Determining a Least-cost Routing and MAC Strategy for a Rural Communications Ad hoc Network

Stephan van Ellewee

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

Supervisor: Dr. R. Wolhuter

December 2006

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature Date

Abstract

Outside the confines of cities and metropolitan areas, telecommunications may still be required. Farmers may, for example, want to communicate with each other or with local municipality or law enforcement. Various factors may make the application of fixed infrastructure telecommunications networks to rural situations like these unfeasible. Fixed infrastructure may prove to be ineffective due to *geographic, social* or *monetary* reasons.

Ad hoc networking seems like an intriguing solution to these elements of the rural telecommunications problem. Instead of using the client-server architecture approach, ad hoc networks use a *peer-to-peer* network architecture that allows the network to change in a more dynamic fashion. Hosts of such a network can join or leave the network dynamically and will share in the forwarding responsibility. Routing is done dynamically.

Transceiver range is still an issue. To counteract this problem satellites can be used to *extend* the communications range of a network. Communication with a satellite can be added by using gateway hosts that are equipped to establish satellite up- or downlinks. Even if one such gateway host is deactivated, ad hoc network hosts should be able to find alternative gateways (if such alternative gateways exist).

For this thesis, various MAC and Network protocols will be evaluated. One protocol set will be selected and adapted to a low-bandwidth situation. Cross layer design will be used in an attempt to decrease overhead of this strategy. A simulation model was devised to predict system performance. These simulations was followed by interpretation of results which rendered a theoretical basis with which network behaviour can be explained and even predicted. A tool-like framework has, in effect, been developed for the simulation and development of ad hoc network protocols. Novel approaches to protocol behaviour analysis have also been devised.

Opsomming

Telekommunikasie word ook benodig in landelike gebiede. Boere mag, byvoorbeeld, in verbinding wil tree met mekaar of met die plaaslike munisipaliteit of wetstoepassing. Huidige vaste infrastruktuur telekommunikasie-netwerke mag egter nie altyd geskik wees vir sulke landelike situasies nie. Faktore sluit in *geografiese, sosiale* of *geldelike* redes wat daartoe kan lei dat vaste infrastruktuur nie effektief toegepas kan word nie.

Om hierdie redes wil dit voorkom asof *ad hoc netwerke* 'n goeie alternatief bied vir die landelike telekommunikasie-probleem. In plaas van die standaard kliënt-bediener benadering word 'n *eweknienetwerk-argitektuur* gebruik, wat toelaat dat die netwerk in 'n meer dinamiese wyse kan verander. Nodusse van so 'n netwerk kan dinamies bykom of weggaan van die netwerk en alle nodusse neem deel aan die aanstuur (*Engels:"forwarding"*) van pakkies. Paaie word dinamies gevind.

Kommunikasieafstand bly steeds 'n probleem. Om die effekte van hierdie probleem teen te werk, kan van satelliete gebruik gemaak word om dus kommunikasie oor lang afstande moontlik te maak. Verbindings met die satelliete kan bewerkstellig word met spesiale toegangsnodusse (*Engels:"gateway hosts"*) wat ook toegerus word met satellietkommunikasie-toerusting. Selfs as een so 'n toegangsnodus gedeaktiveer word, kan die ander nodusse van die ad hoc netwerk paaie na alternatiewe toegangsnodusse vind, indien sulke toegangsnodusse bestaan.

Vir hierdie tesis, word verskeie Medium Toegangs Beheer- (*Engels:"Medium Access Control (MAC)"*) en Netwerkprotokolle ondersoek. Een paar van hierdie protokolle word dan verwerk vir 'n lae bandwydte toepassing. Sogenaamde "Cross Layer" ontwerp word toegepas in 'n poging om 'n verlaging in die oorhoofse bandwydteverbruik te verminder. 'n Simulasiemodel word ontwikkel en gebruik in verskeie toetsgevalle. Hierdie simulasies was gevolg deur 'n ontleding van die resultate wat op hulle beurt 'n teoretiese basis tot gevolg gehad het waarmee die netwerk se werking verduidelik en selfs voorspel kan word. In effek was 'n stelsel ook ontwikkel wat die ontwerp van ad hoc netwerke sal moontlik maak. Nuwe metodes om protokolwerking te beskryf was ontwikkel.

Acknowledgements

I would like to thank:

- My study leader, Dr. Riaan Wolhuter, for his diligent mentorship, in spite of a hefty workload.
- My Parents, for their love, understanding and unfaltering support.
- To my friends, whom I treasure and miss.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

TCP	Transmission Control Protocol
DSR	Dynamic Source Routing
AODV	Ad hoc On-demand Distance Vector Routing
ORA	Optimum Routing Algorithms
LORA	Least-Overhead Routing Algorithms
MAC	Medium Access Control
CSMA	Carrier Sense Multiple Access
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
MACA	Multiple Access with Collision Avoidance
MACA-BI	Multiple Access with Collision Avoidance By Invitation
DCF	Distributed Coordination Function
CCA	Clear Channel Assessment
PLCP	Physical Layer Convergence Protocol (Sublayer)
NAV	Network Allocation Vector
SIFS	Short Interframe Space
DIFS	DCF Interframe Space
BEB	Binary Exponential Backoff
FES	Future Event Set

Chapter 1

Introduction

1.1 Rural Telecommunications

Rural telecommunications is an emerging field that poses many challenges for researchers. Various factors may make established telecommunications infrastructure unfeasible to implement. Critical infrastructure such as cellphone towers would have to exist in areas that are not able to provide the power or maintenance requirements necessary to allow the proper functioning of the communications network as a whole. For this reason *ad hoc networking* is worth investigating as a possible effective approach to the rural communications problem.

1.2 Ad hoc Networks

The term *ad hoc* is a Latin phrase meaning “for this”. In this context, ad hoc refers to the spontaneous connection of hosts to form a communications network. By using a modified (and more generalized) version of the definition that is given in [2] an ad hoc network can be defined as a self-configuring network of routers (and associated hosts) connected by wireless links – the union of which form an arbitrary topology. The network’s wireless topology may change rapidly and unpredictably. Ad hoc networks usually refers to wireless networks in which the forwarding of packets are done by the hosts that the network consists of. This makes up for the limited range of the physical radio infrastructure. In a true ad hoc network, no “client-server” relationship exists. All hosts are similar to each other and act as clients, servers and routers. The benefit of such a network is that when one host is disabled other hosts can take over the forwarding responsibilities of this host. For this reason ad hoc networks can also be referred to as *self-healing networks*.

1.3 Application of Ad hoc Networks in Satellite Enabled Rural Communications

A rural communications infrastructure can benefit from ad hoc networks due to the properties previously stated. Forwarding responsibility is shared by the network's various hosts and this allows power consumption of a single host to be kept to a minimum. Defective hosts do not cause a critical failure of the network, which implies that host maintenance is not as essential as in fixed infrastructure networks. Further extension of the network can be established by the use of hosts with satellite up- and downlink capabilities. When a packet is unable to reach a certain network address, it simply needs to be forwarded to a host with a satellite up/downlink. The message can then be forwarded via the satellite to the network that contains the host with the appropriate network address.

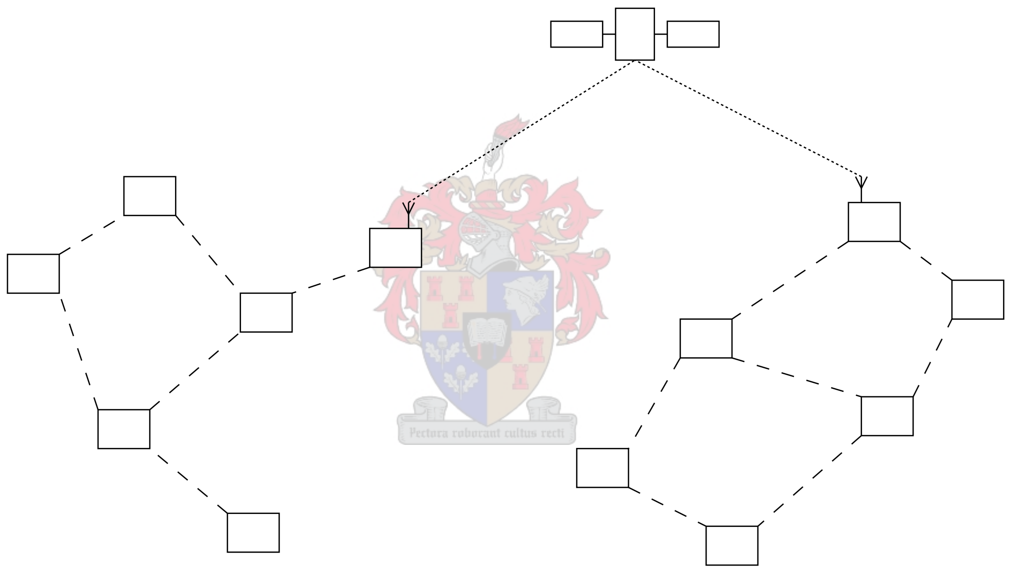


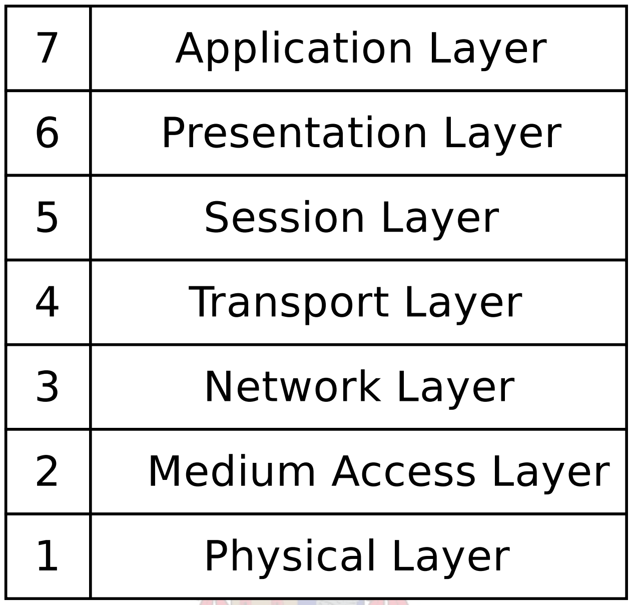
Figure 1.1: *Satellite Enabled/Assisted Ad hoc Rural Communication*

In Figure 1.1, the blocks represent stationary terrestrial hosts which can be deactivated or disabled for any reason. The dashed lines between them indicates a possible radio communication proximity. The satellite acts as a forwarding node, that extends the range that a message can travel. The hosts that are being temporarily connected by the satellite will be known as *satellite up/downlink hosts* for the rest of this thesis. These hosts are also equipped with satellite communications functionality and will function as a gateway to another subnetwork for any packets that may require routing to a remote subnetwork destination (remote subnetwork host). Later, it will be seen that a host will seek hosts with satellite up/downlink capabilities at network initialization.

1.4 Ad hoc Protocol Description

1.4.1 The Protocol's Position in the Protocol Stack

This thesis describes a protocol that will allow ad hoc rural communications. The following is a short description of the protocol's place in the OSI 7 Layer model:



7	Application Layer
6	Presentation Layer
5	Session Layer
4	Transport Layer
3	Network Layer
2	Medium Access Layer
1	Physical Layer

Figure 1.2: *Figure based upon the Open Systems Interconnection (OSI) Reference Model*

7. Application Layer: This layer acts as an interface for the user to the network. Examples of these are web-browsers and mail clients.

6. Presentation Layer: This layer converts lower layer data into a format that the Application Layer can interpret.

5. Session Layer: This layer creates, manages and terminates communication between hosts.

4. Transportation Layer: This layer allows the upper layers to be oblivious to the functioning of the network and provides an interface to the lower protocol stack layers. The most famous example of a layer 4 protocol is TCP.

3. Network Layer: The main task of this layer is to determine routes from a source host to a destination host.

2. Medium Access Layer: This layer must establish connections between neighbouring hosts and in doing so allow efficient use of the medium. The medium can be anything from co-axial cable to an assigned radio frequency. One implementation of a medium access protocol is Ethernet.

1. Physical Layer: This layer is responsible for transmitting bits over the physical medium. It includes all electrical specifications for the network hardware.

The strategy that has been proposed will specify the second and third layers of the protocol stack, namely Medium Access and Network (Routing) Layers.

1.4.2 Design Constraints

In order to design any Network and MAC Layer protocol, the physical layer must be considered. The following constraints were given for the system, due to the design parameters of the satellite deployment:

- Half-duplex radios (Bitrate of 9600 bits/second)
- Low Latency
- Maximum Efficiency
- Long range communication (host separation distance in the `km`-range)

The network would consist of a collection of hosts that would be separated at a distance of a few kilometers. For simulation purposes a maximum distance of 8 `km` was chosen. It could be varied but will not be able to exceed the maximum UHF communications point to point available distance. The network would have to be robust by finding alternate routes for packets after a route breaks. This would occur due to a host failure or deactivation. The rural nature of the network would increase network maintenance requirements and this implies that a *self-healing* approach would have to be applied in which the network adapts to changes (and failures) in its topology. For this network the following parameters must be considered:

Latency refers to the difference between the time the packet is queued at the source and the time it is serviced at its destination. The user perceives latency as the delay needed for a message to arrive.

Throughput is given by the quotient of the packet size and the latency.

Efficiency is given by the following relationship:

$$\text{Efficiency} = \frac{A}{B+A} \tag{1.1}$$

with A : Data Packet Length (bits)
B : Overhead Amount (bits)

From this relationship the obvious conclusion is that a higher overhead will result in a lower efficiency. Overhead can be caused by the protocols used due to MAC (Layer 2) layer handshaking, Routing (Layer 3) techniques and protocol header sizes. The aim of the protocol must therefore be to keep these factors to a minimum.

"Least-cost" Routing and MAC Strategy

When this thesis refers to a *least-cost* Routing and MAC Strategy it refers to a strategy that will incur minimal *overhead*. Due to the low bandwidth that is available overhead will increase latency and decrease overall network throughput. It is, therefore, critical that the strategy developed constantly guards against an unrestricted increase in message overhead.

1.5 Protocol Design Process And Techniques Used

The Network and MAC Layer interaction must endeavour to improve network performance. In order to do this *latency* must be decreased. Latency is increased by delays caused by protocol overhead (MAC and Network Layer) and interframe spacing. This implies that latency can be improved if these factors are minimized. MAC Layer timing adaptations will be attempted in Chapter 3. Network and MAC Layer adaptations will include the following methods:

- *Promiscuous Listening* – This concept refers to modified MAC Layer behaviour but is essential to this Network Layer's function. This technique allows the MAC Layer Data packets to be sent up to the Network Layer, *regardless* of the intended MAC Packet's destination address. Overheard packets like these can be used to gain routing information, thereby avoiding unnecessary route requests.

- *Multipath Routing* – In an ad hoc wireless network situation various routes may exist to a single destination. This means that various routes can share packet routing loads. A method for distributing packet loads over various routes will be described in Subsection 1.6.2.

1.6 Contributions

1.6.1 General Contributions

General contributions submitted for this thesis can be summarised as follows:

- *A Rural Ad hoc Networking model* have been proposed. This model addresses problems such as *Maintenance* and *Range* in rural communications.

  - *Maintenance*: Ad hoc networks are more robust than their fixed infrastructure contemporaries. This implies that a deactivated host or broken link can be resolved by the network itself. No outside maintenance should be necessary to re-establish connectivity after host failure/deactivation.

  - *Range*: When a host from a small rural network needs to communicate with hosts in distant rural networks, it can simply forward its message to a satellite up/downlink host. From the satellite up/downlink host, the message can be transmitted via the satellite to the distant network. This would effect the range to be increased in which a network can allow rural communication. Latency would, off course, be experienced, depending on the duration of the satellite orbit.

- *Ad hoc Network Development Framework* – Simulation models were designed and a theoretical basis was created in order to closely interpret and predict the performance of a topology. A method for illustrating packet route bias and preference have also been proposed. This method will be used in Chapter 6. Various scripts were also devised (in *Bash* and *Python*) in order to gain and interpret results, generate graphs, automate simulation execution and make up for shortcomings of the simulation environment.

1.6.2 Implementation Specific Contributions

The points of Section 1.5 are based on existing techniques. In order to improve network performance for this current application, the following adaptations have been contributed:

- *Implicit Route Error Recovery* – When a Network Layer packet is transferred between two hosts, and the MAC Layer of the transmitter does not receive an acknowledgement from the receiver, the MAC Layer will resubmit the packet to the Network Layer for a rerouting operation. This avoids the overhead of sending route error messages through the network. This works in conjunction with the next aspect in order to make the protocol useful. An improvement is suggested in Chapter 7.

- *Lottery Route Selection* – This method has been proposed in Chapter 5 to govern the packet loads received by different routes. This is a way of biasing multipath routing in order to distribute different loads over routes with different capabilities. With this method, a route metric is chosen to assign a certain amount of "lottery tickets" to a route. The more lottery tickets a route has, the higher its chances of "winning" a packet. In this thesis the metric chosen will be *inverse route length.*

1.7 Work Completed for this Thesis

In the process of working on this thesis the following was accomplished:

- Various Network and MAC layer approaches where evaluated.

- A MAC layer model based on 802.11, but for relatively low speed half-duplex radios was devised.

- A Network Layer model was devised based on Dynamic Source Routing, adapted for compliance with the system constraints already mentioned.

- Simulation software was evaluated and a simulation model was constructed based on this software.

- Jackson queueing network theory was used to determine steady state operating characteristics.

- Using knowledge of system characteristics, simulated situations was described.

1.8 Thesis Description

The rest of this thesis will deal with the process employed in the design of the MAC and Network/routing layers. It can be subdivided into the following subsections:

- *Chapter 2* will deal with previous work regarding to MAC Layer implementations.
- *Chapter 3* will give some insight into the modified MAC Layer that will make up first part of the designed work.
- *Chapter 4* deals with previous Network Layer protocols.
- *Chapter 5* gives a description of how the newly devised Network Layer works (the second part of the design).
- *Chapter 6* will present simulations that will deal with *Latency*, *Efficiency* and *Throughput*.
- *Chapter 7* will discuss how the protocols of this thesis can be expanded, applied and improved.

Chapter 2

The Medium Access Layer: Background

2.1 Wireless Link Layer Problems

In wired networks, protocols like Ethernet (CSMA/CD) are generally sufficient to ensure that collisions of data packets are minimized. This is due to the fact that all hosts work on a medium that is common to all parties involved. In a wireless situation, it is however possible for transmissions of some hosts to leave other hosts unaffected due to limited host transmission range. Two main problems that arise due to this occurrence are the *hidden* and *exposed* terminal problems.

2.1.1 The Hidden Terminal Problem

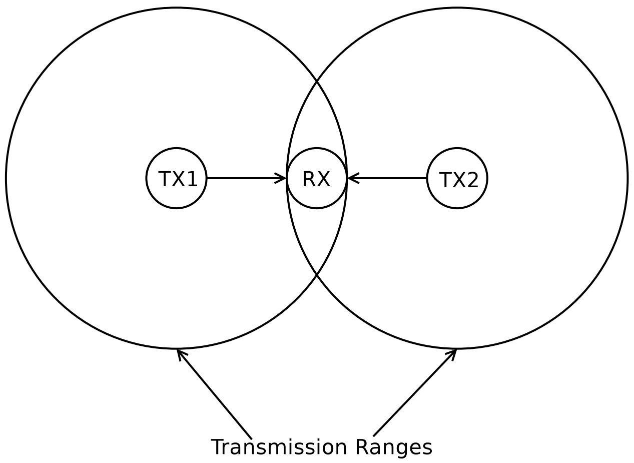


Figure 2.1: *Hidden Terminals: TX1 and TX2 are hidden from each other*

As can be seen from Figure 2.1 TX1 and TX2 are oblivious to each others existence. Normal CSMA will not allow TX1 to detect when TX2 is transmitting or vice versa. This means that TX1 and TX2 are *hidden terminals* to each other.

2.1.2 The Exposed Terminal Problem

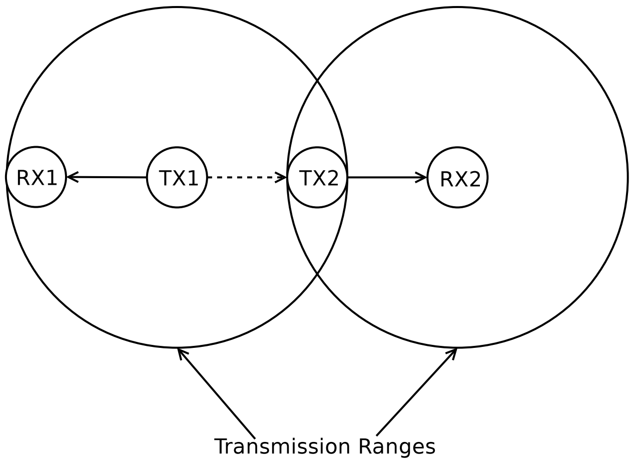


Figure 2.2: *Exposed Terminal: TX1 is blocked by the a transmission from TX2*

Here it is evident that if TX2 is engaged in a transmission to RX2 then TX1 will determine that the channel is busy (by means of Physical Carrier Sense) and will thus not be able to send a message to RX1. This problem is called the *Exposed Terminal Problem* due to the fact that TX1 is exposed to TX2's transmission.

These problems could result in a great loss of the throughput of the network, so a number of protocols have been proposed to deal with these issues.

2.2 Protocols Investigated

The following is a short description of the MAC Layer strategies that have been investigated as well as the reasons why they have been deemed inappropriate for the current application.

2.2.1 CSMA

Carrier Sense Multiple Access is a method (first modelled by Kleinrock & Tobagi in [3]) in order to improve the throughput of a channel. CSMA uses *physical carrier sense* to sense channel activity. This behaviour causes a host that requires to send a message, to sense the medium for channel activity first. If the channel is currently available then the host may choose to transmit. This decision is based on the variation of CSMA that is in use. If the channel is busy, a timer is set to wait for a retransmission. The CSMA protocol has *1-persistent and `p`-persistent* variations. The value in front (such as `1` or `p`) denotes the probability of a transmission taking place after the channel was sensed to be free.

2.2.2 CSMA/CA

This protocol is an adaptation on the basic CSMA technique for wireless networks. Where basic CSMA only uses physical carrier sense to detect a busy medium, CSMA/CA extends CSMA by also making use of *virtual carrier sense*. This method makes use of a RTS (*Ready-to-send*) packet transmission to notify all neighbours that this host intends to transmit. If this notification packet does not collide with the packets of any other host and the packet arrives without errors at the destination host, then the receiving host responds with a CTS (*Clear-to-send*) signal. Both RTS and CTS messages contain information like message source, message destination and message duration. All hosts that overhear RTS/CTS messages defers from using the channel a period that will be sufficient for this handshaking process to complete. A well-known extension of this protocol, namely IEEE 802.11 DCF (or Distributed Coordination Function), has been used as the basis for the MAC protocol that this work will be based upon. In this thesis, the concepts of DCF and CSMA/CA will be used interchangeably, as is done in the IEEE 802.11 specification [4].

2.2.3 MACA

MACA (Multiple Access Collision Avoidance) has been proposed by [5] as an alternative to CSMA/CA. MACA only uses *virtual carrier sense* to determine medium access. The way MACA accomplishes this is by adding the data transmission time in its RTS header. This receiver will respond with the same information by including this same time in its CTS header. This will lead to all nodes overhearing these RTS and CTS messages to know how long to back off from the medium. This protocol could also be enhanced by adding a automatic power control feature. This causes the host to only use the minimum amount of transmission power for each exchange, thereby limiting the interference that it may cause other hosts that are positioned further away. This protocol was not chosen, due to the lack of a proper protocol specification.

2.2.4 MACA-BI

MACA-BI (Medium Access with Collision Avoidance - By Invitation) as explained in [6] is referred to as a receiver-initiated medium access protocol, due to the fact that the receiver "invites" the sender to transmit. The receiver host would have to predict when to send out RTR (*Ready-to-Receive*) messages *and* to which hosts. The polled hosts do not, however, have to send packets to the polling host. The receiver host would have to know how many packets are waiting in each neighbours' packet buffer in order to allow fair sharing of the medium. This protocol was also a very interesting candidate due to the fact that it boasted a low overhead while allowing for a higher possible throughput. This would however require a complex initialization step in order to allow proper flow between the hosts. Another issue is that if traffic is bursty, the performance of MACA-BI degrades to normal MACA.

2.2.5 Polling Medium Access Techniques

Techniques such as Round Robin Scheduling was determined to be unfeasible due to the client-server relationship that is implied in the architecture. All hosts would have to agree to make one host a polling host, while the others agree to await a signal to be polled. This system can not be feasibly applied to ad hoc networks due to their peer-to-peer architechture.

2.3 Summary

In this chapter, some difficulties faced by a wireless MAC Layer was explained. Previous MAC Layer designs were also evaluated and considered. CSMA/CA was chosen as the basis for the MAC Layer of this thesis. In the next chapter, the new MAC Layer will be designed by modifying a popular implementation of this protocol (WiFi or IEEE 802.11). *Seperation distance between hosts* and *modem speed* are factors that will be kept in consideration when developing the newly modified MAC protocol.

Chapter 3

The Medium Access Layer: Protocol Description

The following chapter will deal with the adaptation of WiFi or IEEE 802.11 DCF in order to be used for a long-range network that make use of a low bitrate physical layer. Firstly, the basic function of the network will be described. This will be followed by descriptions of the modifications done.

3.1 Introduction

As explained in the Chapter 1, the Network Layer will require some extra functionality from the MAC Layer. This refers to the following:

- *Promiscuous Listening*
- *Implicit Route Error Recovery/Maintenance* – This refers to the behaviour of the MAC Layer that causes the resubmision of unsuccessful Network Layer Packets to the Network Layer. Packets that do not receive an `ACK`-packet from the destination are considered unsuccessful.

These elements will be incorporated into the MAC Layer, but it will not be essential to the functioning of the basic MAC Layer state machine. These factors will have a larger effect to the function of the new Network Layer and will therefore be dealt with in Chapter 5. Other adaptations to the MAC Layer are as follows:

- *Revised Timing Values* – Due to the low bitrate modems (9600 bits/second) that will be used, longer delays will be required from the MAC Layer. A scaling operation will be done in order to compensate for the low bitrate.
- *Distance Effect Compensation* – The effect of distance to the hosts will be investigated. It will, however, be determined that the effect of distance on the MAC Layer's timing is negligible due to the long delays imposed by the previous point. This would not have been the case in a faster bitrate situation.

3.2 IEEE 802.11b DCF

As previously stated, the MAC protocol proposed by this thesis is based on CSMA/CA or more specifically a modified version of the IEEE 802.11b DCF protocol. Figure 3.1 is a state graph based on simulation code provided in [7]. Reasons detailing the transitions from one state to another is indicated at the beginning of each edge. Please note that the dashed edge in the graph indicates a modification. It was determined that the model failed to function if arrival rates became too high. This modification resolved this problem.

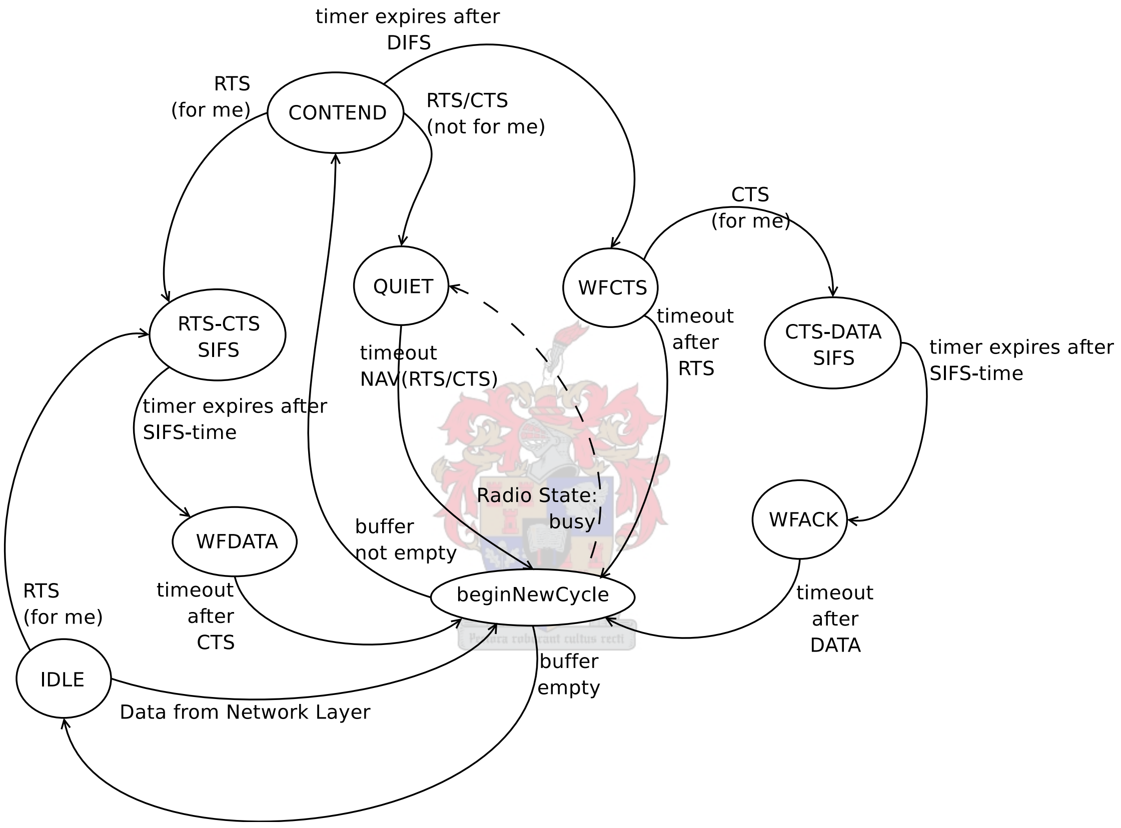


Figure 3.1: *State diagram of the model used for 802.11b DCF simulation*

Transmission Side Function

In order to describe the IEEE 802.11b DCF protocol, the following description will be given, paraphrasing the simulation model source code provided by [7]. The transmission side of IEEE 802.11b DCF is as follows:

1. When the host requires the medium for transmission, it enters a `CONTEND`-state. During this state, a timer is set for one DCF Interframe Space (DIFS) period. This period allows half-duplex radios to sense the channel in order to determine channel availability.

2. A determination that the medium is not in use will then result in the transmission of a `RTS`-packet and start a time-out timer. This will be referred to as the *Wait For Clear To Send State* (`WFCTS`-state).

3. Upon receiving a `CTS`-packet one of two occurrences could take place:

   - If the `CTS`-packet is meant for this host, then the host waits for one Short Interframe Space (SIFS) period before sending the `DATA`-packet. If activity is sensed during the SIFS period, then all timers are reset, and the exchange is then considered to be a failure. The host then returns to a `CONTEND`-state and retries with a time equivalent to one DIFS-period plus a certain backoff time. This time is governed by using the Truncated Binary Exponential Backoff Algorithm (TBEB). If successful, a `DATA`-packet is transmitted. A time-out timer is also set to expire if an acknowledgement message, or `ACK`-packet, is not received. This state's designation for the rest of the thesis will be the *Wait For Acknowledgement State* (`WFACK`-state).
   - If the `CTS`-packet is not meant for this host, it defers for a NAV(CTS)[1]-period while going into a `QUIET`-state. Figure 3.2 demonstrates this behaviour.

4. On the receipt of an `ACK`-packet, the host resets all timers and restarts in a `CONTEND`-state, thereby restarting the whole cycle from the first step.

The Binary Exponential Backoff Algorithm

The *Binary Exponential Backoff Algorithm* refers to the behaviour of a host after a packet collision has occurred. This behaviour causes a host to wait for a uniformly random number of slottimes between zero and $2^i - 1$ before reattempting a retransmission. The i-variable refers to the number of *retry attempts*. The *Truncated Binary Exponential Backoff Algorithms* acts similarly but the maximum amount of possible slottimes have an upper bound that can be reached.

[1]Network Allocation Vector for a Clear-To-Send message

Receiving Side Function

The receiving side functions as follows:

1. After receiving a `RTS`-packet the host cancels all timers.
   - If the packet is not meant for this host, it defers for a time equal to NAV(RTS)[2] while staying in a `QUIET`-state. This can be seen on Figure 3.2. This allows exchanges between neighbour hosts to continue uninterrupted.
   - If the packet is meant for this host, then the host waits one SIFS period before transmitting a `CTS`-packet.
2. After sending the `CTS`-packet, the host will wait for data to be received. In this thesis this state will be denoted as the `WFDATA`-state.
3. After the `DATA`-packet arrives at the receiver host, it then proceeds to wait for one SIFS period before transmitting an `ACK`-packet. This ends the exchange and allows a new contention cycle to begin.

