The Design of a Communication Strategy for an Underwater Sensor Network

by

Jan Abraham du Toit

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

at Stellenbosch University
Department of Electronic Engineering

Study leader: Dr. R.Wolhuter

December 2008

Declaration

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

Date: December 2008

Abstract

The Design of a Communication Strategy for an Underwater Sensor Network

J.A. du Toit
Department of Electronic Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Thesis: M.Sc.Eng (Elec)
December 2008

There is currently a disparity in the amount of research done in underwater communication when compared to terrestrial communication. Therefore, it was the goal of this work to try and make an initial step towards bridging that gap. To start with, an introductory analysis was made of the ocean as a communications medium, focusing on any areas where the ocean characteristics could negatively affect communication. Furthermore, an overview was conducted of current communication schemes, to determine where ocean communication would differ from terrestrial communication, with the idea of determining the limiting parameters of such communication, specifically in terms of protocol design for swarms and sensor networks. Using this research, a *n*-ary tree-based routing algorithm was designed and incorporated into an overall protocol in line with current ISO convention. The strategy was simulated using the Erlang platform and it was found that underwater communication can be achieved with favourable results. It was however also found that using Erlang as a communications tool is currently not the best option and has various shortcomings, although with further work it could be more usable. The implemented strategy appears eminently feasible and should provide a basis for further research and practical implementation.

Opsomming

Die Ontwerp van ‘n Kommunikasiestrategie vir ‘n Onderwater Sensor Netwerk

Daar is tans ‘n wanbalans in terme van navorsing gedoen oor onderwater kommunikasie, in vergelyking met ander kommunikasie mediums. Daar is egter baie interessante potensiele aanwendings in onderwaterkommunikasie. Die doel van hierdie werk was dus om ‘n eerste stap te neem om verkennende ondersoek te doen na tersaaklike moontlike oplossings en dan ook ‘n spesifieke strategie te analiseer en simuleer. Om mee te begin is daar oorsigtelik ondersoek gedoen na die marine omgewing, veral in terme van faktore wat onderwaterkommunikasie beinvloed. Verder is daar ‘n analise gedoen van sommige bestaande kommunikasieskemas om te bepaal of huidige implementerings enigsins prakties, of bruikbaar is vir hierdie doel. Deur van hierdie agtergrond gebruik te maak, is ‘n *n*-êre boomstruktuur ontwikkel om ‘n kommunikasieprotokol daar te stel in lyn met die ISO model. Die eerste vier vlakke van hierdie model is ondersoek en in die protokol ingesluit. Die skema is vervolgens in Erlang gesimuleer, om die bruikbaarheid en werkverrigting te bepaal. Die resultate was belowend. Daar is egter ook gevind dat Erlang self die resultate beinvloed en nie noodwendig die beste simulasietaal vir hierdie doel is nie. Verdere toekomstige werk mag hierdie bevindings nuttig vind en sal dit in ag moet neem.

Ten slotte bied die geimplimenteerde strategie ‘n interessante uitbreiding op bestaande metodes en beloof om goed bruikbaar te wees.

Acknowledgements

I wish to thank the following:

- My Creator,
- Dr. Riaan Wolhuter, my study leader, for his guidance and advice,
- My parents, for their love and support,

Contents

List of Figures

List of Tables

Acronyms

ACK	Acknowledge
BER	Bit Error Rate
CDMA	Code Division Multiple access
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
ESRT	Event-to-Sink Reliable Transport
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
FSK	Frequency Shift Keying
Hz	Hertz
ISI	Inter-Symbol Interference
MAC	Medium Access Control
MACA	Multiple Access with Collision Avoidance
QAM	Quadrature Amplitude Modulation
RAM	Random Access Memory
RF	Radio Frequency
RM	Reed-Muller
RS	Reed-Solomon
RTC/CTS	Request-to-Send/Clear-to-Send
RTT	Round Trip Time
SSB	Single Sideband
TDMA	Time Division Multiple Access
UNACK	Un-Acknowledge
WSN	Wireless Sensor Networks

Chapter 1

Goals and Objectives

The use of technology in the marine environment is on the increase, as everywhere else.

Research indicated that there are possible gaps in underwater communication, specifically in acoustic communication technology aimed at distributed sonar swarm technology. Acoustic sonar swarms are an underwater “wireless” communications network consisting of multiple nodes. These nodes can be deployed for various purposes, i.e. sensor networks, and they use acoustic communication to achieve inter-nodal communication.

A sonar swarm roughly resembles an underwater sensor grid. The sensor grid consists of a number of freely floating un-tethered nodes that communicate via some protocol/acoustic method. Individual sensor tasks however depend on the purpose of the swarm, whether it be for environmental, military or commercial use.

Sonar swarms have great potential and a wide variety of possible implementations. This document investigates some aspects of inter-node swarm application in some depth. The development of underwater sonar swarms has great potential benefit in commercial, military and research areas. This is the background to this project.

The goals of the project were:

1. To do an exploratory investigation into some aspects of a suitable communication protocol for underwater sonar swarms

2. To briefly investigate the marine environmental factors, influencing such a communication strategy
3. To provide an overview of the constraints imposed on such communications, by the technical characteristics of such sonar swarms
4. In view of the above, to consider a few design possibilities for channel access and routing strategies
5. Verification of the selected approach by simulation in Erlang, with emphasis on aspects such as Forward Error Correction (FEC) and error handling in general.
6. Critical evaluation of Erlang for use in future work of this nature
7. Evaluation and presentation of results from the above to assist with decisions regarding future research and development.

Various types of communication protocols for terrestrial communication systems already exist, but this document will look into the possibility of designing such a communication system for marine sonar swarms, taking the problematic environment into account, as well as doing basic simulations. Such a design would then provide some groundwork for further research into underwater communication.

The ocean poses similar but also uniquely different challenges to those found in terrestrial systems, therefore at least some analysis of the ocean communications environment was done to determine these challenges, as they will surely affect the communication protocol designed in this and future work. After designing the communication protocol, it was implemented and tested by running various simulations to determine the viability of the protocol as well as to determine possible shortcomings and/or improvements. The Erlang simulation/programming language was used as a development vehicle. The design of a complete system, including hardware and software, is too comprehensive and the work was, therefore, limited to some of the most important protocol issues.

During the development process, it was decided that the communication protocol to be designed and simulated, would be a deterministic, tree based, routing algorithm with Reed-Muller (RM) error correction, as well as employing the necessary link layer controls such as using an Ack-Unack scheme. The reason for this choice will become clear elsewhere in this document.

It has to be noted that the tree method employed was of an *n*-ary nature, whereas most of the researched implementations were of a binary nature and this increased the degree of complexity in Erlang at the time of implementation. This, however, also creates a degree of flexibility in terms of acoustic sonar swarm dimensions, future research and real world implementation. This is due to the fact that the algorithm is a lot more scalable and adaptable as the value of *n* can be dynamically altered. The simulations were broken up into 4 different parts:

1. Testing of the Forward Error Correction implementation,
2. Determining the value of *n* where *n* is the maximum number of children per tree node for the tree configuration as later described,
3. Determining the efficiency of varying values of *n* under noiseless as well as noisy conditions,
4. Simulation of the communication protocol as a complete system, to determine the ability of the algorithm to handle channel load as well as its ability to relay packets.

The above simulations resulted in the use of a Reed Muller (1, 4) FEC code and a value of *n* = 4 being employed in the algorithm as well as a link layer control system based on a simple Ack-Unack system. Using these values to simulate the algorithm, very good results were obtained and the simulation results proved that the particular tree routing algorithm is a feasible solution to the objectives stated during the course of this work. Furthermore, reasonable throughput was obtained. It was, however, difficult to compare the achieved throughput with the calculated throughput, as Erlang caused some difficulties and introduced unknown delays into the throughput. As a result It was later decided to do a benchmark simulation to determine the variance that Erlang might introduce into the simulations. It was found that Erlang does indeed influence the results and introduces a time varying uncertainty into the simulations.

The results, however, still create a basis for further study and simulation, as well as the possibility of assisting with a real world implementation of the communication protocol. It was found that Erlang is not entirely suitable as a simulation vehicle in this case. More research into Erlang will still have to be done for efficient and effective utilisation, as well as to quantify the influences Erlang might have on communication modelling. It is believed that some of the results documented in this thesis, will contribute towards this process.

This thesis emphasises and contributes towards the following:

- Overview of the marine environment affecting communication and protocol design,
- Design and simulation of a comms strategy in terms of the ISO layer model,
- Initial results favour a deterministic data link layer approach over a contention oriented scheme,
- The *n*-ary tree based routing scheme implemented in the network layer seems to be advantageous over the more common binary structure,
- In the transport layer FEC as well as flow control was implemented, to compensate for uncertainties in the Erlang simulation language,
- Extensive simulations were run to determine optimal parameters for the strategy as presented,
- Some anomalous results, but valuable experience nonetheless, were obtained using Erlang, which would be very useful in future further work in this field.

Chapter 2
Problem Statement, Implementations and Technical Background

2.1. Introduction and Overview

In this chapter the technical background to the project is defined, analyzed and a problem statement generated.

2.2. Problem Statement

Sonar swarms represent a reasonably new technology. A fair amount of research has been done on underwater sensor networks [1, 4, 6, 7], but in almost all of the cases the focus has been on sensor networks that have been tethered or mounted in some manner. This means that all the nodes in the sensor network are stationary and held in place by some means, which in turn is mounted onto a structure, such as an oil rig, or the ocean floor. Most of the documented implementations also had fair to high signal strength and were tested over varying distances.

From the above it can be seen that free floating systems have not had as much exposure as tethered systems. Therefore, the goal of this work is aimed at research into free floating systems and specifically, acoustic sonar swarms. The requirements around a sonar swarm are accordingly:

- The nodes have to be free floating, un-tethered and they have to be able to automatically float at some depth. Therefore, some form of buoyancy has to be achieved.
- They have to be self-contained sealed units, including everything necessary for communication, power, storage and the tasks assigned to

the nodes (determined by application environment). Once they are deployed, no changes can be made to any individual nodes as they will most probably be unreachable.

- They have to be able to operate for limited, but reasonable lengths of time.
- All the nodes have to be able to communicate with each other via some communications protocol and routing algorithm, as well as be able to relay information to some final destination i.e. a control node or a control centre.
- They have to be able to communicate over varying distances as specified by their implementation.
- They have to be robust. If the nodes are placed in dangerous/harmful situations they have to be robust and be able to handle possible damage.
- The communication protocol has to take into consideration that the nodes are free floating and their positions may randomly alter during operation.
- The communication protocol also has to be robust and able to handle node loss.
- The communications protocol cannot be too complex due to restricted communications bandwidth.
- The communications protocol has to be able to handle communications errors.
- The communications protocol has to control the flow of data/information between nodes.
- The communications protocol has to be simulated to check validity and predict performance.

It can be seen from the above that restrictions placed on the sonar swarm will reflect on any and all communication protocols used and designed for sonar swarms. Possible implementations and uses for sonar swarms will be discussed in the next section

2.3. Acoustic Sonar Swarm Applications

The following is a brief description of possible uses and applications of sonar swarms. The possible applications have been categorised into the following preliminary groups, but are not limited to them:

- **Distributed Tactical Surveillance.** One of the most obvious and highest paying of applications would be tactically or militaristically orientated. The sensors could be used for harbour monitoring and protection, eliminating blind spots to submarines, triangulating a target and/or improvements in targeting accuracy, long range and short range underwater monitoring, etc.

- **Environmental Monitoring.** The first application springs from a tsunami related disaster three years ago: i.e. the monitoring of underwater/oceanic seismic activity. Furthermore monitoring of pollution, monitoring of marine life, ocean current monitoring, sea life monitoring and many other applications are possible. Although beyond the scope of this work, the influence and effect that acoustic communication might have on ocean creatures has to be kept in mind when implementing such a sensor network. This is a hardware aspect to take into account.

- **Mapping.** It is said that we know more of outer space than we truly know of the oceans – underwater sensor networks could be used to obtain a wealth of marine environmental information, thus providing us a clearer perspective of just what the ocean holds or hides.

- **Commercial.** Commercial applications such as remote controlled operations, or monitoring in the oil industry. Furthermore, they can be used for the discovery of new natural resources such as oil, natural gases and numerous others. Acoustic sensors could also be used, in an autonomous fashion [4], to monitor ship movements and keep track of shipping in shipping lanes.

2.4. Technical Background

In the next chapter, the focus will be placed on the ocean environment and the unique obstacles posed by it, while in the rest of this chapter possible problems due to the hardware properties of sonar swarms as well as any protocol related issues that have to be resolved, will be discussed.

Later in this chapter, the problems will be discussed in detail, but the following is a quick overview of the technical limitations/requirements:

- The first aspect is the availability of power and type of power supply.
- Another factor would be the communications distance and the bandwidth limit of the sensors, imposed by the sensor hardware.
- Memory availability/memory size for each sensor will influence whether each will have a buffer in which to store incoming and outgoing data.

2.4.1. Sensor Related factors

The sensor characteristics themselves will have a definite effect on the communications architecture used. This section is therefore both a technical specification, as well as a method to determine the design problems posed for the required routing implementation. The following part takes a look at these factors, while solutions will only be discussed in Chapters 4 and 5.

2.4.1.1. Power

Probably the first and foremost factor to take into consideration is power requirements. Power enables all tasks, i.e. data collection and the transmission of the data between nodes. Therefore, it can be said that power is the heart of the sensor, while the communication protocol is the brain of the sensor.

Therefore, when choosing a communication protocol, its influence on power consumption will have to be taken into consideration and the power requirements of such a communication protocol is important. Power influences various factors such as signal strength, communications distance, the size of data packets and the complexity of the routing algorithm itself. The factors in turn influence bandwidth as well as the SNR ratio, which is a very important factor in all communications systems as it affects the probability of error (P_e) and as a result, the bit error rate (BER). Sensor power supply design itself, however is beyond the scope of this report.

2.4.1.2. Limited Communications Range and Signal Strength

This is in part an extension of the previous point, but further considerations play a role. The communications range firstly depends on transmission power (which is limited), but secondly, it also depends on the transmitter configuration and type.

The size and type of the transmitter will be tempered by cost, sensor size, as well as power supply. The communication protocol eventually chosen, has to be compatible with the type of transmitter chosen.

Something that has to be taken into consideration, is that at no time may the signal strength or frequency cause any interference with marine life. This again affects communications bandwidth, speed and reach.

2.4.1.3. Memory

Memory storage space would affect two aspects: the first related to communication, the second related to data capturing and long-term storage.

The first one would affect the communication in terms of buffering: when sending and receiving data, can the data be temporarily stored in a data buffer? This becomes very important when there are long propagation delays (this is the case in underwater communication, as will later be seen), because the data has to be stored in the buffers until all packets have been received. Only then can the data be processed or sent to the next destination. In the sending process a buffer/temporary storage space is also very important, as data will be broken up into smaller packets and these smaller packets have to be stored until all the packets have been sent and all acknowledges have been received. This storage space probably need not be very big due to the limted size of the packets / messagess.

The bigger factor would be the routing table as it would have to be a long term storage space that facilitates quick access, as to increase the total throughput of the routing algorithm. Some form of static, robust, low power memory should be suitable. The exact choice is beyond the scope of this document.

2.4.1.4. Processing Speed

Processing speed and processing power will directly influence the routing algorithm and vice versa. An example would be the assembly and disassembly of packets.

If the current node were the transmitting node it would have to break the message to be sent, into constituent packets, perform encoding if necessary, determine the node to which the data needs be sent (from the routing table) and then initialise transmission. If this node were in turn the receiving node, the packets would have to be taken from the buffer, decoded, checked for errors and then reassembled. Furthermore, the action of deciding whether an Ack (if used) is to be sent, also takes up processing power and processing time.

Therefore, the processing power required will be directly affected by the communications protocol. The cost and size of the required processor will therefore have to be balanced against processing power.

Other factors, such as the specific sensor application, might not directly be protocol related, but will affect the choice of CPU. These factors could require a lot more CPU power and will significantly affect both the size of the sensors as well as the amount of battery power required to run the processor/CPU.

2.4.1.5. Size

Size might not directly affect the protocol, but it does have an effect on all the other components involved in the sensor, the capabilities of the components, as well as cost. Size will be influenced by the purpose and location of the sensor network and the question of disturbance and detection. The smaller the node, the harder it will be to detect, and the less disturbance it will cause to marine life.

2.4.1.6. Cost

Cost as in the case of size, indirectly affects the protocol. Cost will be determined and/or affected by the intended application of the sensors and where they will be used.

2.4.1.7. Inaccessibility

Due to the nature of the environment in which the sensors will be implemented, the nodes will in most likelihood not always be directly accessible. This, in turn, increases the complexity of the scenario and creates certain problems, as was seen in the case of power consumption and availability. The following are a few points to keep in mind:

- No physical interaction is possible between onshore control systems and the monitoring instruments. Therefore, any physical adaptive tuning of the instruments is not possible, nor is it possible to easily reconfigure the system after particular events occur, or after the nodes have been deployed [6].
- Any altering shall have to be done in the next deployment of the nodes or in future versions. An alternative or a solution would be to make sure all instruments are adaptable and/or configurable via the wireless acoustic link.
- If failures or miss-configurations occur, it may not be possible to immediately detect them or to effect changes.
- The amount of data that can be recorded during the monitoring mission by every sensor is limited by the capacity of the onboard storage devices, as above mentioned.
- Power is limited and cannot be immediately replenished, as above mentioned.

From the above technical overview of the sensor nodes, it can be seen that various factors will have to be contended with when designing the final protocol. In the next section protocol related factors will considered.

2.4.2. Protocol Related Factors

Some aspects related to protocol characteristics and design will now be briefly discussed:

2.4.2.1. Duplicate and Lost Messages

The communication protocol would require some way of handling duplicate or lost messages. As was discussed in the hardware overview, a buffer would store all the incoming packets and then try to recombine the packets into a message.

This recombination would be dependent on all the packets having been received and none of the packets being subject to errors of some form, whether they are burst errors or caused by always present white noise. If an error was found (a duplicate packet received or a faulty packet received) some form of retransmission will have to take place if the error(s) cannot be corrected.

The problem with this is that due to the noisy ocean environment, packet loss might actually be so common that retransmission is the norm. The question now is how this is done – whether by sending an un-acknowledge or just forcing the sender to re-send until an ACK is received. All of these will affect overhead, so careful planning would have to take place and a well considered Ack or resend scheme implemented.

2.4.2.2. Error Handling

Further to the previous point, would be the question of error handling. It has to be remembered that error detection and error correction are two separate parts of error handling.

Error detection allows the sensor to determine whether an error occurred but does not repair that error, it merely requests a resend or similar action. On the other hand error correction corrects the error. Furthermore, error correction would reduce the need for a resend request (which itself could also be lost or damaged) and then as a consequence, almost definitely reduce the overhead.

The tempering factor here would be complexity. By introducing error handling of some form (detection or correction), we complicate the communications protocol and in turn increase processing time and power required. This complication could be minimal, or severe in the extreme case of turbo codes. It would therefore affect packet size (thus increasing the overhead and errors), transmission time, as well as transmission power required. A balance is required between complexity and system gain, due to the required error handling.

2.4.2.3. Initial Positioning of Nodes

Something that might also play a role is the positioning of the nodes. It has to be taken into account that the transmitter will most probably work in a broadcast mode and thus will affect or be affected by the positioning of the nodes. Two points out of this to take into consideration are of course the hidden and exposed nodes scenarios. Positioning might also affect the initial routing, but distance between nodes and the number of nodes will definitely affect the designed protocol.

A less significant factor would be whether they will be placed in a two or three-dimensional application. Realistically, the scenario would be three-dimensional, as two-dimensional/”flat” implementation would be most unlikely in the ocean environment. It would also place restrictions on the applicability of the acoustic sonar swarms.

2.4.2.4. Communication Routing: Ad-hoc or Deterministic

Simply stated, communication routing can be done in basically one of three ways, i.e. Ad-hoc, deterministic or a combination of the two. Either of the first two could be implemented in this communications protocol problem, but it would be wise not to rule out a combination of the two, thus utilising the best of both worlds.

To better understand the routing question, each method will be analyzed with a brief explanation of possible advantages/disadvantages of each protocol:

Ad-hoc Routing

- Ad-Hoc routing presents a scenario where all nodes are peers and they communicate in a seemingly random order. Each node has access to every other node and each one is in essence both a server and a peer - transmitting and receiving information as well as relaying information. Communication is usually achieved by contesting for the medium, using methods such as CSMA, and then transmitting when the medium is deemed available.

  The benefit of this is that nodes can be added as required without the other nodes in the network having to be notified – the node just broadcasts its presence and each node in the immediate environment will take notice. The problem with this is that communication is random and collisions can occur frequently, especially on slow noisy networks (i.e. the underwater acoustic channel) which slows down communication and increases packet loss. Ad-hoc systems will, therefore, also be affected adversely by the propagation delay and this has to be taken into consideration when choosing such a scheme.

Deterministic Routing

- Deterministic operation is done in an opposite manner of Ad-hoc and usually consists of one server node with communication controlled and/or overseen by this server. Therefore, all the other nodes are clients and communication takes place in an orderly fashion. This communication control can be done in various ways, but an example would be using a token (token ring), so that nodes may only communicate when they are in possession of the token.

The advantage is obvious in that communication is orderly and thus collisions are minimised and packet loss, due to collisions and other factors other than noise, can be virtually eliminated. The disadvantage in this case would be if a node were to be added, it would have to be added to the routing table of the server. Otherwise the server would not have knowledge of this new node and it wouldn't be able to communicate with that node.

The above need not be as rigid as explained and hybrid methods are available / possible where the best of both worlds are used. The swarm communications environment will have a major influence on the choice of routing structure.

2.4.2.5. Hidden and Exposed Nodes

Hidden and exposed nodes have to be taken into consideration when designing a protocol as they can cause collisions, packet loss as well as introduce delays in the communication network. Hidden and exposed nodes are a product of wireless networks and manifest themselves as follows:

The hidden node problem describes the situation where, say, a sensor A, not within the transmission range of another station C [2], detects no carrier (thus no other transmissions on the medium) and initiates a transmission. If C was in the middle of a transmission, the two stations' packets would collide as far as all other stations that can hear both A and C is concerned. This is a product of wireless communication, because any and all nodes would detect a signal on a physical (i.e. copper) medium, but with wireless this is not the case.

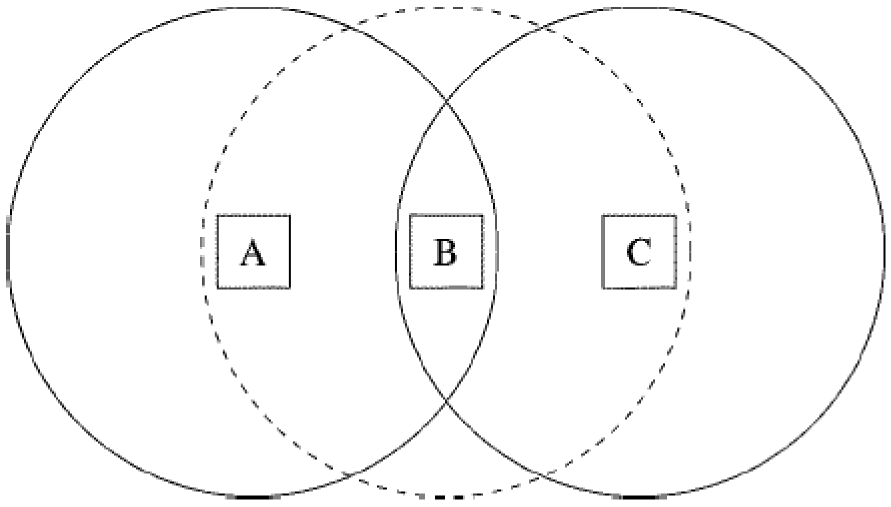

Figure 2.1 Example of the hidden nodes scenario

In the case of the exposed node problem, B defers transmission since it hears the carrier of A. However, the target of B, C, is out of A's range. In this case B's transmission could be successfully received by C. However, this does not happen, since B defers due to A's transmission. The hidden and exposed nodes problems have been shown to significantly reduce the performance of Carrier Sense-based protocols [2].

The above are some of the factors that could and most probably will affect the eventual design of the protocol. Their relevance and what could be done about them, can subsequently be considered and analysed.

The next chapter will look at the operational environment, i.e. the ocean and what affect it will have on protocol selection.

2.5. Conclusion

A problem statement was defined together with the presentation of some general surrounding considerations. This provides insight into the type of obstacles that might have to be overcome when designing a communications protocol. Furthermore, possible applications of sonar swarms were indicated.

In the next chapter the theoretical limitations imposed by the environment will be briefly discussed and should highlight the constraints imparted on protocol design, for this type of application.

Chapter 3

Communications Aspects of the Ocean Environment

3.1. Introduction and Overview

All aspects of the communications system and environment have to be taken into account before solutions can be developed. In the preceding chapter some technical aspects were reviewed and in this chapter the characteristics of the communications medium i.e. the ocean, will be presented. These two chapters are critical as they give insight into constraints that will be placed on any and all communications protocols to be used / developed for underwater acoustic communication.

The ocean environment is harsh and imposes limiting conditions. These complications have to be handled or overcome by the routing and communication protocol if any acoustic communication is to be achieved.

To utilise acoustic and not RF communication is not really a design consideration but a necessity. Radio waves (RF) propagate at long distances through conductive sea water only at extra low frequencies (30 - 300 Hz) [1], which require large antennae and high transmission power. Furthermore, as the frequency is increased to overcome the need for large antennae and high transmission power, the propagation distance (reach) severely decreases.

This again causes a dilemma, as in the previous chapter it was shown that neither high transmission power, nor large antennae, can be accommodated. Another option is optical communication. Optical waves, unlike RF, do not suffer from such high attenuation and do not have the same power / antennae restrictions. On the other hand, optical waves are affected by scattering to varying but prominent degrees in the ocean medium [4].

Therefore, the only viable method for use in underwater communications would be acoustic communication [4]. This choice affects the propagation delay and bandwidth and thus indirectly affects the type of communication protocol implemented in underwater communication.

The problems caused by the ocean include such factors as propagation delay (time it takes for a packet to reach its destination from the time it is sent), effective communication distance between sensors (which will be affected by the nature of the saltwater, the SNR and pro rata the P_e) and effects caused by the nature of the ocean water such as diffraction and reflection. Movement of the sensors due to ocean movement will affect sensor signal detection and inter-sensor communication (hidden or exposed nodes). The position of each node relative to the others is clearly also affected.

The reflective and distorting properties of the ocean floor can cause signals to be lost, duplicated, multiplied and even delayed, as well as create the illusion that one node is in permanent communications range of another when in fact, it is not. This illusion is created by reflected signals that create a temporary communications link between the two nodes, when in actual fact there is no direct communications link. Lastly, the nature of the noise that could be expected, which in turn causes either burst or continuous errors, has to be taken into consideration.

3.2. Theoretical Background and Limitations

The first restriction placed on the protocol by the ocean environment is path loss.

