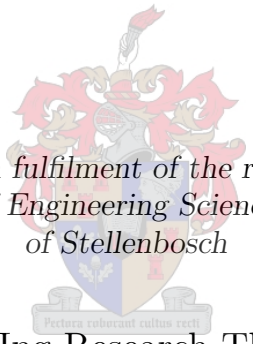
The Development of a Dynamically Configured Wireless Ad-Hoc Multihop Network Protocol

by

Wynand Pretorius



Thesis Presented in fulfilment of the requirements for the degree of Masters of Engineering Sciences at the University of Stellenbosch

MScIng Research Thesis

Department of Electrical Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland

Study leader: Dr. R. Wolhuter

December 2006

Declaration

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

Signature: .....
WJ Pretorius

Date: .....

Abstract

An ad-hoc network encompasses the cooperative engagement of a collection of mobile nodes that are free to move and communicate with each other wirelessly without the required intervention of any centralized access point or existing infrastructure.

The advantage of such a network lies in it's robustness, adaptiveness, the fact that its self-configurable and that it becomes somewhat indestructible due to it's decentralized nature. But such a network layout simultaneously introduces many complex network management issues which are normally taken care of inherently by a rigid network architecture.

The biggest challenge faced by any such protocol is the fact that it needs to be scalable, must maintain a decent stable data throughput, all whilst performing it's own continuous network management and associated routing algorithms.

These mobile nodes need a complex, scalable, compact and essentially real-time algorithm for maintaining an up to date representation of the overall network layout, yet without clogging the system's communications channels with too much overhead traffic, and drastically lowering the effective data throughput.

Since each mobile node only has a limited communications range each node also needs very advanced routing capabilities which will allow it to track who is currently within communications range, and at the same time allow the node to create multihop paths to distant destination nodes, thus connecting nodes which cannot directly communicate.

This report follows the development process of both the software needed to successfully conceptualize, simulate and test the protocol, as well as the hardware needed as proof of concept. It highlights and discusses the various design choices / considerations made in development of such a protocol, the strong- and weakpoints of the developed protocol, as well as providing several possibilities to further evolve the developed protocol.

Uittreksel

'n Ad hoc network bestaan uit 'n aantal nodusse wat elkeen vrylik kan rondbeweeg, saamwerk en kommunikeer sonder om fisies aan mekaar verbind te wees ("wireless") en wat ook nie van 'n sentrale beheerpunt of bestaande infrastruktuur gebruik maak nie.

So 'n tipe network het baie voordele: dit is kragtiger, meer aanpasbaar, kan self konfigureer en is bykans onvernietigbaar as gevolg van die gedesentraliseerde wyse van skakeling. Dit bring egter komplekse netwerk-beheer kwessies na vore, wat nie by vaste argitektuur netwerke voorkom nie, of wat weens die rigiditeit van vaste netwerke inherent hanteer word.

Die grootste uitdaging van so 'n netwerk protokol is dat dit 'n veranderende aantal nodusse moet kan hanteer en nog deurgaans 'n aanvaarbare en stabiele data toevoer handhaaf ten spyte van die komplekse netwerk beheer met al die meegaande roeterings ("routing") algoritmes. Elke beweeglike nodus benodig dus 'n komplekse, aanpasbare, kompakte en basies in-tydse algoritme wat dit in staat sal stel om op hoogte te bly van veranderinge in die oorhoofse netwerk uitleg, sonder om die netwerk se kommunikasiekanale te oorlaai met te veel administratiewe verkeer en sodoende die effektiewe data oordrag te verlaag.

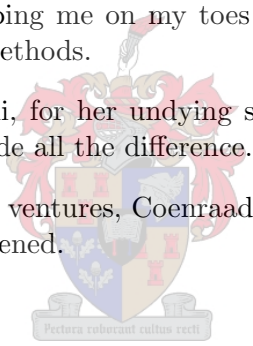
Omdat elke nodus se kommunikasiegebied beperk is tot 'n sekere radius, benodig dit verder gevorderde roeterings-vermoëns om te kan bepaal watter nodusse binne kommunikasie afstand is en moet dit ook in staat wees om multi-sprong paaie of roetes te skep via ander nodusse om sodoende met nodusse wat buite sy direkte kommunikasiegebeid val te kan kommunikeer.

Hierdie verslag beskryf die ontwikkelingsproses van beide die sagteware wat benodig was om die protokol te konseptualiseer, simuleer en toets, asook van die hardeware wat nodig was om te kan bewys so 'n protokol werk. Verder benadruk en bespreek dit die verskillende ontwerp keuses en oorwegings betrokke by die ontwikkeling van die protokol, asook die swak- en sterk punte van die ontwikkelde protokol. Verskillende moontlikhede word ook genoem van hoe die ontwikkelde protokol verder verbeter en uitgebrei kan word.

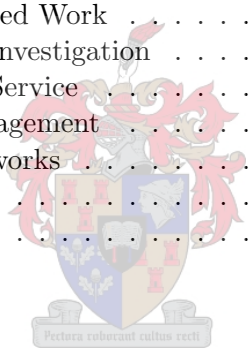
Acknowledgements

I would like to thank the following people:

- My supervisor, Dr. R. Wolhuter, for his support and continuing enthusiasm for this project.
- My parents, for their support, sympathy and always being there for me.
- My flatmates for keeping me on my toes with their challenging and progressive support methods.
- My girlfriend, Annemi, for her undying support, understanding and compassion which made all the difference.
- My colleague in many ventures, Coenraad, for keeping me sufficiently distracted, yet enlightened.



Contents



List of Figures

List of Tables

Nomenclature

Acronyms and abbreviations:

μs	Microsecond
ms	Millisecond
ACK	Acknowledgement
ALOHA	Routing protcol originating from University of Hawaii
AODV	Ad-hoc On-Demand Distance Vector
AP	Access Point
BOR	Brown-Out Reset
C	A high level programming language
CDC	Communication Device Class
CLK	Clock
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
DataAck	Data Acknowledgement packet
DELPHI	Programming language
DSDV	Distance Sequenced Distance Vector
DSR	Dynamic Source Routing
EEPROM	Electrically Erasable Programmable Read-Only Memory
FAQ	Frequently Asked Questions
FCC	Federal Communications Commission
GLUT	OpenGL Utility Toolkit
GUI	Graphical User Interface
ID	A means of identification
IDE	Integrated Development Environment
IO	Input / Output
ISM	Industrial-Scientific-Medical
ISP	In Service Programmer
ISR	Interrupt Service Routine
JAVA	Platform independant programming language

LED	Light Emitting Diode
MCLR	Master Clear
MCU	Micro Controller Unit
MPLAB	An IDE for programming PICMicro devices
OE	Output-Enable
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PIC	Programmable Interrupt Controller
POR	Power-On Reset
RAM	Random Access Memory
RERR	Route Error
RF	Radio Frequency
RIP	Routing Information Protocol
ROM	Read Only Memory
RREQ	Route Request
RREQReply	Route Request Reply
RS-232	A computer serial port interface
RT	Routing Table
RTT	Routing Temporary Table
RX	Receive
STDBY	Standby
SRAM	Static Random Access Memory
SWARM	Self-configurable Wireless Ad-hoc Relay and Messaging
TTL	Time To Live
TX	Transmit
USB	Universal Serial Bus
WDT	Watch Dog Timer
WLAN	Wireless Local Area Network
ZRP	Zone Routing Protocol

Chapter 1

Introduction

1.1 Project Motivation

The Internet was started as a military project to create a communications system which would be virtually indestructible due to it's redundant nature. For instance, if two nodes, A and B wanted to transmit data to one another, the actual data packets could traverse any one of many paths between the nodes. Thus if one path was destroyed, another would be taken. This military project evolved over the years and became what is today known as the Internet.

The Internet connects the whole world together, yet there is still one physical weakness to the system, because even though many complete paths exist between two nodes, closer to a node itself it is likely that there exists only one connection path between the node and it's Internet gateway, be it a telephone line or broadband connection. In large areas of the world, this weakness is more of a disability, since the basic infrastructure which would allow an Internet connection simply does not reach everywhere, the so-called 'Last Mile Problem'. In other countries we find a variation of the 'Last Mile Problem' in which the infrastructure necessary is available almost everywhere, but the cost per connection is too high for the average individual, forming exactly the same kind of connectivity disability as in the first case.

These problems stem from the fact that the current broadband Internet access paradigm relies on cable and DSL being deployed in individual homes, is centrally managed and requires extensive planning, layout, continual management and maintenance to operate successfully. This means that even if two nodes are physically seperated by only a couple of meters the data packets they send each other might be traversing a much longer complete path through the bigger network in order to connect them, a service for which they might be paying dearly.

A possible solution can be found in the form of a mesh network in which local nodes are connected together directly, without the need to rely on the prior existence of any preconfigured network topology. These mesh network nodes should be able to power up, detect existing mesh networks in the area and join these networks without any user intervention. In such a dynamically configured ad-hoc mesh network, each node has two roles besides network management. It has to forward packets from other nodes (acting as a repeater) and be able to transmit it's own packets.

A mesh network in an urban scenario allows users within a community to connect directly to one another, thus forming one big community mesh network where each user is only responsible for their own mesh network node. Such a community mesh network in which neighbours directly connect their home networks together holds many advantages for those involved, for example, when enough neighbours cooperate and forward each other's packets, each user does not need to individually install an Internet connection (gateway), but instead can share faster, cost-effective Internet access via gateways that are distributed in their neighbourhood. Packets dynamically find a route, hopping from one neighbour's node to another to reach the Internet through one of these gateways. Sharing of information inside a community by means of an electronic community bulletin board or direct file transfers becomes accessible to all, and is important because it allows free flow of information without any moderation or selective rate control. An example of a possible urban mesh network scenario is illustrated in Figure 1.1.

From an industrial or military point of view, an ad-hoc network has many advantages. A network can be installed in an environment where infrastructure to transport electronic information simply doesn't exist and an installation is simply not feasible. For example during a military operation in a foreign country where infrastructure installation is hazardous, expensive, temporary and vulnerable to attack, an ad-hoc mesh network can quickly be put together by simply dropping nodes in place. The more nodes there are, the more robust and indestructible the network. For instance, if the nodes were inexpensive, it could be feasible to drop thousands from an aeroplane over the area where an operation is planned. Thus communication is already set up upon arrival and the communications channels should be quite robust. If a tank drove over, or an explosion destroys many of these nodes, those remaining will simply adapt to the changes and relay the information as if nothing had happened. An industrial scenario such as a network installation inside an existing power plant / water pumping station is also a good location for an ad-hoc network, since this is not a network friendly environment and modifications to the building are very time consuming and difficult to make. An example of an ad-hoc network installation in a water pumping station is shown in Figure 1.2. In the figure we see a wireless mesh network pictured,

as was deployed in a water treatment plant. This implementation includes repeaters between the control room and the monitoring nodes to enhance the reliability of information transfer. Some of the characteristical advantageous of an ad-hoc mesh network are demonstrated in Figure 1.2 where some of the nodes are positioned to act merely as repeaters for sending information coming from the pipe galleries back up to the communications room.

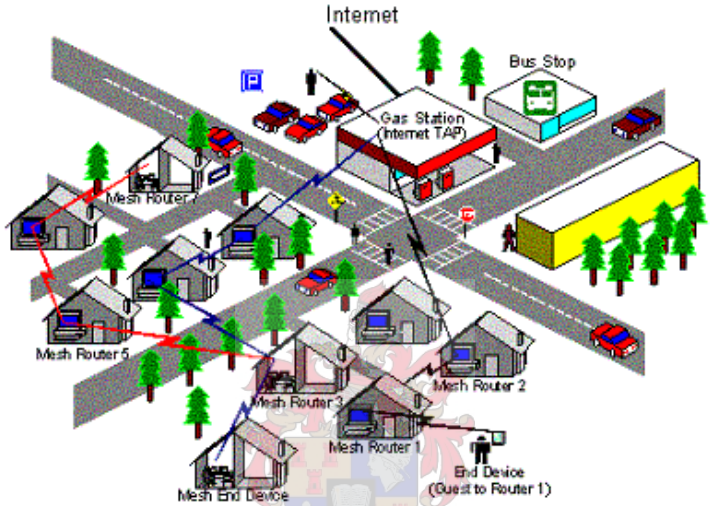


Figure 1.1: An example of a possible community mesh network [1].

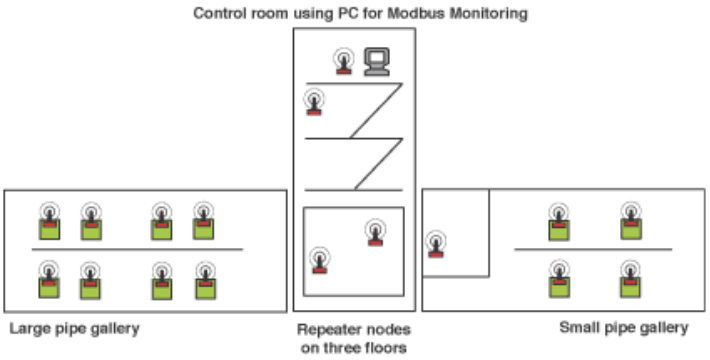


Figure 1.2: A wireless mesh network installation inside a water treatment plant [2].

1.2 Project Description

This project involves the design and implementation of a dynamically configurable wireless ad-hoc network protocol. There exists many ad-hoc / mesh network protocols, albeit most in theory, but each protocol was usually created to solve a specific problem, and outside the scope of that problem it is not very useful. A lot of the protocols were also tested using modified 802.11 transmissions, which simplifies the hardware needed for testing but fails to decently test the new protocol since results are still prejudiced by the 802.11 standard which was not optimized for a non-infrastructure network.

Subsequent to designing the protocol, simulations were used to test the vast number of potential network nodal layouts possible, which an ad-hoc network node would have to automatically be able to organize in such a way that it knows who it can communicate and connect with directly, and how it can communicate with those nodes whom it can't reach directly. This simulation was created in such a way that the final simulation code could be used directly in the firmware, with the only modifications being to the actual message transmission functions.

The implementation of the protocol involved the development of a SWARM (Self-Configurable Wireless Ad-hoc Relay and Messaging) Network Node to be connected via the USB port of a PC, as well as the development of the necessary software to establish communication between the device and the software application. The USB port is chosen for communication with the SWARM device since all modern PC's have more than one USB port which allows for easy connection of one or multiple nodes. USB devices have many other advantages, such as portability and 'plug and play' functionality, as described in Chapter 7. A complete USB prototype device that can wirelessly connect to other such devices within range, and that can interface with and be controlled from the computer's USB port has to be developed. This prototype is to be very adaptable and robust, has to automatically perform a whole range of functions in order to simply remain connected, and must be able to transfer data in an ever changing network, as described in Chapter 3.

1.3 Design Considerations for an Ad-hoc, Mesh Network Protocol Application

Some introductionary design considerations for a mesh network protocol were already indirectly mentioned in Section 1.1 and will be further formalized in Chapter 3.

In developing a complete ad-hoc mesh network protocol application the USB port was chosen as connection type between the network node and PC, however not all nodes will necessarily be connected to a PC. In such a case the node cannot generate new network data since it is not receiving any new input, but must be able to operate as a complete fully functional mesh network repeater node, or destination node (in the event of it being a PowerRelay node) if provided only with power. The host PC application serves merely as a gateway into the network for any prospective user. The ad-hoc mesh network protocol application will be subject to the following design considerations:

- The SWARM device can be activated from the host PC or manually with a switch on the device.
- The expected mobility rate of the SWARM device can be set from the application GUI or remains set to the nodal default in the event of a manual activation.
- The network node needs to be fully self contained if not connected to the host PC application via the USB cable and should need only to be provided with power from an alternate source.
- If a node isn't connected to a host PC it cannot generate any new network data and can only forward messages from other nodes, e.g. it's functionality gets reduced to acting as a repeater only.
- The host PC application should allow the user to see the connectivity status (E.g. link quality, number of active connections) of the node connected to it via the USB cable.
- The host PC application should allow the user to select a messaging node connected somewhere in the mesh network as destination node and subsequently send message to that node. It must also always be able to receive text messages from other messaging network nodes.
- The host PC application should allow the user to select a PowerRelay node connected somewhere in the mesh network as destination node and switch the relay at that node on or off.

1.4 Literature Study

Before any development would be done, a study had to be made of the following:

- Ad-hoc Network Protocols
- Mesh Network Protocols
- Medium Access Control Protocols
- Data Link Protocols
- Programming languages and functions available, with which to build a suitable simulator.
- The hardware components that would allow creation of a low-cost, fully functional node based upon the developed protocol.
- The firmware programming environment, language and tools to program the hardware.

1.5 Project Outcome

A ad-hoc network protocol was developed, performance examined and refined using custom built simulators. As proof of concept, and to verify the results of the simulators 5 ad-hoc network nodes were built using general purpose microcontrollers and WiFi transceivers.

The network nodes developed conformed to all the requirements for dynamic self-configured ad-hoc network nodes as set out in Chapters 1 and 3. These nodes can be powered up by a host PC, allowing the user to set several parameters, or automatically in which case the factory default settings are used. In both cases the nodes scan for other nodes within range, join a network if one is found, otherwise create one, setup an addressable network structure automatically, and maintain this network structure dynamically. They can then communicate directly with one another, or indirectly by relaying messages for one another.

The results obtained from the hardware experiments conducted, confirmed the theoretical results, closely matched the simulation results and indicated which areas could be further investigated.

1.6 Thesis Outline

This thesis starts by giving an overview of the literature study that was done. The protocol development, testing through simulation, and the hardware and software design of the dynamically configurable ad-hoc network node USB device is discussed in the subsequent chapters.

An overview of existing work relating to ad-hoc networks, or more specifically some common routing problems and dynamic routing protocols, as well as some medium access control protocols are given in Chapter 2.

In Chapter 3 a set of functional characteristics for the protocol developed is defined.

Chapter 4 examines the practical implementation of these functional characteristrics.

Chapter 5 gives an overview of the proposed system as a whole and the design process that was followed during development of the project.

In order to refine and test the protocol, it was necessary to be able to test it under various extreme conditions while still maintaining control over the network. Two simulators were developed for this goal, of which an overview and some performance results are provided in Chapter 6.

Chapter 7 takes a look at what hardware would be necessary to provide the functionality required by a SWARM ad-hoc network node. It provides an overview of the Microchip microcontroller chosen, briefly discusses USB I/O devices, the relevant device drivers needed to achieve USB functionality, and provides an overview of the RF transceiver chosen for use in the prototype.

Chapter 8 describes how the PIC USB microcontroller, it's USB functionality and the RF transceiver would be integrated in the prototype hardware design to meet the system specification. The microcontroller code (firmware) necessary to provide a functional framework within which a USB ad-hoc network node can be developed is briefly discussed in this chapter.

Chapter 9 covers the detailed hardware design of the final prototype device and concludes with a complete schematic design, which was used to design the printed circuit board (PCB).

The firmware design (software running on the device's microprocessor) is discussed in Chapter 10.

Chapter 11 presents the host PC user interface software design.

The final prototype is tested and its operation verified. The testing procedures and the test results of the evaluation are given in Chapter 12.

Finally, Chapter 13 presents a summary and conclusions.

Chapter 2

Overview of Existing Protocols

2.1 Overview of Dynamic Routing Protocols

In this chapter, we examine some of the typical characteristics present in most dynamic routing protocols, before examining several specific routing protocols which are directly relevant to the SWARM network protocol being designed.

2.1.1 Distance Vector Routing Protocols

Most of the protocols summarized as part of this chapter fall into the distance vector class of routing protocols. Distance vector algorithms are based on the work done by R.E. Bellman, L.R.Ford, and D.R.Fulkerson; thus are occasionally referred to as *Bellman-Ford* or *Ford-Fulkerson* algorithms.

The name, distance vector, is derived from the fact that routes are advertised as vectors of (distance, direction), where distance is defined in terms of a metric and direction is defined in terms of the next-hop router. For example, "Destination A is a distance of 5 hops away, in the direction of next-hop router X." As the statement implies, each router learns routes from its neighbouring routers' perspectives and then advertises the routes as its own perspective. Because each router depends on its neighbours for information, which the neighbours in turn, may have learned from their neighbours, and so on, distance vector routing is sometimes facetiously referred to as "routing by rumour." [10]

A typical distance vector routing protocol uses a routing algorithm in which routers periodically send routing updates to all neighbours by broadcasting their entire routing tables.

2.1.2 Proactive Protocols

Proactive protocols attempt to continuously evaluate the routes within the network, so that when a packet needs to be forwarded, the route is already known and can be immediately used. The advantage of the proactive schemes is that, once a route is requested, there is little delay until that route is determined [11].

2.1.3 Reactive Protocols

Reactive protocols invoke the route determination procedures on demand only. Thus, when a route is needed, some sort of global search procedure is employed. In reactive protocols, because route information may not be available at the time a routing request is received, the delay to determine a route can be quite significant [11].

2.1.4 Hybrid Protocols

Hybrids are a combination between reactive and proactive protocols. Some functions are performed using a reactive protocol, whilst others are still done utilizing a proactive protocol.

2.1.5 Periodic Updates

Periodic updates means that at the end of a certain time period, updates will be transmitted. The data contained in these updates depends on the routing protocol. The issue regarding updates is the rate at which updates are sent. If sent too frequently, network congestion may occur; if updates are sent too infrequently, the network adaption time period may be unacceptably high [10].

2.2 Common Routing Protocol Problems

2.2.1 Route Invalidation Timers

Assume the network layout shown in Figure 2.1 has stabilized, how would topology changes be taken care of? If node B goes offline, then all three other nodes will still forward messages towards an unreachable destination.

This problem is handled by setting a route invalidation timer for each entry in the route table. For example when node A first hears about node B and enters the information in it's routing table, node A sets a timer for that route. At every regularly scheduled update from node B, node A discards the update's already-known information about node B, but as it does so it resets the timer for node B.

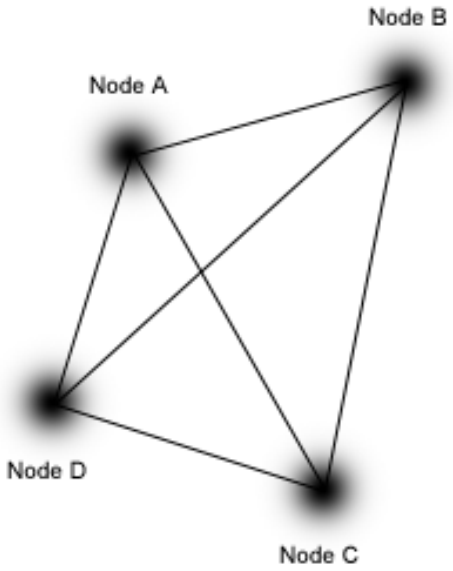


Figure 2.1: A simple mesh network layout

Thus if node B goes down, the RT entry would expire at node A, as well as all the other nodes [10].

2.2.2 Split Horizon

Since nodes broadcast updates, node A will update node B, and vice versa. This could cause a problem if another node, say node E, was connected to node B, but out of range of node A. Thus node A gets all its updates regarding node E via node B. If node E goes offline, and the RT entry for node E expires in node B's routing table, then it could happen that between the entry expiring and node B sending an update, node A broadcasts an update. In this update node A advertises a route to node E, thus node B creates a RT entry for node E via node A.

Now a packet with a destination address of node E arrives at node B, node B consults it's RT and forwards the message to node A. Node A consults its RT and forwards the packet to node B, node B forwards it back to node A, ad infinitum. A routing loop has occurred.

Split horizon is a method whereby information learned from a specific node does not get advertised back to that node thus preventing routing loops as mentioned from occuring [10].

2.2.3 Counting to Infinity

Another form of routing loop which can still take place even if split horizon is implemented, occurs in a network such as the one shown in Figure 2.1, but without the inner links (thus a circular connection of nodes), in which a fifth node recently disconnected.

The same type of situation as in the split horizon scenario occurs, when the false route is advertised around the network from one node to the next, gaining an extra hop with every update. Each node receives the update, reckons that even though the route has a greater hop count it is still the only route available, and stores it. This routing update will continue to circulate, with a hop count approaching infinity.

To prevent this a maximum hop count for a network should be defined. When a route reaches this hop count its destination is classified as unreachable [10].

2.2.4 Triggered Updates

Triggered updates occur as soon as a routing metric changes for better or for worse, it doesn't wait for the normal periodic update timer to expire first. Thus changes are allowed to ripple through a network much quicker than if every node had to wait for regularly scheduled updates.

The problem of counting to infinity is greatly reduced, although not completely eliminated since periodic updates may still occur along with triggered updates. Thus a node might receive bad information about a route after having received correct information from a triggered update, but triggered updates will definitely help to propagate changes more quickly [10].

2.2.5 Holddown Timers

Holddown timers introduce a certain amount of skepticism to reduce the acceptance of bad routing information.

If the distance to a destination increases (for example, the hop count increases from 2 to 4), the node sets a holddown timer for that route. Until the timer expires, the node will not accept any new updates for the route.

Thus holddown timers must be set with care, as they reduce the likelihood of bad routing information getting into a table, but slow down the adaption rate of a network to changes. If the holddown period is too short, it will be ineffective, and if it is too long, normal routing will be adversely affected [10].

2.3 Medium Access Control Protocols

2.3.1 ALOHA

All data transmissions occur immediately when necessary and if a collision does occur, the nodes simply do a retransmission. However, retransmissions need to occur with a random backoff time, else the retransmissions from different nodes will also collide.

The disadvantage here is that the more collisions occur, the more retransmissions are scheduled and the more data transmissions get backlogged at each node waiting to transmit; which only further increases the likelihood of additional collisions. As a result of this retransmitive snowball effect the protocol performs best under light loads [12].

2.3.2 CSMA/CA

The low maximum throughput of the ALOHA schemes is due to the wastage of transmission bandwidth because of frame collisions. This wastage can be reduced by avoiding transmissions that are certain to cause collisions. By sensing the medium for the presence of a carrier signal from the other stations, a station can determine whether there is an ongoing transmission. Hence the name, carrier sensing multiple access (CSMA).

CSMA schemes come in many flavours, with CSMA/CD and CSMA/CA being popular ones. CSMA/CD has collision detection built in, but cannot be practically used in a radio environment since it is difficult to detect collisions in an radio environment (interference from various nodes; and transmitted power being so much greater than receiving power) and the hidden node problem.

The hidden node problem, shown in Figure 2.2, occurs when all nodes aren't within range of one another. For example, node A and node C could both sense that the carrier is free, perform a transmission, and neither would be able to detect the resulting collision.

The Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) medium access control was developed specifically to prevent this type of collision. CSMA/CA requires that nodes first monitor the channel to determine whether the medium is idle or busy. If the medium is busy all nodes waiting to transmit have to wait till the channel becomes idle.

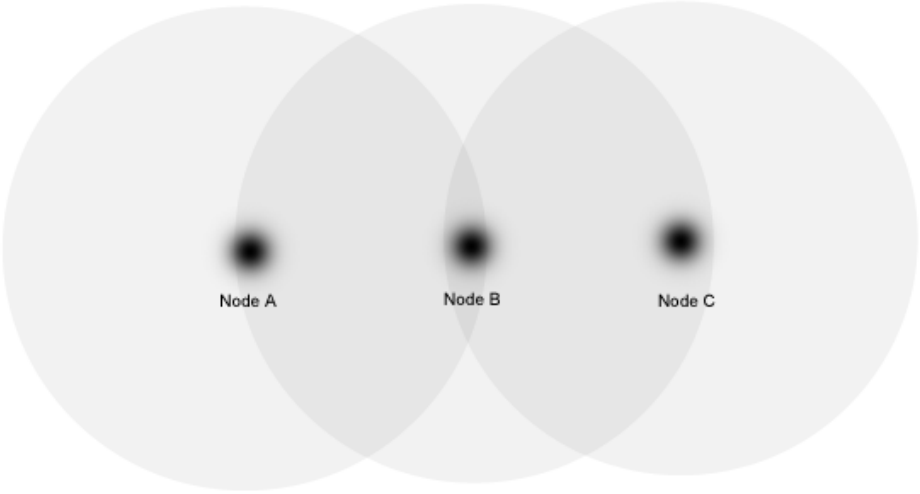


Figure 2.2: Hidden node problem

When it becomes idle all nodes are required to wait a certain fixed amount of time, whereafter each node is assigned a random time, and whenever this time expires a node can perform a transmission[12].

2.4 Dynamic Routing Protocols

2.4.1 ZRP

Zone Routing Protocol (ZRP) is a hybrid routing protocol which effectively combines the best features of both proactive and reactive routing protocols. The key concept employed in this protocol, is to use a proactive routing scheme within a limited zone in the r-hop neighbourhood of every node, and use a reactive routing scheme for nodes beyond this zone.

An *intra-zone routing protocol* (IARP) is used in the zone where a particular node employs proactive routing. The reactive routing protocol used beyond this zone is referred to as *inter-zone routing protocol* (IERP). The *routing zone* of a given node is a subset of the network, within which all nodes are reachable within less than or equal to *zone radius* hops .

Nodes within the routing zone exchange periodic route update packets, hence the larger the routing zone, the higher the update control traffic. The IERP is responsible for finding paths to nodes which are not within the routing zone.

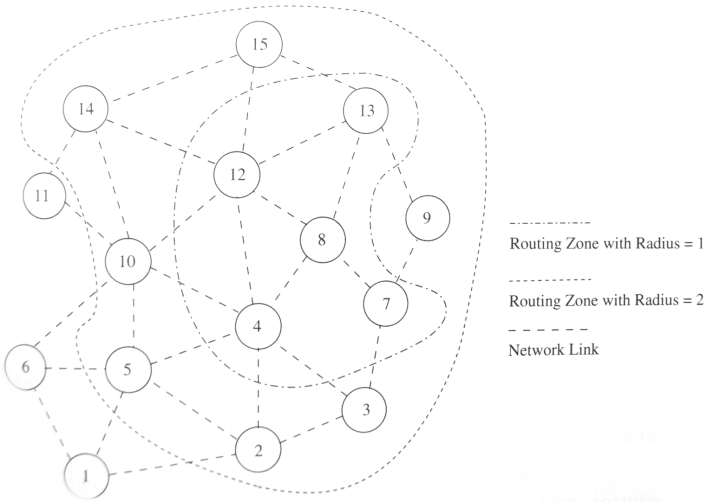


Figure 2.3: Routing zone for node 8 in ZRP[3]

This protocol produces high control overhead if it is not ensured that redundant or duplicate Route Requests are not forwarded, and the exact size of the zone radius has a significant impact on the performance of the protocol[3].

2.4.2 AODV

The Ad hoc On-demand Distance Vector (AODV) is a reactive routing protocol. Hello messages may be used to detect and monitor links to neighbours, otherwise a node monitors communications around itself to maintain a connectivity table.

If Hello messages are used, each active node periodically broadcasts a Hello message that all it's neighbours receive. If a node thus fails to receive several Hello messages from a neighbour, then a link break is detected.

When a source has data to transmit to an unknown destination, it broadcasts a Route Request (RREQ) for that destination. At each intermediate node, when a RREQ is received a route to the source is created. If the receiving node has not received this RREQ before, is not the destination and does not have a current route to the destination, it rebroadcasts the RREQ. If the receiving node is the destination or has a current route to the destination, it generates a Route Reply (RREP). The RREP is unicast in a hop-by-hop fashion to the source. As the RREP propagates, each intermediate node creates a route to the destination. When the source receives the RREP, it records the route to the destination and can begin sending data. If multiple RREPs are received by the source, the route with the shortest

hop count is chosen. As data flows from the source to the destination, each node along the route updates the timers associated with the routes to the source and destination, maintaining the routes in the routing table.

If a route is not used for some period of time, a node cannot be sure whether the route is still valid; consequently, the node removes the route from its routing table. If data is flowing and a link break is detected, a Route Error (RERR) is sent to the source of the data in a hop-by-hop fashion. As the RERR propagates towards the source, each intermediate node invalidates routes to any unreachable destinations. When the source of the data receives the RERR, it invalidates the route and reinitiates route discovery if necessary [13][14].

2.4.3 DSDV

The Destination Sequenced Distance-Vector routing protocol (DSDV) is one of the first protocols proposed for ad hoc wireless networks. It is an enhanced version of the distributed Bellman-Ford algorithm where each node maintains a table that contains the shortest distance and the first node on the shortest path to every other node in the network, thus it is a proactive routing protocol. It incorporates table updates with increasing sequence number tags to prevent loops, to prevent the count-to-infinity problem, and to allow the network to settle faster.

The protocol suffers from excessive control overhead (fairly brute force, it depends for it's correct operation on the periodic advertisement and global dissemination of connectivity informarion) that is proportional to the number of nodes in the network and therefore is not scalable in ad hoc networks which are highly dynamic and where bandwidth is limited [15][3].

2.4.4 RIP

The Routing Information Protocol (RIP) transmits updates periodically, and immediately when the network topology changes. These updates are used to keep RTs up to date, and only the best route (the route with the lowest metric value) to a destination is maintained. RIP uses a single routing metric (hop count) to measure the distance between the source and destination node.

RIP prevents routing loops from continuing indefinitely by implementing a limit (15) on the number of hops allowed in a path from the source to a destination. The downside of this stability feature is that it limits the maximum diameter of a RIP network to less than 16 hops.

RIP includes a number of other stability features that are common to many routing protocols. These features are designed to provide stability despite potentially rapid changes in a network's topology. For example, RIP implements the split horizon and holddown mechanisms to prevent incorrect routing information from being propagated.

RIP uses numerous timers to regulate its performance. These include a routing-update timer, a route-timeout timer, and a route-flush timer. The routing-update timer clocks the interval between periodic routing updates. Generally, it is set to 30 seconds, with a small random amount of time added whenever the timer is reset. This is done to help prevent congestion, which could result from all routers simultaneously attempting to update their neighbors. Each routing table entry has a route-timeout timer associated with it. When the route-timeout timer expires, the route is marked invalid but is retained in the table until the route-flush timer expires [16].

2.4.5 DSR

Dynamic Source Routing (DSR) is a reactive routing protocol designed to restrict the bandwidth consumed by control packets in ad hoc wireless networks by eliminating the periodic table-driven approach. The major difference between this and the other on-demand routing protocols is that it is beacon-less which means that there are no hello-messages used between the nodes to notify their neighbours about their presence.

If a node has data to transmit to a specific destination the node floods the network with a *Route Request* message. This Route Request is forwarded by all nodes, except if this is the second time a node has received a specific route request. When a route request reaches the destination node, the node will send a Route Request Reply back to the source node. The data messages subsequently sent out by the source node each carry the complete path information from source to destination. Thus nodes along the route store this information regarding the network layout in a route cache. The node can then use this route cache in future communications.

If a link between two nodes is broken a *RouteError* message is generated from the adjacent node to inform the source node. The source node will then reinitiate the route establishment procedure.

The DSR protocol was developed for networks with a small diameter, between 5 and 10 hops, and the nodes should only move around at a moderate speed as performance degrades rapidly with increasing mobility. Even though intermediate nodes can utilize the routing information contained in data packets efficiently; the routing overhead introduced is directly propor-

tional to the path length; a property which severely limits the maximum diameter of a DSR network. If a link was broken, and a route is sought again, the route cache information kept at intermediate nodes could also result in inconsistencies during the route reconstruction phase [3].

This chapter has provided a summary of the existing work found relevant to the project, has introduced most of the concepts and schemes which are used throughout this project, albeit in an original context or in an adapted form; and thus provides a good base from which to proceed with the protocol development.

In the next chapter we briefly summarize the characteristics of the protocol as developed hereforth.

Chapter 3

Functional Characteristics of Protocol Developed

The introduction in Chapter 1 provided a brief outline of the goals of the project. Before development is started though, the outline of the project needs to be expanded into more formal functional system requirements.

This chapter provides a short description of the functionality upon which the Self-configuring Wireless Ad-hoc Relay and Messaging (SWARM) Network protocol will be based.

Nodes in a SWARM Network:

- have a distributed routing scheme. Centralized routing involves high control overhead and hence is not scalable. Distributed routing is more fault-tolerant than centralized routing, which involves the risk of single point of failure.
- can move about freely and always remain connected when there are enough nearby nodes, the degree of mobility being indicated as a percentage between 0 and 100%.
- are identified and addressed according to a unique ID or NodeName by all other nodes in the network. The NodeName also indicates to other nodes whether a node is a relay or messaging node.
- send out periodic update messages. The rate at which these updates get sent out is determined by the degree of mobility of the system.
- accept, process and retransmit data messages destined for other nodes, thus forming a link in the chain between a specific source and destination node.

- scan for and connect to any other nodes in the area, which may or may not already be part of a bigger network of nodes, upon startup.
- must be able to find the shortest path between itself and a specific destination node through an interconnected series of nodes.
- identify and remove a route between a specific source and destination node which has become invalid since the last time it was used.
- automatically and continuously reconfigure the network in such a fashion that the efficiency remains at an optimum level.
- identify and remove a node from the RT which has become disconnected from the rest of the nodes.
- must join the network in such a way that they become addressable to other nodes already in the network
- must become addressable to nodes already in a network when moving into their RF range.
- must be able to connect and address a network consisting in total of more nodes than it can store in it's routing table directly. Thus enabling a node to be connected to any number of nodes in theory. The upper limit being determined by the acceptable network throughput rather than the size of the node's routing table size.

In Chapter 2 a brief overview of existing work was given; highlighting some of the main issues relating to ad-hoc networks, and which this protocol will have to deal with.

This chapter has listed the main functional characteristics of the SWARM protocol as developed. The next chapter takes an in-depth look at how the protocol deals with the networking issues mentioned, and how the functional characteristics mentioned in this chapter are implemented.

Chapter 4

System Design Overview

4.1 Overview

The functional characteristics and the key routing protocol building blocks for this project have been discussed in the previous chapters. This chapter discusses the design trajectory that was followed to design and refine the SWARM routing protocol, as well as the design trajectory that was followed to design and develop a USB ad-hoc network node based upon that protocol, and presents an overview of the system as a whole.

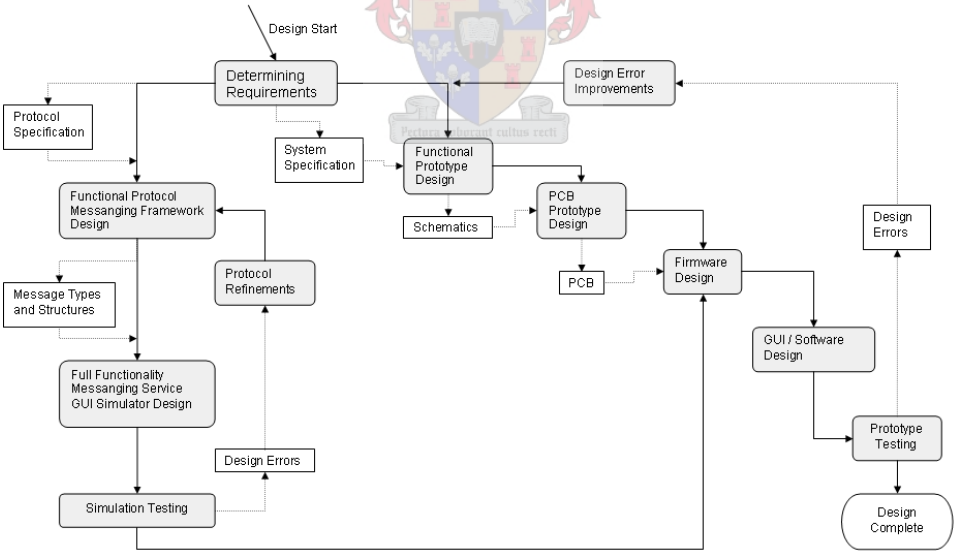


Figure 4.1: System design trajectory.

4.2 Design Process

A design trajectory (Figure 4.1) was followed to develop the SWARM protocol and the USB ad-hoc network node based upon this protocol. This trajectory can be divided into two main categories, the protocol design and the hardware design.

4.2.1 Protocol Design Process

The design trajectory for developing the SWARM protocol consists of the following steps:

1. A protocol specification has to be defined. In order to obtain the specification, an investigation of the protocol requirements for ad-hoc networks and the restrictions imposed on ad-hoc networks needs to be done. An overview of existing ad-hoc network protocols (Chapter 2) and the requirements for a dynamically configurable ad-hoc messaging protocol (Chapter 3) have been discussed already. The design considerations (Chapter 5) for such a protocol will be discussed in the next chapter.

2. A simulation environment has to be designed (Chapter 6), which consists of the following sub-steps:

   - A framework has to be designed which will allow messages to be passed between virtual nodes based upon the routing protocol.
   - All the message types and structures used in the protocol must be defined (Chapter 5)
   - A GUI has to be designed which allows the user access to the full functionality offered by the routing protocol, provides extra debugging information and can be set to run automatically.

