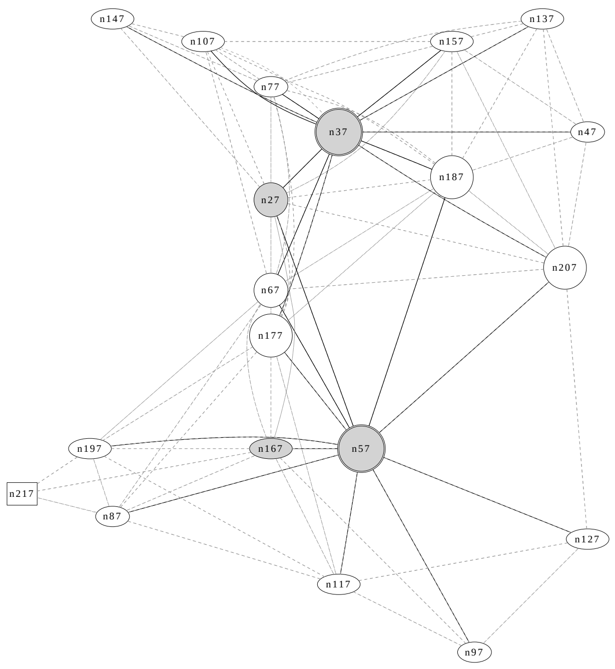


Figure A.1: *Twenty nodes run 1.*

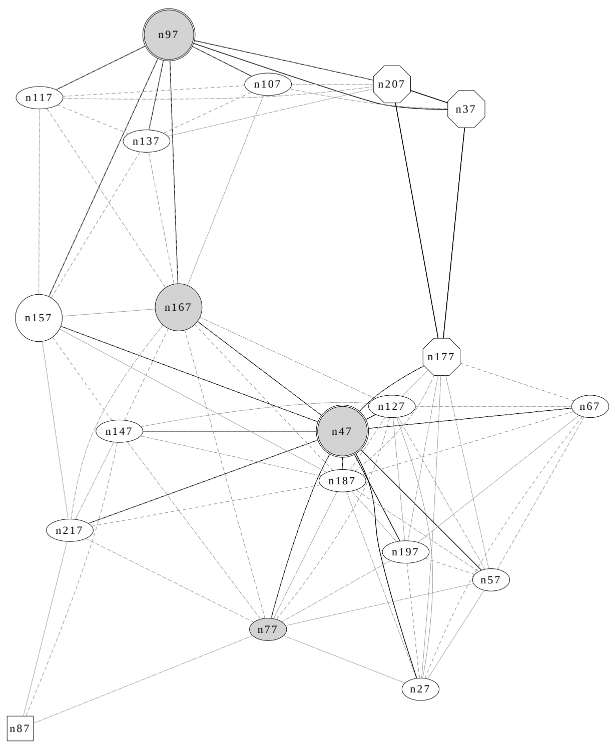


Figure A.2: *Twenty nodes run 2.*

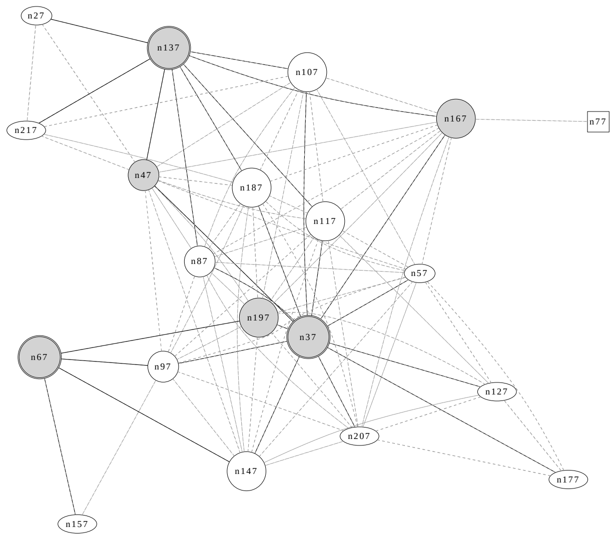


Figure A.3: *Twenty nodes run 3.*

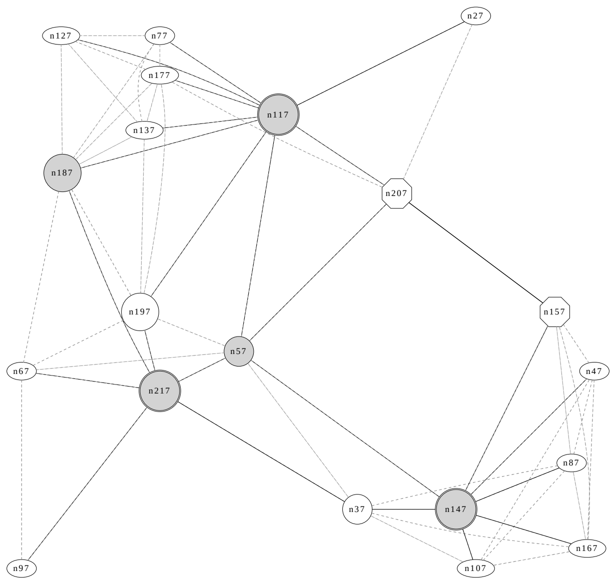


Figure A.4: *Twenty nodes run 4.*

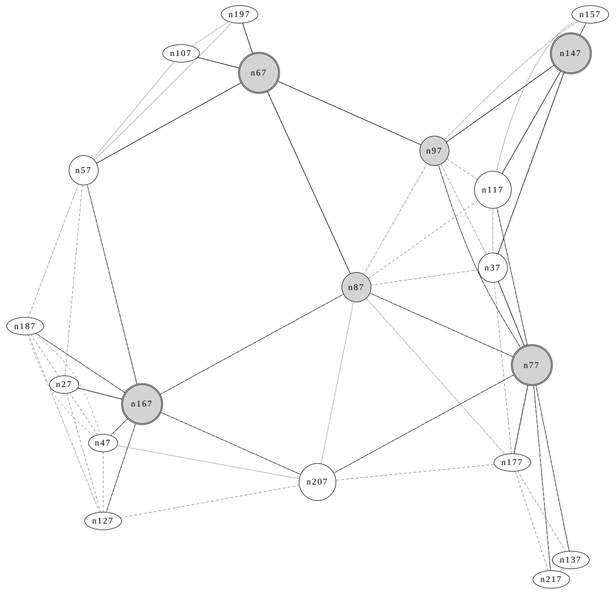


Figure A.5: *Twenty nodes run 5.*

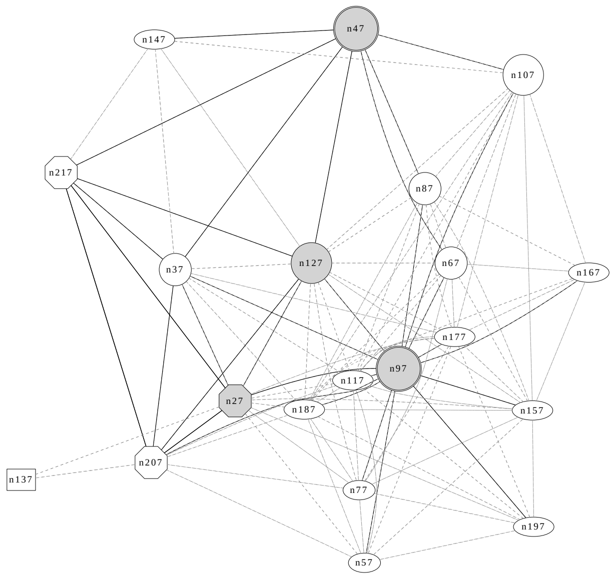


Figure A.6: *Twenty nodes run 6.*

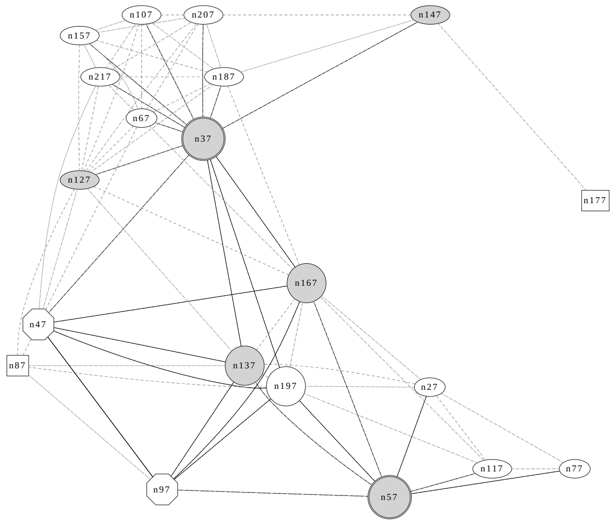


Figure A.7: *Twenty nodes run 7.*

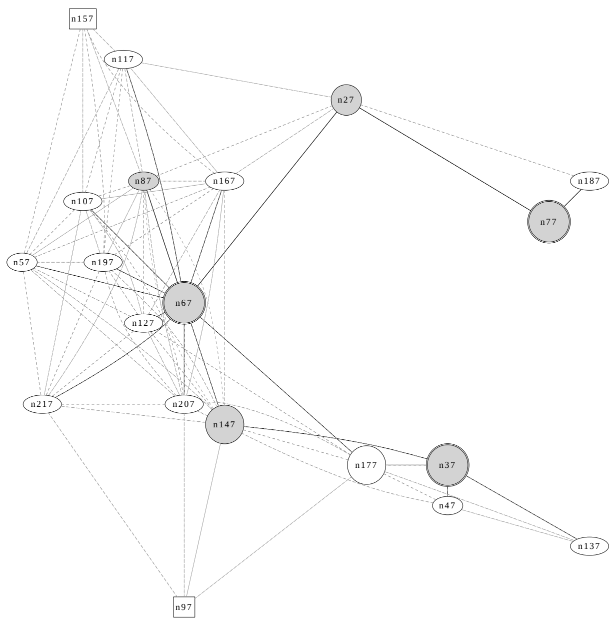


Figure A.8: *Twenty nodes run 8.*

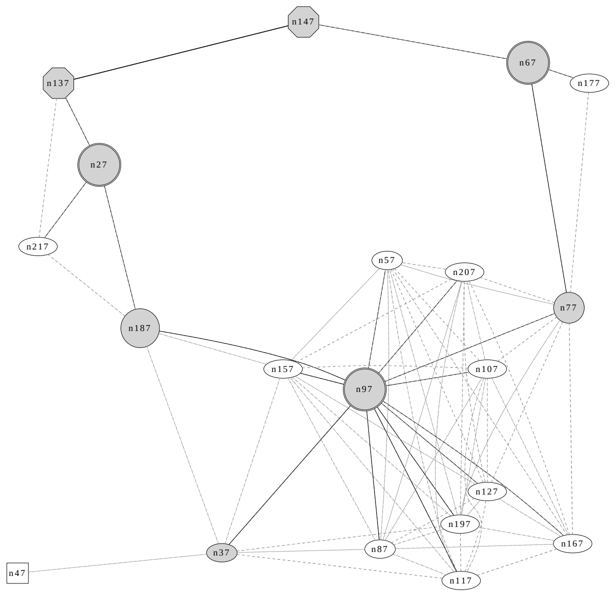


Figure A.9: *Twenty nodes run 9.*

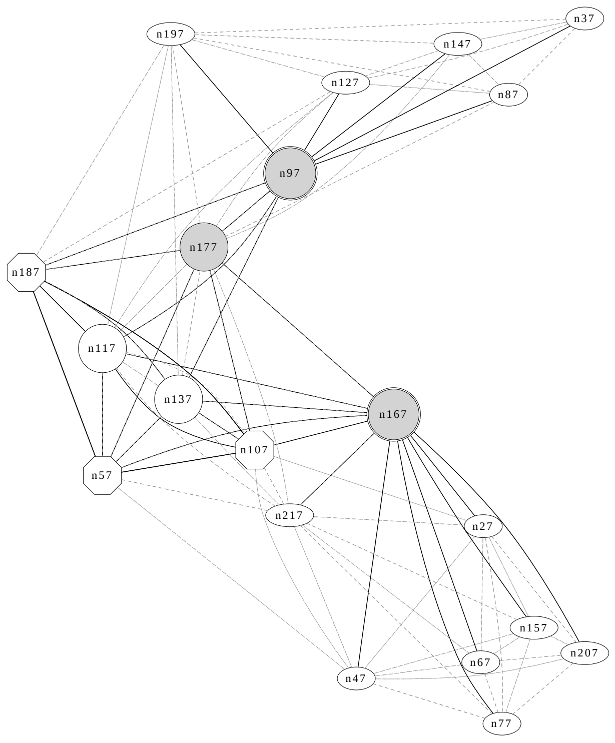


Figure A.10: *Twenty nodes run 10.*

A.1.2 50 Nodes

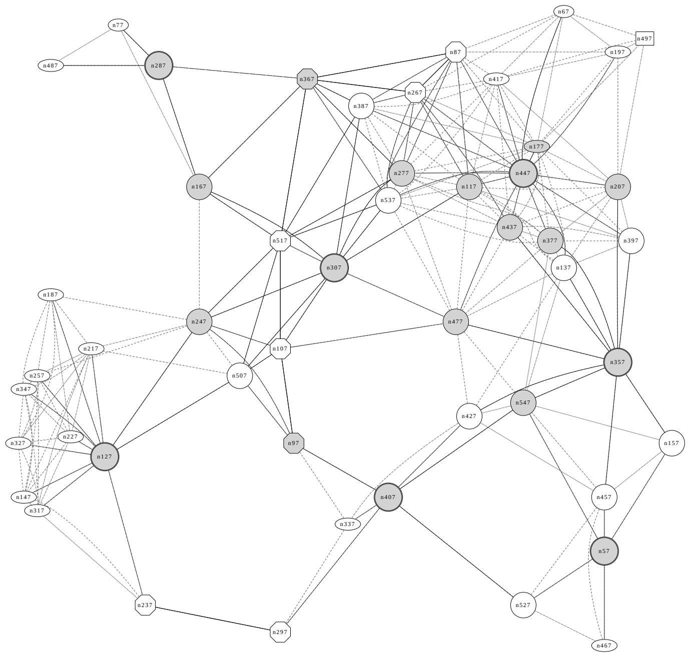


Figure A.11: *Fifty nodes run 1.*

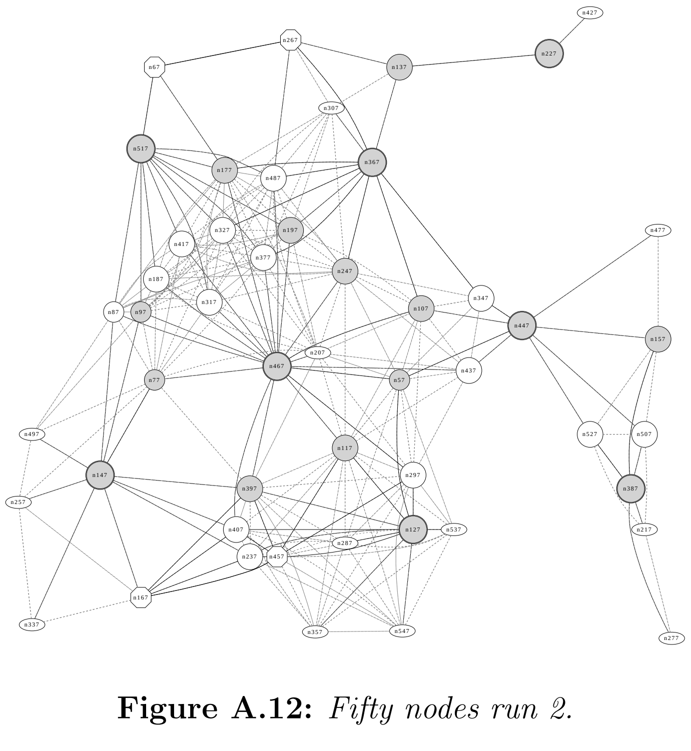


Figure A.12: *Fifty nodes run 2.*

