Development of a Monte Carlo Ad Hoc Routing Protocol for Connectivity Improvement

by

Philippus Rudolf Perold

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

Supervisor: Dr. Riaan Wolhuter

Department of Electrical and Electronic Engineering

March 2010

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2010

Abstract

An ad hoc network is a self-configurable, infrastructureless network where nodes relay packets on behalf of other nodes. With the flexibility and dynamic nature of such a network come added complexity. Since nodes are routed in a multihop fashion, the routing strategy plays a significant role in the network performance, specifically that of throughput and latency.

This thesis proposes a novel *Monte Carlo* based ad hoc routing protocol, incorporating reactive routing, multiple paths, the ETX metric and a load balancing strategy. The load balancing strategy utilises *Monte Carlo* methods to dynamically spread traffic to relatively idle parts of the ad hoc network.

The viability of the proposed methodology was evaluated by means of simulation, an analytical model and a hardware implementation. Extensive simulations of the proposed methodology in various scenarios were executed in the simulation environment, *OMNeT++*. The analytical model is based on probabilistic behaviour and queueing theory. Aspects of the proposed methodology were incorporated into *Tmote Sky* wireless modules. Test were performed to evaluate the difference the inclusion of the *Monte Carlo* load balancing strategy has on latency.

From the results of the different analyses it is concluded that the proposed methodology is promising. The routing strategy generated its best results for networks larger than the physical communication range of a single node, where there was an even physical spread of nodes. The given topology maximises the number of possible multiple routes between any two nodes.

Uittreksel

'n Ad hoc netwerk is 'n selforganiseerbare, infrastruktuurlose netwerk waar nodes pakkies namens ander nodes aanstuur. Saam met die aanpasbaarheid en dinamiese aard van so 'n netwerk kom ekstra kompleksiteit. As gevolg van die multihop wyse waarmee pakkies aangestuur word, speel die roeteprotokol 'n beduidende rol, veral as dit by deurset en tydvertraging kom.

'n Nuwe ad hoc roeteprotokol wat reaktiewe padvindtegnieke, meervoudige paaie, die ETX maatstaf en 'n lasverspreidingstrategie gebruik, word voorgestel in hierdie tesis. Die lasverspreidingstrategie gebruik *Monte Carlo* metodes om die las op 'n netwerk op 'n dinamiese wyse na minder besige dele van die netwerk te versprei.

Die doenbaarbaarheid van die voorgestelde metodologie was geëvalueer deur gebruik te maak van simulasie, 'n analitiese model en 'n hardeware implementasie. Verskeie simulasies van verskeie opstellings was uitgevoer in die simulasie omgewing, *OMNeT++*. Die analitiese model is gebaseer op waarskynlikheids- en touteorie. Sekere aspekte van die voorgestelde metodologie is in *Tmote Sky* draadlose modules geïnkorporeer. Daar was toetse gedoen om te sien of the *Monte Carlo* lasverspreidintegniek 'n verskil maak as dit by tydvertraging kom of nie.

Vanaf die resultate van die verskeie analises word dit afgelei dat die voorgestelde metodologie belowend is. Die roete strategie het sy beste resultate gelewer vir netwerke groter as die fisies kommunikasiebereik van 'n enkele node en waar die nodes eweredig versprei was. Hierdie topologie maksimeer die hoeveelheid moontlike roetes tussen enige twee nodes.

Acknowledgements

I would like to express my sincere gratitude to Dr Riaan Wolhuter for all his guidance, patience, and willingness to lend a helping hand wherever possible. I would also like to thank my family and my girlfriend for their continuous support and motivation when I needed it the most.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

A-Team	Asynchronous Team
AODV	Ad Hoc On Demand Distance Vector Routing
CGSR	Clusterhead-Gateway Switch Routing
CPU	Central Processing Unit
DSR	Dynamic Source Routing
EA	Evolutionary Algorithm
ETX	Expected Transmission Count
GA	Genetic Algorithm
IP	Internet Protocol
LAN	Local Area Network
LCC	Least Clusterhead Change
LQI	Link Quality Indicator
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
MPR	Multipoint Relay
NesC	Network embedded system C
NIC	Network Interface Card
OLSR	Optimised Link State Routing
QoS	Quality of Service
RF	Radio Frequency
RREP	Route Reply
RREQ	Route Request

RSM	Response Surface Methodology
SA	Simulated Annealing
SN(I)R	Signal to Noise (plus Interference) Ratio
TCP	Transmission Control Protocol
TKN	Telecommunication Networks Group
TTL	Time To Live
UDP	User Datagram Protocol
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
ZRP	Zone Routing Protocol

Chapter 1

Introduction

Dependency on network connectivity is a growing need in the technology driven world of today. Whether communication or the transport of large quantities of data is the primary goal, speed and accessibility are major factors that need to be taken into consideration. Sluggish data rates are everything but desirable considering the multimedia rich Internet. With regard to accessibility, wireless technologies have taken a leap in the direction of being connected wherever you are. The dependency on a fixed infrastructure limits the scalability and availability of networks. Even with wireless networks nodes are bound by the range of a given transmitter. Extending the physical range of the Internet, for instance, can thus become a tedious and expensive task.

1.1 Motivation

An alternative to limiting network infrastructures, is the utilisation of an ad hoc network. Since information traverses ad hoc networks in a multihop fashion, the range of a given network is theoretically bounded by the longest chain of communicating nodes. With the flexibility, scalability and dynamic nature of an ad hoc network come various new challenges that must be overcome. As the number of nodes in an ad hoc network grows, the number of potential multihop paths increases as well. Determining the optimal route to transport data along is a problem with multiple, everchanging input variables. The nature of the needs of the network user also influences what is considered to be optimal. A user could require a high throughput, where the speedy transfer of blocks of data is required. Another example of user need is low latency, for example where multimedia is streamed over the network.

The intuitive, and most utilised strategy is to merely select the route between a source and destination node with the least number of hops. This strategy would probably be the most effective in a large percentage of data transfers in ad hoc networks. However, there are situations where the strategy's performance could suffer. A factor with substantial influence in the dynamics of an ad hoc network is the physical layout, or topology, of the nodes. An example of where topology could influence the performance of an ad hoc network as a whole, is where a certain link is common to a large number of routes. When traffic increases in the network, the link would act as a bottle neck, and an alternative route, with possibly more hops could be a better option.

The *status quo* of the network at a given time, which would comprise of physical layout and traffic distribution, thus both play a role in the potential success of a routing strategy. It was decided to investigate an alternative and novel approach to data routing in ad hoc networks, where these factors are taken into consideration.

1.2 Objectives

This investigation set out to explore the merits of a novel ad hoc routing protocol. The proposed protocol would measure route quality by determining the expected number of times a packet needs to be transmitted between nodes before it reaches its destination. By using this metric, the effects of physical network topology and data traffic is taken into account. When data transmission is requested, a discovery process is triggered, which could potentially return multiple routes to a given destination. The metrics of the potential multiple routes are utilised as *Monte Carlo* type weighting functions to determine which route is to be chosen for data transmission. The idea behind the weighting of routes is to spread the load on a network in order to utilise the dynamic infrastructure to its full potential.

In order to investigate the validity if this *Monte Carlo* approach, analysis of the proposed strategy should be done by means of extensive simulation of relevant scenarios. A further objective was to develop an analytical model emulating the dynamics of the strategy. Such a model can subsequently be utilised as a planning tool to provide a reasonably accurate performance indication for such a network, without having to resort to time consuming simulations. Although not a prime objective from the outset, it was however, foreseen to implement a first iteration of a suitable high level hardware topology as a reasonable emulation of the proposed strategy.

1.3 Contributions

In reaching the abovementioned objectives, the following contributions were made:

- Development of a novel ad hoc routing protocol, loosely based on Dynamic Source Routing (DSR)
- Incorporation of the Expected Transmission Count (ETX) metric into the developed protocol
- Incorporation of the *Monte Carlo* load balancing strategy into the developed protocol
- Analysis of the proposed methodology by means of simulation
- Optimisation of critical identified protocol parameters by means of a Genetic Algorithm (GA)
- Verification of the viability of the proposed protocol simulation results by means of an analytical model

- Partial hardware implementation of the proposed routing protocol, inclusive of
  - Synchronisation of individual wireless modules in order to accurately observe packet latency
  - Electromagnetic shielding of wireless modules in order to establish multiple independent communication paths

1.4 Thesis Outline

Chapter 1 The motivation, objectives and outline of the thesis are stated.

Chapter 2 A high level overview of ad hoc networks and classification of existing ad hoc routing protocols are given. Examples of relevant existing strategies and the fundamentals of their operation are highlighted.

Chapter 3 The proposed methodology is stated and explained.

Chapter 4 Analysis of the proposed methodology in the form of simulation is discussed. Simulation environments are also investigated. The development of the proposed routing protocol in the simulation environment of choice, *OMNeT++*, is documented.

Chapter 5 Analysis of the proposed methodology in the form of an analytical model is discussed. Statistics and queueing theory utilised in the model are also covered.

Chapter 6 Analysis of the proposed methodology in the form of a hardware implementation is discussed. Relevant information regarding the hardware and embedded software is given. Alterations to the existing routing strategy is explained.

Chapter 7 The testing process of ad hoc routing protocols is investigated. A parameter optimisation strategy, with its results are also documented. Results obtained from the forms of analysis as stated in Chapters 4-6 are shown.

Chapter 8 This chapter documents the conclusions subsequent to the abovementioned investigation. Possible further research with regard to the work done is also stated.

Chapter 2

Literature Study

In the design and implementation of a routing protocol for an ad hoc network, having extensive knowledge and understanding in the existing technologies in the area, enlightens one as to where improvement is needed and where design choices should not be tampered with. This chapter runs through the basic definition of an ad hoc network, the technology it is built upon, and touches various directions that can be taken when designing an ad hoc routing protocol. A closer look is also taken at the basic strategies behind some of the more prominent existing ad hoc routing protocols.

2.1 Wireless Local Area Networks (WLANs)

The dependency on fixed infrastructure in traditional wired Local Area Networks (LANs) puts a spoke in the wheel of development towards scalable and mobile networks. The advantages of a WLAN can be categorised into the following five broad categories [4]:

- *Installation speed and simplicity.* No cables have to be physically installed.
- *Installation flexibility.* Wireless networks can reach places where wiring proves to be problematic.
- *Scalability.* A variety of topologies can be supported. Configurations can easily be changed for small to large networks, consisting of many users.
- *Improved productivity and service.* Shared resources can be accessed anywhere in an organisation.
- *Reduced cost-of-ownership.* Overall life-cycle costs can be significantly lower than that of its wired counterpart.

The scalability of a WLAN, however, is still limited by the physical range of communication as governed by the wireless technology in place. A central access point also still acts as the backbone of the network as a whole. It could thus be said that the network is still bound by the infrastructure in place.

A more scalable and dynamic alternative to the conventional WLAN, would be a self-configurable, infrastructureless network where nodes relay packets on behalf of other

nodes. Such a network is formally referred to as a Mobile Ad Hoc Network (MANET). From here on these networks would merely be referred to as ad hoc networks, since the nodes are not necessarily implied to be mobile.

2.2 Ad Hoc Networks

An ad hoc network is a (possibly mobile) collection of communications devices (nodes) that wish to communicate, but have no fixed infrastructure available, and have no predetermined organisation of available links [5]. Each node has the responsibility to have an awareness of other nodes that fall in communication range with it. It is not necessary for every node to be in direct communication range with every other node, since network packets are relayed in a multi-hop fashion across the network. Nodes can also enter and leave the network, which is a step in the right direction when it comes to significant improvements in scalability. An ad hoc network can be built around any wireless technology, such as infrared or Radio Frequency (RF).

The difference between an ad hoc and infrastructure-based WLAN configuration is illustrated in Figure 2.1 and Figure 2.2 [4].

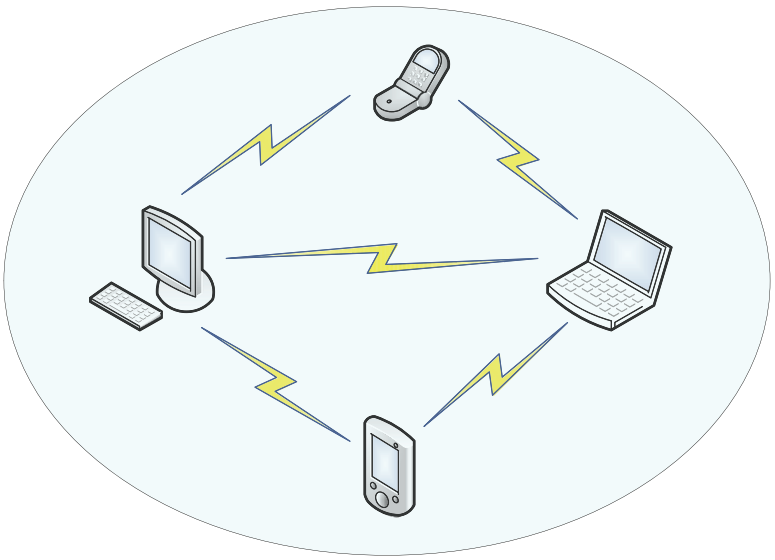


Figure 2.1: Ad hoc wireless network configuration

This alternative approach to a wireless network is accompanied by a new set of problems. The multihop routing across such a network introduces a unique and complex problem. The possibly mobile nature of an ad hoc network complicates matters even further.

2.3 Categorisation of Ad Hoc Routing Protocols

The routing level problem in ad hoc networks has seen many different approaches in the quest for factors such as high efficiency and scalability. In the development of these protocols, different categories of approaches have appeared. An overview of current applicable ad hoc routing protocols, and how they are classified, are illustrated in Figure 2.3 [6].

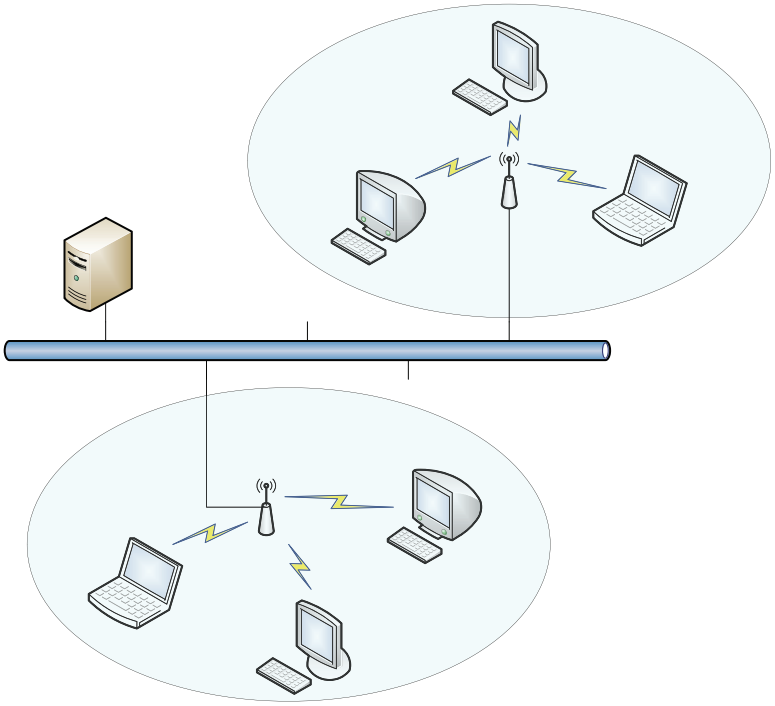


Figure 2.2: Infrastructure-based wireless network configuration

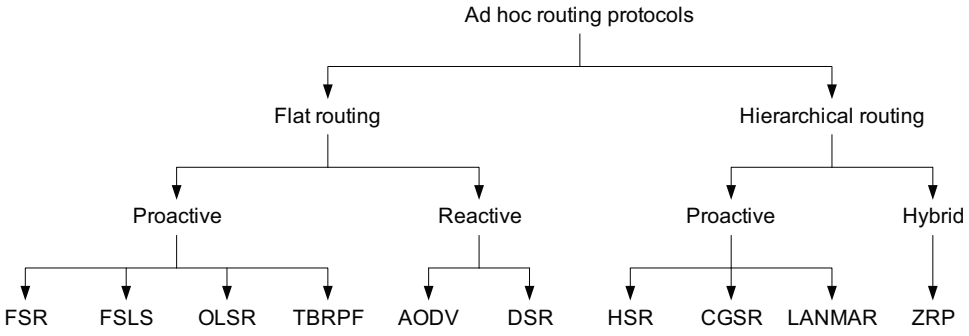


Figure 2.3: Classification of ad hoc routing protocols

2.3.1 Flat Routing

In flat routing, each node plays a role of equal importance. There is no hierarchy in the addressing scheme. Flat routing can be broken down further into two categories: proactive and reactive routing.

2.3.1.1 Proactive (Table-Driven) Routing

Proactive routing protocols ensure the exchange of background routing information, regardless of whether communication is requested between nodes. These route updates are scheduled at certain intervals. The idea is thus to maintain relatively *fresh* routing information. The down side to this approach is the extra traffic generated by the constant flooding of control packets across the network.

The most popular proactive routing protocol, OLSR, is described.

Optimised Link State Routing (OLSR)

In order to reduce the traffic generated by the periodically sent control packets, Multipoint Relays (MPRs) are used by the protocol. An MPR is, for example, the minimum set of one-hop neighbours required by a node to reach all of its two-hop neighbours. Updates are then only sent through the given node's MPRs. OLSR works efficiently in dense networks. In a sparse network configuration, more neighbours would become MPRs, and the efficiency would thus drop.

2.3.1.2 Reactive (On-Demand) Routing

Reactive routing attempts to reduce control traffic by only exchanging routing information when communication is awaiting. When a node wants to exchange data with another node, *route discovery* takes place. The discovery of a route thus only happens on demand.

Two of the more prominent reactive routing protocols will briefly be described.

Ad Hoc On Demand Distance Vector Routing (AODV)

When communication awaits, *route discovery* is initiated, and a Route Request (RREQ) packet is flooded to all of the source node's neighbours [7] [8]. This flooding continues until the destination is reached, or an intermediate node with a *fresh enough* route to the destination is found. Destination numbers, sequence numbers and broadcast identification numbers all help to ensure loop free routing and the maintenance of recent routing information. Intermediate nodes record the address of the neighbour that had sent the RREQ packet in order to establish a reverse path for the Route Reply (RREP) packet to propagate towards the source once the destination has been found. The process is illustrated in Figure 2.4 [7].

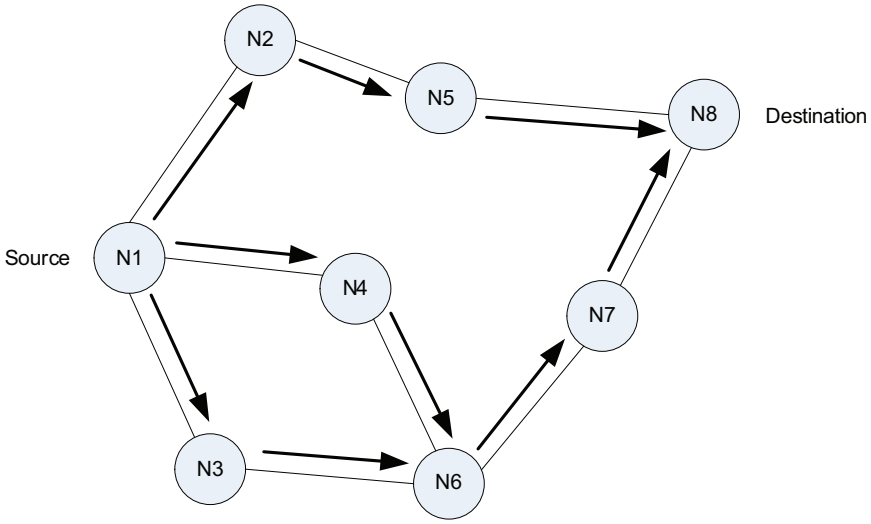
Route maintenance is done by the sending of *link failure notification* packets in the event of broken links. *Route discovery* can then be repeated by the source if communication is still desired. Periodic *hello* messages are sent to neighbouring nodes in order to ensure an awareness of the status of links in their environment.

DSR

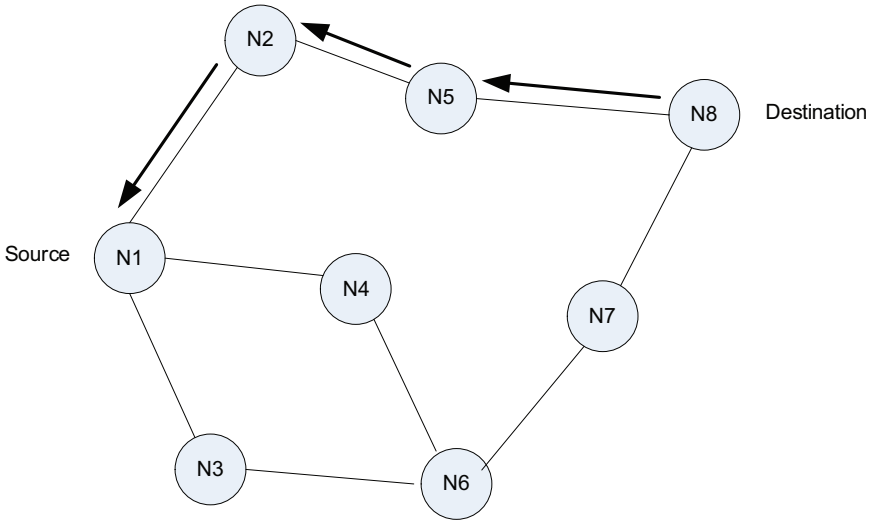
DSR works in very much the same manner as AODV. The difference lies in the fact that DSR packets contain full routing information, where AODV packets only carry the destination address. A route request packet keeps track of the path it follows until it reaches the destination. The packet, with this information, is then sent back along the same route [7] [9]. Figure 2.5 illustrates the dynamics of DSR [7].

2.3.1.3 Protocol Comparison

Table 2.1 [6] shows a comparison between the different flat routing protocols mentioned. N denotes the number of nodes in the network, and e the number of communication pairs.

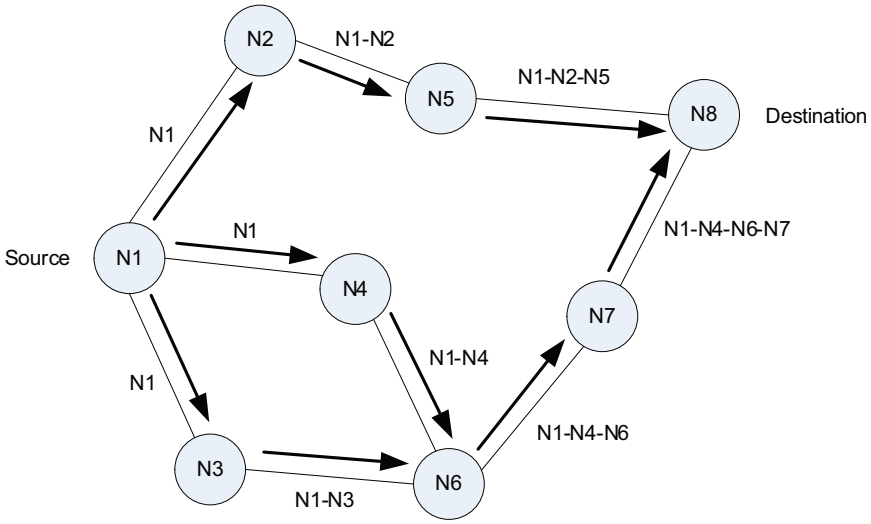


(a) Propagation of the RREQ

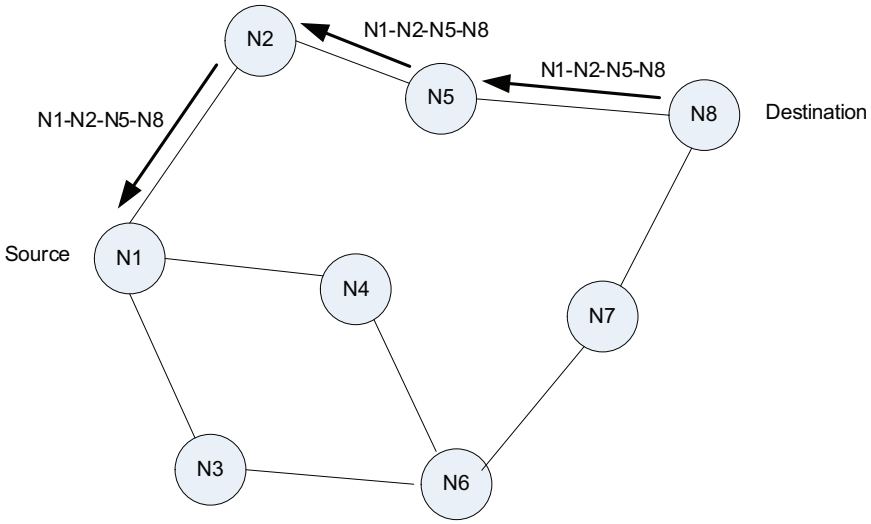


(b) Path of the RREP to the source

Figure 2.4: AODV *route discovery*



(a) Building of the route record during route discovery



(b) Propagation of the route reply with the route record

Figure 2.5: Creation of the route record in DSR

Table 2.1: Characteristics of flat routing protocols

	OLSR	**AODV**	**DSR**
Routing philosophy	Proactive	On-demand	On-demand
Routing metric	Shortest path	Shortest path	Shortest path
Frequency of updates	Periodically	As needed (data traffic)	As needed (data traffic)
Use of sequence numbers	Yes	Yes	No
Loop-free	Yes	Yes	Yes
Worst case exists	Yes	Yes (full flooding)	Yes (full flooding)
Multiple paths	No	No	Yes
Storage complexity	*O(N)*	*O(e)*	*O(e)*
Comm. complexity	*O(N)*	*O(2N)*	*O(2N)*

2.3.2 Hierarchical Routing

Growth in size of an ad hoc network increases link and processing overhead [6]. This limits the scalability of a given network utilising a flat routing scheme. Nodes are grouped in so-called *clusters*, and then different nodes are assigned different responsibilities inside and outside the *clusters*. In effect, a hierarchical configuration is formed in much the same way as the hierarchical Internet.

Clusters are mostly formed by nodes in close geographical proximity to each other. Each cluster assigns a leading node (*clusterhead*) that is in charge of organising communication with other clusters.

Clusterhead-Gateway Switch Routing (CGSR)

CGSR [10] [6] uses the Least Clusterhead Change (LCC) algorithm to partition the network into clusters. Clusterheads are then elected within each cluster. Nodes common to two or more clusters are called *gateway* nodes. When communication commences, packets hop from clusterhead to gateway to clusterhead until they reach the destination cluster. The clusterhead in the destination cluster then forwards the packet to the destination (if the destination is not the gateway or the clusterhead that it has already traversed). The strategy is illustrated in Figure 2.6 [6].

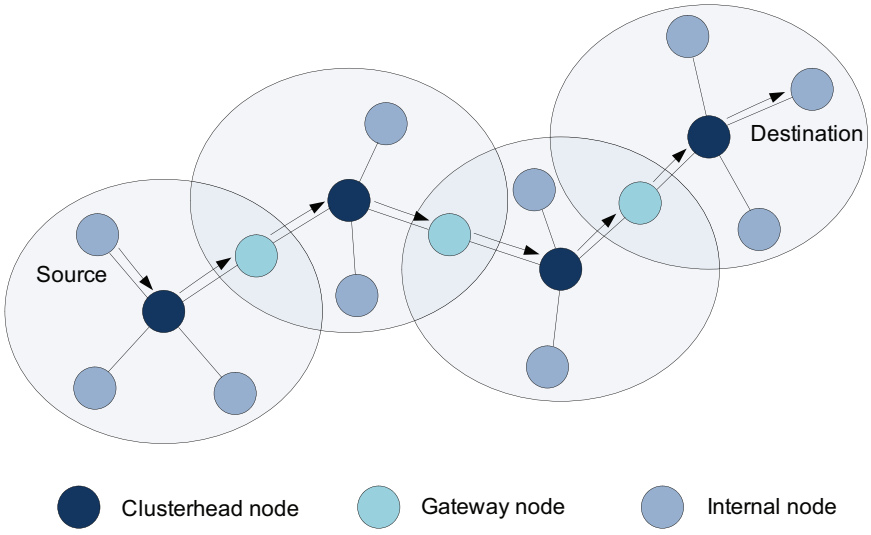


Figure 2.6: CGSR routing: showing a data path from source to destination

Zone Routing Protocol (ZRP)

ZRP [11] [6] is two level hierarchical in nature, but is also classified as a hybrid routing protocol. It is a hybrid between proactive and reactive routing. Each node falls in a predefined *zone* centred around itself. Within this zone proactive routing is used. When communication reaches outside the source node's zone, route discovery by means of RREQ and RREP packets commences.