3.2.1. Path Loss

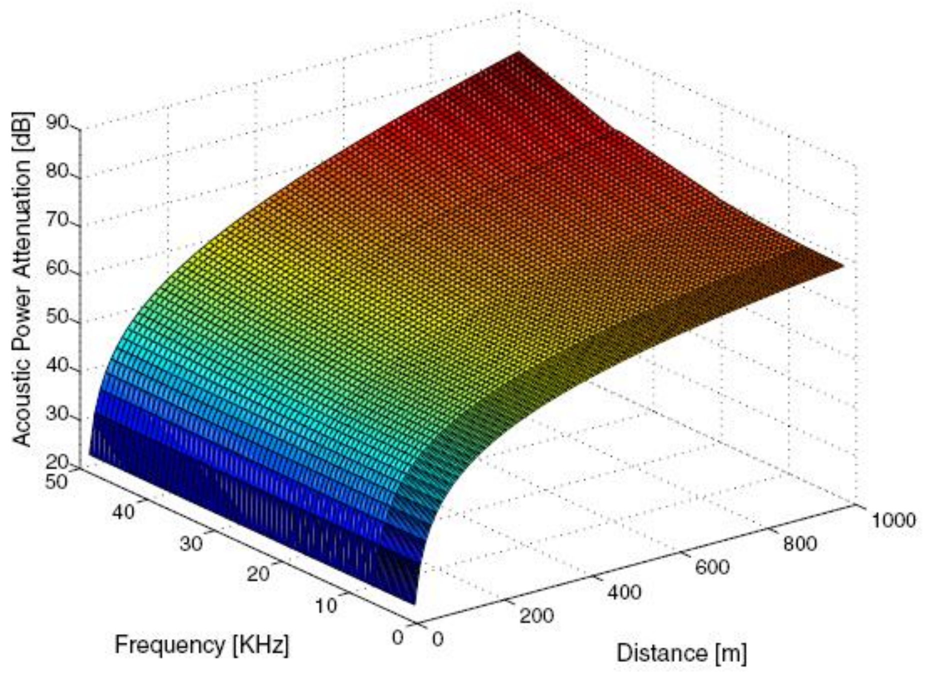

Figure 3.1 Path loss of short range, shallow UW-A channels vs. distance and frequency in the band 1–50 kHz [18]

- *Attenuation:*

Attenuation is mainly caused by absorption – the absorption is due to conversion of acoustic signal energy into heat. Attenuation increases with distance and frequency, as can be seen from Figure 3.1. It may also be caused by scattering and reverberation (due to the rough ocean surface which is affected and/or caused by high winds, rain and tides. It may also be caused due to the irregularity of the ocean bottom), refraction, and dispersion (due to the displacement of the reflection point caused by wind on the surface).

Taking the above into consideration, water depth will play a major role in attenuation. As the depth of the sensor node changes with time, its distance between the ocean bottom and surface will also change, causing the path loss in that direction to alter with time. It can be hypothesized that if the sensors were in deep water, the attenuation might be less as the ocean floor and surface would be a significant distance away in comparison with shallow water operation, thus limiting reverberation, scattering and dispersion. Sensors might not always be used in deep water as the placement of the sonar swarm depends on the application, and the application environment might be either in shallow or deep water. Therefore, this variable has to be taken into consideration.

- *Geometric Spreading*:

  As the wave fronts expand, the sound energy is geometrically spread. The spreading increases with the propagation distance, but is independent of frequency. There are two common kinds of geometric spreading: *spherical* (omni-directional point source), and *cylindrical* (horizontal radiation only). Thus, as the distance between nodes starts to increase, the more losses will be incurred due to geometric spreading.

3.2.2. Noise

The second factor to look at would be noise – noise can be caused by various elements or factors, be it man-made or due to a natural source. Noise is present in all environments and can cause bit errors in distributed or burst format, depending on the noise characteristic in the transmitted signal.

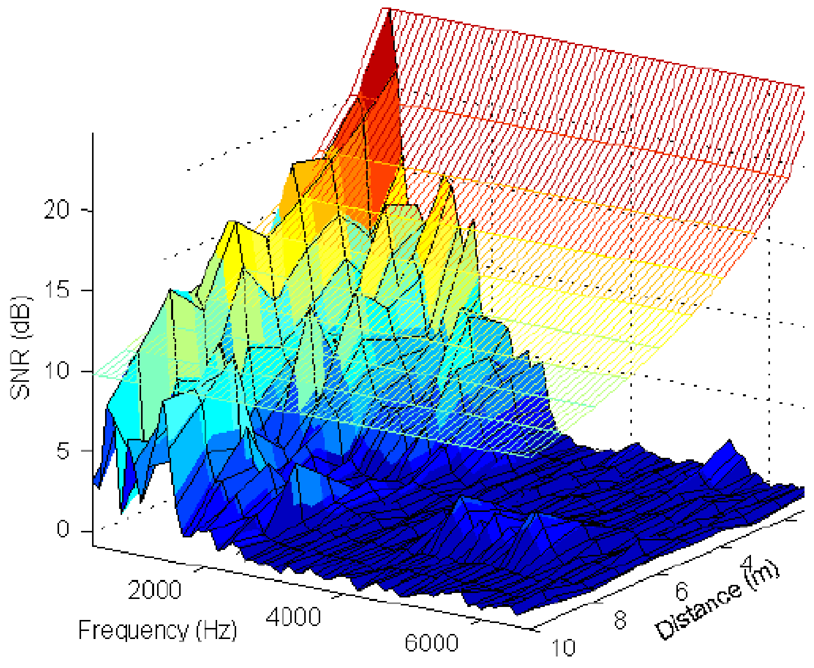

Figure 3.2 Profile of the underwater channel. The solid plot represents the measured SNR and the transparent plot represents the expected SNR [17]

3.2.2.1. Man-Made Noise

Man made noise is mainly caused by machinery noise (pumps, gears, propeller screws, power plants, sonar, etc.), and shipping activity (hull fouling, animal life on hull, cavitations, proximity to shipping lanes). All these factors are external and cannot be controlled. A way to minimise their impact on communication will however have to be found.

The amount of man made noise will depend on positioning and can range from sensor placement in a harbour where noise will most probably be prominent, to deep sea monitoring where man made noise might be minimal and even non existent.

3.2.2.2. Ambient Noise

Ambient noise is related to hydrodynamics (movement of water including tides, currents, storms, wind, rain, etc.), seismic (underwater volcanoes, earthquakes, shifting of the seabed, etc) and biological phenomena (whales, dolphins, etc). Ambient noise is another external factor that cannot be controlled. From Figure 3.2 it can however be seen that at lower frequencies and shorter distances, the SNR (signal-to-noise ratio) profile increases drastically. This in turn will improve the P_e or BER (bit error rate) due to the relation between the SNR and P_e as can be seen from Table 3.1.

Moving on, it is however clear that the ambient as well as the man made noise cannot be accurately calculated beforehand, as the ocean noise will most probably be variable depending on the position in the ocean that the nodes find themselves in. However, it is possible to calculate an estimate using some assumptions:

- Firstly, communication between nodes will in most implementations take place over short distances (< 100 m). This is especially true in scenarios such as harbour protection as well as the elimination of a submarines blind spot (Chapter 2.4),
- Secondly, the communications frequency will in all likelihood be low to very low, typically between 1 kHz and 20 kHz [7],
- Lastly, as will be explained in Chapter 4, the most common modulation scheme used in underwater communication is FSK. This is in a large part due to its simplicity, power efficiency and the fact that it does not require phase tracking [1].

Using the above assumptions as well as Figure 3.2 and Table 3.1 we can calculate an estimated P_e. Choosing a frequency of 2 kHz, employed over a very short distance and using Figure 3.2 we find an approximate SNR of between 10 dB to 14 dB. Using these values as well as the SNR vs. BER curve shown in Figure 3.3, we can see that for a SNR of 10 dB, a P_e of 1 x 10^{-2} can be achieved while for a SNR 15 dB, a P_e of approximately 1 x 10^{-4} can be achieved. Therefore, it can be assumed that the P_e for the ocean as communications medium will lie between approximately 1 x 10^{-2} and 1 x 10^{-4} when using a typical modulation scheme such as FSK. It was found by [4] that for the transmission of sensor data a P_e of between 1 x 10^{-2} and 1 x 10^{-3} would be satisfactory.

However, the sensor nodes will in all likelihood have minimum available power and also the variability of the noise present in the ocean environment is unknown. Therefore, it would be wise to assume a lower SNR of 10 dB, resulting in a P_e of 1 x 10^{-2}, rather than a higher SNR of 14 dB. Therefore, taking the above into consideration, all simulations and assumptions will from here on forward make use of a P_e of 1 x 10^{-2}.

Communication Scheme	Probability of Error P_e (BER)
Non-coherent binary FSK	$P_e = ½ e^{(-E_b / 2N_o)}$
Coherent binary PSK	$P_e = \mathrm{erfc}(\sqrt{(E_b / N_o)})$
Quadriphase-shift keying (QPSK)	$P_e = \mathrm{erfc}(\sqrt{(E_b / N_o)})$

Table 3.1 Comparison of Differing error probabilities for different communication schemes

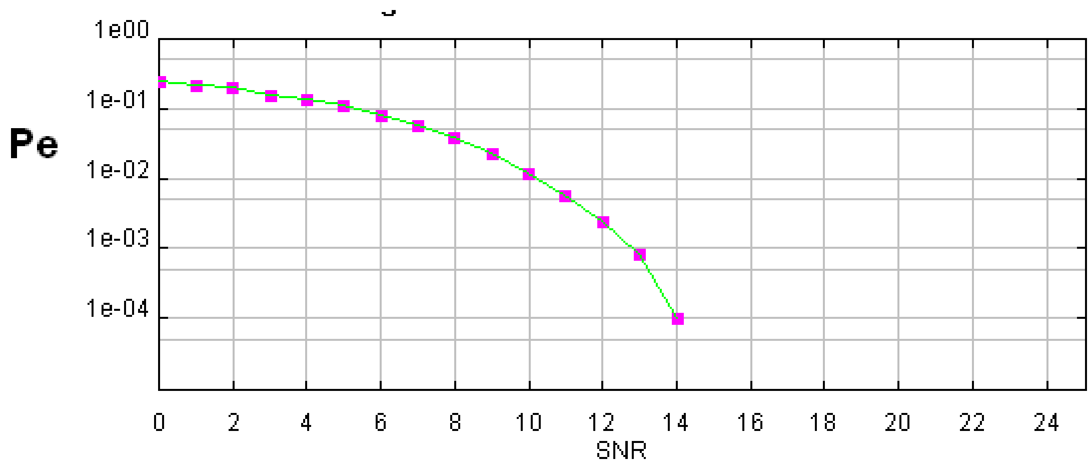

Figure 3.3 SNR vs. P_e for binary FSK. At higher SNR (in dB) ratios the BER decreases drastically

3.2.3. Multi-Path and Fading

The multi-path phenomenon appears when a single signal is sent and multiple time-delayed versions of the same signal arrive at the receiver. What this means is when a signal is sent from point A and travels to point B, that due to the broadcast nature of the signal, the signal becomes reflected off the surrounding environment such as buildings, mountains, ocean bottom in this case, etc. The receiver then receives multiple signals – all of them the same signal, but these versions of the signal are shifted in time and thus shifted in phase.

This then confuses the receiver as to which signal is the correct one and which signals are shifted. Furthermore, these signals interfere with each other and can cause minor and even total cancellation as the signals are shifted through 180° (This is called destructive interference if two signals/waves cancel each other out).

It is logical that this type of interference could also take place in the ocean environment where the water itself, the ocean surface and the ocean floor could all cause multi-paths to be generated.

The multi-path geometry depends on the link configuration. Vertical channels are characterised by little time dispersion, whereas horizontal channels may have extremely long multi-path spreads, of which the value depends on the water depth. Multi-path propagation might be responsible for severe degradation of the acoustic communication signal, since it generates Inter-Symbol Interference (ISI) and resultantly, data packet loss.

Fading on the other hand, has to do with the signal power bleeding away over time/distance in accordance with some probability distribution, rendering detection difficult. This loss of signal power could be greatly increased if the absorption of the medium is very high. This is also a factor of frequency, as higher frequencies will fade much quicker.

ISI is greatly increased when the signalling rate is increased on a given multi-path channel, or equivalently, the symbol duration is shortened. But on the other hand in [10] it was found that increasing the transmission rate allows for faster sampling of the time-varying channel which may in turn, allow for better overall receiver performance as well as more accurate channel tracking. Therefore, a mid-way has to be found where the signalling rate is high enough whilst still keeping ISI and fading down. On a simulation front we now see that the underwater acoustic channel can be modelled as a time-varying (fading) multi path channel [10].

3.2.4. High Delay and Variance

The propagation speed (the speed at which signals travel through the specific medium i.e. signal x travels at y m/s, then y is the propagation speed) in the underwater acoustic channel is five orders of magnitude lower than in the radio channel [1]. This large propagation delay (0.67*s per km*) can reduce the throughput of the system considerably. The very high delay variance is even more harmful for efficient protocol design, as it prevents accurately estimating the round trip time (RTT), and increases the message collision rate, especially in carrier sense protocols.

3.2.5. Doppler Spread

The Doppler frequency spread can be significant in underwater acoustic channels [1], causing a degradation in the performance of digital communications. Transmissions at a high data rate might cause many adjacent symbols to interfere at the receiver, requiring sophisticated signal processing to deal with the generated ISI. Although Doppler shift could also be a factor, the semi-stationary slow moving nature of the sensors will most probably negate this as the sensors will not be moving at a significant velocity, unless propelled by some external force i.e. an underwater ocean current.

From Figure 3.4 it can be seen that as the wind speed increases, so does the Doppler spread. This however only affects communication of shallow water sensor nodes at higher frequencies. The reason for this being that motion of the reflection point causes frequency spreading of the surface reflected signal. Therefore, at greater depth and lower frequency, the effect of wind will be minimal to zero, with corresponding low Doppler spread. In the present application under consideration, Doppler spread can therefore be ignored.

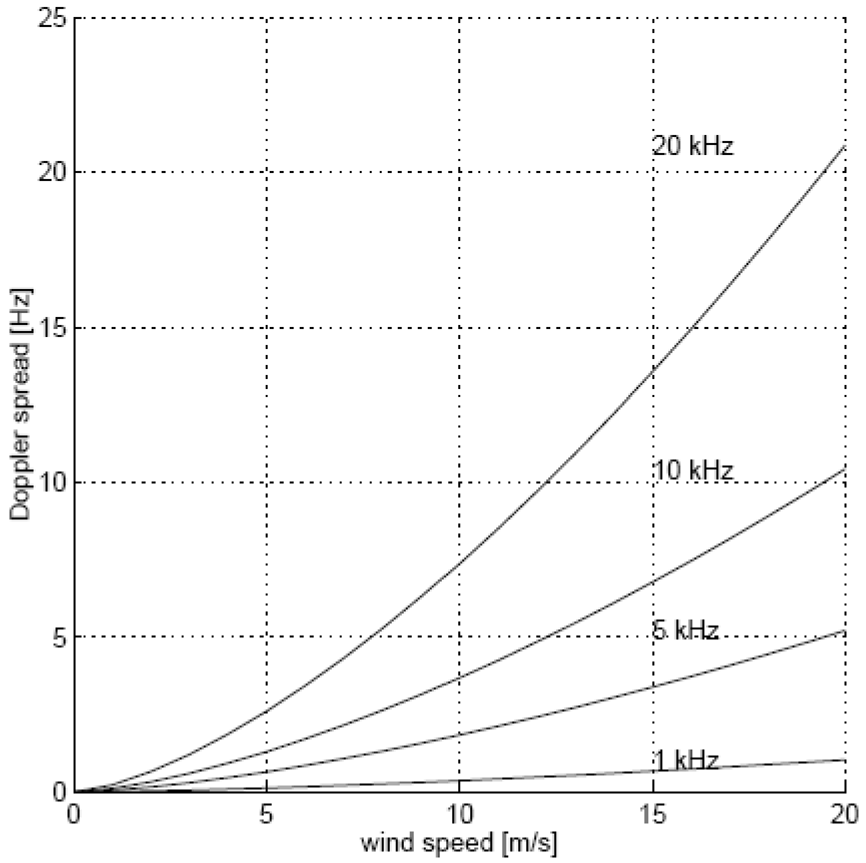

Figure 3.4 Doppler spread on a single surface-reflected path [4]

3.3. Ocean Influence on Communication

From the above, it can be seen that underwater acoustic communications are mainly influenced by the following factors: path loss, noise, multi-path, Doppler spread and high / variable propagation delay. All these factors have an influence on the temporal as well as spatial variability of the acoustic channel [1]. These factors limit the available bandwidth of the underwater acoustic channel and also influence range and frequency selection.

Table 3.2 shows typical bandwidths of the underwater channel for different ranges. From Table 3.2 it is clear that long-range systems that operate over tens, even hundreds of kilometres, may only have a bandwidth of a few kHz, if not in fact a few hundred Hz. In comparison, short-range sensors would fare better but would still be limited to few hundreds of kHz worth of bandwidth – which is several orders less than those obtainable by terrestrial communications systems.

	Range [km]	Bandwidth [kHz]
Very Long	1000	< 1
Long	$10 - 100$	$2 - 5$
Medium	$1 - 10$	≈ 10
Short	$0.1 - 1$	$20 - 50$
Very Short	< 0.1	> 100

Table 3.2 Available bandwidth for different ranges in underwater acoustic channels [18]

In both situations, these factors will ultimately lead to low bit rates. Moreover, communication range is dramatically reduced as compared to terrestrial communication channels due to the marine attenuation characteristics. We can classify underwater acoustic communication links according to their range as: *very long*, *long*, *medium*, *short*, and *very short* links [1]. Acoustic links are also roughly classified as: *vertical* and *horizontal*, according to the direction of the sound wave, but a combination of the two is also possible, especially if the sensors are free floating.

With reference to the brief technical background given, it is clear that some aspects are prominent and important in view of creating a possible communications protocol.

Therefore, it has to take into consideration which problems have to be handled and solved by such a communications protocol and which can be solved by using hardware. Thus the ones that can be solved via a hardware solution can be left out of the scope of the eventual implementation. An example of this would have to be the Doppler spread – it does affect communication but not the communications protocol itself directly, as this problem can be solved using hardware means. Power also falls within these parameters as it influences packet size, communication strength and other factors, but in itself it does not directly affect the protocol.

On the other hand, noise and resultant symbol errors do have a direct effect on a communication protocol, as it interferes with packets sent for routing and communication and will have to be taken into consideration for the final solution.

3.4. Conclusion

It is clear that the marine environment poses limitations that will have to be taken into account in the design of a communications protocol. In the next chapter, current real world implementations will be analysed and an initial investigation will be done into possible ways of implementing a communication system in an underwater environment as well as ways of overcoming the obstacles already analysed.

Chapter 4
Current Real World Implementations and Possible Initial Solutions

4.1. Introduction and Overview

After discussing the theoretical and technical background, it will be possible to create a more specific problem statement, as well as a list of requirements for the creation of a communications protocol for a sonar swarm.

Before moving to the design process it is however prudent to take a look at what types of communication protocols and routing algorithms exist at present, as implemented in the underwater environment. This will help to determine whether it is, in fact, possible to create a suitable algorithm and to estimate its effectiveness. This might also help to compare the final results of this work with other existing approaches and protocols.

In the first part of this chapter however, we shall be looking at the ISO model, specifically the first four layers: physical layer, data link layer, network layer and transport layer. This will be to provide background on the role of each of these layers. The purpose behind this is firstly that most communications protocols are designed using these layers of abstraction. Secondly, this will provide for a means to compare different strategies / communications protocols with each other on a layer by layer basis.

In the second part of this chapter possible solutions to identified problems will be discussed. However, before a design is implemented, existing solutions and implementations will be discussed in the third part of this chapter. This will also help to design a communications protocol that could possibly be implemented as a real world solution. The design phase will therefore be described in Chapter 5.

Problems discussed in Chapters 2 and 3 not directly affecting the protocol will not be dealt with as they are outside the scope of this work.

4.2. Overview of ISO Layer Model

In the next section we shall be looking at the first four layers of the ISO model. This will provide further insight into communications protocols as well as into the design of a possible communications protocol for an underwater sensor network.

4.2.1. Physical Layer

In overview, the physical layer is responsible for converting logical 1’s and 0’s into a suitable signal to be transmitted over the medium. At the receiver side it is responsible for the detection of signals (in the presence of noise), and then converting the signal back into logical form i.e. 1’s and 0’s.

The physical layer defines all the electrical and physical specifications for devices. In particular, it defines the relationship between a device and the physical communications medium. This includes:

- Actual circuitry,
- Voltage levels (defining logical 0’s and 1’s in most cases), and
- Transmission medium i.e. fiber optic, copper, air, etc, or the ocean as in underwater communication.

The following are physical layer devices:

- Hubs,
- Repeaters,
- Host Bus Adapters (HBAs used in Storage Area Networks) [22],
- Modulators and demodulators.

To better understand the concept behind the physical layer in contrast to the way the data link layer functions, think of the physical layer as concerned primarily with the interaction of a single device with the communications medium, where the data link layer is concerned more with the interactions of multiple devices (i.e., at least two) with the shared communications medium [21].
The physical layer will tell one device how to transmit to the medium, and another device how to receive from it. However, with modern protocols, the physical layer does not specify how to gain access to the medium. Obsolete physical layer standards such as RS-232 do use physical wires to control access to the medium. The main responsibilities of the physical layer are:

- Establishment and termination of a connection to a communications medium.
- Modulation of digital data in user equipment and the corresponding signals transmitted over a communications channel.

It should be stressed that for most protocol designs, the physical layer cannot be viewed in total isolation. Error characteristics dependent upon the physical layer, will certainly influence upper layer design and strategies.

4.2.2. Data Link Layer

The Data Link layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical layer. Originally, this layer was intended for point-to-point and point-to-multipoint media [22].

LAN architecture, which included broadcast-capable multi access media, was developed independently of the ISO work, in IEEE 802. IEEE work assumed sub-layering and management functions not required for WAN use. In modern practice, error detection as well as flow control, is present in modern data link protocols such as Point-to-Point Protocol (PPP) [22]. Furthermore, on local area networks the IEEE 802.2 LLC layer is not used for most protocols on Ethernet, while on other local area networks its flow control and ACK mechanisms are used.

Sliding window flow control and acknowledgment is used at the transport layers by protocols such as TCP [22]. Both WAN and LAN services arrange bits, from the physical layer, into logical sequences called frames. Not all physical layer bits necessarily go into frames, as some of these bits are purely intended for physical layer functions.

4.2.3. Network layer

The Network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer. The Network layer performs network routing functions [22], and might also perform fragmentation and reassembly, and report delivery errors. Routers operate at this layer.

This is a logical addressing scheme – values are chosen by the network engineer and the addressing scheme is hierarchical. The best known example of a layer 3 protocol is the Internet Protocol (IP). Perhaps it's easier to visualize this layer as managing the sequence of human carriers taking a letter from the sender to the local post office, trucks that carry sacks of mail to other post offices or airports, airplanes that carry airmail between major cities, trucks that distribute mail sacks in a city, and carriers that take a letter to its destinations. Fragmentation on the other hand is the process of splitting an application or transport record into packets.

4.2.4. Transport Layer

The Transport layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The transport layer controls the reliability of a given link through flow control, segmentation / de-segmentation and error control. Some protocols are state and connection oriented - this means that the transport layer can keep track of the segments and retransmit those that fail.

Although it was not developed under the OSI Reference Model and does not strictly conform to the OSI definition of the Transport Service, the best known example of a layer 4 protocol is the Transmission Control Protocol (TCP) [22]. The transport layer is the layer that converts messages into TCP segments or User Datagram Protocol (UDP), Stream Control Transmission Protocol (SCTP), etc. packets.

Perhaps an easy way to visualize the Transport Layer is to compare it with a Post Office, which deals with the dispatch and classification of mail and parcels sent.

4.3. Current Real World Implementations of Underwater Communication with Relation to the ISO Model

One of the first underwater communication systems was an underwater telephone which was developed in 1945 in the United States for communicating with submarines. This device used a single sideband (SSB), suppressed carrier, amplitude modulation in the frequency range of 8-11 kHz. By using this method it was possible to send acoustic signals over several kilometres [4]. However, in modern times companies have achieved two-way, RS-232 equivalent underwater acoustic communication at 9600 bits/second with an acoustic link of nominally 17.8k bits/second [5]. These communications systems are, however, tethered in a number of cases and also achieve reasonable signal strength from good power supply availability.

The following is an overview of the more common MAC layer techniques and protocols. Possible pros and cons of each are shown, and checked for relevance to the present project. In keeping with the traditional layered communication model, each aspect will be discussed under the headings: Physical, Transport, Network and Data link layer. A brief description of the purpose of each layer is also added as an introduction. Afterwards a brief case study of underwater acoustic communication is also added.

4.3.1. Physical Layer

The main purpose of this layer is modulation and demodulation. Until the beginning of the last decade underwater modem development was based on FSK - non-coherent frequency shift keying [1]. The obvious reason for this was that non-coherent FSK communication techniques do not require phase tracking. Phase tracking can become very difficult in underwater situations due to multi-path and fading phenomena.

Although non-coherent modulation schemes are characterised by high power efficiency, their low bandwidth efficiency makes them unsuitable for high data-rate multi user networks. This bandwidth utilisation is furthermore a problem when taken into consideration that there might be limited bandwidth available in the acoustic communications system, therefore every bit of bandwidth has to be utilised in an efficient manner.

Because of the restrictions caused by non-coherent FSK, coherent modulation techniques became a more actively studied field and protocols were simulated and tested for use in long-range, high-throughput systems [10]. In the last few years [1], fully coherent modulation techniques such as phase shift keying (PSK) and quadrature amplitude modulation (QAM) have become more practical due to the availability of powerful digital processing, faster processors as well as cheaper components. But limitations do still exist and in low cost application environments such as this one, such a solution might be impractical.

In horizontal underwater channels, especially in shallow water, the time-variability of the channel is the primary limitation to the performance of conventional receivers. Multi-path phenomena create two problems. The first one is the delay spread which causes ISI (Inter symbol interference) at the receiver side. The other one is the phase shift of the signal envelope. Thus, high-speed phase coherent communications are difficult because of the combined effect of the time varying multi-path as well as the Doppler spread in shallow water.

4.3.2. Data Link Layer

The data link layer is responsible for medium access control. Multiple access techniques were developed to allow devices to access a common communications medium (in this case the ocean). This is an attempt to utilise the scarce available bandwidth in an efficient and fair way, giving equal access to all the nodes, while maintaining maximum use of the bandwidth.

Channel Access Control in underwater acoustic networks poses an additional challenge due to the peculiarities and unpredictability of the underwater channel, in particular due to limited bandwidth and high variable delay. There are however various multiple access techniques which might be useful. Multiple access techniques can be roughly divided into two main categories [3]:

1. Contentionless (nodes do not contend for access to the medium), such as FDMA, TDMA, polling and CDMA,

2. Contention orientated, which are either based on random access (ALOHA, slotted-ALOHA), or carrier sense access variations (CSMA), or on collision avoidance with handshaking access (MACA, MACAW).

4.3.2.1. Contentionless Multiple Access Techniques

- Frequency division multiple access (FDMA) divides the available frequency band into sub bands and assigns each sub-band to a device. Each device or node may then communicate by using one or more of these assigned sub-bands. Due to the narrow bandwidth in underwater acoustic channels and due to the vulnerability of limited band systems with regard to fading, FDMA is not entirely suitable for underwater acoustic communications [1].