3. The protocol has to be tested in this simulated environment, results verified and analyzed (Chapter 6).

4. The protocol has to be redesigned according to the results of the verification process.

4.2.2 Hardware Design Process

The design process for developing the SWARM network node hardware consists of the following steps:

1. A system specification has to be defined. In order to obtain the specification, an investigation of the system requirements, restrictions and the development environment needs to be done. The design considerations for a device to be used in a mesh network have been mentioned

in Chapter 1, Section 1.3; and 1.2) and the requirements in terms of the hardware interface (USB), communications capability (RF range, data rate), and node processing power are all discussed in Chapter 7.

2. The hardware device has to be designed, which consists of the following sub-steps:
   - Integrating all necessary components into a prototype which meets the system specification (Chapter 8).
   - Schematic design of the prototype device (Chapter 9).
   - Design of a printed circuit board (PCB).
3. The firmware for the PIC18F2550 microcontroller had to be designed (or adapted) to be able to utilize the full hardware functionality (Chapter 10). The firmware also has to be designed in such a way as to encorporate the complete refined protocol, which will enable the microcontroller to assume all the characteristics of a SWARM network node.
4. The software application has to be developed to be able to communicate with the hardware device and enable the user to utilize the full functionality offered by the SWARM network node (Chapter 11).
5. The prototype has to be tested and verified and the results analysed (Chapter 12).
6. The prototype has to be redesigned according to the results of the verification step and to accommodate for the added hardware functionality required in the later prototype.

The development of the hardware was divided into stages, because it allows the components to be individually designed and tested, without the additional complexity or influence of other hardware components.

4.3 System Overview

4.3.1 Simulation Software

The simulator has to enable the creation of virtual nodes, and allow the setup, transmission and reception of custom packets between these nodes. Thus the simulator core is formed by a suitable messanging framework, capable of handling multiple endpoints or nodes.

A GUI has to be built on top of this framework to provide the user with control over the network's nodal layout, provide the user with the ability to message one node from another node, the ability to see a specific node's

routing tables and a means of obtaining debugging information confirming the internal workings of the routing protocol.

4.3.2 Hardware and Firmware

Figure 4.2 shows a conceptual diagram of how the key hardware and software building blocks will be integrated. The shaded blocks indicate those hardware and software components that must be designed.

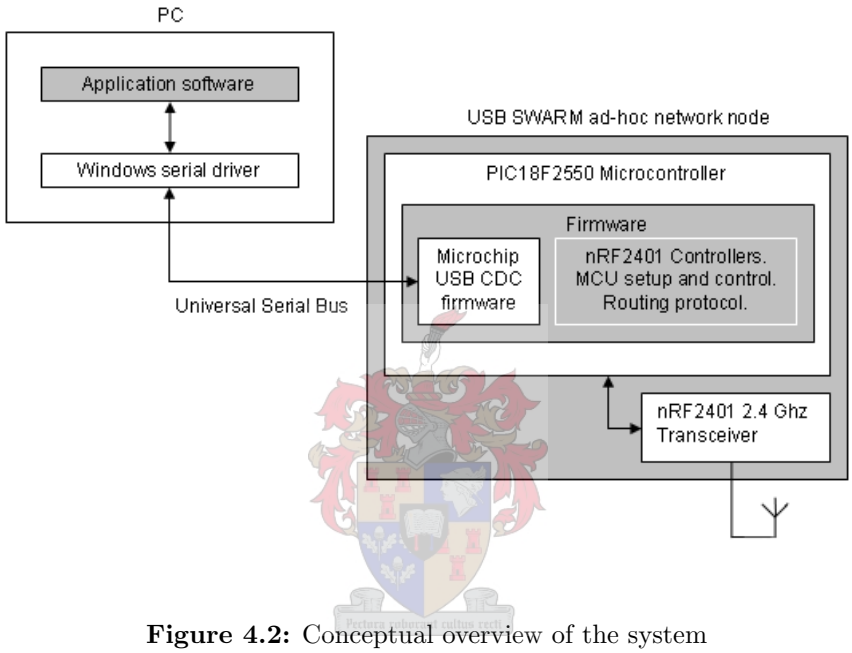


Figure 4.2: Conceptual overview of the system

The hardware device contains all the components shown, each of which will be discussed individually when looking at the design considerations for this device (Chapter 7).

4.3.3 Software GUI

Apart from the simulation software, a seperate graphical user interface to the USB ad-hoc mesh network node must be developed. It must provide enough functionality to enable real-world usefullness to the user (here implemented as a messaging and relay on/off control console) and a proper means with which to thoroughly test the underlying routing protocol. The GUI was written in Delphi and is thus restricted to use on the Windows operating system. Since the USB serial emulation process also relies on Windows drivers this does not in itself provide a further restriction to the usefullness of the software.

Chapter 5

Design Considerations for the SWARM Protocol

5.1 Overview

This chapter illustrates how the characteristics set out in Chapter 3 can be met, discusses all the core protocol functionality, and forms the foundation of the protocol developed from here onwards.

5.2 Mobility Variable

A mobile network is any network in which the shortest path connecting two nodes will change over time in direct relation to the node's current physical position. Such situations do not arise in wired networks where all nodes are stationary. Even though the wired network protocols find alternate routes during path breaks, these breaks don't occur frequently, and the rate at which changes propogate is very slow. An ad-hoc routing protocol must be able to perform fast, efficient and effective mobility management.

In an ad-hoc network where the actual nodal connectivity is dependant on the specific positioning of nodes the maximum mobility rate forms a very important component of the overall protocol. The maximum mobility rate determines the speed at which a node can move and remain connected and is determined by the range of the hardware and density of users. In the SWARM network protocol a mobility variable was included to indicate what the upper limit for the nodal maximum mobility rate is.

Looking back at Chapter 2 we see that the majority of the protocols discussed rely on individual network nodes broadcasting their routing information periodically, thus allowing other nodes in the vicinity to maintain connections pointing back to the broadcasting node. The SWARM network

protocol is no different in this regard, in that update messages are also sent out periodically, with the exception that the transmission period can be adjusted dynamically.

Usually the transmission period is set to an optimum value suitable for all network scenarios most probable to occur, and will be determined through extended experimentation and simulation. However, since one value can never really adequately cater for the many varied network layouts a mesh network node could find itself in, there are many scenarios in which a lot of protocols will perform very poorly, not by bad design, but simply because the protocol was not optimized for that scenario. For example, if a protocol is optimized for fast moving mobile network nodes and requires regular update messages broadcasted, but a node actually remains stationary for most of it's lifetime, then a huge amount of overhead is unneccessarily being introduced into a system, thereby wasting precious network resources.

To this end, the SWARM Network protocol was designed to adjust this update period dynamically in two regards. 1) - When the node starts up the expected mobility of the node can be set, being expressed as a percentage between 0% and 100%; with 0% indicating a stationary node and 100% a very mobile node. This mobility variable value is then used to set various timeouts, such as the maximum time a routing table entry remains valid, the maximum time a route between a source and destination remains valid, as well as the all important update broadcast transmisson period maximum value. 2) - The node will continuously monitor how active the network is in it's surrounding area, if it finds that since having started up the network hasn't changed much and seems to have stabilized, the node will keep decreasing the frequency of it's update broadcasts until it reaches the maximum time limit set according to the mobility variable / expected mobility of the node. The different maximum broadcast periods achievable and their corresponding mobility ratings are summarized in Table 5.2.1.

The effect this has on the total overhead produced by a node is quite dramatic. For a full comparison, refer to Chapter 12, Section 12.1.

5.3 Proactive or Reactive Protocol

Reactive routing protocols have a longer delay period before new data can be transmitted due to the route acquisition process it utilizes, and because of this long delay, pure reactive routing protocols may not be applicable to realtime communication. Nodes in a SWARM network are allowed to have a high mobility rate, thus the network changes may be more frequent than the routing requests, which means most of this routing information never

Mobility	Effective Maximum Update TX Period
0%	2.1sec
10%	2.1sec
20%	4.2sec
30%	8.4sec
40%	16.8sec
50%	33.6sec
60%	76.2sec
70%	134.4sec
80%	268.8sec
90%	537.6sec
100%	1075.2sec

Table 5.2.1: Effect of mobility variable on update period

gets used and is just an excessive waste of network capacity.

However, pure proactive schemes are likewise not appropriate for the SWARM network environment, as they continuously use a large portion of the network capacity to keep their routing information current and aren't scalable to a large number of nodes in terms of memory usage.

What is needed, is a hybrid protocol which effectively combines the best features of both proactive and reactive routing protocols in an optimal way suited to this application. The solution utilized by the SWARM network protocol was implemented by maintaining some routes using a proactive routing scheme within a limited zone, and using a reactive routing scheme for nodes beyond this zone, much like the ZRP, except that it defines its routing zone differently.

5.4 Routing Zone Formations

The single biggest problem with ad-hoc networks, is scalability. How does a single network with no fixed structure or management keep track of all the addressable nodes if the network size becomes very large? One node simply cannot store routing information concerning all other nodes in a big network, which is the main disadvantage to many of the routing protocols mentioned in Chapter 2. In these cases the memory requirements increase in direct proportion to the amount of nodes present in the network, which is not a realizable scenario.

The solution presented here is partly based on the zone routing protocol, but also draws inspiration from traditional wired networks, where a network is broken up into various subnets, with each subnet only being able to directly communicate with other nodes in the same subnet and these subnets interconnected by routers or gateways which perform address translation from one level to another.
This network subnet system, created using a wireless metric, much like the hop count that was used in the ZRP, should result in localised node clusters of a manageable size.

The wireless metric used to determine cluster sizes was simply the size of the cluster itself. Each node in the ad-hoc wireless network will only have space for X number of entries, thus each subnet may only have a maximum allowable size of X.

These subnets were dubbed LANs and the nodes on the edge LANedges. If a node wanted to join a LAN, but the LAN already has too many nodes, then the node would have to form its own LAN. Any new nodes starting up after and around this node could now join this new and still accessible LAN. Shown in Figure 5.1 is a network of 29 nodes and one possible LAN configuration (using a maximum size of 7 nodes per LAN).

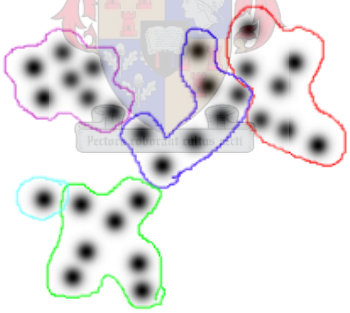


Figure 5.1: Possible LAN division in a 29 node network

The layout shown indicates that 5 LANs were created, each with a maximum size of 7 nodes. Inside each of these LANs the nodes are allowed to message and update one another directly, thus each node maintains a link to every other node in it's LAN in a proactive manner (see section 5.10 for more details).

Looking at Figure 5.2 which shows only one of these LANs we see that from node A's point of view the following can be seen of the network :

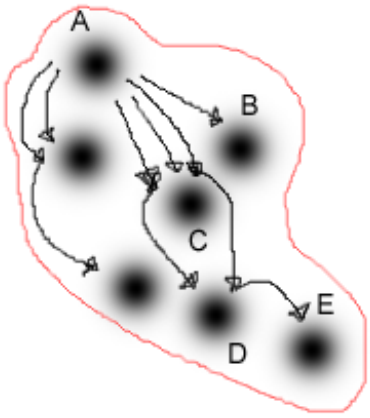


Figure 5.2: Inner LAN routing

```
node B is one hop away and can be reached directly
node C is one hop away and can be reached directly
node D is two hops away and can be reached through node C
node E is three hops away and can be reached through node C
etc..
```

Thus if node A is trying to reach node D or E it knows that if it sends a message to node C, node C will be able to get it to node D or E. All other nodes in the LAN can be reached directly with only one hop (one transmission).

In a similar fashion all the nodes belonging to each of these 5 LANs can now respectively see and reach all other nodes within their specific LANs. The nodes all send out periodic updates containing routing table information and thus keep each other's routing tables up to date. This process is described in more detail in Section 5.10.

But a network split into 5 seperate parts is of no use, links have to be formed between these nodal clusters to create one big network. These links are called 'LANedge links' and the process whereby these links are created, is discussed in Section 5.6.

5.5 Communication Between LANs

When two LANs exist in close proximity, the nodes will be within communications range of one another, enabling the setup and creation of proper communications channels between these two LANs. This Section takes a closer look at the various exchanges the LANs might have with one another.

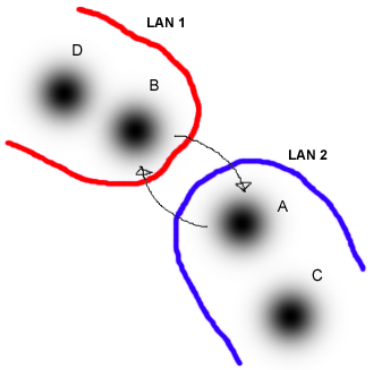


Figure 5.3: A 'LANedge link' between LANs

When a node from one LAN receives an update message from a node in another LAN, then it can use the information contained in that update to perform the following processing:

- create a LANedge link between two LANs (described in Section 5.6).
- validate an existing LANedge link entry (described in Section 5.7).
- recombine two LANs which would function optimally as one LAN (described in Section 5.8).
- removing a LANedge link if it is between a single node and a LAN, combining the two LANs into one LAN (described in Section 5.8).
- do nothing, if updated from a LAN linked to this LAN, but another node is already acting as the LANedge link node between the LANs.

These execution paths are shown in more detail in Figure 5.4.

5.6 LANedge Link Creation

To setup a LANedge successfully two nodes from different LANs need to be within communications range of one another. The process itself is a cross nomination process, where each node upon seeing that a link is possible, makes certain assumptions and tries them out for a while. If these assumptions are wrong the timers will expire, the temporary LANedge link entries created will be removed from the RTT and the node can try again.

Shown in Figure 5.5, is an ideal setup process, where no packets are lost, and no retransmissions occur. This same process was tested for heavy packet

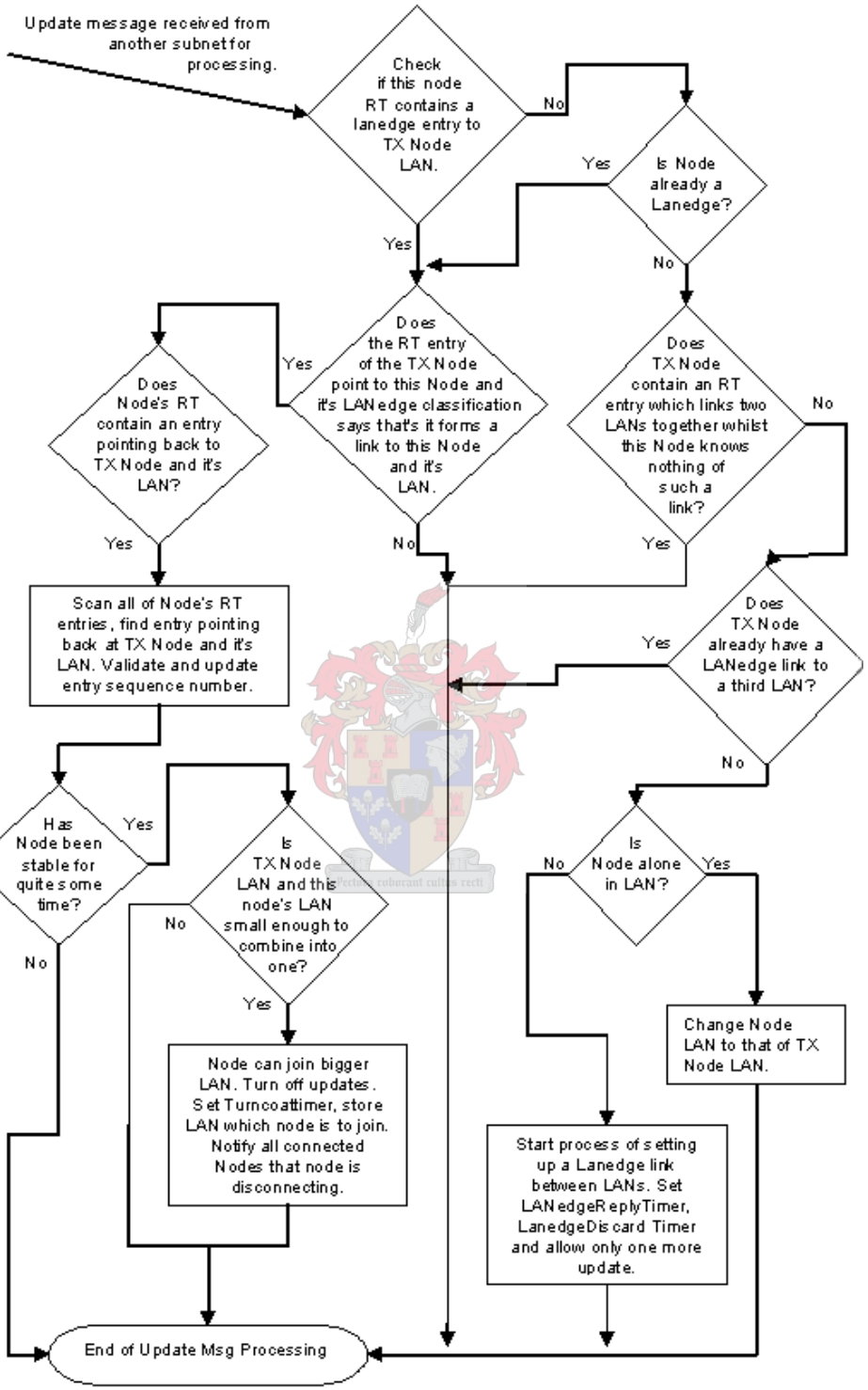


Figure 5.4: Processing of update from another LAN

loss circumstances and confirmed to still work perfectly. The time between nodes becoming aware of one another, and until the link is complete, just becomes much longer due to the many retransmissions.

In Figure 5.5, we see node B receiving an update message from an unknown node (node A), thus a node from another LAN. If node B is itself not already a LANedge, and there is no existing link between this LAN and the other LAN, then node A becomes a possible LANedge link node from node B's point of view.

Node B will now activate it's LANedge creation process, setting timers to govern when a LANedgereply should be sent, and another to act as a LANedge process timeout timer, cancelling the LANedge creation process if it does not succeed within a certain time period. Node B will also only send out one more update whilst the LANedge creation process is active, and then disable it's updates, in order to avoid inconsistencies should the process only complete halfway and then be cancelled.

When node B does send out it's next scheduled update node A will detect an unknown node and start the same process. Should this update have gotten lost, node A would not know about this, and will keep sending updates which node B will ignore, until a timeout occurs and node B cancels this entire process. Upon the next round of updates the entire process will repeat until a link is created.

As shown in Figure 5.5 the update message wasn't lost, and both nodes are now waiting for the LANedgereplytimer to expire. In this case node B will be the one who's timer will expire first, triggering it to send a LANedgereply message. Node A will create an RT entry pointing towards node B upon reception of the LANedgereply message, and update it's LANedge status to node B's LAN. When node A's LANedgereplytimer expires it will send a LANedgereply message to node B, and seeing as it has already received a reply from node B, will assume the LANedge link to be complete, finish the LANedge link creation process and update, as it would normally.

Node B will also received the reply from node A, assume the LANedge link to be complete, finish the lanedge link creation process and update as it would normally.

After a link has been created the update messages passed between them will be used to maintain this link. Each update message contains enough information to prove to the receiving node that the transmitting node has it's side of the link properly setup, enabling the receiving side to revalidate it's side of the link.

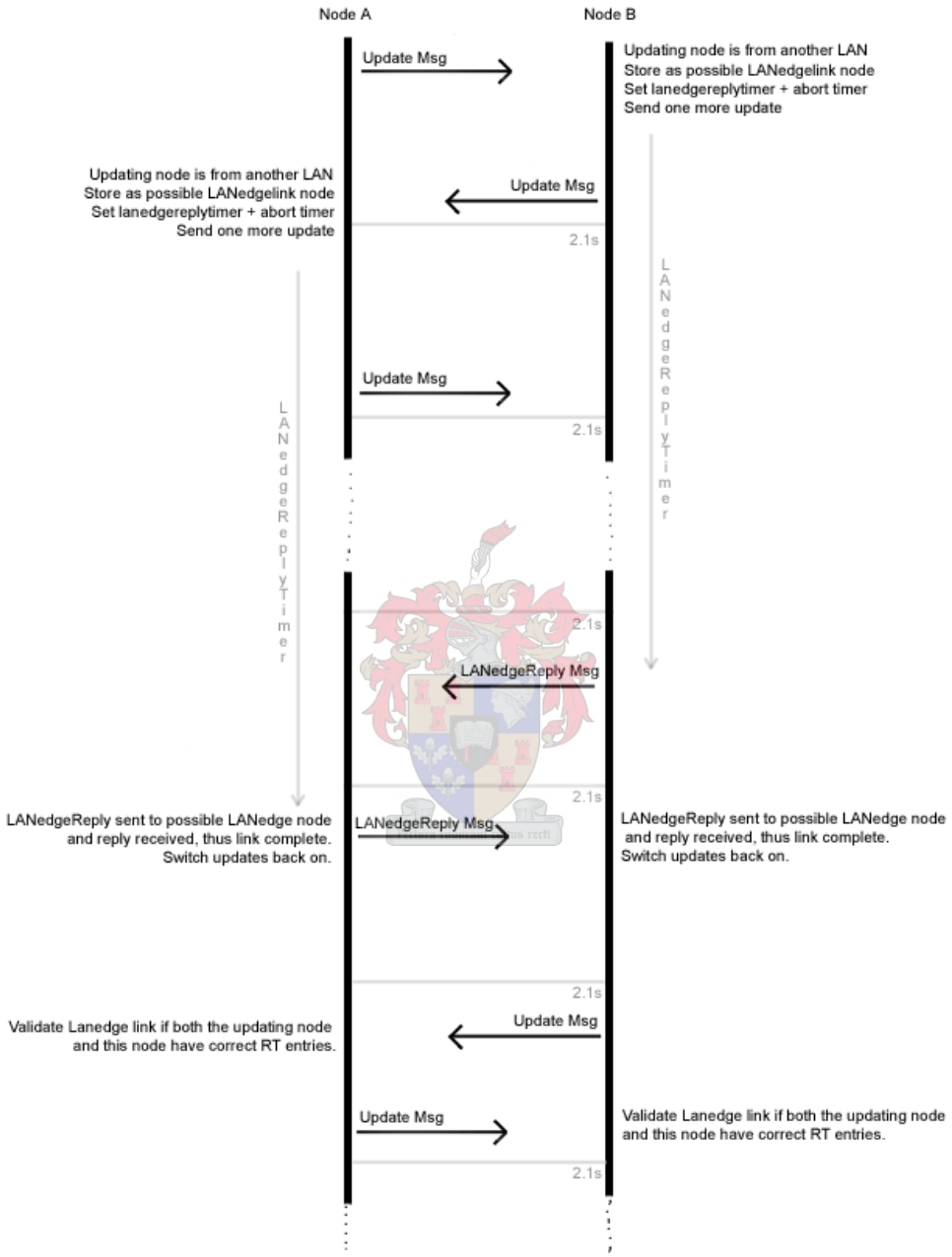


Figure 5.5: LANedge creation process

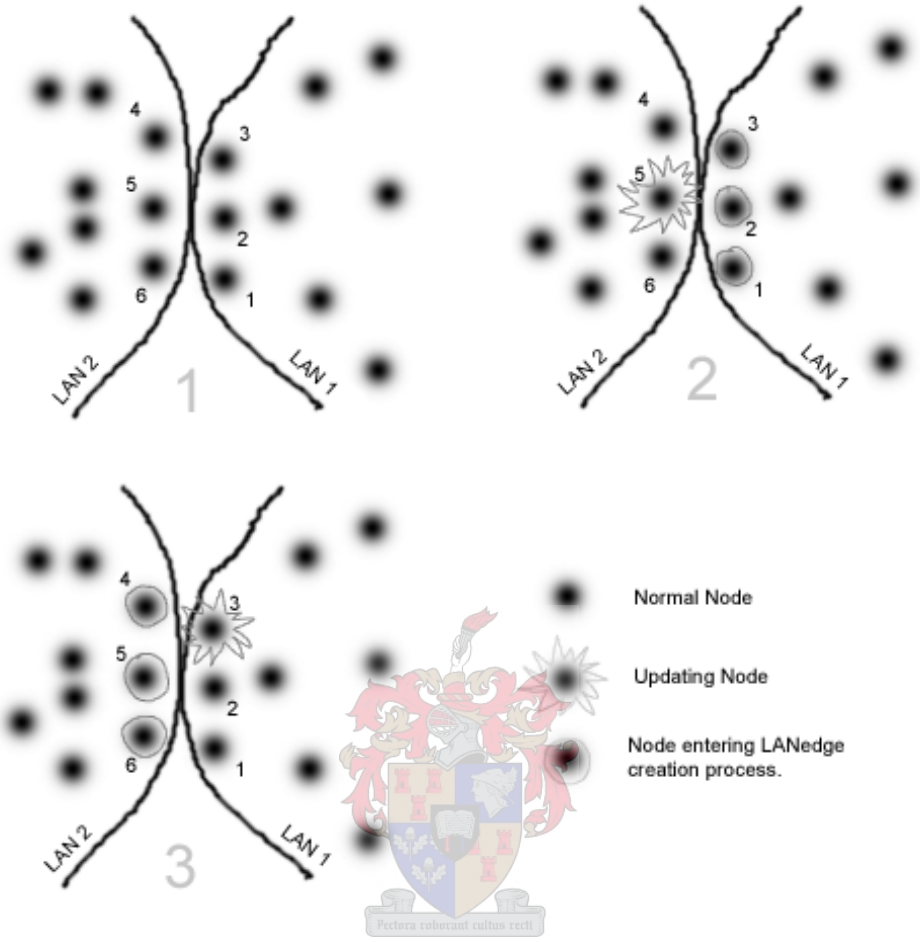


Figure 5.6: LANedge link creation - scenario 2

This process works under all conditions tested, even when there are a couple of nodes all trying to become the LANedge link node for their LAN at the same time, the process still ends up creating a proper link between the LANs due to the nomination process.

Looking at Figure 5.6 we see one such situation, where suddenly three nodes from each LAN can see 3 nodes in the other LAN, as shown in Block 1 of Figure 5.6. This situation is also resolved purely according to who sends out an update first. In Block 2 node 5 broadcasts an update, everyone hears this, it's own LAN nodes ignore it (in terms of LANedge creation process), but all three nodes from LAN 1 start the LANedge creation process. They are each allowed one more update, they have set the relevant timers and all three have assumed node 5 as the node with whom they want to create a LANedge link.

If either node 4 or 6 were to transmit an update now, it would be ignored by nodes 1,2 and 3 since they have already entered their LANedge creation processes. However when either node 1,2 or 3 sends an update the LANedge creation process continues. In Block 3 we see node 3 broadcasting an update. This update is ignored (in terms of the LANedge creation process) by nodes 1 and 2, but nodes 4,5 and 6 all assume node 3 as the node with whom to form a LANedge link and each starts the LANedge creation process.

LANedge reply messages will be sent back and forth between the nodes, as each tries to setup a link to the node it sees as the LANedge link node. However, all attempts to setup a link will fail, except for those attempts by nodes 3 and 5. They are the only pair of nodes whose assumed link nodes will compare correctly (node 5 assumed node 3 as link node, node 3 assumed node 5 as linked node). All other nodes will get it wrong and the entries will expire (if node 2 transmits to node 5 that it wants to form a link, node 5 will not be interested since it has assumed node 3 will be the link node and is waiting to hear from node 3), allowing them to leave the LANedge process and update normally again.

5.7 LANedge Link Maintenance

The LANedge links created are maintained by the periodic update broadcasts that all nodes send and receive. Once a LANedge link is setup between two LANs, all nodes, other than those involved in the LANedge link directly, will ignore update messages coming from nodes in the LAN their LAN is already linked to.

The LANedge link nodes can only validate an entry if the update message is from the node forming the other half of the LANedge link. In this case the receiving node scans the TX node information contained in the update message, and using this information checks that the LANedge link entry is setup correctly and valid at the TX node. It then checks that it has a corresponding entry for the same LANedge link, and if both tests succeed, the RX node will validate it's own entry.

5.8 LAN Recombination

LAN recombination can take place under the following circumstances:

1. a node is in a LAN by itself.
2. the link between two LANs has stabilized, and can now be reconfigured.

Scenario 1 occurs when a node is in a LAN by itself and it receives an update message from another LAN, it will then join that LAN if possible rather than setup a LANedge link between that LAN and itself. The second scenario only occurs after the link has stabilized, and will merge the smaller subnet into the bigger subnet if the combined size is smaller than the maximum size allowed. Thus if two LANs are of the same size, or the combined size is too big then the LANs will stay the same.

This feature was added to create a more optimal solution. Having one LAN instead of two smaller nodes will deliver better performance as the bottleneck at the LANedge link is avoided. And having two LANs of equal size should also result in a good average traffic flow between the two LANs, since the amount of internal and external traffic generated should then the same for both LANs.

5.9 Common Routing Problems

5.9.1 Route Invalidation Timers

To be able to identify routes which have become invalid or stale a node needs to be able to identify which nodes have been inactive for a long period of time. Thus each entry in the node RT gets a validity tag. Everytime a node broadcasts an update, the value of all it's RT entries get decreased by 1. When an update is received from another node, the RT entry corresponding to the updating node has it's validity set to the maximum again.

5.9.2 Split Horizon and Counting to Infinity

To avoid the occurrance of these rooting loops, two steps were implemented. Firstly all RT entries have a sequence number attached to them. This sequence number is assigned when an entry gets created and may only be updated (along with the RT entry) if an update with a larger sequence number is received. Thus each node increments its own sequence number before performing an update broadcast, thereby allowing receiving nodes to use the information transmitted. This means two nodes can't keep a false route alive by advertising it to each other contineously, simply because they cannot change the sequence number and can as a result, only revalidate a route using a specific update message once. Thus after using an update to validate a specific route, RT entries for that route will start expiring until a 'fresh' update is received from that route's source node itself.

Secondly, if we have three nodes in a row, A,B and C, then B gets an update from A directly and advertises this route to C. Node C may now only refresh the validity of the entry to the value advertised, not the maximum validity

for an RT entry which has just been confirmed. This means that validities for a specific route will decrease slightly in direct proportion to the distance from the route source node, but the time it takes for changes to propogate through the network is drastically reduced. Since the LAN size used is not that big, this disadvantage is acceptable.

5.10 LAN Maintenance

The preceeding sections have shown how the protocol uses a hybrid routing protocol to create routing zones (Section 5.4), how it creates and maintains links between these routing zones (Section 5.6), and how the protocol copes with common routing problems (in Section 5.9). In this section the internal LAN update process is examined in more detail.

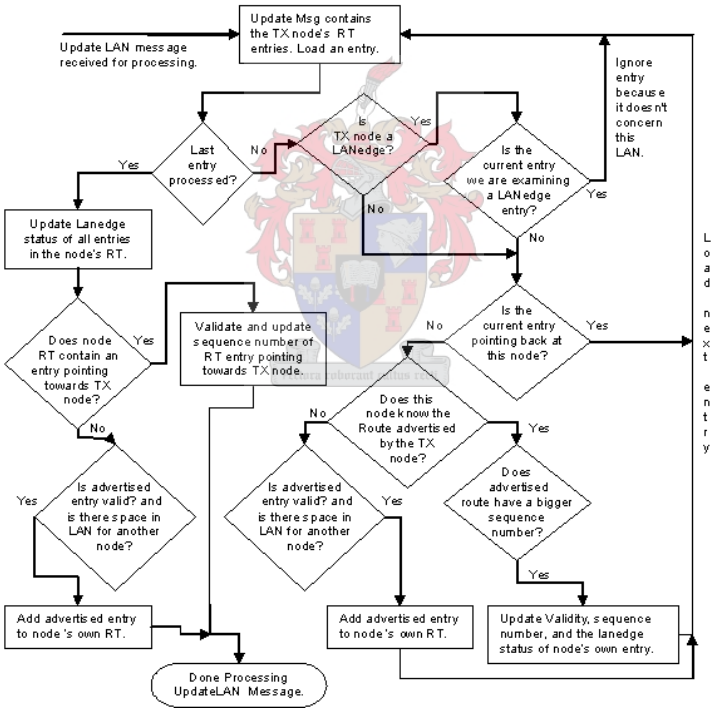


Figure 5.7: Update LAN

Each update message contains the updating node's entire RT (see Section 5.17 for the various message types and their corresponding packet structures) which gets processed one entry at a time by the receiving side. The

entire flow diagram (shown in Figure 5.7) relating to the update message processing procedure can be divided into two main halves, the left half describing what happens when all RT entries sent have been processed, and the right half what happens during processing of the RT entries sent.

In summary the following could occur during processing of an update message:

- If the RX node is a LANedge and the RT entry is a LANedge link entry to other LAN, then ignore it.
- If the RT entry being examined is the one pointing back from the TX node to the RX node then it can be ignored.
- A known entry can be validated.
- A new entry can be added.
- The update message could update the LANedge status of RT entry pointing to the TX node to reflect the current LANedge status of the TX node.
- The update message could add an entry to the RX node's RT pointing from the RX node to the TX node if it was lost somehow. This would then also update the sequence number and validity of the RX node's RT entry pointing to the TX node.

5.11 Medium Access Control Protocol

The Carrier Sense Multiple Access with Collision Avoidance (CSMA / CA) Control Protocol was used as Medium Access Control for the SWARM protocol. The transceivers will need some form of carrier sensing, but if they are all set to broadcast on the same frequency and can indicate when a message reception is detected, then when nothing is detected, the medium can be assumed free for transmission.

A further modification was incorporated though. Inspired by slotted ALOHA the total available transmission time (2.1s - updates are sent at least once every 2.1s) is divided up into blocks, each of these blocks being long enough to accommodate a reception and a transmission. To avoid two nodes broadcasting at the same time a random transmission block is chosen between 1 and 3 from the current block. A FIFO queue was also implemented to buffer any transmissions which may be stalled because of this. This entire CSMA/CA implementation is discussed in detail in Chapter 10, Section 10.3.

All transmissions (except for ones where loss is acceptable) have timers associated with them, and if the timer expires (eg. if no reply was received) then a collision is assumed and a retransmission scheduled. Each retransmission also gets a random time tag associated with it to avoid two nodes retransmitting at the same time.

5.12 Route Acquisition

The route acquisition process is performed reactively, which adds a slight delay initially when messaging a node not recently messaged. Once a route has been found successfully though, it is stored in the node's request routing table and can simply be reused thereafter. All routes do have expiry timers though, the duration of which depends on the mobility of the node.

When a node requests a data transmission, the source and destination nodal pair is compared to the RTT contents, and if a valid entry is found the data message is sent to the first hop along the route. However, if nothing is found then a scan is done on the node's RT to see if the destination is inside the LAN, or whether it belongs to another LAN. A Route Request (RREQ) message is then sent directly to the destination node if the destination node belongs to this LAN, else a RREQ message is sent to all LANedge links known.

At each foreign half of these LANedge link nodes a scan is performed. If the destination is found to be local to that LAN, the RREQ message is sent directly towards the destination, else the RREQ message is forwarded to all LANedge links known.

For example, in Figure 5.8, we see that the source node belongs to LAN 3 and the destination does not, thus the RREQ message needs to be forwarded to all the LANedge links out of this LAN. The RREQ message is then forwarded to the LANedge link node connecting LAN 3 and LAN 4, as well as the LANedge connecting LAN 3 and LAN 2. The LAN 4 LANedge link node did not forward the RREQ anywhere, since LAN 4 does not contain the destination node and does not contain any other LANedge links either.

At the LAN 2 LANedge link node however, as at every LANedge link node when entering a new LAN, the LANedge will perform a scan for the destination node, realize the destination node does not belong to this LAN, and forward the RREQ message to any available LANedge links, in this case the link to LAN 1. At the LAN 1 LANedge link node, the same scan is

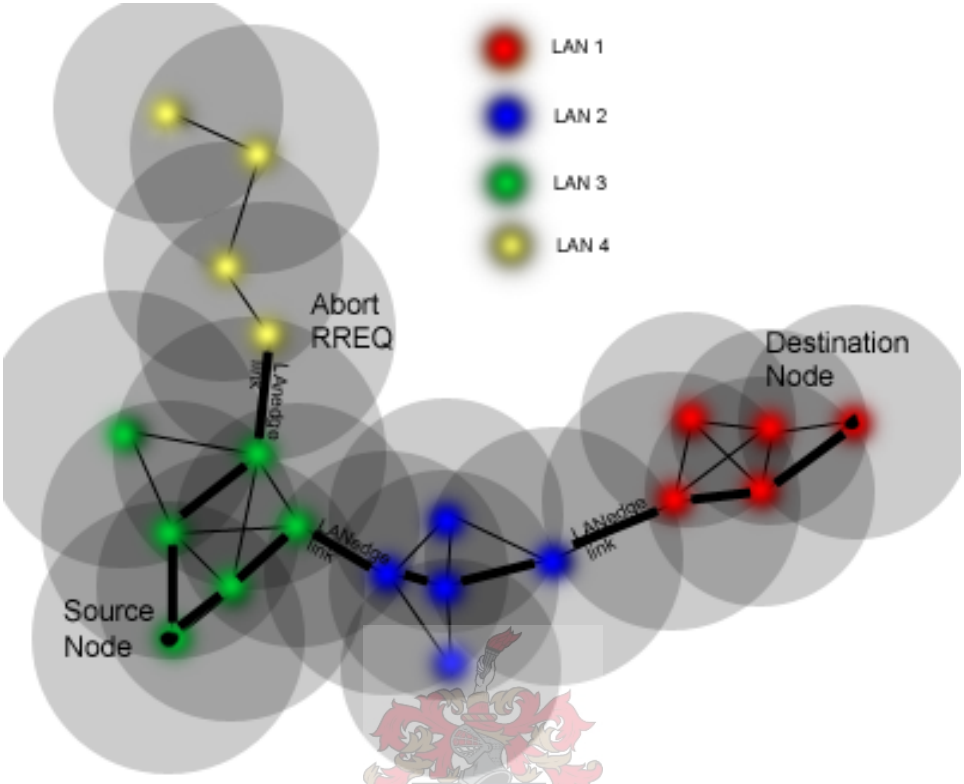


Figure 5.8: Route acquisition - finding the destination

conducted, however this time, the destination node is found to be local and the RREQ message is sent directly to the next hop along the route to the destination node.

When the RREQ reaches it's destination a Route Request Reply (RREQReply) message is generated, and sent back along the temporary route left behind by the RREQ message on it's journey from source to destination. When a RREQReply message traverses the temporary route created by a RREQ message; the route is confirmed as being valid, becomes validated and properly entered into the RTT.

When two routes to a destination node exist, two RREQReply messages will be returned, but only the shortest route will be used for data transfers. Another potential problem occurs when a RREQ gets forwarded from one LANedge link to all the LANedge links out of that LAN, and then they forward it to their LANedge links again, and so forth, then a loop can easily be formed which will choke the network with broadcast packets. For example

a RREQ message can be forwarded from LAN A to LAN B to LAN C and back to LAN A again, just to restart the loop again.

To combat this a TTL tag could be attached to every RREQ message broadcasted, but instead the RREQ forwarding process was setup in such a way that a LANedge link node cannot create two temporary routes for the same source and destination nodal pair. It simply doesn't forward the RREQ message any further. Thus preventing the broadcasting to carry on indefinitely.

To deal with packet loss a *RequestSentTimer* is set when the RREQ gets transmitted, if no RREQReply message is received and the timer has expired the route acquisition is assumed to have failed, the temporary entries created removed, and the route acquisition process will restart with a retransmission of the RREQ message. The number of retransmissions which may take place before a destination is regarded as unreachable is set by the constant *RetryTotal*. When this number of retransmissions have occurred the route acquisition process is abandoned and the user informed that the destination node is unreachable.

5.13 Data Route Maintenance

Data routes are temporarily created by a RREQ and confirmed by a RREQReply. Once stored in the RTT a route can either be validated or removed. A route becomes validated everytime it is used to transmit data succesfully.

A route is removed when either of the following takes place:

- the route has not been used to transmit data in a long time. Such a stale route will be removed due to it's low validity rating.
- when a data transmission fails (see Section 5.14) the route it was using is instantly removed. Any retransmissions are then forced to restart the route acquisition process and seek out a new route.

5.14 Route Error Detection

All critical message types have associated timers which get set when a transmission occurs. These timers expire when a message is sent and an expected reply is not received within a specified period of time. These message types include the LANedge creation messages (process described previously in Section 5.6), the route request (process described previously in Section 5.12), and the data messages.