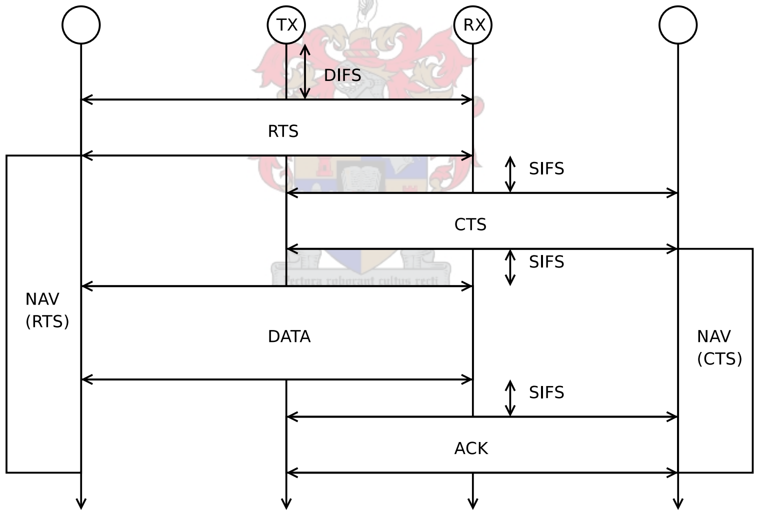


Figure 3.2: *Normal IEEE 802.11(DCF) operation*

As can be deduced from the diagram, NAV(RTS) and NAV(CTS) can be calculated as follows:

$$\text{NAV(RTS)} = 3\text{SIFS} + \text{duration}_{CTS} + \text{duration}_{ACK} + \text{duration}_{DATA}$$
$$\text{NAV(CTS)} = 2\text{SIFS} + \text{duration}_{ACK} + \text{duration}_{DATA}$$

After the source (transmitting) host receives a CTS from the destination host, it sends the data packet to the destination. If successful reception of the data was achieved, the receiver host waits another SIFS period before transmitting an acknowledgement packet.

[2]Network Allocation Vector for a Ready-To-Send message

3.3 Modifications of IEEE 802.11b DCF

This modification of IEEE 802.11b DCF must take into consideration the distances at which transmission takes place, as well as the hardware layer's bitrate. For IEEE 802.11b distances of a few hundred meters does not make a real difference to propagation delay T_p. Longer distances do affect the route propagation delay and this may lead to timing conflicts in normal IEEE 802.11b DCF operation.

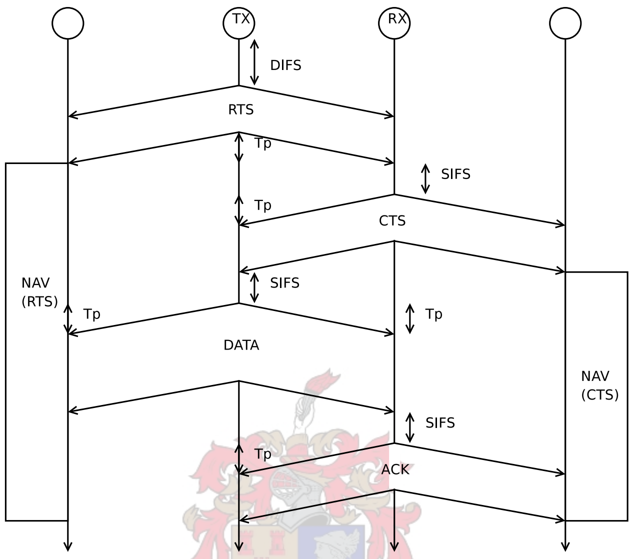


Figure 3.3: *Modified IEEE 802.11(DCF) operation for long range*

The bitrate will require the *slottime* and *SIFS* values to be changed. The definition of a slottime varies between various explanations and implementations, but for this thesis a slottime will be seen as the time it takes for the host to determine if the channel is clear and for a signal to reach the host. A description of the modifications that need to be done to the slottime and SIFS time follows.

3.3.1 Choosing a Slottime Value

The basic backoff time unit for a CSMA-type protocol is a slottime. From this time various other delays are calculated, such as DIFS and EIFS times.

Original IEEE 802.11b DCF Slottime

The definition of slottime follows from [4] as:

$$\begin{aligned}\text{aSlottime} &= \text{aCCATime} + \text{aRxTxTurnaroundTime} + \text{aAirPropagationTime} \\ &\quad + \text{aMACProcessingDelay} \qquad (3.1)\end{aligned}$$

According to [8] the `aMACProcessingDelay` represents the MAC Layer processing delay and `aRxTxTurnaroundTime` represents the switching time for a radio to change from receive to transmit operation mode. The default values for the MAC protocol was chosen to be the following for the IEEE 802.11 DSSS Physical layer:

Parameter Name	*Value*
SIFS	$10\mu s$
aCCATime	$15\mu s$
aRxTxTurnaroundTime	$5\mu s$
aMACProcessingDelay	$0s$
aAirPropagationTime	$1\mu s$
Slottime	$20\mu s$
DIFS	$2 \times \text{Slottime} + \text{SIFS} = 50\mu s$
Physical layer bitrate	11 Mbits/second

Table 3.1: *Default 802.11b Parameters*

Adaptation of 802.11b DCF Slottime values

In order to determine a sufficient slottime for the 9600 bits/second application `aCCATime` and `aRxTxTurnaroundTime` must be modified. It is logical to assume that if Table 3.1 was meant for a 11 Mbits/second hardware layer bitrate, a 9600 bits/second bitrate will require delays that are more than 1000 times longer. To summarise the values:

$$\begin{aligned} \text{aCCATime}_{old} = 15\mu s &\rightarrow \text{bitrate}_{old} = 11 \times 10^6 \text{ bits/second} \\ \text{aCCATime}_{new} &\rightarrow \text{bitrate}_{new} = 9600 \text{ bits/second} \end{aligned}$$

where aCCATime_{new} is the new unknown delay that will be determined for an application using the new bitrate (9600 bits/second). This results in a simple scaling operation to determine this value:

$$\begin{aligned} \text{aCCATime}_{new} &= \left(\frac{1}{\text{aCCATime}_{old}} \times \frac{\text{bitrate}_{new}}{\text{bitrate}_{old}}\right)^{-1} \\ &= \left(\frac{1}{15 \times 10^{-6}} \times \frac{9600}{11,000,000}\right)^{-1} \\ &= 0.017187500000000001s \\ \text{or} &\quad 17.18750ms \end{aligned}$$

A similar scaling can be done for `aRxTxTurnaroundTime`:

$$\begin{aligned} \text{aRxTxTurnaroundTime}_{old} = 5\mu s &\rightarrow 11 \times 10^6 \text{ bits/second} = \text{bitrate}_{old} \\ \text{aRxTxTurnaroundTime}_{new} &\rightarrow 9600 \text{ bits/second} = \text{bitrate}_{new} \end{aligned}$$

From this follows:

$$\begin{aligned}\text{aRxTxTurnaroundTime}_{new} &= \left(\frac{1}{\text{aRxTxTurnaroundTime}_{old}} \times \frac{\text{bitrate}_{new}}{\text{bitrate}_{old}}\right)^{-1}\\ &= \left(\frac{1}{5\times 10^{-6}} \times \frac{9600}{11,000,000}\right)^{-1}\\ &= 0.0057291666666666671s\\ \text{or} &\quad 5.72916667ms\end{aligned}$$

When determining `aAirPropagationTime` a maximum distance of 8000 `m` will be used. `aAirPropagationTime` can then be determined as follows:

$$\begin{aligned}\text{aAirPropagationTime} &= \frac{\text{distance}}{\text{c}}\\ &= 8000/(3\times 10^{8})\\ &= 26.6667\mu s\end{aligned}$$

with c being the *speed of light*.

This shows that the `aAirPropagationTime` value is very small($26.6667\mu s$) in comparison to the other values that make up the MAC Layer slottime (`aCCATime` = $17.18750ms$ and `aRxTxTurnaroundTime` = $5.721916667ms$). This propagation delay would have had a larger effect on the Slottime and SIFS-time in a system with a higher bitrate. This means that the host slottime can be determined by substitution of the values of Equation 3.1:

$$\begin{aligned}\text{aSlottime} &= 17.18750\times 10^{-3} + 5.7219166\times 10^{-3} + 26.6667\times 10^{-6} + 0\\ &= 0.022943333333333336\\ &\approx 23ms\end{aligned}$$

3.3.2 Choosing a SIFS time value

Original 802.11b DCF SIFS time

In 802.11b DCF specification, the SIFS period is defined as follows:

$$\begin{aligned}\text{aSIFSTime} &= \text{aRxRFDelay} + \text{aRxPLCPDelay} +\\ &\quad\ \text{aMACProcessingDelay} + \text{aRxTxTurnaroundTime}\end{aligned} \tag{3.2}$$

In Table 3.1 `aSIFSTime` has been specified to have a default value for IEEE 802.11 DSSS-PHY layers ([4]) of $10\mu s$. The new values have the following meanings:

- *aRxRFDelay* – represents the delay caused by the radio receiver(according to [8]),

- *aRxPLCPDelay* – represents the nominal time that the Physical Layer Convergence Protocol uses to deliver a bit from the medium to the MAC Layer.

The PLCP Sublayer translates/encapsulates MAC sublayer protocol data units (or MPDUs) into PLCP protocol data units (PPDUs). These PPDUs are then sent to the Physical medium dependant sublayer. This sublayer is specific to the actual medium in use. These two sublayers make up the Physical Layer. In normal IEEE 802.11 DCF the three available physical layers include:

- *Frequency Hopping Spread Spectrum (FHSS) for the ISM band*
- *Direct Sequence Spread Spectrum (DSSS) for the ISM band*
- *Infrared Physical layer*

Further descriptions of these layers are outside the scope of this thesis.

Adaptation 802.11b DCF SIFS time

In choosing a SIFS time the following must remain true (according to the IEEE 802.11b specification [4]):

$$\text{SIFS} < \text{ST} < \text{DIFS}$$

The previously determined `aRxTxTurnaroundTime` will be substituted in Equation 3.2. The MAC Layer processing delay will be neglected. The `aRxPLCPDelay` will be assigned a value equal to the time it would take the physical medium to transfer 1 bit to the MAC Layer. This would be $1/bitrate = 1/9600 = 0.0001041667s$ or $104.1667\mu s$. `aRxRFDelay` should also be in this range. This implies that the brunt of the SIFS delay will be attributed to `aRxTxTurnaroundTime` which is a millisecond value ($5.7291666ms$). This number will therefore simply be rounded up and used as the new `aSIFSTime`. The new values will therefore be given be the following table:

Parameter Name	*Value*
SIFS	$6ms$
aCCATime	$17.18750ms$
aRxTxTurnaroundTime	$5.7291666ms$
Slottime	$23ms$
DIFS	$2 \times \text{Slottime} + \text{SIFS} = 52ms$
Physical layer bitrate	9600 bits/second

Table 3.2: *Chosen MAC Layer Parameters*

3.4 Summary

In this section the basic simulation model for the IEEE 802.11b DCF protocol was described. A state machine description was determined of simulation code from [7]. The timing was then adapted to make up for the low bitrate and *possible* propagation delay (T_p) effects that may have occurred due to the fact that the distances in question between hosts of a rural communication situation is in the `km`-range. This was, however, found to be negligible in comparison to the other delays in question, such as the *Clear Channel Assessment*.

The next chapter will deal with previous work regarding the Network Layer and will give an explanation as to the why the Network Layer (that has been developed in this thesis) was based on a certain protocol.

Chapter 4

The Network Layer: Background

4.1 Design Methodology For The Network Protocol

It was decided to base the network protocol on an existing ad hoc routing protocol. Various network/routing layer strategies were investigated, but it was found that most of these are variations on Distance Vector and Link State routing. The most notable of these have been summarised in Sections 4.3.1, 4.3.2 and 4.3.3. The advantages and disadvantages must be considered while keeping the design constraints of Section 1.4.2 in mind, such as the requirement for a low-cost routing strategy, is taken into account. The routing layer must, for example, not cause unnecessary flooding or route discovery, due to the fact that it would violate previously mentioned constraints.

4.2 Network Protocol Classifications

4.2.1 LORA versus ORA

These two acronyms refers to different network protocol design paradigms. *ORA (Optimum Routing Algorithms)* ensures that nodes use the most optimal routes between a source and destination. A characteristic side-effect is the large amount of overhead that is generated due to the protocol's route request and discovery phases. *LORA (Least-Overhead Routing Algorithms)* differs from ORA due to the fact that these protocols do not necessarily use the most optimum routes between source and destination, but follows an approach that will ensure that overhead is kept to a minimum. This is done by ignoring link-failures of a path if another path is still available towards a destination.

4.2.2 Reactive versus Proactive

A network protocol is referred to as reactive when it *reacts* to changes in the network topology instead of actively seeking changes. The class of network protocols that *does* actively seek changes in the network is referred to as a *proactive* protocol.

4.3 A Short Description of Existing Routing Protocols

4.3.1 Link State Routing (LSR)

Hosts using link state routing exchanges link state information with each other in order to attain a mapping of the entire network. Firstly, the network hosts determine who their neighbours are via a reachability protocol. This information is then used to construct a *link-state advertisement* message that is periodically flooded throughout the network. In this way, all the hosts in the network will, in theory, be aware of each other. The main problem that is apparent with such a strategy is that whenever *any* change takes place in the network, the updated link-state information would have to be flooded through the entire network again. This approach and the ad hoc versions of this protocol was originally devised for current Internet and wired network applications. In these situations bandwidth and transmission speed is not as scarce as in situations specified by the design constraints of Section 1.4.2. This can be seen as a proactive protocol.

4.3.2 Ad hoc On-demand Distance Vector Routing (AODV)

Distance vector routing is a protocol that distributes the topology information across the network. No singular host has a complete picture of the entire network. The network maintains information about its neighbours by means of periodic `HELLO`-packet. Routing is then done as follows: the only information that the host has about a destination is the next hop (the successor) and the “distance” to the destination. *This implies that a host is only aware of one route to the destination host.* Ad hoc on-demand distance vector routing (AODV) [9] extends this method by adding a *destination sequence number* for each routing table entry. When faced with more than one route to a destination, the route table entry with the largest sequence number will be added to the host table. This modification also ensures that the routing table may not be updated by obsolete routing information and also prevents the occurrence of routing loops. The following figure will give a graphical representation of the route discovery and maintenance processes. Network discovery and maintenance for the AODV-protocol (and the DSR-protocol) will now be described. Changes of the network are registered reactively.

Basic Route Discovery

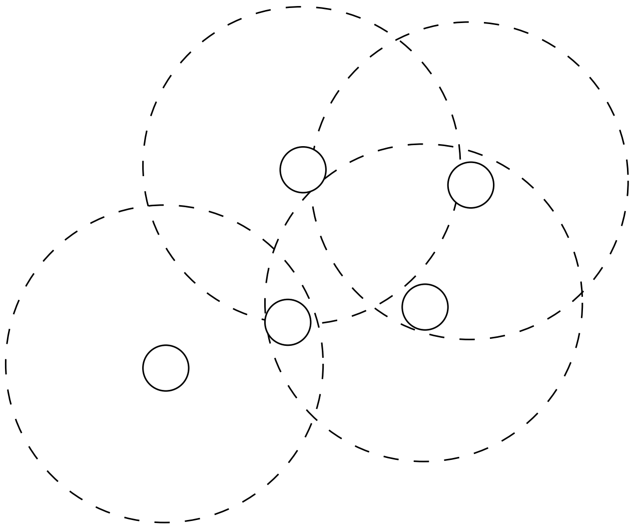


Figure 4.1: *Representation of an ad hoc radio network*

Figure 4.1 is a simple representation of a wireless ad hoc network. This model will be used to give a basic description of a network of nodes.

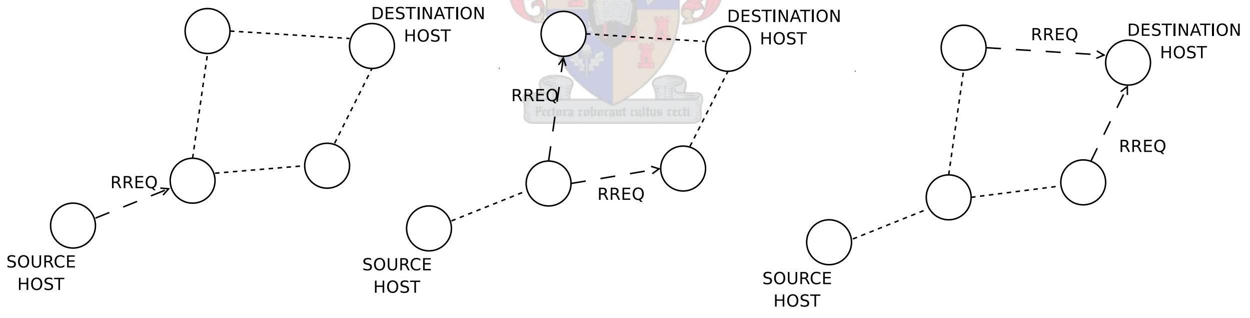


Figure 4.2: *Representation of Route Discovery*

In Figure 4.2 route discovery is done. This occurs when a host needs to determine a route to a new unknown location. It is possible, however, for a host to respond if it is not the destination host, but has information on how to reach the destination. The route discovery process is a flooding operation.

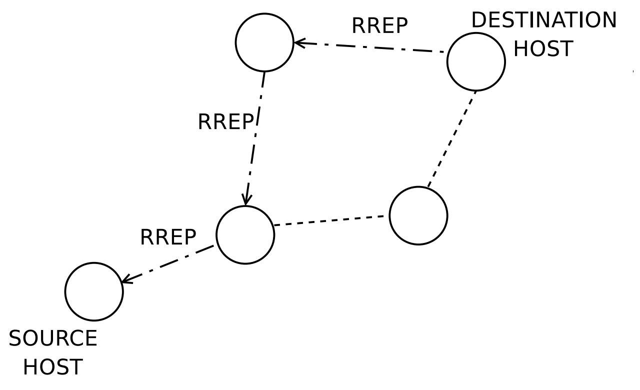


Figure 4.3: *Reply to source host*

Next the responding host unicasts a route reply back to the source host. This informs all hosts along the way how to reach the destination host. In **AODV** this is stored in a table that stores for each destination the "next hop" and "distance" information. The *distance* refers to the amount of hops required to reach the destination. This is shown in Figure 4.3. Data packets can then move between message Source and Destination hosts. (Figure 4.4)

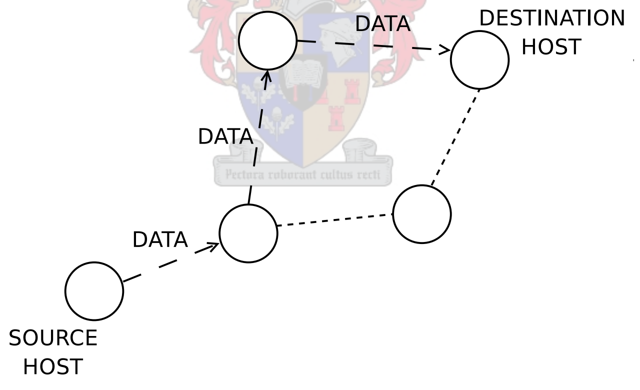


Figure 4.4: *Data uses discovered route*

Route Maintenance

In case a host is not capable of sending a packet to a destination (due to a broken link), it responds by reverse unicasting a route error (RERR) packet to inform the route's preceding hosts of the failure. In **AODV** every host that receives a `RERR`-packet removes the entry out of its distance vector table.

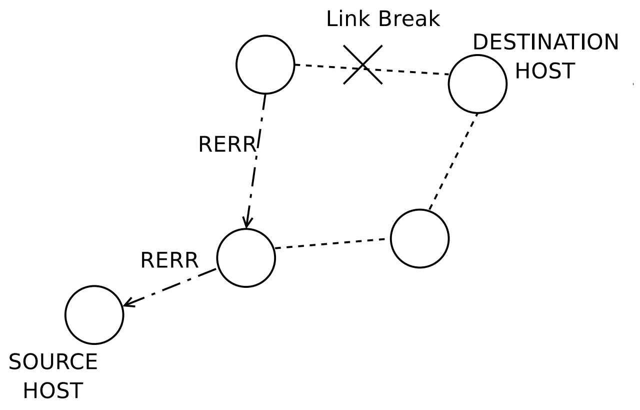


Figure 4.5: *Representation of route maintenance*

AODV and DSR both share similar aspects that make them "On-demand" routing protocols. Next *Dynamic Source Routing* (DSR) will be described in relation to AODV.

4.3.3 Dynamic Source Routing (DSR)

This protocol was described in [10]. Source routing can be seen as a subset of link state routing. The main difference is the way in which link-state information is determined and distributed to other hosts. Firstly the *Route Discovery* phase takes place to determine a route to a required host. This involves the flooding of a `RREQ`-packet throughout the network. The `RREQ`-packet stores the address of each node visited in a route record inside the `RREQ`-packet header. Hosts that overhear these `RREQ`-packets may be able to include route information from these packets into their own route cache. When the target host receives a `RREQ`-packet, it also adds the accumulated route to its route cache, but continues by unicasting a `RREP`-packet back to the source of the RREQ transmission along the path that the RREQ traveled. When the source host receives a `RREP`-packet, it then responds by extracting the source route from its header and creating a data packet that has been equipped with this newly acquired source route information. When a link fails, a `RERR`-packet will propagate (reverse unicasted) through the network in order to inform the source host of the error. The source host will then respond by removing this link from its route cache. The main difference between this and the previous protocol is that *multiple* routes to the host may still exist, making new route discoveries unnecessary.

4.4 Summary

DSR was chosen as the Network Layer protocol on which this work would be based due to the following reasons:

- Multiple RREQ-packets that are received by a target (destination) host from a specific source can be responded to various times. Through employing this method, many routes can be determined by the source. AODV would, for example, only respond to the first RREQ-packet and ignore all other RREQ-packets coming from a source. In DSR this feature can also be amplified by adding *promiscuous listening*. This concept refers to the overhearing of packets by hosts for whom the packet was not intended. It could use this feature in order to extract route information and possibly avoid sending a route request itself. Therefore, this method reuses otherwise wasted packet transmissions.

- According to Perkins [11], DSR outperforms AODV in cases where network size, traffic load and/or mobility of the hosts are low. The reason DSR struggles in more stringent network conditions is due to the protocol's use of route caching and the fact that there is no route "aging" system in DSR as there is in AODV. Due to the characteristics of this rural ad hoc network application such as *low number of nodes* and the *stationary nature* of the nodes, DSR seems like the best choice to base the Network Protocol of this thesis on.

- DSR has been proven to generate less routing overhead. Various routes are maintained in a DSR route cache. This means that if one route expires, other routes will allow packets to still be forwarded without the need for yet another route discovery.

- DSR does not require *any* periodic messaging, such as AODV's HELLO-packets. Any periodic transmission between hosts will result in a waste of bandwidth, which is very limited in this application.

DSR can be classified as a reactive protocol that follows the LORA-approach to route maintenance and usage. When many various routes exist to a destination, the protocol does not need to know which of these offers the most optimum path. This means that when the network changes, DSR only reacts when the packet can no longer find an alternate path to reach a destination. The next chapter will deal with the modified version of DSR designed for use in a rural ad hoc network context. Modifications will include:

- an application of lottery scheduling as a route selection method in order to prevent route congestion

- an implicit route error recovery method that will avoid superfluous route error messages that may consume precious bandwidth

- a protocol that applies the concepts of a DSR-like network protocol in order to determine which hosts are satellite up/downlink hosts, as well as how they can be reached.

Chapter 5

The Network Layer: Protocol Description

5.1 Introduction

To make the Network Layer Protocol viable to deal with the various constraints given in Chapter 1, modifications needed to be made to the standard DSR framework. In Chapter 3 MAC Layer modifications such as *promiscuous listening* and *implicit route error recovery* was referred to but not explained in context of the MAC Layer. This was done due to the fact that these modifications are more important to Network Layer functionality even though these are MAC Layer modifications. The following chapter will explain how these factors are used in order to decrease routing overhead. Some other modifications will be explained:

- *Subsection 5.1.1: Lottery Route Selection* – where a chosen metric for a route decides what the chances are of it being selected for packet forwarding.
- *Subsection 5.1.2: Implicit Route Error Recovery* – no explicit route error message will be broadcast. After link failures, Network Layer packets are readmitted to the Network Layer and retransmitted using different routes.

Here follows a description of the Network Layer's initialization and packet handling functions using the rural ad hoc network context as an example. Initialization of a host will always involve the search for a satellite up/downlink host. Even if this step is unsuccessful, any Network Layer packets that passes a host will allow that host to know what other hosts exist.

5.1.1 Lottery Route Selection

Data packets will be sent using various routes (not just the shortest). Using only the shortest routes will cause routes to become unnecessarily congested. To aid in the spreading of packets over various routes, a route selection algorithm should be used. Assuming route discovery has been done in the past and various routes to the destination have been established, this protocol will select routes via an algorithm based upon *Lottery Scheduling* [1]. Originally this was used to schedule processes in operating systems. This variation of the algorithm basically assigns each route an *inverse* lottery ticket value based upon the route length. The larger the value becomes, the smaller the chance of the route being selected for forwarding a data packet. This implementation of the selection algorithm uses the route length (or hop metric) as the basis upon which route selection probabilities are assigned. This does not imply that another metric could not be used, such as link quality. The latter, however, is more difficult to determine and is bound to be dynamic.

Lottery Scheduling Algorithm

Original purpose: Lottery Scheduling [1] is a randomized resource allocation method used by some operating systems to schedule processes. Rights to a resource is represented as an amount of "lottery tickets". Every process that is competing for a system resource is then assigned a certain number of lottery tickets. This number is decided through the process priority or the popularity of the resource.

Description of the algorithm: Figure 5.1 shows an example of one lottery scheduling situation. A total of 30 lottery tickets are assigned to 4 processes. A random number is selected through a random number generator, that has an uniform distribution over the range [0..30). In this, the draw resulted in the number 25. The algorithm then iterates through processes, cumulatively summing the lottery ticket values. This iteration terminates when the summation reaches a value that is larger than the random number. The process at which this summation stops will then be assigned the resource.

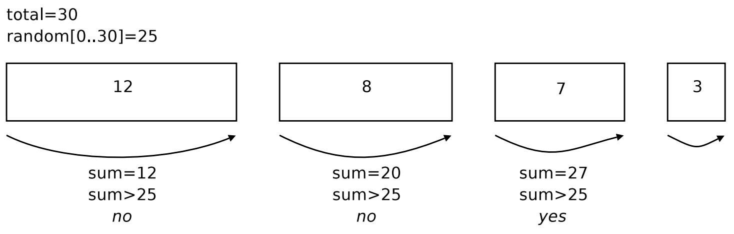


Figure 5.1: *Lottery selection example (Taken from reference [1])*

How the algorithm is applied to the Network Layer: The protocol uses the Lottery Selection Algorithm at the arrival of a `DATA`-packet. Firstly, the protocol determines how many paths can be taken to the `DATA`-packet's required destination. The total probability for route selection is spread out amongst the routes. This means that some routes receives different probabilities (represented by "lottery ticket quantities") in the route selection process. The quantity of "tickets" that a route will receive in order to "win" a `DATA`-packet is inversely proportional to the amount of nodes in a route (the *route length*). The weight can also be squared or cubed in order for the route selection process to be more exaggerated. This means that short routes will have an even higher chance of being selected by a packet and long routes have an even smaller chance of being selected. This implementation uses the inverse route length as the metric for which route bias probability is determined.

5.1.2 Implicit Route Error Recovery

Due to the low speed radios used for the Physical Layer, it was decided that route errors should not be dealt with explicitly. This would cause the network overhead to become too high. The approach this protocol will take will be to evaluate the success of a route for each hop. If a data packet is transmitted and no acknowledgement was received, then the data packet is readmitted to the Network Layer. This will allow the Network Layer to select another route in the route cache using the Lottery Route Selection method described in Section 5.1.1. This implies that a packet does not have to follow the route that was embedded into it. This also allows a path of a packet to be changed dynamically.

5.2 Modified DSR Network Protocol Behaviour

In this section the basic behaviour of the network/routing protocol will be described. This section will also show how this protocol differs from normal DSR-routing. Even though the protocol is based on the concept of DSR, it has been designed in order to determine satellite up/downlink hosts at start-up. The use of *promiscuous listening* allows hosts to use any routing information that may come into wireless contact with them. This implies that a host's MAC Layer should send packets up to the Network Layer, regardless if the packet was meant for that host's MAC address or not.

5.2.1 Initialization of a host

At start-up a node firstly acquires any information on satellite up/downlink hosts. This can be seen as *proactive* behaviour. This is accomplished by sending a `RREQ`-packet with no specific address to discover any possible satellite up/downlink hosts. Hosts that receive these requests will first extract route information and then check whether or not they are equipped with satellite up/downlink hardware. That, in combination with promiscuous listening, can allow a large amount of the network hosts to be informed of their neighbours in the network. Hosts that do have satellite up/downlink capability will respond with a regular `RREP`-packet response.

5.2.2 Handling Behaviour for Application Layer Packets

Packets arriving at the Network Layer from the Application Layer is dealt with using the algorithm of Figure 5.2

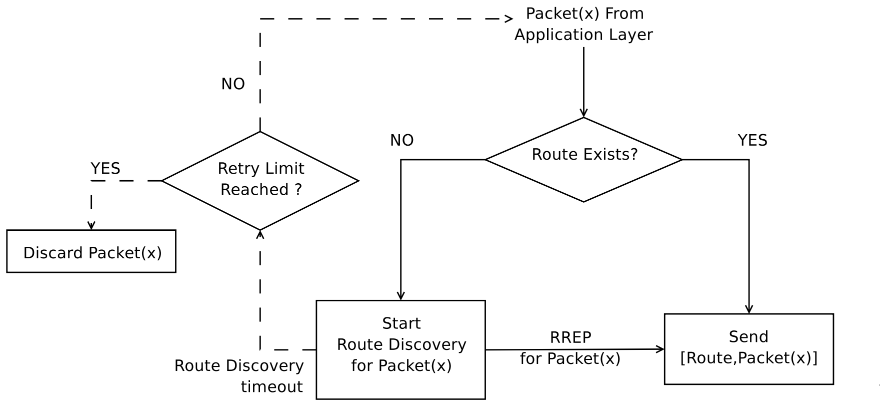


Figure 5.2: *Application Packet handler Flowchart*

When the packet arrives from the Application Layer for transmission, it causes the Network Layer to check its *route cache* for a route to the required destination address. Routes are chosen from the route caches by means of the lottery route selection method described previously in Section 5.1.1. If a route does not exist *Route Discovery* is done. If this specific route discovery is unsuccessful, then a timeout will occur, causing the Network Layer to retry the route discovery process. If a route is determined then the Network Layer will receive a `RREP`-packet with a corresponding destination host. A transmission can then occur of a packet that consists of the newly acquired route and the Application Layer packets' contents. Route discovery retries are limited to a certain amount.

5.2.3 Handling Behaviour for Routing Layer Packets

Packets arriving at the Network Layer from the MAC Layer are categorised as one of the following packet classes:

- `RREQ`-packets
- `RREP`-packets
- `DATA`-packets
- Packets that cannot be forwarded due to non-responsive hosts

The RREQ-packet case: When the Network Layer receives and identifies that a packet is a `RREQ`-packet, the handler firstly extracts the current route from the packet header and

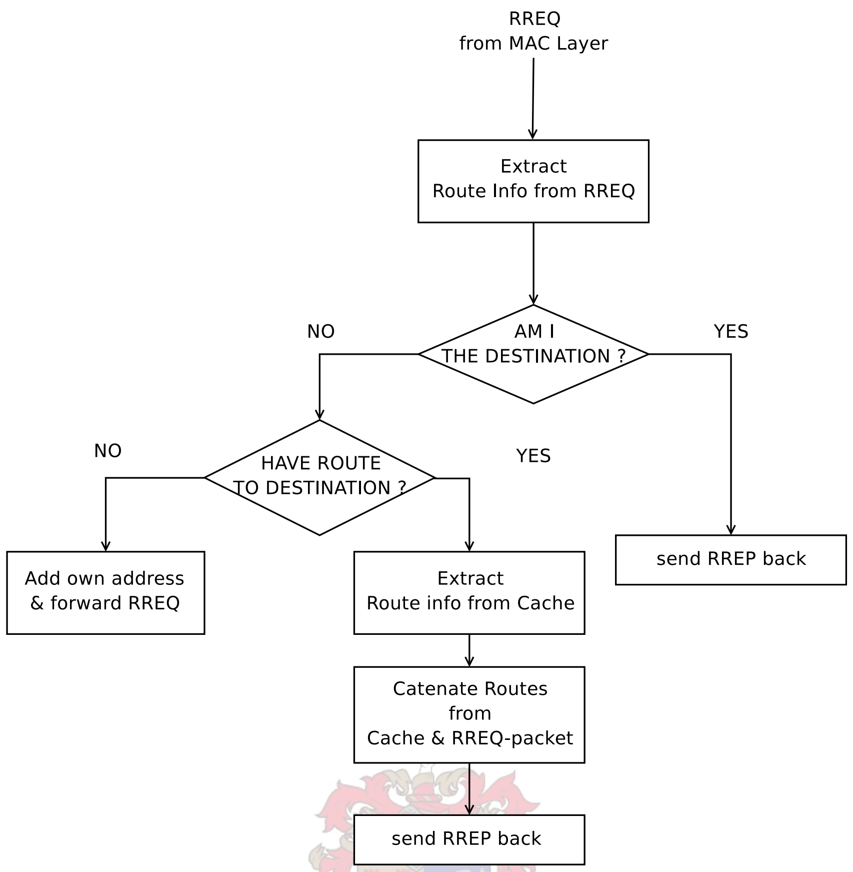


Figure 5.3: *RREQ-packet handler Flowchart*

stores this information in its route cache. This is done with all packets and allows a host to learn of routes from overheard packets, due to promiscuous listening.