- Time division multiple access (TDMA) divides time into slots, providing time guards to limit packet collisions from adjacent time slots. These time guards are designed to be proportional to the propagation delay of the channel (which in this case is quite high). Due to the characteristics of the underwater environment it is very challenging to realise precise synchronisation with a common timing reference, which is required for the proper utilisation of time slots in TDMA. Moreover, due to the high delay and delay variance of the underwater channel, TDMA efficiency is limited because of the long time guards as well as the large size of each slot required for implementation.

- Code division multiple access (CDMA) allows multiple nodes to transmit simultaneously over the entire frequency band. Signals from different nodes are distinguished by means of pseudo-noise codes that are used for spreading the user signal over the entire available band. This makes the signal resistant to frequency selective fading caused by multi-paths. Although the high delay spread which characterises the horizontal link in underwater channels introduces difficulties in maintaining synchronisation among the stations, especially when orthogonal code techniques are used, CDMA is a promising multiple access technique for underwater acoustic networks. However, synchronisation issues will still have to be resolved which might prove to be difficult due to the high propagation delay.

- Polled systems work on the principle where nodes are in-turn solicited for data by a central controlling node in a continuous, scheduled fashion. There are many variations on the theme, i.e. priority, scheduled or adaptive polling. It is simple to implement, but might not be well suited to a dynamic environment.

4.3.2.2. Contention Oriented Multiple Access techniques

In the underwater environment Carrier Sense Multiple Access (CSMA) protocols exhibit low efficiency, mainly due to the noise characteristics of the acoustic channel and resultant collisions. Furthermore, the need for retransmissions increases the power consumption of sensors and ultimately reduces the lifetime of the sensors as well as the entire network.

CSMA is one of the primary contention orientated protocols. In CSMA a node firstly senses the medium and if the medium is not in use the node transmits, otherwise the node delays transmission for later retransmission or until the medium is deemed clear. There are three different CSMA variations namely:

- Non-persistent.
- 1-persistent, and
- P-persistent.

Furthermore, each of the above can be deemed to be slotted or un-slotted, where slotted means that packets are only transmitted within a ‘time slice’, where the size of this slice is determined beforehand in terms of time. Slotted systems however will be impractical in the ocean environment as the high propagation delay will make synchronisation between nodes difficult and in slotted systems synchronisation is critical. Furthermore, the high propagation delay will also negatively affect CSMA and decrease throughput, as the throughput of CSMA is heavily dependent on the propagation delay.

Contention based techniques that use handshaking mechanisms, such as RTS/CTS (request to send/clear to send) in shared medium access (e.g. IEEE 802.11) are impractical in underwater situations. This is due to the following reasons:

- Large delays in the propagation of RTS/CTS control packets lead to low throughput. This high propagation delay also increases the number of collisions.

- The high propagation delay of underwater channels impairs the carrier sense mechanism. To explain this, imagine two nodes at different points of the grid. Node one senses the medium and finds it free and thus starts to send. Node two then senses the medium but because of the high propagation delay (nearly a second per km) it also finds it ‘empty’ and starts sending (This might also be seen as a variation on the hidden node scenario). This then causes a collision and requires retransmission.

  If due to the high propagation delay of 0.67 s per km, two nodes try to send on the open medium, within approximately one second of each other, they will most likely find it to be available more times than not, unless they are located within close proximity to each other. This will therefore impede carrier sense and be aggravated by long inter-nodal distance.

- The high variability of delay in handshaking packets makes it impractical to predict the start and finish time of the transmissions of other stations [1].

Many novel access schemes have been designed for terrestrial sensor networks, whose objectives are to maximise the network efficiency and prevent collisions in the access channel. These similarities would suggest that adaptation, tweaking or even directly applying those schemes in the underwater environment might work.

However, S-MAC for example, aims at decreasing the energy consumption by using sleep schedules with virtual clustering [1]. Although this contention based access scheme is provided with an effective collision avoidance mechanism, it may not be suitable for an environment where the required dense sensor deployment and continuous favourable medium conditions cannot be assumed.

4.3.3. Network layer

The network layer is in charge of determining how messages are routed within the network. In underwater acoustic sensor networks, routing translates into determining which path data packets should follow from the source sampling the physical phenomenon to the onshore sink, or to the mother node if deterministic routing is used, or some other node in a purely peer-to-peer scenario.

In recent years there has been an intensive study in routing protocols for ad-hoc wireless networks. However, due to the difference between the underwater environment and the terrestrial environment, there are several drawbacks with respect to the suitability of using the existing solutions for Underwater Acoustic

Networks. The existing routing protocols are usually divided into three categories, namely:

- proactive,
- reactive and
- geographical routing protocols [1]:

4.3.3.1. Proactive routing protocols

These protocols attempt to minimise the message latency induced by route discovery by maintaining up-to-date routing information at all times from each node to every other node. This is obtained by broadcasting control packets that contain routing table information (e.g. distance vectors). These protocols invoke a large signalling overhead to establish routes for the first time and each time the network topology is modified, even more overhead is invoked because of mobility or node failures, since updated topology information has to be propagated to all the nodes in the network.

This way each node is able to establish a path to any other node in the network, which may not be needed or be redundant in underwater sensor networks. For this reason proactive protocols are not entirely suitable for underwater networks, as they incur large overhead. Also, having to update the routing tables constantly might have a significant effect on power drain as well as storage space.

4.3.3.2. Reactive routing protocols

A node initiates a route discovery process only when a route to a destination is required. Once a route has been established, it is maintained by a route maintenance procedure. These protocols are more suitable for dynamic environments such as the ocean, but still incur a high latency and still require sending out large numbers of packets, called flooding, to discover the route.

Thus, both proactive and reactive protocols incur excessive signalling overhead due to their extensive reliance on flooding [1]. Reactive protocols are deemed to be unsuitable for underwater sensor networks as they also cause a higher latency that may even be amplified by the slow propagation of acoustic signals. Moreover, the topology of underwater networks are unlikely to vary dynamically on a short time scale even if strong water currents are taken into consideration, therefore it can be assumed that the nodes will stay in reasonably the same place or in the same proximity to each other.

4.3.3.3. Geographical routing protocols

These protocols establish source-destination paths by leveraging local network information/positions of neighbour nodes, i.e. each node selects its next hop based on the position of its neighbours and of the destination node. Although these techniques are very promising, it is still not clear how the required accurate localisation information can be obtained in the underwater environment with limited energy expenditure and transmission unreliability.

It will be advantageous if routing schemes that account for the 3D underwater environment could be developed. Especially in the 3D case the effect of currents should be taken into account, since the intensity and the direction of currents are dependent on the depth and application location of the sensor node. Thus, underwater currents can modify the relative position of sensor devices and also cause connectivity holes, especially when ocean column monitoring is performed in deep waters [1].

While most developed protocols for Ad-hoc networks are based on packet switching, i.e. the routing function is performed for each single packet separately, in underwater sensor networks virtual circuit routing techniques could be considered [1]. In these techniques paths are established a priori between each source and sink, and each packet follows the same path. This may require some form of centralised coordination (deterministic routing) but can lead to more efficient data transfer (at the expense of not being able to accommodate dynamic networks).

4.3.4. Transport Layer

In this section existing reliable data transport solutions for Wireless Sensor Networks, along with their shortcomings in the underwater environment, and the fundamental challenges for the development of an efficient reliable transport layer protocol for underwater sensor networks, are discussed.

In sensor networks reliable event detection at the sink should be based on collective information provided by source nodes and not just on individual reports from each single source. Therefore, new ways should be defined to provide reliable event detection, based on the requirements of the ocean, as current methods could feasibly lead to wastage of scarce resources.

Thus, the underwater network paradigm necessitates a new event transport reliability notion rather than the traditional end-to-end approaches. A transport layer protocol is needed in underwater networks (as in other networks) not only to achieve reliable collective transport of event features, but also to perform flow control and congestion control. The primary objective is to save scarce sensor resources and increase network efficiency and to ensure network stability.

Several solutions have been proposed to address the transport layer problems in Wireless Sensor Networks (WSN). For example, in [9], Event-to-Sink Reliable Transport (ESRT) protocol is proposed to achieve reliable event detection with minimum energy expenditure. However, the ESRT mechanism relies on spatial correlation among event flows that may not be easily obtainable in underwater acoustic sensor networks. Hence, further investigation is needed to fully develop this type of spatially dependent strategy.

4.4. Case Study of Existing Implementation

There is a lot of potential for underwater acoustic communication. Some creative solutions have been proposed to overcome the theoretical and practical limitations.

In order to illustrate the point an excerpt is taken from: Stojanovic, M. Proakis, J.G. and Catipovic, J.A. *Performance of High-Rate Adaptive Equalization on a Shallow Water Acoustic Channel* [7]. This is a real world design and implementation. A lot of the source material is taken as is. All the information and results are taken directly from [7]. An entire communication scheme was designed to operate acoustically underwater and after design and implementation it was tested and the results analyzed. Comments relevant to, as well as differing from sonar swarms, are mentioned in italics in brackets.

An underwater acoustic sensor network was required. The setup was as follows:

- Network consisted of numerous nodes operating in shallow water, between 50 – 100 meters under ocean surface,
- Nodes were mounted on the bottom (*In our system the nodes are un-tethered*) and separated by distances of up to 10 km (*Sonar swarm implementation will most probably require much shorter distances i.e. 50m up to a maximum of 2 km, but if required longer distances could be feasible*),
- No special positioning of node placement is assumed, allowing for applications that require rapid, random deployment in unknown areas (*Similar to this work's objective*),
- Nodes' hardware designed with sensors capable of detecting presence of various objects, or discerning activity in the area such as magnetic or thermal disturbances,

- Nodes have acoustic communication modems that transmit and receive information.
- Maximum number of nodes not restricted, with several tens of nodes possibly assigned per master node (*Clearly a combination of Ad-hoc and deterministic*).
- Master node connected to surface buoy from which a radio link is used to send data to and from the shore.
- Nodes connected to the master node in hierarchical manner, similar to that of a tree,
- Desired data rate of 100 bit/s per node,
- Network is asynchronous and nodes transmit information as it becomes available,
- Packet sizes of 256 bits,
- Transmission mode is half-duplex.

To conserve energy, the nodes must have two modes of operation: a sleep mode and an active mode. In the sleep mode a node consumes much less energy than in the active mode. A node is awakened from sleep acoustically and when asleep each node continually looks for a low-power wakeup signal. The goal of the network design is to achieve a good trade-off between quality of service and the energy consumption. In particular, a good quality of service includes maximum information throughput at minimum delay, as well as network reliability.

A. Network Topology and Energy Consumption

One of the network design goals is to minimise the energy consumption while providing reliable connectivity between the nodes in the network and the master node (*A realistic goal in all underwater communication networks*). To assess the tradeoffs involved in such a design, we analyze the large-scale system and channel parameters that determine the energy consumption of the network.

Assuming that any number of connections may be supported by a chosen multiple-access strategy, we want to determine the energy consumption as a function of network topology, i.e., as a function of the connections established between the network nodes. As is pointed out, due to the nature of the application, the actual network geometry is such that node locations cannot be determined precisely or beforehand. On the other hand, the authors claim that it is reasonable to assume in the first approximation that a uniform node distribution provides best area coverage and would be a beneficial starting point.

To quantify the energy consumption of the network, a simplified scenario is investigated in which nodes and a master node is arranged linearly along a straight line. The nodes are uniformly distributed so that the distance between each two adjacent nodes, including the distance from the base station to the node closest to it, is r/N. Two extreme communication strategies are possible in this scenario. In the first strategy, each node has a direct access to the master node (fully connected topology). In the second strategy, each node transmits only to its nearest neighbour, who then relays the information toward the master node (multi-hop peer-to-peer topology) (*Multi-hop is a much more efficient way communicating as it saves quite a significant amount of energy, as shown later in this work*).

The energy consumption for both of these cases is determined below. To determine the energy needed for transmission of a single data packet of duration T_p from a node at distance r to the master node, over N relay hops, we assume that a required quality of reception is achieved if the received power level is P_0. To achieve a power level P_0 at the input to the receiver at distance x, the transmitter power needs to be $P_0/A(x)$, where $A(x)$ is the attenuation. The attenuation is given as

$$A(x) = x^k a^x \qquad (4.1)$$

k is the energy spreading factor (k is 1 for cylindrical, 1.5 for practical, and 2 for spherical spreading), and

$$a = 10^{\alpha(f)/10}, \tag{4.2}$$

is a frequency-dependent term obtained from the absorption coefficient. The absorption coefficient $\alpha(f)$ for the frequency range of interest is calculated according to Thorp's expression [7] as

$$\alpha(f) = 0.11 * (f^2 / (1+ f^2)) + 44 * (f^2 / (4100 + f^2)) + 2.75*10^{-4} * f^2 + 0.003, \tag{4.3}$$

in dB/km for f in kHz. To transmit a data packet from one node to another over a distance r/N, each node needs to transmit at a power level $P_1 = P_0/A/(r/N)$, and each needs to transmit for the duration of a packet T_p. Hence, the total consumed energy for transmission over N hops is

$$E = N^* P_1{}^* T_p = P_0{}^*T_p{}^*N/A(r/N).\ [7] \tag{4.4}$$

When each of the N nodes has a packet to transmit, the total consumed energy for packet relaying is

$$\begin{aligned} E_{rel} &= P_0{}^*T_p/A(r/N) + 2^*P_0{}^*T_p{}^*N/A(r/N) + 3^*P_0{}^*T_p/A(r/N) + \dots + \\ &\qquad N^*P_0{}^*T_p{}^*N/A(r/N) \\ &= (N(N+1)^*P_0{}^*T_p / 2^* A(r/N)).\ [7] \end{aligned} \tag{4.5}$$

For a direct-access strategy, the total consumed energy is

$$\begin{aligned} E_{dir} &= P_0{}^*T_p/A(r/N) + P_0{}^*T_p/A(2r/N) + P_0{}^*T_p/A(3r/N) + \dots + P_0{}^*T_p/A(Nr/N) \\ &= P_0{}^*T_p / \Sigma_{i=1}^{N} A(ir/N).\ [7] \end{aligned} \tag{4.6}$$

Fig. 4.1 shows total consumed energy for both strategies. Dashed curves represent the case of direct access, which obviously requires more energy. For direct access, inclusion of each additional node results in an increase in total energy. For relaying, the situation is reversed: inclusion of each additional node decreases the total energy consumption because the additional node serves as an additional relay along the same distance. (*This is an interesting point to note and should be taken into consideration when discussing possible solutions to the communication protocol paradigm.*)

The price to pay, of course, is the need for a sophisticated communications protocol and the increased packet delay. Hence the strategy that minimises energy consumption is definitely multi-hop routing. The savings in energy increases as the number of hops increase, and become more pronounced at longer distances. Specifically, for the distances in the order of several tens of kilometres that are of interest to the case study, the savings are significant. Consequently, a network topology that implements connections only between the neighbouring nodes is preferable.

B. Media Access Protocol

If a multiple access strategy is chosen it is generally either FDMA or CDMA. In the case of FDMA, acoustic modems that employ Hadamard coded MFSK modulation are implemented. As an alternative to FDMA, direct sequence spread-spectrum signalling and CDMA are also considered for the network. A Rake receiver is employed at the receiver to make use of the multi-path propagation of the shallow-water acoustic channel (*hardware used to resolve an ocean related restriction*). The media access protocol for the shallow-water network is based on the MACA protocol, which uses RTS-CTS-DATA (request to send-clear to send-send data) exchange.

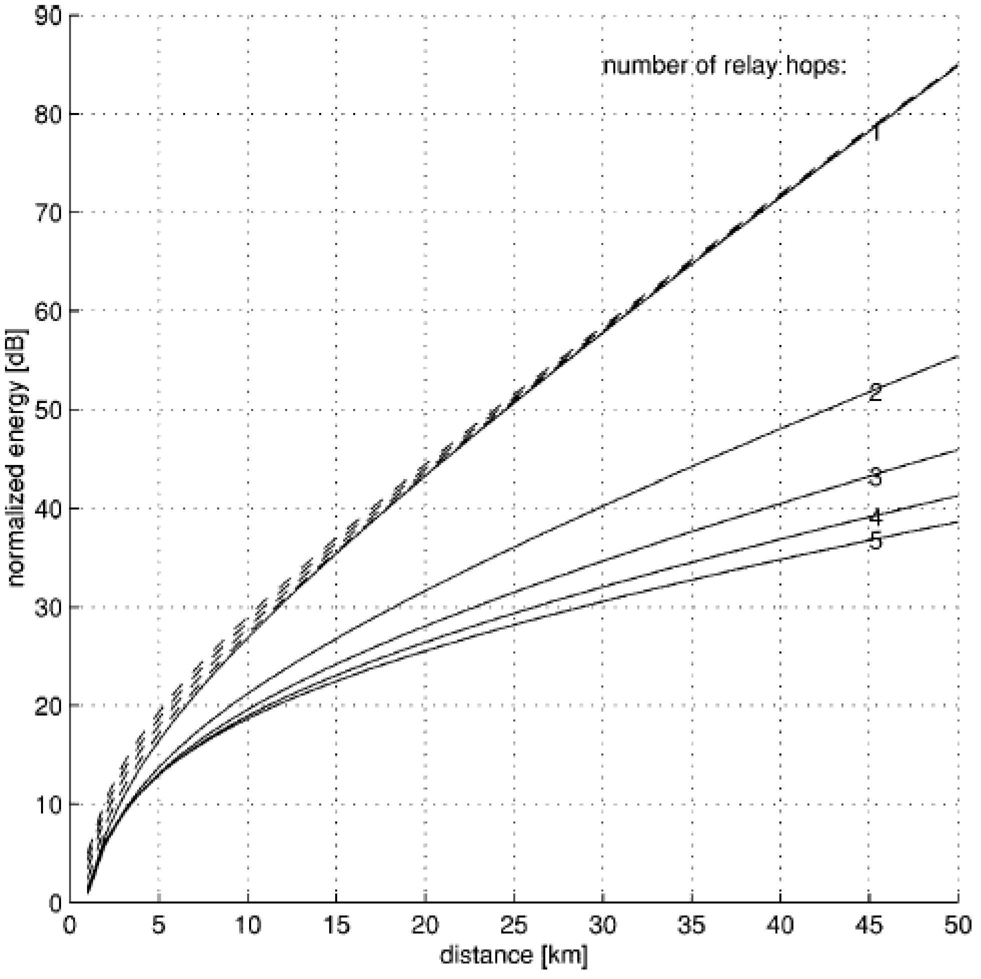

Fig. 4.1 Total (normalised) energy needed to transmit a packet from each of *N* nodes to the master node. Solid curves represent relaying, dashed curves represent direct access. The parameter on the curves is the number of hops. For relaying, the number of hops increases from the top curve downwards: although more packets are sent when there are more hops, total energy consumption is lower. For direct access, there is little difference between the curves, and the situation is reversed: the number of hops increases from 1 for the lowest energy consumption to 5 for the highest [7].

The network employs the Stop & Wait ARQ scheme. If the source cannot receive CTS from the destination after a predetermined time interval, it repeats RTS. If, after trials of RTS, the source cannot receive a CTS, it decides that the link is no longer available and returns to a low-power state. If the source receives a CTS, it immediately transmits the data packet. An ACK (*Flow control is a requirement in the communications protocol to satisfy level 4 of the ISO model)* signal is sent by the destination upon the receipt of a correct data packet to provide positive acknowledgment to the source in the data link layer.

During a transaction, all other transmission requests are declined. If the source of the transmission is not informed, it repeats its request. In doing so, the source increases its power as dictated by the power control algorithm. This situation results in increased battery consumption and increased probability of collisions. To prevent these undesirable effects, a WAIT command is added. This command informs the source node that the destination is busy and will send CTS as soon as possible.

If two nodes send an RTS to each other, a deadlock may occur because both nodes will issue the WAIT command when they receive RTS's from the other node. Each node will then wait forever for the other node to send CTS. This problem is solved by assigning priority to the packets that are directed toward the master node, as explained below. Assume that node A is a lower level node than node B, that is, A is closer to the master and, thus, the parent of B. Node A and node B both send RTS's to each other. Due to transmission delays, packets arrive at their destinations while both nodes are waiting for a CTS packet. When node B receives the RTS, it notices that the packet is from its own destination, node A. Node B checks whether node A is its parent or child. By that time, node A receives the RTS of node B and does the same check. Because node B is its child, node A decides that it should wait for node B to complete its data transmission. Therefore, node A sends a CTS immediately, puts its own data packet into a buffer, and waits for the CTS packet of node B.

C. Initialisation and Routing

Since the network in consideration is an *Ad-hoc* network (*In the case study excerpt they chose an Ad-hoc structure, whereas in the case of this project a choice still has to be made between an ad-hoc and deterministic strategy. This is* *a fundamental part of the communications protocol and will have to be carefully considered in order to make the correct choice)*, an initialisation algorithm is needed to establish preliminary connections autonomously. This algorithm is based on polling, and as such it guarantees connectivity to all the nodes that are acoustically reachable by at least one of their nearest neighbours. During initialisation, the nodes create neighbour tables.

These tables contain a list of each node's neighbours and a quality measure of their link, which can be the received SNR from the corresponding neighbour. The neighbour tables are then sent to the master node and a routing tree is formed. The master node decides on the primary (and secondary) routes to each destination. Initialisation ends when the master node sends primary routes to the nodes. It provides either a single set of connections or multiple connections between the nodes. Multiple connections are desirable to provide greater robustness to failures.

In the design of the case study, the master node creates a routing tree depending on the neighbour tables reported by its nodes. If a node reports that a link's performance has degraded or it is no longer available, the master node selects new routes that take the place of the failed link. The changes in the routing tree are reported to all related nodes. This procedure ensures that nodes will not attempt to use a failed link. In this way, unnecessary transmissions that increase battery consumption are avoided.

D. Results

According to [7] the network performance was tested using Opnet's Radio Modeller tool (*In our case we shall use Erlang. More detail about Erlang will be given in the design and implementation chapter*). The characteristics include the transmission loss as a function of distance and frequency, and the probability of error as a function of the SNR.

To end of the analysis of this case study, two things must be considered:

1. The data packet generation process is simulated as a Poisson process,

2. The purpose of the case study is to indicate a possible direction of design and not to focus on the specific results presented in the reference, therefore, we shall not go into any more detail.

4.5. Initial Investigation into Possible Solutions

Subsequent to presenting some theory and current implementations, possible solutions could be investigated:

- Starting with hardware, the first aspect is power. Power indirectly influences the communications protocol and plays a major role in sonar swarms. The communications protocol then has to be designed in such a way as to facilitate low power consumption. Therefore, when designing the communications protocol, methods such as sleep and listen modes have to be implemented.

- An aspect that is directly related to the communications protocol and also affects power is processing requirements. The more processing is required, the more power is consumed so the algorithm cannot be too complex. If the protocol is too complex we also incur overhead, which slows down the overall communications rate as well as lowering protocol efficiency. The overhead, however, is more critical for this implementation as this more a hardware issue and outside the scope of this document.

- Processing power might be reduced by using smaller packets to send the data, but a balance has to be achieved. If the packets are too small, more overhead is incurred in sending more packets, while if the packets are too large, more transmitting energy is required in which case more power is depleted and the packet is also more noise prone due to increased bit count.

- It can be shown that a multi-hop algorithm is much more efficient than an algorithm that uses direct paths between nodes [6]. It is also more power efficient [10].

- Furthermore the use of multi-hop communication could possibly solve other problems, i.e. limited communications range and signal strength. To conserve power, signal strength can be reduced. It has to be kept in mind though that this in turn limits communications range as well as the size of packets that can be sent.

- Multi-hop routing could also solve, or at least minimise the problem of fading and multi-path. Fading and multi-path increases with distance, therefore if communication takes place over a multi-hop stratagem with a minimum distance between each hop, then each hop regenerates the signal before sending to the next hop, which in turn decreases multi-path and fading, but might introduce added latency. It has to be kept in mind though that minimum distance might not be the optimal distance as all factors influenced by distance have to be weighed against each other.

- Storage space does not directly affect the communications protocol, except in terms of the routing table and the message buffer. Neither of these two would under normal circumstances take up a lot of space, so this does not affect the protocol in a very adverse manner. It will still be advantageous however, if the messages and packets to be sent are kept to a reasonable size. Long-term storage does not directly affect the routing protocol and will thus not be discussed in any more depth than in the previous chapter.

- Cost will only be mentioned briefly as it has no direct effect on the algorithm. It is however desirable to have a protocol that is cost efficient. If it is cost efficient it will fit in better with the overall concept of the swarm, which consists of numerous, small and inexpensive nodes. So, if the protocol results in low design and implementation cost, it will be a bonus.

- Inaccessibility means that the algorithm has to be fault tolerant and as close to 100% reliable as possible. The reason is obvious – once the protocol is embedded and the nodes are deployed, it cannot be altered and must therefore function correctly for all scenarios, as far as possible.

- Error handling and lost messages are two critical factors. Due to the noisy nature of any communications medium, especially the ocean, it is logical that errors will take place. To make sure that these errors do not adversely affect the routing and communication protocol, some form of error handling has to be introduced. There are various methods to handle errors – Forward Error Correction (FEC) which corrects errors, Cyclic Redundancy Checks (CRC) which checks for errors and various other methods that either check for errors or possibly correct them. The final choice will be influenced by reliability requirements (therefore probability of error), processing requirements/overhead, as well as its impact on throughput.

- Some form of link-layer control, or data flow control, will be required to solve the problem of duplicate or lost messages (caused by factors such as noise and multi-path).

- Hidden and exposed nodes are a problem that can be solved either by hardware or software methods, i.e. making sure not more than one node is communicating at one time by having some form of communications control, or each node can have a list of nodes which are within one hop of communication. Whatever choice is made, it has to be in compliance with the overall protocol.

- Another fundamental question is whether to use ad-hoc or deterministic as the routing basis, or perhaps even a hybrid. The pros and cons have already been discussed in the previous chapters and the final choice and motivations will be explained in the next chapter.

- High (propagation and transmission) delay is an important factor that will play a role in the protocol. A high delay means that communications throughput will be severely reduced. This is attributed to the fact that when routing or communication is taking place, all participating nodes will have to wait suitably long when expecting an arrival or before deciding whether to retransmit.

4.6. Conclusion

Chapters 2 and 3 gave an insight into factors that would directly and indirectly influence the communications protocol. In this chapter we looked at the abbreviated ISO/OSI 7 layer communication model, as we only analysed the first four relevant layers. Furthermore, a case study was also included in this chapter to demonstrate the functionality of a current real world implementation, as well as to serve as a possible guide to subsequent work in the current project. Finally, we looked at initial solutions to some of the problems encountered during the background and literature research.

The next chapter will deal with a more detailed protocol design. This protocol framework can be summarised as follows:

- Low processor overhead and minimal complexity,
- Maximise throughput efficiency,
- Multi hop routing,
- Sleep and listen modes,
- Small, reasonably sized data packets,
- Protocol must cater for multiple nodes,
- Routing table must not be too complex and large,
- If possible, cheap to design and implement,
- Fault and error tolerant,
- Error handling and possibly, error correction,
- Some form of flow control required,
- Avoidance of hidden and exposed nodes,
- Strategy should take high delay caused by ocean environment into account.