When data messages are sent a DataAck message is expected back from the destination node upon successfull reception of the Data message, and if no DataAck message is received the data packet is assumed lost or the destination unreachable. A retransmission will then be performed, but if a certain number of retransmissions are lost, the destination is regarded as unreachable.

5.15 Node Identification

Each node is identified using a 8 bit ID. This is programmed into the firmware at manufacture. The ID indicates whether a node will be a messaging node or a relay node. All ID's with a decimal value of smaller than 120 indicate a messaging node, and those above 120 indicate that the node is a PowerRelay node.

PowerRelay nodes do not take any form of user input directly and can only be controlled from another user terminal. Messaging nodes are meant to be connected to a computer terminal and can be used to transmit and receive text messages, or set the state of connected PowerRelay nodes which may in turn be used to switch a bigger auxillary load.

5.16 Routing Table Structure

Each node's routing table has the structure shown in Table 5.16.1. The core

Number of node RT entries	8 bits
Node sequence number	8 bits
LAN node belongs to	8 bits
Node LANedge status	8 bits
n RT entries	n x 8 bits
Total Bytes Required	(4+n) Bytes

Table 5.16.1: Node RT structure

of the RT consists of n number of route entries, each having the structure shown below in Table 5.16.2. The node's RTT is a simpler structure, containing only a list of RT entries (shown in Table 5.16.2), one entry for each route acquired. The node RTT is shown in Table 5.16.3.

Source node	8 bits
Destination node	8 bits
Next hop	8 bits
Previous hop	8 bits
HOP total for route	8 bits
Entry sequence number	8 bits
Entry LANedge status	8 bits
Entry validity	8 bits
Total Bytes Required	8 Bytes

Table 5.16.2: RT entry structure

Number of node RT entries	8 bits
n RTT entries	n x 8 bits
Total Bytes Required	(1+n) Bytes

Table 5.16.3: Node RTT

5.17 Message Packet Structures

The RREQ messages used during route acquisition have the structure shown in Table 5.17.1. The LANedge_link_node field is used to indicate whether the RREQ is being sent directly to the final destination node or whether it is on the way to a LANedge link node. This field will be set to NULL if the destination is local, else it will contain the name of the LANedge link node out of this LAN, or the other half of the LANedge link nodal pair of nodes between LANs if the one LANedge has already been reached. Once a RREQ reaches it's destination, a RREQReply is generated and sent back along the route created. The packet structure for a RREQReply is shown in Table 5.17.2. It is very similar to a RREQ, yet simpler since it only stores which route request process it is part of, what node is the next hop, and what the exact hop count from the source to this node is (which nodes along the route store and decrementing before passing the message on to the next hop along the route).

The Data packet has a similar structure to the RREQ and RREQReply, it also only needs to contain enough overhead information to be able to get from the source to the destination one hop at a time. The data packet also contains 16 Bytes data payload though (limited by the hardware, see Chapter 8). The structure is shown in Table 5.17.3 .

Msg_Type_Id	8 bits
Node_Id	8 bits
Node_Seq	8 bits
Node_LAN	8 bits
Node_LANedge	8 bits
Node_numberofRTentries	8 bits
Next_Node	8 bits
Requested_Destination_Node	8 bits
Request_Source_Node	8 bits
Sequence_number	8 bits
LANedge_link_node	8 bits
HopCount	8 bits
Total Bytes Required	Bytes

Table 5.17.1: RouteRequestMsg

Msg_Type_Id	8 bits
Node_Id	8 bits
Node_Seq	8 bits
Node_LAN	8 bits
Node_LANedge	8 bits
Node_numberofRTentries	8 bits
Next_Node	8 bits
Requested_Destination_Node	8 bits
Request_Source_Node	8 bits
Sequence_number	8 bits
HopCount	8 bits
Total Bytes Required	11 Bytes

Table 5.17.2: RouteRequestReplyMsg

Msg_Type_Id	8 bits
Node_Id	8 bits
Node_Seq	8 bits
Node_LAN	8 bits
Node_LANedge	8 bits
Node_numberofRTentries	8 bits
Next_Node	8 bits
Data_Destination_Node	8 bits
Data_Source_Node	8 bits
Data	128 bits
Total Bytes Required	27 Bytes

Table 5.17.3: Data Message

Msg_Type_Id	8 bits
Node_Id	8 bits
Node_Seq	8 bits
Node_LAN	8 bits
Node_LANedge	8 bits
Node_numberofRTentries	8 bits
Next_Node	8 bits
Data_Destination_Node	8 bits
Data_Source_Node	8 bits
DataAckSequence_number	8 bits
Total Bytes Required	10 Bytes

Table 5.17.4: DataAck Message

The periodic update messages contain all the node's vital information, as well it's entire RT. The message structure used throughout the simulations and protocol development is shown in Table 5.17.5.

This structure was slightly adapted with firmware development as described in Chapter 10, to accommodate some hardware restrictions (discussed in Section 10.2). The variable n is determined by hardware restrictions, albeit memory limitations or transmission packet size restrictions.

Msg_Type_Id	8 bits
Node_Id	8 bits
Node_Seq	8 bits
Node_LAN	8 bits
Node_LANedge	8 bits
Node_numberofRTentries	8 bits
(n) SOURCE	8 bits
(n) DESTINATION	8 bits
(n) NEXT	8 bits
(n) PREVIOUS	8 bits
(n) HOPS	4 bits
(n) SEQUENCE	4 bits
(n) LANEDGE	8 bits
(n) VALIDITY	8 bits
(n+1) SOURCE	8 bits
(n+1) DESTINATION	8 bits
(n+1) NEXT	8 bits
(n+1) PREVIOUS	8 bits
(n+1) HOPS	4 bits
(n+1) SEQUENCE	4 bits
(n+1) LANEDGE	8 bits
(n+1) VALIDITY	8 bits
⋮	⋮
Total Bytes Required	(6+(nx8)) Bytes

Table 5.17.5: Update Message

Chapter 6

Protocol Simulation

6.1 Overview

This chapter follows the development and testing of the two simulators used throughout the protocol design. It also presents results from simulations performed using the second simulator for comparison to the hardware results presented in Chapter 12.

6.2 Simulation 1

The simulation was created to provide the user with a tool with which to effortlessly create a network layout upon which a routing protocol can be tested. Thus a graphical user interface was created, providing the user with a graphical way of displaying a network layout. The routing information for a specific node can also be displayed visually which simplifies the testing and verification of a network.

To provide such a convenient test vehicle for routing protocol testing in a mesh network environment, means that the user has to be able to specifically control the connections between nodes, which means being able to determine exactly which nodes are in range of which nodes.

To ensure that the routing protocol is functioning correctly the user should be able to pause the automatic updates system, manually send updates from specific nodes, and be able to directly access the information currently contained in the routing tables of the nodes.

6.2.1 Graphical Interface Development

The simulation software was developed in C++ using Dev Cpp as develop

ment environment. C++ is a powerful and popular programming language with widespread popularity.

All graphical components of the GUI are directly based on the OpenGL graphics library. The OpenGL based GLUT and GluPlot were used to display and plot the network nodes. The cross-platform OpenGL Utility Toolkit (GLUT) simply provides a context, or a window, in which OpenGL objects can be drawn. The GluPlot library is an OpenGL graph drawing software library which provides many usefull wrapper functions for plotting data without having in-depth knowledge of the underlying OpenGL. For a detailed description of the GLUT API or GluPlot libraries and their abilities please refer to [17] and [18] respectively.

6.2.2 Simulation Characteristics

- Allow manual graphical placement of nodes, also showing the RF range of each node. This allows the user to create very detailed and specific networks by exactly controlling which nodes can see which other nodes. These specifically created network layouts can then be used to test certain routing protocol properties.
- Allow the user to select a specific node and view its RT content. Also indicate graphically which other nodes belong to the same LAN as the selected node.
- Graphically indicate the nodes acting as LANedge links from one LAN to another LAN.
- Allow the user to manually send out an update from a selected node.
- Allow the user to set the system active (all nodes send out updates periodically), and inactive (all nodes become passive and remain as they are).
- Allow the user to immediately remove a specific node.
- Allow the user to select a source and destination node. A route is then sought and displayed graphically.

6.2.3 Simulation Functionality

Each node is represented by a red dot with a specific x,y coordinate, and each node has a certain radius from this x,y coordinate within which it can communicate with other nodes. This radius is displayed as a circle of blue dots around the selected node.

The user has the ability to place nodes anywhere within the network layout screen manually be clicking where the node should be placed, or automatically by letting the computer choose a random x,y coordinate for the node by pressing 'a'. The maximum size of the network is determined by a constant variable, *const int coordindex = 50;* which is currently set to 50, but could be made much higher. The user cannot add more nodes than the number specified by this variable.

The user can select the node they are currently interested in by scanning through all the nodes in the network by pressing 's'. This selects the nodes one by one, allowing the user to stop upon reaching the node interested in.

All nodes forming part of the selected node's LAN are shown as either yellow or green dots when the user requests to view the selected node's RT by pressing 'r'. The green dots represent nodes directly within reach and the yellow dots represent nodes which are reacheable through one of the neighbouring nodes. Red dots represent nodes which belong to a different, possibly unreachable LAN. An example network layout is shown in Figure 6.1. When two LANs exist in close proximity to one another a LANedge link

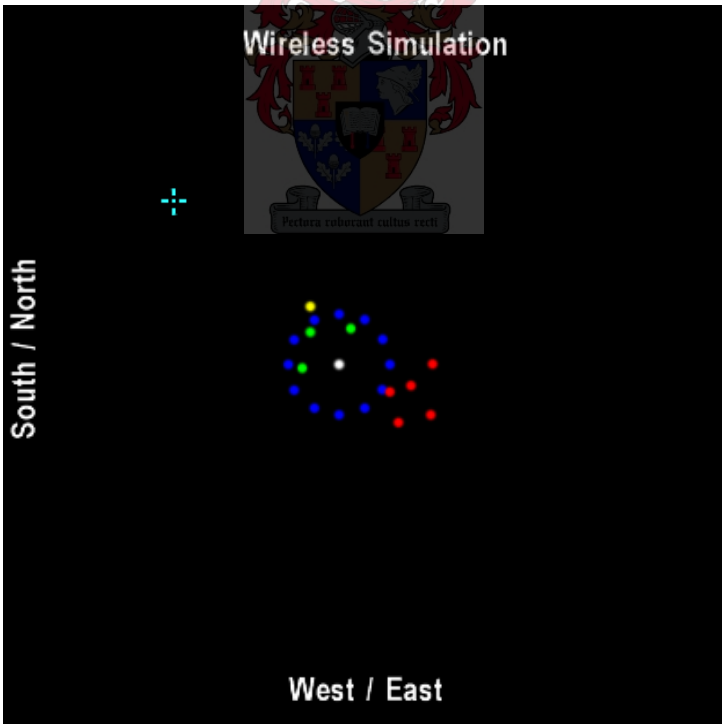


Figure 6.1: Selected node's LAN

would be created between them, with a LANedge link node being indicated as a purple dot, as can be seen in Figure 6.2. Typically the user would add

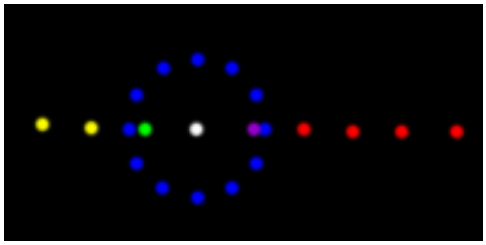


Figure 6.2: LANedge link between LANs

all the nodes into the network layout desired, and then set the system active. The nodes which have already sent out hello and received reply hello messages upon startup will then automatically share update information and structure their LANs accordingly.

The user is also able to scan between all existing LANs, swopping from one to the other each time the user presses 'l'. When a node from a specific LAN is selected all nodes belonging to that LAN are highlighted. An example showing three isolated LANs is shown in Figure 6.3. The user can

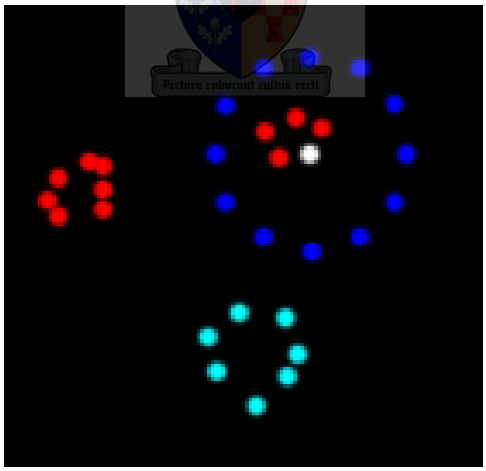


Figure 6.3: Displaying various LANs

also select a source and a destination node in order for a route request to be sent out. This results in all the nodes belonging to the route found to become highlighted in white. An example is shown in Figure 6.4.

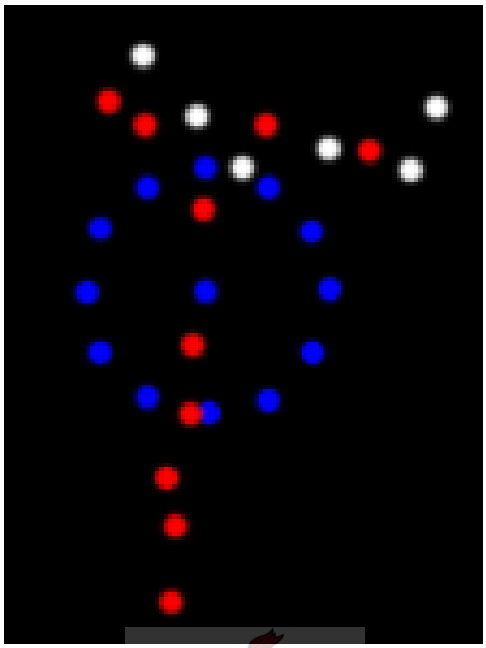


Figure 6.4: Route acquisition between source and destination

6.2.4 Simulation Shortcomings

Even though the simulation provided a clear way of illustrating the network and developing basic routing protocol rules, it had serious fundamental limitations in its messaging framework.

A networking model more closely resembling the real world was needed to effectively build a simulator from which the source code could be more directly translated into firmware code. Thus, existing client server models were researched, but most were found inadequate. A client server model is needed which can accept many concurrent connections, and yet must still be able to easily pass a message from one connection (node) to multiple connections (nodes), or none, depending on who is within 'transmission range'. Range once again being defined as an abstract quantity. Every node is assigned an x,y coordinate, and two nodes are in range if they are within a certain radius from one another.

It was decided to use a form of multicasting as message transmission medium since this closely resembles the air as transmission medium. All nodes will be given location coordinates to determine who is in range of who, but other than that the simulator code can be created in such a way that it is 80-90% directly translatable to firmware. This is discussed in the next section.

6.3 Simulation 2

6.3.1 Overview

The second simulator has exactly the same functionality as the first simulator, but it also adds extra functionality. The first simulator was limited in that it was a single executable program, with no 'real' messages being passed between nodes, just data manipulation within the variables of a program. The second simulator was built on top of the TCP / IP stack, and encapsulated it's messages within TCP packets. Thus nodes could be run on several computers all within the same network and nodes would still be able to communicate.

This second simulation was able to do this since it was built using the JAVA multicasting framework. The idea was based upon the *DeitelMessenger* chat room application developed as a networking tutorial in *Java How to Program, the fourth edition*[19]. The core of the simulator is a client server based model, but the role of the server is to simply connect all the clients together, and multicast messages received from one client to all other clients.

Two Java programs were thus created, *Client.java* and *Server.java*. In every simulation one instance of the server would be created, and one client instance per node in the network. The server process listens for client connections on a specific port, creating a thread for each new client which connects, as shown in Figure 6.5.

```
Server listening on port 5005 ...
Monitoring socket : Socket[addr=/127.0.0.1,port=57735,localport=5005] for incoming client messages
Connection received from: /127.0.0.1
a1 added to Server list : 160 - 141
Monitoring socket : Socket[addr=/127.0.0.1,port=59594,localport=5005] for incoming client messages
Connection received from: /127.0.0.1
a2 added to Server list : 165 - 152
Monitoring socket : Socket[addr=/127.0.0.1,port=40666,localport=5005] for incoming client messages
Connection received from: /127.0.0.1
a3 added to Server list : 176 - 155
Monitoring socket : Socket[addr=/127.0.0.1,port=41708,localport=5005] for incoming client messages
Connection received from: /127.0.0.1
a4 added to Server list : 181 - 143
Monitoring socket : Socket[addr=/127.0.0.1,port=49786,localport=5005] for incoming client messages
Connection received from: /127.0.0.1
a5 added to Server list : 170 - 134
```

Figure 6.5: Server listening for and accepting new client connections

At the startup of each client the user is asked for a Node Name, and provided an opportunity to indicate the node's position on the network layout map. Once started, the simulation interface offers the user with several different functions, all which will be explored in the following couple of subsections.

6.3.2 Network Connectivity

The network connectivity tab provides most of the user control functions, and is the main nodal interface the user is provided with upon startup. The interface is shown in Figure 6.6, and the functionality of the buttons

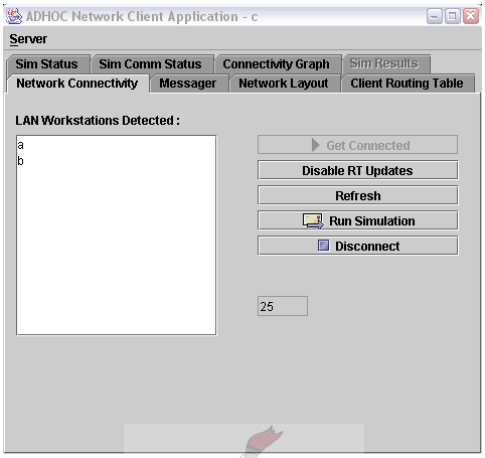


Figure 6.6: Network connectivity tab

described next. The tab also displays general information about the node such as how long the node has been active for, and provides a list of connected nodes (connected at time of last refresh).

Get Connected Button

Sets the node active, transmits a hello message and deactivates the button (greyed out). After activation the node is actively receiving messages when available and performing periodic updates.

Disable RT Updates

Disables the transmission of periodic updates. The button then changes to *Enable RT Updates*, and vice versa, thus it allows the user to toggle updates on and off.

Refresh

This will refresh the list of nodes shown on this tab to reflect all nodes currently reachable, based upon the current contents of the node RT.

Run Simulation

Starts the simulator. When the simulation is running whichever node has the transmit token (the transmit token is called the 'nodesend' token in the simulator) has the option of performing a transmission. When a node starts the simulator it has the transmit token by default.

The node with the token performs transmissions according to a Poisson

probability distribution function [20]. Thus once every second it will determine the probability of performing a transmission, and if the probability is good then the node will choose a random packet size, a random message destination and perform the data transmission. Upon completion of the data transmission the token is given up (as indicated with 'Simulation Running - Node Done' in Figure 6.7), randomly allocated to another node by the server, and then that node has the option of performing a transmission again. This process is shown in Figure 6.7 for node 'a' messaging node 'b'.

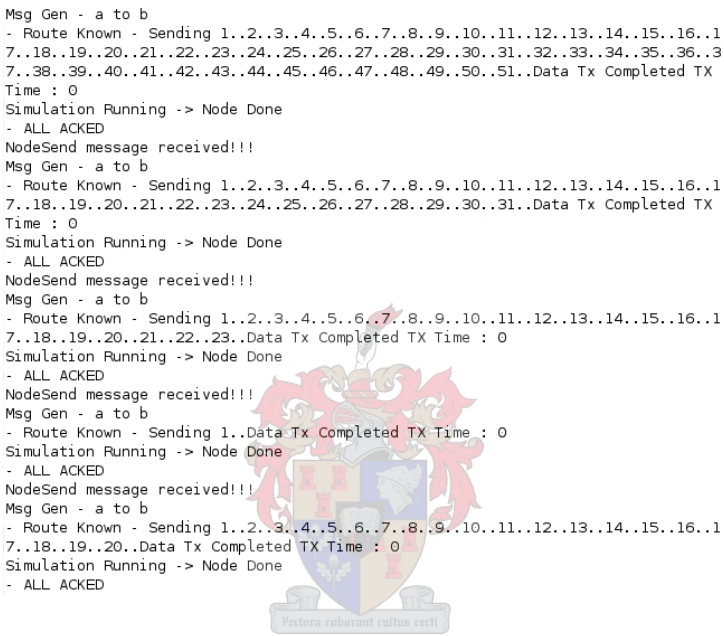


Figure 6.7: Random message size of transmissions

Disconnect
This will disconnect the node from the server and store all simulation results to a file.

6.3.3 Messenger

The messenger tab is used to enter user messages, choose a destination and transmit the messages. The tab allows the user to refresh the list of reachable network nodes, select one of these nodes, enter a message, and send it to the specified source. The message size does not have to be entered, but for testing purposes a large message size may be entered, which will then result in multiple packets being transmitted.

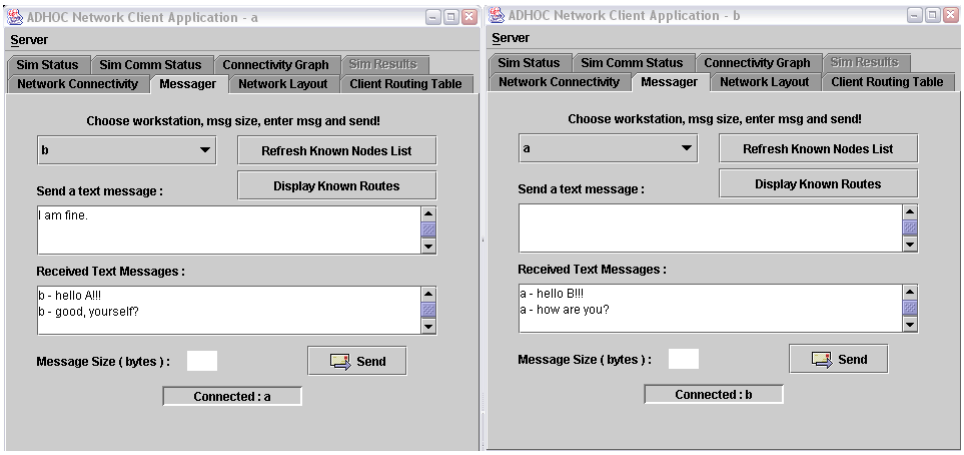


Figure 6.8: 2 Nodes messaging one another

6.3.4 Network Layout

The network layout tab shows the network map containing all currently known nodes plotted upon it. To update the nodes plotted click the refresh button. An example is shown below in Figure 6.9.

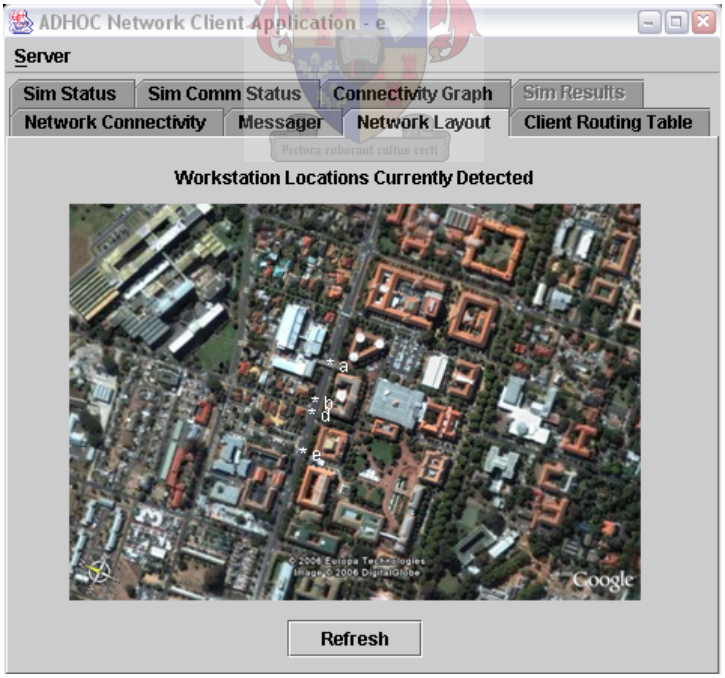


Figure 6.9: Current network layout

6.3.5 Client Routing Table

This tab is used to show the user the current contents of the routing table. To obtain the most up to date version, all the user has to do is click the refresh button. The tab displays general information like the node's X and Y location coordinates, the LAN the node belongs to, and whether the node is a LANedge link, at the top of the screen. The rest of the tab is taken up by a big message box which shows the rest of the routing table in a tabular form, as is shown in Figure 6.10. The information is displayed with one RT entry per line, and with dashes ('#') seperating the various fields.

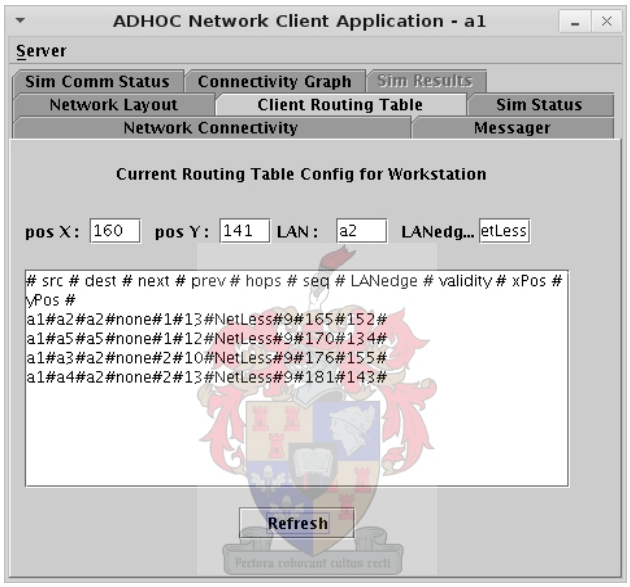


Figure 6.10: Node A1's RT

6.3.6 Sim Status

This tab shows a total of all messages sent and received by each node in the system since the simulation start.

6.3.7 Sim Comm Status

This tab shows a summary of the all the message types and how many of each specific type has been received by the node since the node was initialized. An example is shown in Figure 6.11 of a case where 3 nodes have just been activated, have said hello to one another, and have started updating each other.

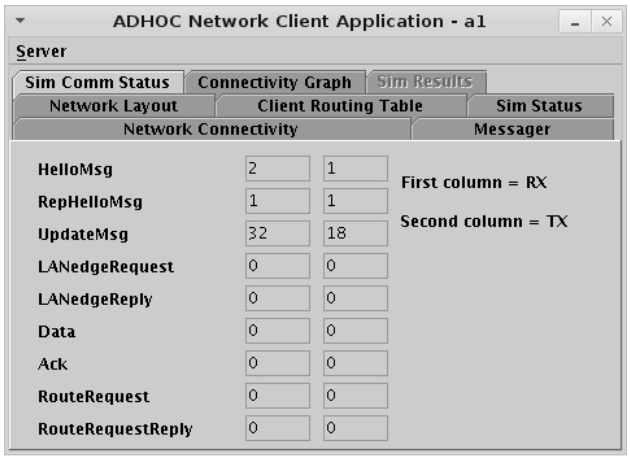


Figure 6.11: Simulation communication summary

6.3.8 Connectivity Graph

The connectivity graph uses data from the routing table to create a graph of the network layout from this node's perspective. It shows the node's immediate LAN, and the gateways from this LAN into surrounding LANs. Nodes not directly reachable (nodes in another LAN) are shown as red dots, nodes directly within reach as blue dots with blue lines joining them to the current node, and gateway nodes are shown as green dots and lines. An example of a 5 node single LAN connectivity diagram from the perspective of node A1 is shown in Figure 6.12 and from the perspective of node A3 in Figure 6.13. A very comprehensive example is illustrated in Figure 6.27 further on in this chapter.

6.4 Results of the Simulations

6.4.1 Overview

Simulation 1 was not developed further when development of Simulation 2 started, since they both have the same functionality. This means that Simulation 1 only has basic functionality and was only used in early stages of protocol development. All experiments were conducted using the refined and completed second simulator.

The experiments are divided into two sections, the first section merely demonstrates that the protocol functions correctly in terms of routing table setup, and route request acquisition given specific situations. The second section shows the performance of the protocol under heavy load.

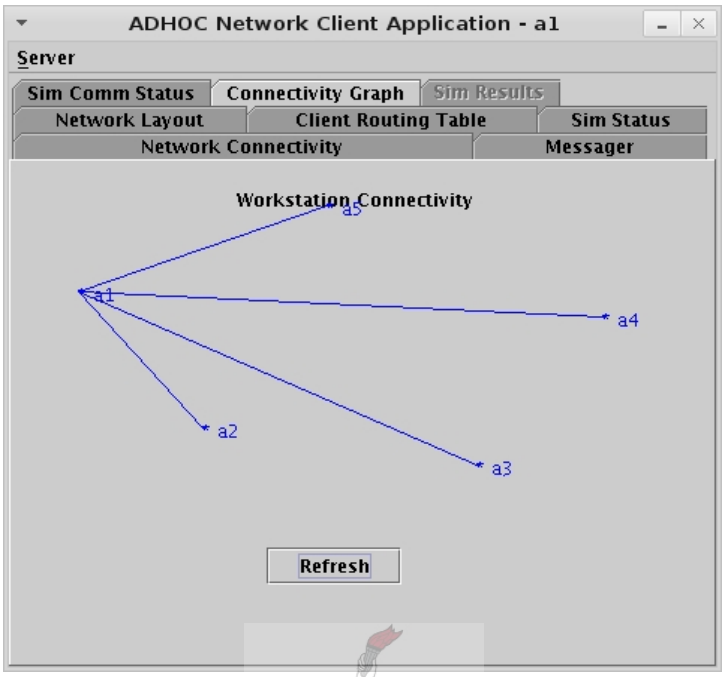


Figure 6.12: Node A1 connectivity diagram

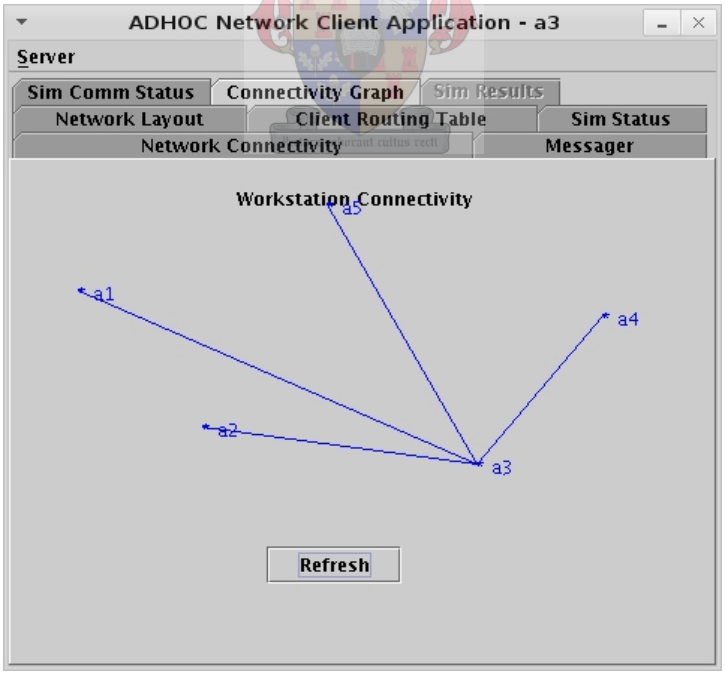


Figure 6.13: Node A3 connectivity diagram

6.4.2 Protocol Functionality Evaluation

Experiment 1 - RT Setup and Manipulation in Small Network Environments
Goals of this experiment:
1) Demonstrate the automatic creation of LANs.
2) Demonstrate the automatic creation of LANedge links between LANs.
3) Demonstrate the automatic reconfiguration and adaptability of remaining nodes after a disconnection occurs.

Initially seven nodes were activated. The way these nodes were located did not allow all nodes to be in contact with one another, thus three seperate LANs were created. The overall network layout, and node b2's perspective of the network is shown in Figure 6.14 .

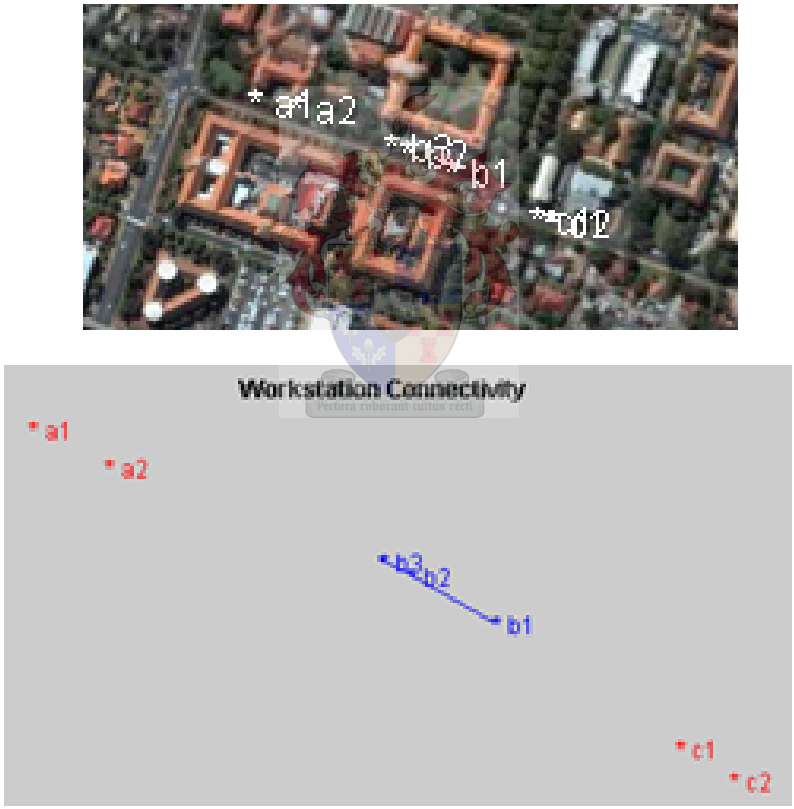


Figure 6.14: Network layout of 7 nodes forming 3 LANs

Figure 6.14 already illustrates the 3 LANs that were created, showing the middle LAN to which node b2 belongs. The other nodes can also each only

see those nodes in close proximity to them, whom currently form part of their respective LAN. The RT for node a2 is shown in Figure 6.15, from where we can see that the node belongs to LAN a2, is not a LANedge link node, and contains only one RT entry. This RT entry points to node a1, which is one hop hop away, is still valid (maximum validity for this experiment = 10 and it is currently 9) and the RT entry indicates that node a1 is not a LANedge link (LANedge = NetLess) .

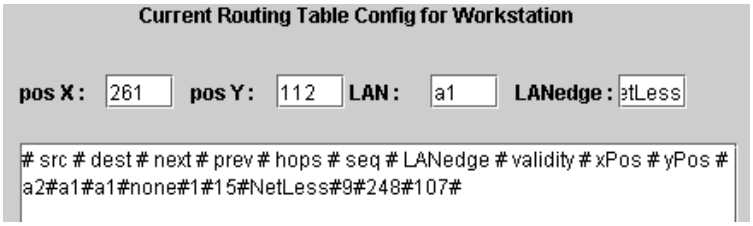


Figure 6.15: Node a2's RT contents

The RT for node b2 and the RT for node c1 are both shown together in Figure 6.16. The contents of these RT's is of a similar nature to the RT of node a1 and are as a result not discussed again .

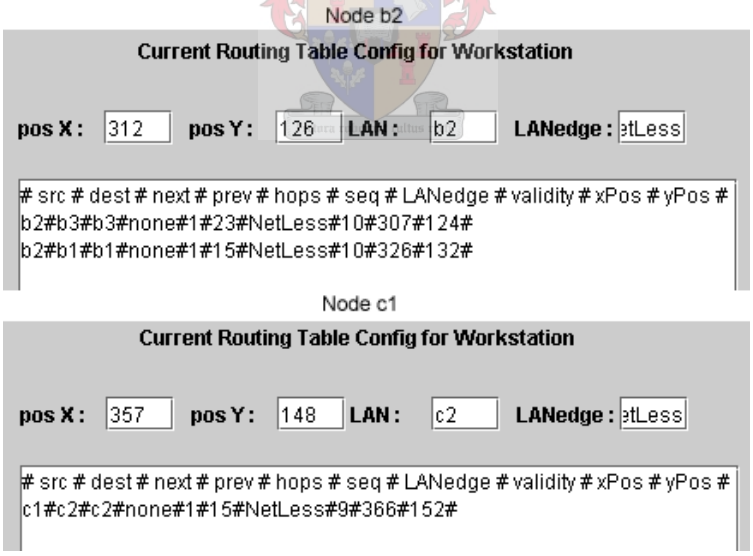


Figure 6.16: RT's of node b2 and c1

Two nodes, cnx and cnx2, were then inserted into the network layout at strategic positions. These node were located in such a way that they would

be in communications range from nodes belonging to more than one LAN. Thus they should be able to form a LANedge link between two LANs .

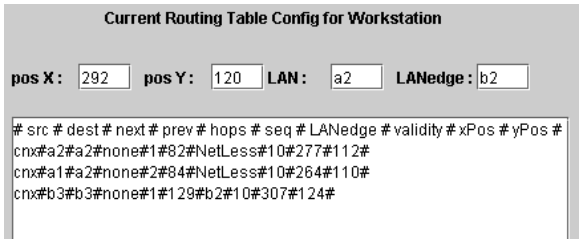


Figure 6.17: The RT of node cnx after LANedge link creation

Node cnx was added to the network layout, successfully joined LAN a2, and then started the LANedge creation process. This resulted in the successfull creation of a LANedge link between the two LANs, with node cnx and node b3 being the self appointed LANedge link nodes.

Figure 6.17 shows the RT of node cnx after the completion of the LANedge link between LAN a2 and b2. Looking at the RT, we see that node cnx has it's LANedge status set to 'b2' to indicate that it is the official LANedge link to LAN b2. Looking at the RT itself we can see that it still has the RT entries for all the other nodes belonging to LAN a2, namely node a1 and a2, but it also has an extra RT entry to node b3. This entry has a LANedge status of 'b2', indicating that it is the official entry for the LANedge link to LAN 'b2' .

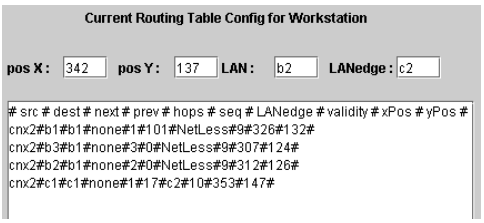


Figure 6.18: The RT of node cnx2 after LANedge link creation

When node cnx2 was added to the network layout, it successfully joined LAN b2, before starting the LANedge creation process. This resulted in the successfull creation of a LANedge link between the two LANs, with node cnx2 and node c2 being the self appointed LANedge link nodes. Figure 6.18 shows the RT of node cnx2 after the completion of the LANedge link between LAN b2 and c2. From this RT we see that node cnx2 has it's

LANedge status set to 'c2', still has all it's normal RT entries to all other nodes in LAN b2, and it has a LANedge link entry to node c1. This results in the connectivity diagrams shown in Figure 6.19.

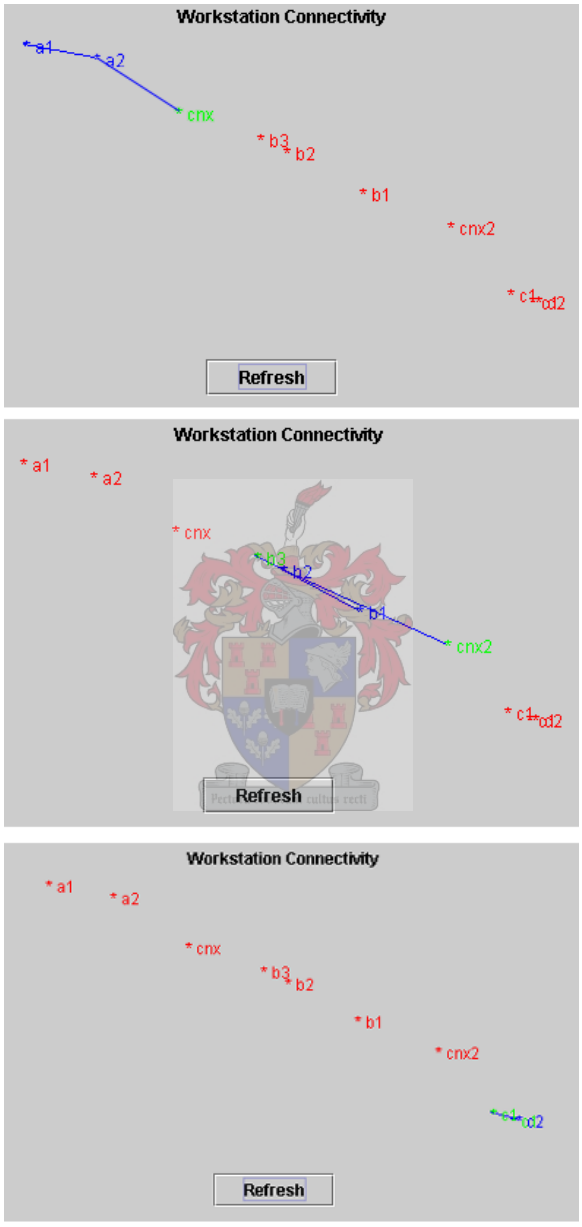


Figure 6.19: Connectivity of each of the 3 LANs after addition of LANedges

After the LANedges have formed between the LANs, the LANs will start re-combining as soon as the system becomes stable, Eg. when no new changes

are being made to the system. Thus we find that the nodes from LAN c2 start joining the bigger LAN, LAN b2. The result of this recombination process is shown in Figure 6.20.