2.3.2.1 Protocol Comparison

Table 2.2 [6] shows a comparison between the two hierarchical routing protocols mentioned. N denotes the number of nodes in the network, M the average number of nodes in a cluster, L the average number of nodes in a node's local scope, and e the number of communication pairs.

Table 2.2: Characteristics of hierarchical routing protocols

	CGSR	**ZRP**
Hierarchy	Explicit two levels	Implicit two levels
Routing philosophy	Proactive	Hybrid
Loop-free	Yes	Yes
Routing metric	Via critical nodes	Local shortest path
Critical nodes	Yes (clusterhead)	No
Storage complexity	$O(N/M)$	$O(L)+O(e)$
Comm. complexity	$O(N)$	$O(N)$

2.4 Routing Metric

All ad hoc routing protocols describe a decision making process where different routes to a destination are evaluated and one is selected according to some predetermined metric. Initially, the least amount of hops to a destination was the metric of choice in ad hoc routing protocol design. Factors such as bad link quality and traffic congestion of links can, however, cause routes with more hops to produce higher throughput.

ETX

The ETX metric [12] is an alternative metric that finds high throughput paths on multi-hop wireless networks. The ETX of a link is the expected number of transmissions required to send a packet over a link. This expected number includes retransmissions. A link with a lower ETX is thus better. The ETX of a link is calculated by using

$$ETX = \frac{1}{d_f \times d_r}, \tag{2.4.1}$$

where d_f, the forward delivery ratio, is the probability that the data packet reaches its destination, and d_r, the reverse delivery ratio, is the probability that an acknowledgement packet is received by the source. By determining both of these ratios, asymmetry in routes is appropriately handled. In order to maintain relevant values for these ratios, dedicated probe packets, of a fixed size, are broadcast at a certain predetermined period, τ. The period, τ, is varied, or *jittered*, by up to $\pm 0.1\tau$. The *jitter* is merely a measure taken to reduce the probability of synchronization, which could lead to packet collisions. Each node remembers how many probe packets it has received from a certain sender in the last w seconds. Defining $count(t-w, t)$ as the number of probes received from a certain neighbour node during the last w seconds, and w/τ as the number of probes that should

have been received had there been no losses, [12] calculates the delivery ratio from a sender at any time, t, as

$$r(t) = \frac{count(t-w,t)}{w/\tau}. \tag{2.4.2}$$

The ETX of a given route, would thus be the summation of the ETXs of all intermediate links falling on this path.

2.5 Ad Hoc Routing Problems

In the design of any routing protocol, there are several hurdles that need to be overcome. This section briefly runs through some of the more prominent problems that need to be taken into consideration in the ad hoc routing protocol development process.

2.5.1 Route Loops

Loops being formed in the routes of a network are probably the biggest problem that needs to be side-stepped. There are different situations that could trigger the forming of a route loop. One strategy to decrease the number of route loops forming, is called *split horizon* [13] [14]. In the implementation of *split horizon*, routing information is never sent back to the source of the given information.

Another type of loop that could occur, is in a circular network configuration, where one node suddenly disconnects. As routing information is sent around the circle, the hop counts increments with every hop. Each node thinks that the route with the ever increasing hop count is still the only available route. This route update will loop indefinitely with the hop count approaching infinity [13]. This problem is more formally referred to as the *count to infinity* problem. By defining a maximum hop count for any route, this problem is solved. Any route with this maximum hop count is classified as unreachable [14].

Another strategy to minimise the formation of route loops is the utilisation of so called *holddown timers*. If a route is declared unreachable or if the metric increases beyond a certain threshold, a router will not accept any other information about that route until the *holddown timer* expires. This approach prevents the router from accepting possibly bad routing information while the network is busy updating its routing information around the recent change in topology [14].

2.5.2 Slow Convergence

Slow reaction to topology changes in an ad hoc network results in sub-optimal routes being utilised, and it also increases the probability of problems such as the *count to infinity* problem. A possible improvement in convergence speed can be found by implementing triggered updates. The moment a change in the network is detected, the routing information is passed along to all applicable nodes [13].

2.5.3 Common Link Congestion

In certain situations particular links, common to more than one route, become congested due to various sources propagating their data over it. The problem is caused by the use of a primitive routing metric that does not take network traffic in consideration. As is the case with most routing protocols, the lowest number of hops is utilised as the metric, which leaves a vulnerability for the problem.

2.6 Summary

This chapter covered the basic theory behind ad hoc networks, and also highlighted various prominent ad hoc routing protocols being utilised today. Various design considerations and possible hurdles in the development of such a routing protocol were also discussed.

Chapter 3

Proposed Methodology

The proposed solution set out in this chapter focuses on load balancing. An ad hoc network possibly comprises of many independent links and paths. The situation where one path takes a large load, while another possible route remains idle, or less active, could be managed in a more efficient manner in order to better utilise the communication potential of the network, as a whole. The approach hinges on spreading the load across a network, by incorporating weighted decision making. A form of random, or *Monte Carlo*, sampling is thus used. This chapter explores different possibilities introduced by *Monte Carlo* methods with respect to ad hoc routing strategies.

3.1 Monte Carlo Methods

3.1.1 Background

Monte Carlo methods are a class of computational algorithms that are based on the principle of repeated random sampling [15]. These methods are often utilised when an exact result cannot be calculated deterministically [16]. They have been found to work well in systems that have many interacting degrees of freedom. Systems that are characterised by uncertainty in their inputs are thus ideal candidates for analysis or simulation by means of *Monte Carlo* methods.

These methods have been used in the simulation of various physical and mathematical processes and systems.

3.1.2 Application Areas

The following applications are examples of where *Monte Carlo* methods have been implemented:

- Modelling of the behaviour of semiconductor current carriers [17]
- Modelling of light transport in biological tissue [18]
- Simulation in finance [19]

- Simulation in statistical physics [20]
- Calculation of π [21]

The idea to incorporate *Monte Carlo* methods in the design of a routing protocol, comes from the implementation of these methods in the modelling of the behaviour of semi-conductor current carriers. Since semi-conductor current carriers move in the path of least resistance, and *Monte Carlo* methods are used to predict this movement very accurately, it was decided to investigate these methods utilised in deciding which route to transmit data along in an ad hoc network. In a semiconductor, certain factors influence the movement of the current carriers. These factors thus add weight to the probability of the current carriers behaving in a certain manner. The equivalent factors in an ad hoc network thus need to be identified in order for a similar probabilistic approach to controlling the *behaviour* (routing decisions) of network traffic to be implemented.

3.2 Routing Strategy

The basic strategy incorporating *Monte Carlo* methods leans heavily on multi-path routing in ad hoc networks. The idea is to do *route discovery* on an on-demand (Section 2.3.1.2) basis, and then keeping track of multiple routes to the destination, if they exist. The implementation is thus roughly based on the DSR routing protocol, as described in Section 2.3.1.2. The choice of reactive, instead of proactive routing, is an effort to minimise excessive control overhead.

The discovered routes would then be ordered according to a metric. The metric is based upon link quality and traffic. The ETX metric (section 2.4) fulfills these needs. Since the ETX metric is dynamic, it provides relevant information regarding the current status of routes, which is exactly what is needed for the proposed solution.

The probability of a specific route being chosen, is then weighted by the metric of the given route. The probability of the *best* route being taken is thus the highest. In the scenario where a considerable amount of data is routed along a specific route, the metric of the route would change with the traffic, and thus lower the probability of the route being chosen for further data transport. By not merely taking the route with the best metric, traffic is spread out over the network. The idea is to utilise the network more efficiently as a whole.

3.3 Strategy Investigation

In order to explore the possible viability of the incorporation of *Monte Carlo* methods, a simple *Java* application was developed that simulates the dynamics of the network configuration as illustrated in Figure 3.1. The source code can be found in Appendix A

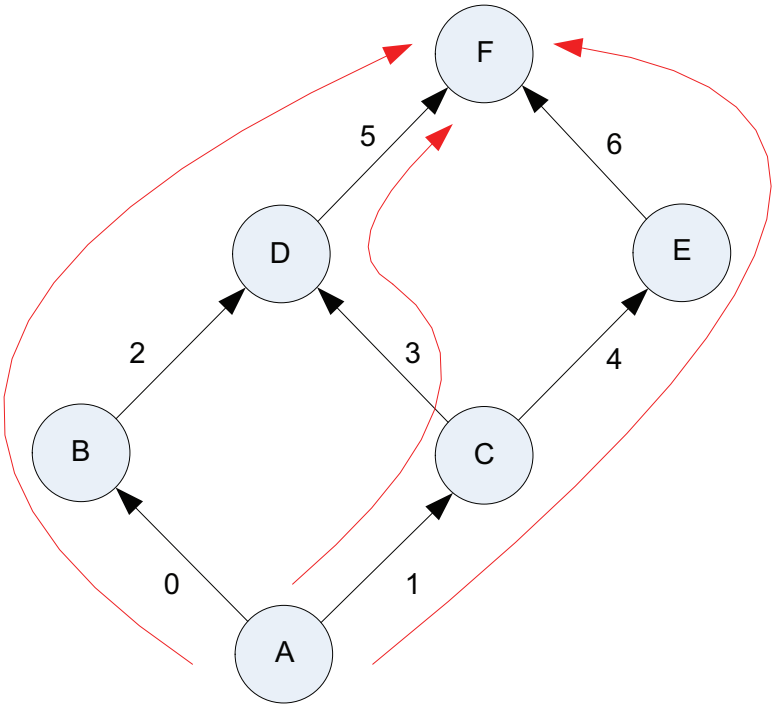


Figure 3.1: Simple network configuration with three possible routes between nodes A and F

3.3.1 Algorithm

It can be seen that three routes exist from source node A to destination node F. For the purpose of this simulation, the routes will be defined by the links that form them.

Route 1: $0 \rightarrow 2 \rightarrow 5$

Route 2: $1 \rightarrow 3 \rightarrow 5$

Route 3: $1 \rightarrow 4 \rightarrow 6$

Care was taken in ensuring certain links are shared among the various routes in the simple network configuration. The traffic flowing along one route, would thus have an effect on the metric of another route.

In this simulation it is assumed that none of the nodes are mobile. The effect of link quality is incorporated by randomly assigning a metric to each link. As traffic increases and decreases across a given link, the metric changes accordingly. The *best* route thus has the lowest combination of link metrics.

In order to observe the improvement in network utilisation and also let the simulation track reality a bit closer, bandwidth constraints on links are also incorporated. When traffic reaches a certain threshold in a certain timeslot, extra traffic is added to the route in order to simulate rising levels of contention.

3.3.2 Results

To have a benchmark to compare results to, simulations were also done where the *best* route was utilised with a probability of one. Results of where the best metric is always used, will thus be compared to where *Monte Carlo* methods are used to weigh the probability of route utilisation.

Figure 3.2 shows how traffic dispersed among the three possible routes when the best metric method was used.

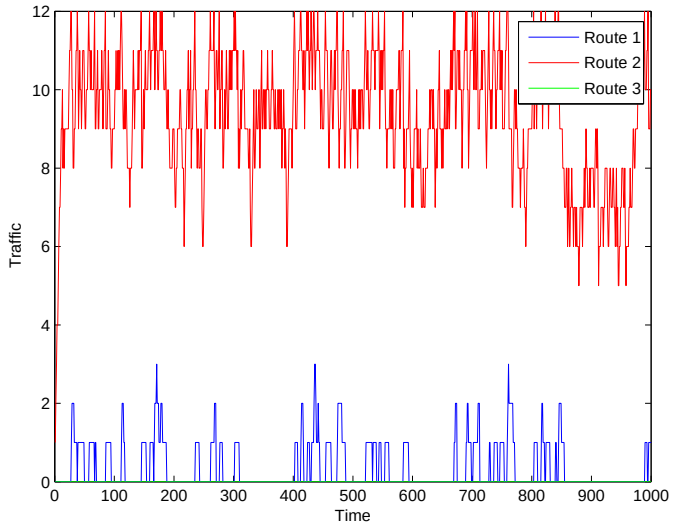


Figure 3.2: Network traffic when best metric determines route choice

The *Monte Carlo* approach delivered the results as seen in Figure 3.3.

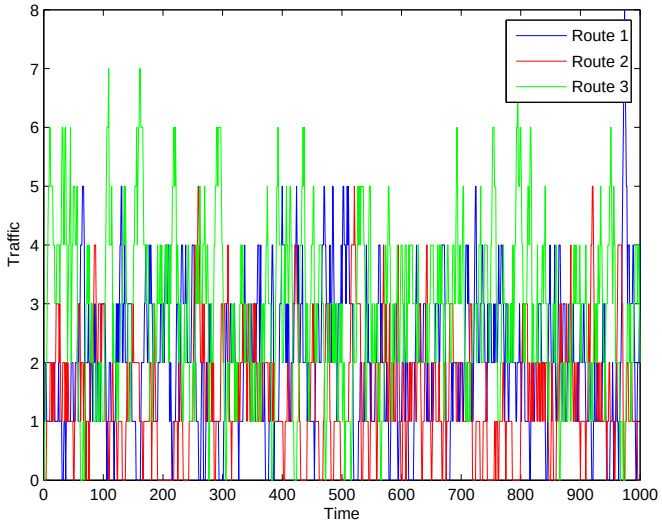


Figure 3.3: Network traffic when metric determines probability of route being chosen

The average network traffic along all three routes, as simulated with the two different strategies, is plotted in Figure 3.4.

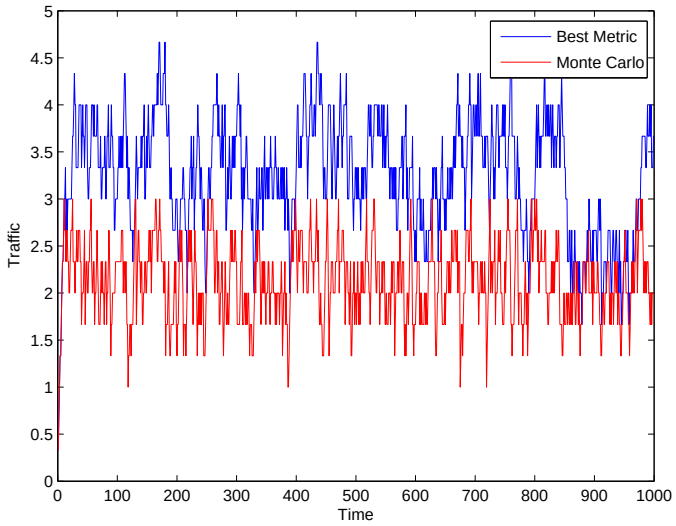


Figure 3.4: Average network traffic of best metric and *Monte Carlo* strategies

It is clear from Figures 3.2 and 3.3 that the *Monte Carlo* approach spreads the load across the available links, while the other method merely loads the route with the best metric. The increased traffic generated by contention on the utilised route increases the average network traffic, as seen in Figure 3.4. The *Monte Carlo* approach balances the traffic load among routes, and thus decreases contention, which leads to less traffic.

It is thus concluded that the *Monte Carlo* approach to load balancing holds promise when it comes to routing strategies in ad hoc networks. Even though various assumptions have been made in this relatively high level simulation, it can be deduced that the basic idea of the strategy shows improvement when merely observing average traffic along a given route. The repercussions of the strategy as a whole are to be discovered in further lower level simulation.

3.4 Summary

This chapter introduced *Monte Carlo* methods, and incorporated it into an ad hoc routing protocol. A high level simulation of a small network was run to gain insight into the potential the described protocol has. The proposed methodology generates less traffic, which would result in better performance. This is now further explored in subsequent chapters.

Chapter 4

Simulation

One way of extensively investigating the performance and dynamics of a developed routing protocol, is to run simulations that mimic a real world network utilising the protocol in question. It is thus desirable to incorporate as many influential factors into the simulation as possible, in an attempt to optimise the mimicry.

Simulation cannot be trusted to be absolutely accurate, but it does provide a rough idea of the dynamics and performance of a proposed protocol.

This chapter starts by exploring different simulation environments, whereafter the steps in the development of the proposed protocol (Chapter 3) are documented.

4.1 Simulation Environment

Developing a simulator from scratch would be a tedious and intricate task if a realistic representation of a real world network is to be simulated. For that reason it was decided to investigate possible simulation environments that could form an extensible base.

Various simulation environments that could fullfill the above mentioned need were found:

- *OMNeT++* [22]
- *NS2 (Network Simulator 2)* [23]
- *OPNET* [24]
- *GloMoSim* [25]
- *NCTuns* [26]
- *JiST/SWANS* [27]

Investigation into the above mentioned simulation environments showed that *OMNeT++* and *NS2* are the two most widely used, and extensive documentation exists for both. The large library of already built network components, and the detailed documentation accompanying *OMNeT++* proved to be the deciding factor in choosing it as the platform to develop the proposed ad hoc routing protocol.

4.1.1 Background

4.1.1.1 OMNeT++

OMNeT++ is an open-architecture discrete event simulation environment [22]. It was primarily designed with communication network simulations in mind. The generic and flexible nature of it enables it to be used for many other purposes as well, such as complex IT systems and queueing network simulations. *OMNeT++* has been developed by András Varga, at the Technical University of Budapest, Department of Telecommunications.

Verification of developed protocols is aided by a strong graphical representation, which includes animation of transmissions. *OMNet++* has built in capabilities dedicated to statistics collection as well.

OMNeT++ is written and extendable in *C++*. Network topologies and connections are defined in the so-called *ned* language, while functionality of the modules is written in *C++*.

Due to the generic nature of *OMNeT++*, various frameworks, or extensions, have been designed to specialise *OMNeT++* in certain areas. The most popular of these extensions are the *INET Framework* [28] and the *Mobility Framework* [29]. The *INET Framework* focusses on high level network protocols, for example Internet Protocol (IP), User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The *Mobility Framework*, on the other hand, accommodates the design of lower level wireless and mobile simulations. For the purpose of the simulation of a newly designed ad hoc routing protocol, it was decided that the *Mobility Framework* provides the tools and framework to be extended.

4.1.1.2 The Mobility Framework

The *Mobility Framework* provides an extensive library of basic modules which can be used in building specific applications and protocol implementations. The *Mobility Framework* has been developed by the Telecommunication Networks Group (TKN) at the Technical University of Berlin. It was developed with extensibility in mind, which makes it ideal for rapidly developing relatively complex network simulations. The core framework, which implements support for node mobility, dynamic connection management and a wireless channel model, provides the perfect base to build upon when a wireless ad hoc network needs to be simulated.

4.2 Implementation

The development of a network communication protocol is a multi layer problem. It ranges from the physical layer all the way up to applications at the top, running oblivious of what is going on at the lower layers. For the development of an ad hoc routing protocol, and the testing of it, only the network and application layers have to be created.

4.2.1 Design Overview

Figure 4.1 is a graphical representation of an ad hoc host in *OMNeT++*, showing the modules and connections between them. The figure is a direct visualisation of the *OMNeT++ ned* file that defines a node in the developed network.

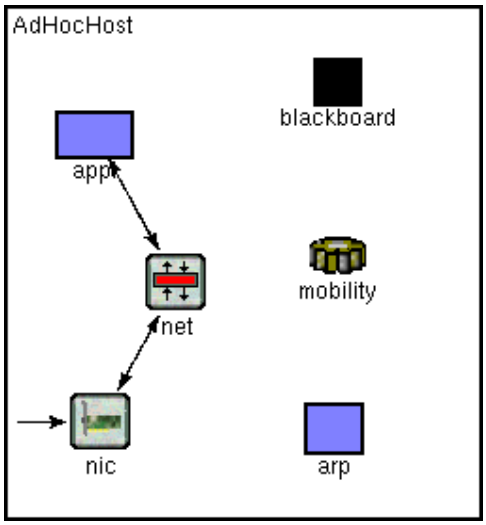


Figure 4.1: Ad hoc host layers and connections in *OMNeT++*

At the top, the application layer, *appl*, is found, which is connected to the network layer, *net*, and then the Network Interface Card (NIC), *nic*. The *blackboard* module is used for communication within a simulation, while the *mobility* module provides functionality for nodes to be mobile. Lastly, the *arp* module is only used for address resolution between certain layers.

4.2.2 Utilised System Components

Figure 4.1 illustrates the modules and connections that form a node that is to be simulated. The power of the *Mobility Framework* lies in the library of modules that can be integrated into the developer's own simulation.

4.2.2.1 The Network Interface Card

The *nic* module takes care of the physical, as well as the Medium Access Control (MAC) layer. The modules that form the *nic* module, as well as the connections between them, are graphically represented in Figure 4.2. The functionality of the physical layer (transmitting, receiving, modulation) is controlled by *snrEval* and the *decider* [29]. The *snrEval* module calculates Signal to Noise (plus Interference) Ratio (SN(I)R) information for a received message. The *decider* then uses this information to decide whether the message has been lost, has received bit errors, or has reached the destination successfully.

If the *decider* determines that a message has successfully arrived at its destination, the message is forwarded to the MAC layer. The *mac* module that is used is an implementation of the *IEEE 802.11b* standard.

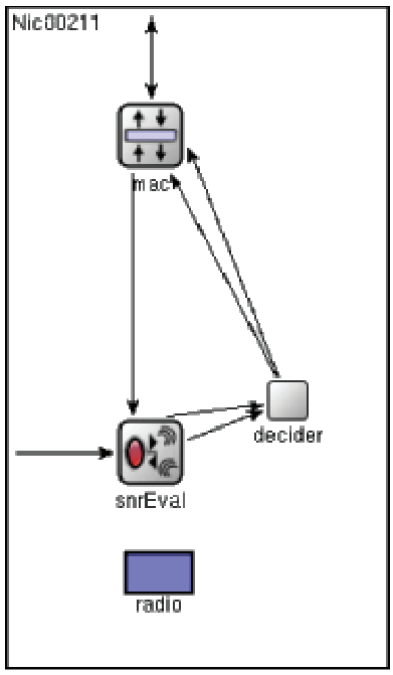


Figure 4.2: NIC module and connections

4.2.3 The Network Layer

The proposed routing protocol (Section 3) would reside within the network layer. The bulk of development covered in this report, thus resides within this module.

4.2.3.1 The ETX Metric

The ETX metric (Section 2.4) is controlled from the network layer. Probe (*HELLO*) packets are defined. A *HELLO* packet has four fields:

- Destination address
- Source address
- Time To Live (TTL)
- Sequence number

While the destination and source addresses establish forward and reverse paths for the *HELLO* packets and acknowledges (*HELLO_ACK* packets) to successful transmissions, the TTL and sequence numbers are used in the prevention of routing problems (Section 2.5.1). Since each node only inspects immediate neighbours, the TTL would always be equal to one for *HELLO* packets.

As explained in Section 2.4, a running window is examined in order to determine the ETX of all links between a given node and its neighbours.

The network layer is the highest layer that ETX control packets ever reach. The application running on top of the system is oblivious of the routing activity taking place.

4.2.3.2 Routing

The process of obtaining relevant information on the quality of links to neighbouring nodes, described in Section 4.2.3.1, runs independently on the network layer. For the purpose of routing, ETX information is thus always available.

At any given time, each node possesses a list of neighbouring nodes that fell in communication range within a predefined time frame. In the event of route discovery, RREQ packets are flooded across the network until the intended destination is reached. RREP packets are sent along the reverse paths of the discovered routes. On the forward and reverse paths of the discovery process, the routing caches on intermediate nodes are updated with *fresh* ETX values for routes to destinations that fall on the path that has been travelled on. Figure 4.3 illustrates a typical route discovery cycle, while Table 4.1 stipulates the route information that is gained as the RREQ packet travels towards the destination, and as RREP packet returns towards the source.

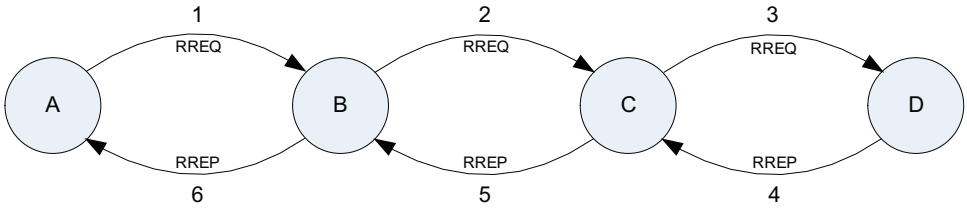


Figure 4.3: Intermediate node route update process

Table 4.1: Intermediate route updates in discovery process

Last link traversed	Routes updated
1	A → B,
2	A → C, B → C
3	A → D, B → D, C → D
4	D → C,
5	D → B, C → B
6	D → A, C → A, B → A

With any route update, the specific route is time stamped in the route cache. In the event of a data transmission request, the source node's route cache is queried for a route to the desired destination that has been discovered within a predefined time frame. The logic behind this strategy is to reduce unnecessary control overhead. If more than one packets are to be sent to a certain destination consecutively, it is most probably unnecessary to search for multiple routes to the destination before every packet is sent. However, as more time goes by between route updates, the risk of route information becoming stale increases. Thus, if data transmission is scheduled, and route information is older than a certain threshold, route discovery is triggered. The optimal time a route remains *fresh* enough, however, is not a value that can be deduced intuitively.

A time interval is defined in which the source node waits while the discovery process commences. After the interval, the Monte Carlo approach (Chapter 3) is used in selecting a route. The flow of the strategy is depicted in Figure 4.4.

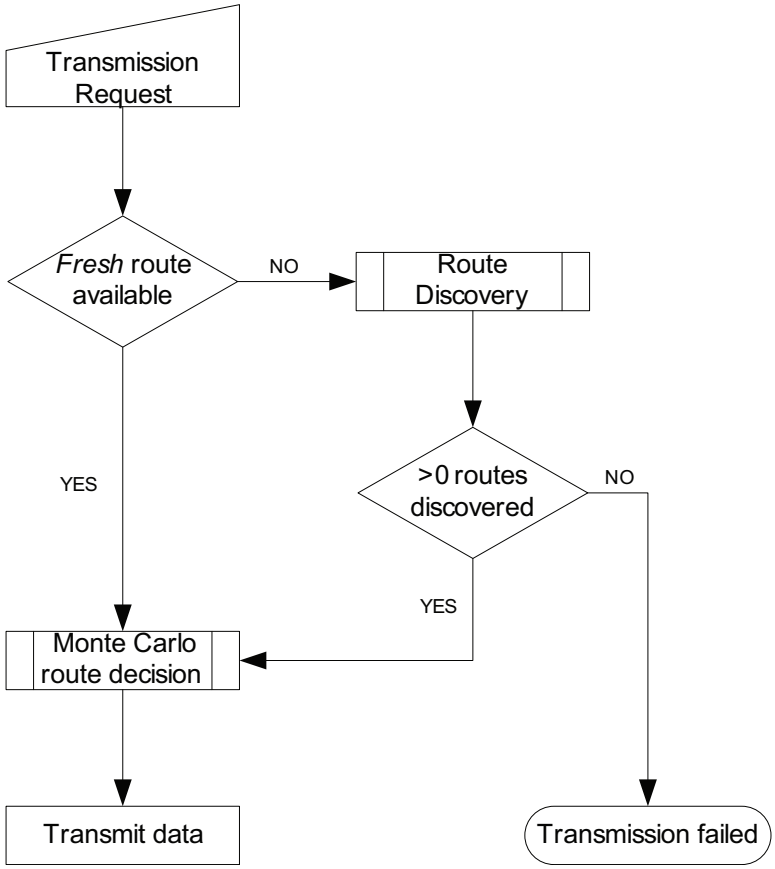


Figure 4.4: Flow Diagram of the routing strategy

4.2.4 The Application Layer

In order to test the proposed solution, data needs to originate somewhere and be transmitted to certain destinations. The application layer takes on the responsibility of making these decisions. To gain relevant output from the simulations, care must be taken in selecting the parameters of the input, as generated in the application layer. The testing process, with design choices, is covered in Chapter 7.