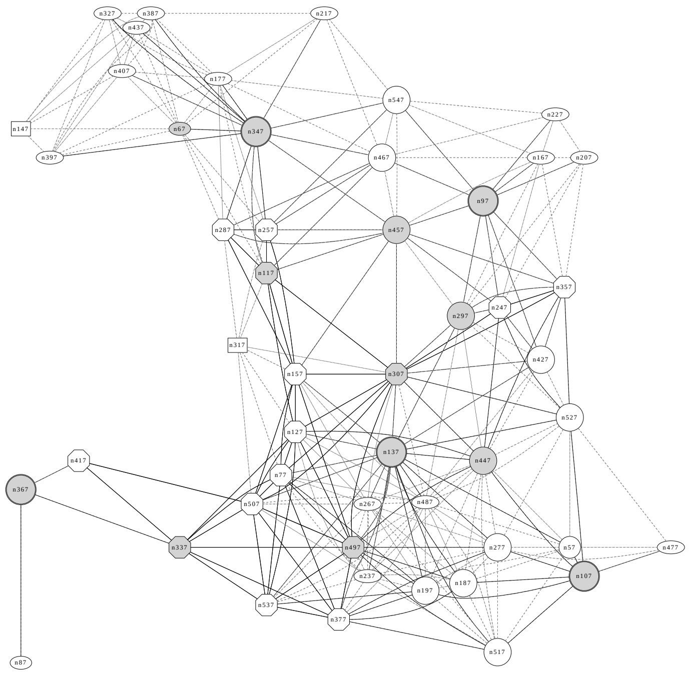


Figure A.13: *Fifty nodes run 3.*

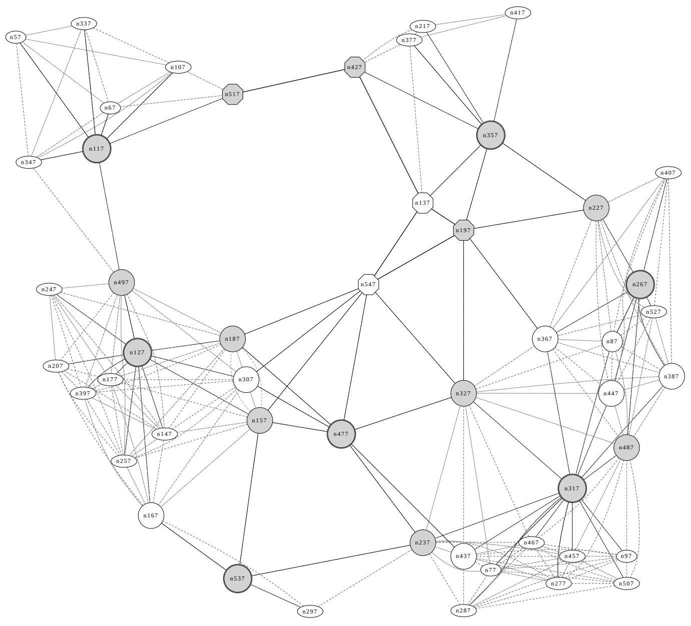


Figure A.14: *Fifty nodes run 4.*

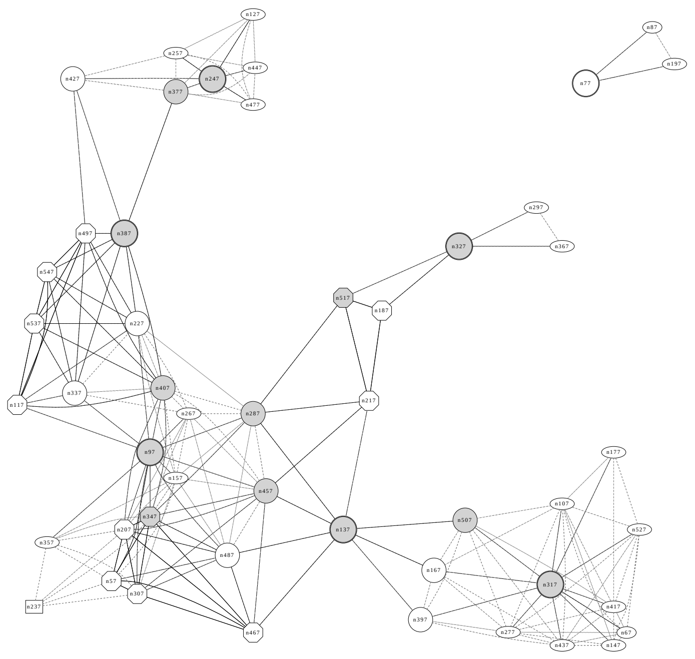


Figure A.15: *Fifty nodes run 5.*

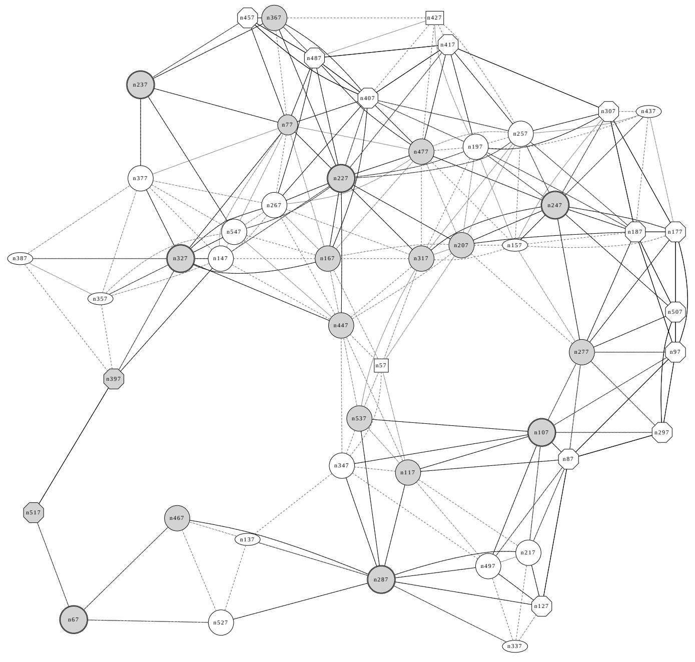


Figure A.16: *Fifty nodes run 6.*

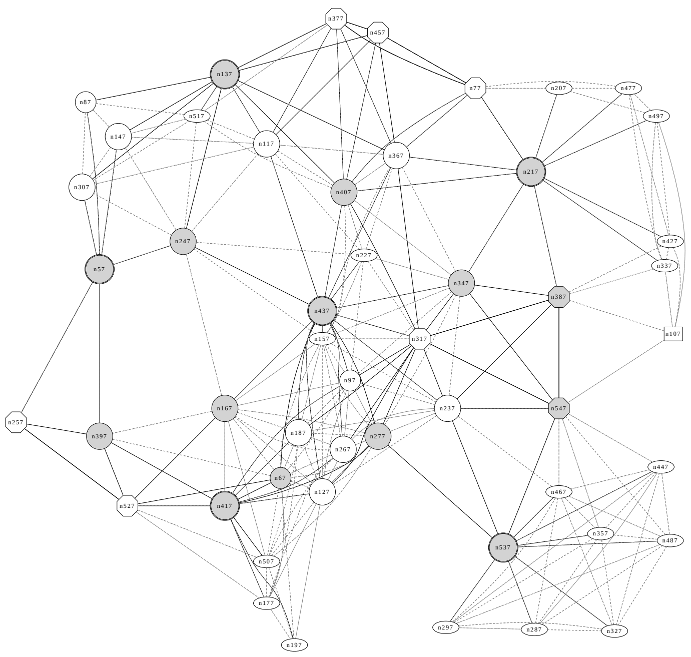


Figure A.17: *Fifty nodes run 7.*

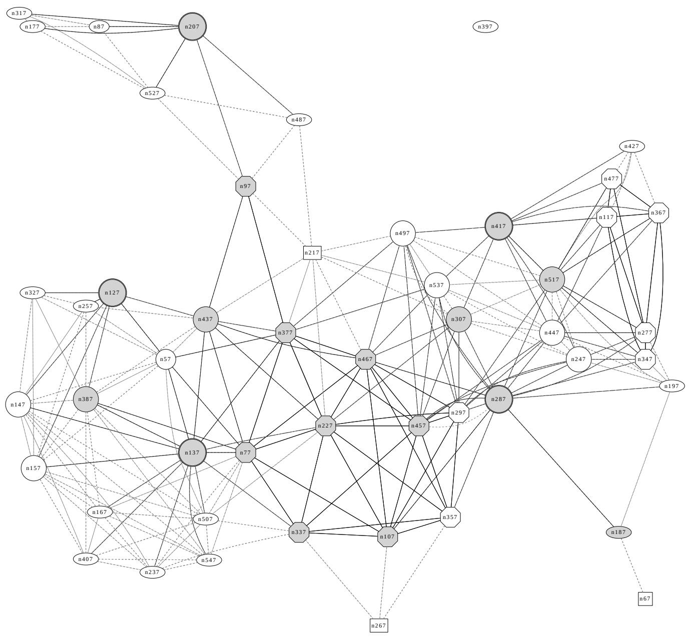


Figure A.18: *Fifty nodes run 8.*

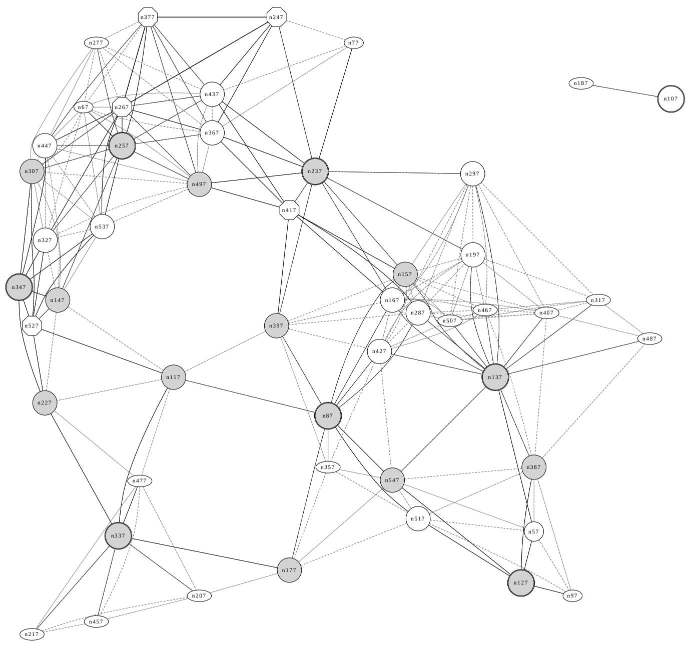


Figure A.19: *Fifty nodes run 9.*

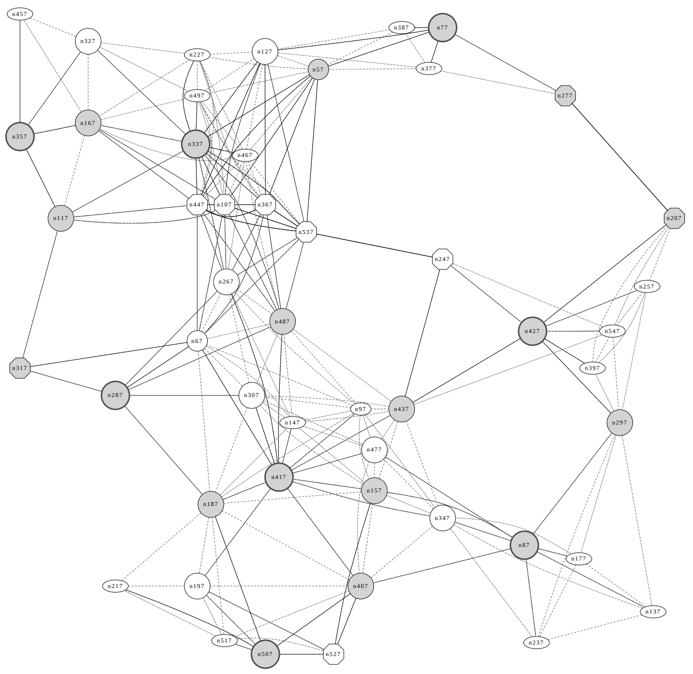


Figure A.20: *Fifty nodes run 10.*

A.1.3 100 Nodes

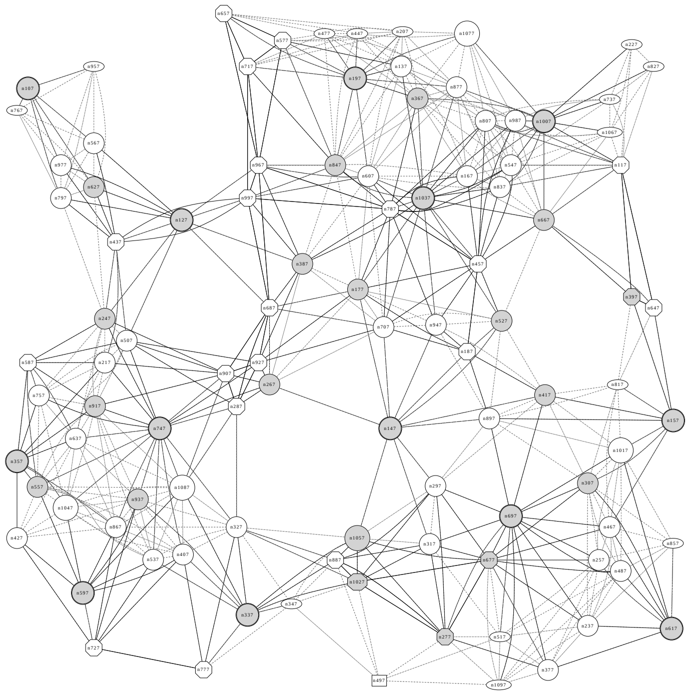


Figure A.21: *Hundred nodes run 1.*

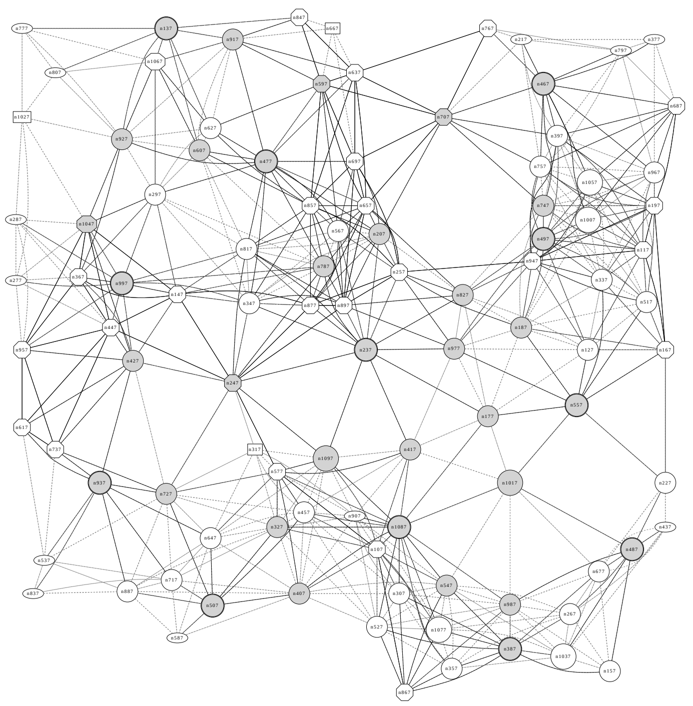


Figure A.22: *Hundred nodes run 2.*

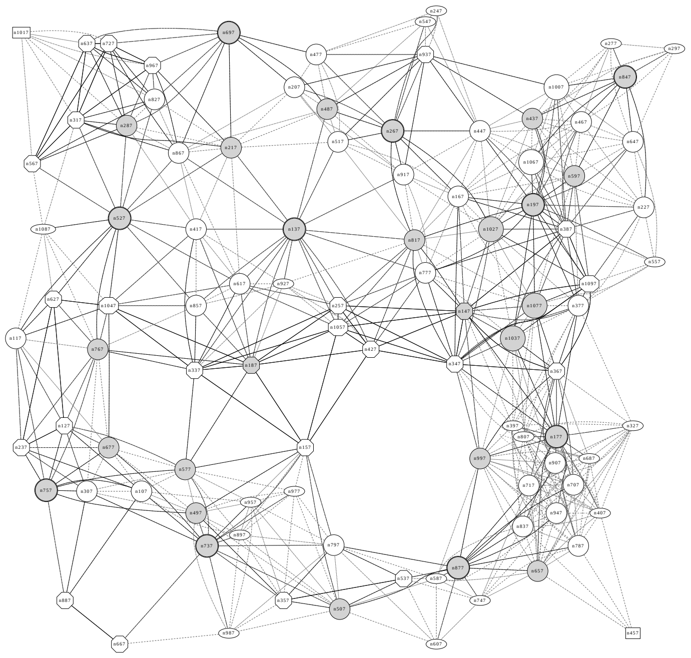


Figure A.23: *Hundred nodes run 3.*

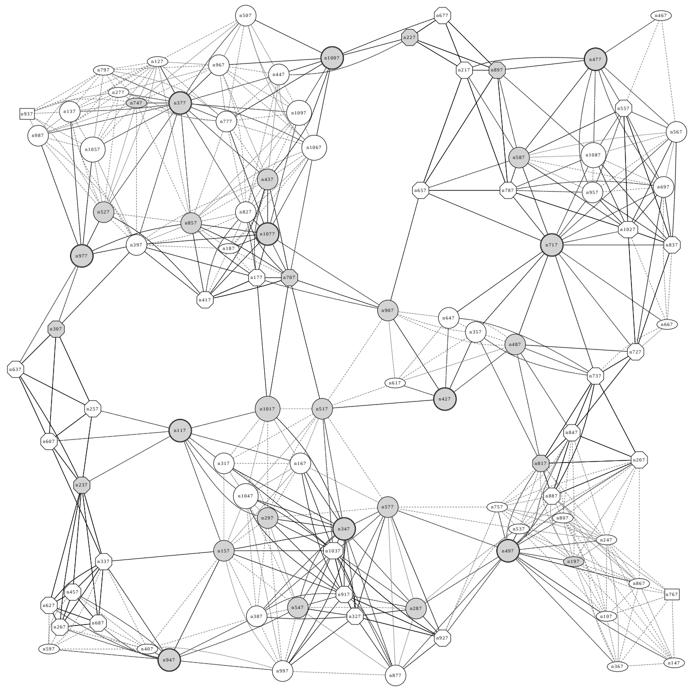


Figure A.24: *Hundred nodes run 4.*