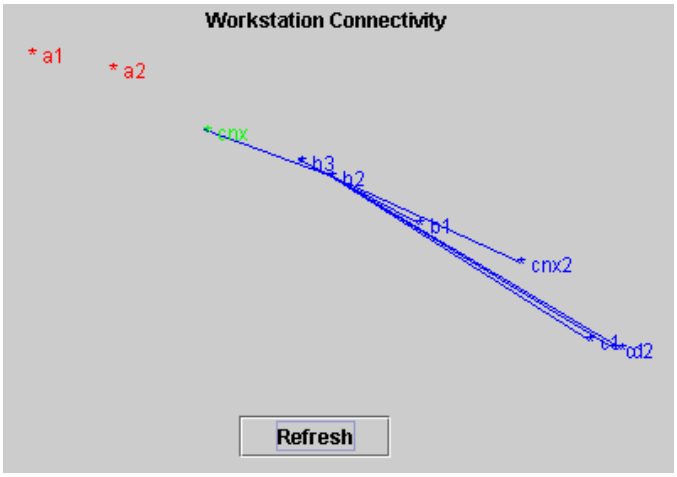


Figure 6.20: Node b2's connectivity diagram after recombination of LANs

The nodes from LAN a2 also want to join the bigger LAN, and would have, but the maximum LAN size for this experiment was set to 7, thus both smaller LANs can't join the bigger LAN. After this recombination process has completed the RT for node cnx has changed to that shown in Figure 6.21 .

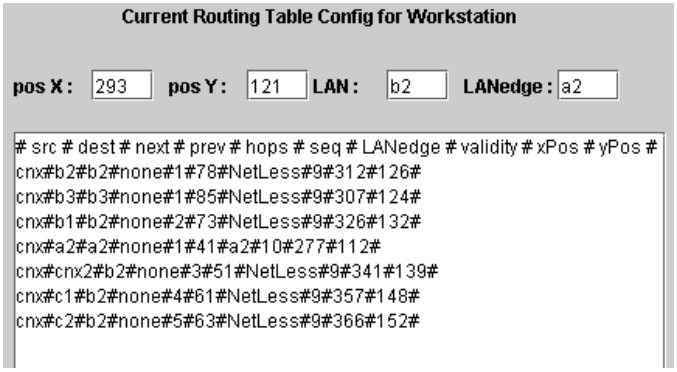


Figure 6.21: The RT for node cnx after recombination

In order to demonstrate the automatic reconfiguration and adaptability of remaining nodes after a disconnection occurs we have chosen to disconnect

node b1 as it forms quite a central node on this series of connected nodes.

When node b1 stops sending out periodic updates, the nodes that were linked to node b1 will keep lowering the validitiy of all entries they are no longer receiving updates for.

Looking at the RT of node b3, shown in Figure 6.22, we see that node b3 is still actively receiving updates from nodes b2 and cnx, but can no longer receive updates from any of the nodes on the far side of node b1, thus the validity of the RT entries pointing to any of these nodes is busy decreasing with every missed periodic update and they will soon expire .

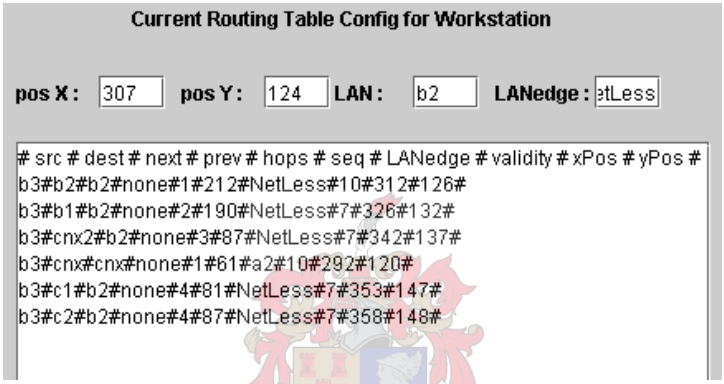


Figure 6.22: RT of node b3 after node b1 has disconnected

Once this disconnection has rippled through the nodes that were in any way connected to node b1, the remaining system will still undergo further changes, since several smaller LANs have now been created. The smaller LANs already have the LANedge links setup earlier, and will now recombine with the neighbouring bigger LANs.

Thus we find that the system stabilizes in the formation of two new LANs, LAN 1 combining nodes c1,c2 and cnx2, and the second LAN combining the rest of the nodes, as shown in Figure 6.23 .

The system will now stay like that until further changes are made, eg. more nodes are added, or removed from the system.

Experiment 2 - Automatic Configuration in a Large Network
In this experiment a large network of nodes was setup in order to illustrate that the protocol can successfully setup and create LANs and LANedges even if the network in question is much bigger. In total, 20 nodes are to be

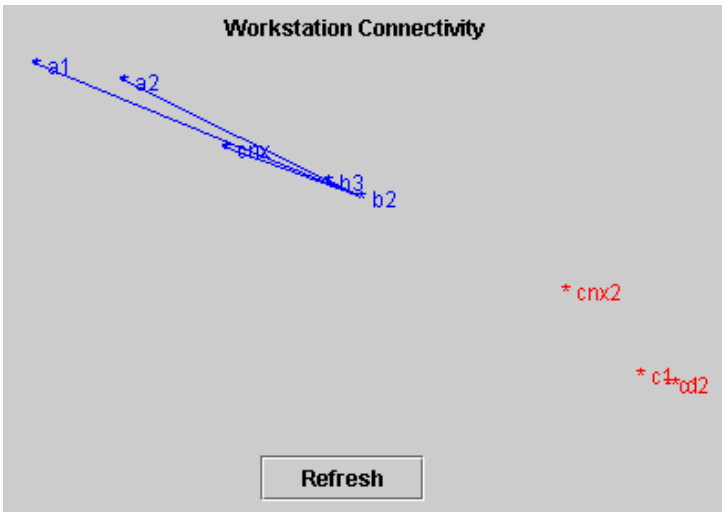


Figure 6.23: Node b2's network connectivity in newly stabilized system

used and the final network structured as is shown in Figure 6.24 .

Figure 6.24: Complete network layout

All the nodes except the nodes shown in red were put into position and activated. They then went through the activation process of initiating contact, setting up an RT, and thereafter sending out periodic updates to keep these RT's updated.

This resulted in the formation of 6 small LANs, all isolated from one another. This can be seen in Figure 6.25 which shows the network connectivity diagram as seen from node a5's perspective.

All the nodes that were marked as red in Figure 12.8 were then added to the network layout and activated. They immediately joined one of the two LANs in close proximity to them and started the LANedge creation process with the second nearby LAN.

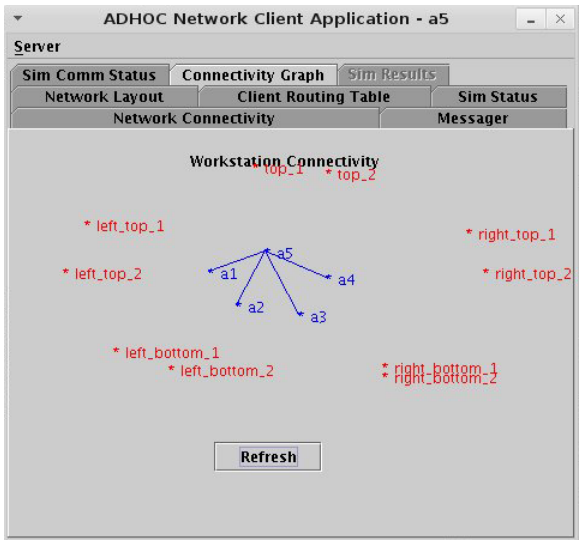


Figure 6.25: Middle LAN

This resulted in the formation of 5 LANedge links, with one LANedge link being created between the middle and each of the outer LANs. This is clearly illustrated in Figure 6.26 which shows the network connectivity from the perspective of a couple of the LANedge link nodes.

Thus the self nominated LANedge link pairs are:
1) left_top_link and a1
2) top_link and a5
3) right_top_link and a4
4) left_bottom_link and a2
5) right_bottom_link and a3

The initial setup doesn't last very long though before the recombination process starts again. Looking at the network from the perspective of node a4 (Figure 6.27) a little while later, the overall network is still intact, but the specific nodes which form the LANedge links have changed.

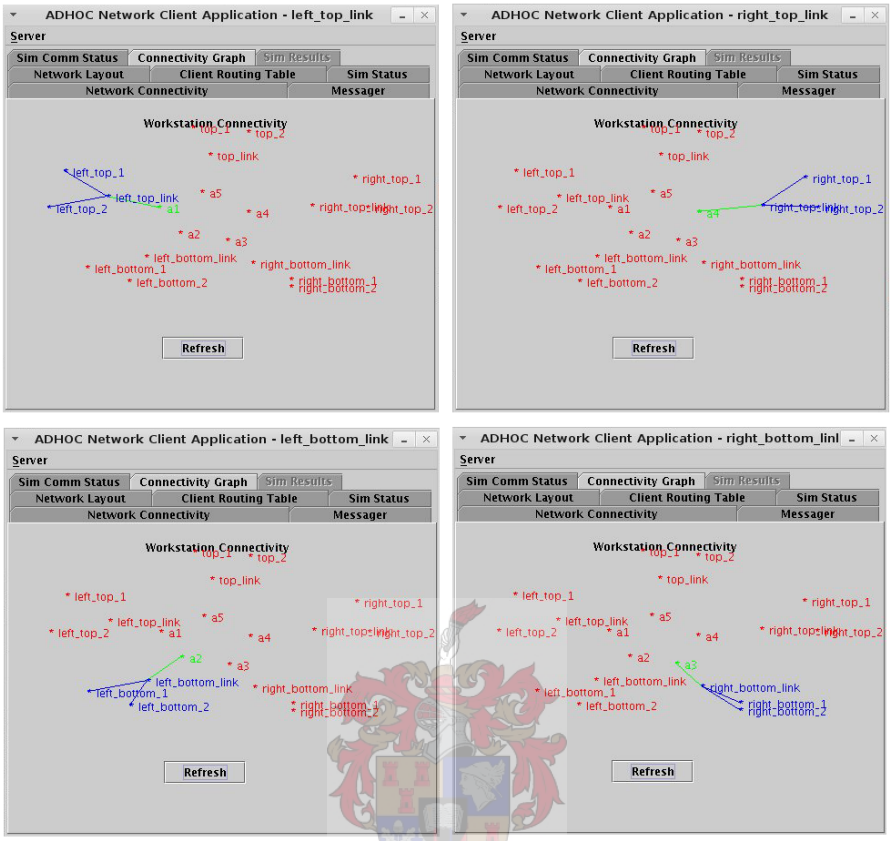


Figure 6.26: Connectivity graph of the left and right outer LANs

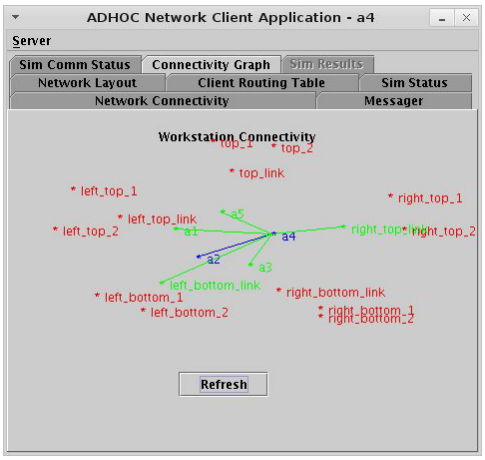


Figure 6.27: Connectivity graph from node a4's perspective

Experiment 3 - Route Acquisition

In this experiment 13 nodes are activated, forming 4 LANs and altogether one big circular network. To get this network laid out, only 9 of the nodes were activated initially, and in such a way that they aren't all within direct reach of one another, as is shown in Figure 6.28. Thus forming 4 isolated LANs .

Figure 6.28: Network layout when only 9 nodes are active

Then the remaining 4 nodes were added. These were located in such a way that each node was within range of two LANs. Thus each of these four nodes will join one LAN and form a LANedge to one other LAN, this network layout is shown in Figure 6.29 .

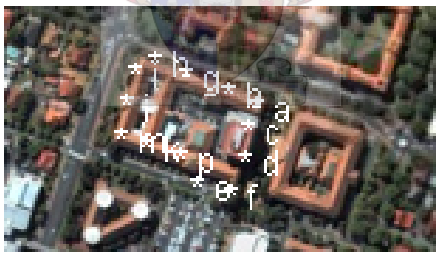


Figure 6.29: Network layout when all 13 nodes have been added

The network connectivity layout for the 4 LANs is shown in Figure 6.30 . To send a message from node 'a' to node 'm' the messenger tab is opened. The list of available nodes is refreshed. Node 'm' is selected and the message to be sent is typed. This is shown in Figure 6.31.

When the user clicks the Send button the route acquisition process will be initiated. When completed node m will choose the shortest of all paths returned as the official route for data transmission between itself and node a. Figure 6.32 shows the RTT of node a .

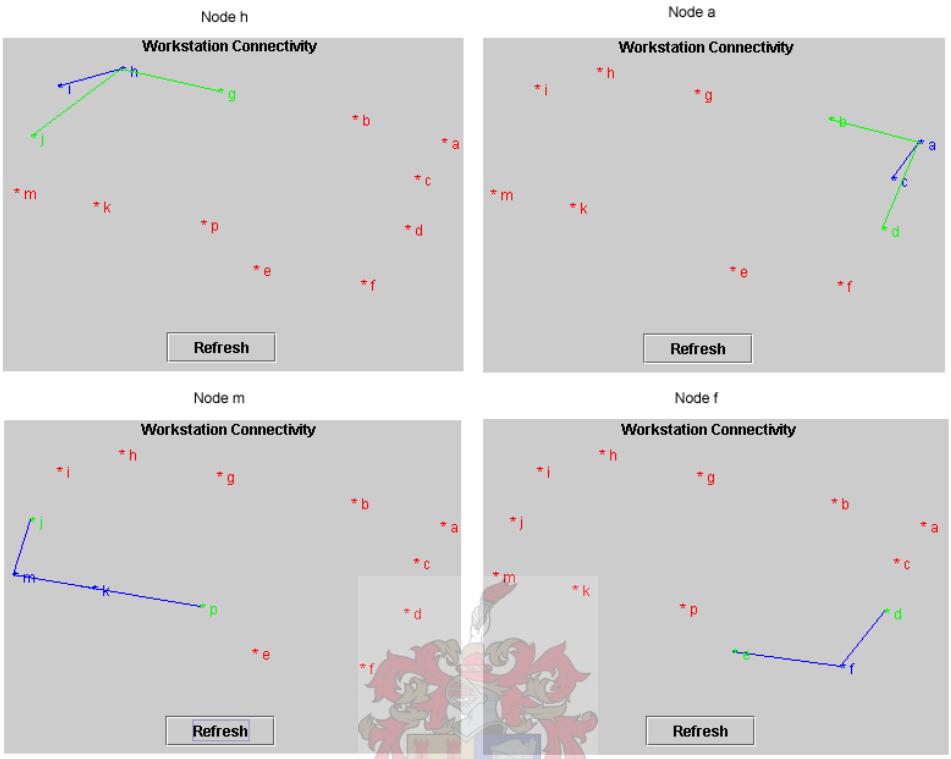


Figure 6.30: Connectivity for each one of the 4 LANs

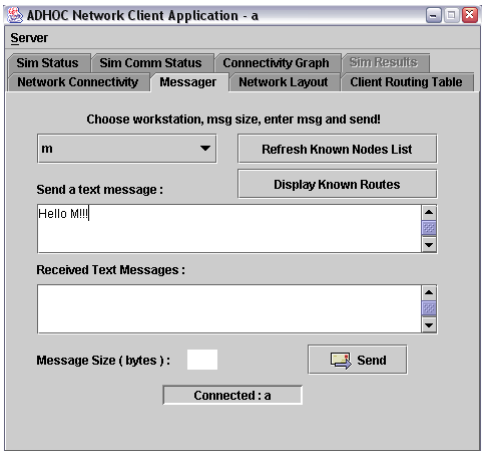


Figure 6.31: Preparing node a to send a message to node m

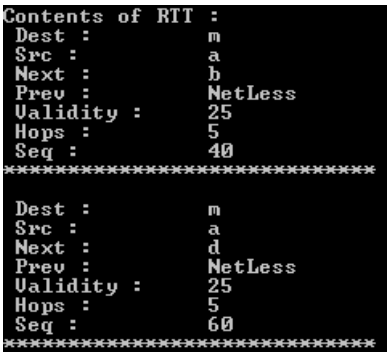
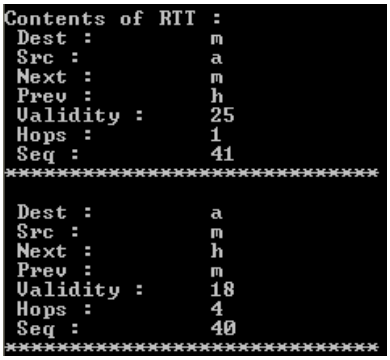


Figure 6.32: RTT for node a

Looking at Figure 6.32 it is unclear which of the two routes would have been chosen, since both routes returned the same hop count. After performing the data transmission the nodes were all left idle, thus the route used will start becoming stale. Then, before it expired, a second message was sent from node a to node m. This time upon examination of the RTT it is clear that only the revalidated route is the one in use. The node going through the gateway at node b has been used recently, and examining the RTT of node j shows that the RTT entry for a route from node a to m has recently been updated, meaning this is the route being used to transmit messages. Thus it seems that when two or more routes of equal length are found to exist then the first one listed is the one to be used for data transmission. The RTT for node j is shown in Figure 6.4.2.



In the RTTs shown there are two RTT entries. This is because node m had also sent a message to node a in reply. Shown in Figure 6.33 is the original message sent from node a to node m (the messenger at node m), and shown in Figure 6.34 is the reply from node m to node a (the messenger at node a is shown).

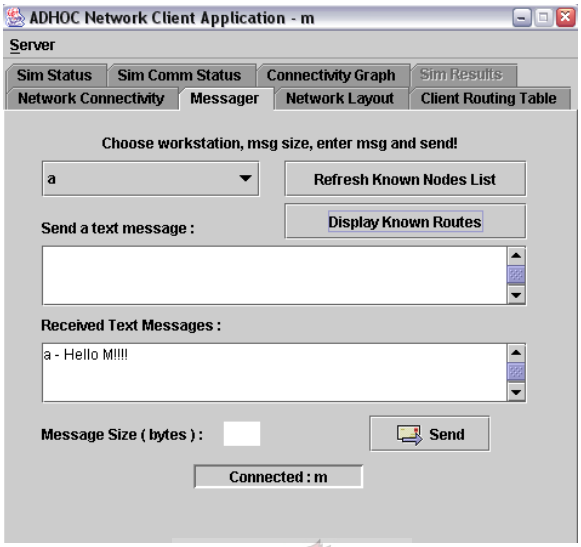


Figure 6.33: Messenger tab for node m

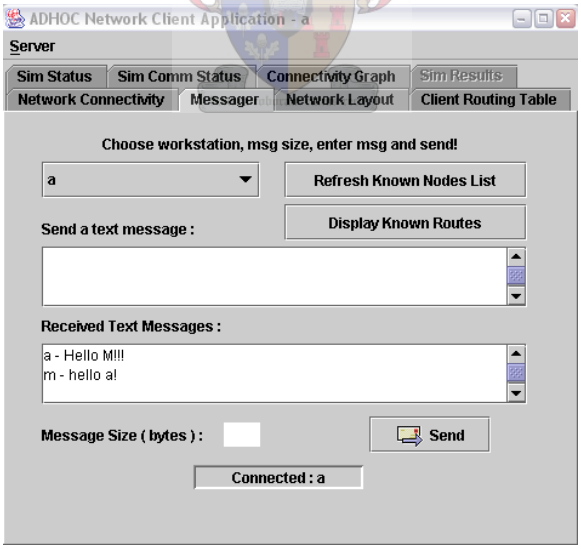


Figure 6.34: Messenger tab for node a

6.4.3 Protocol Performance Evaluation

In the following experiments the RT setup and management will be assumed to always work, and the focus of the experiments will be shifted unto the performance in terms of efficiency, e.g. number of data packets vs number of overhead packets.

Data is defined as including the data sent by a node (node is the source), and the data sent to a node (node is the destination).

Overhead is defined as being everything else, e.g. the hello messages, the reply hello message, RREQ, RREQReply, DataAcks and even Data if the node is neither the source, nor the destination, but is only relaying the data between another source and destination.

Gross Efficiency is defined as the amount of data (including data relayed) divided by the total data and overhead.

Effective Efficiency is defined as the amount of data divided by the total data and overhead.

Experiment 1 - 2 Nodes both transmitting
In this experiment two nodes were connected directly to one another, in other words, both belong to the same LAN and only have a single hop between them, and the simulation was run for 50 minutes.

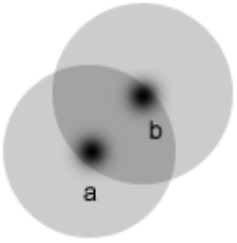


Figure 6.35: 2 Nodes communicating directly

Node Name	A	B
Data packets sent	8547	18037
Data packet total	26584	26584
Overhead packets sent	18069	8574
Overhead (routing only)	32	27
Overhead packet total	26643	26638
Effective Efficiency rating	49.9%	49.9%

Table 6.4.1: Experiment results

Experiment 2 - 3 Nodes, all transmitting
In this experiment three nodes were connected directly to one another, in other words, all belong to the same LAN and only have a single hop between them, and the simulation was run for about 20 minutes.

Figure 6.36: Experimental network layout

Node Name	A	B	C
Data packet total	9837	10981	10194
Data packets relayed	0	0	0
Overhead packet total	9922	11062	10264
Overhead packets relayed	0	0	0
Effective Efficiency rating	49.7%	49.8%	49.8%

Table 6.4.2: Experiment results

From experiment 1 and 2 it would appear that the addition of extra nodes directly connected to one another does not have a big impact on the efficiency. The addition of an extra node node does introduce extra overhead though, since each node now receives 2 updates for every update which it transmits, but this is such a small quantity in comparison to the amount of data sent that it barely affects the overall efficiency.

Experiment 3 - Data transmission incorporating intermediate nodes
In these experiments three or more nodes were connected in such a way that all nodes could only talk to one another through other nodes, except for directly linked neighbours. The simulation was then rerun for various numbers of nodes connected in this way in order to determine the effect this has on the nodal efficiency.

3 Nodes
This experiment, as laid out in Figure 6.37, was left to run for 50 minutes, after which the results shown in Table 6.4.3 were obtained.

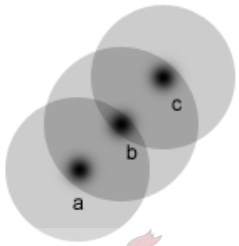


Figure 6.37: Layout for 3 nodes

Node Name	A	B	C
Data packet total	20501	22958	24461
Data packets relayed	0	11002	0
Overhead packet total	20569	67061	24521
Overhead packets relayed	0	11006	0
Gross Efficiency rating	49.9%	49.9%	49.9%
Effective Efficiency rating	49.9%	25.5%	49.9%

Table 6.4.3: Experiment results

4 Nodes
This experiment, as laid out in Figure 6.38, was left to run for 20 minutes, after which the results shown in Table 6.4.4 were obtained.

5 Nodes
This experiment, as laid out in Figure 6.39, was left to run for 27 minutes, after which the results shown in Table 6.4.5 were obtained

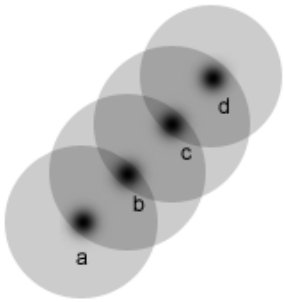


Figure 6.38: Layout for 4 nodes

Node Name	A	B	C	D
Data packet total	6310	8941	8781	8038
Data packets relayed	0	4249	5193	0
Overhead packet total	6380	26041	29651	8094
Overhead packets relayed	0	4257	5201	0
Gross Efficiency rating	49.7%	49.8%	49.8%	49.8%
Effective Efficiency rating	49.7%	25.5%	22.8%	49.8%

Table 6.4.4: Experiment results

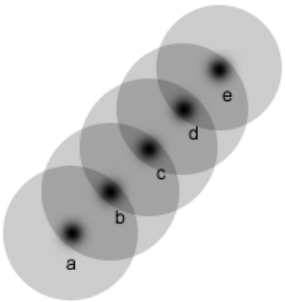


Figure 6.39: Network layout for a chain of 5 nodes

Node Name	A	B	C	D	E
Data packet total	7376	9385	9039	9614	9428
Data packets relayed	0	5759	9187	6850	0
Overhead packet total	7464	32568	45929	37130	9491
Overhead packets relayed	0	5771	9203	6862	0
Gross Efficiency rating	49.7%	49.8%	49.8%	49.8%	49.8%
Effective Efficiency rating	49.7%	22.3%	16.4%	20.5%	49.8%

Table 6.4.5: Experiment results

6 Nodes

This experiment, as laid out in Figure 6.40, was left to run for 20 minutes, after which the results shown in Table 6.4.6 were obtained.

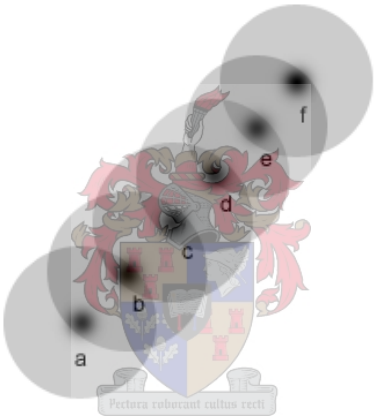


Figure 6.40: Network layout for chain of 6 nodes

Node Name	A	B	C	D	E	F
Data packet total	3795	5420	5322	5432	5620	4755
Data packets relayed	0	2916	5273	5561	1346	0
Overhead packet total	3909	17274	26610	27886	11138	4841
Overhead packets relayed	0	2932	5297	5585	1352	0
Gross Efficiency rating	49.2%	49.6%	49.7%	49.7%	49.6%	49.5%
Effective Efficiency rating	49.2%	23.9%	16.7%	16.3%	33.5%	49.5%

Table 6.4.6: Experiment results

6.4.4 Protocol Performance Evaluation Conclusion

Six experiments have been presented, some where nodes can communicate directly, and some where nodes are only able to communicate through other nodes. No cases where LANedge links were created have been shown directly, since these cases are simply an extension of the scenario where nodes communicate through one another. From a performance point of view it makes no difference, a LANedge link is merely a node through which other nodes can route their messages.

The results did not contain any surprises considering the Data-Ack and routing schemes used. The nodes which are directly linked have very good efficiencies, as expected, since each node can give the destination node it's data directly without causing any unnecessary overhead. The nodes which aren't directly linked, on the other hand, create lots of extra overhead for other nodes who have to relay their information for them. It was to illustrate this property that two efficiency ratings were defined. One to indicate the efficiency from the node's point of view, and the other to illustrate the routing overhead vs data efficiency rate.

In summary, the following main conclusions can be drawn:
1) The amount of routing overhead introduced by the nodes is negligible when the data transmission rate is high enough. The nodes constantly strive towards the maximum efficiency obtainable for the Data-Ack scheme used, as was expressed by the 'Gross Efficiency rating' value in all experiments.
2) The efficiency as experienced by a specific node is directly related to the amount of traffic it has to route for other nodes. This was expressed by the 'Effective Efficiency rating' value in all the experiments conducted.

6.4.5 Protocol Refinement

Throughout protocol development numerous experiments were conducted in order to fine tune the protocol, e.g. the maximum values for timeouts, the effects of various buffer sizes, etc were all investigated through experimentation. These experiments are too numerous to mention or highlight individually, and most results were not saved in formats easily made presentable, but to give an indication of the type of experiments done the following experiment performed in order to find the optimal RTT size is presented.

Experiment 1 - Relation between the size of the RTT and the number of nodes in a network

This experiment was setup to determine the effect of a variable called *MAX_ROUTE_ENTRIES* which determines how many routes can be stored in the RTT before old entries have to be overwritten. The goal is to determine what an optimal value for this variable would be, given a certain number of network nodes. It is important to be aware of the effect this will have on the network performance, since the size of RTT will probably be restricted in hardware and would probably be non optimal.

The experiment was run on different network layouts, mostly networks with 5 nodes or 11 nodes. Figure 6.41 shows the network layout for 11 nodes. The results shown below are for the 5 node 'red LAN' portion of the figure only and the mobility was set to zero.

The experiment was then run for 30 minutes per RTT size being experimented with, eg. with *MAX_ROUTE_ENTRIES* given a value between 4 and 60 everytime.

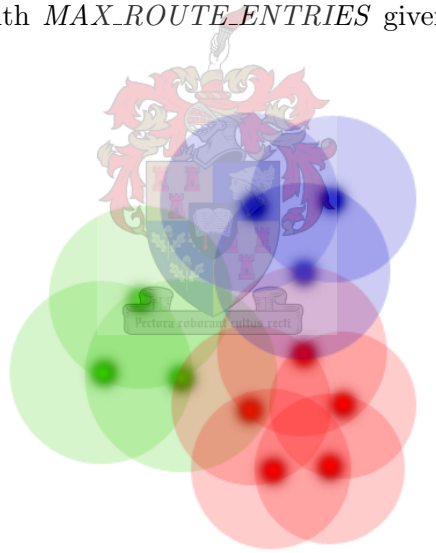


Figure 6.41: Network layout for 11 nodes

In the experiment the expectation is for the RTT to overwrite plenty of entries for small values of *MAX_ROUTE_ENTRIES*, which should have the same effect as having a small RTT expiration timer, meaning lots of routes will have to be rediscovered quite often, leading to a lower efficiency and higher overhead.

Figure 6.42 shows the average number of RREQ messages sent per run of the experiment.

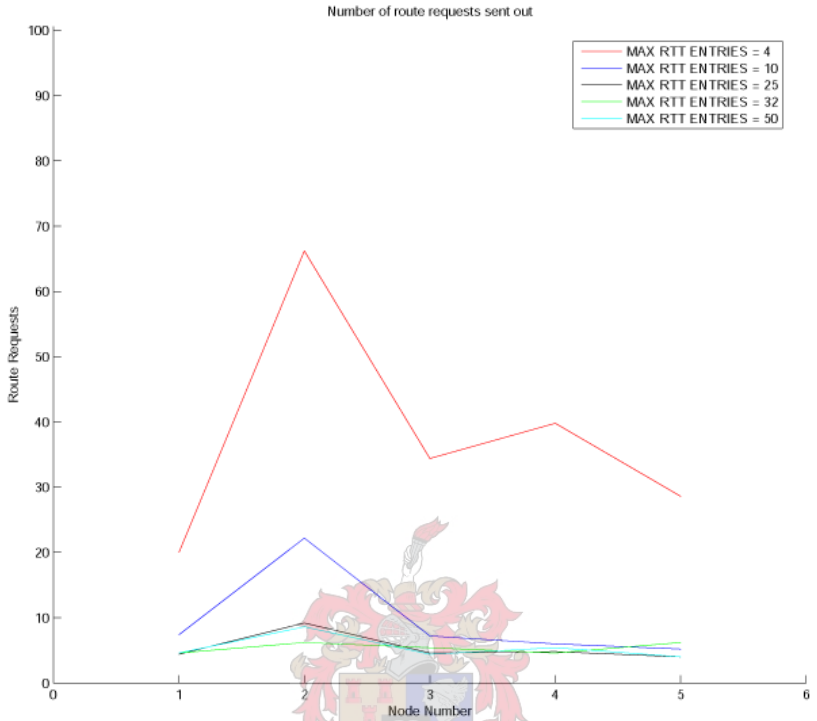


Figure 6.42: Number of RREQ messages sent

Figure 6.43 shows the number of RTT entries overwritten per run of the experiment.

Theoretically 5 nodes could have up to 25 routes, but having an RTT of n^2 is simply not practical for bigger networks. Through the experimentation done it was concluded that a good compromise can be found in $n * 2.5$. This worked well for all of the networks tested, and if memory constraints became a problem then any RTT of a size bigger than the number of nodes (RTT size $>$ n) worked well, especially if messaging is irregular.

However, if all nodes are periodically messaged then it would be optimal to be able to maintain an entry per route to avoid the extra overhead which will be introduced if route requests are regularly sought out.

In this chapter we have presented some of the results obtained with the aid of the second simulator. The results obtained proved that the simulator

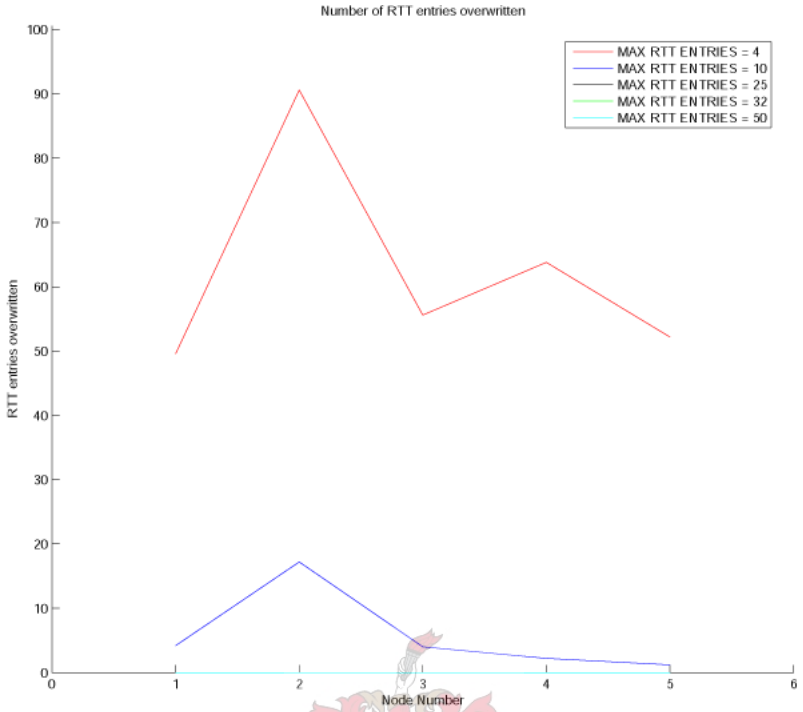


Figure 6.43: Number of RTT entries overwritten

performs as would be expected, and can be used as a tool to even further test and refine the protocol. A brief example of one such refinement performed using the simulator was also provided.

In the next few chapters the hardware design phase of the project will be discussed. Firstly some design considerations will be looked at in Chapter 7, followed by the design of an initial prototype device in Chapter 8 and the final prototype in Chapters 9 and 10.

Chapter 7

Hardware Design Considerations

7.1 Overview

The hardware necessary for implementation of a wireless network node as described in Chapter 1 can be split into three main functional categories, namely processing done by a 8-bit microcontroller, USB connectivity, and wireless connectivity. This chapter examines the design choices made in choosing the specific hardware components in each of these categories, and acts as an introduction to the next chapter in which the components are used to build the first prototype.

7.2 Microcontroller

The microcontroller is responsible for all processing done on the network node since the device has to be able to function even when disconnected from the host PC. It has to contain enough memory to be able to store and process all it's own routing tables and is responsible for the necessary IO buffering and assembly and disassembly of USB and RF messages .

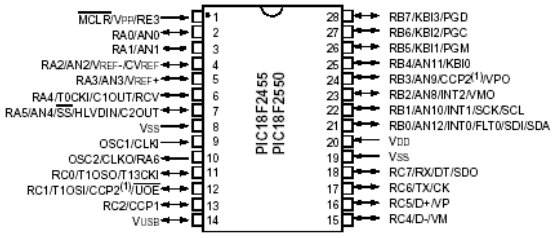


Figure 7.1: PIC18F2550, 28-Pin PDIP pin diagram [6]

Being familiar with PIC microcontrollers, the PIC18F2550 from Microchip was quickly selected as it fulfilled all the requirements set forth for a microcontroller and had the advantage of having a USB port built-in. This means that all USB functionality (discussed in Section 7.3) can be accomplished without adding any extra circuitry to the board, thereby reducing the complexity and cost of the board.

With the USB port only utilizing two pins on the chip it was decided that the 28pin version of the PIC would be sufficient since the RF module would only need a couple of pins, leaving more than enough pins free to connect LEDs to indicate TX and RX, etc. The 28pin PIC18F2550's main features (from [6] pg.1) are briefly summarized in Table 7.2.1 and the chip's pin diagram is shown in Figure 7.1. To program the PIC18F2550 a ICD2 programmer was

Flash Memory	32Kbytes
SRAM (bytes)	2048
EEPROM (bytes)	256
I/O pins	24
8-bit timer	1
16-bit timers	3
External interrupts	3

Table 7.2.1: PIC18F2455/2550/4455/4550 High-Performance,Enhanced Flash USB Microcontrollers [6]

used. This programmer allows for both programming and in-circuit debugging of hardware, and since it plugs into a USB port it is also self-powering. The Microchip MPLAB IDE used to create the entire firmware project also supports the ICD2 programmer used to program the PIC. The firmware was written in C, and compiled using the MPLAB C18 compiler from Microchip. This compiler can also be integrated into MPLAB IDE, thus allowing the user to create a project, write the C source code files, compile them, and program the PIC microcontroller using the same programming interface.

All of the above mentioned software can be downloaded from the Microchip website, so no additional costs were incurred. The only limitation being that the compiler has a 60 day trial period whereafter the optimization feature expires. All necessary documentation, user guides, FAQ, tutorials and installation documentation are available from Microchip. Microchip also hosts a very comprehensive discussion forum, where most problems have already been discussed and solved, with a quick response time to any new queries.

7.3 USB Connectivity

7.3.1 Benefits of using USB

- **Single interface:** A single universal interface is provided that can be used by many kinds of devices. The connectors are small, simple, cannot be plugged in the wrong way, and are also compact in contrast to other connectors.
- **Automatic configuration:** When a USB device is connected to a powered system, it can automatically be detected and configured.
- **No settings :** USB peripherals do not have port addresses or interrupt request (IRQ) lines. This frees hardware resources for use by other devices.
- **Easy to connect and 'hot swappable':** There is no need to open the computer. Most computers have at least two USB ports, and more ports can be added. USB devices can be connected and disconnected as and when needed.
- **No power supply required:** The USB bus provides power on a +5V and ground lines. A device that requires up to 500mA can draw all it's power from the bus, instead of requiring it's own power supply.
- **Speed:** USB supports three bus speeds: high speed (480Mbps), full speed (12Mbps) and low speed (1.5Mbps). Every USB-capable computer supports low and full speed. High speed was added with the USB 2.0 specification. Low speed devices are cheaper as they do not require shielding.
- **Automatic error checking:** The developer does not have to provide error checking algorithms in software to check that the data is correctly transmitted and received. This is done by the hardware (host controller hardware).
- **Flexibility:** The USB protocol defines a number of data transfer modes which makes it very flexible for the application that it is to be used for.

7.3.2 USB as I/O Device

All USB transmissions travel to or from a device **endpoint**, which is a buffer that can store multiple bytes. The USB specification defines a device endpoint as 'a uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device' [21]. All devices must contain at least endpoint 0, which is a bidirectional endpoint used for control purposes. A typical device will have a collection of *IN* and *OUT* endpoints. This collection is called an interface. A pipe describes the logical connection between the PC host software and an interface. A USB device may have multiple interfaces.

A USB device may belong to any of the defined USB classes mentioned briefly in Table 7.3.1 below. Table 7.3.1 shows the USB base class values, what the generic usage is, and where that Base Class can be used (either Device or Interface Descriptors or both). These classes specify protocols and functions that a device must be able to support if it belongs to that class. If a device belongs and adheres to a specific class's definition then it can use the existing drivers included in the OS, otherwise a custom or generic USB device driver must be used to access the USB device.