The host then decides whether or not the `RREQ`-packet is meant for it. If it is, then a `RREP`-packet is sent (with the newly determined route) in order to notify the source of the route discovery of what route was taken to reach it. If the `RREQ`-packet was **not** meant for this host, then it must be determined if the that host possesses a route to the destination. In the case that it does have a route in its route cache that would be a path to the destination, the response would be a `RREP`-packet. The route that is stored in the new `RREP`-packet's header is the catenated result of the route cache and the `RREQ`-packet that is currently under scrutiny.

The RREP-packet case: The Network Layer would handle a `RREP`-packet using the algorithm represented by the flowchart of Figure 5.4. As in the `RREQ`-packet case, the Network Layer extracts route information from the `RREP`-packet's header before doing any other operation on the packet. This route information is stored in the route cache for later usage. The host then decides whether or not the packet is destined for it or for another host. In case the host is not the destination of the route reply, then the host will assume the packet was overheard and discard it.

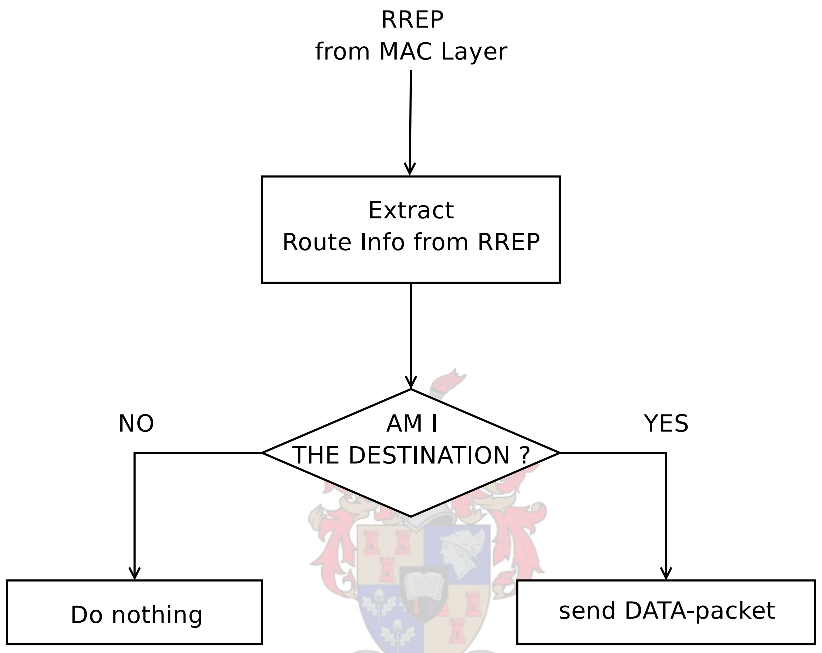


Figure 5.4: *RREP-packet handler Flowchart*

The DATA-packet case: Packets identified as `DATA`-packets are handled similarly to `RREP`-packets. A host also extracts routing information for its own route cache before deciding how to deal with the packet first. This is then also followed by comparing the packet's destination with the host's own network address. If the packet is meant for this host then it will be sent up to the Application Layer. Packets that are not meant for this host should be deleted by *honest* nodes. This does, however, raises a question regarding security, but this is not within the scope of this thesis and was therefore not considered to be an important design consideration.

5.2.4 Forwarding Behaviour

Hosts forwarding or sending `DATA`-packets uses as many different routes to a destination as available. This approach allows *multipath routing* to take place. Later, simulation will confirm this behaviour.

5.3 Summary

This chapter has described a Network Layer that has been based on the Dynamic Source Routing (DSR) Protocol. It discovers routes at initialization to a satellite up/downlink host. This *proactive* initial route discovery phase has a dual purpose.

- Through the use of promiscuous listening other hosts will be informed of each other.
- A route to a satellite up/downlink host will, hopefully, be discovered.

In the event that a satellite up/downlink host does not exist, then the `RREQ`-packet's time-to-live counter will let it expire. The Network Layer will reschedule satellite gateway discovery for a later stage or wait until it overhears a route to a satellite up/downlink host. This protocol attempts to decrease overhead incurred, by using these techniques. Reuse of packets allows proactive routing in normal (non-initial) network behaviour to be avoided. The protocol can thus be described by the following characteristics:

- *Reactive* – The Network Layer only does route discovery if it does not have a route to the destination. The only proactive action that the Network Layer takes is at initialization in order to find the satellite up/downlink host(s).
- *Multipath Routing* – Messages can be split up into packets and sent via various paths to the same destination. The way in which these packets are distributed are determined using the *Lottery Route Selection Algorithm*.
- *Least Overhead Routing Algorithm (LORA)* – If a route fails to a destination but one or more other routes to that destination exists, no special action will be taken, due to the fact that overhead would be decreased if route discoveries are avoided.

Chapter 6

Simulation And Results

The following chapter will deal with the actual test simulations of the protocol using various test scenarios. The parameters that are evaluated will be throughput, latency and efficiency. Each section will explain how the values were calculated and results will be displayed and interpreted. In this chapter the following will be covered:

- The process of elimination of simulation software that was followed.
- The background of simulation using OMNeT++ will be explained.
- A method for modelling route discovery by embedding timing events to the `RREQ`- and `RREP`-packets.
- Simulation of various test scenarios will be shown, as well as interpretation of the results.
- As a result of the previous point a theoretical basis have been established for ad hoc network design. This, in conjunction with the simulation models and scripts, can collectively be seen as a tool for modelling ad hoc networks.

6.1 Simulation Software

Various simulation software packages were considered, such as Network Simulator [12], JiST/SWANS [13] as well as a few other closed source alternatives. The following will shortly summarize all the simulation packages considered as well as reasons for the final choice.

- *Network Simulator* [12] (otherwise known as *NS*) has been developed by various contributors (see [14]). It boasts with a wide community of support and developers, as well as, a large amount of simulation models written for, or ported to it. The software is based on C++ with OTcl as a scripting and simulation setup language. This simulator was not chosen due to the high grade of difficulty it poses to users that are new to any form of simulation software.

- *SimPy* [15] is a Python-based simulation framework that allows fast development of small simulation systems. It is very simple to learn due to its use of Python as its base language. This simulation framework was, however, meant for a wide variety of simulation situations. This lack of specialization causes the development of network-specific simulations to be slow. The medium, the nodes and the protocols would have to be implemented from scratch.

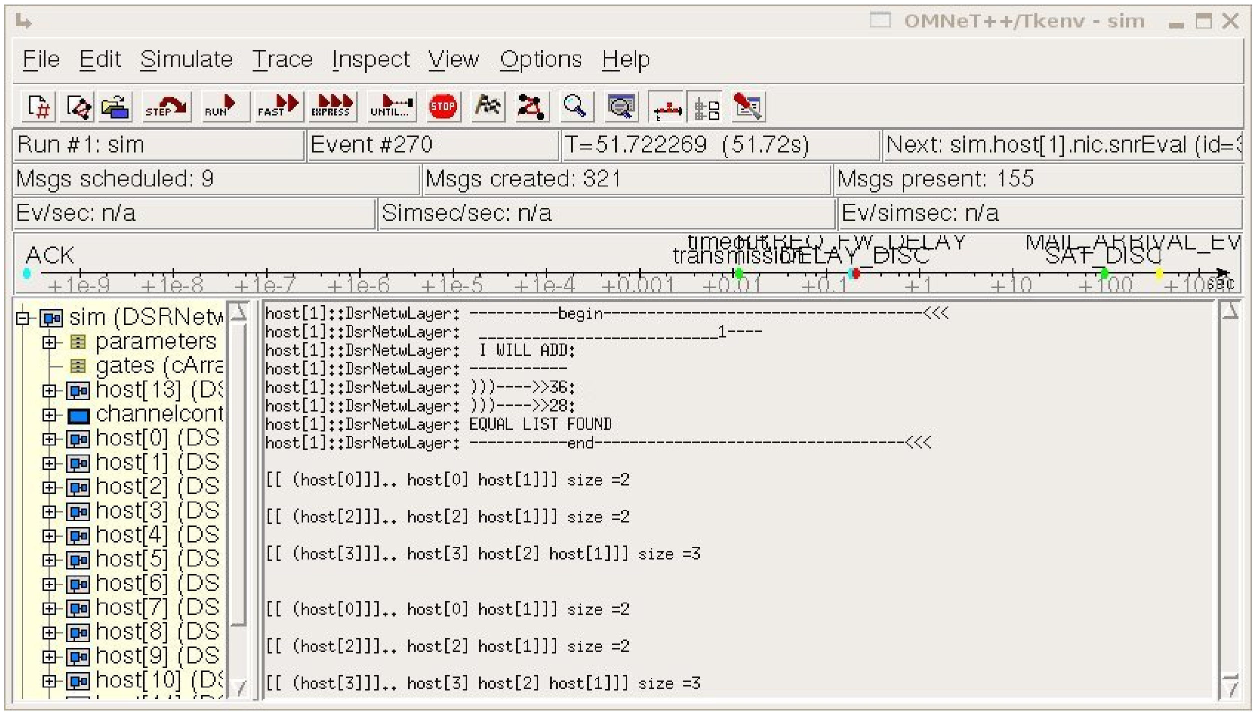
- *JiST/SWANS* [13] or Java in Simulation Time/Scalable Wireless Ad hoc Network Simulator, was developed by Rimon Barr at Cornell University. This simulation software uses various interesting concepts in order to accomplish simulation. The most interesting characteristic of this simulator is that it is Java-based. Normal Java-based simulations would be too slow and resource inefficient. This simulation framework recompiles the simulation model's byte-code and modifies it to interact more efficiently with the hardware platform. It also boasts various ad hoc networking models. The simulator was eventually not chosen due to a lack of support.

- *OMNeT++/OMNEST* [16] is a simulator initially developed by Andras Varga. This simulation package is based on C++. It does not have as many ad hoc network specific models written for it, but it is a fast and simple simulator that has various basic radio models and libraries to its disposal. It also allows one the option between visualizing the network with graphics for single simulation runs or outputting results for various simulation runs while in a console based environment. It only requires simple scripting in order to allow various simulation runs to be accomplished in the console based environment. For these reasons, it was decided to choose OMNeT++ for the ad hoc network simulation for this thesis.

6.2 Simulation Strategy

6.2.1 Background on OMNeT++ Simulation Modelling

OMNeT++ [16] is an open-source object-oriented discrete event simulator software package. It is written in `C++` and implements a custom high-level network modelling language that allows users to reuse models previously implemented in `C++`. The following section will describe the basic structure of the event-queue as well as how it was used and exploited to implement a DSR-like routing protocol model.



Run #1: sim | Event #270 | T=51.722269 (51.72s) | Next: sim.host[1].nic.snrEval (id=

Msgs scheduled: 9 | Msgs created: 321 | Msgs present: 155

Ev/sec: n/a | Simsec/sec: n/a | Ev/simsec: n/a

```
host[1]::DsrNetwLayer: ----------begin----------------------------------------<<<
host[1]::DsrNetwLayer:  ___________________________1----
host[1]::DsrNetwLayer:  I WILL ADD:
host[1]::DsrNetwLayer: -----------
host[1]::DsrNetwLayer: )))---->>36:
host[1]::DsrNetwLayer: )))---->>28:
host[1]::DsrNetwLayer: EQUAL LIST FOUND
host[1]::DsrNetwLayer: -----------end-------------------------------------<<<

[[ (host[0]]].. host[0] host[1]]] size =2

[[ (host[2]]].. host[2] host[1]]] size =2

[[ (host[3]]].. host[3] host[2] host[1]]] size =3


[[ (host[0]]].. host[0] host[1]]] size =2

[[ (host[2]]].. host[2] host[1]]] size =2

[[ (host[3]]].. host[3] host[2] host[1]]] size =3
```

Figure 6.1: *OMNeT++ Graphical Window*

6.2.2 Discrete Event Simulation

The basic algorithm that discrete event simulators follow to execute events are as follows (taken from [16]):

```
initialize: construct model & insert initial events to FES
while (FES not empty and simulation not yet complete)
    {
         retrieve first event from FES
         t:= timestamp of this event
         process event
         (processing may insert new events in FES or delete existing ones)
    }
finish simulation (write statistical results, etc.)
```

Initialization The initialization phase calls all initial object constructors. This implies that objects modelling protocols and hardware, as well as objects that represent initial messages, events and packets are created. In this phase initial network connections are also constructed. Construction of network connections are governed by Formula 6.3 (discussed later).

The Event loop This program construct moves through the FES (Future Event Set) or event queue and executes each message in the sequence that they were scheduled. The FES is similar to a priority queue, due to the fact that a `cMessage`-event with an earlier scheduled simulation time is acted on before an event with a later simulation time. During this process information about the simulation should be logged and recorded to determine results from the simulation.

Finishing the simulation If the simulation's event queue (or FES) is empty, the simulation exits the event loop. In this section object destructors must be called and the necessary object clean-up operations done.

6.2.3 The cMessage Class

This class (and subclasses thereof) is used in OMNeT++ to represent events, packets and timers. These applications are implemented as follows:

- `Event` or `Timer` – When an event needs to occur at a specific time, a `cMessage`-object is placed in the FES with a specific *arrival time*. When the event loop iterates through the FES and reaches this event, it is removed and an user-defined event-handler takes over in order to interpret the message.

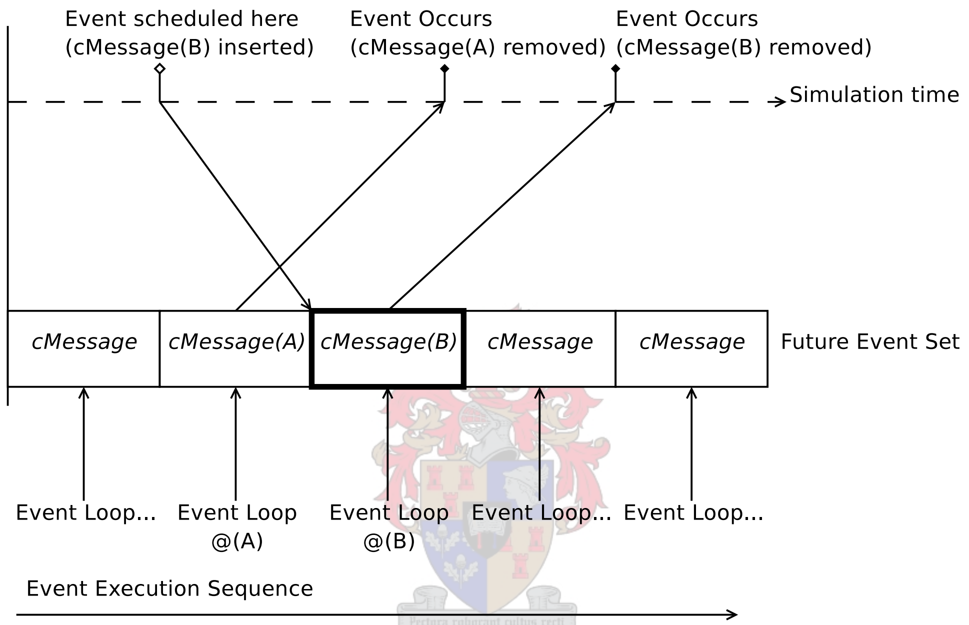


Figure 6.2: *Mapping between the FES, Simulated and Real time*

In Figure 6.2 simulation time and the FES are represented. `cMessage(A)` and `cMessage(B)` are marked. Note that even though the events these `cMessage`s represent occurs after each other, there is no need for the events to be scheduled in the same sequence. This means that `cMessage(A)` could have been scheduled *after* `cMessage(B)`. In Figure 6.2, only `cMessage(B)`'s movements are completely shown. Note also that when a `cMessage` is **inserted** into the FES, it models an event that is being **scheduled**. When such a `cMessage` leaves the FES, it represents that the simulation time has reached the timestamp of the `cMessage` and the scheduled event has **occurred**. In this case, the `cMessage` instance is then handled by an event handler. If such a `cMessage` instance is removed before its simulation time was reached then the event was **cancelled**.

- `Packet` – A packet class is usually subclassed from a `cMessage`-class to include extra information, such as source and destination addresses, time-to-live and packet size. Meta-data, such as overhead incurred or the current age of a packet can also be added. This can be done without changing the simulated size of the packet, due to the fact that the size value is independent of actual packet-object contents. Packets are usually moved between protocol layers and encapsulated or decapsulated with layer-specific headers. Packets can also move between hosts and node objects by means of channel objects.

The context pointer: The `cMessage`-class is also equipped with a general purpose pointer that allows timers and events to be bound to other messages or packet-objects. One usage of such a pointer is to bind a timer-message to the exploits of a certain packet. This usage will be explained further in Section 6.2.4.

Other cMessage attributes:

There are a few other class attributes that are worth mentioning:

- `creationTime` – Attribute that contains the time at which the object was constructed
- `sendingTime` – Attribute that contains at what time the object was entered into the Future Event Set.
- `arrivalTime` – Attribute that contains at what time the timer object will leave the Future Event Set.

These attributes are useful for determining how long the timer was active, in case it was canceled before it could expire. Other attributes include a general-purpose `kind`-attribute that allows user-defined message types. When such a timer "arrives" (from the FES) at the host that set it, the host can simply differentiate between messages by only using the `kind`-field.

6.2.4 DSR Route Discovery Modelling by means of the FES and cMessage class

When the Application Layer sends a packet to the Network Layer, it is determined whether or not the Network Layer has a route to the destination in question. A Dynamic Source Routing (DSR) type protocol would require to start route discovery. If the route discovery is unsuccessful, the simulation model timer must expire. This implies that a timer must be kept for each route discovery that a host is waiting for. In order to implement various timers for various route discoveries, the context pointer of the new `RREQ`-packet is set to a new timer object (`cMessage`-object). The `RREQ`-packet's context pointer is thus taking "ownership" of a timer. This timer will therefore only refer to one route discovery.

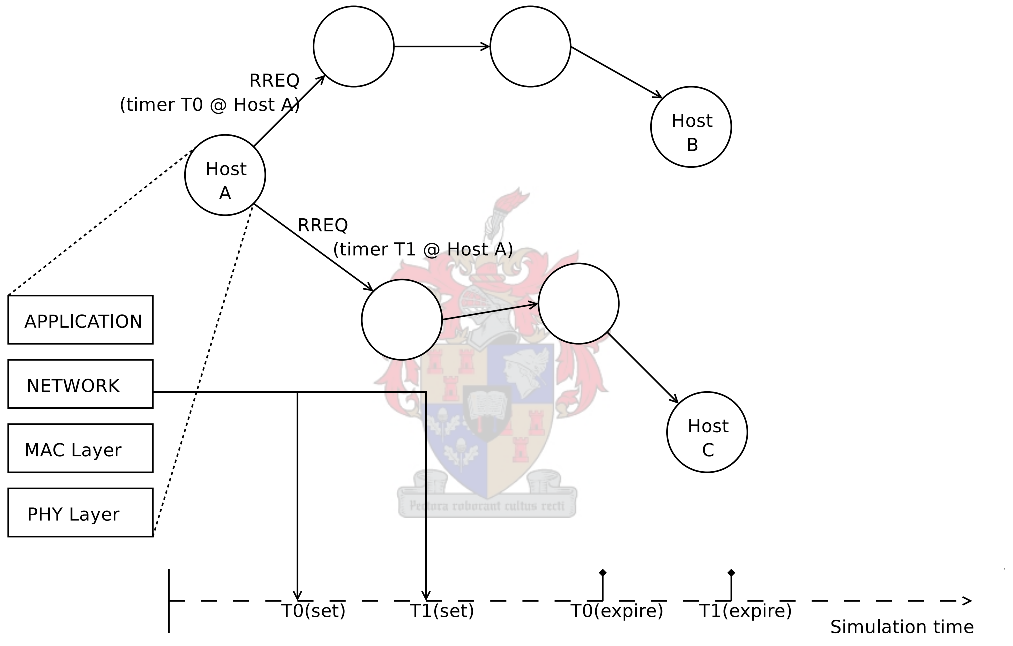


Figure 6.3: *RREQ-packets point to timers at source*

In Figure 6.3 the route discovery timer objects (`Timer T0` and `Timer T1`) were created in order for the Network Layer to remember which route discovery attempt has expired and which was still active. `Timer T0` is set to be active from `T0(set)` and `T0(expire)`. After `T0(expire)`, the Network Layer considers the route discovery to be a failure and a new attempt (with a new timer) is issued. The same holds true for `Timer T1`. Note that `Timer T0` and `Timer T1` are completely independent of each other.

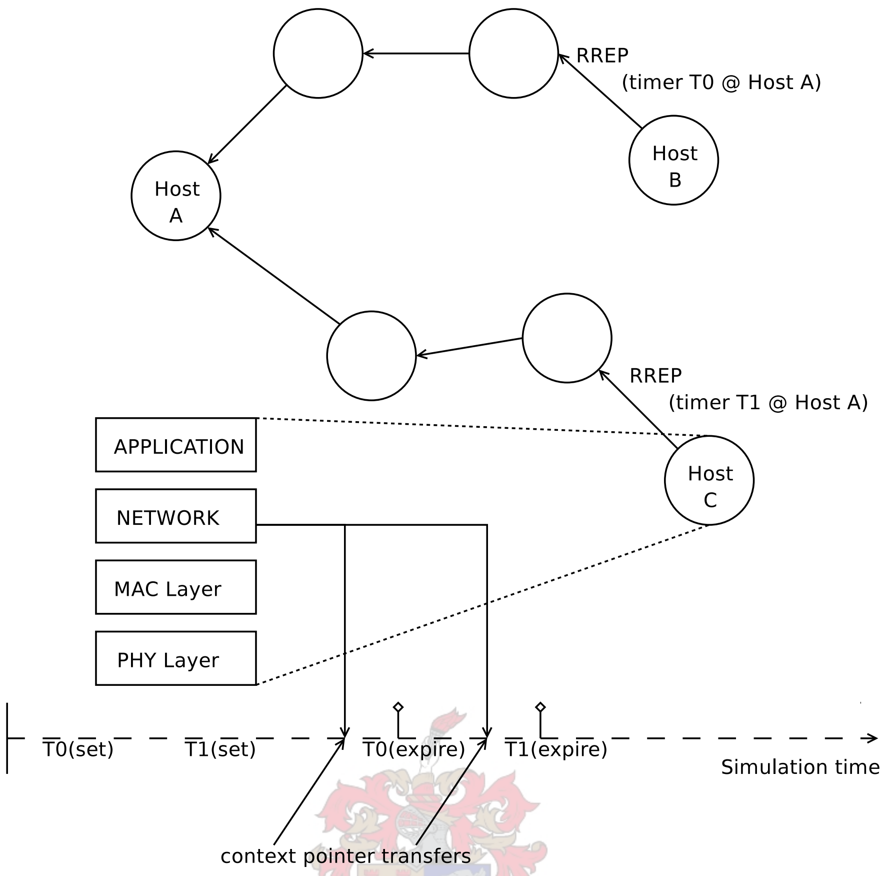


Figure 6.4: *RREP-packets point back to timers at source*

In Figure 6.4 the `RREQ`-packets have reached the Network Layer of the destination host. The context pointers contain memory addresses of that specific route discovery timer object (`cMessage`). The address contained by the context pointer is transferred from these `RREQ`-packets to the newly created `RREP`-packets. *This means that the context pointers are **still** referring to the timers at the source.* These `RREP`-packets will then be unicasted back to the source host. The source host must then know which route discovery timer to cancel. Timer identification can then be done by comparing all **its own** active timers in the FES with the timer that is referred to by the `RREP`-packet's context pointer.

This method allows multiple timers to be implemented by using only the Future Event Set and cMessage-objects. No extra data structures will then be required to maintain the mapping between timers and route discoveries.

6.2.5 Simulation Structure

Hosts were designed to have the following basic structure:

- `ApplLayer` – Models the Application Layer (a short mail message client)
- `NetwLayer` – Models the network and routing layer as described in Chapter 5
- `MacLayer` – Models the medium access layer as described in Chapter 3
- `Physical Layer` – Models the radio and the dynamic connections between hosts

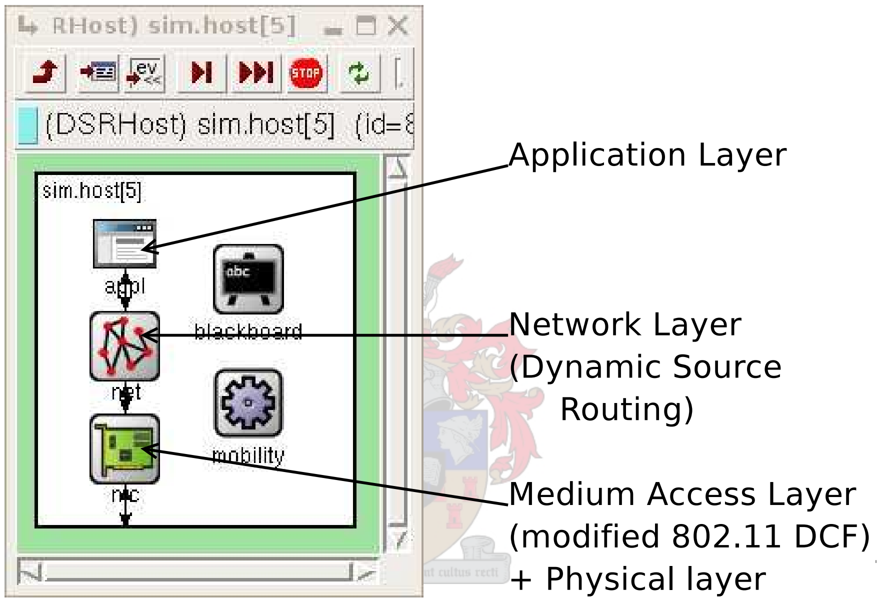

RHost) sim.host[5]
(DSRHost) sim.host[5] (id=8
sim.host[5]
appl
blackboard
net
mobility
nic
Application Layer
Network Layer (Dynamic Source Routing)
Medium Access Layer (modified 802.11 DCF) + Physical layer

Figure 6.5: *Screenshot of host[5]*

The main implemented simulation models will be described.

ApplLayer model

The Application Layer Model was designed to give a host an identity as a peer host or a host with satellite up/downlink capability (or a gateway host). To specify host behaviour, the following parameters must be specified:

- `activationAt` – time at which the host starts its function
- `dieAtTime` – a parameter that can enable random failure or failure at a specified time.
- `isSatHost` – a parameter that sets the host's satellite gateway host status.
- `isSource` – this parameter allows a host to get periodic mail receptions from its Application Layer.

- `mailArrivalInterval` – this parameter is used to specify the mean arrival rate of every mail message.
- `constMailSize` – a parameter that specifies the constant amount of packets that a message can consist of.

Hosts can be set to behave as *satellite gateway hosts* or *normal peer hosts*. Both classes can also be set to an active or inactive state, as well as, source hosts or non-source hosts. This allows simulation hosts to either act as active packet generators, packet sinks or simply forwarding hosts.

NetwLayer Model

The parameters used for the Network Layer is stated below:

- `defaultTtl` – default time-to-live (ttl) used for this module.
- `resendRREQtime` – time it takes before the Network Layer sends another route request.
- `addressLength` – used to accurately simulate packets with attached route records.
- `numberReTX` – the number of times the Network Layer will resend a packet before discarding it. Every time a network packet is retransmitted a route is selected for it by using the Lottery Route Selection Algorithm.
- `weightPower` – used in Lottery Route Selection. A larger weight implies that popular routes will become more popular and less popular routes are used less.
- `rreqAttempts` – specifies how many times a route discovery must take place before the host gives up.

MacLayer Model

The Medium Access Layer Model was based on the 802.11 simulation model that was distributed with the Mobility Framework add-on for OMNeT++. It was changed to take a maximum wait distance as a parameter in order to wait a reasonable amount of time for transmissions to propagate between hosts. This parameter (named *maxNodeSeperationDistance*) as well as the others will now be described:

- `maxQueueSize` – this specifies a FIFO buffer size for waiting packets.
- `broadcastBackoff` – this parameter determines the duration of the backoff window when a message is broadcast.
- `maxNodeSeperationDistance` – the distance (in meters) of how far nodes will be (maximally) from each other.

Physical Layer Model

The physical layer determines which hosts are connected to each other. This layer model also allows for packets to collide when simultaneous transmission occurs. Packets involved in this event will then be marked by this layer model as collided or corrupted packets. The link quality for all the radio links are determined by calculating the interference distance given by the Equation 6.3 found in [7]:

$$\lambda = \frac{c}{f_{carrier}} \tag{6.1}$$

$$P_{RXpower} = 10.0^{SAT/10.0} \tag{6.2}$$

$$\text{InterferenceDistance} = \left(\frac{\lambda \times P_{max}}{16.0 \times \pi^2 \times P_{RXpower}} \right)^{1/\alpha} \tag{6.3}$$

$$\begin{aligned} \textit{with} \quad c &= \text{speed of light} = \ 3 \times 10^8 m/s \\ \text{SAT} &= \text{Signal Attenuation Threshold (in dBm)} \\ \alpha &= \ \text{path loss coefficient} \\ \lambda &= \text{wave length} \\ P_{max} &= \text{Maximum transmission power} \end{aligned}$$

This value is then compared with the actual physical distance between all host pairs. If the distance between a node pair is less than the value determined by Equation 6.3, a connection is modelled. Otherwise, no connection is made in the simulation model. This code was reused later in order to create a Python script that would represent the network topology.

6.2.6 Representation of Statistics

Network Representation

Network topologies are represented as a graph of vertices and edges, where the dots represent the network hosts and the edges represent the radio link in between radio networks.

Figure 6.6 is an example network topology

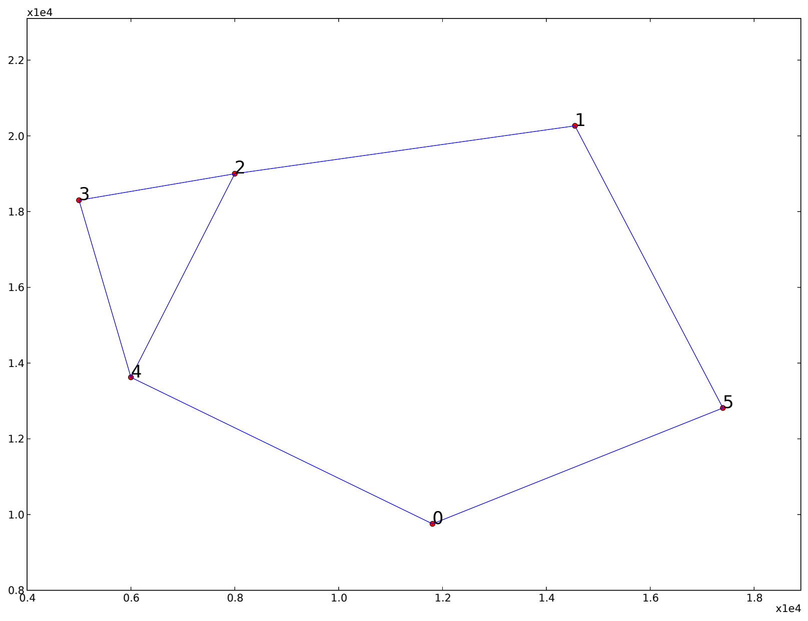


Figure 6.6: *Simple Test Network*

OMNeT++ has a network visualization system that, for short ranges, shows the entire network in a highly user-friendly manner. It was however decided not to be used due to the lack of scaling functionality in cases where the distance between simulated networks nodes became too big. Figure 6.6 was represented by means of a custom made Python-based script using the Matplotlib library. Connectivity of nodes were determined using Formula 6.3.

Route Preference Graph

Packet arrivals can be represented as marks on the time-line as is the case of Figure 6.7 but this does not give a good impression of the routes that are used by arriving packets over a certain interval of time at the destination node.