After reviewing the protocol outline the next steps to take will then be:

- Deciding on a routing structure: Ad-hoc or Deterministic,
- A length for the data packets will be determined,
- A choice made between error detection and error correction and which type of error correction in the case of the latter,
- Type of flow control,
- Simulate the proposed design, and
- Evaluate the performance.

This should give a good idea of whether the protocol is functional and applicable, as well as possible changes that could be made. Furthermore, it will provide insight into doing further work and research in future works. Finally it will lay the basis for the design of a communications protocol for practical implementation, beyond just simulation.

Chapter 5
Analysis of Possible Solutions and Implementations with Regard to ISO model

5.1. Introduction

The goal of this chapter is to consider some possible solutions to designing a communications protocol for an underwater sonar swarm or communication system. Therefore, we shall be taking each of the four levels of the ISO truncated model and discuss certain aspects as well as some possible solutions. We shall also try and quantify some of the choices that are made

From there, the proposed solutions will be simulated and in Chapter 6 we shall be analysing the results. We shall also be doing a critical discussion of the results as well as determining whether more simulations should be run for the purposes of this work.

5.2. Analysis of Possible Solutions with Relation to the ISO Model

5.2.1. Physical Layer

Due to the nature of this work we shall not be focusing on the physical layer any more than we already have. There are various possible solutions that could be employed in terms of the physical layer and in the previous chapter we briefly mentioned quite a few. Overall, this layer has to ensure that the conversion of logical 1’s and 0’s into some signal, takes place in an efficient manner.

Some references place error handling in this layer, but this is uncommon. For the purposes of this work we shall be placing error handling in the transport layer [21, 22].

5.2.2. Data Link Layer

In the data link layer we shall be looking at two critical factors, the first of these being packet lengths, and secondly the choice between using a contention or a contentionless oriented communications scheme.

5.2.2.1. Packet Lengths

Choosing packet lengths is dependent on various factors:

- Type of information to send,
- Bit or packet rate (Throughput),
- Bit Error rate (BER),
- Error handling,
- Propagation Delay.

Therefore, when deciding on a packet length the characteristics of the medium, as well as the communication system will have a definite effect. At this stage it is however prudent to remember that Erlang handles the physical layer of communication. Therefore it handles addressing as well as the baud rate and hides this from the user. Therefore, when deciding on the packet sizes, provision will be made only for the data portion of the packet and not the address as well.

The size will, however, affect the eventual hardware implementation, but the packet sizes are easily adjusted and the hardware implementation is, furthermore, beyond the scope of this specific document. There is, however, a slight problem in that Erlang handles the addressing by itself and errors cannot be introduced into the address portion of the message that is to be sent. If required however, some way of simulating noise in the address portion will have to be devised. At this stage however the focus will only be on the data portion

Underwater acoustic sensor networks will most probably only send telemetry, sonar, seismic or similar types of information and not video or speech data. Therefore, the size of the data packets need not be very large. Furthermore, as discussed previously, the BER requirements are not as stringent as those for video and speech, but the BER requirements will still play a prominent role. The size of the data packets will, therefore, be affected by the type of error handling employed which will in most likelihood increase the size of the packets, but decrease the effect that noise has on throughput.

There is currently no required bit rate or packet rate specified (Throughput), but as with any form of communication a higher data rate would be desirable and even beneficial. This data rate will be tempered by the propagation delay, as with each packet sent the propagation delay will decrease the data rate, especially if flow control is employed. If flow control is employed, then the propagation delay effect will effectively be doubled.

To explain this, take the time to send a packet as T, propagation delay as τ and finally the time taken to send an acknowledge (Ack) as T_a. Then the minimum time taken to send a packet from one node to the next, without flow control and in the absence of noise and with no retransmissions, would be:

$$T + \tau, \tag{5.1}$$

However, if flow control is introduced then the minimum time taken would be:

$$T + \tau + T_a + \tau, \tag{5.2}$$

Therefore the propagation delay is doubled. This is significant as the propagation delay of the ocean is very high - 0.67 s/km. Due to this it would be an advantage to send larger packets, which in turn would cause fewer transmissions (in the absence of noise) and thus decrease the effect that the propagation delay has on the throughput and increase the throughput. However, larger packets are more susceptible to the influence of noise and this in turn could cause the corruption of these data packets. In the case of corrupted data packets the associated data packets have to be resent and this again decreases the throughput.

Somehow a middle ground has to be achieved between the effects of propagation delay and noise, while still attaining a high data rate and at the same time keeping in mind that Erlang hides the baud rate as well as the physical layer implementation. Eventually it was deemed that the final choice of packet length would only be made after the analysis of the error handling. It was, however, decided to choose an initial packet length of either 4 or 5 bits. This choice was made on the need to keep the packet lengths small as well as simplistic.

It is however prudent to show a comparison between the choice of 4 bit packets and 5 bit packets in Table 5.1 and Table 5.2, as well as the pros and cons in Table 5.3. The analysis was made with a normal distribution of errors and in the absence of burst errors. Burst errors are, however, possible and cannot be discounted, but for the purpose of an equal comparison burst errors are ignored, as they are difficult to quantify.

For Table 5.1 assume the following:

- Bit rate of 1kbs for 10 seconds which gives 10 kb (10000 bits) sent
- Choose worst case $P_e = 1 \times 10^{-2} = 0.01$, therefore 1 error per 100 bits

	4 bit	5 bit
Packets in 10 kb	2500	2000
Bit errors per 10 kb	100	100
Assuming normal distribution of errors average number of errors per packet	0.04	0.05
Total introduced propagation delay without flow control in absence of noise	2500	2000
Propagation delay count with flow control in absence of noise	5000	4000

Table 5.1 Comparison between 4 bit packets and 5 bit packets on the basis bit rate equals 1kb/s

For Table 5.2 assume the following:

- Packet rate of 1kp/s for 10 seconds which gives 10 kp (10000 packets) sent
- Choose worst case $P_e = 1 \times 10^{-2} = 0.01$, therefore 1 error per 100 bits

	4 bit	5 bit
Bits in 10 kp	40000	50000
Bit errors per 10 kp	400	500
Assuming normal distribution of errors average number of errors per packet	0.04	0.05
Total introduced propagation delay without flow control in absence of noise	10000	10000
Propagation delay count with flow control in absence of noise	20000	20000

Table 5.2 Comparison between 4 bit packets and 5 bit packets on the basis packet rate equals 1kp/s (1000 packets / second)

From the above results the following observations can be made:

- A choice of bit rate will affect the choice of packet size,
- For the 5 bit case the propagation delay effect is approximately 20 % less than for the 4 bit case (Table 5.1),
- For the 4 bit case the noise effect is reduced by approximately 20 % above that of the 5 bit case (Table 5.1),
- The final choice will be weighted between influence of noise, propagation delay and required bit rate.

4 bit packets	5 bit packets
Higher packet rate when bit rate is equal	More information per packet
Assuming normal distribution, less bit errors per packet, but more packets	Better error correction possibilities when implementing FEC
Fewer total number of bits required to send same number of messages if message size <= 4 bits	More coding flexibility
	Mitigates propagation delay effects

Table 5.3 Advantages of respectively using 4 or 5 bit packets

Finally, to determine how the 4 bit and 5 bit case would perform in the Erlang environment, a simulation was run. During the simulation a server node would send a packet to a receiver and the receiver would then reply with an Ack of exactly the same size. On each leg a propagation delay of 0.67s was simulated, as well as a BER of 1 x 10^{-2}. The noise was simulated using a normal distributed random number generator. The above was then repeated for 10000 iterations and the duration as well as the number of errors logged. The result is shown in Table 5.4

It can be seen that the calculated bit rate for the 5 bit is case is higher than for the 4 bit case, while in turn it also has a higher calculated error rate. The higher bit rate can be attributed to the higher volume of bits sent in the case of the 5 bit implementation. This higher volume, however, also contributes to the higher total error rate calculated with the 5 bit implementation. However, taking Table 5.4 into consideration, the final choice of which packet length to employ will be determined by the type and implementation of the error handling chosen. It is obvious that there are various factors influencing the choice of packet length, but the two largest are the BER as well as the propagation delay influence.

	4 Bit	5 Bit
Number of iterations	10000	10000
Bits sent to receiver	40000	50000
Number of error bits detected at receiver	425	507
Ack bits received by server	40000	50000
Total bits sent	80000	100000
Number of error bits detected at server	345	497
Total number of errors	770	1004
Bit Error Rate	0.9625×10^{-2}	1.004×10^{-2}
Transmission duration (in seconds)	13437.625	13437.5
Effective bit rate (bits / second)	5.9534	7.4418
Effective error rate (errors/second)	0.0573	0.0747

Table 5.4 Comparison of the simulation results of communication using respectively 4 bit and 5 bit packets

However, it is evident that with the high propagation delay of the ocean environment, the focus will be more on the propagation delay than the BER. If a method such as forward error correction is deemed necessary and implemented, it will decrease the effective BER.

Also, since a P_e of 1×10^{-2} is to be simulated (as calculated earlier), it is safe to say that reliability rather than throughput will be of greater importance in an environment as unpredictable and noisy as the ocean.

In Table 6.1 in Chapter 6, a more in depth comparison is made between differing packet sizes, as well as RM encoding schemes. This was done so as to display the difference between using FEC vs. no error correction, but also to show gains / losses between using smaller / larger packet sizes, both with the presence of FEC and the lack thereof.

5.2.2.2. Contentionless vs. Contention Oriented

In this part of the chapter we shall be looking at the benefits of using a contentionless scheme vs. a contention oriented scheme. We shall also be doing a mathematical analysis, in terms of ocean factors, to determine which one would fare better in the ocean environment, and according to the results, make a choice between the two.

We will start by comparing the difference in channel utilisation, *S*, and average delay, *D*, of contentionless scheduled polling with that of non-persistent CSMA, which is a very typical, widely used contention based protocol type. Scheduled polling is used to facilitate a comparison as scheduled polling is a widely used scheme and is suitable for various forms of adaptation. Using the following variables:

λ = *aggregate mean packet generation rate in packets/s*
τ = *propagation delay = 0.67 s/km* [1]
T = *Time in seconds it takes to transmit one packet*
T_a = *Time taken to transmit an Acknowledge (Ack)*
$a = \tau / T$ =*Normalised propagation delay*
X = *Average backoff time*

G = Packets per transmission time T (Normalised channel traffic demand)

P_s = Probability that a packet transmits successfully

$S = \lambda \times T$ = Channel Utilisation (Normalised channel Throughput)

R = Average delay between two consecutive messages

D = Average end-to-end delay for successful packet transmission

Now from [19], *S* can be defined as:

$S = GP_s$ = *Channel Utilisation* (5.3)

Or,

$S = U / (B + I)$ (5.4)

Where *U* is the time spent for successful transmissions, *B* is the time that the channel is busy (time spent on successful transmissions and retransmissions) and *I* is the time that the channel spends idle. It can be seen from above that *U* is a subset of *B*. Using the above equations and those found in [19] and [20], an expression will be derived for the throughput of scheduled polling, as well as the average delay.

Furthermore, the equations found in [19] will be used for CSMA. It was also decided that to be able to compare the results for CSMA from [19] and [20], it would be feasible to normalise the values of *a*, *T* and T_a in such a way that comparison would be easy [19]. Therefore, the following values were chosen to create a basis for comparison:

$T = 1s$

$T_a = T = 1s$

It was also decided to select the distance between nodes at 1km, this would simplify the calculation of *a* (the propagation delay) and maintain the required basis for comparison. Therefore:

$a = \tau / T = 0.67$

5.2.2.2.1 Comparing the Throughput of Non-Persistent CSMA against that of a Scheduled Polling Strategy

In [20] the channel utilisation for non-persistent CSMA is given as:

$$S = (Ge^{-aG}) / (G(1 + 2a) + e^{-aG}) \qquad (5.5)$$

Figure 5.1 shows the curves of *S* versus *G* of non-persistent CSMA for differing values of *a* and *G*. To clarify the significance of *G*, it has to be explained that *G* is the number of packets per second that is offered to the network to transmit. Therefore, as the packet generation rate is increased at each node, G increases. These packets have to be sent or handled before nodes start discarding packets due to limited buffer space. Therefore, *G* can also be seen as the load that has to be handled by the routing algorithm and the communications system. The way in which the load is handled is the normalised channel utilisation – packets sent as soon as possible after they are generated and how effectively the available channel capacity is utilised. It also has to be remembered that *G* not only includes newly generated packets, but also includes packets that have to be resent due to noise corruption and/or packet loss. Therefore, *G* encompasses all packets to be handled by the communications system.

A good benchmark of a system is to determine how the channel utilisation, *S*, varies as the load of the system increases or decreases. Therefore, as will be shown later, the offered load *G* is varied over some range of values to determine how the channel utilisation changes as the offered load changes and as a result how the system handles changing and increasing loads.

Now, from Figure 5.1 it can clearly be seen that as *a* increases, the channel utilisation severely decreases. Therefore, over longer distances, CSMA would also perform decidedly worse as the propagation delay is also a function of distance, *τ = 0.67 s/km.*

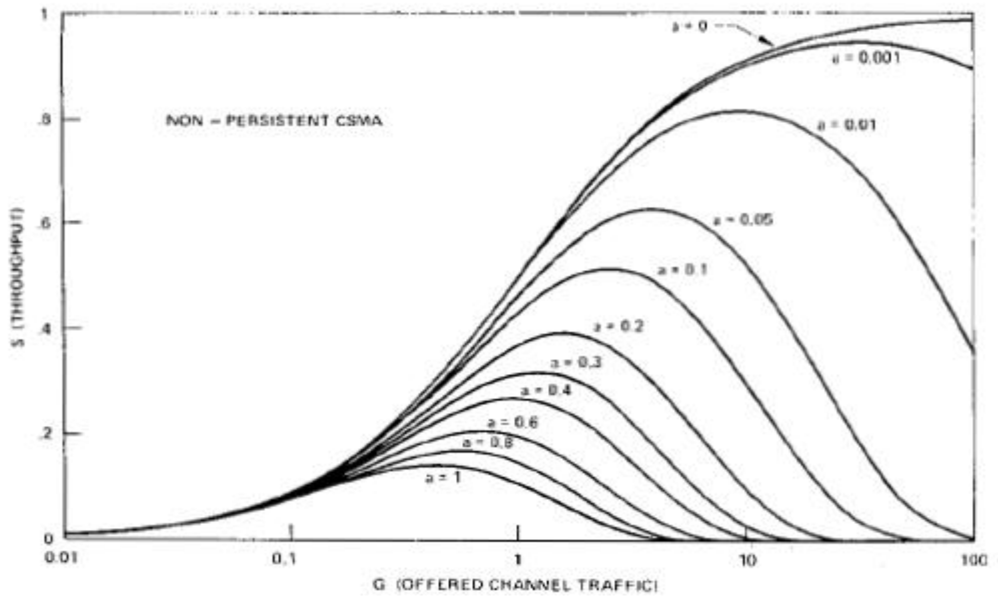

Figure 5.1 Throughput in non-persistent CSMA for differing values of *a* [19]

By implementing Equation 5.4, an equation for the channel utilisation for scheduled polling will be derived to compare with that of CSMA. The first part of the equation to be discussed is that of the value of *I*, where *I* is the average duration of the idle period. Therefore, if *G* is the offered channel load, then the average duration of an idle period would simply be the inverse of *G* [20]. Therefore we have that:

$$I = 1 / G \tag{5.6}$$

Now expressions for U and B have to be derived. To do so, keep in mind that *U* and *B* are metrics of channel busy times. Taking this into consideration and calculating the times when the channel is busy with successful transmissions, it is found that:

$$U = T + T_a + 2a \tag{5.7}$$

This is due to the fact that when the channel is busy with a successful transmission, it is equal to the time spent sending the packet (*T*), plus the time spent sending an Ack (*T_a*), plus round trip propagation delay (*2a*).

To find an expression for *B* first remember that *U* is a subset of *B*, but *B* also consists of the time that the channel is busy with retransmission. Therefore, *B* can be derived as:

$$\begin{aligned} B &= U + p * U \\ &= (T + T_a + 2a) + p(T + T_a + 2a), \end{aligned} \tag{5.8}$$

where *p* is the probability of an error due to noise, or in other words the packet loss % due to the BER. Therefore, the time spent on retransmissions is equal to the time spent on a successful transmission multiplied by the probability of a retransmission. Therefore, the *S* of scheduled polling can now be expressed as:

$$\begin{aligned} S &= U / (B + I) \\ &= U / (U + p * U + I) \\ &= ((T + T_a + 2a)G) / ((T + T_a + 2a) + (p(T + T_a + 2a)G + 1) \end{aligned} \tag{5.9}$$

To give a better idea of the change in *S* as *G* changes, *G* was chosen as a spread of values, starting at 0.0001 and ending at 10. This was also done to show that as the normalised offered channel traffic *G* increased, the algorithm became increasingly more efficient in terms of channel utilisation *S,* up to a point. It was deemed unnecessary to go beyond a value of 10 for *G,* as scheduled polling starts to level off near a value of 1 for the channel utilisation when *G* = 10 and higher, while for CSMA and ALOHA, *S* starts to collapse totally for G > 1.

These values were then plotted and Figure 5.2 shows a comparison of *S* vs. *G* for ALOHA, non-persistent CSMA, as well as scheduled polling for *a = 0.67* and *p = 0.01* (The reason for this choice was defined in Chapter 3 where a desired P_e between 10^{-2} and 10^{-3} was stipulated. For this comparison a worst case scenario was chosen, thus the use of *p = 0.01*).

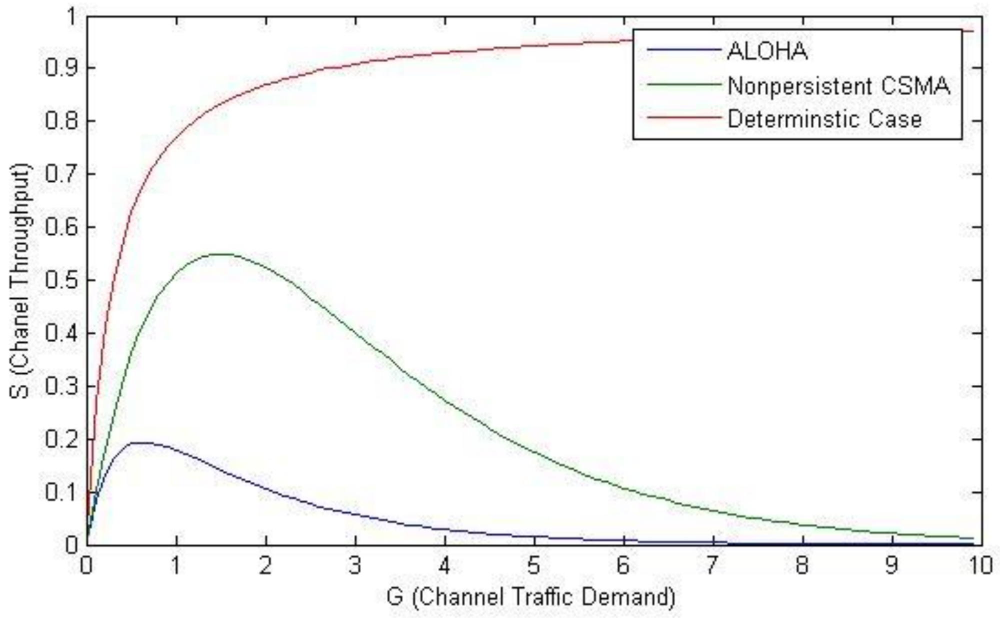

Figure 5.2 *S* vs. *G* for the three protocols for *a* =0.67. Scheduled polling and a deterministic tree perform very similarly.

From Figure 5.2 it can clearly be seen that due to the high value of *a* (normalised propagation delay) as applicable in the marine environment, the scheduled polling algorithm performs far better as CSMA and ALOHA. If *a* were lower (< 0.01) then CSMA would approach a value of 1 for *S,* as can be seen from Figure 5.1, but taking into account Figure 5.1 and 5.2, as *a* increases CSMA decreases in efficiency and as *a* reaches 1, CSMA's curve approaches that of scheduled polling. It also has to be taken into account that as distance increases, scheduled polling would also remain better, as the propagation delay is a function of distance, *τ = 0.67 s/km.* Therefore, taking into account the high propagation delay of the ocean, the deterministic approach would fare better than CSMA in terms of channel traffic demand versus channel throughput.

5.2.2.2.2 Comparing *D* of Non-persistent CSMA against that of the Deterministic Strategy

The average delay between two consecutive messages for CSMA (and for most other communication schemes) can be derived as follows:

$$R = T + \tau + T_a + \tau + X \qquad (5.10)$$

This means that the average delay between two consecutive messages for CSMA is equal to the time it takes to send the original message, plus the time it takes to send an ACK, plus the round-trip propagation delay as well as the average backoff time (x). It has to also be kept in mind that $X >> T$. For the purposes of simulation X was chosen as 100 to be suitably larger than T.

In Figure 5.3 [19], we see the comparison of ALOHA versus the different types of CSMA. As the throughput increases, the delay also increases and ALOHA’s characteristic curve starts to steeply incline, whereas the other protocols have a more even incline. However, it can be seen that beyond some threshold of S, for every scheme, the delay increases dramatically.

For the contentionless case the average delay between two consecutive messages would most probably be:

$$R = T + \tau + T_a + \tau \qquad (5.11)$$

As can be seen the average backoff time is not present in this case. The reason for this is that in contentionless schemes collisions are avoided by the use of a controlled communication structure. Therefore nodes need not employ an average backoff time, due to the absence of collisions. Errors will however occur, but this is taken into consideration in the calculation of *S* as above. Therefore these values are now used in conjunction with the following equation, which represents the average delay for a successful delivery:

$$D = ((G / S) - 1) * R + 1 + a, \qquad (5.12)$$

Where *G* = 0.0001 up to 10 and the values of *S* were calculated using this spread of varying values of *G*. the values of *G / S* were then calculated using the above values and *D* calculated using Equation 5.12 and plotted in Figure 5.4.

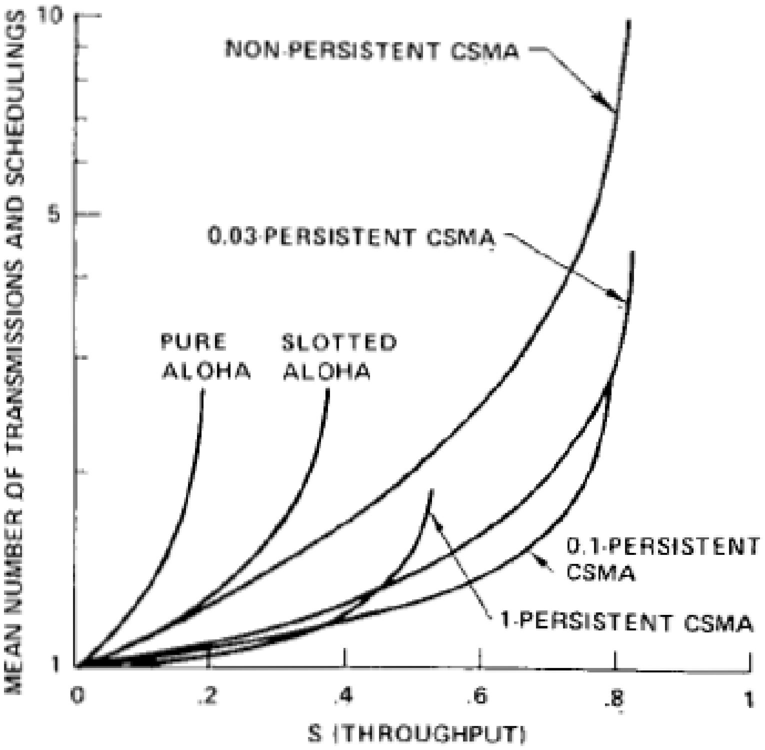

Figure 5.3 Throughput vs. delay for contention orientated protocols [19]

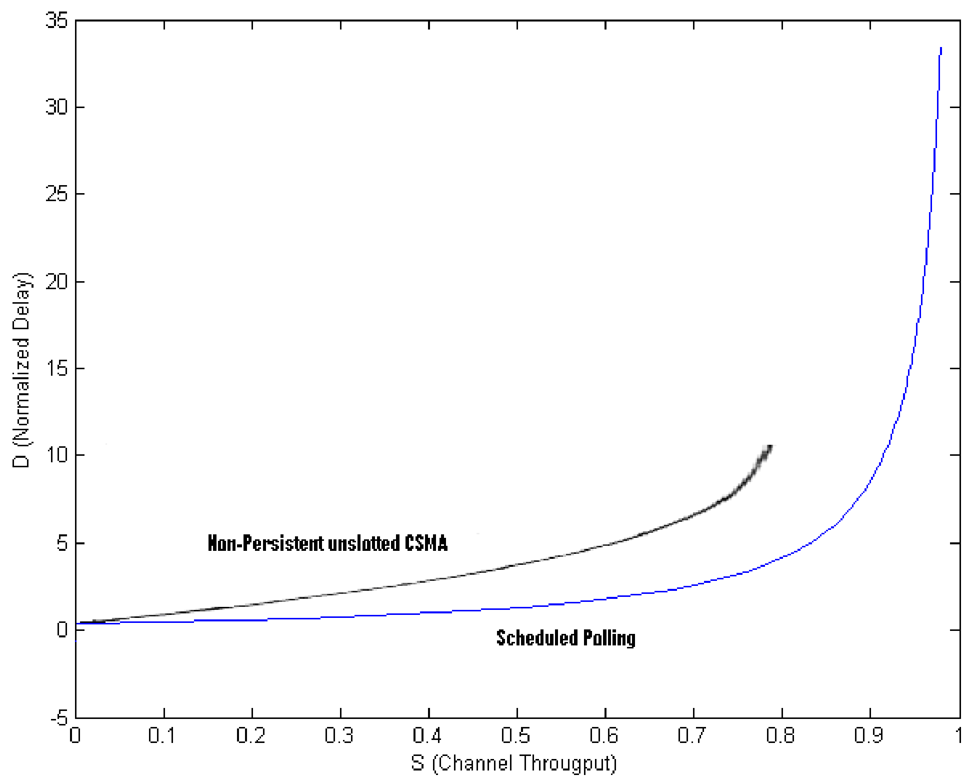

Figure 5.4 Channel Throughput vs. Delay for deterministic routing as well as non-persistent unslotted CSMA.

From Figure 5.4 it can clearly be seen that scheduled polling offers a very good *S* vs. *D* characteristic. The curve of most interest for this work is that of the unslotted non-persistent CSMA in Figure 5.3 and the scheduled polling as displayed in Figure 5.4.Therefore, taking Figure 5.3 vs. Figure 5.4, scheduled polling has a much better characteristic curve in that as *S* (the throughput or normalised channel utilisation) increases, the normalised delay stays reasonably constant. However, from a value of approximately 0.8 the scheduled polling curve does start to incline very rapidly. This however is not unexpected as delays should increase at some point as more packets are being generated at the nodes and this in turn increases the average time that a packet waits to be sent. While this increase does take place, scheduled polling still offers a better result than both ALOHA and CSMA.