Base Class	Descriptor Usage	Description
00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Ch	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

Table 7.3.1: USB Base Class Values [9]

.

7.3.3 Device Drivers

A device driver is a type of computer software, specifically developed to allow interaction between the computer's OS, application software and hardware devices [22].

The key design goal of device drivers is abstraction. Every model of hardware (even within the same class of device) is different. Newer models are also continuously released by manufacturers that offer an improvement over the older model, and these newer models are often controlled differently. The computer's operating system cannot be expected to know how to control every device, both current models and future models.

In order to solve this problem operating systems have developed a set of interfaces which dictate how every type of device should be controlled. Thus a device driver insulates applications from requiring details about the physical connections, signals, protocols, addresses or ports the device attaches to. The driver has to comply with the interface required by the OS on the one level, and on the other it has to provide hardware specific code and be able to handle the protocols necessary to access the peripheral's circuits. Thus by constantly translating between application-level and hardware-specific code, it allows the system to function and interact transparently.

Device drivers also differ, some drivers take care of absolutely everything necessary to allow the application program to control and use the hardware device. Other drivers make use of a layered driver model, in which each driver performs a portion of the communication between the application and the physical device. The top layer contains a function driver that manages communications between applications and the lower-level bus drivers. The bottom layer contains a bus driver that manages the communication between the function driver and the device.

The layered driver model is more complicated, but it simplifies the task of writing a device driver, as many of the low-level hardware drivers may already exist, and only function drivers need to be written. It is also more efficient, because it enables different devices that have tasks in common to use the same driver for those tasks.

For this project, two Windows drivers were used, usbser.sys and ccport.sys, in conjunction with the Microchip USB CDC firmware (see next section). This combination of firmware and drivers provide adequate functionality and performance for the needs of the SWARM network node.

7.3.4 Microchip CDC USB

The two Windows drivers, usbser.sys and ccport.sys, provides the necessary services to allow the creation of a virtual COM port with which the application PC software can reference the connected USB device, in this case the SWARM network node. Thus the SWARM network GUI as developed in Chapter 11 will see the USB connection as a COM port, same as if a physical RS-232 connection had been made, and can communicate with the attached device using the CreateFile, ReadFile, and WriteFile functions (standard Windows functions used to communicate to a serial port [23]).

The Microsoft Windows driver, usbser.sys, conforms to the Communication Device Class (CDC) specification 1.1 which defines many communication models, including serial emulation. Therefore, the embedded device must also be designed to conform with this specification in order to utilize this existing Windows driver.

Features in version 1.0 of the RS-232 Emulation firmware include:

- a relatively small code footprint of 3 Kbytes for the firmware library.
- data memory usage of approximately 50 bytes (excluding the data buffer).
- maximum throughput speed of about 80 Kbps.
- data flow control is handled entirely by the USB protocol.
- no additional drivers are required; all necessary files, including the .inf files for Microsoft® Windows® XP and Windows® 2000, are included

The CDC specification describes an abstract control model for serial emulation over USB in Section 3.6.2.1 of [7]. In summary, two USB interfaces are required. The first one is the Communication Class interface, using one IN interrupt endpoint. This interface is used for notifying the USB host of the current RS-232 connection status from the USB RS-232 emulated device. The second one is the Data Class interface, using one OUT bulk endpoint and one IN bulk endpoint. This interface is used for transferring raw data bytes that would normally be transferred over the real RS-232 interface.

Using this solution vastly improves development cycle time since it provides a fully functional framework capable of meeting all the requirements set by the SWARM network node. This elliminates the need to create new descriptors from scratch or write function handlers for Class-Specific Requests. Descriptors for a USB RS-232 emulated device and all required handlers for

Class-Specific Requests listed in Table 4 of the CDC specification [9] are provided with the Microchip USB CDC firmware.

7.3.5 Microchip CDC USB Firmware Core Functions

mUSBUSARTIsTxTrfReady:
Prototype: `bool mUSBUSARTIsTxTrfReady(void)`

This function is called before a USB data transmission using the CDC class is initiated, to check whether the CDC class is ready to send more data. It is a non-blocking function typically used as part of an 'if' statement [7].

putrsUSBUSART:
Prototype: `void putrsUSBUSART(const rom char *data)`

This procedure writes a string of data to the USB including the null character. This version is used to transfer literal strings of data, or data located in program memory, to the host. mUSBUSARTIsTxTrfReady() must return '1' before this function can be called. The amount of data to be sent must be equal to or smaller than 255 bytes, including the null-terminated character [7].

putsUSBUSART:
Prototype: `void putsUSBUSART(char *data)`

This function writes a string of data to the USB including the null character. Use this version to transfer data located in data memory to the host. mUSBUSARTIsTxTrfReady() must return '1' before this function can be called. The amount of data must be equal to or smaller than 255 bytes, including the null-terminated character. This function formed the backbone of the USB to host PC application data transfers. An example [7] of how it would be used is shown below :

```
char example_string[6];

void SWARM_example(void) {
   example_string[0]='S';
   example_string[1]='W';
   example_string[2]='A';
   example_string[3]='R';
   example_string[4]='M';
   example_string[5]=0x00;
   if(mUSBUSARTIsTxTrfReady())
      putsUSBUSART(example_string);
}//end example_1}
```

7.4 Wireless Connectivity

Several different types of wireless devices were considered, primarily those that operate in the 2.4 GHz band as these were the most common. The prerequisites set were that the device:

- had to be a transceiver (capable of both receiving and transmitting information).
- had to be inexpensive.
- had to be able to operate with only the 8bit PIC microcontroller controlling it.
- must have a decent range.
- must not require too much RF knowledge to get started (to comply with time constraints).
- must have a data throughput rate that would make for a decent proof of concept.

The Nordic Semiconductor family of chips was selected in the end since they conformed with the requirements set, and several of the staff at the University of Stellenbosch had already previously worked with these chips. An application note from Nordic Semiconductor for the Universal low cost USB DuoCeiver which uses a Nordic chip, the nRF2401 together with a Cypress USB MCU, [24] made quite an impressive introduction to the nRF2401, especially since the kind of application it was implemented in, closely resembled the SWARM network node to be built. Thus the specific Nordic chip selected was the 2.4GHz nRF2401 transceiver.

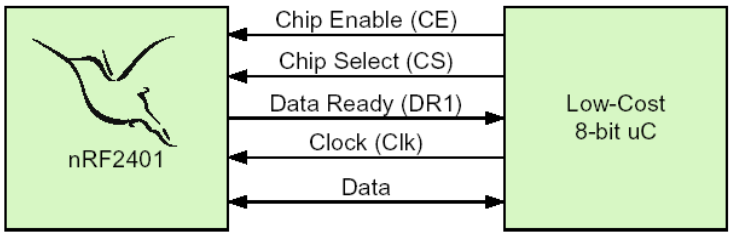


Figure 7.2: Physical connection of a nRF2401 and a low-cost uC [25].

7.4.1 nRF2401 2.4 GHz Transceiver

The nRF2401 is a single-chip radio transceiver for the world wide 2.4 - 2.5 GHz ISM band. Features of the nRF2401 :

- Multi channel operation - 125 available channels.
- Data rate 0 to 1 Mbps.
- Address and CRC computation.
- Shockburst mode for ultra-low power operation and relaxed MCU performance.
- Maximum output power 0 dBm.
- World wide use.

The ShockBurst feature of the transceiver automatically handles preamble and CRC. Configuration is programmable via a 3-wire interface by serially clocking the data in bit by bit. Current consumption is very low, in transmit it consumes only 10.5mA at an output power of $-5dBm$, and in receive mode only 18mA. Built in 'power down' modes make power saving easily realizable. See [25] for the complete Nordic semiconductors nRF2401 datasheet .

Mode	PWR_UP	CE	CS
Active (RX/TX)	1	1	0
Configuration	1	0	1
Stand by	1	0	0
Power down	0	X	X

Figure 7.3: nRF2401 main modes [25].

The operational modes shown in figure 7.3 are available and perform the following roles :

Active Mode (RX / TX)
The nRF2401 is in either Shockburst or direct mode, depending on configuration.
Configuration Mode
The nRF2401 can be configured by means of a 15 byte configuration word.
Stand-By Mode
The nRF2401 uses minimal current in this mode, whilst maintaining short start up times. The configuration word is also maintained.

Power Down Mode
The NRF2401 is disabled. This results in minimal current consumption, maintains the configuration word and is used to maximize battery life when the device is not active.

These operational modes are selected using the three inputs, PWR_UP, CE and CS as indicated in Table 7.3, otherwise the nRF2401 module has one output to indicate it's message reception status and the bidirectional data pin to transfer data to the unit before a message is transmitted and from the unit after a successfull packet reception has been signalled.

7.4.2 nRF2401 2.4 Ghz Transceiver Shockburst Mode

The whole idea with the ShockBurst technology is to put as much low level protocol handling into the nRF chip as possible without removing any flexibility from the user.[26]

This means that the MCU can look at the nRF2401 as an 'advanced register' where data to be transmitted can simply be clocked in at a speed set by the MCU itself. Thus even if a very low cost MCU is used which utilizes it's built in oscillator it can still make use of the high bit rate of 1Mbit/s on the air.

In receive mode it is even simpler, the nRF2401 will detect and receive a transmission, check that the TX address of the received packet matches the device address configured for this nRF2401, and check that the CRC was correct. If both tests are passed then the nRF2401 has a valid packet and sets the DR1 (Data Ready 1) output pin high. The MCU sees the DR1 pin is high and starts clocking out the data at its own speed. In doing it this way the nRF2401 does a lot of the processing itself, like all the low level protocol handling, bit sampling, address checking and checksum calculation, thereby placing a much smaller role on the MCU. Thus shockburst mode allows for both RX and TX without the need for precise timing or high speed operation, making it possible to choose among even the cheapest micro-controllers.

Other advantages to clocking data in at a speed determined by the MCU and transmitting that data at 1Mbps is that not only is the current consumption greatly reduced, but the risk of 'on-air' collisions gets drastically reduced due to the short transmission time. This process is shown graphically using a 10kbps example in Figure 7.4.

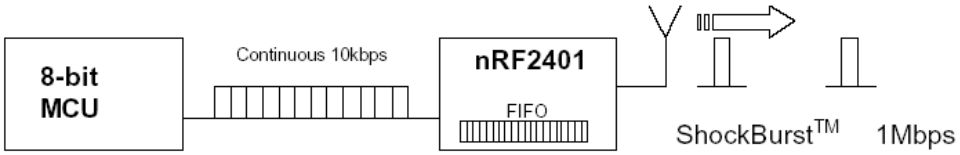


Figure 7.4: Clocking in data at the MCU speed and transmitting with Shockburst technology at 1Mbps [25].

7.4.3 nRF2401 2.4 GHz Transceiver and Link Integrity

When venturing into the 2.4 GHz ISM band which also contains known wide spectrum and powerful systems like wireless LAN (WLAN) and Bluetooth, quality-of-service and system integrity are issues of great concern.

Traditionally the RF specification of the device itself (utilizing SAW filters etc.) and limiting the number of operators (licensing) in a band has carried the main burden in accomplishing this.

But in open bands, such as the ISM, there can be an unlimited number of devices and systems in operation. RF protocols that can cope with the occasional collision while maintaining system integrity become increasingly important. The shockburst feature as discussed previously in Section 7.4.2 helps a lot by enabling all transmissions to take place at maximum transmission speed, thus lowering the time spent broadcasting.

$$TimeOnAir = 216bits/1Mbps = 216\mu s$$

The short time on the air of course significantly reduces the risk of being jammed by other systems. Since most of the other operators in the 2.4 GHz band also operate on similar data rates, no single channel is occupied for long periods of time which partly replaces the need for the strict frequency separation.

However, to make sure a system built using the nRF2401 can coexist with other systems and to build a robust system, a quick summary of the other 2.4 GHz systems needs to be done. How to handle collisions also depends on what kind of system is interfering with the transmission.

The different 2.4 GHz systems can be divided into 4 categories [4]:
1. Direct spread spectrum devices (WLAN)
2. Burst and frequency jumping systems (nRF24xx and Bluetooth)
3. Single channel, low data rate systems.
4. Multiple systems of same type (Multiple nRF24XX systems)

Wireless LAN

Powerful systems with large amounts of data being transferred like WLAN use direct spread spectrum to spread the RF energy across a wide frequency range. One WLAN utilizes a 22 MHz wide channel and the energy density towards the outer reaches of the direct sequence transmission is down to at least $-10dBm$ (according to FCC guidelines).

Furthermore, any physical separation between a nRF24XX device and WLAN will reduce the received RF energy from the WLAN. The inverse square law dictates that 6dB of output power is lost with each doubling of distance from the transmission source, thus placing a nRF2401 module right next to a WLAN access point might give it very poor performance[4].

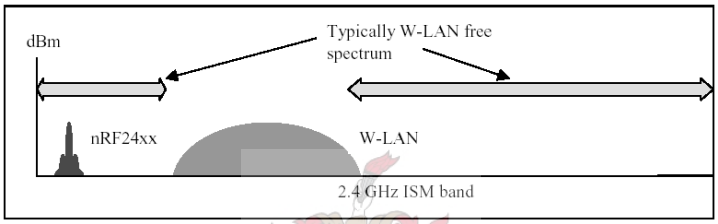


Figure 7.5: 1 nRF24xx coexisting with WLAN (measured at receiver) [4]

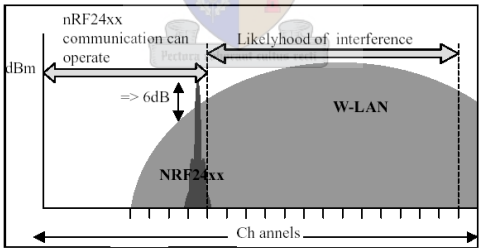


Figure 7.6: nRF24xx operating at outer limits of W-LAN spectrum [4]

It can be seen in Figure 7.5 that typically, only one WLAN network will be physically co-existing in any particular area. This means that 75 % of the ISM band is WLAN free, this equates to 60+ channels free for nRF24xx operation. Figure 7.6 shows that at the outer channels utilised by the WLAN transmission there is a sharp fall-off in signal strength and therefore the nRF communication may well be able to operate in this region with only some degradation of system sensitivity as a result.

Bluetooth

Bluetooth has become the most popular and common system utilizing the 2.4 GHz space and the nRF24XX devices share the same basic radio architecture. However, Bluetooth and the nRF2401 can coexist rather well since Bluetooth needs a complex protocol engine which jumps between 79 channels, but only stays on one channel for 625μs. Thus if a collision were to occur at the channel used by the nRF24xx system, it can be taken as a fact that the Bluetooth will swap to another channel within 625μs, leaving the channel free again for the nRF2401 to re-transmit it's package.

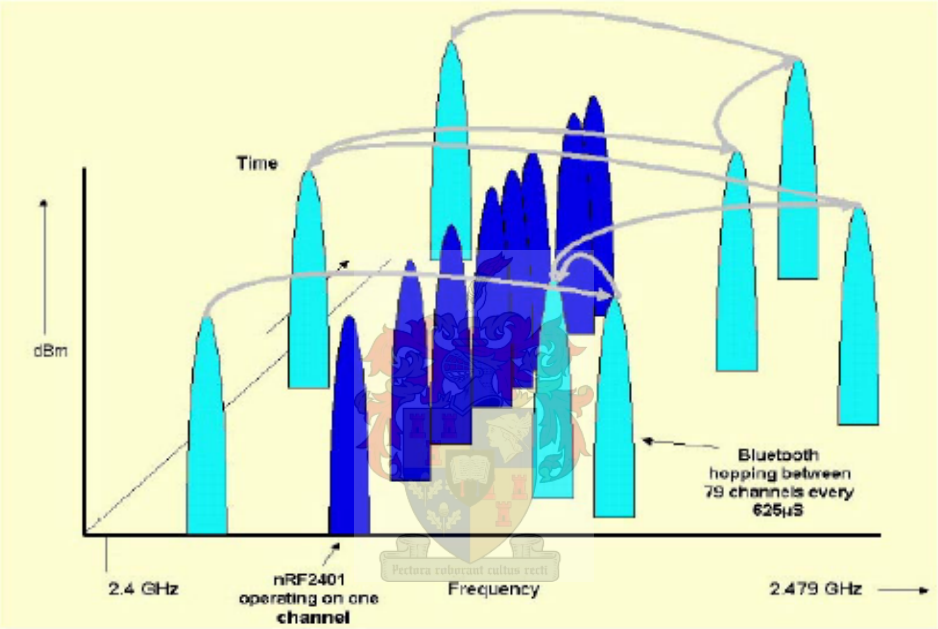


Figure 7.7: nRF2401 coexisting with Bluetooth [4]

Other 2.4 GHz Single Channel Systems

Transmitters sending almost continuously are required to have a RF output of less than 0dBm, so any influence it has is limited in space (distance) and is for the main part handled by the RF blocking in the nRF2401 receiver. Thus if such a problem is detected the solution is to create a physical seperation between the devices, or a manual way of setting the nRF2401 to a new frequency. The low data rate system on the other hand is more susceptible to interference from a lot of systems due to the long time they stay on the air. The detection schemes used are also not that effective against devices which have a much shorter turn around time than the device itself, such as Bluetooth or even the nRF2401.

Other nRF2401 or Similar Devices
When multiple nRF2401 systems are operating within a confined area the high data rate and ShockBurst keeps the probability of collisions low, even if the systems share the same frequencies. On a non-hardware level it is up to the transceiver communications protocol to ensure link integrity. In the next two Chapters, 8 and 9, we take a look at the SWARM network protocol implementation and discuss how collision avoidance was built in, and also how the transceiver automatically deals with collisions if and when they do actually occur.

7.4.4 nRF2401 2.4 Ghz Transceiver PCB layout

Nordic Semiconductors provide an application example of the nRF2401 with a single ended matching network. A design based on this should simply work, however as quoted from the Nordic nRF2401 datasheet '*A well-designed PCB is necessary to achieve good RF performance. Keep in mind that a poor layout may lead to loss of performance, or even functionality, if due care is not taken.*'. The nRF2401 also comes in a very small package, 5mm x 5mm, which means the pins of the chip are 0.3mm apart, and that's too fine to guarantee reliable manufacturing of PCB boards manufactured by the University of Stellenbosch.

Thus all things considered it was decided to purchase the nRF2401 transceiver as part of a complete module. This was decided since the building, fine-tuning of the transceiver and it's antenna is not the point in constructing a SWARM network node. By purchasing a module at least it can be assumed to be reliably constructed.

The module purchased was the TRF-2.4G designed by Laipac Technologies Inc. It is simply the nRF2401 connected to a closed loop PCB printed antenna, with a 16Mhz clock, and all small components necessary to build the proper impedance matching circuit already built into the module. It has a claimed range of 280m @250Kbps and 150m @1Mbps and was bought from Sparkfun Electronics.

In this chapter we have investigated the hardware components necessary to design an ad-hoc network device with a general purpose CPU, low cost components and non-proprietory software tools and libraries. In the next chapter we design a prototype device using these components.

Chapter 8

Prototype Design

8.1 Overview

In designing the first prototype a modular approach was taken, where all the main components necessary for the implementation of a SWARM network node were given a rating according to how vital it's role in the overall design is and then it was setup, configured and tested individually. This allowed for smaller, more specialized pieces of test code which thoroughly tests those features of a specific component that will be used later, and when completed provides a good overall design framework from which to create the final prototype.

This chapter briefly describes the development of the prototype using a router board in order to test all system components, determine their external component values, determine their position relative to one another, and how they are all interconnected, before Chapters 9 and 10 describe the hardware and firmware of the final prototype developed.

8.2 First Prototype

The prototype was designed to be a stand-alone fully functional layout of the SWARM network node to be constructed, except that it contains two transceivers. In this way one can be setup as a transmitter and one as a receiver for testing whether the RF modules are operational (see Figure 8.1).

8.2.1 Hardware Design

The typical interconnections circuit between the ICD2 programmer and the Microchip MCU ([27], pg.8) was used to enable programming and debugging of the MCU. The ICD2 was not used to power the board, instead the

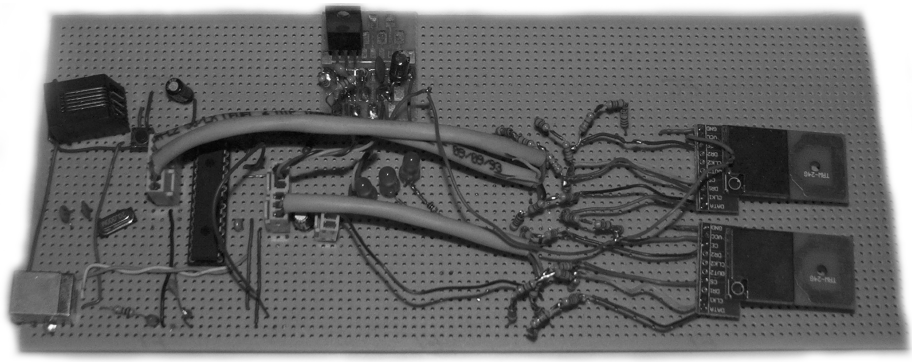


Figure 8.1: Prototype board

USB port itself was chosen as power source since this is the simplest power method for this device and one that would normally be available, unlike the ICD2. This meant adding a capacitor of 470nF between GND and the V_{USB} pin of the PIC, as is shown in Figure 8.2. The USB connector's +5V and GND can then be connected to the board's VDD and VSS as done in Section 9.4. The nRF2401 transceivers however, use a vdd of between +1.9V and

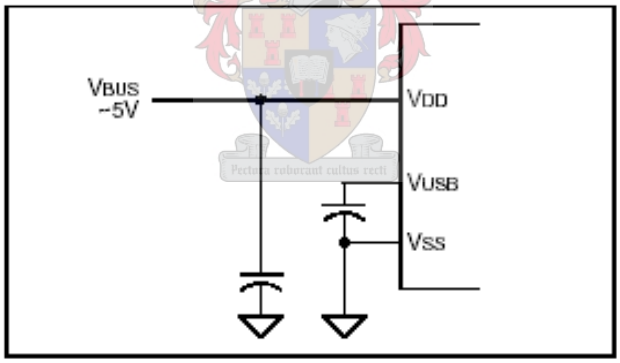


Figure 8.2: USB connection in bus power only mode ([5],[6] pg.182)

+3.6V. The lower power consumption of 3V devices is the motivation behind the transition from 5V to 3V systems (and even lower operating voltages) in new systems. Very often, not all components are available as +3V devices. Mixed voltage designs are possible as long as there is a reliable signal transfer between the +5V and +3V systems. The required supply voltages are not a problem since the USB bus supplies +5V, which a voltage regulator uses to provide the 3.3V to the lower voltage components. The voltage regulator circuit constructed uses the LM317 to obtain a steady +3.2V from

the +5V USB power. In the final prototype this was replaced with a single chip voltage regulator (see Section 9.4).

A voltage divider circuit was also constructed to convert the signals travelling to and from the MCU to the nRF2401. This provided a cheap option good enough for an initial prototype board. This option was made possible because the MCU CMOS I/O levels and the nRF2401 I/O levels are compatible enough for a logical high output to be picked up as a logical high at the other device, and only a small resistor circuit was constructed to limit the current flow. In the final prototype this crude solution was replaced by a +5V to +3V signal level shifter (see Section 9.6).

8.2.2 PIC18F2550 USB Firmware Design

Getting Started

The CDC firmware framework as provided from Microchip was created directly for the PIC18f4550 which is the bigger brother of the PICmicro MCU used here. Thus the code would have to be adapted to work properly. To do this help was sought on the Internet, and it soon became apparant that there are a lot of people out there who are already trying to convert the code to other USB enabled MCUs, but with varying degrees of success.

The following methodology was used: first use the code exactly as it is, even though it is compiled for the 44pin version of this family of PICs, in order to test that the code does in fact work, to see how it works, and to test which parts can be removed while still leaving all functionality behind. Then, with a clear understanding of which are the core parts that need to be changed the firmware can be adapted and recompiled for the PIC18F2550. This method was made possible by the fact that all the important pins, like those used for power, programming, and USB, are all at the same locations, thus physically it is only some peripheral I/O pins which are absent in the smaller PIC.

In order to program the PIC18F2550 with the PIC18F4550 firmware (MPLAB wouldn't allow it) the MCU was first programmed with a bootloader which can then be used to load the pre-compiled CDC code into the MCU ROM.(This process is described in [28] and [5]) The bootloader creates a very fast and effective way of programming the MCU using only software and a USB port on the host PC. This would be of great use in a classroom environment since there is no need for any PIC programmers, however, by using an ICD2 programmer, the extra RAM and ROM used by the bootloader becomes an unnecessary and expensive loss of resources. The memory map for this setup is shown in Figure 8.3. It was also noted that the bootloader code and the CDC code are not very different, thus there is a lot of redundancy.

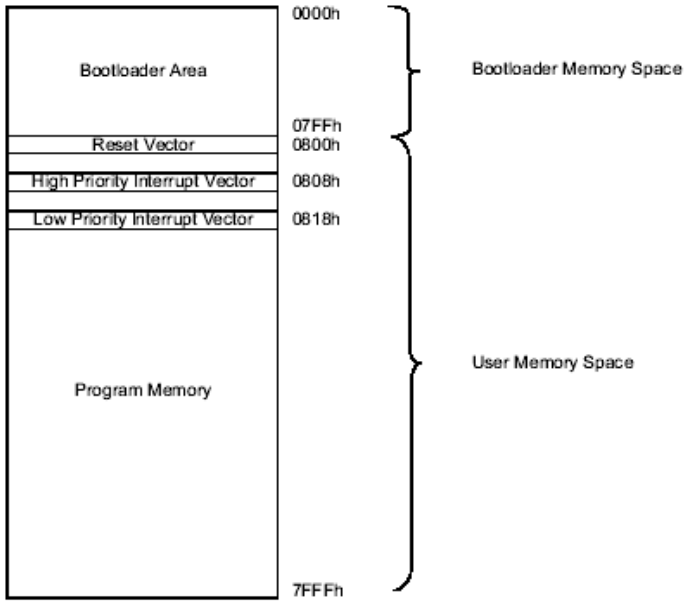


Figure 8.3: PIC18F4550 program memory map (bootloader implemented)

In the next section we adapt the code for the PIC18F2550 and shrink the code down to it's bare functional minimum.

CDC-based USB Firmware Code for PIC18F2550

The code was originally created for the PICDEM FS-USB board utilizing a PIC18F4550, thus there were some changes which had to be made. The demo board has circuitry to sense whether an alternative power source was available and also to check if the USB was connected or not, which had to be removed. All configuration settings, as well as the device and memory linker files had to be replaced with those for the PIC18F2550. There were also routines which used four LEDs to indicate the USB connectivity state that were removed since IO pins are limited on the PIC18F2550.

The main challenge came in implementing the USB CDC functions within the framework of a larger program. The application note from Microchip [7] states that the Microchip USB firmware is a cooperative multitasking environment where there should not be any blocking functions in the user code. The USB tasks are polled and serviced in the main program loop, and any blocking functions that are dependent on the state of the USB may cause a deadlock. It recommends the use a state machine in place of a blocking function. For instance, when checking if data can be transmitted on the USB port, *while(!mUSBUSARTIsTxTrfReady());* cant be used as it

Incorrect Method (back-to-back calls):

```
if(mUSBUSARTIsTxTrfReady())
{
    putrsUSBUSART("Hello World");
    putrsUSBUSART("Hello Again");
}//end if
```

Correct Method (state machine):

```
byte state = 0;
if(state == 0)
{
    if(mUSBUSARTIsTxTrfReady())
    {
        putrsUSBUSART("Hello World");
        state++;
    }//end if
}
else if(state == 1)
{
    if(mUSBUSARTIsTxTrfReady())
    {
        putrsUSBUSART("Hello Again");
        state++;
    }//end if
}//end if
```

Figure 8.4: Recommended use of USB UART function calls [7]

is a blocking function.

The reason bocking functions should not be used is because of the way that putrsUSBUSART, putsUSBUSART, mUSBUSARTTxRom and mUSBUSARTTxRam operate. They do not send out data to the USB host immediately, nor wait for the transmission to complete. All they do is set up the necessary Special Function Registers and state machine for the transfer operation. The routine that actually services the transfer of data to the host is CDCTxService(). It keeps track of the state machine and breaks up long strings of data into multiple USB data packets. In the firmware provided it is called once per main program loop in the USBTasks() service routine. Because of this, back-to-back function calls will not work; each new call will override the pending transaction. See Figure 8.4 for examples of correct and incorrect use of these functions. Always check whether the firmware driver is ready to send more data out to the USB host by calling mUSBUSARTIsTxTrfReady() first.

In the larger context of the SWARM network node firmware it will be necessary to send data to the USB port and host PC throughout the code, from

within procedure calls and even interrupts, thus the use of such a state machine as recommended is simply not feasible. To solve the problem of data being sent from anywhere in the firmware two global buffers were created, namely *USB_output_buffer* and *USB_TX_Buffer* which can both store up to 27 bytes of data.

The *USB_output_buffer* allows data to be *sent* to the USB from anywhere in the code. It uses two functions called *Single_USB_Msg(unsigned char)* and *Two_USB_Msgs(unsigned char,unsigned char)* to store the data. When ready (more detail given in Section 10.4.9) to send all the buffered USB data, the contents of the *USB_output_buffer* will be transferred into the *USB_TX_Buffer* and a flag set to indicate that USB data is ready to be sent to the host PC.

If this flag is set, the main loop calls a function called *SendUSB*, which polls *mUSBUSARTIsTxTrfReady()* to see if the state machine is ready for a USB transfer. It it is ready to perform a transfer, the entire *USB TX Buffer* contents will be shifted into the Special Function Registers. The CDCTxService() routine that actually services the transfer of data to the host is called from within an interrupt (see Section 10.2.2) and will conduct the transfer as soon as possible. The routine was placed inside an interrupt because it must be called at least once every 1ms for the USB connectivity to remain active and the interrupt guarantees a specific frequency compared to the normal code execution which has varying inherent processing delays.

8.2.3 PIC18F2550 Interrupt Handling

Overview

The firmware makes use of three interrupts, of which two are timed interrupts and the other is a interrupt-on-change interrupt. A timed interrupt increments a counter with every clock pulse of the MCU until a specific value is reached, thus creating an event which occurs with a precise frequency period. This overflow value is calculated by using a fixed base value of either 8 or 16 bits, which gets multiplied by a prescaler (and sometimes a post scaler) of between 0 and up to as much as 256, to give us the total overflow value. The interrupt-on-change interrupt routine fires when the input to a specific input pin on the MCU changes level, eg. goes from a low to a high state.

The slower timed interrupt, Timer0, utilizes a 16-bit base value with a prescaler of 128. That means this interrupt fires every time the overflow counter reaches a value of 8388608, thus giving an effective delay period of 0.699s between interrupts, as given by :

$$(2^{16}) \times 128 = 8388608 \text{ clockcycles}$$

$$\frac{8388608}{12000000} = 0.69905s$$

Thus the Timer0 ISR will be executed approximately once every 0.7seconds. To boost this time up to once every 2.1 seconds the ISR was written in such a way that it only executed every third time the interrupt gets triggered. The TMR0 ISR is responsible for scheduling updates, decreasing the validity of RT and RTT entries, merging subnets, and setting up links between subnets.

The faster timed interrupt, Timer2, utilizes a 8-bit base value with a prescaler of 1, and a postscaler of 2. That means this interrupt fires every time the overflow counter reaches a value of 512, thus giving an effective delay period of $42.67\mu s$ between interrupts, as given by :

$$(2^8) \times 2 = 512 \text{ clockcycles}$$

$$\frac{512}{12000000} = 42.67\mu s$$

Thus the Timer2 ISR will be executed approximately once every $42.67\mu s$. The TMR2 ISR is mainly responsible for sending data to, and receiving data from, the nRF2401 transceiver. Since all data transfers occur in a serial fashion a '*bit-banging*' routine was implemented using the ISR to generate the necessary clock pulses to clock the data bits in/out.

The '*bit-banging*' method uses general I/O lines to emulate a serial port. It is the users responsibility to provide the correct number of transitions to obtain the desired waveform and to ensure that the timing requirements (particularly setup and hold times for reading and writing data) are met. Due to the nature of this specific '*bit-banging*' method implementation the data throughput is not very high, but it doesn't place a great strain on the MCU and provides a good solid timing framework with which to operate the nRF2401. The nRF2401 requires several settling delays during operation which are easily incorporated into this framework.

The timing diagrams for the nRF2401 ([25] p. 25-29) and the PIC18F2550 Timer2 Module notes ([6] p. 135-136) were consulted to develop the firmware functions to enable communication between the PIC18F2550 and the nRF2401.

Thus, apart from communicating with the nRF2401, the TMR2 ISR is also responsible for the following functions, (shown in Figure 10.2, discussed in Section 10.2.2) :

- After an update is scheduled to take place a random time is assigned to it (CSMA), the ISR decrements and checks whether that time has run out.

- Before entering Shockburst RX from standby mode the nRF2401 needs a certain amount of settling time, the ISR decrements and checks whether this time period has expired.
- After performing a Shockburst TX, the nRF2401 needs a certain amount of settling time while the transmission completes, the ISR decrements and checks whether this time period has expired.

The interrupt-on-change interrupt, or INT0 external interrupt, is connected to the DR1 pin of the nRF2401. The DR1 pin is the pin that the NRF2401 uses to signal the successfull reception of a shockburst packet, thus everytime this interrupt routine is called the MCU can start clocking out the packet payload from the nRF2401.

8.2.4 PIC18F2550 nRF2401 Firmware Design

nRF2401 2.4 GHz Transceiver Configuration
At this point everything is in place and the configuration of the transceiver could be performed. Being carefull to give the nRF2401 more than 3ms to settle after power on, the configuration mode can be entered by setting CE = 0 and CS = 1. The configuration consists of shifting a 144 bit configuration word into the configuration register MSB first on positive CLK1 edges. Shifting in the full 144 bit configuration word is only necessary for the initial configuration following a power up. Once the wanted protocol, modus and RF channel are set by the initial configuration, only one bit (RXEN) is shifted in to switch between RX and TX. New configurations are enabled on the falling edge of CS, in other words when leaving configuration mode; then the nRF2401 updates the number of bits actually shifted in during the last configuration.

The 144 bits consist of many configuration words which are used to set the following characteristics of the nRF2401:

- Operating channel frequency
- RF output power
- Enable one or two channel receive mode
- Communications mode (Shockburst or direct)
- Enable on-chip CRC generation/checking
- 8 or 16 bit CRC
- Number of address bits (both channels)

- Up to 5 byte channel 1 address
- Length of data payload section for channel 1
- Up to 5 byte channel 2 address
- Length of data payload section for channel 2
- RF maximum data rate
- Crystal frequency
- Enable RX or TX operation

For use in the SWARM network node the nRF2401 transceiver was set up to operate in Shockburst mode, have the maximum RF output power of 0dBm, operate in one channel receive mode (instead of receiving on two channels at two different frequencies), use a 16 bit CRC, a 24 bit channel address, and a maximum data rate of 1Mbps.

Most note worthy of the configuration settings is the transmit and receive address used and the 16 bit CRC. The 24 bit address used, namely 1100'1110'0000'0101'0100'1001, is the least autocorrelated binary sequence[1] one can create from 24 bits. This lowers the probability that a false address detection occurs as a result of noise in the air. The 16 bit checksum on the other hand can catch all the following types of errors :

- All single and double errors
- All errors with an odd number of bits
- All burst errors of length 16 bits or less
- 99.997% of all 17 bit errors
- 99.998% of all errors $\geq$ 18 bits

Thus, by combining the 24 bit address with a 16 bit CRC should effectively result in zero false detections.

Figure 8.5: Data packet set-up

[1] See http://www.cecm.sfu.ca/ jknauer/labs/

nRF2401 2.4 GHz Transceiver Data Packet Structure
The data packet as used in Shockburst communication and shown in Figure 8.5 can be divided into 4 sections. These are:

1. Preamble - In Shockburst mode the preamble is added and removed to the packet structure automatically.

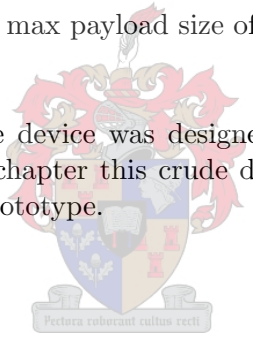
2. Address - The address field is required in ShockBurst mode, can be 8 to 40 bits in length and is automatically removed from received packet in ShockBurst mode.

3. Payload - The data to be transmitted. In ShockBurst mode the payload size is 256 bits minus the address and CRC bits.

4. CRC - The CRC is optional in ShockBurst mode, can be 8 or 16 bits in length and gets removed from the received output data automatically in ShockBurst RX.

Thus, using the configuration as set in Section 8.2.4, the data packet has an address size of 24 bits, a max payload size of 216 bits (27 bytes), with a CRC of 16 bits.

In this chapter a prototype device was designed and constructed using a router board. In the next chapter this crude device's design is formalized and refined into the final prototype.



Chapter 9

Final Prototype: Hardware Design

9.1 Overview

The final prototype incorporates all the components, design aspects and solutions of the previous prototype, except that it only contains one transceiver, since two were only required to enable testing on a single board. The result is a schematic design (Appendix A), that is used for design and layout of the Printed Circuit Board (PCB). Refer to Appendix B for PCB design considerations and layout. The schematic design is a logical representation of the hardware device that includes all components and electrical connections.

The Easily Applicable Graphical Layout-Editor (EAGLE) software application was used to create the schematic design. EAGLE comes standard with libraries full of predefined components, standard footprints, and allows the user to relatively easy create components which don't exist in the library. However, EAGLE was used primarily due to the fact that it is available in a Light Edition (Freeware). In this edition all normal features are enabled, the only restrictions being :

- The board area is restricted to 100 x 80 mm (about 3.9 x 3.2 inches). It is not possible to place packages and draw signals outside of this area.
- Only two signal layers can be used (no inner layers).
- A schematic can consist of only one single sheet.

The prototype being designed only contains a few components, and falls well within these restrictions.

The following steps are typically performed in order to create the schematic design:

1. Lookup of components in existing libraries; creation of components which cant be found.
2. Placement of logic symbols in schematic.
3. Creation of electrical connections between components using nets (explained below) or wires.
4. Verification of schematic design. Verification is done by using *Eagle's Electrical Rule Checker (ERC)*. The *ERC* examines the schematic design for electrical inconsistencies (short circuits, floating pins etc.) and drafting inconsistencies (duplicate designators, unconnected net labels etc.)

The following sections describe the design of the final prototype device. Although the schematic entries are discussed individually, signal connections exist between the schematic entries using *nets*. Nets with the same name define an electrical connection, thus connecting two pins or wires which might be shown on different parts of the schematic. The complete circuit schematic can be found in Appendix A.

9.2 User I/O Interface

The only user I/O provided are in the form of LEDs and two buttons. One button functions as a reset button to restart the SWARM network node and the second button is used to indicate the presence of a host PC connection.

9.2.1 LEDs

Light-emitting diodes (LEDs) are used to show the device's current status. A red LED is connected to the power supply and used to indicate whether the node is powered or not. Two of the four green LEDs indicate the current communications activity (RX or TX). The third acts as a placeholder for a relay connection, since only some boards have the relay circuitry used to power auxilary hardware attached, and is used to indicate the status of this relay had one been attached. The fourth LED indicates whether or not the node is currently actively connected to other network nodes.

These four green LEDs are connected to the normal I/O pins of the PIC18F2550. The maximum output of a PIC18F2550 I/O pin is 3mA [6]. Thus the current through the LEDs was limited at 2.75mA. At a forward current (I_f) of 2.75 mA the LEDs have a forward voltage (V_f) of approximately 1.93V.

This implies a voltage drop of 2.75V across the series current limiting resistor (see schematic entry of Figure 9.1). The value of this resistor should then be :

$$R = \frac{V}{I} = \frac{2.75}{2.75 \times 10^{-3}} = 1000\Omega$$

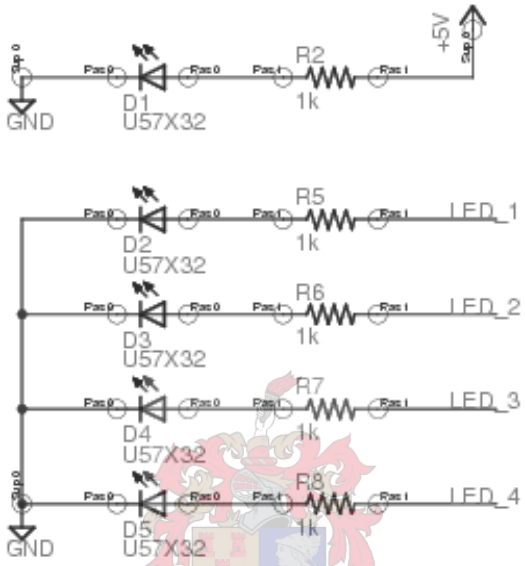


Figure 9.1: Schematic for LED circuit.