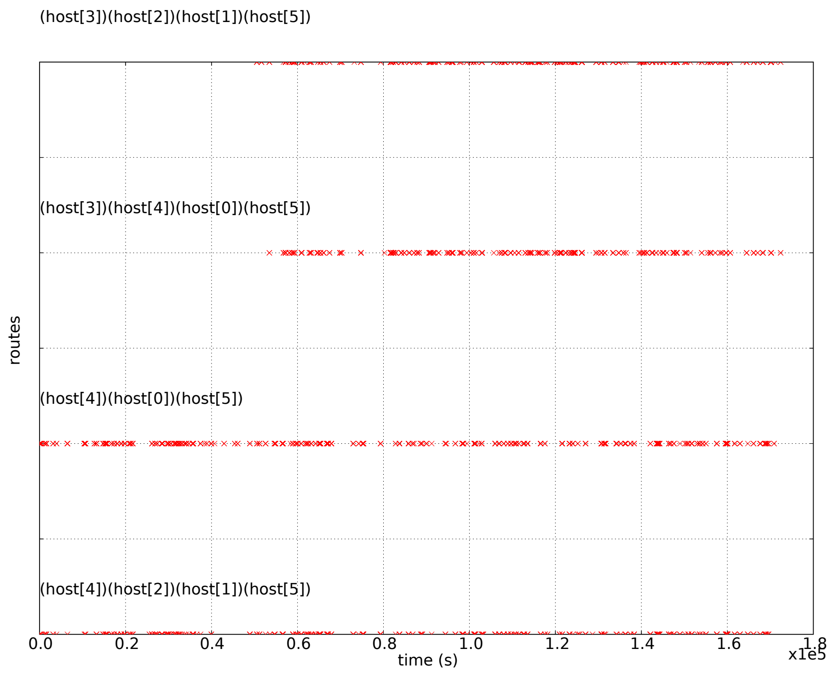


Figure 6.7: *Route arrivals over time as seen from host[5]*

In an attempt to create a more readable representation of packet route "preferences" the graph of Figure 6.8 was rendered. Figure 6.7's data is used and a small interval is taken over which a certain amount of data will be considered. The percentage of packets that used a certain route is registered on the graph in Figure 6.8.

Please note: that more popular routes inhabit *larger areas* on the graph than less popular routes. As previously stated, for this implementation (of the Lottery Route Selection Algorithm) route popularity is inverse proportional to the route hop metric.

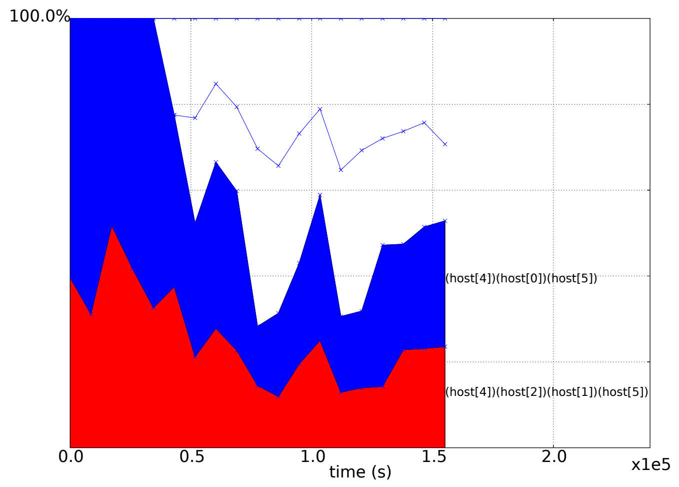


Figure 6.8: *Graph highlighting traffic from host[4] as seen from host[5]*

In Figure 6.8 the routes used by `host[4]`'s traffic are highlighted. From the figure it can easily be seen that `host[4]` is always active. Before 50000s packets originating from `host[4]` makes up for 100 percent of the packets arrivals at `host[5]`.

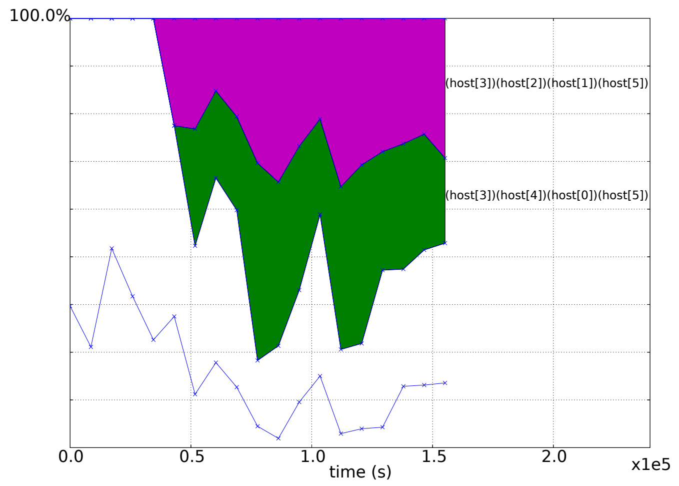


Figure 6.9: *Simple Route Preference Graph (traffic from host[3]) as seen from host[5]*

In Figure 6.9 the routes used by `host[3]`'s traffic are highlighted. It can be seen from the graph that `host[3]` activates at 50000s. A smaller percentage of packets arriving at `host[5]` now originates from `host[4]`, while the rest of the arrivals originates from `host[3]`.

6.3 Host Introduction to the Network

For this simulation the network of Figure 6.10 was used. All hosts are active in the beginning of the simulation, except for `host[7]` which only activates at 50000s. `Host[5]` was designated as the source host and `host[8]` was designated as the destination.

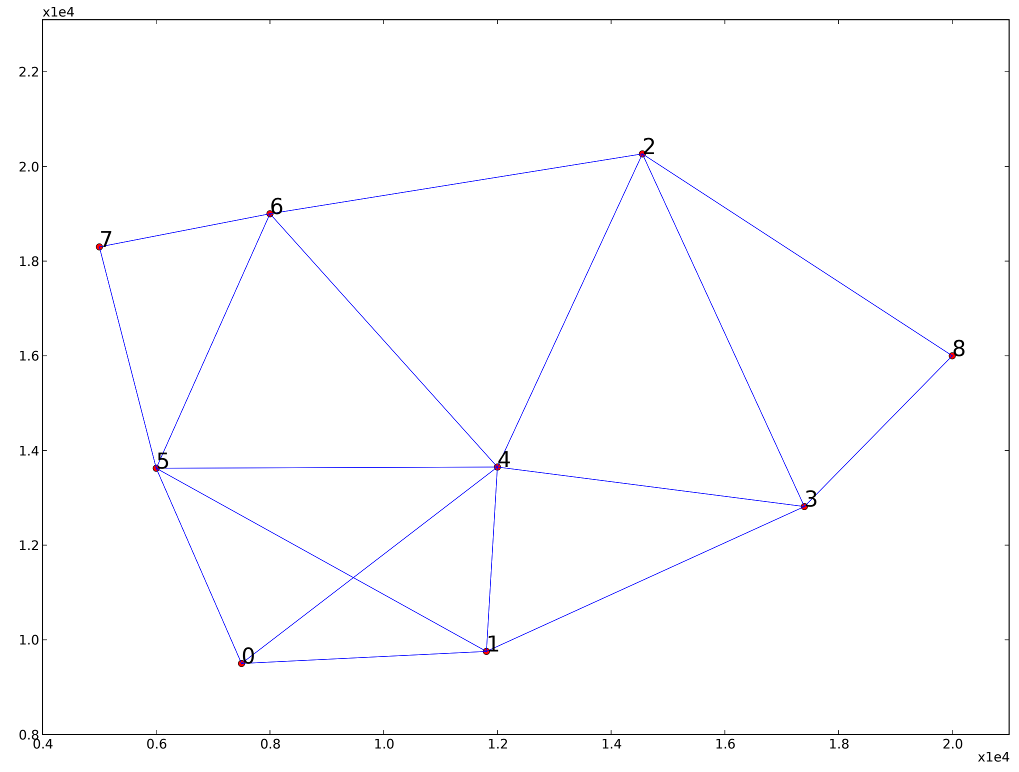


Figure 6.10: *Test Network*

Figures 6.11 and 6.12 gives a graph of what routes were taken to reach the destination host. Figure 6.11 highlights the routes used by packets to move between `host[5]` and `host[8]`.

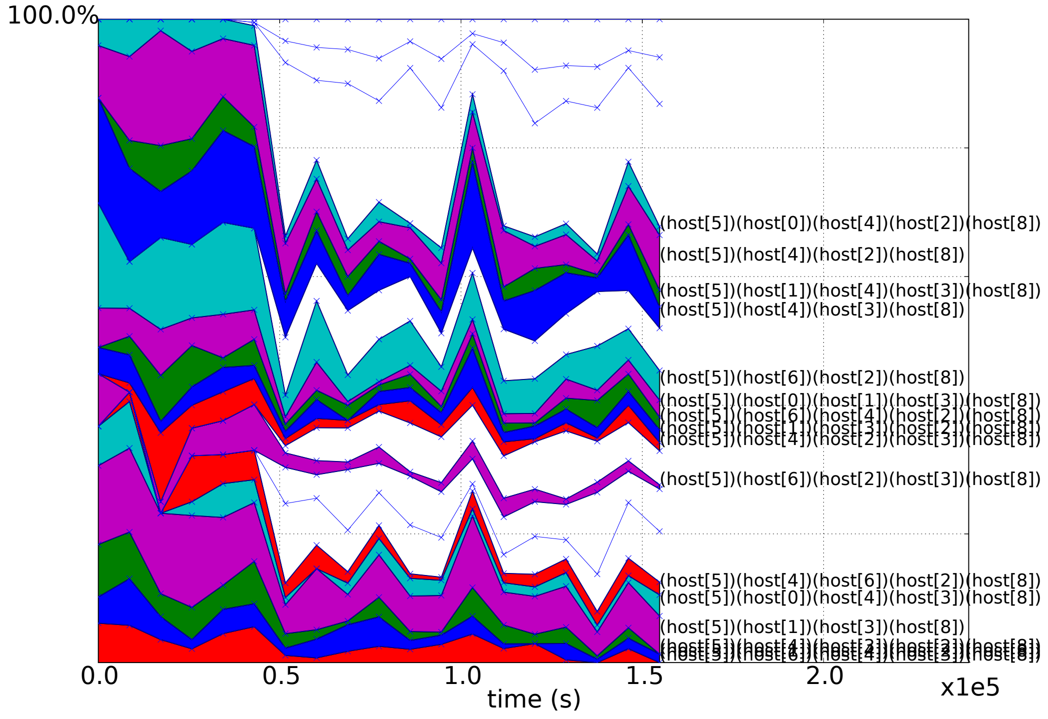


Figure 6.11: *Host[5] packet routes highlighted*

It can be seen on Figures 6.11 and 6.12 that the route change does not appear to take place immediately at 50000s. This can be explained, due to the fact that the graph is generated by checking a subsection (or window) of time and determining what percentage of the packets follow what routes. This process is repeated while moving the window over the total simulation time. The size of the window was modified to make the results more visible, but this causes accuracy to be decreased. Smaller windows would allow for more accurate representations, but the graph would not be as smooth or readable. The larger the window, the more smoothing is done but this decreases accuracy.

It is evident, according to the graph of Figures 6.11 and 6.12, that `host[7]` activates at 50000s. The result is that a smaller portion of the arrivals at `host[8]` can be attributed to `host[5]`.

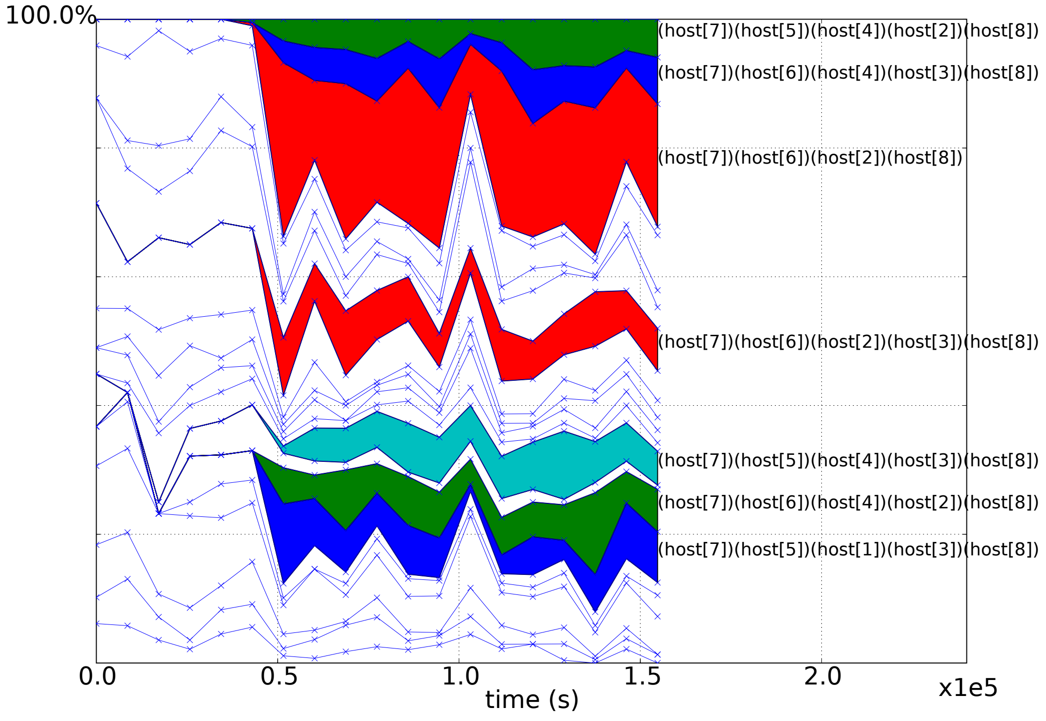


Figure 6.12: *Host[7] joins the network (packet routes highlighted)*

Also note that the surface occupied of each flow represents the popularity of a route. This implies that route `7-6-2-8` is the most popular due to the fact that it is occupies a bigger area on the graph than any other route. *This is due to the shortness of the route.* This representation of diverging traffic flows are, in fact, caused by the Lottery Route Selection Algorithm described previously in Chapter 5.

6.4 Broken Host Route Recovery

The following section will prove that the routing protocol can recover from network errors. This will be done by using the test network of Figure 6.10. `Host[5]` and `host[8]` have been designated as source and destination nodes respectively. `Host[4]` have been set to die at certain times. Simulation runs will be used to demonstrate each case.

6.4.1 Test Case 1: Host[4] dies at 50000s

In Figure 6.13 the highlighted packet route preferences are that of routes that included `host[4]`. `Host[4]` was scheduled to fail at time instant 50000 seconds.

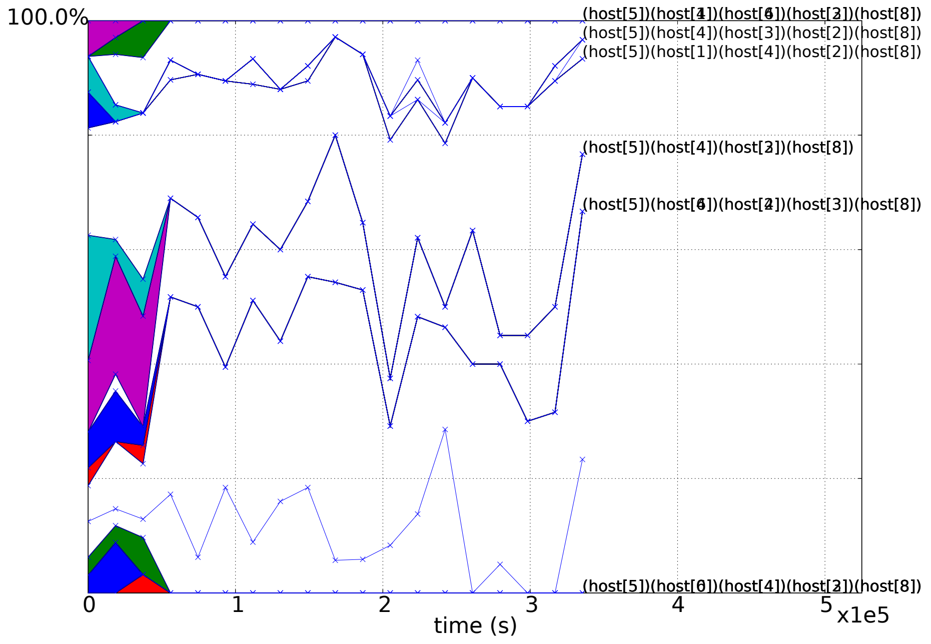


Figure 6.13: *Data Packet arrival rates using different routes (death of host[4] at 50000s)*

The figure clearly shows that route preferences for routes with `host[4]` becomes zero after 50000s in simulation time. The graph does however not stop after this time, but continues until the termination time of the simulation. Due to the Least Overhead Routing Approach (LORA), route discovery is not necessary because useful routes still exist.

6.4.2 Test Case 2: Host[4] dies at 150000s

When the simulation is run again with `host[4]`'s time of death set to 150000s the route preferences will be given by Figure 6.14.

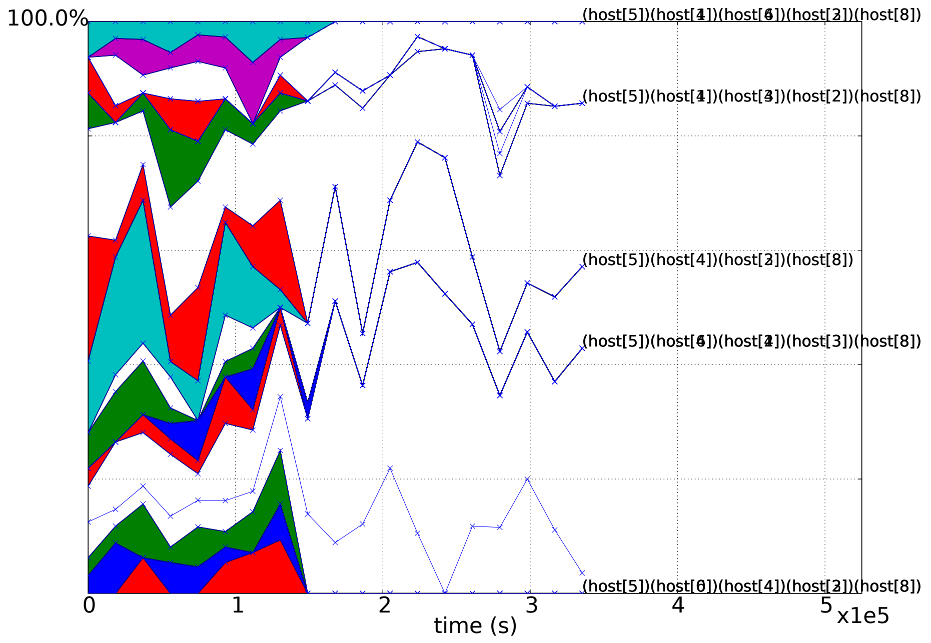


Figure 6.14: *Data Packet arrival rates using different routes (death of host[4] at 150000s)*

Once again, packets from `host[5]` find their way to `host[8]`. These simulations indicate that the routing protocol is, in fact, capable of finding alternate destination routes. A reason for this, is that packets follow alternate paths to the destination and a route failure would only decrease the amount of probable paths that the packets may follow. As described in Chapter 5, forwarding to non-responsive hosts are dealt with by replacing the route header information of the packet and re-attempting the transmission.

6.5 Network Simulation Evaluation via Queueing theory

6.5.1 Scenario Description

The topology of Figure 6.15 was devised to show how the protocol functions after route discovery has been done and routes have been established. `Host[0]` is configured to act as the traffic source host and `host[3]` has been activated as a satellite gateway host. `Host[0]` generates traffic that have exponentially distributed arrival intervals at a rate of λ. The hosts then deal with these packets at a service rate of μ. In this case, a host has completed *servicing* a packet, after the packet has been transferred to the next host.

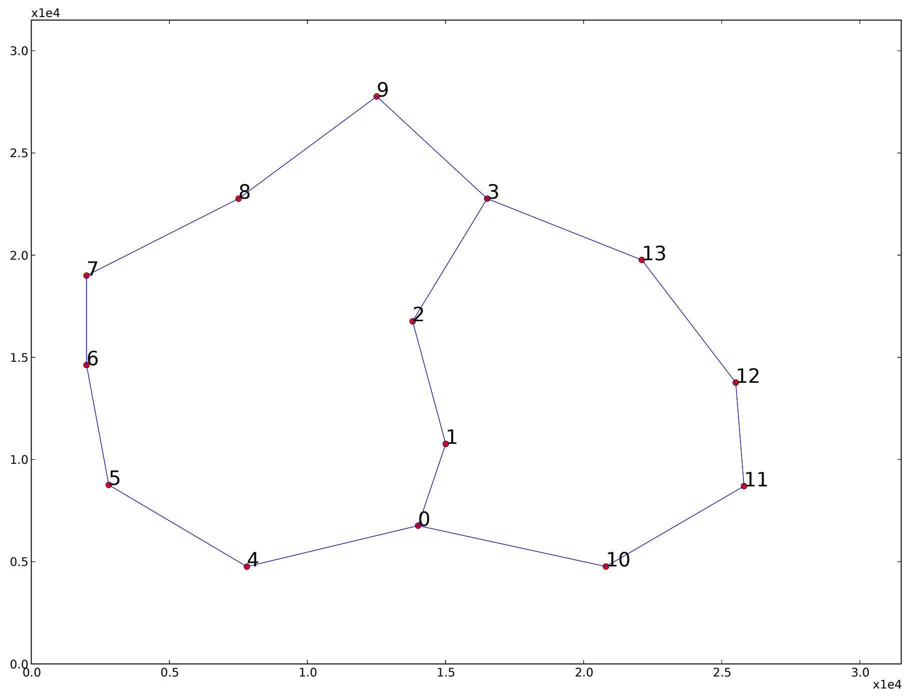


Figure 6.15: *A network with host[0] as source and host[3] as destination*

6.5.2 Theoretical Background

In order to evaluate some of the work of this section, queueing theory needs to be investigated. Figure 6.16 is a simplified representation of a queue-server system. Items (such as clients or packets) reaches the host at a rate λ. The server host services such items at a service rate of μ. Note that the service rate does not change the rate at which the items leave the host.

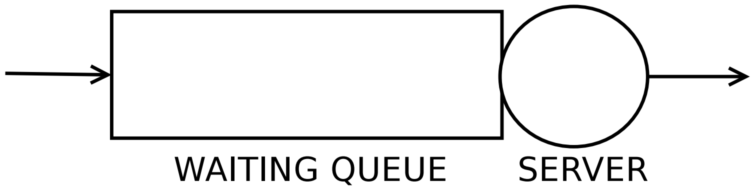


Figure 6.16: *A Basic Queue-Server representation*

In order to determine the average host queue length, Equation 6.4 can be used given a service rate (μ) and mean arrival rate (λ) as parameters. Equation 6.5 states ρ, the traffic intensity value, as the ratio of arrival to service rates. Note that a ρ-value of one results in a queue that grows boundlessly [17].

$$N = \frac{\rho}{1-\rho} \tag{6.4}$$

$$with \quad \rho = \lambda/\mu \tag{6.5}$$

Equation 6.4 can also be used to determine the sum of the waiting and service times of packets in the hosts using Equation 6.6 in a substitution for N.

$$N = \lambda T \tag{6.6}$$

This leads to:

$$T = \frac{1}{(\mu-\lambda)} \tag{6.7}$$

Equation 6.7 is known as *Little's result.*

6.5.3 Modelling Service Rate

Figure 6.17 is a block-diagram representation of the traffic generator (`host[0]`) with one of the successor hosts. It can be seen on Figure 6.15 that the successor hosts (1 hop away) are hosts 1, 4 and 10. The application and physical layers are not explicitly depicted. To determine the theoretical service rate, one must first consider where processing delays may occur.

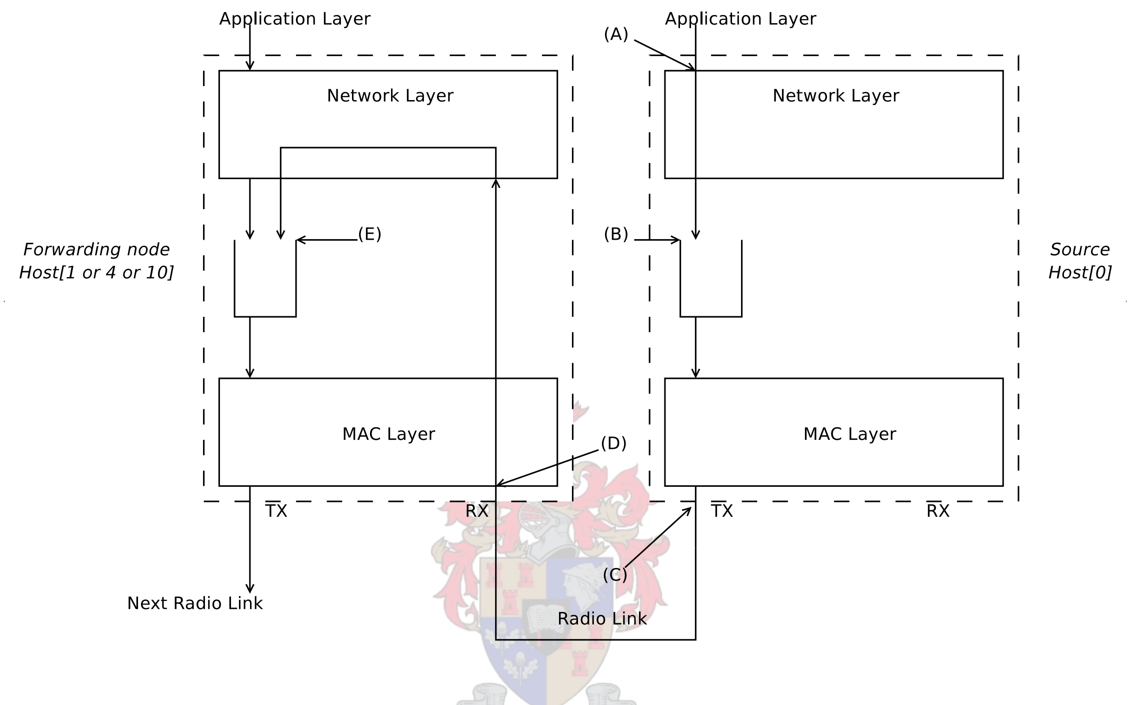


Figure 6.17: *Block diagram of host[0] and one successor host*

Between points A and B: In this section there may be some processing delay caused due to route discovery. However, due to the fact that it is assumed that route discovery has already been done, it can be assumed that the delay will be negligible.

Between points B and C: Section B-C was initially (and incorrectly) considered to be a source of negligible delay. It will later be determined that delays such as DIFS and Truncated Binary Exponential Backoff times are the main sources for delay in this section. This will be investigated in Subsection 6.5.7.

Between points C and D: This section was initially considered to be the main influence of the host's service rate (μ). Initially, determination of the service rate was done by determining the MAC Layer transmission delay.

$$\begin{aligned} T_{MAC} &= \frac{(bits_{RTS} + bits_{CTS} + bits_{DATA} + 3bits_{header})}{bitrate} + 2SIFS + 3Tp \\ &= 0.311246666667 \text{ seconds/packet} \end{aligned} \tag{6.8}$$

Initial service rate calculation can then be determined by using the inverse of this value.

$$\mu = 1/T_{MAC} = 3.21288581404 \text{ packets/second} \tag{6.9}$$

This value of μ will later be shown to be incomplete when one inspects the difference between simulated and theoretical results.

Between points D and E: For Section D-E it was decided that processing delay from the MAC Layer to the Network Layer should be modelled as negligible. This is only possible if route discovery has already been done for all possible locations.

6.5.4 Modelling Arrival Rate

In order to understand what the different arrival rates of hosts in the paths of Figure 6.15 will be, the hosts will be represented as M/M/1 queues. Figure 6.18 is a representation of the source host (`host[0]`) with successors hosts (hosts 1, 4 and 10) that exists one hop away.

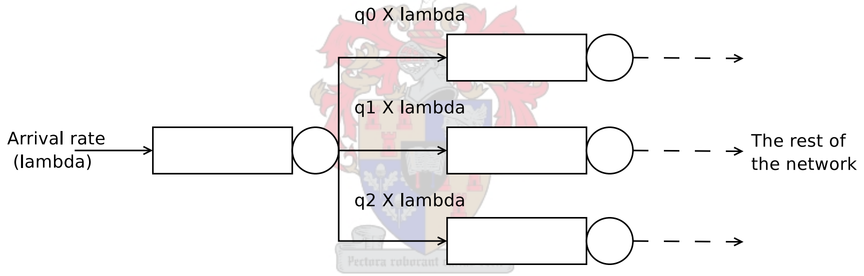


Figure 6.18: *Queues for host[0] and the 3 successor hosts*

According to Jackson [18] the incoming stream at the source of Figure 6.18 diverges into 3 differing streams with arrival rates indicated on the figure. The routing probabilities ($q_0,\ q_1,\ q_2$) are determined by the Lottery Route Selection Algorithm. Table 6.1 shows the calculated routing probabilities for each case. It is evident, from the table, that longer routes have a lower chance of being selected than shorter routes. Please note the correlation between the calculated (theoretical) values and the simulated values. The next section will deal with the simulation proof of these results.

6.5.5 Verifying Arrival Rates through Simulation

The simulation was rerun for various inter-arrival times. To verify that theoretical and simulated arrival rates are the same, the following results were depicted in Figure 6.19.

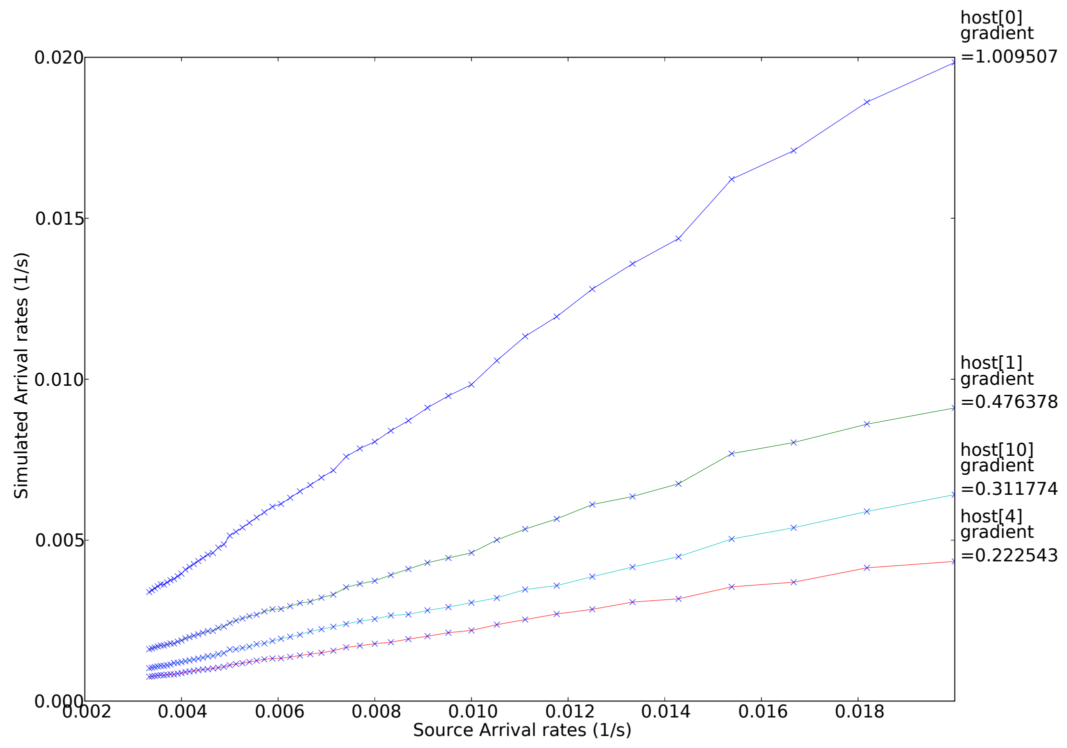


Figure 6.19: *Arrival times versus Perceived Arrival times*

It can be seen from Figure 6.19 that the source host's gradient is approximately equal to unity. The gradients of the other lines are equal to the routing probabilities given by Table 6.1 (column three) and determined by the Lottery Route Selection Algorithm. The longer routes experience lower packet arrival rates than the shorter routes. *This verifies that the Lottery Route Selection Algorithm is functioning in the way it was intended.*

Route Lengths	*Calculated Route Probabilities*	*Simulated Route Probabilities*
4 hosts	$q_0 = 0.4615$	$q_0 = 0.47637$
6 hosts	$q_1 = 0.3076$	$q_1 = 0.31177$
8 hosts	$q_2 = 0.2307$	$q_2 = 0.22254$

Table 6.1: *Route lengths with appropriate route probabilities (Theoretical and Simulated)*

6.5.6 Determining Average Queue length from Simulation

To ascertain the average queue length at a specific host, every change in a host's MAC Layer queue length has been recorded during the simulation as well as the corresponding timestamp of each change. The *queue length time-average* of one host was then determined using Formula 6.10:

$$\begin{aligned} Average\ N_{host\ x} &= N_{t_o}\frac{t_0}{\sum t_i} + N_{t_1}\frac{t_1}{\sum t_i} + N_{t_2}\frac{t_2}{\sum t_i} + \cdots \\ \text{where } i &= 0\ldots \text{ Number of Samples} \end{aligned} \tag{6.10}$$

Figure 6.20 was plotted using this information. The expected buffer occupancy that is (incorrectly) calculated using the $\mu = 1/T_{MAC}$ from Equations 6.8 and 6.4 is shown with $\times$-marks. The corresponding theoretical average queue length plots for the other hosts were not added for the sake of clarity. The only conclusion that could be made from these graphs is that the theoretical values did not correspond to the simulated values. An improved calculation will be determined later and shown in Figure 6.25.