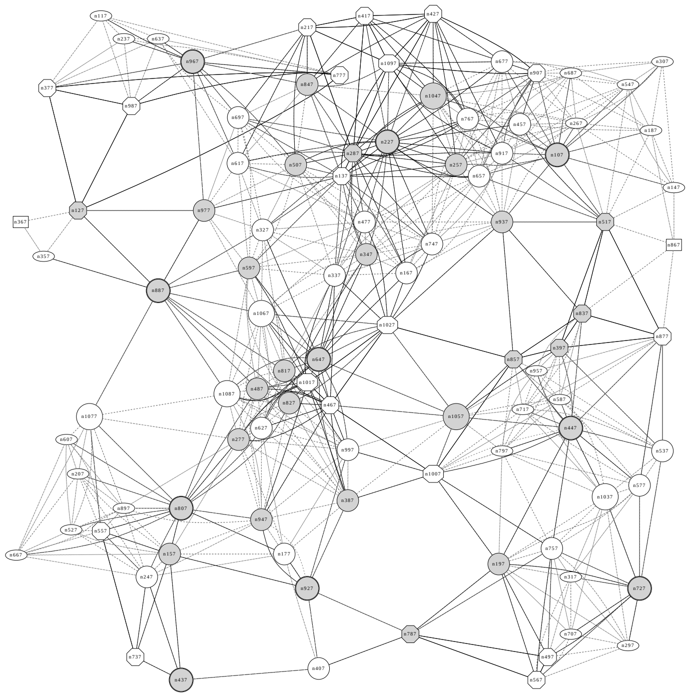


Figure A.25: *Hundred nodes run 5.*

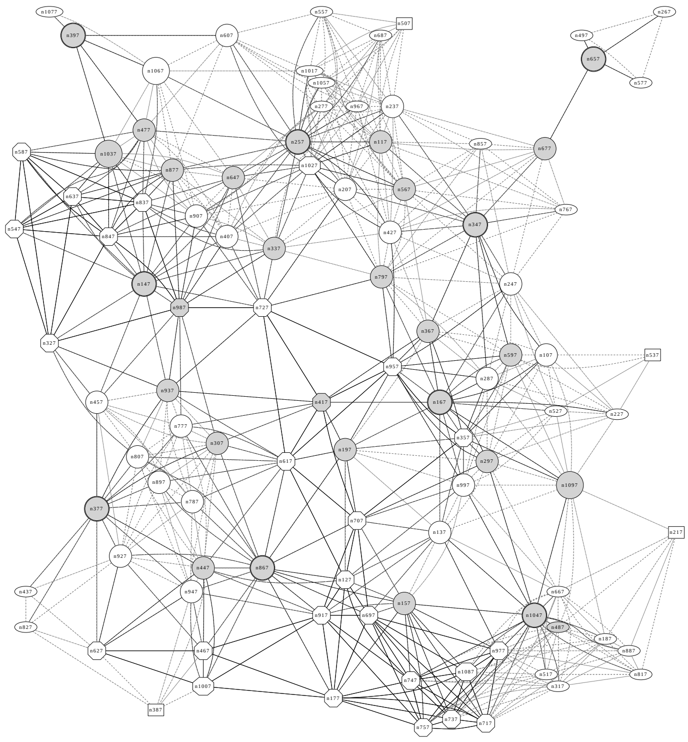


Figure A.26: *Hundred nodes run 6.*

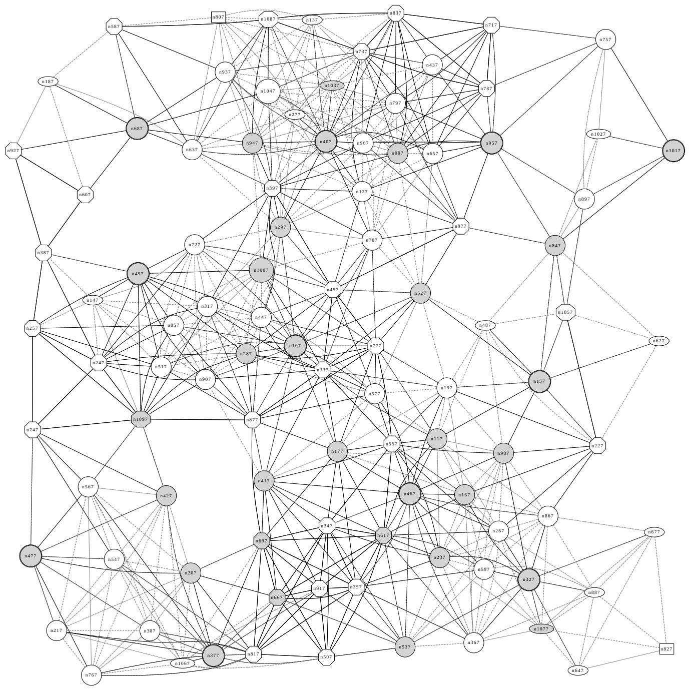


Figure A.27: *Hundred nodes run 7.*

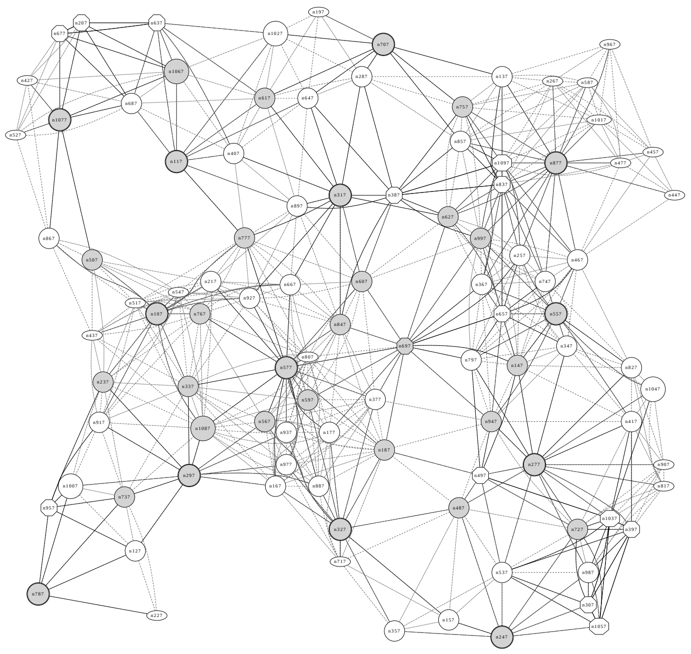


Figure A.28: *Hundred nodes run 8.*

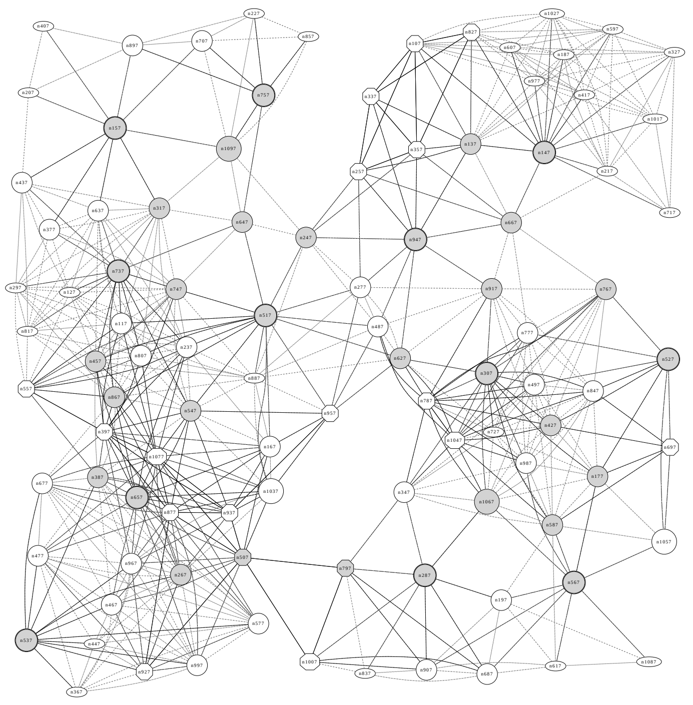


Figure A.29: *Hundred nodes run 9.*

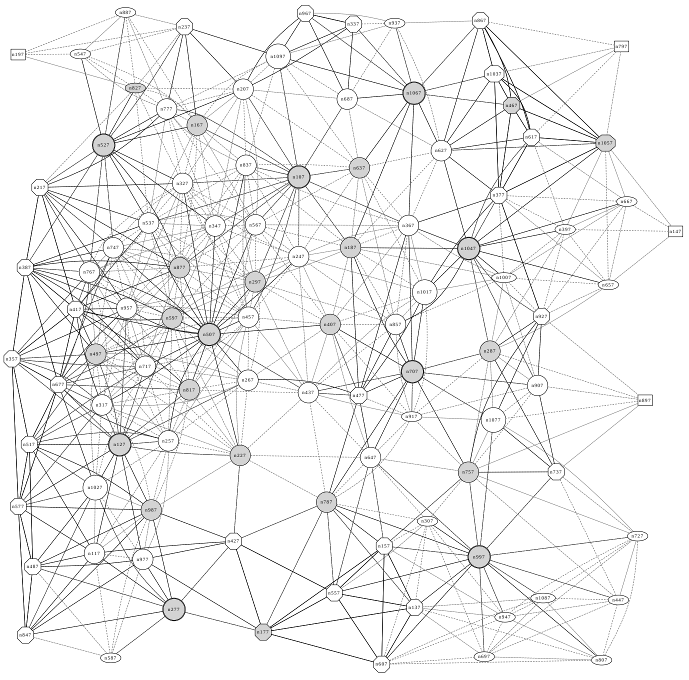


Figure A.30: *Hundred nodes run 10.*

A.2 Increasing Node Density Networks

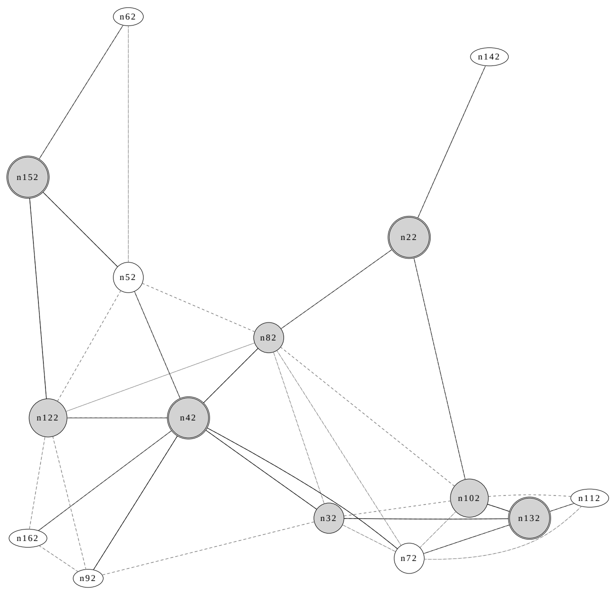


Figure A.31: *Increasing node density run 1.*

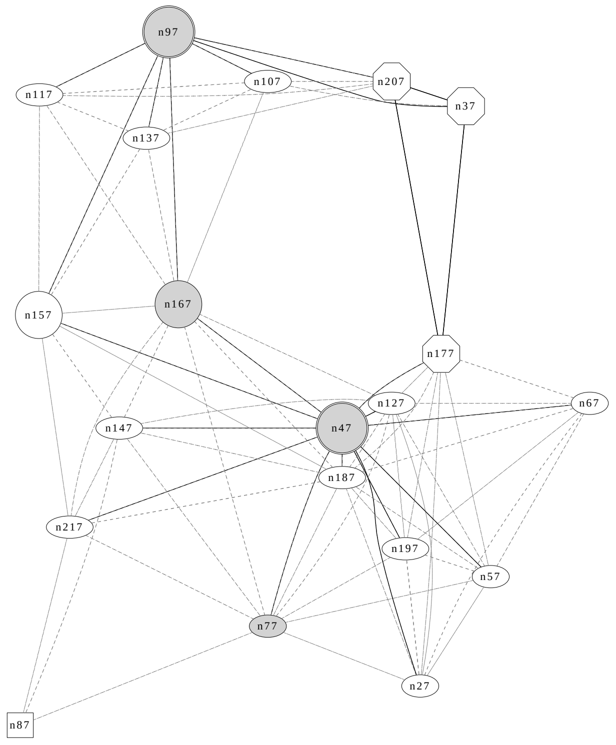


Figure A.32: *Increasing node density run 2.*

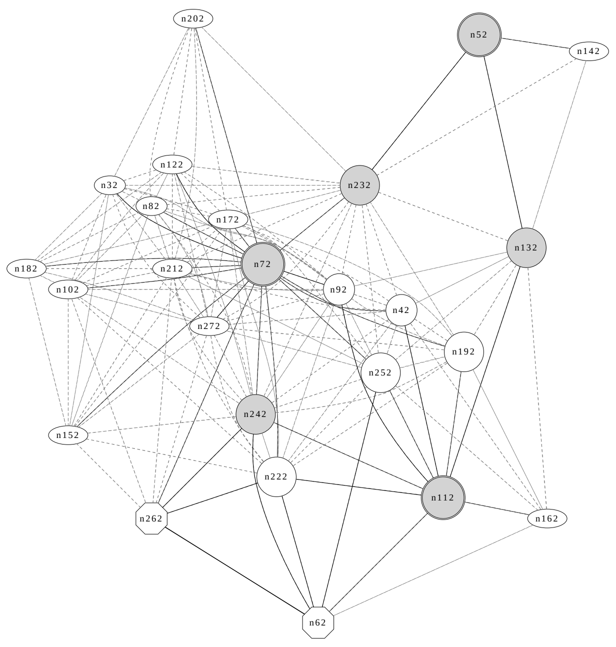


Figure A.33: *Increasing node density run 3.*

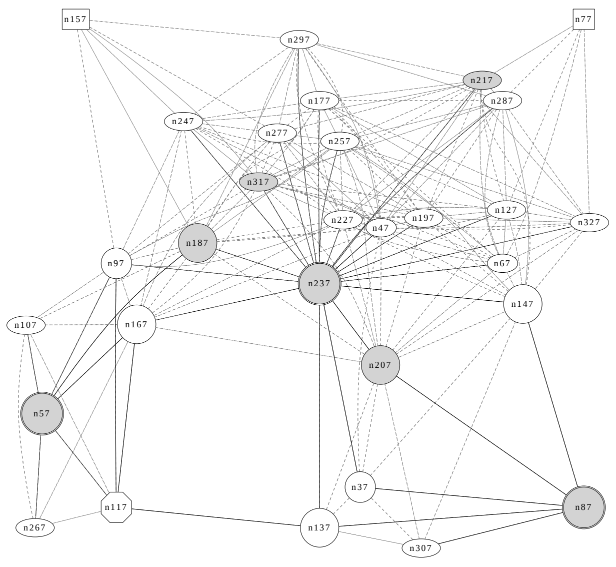


Figure A.34: *Increasing node density run 4.*

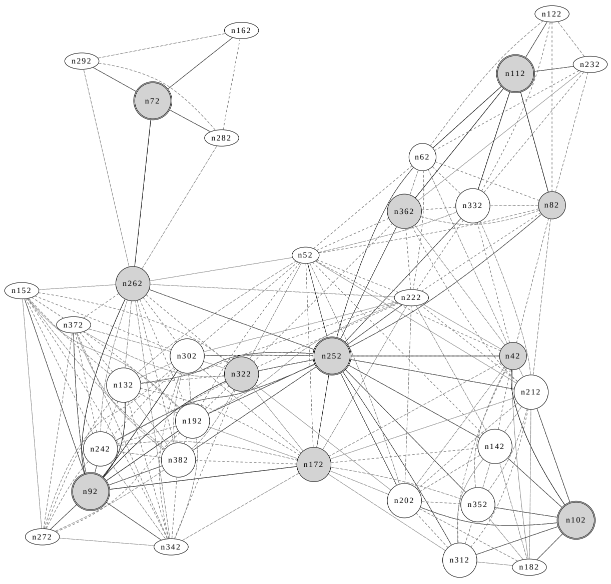