9.2.2 Manual Override Button

This button was added to enable the SWARM network node to automatically activate when not connected to a host PC. By pressing the button the node will know it is to be connected to a host PC and enable all USB functionality and await input from the user terminal. If this button is not pressed the node is setup as a self-sufficient node with no USB functionality. Since a self-sufficient node would have to activate itself (no user input enabled), an automatic default activation is provided (discussed in Section 10.9).

The manual override button works by creating an active low signal, when pressed, on a pin which is usually pulled to an active high signal. The resistor R4 is there to limit the amount of current into the pin when the button is in the idle position, and offer enough resistance that when the button is pressed any resistance offered by the switch is negligible in comparison, thus

setting the MCU input pin equal to ground.

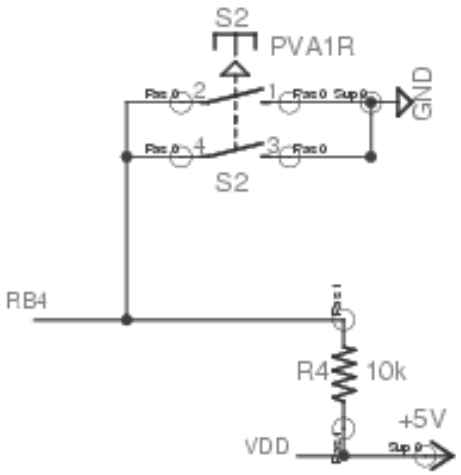


Figure 9.2: Schematic entry for override button circuit.

9.3 Reset Generation

The PIC18F2550 can differentiate between various kinds of Reset:

1. Power-on Reset (POR)
2. MCLR Reset during normal operation
3. Watchdog Timer (WDT) Reset (during execution)
4. Programmable Brown-out Reset (BOR)
5. RESET Instruction
6. Stack Full Reset
7. Stack Underflow Reset

The MCLR pin provides a method for triggering an external Reset of the device. A Reset is generated by holding the pin low. These devices have a noise filter in the MCLR Reset path which detects and ignores small pulses. Thus it is not necessary to implement a Schmitt trigger to create a switching voltage less susceptible to noise. A pushbutton is used to create the active low signal required to reset the PIC18F2550 in the event of firmware failure.

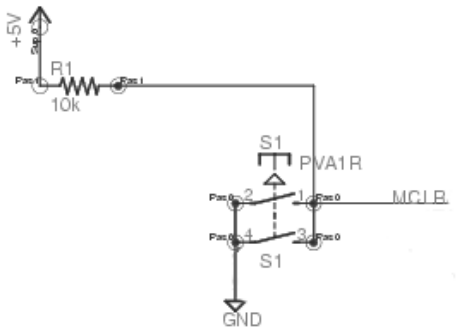


Figure 9.3: Schematic entry for reset generation circuit.

9.4 Power Supply

Table 9.4.1 shows the maximum power supply current required by the components in the design (from datasheets [6], [25], [29], [30], [31]).

Component	Max. supply current required (mA)
Microchip PIC18F2550	50
Nordic nRF2401	15
LEDs	8
TPPM0301 Voltage Regulator	5
Voltage Level Shifter	2
*Relay	40
Total	120

Table 9.4.1: Maximum Power Consumption

The maximum current that can be supplied by the USB port is 500mA. The maximum power supply current requirement of all the components is estimated to be 120mA (Table 9.4.1 - * only present if the SWARM network node is setup as a PowerRelay node), therefore the device can receive all its power from the bus without the need for an external power supply.

The schematic entry for the power supply, regulation and filtering is shown in Figure 9.4. A +5V and +3.3V power supply is required. The +5V is supplied by the USB bus, and +3.3V can be generated from this by using a voltage regulator. The TPPM0301, a single chip voltage regulator from Texas Instruments was selected. This is a low-dropout regulator with auxiliary power management that provides a constant +3.3V supply at the output capable of driving a 400-mA load.

To filter low-frequency noise caused by power supplies, electrolytic capacitors must be placed in the power supply circuit. For this purpose, 5 electrolytic capacitors (C5,C6,C7,C8 and C10) are used for the +5V and +3.3V supplies. These capacitors also provide extra current when needed, such as when many outputs switch simultaneously in a circuit.

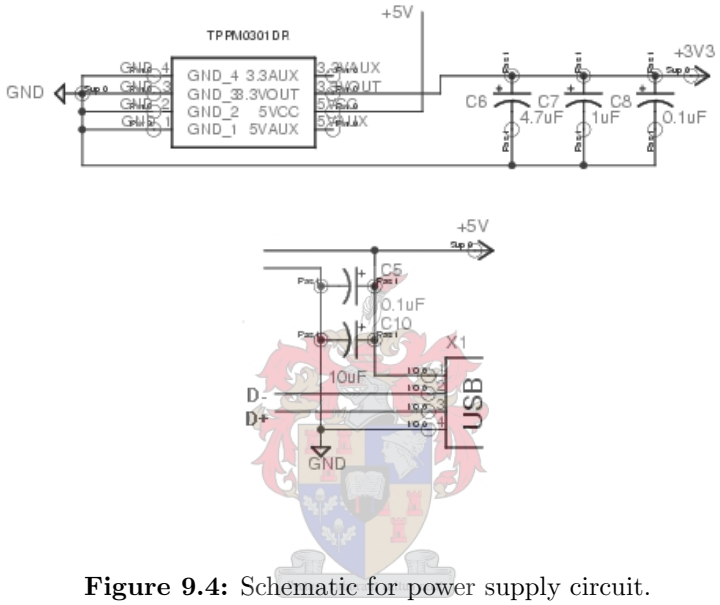


Figure 9.4: Schematic for power supply circuit.

9.5 Microchip PIC18F2550 Microcontroller

The PIC18F2550 requires very few external components. All power supply pins (Vcc, Vss) are connected to the power lines shown (Figure 9.6) in the power supply schematics ($+5V$, GND) where they are already decoupled with appropriate capacitors. The USB data lines (D+ and D-) coming from the USB connector, as shown in the power supply schematic, are connected to the D+ and D- pins on the PIC. The V_{USB} pin of the PIC is provided in case an external +3.3v regulated power supply is to be used for driving the USB Serial Interface Engine. Here the internal USB +3.3V regulator was used, thus the pin had to be connected to ground via a 470nF capacitor.

The MCLR pin forms part of the reset circuitry already mentioned in Section 9.3, but is also used in programming the PIC. This pin, together with

the other two programming pins, PGD and PGC, were connected to a 6-pin male header even though the ICD2 programmer requires a RJ11 female connector, in order to save space on the PCB. Thus a small adpater circuit was constructed consisting of a female RJ11 plug connector and a 6-pin female header .

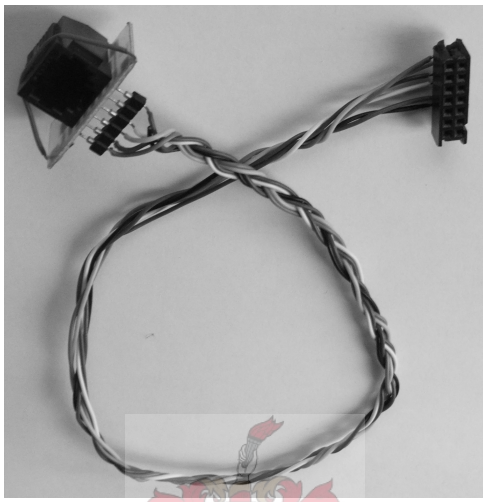


Figure 9.5: ICD2 programmer RJ11 connector converter

The PIC18F2550 supports several oscillator modes, the 'HSPLL' or 'High-Speed Crystal/Resonator with PLL enabled' mode was selected. Thus the MCU has a 48Mhz internal clock speed, and full speed USB functionality. A crystal or ceramic resonator of 20MHz is then required to be connected between the OSC1 and OSC2 pins to establish oscillation, as well as two 15pF capacitors ([6], p. 23-32).

9.6 Voltage Level Shifter

The Texas Instruments SN74CBTD3384 provides ten bits of high-speed TTL-compatible bus switching. The low on-state resistance of the switches allows connections to be made without adding propagation delay. A diode to VCC is integrated on the die to allow for level shifting from +5V signals at the device inputs to +3.3V signals at the device outputs [30].

These devices are organized as two 5-bit switches with separate output-enable (OE) inputs. When OE is low, the switch is on, and port A is connected to port B. When OE is high, the switch is open, and the high-impedance state exists between the two ports.

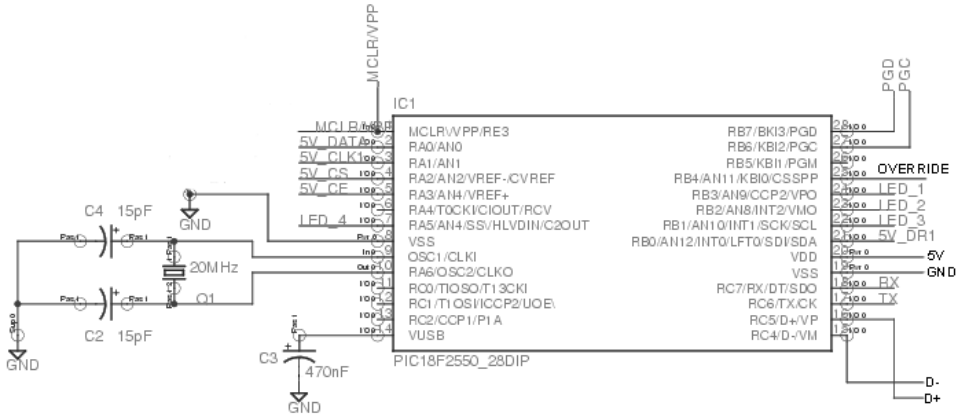


Figure 9.6: Schematic entry for PIC18F2550 circuit.

The level shifter is used to adjust the signal levels between the PIC and the nRF2401 transceiver. The 5 I/O pins from the PIC (+5V) are all routed to one of the pins labelled 'zAx' (where z denotes port 1 or 2, and x denotes a bit 1 to 5) on the level shifter, and the pins from the nRF2401 (+3,3V) are all routed to one of the pins labelled 'zBx' (where z and x are equal to the signal's corresponding +5V connection's z and x) on the level shifter. These signal pairs can be seen named 5V_XXX and 3V3_XXX in Figure 9.7 .

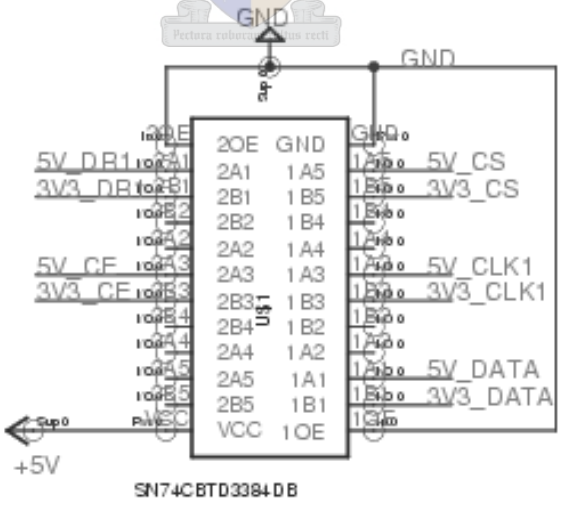


Figure 9.7: Schematic for level shifting circuitry

9.7 Nordic Semiconductors nRF2401

Since the Nordic nRF2401 2.4GHz transceiver was bought as a complete module the unit required no external components. The module purchased was the RF-24G manufactured by Laipac Tech. The module comes out with a non standard 1.25mm spacing between its pins, thus an additional break-out board was purchased which has a standard 0.1 inch spacing.

The nRF2401 has 5 data lines coming from the level shifter, these 5 lines are routed straight to a 10pin male header placed on one side of the PCB. This allows for easy attachment of the breakout board as shown in Figures 9.8 and 9.9.

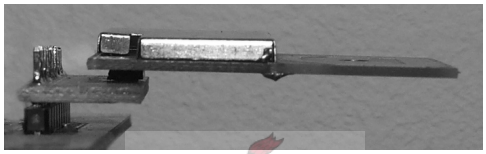


Figure 9.8: nRF2401 connection to 10-pin male header (side view)

Figure 9.9: Schematic of nRF2401 10-pin male header circuitry and photo of nRF2401 connected to it's header.

9.8 Relay Control Board

The SWARM network node can be used to act as a remote on / off switch by means of a relay, in which case the third LED on the PCB (the relay placeholder LED) won't be mounted on the PCB, instead a wire connecting the PCB board to the Relay Control Board (RCB) would be connected in it's place.

The core of the RCB circuit is a simple npn bipolar inverter circuit used as a switch, which uses a JRC-27F Subminiature 5V Dip Relay [29] and a 2N222 transistor [32]. This network can be simplified to the symbolic circuit shown in Figure 9.10 which was used to perform the necessary circuit design analysis .

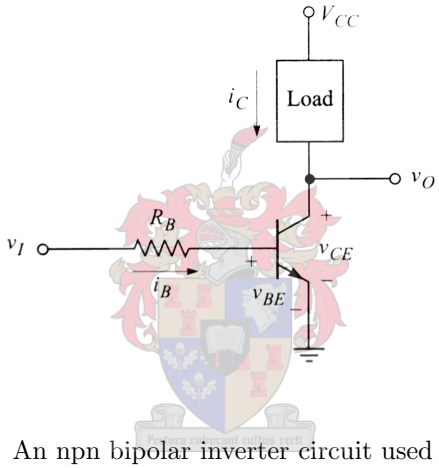


Figure 9.10: An npn bipolar inverter circuit used as a switch [8].

The V_{CC} shown is a 5V source drawn straight from the USB port's power lines. The relay is represented simply as 'Load'. For $v_I = 0$, the transistor is cut off, $i_B = i_C = 0$, $v_O = V_{CC} = 5V$, and the power dissipated in the transistor is zero. For $v_I = 4.3V$,

$$i_B = \frac{(v_I - V_{BE}(on)}{R_B} = \frac{4.3 - 1.2}{4700} = 0.65957mA$$

Assuming the transistor is in saturation, we find that

$$i_C = \frac{(V_{CC} - V_{CE}(sat)}{R_C} = \frac{5 - 0.3}{122.78} = 38.27985mA$$

If we take the ratio of collector current to base current, we have

$$\frac{i_C}{i_B} = \frac{38.27985}{0.65957} = 58$$

Since the ratio of $\frac{i_C}{i_B} < \beta$, $(40 < \beta < 300)$, the transistor is indeed in saturation, as was initially assumed.

In the RCB the relay was provided with a diode in parallel, this was done since relay coils have a large amount of inductance and when they are released (when the circuit is cut off) they generate a very large voltage spike which needs to be clamped or it will damage the rest of the circuit .

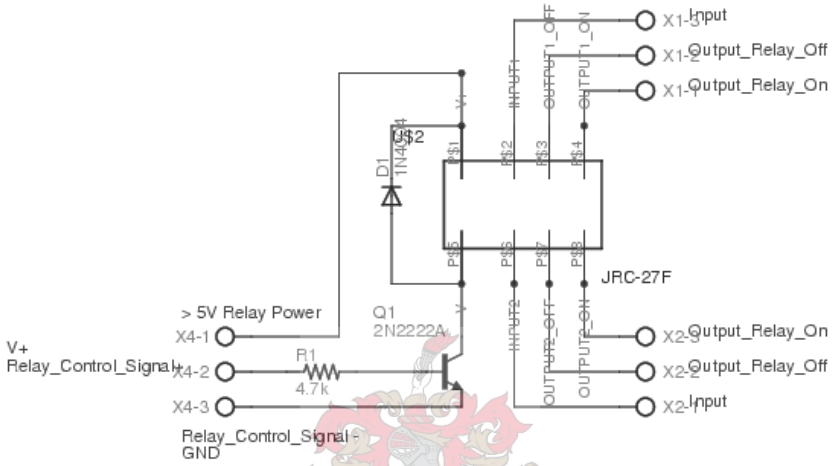


Figure 9.11: Schematic of relay control board.

This chapter has mentioned and motivated all the schematic design choices made, and resulted in a complete PCB design being created, as shown in Appendix A. The next chapter discusses the firmware which had to be developed for the PCB components to give the SWARM network node all it's required functionality.

Chapter 10

Final Prototype: Firmware Design

This Chapter describes the firmware as created for the PIC18F2550 microcontroller. The actions performed by the main loop of the microcontroller firmware are illustrated in the flowchart of Figure 10.8. Upon power up, the first process executed is InitialiseSystem().

10.1 InitialiseSystem()

InitialiseSystem() sets the MCU up in such a way that it is capable of performing all it's functions as central processor to the SWARM network node. The initialisation process is divided into two sections, initialising the protocol related variables and initialising and setting up the MCU itself. In summary the following actions are performed :

- Disabling the comparator on port A.
- Disabling the A/D convert on port A.
- Set the pins of port A and B as either output or input pins.
- Initialize the USB driver.
- Load the 144 bits nRF2401 configuration data into a temporary buffer.
- Set all timers to zero.
- The Routing Table (RT) is cleared, and all entries set to NUL_CLIENT.
- The Routing Temporary Table (RTT) is cleared, and all entries set to NUL_CLIENT.
- The data transmsission queue is set to zero.

- All node status flags are set to zero.
- Message types are loaded.
- Setup TIMER 0 to fire the interrupt every 0.699ms.
- Setup TIMER 2 to fire the interrupt every 42.5us.
- Set INT0 External Interrupt to fire on change.
- Enable all configured interrupts.
- Clear all buffers used.
- Load seed for random number generator.
- Perform initial nRF2401 configuration, placing the device in Shockburst RX.

After initialisation the firmware will remain in a loop, see Figure 10.8. From here all tasks are either interrupt driven (data receptions and transmissions) or initiated by the user (user I/O).

10.2 Interrupt Handling

The firmware makes use of three interrupts, of which two are timed interrupts (Timer0 and Timer2) and the other is a interrupt-on-change interrupt (INT0).

10.2.1 Timer 0

Timer0 ISR will be executed approximately once every 0.7seconds. To boost this time up to once every 2.1 seconds the ISR was written in such a way that it is only executed every third time the interrupt gets triggered. The TMR0 ISR is responsible for scheduling updates, decreasing the validity of RT and RTT entries, merging subnets, and setting up links between subnets. A rough outline is given in Figure 10.1.

The first thing the ISR does is increase the NodeLifeTimer which is used to calculate how long a system has been stable, and reset the TX bracket variables. The TX bracket variables are used to split the 2.1 seconds between interrupts into 21 possible message transmission slots as will be discussed in more detail in Section 10.3. The other main processes are discussed individually in the following subsections.

UpdateMsgscheduler()

UpdateMsgscheduler makes use of the mobility variable to adjust the frequency with which the node sends out updates, but this feature only becomes active if the node has been stable long enough. This stability period is governed by a variable *UpdateThreshold*, a constant declared in the file *ProtocolConstants.h*. The maximum time between two consecutive updates being sent out is governed by a variable called *Max_updateThreshold* which has a value between 1 and 512 depending on how mobile the node is (value of the mobility variable).

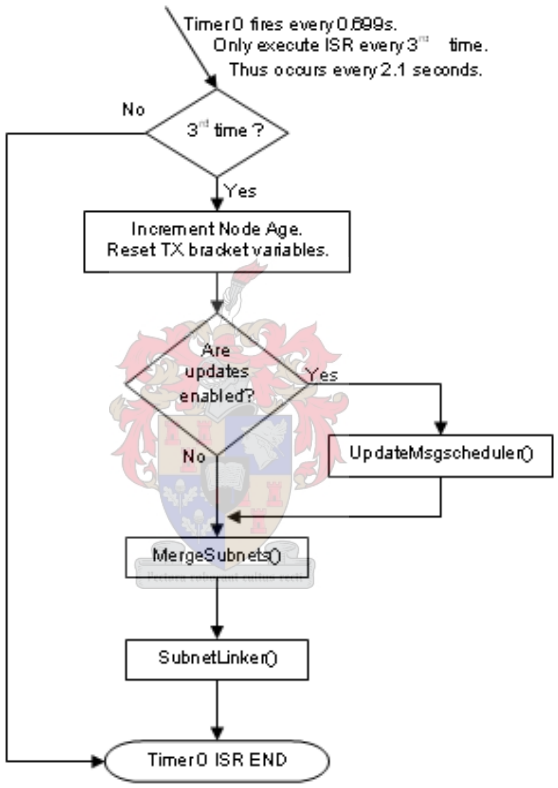


Figure 10.1: Timer0 ISR overview

Should the system not be stable (an RT entry was added or removed recently for example), then the update frequency won't be adjusted and will remain fixed at 1. This was done to make sure that any changes made are reflected throughout the system as quickly as possible, if update frequencies are too low then the changes propagate through the system too slow, which causes inconsistencies in the network.

If the Node is classified as being a LANedge link, then the update frequency also won't be adjusted and will remain fixed at 1. This was done to make sure that the link between subnets does not expire simply because nodes in the one subnet have a slow update frequency and nodes in the other subnet have a very fast update frequency.

The second half of UpdateMsgscheduler() only occurs everytime an update is scheduled to take place. When an update is scheduled to take place the node sequence number is incremented, a random transmit bracket is chosen as part of the CSMA scheme used in transmitting updates (discussed in Section 10.3), the validities of all RT and RTT entries gets decreased, and the type of update to be sent is selected.

A difference was created here between the protocol simulation and the firmware, in order to solve a problem with the size of the update packets. In the simulation bigger packets were used, thus it was possible to send an entire node RT in one packet. This made processing update messages much easier since all the other node's route information was available simultaneously.

SOURCE	8 bits
UPDATEMSG	8 bits
SEQUENCE	8 bits
SUBNET	8 bits
SUBNETEDGE	8 bits
RTINDEXER	8 bits
SRC(n)	8 bits
DEST(n)	8 bits
NEXT(n)	8 bits
PREV(n)	8 bits
HOPS(n)	8 bits
SEQ(n)	8 bits
LANEDGE(n)	8 bits
VALIDITY(n)	8 bits
SUBNET	8 bits
Total Bytes Required	6+(n*9)Bytes

Table 10.2.1: Update LAN Message Structure

In the firmware however the packets were only big enough to contain two RT entries along with the usual packet overhead. This meant that three update messages are needed simply to transmit a node's entire RT. This solution had further complications though; if only two entries are sent at a time, how

would the LANedge creation and maintenance process described in Sections 5.6 & 5.7 be able to function? This process relies on one LANedge node having up to date information of all the RT entries linked to it's opposite LANedge node in the other subnet all at once.

To solve this problem two types of update messages were created, the *UpdateMsgSubnet* and the *UpdateMsg*, of which the *UpdateMsgSubnet* message is used by nodes to advertise the contents of their RT to those nodes around them, albeit two RT entries at a time then. The *UpdateMsg* is sent out every fourth time, and contains a summary of a node's RT. It contains only the destination, next nodal hop, and LANedge status of each RT entry. See Tables 10.2.1 and 10.2.2 for details concerning these message structures.

Msg_Type_Id	8 bits
Node_Id	8 bits
Node_Seq	8 bits
Node_LAN	8 bits
Node_LANedge	8 bits
Node_numberofRTentries	8 bits
(n) SOURCE	8 bits
(n) DESTINATION	8 bits
(n) LANEDGE	8 bits
(n+1) SOURCE	8 bits
(n+1) DESTINATION	8 bits
(n+1) LANEDGE	8 bits
(n+2) SOURCE	8 bits
(n+2) DESTINATION	8 bits
(n+2) LANEDGE	8 bits
(n+3) SOURCE	8 bits
(n+3) DESTINATION	8 bits
(n+3) LANEDGE	8 bits
(n+4) SOURCE	8 bits
(n+4) DESTINATION	8 bits
(n+4) LANEDGE	8 bits
(n+5) SOURCE	8 bits
(n+5) DESTINATION	8 bits
(n+5) LANEDGE	8 bits
Total Bytes Required	10 Bytes

Table 10.2.2: Update LANedge Message Structure

Thus four updates are sent out before one update cycle is complete. Three of the four update message are used to check, update and maintain the nodal routing tables, and the fourth update message is used to check, update and maintain the network layout (subnet layout and interconnections).

This scheme works well even if packet losses are high. For instance, for one RT entry to expire due to packet loss it would mean the update containing that specific RT entry, which is only present in every fourth update message sent, and whose packet is sent randomly according to a CSMA scheme (discussed in Sections 5.11 and 10.3) has to be lost 2-4 times (depends on mobility) in a row.

MergeSubnets()
This implements the process of recombining two LANs which would function optimally as one LAN, as mentioned in Section 5.8. Briefly, if an *UpdateMsg* is received, and a link between this node's subnet and the updating subnet already exists, but this node belongs to the smaller of the two subnets and both the subnets combined still form a subnet of allowable size, then a flag and timer will be set, indicating that this node is moving to the larger subnet, and updates will be turned off.

When the timer expires all knowledge of the node would have expired in the RT's of those nodes around it, unless they were already performing adaptive updating, in which case it takes longer. Either way MergeSubnets() is executed and the Node changes it's subnet to that of the other subnet, and enables updates again. When nodes around it now pick up this new node they will reset their adaptive updating (since changes are being made to subnet), thus ridding themselves of the old RT entry quicker.

SubnetLinker()
SubnetLinker() is only executed if a LANedgelink process has already been started, as was described in Section 5.6. Briefly it means that a node had picked up an update message from an unknown subnet, and had set two timers, one to monitor if it's nomination that it sent out to become the link node between the subnets has been rejected or lost, and the other to monitor whether the other node's nomination to become the lanedge link is still valid.

Thus all SubnetLinker() does is to decrement these timers, and to send out one last nomination for itself to become the lanedgelink when it's previously sent nomination times out.

10.2.2 Timer 2

The Timer2 ISR will be executed approximately once every 42.67μs. The TMR2 ISR is mainly responsible for sending data to, and receiving data from, the nRF2401 transceiver. All communication is performed using a *bit-banging* method where data is loaded bit by bit. A set of flags control the program flow inside the ISR and governs what is loaded next.

The example below is used to illustrate the initial configuration procedure (shown as *Configure_nRF2401()* in Figure 10.2) performed by calling *NRF_INITIAL_C ONFIG()* (shown below) from the InitialiseSystem() procedure during startup.

```
void NRF_INITIAL_CONFIG(void)
{
  // configure nRF2401 for the first time
  // enable configuration of nRF2401 device
  Flags.Bit.Config = 1;
  // enter configuration mode of nRF2401,
  // automatically includes a delay of longer than 5us
  Flags.Bit.Config_Setup = 1;
  // first time, thus give it default setup
  Flags.Bit.Config_Default = 1;
  // make sure all other flags in resting position
  Flags.Bit.Config_RX = 0;
  Flags.Bit.Config_TX = 0;
  Flags.Bit.Config_Setup_RX_Done = 0;
  Flags.Bit.Config_Setup_TX_Done = 0;
  Flags.Bit.Config_Setup_Initial_Done = 0;
  // MSB will be loaded first
  config_bit = 119;
}
```

Thus the three flags that will enable configuration of the nRF2401 are set, as well as the *config_bit* variable which indicates at which bit to start loading. The piece of code in Timer2 which actually clocks in the data is shown below:

```
...
.... start of Timer 2 ISR

// if busy configuring nRF2401
if ( Flags.Bit.Config )
{
  if ( Flags.Bit.Config_Setup )
  {
    //Enter Config Mode
    RF_CE = 0;RF_CS = 1;
```

```
  Flags.Bit.Config_Setup = 0;
}
// load initial configuration into nRF2401
else if (Flags.Bit.Config_Default )
{
  // make data available on RF_DATA while
  // performing low transition of clock cycle
  if ( Flags.Bit.RF_Clk1_High )
  {
    RF_CLK1 = 0;                // actual pin going to nRF2401
    Flags.Bit.RF_Clk1_High = 0; // flag used to track clock status
    // if done clocking in config bits
    if ( config_bit < 0 )
    {
      // put flags used back in default state
      Flags.Bit.Config_Default = 0;
      Flags.Bit.RF_Clk1_High = 1;
      //Configuration is actived on falling edge of CS
      RF_CE = 0; RF_CS = 0;
      Flags.Bit.Config_Setup_Initial_Done = 1;
    }
    else
    {
      // clock next data / config bit out
      RF_DATA_OUT = RF_LOAD_CONFIG(config_bit);
      config_bit -= 1;
    }
  }
  // data still available on RF_DATA, now
  // perform high transition of clock cycle
  else
  {
    RF_CLK1 = 1;
    Flags.Bit.RF_Clk1_High = 1;
  }
}

...
.... and
...

else if ( Flags.Bit.Config_Setup_Initial_Done )
{
  Flags.Bit.Config = 0;
  state = Initial;
  //Set Mode to Active
  //Start monitoring the air by setting Mode to Shockburst RX
  RF_CE = 1; RF_CS = 0;
  Flags.Bit.Config_Setup_Initial_Done = 0;
}

...
.... rest of Timer 2 ISR
```

The code sets the nRF2401 into configuration mode, thereafter the *Flags.Bit .Config_Default* part iterates until all bits have been clocked out. It does this by setting the clock low and loading the next bit of data on one iteration, then setting the clock high on the next iteration with the same bit still loaded, repeating until all bits have been loaded. When it is done, the *Flags.Bit.Config_Setup_Initial _Done* part gets run, which resets the last of the flags used and puts the nRF2401 into active Shockburst RX mode.

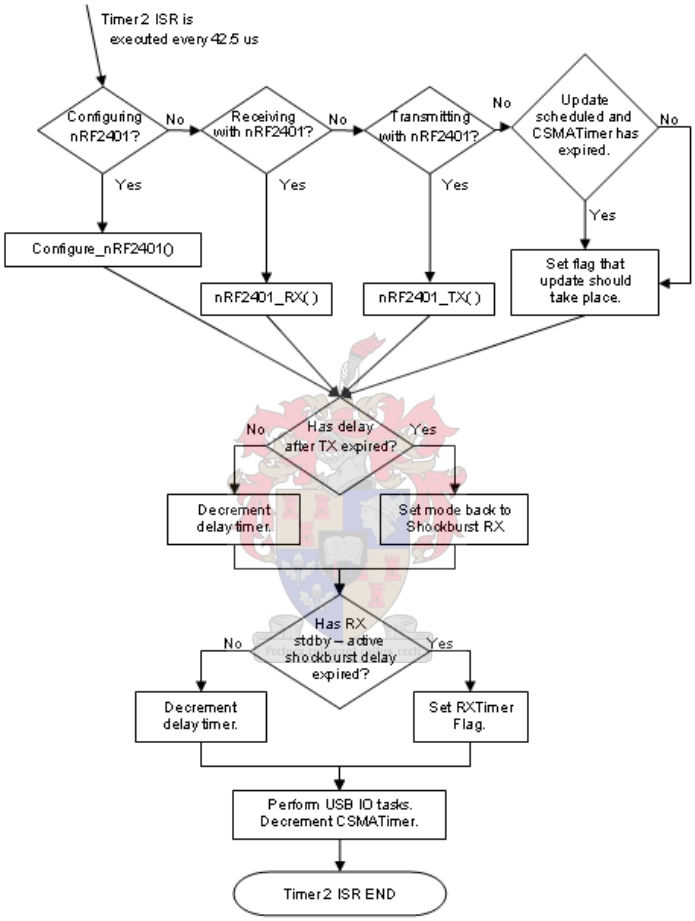


Figure 10.2: Overview of the Timer2 ISR

The other two bit-bashing routines, *nRF2401_RX()* and *nRF2401_TX()* both function in a very similar way. The only difference is that they did not have a preloaded construct (eg like RF_LOAD_CONFIG(*bit_to_clock_out*) used above) containing all the bits they simply need to name to clock out, thus byte to bit conversion functions had to be added. A function had to be

written to extract single bits out of an unsigned char (a byte) when writing to the nRF2401, and functions to set the bits in the buffer to either 0 or 1 depending on what was clocked out of the nRF2401. These are shown below:

```
/*******************************************************
**     bit_clear(byte to change, bit to change )   ***
*******************************************************
**   eg. if var = 1111 1111 then bit_clear(var,2)  ***
**   would return 1111 1011 ***
*******************************************************/
unsigned char bit_clear(unsigned char var, char c )
{
  unsigned char temp = var;
  temp &= ~(1<<c);
  return temp;
}

/*******************************************************
**    bit_set(byte to change, bit to make 1 )     ***
*******************************************************/
unsigned char bit_set(unsigned char var, char c )
{
  unsigned char temp = var;
  temp |= (1<<c);
  return temp;
}

/*******************************************************
**      bit_test(byte to test, bit to test )       ***
*******************************************************
**  test the bit specified, return 1 if 1, 0 if 0   ***
*******************************************************/
short bit_test(unsigned char var, char c ) {
  unsigned char b,t;
  b = 1 << c;
  t = (var&b) >> c;
  if (t == 1) return 1;
  else return 0;
}
```

The rest of the ISR sets flags used to trigger events in the main() loop or in other interrupts. When an update is scheduled and the CSMATimer has expired a flag is set to indicate that the random timer period before the update can be sent has expired, and the next time the main() loop is in it's

Idle state the update can be put into the nRF2401's transmit queue and transmitted .

ShockBurst™ TX:

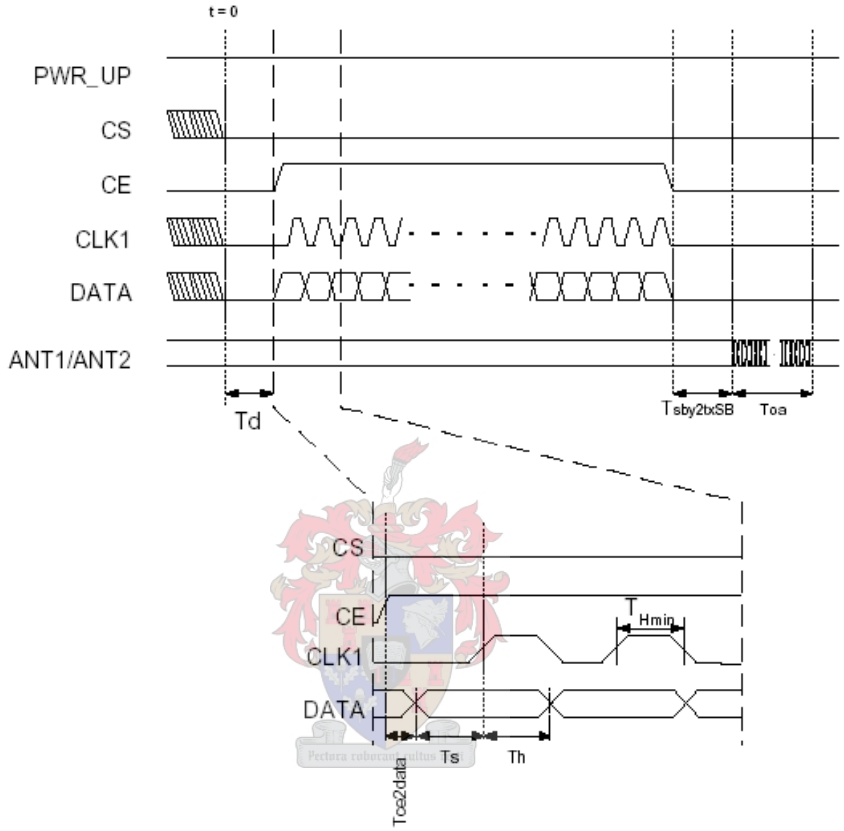


Figure 10.3: Timing of Shockburst in TX.

The nRF2401 also has two delays, one relating to Shockburst RX and one relating to Shockburst TX, to heed for proper functioning, that are both taken care of by Timer2. The timing diagram related to performing a Shockburst TX is shown in Figure 10.3. After being setup as a transmitter, loaded with the data to transmit, and told to transmit, the nRF2401 needs time to settle, *Tsby2txSB*, and time to actually perform the data transmission, *TOA*. Tsby2txSB is given as a maximum of 195μs, and TOA (time on air) can be calculated as follows :

$$T_{OA} = \frac{(number of data bits to TX + 1)}{data rate} = \frac{257}{1000000} = 257\mu s$$

Thus a maximum delay of $452\mu s$ is expected, which equals 11 iterations of Timer2 ISR. It was found that the nRF2401 operated more reliably if this period was increased to $595\mu s$, thus 14 iterations of Timer2 was used. This delay is the one shown in Figure 10.2 as 'Has delay after TX expired?'. When this time period has passed the nRF2401 is set back to Shockburst

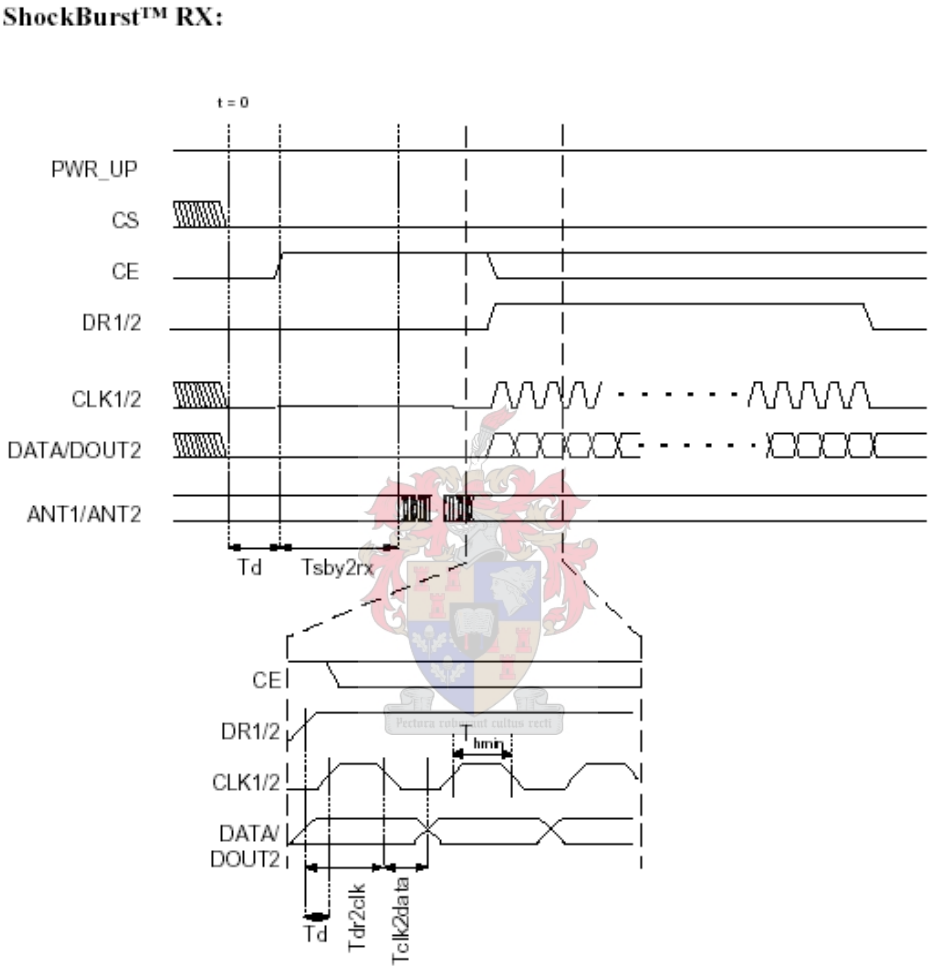


Figure 10.4: Timing of ShockBurst in RX.

RX mode, the default functional mode for a SWARM network node.

The timing diagram related to performing a Shockburst RX is shown in Figure 10.4. After being setup as a receiver, and set into active Shockburst RX mode, the nRF2401 needs a settling time, *Tsby2rx*, before it is ready to receive data or be set into another operational mode. Tsby2rx is given as a

maximum of 202μs, but the datasheet also says that *The CE may be kept high during downloading of data, but the cost is higher current consumption (18mA) and the benefit is short start-up time (*200μs*) when DR1 goes low.* Unless the 202μs given is the longer start-up time no other indication is given as to what this longer start-up time might be.

Through extensive experimentation a value of 595μs was found to provide error-free operation. This delay is the one shown in Figure 10.2 as 'Has RX stdby - active shockburst delay expired?'. When this time period has passed, or when the nRF2401 has successfully received a packet (in which case the timer is cleared) the normal operation of states Initial, Idle and RX can continue. Otherwise these states have to wait until the timer expires before doing any further processing.

