2008-12

A discovery process for initializing ad hoc underwater acoustic networks

Ong, Chee Wei.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A DISCOVERY PROCESS FOR INITIALIZING AD HOC UNDERWATER ACOUSTIC NETWORKS

by

Ong, Chee Wei

December 2008

Thesis Advisor:	Joseph A. Rice
Second Reader:	John C. McEachen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2008	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE A Discovery Process for Initializing Ad Hoc Underwater Acoustic Networks	5. FUNDING NUMBERS
6. AUTHOR(S) Ong, Chee Wei	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

Seaweb is an underwater acoustic wide-area network connecting autonomous, distributed nodes. Prior iterations of Seaweb relied on operator intervention to initialize and manually configure the network routes. This thesis implements a network discovery process that enables a field of spontaneously deployed, ad hoc nodes to auto-configure for networking purposes. Network routing is initialized as nodes in the network are discovered, with routes chosen according to comparative evaluation of a cost function for all candidate routes. The implemented network discovery process is tested using computer simulation and sea trial data. The resultant network routes obtained upon completion of the ad hoc network discovery process are compared with those derived from Dijkstra's algorithm. It is concluded that the network discovery process always produces a shortest-path route from a master node to any other discovered nodes in the network. Sensitivity studies on the route cost evaluation function are performed, and an alternative network discovery scheme is discussed.

14. SUBJECT TERMS Underwater networks, acoustic networks, ad hoc networks, network discovery, Seaweb, acoustic communications, acomms, telesonar	15. NUMBER OF PAGES 115
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A DISCOVERY PROCESS FOR INITIALIZING AD HOC UNDERWATER ACOUSTIC NETWORKS

Ong, Chee Wei
Lieutenant-Colonel, Republic of Singapore Navy
B. Eng. (Hons), Nanyang Technological University, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING ACOUSTICS

from the

NAVAL POSTGRADUATE SCHOOL
December 2008

Author: Ong, Chee Wei

Approved by: Joseph A. Rice
Thesis Advisor

John C. McEachen
Second Reader

Daphne Kapolka
Chair, Engineering Acoustics Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Seaweb is an underwater acoustic wide-area network connecting autonomous, distributed nodes. Prior iterations of Seaweb relied on operator intervention to initialize and manually configure the network routes. This thesis implements a network discovery process that enables a field of spontaneously deployed, ad hoc nodes to auto-configure for networking purposes. Network routing is initialized as nodes in the network are discovered, with routes chosen according to comparative evaluation of a cost function for all candidate routes. The implemented network discovery process is tested using computer simulation and sea trial data. The resultant network routes obtained upon completion of the ad hoc network discovery process are compared with those derived from Dijkstra's algorithm. It is concluded that the network discovery process always produces a shortest-path route from a master node to any other discovered nodes in the network. Sensitivity studies on the route cost evaluation function are performed, and an alternative network discovery scheme is discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank the following people for making my learning journey at the Naval Postgraduate School an enjoyable and fruitful experience:

First and foremost, to my lovely wife, Lee Lian, thank you for the unwavering support, love and patience. Thank you for taking such good care of our beautiful daughters, Jolene and Jasmine.

To my thesis advisor, Professor Joe Rice, thank you for offering me an opportunity to be part of the exciting development involving Seaweb networks. I appreciate your guidance, patience and words of encouragement as I worked on the thesis.

To my thesis second reader, Professor John McEachen, thank you for taking precious time off to comment on my drafts.

To the tireless staff at SPAWAR Systems Center Pacific comprising Bill Marn, Chris Fletcher, Bob Creber, Doug Grimmett and Lonnie Hamme, thank you for sharing the practical knowledge associated with the deployment and configuration of the Seaweb networks.

To the software engineers at Teledyne Benthos, namely, Mike Coryer and Rob Pinelli, thank you for sharing ideas regarding ad hoc network discovery and for providing suggestions on how I should develop my simulation.

To Dr Garry Heard of the Defence Research and Development Canada (DRDC) Atlantic, thank you for making the Unet 08 trial log available to me.

To Dr Tom Drake and Ms Jody Wood-Putnam of ONR 321, thank you for sponsoring the ad hoc network discovery implementation on Seaweb, and US participation in the June sea trial.

To Danna Hesse of ONR 321, thank you for sponsoring US participation in the September sea trial.