Figure A.35: *Increasing node density run 5.*

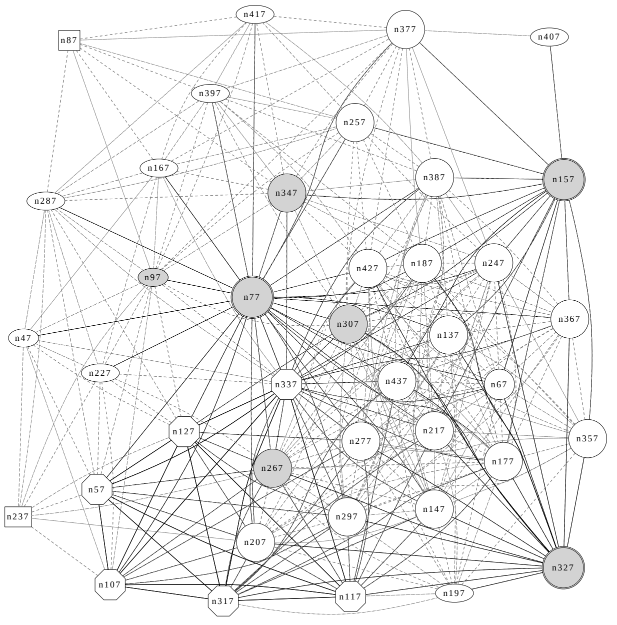


Figure A.36: *Increasing node density run 6.*

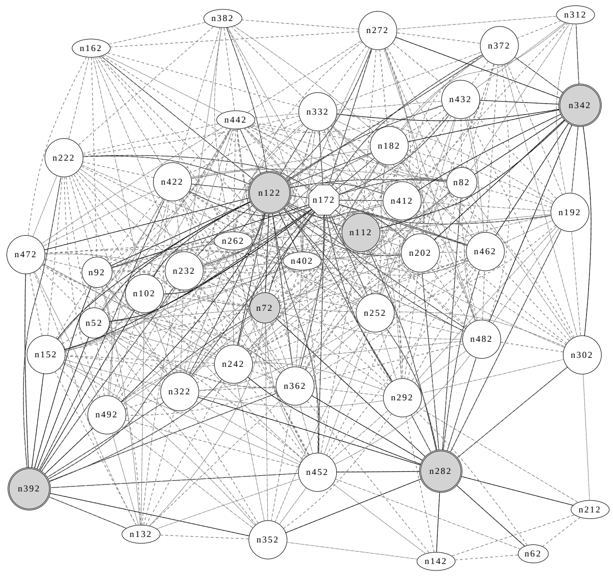


Figure A.37: *Increasing node density run 7.*

A.3 Effect of Token Prioritisation on Mean Packet Latency Networks

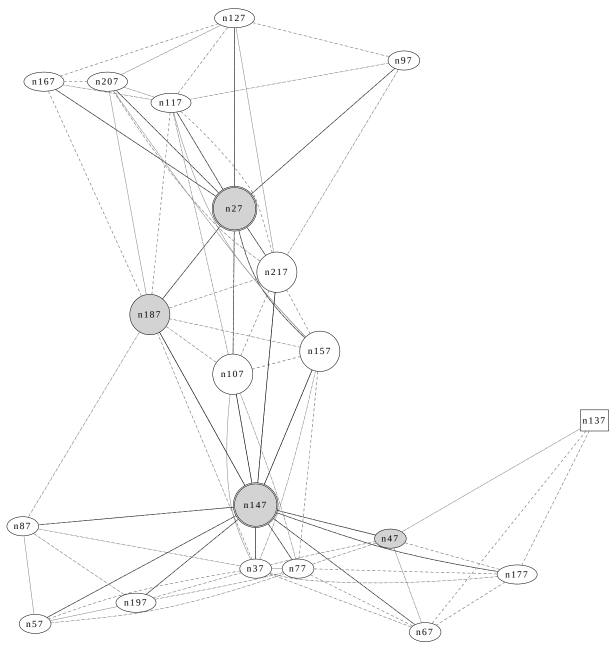


Figure A.38: *Latency network 20 nodes run 1.*

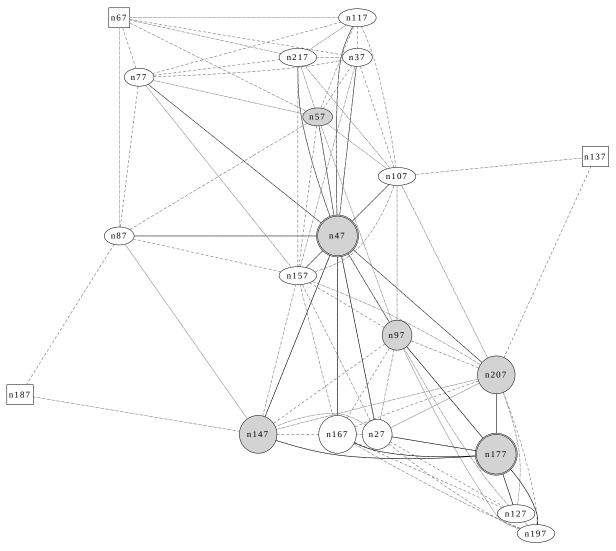


Figure A.39: *Latency network 20 nodes run 2.*

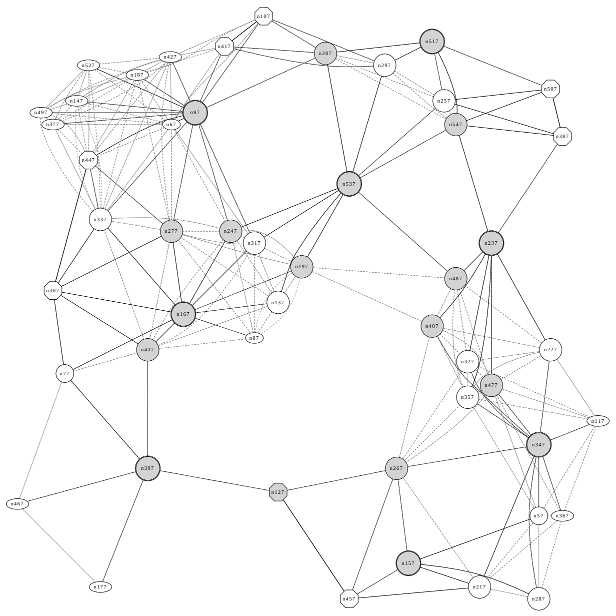


Figure A.40: *Latency network 50 nodes run 1.*

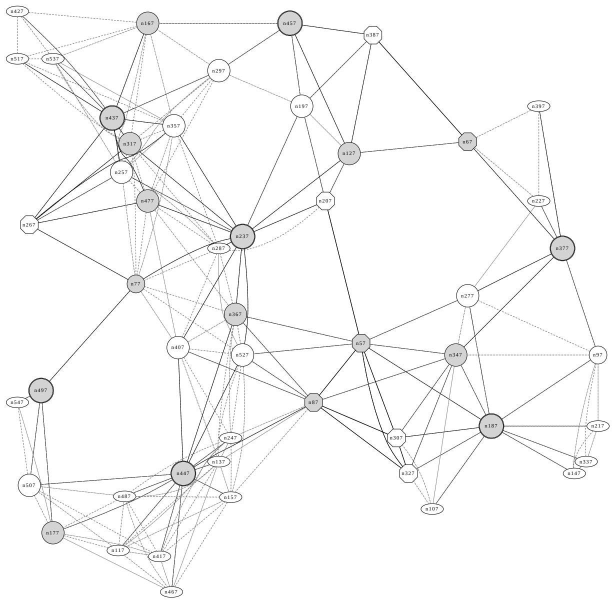


Figure A.41: *Latency network 50 nodes run 2.*

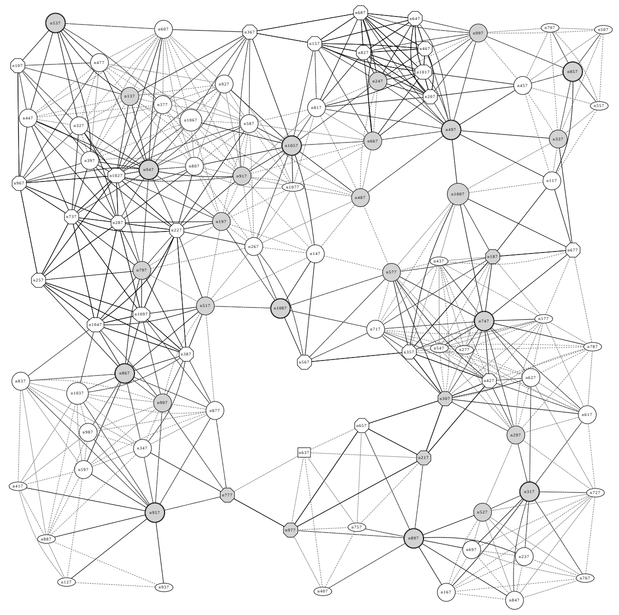


Figure A.42: *Latency network 100 nodes run 1.*

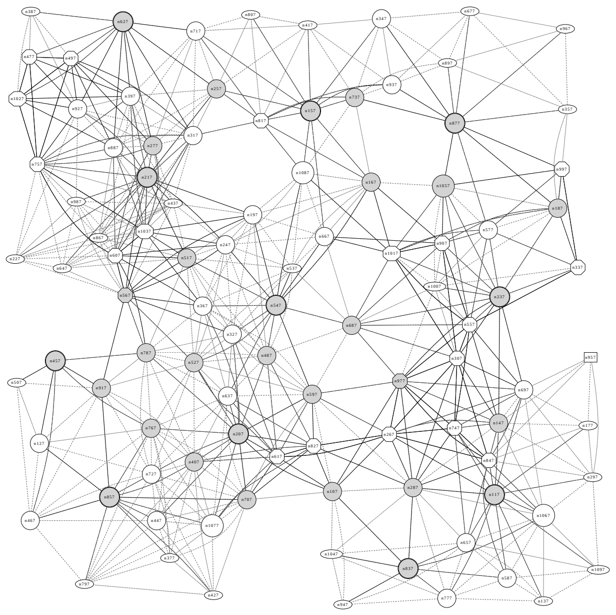


Figure A.43: *Latency network 100 nodes run 2.*

Appendix B

Programming Scripts

This appendix shows various scripts and configuration files, used by the simulator.