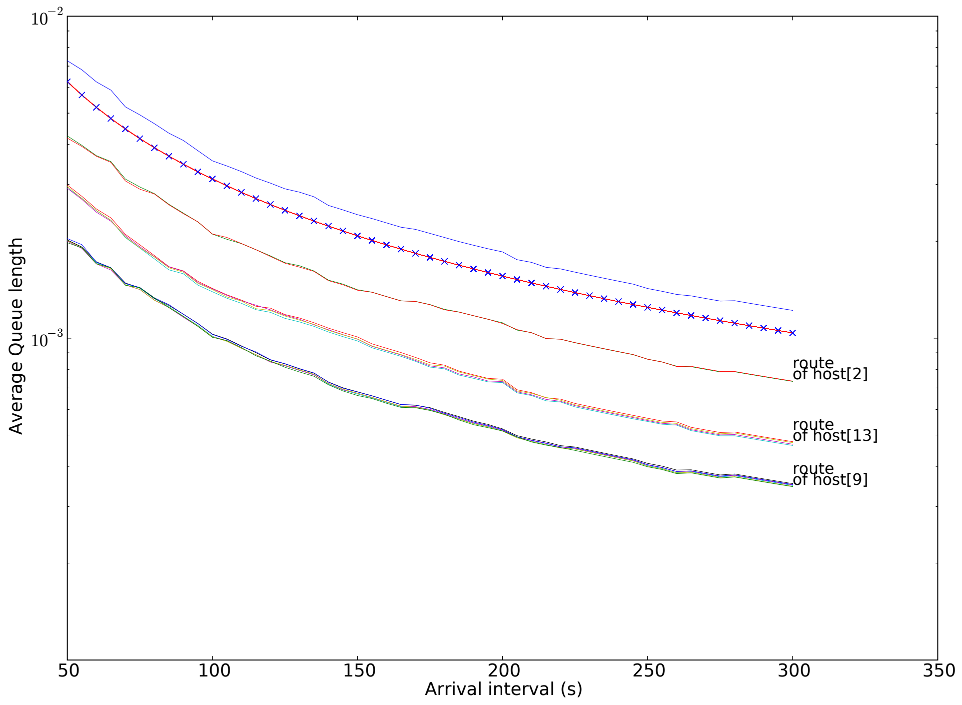


Figure 6.20: *Semilog graph of Queue length with One Theoretical Determined Queue Length ($\times$-marks)*

Figure 6.21 is a closer inspection of Figure 6.20 over a certain interval. It can be seen that hosts of certain routes have similar queue lengths. The top queue length plot in Figures 6.20 and 6.21 belongs to the source host. The next bundle of plots (not marked) belongs to the center route (hosts 1,2). The following bundle of plots belongs to the hosts of route `10-11-12-13`. The last bundle of lines depicts the queue length of the hosts of the route `4-5-6-7-8-9`. The calculated graph with constant μ (of Equation 6.9) was also superimposed onto the figure. This implies however that the actual μ-value varies for different hosts. The next section will investigate and explain the reason for the discrepancy.

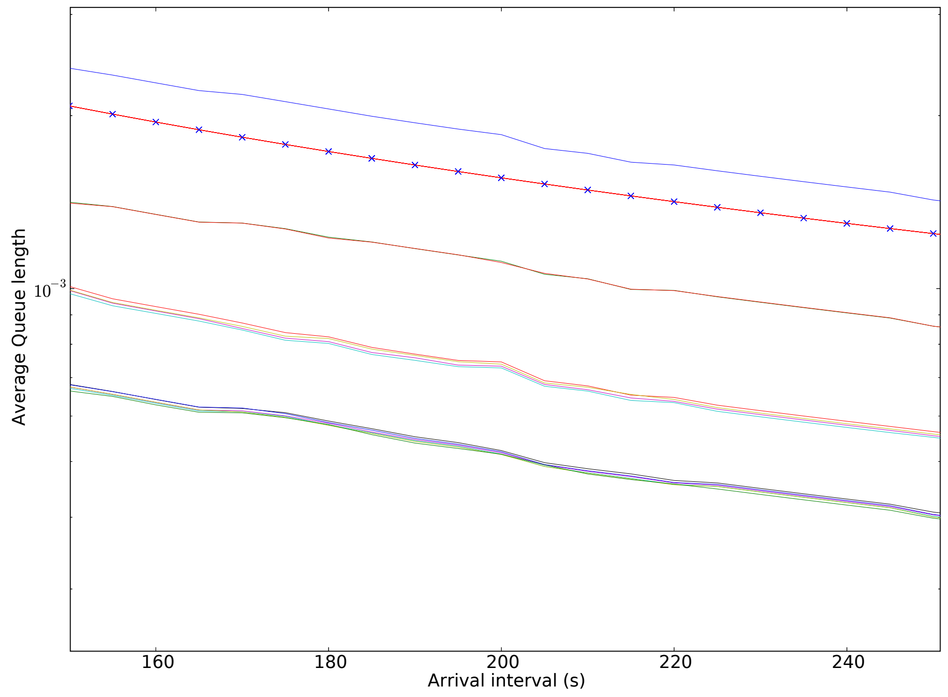


Figure 6.21: *Semilog graph of Queue length at closer inspection*

The following subsections will deal with an **improved** theoretical understanding of service rate, followed by another attempt to determine the average *Queue Length* of a host. This will then be followed by a comparison of theoretical and simulated average *latency* quantities.

6.5.7 Determining Service rates out of Simulation

The following section will help explain service rate quantities and allow a better theoretical model for service rate to be determined. For increasing inter-arrival times the service rates were determined and Figure 6.22 was rendered.

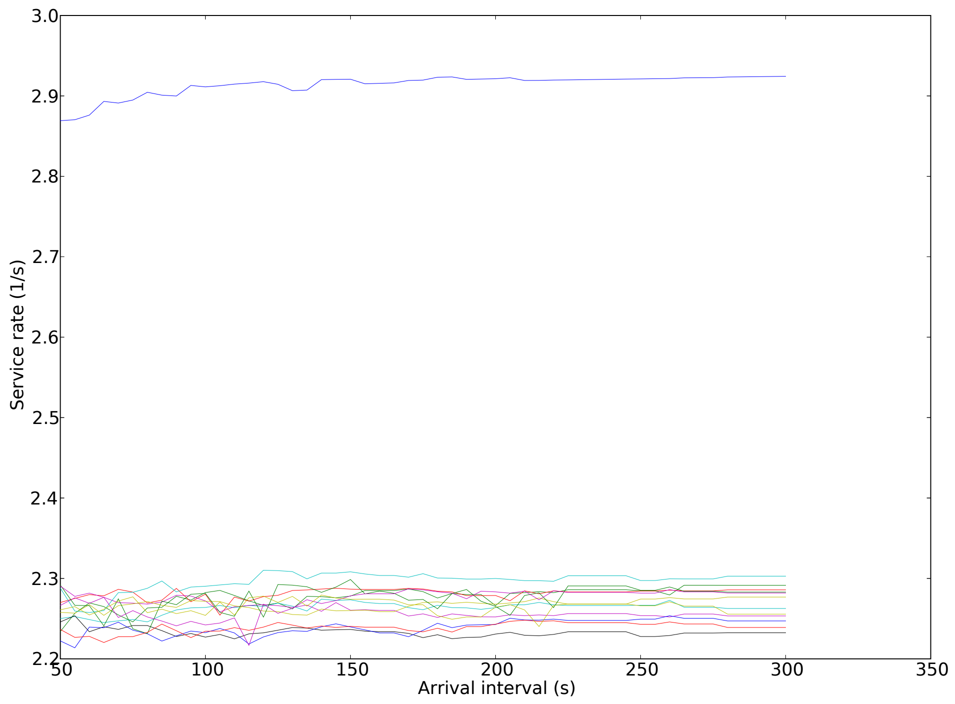


Figure 6.22: *Perceived Service rate versus interval*

In the figure, the y-axis represents the *average service rates* of all the hosts for various simulation runs. Each run contributes a data point to the figure. Each simulation run uses a different *average packet arrival interval* and this is represented by the x-axis. The inverse of this interval can be seen as the source host's *average packet arrival rate*. A shorter arrival rate would imply a larger arrival interval. On the figure it is shown that the source host (`host[0]`) achieves "steady state" after the 100 second mark. This means that the simulations using an average arrival interval of 100 seconds or longer, exhibited similar service rates.

Reasons for Service rate Inequality

It was initially determined that the service rate of all the hosts would be the constant amount given by the value $\mu = 1/T_{MAC} = 3.21288581404$ packets/second. According to simulation however, this is not the same value for all hosts as can be seen in the Figure 6.22. This inconsistency can be attributed to the fact that the processing delay of the host *while inside the buffer* was not kept into consideration. This delay occurs in section B-C in

Figure 6.17. The main reason for the large differences in service rates can be attributed to the backoff behaviour of the host. Hosts that transmit and receive (such as *forwarding hosts*) are more likely to contend for the medium. These hosts would then use the *truncated binary exponential backoff (BEB) algorithm* in order to resolve medium access conflicts.

The Truncated Binary Exponential Backoff algorithm model

The *Truncated Binary Exponential Backoff Algorithm* was modelled in [7]. The backoff calculation functions can be described as follows:

```
Global Variables: retryCounter = 1
Global Constants: ST, CWmin = 7, CMmax= 255
Calculation Functions: CW(), backoff()

        CW()
        {
           window = CWmin * 2^(retryCounter-1)-1
           if window >= CWmax then return CWmax
           else return window
        }

        backoff()
        {
           // returns a value that is chosen
           // from a uniform distribution
           // and multiplies with ST
           return uniform(CW())*ST
        }
```

The `CW()`-function calculates the current *contention window* value. This method also implements the truncation function for the algorithm by checking that the newly determined window is no bigger than the maximum value (`CWmax = 255`). The `retryCounter`-value is a global value that is incremented every time the MAC Layer sends a ready-to-send message. Every RTS message implies a new retry attempt. Figure 6.23 shows a flowchart representation of a section of the IEEE 802.11 DCF state machine in order to explain how the backoff section fits into the MAC Layer system as a whole.

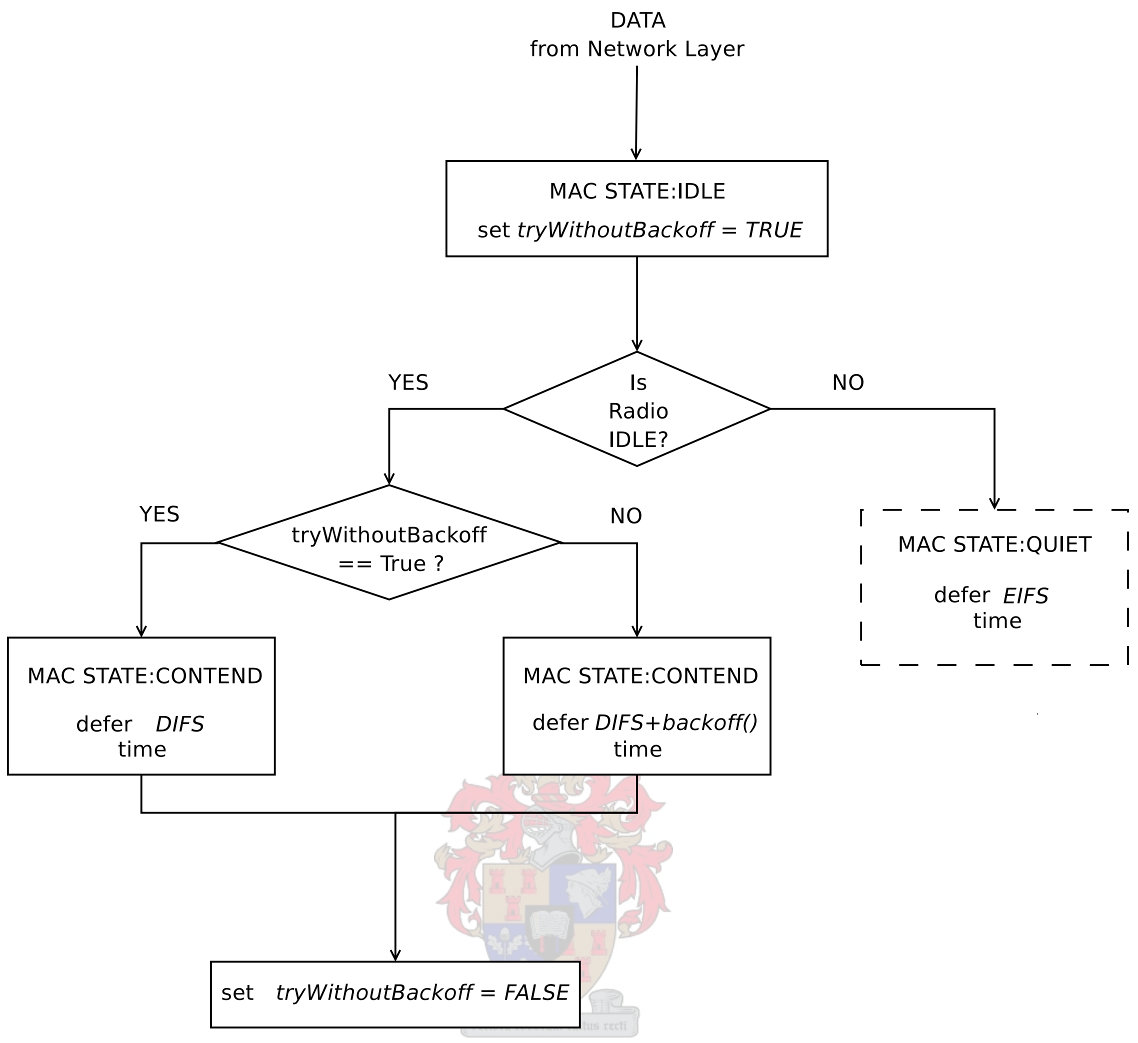


Figure 6.23: *Application of the BEB-Algorithm in the MAC Layer model*

The model uses a Boolean variable called `tryWithoutBackoff` that acts as a switch in the algorithm. The '`--`'-line block indicates the modification that was made to the model in order to compensate for the high arrival rate problem previously mentioned in Section 3.2. It can be seen from Figure 6.23 that the `tryWithoutBackoff` switch is set to `true` in during the MAC Layer `IDLE`-state. This switch then changes if the radio hardware layer is in an `IDLE`-state (not receiving or transmitting). This acts as a *latch* that enables the MAC Layer to use backoff behaviour for future attempts to transmit the packet.

If a packet is not transmitting, then it must be waiting while the MAC Layer is in a CONTEND-state. This implies that in order to determine what causes the delay in section B-C of Figure 6.17, the timing activity of the CONTEND-state timer must be logged. The average B-C delay (CONTEND-state delay) for each simulation run is shown in Figure 6.24. The x-axis is the average arrival interval ($1/\lambda$) used for each specific simulation run.

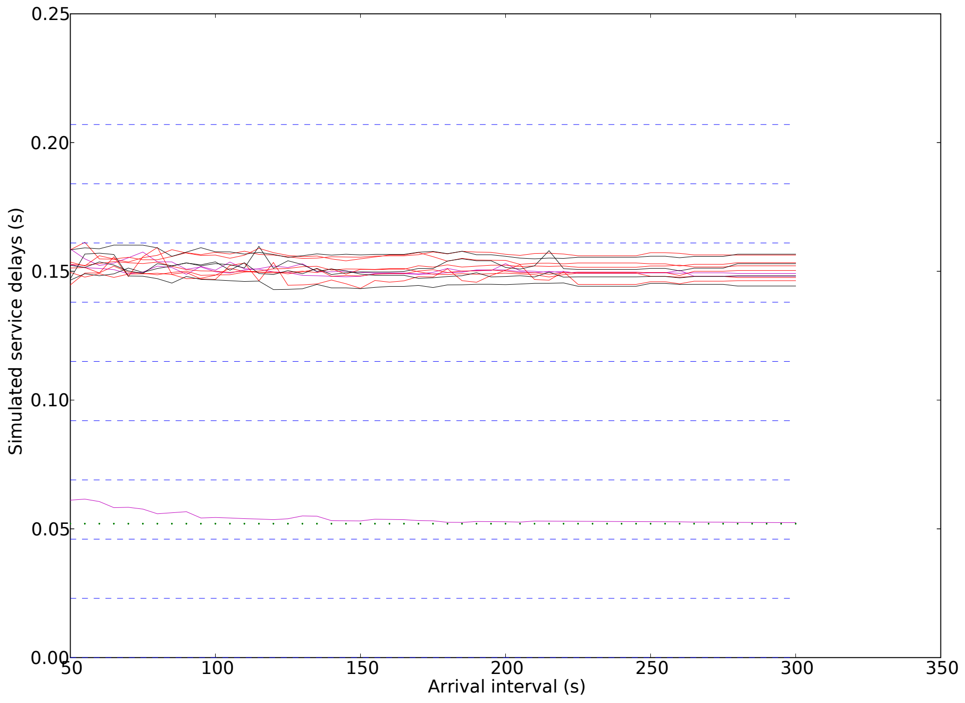


Figure 6.24: *Backoff Delay versus interval (multiples of Slottime indicated)*

Please note the average source (host[0]) section B-C delay is given by the bottom line, that reaches a "steady state" value of DIFS ($= 2 \times ST + SIFS$) after the 100 second interval mark. The other hosts have average delays over B-C of approximately equal to 1 DIFS period plus 4 slottimes. It can be proven that this is the source of the delay that is causing the difference in the simulated service rate, by increasing both graphs with an offset equal to the transmission time ($T_{MAC} = 1/\mu = 0.311246666667$). The inverse graph of this summation is then equivalent to Figure 6.22.

- The reason for this delay difference is that host[0] is the only source host. This fact suggests that its chances to be able to transmit is much greater than that of any other host. This, in turn, implies that host[0] does not need to employ a *backoff* as much as the other hosts. Its average CONTEND-state period is therefore equal to 1 DIFS period.
- Other hosts only act as forwarders for packets. Once they have forwarded packets to the previous host they are inactive until they receive another packet to forward. These

hosts are, however, in a situation where their successors and predecessors are both contending for transmission, due to the continuous stream of packets that are being sent through each route. These contention situations makes links on both sides of a host unavailable for the host causing the host to start a truncated binary exponential backoff.

- The truncated binary exponential backoff algorithm uses a uniform distribution of which the upper limit varies with every retransmission attempt. (For example: "0 . . . 7 for attempt 1, 0 . . . 14 for attempt 2 and eventually 0 . . . 255 for the last attempts.)
- It has been determined that the MAC Layer backs of mostly once. This means that the uniform distribution will lie over 0 . . . 7. This distribution has an average value of 3.5.
- This implies that a non-source host will then have an average backoff at least once and the average of all these backoff times is $3.5 \times ST$. This would make the total average backoff time for a node $DIFS + 3.5 \times ST$. Which is approximately the average delay value ($DIFS + 4 \times ST$) of the cluster of graphs that are visible on Figure 6.24.

The average service rate can be computed by taking the previously mentioned points into consideration and applying them as follows:

1. Firstly the average service delay must be determined. For the forwarding hosts this average service delay is given by:

$$\begin{aligned} T_{FW(service\ delay)} &= DIFS + 3.5 \times ST + T_{MAC} + SIFS \\ &= 52.0 \times 10^{-3} + 3.5(23 \times 10^{-3}) + 0.311246666667 + 6 \times 10^{-3} \\ T_{FW(service\ delay)} &= 0.449746s \text{ seconds} \end{aligned} \tag{6.11}$$

   The source hosts can be determined in a similar fashion:

$$\begin{aligned} T_{SRC(service\ delay)} &= DIFS + T_{MAC} + SIFS \\ &= 52.0 \times 10^{-3} + 0.311246666667 + 6 \times 10^{-3} \\ T_{SRC(service\ delay)} &= 0.369246s \text{ seconds} \end{aligned} \tag{6.12}$$

2. This then leads to forwarding average service rate of :

$$\mu_{FW(service\ rate)} = 1/T_{FW(service\ delay)} = 2.2234739 \text{ packets/second} \tag{6.13}$$

   The average source service rate is given by:

$$\mu_{SRC(service\ rate)} = 1/T_{SRC(service\ delay)} = 2.708216 \text{ packets/second} \tag{6.14}$$

Improved Theoretical and Simulated Average Queue Length Comparison

1. The arrival rate of a host can be determined from the product of the route probability and the source arrival rate. The chosen average arrival interval is 200 seconds. Average arrival rate is therefore 1/200 packets/second.

$$\lambda_{FW} = q_x \times \lambda_{SRC} \tag{6.15}$$

The route selection probability of route `0-10-11-12-13-3` is equal to 0.3076 according to Table 6.1. In this route all nodes in the route section `10-11-12-13` (forwarding hosts) use a common average arrival rate (λ-value) given by Equation 6.15. For example, at `host[13]` the average arrival rate would be given by:

$$\begin{aligned}\lambda_{FW(host[13])} &= q_1 \times \lambda_{SRC}\\ &= 0.3076 \times 1/200\\ \lambda_{FW(host[13])} &= 0.0015379 \text{ seconds}\end{aligned}$$

All nodes in the route section `4-5-6-7-8-9` of route `0-4-5-6-7-8-9-3` also use a common average arrival rate (λ-value). Equation 6.15 can also be applied in this situation. For example, `host[7]`'s average arrival rate can be given by:

$$\begin{aligned}\lambda_{FW(host[7])} &= q_2 \times \lambda_{SRC}\\ &= 0.2307 \times 1/200\end{aligned}$$

Average arrival of hosts in route section `1-2` of route `0-1-2-3` can also be determined. This means that, for example, `host[2]`'s average arrival rate is computed as follows:

$$\begin{aligned}\lambda_{FW(host[2])} &= q_0 \times \lambda_{SRC}\\ &= 0.4615 \times 1/200\end{aligned}$$

2. The average latency incurred over a host (T) can be determined using Little's result (Equation 6.7). For the source host the latency is given by:

$$T_{SRC} = \frac{1}{\mu_{SRC(service\ rate)} - \lambda_{SRC}} \tag{6.16}$$

By incorporating this with Equation 6.6 the average queue length (N) of a source host is given by:

$$N_{SRC} = \frac{\lambda_{SRC(host)}}{\mu_{SRC(service\ rate)} - \lambda_{SRC}} \tag{6.17}$$

3. The latency and queue length of a forwarding host can similarly be computed by:

$$T_{FW} = \frac{1}{\mu_{FW(service\ rate)} - \lambda_{FW(host)}} \tag{6.18}$$

$$N_{FW} = \frac{\lambda_{FW(host)}}{\mu_{FW(service\ rate)} - \lambda_{FW(host)}} \tag{6.19}$$

By substitution of the values determined by Equations 6.14 and 6.13 into Equations 6.17 and 6.19, the average queue lengths determined by simulation in Subsection 6.5.6 can be determined. This set of ***improved*** *theoretical calculations* has been indicated by the ×-marks on the Figure 6.25.

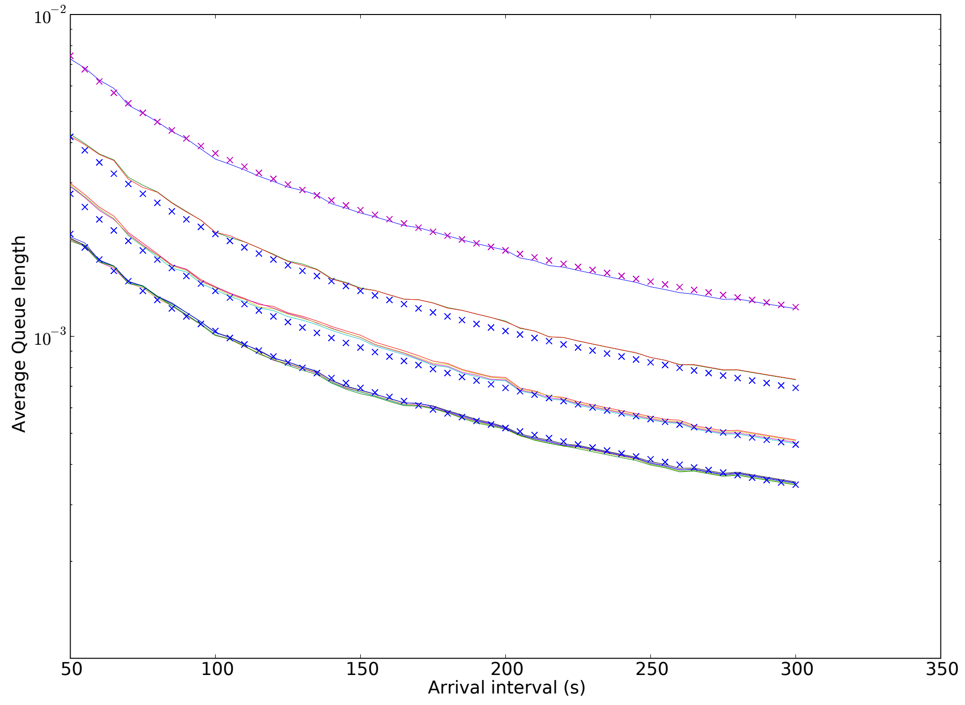


Figure 6.25: *Semilog graph of Queue length (Simulated and Improved Theoretical Results (×-marks))*

As can be seen from the figure, the average queue lengths for each route are closely approximated by the improved theoretical values. The following section will now deal with the theoretical and simulated analysis of average route latency.

Theoretical and Simulated Average Latency Comparison

The delay over the route (the latency) can be seen as the superposition of delays incurred over the source and forwarding hosts. By using the Equations 6.16 and 6.18 this value can be determined. The average latency over all three routes will now be determined. The last host (`host[3]`) will be left out of the calculation.

Please note: In this section, N represents route length in hosts.

- Latency to `host[13]` of route `0-10-11-12-13-3` can be determined as follows. The source (`host[0]`) and four other hosts latencies/delays must be summed. Latency ($T_{latency}$) over a route will be given by:

$$\begin{aligned} T_{latency} &= (N-1)T_{FW} + T_{SRC} \\ \text{where } N & \text{ is the number of hosts from the source} \\ & \text{ to the last host before the destination} \end{aligned} \tag{6.20}$$

  Take the number of hosts to `host[13]` (including source host) as $N = 5$.

$$\begin{aligned} T_{latency} &= \frac{\text{N-1}}{(\mu_{FW(service\ rate)} \text{ - } \lambda_{FW(host)})} + \frac{1}{(\mu_{SRC(service\ rate)} \text{ - } \lambda_{SRC})} \\ &= 4/(2.2234739 - 0.0015379) + 1/(2.708216 - 0.005) \\ &= 2.1701615s \text{ seconds} \end{aligned} \tag{6.21}$$

  The other route latencies can be computed similarly.

- For the route ending with `host[2]` the following average route latency calculation can be done ($N = 3$):

$$\begin{aligned} T_{latency} &= \frac{\text{N-1}}{(\mu_{FW(service\ rate)} \text{ - } \lambda_{FW(host)})} + \frac{1}{(\mu_{SRC(service\ rate)} \text{ - } \lambda_{SRC})} \\ &= 2/(2.2234739 - 0.4615 * 0.005) + 1/(2.708216 - 0.005) \\ &= 1.2703574s \text{ seconds} \end{aligned} \tag{6.22}$$

- Similarly the average route latency ending with `host[9]`. N is given by 7:

$$\begin{aligned} T_{latency} &= \frac{\text{N-1}}{(\mu_{FW(service\ rate)} \text{ - } \lambda_{FW(host)})} + \frac{1}{(\mu_{SRC(service\ rate)} \text{ - } \lambda_{SRC})} \\ &= 6/(2.2234739 - 0.2307 * 0.005) + 1/(2.708216 - 0.005) \\ &= 3.0698102s \text{ seconds} \end{aligned} \tag{6.23}$$

The next paragraph will deal with latency simulation. Figure 6.26 indicates the calculated results given in Equations 6.21 to 6.23 as a '--'-line. The simulated values are represented by the solid line.

Simulated Latency Determination

The latency of a packet is, in this case, defined to be the time it takes a packet to reach the destination host after begin created at the source host's Application Layer. It was therefore decided to set a timestamp on the application packet at the time of creation. The latency would then be determined at the destination by checking the difference between the timestamp and the current simulation time. By using Equations 6.21 to 6.23 the theoretical latency values can be determined for all routes.

Route of host..	*Theoretical latency values*
`host[9]`	3.069810s
`host[13]`	2.170161s
`host[2]`	1.270357s

Table 6.2: *Theoretical Values determined with Equations 6.21-6.23*

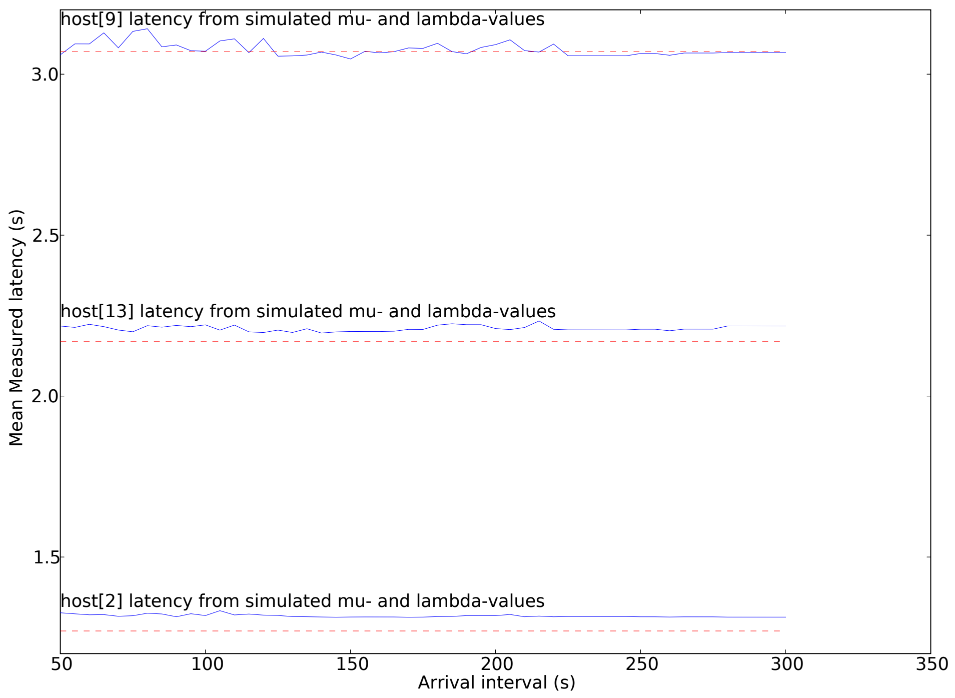


Figure 6.26: *Latency versus Arrival Interval*

In Figure 6.26, the simulated latencies are plotted as simple lines while the theoretical values are marked by dashed lines. In the beginning the values deviate, due to varying perceived (simulated) arrival rates.

6.6 Variation of Packet Size and Bitrate

The experiments of this section will vary *data packet size* and *bitrate*. In this section various networks will be simulated and metrics such as *latency*, *efficiency* and *throughput* will be determined. The values to be determined through simulation will be defined as follows:

- *Latency* – In the context of this simulation latency will be defined as the time it takes for a packet to progress from the source host to the destination.
- *Efficiency* – Efficiency is the ratio of data bits to total bits sent. This can be seen in Equation 1.1 in Chapter 1. Any retransmissions and headers are counted as overhead.
- *Throughput* – This quantity is determined as the quotient of the total amount of data bits and the latency of a data packet.

There are three cases that will be considered in the next simulation. They will be referred to as the *linear 7 node*, *linear 14 node* and *three prong cases*. A description of each will follow.

Please note:

- *Varied Packet Size Simulation* – Efficiency values will increase, but these simulations do not take in account the bit error rate that changes with packet size.
- *Varied Bitrate Simulation* – Even though the Slottime and SIFS parameters of the MAC Layer were scaled for a bitrate of 9600 bits/second, faster bitrates will still be compatible with these timing values. Optimal throughput and latency for the high bitrate cases will not be achieved. Lower bitrates would, however, experience timing conflicts and errors.