Since data to be transmitted originates in the application layer, the distribution of packets sent is controlled here. By determining time intervals between consecutive packets, the average data rate and distribution is controlled. Determination of inter-packet time intervals is trivial if a constant data rate is implemented, since the intervals all have the same length. If λ is the average transmission rate, then the time interval between packets would be

$$T = \frac{1}{\lambda}. \tag{4.2.1}$$

Another traffic distribution that is very applicable when it comes to arrivals, is the *Poisson* distribution. To get *Poisson* distributed traffic generated at a node, inter-packet time

intervals, $A(t)$, need to be exponentially distributed [1]:

$$A(t) = \lambda e^{-\lambda t}, \tag{4.2.2}$$

where λ is the average data rate.

4.3 Summary

This chapter evaluates available simulation environments. *OMNeT++* is selected as the platform to be utilised in the simulation of the proposed routing strategy. Design choices and development within the structure of the simulation environment are documented.

Chapter 5

Analytical Model

Contrasting the simulation approach (Chapter 4), the performance of the proposed routing protocol was evaluated by means of an analytical model. The model hinges on expected probabilistic behaviour and queueing theory.

5.1 Queueing Theory

Queueing theory entails the mathematical analysis of queues. A queue is a waiting line, consisting of a number of entities waiting for a certain service. The source population of the queue could be finite or infinite. Other factors that also define a queue, include the probability density function of the arrival process, the probability density function of the service process, the number of servers, the queueing discipline and the queueing buffer space [1]. Considering these factors, certain performance measures can be calculated, such as queue length, average waiting time in the queue or the probability of the queue finding itself in a certain state.

5.1.1 Classification of Queues

The notation most often used to classify different queueing processes, is Kendall's Notation [30]. D. G. Kendall developed this notation in 1953. It initially consisted of only three factors, $A/B/C$, but was extended to six factors, $A/B/C/K/N/D$. The different factors respectively represent the arrival process, service process, number of servers, number of places in the system, calling population and the queue's discipline. For the purpose of this study, only the first three factors are considered.

The most common case of the arrival process characteristic, A, would be Markovian (M). This entails the arrival probability density function to be a Poisson distribution. The generic version of the Poisson distribution function is

$$P_n(t) = \frac{(\lambda t)^n . e^{-\lambda t}}{n!}, \tag{5.1.1}$$

with λ representing the mean arrival rate in arrivals per second, and $P_n(t)$ representing the probability that n arrivals occurred in a time frame of length, t.

As stated in Section 4.2.4, inter-arrival times, $A(t)$, are exponentially distributed [1]:

$$A(t) = \lambda e^{-\lambda t}. \tag{5.1.2}$$

A short inter-arrival time is thus more likely than a longer one. Figure 5.1 illustrates the probability density function of the inter-arrival times if the arrivals are assumed to be Poisson distributed. Other cases of A can be General (G) or Deterministic (D).

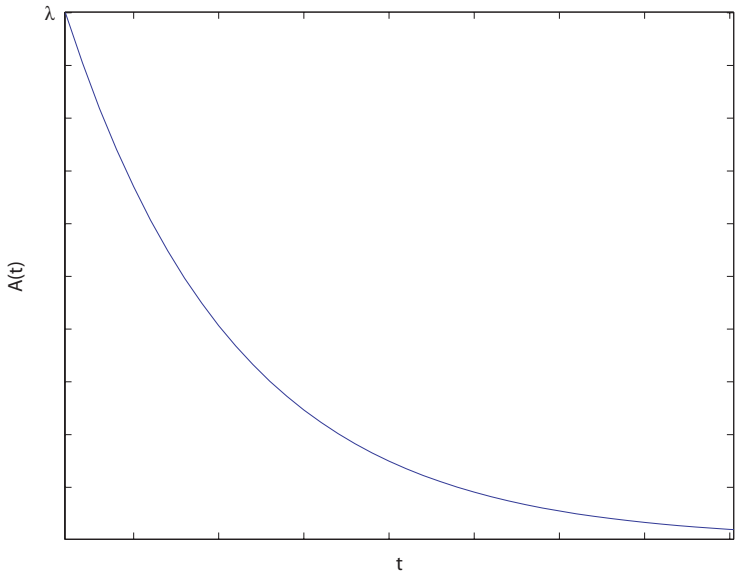


Figure 5.1: Inter-arrival time distribution function, $A(t)$

The most convenient case of the service process characteristic, B, would be to assume the service distribution also to be Poisson. In the case of this assumption, the process would thus also be classified as Markovian.

5.1.2 The $M/M/1$ Model

The arrival and service processes of this queueing model are Markovian, and the system only has one server.

A state approach will be used to analyse the queueing model. The probability of the queue finding itself in state k at a given time, thus the probability of the queue having a length of k at that time, is expressed by P_k. The arrival rate, λ, and the service rate, μ, respectively have the units *events/second*, and *customers/second*. Figure 5.2 illustrates the state approach of the queueing model. The average transition rate between state k and $k+1$ is λP_k. The average transition rate between state $k+1$ and k is μP_{k+1}. For a system to be stable, the rate of transition between states k and $k+1$ must equal the rate of transition between states $k+1$ and k:

$$\lambda P_k = \mu P_{k+1}. \tag{5.1.3}$$

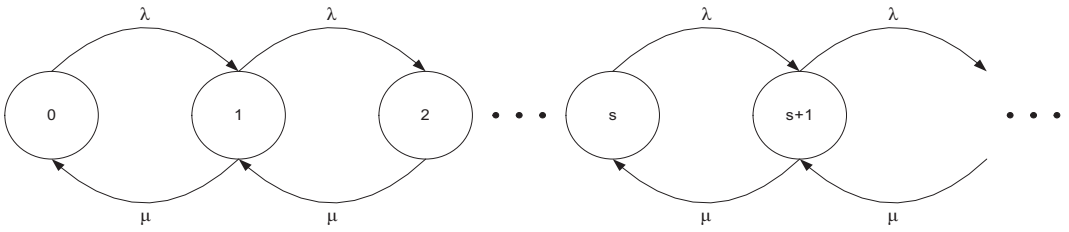


Figure 5.2: Single server queue state diagram (from [1])

The traffic intensity, ρ, is defined as

$$\rho = \frac{\lambda}{\mu}. \tag{5.1.4}$$

From (5.1.3), (5.1.4), and the knowledge that the sum of all the state probabilities must equal one, [1] derives an equation for the average queue length:

$$N = \frac{\rho}{1-\rho}. \tag{5.1.5}$$

An interesting observation is that, if the mean arrival and service rates are equal, the average queue length would strive towards infinity. This is mainly due to the Poisson distribution of the arrivals. The arrival rate is not constant. During certain intervals, the inter-arrival times would be small enough that the queue length would have a value larger than one. Whenever an entity has to wait in the queue before being serviced, the time it waits is lost. The single server can only service one entity at a time at the given service rate. The server cannot make up the time it lost by servicing the entities at a higher rate. This lost time builds up, and a boundlessly growing queue is the result.

The derivation of the average waiting time for an entity in the queue was first done by John Little in 1961. Little's result, as it is known, states that if an entity spends T seconds in the queue and is eventually in the front, all entities in the queue that arrived after him, arrived within the time slot, T [31]. The queue length (N) at the given time would thus be described by the following equation:

$$N = \lambda T. \tag{5.1.6}$$

From (5.1.6) the average waiting time can calculated as follows:

$$T = \frac{N}{\lambda}. \tag{5.1.7}$$

5.2 Assumptions

Since there are many factors that influence the behaviour evoked by a routing protocol, such as the one investigated here, certain assumptions need to be made to simplify the model and get insight into the average performance:

- Each node generates Poisson distributed traffic with a common mean arrival rate, (λ)
- The service time distribution at any node is exponential, (μ)

- Nodes are uniformly distributed and are spaced the maximum communication distance from each other, as illustrated in Figure 5.3
- The probability for node x to send a packet to node y is uniformly distributed for all values of y, except $y = x$, which entails that each node's generated traffic is distributed evenly among its neighbours, as seen in Figure 5.3
- Different metric values are uniformly distributed among all possible links in the network, which implies that an average ETX for all links exists
- Transmission is limited to a maximum amount of hops, H_{max}

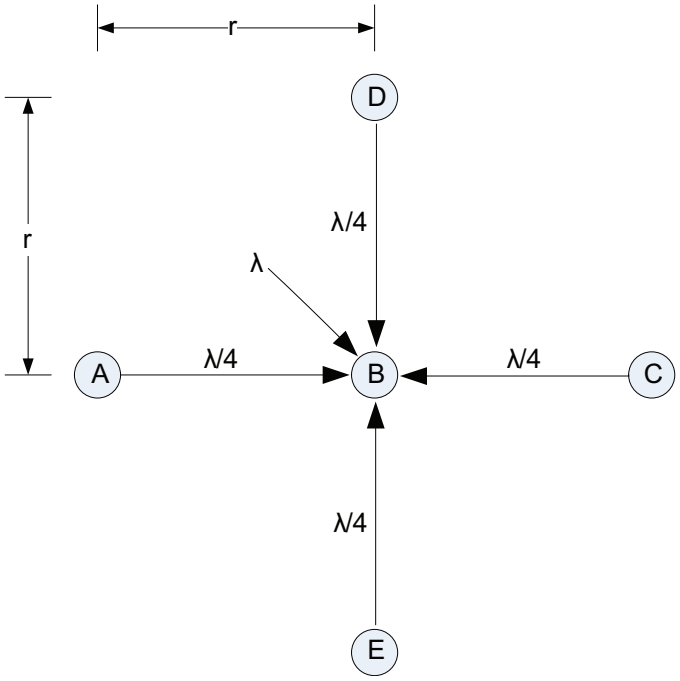


Figure 5.3: Node spacing and average generated and received traffic

5.3 Derivation

Each node is considered to be a *server*, where the *service* process is defined as sending a packet to the next hop on a given route. With regard to the assumptions made, each node can be seen as an *M/M/1* queue (Section 5.1.2), with a certain arrival rate and service tempo. For a transmission consisting of more than one hop, all conditions are met for the system to be analysed as a *Jackson Network* [32]. If the average number of hops for all transmissions in a network (H_{avg}) can be determined, the system can be seen as a single queue, but with a queue length:

$$N_{avg} = \sum_{i=1}^{H_{avg}} N_i, \tag{5.3.1}$$

where N_i is the average queue length at intermediate node i. N_i can be calculated using 5.1.4 and 5.1.5. Since N_i is equal for all nodes, (5.3.1) can be simplified:

$$N_{avg} = H_{avg} \times N_i. \tag{5.3.2}$$

H_{avg} can be calculated using the following equation:

$$H_{avg} = \sum_{i=1}^{H_{max}} i \times P_i, \tag{5.3.3}$$

where H_{max} is the maximum number of hops specified, and P_i is the probability of a transmission to consist of i hops. This leaves the problem of determining P_i.

To begin, the probability for a given node to send a packet along a route that has a minimum number of i hops, $P_{(min)i}$, is calculated by looking at the problem purely geographically. Figure 5.4 illustrates the range of communication for any given node, where r is the maximum distance of communication for any node. The concentric circles depict

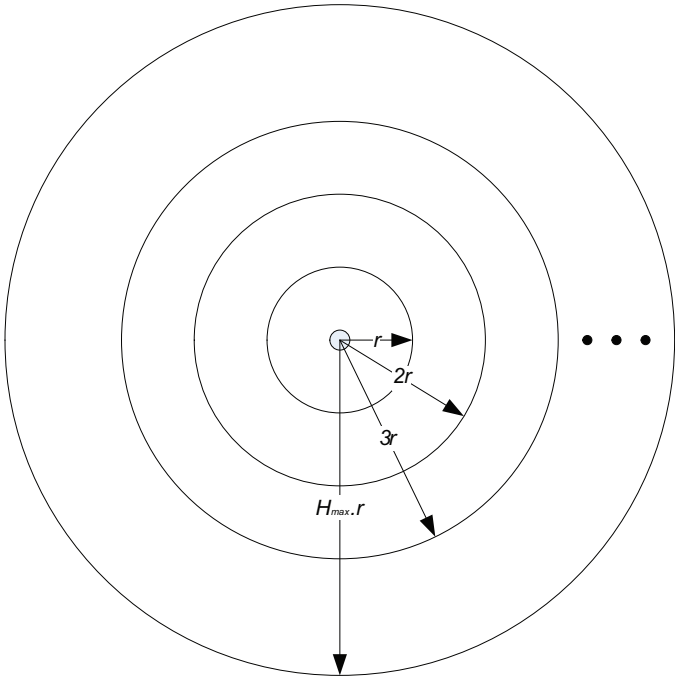


Figure 5.4: Areas reachable within a certain number of hops

the distance from the node where another hop would be needed for transmission. The area between two consecutive circles is thus the area where destinations with a common minimum number of hops between themselves and the centre node can find themselves. Since nodes are assumed to be uniformly distributed, $P_{(min)i}$ is calculated as the ratio between the area where a minimum number of i hops is required for transmission, and the entire area that can be reached within H_{max} hops. The derivation of $P_{(min)i}$ follows:

$$P_{(min)i} = \frac{\pi(ir)^2 - \pi((i-1)r)^2}{\pi(H_{max}r)^2}, \tag{5.3.4}$$

which can be simplified:

$$P_{(min)i} = \frac{2i-1}{H_{max}^2}. \tag{5.3.5}$$

Since the decision of which route to transmit data on is governed by a weighted probability, the route with the least amount of hops would thus not necessarily be utilised. The

assumption that an average ETX (ETX_{avg}) exists for all links implies that the likelihood of a given route being utilised would be higher than that of another route with a higher hop count. Since a lower ETX is more desirable, the probability of a route consisting of x hops being chosen, while it is given that the destination can only be reached in a minimum of y hops, is calculated using:

$$P_{xy} = \frac{P_{(mul)xy} \cdot \frac{1}{x(ETX_{avg})}}{\sum_{i=y}^{H_{max}} P_{(mul)iy} \cdot \frac{1}{i(ETX_{avg})}}, \tag{5.3.6}$$

which can be simplified:

$$P_{xy} = \frac{P_{(mul)xy} \cdot \frac{1}{x}}{\sum_{i=y}^{H_{max}} P_{(mul)iy} \cdot \frac{1}{i}}. \tag{5.3.7}$$

$P_{(mul)xy}$ is the weight representing the likelihood of a transmission with x hops, given a minimum number of hops, y. This weight is incorporated since the larger the number of hops, the larger the number of possible routes with the same number of hops. Since it is assumed that all nodes are uniformly distributed and are spaced the maximum communication distance from each other, $P_{(mul)xy}$ can be determined by counting the amount of possible routes with hop count x, given a minimum amount of y hops between two nodes.

With the probability of transmission with a minimum number of hops, $P_{(min)i}$, and the probability of transmission with more hops, given a minimum number of hops, P_{xy}, known, the probability of transmission with x hops can be calculated:

$$P_x = \sum_{i=1}^{x} P_{(min)i} . P_{xi}. \tag{5.3.8}$$

From equations (5.3.1) - (5.3.8) it is possible to calculate an average queue length for the network as a whole (N_{avg}). Substituting N_{avg} into Little's equation, 5.1.7, average latency (T) is derived.

5.4 Summary

This chapter introduces an analytical approach to evaluating the performance of the developed routing protocol. A set of assumptions is made in an effort to emulate the average network configuration. Average latency can be calculated by means of incorporating the assumptions into certain queueing theory fundamentals. By taking a completely different approach to analysing a certain aspect of the developed routing protocol, additional insight and confidence in the strategy can be gained.

Chapter 6

Hardware Implementation

In order to increase confidence in the proposed solution (Chapter 3), it was decided to investigate its dynamics when integrated into an ad hoc network hardware implementation. A routing protocol was not developed from scratch, but due to certain hardware limitations an existing protocol was rather utilised and altered to incorporate the *Monte Carlo* load balancing specifically.

6.1 Utilised Hardware

Due to the availability of hardware and literature, *Tmote Sky* modules were used in the implementation. The modules are manufactured by the *Moteiv Corporation*. The *Tmote Sky* module, as shown in Figure 6.1, is a platform designed for the rapid development of low power and high data-rate multihop network applications [2] [33]. The modules offer a

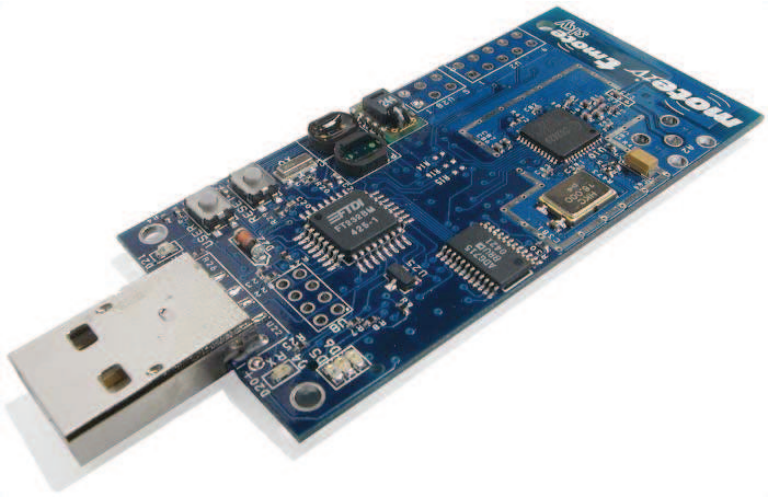


Figure 6.1: *Tmote Sky* module (from [2])

wide range of applications due to its utilisation of industry standards and range of sensors (humidity, light and temperature). Key features of the module are (reproduced from [2]):

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver

- Interoperability with other IEEE 802.15.4 devices
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption
- Fast wakeup from sleep ($< 6\mu s$)
- Hardware link-layer encryption and authentication
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector
- TinyOS support : mesh networking and communication implementation
- Complies with FCC Part 15 and Industry Canada regulations

For the purpose of testing the performance of an ad hoc routing protocol, the modules would merely transmit data between each other. The lightweight operating system, TinyOS [34], which is installed on the modules, provides numerous functional components which can be altered and extended to fit the developer's needs.

6.2 TinyOS

TinyOS is an open-source operating system which is specifically designed for wireless embedded sensor networks [1] [34]. Strong focus in the system architecture falls on component-based design. Extensive component libraries, which include network protocols, distributed services, sensor drivers and data acquisition tools, aid in the rapid development of the user's specific application.

The distribution of TinyOS which is compatible with the *Tmote Sky* module, is called *Boomerang*. *Boomerang* is certified by the *Moteiv Corporation*, and is thus compatible with all *Moteiv* hardware [33]. A ready to use multihop communication system is included in the distribution. Extension and alteration are thus all that is needed to test aspects of the proposed solution (Chapter 3).

6.2.1 Development Environment

Component functionality in TinyOS is developed in Network embedded system C (NesC), which is a dialect of C which has some additional language features for components and concurrency [34]. A NesC program consists of one or more components which are *wired* together. Various components are thus defined, with a top level configuration file linking the components together.

[1] A sensor network is an ad hoc network typically consisting of small modules, or *motes*, cooperatively monitoring environmental conditions, such as temperature, humidity, light, and many more [35].

6.3 Routing Strategy

The utilised routing strategy, *MultihopLQI*, is proactive in nature, since each node sends so called *beacon* messages to its neighbouring nodes[2] routinely with a predefined time interval. Since the strategy is primarily aimed at sensor network operation, a base node exists, which acts as the destination node for all communication. The metric used to determine the *cost* of a given link is called the Link Quality Indicator (LQI). The LQI of a given link is determined by a sampling of the error rate of the first eight symbols of each incoming packet [36]. Nodes neighbouring the base node would receive *beacon* messages from the base node, and would thus know that they are within a single hop from the destination. As neighbouring nodes send their own *beacon* messages to their respective neighbours, information regarding the link *costs* and number of hops to the base node is propagated across the network.

When a data packet is to be sent to the base node, the route with the smallest *cost* is then selected from the entries in the route table. For the purpose of testing the principle of the proposed methodology, as presented in Chapter 3, the route selection from the route table is merely altered to incorporate a weighted decision making. Route *costs* are thus used in the *Monte Carlo* process to determine the probability of a given route to be chosen for data transmission. The proposed methodology is thus not replicated in hardware. The principle behind the *Monte Carlo* load balancing strategy is explored.

6.4 Practical Considerations

In recording data from such a network setup, certain considerations needed to be taken into account. Recording statistics is not as trivial as is the case for network simulation in software, since nodes operate independently. Placement of the nodes, in order to manifest multiple independent routes proved to be challenging as well.

6.4.1 Recording Statistics

The recording of throughput proved extremely troublesome. The data rate is set by defining the data packet size and the interval between consecutive packets being sent. Knowledge of when data recording commences and ends is thus theoretically enough information to determine the number of packets sent by each node. However, it was practically found not to be this trivial. An extremely low data rate network consisting of merely two nodes proved to drop packets on occasions. Insufficient information regarding the operation of the MAC layer utilised on the *Tmote Sky* led to the decision to rather not attempt recording of throughput.

The recording of packet latency introduced another problem. Latency is calculated by time stamping a packet when it is sent and received, and then calculating the difference between the two stamps. Nodes need to be synchronised in order to establish relevance between send and receive times of packets. The problem was overcome by giving each node a millisecond counter. A program was written to reset each node's counter over

[2]Nodes within direct communication range

a Universal Serial Bus (USB) connection, and simultaneously writing the computer's current time to a file. Since the base node is connected to the computer, the computer's clock is used for the timestamp upon reception of a data packet. With the sender node's millisecond counter value embedded in the data packet's metadata, the synchronisation file provides enough information to determine when the packet was sent, according to the computer's clock. Latency can thus be calculated accurately. The only assumption that is made in the process is that USB communication delays are negligible.

6.4.2 Node Placement

Placing nodes in positions as to ensure independent multiple paths exist proved challenging, since the range of communication for the modules turned out to be relatively far. It was decided to rather introduce some form of attenuation into the network to shield certain nodes from one another. By shielding nodes, it was possible to ensure that multiple paths are available between certain nodes.

Various shielding techniques were attempted. Nodes placed in aluminium foil cylinders proved to only be able to communicate with nodes placed at the entrances of the cylinders. Figure 6.2 illustrates an example of how multiple independent paths to a destination node can be manifested by using such cylinders in shielding the nodes from one another.

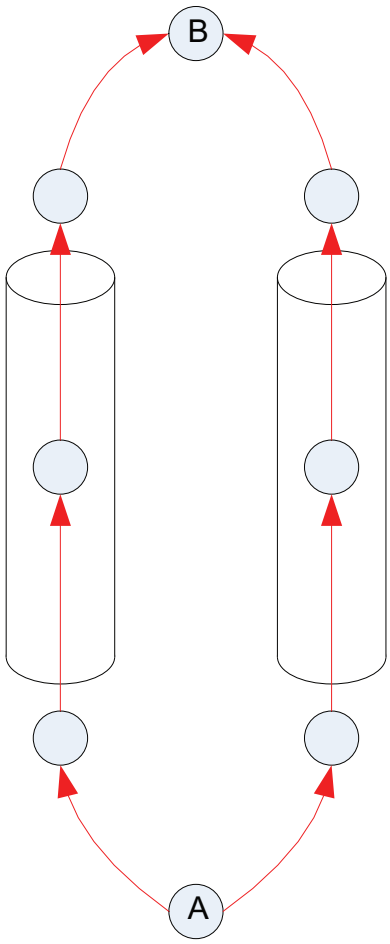


Figure 6.2: Example of multiple independent routes created by shielding effect of aluminium foil cylinders

A *Java* application, called *Trawler*, came in handy when determining whether nodes could communicate or not. The application gives a visual representation of the nodes as discovered in the network. It also gives an indication of which nodes are within communication range of one another, and also shows metric values of the given links. Data packet transmission is also logged. Figure 6.3 illustrates typical output as generated by *Trawler*.

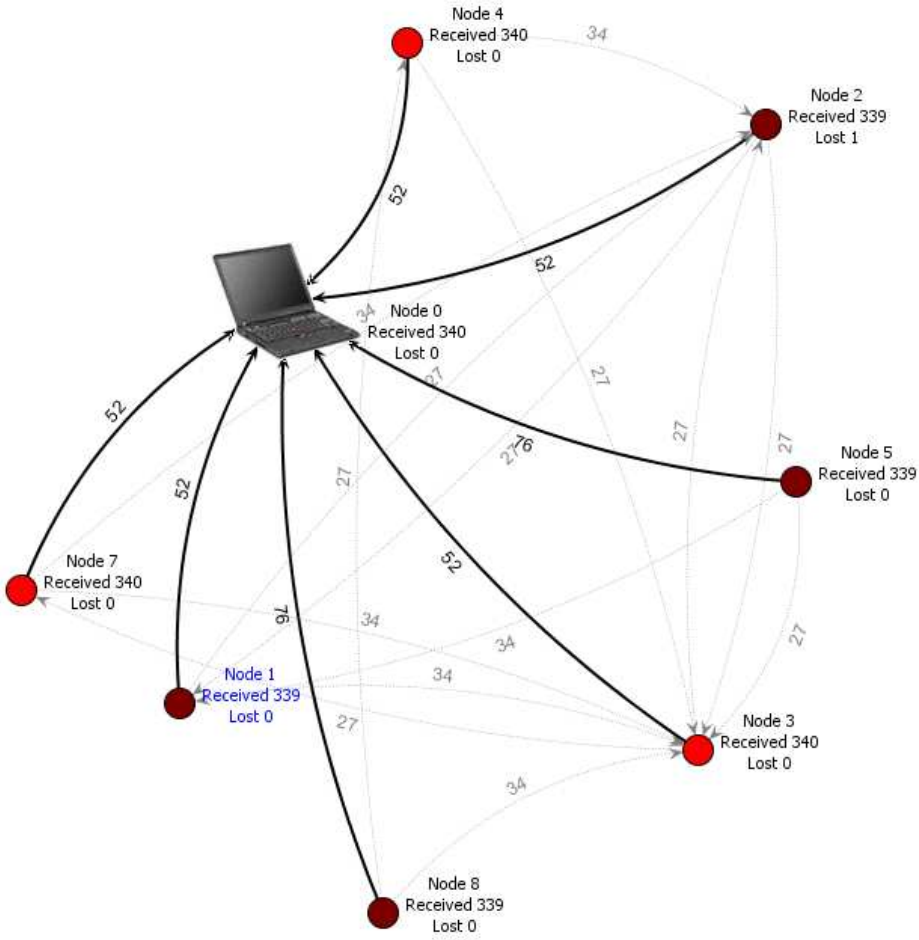


Figure 6.3: Typical output as generated by *Trawler*

6.5 Summary

This chapter introduced a hardware platform that could be utilised to develop and test certain aspects of the proposed routing protocol (Chapter 3). Relevant information regarding the operation of the hardware modules is set out. Practical considerations in the testing situation are also stipulated.

Chapter 7

Testing and Results

This chapter does not merely document results of various random test simulations or calculations, but also takes a deeper look into the process of efficient and relevant testing, with regard to an ad hoc routing protocol. With communication optimisation within an ad hoc network the primary goal that is strived towards, care has to be taken in choosing relevant test setup variables. Tests must provide realistic results that produce confidence in a real-world implementation of the proposed solution.