B.1 Run Simulation BASH Script

```
#!/bin/bash

date >> ./LogFiles/lastSimulationRun.log

runTokenSimulation()
{
rm ./tokenProgress.log
rm ./ConfigFiles/token_30_nodes_extra_buffer.sca
for alpha in 200000 400000 600000 800000 1000000 1200000 1400000 1600000 1800000 2000000 2200000 2400000 2600000 2800000 3
echo "description = \"Using token passing\"" > ./ConfigFiles/varParameters.ini
echo "chgrp.host[*].net.useTokenPassing = true" >> ./ConfigFiles/varParameters.ini
echo "chgrp.host[*].net.averageTotalNetworkBitRate = $alpha" >> ./ConfigFiles/varParameters.ini
echo "chgrp.host[*].nic.mac.useRtsCts = true" >> ./ConfigFiles/varParameters.ini
echo "output-scalar-file=\"./token_30_nodes_extra_buffer.sca\"" >> ./ConfigFiles/varParameters.ini
cat ./ConfigFiles/varParameters.ini
./CHGRP
echo $alpha >> tokenProgress.log
done
}

runNoTokenSimulation()
{
rm ./noTokenProgress.log
rm ./ConfigFiles/notoken_30_nodes_extra_buffer.sca
for alpha in 200000 400000 600000 800000 1000000 1200000 1400000 1600000 1800000 2000000 2200000 2400000 2600000 2800000 3
echo "description = \"Using the 802.11 without tokenpassing\"" > ./ConfigFiles/varParameters.ini
echo "chgrp.host[*].net.useTokenPassing = false" >> ./ConfigFiles/varParameters.ini
echo "chgrp.host[*].net.averageTotalNetworkBitRate = $alpha" >> ./ConfigFiles/varParameters.ini
echo "chgrp.host[*].nic.mac.useRtsCts = true" >> ./ConfigFiles/varParameters.ini
echo "output-scalar-file=\"./notoken_30_nodes_extra_buffer.sca\"" >> ./ConfigFiles/varParameters.ini
cat ./ConfigFiles/varParameters.ini
./CHGRP
echo $alpha >> noTokenProgress.log
done
}

cleanRouteStats()
{
if [ -f ./routeStats.txt ]
```

```
then
echo "routeStats.txt file exists: Cleaning file"
rm ./routeStats.txt
fi
}

runXMLSimulation()
{
local simulationName=$1

echo "Creating Simulation Output Directory"
outputDirectory="$simulationBase/$output/$(createSimulationDirectory $simulationName)"

echo "Running Simulation Scenario : $simulationName"
cleanRouteStats
xmlDir="$simulationBase/ConfigFiles/Scenarios/${simulationName}"
echo "Simulation Input Data Directory : $xmlDir"

cat $xmlDir/index.list
index="$(cat $xmlDir/index.list)"
#xmlDir="./Scenarios/${simulationName}"

    echo "Simulation Scenario $simulationName" >> $outputDirectory/simulationsCompleted.txt

for fileNumber in $index
do
echo "Running Simulation $fileNumber"
echo "description = \"Connection Oriented Simulation $fileNumber\"" > "$varParametersFile"
echo "chgrp.host[*].net.useTokenPassing = true" >> "$varParametersFile"
echo "chgrp.host[*].net.averageTotalNetworkBitRate = 0" >> "$varParametersFile"
echo "output-scalar-file=\"./scenario${simulationName}.sca\"" >> "$varParametersFile"
echo "chgrp.host[*].net.sendMessageRandomDestination = false" >> "$varParametersFile"
echo "chgrp.host[*].net.xmlParameters = xmldoc(\"${xmlDir}/file${fileNumber}XML.xml\")" >> "$varParametersFile"
echo "chgrp.performance.simulationOutputDirectory = \"$outputDirectory\"" >> "$varParametersFile"

#Create Output Directory for Run
mkdir "$outputDirectory/Run$fileNumber"
mkdir "$outputDirectory/Run$fileNumber/ModuleOutput"
echo "chgrp.performance.runOutputDirectory = \"$outputDirectory/Run$fileNumber/ModuleOutput\"" >> "$varParametersFile"

#Setup simulation specific parameters
echo "chgrp.host[*].appl.sendUdpStreams=false" >> "$varParametersFile"
echo "chgrp.host[*].appl.testLinkBreak=false" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbStartSendingDataMessagesTime=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbLinkBreakTime=10" >> "$varParametersFile"

echo "chgrp.host[*].appl.minimumStreamLength=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.maximumStreamLength=5" >> "$varParametersFile"

echo "chgrp.host[*].net.oneTransmissionChance=true" >> "$varParametersFile"
echo "chgrp.host[*].net.sendDataMessages = true" >> "$varParametersFile"

        #Setup performance measurements
        echo "chgrp.performance.simulationTimeConnectivity = false" >> "$varParametersFile"

        #Setup node positions
        if [ $simulationName == "RandomOne" -o $simulationName == "RandomTwo" ]
        then
            echo "include ./NodeSetups/nodeSetup_Random_1.ini" >> "$varParametersFile"
        else
            echo "include ./NodeSetups/nodeSetup6.ini" >> "$varParametersFile"
        fi
```

```
echo "chgrp.performance.runNumber = $fileNumber" >> "$varParametersFile"
./CHGRP
echo "Completed Simulation $fileNumber" >> $outputDirectory/simulationsCompleted.txt
done
}

writeNodeSetup()
{
local xSize=$1
local ySize=$2
local numberOfNodes=$3

local nodeSetupPath="$simulationBase/ConfigFiles/NodeSetups"

echo "chgrp.playgroundSizeX = $xSize" > $nodeSetupPath/nodeSetupVar.ini
echo "chgrp.playgroundSizeY = $ySize" >> $nodeSetupPath/nodeSetupVar.ini
echo "" >> $nodeSetupPath/nodeSetupVar.ini
echo "chgrp.numHosts = $numberOfNodes" >> $nodeSetupPath/nodeSetupVar.ini
echo "chgrp.host[*].numHosts = $numberOfNodes" >> $nodeSetupPath/nodeSetupVar.ini

echo "" >> $nodeSetupPath/nodeSetupVar.ini

echo "chgrp.host[*].mobility.x=-1" >> $nodeSetupPath/nodeSetupVar.ini
echo "chgrp.host[*].mobility.y=-1" >> $nodeSetupPath/nodeSetupVar.ini
}

generateCircleClusterNetwork()
{
    local clusterHeadNeighbours=$1
    local nodeSetupPath="$simulationBase/ConfigFiles/NodeSetups"
    local nodeSetupFile="$nodeSetupPath/nodeSetupVar.ini"
    local xSize=600
    local ySize=600

    echo "chgrp.playgroundSizeX = $xSize" > $nodeSetupFile
    echo "chgrp.playgroundSizeY = $ySize" >> $nodeSetupFile
    echo "" >> $nodeSetupFile
    echo "chgrp.numHosts = $(($clusterHeadNeighbours+1))" >> $nodeSetupFile
    echo "chgrp.host[*].numHosts = $(($clusterHeadNeighbours+1))" >> $nodeSetupFile
    echo "" >> $nodeSetupFile

    pi=$(echo "scale=15; 4*a(1)" | bc -l)
    radius=150
    xOffset=300
    yOffset=300

    echo "chgrp.host[0].mobility.x=$xOffset" >> $nodeSetupFile
    echo "chgrp.host[0].mobility.y=$yOffset" >> $nodeSetupFile

    for ((i=0; i<$clusterHeadNeighbours; i=$i+1 )) do
        xPos=$(echo "(s(($pi*2*$i)/$clusterHeadNeighbours)*$radius+$xOffset)" | bc -l | xargs printf "%1.0f")
        yPos=$(echo "(c(($pi*2*$i)/$clusterHeadNeighbours)*$radius+$yOffset)" | bc -l | xargs printf "%1.0f")
        echo "chgrp.host[$(($i+1))].mobility.x=$xPos" >> $nodeSetupFile
        echo "chgrp.host[$(($i+1))].mobility.y=$yPos" >> $nodeSetupFile
    done
}

singleRunSimulation()
{
local networkSize=$1
echo "Creating Simulation Output Directory"
```

```
outputDirectory="$simulationBase/$output/$(createSimulationDirectory "SingleRun")"

echo "Starting Single Run Simulation"

xmlDir="$simulationBase/ConfigFiles/SingleRun"
echo "Simulation Input Data Directory : $xmlDir"

echo "description = \"Single Simulation Run\"" > "$varParametersFile"

echo "chgrp.host[*].net.useTokenPassing = true" >> "$varParametersFile"
echo "chgrp.host[*].net.averageTotalNetworkBitRate = 0" >> "$varParametersFile"
echo "output-scalar-file=\"$outputDirectory/scenarioSingleRun.sca\"" >> "$varParametersFile"
echo "chgrp.host[*].net.sendMessageRandomDestination = true" >> "$varParametersFile"

echo "chgrp.host[*].net.xmlParameters = xmldoc(\"${xmlDir}/SingleRun.xml\")" >> "$varParametersFile"
echo "chgrp.performance.simulationOutputDirectory = \"$outputDirectory\"" >> "$varParametersFile"

#Create Output Directory for Run
mkdir "$outputDirectory/RunSingle"
mkdir "$outputDirectory/RunSingle/ModuleOutput"
echo "chgrp.performance.runOutputDirectory = \"$outputDirectory/RunSingle/ModuleOutput\"" >> "$varParametersFile"

    #Setup simulation type

    #Setup performance measurements
    echo "chgrp.performance.simulationTimeConnectivity = false" >> "$varParametersFile"

#Setup node positions
if [ $networkSize == "Twenty" ]
then
echo "include ./NodeSetups/nodeSetup3.ini" >> "$varParametersFile"
elif [ $networkSize == "Fifty" ]
then
echo "include ./NodeSetups/nodeSetup50nodes.ini" >> "$varParametersFile"
elif [ $networkSize == "Hundred" ]
then
echo "include ./NodeSetups/nodeSetup100nodes.ini" >> "$varParametersFile"
else
echo "include ./NodeSetups/nodeSetup3.ini" >> "$varParametersFile"
fi

echo "chgrp.performance.runNumber = 1" >> "$varParametersFile"
./CHGRP
echo "Completed Single Simulation" >> $outputDirectory/simulationsCompleted.txt
}

networkSetupSimulationRun()
{
local name
local networkSize="Twenty"
local runNumber=""
local testConnectivity="false"
local simulationName="networkSetupTest"
local priorityToken="false"
local currentRunSetup
local testLinkBreak="false"

OPTIND=1

while getopts s:r:cn:pb name $@
do
if [ $name == "s" ]
```

```
then
networkSize=$OPTARG
elif [ $name == "r" ]
then
runNumber="$runNumber$OPTARG "
elif [ $name == "c" ]
then
testConnectivity="true"
elif [ $name == "n" ]
then
simulationName=$OPTARG
elif [ $name == "p" ]
then
priorityToken="true"
elif [ $name == "b" ]
then
testLinkBreak="true"
else
echo "Invalid option for script function networkSetupSimulationRun!!"
fi
done

#Determine which simulation runs should be executed
if [ -z $runNumber ]
then
if [ $networkSize == "Increase" ]
then
runNumber="1 2 3 4 5 6 7"
elif [ $priorityToken == "true" ]
then
runNumber="1 2 3 4"
elif [ $testLinkBreak == "true" ]
then
runNumber="1 2"
else
runNumber="1 2 3 4 5 6 7 8 9 10"
fi
fi

echo "Network size : $networkSize"
echo "Run number : $runNumber"
echo "Test connectivity : $testConnectivity"
echo "Priority Token Test : $priorityToken"
echo "Test Link Break : $testLinkBreak"

echo "Creating Simulation Output Directory"
outputDirectory="$simulationBase/$output/$(createSimulationDirectory "$simulationName")"

echo "Starting Network Setup Test Simulation"

#Setup xml configuration file path
xmlDir="$simulationBase/ConfigFiles/SingleRun"
echo "Simulation Input Data Directory : $xmlDir"

for simulationRun in $runNumber
do
echo "description = \"Network Setup Simulation Run $simulationRun\"" > "$varParametersFile"

echo "chgrp.host[*].net.useTokenPassing = true" >> "$varParametersFile"
echo "chgrp.host[*].net.averageTotalNetworkBitRate = 100" >> "$varParametersFile"
echo "output-scalar-file=\"$outputDirectory/scenarioSingleRun.sca\"" >> "$varParametersFile"
echo "chgrp.host[*].net.sendMessageRandomDestination = false" >> "$varParametersFile"
echo "chgrp.host[*].net.xmlParameters = xmldoc(\"${xmlDir}/SingleRun.xml\")" >> "$varParametersFile"
echo "chgrp.performance.simulationOutputDirectory = \"$outputDirectory\"" >> "$varParametersFile"

#Create Output Directory for Run