6.6.1 Simulation Cases

Linear 7 Node Path Network

The *linear 7 node case* is depicted in Figure 6.27. This network configuration consists of 7 hosts linked together in the following fashion:

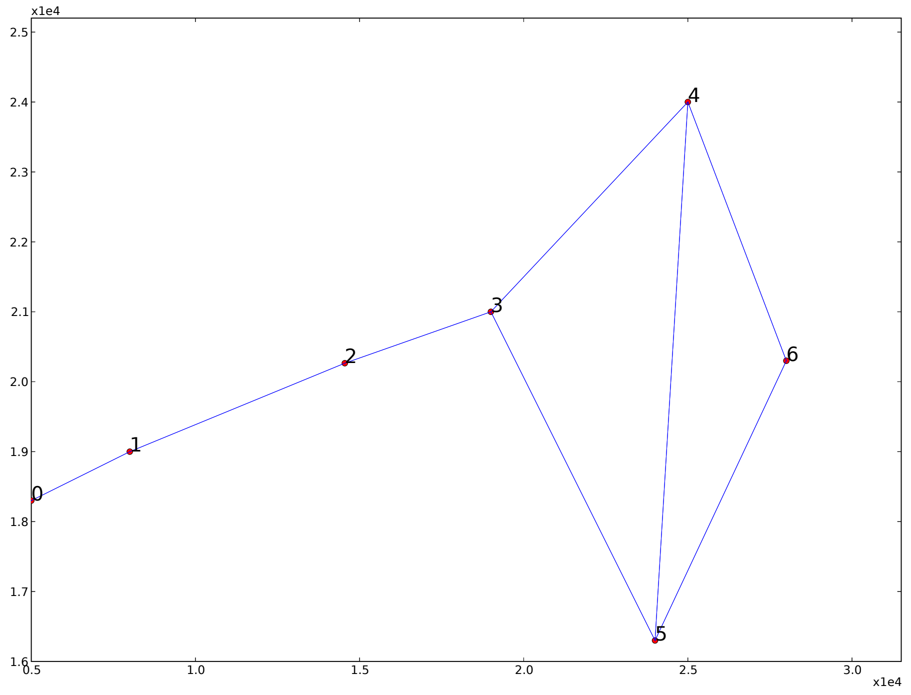


Figure 6.27: *Simple seven node path network*

In all these cases `host[0]` will be set as a source node with an exponential arrival rate of which the average arrival interval is 200 seconds which implies an arrival rate of 1/200 packets/second. The destination host is `host[6]`.

Linear 14 Node Path Network

This network is two 7 node path networks placed in series to each other. `Host[13]` is set to be the destination host.

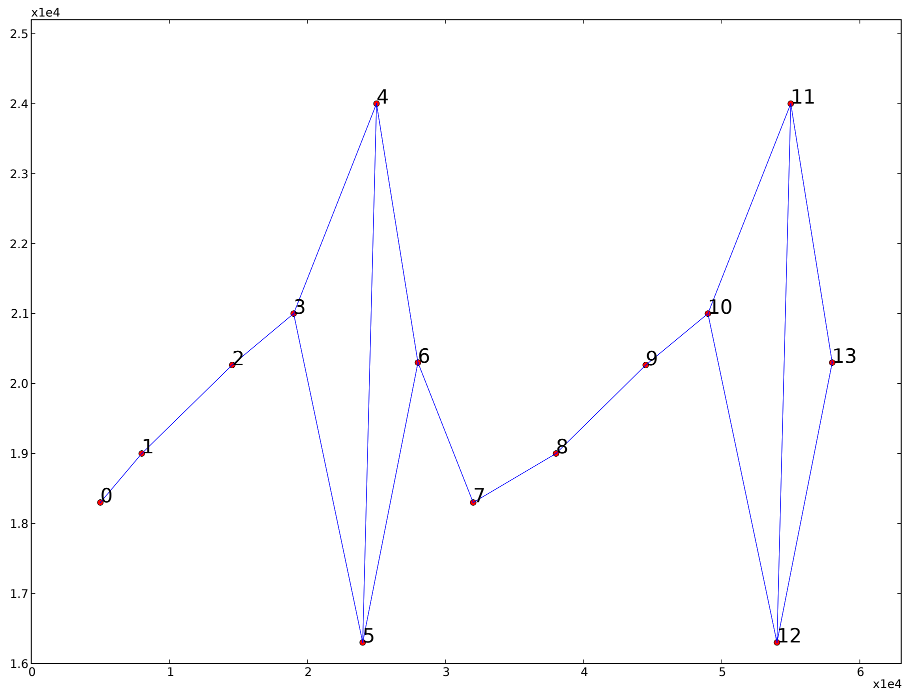


Figure 6.28: *Extended 14 node network*

The third test network (the *three prong case*) will be reused from Figure 6.15.

6.6.2 Route Preference Graphs

Linear 7 Node Path Route Preference

Figures 6.29 to 6.32 shows the route preference graphs for the three test cases. For Figure 6.27 the link between hosts 4 and 5 can be ignored. This is evident when inspecting the route preference graph.

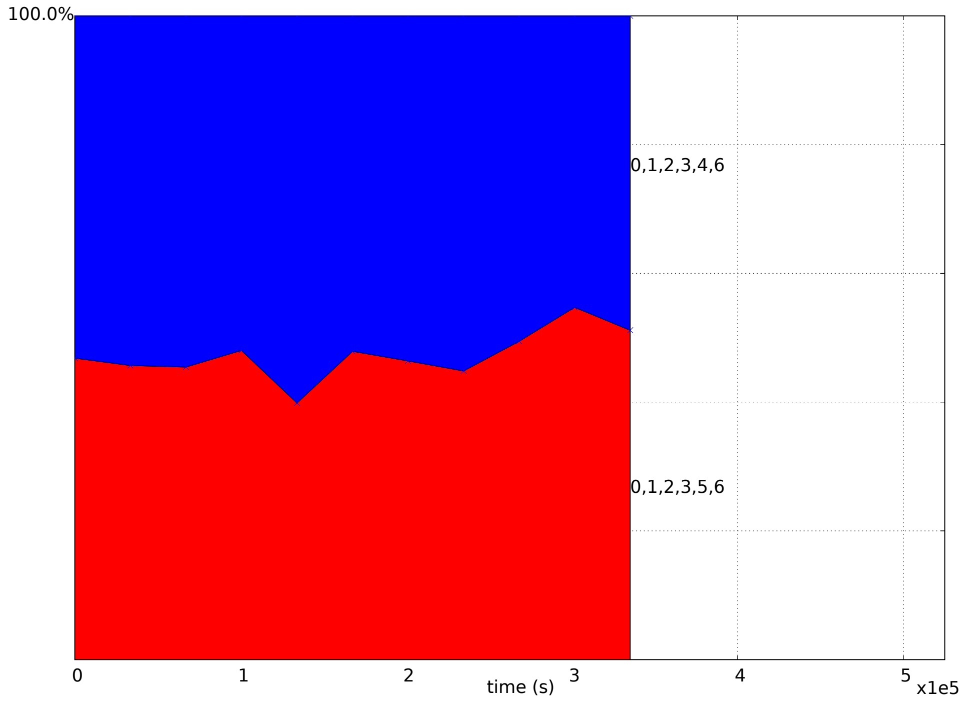


Figure 6.29: *Route Preferences for packets of 7 node path case (TTL = 6)*

From the figure the route popularity for each route is very close to each other. This means that the route usage is divided equally amongst the two routes. The routes weight power for this experiment is 3. It can be seen that the routes do not use link 4-5. This is due to the fact that the `RREQ` time-to-live is set to 6. When this value is increased to 7 the route preference graph can be seen by Figure 6.30.

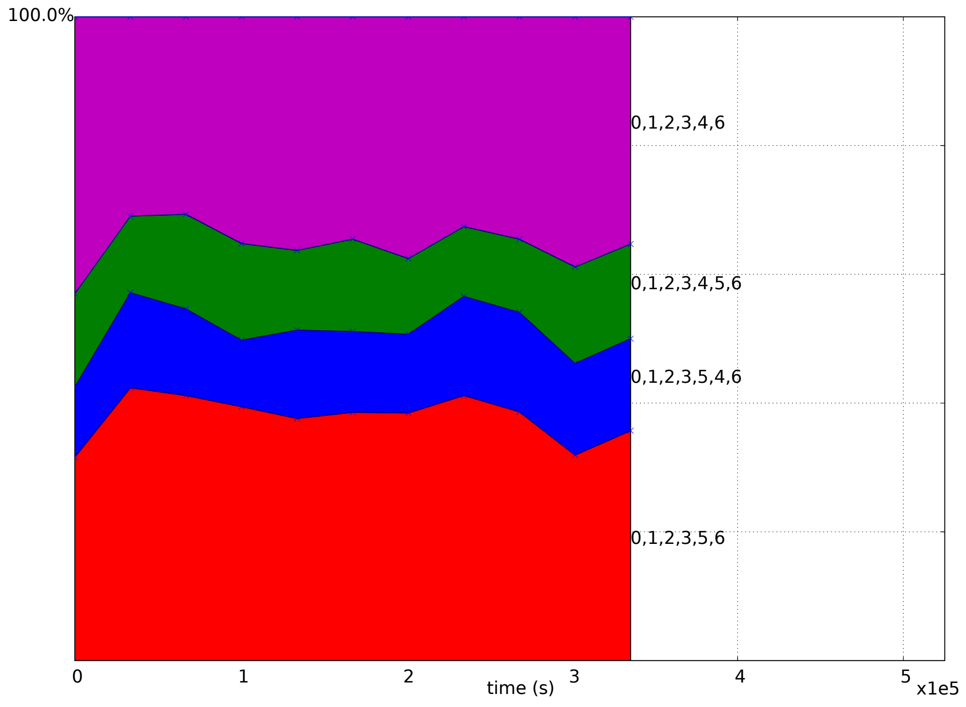


Figure 6.30: *Route Preferences for packets of 7 node path case (TTL = 7)*

It can be seen that the route popularity is much greater for the shorter routes (route `0-1-2-3-4-6` or route `0-1-2-3-5-6`) than for the long routes (route `0-1-2-3-4-5-6` or route `0-1-2-3-5-4-6`). For the rest of the experiment the time-to-live will be kept at 6 for the 7 node case.

Linear 14 node Path Route Preference

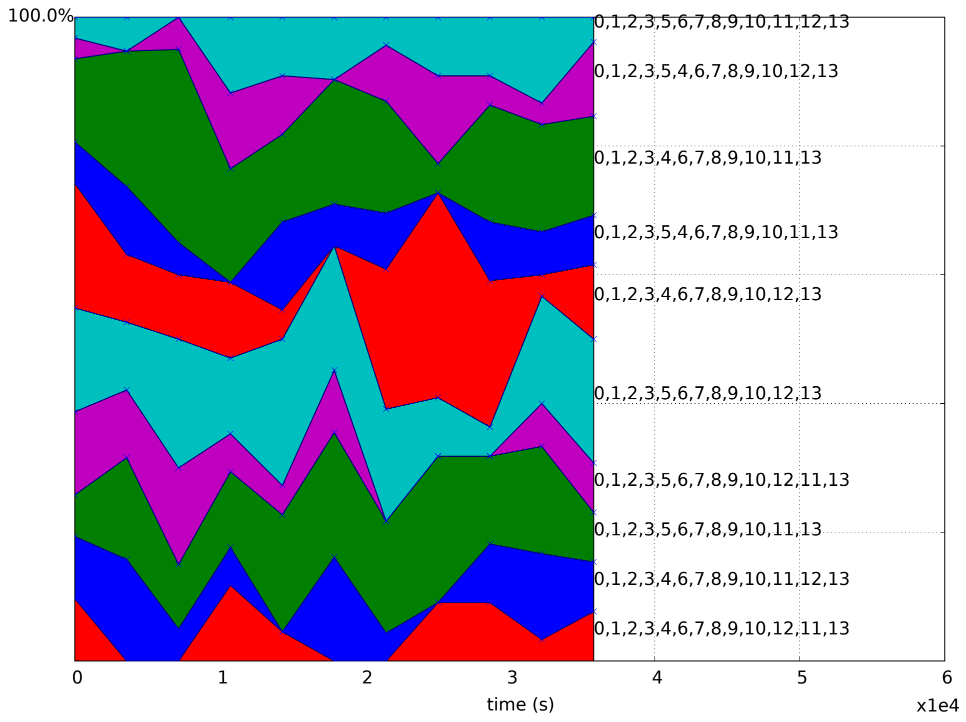


Figure 6.31: *Route Preferences for packets of 14 node path case*

The route weight power used for this experiment is 3. Note that routes using links `4-5` or `11-12` or both do not take up as much space on Figure 6.31 than the other routes. This implies that these routes are less popular than the other routes. The time-to-live for this network was set to 13. This means that there is a higher chance of link `4-5` begin used in this network than in the 7 node case.

Three Prong Network Route Preference

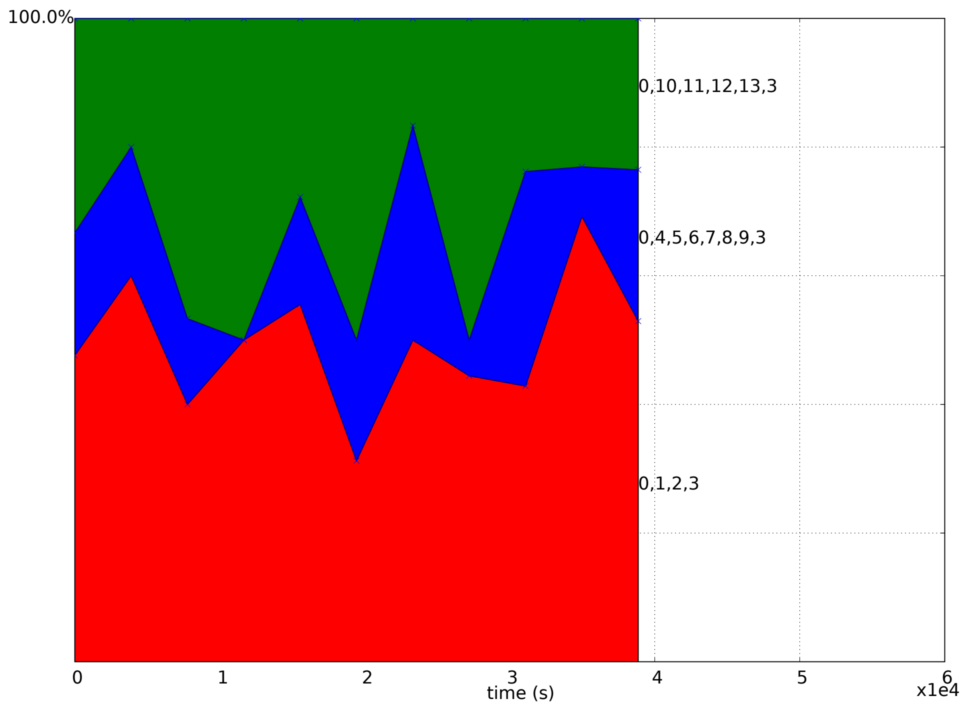


Figure 6.32: *Route Preferences for packets the three prong network*

The route weight power used for this experiment is 1. Once again, shorter routes get preference in this situation over larger routes.

6.6.3 Varied Packet Size

Figure 6.33 displays the efficiency of the three test network cases.

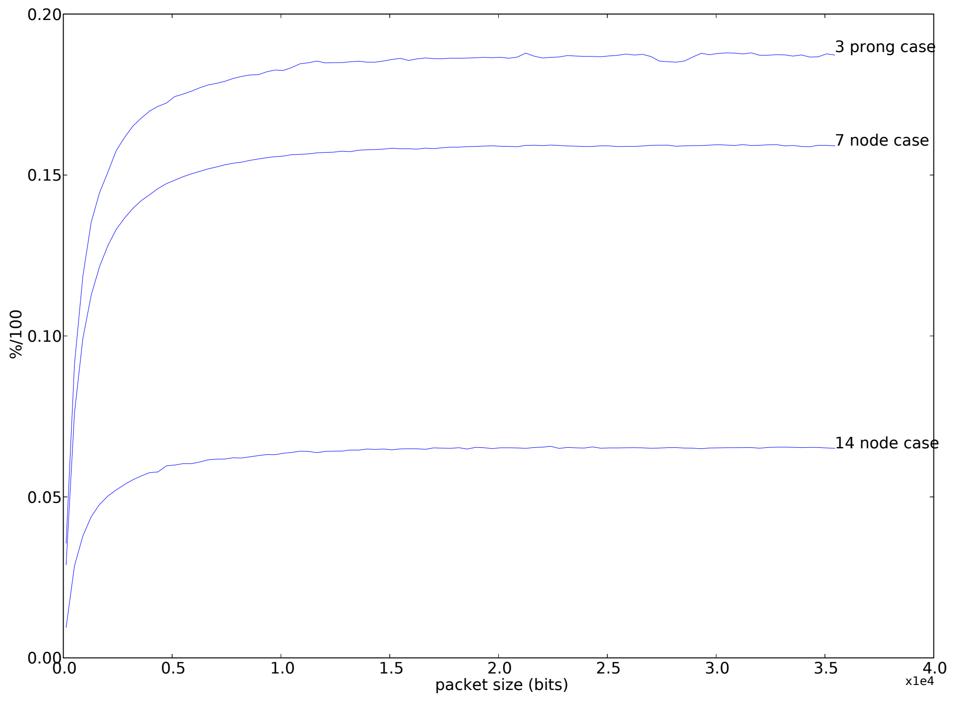


Figure 6.33: *Average Efficiency versus Data Packet Size*

According to the graph the efficiency values for each graph tends toward a certain constant value after approximately 1×10^4 bit-packets. The maximum reached values can be summarised in the following table.

Test Name	*Max Efficiency Value*
Linear 14 node network	0.0657207054534
Linear 7 node network	0.159447499296
Three Prong network	0.187945974805

It can be seen that the Three Prong network has the highest achieved efficiency in the simulation even with the same amount of nodes as the 14 node network. This can be attributed to the Network Layer. The longer the route that a packet needs to take the larger the header information will be, due to source routing. This increases the overhead and therefore decreases efficiency. In the three prong network, the lottery selection favoured the shorter routes. This caused average routing overhead to be minimal, due to the fact that a shorter route needs to be added to the packet header.

Latency Versus Packet size

The latency of the three cases are shown in Figure 6.34. Latency predictably increases with packet length. This gradient of the Three Prong network case is lower than the two other network cases.

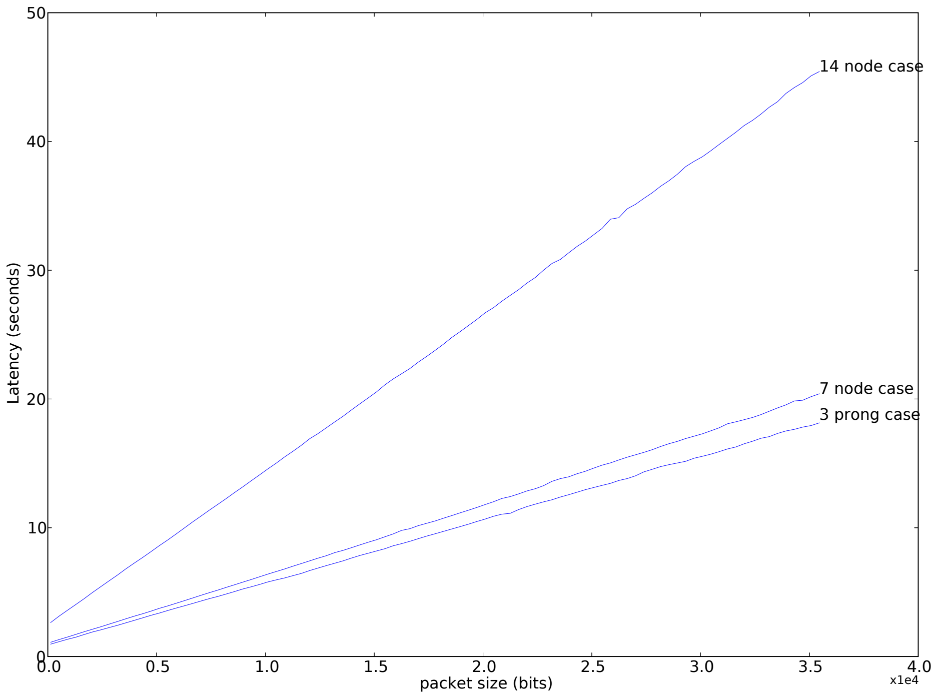


Figure 6.34: *Average Latency versus Data Packet Size*

To explain the characteristics of the network latency graphs such as Figure 6.34, the following should be considered:

- The latency of the entire network can be determined by:

$$y_{latency(TOTAL)} = \sum_{i}^{\#\ of\ routes} P_i \times y_{latency(ROUTE)} \tag{6.24}$$

The P-values refers to the route probability. An example of this was given by Table 6.1 (second column).

- The latency of one route is given by:

$$y_{latency(ROUTE)} = (N_i - 2) \times y_{latency(FW)} + y_{latency(SRC)} \tag{6.25}$$

where N_i is the total nodes in the $route_i$ (including source and destination nodes)

The `N`-value must be subtracted by 2 in order to exclude the source and destination nodes. The source node's effect is considered in the second term of the formula. The destination node is not an active traffic generator and will therefore not cause delays in this sense.

- The following can be used in order to describe the latency of a *forwarding host*:

$$y_{latency(FW)} = \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS + 3.5ST\right) \tag{6.26}$$

The parameters defined are as follows:

$$\begin{aligned} B_{overhead} &= B_{RTS} + B_{CTS} + B_{headers} + B_{route} \\ B_{route} &= \text{the amount of bits the source route includes} \\ B_{headers} &= \text{the total bits for all the packet headers (RTS, CTS, Data)} \\ bitrate &= \text{9600 bits/second} \\ x &= \text{ the current packet size} \end{aligned}$$

$y_{latency(FW)}$ can also be seen as the time a packet would have to wait to move from one node to another. The `ACK`-packet delay and its corresponding SIFS delay was left out, due to the fact that the acknowledgment is exchanged after the data packet is transmitted.

- Similarly the following can be used in order to describe the latency of the *source host*:

$$y_{latency(SRC)} = \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS\right) \tag{6.27}$$

- By substitution of Equations 6.26 and 6.27 into Equation 6.25:

$$\begin{aligned} y_{latency(ROUTE)} = {} & (N_i - 2) \times \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS + 3.5ST\right) + \\ & \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS\right) \end{aligned}$$

- Applying this result to Equation 6.24 will lead to:

$$\begin{aligned} y_{latency(TOTAL)} = {} & \sum_{i}^{\#\ of\ routes} P_i \times \left\{ (N_i - 2) \times \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS + 3.5ST\right) + \right. \\ & \left. \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS\right) \right\} \end{aligned}$$

$y_{latency(TOTAL)}$ is a weighted average, with the weights being P_i or the route probabilities. This implies that $y_{latency(TOTAL)}$ can actually be seen as the *expected value of the network latency.* When compared to the simulated values, it can be seen in Figure 6.35 that the theoretical values follow the simulated values very closely.

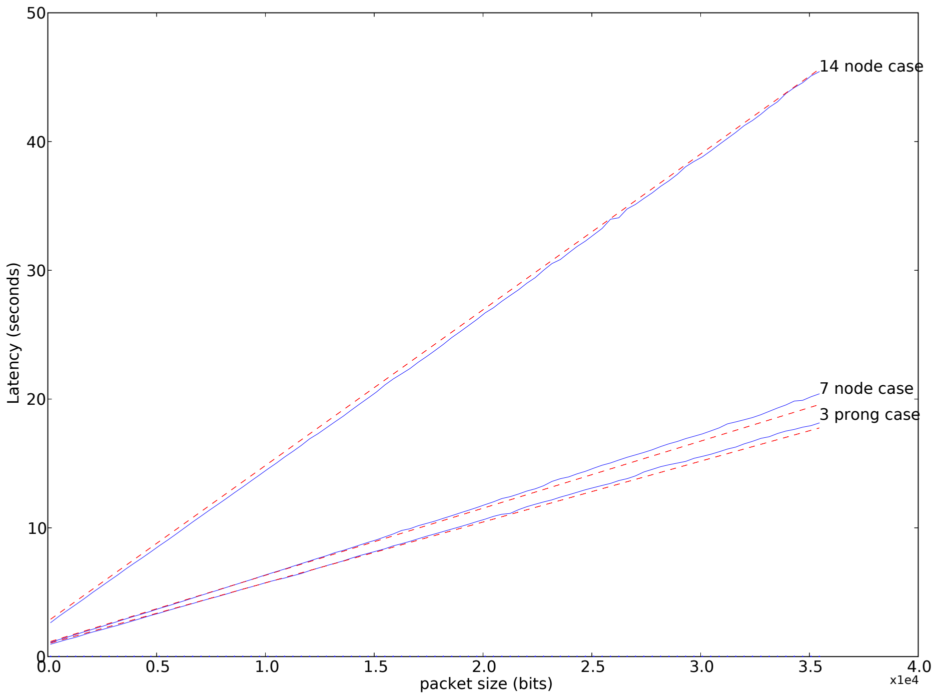


Figure 6.35: *Average Latency versus Data Packet Size*

- Differentiation of the previous formula will have the following result:

$$\begin{aligned} \frac{dy_{latency(TOTAL)}}{dx} &= \sum_{i}^{\#\ of\ routes} P_i \times \left\{((N_i - 2) + 1) \times \left(\frac{1}{bitrate}\right)\right\} \\ &= \sum_{i}^{\#\ of\ routes} P_i \times \left\{\frac{N_i - 1}{bitrate}\right\} \\ &= \frac{1}{bitrate} \sum_{i}^{\#\ of\ routes} P_i \times (N_i - 1) \end{aligned} \qquad (6.28)$$

By using Formula 6.28 the theoretical gradient values can be determined. Determination of this gradient will now be demonstrated for each network case.

- For the Three Prong Network the values from Table 6.1 were used. These values have been reproduced in Table 6.3 in context. Using these values in Equation 6.28 leads to

Route Name	*Route Length (N-Value)*	*Route probability (P-Value)*
`0-1-2-3`	$N_0 = 4$	$P_0 = 0.4615$
`0-10-11-12-13-3`	$N_1 = 6$	$P_1 = 0.3076$
`0-4-5-6-7-8-9-3`	$N_2 = 8$	$P_2 = 0.2307$

Table 6.3: *Three Prong Network Values*

  a gradient result of 0.000472645833333 *second/bit.*

- For the Seven Node Network, the values from Table 6.4. These values lead to a gradient

Route Name	*Route Length (N-Value)*	*Route probability (P-Value)*
`0-1-2-3-4-6`	$N_0 = 6$	$P_0 = 0.5$
`0-1-2-3-5-6`	$N_1 = 6$	$P_0 = 0.5$

Table 6.4: *7 Node Network Values*

  of 0.000520833333333 *second/bit.*

- For the 14 Node Network case, the routes shown in Figure 6.31 were enumerated from the bottom upwards. The values are summarised in Table 6.5. The last theoretical

Route Length Numbers	*N-Value*	*Route probability*
$N_0,\ N_1,\ N_3,\ N_6,\ N_7,\ N_9,\ N_{10},\ N_{11}$	13	0.0764196
$N_2,\ N_4,\ N_5,\ N_8$	12	0.0971608

Table 6.5: *14 Node Network Values*

  gradient result is 0.00120951633351 *second/bit.*

Table 6.6 summarises the simulated and theoretical latency gradients results.

Test Name	*Simulated Gradient*	*Theoretical Gradient*
Linear 14 node network	0.00121138787163	0.00120951633351
Linear 7 node network	0.00054632145697	0.000520833333333
Three Prong network	0.0004865025557	0.000472645833333

Table 6.6: *Simulated and Theoretical Latency Gradients*

It was shown that the latency of a packet can be described in terms of the route probability (determined by the Network Layer during route discovery), the number of nodes in a route as well as the chosen timing values. Equations 6.24 and 6.25 will later be reused for determination of latency in simulations with varying transfer rates.

Throughput Versus Packet size

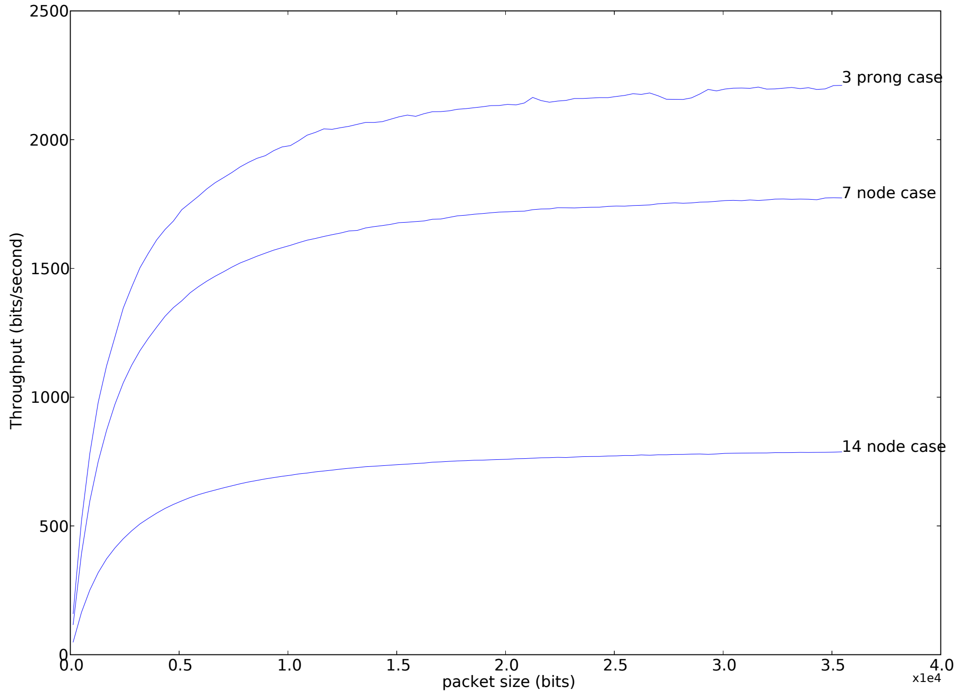


Figure 6.36: *Average Throughput versus Data Packet Size*

The formula for determining throughput of the network is similar to Equation 6.24:

$$y_{tput(TOTAL)} = \sum_{i}^{\#\ of\ routes} P_i \times y_{tput(ROUTE)} \tag{6.29}$$

The equation for determining a route's throughput can be given by:

$$\begin{aligned} y_{tput(ROUTE)} &= \frac{x}{y_{latency(ROUTE)}} \\ &= \frac{x}{(N_i - 2) \times y_{latency(FW)} + y_{latency(SRC)}} \end{aligned} \tag{6.30}$$

where N_i is the total nodes in the $route_i$ (including source and destination nodes)

x is the packet size

This implies that if one incorporates Equations 6.26 and 6.27 into Equation 6.30, the result can be written as follows:

$$\begin{aligned} y_{tput(ROUTE)} &= x \times \Bigg\{ (N_i - 2) \times \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS + 3.5ST \right) + \\ & \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS \right) \Bigg\}^{-1} \end{aligned}$$

This equation, in turn, causes the total network throughput to be written as follows:

$$\begin{aligned} y_{tput(TOTAL)} &= \sum_{i}^{\#\ of\ routes} P_i \times x \times \Bigg\{ (N_i - 2) \times \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS + 3.5ST \right) + \\ & \left(\frac{B_{overhead} + x}{bitrate} + 2SIFS + DIFS \right) \Bigg\}^{-1} \end{aligned} \tag{6.31}$$

The theoretical curves given by Equation 6.31 are plotted on Figure 6.37 and indicated by the dashed (`'--'`) lines.