The goals of the chapter are:

- Identification of critical parameters that have a significant effect on protocol performance.
- Determination of the parameter values that approximate an optimised solution with regard to protocol performance
- Determination of whether the proposed methodology (Chapter 3) does introduce improvement over similar routing strategies currently being utilised.

7.1 Test Scenarios

The testing of a proposed ad hoc routing protocol is extremely important, since the feasibility of the design is judged on the results obtained in the tests. It is thus imperative to explore various test scenarios.

7.1.1 Performance Measures

Exactly what makes a certain routing protocol *better* than another depends on a list of factors. The first consideration to have in mind is what the user defines as being *good*, which comes down to a user's specific needs. A few examples of a user's needs, with functionality provided by a routing protocol in mind, could be:

- Low latency transmission

- Minimal network congestion due to control overhead
- High delivery ratio

These user needs are not the only possible needs, but assessment of numerous ad hoc routing protocol performance studies ([6, 7, 8, 37, 38, 39]) led to the focus falling on the above mentioned factors.

For every scenario, there needs to be a *control* test to compare results to.

7.1.2 Considerations

With the testing of different implementations of ad hoc networks, various factors need to be considered carefully.

7.1.2.1 Network Topology

The network topology in a test setup has a dramatic effect on the results. This is extremely relevant in the testing of the proposed methodology, as described in Chapter 3, since the strategy relies heavily on multi-path routing. A network topology that does not have multiple routes between nodes would thus not benefit from the strategy. Figure 7.1 illustrates a topology that would suffer from this problem. Each node in Figure 7.1 has a circle around it which represents the given node's communication range. There only exists one possible route between any of the nodes. A situation such as this is highly unlikely, but this extreme situation does illustrate the effect different topologies could have on test results.

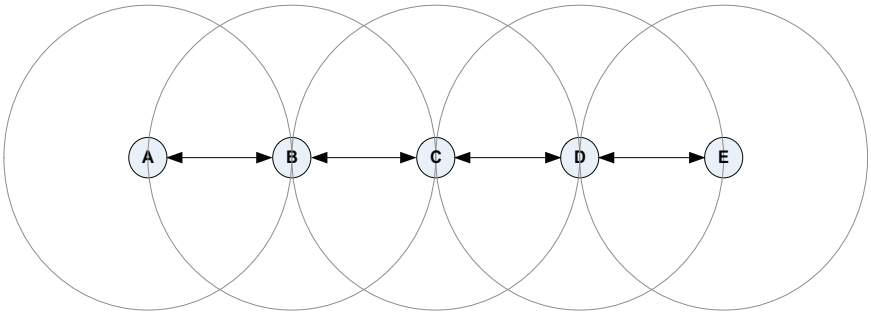


Figure 7.1: Network topology without multiple routes

Another network topology that could have a significant effect on the *performance* (Section 7.1.1) would be where there are certain links that are common to a significant number of routes. The common link would become a bottle neck, and the advantage that multiple routes provided would be mitigated by this. Figure 7.2 illustrates an example of such a situation. Since nodes C and D are common to many of the possible routes in the figure, they would serve as intermediate nodes in many transmissions, which would limit throughput and increase latency. This phenomenon is likely to be found to some extent in any given network setup. It does, however, have a significant effect on communication within an ad hoc network.

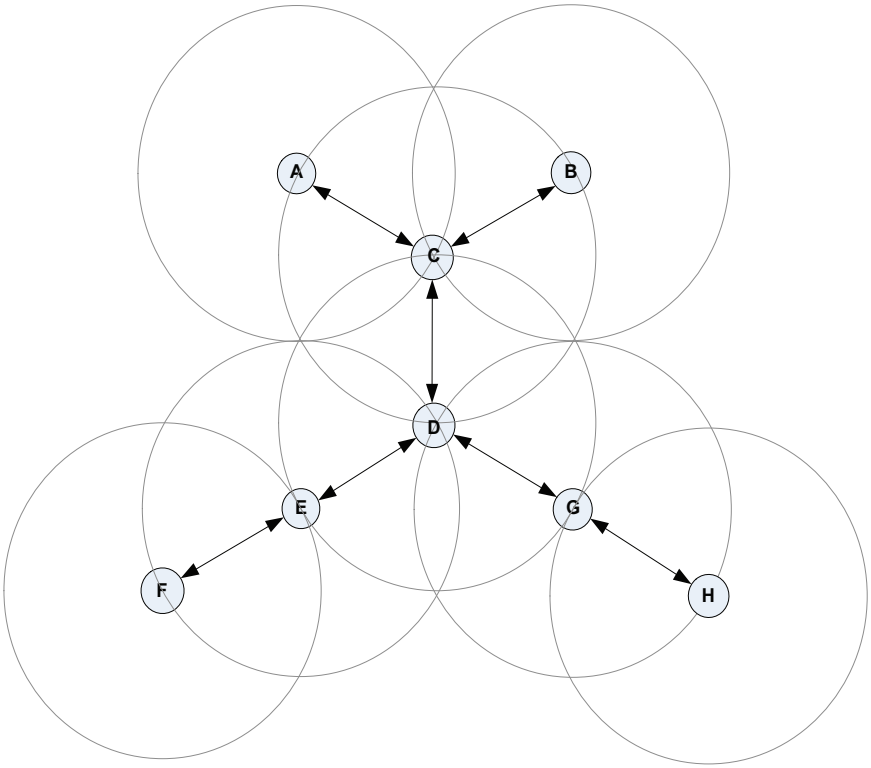


Figure 7.2: Network topology with bottle neck link

A third topology factor that might have significant consequences in an ad hoc network, is the density of nodes. A very dense network, where nodes are so close together that most of them are within direct communication range, would not benefit from an ad hoc routing strategy. There would be a large number of possible routes, where most of them share the same communication medium, which would be the given frequency spectrum in that area. Too many nodes in the same area can only interfere with each other [40]. Since each node generates control overhead, congestion would increase with every node in proximity of one another trying to access the spectrum simultaneously. Multi-path multi-hop routing, where nodes are spaced further away from each other, where multiple routes between nodes exist that are in part independent from one another, would benefit much more from the strategy. Figure 7.3 illustrates an example of where multiple routes that are in part independent from one another exist between two nodes. There are three different routes available between nodes A and B. Depending on the metrics of the three routes, data transmitted between these two nodes can be routed along a certain path while other parts of the network remain unaffected.

It is thus evident that topology could have a significant effect on communication within an ad hoc network. There would be topologies that play to a certain routing strategy's strengths, while other configurations would deliver inferior results. Identification of topologies where the proposed routing strategy thrives, as well as where it struggles, would provide insight into when the protocol should be used, or how it can be altered and improved.

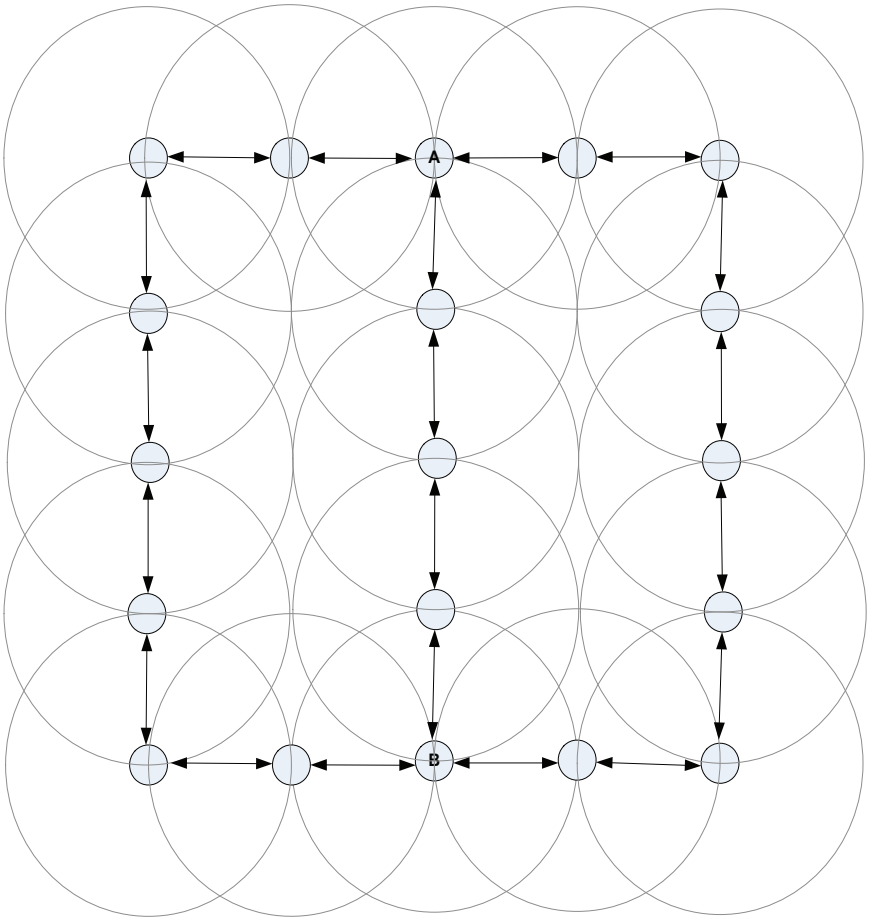


Figure 7.3: Network topology with independent routes

7.1.2.2 Network Size

The number of nodes, and the distance between them play a role in the overall network performance as well. As the number of nodes in a network increases, the control traffic also increases. If physical size of the network (average distance between nodes) is increased with the number of nodes, the increase in network traffic would not be as severe, since less nodes would then be in direct communication range with each other, which would decrease the likelihood of packet collisions.

7.1.2.3 Load Distribution

The distribution of data transmitted over a given network also plays a critical role in the performance evaluation of a developed routing protocol. Not only the distribution of data originating at nodes, but also the destination of the data has significant impact on the dynamics of the test.

The challenge lies in determining the average load distribution of all networks that might implement the proposed routing protocol. If performance proves to show improvement over existing technologies, while such a distribution of data is injected over the network, it would ensure confidence that the developed protocol is a feasible alternative on most potential target networks. Finding such a *golden* average is something that is not necessarily achievable. It was thus decided to attempt to recreate an average distribution of data,

but also to consider other distributions that are more specific to certain circumstances, as done in Section 7.3.1.

7.2 Parameter Optimisation

In order to strive towards obtaining the best possible results from the given solution (Chapter 3), the parameters need to be optimised. Since there are a number of variables, which potentially fall within a large range of values, the possible combinations of values are extremely large. Simulation of every single combination becomes an extremely resource intensive exercise. For that reason it was decided to use simulation optimisation methods [3] for this purpose. The simulation setup, as explained in Chapter 4, is used in the optimisation process.

7.2.1 Parameter Identification

In the design and simulation process, a set of critical parameters were defined that each play a significant role in the protocol performance, where performance is measured as stated in Section 7.1.1.

Metric Bound (considerFact)

After route discovery has taken place, it is possible that multiple routes are available that the data can be propagated along. The proposed methodology (Section 3.2) states that the probability of a given route to be utilised is proportional to its metric. Even though the probability would be relatively small, it is thus still possible for a route with an extremely *bad* metric to be selected for data transportation. A maximum percentage value is thus set that states how much *worse* a certain route is allowed to be compared to the *best* route found in the given route discovery iteration. How *good* a route is is determined by its metric (Section 2.4). If a route does not fall within the defined bounds, it is not even considered for communication.

Number of best routes (routeNum)

Another measure put in place to ensure that load balancing is performed without hindering overall performance by utilising *bad* routes, is to only consider the *best* routes for communication. How many of the *best* routes should be considered is thus another variable to be investigated.

Interval between HELLO packets (helloInterval)

As explained in Section 4.2.3.1, HELLO packets are probes sent at a constant rate from every node in order to discover information regarding the link quality between the node itself and nodes within communication range from it. A smaller interval between the

HELLO packets would result in more trustworthy information regarding route metrics, but comes at the cost of increasing network traffic.

Time frame a route remains fresh (freshFactor)

Route discovery only commences if a route to the given destination has not been found within a certain time frame ago (Section 4.2.3.2). If the time frame is too short, unnecessary control overhead would be produced, but if it is too long, the network could be utilised sub-optimally, since out of date routes could be used while much better options are available.

7.2.2 Methods

The idea behind a simulation optimisation method is to find the optimal input values without iterating through every single possible combination of parameter values [3].

Figure 7.4 illustrates a general simulation model. Simulation optimisation aims to find

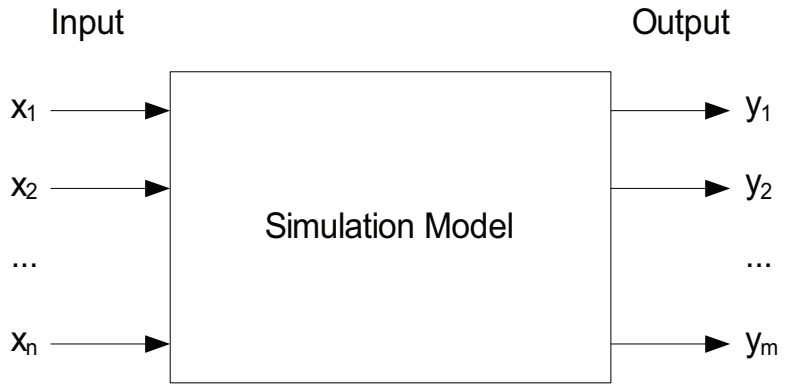


Figure 7.4: A simulation model [3]

the values of the n input variables in order to optimise the m output variables.

It works on the principle of a feedback system. Figure 7.5 illustrates the dynamics of the method. Significant research has been done in the field of optimisation strategies. The

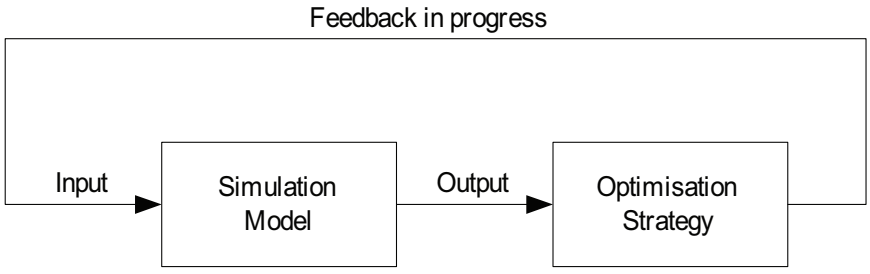


Figure 7.5: A simulation optimisation model [3]

main categories of these methods are:

- Gradient Based Search Methods
- Stochastic Optimisation
- Response Surface Methodology (RSM)
- Heuristic Methods
- Asynchronous Teams (A-Teams)
- Statistical Methods

Due to the nature of the simulation inputs and setup, it was decided to use GAs for the problem. GAs form part of the Heuristic Methods category. Before optimisation the input variables mostly have large ranges which they could fall into, since they are defined by means of educated guesses. The idea behind the GAs is to get those ranges as small as possible.

7.2.2.1 Genetic Algorithms (GAs)

GAs are a class of Evolutionary Algorithms (EAs). EAs use principles inspired by evolutionary biology [3], such as, selection, reproduction, crossover and mutation [41].

The *gene pool* of the evolutionary process in the method is a population of candidate solutions. Usually the evolution begins with population of candidate solutions randomly selected (*Initialising*). The process progresses in generations. The first, randomly selected population, would thus be the first generation. The candidates are then evaluated according to a predefined *fitness* function. A subset of the population is then selected based on their *fitness* (*Selection*). A new population is then created by modifying the selection (*Reproduction*). The modification is usually a recombination and possibly mutation of the selection. The new population, or generation, is then used in the next iteration of the algorithm. Termination of the algorithm usually occurs when a maximum number of iterations has been executed, or a satisfactory *fitness* level has been reached. Figure 7.6 illustrates the flow of the process.

7.2.3 Implementation

7.2.3.1 Setup

Taking the considerations into account as highlighted in Section 7.1.2, it was decided to perform optimisation with a uniformly distributed network topology, as illustrated in Figure 7.7, where traffic is generated with a Poisson distribution at every node. If traffic at all nodes is Poisson distributed, then the total traffic is also Poisson distributed, where the average data rate introduced to the network as a whole, is the sum of the average data rates introduced to the individual nodes [42]. Destination nodes are randomly selected with a uniform distribution. More elaborate testing of different scenarios is done in Section 7.3, where the proposed solution is not compared to itself, but to other protocols.

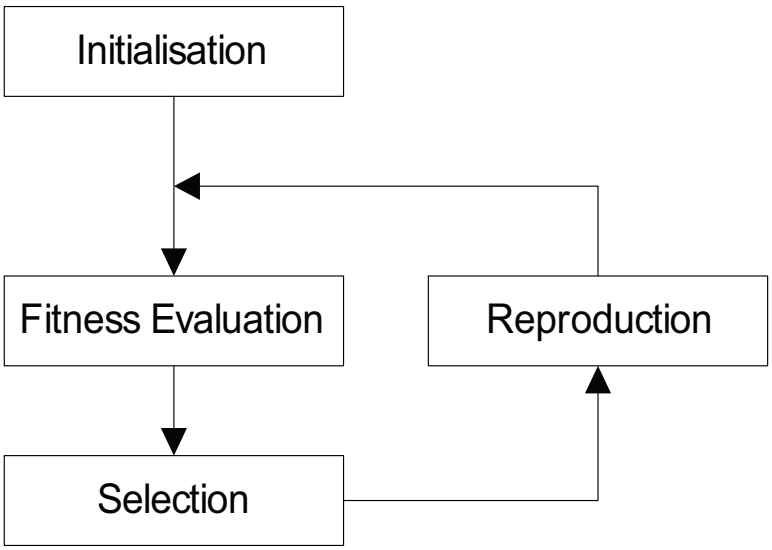


Figure 7.6: High level flow diagram of the Genetic Algorithm

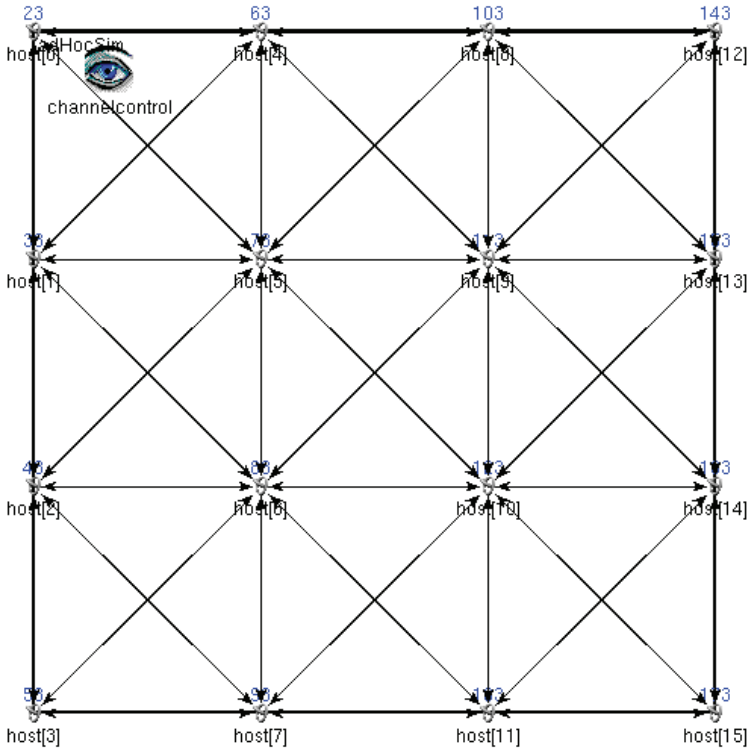


Figure 7.7: Network topology utilised in parameter optimisation simulations

The accuracy of the process increases with the number of simulation runs performed. It was decided to generate twenty sets of random values for the parameters under investigation for each generation. Each parameter set was used in three separate simulation runs, where the data rate was set at a quarter, a half, and one time the maximum data rate possible across a given link.

Table 7.1 shows the values assigned to the network variables.

Table 7.1: Optimisation setup variables

Simulation time	500 s
Number of nodes	16
Distance between nodes	150 m
Carrier Frequency	2.4 GHz
Maximum possible bit rate	5.5 Mbps
Transmit power	110.11 mW
Data packet size	1 kB

7.2.3.2 Initialisation

According to the steps as illustrated in Figure 7.6, the parameters in question (Section 7.2.1) need to first be initialised. Since the optimal values for the variables are initially unknown, a fairly large range is defined for every one of them, as stipulated in Table 7.2.

Table 7.2: Initial range of variables to be optimised

considerFact	$0 \rightarrow 50\%$
freshFactor	$0 \rightarrow 30 seconds$
helloInterval	$1 \rightarrow 10 seconds$
routeNum	$1 \rightarrow 3$

For the first generation in the process, it was decided to randomly select the different parameter sets with uniform distributions across the chosen ranges for the parameters.

7.2.3.3 Fitness Evaluation

The *fitness* function needs to be defined. Section 7.1.1 defines the performance measures chosen for the analysis of the developed routing protocol. For the purpose of optimisation, it was decided to perform two separate runs of the entire GA, with the one having latency, and the other having throughput as its *fitness* functions. The effect control overhead has on network performance is already taken into account since the interval between HELLO packets (helloInterval) is one of the parameters under investigation.

Since each random parameter set is evaluated at three different data rates, a single metric needs to be extracted from the three results obtained. It was decided to merely calculate the average value obtained from the three runs. The factor itself does not mean anything,

but when compared to other runs, it is a sufficient performance measure, since the other factors were calculated in the same manner. In the case where throughput is investigated, a higher value is desirable, where a lower latency suggests better performance.

7.2.3.4 Selection

After the simulations comprising a generation are completed, and the *fitness* of the the respective parameter sets have been determined, the top 25% parameter sets are selected to be used in the reproduction process.

7.2.3.5 Reproduction

The parameters selected as described in Section 7.2.3.4, are used as inputs in determining the distribution function governing the random generation of parameter values for the next generation of the GA. In order to ensure a converging process, it was decided to utilise a Gaussian distribution,

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)}, \tag{7.2.1}$$

where the parameter values from the selection determine the mean (μ) and variance (σ^2) of the function. The mean is determined by calculating the average of a given parameter from the selection,

$$\mu = \frac{1}{N} \sum_{k=1}^{N} x_k, \tag{7.2.2}$$

where x_k is the simulation run parameter value of a selected parameter set, and N is the number of selected parameter sets. The variance is calculated using

$$\sigma^2 = \frac{1}{N} \sum_{k=1}^{N} (x_k - \mu)^2. \tag{7.2.3}$$

The next generation is thus produced by means of mutation, and not crossover, as done in Simulated Annealing (SA).

7.2.4 Results

Since the optimisation process was performed for the *fitness* function being network throughput, as well as packet latency, the two respective processes were investigated separately. The idea behind the GA is to have the parameters converge to certain values, where the performance measures converge towards their maximum possible values. Since each generation consists of twenty different parameter sets, it was decided to investigate the mean values and standard deviations of the respective parameters. The mean is calculated using equation 7.2.2, while the standard deviation is derived by taking the square root of the variance, which is in turn calculated using equation 7.2.3. The reason why the standard deviation is applicable, is because the majority of the samples lie within the range of the standard deviation from the mean. The two measures are thus comparable

on the same scale. These measures would indicate the level of convergence, as well as the value it converges towards. Alongside the converging parameters, the mean and standard deviation of the performance measure (*fitness*) was also investigated in order to obtain an indication of the confidence that can be obtained from a specific generation of parameters.

7.2.4.1 Optimised for Throughput

As stated in Section 7.2.3.2, the parameters under investigation were initialised with uniform distributions over the defined ranges. Table 7.3 shows the randomly generated initial input variables.

Table 7.3: Parameter optimisation initialisation values

Run	considerFact	freshFactor	helloInterval	routeNum
1	1	13	2	3
2	3	16	2	3
3	38	16	5	2
4	20	28	1	1
5	18	16	5	2
6	49	20	5	2
7	38	3	2	1
8	19	11	2	1
9	44	24	4	3
10	42	14	2	1
11	45	0	2	1
12	30	25	4	3
13	5	6	4	3
14	2	30	4	2
15	20	3	2	3
16	32	21	1	1
17	23	4	4	3
18	3	28	2	3
19	41	3	1	2
20	7	2	3	1

Figures 7.8-7.11 show the dynamics of the four respective parameters throughout the five generations of the GA. Figure 7.12 illustrates how the fitness measure was influenced by the changing input variables.