mkdir "$outputDirectory/Run$simulationRun"
mkdir "$outputDirectory/Run$simulationRun/ModuleOutput"
echo "chgrp.performance.runOutputDirectory = \"$outputDirectory/Run$simulationRun/ModuleOutput\"" >> "$varParametersFile"

        #Setup simulation type

        #Setup performance measurements
        echo "chgrp.performance.simulationTimeConnectivity = $testConnectivity" >> "$varParametersFile"

        #Setup node positions
if [ $networkSize == "Twenty" ]
then
         echo "include ./NodeSetups/nodeSetup3.ini" >> "$varParametersFile"
elif [ $networkSize == "Fifty" ]
then
echo "include ./NodeSetups/nodeSetup50nodes.ini" >> "$varParametersFile"
elif [ $networkSize == "Hundred" ]
then
echo "include ./NodeSetups/nodeSetup100nodes.ini" >> "$varParametersFile"
        elif [ $networkSize == "Test1" ]
        then
            echo "include ./NodeSetups/nodeSetup1.ini" >> "$varParametersFile"
        elif [ $networkSize == "Circle" ]
        then
            generateCircleClusterNetwork $simulationRun
            echo "include ./NodeSetups/nodeSetupVar.ini" >> "$varParametersFile"
elif [ $networkSize == "Increase" ]
then
writeNodeSetup 600 600 $((10 + (5*$simulationRun)))
echo "include ./NodeSetups/nodeSetupVar.ini" >> "$varParametersFile"
else
echo "include ./NodeSetups/nodeSetup3.ini" >> "$varParametersFile"
fi

#Perform Simulation Specific Test Setup
if [ $priorityToken == "true" ]
then
#Setup token priority
currentRunSetup=$((($simulationRun+1)/2))
if [ $(($simulationRun%2)) == 0 ]
then
echo "Using priority token : false"
oneTransmissionChance="true"
else
echo "Using priority token : true"
oneTransmissionChance="false"
fi

echo "chgrp.host[*].appl.sendUdpStreams=true" >> "$varParametersFile"
echo "chgrp.host[*].appl.testLinkBreak=false" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbStartSendingDataMessagesTime=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbLinkBreakTime=10" >> "$varParametersFile"

echo "chgrp.host[*].appl.minimumStreamLength=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.maximumStreamLength=5" >> "$varParametersFile"

echo "chgrp.host[*].net.oneTransmissionChance=$oneTransmissionChance" >> "$varParametersFile"
echo "chgrp.host[*].net.sendDataMessages = false" >> "$varParametersFile"
elif [ $testLinkBreak == "true" ]
then
currentRunSetup=$simulationRun
echo "chgrp.host[*].appl.sendUdpStreams=true" >> "$varParametersFile"
echo "chgrp.host[*].appl.testLinkBreak=true" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbStartSendingDataMessagesTime=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbLinkBreakTime=5" >> "$varParametersFile"

echo "chgrp.host[*].appl.minimumStreamLength=600" >> "$varParametersFile"
echo "chgrp.host[*].appl.maximumStreamLength=700" >> "$varParametersFile"

echo "chgrp.host[*].net.oneTransmissionChance=false" >> "$varParametersFile"
echo "chgrp.host[*].net.sendDataMessages = false" >> "$varParametersFile"
        elif [ $networkSize == "Circle" ]
        then
            currentRunSetup=$simulationRun
            echo "chgrp.host[*].appl.sendUdpStreams=false" >> "$varParametersFile"
            echo "chgrp.host[*].appl.testLinkBreak=false" >> "$varParametersFile"
            echo "chgrp.host[*].appl.lbStartSendingDataMessagesTime=3" >> "$varParametersFile"
            echo "chgrp.host[*].appl.lbLinkBreakTime=10" >> "$varParametersFile"

            echo "chgrp.host[*].appl.minimumStreamLength=3" >> "$varParametersFile"
            echo "chgrp.host[*].appl.maximumStreamLength=5" >> "$varParametersFile"

            echo "chgrp.host[*].net.oneTransmissionChance=true" >> "$varParametersFile"
            echo "chgrp.host[*].net.sendDataMessages = false" >> "$varParametersFile"
else
currentRunSetup=$simulationRun
echo "chgrp.host[*].appl.sendUdpStreams=false" >> "$varParametersFile"
echo "chgrp.host[*].appl.testLinkBreak=false" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbStartSendingDataMessagesTime=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.lbLinkBreakTime=10" >> "$varParametersFile"

echo "chgrp.host[*].appl.minimumStreamLength=3" >> "$varParametersFile"
echo "chgrp.host[*].appl.maximumStreamLength=5" >> "$varParametersFile"

echo "chgrp.host[*].net.oneTransmissionChance=false" >> "$varParametersFile"
echo "chgrp.host[*].net.sendDataMessages = true" >> "$varParametersFile"
fi

echo "Run Command : ./CHGRP -r $currentRunSetup"

echo "chgrp.performance.runNumber = $simulationRun" >> "$varParametersFile"
./CHGRP -r "$currentRunSetup"
success=$?
if [ $success -eq 0 ]
then
result="(FAILED)"
else
result="(SUCCESS)"
fi

echo "Completed Network Setup Test Simulation $simulationRun $result" >> $outputDirectory/simulationsCompleted.txt
done
}

createSimulationDirectory()
{
local simulationName=$1
local cOutputDirectory="Simulation.$simulationName.$(date +%d_%b_%y_%H%M_%S)"
mkdir "$simulationBase/$output/$cOutputDirectory"
```

```
echo "$cOutputDirectory"
}

simulationBase="$WORKDIR/CHGRP"
output="SimulationOutput"

echo ""
varParametersFile="./ConfigFiles/varParameters.ini"
firstVarParametersFile="./ConfigFiles/firstVarParameters.ini"
simulationTimeLimit="200"

echo "[General]" > $firstVarParametersFile
echo "sim-time-limit = ${simulationTimeLimit}s" >> $firstVarParametersFile

#XML Simulations

# runXMLSimulation "One" #Change networklayer values back to default values.
#runXMLSimulation "Two"
#runXMLSimulation "Three"
# runXMLSimulation "Four"
#runXMLSimulation "Five"

# runXMLSimulation "RandomOne"
# runXMLSimulation "RandomTwo"

#Routing Overhead Simulation
#networkSetupSimulationRun -s Increase -n "ControlOverhead"

#Priority Token Simulations
#networkSetupSimulationRun -s Twenty -n "PriorityTokenTwenty" -p
#networkSetupSimulationRun -s Fifty -n "PriorityTokenFifty" -p
#networkSetupSimulationRun -s Hundred -n "PriorityTokenHundred" -p

#Quick Reroute Simulations
#networkSetupSimulationRun -s Twenty -n "QuickRerouteTwenty" -b
#networkSetupSimulationRun -s Fifty -n "QuickRerouteFifty" -b
#networkSetupSimulationRun -s Hundred -n "QuickRerouteHundred" -b

#Connectivity Simulations
#networkSetupSimulationRun -s Twenty -r 1 -r 2 -n "ConnectivityTwenty" -c
#networkSetupSimulationRun -s Fifty -r 1 -r 2 -n "ConnectivityFifty" -c
#networkSetupSimulationRun -s Hundred -r 1 -r 3 -n "ConnectivityHundred" -c

#Test Networks
#networkSetupSimulationRun -s Test1 -r 1 -n "Test1Network"

#singleRunSimulation Twenty

#networkSetupSimulationRun -s Circle -n "CircleNetworks"

#Run Simulation to Generate Random Network

#networkSetupSimulationRun -s Twenty -r 11 -n "ARandomNetwork"

echo ""
```

B.2 Configuration Files

B.2.1 General Configuration File

```
[General]
;ini-warnings = true
rng-class="cMersenneTwister"
network = chgrp
random-seed = 1
sim-time-limit = 20s

[Tkenv]
bitmap-path="/home/vier/omnetpp-3.4b2/mobility-fw2.0p3/bitmaps"
default-run=1
runs-to-execute = 1
use-mainwindow = yes
print-banners = yes
slowexec-delay = 300ms
update-freq-fast = 10
update-freq-express = 100
breakpoints-enabled = yes


[Cmdenv]
runs-to-execute = 1
event-banners = yes
module-messages = yes
express-mode = yes

[DisplayStrings]


[Parameters]

chgrp.modifiedNicPath = "./ModifiedNic/"
chgrp.useModifiedProtocolStack = false

##############################################################################
#       Parameters for the entire simulation                                 #
##############################################################################

#Using number in nodeSetup*.ini file
#chgrp.numHosts = 20
#chgrp.host[*].numHosts = 20

# uncomment to enable debug messages for all modules
#**.debug = 1
**.coreDebug = 0

##############################################################################
#       Parameters for the Mobility Module and PlaygroundSize                #
##############################################################################

include ./ConfigFiles/mobilitySetup.ini


##############################################################################
#       Parameters for the ChannelControl                                    #
##############################################################################
chgrp.channelcontrol.carrierFrequency = 2.4e+9
```

```
# max transmission power [mW]
chgrp.channelcontrol.coreDebug = 1

#Transmission Power Gt*Gr*Pt = 2dBi + 2dBi + 15dBm = 79,4 mWatt
#Scaling Factor = 6,69
chgrp.channelcontrol.pMax  = 159208;3200000 ; Pt = Pto*Scaling factor; was 110.11

# signal attenuation threshold [dBm]
# SAT -85dBm for 802.11b
chgrp.channelcontrol.sat   = -85;-72 ; was -120
# path loss coefficient alpha
chgrp.channelcontrol.alpha = 4 ; was 4
chgrp.channelcontrol.sendDirect = 0
chgrp.channelcontrol.useTorus = 0

##############################################################################
#       Parameters for the Host                                              #
##############################################################################
chgrp.host[*].color = "white"
chgrp.host[*].appendDisplay = "b=20,20,oval;o=blue,black,2"
chgrp.host[*].applLayer="TestApplLayer"

##############################################################################
#       Parameters for the Application Layer                                 #
##############################################################################
include ./ConfigFiles/applParameters.ini

##############################################################################
#       Parameters for the Network Layer                                 #
##############################################################################

include ./ConfigFiles/netParameters.ini

##############################################################################
#       Parameters for ARP
##############################################################################
chgrp.host[*].arp.debug = 0

include ./ConfigFiles/macParameters.ini

##############################################################################
#       Parameters for Performance Module                                    #
##############################################################################
include ./ConfigFiles/performanceModuleParameters.ini

##############################################################################
#       Parameters for Different Runs                                        #
##############################################################################
include ./ConfigFiles/runParameters.ini
```