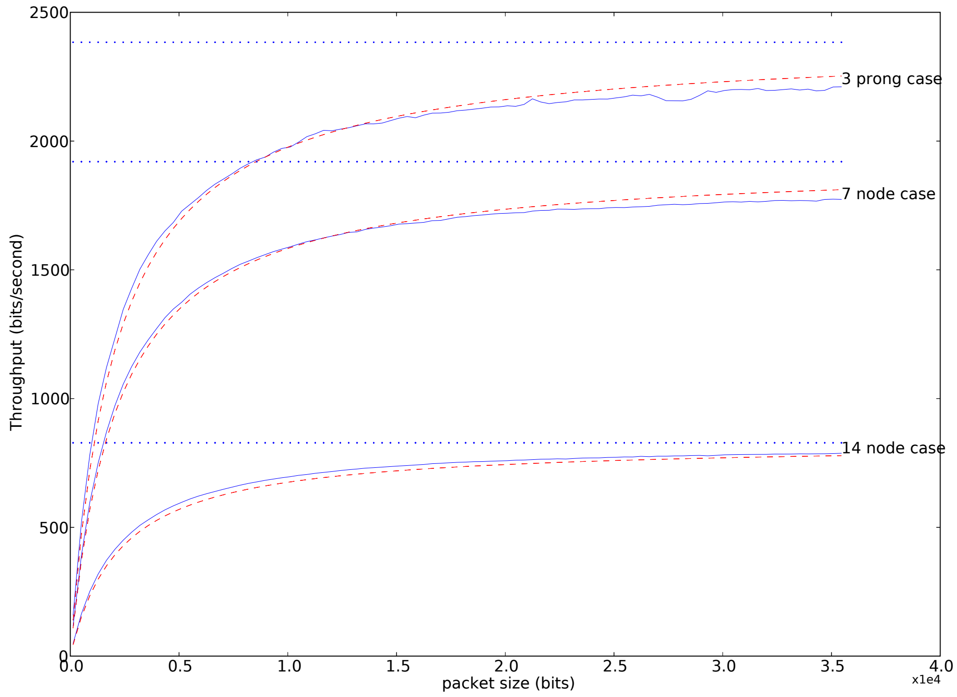


Figure 6.37: *Average Throughput versus Data Packet Size (with theoretical curve)*

When packetsize (x) tends toward infinity, throughput ($y_{tput(TOTAL)}$) strives towards a limit. This limit can be determined by recognising that Equation 6.31 follows the following form :

$$\begin{aligned} y &= \frac{x}{mx+c} \\ &= \frac{1}{m+c/x} \end{aligned}$$

This leads to the following implication:

$$x \rightarrow \infty \implies y \rightarrow 1/m$$

Similarly Equation 6.31 will tend towards:

$$\begin{aligned} y_{tput(TOTAL)} &\rightarrow \sum_{i}^{\#\ of\ routes} P_i \times \left\{ (N_i - 2) \times \frac{1}{bitrate} + \frac{1}{bitrate} \right\}^{-1} \\ &\rightarrow \sum_{i}^{\#\ of\ routes} P_i \times \left\{ \frac{N_i - 1}{bitrate} \right\}^{-1} \\ &\rightarrow bitrate \times \sum_{i}^{\#\ of\ routes} \frac{P_i}{N_i - 1} \end{aligned} \tag{6.32}$$

Using values specified in Tables 6.3, 6.4 and 6.5 in Equation 6.32 the throughput limits can be determined. The throughput limits for all the network cases are summarised by Table 6.7:

Test Name	*Theoretical Throughput Limit*
Linear 14 node network	828.264959876
Linear 7 node network	1920.0
Three Prong network	2383.78057143

Table 6.7: *Throughput Limits for Varying Packet sizes*

6.6.4 Varied Bitrate

In this section the bitrate was varied in order to determine how this would effect latency and throughput. Efficiency was determined to be constant values. The is due to the fact that ratio of data packet size to overhead does not change as bitrate changes. This means that the perceived(simulated) values do not have a great variance and can therefore be summarized by mean values:

Test Name	*Mean Efficiency Value*
Linear 14 node network	0.0504612963933
Linear 7 node network	0.128034148002
Three Prong network	0.150715578719

Now follows the average latency and throughput graphs for the various simulations using 2 sets of the *Slottime (ST)* and *SIFS* values. The first graphs were determined using the values chosen in Chapter 3. The other graphs used values that were randomly chosen and modified through experimentation. The reason these were also included was due to the fact these values were chosen for simulation first. The results gained might add some insight into the system when compared to the Chapter 3 MAC Layer values.

The Chapter 3 values are:

$$ST = 23ms \quad \text{and} \quad SIFS = 6ms \tag{6.33}$$

The "tweaked" values are:

$$ST = 47.3333ms \quad \text{and} \quad SIFS = 10\mu s \tag{6.34}$$

Average latency will first be determined for the three network cases using the Value set (6.33). This will be followed by the same simulations using Value set (6.34).

Latency Versus Bitrate

For the Values set (6.33) the latency will be as depicted in Figure 6.38. In order to determine

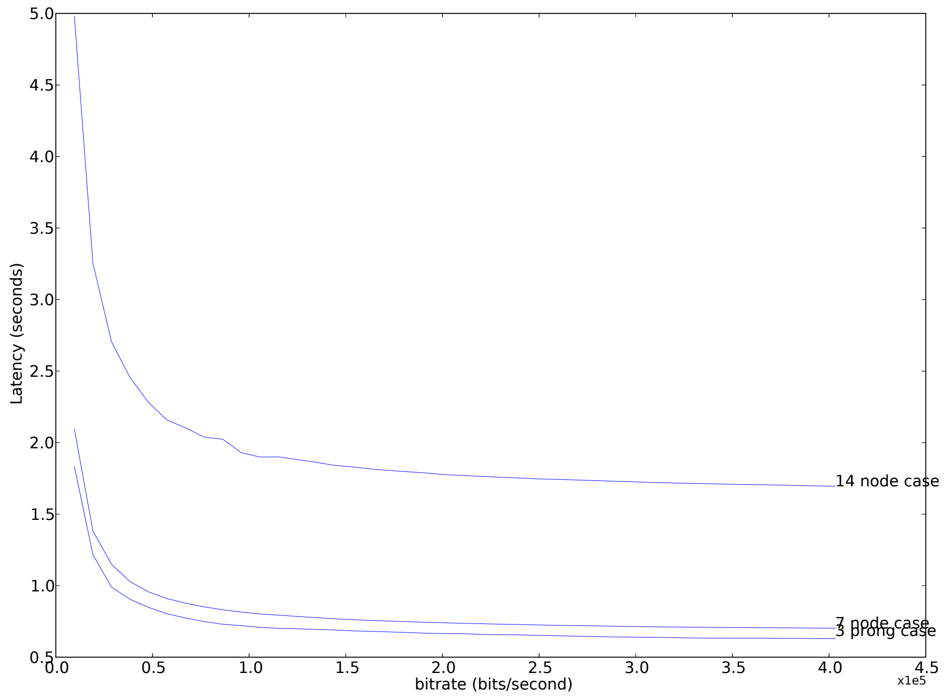


Figure 6.38: *Average Latency versus Bitrate of the Three Network Cases (ST = 23ms; SIFS = 6ms)*

an explanation for the latencies over the nodes, please keep the following in mind: The formula for latency *over one forwarding node* on Figure 6.38 is:

$$\begin{aligned} y_{latency(FW)} &= \left(\frac{B_{total}}{x} + 2SIFS + DIFS + 3.5ST\right) && (6.35) \\ \text{where } B_{total} &= B_{RTS} + B_{CTS} + B_{data} + B_{headers} + B_{route} \end{aligned}$$

The formula for latency *over the source node* would be:

$$\begin{aligned} y_{latency(SRC)} &= \left(\frac{B_{total}}{x} + 2SIFS + DIFS\right) && (6.36) \\ \text{where } B_{total} &= B_{RTS} + B_{CTS} + B_{ACK} + B_{data} + B_{headers} + B_{route} \end{aligned}$$

x the bitrate of the receiver/transmitter

B_{total} the amount of transmitted bits between transmitter and receiver (without ACKs)

The other B-values are indications of the various packet lengths, such as Ready To Send (RTS) and Clear To Send (CTS) packets (excluding packet headers). Equations 6.24 and 6.25 can be reused. After substitution of Equations 6.35 to 6.36 into Equation 6.24 latency can be described as follows:

$$\begin{aligned} y_{latency(TOTAL)} &= \sum_{i}^{\#\ of\ routes} P_i \times \left\{ (N_i - 2) \times y_{latency(FW)} + y_{latency(SRC)} \right\} \\ &= \sum_{i}^{\#\ of\ routes} P_i \times \left\{ (N_i - 2) \times \left(\frac{B_{total}}{x} + 2SIFS + DIFS + 3.5ST \right) + \right. \\ & \left. \left(\frac{B_{total}}{x} + 2SIFS + DIFS \right) \right\} \end{aligned}$$

where	P_i	the calculated route probability of route i given in Table 6.1 column 2
	N_i	the length (in number of hosts) of route including source and destination

This latency equation's limit was then determined as bitrate (x) strives to infinity. The result is as follows:

$$\begin{aligned} y_{latency(TOTAL)} \rightarrow & \sum_{i}^{\#\ of\ routes} P_i \times \left\{ (N_i - 2) \times \left(2SIFS + DIFS + 3.5ST \right) + \right. \\ & \left. \left(2SIFS + DIFS \right) \right\} \end{aligned} \qquad (6.37)$$

Firstly, the Three Prong Network's latency limit will be determined, followed by the latency explanation for the other networks.

Three Prong Network case By substituting the values of Table 6.3 into Equation 6.37 the it can be determined that the fitted curve tends toward a latency of $0.5751704s$.

The 7 Node Network can be determined in the same way using Equation 6.37. As can be seen in the route preference graph of Figure 6.29 that the packet traffic is split in half between the two routes. This is also consistent with the route preference algorithm. The substitution of the values of Table 6.4 into Equation 6.37 will produce a value of $0.642s$.

The 14 Node Network The substitution of the values found in Table 6.5 in Equation 6.37 will produce a value of $1.59734105785s$.

Figure 6.39 shows the Equation 6.37 as a dashed ('--') line. The dotted lines indicate the limits.

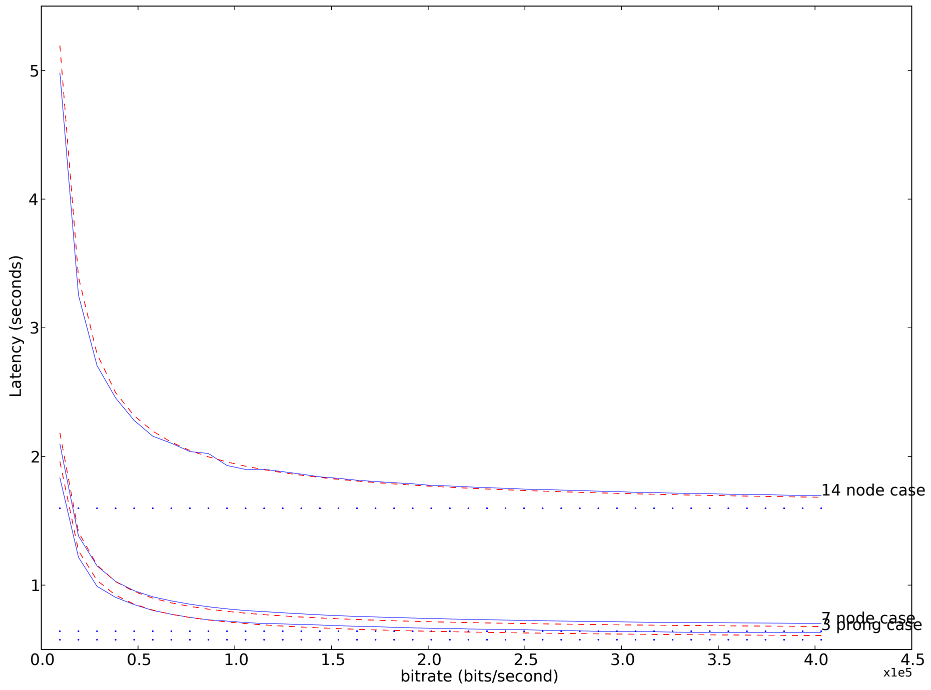


Figure 6.39: *Average Latency versus Bitrate of the Three Network Cases (Theoretical Value and limits indicated)*

To summarise the fitted latency curves tend toward the following values:

Test Name	*Latency curve tends towards*
Three Prong network	0.5751704s
Linear 7 node network	0.642s
Linear 14 node network	1.59734105785s

Table 6.8: *Approximate latency Curve limits for varying bitrate*

Please note that the values of Table 6.8 are not accurate, due to the nature of the data that it is trying to match. These values are merely indications to give an idea of how the 3 network cases compare with each other. The lowest latency limit according to Figure 6.38 and Table 6.8 will always be the Three Prong network due to the fact that it is always smaller than the 7 node network average latency. *Latency will be low in a network that has multiple paths to its disposal.*

Resimulation of system with different SIFS and Slottimes

The network simulations were rerun for Value Set (6.34). The values are increased for all cases. It is clearly a side-effect of the slottime (ST) value that is approximately double the previous simulation situation. Care should therefore be taken in MAC Layer design not to make delays too long.

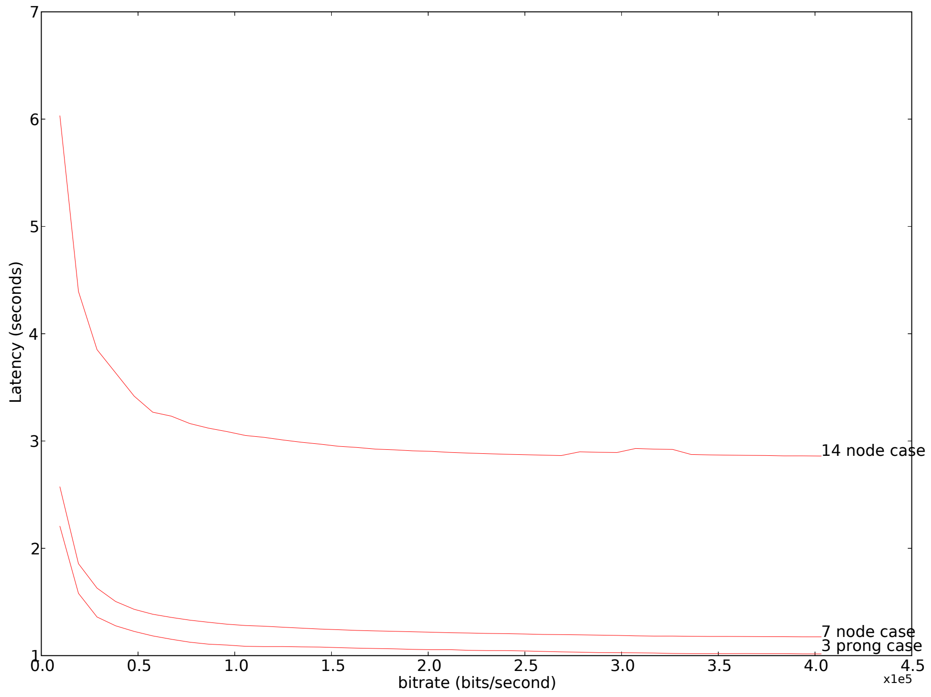


Figure 6.40: *Average Latency versus Bitrate of the Three Network Cases ($ST = 47.3333ms$; $SIFS = 10\mu s$)*

Incorrect timing values may slow the network unnecessarily. These simulations try to be as generic as possible for various makes of hardware layers. ST and SIFS parameter values should be refined by keeping into consideration the actual hardware used. An example of this can be found in the [4]. Values such as `aRxTxTurnaroundTime` and `aRxRFDelay` are very hardware specific and does affect the interaction between the PHY layer (hardware layer) and MAC Layer.

Throughput Versus Bitrate

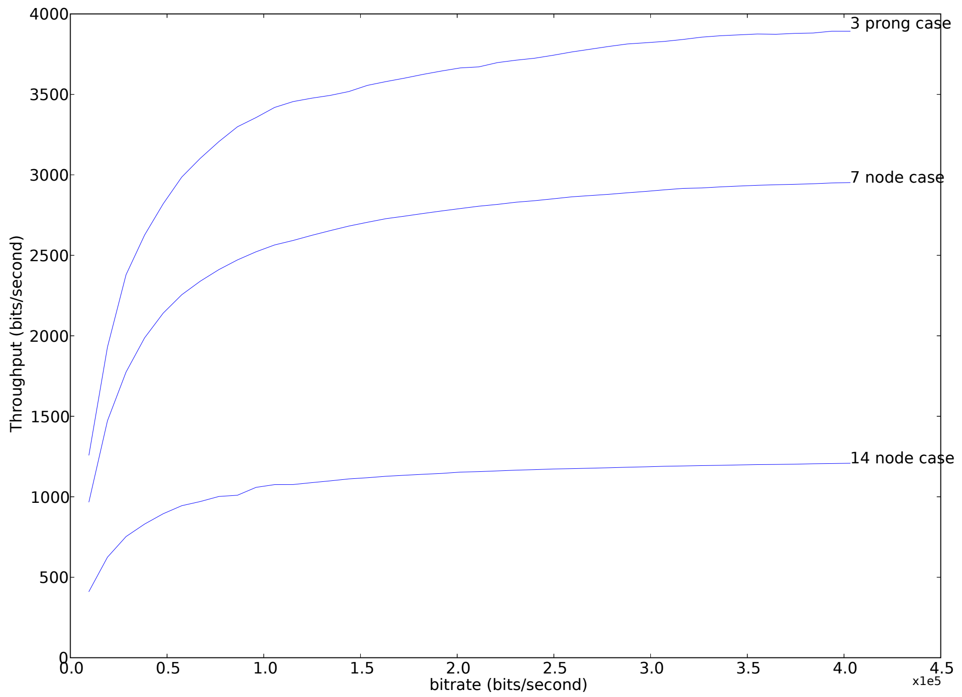


Figure 6.41: *Average Throughput versus Bitrate of the Three Network Cases (ST = 23ms; SIFS = 6ms)*

The throughput value, as previously stated is the quotient of the packet size over the latency. For throughput calculation, Equations 6.30 and 6.29 can be reused. The latency calculations for the latency is given by substituting Equation 6.26 and 6.27 into Equation 6.29. This leads to a throughput formula that is given by:

$$y_{tput(TOTAL)}$$

$$= \sum_i^{\#\ of\ routes} P_i \times \frac{packetsize}{\left\{(N_i - 2) \times y_{latency(FW)} + y_{latency(SRC)}\right\}}$$

$$= \sum_i^{\#\ of\ routes} P_i \times \frac{packetsize}{\left\{(N_i - 2) \times \left(\frac{B_{total}}{x} + 2SIFS + DIFS + 3.5ST\right) + \left(\frac{B_{total}}{x} + 2SIFS + DIFS\right)\right\}}$$

Where packetsize is constant (2024 bits).

The latency limit as bitrate approaches infinity ($x \to \infty$), is then given by:

$$y_{tput(TOTAL)} \to \sum_i^{\#\ of\ routes} P_i \times \frac{packetsize}{\left\{(N_i-2)\times\left(2SIFS+DIFS+3.5ST\right)+\left(2SIFS+DIFS\right)\right\}} \tag{6.38}$$

The theoretical curves with the limit values have been indicated on Figure 6.42. The limit values are also given in Table 6.9.

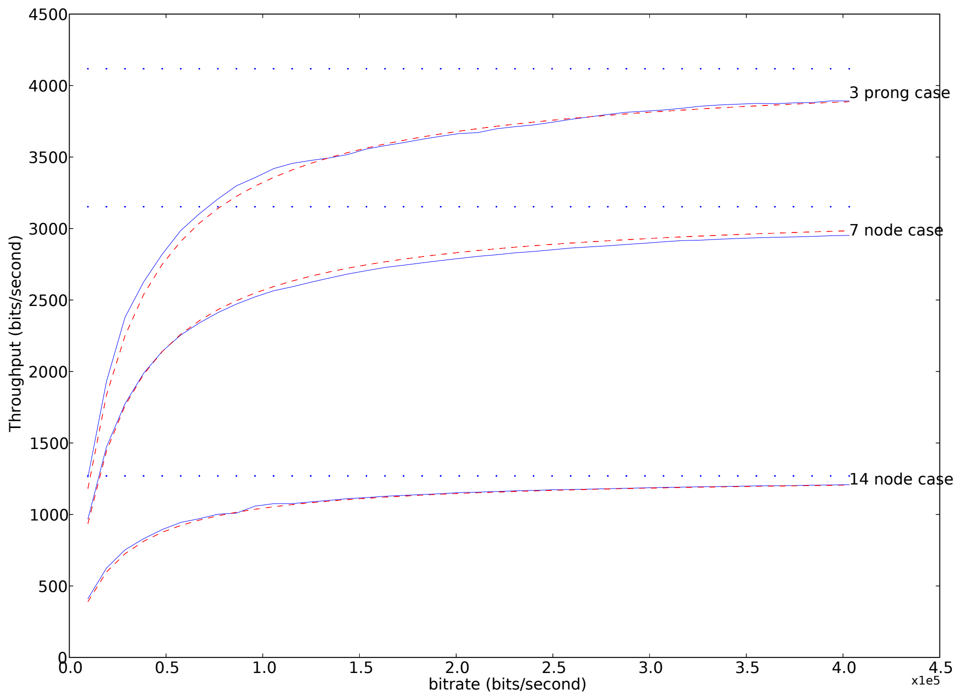


Figure 6.42: *Average Throughput versus Bitrate of the Three Network Cases (Theoretical and Limit values indicated)*

Test Name	*Throughput Value as bitrate* $\to \infty$
Three Prong network	4117.40545265 bits/second
Linear 7 node network	3152.64797508 bits/second
Linear 14 node network	1269.62514758 bits/second

Table 6.9: *Throughput Limits for Varying Bitrates*

Resimulation of system with different SIFS and Slottimes continued...

In Figure 6.43, the SIFS and Slottimes of Value Set 6.34 was used in order to redetermine the average throughput with varying bitrate. It can be seen that throughput in this case is not as high the throughput achieved in Figure 6.41.

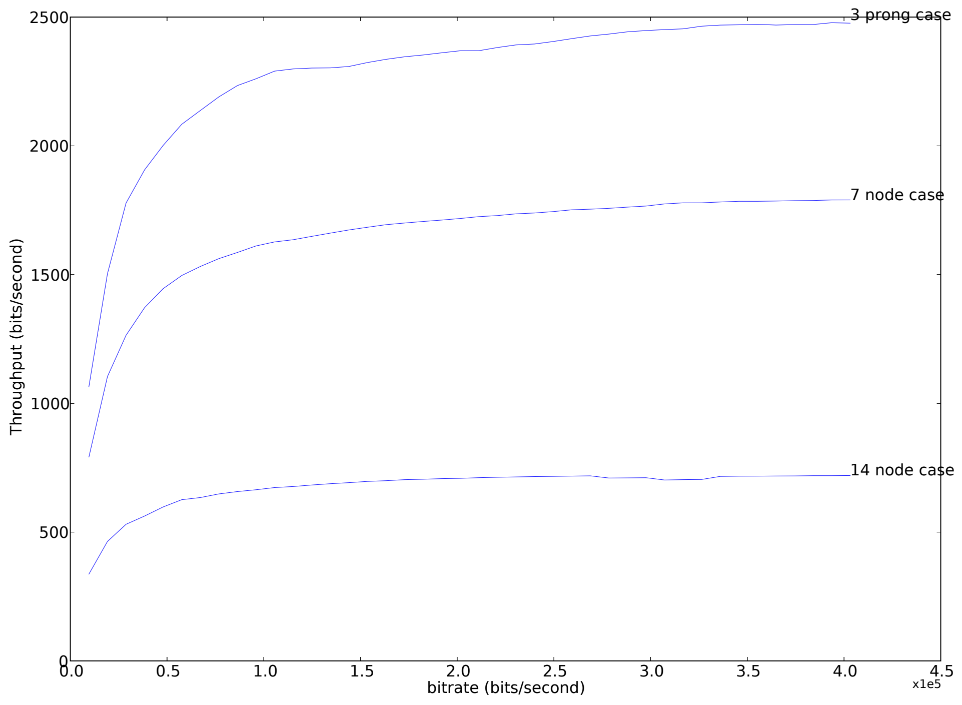


Figure 6.43: *Average Throughput versus Bitrate of the Three Network Cases (ST = 47.3333ms; SIFS = 10μs)*

Test Name	*Maximum Throughput Value*
Linear 14 node network	719.678450251
Linear 7 node network	1790.02909193
Three Prong network	2478.16581376

6.6.5 Simulation of a Cluster Topology Network Situation

The topology of Figure 6.44 was set up in order to test the average performance of the network with a more cluster-like topology. For ad hoc networks, topologies can vary from clusters to paths to combinations of the two. The network does not even have to be fully connected. These factors imply that there is no one network that can represent all network subclasses. Even if such a method of abstraction did exist, it is beyond the scope of this work. For the following network, the average of success rate, throughput, latency and efficiency was monitored.

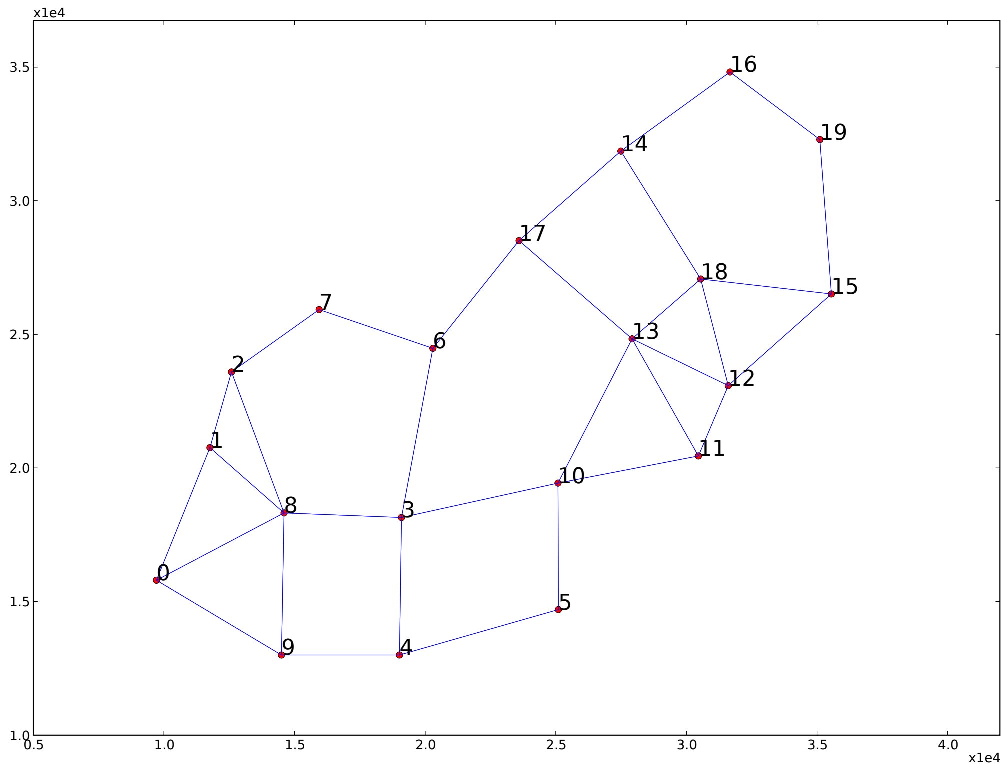


Figure 6.44: *20 Node Cluster*

`Host[18]` was chosen as the satellite up/downlink host. The other nodes were all set up to be source hosts. Simulations were rerun for various average arrival intervals, starting at 10s and ending in 200s. The other hosts were instructed to transmit to all their neighbours. Packets were logged at the source and arrivals were logged at the destination. Every exchange's success rate is then determined and the average of all the exchanges are then determined to get a data point at a specific arrival interval. The results have been plotted in Figure 6.45.

Note that the efficiency has also been indicated on the first subplot as a dashed line and (as expected) remains fairly constant. The average throughput does have an inverse relationship to the average latency. Success rate is close to 100% after an average arrival interval of 60s.

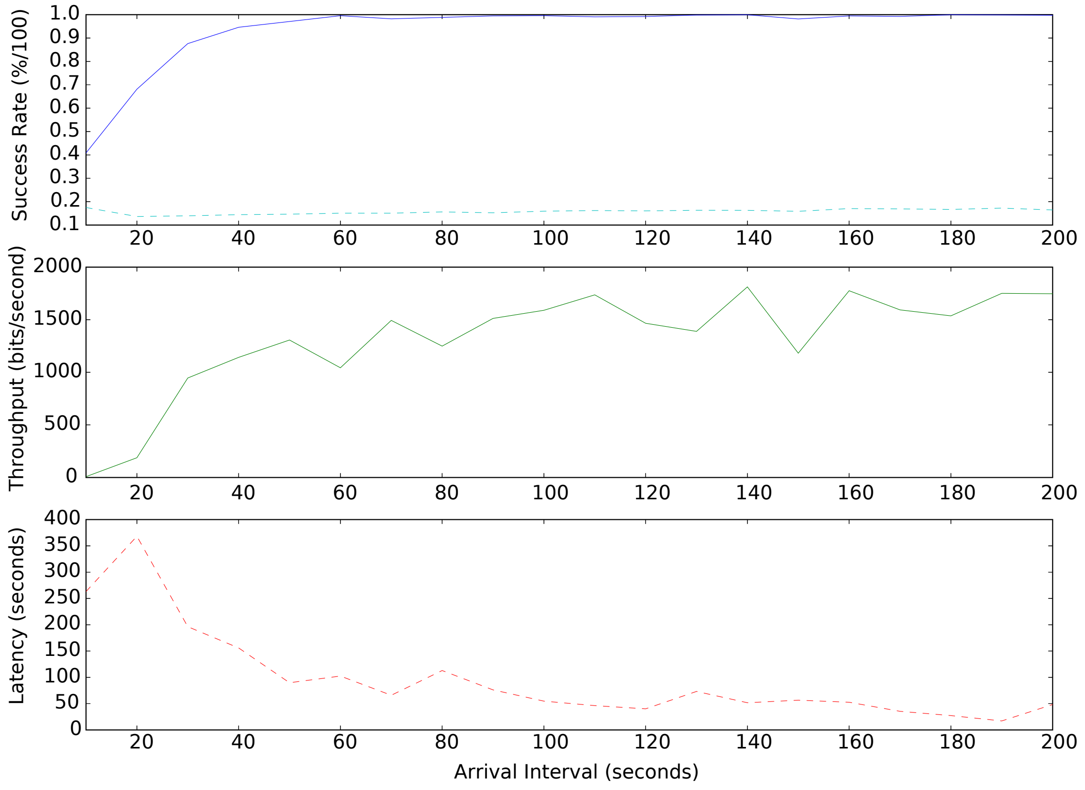


Figure 6.45: *Cluster Average Characteristics*

Average latency is very high at low average arrival interval, but drops dramatically as the arrival interval decreases.

6.7 Summary

In this chapter, various factors of protocol simulation was covered. The following was done in order to accomplish this:

- Various simulation platforms were found and evaluated for their relative "ease-of-use" without losing the level of sophistication required in simulation software. Determining which simulation framework to use is a non-trivial and time-consuming undertaking.
- A background in discrete event simulation was given in terms of two main simulation objects-classes namely:

  1. the `cMessage`-class and
  2. the *Future Event Set* (FES).

  Modelling of timers and events was also covered in this section.
- A more protocol specific simulation model description was given. Binding timers in the event queue (or FES) was a technique to allow an indefinite amount of route requests to be timed independently. This method also decreases data structure usage by avoiding explicit timer-`RREQ` mapping.
- In order to avoid simply outputting raw packets arrival events (with their corresponding routes) a new graphing method was developed. These *Route Preference graphs* indicate the bias that some hosts may have for certain routes. This can be seen by the area the graph occupies on the figure. A larger area for a route implies that the route is more *popular*. This means that the Network Layer uses this path more in order to forward packets. Smaller areas on graphs indicate less popular routes. Route failures and activations are easily discernible through this method.
- A simulated situation was developed and explained using queueing theory and Jackson queueing networks. It was shown that the Lottery Route Selection system has the desired result as was seen in the gradients of Figure 6.19. A initial service rate value was determined and refined based on simulation and further insights gained. Better theoretical models were then determined. *The Binary Exponential Backoff was determined to be an important factor in the service rate determination.*
- Latency over a route was also determined using previously determined theoretical knowledge and compared to simulated results.
- Three network topologies was used in order to show how variable packet size and bitrate would allow different average values for *Latency*, *Throughput* and in some cases *Efficiency*. Using theoretical understanding of the networks, the behaviour could be explained and values matched with a low amount of error.

- An “incorrectly” chosen set of simulation values was also included to show, by contrast, what difference the MAC Layer parameters would make to system performance.

- Finally, a more realistic network topology was created and simulated with varying (increasing) arrival intervals (which implies decreasing arrival rates). Success rate was shown to be low until about average arrival intervals of 60s.

The following chapter will give a thesis summary and will then deal with possible usages and applications.

Chapter 7

Discussions And Comments

7.1 Project Summary

The goal of this thesis was to determine a least cost routing and MAC strategy for a satellite assisted rural ad hoc network. This "strategy" can be divided into two parts:

- *A Routing (or Network) Layer* – Based on the Dynamic Source Routing Protocol.
- *A Medium Access Control (MAC) Layer* – Based on the IEEE 802.11 Distributed Coordination Function (CSMA/CA).

This strategy was developed by means of *cross layer design*. Designs of this nature allows some tasks/responsibilities that are traditionally done by one layer be passed to another. For example, a technique used, namely *Promiscuous Listening*, causes the MAC Layer not to discriminate which packets are meant for it and which packets are not. (This causes the Network Layer to *overhear* packets meant for other hosts.) The responsibility for this distinction then rests with the Network Layer. This, however, allows the Network Layer to gain important routing information by simply overhearing packets.

Additional Developement and Investigation:

- *Simulation Modelling* – A Network and MAC Layer model was devised that will allow multiple network topologies to be created. Some previous attempts to devise a `MACA` MAC Layer protocol was also attempted, but was not used in the final simulation model. Eventually DSR and IEEE 802.11 DCF (CSMA/CA) was chosen and modelled.
- *Theoretical Analysis* – In order to interpret the behaviour of simulations, a theoretical framework was devised. By using this framework various model parameters can be predicted.
- *A Tool for Rural Ad hoc Network Development* – A collection of programs and scripts have been devised that implement methods for determining and visualising various performance parameters. By using OMNeT's initialization script various simulation

situations can be created. Any performance parameters (results) that were generated by the simulation model can then interpreted (by various scripts in *Python* and *Bash*) and processed for inspection by the user.