It can be seen that every one of the four parameters under investigation converges to a certain extent towards a certain mean value. The mean stabilises while the standard deviation declines. For *freshFactor*, *helloInterval* and *routeNum*, the standard deviation converged to zero within the five generations. This implies that every single one of the twenty runs in the simulation utilised the same value of the specific parameter. Figure 7.12 shows how the *fitness* metric improved, but also how it stabilised. The decreasing standard deviation increases the confidence in the improved *fitness* metric. The standard deviation, being as small as it is in the last generation, implies that the parameters, within

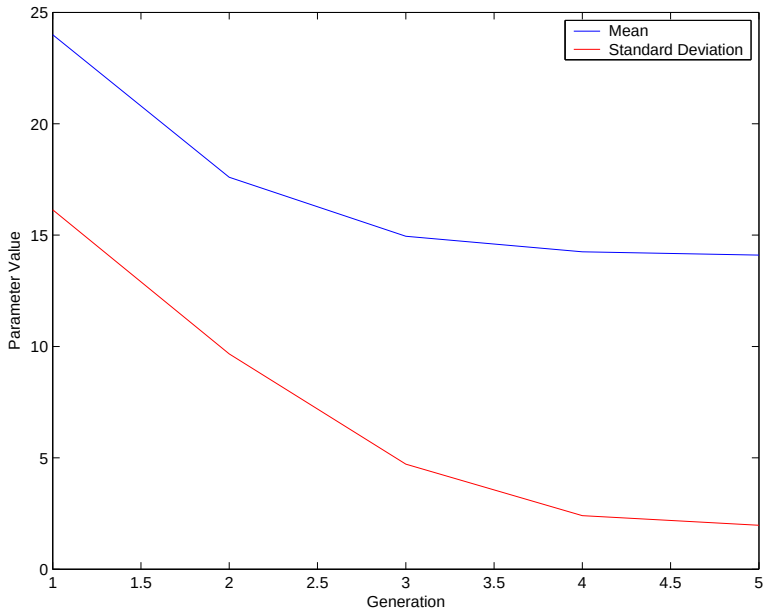


Figure 7.8: Throughput parameter optimisation dynamics of the parameter: considerFact

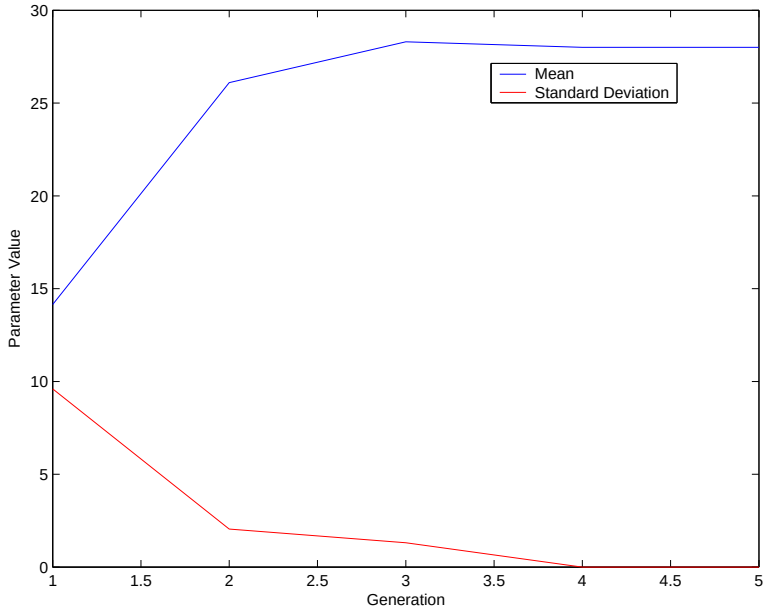


Figure 7.9: Throughput parameter optimisation dynamics of the parameter: freshFactor

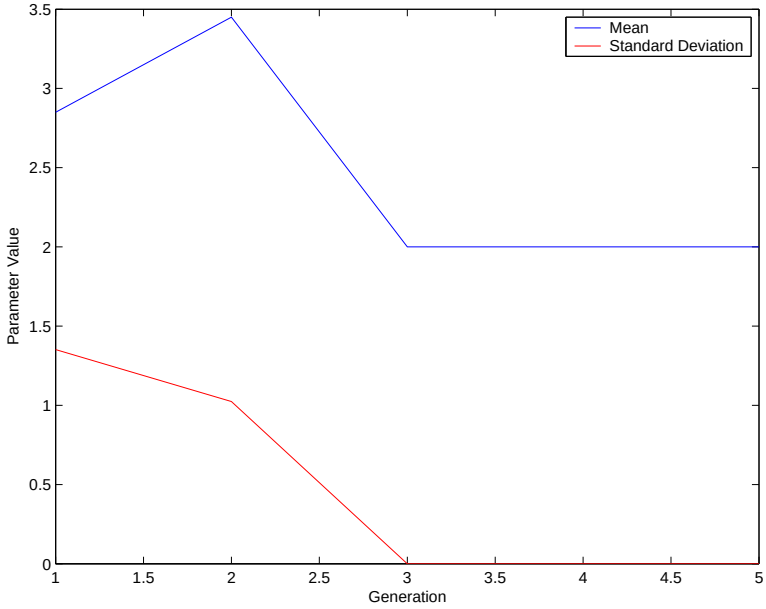


Figure 7.10: Throughput parameter optimisation dynamics of the parameter: helloInterval

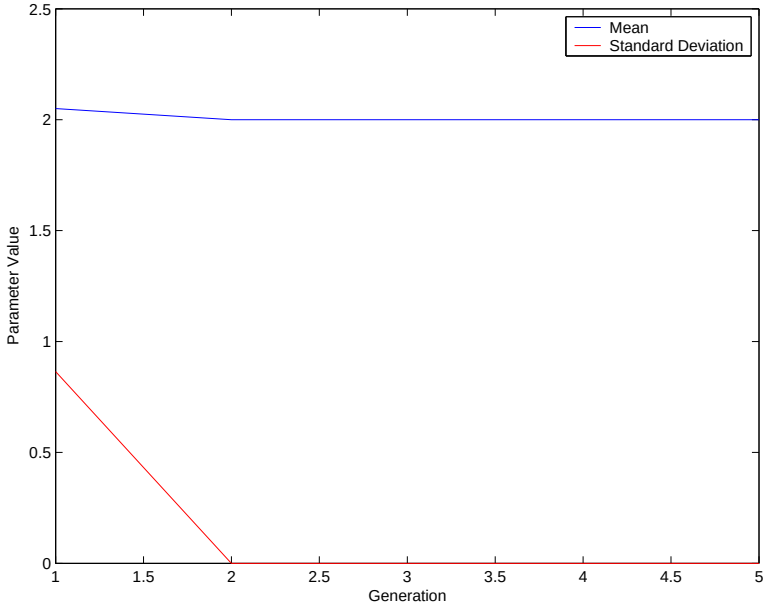


Figure 7.11: Throughput parameter optimisation dynamics of the parameter: routeNum

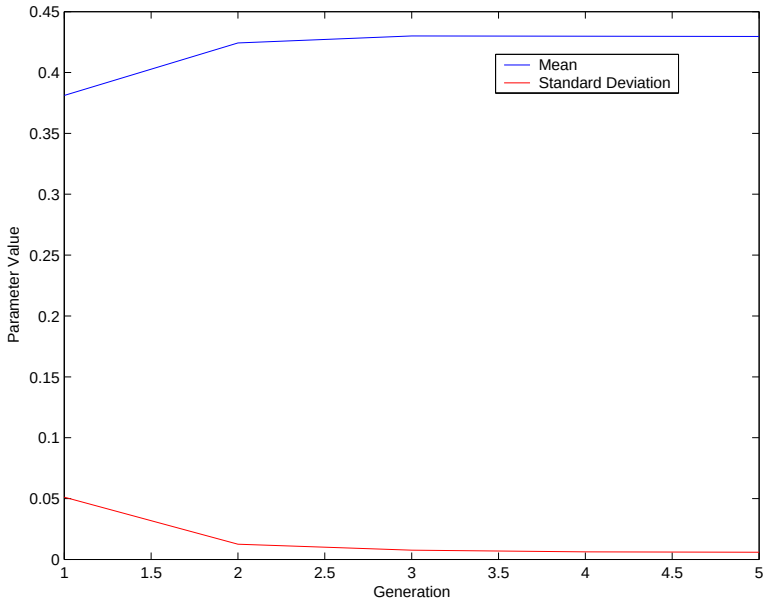


Figure 7.12: Throughput parameter optimisation dynamics of the fitness metric

their ranges for the given generation, would produce fitness metrics very close to the mean with a very high probability.

Table 7.4 shows the input values to the final generation of the GA. The convergence is evident when the values are compared to those in Table 7.3. The complete output set from the optimisation for throughput GA can be found in Appendix B.1.

Table 7.4: Final generation throughput optimisation parameter values

Run	considerFact	freshFactor	helloInterval	routeNum
1	17	28	2	2
2	11	28	2	2
3	14	28	2	2
4	12	28	2	2
5	14	28	2	2
6	14	28	2	2
7	16	28	2	2
8	11	28	2	2
9	12	28	2	2
10	14	28	2	2
11	12	28	2	2
12	15	28	2	2
13	16	28	2	2
14	15	28	2	2
15	15	28	2	2
16	13	28	2	2
17	18	28	2	2
18	17	28	2	2
19	13	28	2	2
20	13	28	2	2

7.2.4.2 Optimised for Latency

The exact initial parameter values as shown in Table 7.3 were also fed into the GA where latency was used as the defining *fitness* parameter. Figures 7.13-7.16 show the dynamics of the four respectable parameters throughout the six generations of the GA. Figure 7.17 illustrates how the fitness measure was influenced by the changing input variables. An additional generation of simulation runs was executed since some of the parameters still had relatively large standard deviations after the fifth generation.

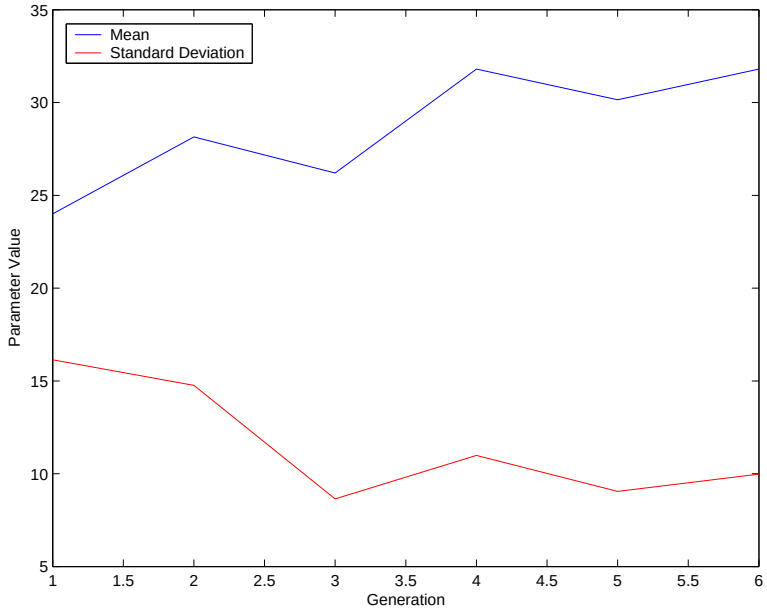


Figure 7.13: Latency parameter optimisation dynamics of the parameter: considerFact

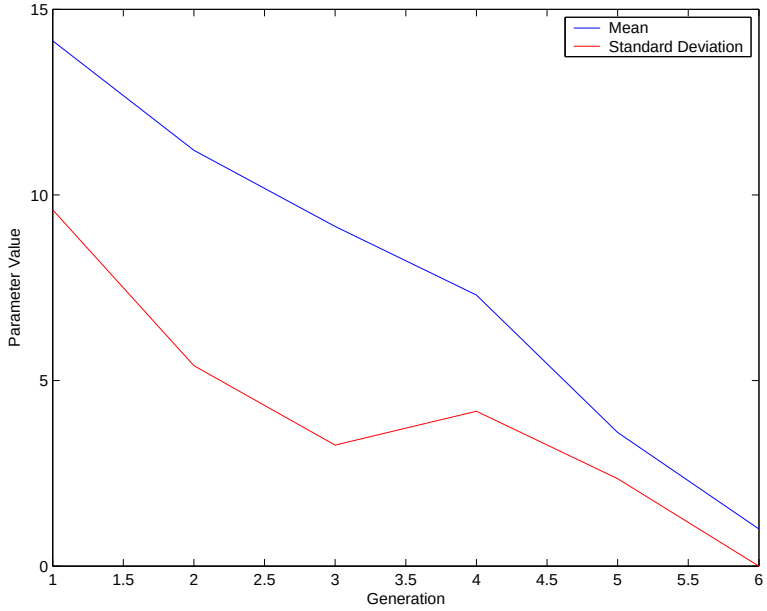


Figure 7.14: Latency parameter optimisation dynamics of the parameter: freshFactor

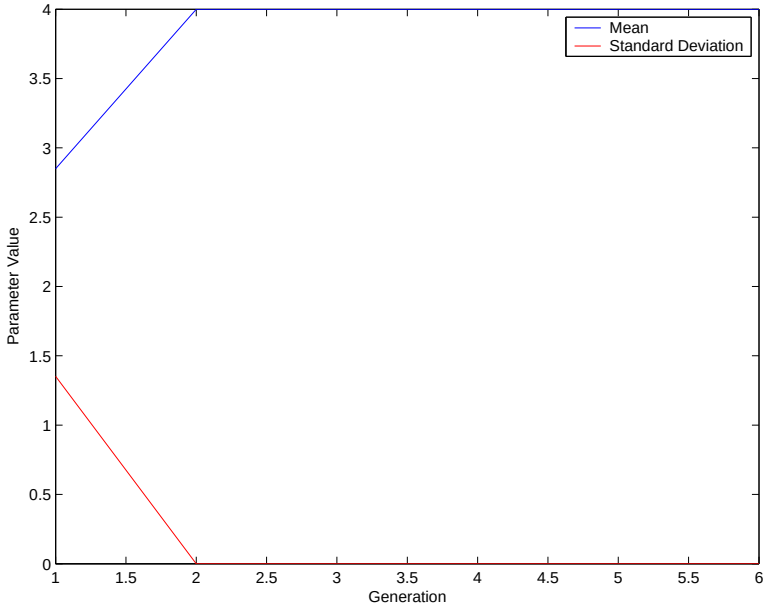


Figure 7.15: Latency parameter optimisation dynamics of the parameter: helloInterval

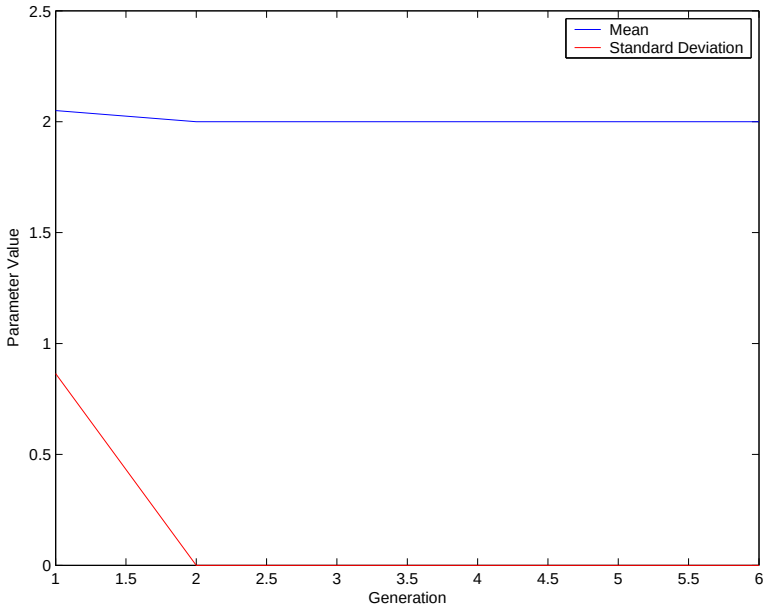


Figure 7.16: Latency parameter optimisation dynamics of the parameter: routeNum

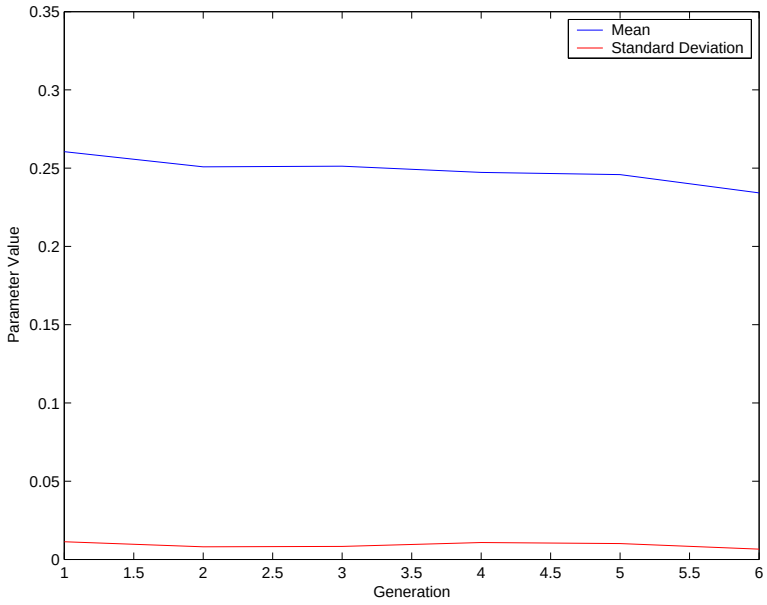


Figure 7.17: Latency parameter optimisation dynamics of the fitness metric

It can be seen from Figures 7.15 and 7.16 that *helloInterval* and *routeNum* converge quickly, since their standard deviations are both equal to zero from the second generation onwards. For *freshFactor* it takes six generations until it converges to a certain mean value. However, *considerFact* does not seem to converge since the standard deviation remains at a value in the same order of magnitude to the mean value. The reason why it was decided not to continue simulations and execute more generations of the GA, was because the *fitness* parameter's standard deviation remained stable and relatively small compared to its mean value. This implies that the mean value of the fitness parameter, as illustrated in Figure 7.17, can be obtained with a relatively high level of confidence, regardless of where the parameter value falls within the range as used in the latter simulation generations.

Table 7.5 shows the input values to the final generation of the GA. The convergence is evident when the values are compared to those in Table 7.3. The complete output set from the optimisation for latency GA can be found in Appendix B.1.

7.2.4.3 Conclusion

From the results obtained in Sections 7.2.4.1 and 7.2.4.2, it is concluded that the GA with throughput as its *fitness* measure produced parameter values with a slightly higher confidence level. The lack of convergence of the parameter, *considerFact* (Figure 7.13), is the reason for the statement. Since the latency *fitness* measure did not vary much (Figure 7.17) throughout the generations, it was decided to investigate the latency *fitness* measure with the parameter values as used in the final generation of the GA where there was optimised for throughput. Figure 7.18 shows how the the latency *fitness* parameter of the two separate implementations of the GA compare to each other. As stated in Section 7.2.4.2, an additional generation of simulation runs was executed for the GA with latency as its *fitness* measure, since certain parameters had not converged sufficiently after five generations, as was the case in the GA where throughput was the *fitness* measure (Section 7.2.4.1). Since the latency *fitness* measure is merely the average of the latencies for a certain set of parameters, where three different data rates are introduced to the

Table 7.5: Final generation latency optimisation parameter values

Run	considerFact	freshFactor	helloInterval	routeNum
1	39	1	4	2
2	34	1	4	2
3	39	1	4	2
4	38	1	4	2
5	26	1	4	2
6	29	1	4	2
7	30	1	4	2
8	47	1	4	2
9	21	1	4	2
10	34	1	4	2
11	46	1	4	2
12	27	1	4	2
13	9	1	4	2
14	40	1	4	2
15	25	1	4	2
16	19	1	4	2
17	17	1	4	2
18	34	1	4	2
19	45	1	4	2
20	37	1	4	2

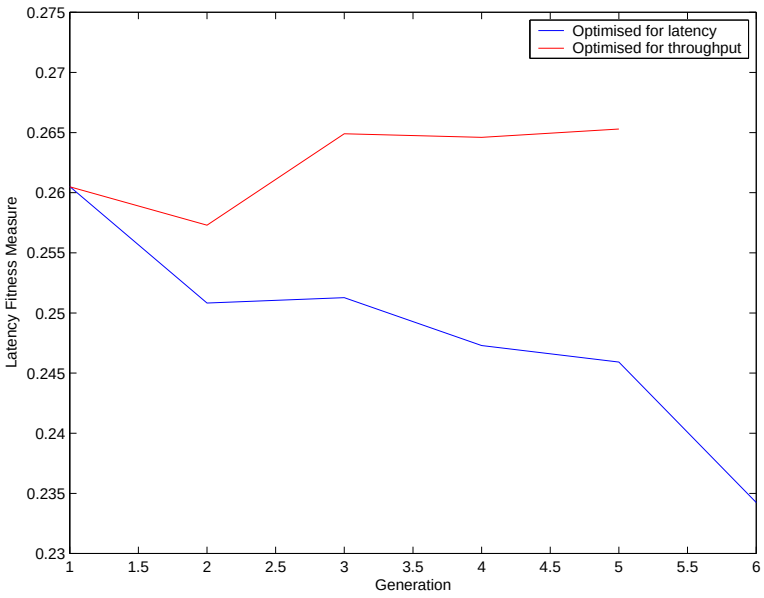


Figure 7.18: Latency *fitness* measure for both implementations of the GA

network, a lower value is more favourable. The latency *fitness* measure improved by 10% throughout the generations when optimised for latency, while the latency *fitness* measure worsened by 2.3% when optimised for throughput.

The throughput *fitness* measure with the parameter values as used in the final generation of the GA, where latency was the *fitness* measure, was also investigated. The results are shown in Figure 7.19. A higher throughput is better. The throughput *fitness* measure improved by 12.7% when optimised for throughput, while the throughput *fitness* measure worsened by 25.7% when optimised for latency.

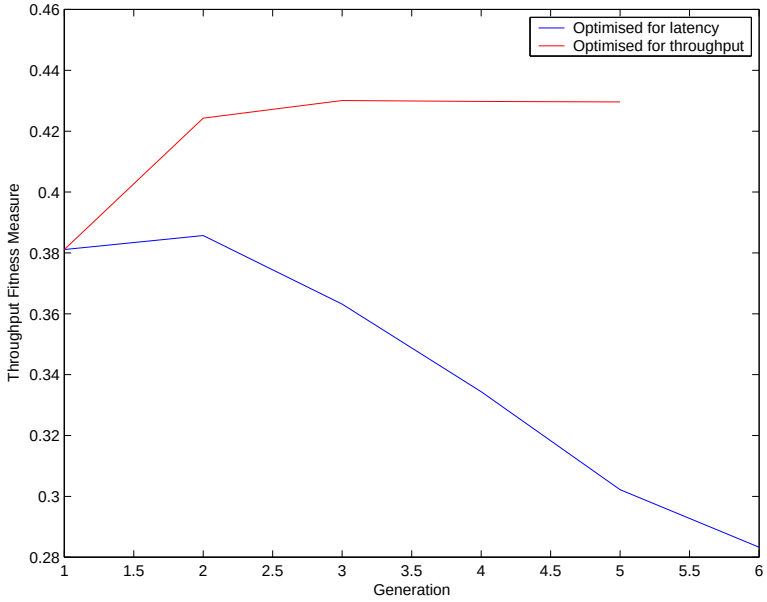


Figure 7.19: Throughput *fitness* measure for both implementations of the GA

Since the user's need determines the importance of throughput or latency, there is no golden ratio of weighing the relevance of the two performance measures. In a network where the streaming of multimedia is a priority, low latency is extremely important, while in the scenario were large amounts of data just need to be moved between two nodes, throughput would be important. The optimisation process in this Section did not merely provide one with an indication of which parameter values produce the most favourable output, but it also provided the insight that the initial parameter values were not hampering the proposed solution's (Chapter 3) performance by an order of magnitude. The possibility of such an occurrence would not have been ruled out had the process not been followed.

Because of the dramatic drop in the throughput *fitness* parameter when optimised for latency, it was decided to utilise the parameters as determined by the GA when optimising for throughput. The latency measure is 12.3% lower than what it could be, as determined in the alternative GA, but the throughput is higher by 38.4%. The mean values of the parameters, as they were in the final generation of the GA, were thus decided upon (Table 7.6).

Examples of the *Python* code used to perfrom the GAs are listed in Appendix B.2.

Table 7.6: Parameter values after optimisation process

considerFact	14.1%
freshFactor	28 seconds
helloInterval	2 seconds
routeNum	2

7.3 Performance Comparison

In order to test the feasibility of the proposed solution (Chapter 3), various network setups were simulated, with the *Monte Carlo* load balancing strategy enabled and disabled.

7.3.1 Test Scenarios

As stated in Section 7.1.2, there are various considerations to be taken into account when the performance of a routing protocol is investigated. The network topology plays a significant role in the dynamics of traffic in an ad hoc network (Section 7.1.2.1), it was decided to investigate three different topologies:

- Uniform
- Minimal multiple routes
- Random

Figure 7.7, illustrates an example of a uniformly distributed topology, as also used in the optimisation process. A uniform topology represents a scenario where multiple routes between nodes exist. Figure 7.20 illustrates a network topology with minimal multiple routes, while Figure 7.21 is an example of a randomly selected network topology. A randomly selected network topology would most probably consist of certain links which would be utilised more than others. Multiple routes would most probably exist between certain nodes, while other node pairs could only have one possible route between them.

In addition to the three different topologies under investigation, it was also decided to look at the effect the network size (Section 7.1.2.2) and the load distribution (Section 7.1.2.3) has on performance. A relatively small, spread out network (Section 7.3.2.1) and a much larger and denser network (Section 7.3.2.2) were investigated. With regards to load distribution, it was decided to consider a Poisson distributed data rate, as well as a distribution where a certain number of consecutive data packets are sent sequentially time.

Since the performance of when the *Monte Carlo* strategy is utilised is compared to when it is not used, all other setup parameters need to be kept constant. The variables under investigation in Section 7.2 were awarded the values as stated in Table 7.6. Table 7.7 shows the chosen constants used throughout the comparative simulations.

With the maximum possible data rate across any given link as defined in Table 7.7, it was decided to run simulations for the various different scenarios at the following normalised data rates (G)[1]: 1/4, 1/2, 1, 2 and 4. The reason for the wide range of data rates is to

[1]G = (actual data rate)/(maximum possible data rate)

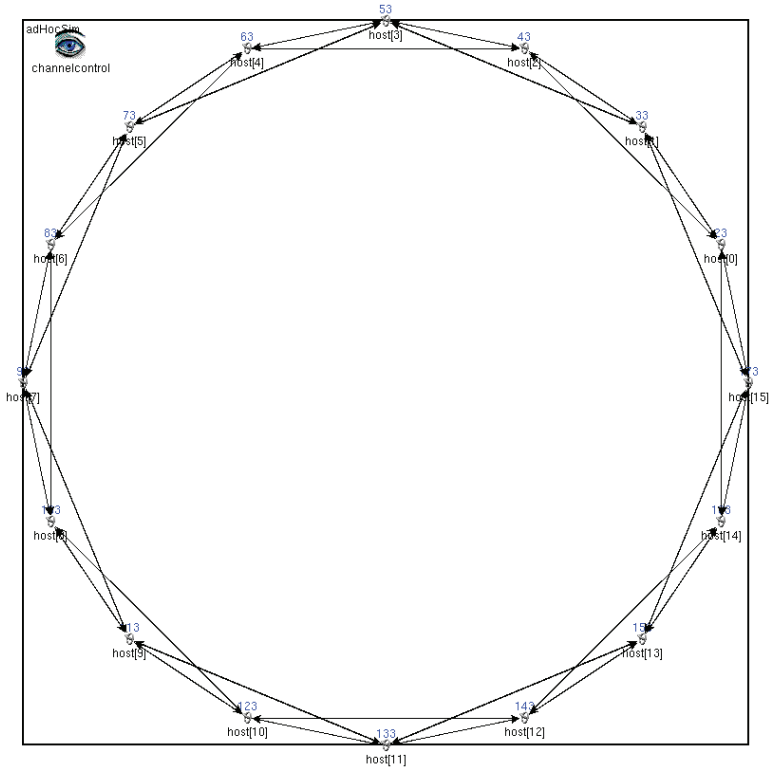


Figure 7.20: Network topology with minimal multiple routes

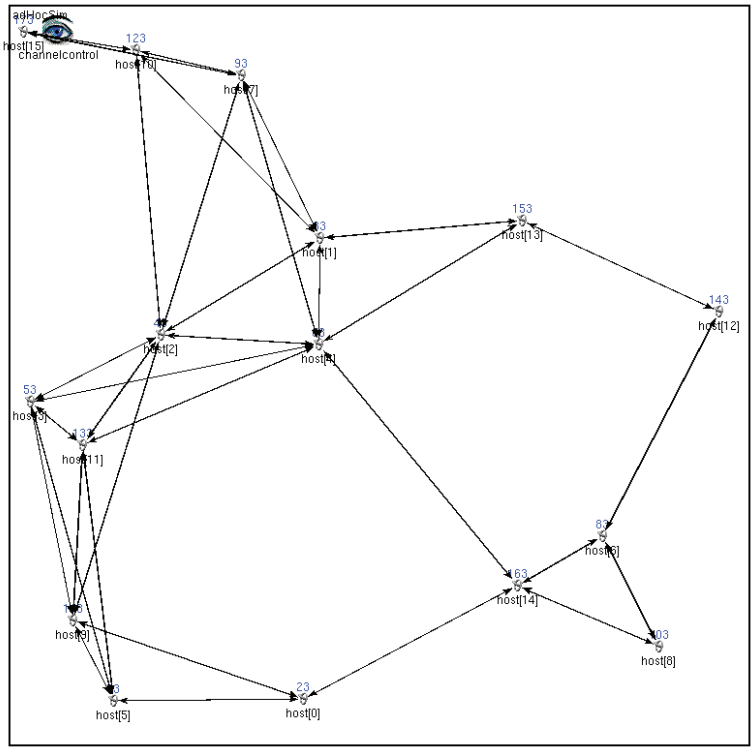


Figure 7.21: Randomly distributed network topology

Table 7.7: Comparative simulation setup parameters

Carrier frequency	2.4 GHz
Maximum possible bit rate	5.5 Mbps
Transmit power	110.11 mW
Data packet size	1 kB
Data block size	500 packets

attempt to obtain a visualisation of how the routing protocol holds up under conditions where the load on the network is relatively low, up to where the load is so high that the technology upon which the routing protocol is built surpasses its limit. The given data rate is the average rate introduced to the network as a whole. Each node produces the same average data rate. The destination node, in the case of Poisson traffic, is random with a uniform distribution. The probability that any other node in the network could be the destination of every packet sent is thus the same. In the case of where blocks of data are sent, the destination node to receive an entire block of data is chosen with a uniform random distribution.