10.2.3 INT0 - Interrupt on Change

The interrupt-on-change interrupt, or INT0 external interrupt, is connected to the DR1 pin of the nRF2401. The DR1 pin is the pin that the nRF2401 uses to signal the successfull reception of a shockburst packet, thus every-time this interrupt routine is called data is ready to be clocked out. The ISR allows for clocking out of information under two conditions, when the MCU is in an Idle or Initial state, then a flag is set which will allow the MCU to start clocking out the packet payload from the nRF2401.

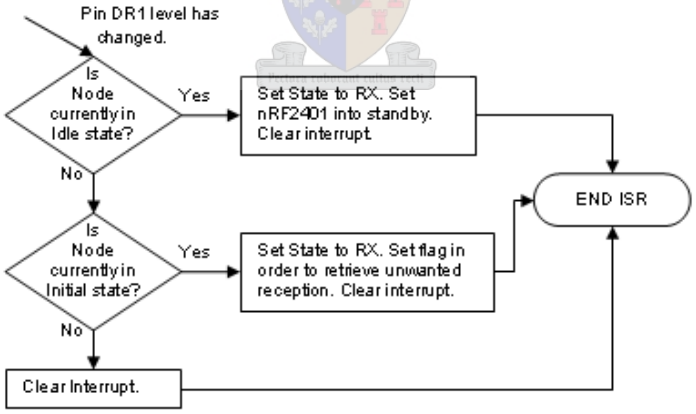


Figure 10.5: INT0 - Interrupt on change ISR.

10.3 SWARM Network CSMA/CA Implementation

Messages sent across the nRF2401 need to be transmitted with a certain degree of randomness, else two nodes which start transmitting at exactly the same time will have an on-air collision, their transmission timers will time-out, a retransmission will occur, but for both nodes simultaneously again, over and over, and in the end neither will ever be successfull. To avoid this situation a carrier sense multiple access with collison avoidance scheme was implemented.

If the nRF2401 can't sense anything in the air to receive, the carrier is assumed available and a transmission may occur if there is a transmission ready and waiting. This in itself is no solution, because if two nodes sensed the carrier as free and transmitted simultaneously we get the problem described earlier. To avoid this problem, each message needs to be tagged with a random transmission time tag.

There are two types of transmissions which occur in the SWARM network nodes, messages sent out periodically (update messages) and the sporadic user initiated application software messages. The update messages are sent out once every 2.1 seconds. When an update gets scheduled it chooses a random transmission time, between 0 and 2.1 seconds, which it assigns to *UpdateTimer*. This is a completely random continuous time chosen in the form of a counter which is decremented in Timer2. When UpdateTimer expires, a flag is set to indicate that an update should be sent as soon as the nRF2401 is available (in it's Idle state).

To send sporadic user messages, the total time available per transmission cycle (2.1seconds), was divided up into as many transmission slots as possible using the average round trip time between two nodes as guideline. Doing it this way there are 21 possible slots for transmission. These slots are created by the CSMA_TX_Block_Creation() process, as described in Section 10.7.

Every message which gets scheduled for transmission gets a random transmission slot tag attached to it of either 1, 2, or 3 transmission brackets. If a message is delayed by three transmission brackets though, then any messages that may result from received messages during this time will also have to be stored until after the delayed message has been sent, in order to prevent interfering with it's transmission process. Thus if the maximum delay is three transmission slots, then a buffer capable of storing at least three messages is needed.

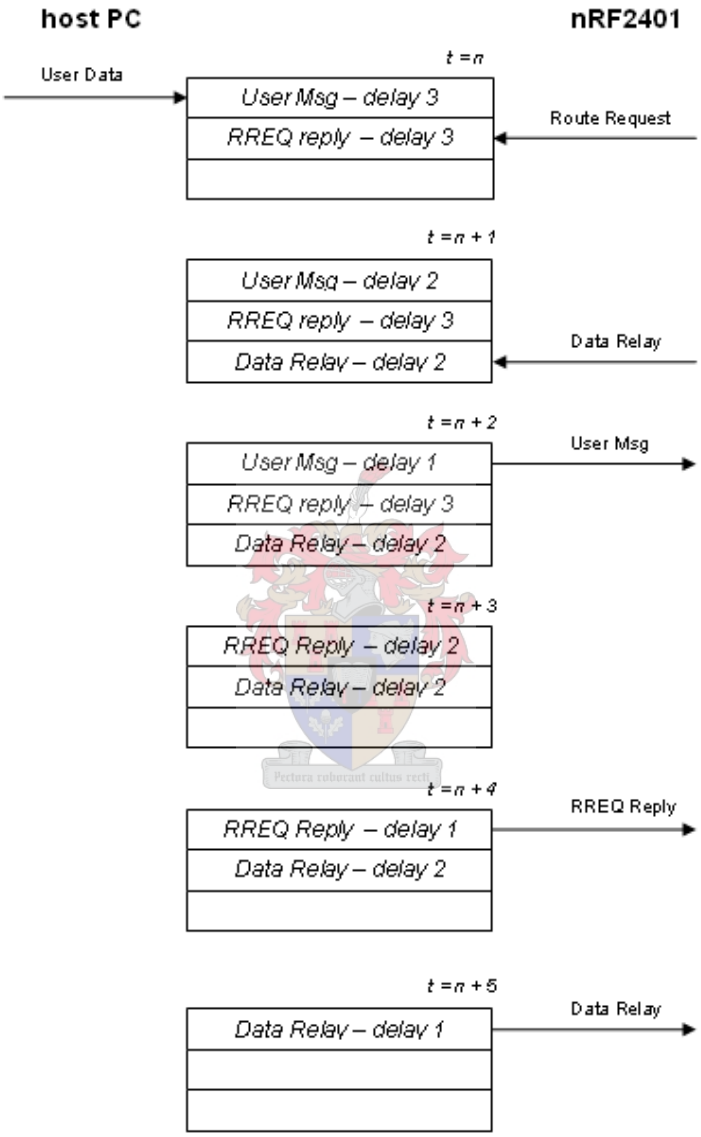


Figure 10.6: FIFO buffer used for nRF2401 transmissions.

The nRF2401 can only be either transmitting or receiving at any one point in time, and has to receive and transmit messages from and to other nodes (forwarding packets), in addition to sending its own packets when prompted by the software application (user messages). All messages to be sent receives a message transmission random transmit tag, this makes for a range of possible reception / transmission scenarios, an example of which is shown in Figure 10.6.

Only the 'topmost' item in the FIFO buffer has it's transmission delay decremented, all other messages in the buffer remain idle until they reach the top. Each message in the buffer is also assigned a time to live (TTL) field, which gets decremented every transmit bracket, if the TTL runs out the message wont ever be transmitted. This was done to prevent a message being delayed for such a long period of time in the buffer that the timers set to determine whether it was lost had already expired by the time it gets transmitted. All timers were also carefully set to cater for the effects of the FIFO buffer, since it could happen that three messages each have a delay of 3 transmission brackets, in which case a message could be delayed for up to 9 transmission blocks in total.

10.4 SWARM Network Node Data Transfers

All data transfers from the PC to the SWARM network node are initiated by the user, the bulk of these being data transfers occurring when a network text message or relay state change message is sent to another node in the network.

10.4.1 USB_msg_parser()

The PIC has a process called *ScanUSB* which forms the core of the *USB_msg_parser()* process block, forming part of the main() firmware loop, shown in Figure 10.8. ScanUSB() checks continuously and if found to be available it will read a 27-byte packet from the *OUT* endpoint associated with the USB CDC class, and transfer it to RAM. Only 27-bytes packets are read from the endpoint since 27-bytes is the maximum amount of data which will ever be sent in a single transfer from the host PC user interface.

Once transferred into RAM the first byte gets scanned to determine what type of data has been received, and what processing should be done on the rest of the data sent across.

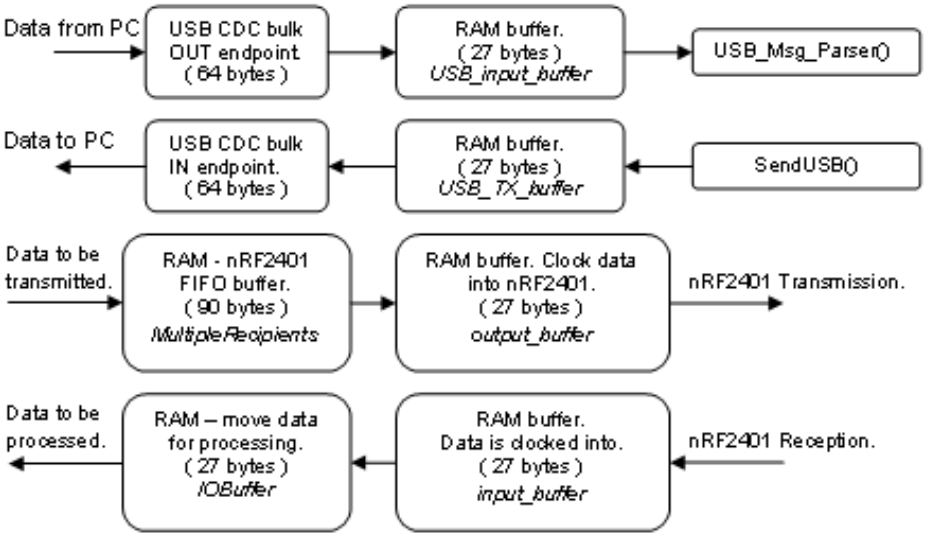


Figure 10.7: Data flow between endpoint, USB buffers, FIFO TX buffer, and RX buffer.

Currently the following different types of data transmission are possible, (shown by first identifying byte, and type of message) :

- 01010100 - Initialize connection between PC and SWARM network node.
- 01000011 - Set mobility + related variables.
- 01001101 - Activate node.
- 01010010 - Send RT across USB.
- 01010011 - Send list of connected nodes across USB.
- 01010101 - User has typed a text message and wants to transmit to another node.
- 01010000 - User wants to transmit a relay control message to another node to set it's relay status.

10.4.2 01010100 ('T') - *ConnectPC()*

This transfer is used to initiate communication between the application software and the SWARM network node. The host PC sends out a byte, 01010100 ('T') and if the SWARM network node replies with a byte, 01011000 ('X') then the application software knows they are properly connected and further communication may commence.

10.4.3 01000011 ('C') - *SetMobility()*

The node essentially needs a name and a mobility value before it can function normally. These values cannot simply be left to the default preprogrammed values if the user intends to activate the node immediately, and have to be set before such activation can take place. The application software provides the necessary functionality to enter these values, and if the user activates the node a byte, 01000011 ('C'), is sent, followed by a byte indicating a mobility value of between 0 and 10 (read as char '0' up to char ':', thus minus 48 to get the correct decimal value), and followed lastly by the node name, also a byte.

Upon successfull reception of these 3 bytes all variables dependant on the mobility variable are set, as well as the node name, and then the SWARM network node replies to the application software with a byte, 01001101 ('M'), and the char it had received as mobility variable (sent back in this way as confirmation that it had been received correctly). If an invalid value was received, a response is sent back to the host pc, also in the form of two bytes, the first 01000101 ('E'), and the second is the char received as mobility value.

10.4.4 01001101 ('M') - *ActivateNode()*

If the mobility variable had been set successfully then activation continues with the application software sending out a byte, 01001101 ('M'), which calls the *ActivateNode()* procedure. ActivateNode() sets up and transmits a hello message, sets the SWARM network node's mode to **online** and enables the periodic broadcasting of update messages.

10.4.5 01010010 ('R') - *DisplayRT()*

This enables or disables whether or not the node should share it's routing table contents with the software application. If enabled it means that the SWARM network node will send one field from it's routing table to the application software everytime the TIMER0 interrupt fires. By only sending one field (one byte) everytime this operation never places additional strain on the USB communication buffers.

10.4.6 01010011 ('S') - *RefreshNodeList()*

Before a message can be sent out the user needs to know who is reachable, when requesting a refresh nodal list the application software sends out this byte, 01010011 ('S'), to which the SWARM network node replies by sending the name of each destination node listed in it's RT. Before each name a byte is attached to indicate that a destination node name follows, thus 01010011 ('S') followed by a destination node name will be sent until all RT entries have been covered.

10.4.7 01010101 ('U') or 01010000 ('P') - *ReceiveUserMsg()*

The application software sends a byte, 01010101 ('U') or 01010000 ('P'), when there is either text message data or relay control information that the user wants to transmit to another node in the network. The second byte received from the application software is the message destination, and the 16 bytes following that contain the actual text data. *ReceiveUserMsg()* sets the message destination, resets all timers used to determine the success of a route request and sets all flags which indicate that a user data transmission is in progress.

Then *ReceiveUserMsgHandler()* gets called. ReceiveUserMsgHandler() forms the starting point of every data transmission performed. Upon reception of a new user data message it will check whether a new route is being opened up, or does a route to the destination already exist. If a route does exist it will load the FIFO buffer and instruct the packets to follow this known route. If no existing route was found, then a route request needs to be sent before any data can be sent. The route request can be sent directly if the destination node is within this node's subnet, else if the destination node is completely unknown then the route request will be sent to all LANedges that this node is aware of. To send a route request an RTT entry needs to first be created, and the FIFO buffer loaded with the actual route request information.

10.4.8 nRF2401 RX / TX

Data to be sent from the SWARM network node across the nRF2401 gets put into the FIFO buffer as described in Section 10.3. When scheduled for transmission the data is transferred from the FIFO buffer into *output_buffer* from where that data is clocked out into the nRF2401 using Timer2 (as described in Section 10.2.2) and subsequently transmitted.

Data received on the nRF2401 is also clocked out using Timer2(as described in Section 10.2.2) and *bit-banged* directly into *input_buffer*. Once the full payload has been clocked out the contents are shifted into the *IOBuffer*. The *input_buffer* is then free for clocking in any new payloads received, while the data stored in *IOBuffer* gets processed. Based on tha results of the processing, it might be necessary to transmit again, thus transferring data into the FIFO buffer again, or to transmit data back to the software application, eg information that a data transmission was successfull (if source node), or the actual data received from another node (if destination node).

10.4.9 SendUSB()

As was stated briefly in Section 8.2.2 it will be necessary to send data via the USB port to the host PC from throughout the firmware code of the SWARM network node, from within procedure calls and even interrupts. To solve the problem of data being sent from anywhere in the firmware two global buffers were created, namely *USB_output_buffer* and *USB_TX_Buffer* which can both store up to 27 bytes of data.

The global buffer, *USB_output_buffer*, uses two functions called *Single_USB_Msg(unsigned char)* and *Two_USB_Msgs(unsigned char,unsigned char)* to gather the bits of data to be sent together. The difference between the two functions is simply that *Two_USB_Msgs(unsigned char,unsigned char)* makes sure that both bytes are sent together, instead of one byte being sent now and one byte later. The contents of this temporary buffer are transferred to the USB transmission buffer, called *USB_TX_Buffer*, when the temporary buffer is full or after 1ms has passed since the last transmission and the buffer is still not full. When the contents have been transferred a flag is set to indicate that USB data is waiting to be sent.

If this flag is set in the main loop a function called *SendUSB* will be called which uses the CDC-based USB firmware code functions *mUSBUSARTIsTxTrfReady()* and *putsUSBUSART()* to shift the contents of *USB_TX_Buffer* into the Special Function Registers. The CDCTxService() routine called from within the Timer2 interrupt (see Section 8.2.2) then transfers the data into the CDC USB bulk IN endpoint.

10.5 Firmware Main() Loop

Upon power up the MCU enters the main loop, which it repeats indefinitely.

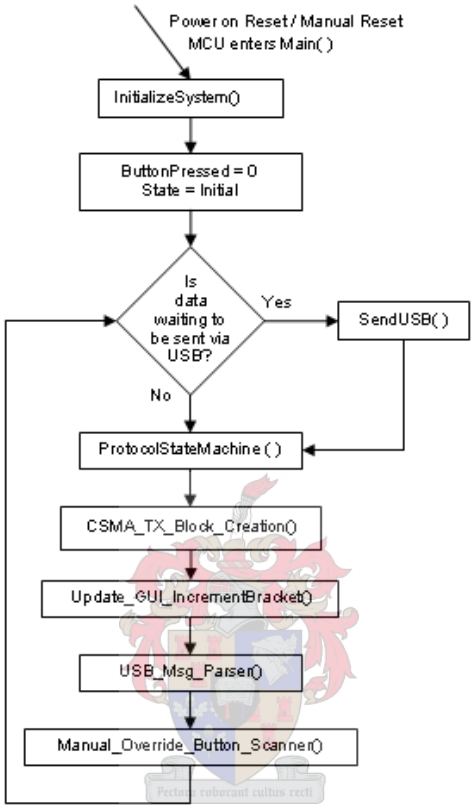


Figure 10.8: Main() of SWARM network node.

Some of these main processes have been described already, whilst others are described in the next couple of sections:

- InitialiseSystem() in Section 10.1.
- SendUSB() in Section 10.4.9.
- ProtocolStateMachine() in Section 10.6.
- CSMA_TX_Block_Creation() in Section 10.7.
- Update_GUI_IncrementBracket() in Scetion 10.8.
- USB_msg_parser() in Section 10.4.1.
- Manual_Override_Button_Scanner() in Section 10.9.

10.6 Protocol State Machine

The protocol state machine consists of 6 states, namely :

- Initial
- Idle
- RX
- TX
- Busy
- Processing

10.6.1 Initial State

After initialization the node is put into the Initial state by default. In this state the nRF2401 is configured as a receiver in Shockburst RX mode (the default mode for a SWARM node). Any data received is immediately clocked out of the nRF2401 by the Timer2 ISR and discarded. This was found to work the best, as the pin which notifies the MCU of a transmission reception only goes low after all data has been clocked out, thus if it is not extracted at all, the pin stays high and no longer notifies the MCU of new packet receptions.

The node leaves this state upon activation by either the host PC controlling software, the manual override button, or automatically if not activated 10 minutes after being powered up.

10.6.2 Idle State

The Idle state is the default state, and there are two ways of leaving the state, by either moving to the TX or RX state. To move into the RX state a valid packet has to be received while in the Idle state, and to move into TX state the nRF2401 has to have settled after a previous reception and one of the following conditions must occur :

- sendMsg flag has been set (a Reply Hello or LANedgereply msg needs to be sent immediately).
- one or more msgs are queued for transmission across nRF2401.
- an update has been scheduled and is ready to be sent.
- timer which gives multiple Route Request Replies time to arrive has expired.

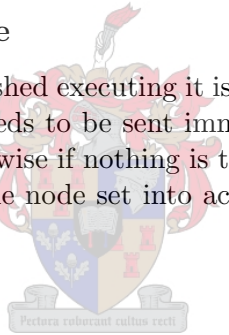
- timer to control Data sent - Ack received period has expired, possible retransmit.
- timer to control Route Request sent - Route Request Reply received period has expired, possible retransmit.

10.6.3 RX State

If the nRF2401 has finished settling after a previous reception, and INT0 has indicated that data is ready to be extracted from the nRF2401 then the flag used by Timer2 to clock out the data gets set. Timer2 will automatically start extracting the data. Once all the data has been successfully extracted by Timer2 it can be moved from the nRF2401 input buffer into the IO buffer used for processing. The message processing procedure MsgProcessor() is called, which parses the data in the IO buffer, and processes the data according to the message type identified. When done processing the message, the state gets changed to processing.

10.6.4 Processing State

After MsgProcessor() has finished executing it is possible that a Reply Hello or LANedgereply message needs to be sent immediately, in which case the state is changed to TX, otherwise if nothing is to be sent immediately, then the state is set to Idle and the node set into active the default Shockburst RX mode.



10.6.5 TX State

The process flow diagram shown in Figure 10.9 shows quite clearly the procesing conducted during the TX state.

The sendmsg flag is used to indicate whether a LANedgereply msg or replyhello msg is ready to be sent immediately. The *SendMsg()* procedure is used to place the nRF2401 into standby, configure the nRF2401 for transmission, and set the flag indicating that data is ready to be loaded into the nRF2401 and transmitted.

All other processes shown in Figure 10.9 have been discussed previously and are thus seen as self explanatory.

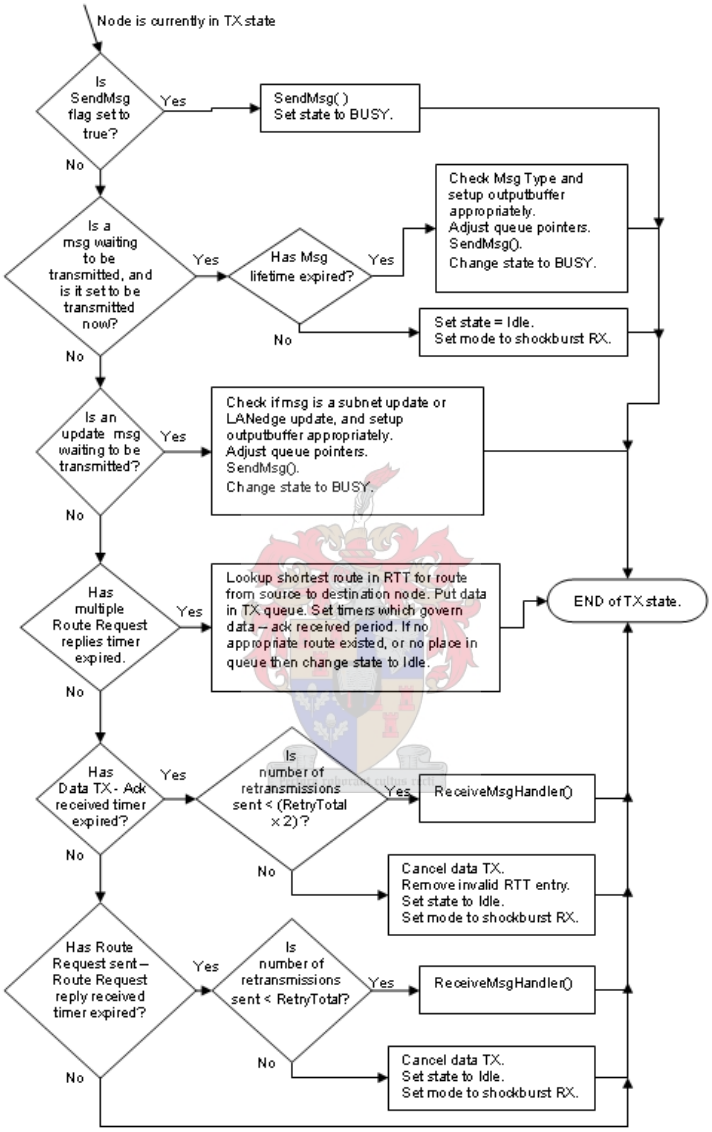


Figure 10.9: TX state of protocol state machine.

10.6.6 BUSY State

This is a 'NOP - no operation' state, in which the MCU simply executes nothing but a short delay. All processing is being done by the interrupts according to flags which have been set. Once these interrupts are done the state will automatically be changed back to Idle when the nRF2401 is set into Shockburst RX.

10.7 CSMA_TX_Block_Creation()

This process uses two variables *CSMACounter* and *CSMABlock* to divide 2.1 seconds up into 21 blocks. Both variables are reset everytime a periodic update gets scheduled to take place, *CSMABlock* to zero, and *CSMACounter* to an initial value. *CSMACounter* is then decremented everytime the TMR2 interrupt occurs.

CSMA_TX_Block_Creation() only executes once every transmission block. This takes place when *CSMACounter* runs out, *CSMABlock* is incremented and *CSMACounter* is reset to it's new initial value. CSMA_TX_Block_Creation() also decrements the TTL tags of all messages waiting in the FIFO queue, and decrements the following transmission timers:

- time to wait for multiple RREQReplies before choosing shortest of all received.
- timer set when data is sent, which determines when a retransmission should occur.
- timer set when a RREQ is sent, which determines when a retransmission should occur.

10.8 Update_GUI_IncrementBracket()

Update_GUI_IncrementBracket() simply checks whether the node has further stabilized, and if it has it will send a message to the GUI telling it to adjust the value of the bar indicating the node stability and update frequency.

10.9 Manual_Override_Button_Scanner()

Manual_Override_Button_Scanner() is used to set the node into user terminal mode and prevents the node from automatically activating itself after 10 minutes. In this case the node assigns itself it's default NodeName ID and mobility rating.

This chapter has explored the firmware created for the PIC18F2550 microcontroller, has shown how all the PCB components are integrated, and how the protocol developed in previous chapters has been incorporated into the firmware in order to create a fully functional SWARM network node. The next chapter looks at the development of the host PC application used to control the SWARM network node when it is set to user terminal mode.

Chapter 11

Software Application

A software application had to be developed for the host PC to allow it to communicate with, control and operate the SWARM network node. A GUI (graphical user interface) was developed which had to provide the user with the following functionality :

- space for the user to enter a node name
- a way to set the node's mobility
- setup and initiate a connection to the SWARM network node
- activate the node
- indicate number of active connections to the node
- indicate the node's link quality
- display received messages
- enter a message and select it's destination node
- set the relay status of a selected destination node

The GUI was developed to run on any Windows OS using Delphi 6, since Delphi allows for the quick layout of a graphical system, and some prior work had already been done using Delphi and a serial port which aided in a fast development cycle. The rest of this chapter examines the various components of the overall GUI shown in Figure 11.1.

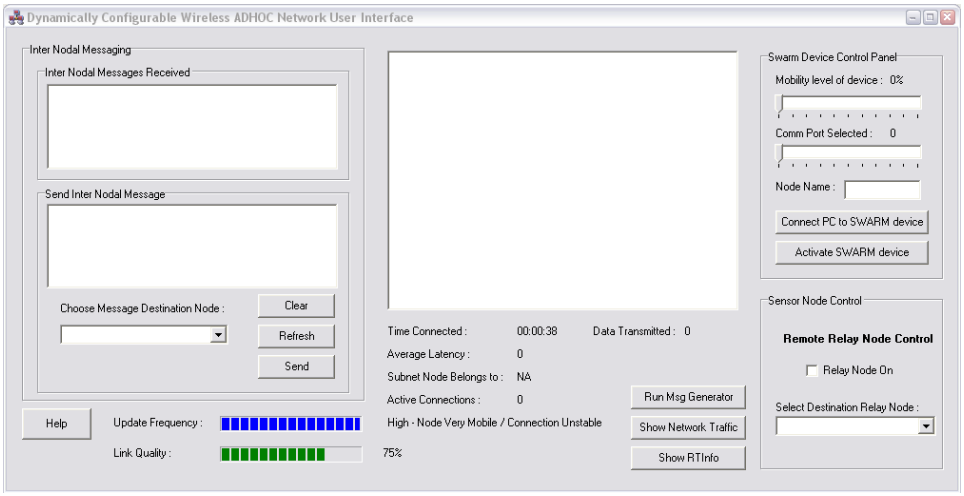


Figure 11.1: SWARM Network GUI

11.1 Serial I/O

The serial port connectivity was setup according to the information provided by Microsoft MSDN [23]. The main Window's function used were *CreateFile()*, *BuildCommDCB*, *SetCommState*, *SetCommTimeouts*, *WriteFile()* and *ReadFile()*.

The serial port functions were written to form a connection with the COM port as specified by the variable *portname*, at a speed of 115200 bits per second, with no parity, 8 data bits and 1 stop bit. When attempting to read from the connected serial port the operation will timeout if nothing is received within 1ms, and when doing a write the operation will timeout if noting can be written after 500ms.

The main procedure used to perform this setup is shown below :

```
function initcomms(portname,portparams: pchar): boolean;
var
  devDCB: DCB;
  devt: COMMTIMEOUTS;
begin
  dev:=CreateFile(portname,GENERIC_READ or GENERIC_WRITE,0,NIL,OPEN_EXISTING,0,0);
  if dev=dword(-1) then begin result:=false; exit; end;
  //  GetCommState(dev,devDCB);
  BuildCommDCB(portparams,devDCB);
  SetCommState(dev,devDCB);
  with devt do begin
```

```
    ReadIntervalTimeout:=0;
    ReadTotalTimeoutMultiplier:=1;
    ReadTotalTimeoutConstant:=0;
    WriteTotalTimeoutMultiplier:=500;
    WriteTotalTimeoutConstant:=0;
  end;
  SetCommTimeouts(dev,devt);
  result:=true;
end;
```

The functions to read and write from the com port are shown below :

```
// writes a char to PIC
procedure comoutc(c: char);
var i: dword;
begin
  writefile(dev,c,1,i,NIL);
end;

// reads a char from port
function cominc: char;
var c: char;
    i: dword;
begin
  readfile(dev,c,1,i,NIL);
  if i=0 then result:=#0
  else result:=c;
end;
```

These form the core functions of serial port connection and communication. Using these as building blocks it is possible to create all the other necessary functions. Eg a function which writes 27 bytes to the serial port at once, or a function which repeats in a loop until it reads an expected character.

11.2 Node Setup and Activation

Before any of the SWARM GUI functionality can be utilized the user needs to select a COM port and create a serial connection with the SWARM network node using the functions described above.

The USB CDC serial port emulation takes place when the unit is connected to a USB port on the host PC. To see which communications port it has emulated itself as, the user can check the windows 'Device Manager', (Found under **System Properties** ⇒ **Hardware** ⇒ **Device Manager**), and look at the communications ports (***Ports(COM & LPT)***) listed before and

after plugging in the unit.

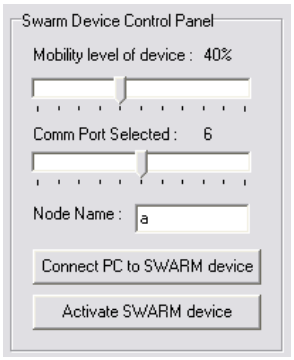


Figure 11.2: GUI - Nodal Control Settings

Once the user has connected the host PC to the SWARM network node, then the user has to set a mobility between 0 and 100%, and enter a node name. The unit is now ready to be activated.

Upon activation the unit will transmit a hello message, starts sending out periodic updates, and scan the air for messages from any other nodes.

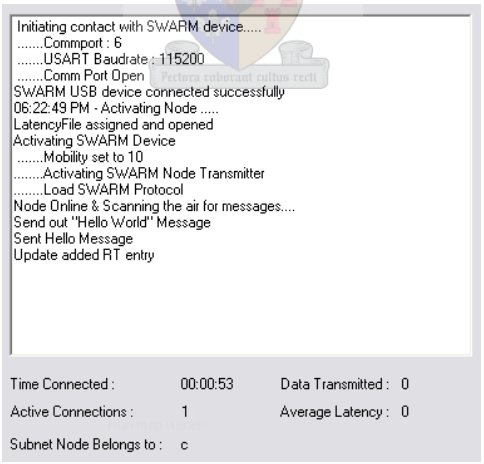


Figure 11.3: Node activation sequence, the node will also hear and join other network nodes

When other nodes are heard, and successfully added to the RT, then the GUI will update the 'Active Connections' field, and the link quality bar will give a measure of the quality of this link .

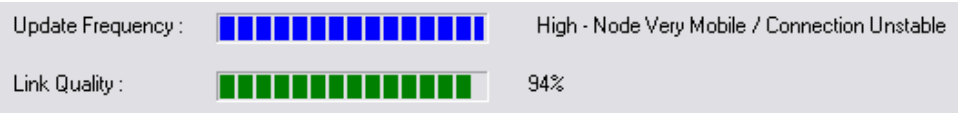


Figure 11.4: Node link quality and update frequency measurement bars

Other connection information is also displayed, such as the current subnet the node is connected to, the amount of time that the unit has been connected, the average latency and amount of data transmitted (both only available if data has been successfully transmitted). The other bar indicates the current update frequency if adaptive updating is possible, and is a direct indication of the system stability.

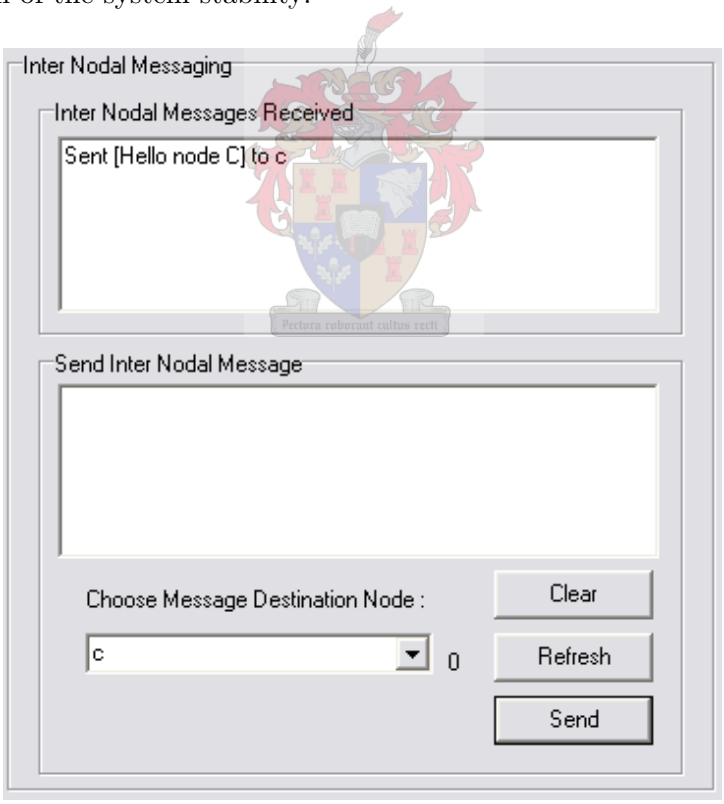


Figure 11.5: Messaging system component

When the node has at least one active connection then the node can message other nodes. To get a list of connected nodes, click refresh and a list of connected nodes will be available in the 'Choose Message Destination Node:' selection box.

When sending a text message to a messaging node simply select the destination node, enter a message in the 'Send Inter Nodal Message' box and click send. The 'Inter Nodal Message Received' box shows all the messages sent and received by the node. Shown in Figure 11.5.

To set the relay status of a relay node, the user also clicks on refresh, selects the relay node to contact, sets the relay level and clicks on send. Shown in Figure 11.6.

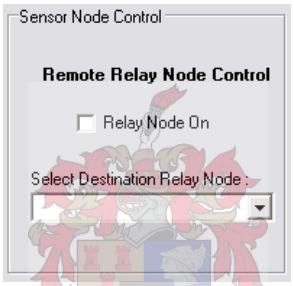


Figure 11.6: Relay node status control

The 'Advanced' section of the GUI is there to provide more information to interested parties and to be able to understand the behaviour of the unit better if it does not behave as expected.

The user has two options in this category, the first being an 'Show RT Info' option which enables the user to monitor the node's RT, thus showing the source, destination, hop count, next hop etc. Shown in Figure 11.7.The individual fields are updated one by one, with one being updated every second.

The second option allows the user to see the node's network traffic statistics, thus showing a list of all the message types and how many of each type has been received already. Shown in Figure 11.8.

In this chapter a GUI has been developed for the host PC, thus allowing the user to control the connected SWARM network node. In the next chapter this GUI is used to thoroughly test the SWARM network node's performance and chracteristical routing properties.

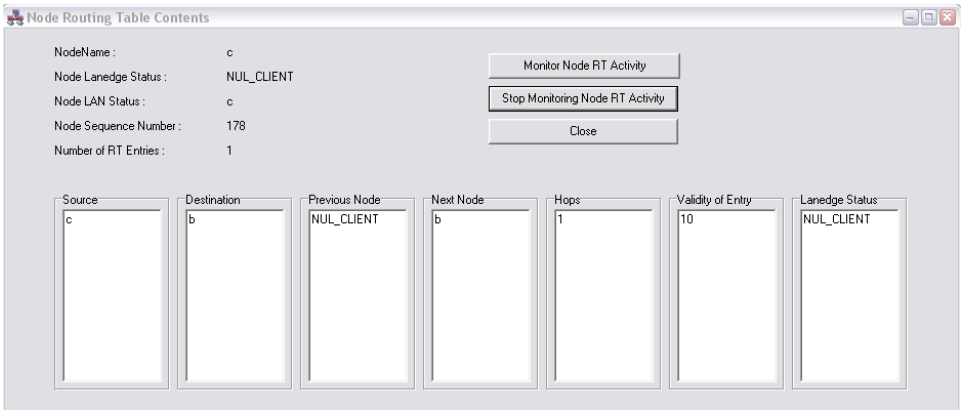


Figure 11.7: Node routing table information

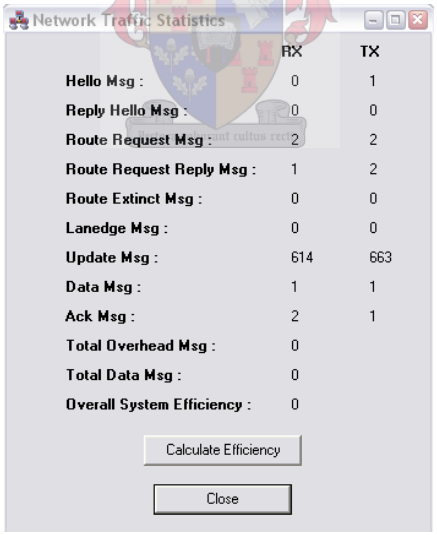


Figure 11.8: Network traffic statistics

Chapter 12

Testing and Evaluation

Five SWARM network node prototype units were constructed. This meant that all the different scenarios could be tested albeit on a very small network scale, which is good enough, since the main aim of the testing procedure was simply to show that the protocol built and tested on the simulators, where there were no packets losses and no real-world interference issues included in the model; is really powerful enough to cope when these factors are present.

The testing was started with a close inspection of all the main features of the protocol, then comparisons were done between a 5 node network run on the simulator, and a 5 node prototype network. Characteristics of the SWARM protocol that were tested :

1. When a node powers up, it scans the surrounding area, shouts out hello, and upon hearing no reply becomes it's own subnet.
2. When two or more nodes within range of one another are powered up, they detect each other and setup a subnet between all of themselves automatically.
3. Two or more nodes maintain the network which they have setup between themselves automatically.
4. When a node powers up and sees a network subnet within range, it joins that subnet automatically, unless that subnet already contains too many nodes, in which case it forms it's own subnet.
5. A subnet of two or more nodes allows any connected subnet node to send data directly to any other node in the subnet.
6. Data messages sent between subnet nodes who don't have direct contact with one another will be relayed by all the nodes that are forming a linked chain between them (results shown below).

7. If nodes are connected together into a subnet and come into contact with nodes from another subnet, then the subnets become connected together in such a way that nodes from one subnet can address nodes in the other subnet simply by messaging the node in their subnet which is acting as a gateway to the other subnet.

8. If two subnets which are connected together as a result of two nodes actively maintaining a subnet link as described above, can be recombined into one subnet, then they will merge into one subnet, one node at a time.

9. When a subnet of two or more nodes lose one node, this change is reflected throughout the subnet automatically.

10. If a node wants to message a node to whom a route does not already exist, a route is sought and stored in a Request Routing Table.

11. More than one route can be found to a node, these routes will be temporarily stored and considered, but the official route to the destination will be the shortest route of all routes found to exist.

12. A node will maintain known routes by updating their validities everytime they are used.

13. If a route has not been used for a long time then its validity will expire and the route will be removed from the RTT completely.

14. If the RTT is full and a new route has been found which needs to be added, then the oldest route in the RTT will be replaced by this new route.

15. If a node wants to send data to another node not in it's own subnet then the route request is sent out through all subnet gateways to other subnets, until the destination node subnet is found, all subsequent data packets will also travel through the subnet gateways along the route to the destination subnet then.

16. Nodes with a low mobility value have the ability to adapt the frequency with which periodic updates are sent out (results shown below).

12.1 Adaptive Updating - The Mobility Variable

The mobility variable sets the maximum timeout value for the Data TX - DataAck RX and RREQ TX - RREQReply RX timers, but more importantly it enables adaptive updating and determines the maximum update period. Adaptive updating makes a huge difference to the amount of overhead messages generated, which in turn makes a big difference to the achievable nodal efficiency.

To prove this a comparison between all the different mobility ratings, showing the effect on the total overhead over a time period of 1hr 37min was compiled. The results are shown in Figure 12.1. For clarity sake this experiment was run without any data transmissions taking place, thus highlighting the effect of the mobility variable on the amount of periodic update messages generated. For a full comparison including data transmissions see section 12.2.