5.2.2.2.3 Discussion of the Results

From the above results for both *S* and *D* of the non-persistent CSMA strategy and the scheduled polling strategy, it was found that for the ocean environments characteristically high propagation delay, scheduled polling performs better than typical contention orientated protocols such as CSMA. It was also found that as *a* increases, CSMA becomes decidedly worse.

Therefore, we can now come to the conclusion that scheduling polling and thus a contentionless scheme, is a better option than a contention orientated one for use in the ocean environment, due to the high propagation delays. Therefore, employing a deterministic scheme should provide better performance.

Before moving on to the next section, it is important to draw some comparison between straight scheduled polling and a tree based routing scheme (we will be using a tree based routing scheme as will be discussed next). In both cases communication is controlled by some mother node. Furthermore, each has some method for determining whether any of the nodes have data to transmit. The difference lies herein, in that with scheduled polling the mother node polls each node to determine whether it has data to transmit and therefore has to check each node. In the case of a tree based scheme the mother node checks each child to determine which of those nodes and/or children of those nodes have data to transmit, until it finds a node which has data to transmit. The tree then continues checking each child separately until it finds the next child node, or subsequent children that requires transmission.

This then causes the tree to behave in a similar, but improved way than scheduled polling, in that it performs the same as scheduled polling in the worst case where it has to check each child of each node. However, a tree based routing scheme will not necessarily need to check all nodes during each iteration, especially when packet generation of the communication system is low. At maximum load a tree strategy reduces to scheduled polling. It can therefore be assumed that in a worst case scenario a tree based routing scheme would perform similar to scheduled polling, as depicted in graphs of Figure 5.2 and Figure 5.4, in terms of *S* vs. *G* and *D* vs. *S.*

5.2.3. Network Layer

In the next layer we shall be discussing routing a little more in depth as well as a possible solution for this layer of the communications scheme.

A fundamental choice of protocol design would be a decision for ad-hoc or deterministic operation. This will determine the way routing is done, the format of the routing table, communications rights and hierarchies, etc. In this case deterministic routing was chosen as the most favourable option for an acoustic sonar swarm. The main reasons for this choice are as follows:

- When a node has information to send, the final destination of that information is virtually always some data sink/mother node. Therefore, all nodes need not be able to communicate directly with each other eliminating one of the potential advantages of ad-hoc operation.
- In a deterministic orientated communications protocol all control is placed with the mother node. Therefore, communication will always be orderly as communication cannot take place without authorisation from the mother node. This is an advantage where packet loss could be high. Unnecessary routing operations are also avoided.
- Nodes stay in a sleep status until a request is received from the mother node, thus conserving power.
- Each node need only know of its parent and immediate siblings. Therefore storage requirements as well as processing requirements are lowered.
- One of the deterministic routing algorithms advantages would be a multi-hop approach. This, as mentioned in the previous chapter, was proven by

[7] to be much more effective and power friendly than nodes communicating directly with the mother node.

- Most of the computations in terms of routing and communication are handled by the mother node. Operational complexity on the sonar swarm nodes is therefore reduced.
- Deterministic routing is much less complex to implement than ad-hoc routing with more predictable performance.
- As will be seen in the next section, a contentionless scheme performs better under high propagation delay than a contention oriented scheme. Contention oriented schemes form the backbone of ad-hoc networks in almost all implementations, but these conventional networks are seldom subject to excessive propagation delays.

5.2.3.1. Tree Based Algorithm

A brief description of tree based algorithms and their operation will now be given. Trees consist of multiple nodes, each node having one parent, except the root node, and each node having the possibility of at least one child. A tree starts out as a collection of nodes (Figure 5.5) consisting of various nodes as well as a root. The root node is the only node without a parent. The root node can be dynamically chosen or chosen beforehand depending on requirements. A tree grows by connecting nodes to the root node (Figure 5.6). Every node need not connect directly to the root node, but may connect to a node which itself is connected to the root node (Figure 5.7).

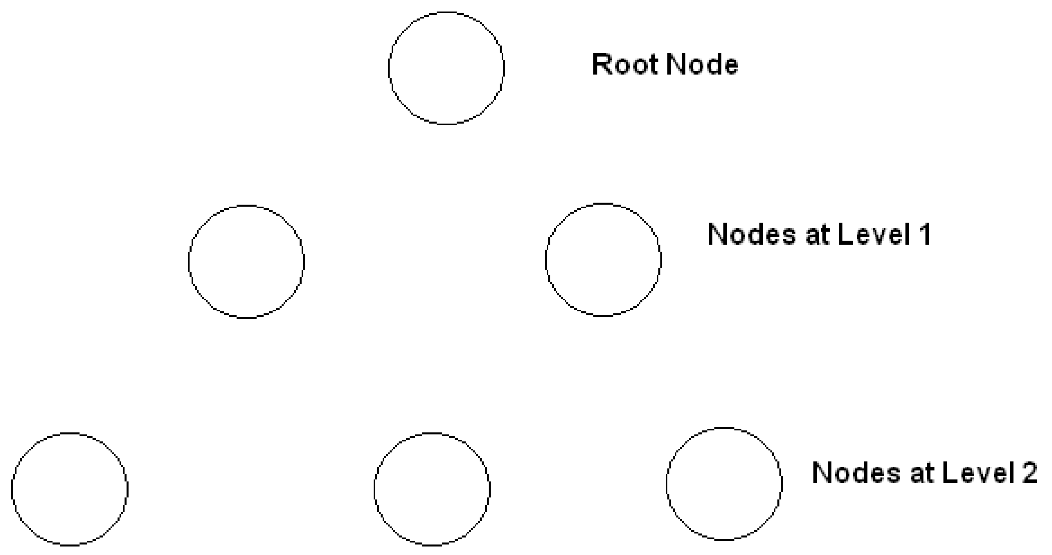

Figure 5.5 Unconnected nodes

The tree then grows until all nodes are connected to each other in some configuration. Figure 5.8 then illustrates the point with all nodes at Level 1 connected to the root and all nodes at Level 2 connected to nodes at Level 1. One node cannot however be connected to more than one higher level node i.e. a node at Level 2 can be connected only to one node at Level 1.

A tree is said to be *n*-ary, where *n* denotes the number of children that each node may have. In the case of Figure 5.8 the tree is said to be binary as each node has at most 2 children.

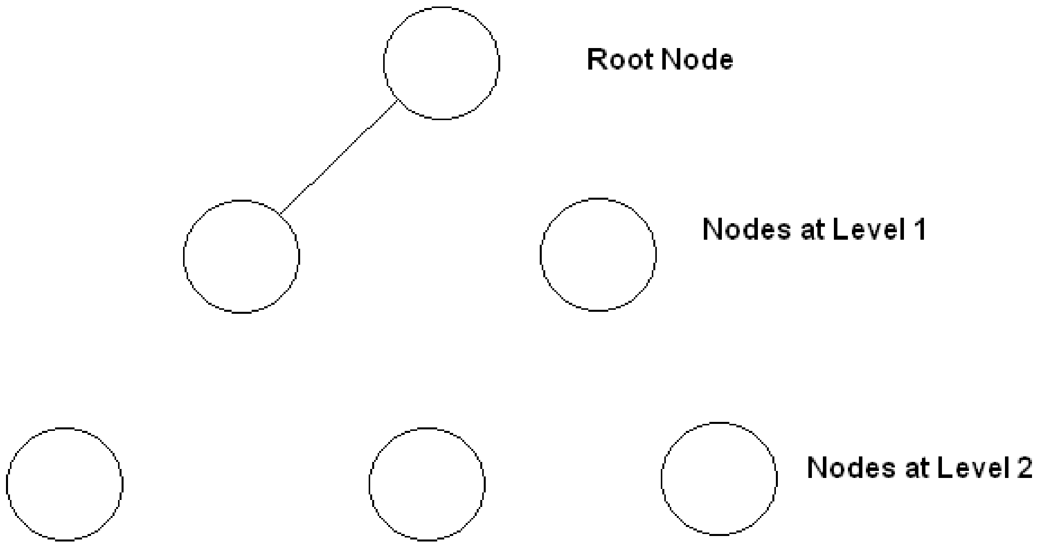

Figure 5.6 One node connected to the root node

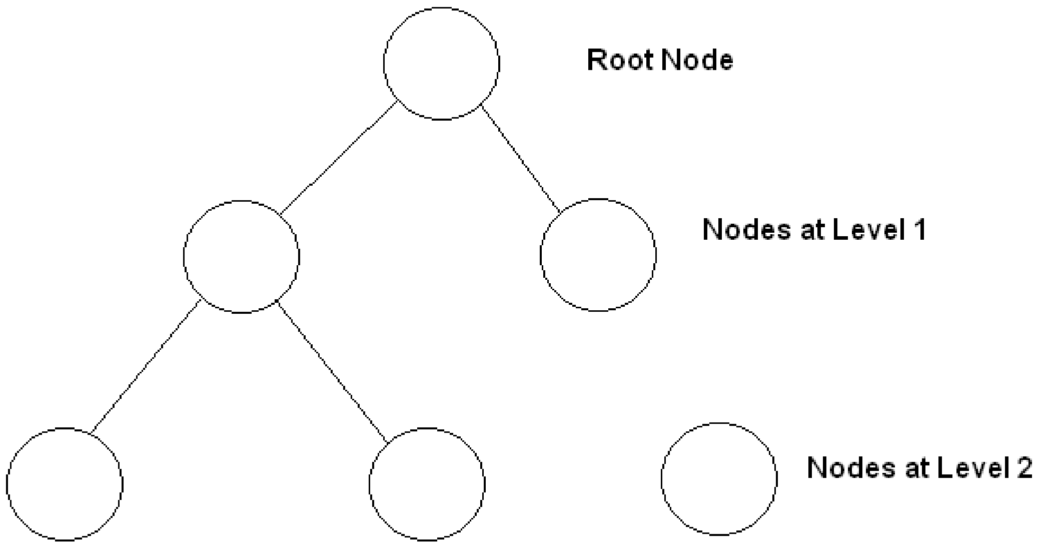

Figure 5.7 Tree grows as more nodes are connected

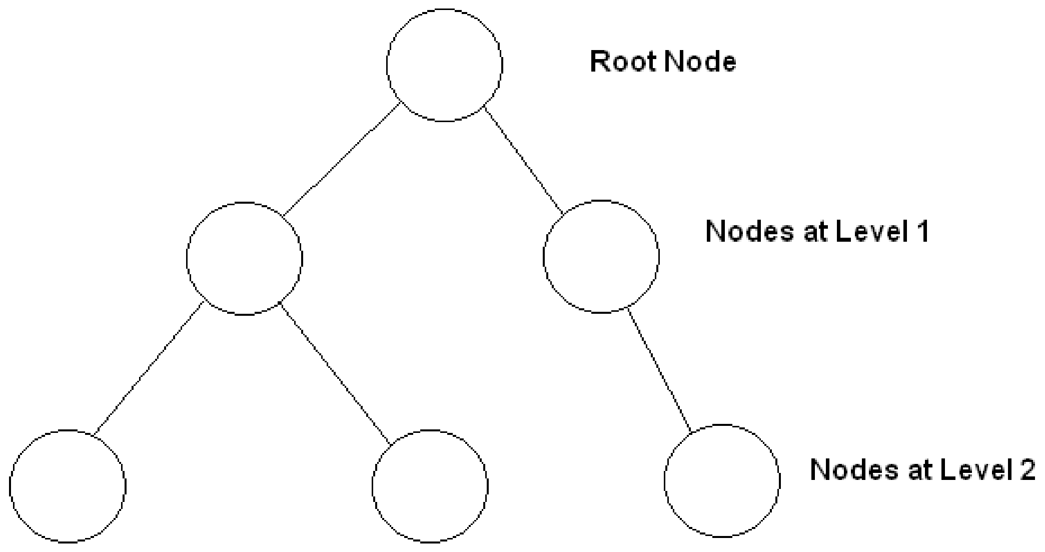

Figure 5.8 fully connected binary tree

The contentionless scheme eventually chosen was of a tree based nature for two reasons. The first of these follows from the explanation given at the end of the previous section, in that under worst case scenarios a tree based routing scheme would perform similar to that of scheduled polling, while under better scenarios it would likely perform better than scheduled polling. It would therefore also follow the same characteristic curve as that in Figure 5.2, of scheduled polling, under a worst case scenario. Furthermore, it would then be that a tree based routing algorithm would in turn perform better than a contention based scheme such as CSMA.

In addition to the numeric motivation provided earlier, a deterministic tree routing algorithm would satisfy various requirements of a communications system, such as:

- The root would be the mother node that deterministically controls communication, therefore deterministic communication,

- Trees employ a multi-hop connection method which has been proved to be better than direct links [7],
- A tree algorithm is relatively easy to implement in various programming languages and would most likely reduce computational overhead due to its simplistic nature,
- Each node only knows of its parent and its direct children, but via this knowledge the root can communicate with any and all other nodes in the tree (and this in turn reduces storage and computational requirements),
- A tree allows for the optimisation of the number of hops in ideal circumstances. It also allows for more flexibility in terms of the amount of hops taken when a message is routed from some destination to some source. The number of hops can be adjusted by adjusting the number of children. This adjustment would be done to improve firstly the overall power efficiency as necessary. Secondly, as will be discussed in the next point, altering the amount of children allows for adjustment of the routing table dimensions as required by some specification or processing requirements,
- Furthermore, minimising the number of hops will reduce routing table complexity. This will then lead to less computational overhead as calculating destinations will not require traversing long arrays or handling a large number of hops,
- Loading a basic, non-complex client program on all nodes except the mother node will reduce complexity, power usage and overhead on each of the nodes. It will also reduce configuration complexity as each node will use the same client program,
- The mother node will then host the sole server program which will be slightly more complex. This is acceptable and furthermore, a stationary or base station might even be used as the mother node if required,
- Lastly, a tree structure is not dependent on the initial positioning of nodes as long as a communications link is present/can be established between the nodes. If the nodes were to be randomly deployed or if the nodes were

to move during operation, a tree could still be established and dynamically altered/updated as required as long as nodes stay within communications range of each other. Therefore, routing is more reliant on there being a communications path between nodes, than the specific positioning of nodes.

The true test of whether a tree based routing algorithm is effective will only be discovered once it is tested.

5.2.4. Transport Layer

In the transport layer we shall be taking a look at error handling as well as flow control. Without error handling/correction the routing algorithm cannot be realistically simulated or implemented. Additionally a noise algorithm will have to be introduced to simulate the noisy ocean environment. The noise algorithm is a simple implementation, resulting in an error probability of 1×10^{-2}.

It was calculated earlier that with the oceanic noise kept in mind, that a simulated approximate probability of error between 1×10^{-2} would be suitable for an underwater sensor network. The noise algorithm was implemented in Erlang as follows: for every bit that is sent over any hop, a random number generator with normal distribution generates a random number between 0 and 100.

However, before the start of every iteration the random number generator is seeded using the current CPU clock time, taken in microseconds, effectively randomising the number generator. This then allows for both the occurrence of uniformly distributed errors as well as burst errors, as each number generated is independent of the previously generated number and therefore it is possible for a series of errors to be generated.

Now, if this number generated is equal to one then the bit for which noise is currently being simulated is flipped or inverted, i.e. a one becomes a zero and vice versa. If however the generated number is larger than one, the bit is left in the same state. The same process is repeated for each bit until each bit of the entire message has had noise simulated.

There are various error handling and correction techniques, with some of them more applicable in this case than others. In this case, error detection and/or correction could be considered. Due to the noisy environment of the ocean coupled with the projected signal unreliability it was decided that both types would be implemented.

5.2.4.1. Error Detection

Error correction will be employed according to the methods discussed in the next section, but it is also necessary to implement some form of error detection. Since Erlang handles all communication through its own methods it is assumed that it also employs some form of error handling/error detection. Error handling increases the reliability of communications which will be beneficial in an environment such as the ocean.

It is however unknown what type of error handling and/or flow control Erlang employs and the only information that the Erlang OTP R10B (the release in which all simulations were coded) documentation provides is the following statement: “Message sending is asynchronous and safe, the message is guaranteed to eventually reach the recipient, provided that the recipient exists”.

This, therefore, does not provide any information on any possible error handling that Erlang employs when initiating communication between nodes. However, a relevant and realistic communications protocol will still have to be simulated for the purposes of this work. Therefore, to be able to simulate a complete communications protocol, some form of error detection or flow control will have to be implemented. Furthermore, implementing flow control provides some control over the influence of the simulated communications channel, as well as providing a method to determine the amount of retransmissions vs. successful transmissions required by measuring the amount of Ack's vs. Unack's implemented by the communications protocol during simulations.

Therefore, flow control will be implemented using an Ack/Unack scheme. The effect that Erlang might have still remains unknown, but at least the results of the simulated channel should reflect that of a real world implementation in that all relevant parts of the communications protocol, within the scope of this work, will be simulated, and furthermore, flow control can now be achieved with certainty, which should increase the performance of the intended communications protocol simulation.

However, it still has to be remembered that the underlying communication methods employed by Erlang might in all likelihood affect the results obtained from any simulations, more so in the case of the measured throughput. If possible these influences have to be quantified and measured either during the duration of this project, or at some later stage. Any anomalous values will have to be analysed with this factor in mind.

5.2.4.2. Error Correction

Making the decision on what type of forward error correction (FEC) to use is slightly more difficult, as there are various effective means, where each has its own upsides, downsides and operational costs. Three types of FEC which could possibly be employed for this work will be described in brief and then one will be chosen according to optimal performance in this case.

FEC methods vary in complexity and performance [13]. Methods that deliver high performance could be very difficult to implement and will cause high overhead, which in this case should be avoided. On the other hand, the simplest methods do not always perform that well in a very noisy environment. Therefore, an FEC method has to be chosen that has little/minimum overhead while at the same time providing a reasonable degree of error correction. For these purposes, three methods will be considered and briefly described.

5.2.4.2.1. Hamming Codes

Hamming codes are one of the simplest FEC codes that can be implemented. Every bit position that is a power of two (i.e. position 1, 2, 4, 8, etc.) is reserved as a check bit position. Combinations of these bits may represent any other bit position in the code. As an example, take position 15. 15 Can be reconstructed using 8 + 4 + 2 + 1. During encoding each check bit is set according to the parity of the bits corresponding to that check bit [13]. When decoding then takes place, a parity check is performed on the same sets that were involved in creating the bit, resulting in a syndrome value. The syndrome value indicates the position of the error. Unfortunately this method is only capable of correcting one error, which might not be sufficient with such a high error probability of 1×10^{-2}.

5.2.4.2.2. BCH Codes

On the other hand we have BCH codes that are binary, bit based, cyclic FEC codes. BCH codes make use of polynomials derived from finite fields to encode and decode data. Their performance can be calculated by using the following equations:

$$n = 2^m - 1 \qquad (5.13)$$

$$t \leq (n - k) / m \qquad (5.14)$$

BCH codes are capable of correcting *t* errors in a segment of *n* bits containing *k* bits of original data. BCH codes are sometimes hard to implement when available processing power is minimal. Although only the routing algorithm itself will be designed in this work, the FEC design still has to be simplistic enough to facilitate minimal processing power.

5.2.4.2.3. Reed-Muller Codes

Reed-Muller codes are also cyclic but are similar to Hamming in the sense that they use sectional message parity checks to overcome the effect of an error [14]. The ratio of original *k* data bits to the length *n* of the code is fixed and is capable of correcting *t* errors. The following equation expresses the performance of Reed-Muller codes:

$$n = 2^{k} - 1 \tag{5.15}$$

$$t \leq (2^{k-2} - 1) / m \tag{5.16}$$

Encoding for Reed-Muller is very simple [13], due to the fact that each row of the generator matrix is only incremented by two and it is not necessary to store the entire matrix in memory. Figure 5.9 presents an example of a generator matrix for a Reed-Muller code where $m = 3$. To encode a message using this generator matrix, we take a message, say A = 1011, and multiply it with the generator matrix as in the case of Figure 5.10.

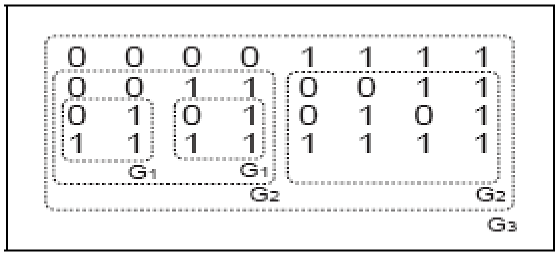

Figure 5.9 Reed-Muller generator matrix for m = 3

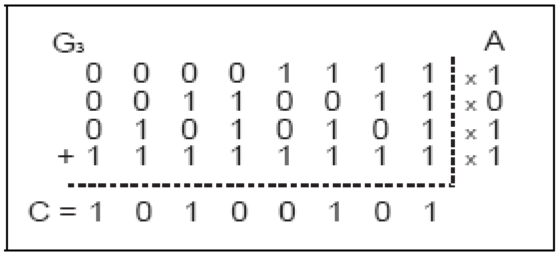

Figure 5.10 Reed-Muller generation of a codeword C from a message A

It is then found that the codeword that will be sent is C = 10100101. Decoding is also not too complex in that it is done with a series of XOR bit operations.

5.2.4.2.4. Reed-Solomon Codes

Reed-Solomon is a symbol based cyclic code. It has good performance but would be difficult to implement in a scenario where processing power is minimal. It requires solving a large number of simultaneous equations that are very time and processing intensive on the CPU. This in turn will lead to time delays and overhead which is undesirable.

5.2.4.2.5. Other Codes

Other methods that are more advanced and effective include convolution codes which are intended for long continuous streams of data and turbo codes, which are extremely powerful, but require a powerful CPU to process the calculations. Also in this case, long continuous streams of data are required and would not be logical to implement. Furthermore, such codes will result in high overhead and consequent time delays which are very undesirable [16].

5.2.4.2.6. Comparison of the different FEC codes

A.S. Labuschagne [13] did a comparison of the above codes using MSK modulation. The following parameters were used as test cases:

- Un-encoded data consisting of 128 bits
- Hamming encoding using a 7 bit codeword with 4 data bits, which is capable of correcting 1 bit error.
- BCH encoding using a 31 bit codeword with 6 data bits, which is capable of correcting 5 bit errors
- Reed-Muller encoding using a 32 bit codeword with 6 data bits, which is capable of correcting 7 bit errors.

The following graph (Figure 5.11) shows the results after extended testing [13]. It shows that Reed-Muller is the optimal solution and that an overall coding gain can be achieved, of approximately 6 dB, above that of no encoding, for an error probability of 1 x 10^{-2}. This coding gain will clearly vary with SNR and the resultant BER. The ocean environment's noise characteristic will most probably differ from that of over the ground point-to-point transmission. It is also not known whether this noise is constant or changing as a function of time.

It can be safely assumed that a small coding gain would increase the effectiveness of the communications protocol. It is also logical that the same relationship between each coding scheme would hold in most situations. Reed-Muller will therefore be chosen as the FEC scheme for the current work, due to the simplicity of implementation as well as its good performance.

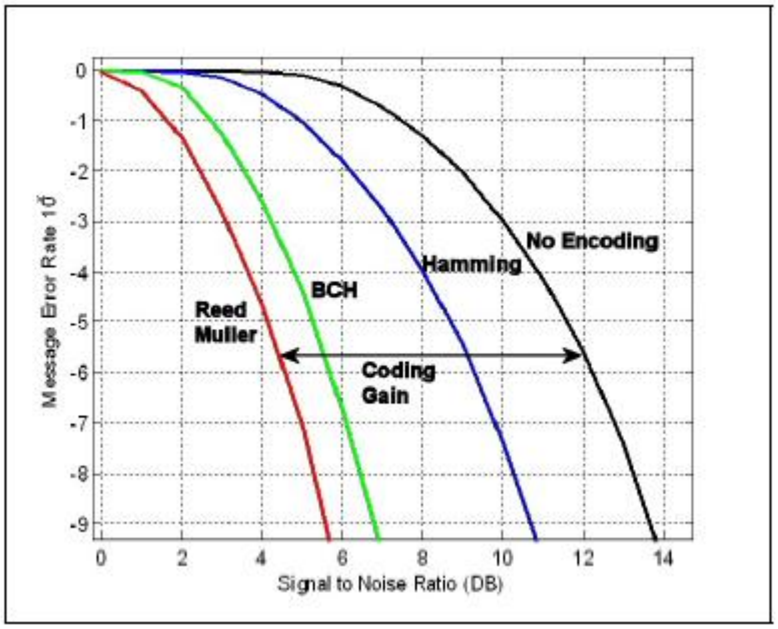

Figure 5.11 Comparison of FEC coding schemes [13]

Earlier the possibility of using either 4 or 5 bit packets was mentioned, but this will be decided by the choice of FEC. Therefore, a decision now has to be made between the following two Reed-Muller codes:

- A RM(1, 3) that takes a 4 bit data word and encodes it to an 8 bit codeword. It can correct one error.
- A RM(1, 4) that takes a 5 bit data word and encodes it to a 16 bit codeword. It can, however, correct three errors.

The first code will increase throughput while correcting one error with relatively low overhead, whereas the second one will have more overhead but has the advantage of correcting three errors, which in turn might reduce the total overhead. The requirement of the Reed-Muller FEC is to maximise the number of nodes discovered during route discovery and secondly to maximise the effective throughput. A simulation will be run to compare the results between RM(1, 3) and RM(1, 4), from which a decision can be made.

The following has to be kept in mind:

- The above sizes are chosen only as part of the simulation and can be altered as required,
- If a message to send contains more than the above number of bits, it can either be encoded using larger Reed-Muller codes or the message can be broken down into smaller data words. This will have to be balanced against increased overhead,
- Erlang sends packets using its own addressing scheme and therefore errors cannot be introduced into the addresses. If required some method will have to be implemented to simulate for noise introduced into the address of a packet.
- Although Erlang might employ its own flow control, it was decided to employ a flow control scheme comprising Ack's and Unack's to be able to exert some form of control over the simulated communications channel as well as to improve the effectiveness of the simulated routing algorithm.

5.3. Possible Solution Related to the ISO Model with Discussion, Analysis and Justification of Particular Choices

In the previous sections we looked at possible solutions and implementations for the simulation of a communications protocol. In brief we will now specify what methods were chosen as part of the implementations:

5.3.1. Physical Layer

Since the physical layer is beyond the scope of this work, we shall not be implementing any direct solutions.

5.3.2. Data Link Layer

After analysing and discussing the possibilities, as well as doing some in depth calculations, it was decided that a contentionless scheme would most probably fare best under the harsh ocean conditions.

5.3.3. Network Layer

For the network layer we chose a tree based implementation as this satisfied most of the requirements stipulated earlier on in this document. It also fits in with the contentionless choice chosen in the Data Link Layer. Furthermore, using a tree based scheme allows for flexibility as well as scalability.

5.3.4. Transport Layer

For the transport layer we chose firstly to implement a simple Ack/Unack system to facilitate flow control. Secondly, from the results of simulations it was found that the best choice for error correction would be a Reed-Muller based code. Therefore, utilising the above two choices, we have provided for both error correction as well as error detection and therefore, we have provide for error handling.