7.3.2 Simulation Results

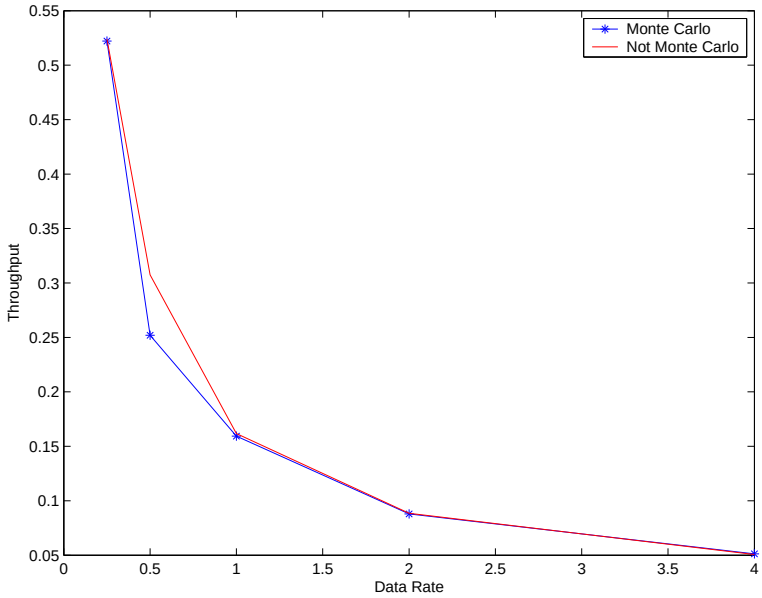
7.3.2.1 Small Network

For the small network configuration, the remaining undefined parameters, as shown in Table 7.8, were utilised. Nodes were distributed in the defined area according to the three topology choices as defined in Section 7.3.1.

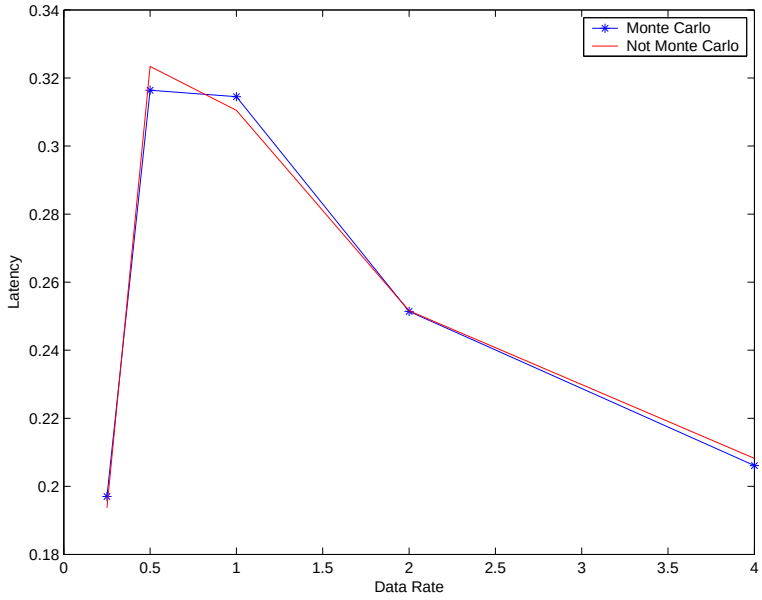
Table 7.8: Small network setup parameters

Simulation time	400 s
Number of nodes	16
Physical network area	450m×450m

For the topology with minimal routes between any two nodes, the nodes were spaced in a circle, as illustrated in Figure 7.20, where there is about 150m of space between consecutive nodes. The physical network area for that particular topology is thus larger than the value as stipulated in Table 7.8

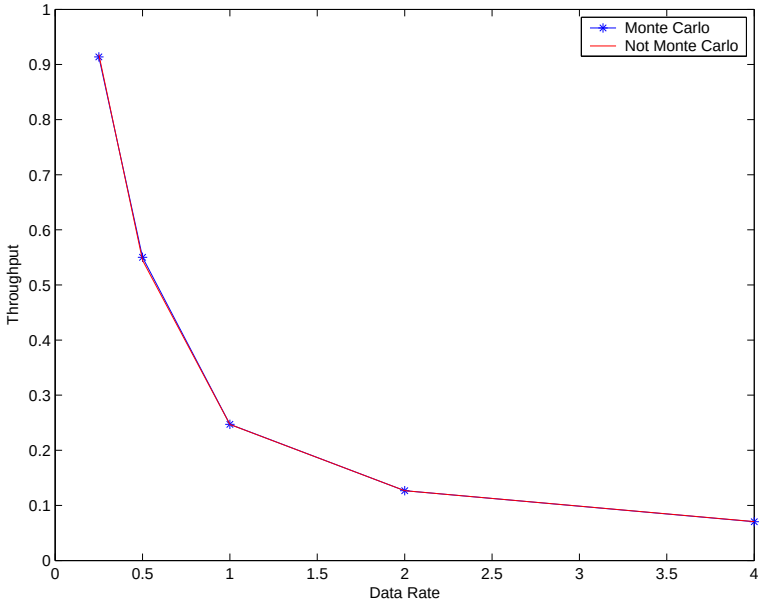


(a) Throughput

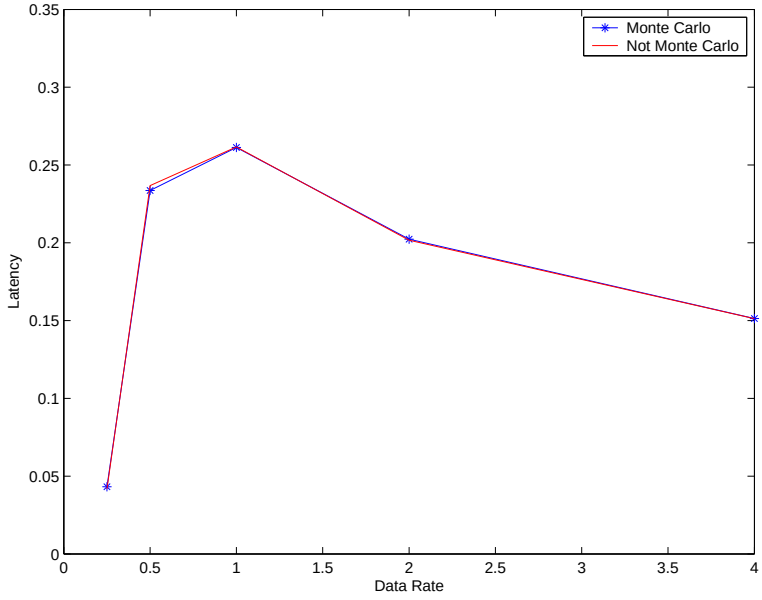


(b) Latency

Figure 7.22: 16 node network performance of poisson distributed data traffic on a uniformly distributed topology

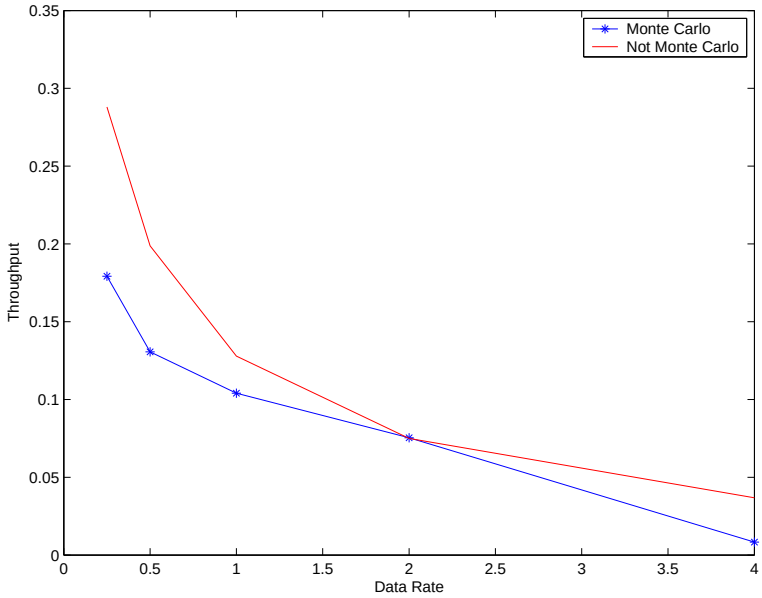


(a) Throughput

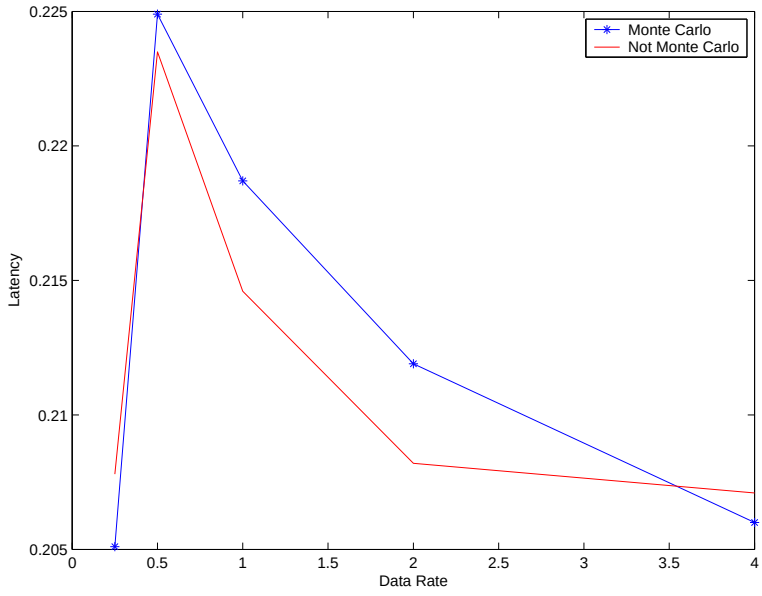


(b) Latency

Figure 7.23: 16 node network performance of poisson distributed data traffic on a topology with minimal multiple paths

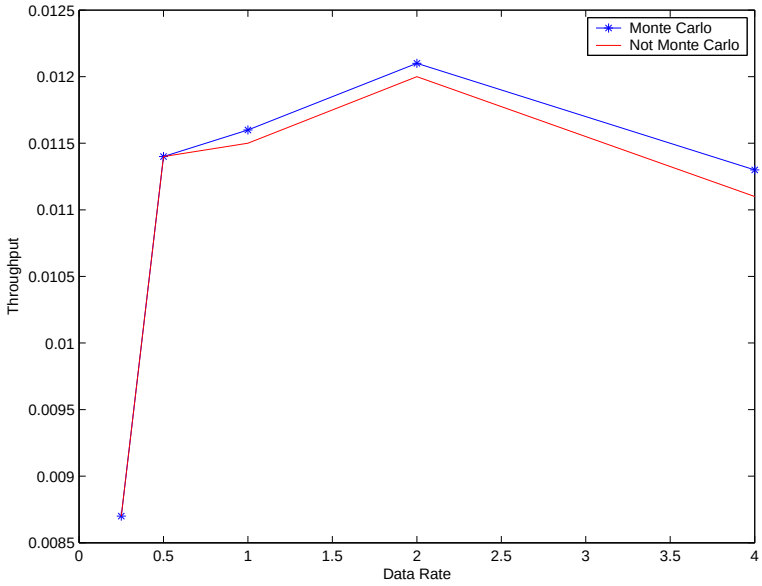


(a) Throughput

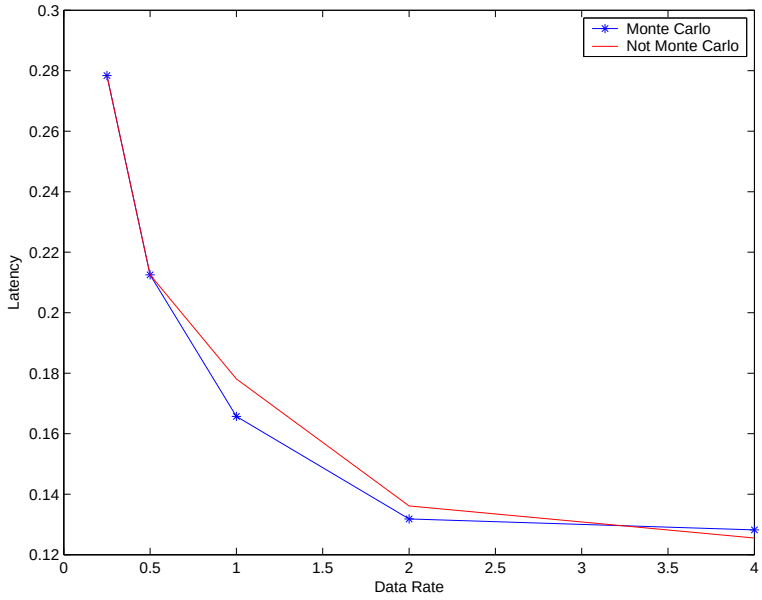


(b) Latency

Figure 7.24: 16 node network performance of poisson distributed data traffic on a randomly distributed topology

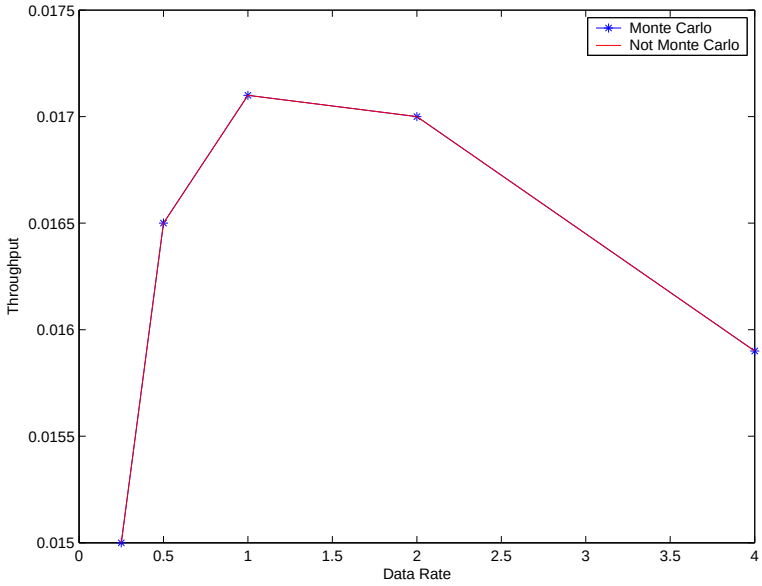


(a) Throughput

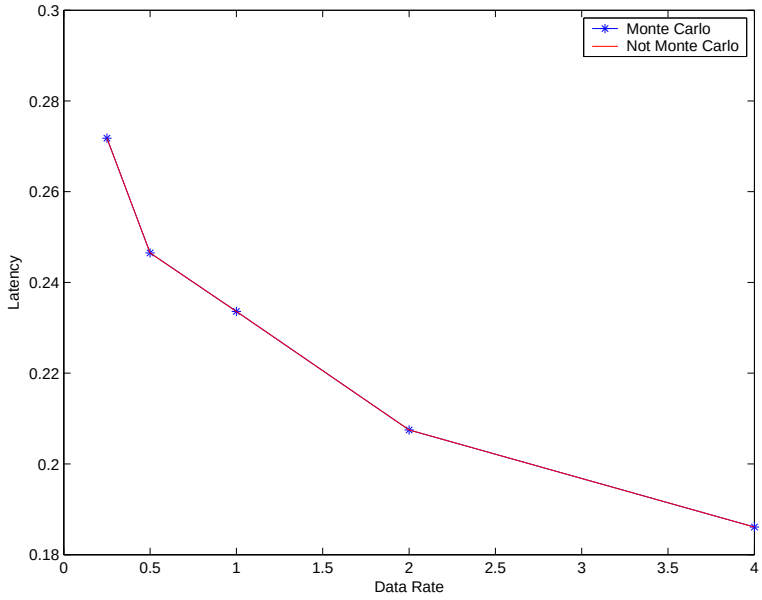


(b) Latency

Figure 7.25: 16 node network performance of blocks of data sent on a uniformly distributed topology

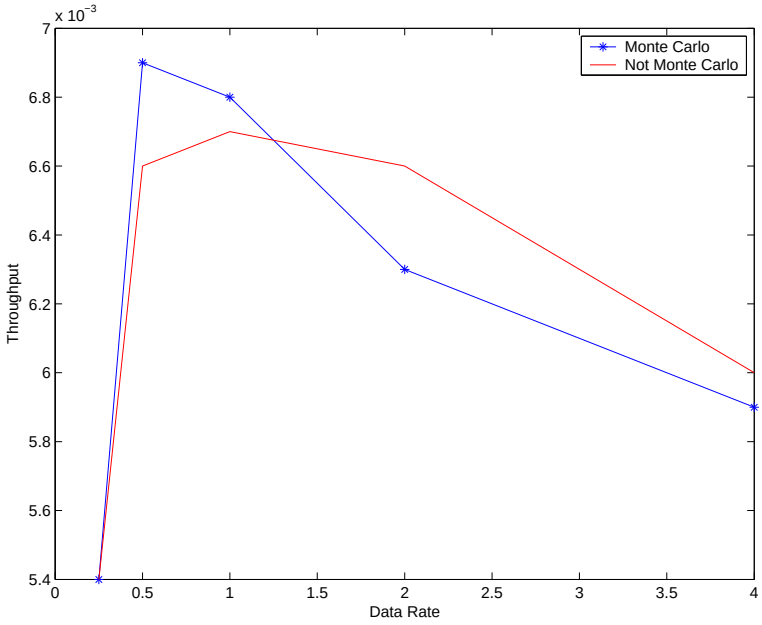


(a) Throughput

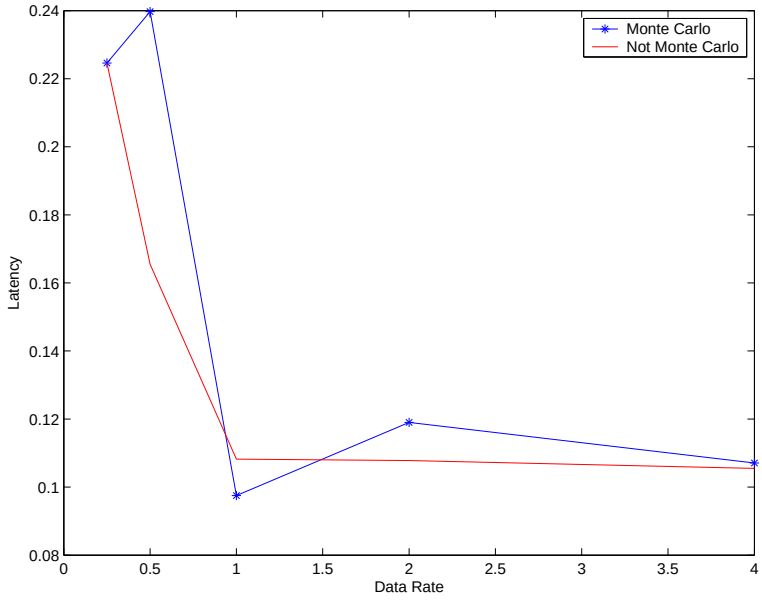


(b) Latency

Figure 7.26: 16 node network performance of blocks of data sent on a topology with minimal multiple paths



(a) Throughput



(b) Latency

Figure 7.27: 16 node network performance of blocks of data sent on a randomly distributed topology

It can be seen in Figures 7.22 to 7.27 that throughput and latency performance in the sixteen node networks are very similar whether the *Monte Carlo* strategy is utilised or not. It is clear how throughput falls as the data rate increase. In the instances where blocks of data were sent each time communication was initiated, the throughput drops significantly. That is because the increased number of consecutive packets congests the surrounding links, which in turn would cause other nodes to find longer routes around the congested area, or they also have to communicate via the congested links, which would all decrease performance.

For the case where a topology with minimal multiple routes was used, the results were

expected to be very similar, since probability of finding multiple routes between destinations, with metrics close in value to each other, is very low. The same routes would thus most probably be chosen every time, hence the performance measures tracking each other closely, as seen in Figures 7.23 and 7.26.

For the random topology, multiple routes exist, but most probably not that many, since the network only consists of sixteen nodes. The proposed solution (Chapter 3) needs multiple routes, with nodes in each route that are not within communication range of each other, to effectively balance the load out across the network. Thus the probability of improvement in performance is even lower in a network with a size as defined in Table 7.8. Multiple routes within an area where the nodes in all or most of the routes are in communication range of each other would not better performance, since the radio spectrum can only *carry* one packet at a time. This is most probably the reason for the similar simulation results obtained from the uniform topology, as seen in Figure 7.24.

It is thus concluded that a larger network, with regard to area, is required in order for the *Monte Carlo* strategy to introduce performance improvement.

7.3.2.2 Large Network

For the large network configuration, the remaining undefined parameters, with values as shown in Table 7.9, were utilised. Nodes were also distributed in the defined area according to the three topology choices as defined in Section 7.3.1.

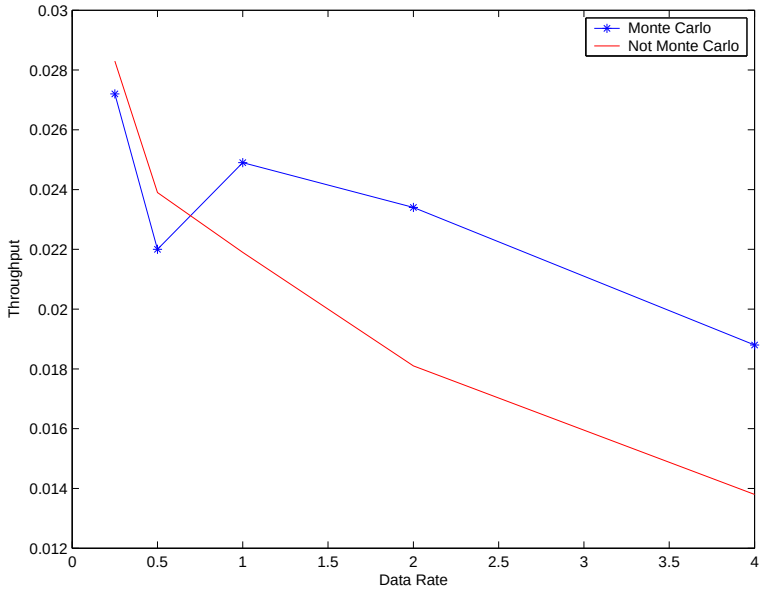
Table 7.9: Large network setup parameters

Simulation time	250 s
Number of nodes	100
Physical network area	800m×800m

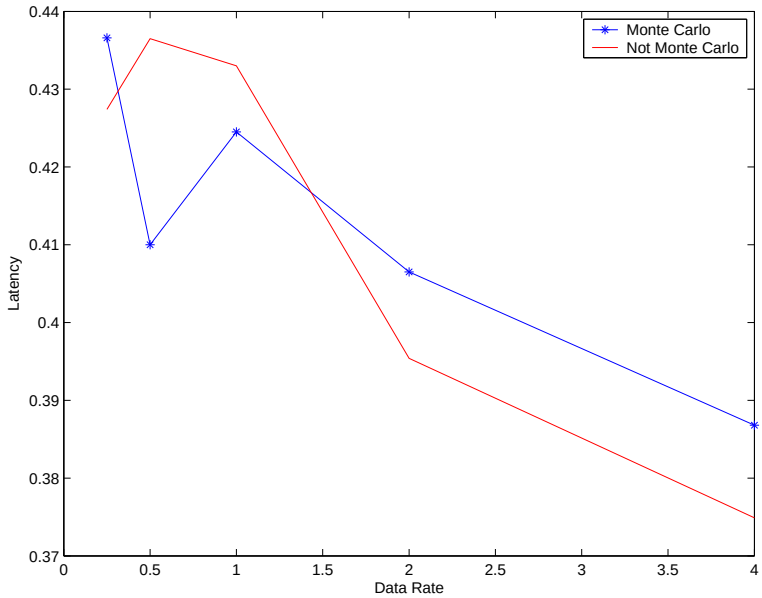
The reason for the shorter simulation time is the increased activity introduced by the large number of nodes. The significantly larger number of events generated by the simulator requires more Central Processing Unit (CPU) cycles. The chosen simulation time, as shown in Table 7.9, was decided upon since available hardware could finish the simulations in a realistic time, while still generating a substantial amount of output data.

It was also decided not to simulate the scenario where minimal multiple routes exists since a larger circle of nodes, each positioned the same distance from each other would not introduce any additional dynamics to the simulation. Even though the nodes are more, there would not be any more optional multipaths available. The test scenario is thus not dependant on the number of nodes, so the simulation with the larger network is not needed.