Last but not least, to all the professors and friends with whom I have interacted, thank you for making the NPS experience a memorable one.

I. INTRODUCTION

Seaweb is an underwater acoustic wide-area network that interconnects fixed and mobile nodes [1]. Each node is equipped with a digital signal processor (DSP)-based telesonar (i.e., *tele*communications *so*und *na*vigation and *r*anging) modem for through-water acoustic communications [2]. The Seaweb network enables data-telemetry and command-and-control capabilities across a set of deployable autonomous undersea sensors and vehicles [3].

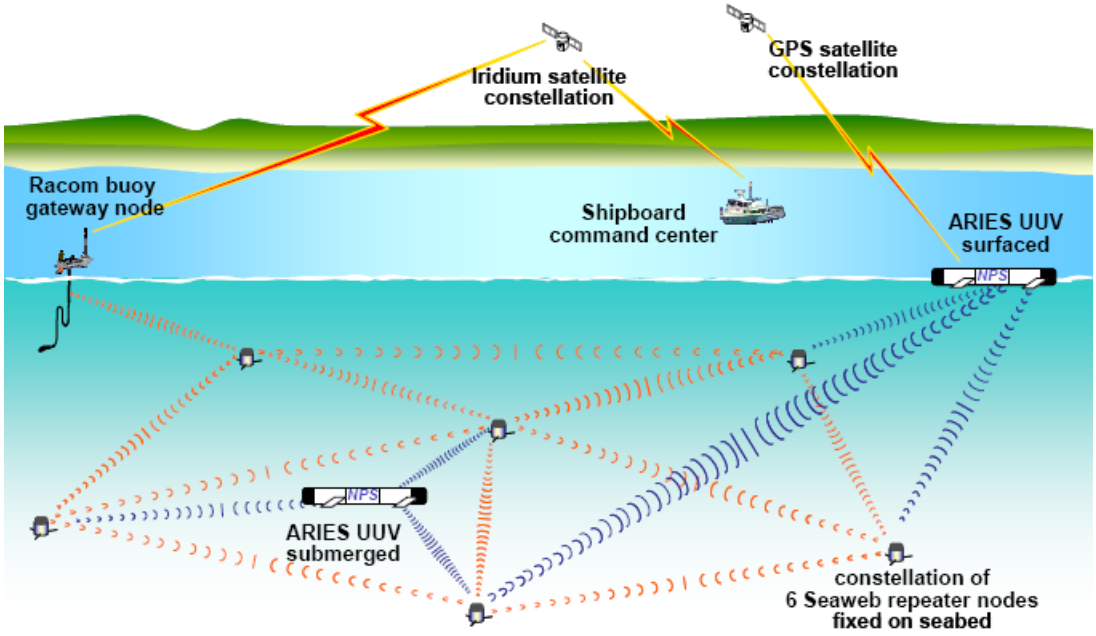


Figure 1. An example of a Seaweb acoustic network comprising fixed and mobile nodes with satellite communications back to a command center via the radio/acoustic communication (racom) gateway buoy [From 4].

The ability of the submerged network to "reach back" to a command center that may be situated at a remote locality is provided by the racom buoy gateway node as depicted in Figure 1.

A Seaweb server resides at the manned command center. Its role is to configure, monitor and manage the underwater network including its routing strategies [5, 6]. The server also archives and publishes sensor data reported

from the network nodes, enabling near-real-time access by clients, including those interested in ascertaining underwater situational awareness.

A. PROBLEM STATEMENT

Prior iterations (prior to May 2008) of Seaweb networks have relied exclusively on operator intervention to initialize and manually reconfigure the network routes, with emphasis on routes from a master node (usually the gateway node) to all other nodes in the network. Such a network initialization process necessitates two assumptions – that the total number of nodes in the network be known a priori to the operator, and that the operator has the necessary tools to decide on a network routing strategy that is consistent with the prevailing propagation conditions.

The need for operator intervention to manually configure the network routes, either pre- or post-deployment, is contrary to the desire of having a field of spontaneously deployed, autonomous nodes that are capable of auto-configuration for networking purposes.

B. SCOPE OF THESIS

This thesis seeks to address the aforementioned issues regarding initialization of an ad hoc Seaweb network by designing an underwater network discovery process, implementing the process in simulation, testing the process with experimental and synthetic data, and evaluating the resultant network routes that come as a natural by-product of the discovery process.