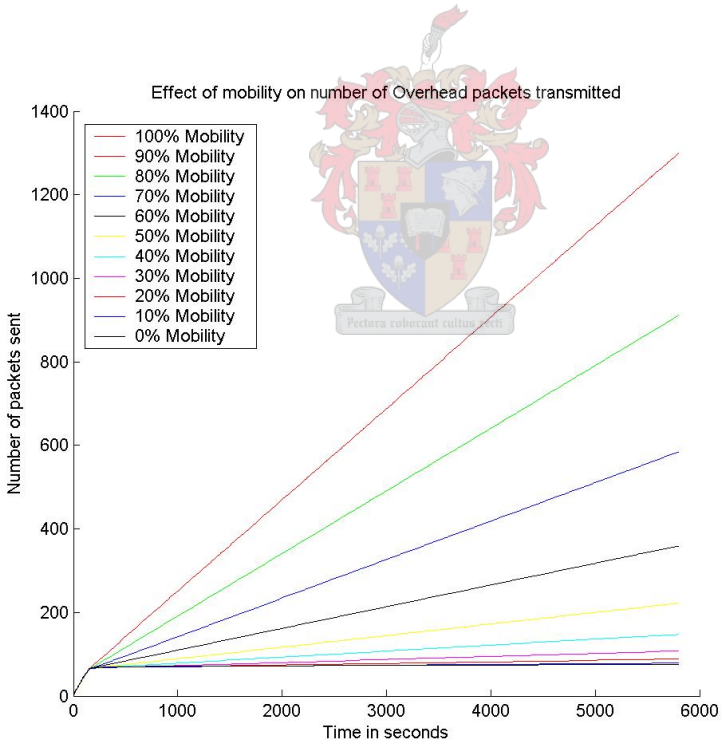


Figure 12.1: The effect of the mobility variable on total node overhead produced during setting up and maintaining nodal network connections over a 1hr 37min time period

12.2 Nodal Efficiency

The SWARM protocol with it's specific *1 data packet - 1 acknowledgement packet* scheme does not have a very high maximum achievable efficiency rating. Even if there was zero extra overhead then a data transmission could only have a 50% efficiency rating. This is a high price to pay, but combined with a CRC-16 implementation (Section 8.2.4) it does provide a very fast, accurate, and reactive way of detecting transmission errors incurred in the system. In this section a theoretical maximum efficiency is calculated and plotted, whereafter several experiments were conducted to show how real world results compare with the theoretical results.

12.2.1 Maximum Theoretical Nodal Efficiency

To calculate this theoretical limit it was assumed that two nodes were activated, immediately formed a network, and updated each other periodically as if their mobility was set to 0%. They then did a route request, got a reply, and subsequently sent all the data packets as necessary to transfer a file of up 64KBytes in size.

Figure 12.2 shows the maximum theoretical limit if no retransmissions are included (the ideal case) and Figure 12.3 shows the maximum theoretical limit if a 10% retransmission rate was used as part of the calculation.

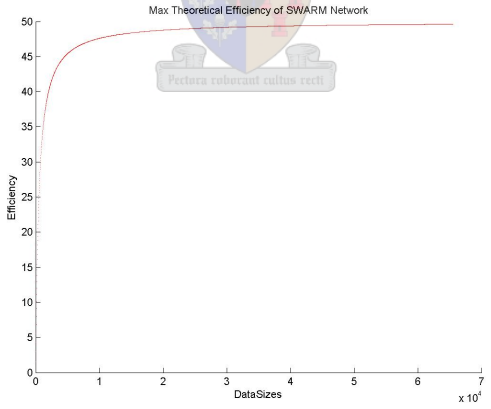


Figure 12.2: Maximum theoretical nodal efficiency, no retransmissions

Looking at Figure 12.2 we see that without retransmissions introducing extra overhead into the system the efficiency does strive towards the 50% efficiency rating as expected from the type of Data-Ack scheme used.

Figure 12.3 clearly shows the effects of a 10% retransmission rate, showing a slightly lower efficiency than the ideal case, with the efficiency curve only reaching a value of 47.2% at it's peak value .

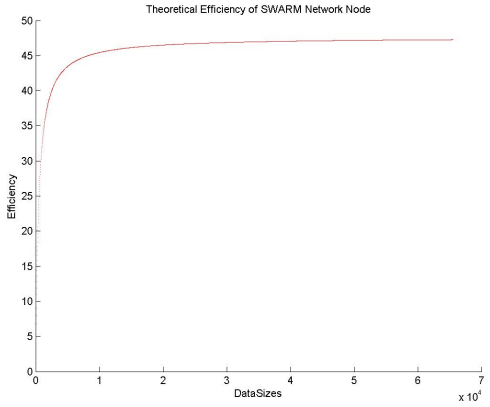


Figure 12.3: Maximum theoretical nodal efficiency, 10% retransmission rate

12.3 Retransmissions and Latency

During each of the experiments conducted the retransmissions and the latency per successful transmission gets logged, thus it is possible to plot a graph of the experiment showing how many retransmissions took place and what the resulting latency was at a specific point in time. These graphs are presented for some of the experiments, but if not presented then the number of retransmissions, the minimum latency, and the average latency are provided merely as overall totals for that experiment.

Data is defined as being data a node sends (node is the source), and the data a node receives where it is the destination of the transfer.

Overhead is defined as being everything else, e.g. the hello messages, the reply hello message, RREQ, RREQReply, DataAcks and even Data if the node is neither the source, nor the destination, but is only relaying the data between another source and destination.

Efficiency is defined here as the amount of data divided by the total data and overhead.

12.4 Results from Experiments

12.4.1 Experiment 1

For this experiment two nodes were placed in the same subnet, one node messaging the other node roughly once every 2 to 4 seconds, mobilities first set to 0% and then to 100%, with each run lasting 40 minutes. Node B was setup as the transmitter, sending a message to node A at least once every 4 seconds.

Looking at Figure 12.4 it can be seen that the amount of data received at node A is higher than the amount of data sent by Node B. This is due to the fact that from the transmitter's side the retransmissions are seen as overhead, but the receiving node still sees data received from a retransmission as data received from node B, and not as overhead.

The efficiency is shown in Figure 12.4 can be seen striving towards an upper limit of around 44 - 45%, which is actually very good considering a 12% retransmission rate, and that the theoretical maximum obtained with a 10% retransmission rate was 47%.

The results for a 0% mobiliy rating are summarized in Table 12.4.1, while a summary of the results for the same experiment, but with the mobility variable set to 100% is shown in Table 12.4.2. The increased mobility

Node Name	**A**	**B**
Data packet total	1014	947
Overhead packet total	1168	1211
Total retransmissions	NA	117
Average packet latency	NA	225ms
Minimum packet latency	NA	109ms
Efficiency rating	46.5%	43.9%

Table 12.4.1: Summary of results for 0% mobility

has drastically increased the overhead total, as is expected, due to the ever present and more frequent periodic updates being sent out. These updates are sent out at the same rate, if not faster than the rate at which data is sent. The effect off this on the efficiency can be quite clearly seen in Figure 12.5, with a new upper limit of around 29% being formed. All remaining experiments will be run using a 0% mobility rating, since the effect of the mobility variable has been illustrated already and it will merely make the results obtained less open to interpretation if included.

Node Name	A	B
Data packet total	890	830
Overhead packet total	2272	2300
Total retransmissions	NA	108
Average packet latency	NA	236ms
Minimum packet latency	NA	109ms
Efficiency rating	28.1%	26.5%

Table 12.4.2: Summary of results for 100% mobility

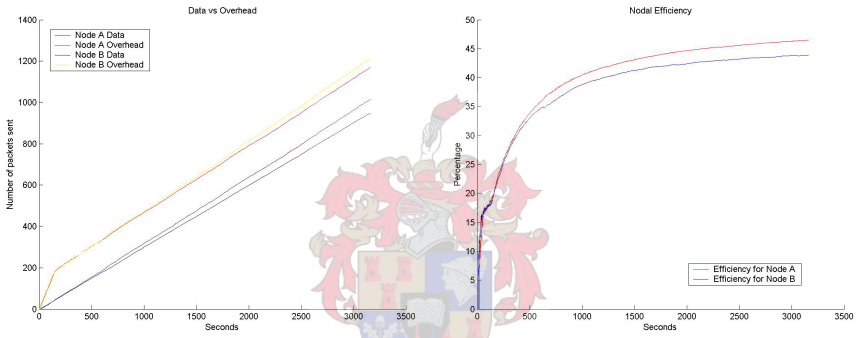


Figure 12.4: Results for 0% mobility

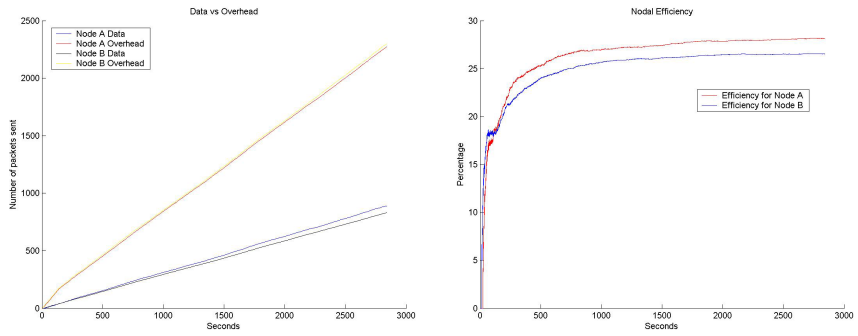


Figure 12.5: Results for 100% mobility

12.4.2 Experiment 2

This experiment was setup similarly to the previous experiment, only with both node A and node B sending a message to each other once every 2 to 4 seconds, and with their mobilities fixed at 0%. The experiment was then allowed to run for 40 minutes before results were gathered, these results are summarized in Table 12.4.3.

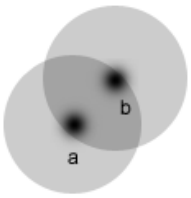


Figure 12.6: Layout of experimental network

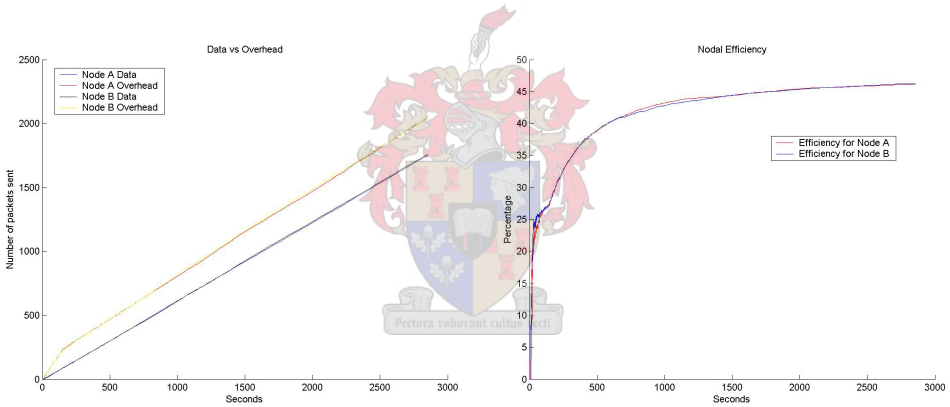


Figure 12.7: Efficiency, and Data vs Overhead ratio plots

Node Name	A	B
Data packet total	1753	1758
Overhead packet total	2041	2046
Total retransmissions	141	137
Average packet latency	234ms	331ms
Minimum packet latency	109ms	109ms
Efficiency rating	46.1%	46.3%

Table 12.4.3: Experiment Results

12.4.3 Experiment 3

In this experiment three nodes were connected directly to one another, in other words, all belong to the same LAN and only have a single hop between them. They were then all setup to message one another once every second, thus each node will transmit a message to one of the other nodes at least once every second, with the entire simulation lasting 40 minutes. The results were then gathered and summarized, as shown in Table 12.4.4.

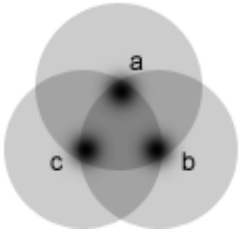


Figure 12.8: Layout of experimental network

Node Name	A	B	C
Data packet total	3008	3114	3121
Overhead packet total	3269	3357	3365
Total retransmissions	272	523	358
Average packet latency	349ms	371ms	288ms
Minimum packet latency	109ms	109ms	109ms
Efficiency rating	47.9%	48.1%	48.1%

Table 12.4.4: Experiment Results

12.4.4 Experiment 4

In this experiment three nodes were connected in such a way that all the nodes could only talk to one another through other nodes, except for directly linked neighbours, as shown in Figure 12.9.

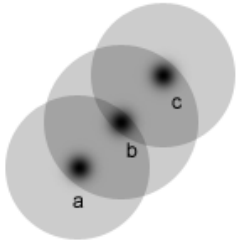


Figure 12.9: Layout of experimental network

The nodes were then all setup to message one another at least once every 2 to 4 seconds. In this way each node will transmit a message to one of the other nodes at least once every 4 seconds, and then the entire simulation was left to run for 40 minutes. After 40 minutes the results shown in Table 12.4.5 were obtained.

Node Name	A	B	C
Data packet total	1440	1507	2040
Overhead packet total	1952	7016	2864
Total retransmissions	94	212	304
Average packet latency	253ms	868ms	717ms
Minimum packet latency	109ms	109ms	109ms
Efficiency rating	42.5%	17.7%	41.6%

Table 12.4.5: Experiment Results

Most notable of all the results shown is the low efficiency rating for node B. Due to its position in the chain of nodes, node B has to relay information for both nodes A and C. These relayed packets are seen as overhead by node B, which explains why node B has nearly 4 times more overhead, and a much lower efficiency rating in comparison to the other nodes.

In this experiment it is interesting to note that node C has the highest retransmission rate (shown in Figure 12.12), this is due to the fact that it had been the node with the lowest link quality rating as a result of it's

peculiar placement (walls and metal plates were used to create the test environment without having to place nodes a hundred meters apart .

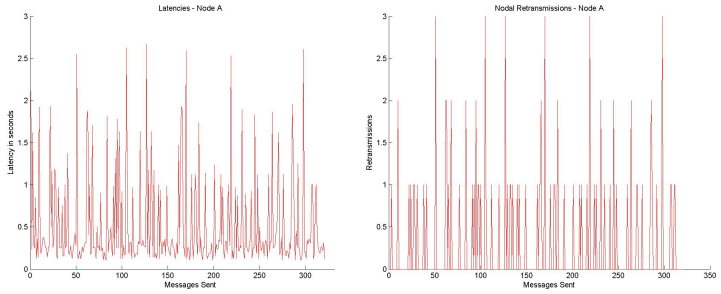


Figure 12.10: Node A - Latencies and Retransmissions

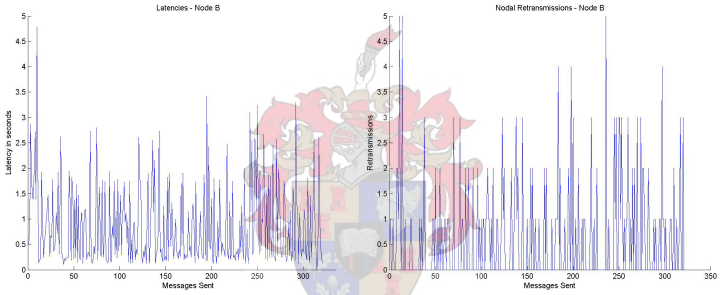


Figure 12.11: Node B - Latencies and Retransmissions

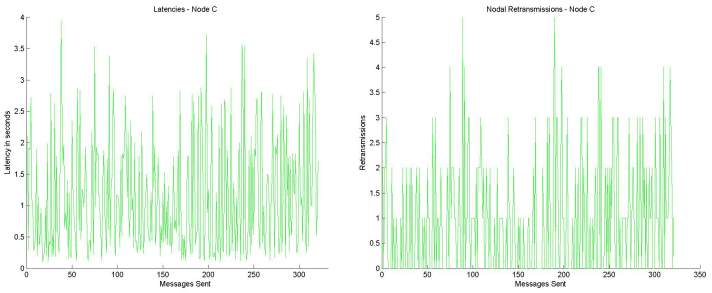


Figure 12.12: Node C - Latencies and Retransmissions

The experiment in general has a high retransmission rate, as should be expected, since each data transmission requires the successfull transmission of four packets in order to succeed. If any one of the four packets gets lost a complete retransmission occurs.

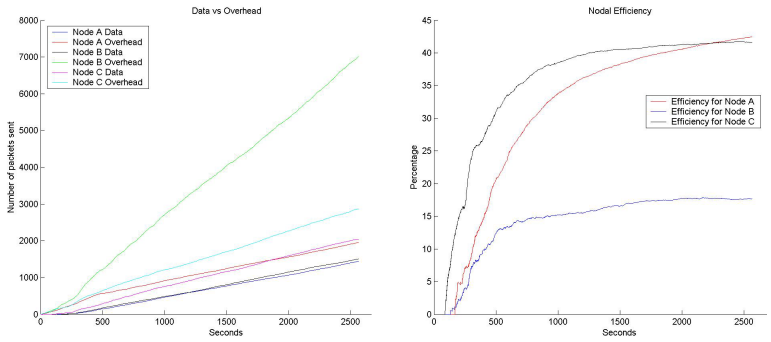


Figure 12.13: Data vs Overhead Ratios and Resulting Efficiency

12.5 Conclusions Drawn from Experiments

This chapter has presented some of the experiments that were conducted whilst testing the hardware device to ensure that it meets the protocol characteristics set forth in Chapter 1.3, Chapter 3, and upon which the behaviour of such a device is based. Most of the results presented are for experiments which were also conducted during simulation to allow a direct comparison.

Comparing the results presented in experiment 2 and 3 of this section to their corresponding simulator experiment results we see that the efficiency ratings obtained using the simulator is slightly higher in both cases, as is expected, since the difference in maximum theoretical efficiencies suggested that this will be the case.

This correlation between results is very good, indicating thet the results from the simulator are reliable, and can be used to test network layouts which are to difficult to test using actual hardware. Performing real life tests, even using only 3 nodes not directly connected, becomes a problem due to the logistics involved in setting up a specific testbed spread across such an area. Coordination of test units, especially since data logging has to be synchronised, becomes a big problem. The protocol itself also made testing more difficult due to it's adaptive nature. A setup had to be very

carefully thought out, else the experiment might start with nodes interconnected in a certain way, but then if the circumstances allow it, the protocol might make network layout changes after having stabilized, thus modifiying the experimental network layout and voiding all test results.

Although not as exhaustively demonstrated here as in Chapter 6, the experiments conducted just confirmed what was found in the simulation environment. The mobility variable has a big impact when the data transmission rate is low, but is negligible otherwise and the efficiency for a specific node is directly related to the amount of traffic it has to route for other nodes. In the hardware the nodes could only maintain a RT of 6 entries, thus the case for 6 nodes in a chain formation, as illustrated in Section 6.4.3 could be considered a worst case scenario. If a high bandwidth carrier was available this would still deliver acceptable results, and the area covered by such a configuration is large. Even with the low power hardware used an area of roughly 1800m in diameter could be covered.

In all the experiments done in this chapter the minimum latency obtained was 109ms. This rate is a direct result of the Timer2 ISR (see Section 10.2.2) used to load and unload the transceiver. In summary, one data transmission consists of the following sequence:

source TX - roughly 21ms
destination RX - roughly 21ms
destination TX - roughly 21ms
source RX - roughly 21ms

Thus it takes 84ms just to load and unload the transceiver, add the time taken in processing, the time taken by the transmission queue, and the time between the network node receiving the last message and the time it gets it's notification through to the data logger (USB buffer queue), then 109ms is a very realizable time period.

Chapter 13

Summary and Conclusions

This thesis has described the research, compilation and simulation of a dynamically configurable ad-hoc wireless network protocol, and the development of a low-cost USB WiFi device utilizing the aforementioned protocol.

13.1 Review of Conducted Work

A study was made of the various network protocols proposed for use in ad-hoc WiFi networks. It was attempted to use all the advantages from protocols which contribute towards building a scalable, self configuring, self maintained network with a tolerable overhead level, while avoiding or minimizing the associated disadvantages. Through this research a rough protocol was conceptionalized, which was then simulated and further refined using two simulators. These simulators were custom built using common programming languages and functionality. This degree of customization meant that the simulators provided exactly the functionality needed to fullfill the requirements set, but also meant considerable amounts of time and effort had to be put into ensuring the correct functioning of all aspects of the simulator.

The main contributions of the protocol developed are as follows:

1) The protocol is a hybrid protocol, utilizing the best characteristics of both reactive and proactive protocols in an adapted ZRP scheme using the abstract concept of LANs and LANedges.

2) The protocol is scalable to big networks whilst still using a fixed RT size through the use LANs and LANedge links.

3) The protocol allows established smaller LANs to recombine dynamically if the current layout is non-optimal.

4) The protocol dynamically adjusts it's periodic update broadcast frequency in order to obtain a optimal value resulting in minimal resource wastage for any given network layout.

5) The protocol allows nodes to configure themselves without any user intervention, and relay information for each other, thus no existing infrastructure is required.

A study was then undertaken to investigate whether a low cost hardware network node prototype could be constructed as proof of concept. This prototype would have to be small, must interface easily with the host pc and have a transceiver with a usable range. The Universal Serial Bus was selected as a communication interface with the network node, thus providing a power source and fast data transfers. A suitable microcontroller was chosen to not only control the nodal networking operations and perform all the routing processing, but also because it could perform USB communications directly. A USB-serial port emulation driver was used to avoid creation of custom USB port drivers and descriptors. This meant that the PC software application developed could now interface the USB device as if it had been just another serial device, thus resulting in faster device design time.

A methodological approach of prototype design and testing was followed to develop the hardware, firmware and software application for the network device. Finally the device was tested to verify proper operation and to evaluate the 'real-world' performance of the proposed network protocol.

13.2 Topics for Future Investigation

13.2.1 Quality of Service

The protocol does nothing to address issues regarding quality of service (QOS) at present. One QOS scheme which could very easily be added to the current protocol is adding an extra metric to each of the routes returned which indicates the average amount of traffic at nodes along that route. Many QOS schemes exist, and some were even researched, [33], [34], but none were implemented due to time constraints.

13.2.2 Power Management

The protocol is not in any way power aware, it assumes all nodes will always have equal and enough power, and does not in any way include this in it's routing metrics. This is however one extra aspect which becomes a big factor when nodes are operating off battery power. For instance the protocol

as developed could be made to change the current LANedge link node to another available node depending on the node's current battery level.

13.2.3 Sensor Networks

This is another area which could easily be incorporated into the existing SWARM nodes and protocol. But with some modifications the nodes could be organized to perform periodic sensor readings and report these to an specific base station node.

13.2.4 Hardware

The greatest limitations to the current design are the range and the data rate. These are both related to the transceiver though, and not so much the protocol directly, which means the system could easily be ported into a different hardware solution and a higher throughput would then be realizable.

The range creates a problem regarding the viability of such an ad-hoc mesh network, since the network relies directly on the availability / reachability of other network nodes to be able to function. The datarate not only means transfers will occur quicker and more realtime, but it is also directly related to the functionality of a large network. If a message has 100 hops to perform to get to the destination, then each hop should not take more than a couple of ms at the most, or your system becomes unusable.

The specific hardware implementation allowed a functional, but limited throughput of 5120bps, or effective data throughput of maximum 2160bps. This throughput could be pushed up to 51200 or 102400bps simply by removing the Timer2 ISR, rewriting it's functionality in assembler and using that to load / extract data from the nRF2401 in a sequential fashion using the system clock as clock, rather than using the timed interrupt as clock.

The other possibility for a much higher throughput without much modification to the system would be the creation of a dual radio system in which two transceivers are present, one a dedicated receiver and one a dedicated transmitter. This would eliminate the time taken to configure the transceiver before and after every transmission and would enable the clocking in / out of data from one transceiver while the other is still busy settling after it's previous operation.

Using a transceiver with a much higher data rate is the other alternative, but then a much more powerful MCU would also be necessary and thus a complete redesign.

13.3 Final Conclusion

The network nodes developed conformed to all the requirements for dynamic self-configured ad-hoc network nodes as set out in Chapters 1 and 3. These nodes can be powered up by a host PC, allowing the user to set several parameters, or automatically in which case the factory default settings are used. In both cases the nodes scan for other nodes within range, join a network if one is found, otherwise create one, setup an addressable network structure automatically, and maintain this network structure dynamically. They can then communicate directly with one another, or indirectly by relaying message for one another.

The performance results obtained from both the simulations and the network nodes constructed are very promising, and indicate that the protocol itself does have some very strong functional characteristics. The hardware itself is not robust enough yet to allow for problem free operation by any user, as it still require some patience and knowledge of the system to operate properly, but this is clearly not an unsurmountable problem.

For the goals of this project the hardware developed fullfilled its role perfectly, although with limited throughput. However, it still successfully demonstrated the feasibility of the protocol and of designing an ad-hoc network device with a general purpose CPU, low cost components and non-proprietory software tools and libraries.

There are minor firmware adjustments and refinements that would have to be made, before such a system could realistically serve a commercial purpose in a network with more than 256 nodes, but with a unit cost of below R100 the SWARM network node constructed is a very functional proof of concept.

Appendix A

Schematic Design

The next page presents the complete schematic design for the final prototype USB ad-hoc network node.

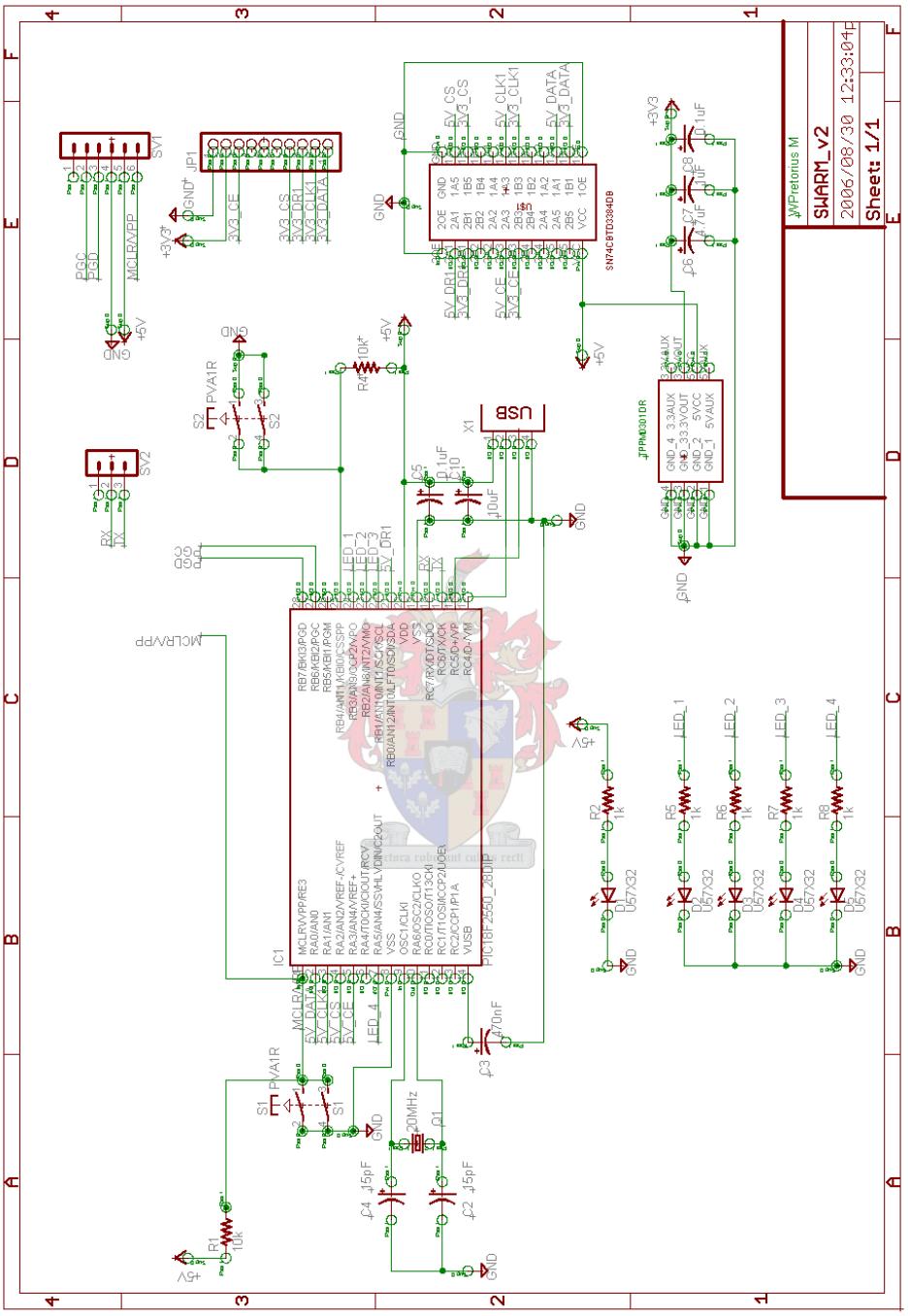


Figure A.1: PCB Schematic

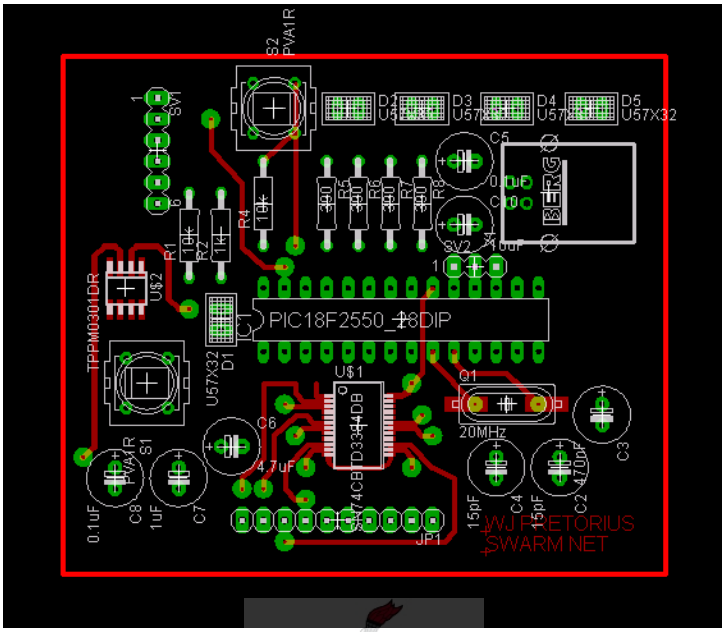


Figure A.2: PCB Top Layer - No GND

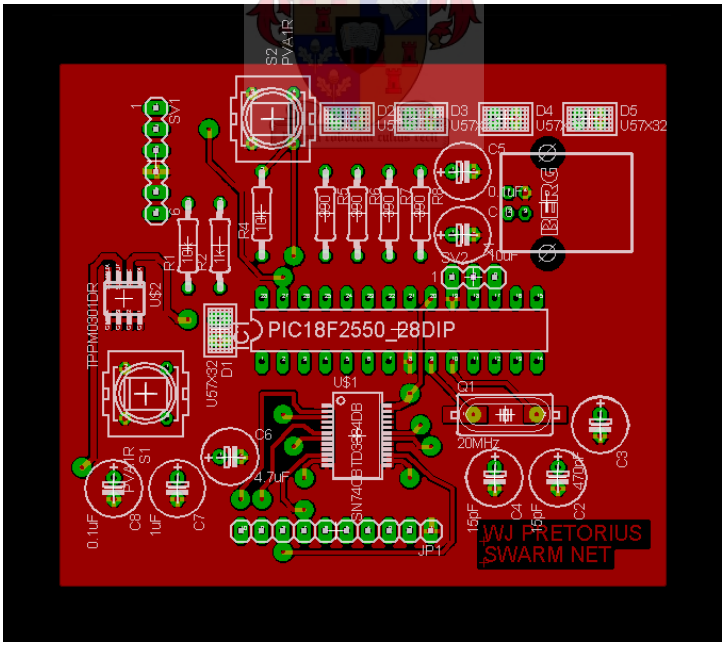


Figure A.3: PCB Top Layer - GND Included

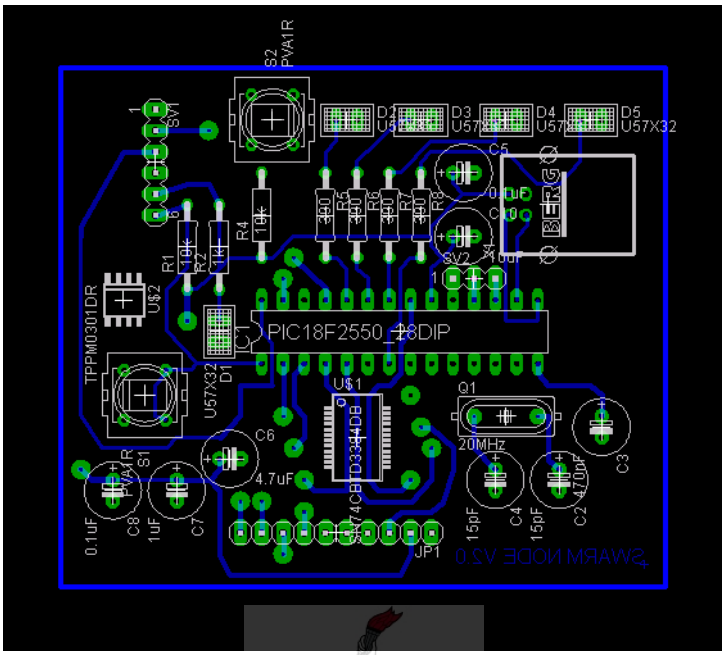


Figure A.4: PCB Bottom Layer - No GND

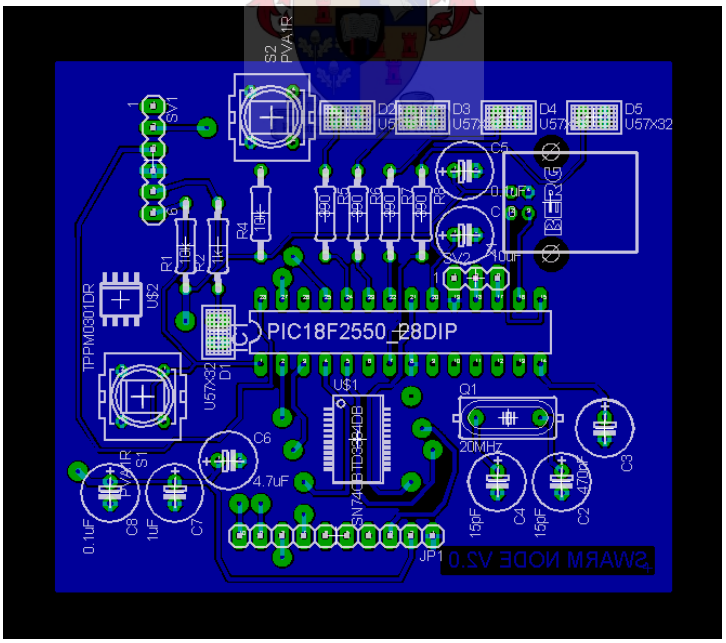


Figure A.5: PCB Bottom Layer - GND Included

List of References

[1] Microsoft Mesh Networking Research HomePage [Online]: 2005. [2005, February 18].
Available at: http://www.research.microsoft.com/mesh/

[2] SensorMag - Wireless Mesh Networks Article [Online]: 2003. [2005, February 18].
Available at: http://www.sensorsmag.com/articles/0203/

[3] C. Siva Ram Murthy, B.M.: *Ad Hoc Wireless Networks - Architectures and Protocols*. 1st edn. Prentice Hall, Upper Saddle River, NJ 07458, 2004.

[4] Nordic Semiconductor, White Paper - nRF24xx Link Integrity [Online]: 2003. [2003, April].
Available at: http://www.nvlsi.no/

[5] pic18Fusb.online.fr [Online]: 2006. [2006, March].
Available at: http://pic18fusb.online.fr/wiki/wikka.php?wakka=UsbBoot%load

[6] Microchip Technology Inc. - PIC18F2455/2550/4455/4550 Data Sheet[Online]: 2004. [2004, March].
Available at: http://ww1.microchip.com/downloads/en/devicedoc/39632c.%pdf

[7] Microchip USB CDC Application Note [Online]: 2006. [2006, March 16].
Available at: http://ww1.microchip.com/downloads/en/Appnotes/00956b.p%df

[8] Neamen, D.: *Electronic Circuit Analysis and Design*. 2nd edn. McGraw-Hill Book Co, New York, 2001.

[9] Universal Serial Bus Class Definitions for Communication Devices [Online]: 2006. [2006, August 22].
Available at: http://www.usb.org/developers/devclass_docs/usbcdc11.pd%f

[10] Doyle, J.: *Routing TCP/IP Volume 1 (CCIE Professional Development)*. 1st edn. Cisco Press, 800 East 96th Street, Indianapolis, 1998.

[11] Zygmunt J. Haas, A New Routing Protocol for the Reconfigurable Wireless Networks [Online]: January 2003. [2005, February 24].
Available at: http://www.ee.cornell.edu/~haas/wnl.html

[12] Alberto Leon-Garcia, I.W.: *Communication Networks - Fundamental Concepts and Key Architectures*. 2nd edn. McGraw Hill, 1221 Avenue of the Americas, New York, NY, 2004.

[13] I.D.Chakeres. AODV-UCSB Implementation from University of California Santa Barbare.[Online]: 2003. [2005, March 1].
Available at: `http://moment.cs.ucsb.edu/AODV/aodv.html`

[14] C.E.Perkins, E.M.Belding-Royer, and S.Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 [Online]: July 2003. [2005, March 1].
Available at: `http://ieeexplore.ieee.org/`

[15] C.E.Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) for Mobile Computers," Proceedings of ACM SIGCOMM 1994, pp 234-244: August 1994. [2005, March 1].

[16] IP Routing Fundamentals - RIP [Online]: 2004. [2006, September].
Available at: `http://www.cisco.com/univercd/cc/td/do/cisintwk/itc_doc%/rip.htm`

[17] Kilgard, M.: *The OpenGL Utility Toolkit (GLUT) Programming Interface*, 1996.

[18] Scholtz, P.: *An OpenGL graph drawing software library*. Final Year Project, BEngSci, Electrical Engineering, University of Stellenbosch, Stellenbosch, South Africa, 2004.

[19] Paul J. Deitel, D.H.M.D.: *Java - How to Program*. 4th edn. Prentice Hall, Upper Saddle River, NJ 07458, 2002.

[20] Random Numbers [Online]: 2005. [2005, August].
Available at: `http://physics.tamuk.edu/~susan/html/4390/Random.html`

[21] Compaq Computer Corporation Hewlett-Packard Company Intel Corporation Lucent Technologies Inc Microsoft Corporation NEC Corporation Koninklijke Philips Electronics N.V - Universal Serial Bus Specification, Revision 2.0, April 27, 2000: .

[22] Wikipedia - Device Driver [Online]: 2006. [2006, August 22].
Available at: `http://en.wikipedia.org/wiki/Device_driver/`

[23] Microsoft MSDN HomePage - File Management Functions [Online]: 2005. [2006, July 10].
Available at: `http://msdn.microsoft.com/`

[24] Nordic Semiconductor, Application Note nAN24-04 USB reference design for nRF24XX rev1 [Online]: 2003. [2003, February].
Available at: `http://www.nvlsi.no/`

[25] Nordic Semiconductor - nRF2401 Single Chip 2.4GHz Transceiver Datasheet [Online]: 2004. [2004, June].
Available at: `http://www.nvlsi.no/`

[26] Nordic Semiconductor, White Paper - nRF240x Shockburst Technology [Online]: 2003. [2006, May].
Available at: `http://www.nvlsi.no/`

[27] Microchip Technology Inc. - ICD2 User Guide [Online]: 2003.
Available at: `ww1.microchip.com/downloads/en/devicedoc/51331B.pdf`

[28] pic18Fusb.online.fr [Online]: 2006. [2006, March].
Available at: `http://pic18fusb.online.fr/wiki/wikka.php?wakka=CdcCOMx`

[29] Hongfa Relay, JRC-27F, MINIATURE TELECOM RELAYS[Online]: 2004. [2006, July 18].
Available at: `http://www.cec-mc.ru/comp/other/hongfa/Telecom/JRC-27F.%pdf`

[30] Texas Instruments - SN74CBTD3384 Data Sheet[Online]: 2004. [2004, dunno].
Available at: `http://www-s.ti.com/cs/ds/sn74cbtd3384.pdf`

[31] Texas Instruments - TPPM0301 low-dropout voltage regulator[Online]: 2004. [2006,August].
Available at: `http://www-s.ti.com/sc/ds/tppm0301.pdf`

[32] Philip Semiconductors, 2N2222 NPN Switching Transistors[Online]: 1997. [2006, August 21].
Available at: `http://www.alldatasheet.co.kr/datasheet/2-9.html`

[33] Boaxin Zhang, H.T.M.: Qos routing for wireless ad hoc networks: Problems, algorithms, and protocols. *IEEE Communications Magazine*, vol. 43, no. 10, pp. 110–117, 2005.

[34] Chunhung Richard, J.-S.L.: Qos routing in ad hoc wireless networks. *IEEE Communications Magazine*, vol. 17, no. 8, 1999.