The first thing that is noticeable from the results, as shown in Figures 7.28 to 7.31, is that the throughput is extremely low throughout the range of data rates. The effect is seen even more dramatically in the instances where blocks of consecutive packets are sent. Packets are dropped by the MAC layer after a certain number of retries. The significant increase in nodes causes an increase in control overhead, which in turn increases congestion, and thus a larger number of packets are dropped by the MAC layer. The extremely low throughput values does, however, seem unrealistically low. It was also

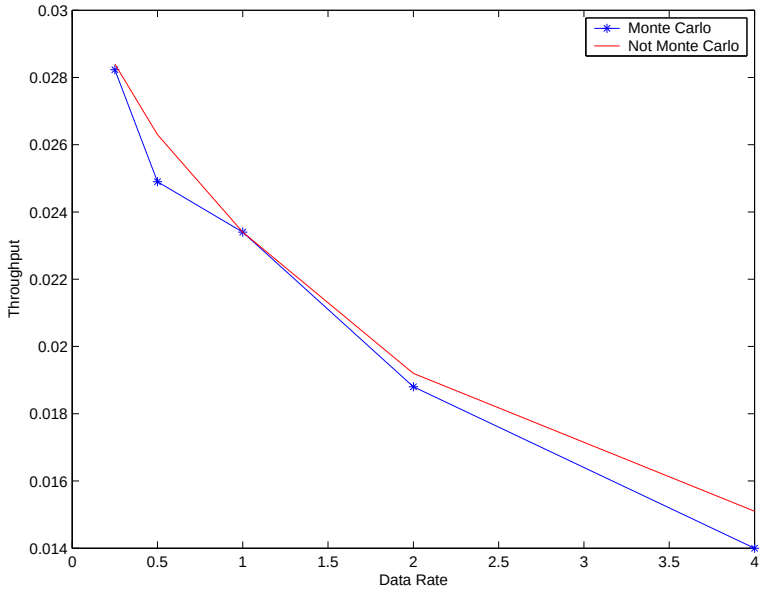


(a) Throughput

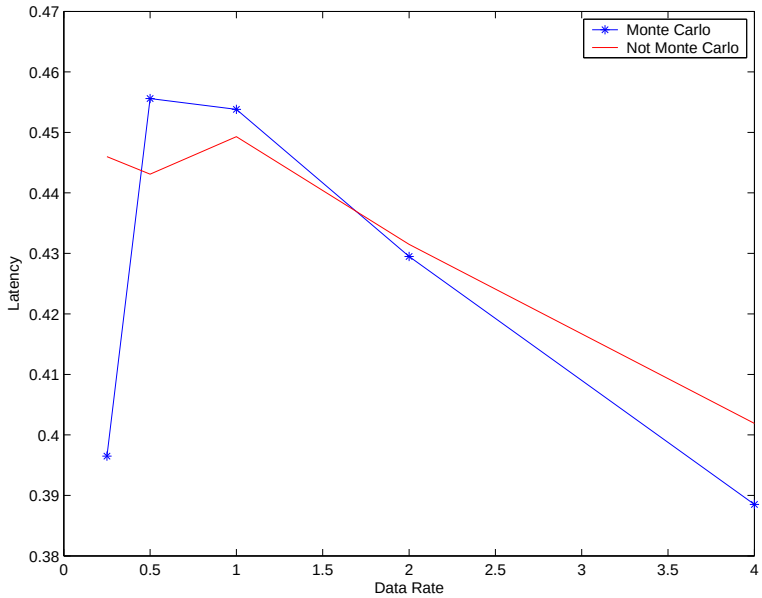


(b) Latency

Figure 7.28: 100 node network performance of Poisson distributed data traffic on a uniformly distributed topology

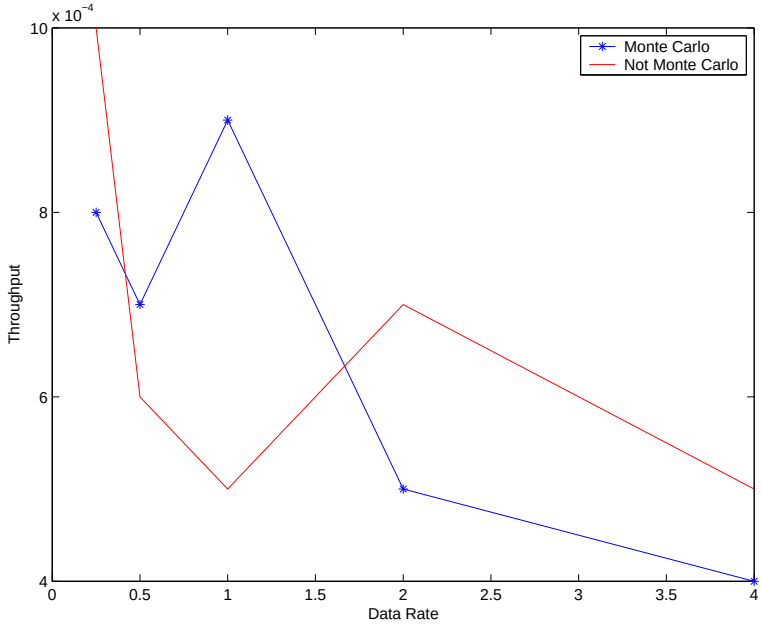


(a) Throughput

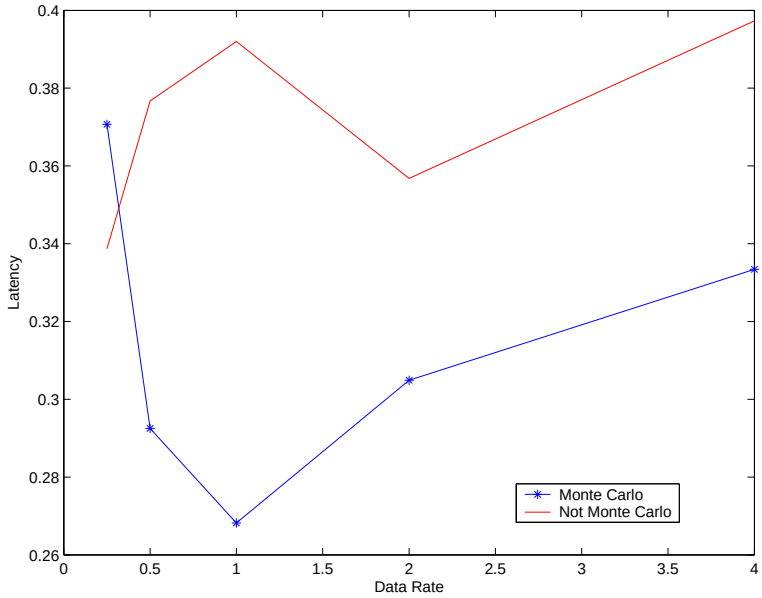


(b) Latency

Figure 7.29: 100 node network performance of Poisson distributed data traffic on a randomly distributed topology

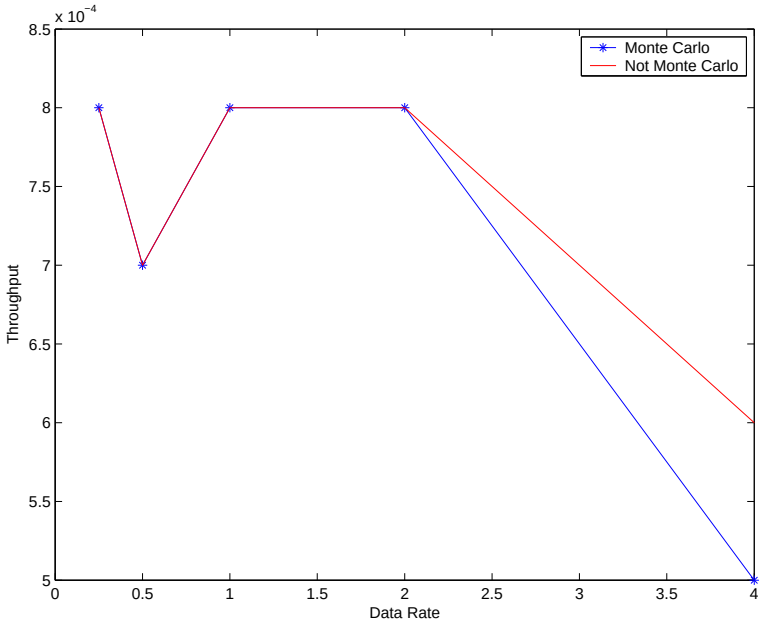


(a) Throughput

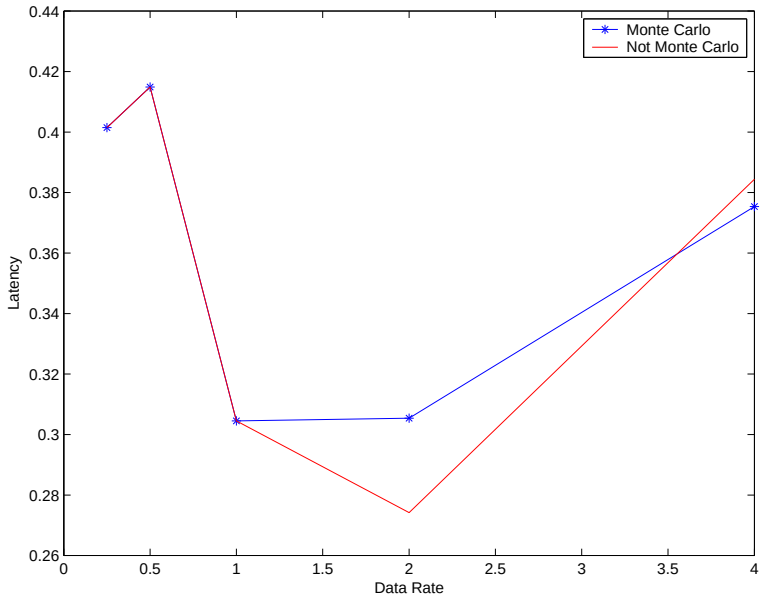


(b) Latency

Figure 7.30: 100 node network performance of blocks of data sent on a uniformly distributed topology



(a) Throughput



(b) Latency

Figure 7.31: 100 node network performance of blocks of data sent on a randomly distributed topology

found that throughput is extremely sensitive for changes in transmit power, as well as data packet size, which introduces extra scepticism as to the exact value of throughput as determined by simulation. Regardless of the precise value the throughput takes on, the simulations are primarily aimed at testing the effect the inclusion of the *Monte Carlo* load balancing strategy has on network performance. The only variable in the respective simulations for every topology and data rate is the *Monte Carlo* strategy being turned on and off, so unbiased comparisons can thus be made.

Performance measures track each other relatively closely for all of the configurations, except for the case where blocks of data were sent on a uniformly distributed topology. Significant improvements for latency, especially when the data rate is higher, can be seen in Figure 7.30(b). The improvement in performance is to be expected in the given configuration, since the uniform distribution over the large area introduces various multiple paths that consist of certain nodes that are not within communication range of one another. The spreading of the load is only possible if multiple routes exist between two nodes where each route contains at least one link that does not fall within communication range of any of the nodes of the other route. The probability of this occurrence is the highest in a uniform, evenly spaced, topology that stretches over a area that is much larger than a single node's communication range.

Multimedia rich networks of today regularly require the transfer of large blocks of data. Large file transfers or multimedia streaming are examples of typical services required from networks. Chunks of data being propagated over large networks, where nodes are relatively uniformly spread out, can expect smaller latencies[2] if the proposed methodology (Chapter 3) is utilised. Another positive that can be taken from the results is that performance does not really suffer when the *Monte Carlo* strategy is applied in any of the test scenarios. It therefore, seems to only offer benefits.

7.4 Hardware

Chapter 6 introduced a hardware platform that was altered to incorporate the *Monte Carlo* load balancing strategy as best practically possible. Figure 7.32 illustrates the eight node network topology as implemented for the purpose of testing. Node 0 is connected to a computer and acts as the base node. It was initially attempted to perform latency measurements with a range of data rates introduced to the network. Initial tests were done where each node attempted to send a single data packet per second to the base node. Increasing the data rate resulted in low throughput. Modules reached a freezed state where a reset was the only option to resume activity. A reset, however, needs to be logged in order to maintain enough information to record realistic latencies (as explained in Section 6.4.1). Reliable communication, while still maintaining accurate synchronisation information, was easily achievable at the initial, single packet per second, data rate. It was thus decided to only perform runs at the low data rate. Table 7.10 shows the average packet latencies obtained, as recorded in a five minute simulations where the *Monte Carlo* load balancing strategy was utilised and where the route with the lowest *cost* was merely selected every time for data transmission.

[2]Compared to the alternative protocol where the *Monte Carlo* strategy is not utilised. Effectively the proposed solution is compared to a version of Dynamic Source Routing (DSR) with the Expected Transmission Count (ETX) metric incorporated

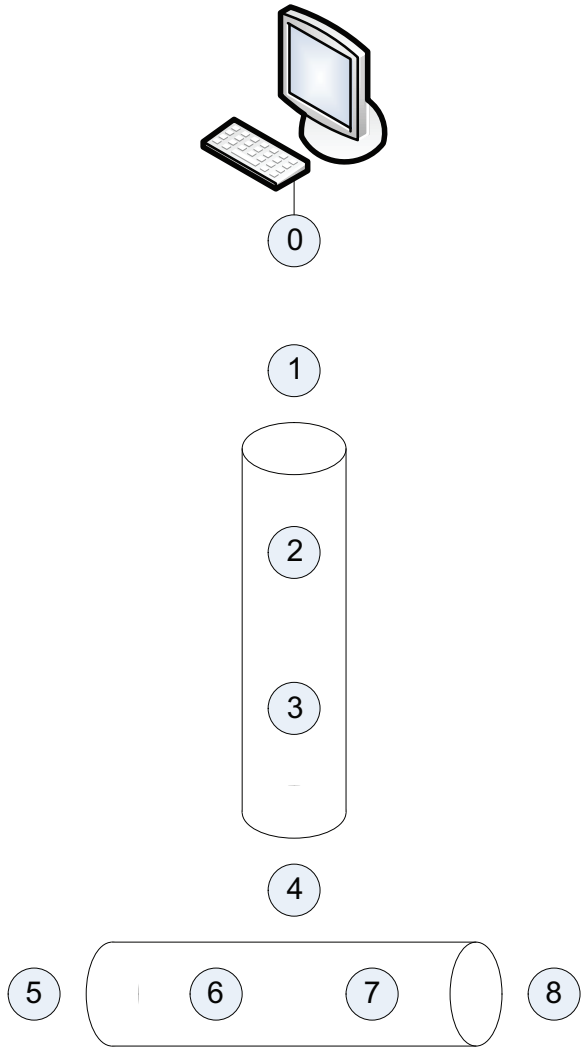


Figure 7.32: Hardware implementation physical network topology

Table 7.10: Latencies recorded with the hardware implementation

Monte Carlo	5.209 seconds
Not Monte Carlo	15.706 seconds

The *Monte Carlo* yielded a significantly lower average latency. However, insufficient information of the the lower level operation of the deployed modules does, however, evoke a degree of scepticism in the obtained results. The extremely large latency values, quite likely due to the medium access strategy, and in particular how communication retries are handled, also introduces doubt.

While there can be little doubt that the hardware implementation proved the relative merits of the *Monte Carlo* approach, it is difficult to quantify such improvement, due to some key unknowns in the hardware platform itself.

7.5 Analytical Model

Chapter 5 introduced an analytical approach to evaluate the performance of the developed routing protocol. The approach hinges on probabilistic behaviour and queueing theory. The aim of developing the analytical model in parallel with simulation, is to strengthen the credibility of the generated results and to develop an analytical tool. In order to create comparable results, the simulation setup was done in a fashion to mimic the assumptions made in the analytical model.

The generated results can be seen in Figure 7.33, where the latency is measured in seconds, and the data rate is the offered load normalised to the maximum transfer rate a single link can handle. Only loads under unity were considered, since as the load approaches unity,

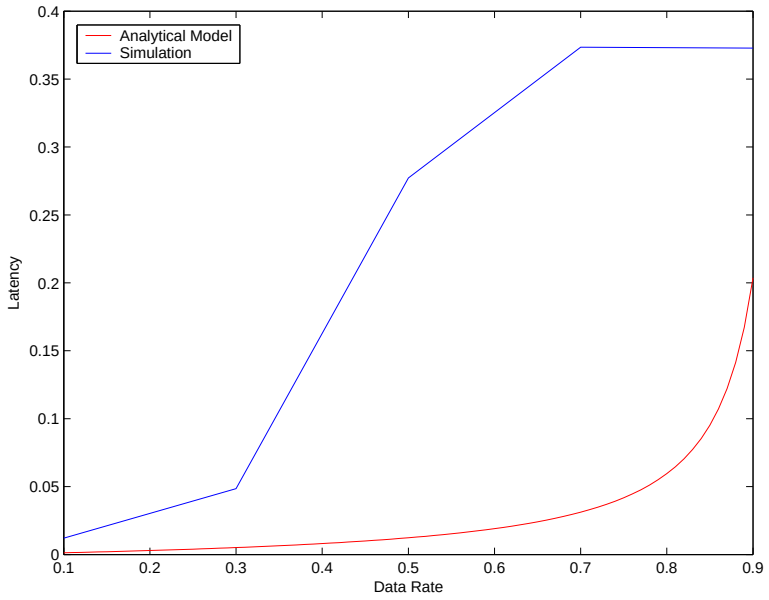


Figure 7.33: Latency as generated by analytical model and simulation compared

the queue length, as used in the analytical model, would strive towards infinity, and thus produce unrealistic results. Results do not coincide precisely, but are in the same order of magnitude, and the curves show similar characteristics. The latency as simulated is expected to be significantly higher since the effect of retransmissions due to bad links is incorporated while the analytical model ignores this effect. The limitations of the MAC layer thus plays a definite role in the quick increase in latency for the simulated results.

7.6 Summary

This chapter began by critically examining the testing process of an ad hoc routing protocol. Influential factors in the process were identified. Thereafter the implementation of a parameter optimisation process, utilising GAs, is fully documented. The results of various network simulations are shown and discussed. Results from a high level hardware implementation are documented, and lastly, the output from an analytical model is compared to simulation results of a similar setup.

Chapter 8

Conclusion

In the dynamic, fast paced and information driven world of today, network connectivity is a growing need. The proverbial *global village* is decreasing in size at a tremendous rate. Infrastructure is, however, needed to connect people from all the corners of the world. At the peripheries of the Internet, where the necessary infrastructure is not available, an alternative self organising, dynamically growing, self-configurable network could be an ideal and affordable option. An efficient ad hoc network solution is thus a technology with significant potential and merit. This thesis investigated a novel *Monte Carlo* based ad hoc routing strategy in an effort to improve overall network performance.

8.1 Summary of Investigation and Results

An in depth background study into the technologies surrounding ad hoc networks, specifically routing protocols for the platform, was done. The study showed the various different approaches and strategies utilised in ad hoc routing. Typical problems were considered and noted.

A novel routing protocol, similar to the Dynamic Source Routing (DSR) protocol, was proposed. The proposed methodology differs from DSR in two ways. Firstly, the Expected Transmission Count (ETX) metric was used to compare links and routes to one another. In most of the ad hoc routing protocols investigated, the route consisting of the lowest number of hops between a source and destination node was selected for transmission. The ETX metric is an estimation of the number of transmissions required to send a packet between a source and destination node. It thus incorporates the effect of a higher bit rate error due to the physical topological nature of a network, as well as the effect of data congestion. The second differential aspect of the proposed methodology is the incorporation of the so-called *Monte Carlo* load balancing strategy. When communication awaits, possible multiple routes between the source and destination nodes are discovered. Since the ETX theoretically generates a dynamic view of the *status quo* of available routes in the network at a given time, the discovered routes are weighted with their respective ETX metric values. The weights determine the likelihood of the given route to be utilised for transmission of data. An initial high level *Java* simulation of the proposed strategy generated positive results.

In order to comprehensively evaluate the merit of the proposed methodology, it was

decided to approach the problem by means of simulation, an analytical model and a hardware implementation. The idea is to increase confidence in the proposed methodology with every different approach.

The routing protocol was fully developed in the simulation environment, *OMNeT++*. Before performance simulations commenced, certain parameters, which are influential in the operation of the protocol, were sent through a series of simulations, in an attempt to optimise them. The optimisation process also aimed to resolve uncertainty regarding the interdependencies between the identified paramters. Genetic Algorithms (GAs) were used in the optimisation process. The process was completed once for latency optimisation, and once for throughput optimisation. Various topologies and scenarios were simulated. For each scenario, simulation runs through a range of data rates were done where the *Monte Carlo* load balancing strategy was incorporated into the routing strategy, and where it was not. For a small network (16 nodes), throughput and latency correlated very closely whether *Monte Carlo* was used or not in all of the test scenarios. Promising latency results where found at extremely high data rates (where blocks of consecutive packets were transmitted every time) for a large network (100 nodes) where all nodes were evenly spaced. The size of the larger network ensured that numerous routes existed independently of one another[1]. The uniform physical layout of the nodes also helped ensuring that multiple paths consistently existed between any two of the nodes. The blocks of consecutive data transmitted caused congestion of certain links, where the *Monte Carlo* strategy then helped spread the load to other routes. The scenario which generated the most favourable results thus fits the profile perfectly of the scenario which the proposed methodology was designed for. The throughput, however, was extremely low in the simulations of the large network. In the case where the blocks of data were sent, the throughput suffered even more. The low throughput brought scepticism into the workings of the IEEE802.11 MAC layer that was used in the simulations. The simulation comparisons were, however, unbiased since the variable in the two simulations compared was the *Monte Carlo* strategy being turned on and off.

An analytical model, based on probability and queueing theory was developed for a typical network configuration. A simulation run, set up to coincide with assumptions made in the analytical model, was executed. The analytical model was specifically designed to determine latency. Latencies, as calculated by the analytical model, fell within the same order of magnitude of those as simulated. The basic tendency of the results across a range of input data rates correlated well. The simulation run delivered slightly higher latencies, which was expected since the simulations incorporated lower layer protocols, which detected physical packet collisions, while the analytical model disregarded it. It is concluded that the analytical model strengthens confidence in the latency results obtained from simulation.

For the hardware implementation, the *Monte Carlo* load balancing strategy was merely incorporated into the routing strategy utilised by a wireless sensor network module called *Tmote Sky*. The operating system, *TinyOS*, runs on the modules. The routing protocol utilised in the modules, *MultihopLQI*, does not work in exactly the same manner as the proposed methodology, but there is a decision that has to be made between multiple routes according to their metrics. The *Monte Carlo* strategy was thus incorporated where this decision had to be made. Numerous test runs proved that the underlying hardware

[1]Data can be transmitted on both routes simultaneously because the nodes forming the respective routes are physically out of communication range form one another.

and/or communication protocols were not developed to operate at high data rates. Low data rate tests were thus run. Another hurdle was the placement of the modules in order to ensure that multiple routes existed independently of one another. Tin foil cylinders where used to provide shielding between certain modules. Careful placement in and around the cylinders ensured that multiple independent routes existed. The test runs showed a significant improvement in latency when the *Monte Carlo* strategy was utilised. However, the order of magnitude of the values introduced scepticism. The result is thus seen as positive, but not conclusive, since the operation of the underlying communication protocols proved to be slightly volatile and aimed at very low data range communication.

8.2 Contributions

In reaching the objective to develop a *Monte Carlo* based ad hoc routing protocol for connectivity improvement, the following contributions were made:

- Development of a novel ad hoc routing protocol, loosely based on DSR
- Incorporation of the ETX metric into the developed protocol
- Incorporation of the *Monte Carlo* load balancing strategy into the developed protocol
- Analysis of the proposed methodology by means of simulation
- Optimisation of critical identified protocol parameters by means of a Genetic Algorithm (GA)
- Verification of the viability of the proposed protocol simulation results by means of an analytical model
- Partial hardware implementation of the proposed routing protocol inclusive of
  - Synchronisation of individual wireless modules in order to accurately observe packet latency
  - Electromagnetic shielding of wireless modules in order to establish multiple independent communication paths

8.3 Possible Future Work

Through the extensive simulation runs, statistical analyses and hardware implementation, it is concluded that the proposed methodology is promising. Where there lies significant scope to dramatically improve the proposed methodology, is in incorporating the *Monte Carlo* load balancing strategy into the MAC layer. Instead of retries implying that data packets be sent on the same links over and over again, the load balancing strategy needs to be incorporated here as well. In the simulations the weighing of the routes only happened when a packet was scheduled, not in the case of retries.

Significant confidence in the methodology could be obtained with a complete hardware implementation, utilising similar lower level protocols as simulated, and incorporating the designed routing protocol precisely. Extensive testing and comparisons to established ad hoc routing protocols running on the same platform would provide definitive insight into the feasibility of the proposed methodology.

8.4 Summary

This chapter summarised the contributions made in the thesis, as well as proposed areas in which the research could possibly be extended in the future.

Appendices

Appendix A

Monte Carlo Strategy Investigation

Java code developed in order to test the viability of the Monte Carlo load balancing strategy is listed.

A.1 Main.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package montecarlo;

/**
 *
 * @author prperold
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Network network = new Network();
        network.run(50);
    }

}
```

A.2 Network.java

```
/*
 * To change this template, choose Tools | Templates
```

```
 * and open the template in the editor.
 */

package montecarlo;

import java.io.*;
import java.util.Random;

/**
 *
 * @author prperold1
 */
public class Network {

    private  int metric[];
    private int route[];
    private int count[];
    private Random rand;
    private int trafficLoad[][];

    public Network(){
        doSetup();
    }

    public void doSetup(){
        metric = new int[7];
        route = new int[3];
        count = new int[7];

        rand  = new Random();

        for(int i = 0 ; i < 7 ; i++){
            metric[i] = rand.nextInt(10)+1;
            //System.out.println(metric[i]);
        }
    }

    public void run(int num){

    int routeChoice;
    int traffic;
    trafficLoad = new int[3][num];

        for(int i =0 ; i < num ; i++){
            route[0] = metric[0]+metric[2]+metric[5];
            route[1] = metric[1]+metric[3]+metric[5];
            route[2] = metric[1]+metric[4]+metric[6];
```

```
            if((route[0] <= route[1]) && (route[0] <= route[2]))
               {
                routeChoice = 0;
            } else if(route[1] <= route[2]){
                routeChoice = 1;
            } else{
                routeChoice = 2;
            }

            traffic =  rand.nextInt(10)+1;

            for(int j = i ;( j<=(i+traffic)) && (j<num) ; j++){
                System.out.println("DEBUG: j = "+j+"  num = "+
                   num);
                trafficLoad[routeChoice][j]++;
            }

            //update routing metrics
            if(trafficLoad[0][i] >= 3){
                metric[0]++;
                metric[2]++;
            }
             if(trafficLoad[1][i] >= 3){
                metric[3]++;
            }
             if(trafficLoad[2][i] >= 3){
                metric[4]++;
                metric[6]++;
            }
             if((trafficLoad[0][i]+trafficLoad[1][i]) >= 3){
                metric[5]++;
            }
             if((trafficLoad[1][i]+trafficLoad[2][i]) >= 3){
                metric[1]++;
            }
        }
        printStats(num, trafficLoad);
    }

    public void printStats(int num, int load[][]){
        System.out.println("Route 0    Route 1    Route 2");
        for(int i =0 ; i <num ; i++){
            System.out.println(load[0][i]+"\t\t"+load[1][i]+"\t
                \t"+load[2][i]);
        }

        File fileOut = new File("/home/prperold/", "out.txt");
         try {
```

```
            BufferedWriter out = new BufferedWriter(new
                FileWriter(fileOut));
            for(int i =0 ; i <num ; i++){
                out.write(load[0][i]+"\t"+load[1][i]+"\t"+load
                    [2][i]+"\n");
            }
            out.close();
        } catch (IOException e) {
        }