5.4. General Overview and Implementation of the Algorithm

A general overview of the proposed Protocol as well as a step-by-step explanation of the protocol will now be provided. The explanation will follow a single iteration of the protocol as it would run in a complete simulation. As will be shown and explained in Chapter 6, it was however decided to simulate each part separately i.e. to determine the optimal value of *n* as well as the effect of noise on *n*, to determine an optimal Reed-Muller code and finally to determine the throughput using a routing table, after routing has been completed.

Firstly, it has to be said that Erlang does not allow for true broadcasting. This in itself makes the simulation of routing difficult as some way has to be implemented to simulate broadcasting. Broadcasting is important as it provides some way for each node to determine which other nodes are in its vicinity by broadcasting a message and then seeing who replies to that message. This also then allows nodes to determine which other nodes are within communications distance as well as allowing the determination of the approximate distance between themselves.

A novel way had to be designed to overcome this restriction. To do so, the following method was employed: assume the mother node is pre-selected and knows how many nodes, *N*, in total do exist (offline or available). The node however does not know which other nodes are in its vicinity, which are available or what each nodes' initial position is. To simulate nodes within this mother nodes vicinity, a uniform random number generator will be employed to generate *n* random numbers, where:

$$0 < n <= \textit{max number of nodes } N \qquad (5.17)$$

Each of these *n* numbers represents some node within the total number of nodes *N*. These *n* nodes, which then correspond to the randomly generated numbers, are then assumed, or simulated, to be the *n* closest nodes. Therefore, a routing message is then sent to each of these nodes and these nodes will comprise the first level of the routing tree. After the mother node has sent messages to these nodes, it will place itself in a listening mode wherein it waits for these and subsequent nodes to reply, so it can build the routing table.

From there each of these first level nodes will repeat the process of generating *n* random numbers (and in essence nodes) which in turn are the simulated closest nodes to each of these nodes (excluding the respective nodes parent). Each first level node will then send a routing message to its own *n* nearest simulated nodes by using the generated numbers and will then enter a listen mode in much the same way as the mother node. However the first and subsequent level nodes only relay messages to the mother node and do not themselves create routing tables. Each node only has information about its parent and children. This process will then be repeated for each subsequent simulated discovered node.

This method of generating uniform random integers is useful for the following reasons:

- It overcomes the problem of no broadcast mode available in Erlang,
- It is not guaranteed that all *N* nodes will be simulated. Therefore it facilitates the simulation of lost nodes, where these lost nodes are represented by the numbers that are not generated,
- It is possible for a number to be generated more than once, which simulates the possibility of nodes being within communications distance of more than one other node, other than its parent,

As an example of how this will take place observe the following: Say 30 nodes exist. The mother node then generates two random numbers between 0 and 31, say 5 and 11. 5 and 11 are then the simulated closest nodes to the mother node and routing messages are then sent to 5 and 11 (Figure 5.12), where after the root enters a listening mode. 5 and 11 then each generates two random numbers and sends routing messages to those nodes before going into a listening mode (Figure 5.13) themselves.

This is repeated until each node has generated *n* random numbers and messages have been sent to these generated numbers. It is however not guaranteed that each node will reply to those messages or that each node will have *n* children. *n* is just the maximum number of children allowable per node.

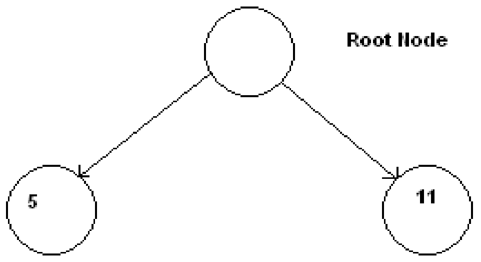

Figure 5.12 Nodes 5 and 11 generated and messages sent to each

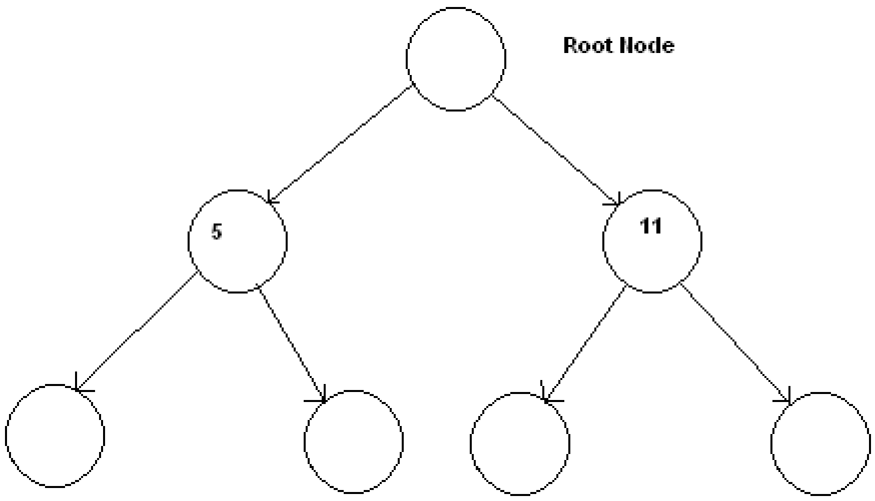

Figure 5.13 Nodes 5 and 11 send routing messages to their generated nodes

Again it is however not guaranteed that all 30 numbers/nodes will be generated due to the random number generation. However for higher values of *n*, the chance that all nodes will be discovered greatly increases, as will be shown in chapter 6.

Therefore, the numbers that have not been discovered will be considered to correspond to nodes that are either out of any communication range or correspond to nodes that have died. Furthermore it is possible that 5 and 11 may generate the same random number. For this reason the mother node checks all incoming routing information and if a node already exists in the routing table, say via 5, it will not be added again, say via 11. Therefore, a node can only be accessed via one path. This then simulates a random discovery tree as well as the possibility of lost nodes, or an environment where nodes could be lost.

Using simulations, *n* will be determined in the next chapter according to the criteria that will be set out. In other words binary, ternary, quaternary and 5-ary routing trees will be simulated and the most effective one selected as the value of *n*. Each routing tree will be run for some measure of time and its effectiveness at discovering all possible nodes (equation 5.18) will be measured. The efficiency at discovering all possible nodes will be measured by taking *N* nodes (i.e. 20) and simulating the routing algorithm. The average number of nodes generated, during each iteration, through the above explained method, will be measured, i.e. say each iteration simulates the discovery of 12 nodes on average per iteration. Therefore the average discovery efficiency will be:

Number of nodes discovered / total number of nodes * 100 (5.18)
= 12 / 20 * 100
= 60 % average discovery efficiency

This will be repeated for *n* = (2, 3, 4, 5) and the efficiency together with other performance parameters measured. The optimal value for *n* will be one with a high average efficiency while still minimising overhead and computation, as well as keeping the number of hops per node optimal, which in this case would be to keep the number of hops to a minimum per node while not attaching too many children to a single node. The average power usage per hop is thus reduced and high *n* values are not even considered as they will minimise the number of hops to an unrealistic value.

As an example let *n* = 10. After the first iteration 11 nodes will have been simulated and Level one will equal 10 nodes. On the second iteration 111 nodes will have been discovered and the second level will equal 100 nodes. Logically, it can be seen that this is not optimal, as to reach any node only one hop has to be taken, but this one hop could mean sending a message from a node on the far side of the swarm to a node on the completely opposite side of the swarm, thus requiring a large amount of signal power and increasing the probability of corruption by noise as well as complete packet loss. Therefore, *n* must be chosen as to maintain an optimal multi-hop (Figure 5.14) strategy, while maintaining high protocol effectiveness.

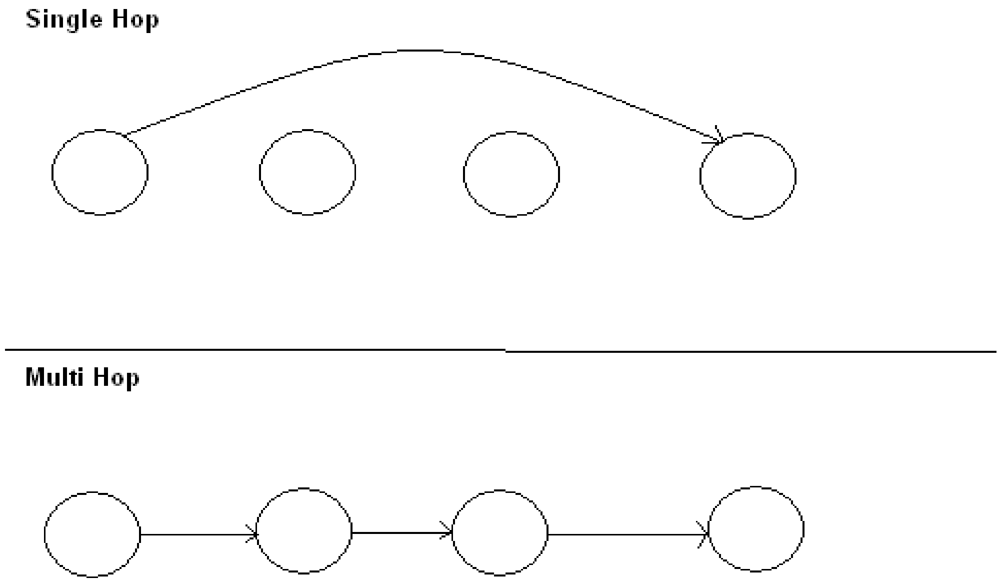

Figure 5.14 Multi hop vs. Single hop

The above simulations for the determination of *n* will be done for 3 scenarios to give a basis for comparison and to provide a better idea of the efficiency of using a tree based method:

- One where no noise is present,
- One where noise is present between hops,

- One where noise and some form of FEC are present.

The second simulation as well as the third simulation has a second goal in mind. Since it is impossible to introduce noise into the address portion of Erlang's messages, an address portion will be simulated in those two cases. This address portion will also be subject to noise and each address bit is subject to a P_e of 1×10^{-2}. This then allows for the possibility of packets not reaching their destination, or destination nodes discarding packets because the address portion is corrupted. This would then provide results that can be used to determine the effect that corrupted addresses would have on the communications protocol, as well as its efficiency at discovering nodes. This will then further help to determine whether using this routing method will be effective in the ocean environment and whether the noise characteristic of the ocean will affect the communications protocol in too detrimental a way.

After all routing messages have been sent, each node sends its own address to its parent with a routing received message and each parent then appends its own address to that message and in turn sends it to its own parent, in essence creating a routing path. Each node then enters a listening mode and sets a timer. When a routing received message is received, it again appends its own address and relays it to its own parent and resets the listening timer. This is repeated until the message reaches the mother node.

The mother node then takes the routing received message, extracts the routing path/list which was created by each node appending its own address and then places it in a routing table. This is done for every node that sends a routing received message, except in the case where a path to a node already exists. To now send data to a node, the node only has to be searched for in the first column of the table. Once it is found, the corresponding row is extracted and used as the routing path to the required destination.

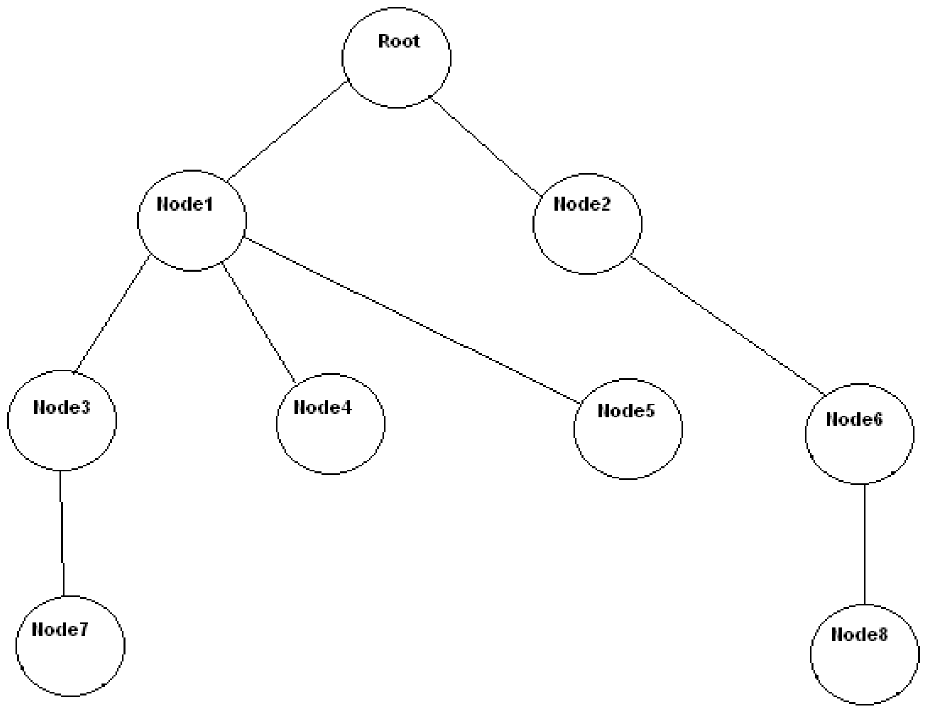

Figure 5.15 Example of a routing tree

Figure 5.15 serves as an example. To send data to Node8, simply look for it in the first column of the table (Table 5.5). When it is found, it can be seen that to send data from the Root to Node8, the first hop would be Node2, then Node6 and then finally Node6 will relay the message to the destination node, Node8.

Should the timer of a node expire however, the node enters a different listening function. This function has been designed to accept data requests and to send data to the root node. In essence, the node then enters a low power-consumption mode in which it waits for requests from the mother node.

Root			
Node1	Root		
Node2	Root		
Node3	Node1	Root	
Node4	Node1	Root	
Node5	Node1	Root	
Node6	Node2	Root	
Node7	Node3	Node1	Root
Node8	Node6	Node2	Root

Table 5.5 Routing table for Figure 5.15

After routing has been completed, messages can now be sent using the routing table. To simulate this, the server generates a random number, which is again representative of some node. The root or server node then sends a data packet to the selected node and waits for an Ack or a Unack. Before this however takes place, the packet has to be encoded using the chosen Reed-Muller FEC. This is done by calling a Reed-Muller algorithm which encodes the data bits into a coded packet. Subsequently the server determines the first hop of the multi-hop path and sends a message to the first hop, with the rest of the route and the encoded data as well as the destination node enclosed in the message.

On each hop noise addition has to be simulated. This is done by creating an extra noise node called 'noise'. Each message between nodes is relayed via this node, which then adds noise. The way this is implemented, is briefly set out in Section 5.6. Running simulations without any FEC found that this method gives a good approximation of real world noise and that the calculated BER is almost exactly the same as the theoretical BER (deviating with less than 1 %).

This is repeated for every hop in the communications path until the message reaches the final destination. Only the data to be sent is corrupted and not the destination or multi-hop node addresses, as Erlang requires viable addresses to function correctly and also Erlang hides all addressing as well as communication link implementations. This is acceptable as the number of noise susceptible bits in the data portion can be increased as required. Simulation of address corruption by noise was discussed earlier.

Each node receiving the message after the root node checks which node is the next hop or whether it is the destination node. If the current node is not the final destination it just relays the entire packet to the next hop. If it is however the final destination, it decodes the message using Reed-Muller decoding and displays that it has in fact received a message. This message is then checked (against the originally sent packet) to determine whether any errors were found and if these errors could be corrected. If they could be corrected an Ack will be sent to the root node via the same path as with which the message arrived. If however the message cannot be correctly decoded due to too many errors, an Unack will be sent to the root and the root will resend the message until it receives an Ack and a valid/correct message is received by the destination node.

A question that has to be addressed, is what happens when nodes die or are lost to some reason or other such as communications paths changing. For future work, some method could be included into the comms protocol to check how many of the originally discovered nodes are still reachable.

The above parameters and methods of operation present a possible solution for a communications protocol for an acoustic sonar swarm. The performance of the proposed algorithm and associated parameters will be further determined through proper simulation, as discussed in the following chapter. The results of these simulations will indicate if and what changes are required.

5.5 Conclusion

The implementation of the proposed communications protocol, based on the parameters as set out, appeared viable. A decision was made between an Ad-hoc and deterministic strategy and a tree based deterministic routing algorithm was finally chosen after due consideration. Furthermore, it was found that the implementation of FEC will be beneficial, with a Reed-Muller approach offering the best balance as well as the utilisation of flow control using a basic Ack/Unack scheme. A brief description of the proposed algorithm was also provided.

Chapter 6

Results of Simulations and Analysis of Results

6.1. Introduction

In this chapter we shall be simulating the proposed solution using Erlang. The communications protocol will first be broken down into some of its constituent parts, to better determine the viability of each part of the solution on its own.

From there however, we shall be doing a complete simulation off the communications protocol as a whole. After all simulations have been run, an analysis of the results will be made. Furthermore, we shall also be doing Erlang specific simulations to determine the effect that Erlang has on communication and furthermore to determine whether Erlang introduces any anomalies, if at all.

6.2. Simulations

6.2.1. FEC testing and simulation

To start with, a decision had to be made in terms of the specific Reed-Muller FEC implementation. The two main options that were considered for this implementation were:

- RM(1, 3) where a 4 bit data word is encoded to an 8 bit codeword with error correction of one,
- RM(1, 4) where a 5 bit data word is encoded to a 16 bit codeword with error correction of three.

However, it was thought prudent to also look at RM(1,7), RM(1,15), RM(1,31) as well as RM(1,63). This would give us an indication of the performance of higher order Reed-Muller encodings, as well as their feasibility in our underwater application. With the higher order RM encodings, we have to, however, remember that they will be much more complex than the lower order codes and therefore, not necessarily applicable for our use.

Therefore, one of the reasons behind using smaller packets is that Erlang does not allow for matrix calculations and arrays. It was therefore found that using larger Reed-Muller codes would make the implementation of the generator matrix G very complex and this would in turn increase computational overhead as well. Therefore the choice of using either 4 bit or 5 bit data packets to encode was satisfactory on this basis, but the effectiveness of RM(1, 3), RM(1, 4). RM(1,7), RM(1,15), RM(1,31) as well as RM(1,63), in reducing bit errors, will be shown by simulation.

The comparison was made on the basis of comparing the use of FEC with the non-use of FEC. Basically, we chose 6 different packet sizes (4, 5, 8, 16, 32, 64), while keeping the total amount of bits to send (1000), as well as the ACK size (4 bits), constant. Furthermore, we kept the error probability constant at $P_e = 0.01$, as we have done through the duration of this work. We then calculated various results for non-FEC as well as FEC, which included Packet Error Probability (P_n), number of packet retransmissions, number of extra bits sent, total number of bits sent and finally, the time taken to transmit the 1000 data bits. To calculate the last value, we kept the channel bit rate constant at 1200bps (bits per second). The results are tabulated in the following section

6.2.1.1. Simulation results of FEC simulations

Packet size (bits)	4	5	8	16
Data to send (bits)	10000	10000	10000	10000
Ack size (bits)	4	4	4	4
Bit rate (bps)	1200	1200	1200	1200
Error probability (Pe)	0.01	0.01	0.01	0.01
Without forward error correction:				
No of packets to send 10000 data bits	2500	2000	1250	625
Packet error probability (Pn) without FEC				
Pn = 1 - (1 - Pe) ^ n	3.940E-02	4.901E-02	7.726E-02	1.485E-01
Number of corrupted packets expected	99	98	97	93
Number of Packet retransmissions	99	98	97	93
Number of extra bits sent	394	490	773	1485
Total number of bits sent including Ack bits	20786	18882	16159	14357
Time taken to transmit 10000 data bits successfully (s)	1.732E+01	1.574E+01	1.347E+01	1.196E+01
With forward error correction:				
Reed - Muller encoding	RM(1,3)	RM(1,4)	RM(1,7)	RM(1,15)
Code word length	8	16	128	32768
Message Length	4	5	8	16
Error bits correctible	1	2	9	1092
No of packets to send 10000 data bits	1250	625	78.125	0.305175781
Packet error probability (Pn) with FEC				
Pn = (n x)*p^x*(1-p)^n-x	2.636E-03	4.914E-04	6.929E-07	1.420E-246
Number of corrupted packets expected	3.295	0.307	0.000	0.000
Number of Packet retransmissions	3.295	0.307	0.000	0.000
Number of extra bits sent	39.542	6.143	0.007	0.000
Total number of bits sent including Ack bits	1.503954E+04	1.250614E+04	1.031251E+04	1.000122E+04
Time taken to transmit 10000 data bits successfully (s)	1.253E+01	1.042E+01	8.594E+00	8.334E+00

Table 6.1 Comparison of Simulation results using differing RM orders, as well as differing packet sizes.

6.2.1.2. Analysis of FEC Simulation Results

From the above results we can obtain quite a bit of useful information. First off we can immediately see there is a gain in using FEC over not using FEC. In all cases, using FEC proved advantageous and provided a transmission gain. For example, in the case of the 8 bit packet, the transmission time was 13.4 seconds without FEC, whereas with the use of FEC this dropped by almost 5 seconds to 8.6 seconds. This is a considerable gain.

Secondly, we find that when not using FEC, there is a clear advantage in using bigger packets rather than smaller packets. We can clearly see a decline in transmission time as we move from 4 bits to 5 bits and so on. However, when we reach 16 bit packets we seem to hit a maxima and from there on further, we start to see the transmission time increase again. This can be attributed to the size of the packet. When we encounter an error in a 8 bit packet, we only have to resend 8 bits, however, when we encounter an error in a 64 bit packet, we have to resend 64 bits. Therefore, logically, at some stage this would unacceptably increase transmission time.

As in the above, we find a similar result when using FEC. However, in comparison to not using FEC we have a significant transmission gain. Furthermore, we find that in each case, as the bits are increased, we have an increase in transmission time. The reason for this is again the increased packet size.

Using the above results, we can clearly see that:

- FEC provides significant gains above non-use of FEC
- We have a minimum transmission time, in the case of FEC, when we use RM(1,15)

- For each higher order of RM, the code word length significantly increases and so also the complexity of the generator matrix G, as discussed in Chapter 5

Therefore, using the above points, as well as what was discussed in Chapter 5. Taking into consideration the complexity to generate RM(1,15) codes, they are obviously not ideal for our use. However, RM(1,4) and RM(1,7) still provided significant gains over the non-use of FEC. Implementing both codes, it was found that using RM(1,4) above RM(1,7), reduced coding complexity, as well as implementation complexity, by a significant amount.

Therefore, we decided to rather use the RM(1,4) encoding for our purposes.

6.2.2. Determining the Value of *n*

The next simulation will be to determine the value of *n* to be used for the routing tree. To do this the tree routing algorithm will run for a number of values of *n* (1,…,6). In this test, all simulations were run only in terms of routing and not to send any data. The testing was done with 20 nodes and was run for at least 6 hours. The simulations were broken up into three stages. The first was to determine a value of *n* under ideal noiseless circumstances and the results analysed. From there, two additional simulations were added, one where noise was introduced between each hop and finally one where noise as well as some form of FEC was employed. The results of the second and third simulations were then compared and discussed.

6.2.2.1. Efficiency of Differing Values of *n* in the Absence of Noise

This part of the simulation was done in the following manner:

- The mother node and each following node used the routing method as described in the previous chapter.
- Each node logs the node from which it received the routing message, as well as to which node it sends a routing message.
- Each time a node receives a routing message, it sends a message to the mother node as well and the server increments a counter with each message. The mother node thus keeps track of the number of routing iterations as well as the number of routing messages received by each node.