7.2 Summary Of Results

7.2.1 Network Simulation Evaluation via Queueing theory

Section 6.5 described a method to explain the protocol behaviour in terms of Queueing theory and Jackson Queuing networks. A host was modelled as a server and queue. The server has an average service rate, μ, and packets arrive at the queue with an exponential arrival interval. This implies that the packets arrivals can be characterized by a *Poisson process* (according to [19]).

- The Lottery Route Selection Algorithm's function was verified, by determining the arrival rate for packets of a route in a three route network. The arrival rate of a route was shown to coincide with the formula:

  $$\lambda_{route_x} = \lambda_{source} \times q_{route_x}$$

  λ_{source} refers to the source host arrival rate the rate at which packets are ready to be sent to the rest of network. Table 6.1 shows a summary of the theoretical route probability values as well as the simulation determined route probability values. The small difference between these two columns proves that the Lottery Route Selection Algorithm functions as expected.

- A theoretical approach to calculating service rate was determined iteratively. It was finally determined that the Binary Exponential Backoff (BEB) strategy, caused an average backoff delay for a forwarding host that can then be determined by:

  $$\begin{aligned} T_{avrg.backoff} &= DIFS + 3.5 \times ST \\ \text{with} \quad DIFS & \quad \text{refers to DCF Interframe Space} \\ ST & \quad \text{refers for Slottime} \end{aligned}$$

  This equation assisted in the deduction of Equations 6.11 and 6.12. This enabled calculation of the final service rate values. The *average source host service rate*, $\mu_{SRC(service\ rate)}$, was determined to be given by Equation 6.14. Similarly the *average forwarding host service rate*, $\mu_{FW(service\ rate)}$, was given by Equation 6.13 for the source host.

- By applying these λ- and μ-values to Little's Law, the *packet's delay* over the host can be determined (Equations 6.16 and 6.18):

$$T_{FW} = \frac{1}{\mu_{FW(service\ rate)} - \lambda_{FW(host)}}$$
$$T_{SRC} = \frac{1}{\mu_{SRC(service\ rate)} - \lambda_{SRC}}$$

The *average queue length* experienced by a host can also be determined by multiplying T_{FW} (for a forwarding host) or T_{SRC} (for the source host) to the arrival rate values $\lambda_{FW(host)}$ and λ_{SRC} respectively.

- The *Route Latency* can be determined by applying Equations 6.16 and 6.18 of a route to Equation 6.20. For N hosts the latency of a route is given by:

  $$T_{latency} = (N-1)T_{FW} + T_{SRC}$$

  This equation was applied for all three routes in the topology and according to Figure 6.26 resulting average latency determined was a very close approximation to the simulated average latency.

7.2.2 Variation of Packet Size and Bitrate

Three network topologies were used to test the effects of various Packet Size and Bitrate values. The parameters tested were *Average Latency, Average Throughput and Average Efficiency.* These simulated results were then interpreted and explained in terms of:

- the *Timing Values* specified previously in the MAC Layer specification (Chapter 3).
- the *Lottery Route Selection Algorithm* and the effect of route probabilities.

This theoretical description was also compared with the simulated values in order to verify accuracy. *Theoretical values matched simulated values very closely and with a very small error.*

Simulations that Varied Packet Size

- *Average Efficiency* – Average efficiency follows the following relationship (according to Equation 1.1):

  $$\text{Efficiency} = \frac{A}{B+A}$$

  where A refers to the packet length

  B refers to the total overhead in an exchange

  From this equation it can be seen that as the packet size increases efficiency increases as well. *Please note that bit error rate would also change if packet size increases.* During simulation a summation of all overhead was registered for each of the Network Layer packets exchanged. It can be seen that networks with multiple routes are much more efficient than networks that have a more linear topology. This is partly due to the routing overhead that source routing incurs. Other overhead is caused due to MAC Layer control packets (`RTS`, `CTS` and `ACK`) and header bits. Retransmissions can also be seen as overhead.

- *Average Latency* – As packet size increases, average latency increases linearly. The gradient is given by Equation 6.28:

$$\frac{dy_{latency}}{dx} = \frac{1}{bitrate}\sum_{i}^{\#\ of\ routes} P_i \times (N_i - 1)$$

  where P_i refers to the $route_i$'s probability to be chosen (or packet route bias)
  N_i refers to the total hosts in $route_i$ including source and destination

  Bitrate is constant. It can be seen that the gradient with which the route latency increases is directly proportional to the route length N_i. In Figure 6.35 the theoretical gradient determined by Equation 6.28 matches the gradient from simulation results very closely.

- *Average Throughput* – The simulated average throughput was analysed and a theoretical analysis was subsequently also done. Equation 6.31 was the result of the analysis. It was also shown that the throughput tends toward Equation 6.32:

$$y_{tput} \rightarrow bitrate \times \sum_{i}^{\#\ of\ routes} \frac{P_i}{N_i - 1}$$

  where P_i refers to the $route_i$'s probability to be chosen (or packet route bias)
  N_i refers to the total hosts in $route_i$ including source and destination

  Bitrate is constant. In this case it has been shown that the upper bound of the average throughput of a route is inversely proportional to the number of hosts in the route (N_i). Once again, simulated and theoretical values were shown to match each other very closely, as can be seen in Figure 6.37.

Simulations that Varied Bitrate

In this class of simulations, the three network topologies was once again simulated multiple times. In this case the effect of various bitrates for the physical layer was determined. The effect on *Average Latency, Average Throughput and Average Efficiency* was once again determined.

- *Average Efficiency* – As expected, average efficiency will not be influenced by the bitrate of the system. Average efficiency will remain constant as long as the exchanges does not cause too many collisions and retransmissions.

- *Average Latency* – Average latency does decrease with an increase in bitrate. This can be seen by Figure 6.39. It is apparent however, that average latency does tend towards a certain value as bitrate increased. This was determined and specified in Equation 6.37:

$$y_{latency} \;\rightarrow\; \sum_{i}^{\#\ of\ routes} P_i \times \left\{ (N_i - 2) \times \Big(2SIFS + DIFS + 3.5ST \Big) + \Big(2SIFS + DIFS \Big) \right\}$$

- *Average Throughput* – An increase in bitrate causes an increase in the average throughput of the system. This can be seen in Figure 6.42. The average throughput also tends towards a value given by Equation 6.38:

$$y_{tput} \;\rightarrow\; \sum_{i}^{\#\ of\ routes} P_i \times \frac{packetsize}{\left\{ (N_i - 2) \times \Big(2SIFS + DIFS + 3.5ST \Big) + \Big(2SIFS + DIFS \Big) \right\}}$$

It has been determined that changing the bitrate will change the latency and throughput in a predictable fashion. This was shown and proven through theoretical analysis of simulation results. It must be made clear, that even though the Slottime and SIFS time was not designed for higher speeds than 9600 bits/second that they will still be compatible with these values. Higher bitrates would simply not be treated as optimally as possible. Lower bitrates would, however, experience timing conflicts and errors.

In this set of simulations the effects of increasing bitrate and packet size was determined. These values have then been interpreted which, in turn, resulted in a theoretical framework that allows interpretation (and prediction) of results.

Simulation of a Cluster Topology Network Situation

In this simulation the averages of *success rate, throughput and latency* for a cluster of nodes was determined for an increasing average arrival interval. With 19 out of 20 source hosts it was shown that the success rate is approximately 100% for an average arrival interval of 60 seconds and higher. Throughput and latency experienced an inverse relationship. This implies that, throughput increased while latency decreased for an increasing arrival interval.

7.3 Summary of Contributions

- *The Routing (or Network) Layer* – This layer has been based on the Dynamic Source Routing Protocol. Modifications that were devised and described in Chapter 5 are as follows:

  - *Lottery Route Selection Algorithm* – In which the Lottery Process Scheduling Algorithm was adapted to cause packets to be more *biased* towards certain routes based on a chosen metric. In this implementation inverse route length was the metric of choice. Packets tended to be more biased towards routes with shorter route lengths.
  - *Implicit Route Maintenance* – Instead of dealing with route failures in an explicit fashion (such as `RERR`-packets) it was decided to **resubmit** the `DATA`-packet to the Network Layer after an unsuccessful exchange. Unsuccessful exchanges include cases were destination hosts fails to respond with the customary acknowledgement (`ACK`) packet. After resubmission to the Network Layer, another route will be chosen and embedded into the header of the `DATA`-packet. Retransmission will then be attempted.

- *The Medium Access Control (MAC) Layer* – For this layer the main contribution is the adaptation of timing values such as Slottime and `SIFS`-time to be compatible with the 9600 bits/second bitrate specified in the design constraints of Chapter 1. It was also interfaced with the Network Layer by techniques such as promiscuous listening.

- *A Theoretical Model* – That can be verified by simulation resulted in a *tool* that may be used for a future deterministic approach to predict the performance of networks of this type, that typically contain satellite based gateway nodes.

7.4 Possible Areas for Improvement

7.4.1 The Network Layer

Route Bias Adaptation: The route bias (preference) value of the route is currently determined before every data packet transmission. Optimization can be done by modifying this behaviour as follows:

- Currently, all routes that have been determined for one destination are taken into account when determining route preference values. This is done before a data packet transmission. This could be optimised by changing the behaviour to precalculating the route preference value of the route after every route reply from the destination and storing it explicitly.

- This stored value can then be *increased* by a certain value whenever a successful routing operation has taken place. A *decrease* could also be applied to this value for any failures of a route during packet delivery. Implementing such a system might however be difficult, due to the lack of any acknowledgement system on the Network Layer. A cross layer approach that piggybacks the success information over the MAC Layer's `ACK`-packets may be a solution.

Caching: This technique allows a network to cut down on request overhead by storing frequently requested information close to requesting hosts. This approach may even improve the perceived latency of the network from the user's perspective. It may also decrease bandwidth usage of the network in its entirety.

7.4.2 The MAC Layer

In order to improve performance on the MAC Layer level a few possible optimizations could be suggested:

- *Determination of Optimal Timing Values.* – Timing values (such as SIFS, DIFS or Slottime) must not be too short or too long due to the fact that both occurrences will lead to an inefficient wastage of bandwidth. Parameter Optimization may prove useful in determining an effective set of delay values. A parameter such as latency can be optimized for. The set of values that gives a lowest latency solution can be seen as the optimal solution.

- *A solution is required for the Exposed Terminal Problem.* – The MAC Layer protocol of this thesis does not deal with this problem. Inefficiency is the result of the unnecessary delays caused by exposed and hidden terminal problems. This is mainly due to the fact that the MAC Layer protocol of this thesis was based on the CSMA/CA medium access method as described in specification [4]. Protocols such as MACA and MACAW does deal with CSMA/CA's exposed terminal problem but causes, in turn, different kinds of hidden/exposed terminal problems [20].

7.4.3 Analysis of Network Performance when taking various Subnetworks into Consideration

This thesis only dealt with the network system on a subnetwork level. Simulation of various hosts from a subnetwork communicating via a satellite to hosts in another subnetwork could be done. For the case of the supernetwork (combination of these smaller networks) variables such as latency and throughput should be determined. Some of the traffic generated could be forwarded to satellite up/downlink hosts in order for the satellite to transfer packets from the current subnetwork (subnetwork of the source host) to the subnetwork of the destination host. The delay incurred by waiting for the satellite to arrive and may even cause some packets to be lost (due to packet-aging). Flow control must therefore take into consideration how often the satellite would cross a certain point on the surface (when assuming a low earth orbit). Satellites in geostationary orbit can simply be seen as another stationary host (from a Network Layer point of view).

7.4.4 "Long-range" Communication in the context of Rural ad hoc networks

Low Cost Rural Networks

Rural communications can benefit from 802.11(WiFi)-like communications infrastructure. Hardware is relatively inexpensive and highly adaptable due to open-source software drivers (such as MadWifi [21]) that allows users to set various parameters of the MAC protocol. In regards to hardware, some inexpensive alternatives have been suggested to WiFi-Antennas using tin-cans in [22]. In a rural situation, cheap alternative solutions of this nature may prove invaluable.

Private Neighbourhood networks

Ad hoc networks (sometimes inaccurately known as mesh networks) can also be used in order to allow relatively low-cost communication in a neighbourhood. Users may install ad hoc hardware into residences, thereby allowing applications such as Voice-over-IP (or VoIP), file-sharing and allow users to communicate with each other on a much lower cost than current available infrastructure will allow. Due to the low maintenance that such a network requires, as well as the ease at which a host becomes part of a network, the network can grow simply by users adding hosts to the network. *One drawback of such a VoIP communication network is that it will make per-call billing for a company close to impossible.* WiFi based networks incorporating VoIP exists already as can be seen in [23]. Other services may be billed in a "per-transaction" basis. If one host has access to a resource any request can be billed to that source of specific packets.

Example: An user of an ad hoc network may have need of a service such as Internet access. Another host may be owned by an Internet Service Provider. The only way the two parties are connected would be via the ad hoc mesh network. Transmitting the packets to and from the ISP would cost the user nothing[1]. Due to the fact that the packets are marked by the user (the source address) the billing info can be stored at the ISP-host.

[1]Actually, node power usage would be shared by all the users over the network.

Bibliography

[1] Carl A. Waldspurger and William E. Weihl, “Lottery Scheduling: Flexible Proportional-Share Resource Management,” *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 1–11, Nov. 1994.

[2] Webster’s Online Dictionary. (2006, Aug.) Webster’s online dictionary. [Online]. Available: http://www.websters-online-dictionary.org/definition/AD+HOC+NETWORK

[3] L. Kleinrock and F. Tobagi, “Random Accesss Techniques For Data Transmission Over Packet-switched Radio Channels,” *Proceedings National Computer Conference*, 1975.

[4] “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications,” *IEEE std 802.11-1999*, 1999.

[5] P. Karn. (2006, Aug.) MACA - A New Channel Access Method for Packet Radio. [Online]. Available: http://www.ka9q.net/papers/maca.html

[6] F. Talucci and M. Gerla, “MACA-BI (MACA By Invitation): A wireless MAC protocol for high speed ad hoc networking.” *Proceedings of IEEE ICUPC 1997*, vol. 2, no. 6, pp. 913–917, Oct. 1997.

[7] M. Löbbers and D. Willkomm. (2006, Aug.) A mobility framework for OMNeT++ version 1.0a6. [Online]. Available: http://www.omnetpp.org/

[8] Michael Ignatius Brownfield, “Energy-efficient wireless sensor network mac protocol,” Master’s thesis, Virginia Polytechnic Institute and State University, Mar. 2006.

[9] C. Perkins and E. Royer, “Ad hoc on-demand distance vector routing,” *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, Feb. 1999.

[10] D. Johnson, D. Maltz and Y. Hu. (2004, July) The dynamic source routing protocol for mobile ad hoc networks. [Online]. Available: http://www.ietf.org/internet-drafts/dratf-ietf-manet-dsr-10.txt

[11] Samir Ranjan Das and Charles E. Perkins and Elizabeth E. Royer, “Performance comparison of two on-demand routing protocols for ad hoc networks,” *INFOCOM*, pp. 3–12, 2000.

[12] (2006, Aug.) The network simulator (ns-2). [Online]. Available: `http://nsnam.isi.edu/nsnam/index.php/User_Information`

[13] R. Barr. (2005, Apr.) JiST/SWANS java in simulation time / scalable wireless ad hoc network simulator. [Online]. Available: `http://jist.ece.cornell.edu/`

[14] Wikipedia. (2006, Aug.) The Network Simulator (ns-2). [Online]. Available: `http://en.wikipedia.org/wiki/Ns2`

[15] K. Müller, S.D.W. Frost and T. Vignaux. (2006, Oct.) SimPy simulation in python. [Online]. Available: `http://simpy.sourceforge.net/`

[16] A. Varga. (2006, Aug.) OMNeT++: Discrete event simulation system. [Online]. Available: `http://www.omnetpp.org/`

[17] W. Giffin, *Queueing, Basic theory and applications.* Grid, Ohio.

[18] E. Gelenbe and G. Pajolle, *Introduction to Queueing Networks.* Wiley, 1987.

[19] Wikipedia. (2006) Exponential Distribution. [Online]. Available: `http://en.wikipedia.org/wiki/Exponential_distribution#Occurrence_and_applications`

[20] M. Wu. (2006, Nov.) A Survey of MAC Protocols in Ad Hoc Networks. [Online]. Available: `http://www.utdallas.edu/`

[21] MadWifi: Long distance links with madwifi. [Online]. Available: `http://madwifi.org/wiki/UserDocs/LongDistance`

[22] How To Build A Tin Can Waveguide WiFi Antenna for 802.11(b or g) Wireless Networks or other 2.4GHz Applications. [Online]. Available: `http://www.turnpoint.net/wireless/cantennahowto.html`

[23] Fon Community Website. [Online]. Available: `http://en.fon.com/`

Appendix A

Assorted Scripts

Route Preference Graphing Script

This script uses the simulation scalar filename and the host number to be highlighted as parameters. Matplotlib must be installed.

Written by Stephan van Ellewee

```
#!/usr/bin/python
from pylab import *;
from math import *;
import sys
import re;

#Regular expression to parse the output.sca file from OMNeT++
r_re = re.compile('scalar([\ \t]*)"sim.host\[([0-9]*)\].net"([\ \t]*)"ACCUM...⇒
...ROUTE:INIT:host\[([0-9]*)\] ([host0-9\[\]\(\)]*)"([\ \t]*)([0-9e\-\+\.]*)')...⇒
...;
# read in all lines of a file....
filename = sys.argv[1]
print "looking for",filename
l = ''
l = "".join(file(filename).read())
routeT ={}
for i in l.split("\n"):
    if (len(i)>0) and (i[0] == "#"): # ignore comments
        pass
    else:
        # ... and parse
        pp = r_re.findall(i)
        if len(pp) > 0:
            #print i,pp
            if routeT.has_key(pp[0][4]):
                routeT[pp[0][4]].append(float(pp[0][-1]))
            else:
                routeT[pp[0][4]] = [float(pp[0][-1])]

# find the maximum value
maxVal = 0
for i in  routeT:
    if maxVal < max(routeT[i]):
        maxVal = max(routeT[i])
    print i, maxVal

#determine the window size
percentage = 0.05
samplingPeriod = round(percentage*maxVal)
print samplingPeriod
xaxis = list(arange(0,maxVal,samplingPeriod))

# process arrivals...
R = {}
T = []
for w in range(0,len(xaxis)-1):
    S = 0

    for i in routeT:
        l = len(find([q > xaxis[w] and q < xaxis[w+1] for q in routeT[i]]))
        if R.has_key(i) == False:
            R[i] = [ l ]
        else:
            R[i].append( l )

N = {}
fig_i = 0;
figure(fig_i);fig_i += 1;grid(True);

xlabel("time (s)",fontsize = 25);
for w in range(0,len(xaxis)-1):
    S = 0
    for i in R:
        S += R[i][w]

    if S > 0:
        for i in R:
            if N.has_key(i):
                N[i].append((xaxis[w] ,R[i][w]/(1.0*S)))
            else:
                N[i] = [ (xaxis[w],R[i][w]/(1.0*S)) ]

Y =[ 0 for z in range(0,len(N.values()[0]))]
layerY = [ 0 for z in range(0,len(N.values()[0]))]
colors = ['r','b','g','m','c']
c_i =0;
for  i in N:
    # regular expression searches host under scrutiny
    re_subcheck = re.compile('(host\[([0-9]*)\]*)');
    findings = re_subcheck.findall(i);
    coords = []
    # for looping through colors
    col = colors[c_i]; c_i += 1; c_i = c_i % len(colors)
    if len(findings) > 0:
        # if host was found...
        if len(list(find([ww[1]== sys.argv[2]  for ww in findings]))) > 0:
            # plot the polygons...
            layerY=[ii for ii in Y]
            Y1 = layerY[-1];
            reverse_X = [z[0] for z in N[i]]
            X1 = reverse_X[-1]
            reverse_X.extend(reversed([z[0] for z in N[i]]))
            Q= add(layerY,[z[1] for z in N[i]])
            layerY.extend(reversed(Q))
            Y2 = Q[-1]
            fill(reverse_X,layerY,col)
            # draw the route names..
            middel = (X1,mean([Y1,Y2]))
            t = text(middel[0],middel[1],i);t.set_fontsize(18);
    # draw the outlines of the routes preferences in blue
    Y = add(Y,[z[1] for z in N[i]])
    X= [z[0] for z in N[i]]
    plot(X,Y,'b');
    plot(X,Y,'bx')
ax = axes(); setp(ax.get_xticklabels(), fontsize=25);

AXIS =  axis()
AXIS[1] = AXIS[1]*1.5
axis(AXIS)
show();
```

Lottery Scheduling Algorithm demonstration

This script is a demonstration of lottery scheduling.

Written by Stephan van Ellewee

```
#!/usr/bin/python
from random import *
from pylab import *

#test case....
P = [ 1, 2, 3 ,10 ]

# the actual lottery scheduling function...
def lottoSch(p):
    total = sum(p)
    #print total
    sigma = 0;
    rndnumber = uniform(0,total);
    #print rndnumber
    for i in range(0,len(p)):
        sigma += p[i]
        if (sigma > rndnumber):
            return i

# get the inverse weight.. for application in Lottery Route Selection...
Pinverse= []
for i in P:
    if i <=0 :
        Pinverse.append(0)
    else:
        Pinverse.append((1.0/i)**2)
```

```
# generate a few random variables
values = []
ivalues= []
for i in range(0,1000):
    values.append(lottoSch(P))
    ivalues.append(lottoSch(Pinverse))

# check the values by means of a histogram
close("all")
figure(0)
hist(values)
# check the inverse values by means of a histogram
figure(1)
hist(ivalues)
show()
```

Topology Rendering Script

This script uses initialization file to render a network.

Written by Stephan van Ellewee

```
#!/usr/bin/python

# This Script reads in initialization files and renderes the network connec...    ⇒
...tions
# based on the network radio proximity
# requires  matplotlib !!!

from math import *;
import sys
import re;
from pylab import *

# set up regular expressions...
hostcheck = re.compile('sim.numHosts([\ \t]*)=([\ \t]*)([0-9]*)');
fX = re.compile('sim.host\[([0-9]*)\].mobility.x([\ \t]*)=([\ \t]*)([0-9\-\...⇒
...+\.]*)')
fY = re.compile('sim.host\[([0-9]*)\].mobility.y([\ \t]*)=([\ \t]*)([0-9\-\...⇒
...+\.]*)')
s = re.compile('channelcontrol.([a-zA-Z]*)([\ \t]*)=([\ \t]*)([a-zA-Z0-9\-\...⇒
...+\.]*)')

# check input parameterlength...
if len(sys.argv) <= 1:
    sys.exit()

#check for input filename
print "looking for",sys.argv[1]
l = ''
l = "".join(file(sys.argv[1]).read())

X = dict([])
Y = dict([])

# Parse parameters from input ini
numHosts = -1;
carrierFrequency = 0.0
pMax = 0.0
sat = 0.0
alpha =0.0
names ={}
for i in l.split("\n"):
    if (len(i)>0) and (i[0] == "#"):
        pass
    else:
        xx = fX.findall(i)
        yy = fY.findall(i)
        ss = s.findall(i)
        h = hostcheck.findall(i);
        if len(h) >0:
            if numHosts == -1:
                numHosts = int(h[0][-1])

                #           if len(X) != numHosts:
        if len(xx) > 0:
            print i,xx,int(xx[0][0]), float(xx[0][-1])
            if X.has_key(int(xx[0][0])) == False:
                X[int(xx[0][0])] = float(xx[0][-1])

        #if len(Y) != numHosts:
```

```
        if len(yy) > 0:
            print i,yy,int(yy[0][0]), float(yy[0][-1])
            if Y.has_key(int(yy[0][0])) == False:
                Y[int(yy[0][0])] = float(yy[0][-1])
                names[int(yy[0][0])] = yy[0][0];



        # Determine Parameters for network connection setup
        if len(ss) != 0:
            print i,ss
            if ss[0][0] == "sat":
                sat = float(ss[0][-1])
                print sat
            if ss[0][0] == "pMax":
                pMax = float(ss[0][-1])
                print pMax
            if ss[0][0] == "alpha":
                alpha = float(ss[0][-1])
                print alpha
            if ss[0][0] == "carrierFrequency":
                carrierFrequency = float(ss[0][-1])
                print carrierFrequency



print i,X,Y,'---',len(X),len(Y) #,int(xy[0][0]), float(xy[0][-1])
print X.values()
coords = zip(X.values(), Y.values())

close("all");

# for all coordinates determined
n = 0;
for pos in coords:
    #print pos
    if names.has_key(n):
        # print a node
        plot([pos[0]],[pos[1]],'ro')
        # print a node label
        tt = text(pos[0],
                  pos[1],
                  names[n])
        tt.set_fontsize(20)
    n+=1;



def distance(x,y,x2,y2):
    return ((x2-x)**2 + (y2-y)**2)**0.5

#------------------------------------------------

print "carrierFrequency ",carrierFrequency
print "pMax ",pMax
print "sat ",sat
print "alpha ",alpha

SPEED_OF_LIGHT = 3e8
waveLength = SPEED_OF_LIGHT/carrierFrequency;
minReceivePower = 10.0**(sat/10.0)
interfDistance = (((waveLength**2) *pMax )/(16.0*pi*pi*minReceivePower) )**...      ⇒
...(1.0/alpha);

# TAKEN FROM THE C++ CODE....
# double speedOfLight = 300000000.0;
# double waveLength = speedOfLight/carrierFrequency;
#  return (pSend*waveLength*waveLength / (16*M_PI*M_PI*pow(distance,pathLos...  ⇒
...sAlpha)));
#rxedPower = (transmitterPower*(waveLength**2))/(16*pi*pi*distance**pathLos...      ⇒
...sAlpha);

print 20*"-","\n  ",interfDistance,"m\n",20*"-","\n",interfDistance/1000,"k...       ⇒
...m \n"

#------------------------------------------------
distanceV = []
for i in range(0,len(coords)):
    for ii in range(0,len(coords)):
        dist = distance(coords[i][0],coords[i][1],coords[ii][0],coords[ii][...       ⇒
...1]);
```

```
            print coords[i], coords[ii],dist
            if dist <= interfDistance:
                print " CONNECT !", dist
                distanceV.append(dist)
                plot([coords[i][0],coords[ii][0]],[coords[i][1],coords[ii][1]],...        ⇒
...'b')


if len(distanceV) > 0:
    print "min distance = ",min(distanceV), "max distance = ",max(distanceV...  ⇒
...)
print "MAX X = ",max(X.values()),"MAX Y =", max(Y.values())
print axis([axis()[0],axis()[1]*1.05,axis()[2],axis()[3]*1.05])

show()
```

Appendix B

Initialization Script Example

The Three Prong Network Case is shown by Figure 6.15. The initialization file for this test situation is given here:

omnetpp.analy.846.ini:

Written by Stephan van Ellewee

```
[General]
;ini-warnings = true
network = sim
sim-time-limit = 43200 # 10800 #3600 #86400#21600#10800 #7200 #21600# 6 hours #86400s ;

[Tkenv]
bitmap-path="../bitmaps"
default-run=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
#verbose-simulation = yes
#event-banners = yes
#module-messages = yes

verbose-simulation = no
event-banners = no
```

```
module-messages = no
[DisplayStrings]

###############################################################################
#       Parameters for the entire simulation                                  #
###############################################################################
[Parameters]

# uncomment to enable debug messages for all modules
**.debug = true

**.coreDebug = 0
#sim.host[*].applLayer="TargetApplLayer"
#sim.host[*].applLayer="BurstApplLayer"
sim.host[*].applLayer="DirectedApplLayer"

###############################################################################
#       Parameters for the ChannelControl                                     #
###############################################################################

# carrier frequency in hertz
sim.channelcontrol.carrierFrequency = 868e+6


# signal attenuation threshold [dBm]
sim.channelcontrol.sat   = -110
# path loss coefficient alpha
sim.channelcontrol.alpha = 2.6
sim.host[*].nic.snrEval.pathLossAlpha=2.6

# max transmission power [mW]
sim.channelcontrol.pMax  = 200.0
sim.host[*].nic.snrEval.transmitterPower=200.0

sim.host[*].nic.snrEval.carrierFrequency=868E+6
sim.host[*].nic.snrEval.thermalNoise=-120
sim.host[*].nic.snrEval.sensitivity=-110

###############################################################################
#       Parameters for the Mobility Module                                    #
```

```
################################################################################
# MF debug switch for mobility
sim.host[*].mobility.debug = true

################################################################################
#       Parameters for the Application Layer                                   #
################################################################################
# debug switch
sim.host[*].appl.debug = true

# application message header length
sim.host[*].appl.headerLength=2024

sim.host[0].appl.isSource = true;
sim.host[0].appl.onlyToSat = true;
sim.host[3].appl.isSatNode = true;

sim.host[*].appl.dieAtTime=-1
sim.host[*].appl.activationAt = 0;

sim.host[*].appl.isSatNode = false;
sim.host[*].appl.isSource = false;
sim.host[*].appl.onlyToSat = false;

sim.host[*].appl.mailArrivalInterval = 250.177777778 #900 # 30mins#3600; # 1 hour
sim.host[*].appl.constMailSize = 1

sim.host[*].appl.avrgMailSize = 16


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

# debug switch
sim.host[*].net.debug = true

# application message header length
sim.host[*].net.headerLength=300
sim.host[*].net.defaultTtl=9 # 10
```

```
##############################
sim.host[*].net.expandingRing = false; true
sim.host[*].net.addRoutesToData = true ;false
sim.host[*].net.rreqLength=150
sim.host[*].net.rrepLength=150
sim.host[*].net.rerrLength=150
sim.host[*].net.resendRREQtime=10 ;

sim.host[*].net.rreqAttempts = 10
sim.host[*].net.addressLength=8;
sim.host[*].net.useCheapestRoutesMethod = false;

sim.host[*].net.numberReTX = 0 #20 ; # 10
sim.host[*].net.weightPower = 1;  # how sensitive is the algorithm to different route l

##############################
sim.host[*].net.addressLength=8;
sim.host[*].net.useCheapestRoutesMethod = false;
#sim.host[*].nic.mac.deltaSampleTime = 1000

###############################################################################
#       Parameters for the MAC Layer                                          #
###############################################################################

# debug switch for the MAC layer
sim.host[*].nic.mac.debug = true


# MAC message header length
sim.host[*].nic.mac.headerLength=24
# debug switch
sim.host[*].nic.mac.debug = true


###############################################################################
#       Parameters for the SnrEval                                            #
###############################################################################

# debug switch for the snrEval
sim.host[*].nic.phy.coreDebug = true
sim.host[*].nic.phy.debug = true

# AirFrame header length
sim.host[*].nic.phy.headerLength=16

# debug switch
sim.host[*].nic.mac.debug = true
sim.host[*].nic.snrEval.debug = false
sim.host[*].nic.snrEval.headerLength=0
sim.host[*].nic.mac.maxQueueSize=10000

# transmission power [mW]
#sim.host[*].nic.mac.maxQueueSize=1000
sim.host[*].nic.mac.bitrate=9600 # in bits/second
sim.host[*].nic.mac.rtsCts=true
sim.host[*].nic.mac.broadcastBackoff=31

sim.host[*].nic.decider.debug = true

sim.host[*].nic.decider.snirThreshold=4; in dB
sim.host[*].nic.snrEval.headerLength=192
sim.host[*].nic.snrEval.bitrate=9600 # in bits/second
sim.host[*].nic.decider.bitrate=9600 # in bits/second

sim.numHosts = 14
sim.host[*].nic.mac.maxNodeSeperationDistance = 8000
sim.playgroundSizeX = 26000.0
sim.playgroundSizeY = 28000.0

###############################
sim.host[0].mobility.x = 14007
sim.host[0].mobility.y = 6765

sim.host[1].mobility.x = 15007
sim.host[1].mobility.y = 10765


sim.host[2].mobility.x = 13807
sim.host[2].mobility.y = 16765
##############################
```

```
# LOOP 1
#############################
sim.host[5].mobility.x = 2807
sim.host[5].mobility.y = 8755

sim.host[9].mobility.x = 12507
sim.host[9].mobility.y = 27765

sim.host[8].mobility.x = 7507
sim.host[8].mobility.y = 22765

sim.host[7].mobility.x = 2000
sim.host[7].mobility.y = 19000

sim.host[6].mobility.x = 2000
sim.host[6].mobility.y = 14624

sim.host[4].mobility.x = 7807
sim.host[4].mobility.y = 4760

##################################
# LOOP 2
#############################
sim.host[13].mobility.x = 22107
sim.host[13].mobility.y = 19765

sim.host[12].mobility.x = 25507
sim.host[12].mobility.y = 13765

sim.host[3].mobility.x = 16507
sim.host[3].mobility.y = 22765

sim.host[10].mobility.x =20807
sim.host[10].mobility.y = 4765

sim.host[11].mobility.x = 25807
sim.host[11].mobility.y = 8700
######################################
```