    }


}
```

Appendix B

Parameter Optimisation

B.1 Output

The full input and output set of the optimisation Genetic Algorithms (GAs) for throughput and latency are shown in this chapter. For each generation of the respectable GAs, the input parameter set and output is shown. The output is given for each of the three data rates (*Max* is the maximum data rate achievable across a single link in the network) used in the simulations, as well as their average, which is used as the fitness parameter in the GA. Certain cells in the output tables are blank, which means that the simulation did not generate sufficent data in the alotted simulation time.

Table B.1: Throughput parameter optimisation input values - Generation 1

Run	considerFact	freshFactor	helloInterval	routeNum
1	1	13	2	3
2	3	16	2	3
3	38	16	5	2
4	20	28	1	1
5	18	16	5	2
6	49	20	5	2
7	38	3	2	1
8	19	11	2	1
9	44	24	4	3
10	42	14	2	1
11	45	0	2	1
12	30	25	4	3
13	5	6	4	3
14	2	30	4	2
15	20	3	2	3
16	32	21	1	1
17	23	4	4	3
18	3	28	2	3
19	41	3	1	2
20	7	2	3	1

Table B.2: Throughput parameter optimisation output values - Generation 1

Run	Max/4	Max/2	Max	Average
1	0.66		0.18	
2	0.69		0.16	
3	0.62	0.37	0.17	0.39
4	0.73	0.38	0.16	0.42
5	0.62	0.36	0.18	0.39
6	0.67	0.35	0.16	0.39
7	0.51	0.27	0.14	0.31
8	0.68	0.33	0.16	0.39
9	0.71	0.37	0.16	0.42
10	0.7	0.34	0.16	0.4
11	0		0	
12	0.75	0.37	0.17	0.43
13	0.58	0.3	0.16	0.34
14	0.73	0.4	0.17	0.43
15	0.45	0.29		
16	0.73	0.35	0.17	0.41
17	0.52	0.28		
18	0.71	0.39	0.17	0.42
19	0.45	0.26	0.14	0.29
20	0.43	0.26	0.14	0.28

Table B.3: Throughput parameter optimisation input values - Generation 2

Run	considerFact	freshFactor	helloInterval	routeNum
1	22	26	1	2
2	16	22	4	2
3	20	25	5	2
4	15	23	5	2
5	10	26	3	2
6	9	26	3	2
7	7	30	4	2
8	20	29	3	2
9	35	27	3	2
10	21	30	2	2
11	13	24	4	2
12	14	26	2	2
13	16	26	4	2
14	42	24	4	2
15	28	27	4	2
16	5	28	4	2
17	14	25	4	2
18	2	26	2	2
19	15	27	4	2
20	28	25	4	2

Table B.4: Throughput parameter optimisation output values - Generation 2

Run	Max/4	Max/2	Max	Average
1	0.73		0.18	
2	0.68	0.37		
3	0.64	0.37	0.17	0.4
4	0.69	0.36	0.16	0.4
5	0.69	0.4	0.19	0.43
6	0.74	0.39	0.17	0.44
7	0.76	0.38	0.16	0.43
8	0.76	0.39	0.17	0.44
9	0.71		0.16	
10	0.77	0.38	0.17	0.44
11	0.74	0.37	0.16	0.42
12	0.74		0.16	
13	0.7	0.39	0.17	0.42
14	0.72	0.4	0.17	0.43
15	0.7	0.39		
16	0.75	0.38		
17	0.74	0.38	0.17	0.43
18	0.7	0.38	0.17	0.42
19	0.7	0.4	0.17	0.42
20	0.74	0.38	0.16	0.43

Table B.5: Throughput parameter optimisation input values - Generation 3

Run	considerFact	freshFactor	helloInterval	routeNum
1	18	26	2	2
2	26	29	2	2
3	16	29	2	2
4	24	29	2	2
5	14	30	2	2
6	13	28	2	2
7	14	28	2	2
8	17	29	2	2
9	11	25	2	2
10	21	29	2	2
11	10	29	2	2
12	16	29	2	2
13	14	30	2	2
14	16	29	2	2
15	16	28	2	2
16	11	29	2	2
17	7	26	2	2
18	9	29	2	2
19	10	27	2	2
20	16	28	2	2

Table B.6: Throughput parameter optimisation output values - Generation 3

Run	Max/4	Max/2	Max	Average
1	0.73		0.18	
2	0.74	0.39	0.17	0.43
3	0.69	0.39	0.18	0.42
4	0.74	0.36	0.17	0.42
5	0.71	0.39	0.18	0.43
6	0.75	0.4	0.17	0.44
7	0.77	0.37	0.17	0.43
8	0.76	0.39	0.18	0.44
9	0.72	0.38	0.17	0.42
10	0.76	0.38	0.17	0.43
11	0.76		0.17	
12	0.76	0.37	0.17	0.43
13	0.72	0.4	0.18	0.43
14	0.73	0.4	0.17	0.43
15	0.69	0.4		
16	0.76	0.39	0.17	0.44
17	0.73	0.36	0.17	0.42
18	0.72	0.39	0.17	0.43
19	0.71	0.38	0.16	0.42
20	0.75	0.39	0.17	0.44

Table B.7: Throughput parameter optimisation input values - Generation 4

Run	considerFact	freshFactor	helloInterval	routeNum
1	18	28	2	2
2	19	28	2	2
3	16	28	2	2
4	15	28	2	2
5	15	28	2	2
6	15	28	2	2
7	10	28	2	2
8	18	28	2	2
9	13	28	2	2
10	12	28	2	2
11	13	28	2	2
12	14	28	2	2
13	13	28	2	2
14	17	28	2	2
15	14	28	2	2
16	13	28	2	2
17	10	28	2	2
18	14	28	2	2
19	12	28	2	2
20	14	28	2	2

Table B.8: Throughput parameter optimisation output values - Generation 4

Run	Max/4	Max/2	Max	Average
1	0.73		0.18	
2	0.72		0.17	
3	0.69	0.39	0.17	0.42
4	0.76	0.37	0.17	0.43
5	0.71	0.39	0.18	0.43
6	0.74	0.4	0.17	0.43
7	0.76	0.38		
8	0.75		0.18	
9	0.73	0.37	0.17	0.42
10	0.76	0.38	0.17	0.44
11	0.75		0.17	
12	0.76	0.38	0.17	0.44
13	0.71	0.39	0.17	0.43
14	0.74	0.4	0.17	0.44
15	0.69	0.4		
16	0.75	0.4	0.17	0.44
17	0.75	0.37	0.17	0.43
18	0.71	0.39	0.17	0.42
19	0.71	0.38	0.17	0.42
20	0.75	0.37	0.17	0.43

Table B.9: Throughput parameter optimisation input values - Generation 5

Run	considerFact	freshFactor	helloInterval	routeNum
1	17	28	2	2
2	11	28	2	2
3	14	28	2	2
4	12	28	2	2
5	14	28	2	2
6	14	28	2	2
7	16	28	2	2
8	11	28	2	2
9	12	28	2	2
10	14	28	2	2
11	12	28	2	2
12	15	28	2	2
13	16	28	2	2
14	15	28	2	2
15	15	28	2	2
16	13	28	2	2
17	18	28	2	2
18	17	28	2	2
19	13	28	2	2
20	13	28	2	2

Table B.10: Throughput parameter optimisation output values - Generation 5

Run	Max/4	Max/2	Max	Average
1	0.73		0.18	
2	0.72	0.39	0.17	0.43
3	0.69	0.39	0.17	0.42
4	0.73	0.37	0.17	0.42
5	0.7	0.39	0.18	0.43
6	0.74	0.4	0.17	0.44
7	0.76	0.37	0.17	0.43
8	0.75	0.38	0.18	0.44
9	0.72	0.38	0.17	0.42
10	0.75	0.38	0.17	0.43
11	0.77		0.17	
12	0.76	0.38	0.17	0.43
13	0.73	0.4	0.17	0.43
14	0.73	0.4	0.17	0.43
15	0.7	0.4		
16	0.75	0.4	0.17	0.44
17	0.75	0.37	0.17	0.43
18	0.7	0.39	0.17	0.42
19	0.38	0.17		
20	0.74	0.37	0.17	0.43

Table B.11: Latency parameter optimisation input values - Generation 1

Run	considerFact	freshFactor	helloInterval	routeNum
1	1	13	2	3
2	3	16	2	3
3	38	16	5	2
4	20	28	1	1
5	18	16	5	2
6	49	20	5	2
7	38	3	2	1
8	19	11	2	1
9	44	24	4	3
10	42	14	2	1
11	45	0	2	1
12	30	25	4	3
13	5	6	4	3
14	2	30	4	2
15	20	3	2	3
16	32	21	1	1
17	23	4	4	3
18	3	28	2	3
19	41	3	1	2
20	7	2	3	1

Table B.12: Latency parameter optimisation output values - Generation 1

Run	Max/4	Max/2	Max	Average
1	0.1	0.36	0.36	0.27
2	0.08	0.32	0.38	0.26
3	0.08	0.31	0.35	0.25
4	0.06	0.33	0.38	0.26
5	0.09	0.35	0.35	0.26
6	0.08	0.29	0.34	0.24
7	0.19	0.32	0.3	0.27
8	0.11	0.34	0.36	0.27
9	0.07	0.34	0.36	0.26
10	0.09	0.36	0.37	0.27
11	0.06		0.32	
12	0.06	0.33	0.38	0.26
13	0.13	0.29	0.31	0.24
14	0.06	0.33	0.37	0.25
15	0.2	0.32	0.31	0.28
16	0.07	0.32	0.41	0.27
17	0.17	0.29	0.3	0.25
18	0.07	0.36	0.38	0.27
19	0.19	0.33	0.33	0.28
20	0.2	0.28	0.26	0.25

Table B.13: Latency parameter optimisation input values - Generation 2

Run	considerFact	freshFactor	helloInterval	routeNum
1	37	8	4	2
2	47	8	4	2
3	1	3	4	2
4	31	11	4	2
5	32	13	4	2
6	11	14	4	2
7	24	1	4	2
8	50	13	4	2
9	23	16	4	2
10	50	10	4	2
11	19	8	4	2
12	9	16	4	2
13	31	24	4	2
14	17	13	4	2
15	18	13	4	2
16	13	10	4	2
17	41	20	4	2
18	50	3	4	2
19	17	10	4	2
20	42	10	4	2

Table B.14: Latency parameter optimisation output values - Generation 2

Run	Max/4	Max/2	Max	Average
1	0.13	0.3	0.33	0.25
2	0.12	0.3	0.34	0.25
3	0.17	0.29	0.28	0.25
4	0.1	0.28	0.34	0.24
5	0.09	0.34	0.34	0.26
6	0.09	0.29	0.34	0.24
7	0.19	0.26	0.25	0.23
8	0.1	0.31	0.35	0.25
9	0.09	0.33	0.34	0.25
10	0.1	0.32	0.34	0.25
11	0.12	0.31	0.29	0.24
12	0.08	0.32	0.38	0.26
13	0.06	0.31	0.36	0.24
14	0.11	0.32	0.34	0.25
15	0.1	0.33	0.36	0.26
16	0.11	0.29	0.35	0.25
17	0.07	0.33	0.36	0.26
18	0.16	0.29	0.31	0.25
19	0.11	0.33	0.36	0.27
20	0.11	0.31	0.3	0.24

Table B.15: Latency parameter optimisation input values - Generation 3

Run	considerFact	freshFactor	helloInterval	routeNum
1	32	8	4	2
2	25	11	4	2
3	32	11	4	2
4	44	4	4	2
5	31	11	4	2
6	35	14	4	2
7	27	9	4	2
8	17	6	4	2
9	14	6	4	2
10	12	5	4	2
11	33	8	4	2
12	16	12	4	2
13	19	6	4	2
14	19	9	4	2
15	24	8	4	2
16	31	12	4	2
17	37	12	4	2
18	21	11	4	2
19	19	16	4	2
20	36	4	4	2

Table B.16: Latency parameter optimisation output values - Generation 3

Run	Max/4	Max/2	Max	Average
1	0.12	0.3	0.33	0.25
2	0.12	0.31	0.35	0.26
3	0.09	0.32	0.33	0.25
4	0.16	0.26	0.3	0.24
5	0.11	0.34	0.34	0.26
6	0.09	0.29	0.34	0.24
7	0.11	0.33	0.32	0.25
8	0.15	0.3	0.32	0.26
9	0.14	0.29	0.3	0.25
10	0.15	0.31	0.31	0.25
11	0.12	0.3	0.3	0.24
12	0.09	0.31	0.36	0.26
13	0.12	0.28	0.3	0.24
14	0.12	0.31	0.31	0.25
15	0.12	0.31	0.34	0.26
16	0.09	0.31	0.37	0.26
17	0.1	0.32	0.35	0.26
18	0.1	0.32	0.34	0.25
19	0.09	0.34	0.37	0.27
20	0.16	0.28	0.28	0.24

Table B.17: Latency parameter optimisation input values - Generation 4

Run	considerFact	freshFactor	helloInterval	routeNum
1	11	11	4	2
2	20	6	4	2
3	35	8	4	2
4	49	7	4	2
5	50	8	4	2
6	35	1	4	2
7	40	15	4	2
8	31	5	4	2
9	42	7	4	2
10	28	13	4	2
11	45	4	4	2
12	25	2	4	2
13	28	6	4	2
14	39	1	4	2
15	14	10	4	2
16	42	12	4	2
17	34	1	4	2
18	21	8	4	2
19	28	7	4	2
20	19	14	4	2

Table B.18: Latency parameter optimisation output values - Generation 4

Run	Max/4	Max/2	Max	Average
1	0.1	0.32	0.34	0.25
2	0.13	0.29	0.33	0.25
3	0.11	0.32	0.31	0.25
4	0.13	0.27	0.32	0.24
5	0.12	0.33	0.33	0.26
6	0.17	0.24	0.25	0.22
7	0.09	0.34	0.34	0.26
8	0.16	0.29	0.31	0.25
9	0.14	0.3	0.31	0.25
10	0.09	0.34	0.35	0.26
11	0.17	0.29	0.27	0.24
12	0.18	0.27	0.29	0.25
13	0.13	0.29	0.31	0.24
14	0.2	0.26	0.23	0.23
15	0.1	0.32	0.34	0.25
16	0.09	0.31	0.37	0.26
17	0.21	0.26	0.25	0.24
18	0.11	0.31	0.34	0.25
19	0.14	0.32	0.34	0.26
20	0.09	0.31	0.3	0.23

Table B.19: Latency parameter optimisation input values - Generation 5

Run	considerFact	freshFactor	helloInterval	routeNum
1	17	3	4	2
2	35	4	4	2
3	15	1	4	2
4	34	1	4	2
5	38	1	4	2
6	45	2	4	2
7	43	6	4	2
8	22	6	4	2
9	25	3	4	2
10	47	1	4	2
11	20	3	4	2
12	34	8	4	2
13	26	1	4	2
14	35	7	4	2
15	27	3	4	2
16	35	7	4	2
17	26	3	4	2
18	20	1	4	2
19	24	7	4	2
20	35	4	4	2

Table B.20: Latency parameter optimisation output values - Generation 5

Run	Max/4	Max/2	Max	Average
1	0.18	0.28	0.3	0.26
2	0.15	0.29	0.32	0.25
3	0.2	0.26	0.24	0.23
4	0.2	0.23	0.26	0.23
5	0.21	0.27	0.27	0.25
6	0.17	0.26	0.27	0.23
7	0.13	0.31	0.31	0.25
8	0.16	0.3	0.32	0.26
9	0.18	0.29	0.27	0.25
10	0.2	0.26	0.24	0.23
11	0.18	0.29	0.26	0.24
12	0.11	0.31	0.34	0.25
13	0.2	0.23	0.24	0.23
14	0.15	0.3	0.31	0.25
15	0.17	0.3	0.3	0.26
16	0.12	0.29	0.34	0.25
17	0.18	0.28	0.29	0.25
18	0.19	0.27	0.27	0.24
19	0.13	0.31	0.33	0.26
20	0.16	0.28	0.28	0.24

Table B.21: Latency parameter optimisation input values - Generation 6

Run	considerFact	freshFactor	helloInterval	routeNum
1	39	1	4	2
2	34	1	4	2
3	39	1	4	2
4	38	1	4	2
5	26	1	4	2
6	29	1	4	2
7	30	1	4	2
8	47	1	4	2
9	21	1	4	2
10	34	1	4	2
11	46	1	4	2
12	27	1	4	2
13	9	1	4	2
14	40	1	4	2
15	25	1	4	2
16	19	1	4	2
17	17	1	4	2
18	34	1	4	2
19	45	1	4	2
20	37	1	4	2

Table B.22: Latency parameter optimisation output values - Generation 6

Run	Max/4	Max/2	Max	Average
1	0.2	0.25	0.26	0.24
2	0.2	0.25	0.27	0.24
3	0.19	0.26	0.25	0.23
4	0.2	0.23	0.26	0.23
5	0.21	0.27	0.27	0.25
6	0.18	0.24	0.24	0.22
7	0.19	0.26	0.25	0.23
8	0.21	0.24	0.25	0.23
9	0.2	0.26	0.24	0.23
10	0.19	0.26	0.25	0.23
11	0.2	0.26	0.23	0.23
12	0.18	0.25	0.26	0.23
13	0.2	0.24	0.24	0.23
14	0.2	0.26	0.23	0.23
15	0.19	0.26	0.26	0.24
16	0.2	0.25	0.27	0.24
17	0.2	0.26	0.25	0.24
18	0.19	0.27	0.27	0.24
19	0.19	0.27	0.26	0.24
20	0.19	0.24	0.23	0.22

B.2 Python Code

B.2.1 inputgenerator.py

```
import string, StringIO, random

def initInput(numRuns, dataPacketSize, dataDistro, blockSize):

        fout = open("runs.ini", 'w')
        fsum = open("inputsummary.txt", "w")
        dataRate = [(5.5*10**6)/4,(5.5*10**6)/2,(5.5*10**6)]
        count = 1
        for i in range(numRuns):
                count2 = 1
                considerFact = str(random.randrange(51))
                freshFactor = str(random.randrange(31))
                helloInterval = str(random.randrange(1,6))
                routes2consider = str(random.randrange(1,4))
                fsum.write(str(i+1)+"\t"+considerFact+"\t"+
                    freshFactor+"\t"+helloInterval+"\t"+
                    routes2consider+"\n")
                for j in range(3):
                        fout.write("[Run "+str(((count-1)*3)+
                            count2)+"]\n")
```

```
                        fout.write("output-vector-file = run"+
                            str(count)+"_"+str(count2)+".vec\n")
                        fout.write("adHocSim.host[*].appl.
                            dataRate = "+str(dataRate[count2-1])+
                            "\n")
                        fout.write("adHocSim.host[*].appl.
                            dataPacketSize = "+str(dataPacketSize
                            )+"\n")
                        fout.write("adHocSim.host[*].appl.
                            dataDistro = "+str(dataDistro)+"\n")
                        fout.write("adHocSim.host[*].appl.
                            blockSize = "+str(blockSize)+"\n")
                        fout.write("adHocSim.host[*].net.
                            considerFact = "+considerFact+"\n")
                        fout.write("adHocSim.host[*].net.
                            freshFactor = "+freshFactor+"\n")
                        fout.write("adHocSim.host[*].net.
                            helloInterval = "+helloInterval+"\n")
                        fout.write("adHocSim.host[*].net.
                            routes2consider = "+routes2consider+"
                            \n\n")
                        count2+=1
                count+=1

        fout.close()
        fsum.close()

if __name__ == "__main__":

        numRuns = 20
        dataPacketSize = 1024*8 # 1kB
        dataDistro = 1 # 1 = poisson, 2 = blocks, 3 = uniform
        blockSize = 750

        initInput(numRuns, dataPacketSize, dataDistro, blockSize
            )
```

B.2.2 dataparser.py

```
import string, StringIO, random, operator, math

def throughputparser(generation):
        fn = "/home/prperold/MScIng/Simulation/BlockTraffic/
            AdHocNetwork2/Output/generation"+str(generation)+"/
            raw_input/omnetpp.sca"
        foutn = "/home/prperold/MScIng/Simulation/BlockTraffic/
            AdHocNetwork2/Output/generation"+str(generation)+"/
            parsed_output/throughput.txt"
```

```
favgn = "/home/prperold/MScIng/Simulation/BlockTraffic/
    AdHocNetwork2/Output/generation"+str(generation)+"/
    parsed_output/throughputavg.txt"
fp = open(fn,"r")
foutp = open(foutn, 'w')
favgp = open(favgn, 'w')
foutp.write("Generation\tMax/4\t\tMax/2\t\tMax\n")
foutp.write("
    ------------------------------------------------------
    ")
data = fp.readlines()
run = 0
oldgeneration = 0
generation = 0
out=[]
for line in data:
        if line.count("run"):
                runnum=""
                run+=1
                totalsent = 0
                totalreceived = 0
                i = 0
                while(line[len("run ")+i].isdigit()):
                        runnum += line[len("run ")+i]
                        i+=1
                oldgeneration = generation
                generation = (int(runnum)-1)/3+1
        if line.count("sent"):
                sent = ""
                sent+=line[line.rfind(" "):-1]
                totalsent+=int(sent)
        if line.count("received"):
                received = ""
                received+=line[line.rfind(" "):-1]
                totalreceived+=int(received)
        if line.count("[15]") and line.count("received")
            :
                if run<int(runnum):
                        foutp.write("\t\t")
                        run = int(runnum)
                if oldgeneration<generation:
                        foutp.write("\n"+str(generation)
                            )
                        if len(out) == 3 and 0 not in
                            out:
                                favgp.write(str(
                                    oldgeneration)+"\t"+
                                    str(round(1.0*(sum(
                                    out)/len(out)),4))+"\
```

```
                                        n")
                                out = []

                            foutp.write("\t\t"+str(round(float(
                                totalreceived)/float(totalsent),4)))
                            out.append(round(float(totalreceived)/
                                float(totalsent),4))
        if len(out) == 3 and 0 not in out:
                favgp.write(str(generation)+"\t"+str(round(1.0*(
                    sum(out)/len(out)),4))+"\n")
        fp.close()
        foutp.close()
        favgp.close()

def newInputGenerator(numRuns,dataPacketSize,dataDistro,
    generation):

        fin = "/home/prperold/MScIng/Simulation/BlockTraffic/
            AdHocNetwork2/Output/generation"+str(generation)+"/
            parsed_output/throughputavg.txt"
        fp = open(fin,"r")
        data = fp.readlines()

        fsum = "/home/prperold/MScIng/Simulation/BlockTraffic/
            AdHocNetwork2/Output/generation"+str(generation)+"/
            parameters/inputsummary.txt"
        fsump = open(fsum,"r")
        sumdata = fsump.readlines()

        out = []
        for line in data:
                outline = line.split()
                out.append((int(outline[0]),round(float(outline
                    [-1]),4)))
        out = sorted(out,key=operator.itemgetter(1))
        out.reverse()
        selection = out[0:len(out)/4] #/4 so that only top 25%
            is selected
        fp.close()

        input = {}
        for line in sumdata:
                outline = line.split()
                input[int(float(outline[0]))] = (int(float(
                    outline[1])),int(float(outline[2])),int(float
                    (outline[3])),int(float(outline[4])))


        considerFact = []
```

```
        freshFactor = []
        helloInterval = []
        routes2consider = []

        for i in selection:
                considerFact.append((input[i[0]])[0])
                freshFactor.append((input[i[0]])[1])
                helloInterval.append((input[i[0]])[2])
                routes2consider.append((input[i[0]])[3])

        createNewInputFile(considerFact,freshFactor,
           helloInterval,routes2consider,numRuns,dataPacketSize,
           dataDistro)

def createNewInputFile(considerFact,freshFactor,helloInterval,
   routes2consider,numRuns,dataPacketSize,dataDistro):
        fout = open("runs.ini", 'w')
        fsum = open("inputsummary.txt", "w")

        dataRate = [(5.5*10**6)/4,(5.5*10**6)/2,(5.5*10**6)]
        count = 1

        for i in range(numRuns):
                count2 = 1
                considerFactStr = round(random.gauss(sum(
                   considerFact)/len(considerFact), getStdDev(
                   considerFact)))
                freshFactorStr = round(random.gauss(sum(
                   freshFactor)/len(freshFactor), getStdDev(
                   freshFactor)))
                helloIntervalStr = round(random.gauss(sum(
                   helloInterval)/len(helloInterval), getStdDev(
                   helloInterval)))
                routes2considerStr = round(random.gauss(sum(
                   routes2consider)/len(routes2consider),
                   getStdDev(routes2consider)))

                if considerFactStr < 1:
                        considerFactStr = 1
                if considerFactStr > 50:
                        considerFactStr = 50
                if freshFactorStr < 1:
                        freshFactorStr = 1
                if freshFactorStr > 30:
                        freshFactorStr = 30
                if helloIntervalStr < 1:
                        helloIntervalStr = 1
                if helloIntervalStr > 10:
                        helloIntervalStr = 10
```

```
                if routes2considerStr < 1:
                        routes2considerStr = 1
                if routes2considerStr > 3:
                        routes2considerStr = 3

                fsum.write(str(i+1)+"\t"+str(considerFactStr)+"\
                    t"+str(freshFactorStr)+"\t"+str(
                    helloIntervalStr)+"\t"+str(routes2considerStr
                    )+"\n")
                for j in range(3):
                        fout.write("[Run "+str(((count-1)*3)+
                            count2)+"]\n")
                        fout.write("output-vector-file = run"+
                            str(count)+"_"+str(count2)+".vec\n")
                        fout.write("adHocSim.host[*].appl.
                            dataRate = "+str(dataRate[count2-1])+
                            "\n")
                        fout.write("adHocSim.host[*].appl.
                            dataPacketSize = "+str(dataPacketSize
                            )+"\n")
                        fout.write("adHocSim.host[*].appl.
                            dataDistro = "+str(dataDistro)+"\n")
                        fout.write("adHocSim.host[*].appl.
                            blockSize = "+str(blockSize)+"\n")
                        fout.write("adHocSim.host[*].net.
                            considerFact = "+str(considerFactStr)
                            +"\n")
                        fout.write("adHocSim.host[*].net.
                            freshFactor = "+str(freshFactorStr)+"
                            \n")
                        fout.write("adHocSim.host[*].net.
                            helloInterval = "+str(
                            helloIntervalStr)+"\n")
                        fout.write("adHocSim.host[*].net.
                            routes2consider = "+str(
                            routes2considerStr)+"\n\n")
                        count2+=1
                count+=1

        fout.close()
        fsum.close()

def getStdDev(input):
        mean = sum(input)/len(input)
        out = 0
        for i in input:
                out+=(i-mean)**2
        std=round(math.sqrt(out/len(input)))
        return std
```

```
if __name__ == "__main__":

        numRuns = 20
        dataPacketSize = 1024*8 # 1kB
        dataDistro = 1 # 1 = poisson, 2 = blocks, 3 = uniform
        blockSize = 750

        generation = 5

        #newInputGenerator(numRuns,dataPacketSize,dataDistro,
            generation)
        throughputparser(generation)
```

List of References

[1] R. Wolhuter, "Basic queueing theory," session 7.

[2] "Tmote sky: Datasheet," Moteiv Corporation.

[3] Y. Carson and A. Maria, "Simulation optimization: methods and applications," in *WSC '97: Proceedings of the 29th conference on Winter simulation*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 118–126.

[4] G. Aggelou, *Mobile Ad Hoc Networks: From Wireless LANs to 4G Networks*, 1st ed. McGraw-Hill Professional, 2004.

[5] R. Ramanathan and J. Redi, "A brief overview of ad hoc networks: challenges and directions," *IEEE Communications Magazine*, vol. 40, pp. 20–22, 2002.

[6] X. Hong, K. Xu, and M. Gerla, "Scalable routing protocols for mobile ad hoc networks," *IEEE Network*, vol. 16, pp. 11–21, 2002.

[7] C.-K. T. E Royer, "A review of current routing protocols for ad hoc mobile wireless networks," *Personal Communications, IEEE [see also IEEE Wireless Communications]*, vol. 6, no. 2, pp. 46–55, 1999.

[8] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, 1999, pp. 90–100.

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

[10] C.-C. Chiang and M. Gerla, "Routing and multicast in multihop, mobile wireless networks," in *Universal Personal Communications Record, 1997. Conference Record., 1997 IEEE 6th International Conference on*, vol. 2, 1997, pp. 546–551 vol.2.

[11] Z. J. Haas and M. R. Pearlman, "The performance of query control schemes for the zone routing protocol," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 427–438, 2001.

[12] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wirel. Netw.*, vol. 11, no. 4, pp. 419–434, 2005.

[13] W. Pretorius, "The development of a dynamically configured wireless ad-hoc multihop network protocol," Master's thesis, Stellenbosch University, December 2006.

[14] J. Doyle, *Routing TCP/IP, Volume II (CCIE Professional Development)*, 2nd ed. Cisco Press, October 2005, vol. 1.

[15] http://www.brighton-webs.co.uk/montecarlo/concept.asp.

[16] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, September 1949.

[17] R. V. Zyl, "Die implementering van die monte carlo partikelsimulasie algoritme op gaas strukture," Master's thesis, Stellenbosch University, 1994.

[18] S. L. J. Lihong Wang and L. Zheng, "Mcml - monte carlo modeling of light transport in multi-layered tissues," *Computer Methods and Programs in Biomedicine 47*, pp. 131–146, 1995.

[19] P. Jaeckel, *Monte Carlo Methods in Finance*. Wiley, 2002.

[20] K. Binder and D. Heermann, *Monte Carlo Simulation in Statistical Physics, An Introduction*, 4th ed. Springer, 2002, vol. 80.

[21] http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html.

[22] http://www.omnetpp.org/.

[23] http://www.isi.edu/nsnam/ns/.

[24] http://www.opnet.com/.

[25] http://pcl.cs.ucla.edu/projects/glomosim/.

[26] http://nsl10.csie.nctu.edu.tw/.

[27] http://jist.ece.cornell.edu/.

[28] http://inet.omnetpp.org/.

[29] http://mobility-fw.sourceforge.net/.

[30] H. Tijms, *Algorithmic Analysis of Queues*. Wiley, 2003.

[31] J. D. C. Little, "A proof for the queuing formula: $l = \lambda w$," Case Institute of Technology, Cleveland, Ohio, November 1960.

[32] L. Breuer and D. Baum, *An Introduction to Queueing Theory and Matrix-Analytic Methods*. Springer, 2005.

[33] G. Spreeth, "Design of a low power wireless sensor network for environmental monitoring," Master's thesis, Stellenbosch University, December 2008.

[34] http://www.tinyos.net/.

[35] K. Romer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, December 2004.

[36] Q. Cao, T. He, L. F. T. Abdelzaher, J. Stankovic, and S. Son, "Efficiency centric communication model for wireless sensor networks," in *IEEE Infocom*, 2006, pp. 1–12.

[37] Z. Haas, J. Halpern, and L. Li, "Gossip-based ad hoc routing," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1707–1716 vol.3.

[38] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen, "Scalable routing strategies for ad hoc wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1369–1379, Aug. 1999.

[39] M. K. Marina and S. R. Das, "Ad hoc on-demand multipath distance vector routing," *Wirel. Commun. Mob. Comput.*, vol. 6, no. 7, pp. 969–988, 2006.

[40] J. Li, C. Blake, D. S. J. D. Couto, D. S. J. De, C. Hu, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," 2001, pp. 61–69.

[41] M. Gen, R. Cheng, and L. Lin, *Network Models and Optimization: Multiobjective Genetic Algorithm Approach (Decision Engineering)*, 1st ed. Springer, 2008.

[42] C.-H. Ng, *Queueing Modelling Fundamentals.* Wiley, 1996.