B.2.2 Network Layer Configuration File

```
##############################################################################
#       Parameters for the Network Layer                                     #
##############################################################################

[Parameters]

chgrp.host[*].net.headerLength=64 #in bits
chgrp.host[*].net.debug = 1
chgrp.host[*].net.Election_Threshold = 2
chgrp.host[*].net.goodConnectionThreshold = 0.75

#chgrp.host[*].net.useTokenPassing = false

#Link detection parameters
chgrp.host[*].net.generateNewHelloMessageShort_TIME = 0.04
chgrp.host[*].net.generateNewHelloMessageLong_TIME = 0.1
chgrp.host[*].net.decreaseTimeToLive_TIME = 0.1

#Routing update paramters
chgrp.host[*].net.clusterTableRoutingUpdateTime = 0.5
chgrp.host[*].net.decreaseClusterTimeToLive_TIME = 0.5
chgrp.host[*].net.clusterTableTimeToLiveMaximum = 10

#Insert Damping Fluctuations Parameters

#Token parameters
chgrp.host[*].net.tokenTimeOut_TIME = 0.0075
chgrp.host[*].net.generateNewTokenMessage_TIME = 0.0075
chgrp.host[*].net.generateNewTokenMessageAfterTimeOut_TIME = 0

chgrp.host[*].net.averagePacketInterarrival_TIME = 0.05

chgrp.host[*].net.displayNetworkStatsGraphicDebug = false
chgrp.host[*].net.bufferDebug = false
chgrp.host[*].net.showNetworkAddress = false

#Routing Layer Parameters
#One transmission chance added to variable configuration file
#chgrp.host[*].net.oneTransmissionChance = false
#Send data messages moved to variable configuration file
#chgrp.host[*].net.sendDataMessages = false

#Output File Switches
chgrp.host[*].net.printNeighbourTimeOutFile = false
chgrp.host[*].net.writeRoutingTableFile = true
chgrp.host[*].net.writeNetworkLayerInformationFile = true
chgrp.host[*].net.writeNodeDataFile = true

#Debug Parameters
chgrp.host[*].net.tokenDebug = false
```

B.2.3 Application (Transport) Layer Configuration File

```
##############################################################################
#       Parameters for the Application Layer                                 #
##############################################################################

[Parameters]
# debug switch
chgrp.host[*].appl.debug = 1
chgrp.host[*].appl.headerLength=1024
chgrp.host[*].appl.burstSize=3

#Send Udp Streams added to variable configuration file
#chgrp.host[*].appl.sendUdpStreams=true

chgrp.host[*].appl.offsetLengthTime=0.8
chgrp.host[*].appl.minimumOffsetLength=2
chgrp.host[*].appl.maximumOffsetLength=4
chgrp.host[*].appl.sendingRate=250000 # in bits per second

#minimumStreamLength and maximumStreamLength moved to simulation setup script
#chgrp.host[*].appl.minimumStreamLength=600
#chgrp.host[*].appl.maximumStreamLength=700

#lbStartSendingDataMessagesTime and lbLinkBreakTime moved to simulation setup script
#chgrp.host[*].appl.lbStartSendingDataMessagesTime=3
#chgrp.host[*].appl.lbLinkBreakTime=5

#Test Parameters
#testLinkBreak moved to simulation setup script
#chgrp.host[*].appl.testLinkBreak = true

#XML Parameters file
chgrp.host[*].appl.xmlParameters = xmldoc("/home/daniel/Personal/Thesis/CHGRP/ConfigFiles/QuickReroute/quickReroute.xml")
```

Appendix C

Published Article

Design and Optimization of a Cluster Based Ad Hoc Wireless Network Routing Protocol

Daniël Kotze and Riaan Wolhuter
Department of Electronic Engineering
University of Stellenbosch
7602 Matieland, South Africa
Email: 14380552@sun.ac.za, wolhuter@sun.ac.za

***Abstract*—Cluster based routing protocols has been shown to reduce routing information overhead at high node densities. This paper documents the design process of a cluster based routing protocol. A simulation was created to implement the protocol. It is shown how the cost to maintain the cluster structure, can be reduced further by decreasing the number of nodes that transmit routing information. The formation of clusters is illustrated and the roles of different nodes are described, due to their place in the network topology. A mechanism for the fast detection of broken routes is presented. Basic congestion control is implemented with token scheduling. Performance is measured by a separate module for easier aggregate simulation time measurements.**

Index Terms—Ad hoc wireless network, cluster generation, congestion control, fast link break detection, simulation.

I. Introduction

Ad hoc wireless networks are communication networks without an inherent infrastructure. Nodes make use of neighbours to route packets to remote destinations that are multiple hops away. Using neighbours as routers extends the coverage of the network at a low cost. Ad hoc networks are ideal for deployment in rural areas, disaster recovery operations, environmental sensing and military operations [1]. This study focuses on ad hoc wireless networks where nodes have a high available bandwidth and high density, such as would be found operating in the K_a band. The Federal Communications Commission (FCC) has assigned the 57-64 GHz band for unlicensed use [2].

Cluster based hierarchical routing protocols are effective at high densities, because routing overhead can be reduced [3]. However maintaining the cluster structure comes at a cost. It will be shown how to reduce the cluster maintenance cost by using less overhead than previous implementations of the Cluster-head Gateway Switch Routing protocol (CGSR) [4], [5].

When network traffic increases, links tend to become saturated degrading the overall network performance. A simple mechanism is used to control congestion. Another drawback of current routing protocols is the slow detection of link breakages, because hello packets are sent periodically at the order of 2 seconds [6]. A link breakage can be detected sooner by sending smaller hello packets at a faster rate.

Background on ad hoc network routing protocols will be discussed first, in Section II. Next, the design steps and implementation environment will be documented in Section III. Section IV will present various tests and results.

II. Background

Routing protocols for wireless networks have different requirements, because nodes normally have limited bandwidth and mobile nodes frequently change the topology [1]. However, for this work, we studied an ad hoc network with high bandwidth and immobile hosts. Routing protocols follow two paradigms. Proactive routing protocols maintain routes to all destinations, while reactive protocols only set up a route when one is needed.

Destination Sequenced Distance Vector (DSDV) is a distance vector routing protocol based on the Bellman-Ford algorithm. With the use of sequence numbers loop-free paths to all destinations are provided [7]. Routing overhead of the DSDV protocol grows by $O(N^2)$ where N is the number of nodes [8]. Therefore, it becomes unsuitable for large networks. Optimised Link State Routing (OLSR) is a proactive link state protocol that provides an effective flooding mechanism with the use of Multi-Point Relays (MPRs) [8].

Dynamic Source Routing (DSR) and Ad hoc On-demand Distance Vector (AODV) are on demand routing protocols. Both rely on flooding Route Request packets for determining a route in the network. A Route Reply packet is generated by the node that provides a route to the destination [8], [9].

Hierarchical routing, groups sets of neighbouring nodes into a cell. A cluster-head is elected that coordinates the cluster. Hierarchical protocols allow for greater scalability. With a high node density, less control overhead is used and routes converge faster. [3], [10], [11]

Cluster-head Gateway Switch Routing (CGSR) is a hierarchical proactive routing protocol. Normal nodes only have to maintain the route to their cluster-head and only the cluster-head transmits routing information. One drawback of the protocol however, is that every node needs to transmit its cluster member table periodically, increasing the overhead. The cluster based protocol favours environments where nodes are not highly mobile. [4], [5], [8]

A. Clustering Algorithms

Mario Gerla et al. presented two methods of how clusters can be formed, namely the lowest-id and highest-connectivity clustering algorithms [5]. According to the lowest-id algorithm, the node in the neighbourhood with the lowest address becomes the cluster-head. With the highest connectivity algorithm the node that is the most highly connected in its neighbourhood, becomes the cluster-head.

organize themselves into a cluster by electing a cluster-head. After the clusters have been formed, routing information can start to be spread through the cluster. The DSDV protocol will be used as a basis, but only cluster-heads and other selected nodes will forward routing information. Lastly, when the routing information is known, traffic can be generated on the network and analysed.

D. Cluster Generation

The highest-connectivity clustering algorithm was chosen for creating clusters. If the situation arises where two cluster-heads are elected simultaneously within transmission range of one another, the cluster-head with the least connections will give up its cluster-head status. Otherwise, if the number of connections are equal, the cluster-head with the highest address will give up its cluster-head status. [4]

The cluster-head election process will now be described. When the network is started every node has the status of *unassigned*. In the unassigned state, the node continually broadcasts hello messages at a high rate. The hello message contains various data fields. Each hello message has a sequence number incremented with each transmission. Lost hello messages can be determined by inspecting the sequence number. The number of neighbours a node has, is included in the hello message. Each message contains an *election address* field. Every time a node receives a message and the election address is equal to its own address, it increments a counter. If the counter value exceeds a certain threshold, the node is elected as a cluster-head. The threshold can be calculated with the following equation:

$$election\ threshold = k_{threshold} \times connections \qquad (1)$$

The threshold prevents wrong nodes being elected as cluster-heads too quickly.

The node state field in a node's hello message is updated accordingly, when a node becomes a cluster-head. All nodes hearing the newly elected cluster-head change their state to *assigned*, to indicate that they are part of the cluster. Figure 3 shows how a cluster is created.

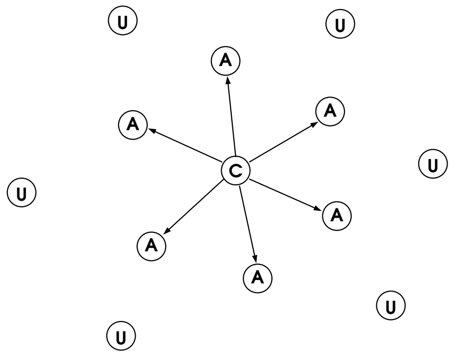


Fig. 3. The diagram illustrates cluster formation. "C" is the *cluster-head* node, "A's" are the *assigned* nodes and "U's" are the *unassigned* nodes.

Nodes that are part of a cluster now generate hello messages at a slower rate to maintain connections with their neighbours.

E. Node States

The protocol makes use of various states. In each state, a node has a different function or behaviour. When a node goes online it starts in the *unassigned* state. In the unassigned state, nodes try to elect a cluster-head. When a node can hear one cluster-head it goes into the *assigned* state. A node that can hear two cluster-heads becomes a *gateway*. Gateways provide a communication path from one cluster to another. A *distributed gateway* is formed between nodes that are in different clusters. *Satellite nodes* are nodes that do not have a cluster-head as a neighbour, but is a neighbour of nodes that can hear a cluster-head.

1) Motivation for Satellite Nodes: Rarely, instances occur where after the election process, a single uncovered node remains at the edge of the network. An uncovered node is a node not associated with a cluster. Only uncovered nodes can take part in cluster-head election and a single uncovered node cannot elect itself. A *satellite node* joins a cluster by relaying its information to the cluster-head through a common neighbour.

F. Routing Tables

The routing tables enable nodes to determine the next hop to a destination. The main routing tables are the neighbour table and the cluster table set. The cluster table set contains all nodes in the network's cluster-node associations and routing information to the specific cluster. Hello messages update the neighbour table and cluster table messages update the cluster table set.

Other tables assist the protocol by maintaining specialised information. The cluster-head table maintains information on cluster-heads within the two hop vicinity. The gateway table summarizes information on neighbour gateway or distributed gateway nodes and to which cluster they link. Routing information on satellite nodes within two hops is recorded in the satellite node table. Only cluster-heads maintain the neighbour cluster table that selects a designated gateway for each neighbouring cluster. The network node table contain all routable nodes and to which clusters they belong.

1) Cluster Table Set Updates: The cluster table set consists of cluster tables. Each cluster table represents a different cluster in the network. Similar to DSDV [7], when a cluster table message (routing update) is received, a cluster table inspects its sequence number to determine if it must be updated. A cluster table in the set with sequence number s_L will be updated if the sequence number in the cluster table in the update message s_R is greater. If however $s_L = s_R$, the cluster table in the set is only updated if the the cluster table in the update message has a smaller hop count.

G. Link Stability

Entries in the neighbour table and the cluster table set both have a TTL (Time-To-Live) field. When a node receives a hello message from a neighbour, a TTL field associated with the neighbour is set to its maximum 5. As long as the TTL is more than 0 the link state is *stable*. After a time nearly equal to the generation time of a hello message, the TTL field of every neighbour is decremented. The generation time of a hello message is 0,1 s. Therefore, if no hello message is received from a neighbour, the TTL field will reach 0 after 0,5 s. The link state is set to *unstable* if the TTL becomes 0.

Note that the hello message is kept small by only including information on critical nodes (maximum 4 of each type) within a two hop range. Critical node types include cluster-heads and satellite nodes.

H. Routing Information Distribution

Only selected nodes generate full routing information. Cluster-heads and forwarder nodes transmit cluster table messages. Forwarder nodes are selected by the cluster-head and are included in the header of the cluster table message. When a node is informed by a cluster table message that it is a forwarder node, it acquires a forwarding responsibility. Designated gateways indicated in the neighbour cluster table and the satellite service node (node between a cluster-head and satellite node) are selected as forwarder nodes. When a forwarder node receives a cluster table message and the header indicates that it is no longer a forwarder node, the node is relieved of its forwarding responsibility. Figure 4 shows how routing information is distributed.

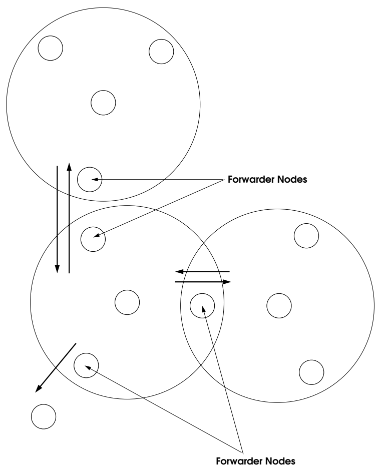


Fig. 4. The routing information is distributed to neighbouring clusters and *satellite nodes* through the forwarding nodes.

I. Routing Function

The routing function has two main parts. One part determines if the destination is available locally. The other part determines the next hop if the destination node is part of a remote cluster.

The neighbour table and the satellite node table are inspected first to find the destination's routing information. If the destination is found in the neighbour table, the destination is set as the next hop address. If the satellite node table contains the destination, an intermediate node is set as the next hop.