Thus each node has a value corresponding to the number of routing commands it has received, from whom and how many routing messages it has sent. The mother node has values corresponding to the number of routing iterations, as well as how many routing commands each node has received. The server values will be used to determine the efficiency as defined and each node logs its own data to correlate the data received by the server/root node. The values will then be used to determine the efficiency of the varying values of *n*, which is given by the following formula:

Efficiency
= ((Number of nodes found) / (Number of nodes to be found)) * 100 (6.1)

where,

Number of nodes to be found
= (((Number of routing messages sent) / *n*) * 20) (6.2)

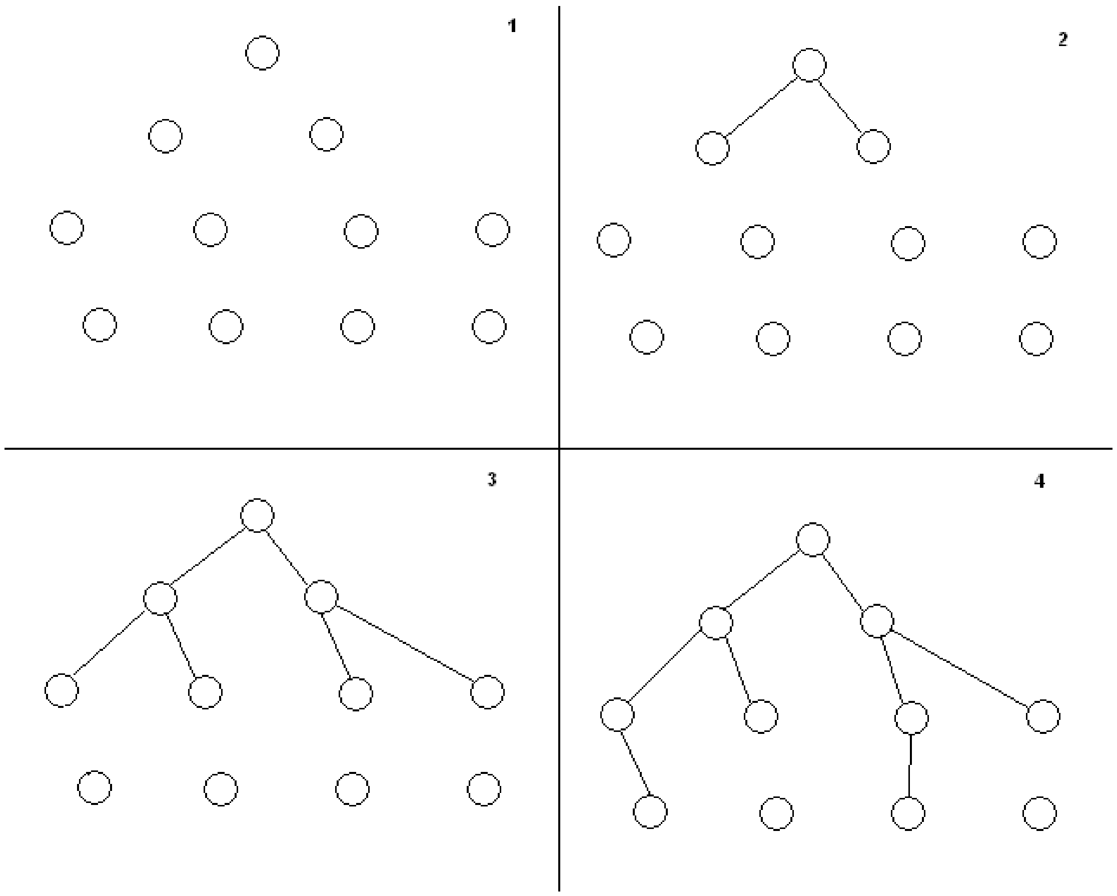

Figure 6.1 From Block 1 to 4, the above Figure shows a routing example where *n* = 2. Each iteration adds more nodes to the tree until at block 4 we have a completed tree. Routing does not guarantee that all nodes will be discovered

6.2.2.2. *n* = 1

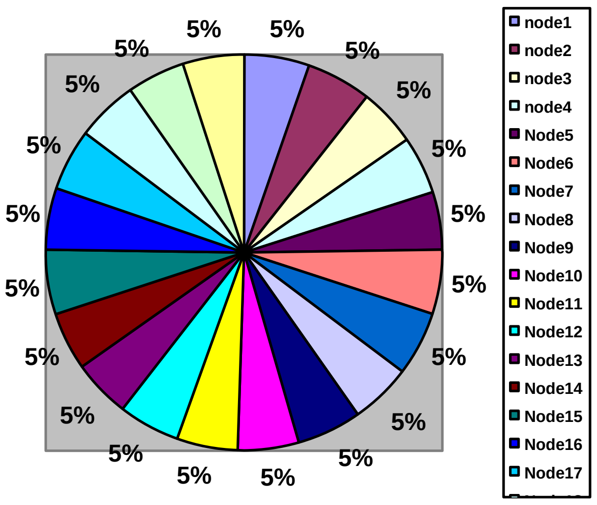

Figure 6.2 Percentage of routing messages per node. Each node receives the routing message with equal probability during each routing operation

Number of Routing messages sent = 1369

Number of nodes to be found = ((1369 / 1) * 20) = 27380

Number of Nodes found = 7276

Efficiency = (7276 / 27380) * 100 = 26.57 %

6.2.2.3. *n* = 2

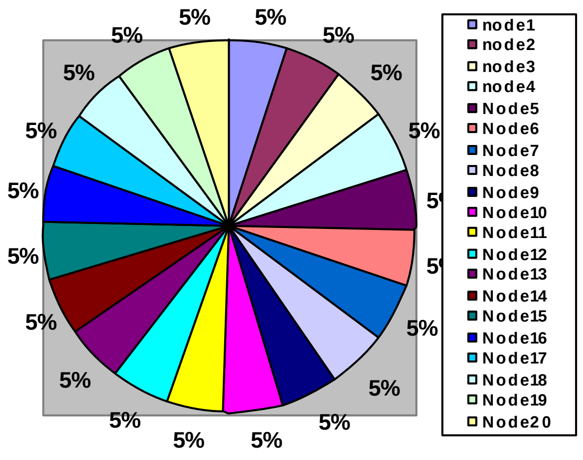

Figure 6.3 Percentage of routing messages per node. It can be clearly seen that each node receives the routing message with equal probability during each routing operation

Number of Routing messages sent = 3404

Number of nodes to be found = ((3404 / 2) * 20) = 34040

Number of Nodes found = 27384

Efficiency = (27384 / 34040) * 100 = 80.44 %

6.2.2.4. *n* = 3

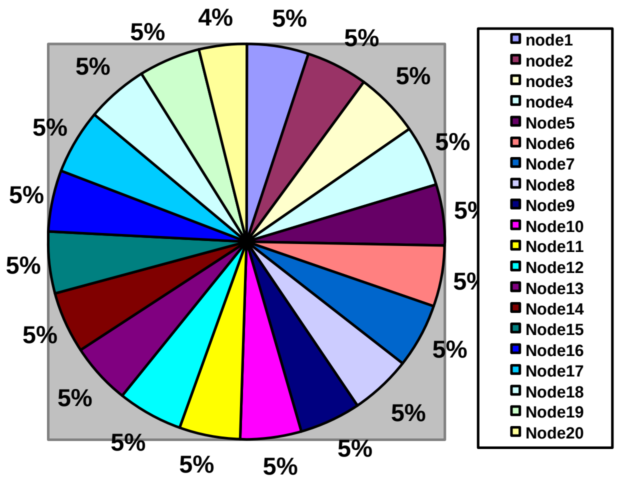

Figure 6.4 Percentage of routing messages per node. Each node receives the routing message with approximate equal probability during each routing operation. Only node 3 had 1% less. This is a small deviation and thus not a problem

Number of Routing messages sent = 10563

Number of nodes to be found = ((10563 / 3) * 20) = 70420

Number of Nodes found = 63944

Efficiency = (63944 / 70420) * 100 = 90.80 %

6.2.2.5. *n* = 4

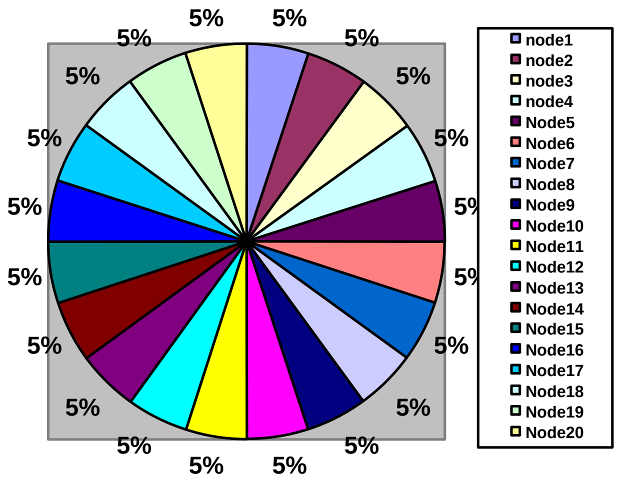

Figure 6.5 Percentage of routing messages per node. Each node receives the routing message with equal probability during each routing operation

Number of Routing messages sent = 9080

Number of nodes to be found = ((9080 / 4) * 20) = 45400

Number of Nodes found = 44423

Efficiency = (44423 / 45400) * 100 = 97.84 %

6.2.2.6. *n* = 5

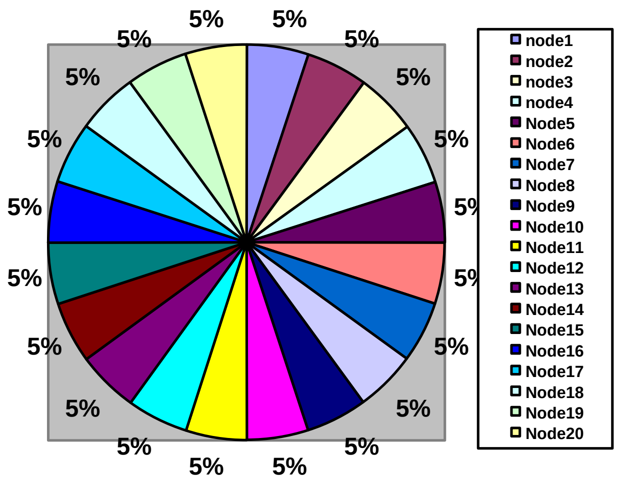

Figure 6.6 Percentage of routing messages per node. Each node receives the routing message with equal probability during each routing operation

Number of Routing messages sent = 13480

Number of nodes to be found = ((13480 / 5) * 20) = 53920

Number of Nodes found = 53549

Efficiency = (53549 / 53920) * 100 = 99.31 %

6.2.2.7. *n* = 6

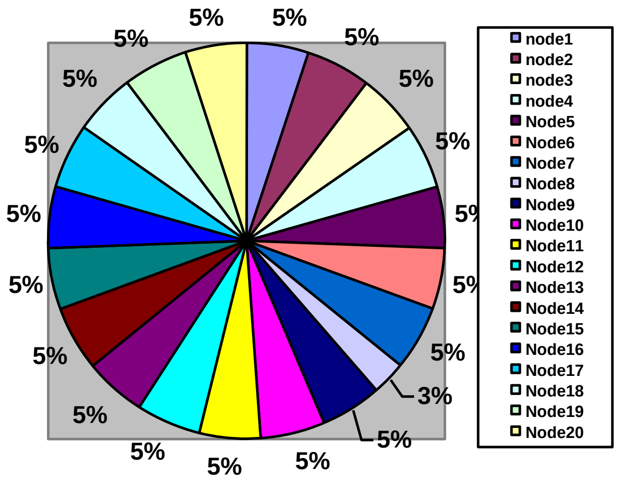

Figure 6.7 Percentage of routing messages per node. Each node receives the routing message with equal probability during each routing operation. The only exception in this case is node8, which in this simulation was only activated at a later stage

Number of Routing messages sent = 9318
Number of nodes to be found = ((9318 / 6) * 20) = 31060
Number of Nodes found = 30091
Efficiency = (30091 / 31060) * 100 = 96.88 %

6.2.2.8. Analysis of Results for Determining *n*

From the above it can be clearly seen that as the size of *n* increases, so does the efficiency of the routing algorithm, as far as node discovery is concerned. Between 1 and 2 there is a huge leap in efficiency. Between 2 and 3 that jump decreases significantly and from there onwards, it becomes reasonably stable. The result for *n* = 6 is slightly different from what is expected as it drops by 3 % compared to *n* = 5. It was however thought with increasing values of *n* that the efficiency would constantly increase in accordance with some relationship. The fact that this is not so could be attributed to internal computer factors (delays, interrupts, increased computational overhead.). The effect that Erlang has on the routing protocol also cannot be quantified and therefore, the possible role Erlang has to play in this drop is uncertain. However the value is still above 95 %, as is the case with *n* = 4 and *n* = 5. This is still more than acceptable and does not necessarily indicate any possible problems when using values of *n* ≥ 6.

Furthermore, it can be seen from the pie charts for each value of *n*, that each node receives the routing message with equal probability, whether it is from the server node or from another node via the generated tree. It is therefore clear that the routing algorithm does not favour any nodes and that depending on the initial placement of the sonar swarm in its application environment, routing would take place fairly and evenly.

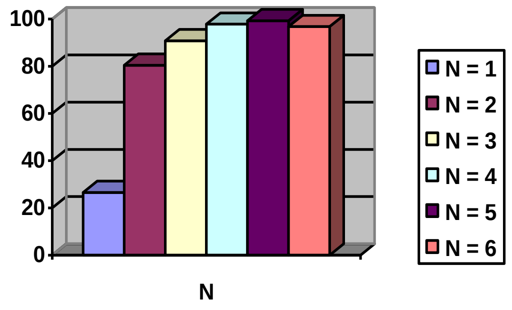

Figure 6.8 Efficiency for differing values of *n*

Figure 6.8 indicates that from *n* = 2 onwards we already have very good efficiency. A selection must be made that will result in the least overhead while maintaining a routing table with reasonable dimensions.

This question might best be answered by inspecting Table 6.2 and Figure 6.9:

	n = 1	*n* = 2	*n* = 3	*n* = 4	*n* = 5	*n* = 6
Depth = 1	2	3	4	5	6	7
Depth = 2	3	7	13	21	31	43
Depth = 3	4	15	40	85	156	259
Depth = 4	5	31	121	341	781	1555
Depth = 5	6	63	364	1365	3906	9331
Depth = 6	7	127	1093	5461	19531	55987

Table 6.2 the maximum number of nodes per depth for each value of *n*

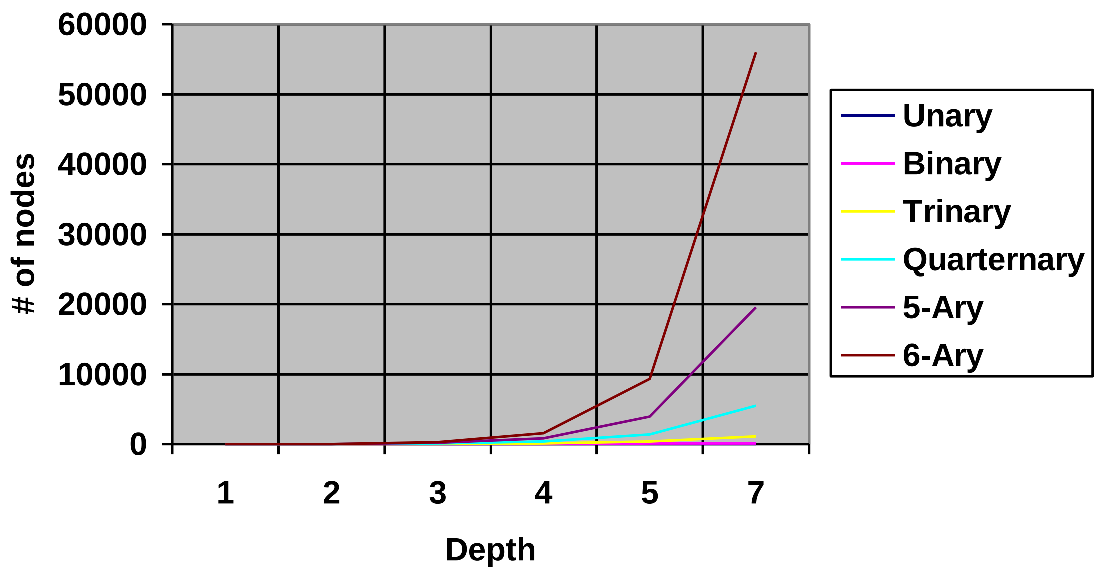

Figure 6.9 Graph depicting maximum discoverable number of nodes per depth for values of *n*. It can clearly be seen that for higher values of *n* the graph increases rapidly

Table 6.2 and Figure 6.9 indicate that as *n* increases, so the depth of the tree decreases. This directly translates to less hops and an altered routing table. However, before a final decision can be made, the format of the routing table will have to be considered. The format of the routing table is directly affected by the value of *n* and therefore, the maximum number of hops. The concepts of routing overhead and routing complexity are illustrated by the routing table in Table 6.3.

The first node represents the root of the tree and each following row represents a hop away from the root. For example, Node 11 is 2 hops away from the root and it has 5 as its parent. Node 17 is three hops away and has Node 23 as its immediate parent. Node 23 has Node 7 as its parent. If the root wants to send a message to Node 17, it first retrieves the routing table and searches through the first column until it finds Node 17. It then extracts the row into an array and checks the first hop, which is Node 7. The rest of the array is then sent to the next hop which is Node 23, with the routing message and the final destination. From Node 23 the message is then sent to node 17 which is the final destination.

Third Hop	Second Hop	First Hop	Start
15	11	5	Root
30	11	5	Root
3	11	5	Root
27	26	5	Root
1	26	5	Root
22	26	5	Root
17	23	7	Root
25	18	10	Root
2	28	10	Root
16	28	10	Root
19	29	10	Root
20	29	10	Root
13	29	10	Root

Table 6.3. An example routing table. For this example *n* = 3 and 30 nodes were in the swarm. 22 of the 30 were found with an efficiency of 73 %

Each consecutive node then uses the array and destination information to route the message to its final destination. The more hops and thus columns are involved, the better power efficiency we have [7]. This in turn increases the size of the routing array with an increase in overhead. On the other hand, increasing *n* gives a higher efficiency but decreases the number of hops as well as power efficiency. A suitable mean has to be found. In this case a decision was made to limit the theoretical implementation of a swarm to a maximum of 100 - 150 nodes.

Taking this into consideration, as well as Table 6.2 and Figure 6.9, the values that offer a suitable mean are *n* = 3, 4 or 5. *n* = 5 has the highest efficiency (99%) and will typically offer a minimum number of hops between 3 to 4 (but probably 3), while on the other side of the spectrum, *n* = 3 provides a lower but still high efficiency (90%) and typically offers a maximum of 4 to 5 hops. This leaves us with *n* = 4 which appears to offer a very good, if not optimal, mean: it has a very high efficiency of 97% and it offers a maximum number of hops of 3 to 4 (most probably 4).

The above indicates that a routing table will not exceed 5 columns (Table 6.3 consists of four) and a message will have to traverse at most 4 hops to reach a final destination. This means routing as well as processor complexity is reduced. Furthermore, using 4 hops allows for a good efficiency in terms of power usage as increasing hops also increases power efficiency. Based on the above, the rest of the simulations will take place using *n* = 4. However, before moving onto any other simulations, the effect that noise has on efficiency has to be analyzed. This will now be done in the next two sections.

6.2.2.9 Efficiency of Differing Values of *n* in the Presence of Noise

From the above it can be seen that a value of *n* = 4 would be a very good and applicable choice. In this section however, the same simulations were done to determine the efficiency for *n* = {2,3,4,5}, however they were now done under noisy conditions (BER = 1 x 10^{-2}) with no FEC employed. *n* = {1,6} were omitted as their simulation data would be redundant as with *n* = 1 the efficiency was very low to start with and with *n* = 6 it was very high and *n* = 6 would most likely not differ much from the results of *n* = 5.

The simulations were run in exactly the same manner as before, however in this case, a 5 bit address header was simulated. Therefore the entire node discovery process employed a simulated address between hops. If this address part could not be properly decoded, then the node receiving that message could not know from whom that packet originated and the packet would be discarded. The result of this was that routing messages might not arrive at their intended destinations and also nodes might not be added to the routing table, as their respective messages would not reach the mother node. To determine the full effect that noise would have, no methods for retransmission were employed.

	n = 2	*n* = 3	*n* = 4	*n* = 5
Number of Iterations	2854	2017	3178	2407
Number of nodes to be found in Total	57080	40340	63560	48140
Number of nodes found	35902	31832	54149	42082
Efficiency (Equation 6.1)	62.897 %	78.909 %	85.193 %	87.415 %

Table 6.4 Efficiency of differing values of *n* under the influence of noise

It can immediately be seen that noise affects the efficiency of the routing tree (Table 6.4). There is a substantial drop in the percentage of nodes found. This is directly linked to messages that were affected by noise and could thus not be correctly interpreted by the receiving node in each case. However, for *n* = 3, an efficiency of 78 % was still achieved and for *n* = 4 an efficiency of 85 % was achieved which is very good and would in most circumstances be more than acceptable. For *n* = 5 an efficiency of 87 % was achieved, however, as discussed in Chapter 6.2.2.8., a value of *n* = 5 would not be optimal so a choice of *n* = 4 would still be the first choice.

In the next section, noise was again added. However in this scenario, the previously simulated RM(1, 4) encoding was employed to counteract the influence and effects of noise. This will then provide insight firstly into the effectiveness of the tree based routing algorithm under the influence of noise while FEC is employed, but also what role FEC will play in the routing.

6.2.2.10. Further simulations on the value of n when noise is present and when FEC is employed

In this section, as in the previous, noise was added on each hop. However FEC was now employed using a RM(1, 4) code. Again *n* = {1,6} were omitted as in the previous section. The results were then logged and compared to determine the efficiency of the routing algorithm under noisy conditions, but with FEC employed.

	n = 2	*n* = 3	*n* = 4	*n* = 5
Number of Iterations	3320	2538	2051	2747
Number of nodes to be found in Total	66400	50760	41020	54940
Number of nodes found	52163	46617	35081	53552
Efficiency (Equation 6.1)	78.558 %	91.838 %	85.521 %	97.473 %

Table 6.5 Efficiency of differing values of *n* under the influence of noise with a RM(1, 4) FEC employed

As in the above case, a 5 bit address portion was simulated. However this 5 bit address portion was encoded using RM(1, 4) and decoded at the destination using the RM(1, 4) decoding matrix. Again, if the 5 bit address portion could not be correctly decoded, the packet/routing message was discarded. Furthermore, no methods for retransmission were employed to determine the effect of noise and the use of FEC on discovering nodes.

Employing FEC in the presence of noise provided a significant improvement in efficiency from Table 6.4, as can be seen in Table 6.5. This then supports the use of FEC in a noisy environment such as the ocean, as in most of the cases (except for *n* = 4) there was an improvement of at least 10 %. In large scale implementations this can span over for hundreds of nodes. Furthermore, it also shows that with noise detection and prevention methods employed, that the routing algorithm remains effective in discovering a high percentage of nodes. At this stage it has to be remembered that no methods for retransmission were employed as to determine the full effect that noise has on routing and on routing messages. However, retransmission methods would also increase the efficiency to some degree.

Lastly, although there was no significant improvement between Table 6.4 and Table 6.5 for *n* = 4, this value still provides a high efficiency (> 85 %), while providing for an optimal amount of hops versus routing table dimensions as discussed in Chapter 6.3.1. However, the fact that *n* = 3 performs better than *n* = 4 is very anomalous. It is uncertain why this drop of 6% occurred, but it could be attributed to the way that Erlang handles communication, as well as the fact that it is unknown how Erlang is influenced by processor overhead as well as interrupts from external sources such as Windows.

Using averages, a value of approximately 94 % would have been in line with expected values, if the upward trend of *n* = 2, *n* = 3 and *n* = 5 were to be followed. However, as will be shown in chapter 6.5, it was found that Erlang does introduce a random variance into the measured results. Taking those results into consideration here, it was found that Erlang introduces a variance of anywhere between 17 % and 54 % above or below the expected average value. Therefore, a drop of 6 % between *n* = 3 and *n* = 4 and thus a negative variance of 9.57 % with respect to the expected value, is much lower than that calculated in chapter 6.5. It is therefore within limits of the variance that Erlang introduces into the measured results.

Having shown the efficiency of a tree based routing algorithm in this section, a complete simulation of the communications protocol will be done in the next section. This will be done to determine the throughput attainable as well as to determine the ability of the communications protocol to handle channel traffic as well as to relay messages as required.

6.2.3. Complete Simulation of the Communications Protocol

In this section a simulation will be done, using the designed communications protocol as a whole, with each part functioning as part of a greater whole.

6.2.3.1. Initial Tests

To make sure realistic results are achieved, the simulations were done with three scenarios, i.e. for *N* equals 10, 20 and 40 nodes respectively. Noise was introduced into the messages to result in a $P_e = 1 \times 10^{-2}$ using the method described earlier. Each node was simulated using an instance of Erlang for each node and running the appropriate client software on that node, whereas one instance of Erlang will be used on the mother node to run the server side software. Ack's were implemented and all relevant results logged for analysis. The difference between the groups lay in the number of nodes per simulation.

The idea behind using three scenarios where the nodes differ, is scalability. It will help to prove the effectiveness of the protocol in relaying messages as well as determining the effective communications throughput. Using three different values of *N* will provide an indication of the adaptability of the comms protocol to deliver messages and control communication under changing implementation sizes. It will also offer a better evaluation of the protocol.

The main focus of the analysis will be the values logged by the server in each case. The values logged by the nodes will be used as correlation values. Table 6.6, 6.7 and 6.8 show the base results obtained from the three simulations.

	Bits Sent / Messages Sent (Un-encoded)	Bits Sent / Messages Sent (Encoded)	Ack's received	Ack bits received (Un-encoded)	Ack bits received (Encoded)	Un-Ack's received	Throughput	Failed Ack's
Server	173010 / 34602	553632 / 34602	34590	173010	553632	11	90 bits/second	1

Table 6.6 Results logged by the server for the simulation of 10 quaternary (*n* = 4) nodes

	Bits Sent / Messages Sent (Un-encoded)	Bits Sent / Messages Sent (Encoded)	Ack's received	Ack bits received (Un-encoded)	Ack bits received (Encoded)	Un-Ack's received	Throughput	Failed Ack's
Server	104345 / 20869	333904 / 20869	20852	104345	333904	17	18 bits/second	0

Table 6.7 Results logged by the server for the simulation of 20 quaternary (*n* = 4) nodes

All three simulations ran for an extended time period of between 6 and 9 hours. During this time between 20000 and 35000 messages were sent by each server. The results obtained are therefore spread over a reasonable number of messages.

	Bits Sent / Messages Sent (Un-encoded)	Bits Sent / Messages Sent (Encoded)	Ack's received	Ack bits received (Un-encoded)	Ack bits received (Encoded)	Un-Ack's received	Throughput	Failed Ack's
Server	170065 / 34013	544208 / 34013	33968	170065	544208	45	29 bits/second	0

Table 6.8 Results logged by the server for the simulation of 40 quaternary (*n* = 4) nodes

6.2.3.2. Analysis of Initial Tests

Comparing the number of messages sent to the number of Ack's received, it is found that the number of messages received correctly by the nodes, is almost equal to the number of messages originally sent. In the 10 nodes simulation, we find that only 11 retransmissions had to be made out of 34602 sent messages. In the 20 nodes simulation, this increased, but only slightly, to 17 retransmissions out of 20869 sent messages. Lastly, in the 40 nodes simulation, the value again increased to 45 encoded data words out of 34013, but again this translates to a retransmission rate of only 0.1323 %.

The retransmission rate of 0.1323 % is very low and quite acceptable if taken into consideration that 40 nodes were used in the testing and that the maximum number of hops between points would be approximately 4. Furthermore, the increase in retransmissions is not in any way unexpected, as in each case the number of nodes was doubled. Therefore, the number of errors would logically increase and therefore, also the retransmission rate. The retransmission rate almost doubles as the number of nodes double. These results show that an approximate retransmission rate could be calculated successfully for larger sonar swarms. The retransmission rate would still remain relatively small.

Lastly, out of all the messages that were sent, only one Ack was found to be non-decodable. From this it can be assumed that almost all messages were relayed to their intended destinations. The number of un-decodable Ack's would most probably increase with an increase in the simulation period, but it is fair to assume that the % would remain reasonably constant and acceptably small. It is clear, by using a simple ACK based flow control to determine reliability, that FEC greatly decreases the number of retransmissions required.

6.2.3.3. Further Trials

From the above it can be seen that a tree routing algorithm works well when routing messages from some destination to some source. However, a problem was found in that the throughput for the three simulations differed and varied too greatly. The throughput does not seem to vary with some form of correlation i.e. steady/linear/some other form of decline. We find a throughput of 90 bits/second with the 10 node scenario, then a sudden drop to 18 bits/second with the 20 node scenario and finally a small increase to 29 bits/second with the 40 node scenario.

After reviewing these results the three simulations were run again, but this provided no further insight into the problem as the 10 node scenario remained the same, the 20 node scenario increased to 24 bits/second and the 40 node scenario also slightly increased to 31 bits/second, but these results are still too varied. To try and effect a change and obtain better results, a change to the code was made and secondly, controlling data was generated via a benchmark simulation.

The second part was effected by creating a single client/server scenario, as before, wherein data will be sent from a server to a single node and the throughput of the data transfer will be measured. The simulations were again run for an extended period to make sure the data was realistic.

The data was then taken and the throughput calculated, and this throughput measured against the throughput of the communications protocol. This was done to have some form of benchmarking values and to determine the theoretical maximum throughput.

The first part was effected in the following manner: During the initial simulations an 800ms delay was introduced to facilitate the monitoring of the algorithm while it executed. This 800 ms delay was removed, but this still did not explain the random values, and therefore all output to the screen was also inhibited. Furthermore, small code changes were also made to streamline the protocol without altering the functionality in any way. The first two changes were made to remove any delays that are not directly linked to the operation of the algorithm. This would be better explained by taking into consideration that each time data is written to the screen an interrupt has to be generated, introducing delay and consequently throughput delay not associated with the protocol itself.

6.2.3.4. Implementing and Testing the New Changes

After the above changes were made, the simulations were run again. At least three iterations each of the 10, 20 and 40 node scenario were run and the data logged as before. The single server client simulation was also run. The results obtained are given in Table 6.9 through Table 6.12.

It can be seen that the throughput of the two simulations, whose data will be used as the correlation data for the theoretical maximum throughput, stayed reasonably the same during each simulation. Therefore, the expected maximum throughput of a tree with only one node, other than root, would be about 645 bits/second. However, once more nodes and thus multiple hops are introduced, this throughput should start to decrease as the swarm increases with size.