C. STRUCTURE

The coverage of this thesis demands an understanding of underwater acoustics and the ocean as a communications channel, and an appreciation of basic communications and network flow theory.

Chapter II of this thesis covers the current state of the art in underwater ad hoc network discovery and related research in network routing protocols.

Chapter III provides an overview of the challenges posed by the physical ocean medium on acoustic communications in the context of Seaweb physical layer and the June 2008 Halifax trial environment, where the implemented network discovery process was first tested at sea.

Chapter IV describes the Seaweb network architecture, specifically its link-layer and network-layer implementation. Existing features such as *ping/echo*, and *broadcast ping* utility packets that are instrumental in the implementation of the proposed discovery process are discussed.

Chapter V develops considerations for the design of the ad hoc network discovery process. Network topology discovery methods such as breadth-first search and depth-first search are examined. The proposed network discovery process for initializing an ad hoc Seaweb network is then presented. Parameters associated with the cost function used for network routing, executed in tandem with the discovery process, are also explained.

Chapter VI reports the June 2008 Halifax sea trial and the analysis of experiment versus simulation results. Sensitivity studies involving the cost function parameters are presented.

Chapter VII proposes a slight variation to the implemented discovery process and discusses its pros and cons vis-à-vis the incumbent scheme.

Chapter VIII presents the conclusions of this thesis and offers recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter covers the current state of the art in underwater ad hoc network discovery and related research in underwater network routing strategies.

A. RELATED WORK

While discovery and routing protocols for terrestrial ad hoc and wireless sensor networks have been extensively studied [7, 8], their counterparts in underwater acoustic networks have received far less attention. While it might be tempting to try to adopt terrestrial wireless solutions to the undersea environment, the unsuitability of terrestrial-derived proactive, reactive and geographical routing protocols in the underwater medium was discussed in [9].

There is, therefore, a need to design equivalent network-layer protocols tailored for underwater acoustic networks.

Most research papers dwelling on underwater acoustic networks advocate some form of centralized planning of network topology and data paths in order to optimize scarce network resources, given that underwater networks are often smaller in scale and that network reliability over a prolonged period of deployment is a primary consideration [9].

In [10], a centrally-controlled underwater network discovery scheme was proposed in conjunction with a node localization algorithm. Network discovery starts with a primary seed node S_1 in a known position. Node S_1 broadcasts a discovery command packet that enables neighbors to establish distances from it, and waits for replies from nodes within earshot. When replies are received, the information such as node ID and distances are kept in the memory of S_1. The most distant node in the region of S_1 is then chosen as the second seed node S_2. The selection of the farthest node as the second seed node is to cover a larger area more quickly.

S_1 then broadcasts the information set of its discovery (containing node ID and distance measures) together with a discovery command specifying the node ID of S_2, to its neighboring nodes. As a result, all nodes in S_1's region have the discovery information set from S_1. The node designated as the second seed node S_2 then proceeds with the same manner of discovery. It then broadcasts the newly discovered information back to its neighbors. At this point, nodes within the intersect region of S_1 and S_2 have both information sets. In order to localize discovered nodes, a third seed node S_3 is chosen from nodes that lie in the intersection of regions S_1 and S_2. The discovery cycle then repeats with S_3 as the next seed node. Upon completion, nodes that lie in the intersection of regions S_1, S_2 and S_3 will have three sets of range measure and can thus be localized. Figure 2 depicts the intersections.

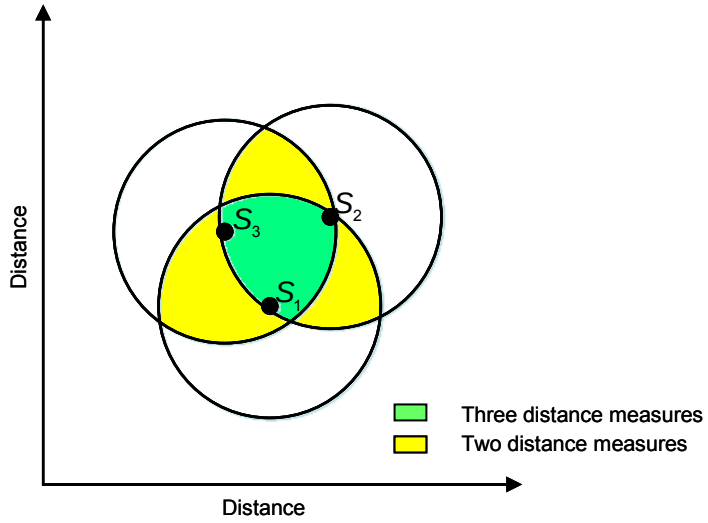


Figure 2. Intersections of two and three known distance measures upon completion of discovery cycles by the first three seed nodes S_1, S_2 and S_3 [After 10].