Now, if the local search has failed to deliver a result, the network node table is searched for the destination. If the node is found in the network node table, the cluster table set is utilised to determine the nearest cluster to which the destination belongs. The cluster table containing the destination node, contains a *reachable through cluster* field. The gateway table is searched for a gateway that links to the *reachable through cluster*. If such a gateway can be found, it is selected as the next hop, otherwise the cluster-head of the *current cluster* is selected as the next hop. Figure 5 explains the terms *reachable through cluster* and *current cluster*.

If the current node is a satellite node, the satellite service node is selected as the next hop.

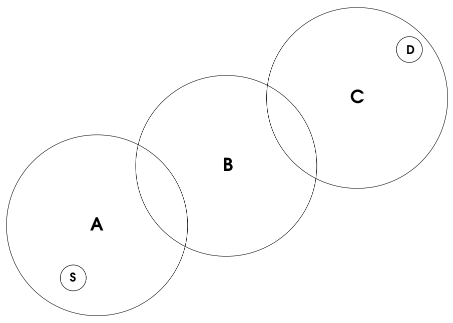


Fig. 5. Suppose a node "S" wants to route data to node "D". From the perspective of node "S", cluster "A" is the **current cluster**, cluster "B" is the **reachable through cluster** and cluster "C" is the **destination cluster**.

J. Token Scheduling

Token scheduling [4] was implemented as a measure of congestion control. Since cluster-heads transmit more routing information and nodes choose the cluster-head as a next hop if no shorter route can be found, cluster-heads are given more transmission chances. Note that different spreading codes for separate clusters were not used, as in the original CGSR, to simplify implementation of the protocol. The focus of token scheduling is to give a fair chance to each node to transmit data in the cluster. The token is generated by the cluster-head and passed back and forth between itself and its neighbours. Every time a token is received, the node can transmit one data message. The token can be regenerated after a time-out. The time-out is calculated by the following equation:

$$t_{timeout} = 2 \times t_{token\ transmission} + t_{data\ transmission} \quad (2)$$

If nodes have more queued traffic, they can be given more transmission chances in a transmission cycle. The number of transmission chances is equated with the following equation:

$$\textit{transmission chances} = \frac{m_i}{\max(1, \min(m_1, m_2, ..., m_n))} \quad (3)$$

Where m_i is the number of messages queued at a node and $m_1, m_2, ..., m_n$ are the messages queued at each cluster member.

Tokens are passed implicitly to satellite nodes.

K. Weighted Fair Queueing with Message Buffers

Weighted fair queueing [15] was implemented to prevent individual nodes from becoming congested. Two buffers are used to queue messages, namely the personal message buffer and general message buffer. The personal message buffer stores all messages originating from the node itself. All forwarded messages enter the general message buffer directly. When a token message is received, one data message from the general message buffer is sent. Messages proceed from the personal message buffer to the general buffer to maintain an acceptable ratio of forwarded messages to generated messages in the general message buffer. For each cluster a node which is part of the cluster has a personal and general message buffer set. Different message buffer sets allows flows to different clusters to be separated. Routing packets receive preference (hello message and cluster table message) and are passed directly to the MAC-Layer for transmission.

L. *Performance Monitoring*

It was decided to create a separate module in the simulation for performance monitoring. This approach keeps the performance monitoring and simulation code separate for greater clarity. Simulation nodes have write access to the performance module to record certain values. At the start of the simulation, each node registers with the performance module by giving access to important data structures, for instance, the routing table. A *graphviz* (drawing tool for graphs) source file is generated by the performance module and compiled with the *neato* program [16] to generate a graphic of the network. The configuration is shown in Figure 6.

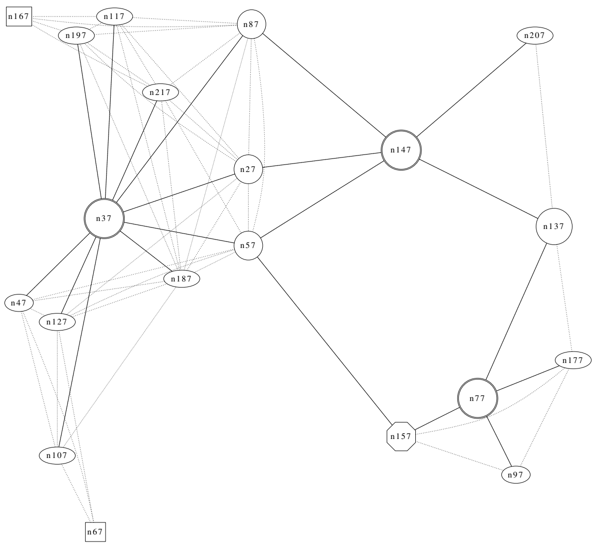


Fig. 6. Big circles represent cluster-heads, gateways are circles, ellipses indicate assigned nodes, distributed gateways are octagons and satellite nodes are rectangles.

IV. Tests and Results

Network setup tests were performed on various different randomised networks. An aggregate connectivity test was performed on the network shown in Figure 6. A test to show the scalability of the protocol at high node densities was conducted and the results are shown in Figures 8 and 9. Note that the hello message generation time is $0{,}1\,\mathrm{s}$ and the cluster table message generation time is $0{,}5\,\mathrm{s}$. Each simulation used 20 nodes with a transmission range of approximately $40\,\mathrm{m}$ bounded by an area of $90\,\mathrm{m}$ by $90\,\mathrm{m}$.

The network setup tests show that the cluster generation algorithm is successful. The results of 9 simulated random networks are given in Table I. For each network, the number of clusters, the average cluster size, number of forwarder nodes, the percentage of nodes that transmit full routing information and the last cluster-head setup time were determined. The number of clusters range from 2 to 4. By only letting cluster-heads and forwarder nodes transmit full routing information, the number of nodes generating full control overhead can be reduced at least to $40\,\%$. Cluster-heads are all elected within $1\,\mathrm{s}$.

TABLE I
Network setup test results.

Simulation Number	Number of Clusters	Average Cluster Size	Forwarder Nodes	Percentage Full Routing	Last Cluster Election Time (s)
1	4	8	7	0,35	0,624
2	2	11	4	0,2	0,403
3	3	10	6	0,3	0,537
4	3	8,33	5	0,25	0,402
5	4	7	6	0,3	0,446
6	2	12,5	4	0,2	0,575
7	2	12	6	0,3	0,406
8	3	8	6	0,3	0,535
9	3	7,33	8	0,4	0,334

The aggregate network connectivity shows the percentage of nodes that are routable for each node in the network. Thus an aggregate network connectivity of 1 indicates that every node knows how to reach every other node in the network. Figure 7 provides the aggregate network connectivity at specific points in time. At approximately $1{,}5\,\mathrm{s}$ the network reaches an aggregate connectivity of 1.

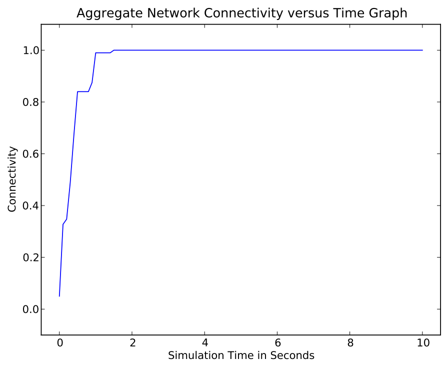


Fig. 7. Aggregate network connectivity versus simulation time graph.

To test network scalability, a test was conducted by increasing the node density in the $90\,\mathrm{m}$ by $90\,\mathrm{m}$ area. First we started with a 15 node simulation and gradually increased the number of nodes by 5 with each $30\,\mathrm{s}$ simulation run. The routing overhead of the cluster table messages and hello messages per node was measured in bytes and shown in Figure 8. The total control overhead and token control overhead is shown in Figure 9. The hello message control overhead per node remains fairly constant over the increasing node density, but it is noted that to send the hello messages frequently, has a high cost. The high cost is allowable, because of the high available bandwidth and it keeps the local routing information up to date. However the cluster table message overhead per node decreases slightly with increasing density, as expected. The token control messages make up the most of the control overhead and also decreases with node density. The less token control overhead per node simply means that every node in the network has less transmission chances as the node density increases.

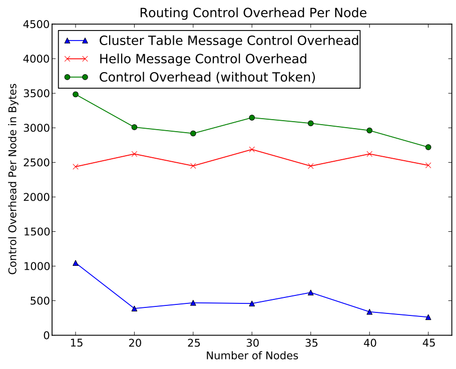


Fig. 8. Routing Control Overhead.

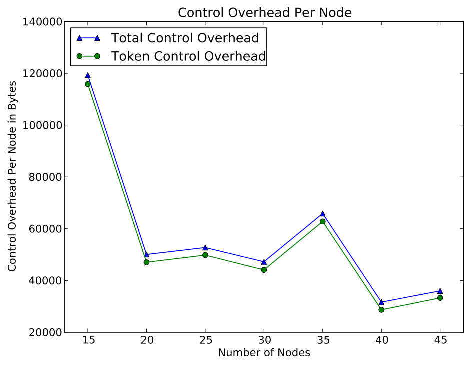


Fig. 9. Total Control Overhead.

V. Conclusion

A routing protocol for high density wireless ad hoc networks was investigated and background given on previous work which proved that cluster based routing protocols provide less control overhead for nodes at high densities. From this point of departure, the design process of an improved cluster based protocol was demonstrated. Two improvements to the CGSR protocol were stated and implemented in a simulated environment. Cluster generation, node states and routing tables of the protocol were discussed. A mechanism for the fast detection of link breakages by sending frequent small hello messages was introduced. The distribution of routing messages by selected nodes was shown. A routing function for the determination of a route to local and remote nodes was discussed.

Simple congestion control based on a token scheduling scheme was implemented, giving preference to nodes with more traffic. Congestion at individual nodes is prevented with weighted fair queueing by separating the forwarded traffic and the traffic generated by the node itself.

A separate performance module was implemented that allows aggregate simulation time measurements of parameters.

Lastly, tests and results were presented. The first test proved that clusters can be successfully set up and the possibility of reducing the number of nodes that transmit full routing information, ie. node-cluster associations and route information. A second test further importantly showed that an aggregate connectivity of 1 can be achieved and therefore, every node knows how to reach every other node.

Another test was conducted to show how the protocol scales in a high density environment. It showed that the cluster table message overhead per node, that contains the most routing information, decreases slightly with a higher density. The hello message overhead per node remains the same for increasing node density. The results show that the control overhead scales well with greater density and does not grow out of bounds. The only disadvantage is that each node gets less transmission chances with the higher node density. This work demonstrated the feasibility of the proposed improvements and desirability of further development and practical implementation.

References

[1] P. Mohapatra and S. V. Krishnamurthy, Eds., *Ad Hoc Networks, Technologies and Protocols*. Springer, 2004, ch. 1, p. 2.
[2] N. Guo, R. C. Qiu, S. S. Mo, and K. Takashi, “60-ghz millimeter-wave radio: Principle, technology and new results,” *EURASIP Journal on Wireless Communication and Networking*, vol. 2007, no. 68253, 2007.
[3] I. F. Akyildiz and X. Wang, “A survey on wireless mesh networks,” *IEEE Radio Communications*, Sep. 2005.
[4] C.-C. Chiang, H.-K. Wu, W. Liu, and M. Gerla, “Routing in clustered multihop, mobile wireless networks with fading channel,” 1997.
[5] M. Gerla and J. T.-C. Tsai, “Multicluster, mobile, multimedia radio network,” *Wireless Networks*, vol. 1, pp. 255–265, 1995.
[6] T. Clausen and P. Jacquet, *Optimized Link State Routing Protocol (OLSR) RFC 3626*, 2003.
[7] C. E. Perkins and P. Bhagwat, “Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers,” *SIGCOMM*, vol. 8, no. 94, 1994.
[8] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, “A review of routing protocols for mobile ad hoc networks,” *Ad Hoc Networks*, vol. 1, no. 22, June 2004.
[9] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” 1996, will appear as chapter in the book Mobile Computing.
[10] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen, “Scalable routing strategies for ad hoc wireless networks,” *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, vol. 17, no. 8, pp. 1369–1379, Aug. 1999.
[11] C. S. R. Murthy and B. S. Manoj, *Ad Hoc Wireless Networks, Architectures and Protocols*, 2004.
[12] J. Banks, I. John S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, W. J. Fabrycky and J. H. Mize, Eds. Prentice Hall, 2001.
[13] M. Löbbers and D. Willkomm, *A Mobility Framework for OMNeT++ User Manual Version 1.0a4*.
[14] Wireless 802.11b transceiver. [Online]. Available: http://goods.us.marketgid.com/goods/19387/
[15] L. L. Peterson and B. S. Davie, *Computer Networks*, 3rd ed., R. Adams, Ed. Morgan Kaufmann, 2003.
[16] S. C. North, *Neato Users Manual*, April 2004.

Daniël Kotze completed his Baccalaureate in Electronical and Electrical Engineering in 2007 at the University of Stellenbosch. He is currently busy with his Masters degree at the same institution. His thesis is about ad hoc wireless networks with a high node density and high available bandwidth. Research focus: Ad hoc wireless networks, signal processing (echo hiding in sounds).

Riaan Wolhuter has a B.Sc. B.Eng, M.Eng and a Ph.D. from Stellenbosch University, and a B.Sc(Eng)(Hons) from the University of Pretoria, in Electronic Engineering. He is a senior researcher at the Faculty of Engineering at Stellenbosch University, South Africa.

Appendix D

Multimedia Guide

The multimedia disk is composed as follows:

- The CHGRP folder contains the simulation code (part of git repository).
- The Thesis_Write folder contains the thesis document, SATNAC article and pictures.
- The Latency_Calculations folder contains the latency analysis Python script.
- The Literature Study folder contains articles and theses.
- The Analysis folder contains Python scripts for the generation of graphs.