	Simulation Duration	Original Data Bits Sent	Throughput
Iteration 1	24931 seconds. 6.925 hours	16091880	645.456 bits/second
Iteration 2	16819 seconds. 4.671 hours	10858450	645.606 bits/second

Table 6.9 Results obtained from the correlation simulations

The reasoning for this is twofold: firstly, as the number of nodes increase, so too does the number of hops and therefore the transmission time is increased.

Secondly, node paths have to be allocated from the routing table and routing table access time and route extrapolation time will come into effect. This adds to the computational overhead and also decreases the maximum possible throughput. To determine what effect these factors have, the results obtained from the simulations will have to be analyzed. The following three tables contain the data collected by the server node during the three iterations per simulation.

	Bits Sent / Messages Sent (Un-encoded)	Bits Sent / Messages Sent (Encoded)	Ack's received	Ack bits received (Un-encoded)	Ack bits received (Encoded)	Un-Ack's received	Failed Ack's	Simulation Duration	Throughput
Server Iteration 1	2061575	6597040	412247	2061575	6597040	61	7	15192 seconds	135.701 bits/second
Server Iteration 2	2636100	8435520	527071	2636100	8435520	144	5	13481 seconds	195.541 bits/second
Server Iteration 3	2002425	6407760	400415	2002425	6407760	64	6	10444 seconds	191.729 bits/second

Table 6.10 Results obtained from new simulations, after streamlining, for 10 nodes

	Bits Sent / Messages Sent (Un-encoded)	Bits Sent / Messages Sent (Encoded)	Ack's received	Ack bits received (Un-encoded)	Ack bits received (Encoded)	Un-Ack's received	Failed Ack's	Simulation Duration	Throughput
Server Iteration 1	15385	49232	3075	15385	49232	2	0	81 seconds	189.938 bits/second
Server Iteration 2	79260	253632	15846	79255	253632	4	1	401 seconds	197.655 bits/second
Server Iteration 3	4505120	14416384	900603	4505120	14416384	409	12	27659 seconds	162.880 bits/second
Server Iteration 4	2139850	6847520	427790	2139850	6847520	177	3	13196 seconds	162.158 bits/second

Table 6.11 Results obtained from new simulations, after streamlining, for 20 nodes

	Bits Sent / Messages Sent (Un-encoded)	Bits Sent / Messages Sent (Encoded)	Ack's received	Ack bits received (Un-encoded)	Ack bits received (Encoded)	Un-Ack's received	Failed Ack's	Simulation Duration	Throughput
Server Iteration 1	991375	3172400	198121	991375	3172400	148	6	10679 seconds	92.834 bits/second
Server Iteration 2	5701025	18243280	1138411	5701025	18243280	1776	18	31473 seconds	181.140 bits/second
Server Iteration 3	4650030	14880096	929004	4650030	14880096	986	16	26362 seconds	176.491 bits/second

Table 6.12 Results obtained from new simulations, after streamlining, for 40 nodes

6.3. Evaluation of Results

Comments on the above mentioned results are as follows:

- To start with, from the above results it can be seen that on average the throughput lies between 160 bits per second and 190 bits per second. There are two highly anomalous values of 135 b/s (Table 6.10 Row 1) and 92 b/s (Table 6.12 Row 1), while the rest of the values stayed between 160 b/s and 190 b/s with some variance. This variance is a concern and although variance was expected between the differing values of *N*, such a large variance between iterations was not expected.

- A variance of 30 b/s between the highest and lowest values (excluding the anomalous values) may seem large, but the environment in which the simulations took place as well as other uncontrollable factors, must be taken into consideration. These factors include interrupt delays, processor bottlenecks, memory access times as well as various other hardware and software related factors. Taking this into consideration a variance between iterations can be expected, but not to such an extent of the two values of 135 b/s and 92 b/s. The great variance between the steady-state and anomalous values is of concern. The difficulty in making a quantified judgement on these values is highlighted as follows:

- The reason for the above variance is unknown but can be attributed to Erlang. The reason for this is that the processing overhead incurred by Erlang is unknown and currently unquantifiable, and not obvious. Furthermore, since Erlang handles all communication, it is unknown what method of error correction, or error handling for that matter, Erlang employs or what the baud rate is set to, or whether Erlang employs fixed/variable length packets to transmit messages. This in itself would incur overheads and severely affect the throughput. Lastly, it is unknown how the simulation system environment would affect Erlang itself.

- Ignoring the two highly anomalous values, as well as the variance for each value of *N* for now, a further question is why there isn't a steady decline in the throughput starting from the 10 node scenario and decreasing to the 40 node scenario. This can be explained by taking into consideration the maximum number of hops required to reach a certain node. From the previous chapter we use the result of the quaternary (n = 4) case, where the maximum number of hops was approximately 5 and the average approximately 3. Taking this, we can make the assumption that irrespective of 10, 20 or 40 nodes, the average number of hops will remain 3. If the average number of hops remains the same then, therefore, the average throughput will also stay the same, which explains why the throughput between the three cases does not vary greatly.

- It was found that the maximum theoretical throughput is 645 b/s, but it was also found that the average throughput of the algorithm was at least two thirds less than this value. This decrease was, however, expected for multiple reasons:

  1. Firstly the increase of the number of hops from maximum one (benchmark values) to maximum six (Simulations). This would decrease the throughput as each hop would introduce a delay and with resultant decrease in throughput.
  2. Secondly, in the correlation simulation no routing was introduced as it was not necessary, but accessing the routing table and finding the route would also decrease the throughput, as is to be expected.
  3. As stated before, Erlang provides no knowledge of its inner workings. Therefore, Erlang might very well affect the throughput in some unknown way. This could easily compensate for the loss of throughput

encountered and therefore, could be the main cause of this sudden drop. At this stage, the question is once more presented whether Erlang would really be suitable for this type of simulation?

- From the above results the RM(1, 4) FEC that was chosen performed as was expected, and kept the number of losses down, in actual fact the losses were less than 1% (for a $P_e = 10^{-2}$). This then justifies the use of FEC and also shows that employing FEC greatly increases the reliability of any communications protocol. Although it initially decreases throughput, it increases the channel utilisation and reliability in that it decreases the number of retransmissions required.

- Lastly, a comparison has to be made between the simulation results, and the expected results calculated in chapter 5. This however will be difficult due to various reasons:

    1. The first of these was covered earlier in this chapter. It was found that for varying packet sizes the throughput still remained the same. This means that because the size of the packets that Erlang uses in its communication is unknown, it is difficult to determine the actual throughput taking all communication into consideration.
    2. It is currently very difficult to measure the offered channel load. Packets are generated at regular intervals, but since the variation that Erlang introduces into the packet generation is unknown, it cannot be calculated sufficiently for comparison.
    3. Due to the variance found in the above simulations, it is difficult to establish from the measured results the exact channel utilisation versus offered channel load per value of *N* as the throughput does not seem to be an accurate description of the channel utilisation.

Furthermore, although packet generation was handled by the communications protocol and packets were in 99 % of the cases routed to their specific destination without being lost, some information on the true effectiveness and throughput capabilities of the protocol is lost due to difficulties caused by Erlang. It seems that Erlang might not be suitable for this kind of implementation or simulations of this kind and in section 6.4 this will be discussed in a little more detail.

From the above it is clear that Erlang has had a major effect on the routing of messages as well as on the throughput. Due to the limited time, it is currently impossible to properly quantify these influences, but this will have to be done in future work if Erlang is to be considered reliable for this purpose. However, it was decided that before coming to a final conclusion, a few extra basic benchmark simulations would be run.

With these simulations, the goal was to try and establish whether Erlang constantly introduces some form of variance between iterations, as well as to determine whether this variance is measurable and constant. Furthermore, the effect that I/O (input/output) has will also be measured by running comparative simulations where no I/O is employed, where I/O to file is employed and finally where I/O to the screen is employed.

It has to be mentioned that the protocol simulated adhered to all the algorithm requirements as set out in he beginning of this work. Furthermore, although no direct comparison could be made between the theoretical and measured results, due to the effects that Erlang had on the simulation results, the theoretical values are still valid and should further simulations be done using a different simulation tool/platform, it is expected that the measured results of such a further simulation would be similar, although slightly less, than those calculated in chapter 5.

6.4. Benchmark simulations to verify Erlang

This section was added after all simulations and testing was completed. The goal of this section is to try and quantify the variance that Erlang introduces into simulations and simulation results. If possible, this variance will be measured to try and determine whether it is constant or variable with some factor.

To achieve this, 3 simulations of three iterations each were completed, each iteration comprising 1000000 packets being sent. In each simulation a 5 bit packet was encoded using RM(1, 4) and sent using Erlang's communications methods. Each packet was sent from a sending node to a receiver node and noise was added to the packet in the same manner as in the rest of the chapter 6 simulations. The packet was then decoded using the correct FEC method and two values measured after decoding. The first value is the number of bit errors detected after decoding and secondly the number of faulty packets received.

The difference in the three simulations lies in the following:

- The first simulation was run without any I/O and results were logged after 1 million packets were sent,
- A second simulation was run with I/O to file. Therefore, during each iteration the measured values were output to file,
- The third was implemented without I/O to file, but with I/O to the screen. During each iteration the measured values were output to the screen and the values logged, after 1 million packets were sent.

Each of the above simulations was run for three iterations each to obtain a better spread and to determine whether Erlang introduces variance within each test case or only across each different test case. The results are displayed in Table 6.13.

	Corrupted bits	Corrupted packets	Bit corruption percentage	Packet corruption percentage
No I/O				
Simulation 1 Iteration 1	1593	1552	0.1593 %	0.1552 %
Simulation 1 Iteration 2	2240	2190	0.224 %	0.219 %
Simulation 1 Iteration 3	1950	1897	0.195 %	0.1897 %
File I/O				
Simulation 2 Iteration 1	254	246	0.0254 %	0.0246 %
Simulation 2 Iteration 2	331	322	0.0331 %	0.0322 %
Simulation 2 Iteration 3	155	151	0.0155 %	0.0151 %
Screen I/O				
Simulation 3 Iteration 1	1444	1410	0.1444 %	0.141 %
Simulation 3 Iteration 2	1055	1013	0.1055 %	0.1013 %
Simulation 3 Iteration 3	448	400	0.0448 %	0.04 %

Table 6.13 Results of benchmark simulations to determine possible variance caused by Erlang

From the above, as well as Table 6.14, it can clearly be seen that Erlang has a significant influence on communication as well as the results obtained.

In the first case there is a variance between the results obtained in the different iterations within each simulation. This deviation is low to high, as can be seen in Table 6.14, where the maximum deviation for simulation 1 was 17 %, for simulation 2 it was 37 % and for simulation 3 it was up to 54 %. The deviation of 17 % can be attributed to factors such as processor overhead, external software and operating system related factors, etc. However, a maximum deviation of 54 % for iteration 3 is alarming. Therefore, it is certain that Erlang introduces variance into the results obtained by simulations.

	Equation	No I/O	File I/O	Screen I/O
Minimum value		0.1593 %	0.0155 %	0.0448 %
Maximum Value		0.224 %	0.0331 %	0.1444 %
Average between all 3 values		0.1927 %	0.02466 %	0.0982 %
Positive deviation (between avg. and max value)	(Max – Avg.) / Avg.	16.24 %	34.23 %	47.04 %
Negative deviation (between avg. and min value)	(Min – Avg.) / Avg.	-17.33 %	-37.14 %	-54.37 %
Maximum deviation (absolute value)		17.33 %	37.14 %	54.37 %
Deviation between maximum and minimum value	(Max – Min) / Max	40.615 %	53.17 %	68.97 %

Table 6.14 Calculated deviation of each simulation. This provides and idea of how much variance Erlang could possibly introduce into each simulation performed. This also that Erlang introduces quite a high variance into values obtained.

Furthermore, there seems to be no constant difference between the values of the iterations within each simulation, i.e. a variance between values with some constant percentage/value. This means that Erlang introduces some form of variance, in a seemingly random way. Furthermore, this also shows that the variance Erlang introduces might be either of an internal (Erlang related) or external (Computer, Operating system, etc.) nature. It is however difficult to ascertain exactly as to why or how Erlang introduces this variance.

Furthermore, Erlang also introduces variance between the different simulations. This variance is markedly higher as can be seen in the differing results obtained between no I/O, File I/O and Screen I/O. There is especially a significant drop in errors detected in the File I/O case which introduces a high variance between the three simulations. It is strange that such a drop should occur as writing to a file should realistically not alter the measured values. Furthermore, we can see that there is a significant variation difference between each of the three simulations, as shown in Table 6.14 (17 % up to 54 %). This again is hard to quantify, but goes to show that Erlang introduces variances into all the simulations that were completed, irrespective of the way they were done.

Furthermore, the results show that Erlang's I/O primitives have an effect on simulation as well as the results obtained. This means that al simulations will from hereon have to be designed with this factor kept in mind. Therefore, to achieve more stable results in future, the use of Erlang's I/O primitives, and in fact other primitives as well, will have to be re-evaluated.

From the above, it is clear that Erlang does not provide invariant simulation results. This variance was clearly introduced into all the other simulations completed in chapters 5 and 6. On this basis, it can therefore be assumed that the anomalous values measured during all of the simulations, were more likely than not introduced by Erlang and not by the communications system simulated.

Although hard to quantify, it is safe to assume that the anomalous values were not caused by the communications protocol itself, as in this section we can clearly see that anomalous values and variance were present in even simple communications and simulations. Therefore, it can be surmised that were another simulation tool, other than Erlang, employed for the purpose of this work, it would have been shown that the routing algorithm does in effect work and should be reasonably effective. There is no good reason to expect otherwise. This is also borne out by the results, if the anomalous values are ignored.

6.5. Conclusion

The simulations and the results were very useful and interesting. It proved that the implemented protocol is successful and it held up under various tests. Very promising results were obtained, granted that a few anomalies were found, as discussed above. However, an attempt was made to quantify and identify these anomalies, as best possible. Furthermore, all of the following initial requirements were met:

- The protocol employed multi hop communication,
- Unused nodes went into a standby mode,
- Error correction and error handling were implemented to minimise the influence of noise,
- The protocol caters for a varying number of nodes and is scaleable,
- The protocol employed deterministic routing,
- Hidden and exposed nodes were eliminated as communication was controlled by the mother/root node,
- The routing table was designed in such a way as to avoid complexity,
- Operational complexity overall was kept to a minimum.

The simulation results proved that the particular communications protocol is a feasible solution to the objectives stated in Chapter 2. Reasonable throughput was obtained, in line with expected values as shown at the end of Chapter 6.4.5. The algorithm was stable and performed well under the test conditions and met all initial requirements. It has to also be mentioned that a novel n-ary tree implementation was used, whereas most other implementations are of a binary tree nature, and this worked quite well.

Furthermore, as can be seen in this chapter, the routing algorithm provided reasonable efficiency under noisy conditions, and provided very satisfactory results under noisy conditions when FEC was employed. For most values of $n > 2$ the efficiency was above 78 % for all conditions, whether noisy or noiseless. This shows that the routing algorithm is efficient in discovering nodes. The only question though is how the algorithm would perform in a simulation language where broadcasting is available and whether the efficiency would be different or similar from that measured in this work. It is however believed that a high efficiency will still be achieved due to the efficient nature of tree structures.

There are however, possible changes that can be recommended to further improve the protocol. The next logical test of the protocol would have to be a real world implementation.

Chapter 7
Summary of Work Done, Critical Analysis of Results, Contributions and Future Work

7.1. Introduction

During the course of this work we looked at various factors influencing and affecting underwater communication. Initially we looked at the factors directly involving communication as well as the nodes used in a sonar swarm. From there an overview of the marine comms environment was given. It was shown that all of these factors play a differing, but significant, role in underwater communication and specifically acoustic sonar swarms.

From there the choice was made to analyse current solutions found in the implementation arena as well as to postulate possible solutions were we to create our own communications protocol. It was further deemed worthwhile to design a simple but effective communications protocol and test some possible solutions in accordance with the ISO standard.

It was found that a relatively simple tree-based communications protocol is not only possible, but feasible, promising satisfactory for underwater communication. However, certain pitfalls were discovered in using Erlang as a simulation vehicle as well as possible implementation vehicle.

The final results were, however, favorable and it is possible to say that initial groundwork has been made and a solid foundation from which to work from has been laid.

7.2. Critical Discussion of Results

The results were quite favourable in terms of what was expected, and it was proved that the implemented protocol adhered to all requirements and is well behaved. Some obstacles have, however, been encountered, specifically in terms of using Erlang. This will however be addressed in Chapter 7.4.

The following was determined during the work:

- The underlying ocean environment (propagation delay, high noise characteristic, etc.) clearly affects the design of the communications protocol,
- Due to the high propagation delay and high P_e of the ocean, a high throughput is not easily achievable,
- Using a contentionless scheme presented a simple solution and proved favourable in handling errors as well as providing a suitable throughput,
- The chosen value for *n* for the tree-based scheme proved effective and useful. Higher values might have negatively affected the advantage provided by a multi hop communication scheme. Lower values would introduce increased hops and lower the chance of discovering as many nodes as possible with the initial broadcast,
- Flow control and FEC is a definite must in underwater communication,
- Using a broadcast based initial discovery scheme proved useful,
- Using Erlang, although not ideal, proved viable, although Erlang introduces anomalies,
- Employing wait/sleep and active modes for nodes worked and did not detrimentally affect functioning of the protocol,
- Using a deterministic communication scheme provides better control over communication between nodes and does not negatively affect throughput, as sensors tend to have a low arrival rate for packets,

- All the separate components (FEC, Routing, Flow Control) all contributed to create a more effective communications protocol.

7.3. Contributions

During the course of this work several contributions were made. They also serve as groundwork for future work. These are as follows:

- An overview of the ocean environment, relating to underwater comms, with factors affecting the nodes, positioning of nodes as well as communication between nodes,
- In depth analysis of the technical aspects of an acoustic sonar swarm, indicating protocol design constraints,
- Analysis of communications in terms of the abbreviated four layer ISO model. Analysis of each level as well as the relevance of each level in underwater communication,
- Analysis and simulation of possible options for the design of a communications protocol, specifically the type of FEC, flow control, type of contention scheme and finally the type of routing scheme,
- A novel, scalable, *n*-ary, tree based access control scheme was used. Most current implementations are of a binary nature,
- A method was developed to simulate a broadcast mode in Erlang, as Erlang does not have a built in broadcast mode,
- A simple flow control method, for use in Erlang, was developed, as it was uncertain whether Erlang employed it’s own flow control,
- FEC was implemented above any FEC that Erlang itself employs, as it is unknown whether Erlang employs FEC in any form,
- Benchmark simulations were run to determine whether Erlang has any external effect on communication as well as to try and determine what that effect might be,

- The strategy presented took the physical layer into account and subsequently addressed the ISO data link-, network- and transport layers respectively,
- Simulations were run with and without the presence of noise to determine what effect noise has on packet loss in a system with Flow control, FEC and using a deterministic access scheme,
- Finally, an analysis and simulation of Erlang was made with benchmark simulations to provide further insight into the usability as well as the feasibility if Erlang. Also certain pitfalls and shortcomings were discovered and mentioned.

7.4. Critical discussion of Erlang

Erlang clearly has its advantages and disadvantages for use in developing a communications protocol such as described. Erlang allowed communication between nodes as well as the ability of creating client and server programs that could be run on several distinct and separate nodes. At the same time, Erlang displayed significant shortcomings:

- Erlang has a steep learning curve. This is due to the fact that Erlang is concurrency orientated (with strong recursion factors) and in no way object orientated. Some time has to be taken to familiarise oneself with the mechanics and primitives that Erlang employs. Part of this work was to determine the applicability of Erlang as a simulation tool in this particular field, as well as to determine some of the shortcomings that Erlang might pose.
- Secondly it takes a bit of a mind shift away from ideas such as inheritance and dynamic variable assignment, as Erlang does not provide coding tools such as these, which would be present in a language such as Java. What compounds this is that although useful information regarding Erlang does

exist, they are incomplete and provide limited information on the inner workings of Erlang.

- Furthermore, Erlang does not allow true broadcasting. Erlang uses its own methods to send messages between nodes and requires a destination address, as well as a destination method, when sending messages. Therefore, when sending a message the destination has to be specified in all cases. This also leads to the next problem,
- Erlang hides its method of sending messages. This denies the user knowledge of the particular algorithm as well as possible problems that might be caused by the particular implementation.
- Erlang also employs a type of fire-and-forget strategy in terms of sending messages. Erlang provides no way for the user to know whether the message safely reached its intended destination. The documentation on the other hand does state that if a node exists, then any message sent to that node, will correctly reach that node.
- It is for the above reason as well as others, as mentioned earlier, that for the purposes of this work, flow control was implemented via an Ack/Unack system. This was done to allow more control over the communications simulations, as well as to ensure that flow control was implemented.
- The following is still unsure: How does Erlang ensure safe and reliable transmission of packets and what packet sizes does Erlang employ when sending messages. Is there some fixed length or is it a variable length as required. It seems from the simulation results in section 6.2, that Erlang employs fixed packet lengths, but this is still uncertain.
- Lastly, since this is the first time that Erlang has been utilised at the university for this type of purpose, there is little knowledge of the internal functioning of Erlang. Without that detailed internal knowledge, it cannot really be discerned in which ways Erlang itself might affect the simulation results obtained. Certain questions and concerns have, however, been discovered and documented during the duration of this work.

- An indication of pitfalls to be avoided in future work, is also valuable, although not always very satisfying.

Despite the above, Erlang proved very useful for the simulation requirements of the work described and appears to hold some promise for future work, provided it can be calibrated properly.

7.5. Recommendations and Future Work

Since this was a first attempt at the creation of an underwater sensor network comms protocol, there are recommendations and possible changes that could be made. These can be summarised as follows:

- If Erlang is to be used as simulation tool it would be essential to gather more information on the internal workings of Erlang. With this in mind it would also be beneficial to run various benchmark simulations to determine base values for Erlang. It will be necessary to determine the range of applications for which it would be suitable and the confidence with which the various results could be accepted. It might also be beneficial to attend have personal contact with the developers of Erlang.
- Using a different simulation language might give comparative results or help to further prove the validity of this algorithm, as well as lower the learning curve encountered with the use of Erlang.
- It is also feasible, that if Erlang were to be used for this type of simulation in future, that the lower level communications channel currently being controlled by Erlang, be simulated/programmed as well. This means creating an entire simulations channel in Erlang, beyond that which Erlang provides. This should provide for more flexibility in terms of factors such as setting the baud rate and broadcast abilities. This should then provide for better and more stable results.

- Although the simulations provided quite positive results, it is possible that the code might be even more streamlined than it is at present. The streamlining of the code in this case would have been aided if it were possible to attain more detailed information on the structure of Erlang.
- Using different values of *n* could provide further insight towards an operational system. Further development fine tuning of the tree based structure appears to hold much promise,
- Trying different types of message sizes combined with different RM encoding schemes could broaden the amount of results obtained.
- The next logical step would be to create the sensor nodes and do a real world implementation by embedding the client and server programs in hardware and testing the routing algorithm in real life.
- It would also be interesting to do some research into the application of other tree structures to this type of network and to establish some comparative results against this first effort,

7.6. Final Conclusion

The work done during the course of the project was interesting and provided a substantial amount of information and useful insight – not only in respect of the promise of the actual protocol implemented, but also as far as the possible usefulness of Erlang is concerned, for this type of application.

It is clear that much scope remains for future work in this field and the actions recorded in this document, should be seen as an exploration effort to establish a better knowledge base in this regard.

Bibliography

1. Akyildiz, I.F. Pompili, D. Melodia, T. *Challenges for Efficient Communication in Underwater Acoustic Sensor Networks*. Broadband & Wireless Networking Laboratory, Georgia Institute of Technology, Atlanta USA.

2. Nicopolitidis, P. Papadimitriou, G. I. Pomportsis, A. S. *Distributed protocols for Ad-Hoc wireless LANs: a learning-automata-based approach*. Science Direct. http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B7576-49KS3HD-3&_coverDate=10%2F31%2F2004&_alid=254552660&_rdoc=1&_fmt=&_orig=search&_qd=1&_cdi=12890&_sort=d&view=c&_acct=C000032099&_version=1&_urlVersion=0&_userid=613892&md5=0cd29bb39af687b9cdb5d03663a9a574 .

3. Jurdak, R. Lopes, C.V. and Baldi, P. *A survey, classification and comparative analysis of medium access control protocols for Ad-Hoc networks*. IEEE Communications Surveys & Tutorials, vol. 6, no. 1, First Quarter 2004.

4. Stojanovic, S. *Underwater acoustic Communication.* Department of Electrical and Computer Engineering, North-eastern University, Boston MA.

5. *Underwater Acoustic Modem*. http://www.link-quest.com .

6. Melodia, T. and Pompili, D. *Underwater Acoustic Sensor Networks (UW-ASN)*. http://www.ece.gatech.edu/research/labs/bwn/UWASN/ . BWN Lab.

7. Sozer, E.M. Stojanovic, M. and Proakis, J.G. *Underwater Acoustic Networks*. IEEE journal of Oceanic Engineering, Vol. 25, Nr. 1, January 2000.

8. *Underwater Acoustic Communications*. http://www.arl.nus.edu.sg/web/research/acomms

9. Bessios, A.G. and Caimiim, F.M. *High-Rate wireless Data Communications – an Underwater Acoustic Communications Framework at the Physical Layer.* Department of Electrical Engineering, Harbour Branch Oceanographic Institution, Inc. Florida, USA. 10 October 1995

10. Stojanovic, M. Proakis, J.G. and Catipovic, J.A. *Performance of High-Rate Adaptive Equalisation on a Shallow Water Acoustic Channel.* Department of Electrical and Computer Engineering, North-eastern University, Boston, USA. 1994.

11. http://www.ericsson.se/erlang

12. Armstrong, J. Virding, R. Wikstrom, C. Williams, M. *Concurrent programming in Erlang*. Second Edition. Ericsson Telecommunications Systems Laboratories. Prentice Hall, Englewood Cliffs, New Jersey,

13. Labuschagne, A.S. *The design of a telemetry system for Grumetri Reserves*. Department of electronic engineering, University of Stellenbosch, South Africa. April 2006.

14. Cooke, B. *Reed-Muller Error Correcting Codes.*

15. Raaphorst, S. *Reed-Muller Codes.* Carleton University. May 9, 2003.

16. Wolhuter, R. *Telecoms 823 Class Notes*. University of Stellenbosch, South Africa. 2004.

17. Jurdak, R. Lopes, C V. Baldi, P. *Software acoustic modems for short range Mote-based underwater sensor networks.* California institute for telecommunications and information technology. University of California, USA.

18. Melodia, T. Akyildiz, I, F. Pompili, D. *Underwater acoustic sensor networks: research challenges*. Broadband and wireless networking laboratory. Georgia institute of technology, USA. 21 January 2005.

19. Kleinrock, L. Tobagi, F, A. *Packet switching in radio channels: part I – Carrier sense multiple-access modes and their throughput-delay characteristics*. IEEE Transactions on communication, vol. COM-23, no. 12. December 1975.

20. Takagi, H. Kleinrock, L. *Throughput analysis for persistent CSMA systems*. IEEE transactions on communications, Vol. COM-33, No. 7, July 1985.

21. *ISO/OSI Network Model*. http://www.ussg.iu.edu/usail/network/nfs/overview.html

22. *OSI Model.* http://www.wikipedia.org