The above network discovery protocol offers insight into some of the issues that this thesis is trying to address. The only drawback is that the discovery process is optimized for node localization. Network routing is not a consideration.

In [11], another centrally-controlled network-layer discovery and routing protocol was proposed for underwater acoustic networks. It relies on a master node to discover the topology of the nodes that comprise the network. Topology discovery is done by the transmission of a probe by the master node to its nearest neighbors as shown in Figure 3. A probe is a topology discovery message (TDM) broadcast. The transmit level of the probe is set to a predetermined signal strength to limit the range of the probe. Upon receipt of a TDM, each neighbor appends its node ID to it and relays it to the next "ring of nodes", so that the probe propagates outward from the master. In addition, each neighbor selects a communication channel from a set of channels not already allocated. Therefore, the probe contains node IDs of nodes it traversed, as well as channel allocation for each of those nodes that relayed the TDM.

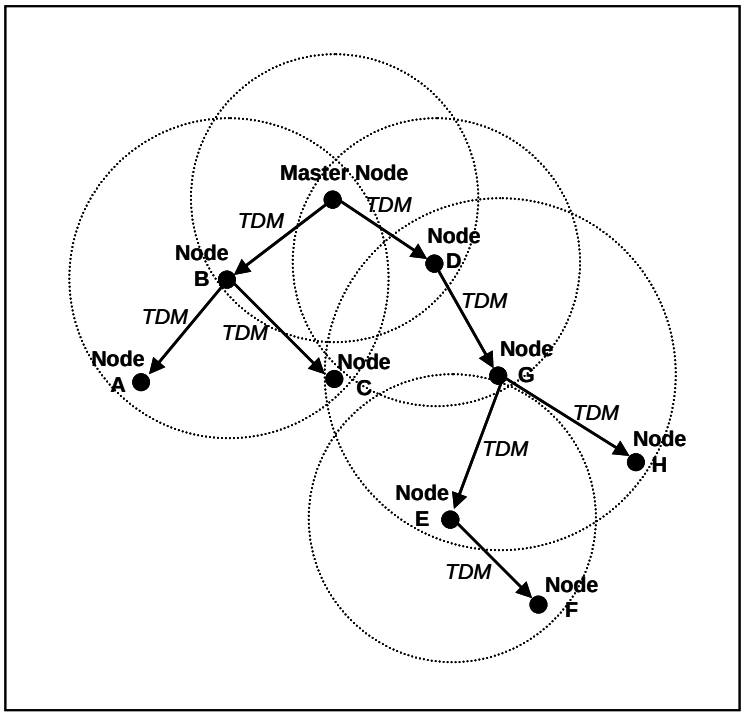


Figure 3. Topology discovery message (TDM) propagation in an underwater acoustic network. Each node forwards the TDM upon receipt. Circles represent the signal propagation radius about a given node [After 11].

When the probe reaches a leaf node, a topology completion notice is initiated. The topology completion notice is returned to the master node along the reciprocal route the topology discovery probe propagated. As the notice

transits each node in the route, the node appends the neighbor information it has collected from the discovery probes it received. As a result, when the topology completion notice reaches the master node, it contains the information necessary for the master to establish paths between each pair of nodes and to manage traffic across the network.

For this protocol to achieve the desired functionality, the communication links throughout the network were proposed to be full duplex. The full-duplex requirement is not readily met by current acoustic communications technology.

B. INSIGHTS

A survey of articles related to network discovery and routing schemes pertaining to an ad hoc network of underwater sensors points to the advantage of having a central manager (a master node) to initiate and propagate the network discovery process. The discovery process inevitably involves some form of broadcast message so that all nodes within earshot of this message will respond. The most significant result of an initial pair-wise discovery is the measure of round-trip sound propagation time, which is proportional to the node-to-node range. This discovery process ripples through the network, and the master node's knowledge of the network topology expands with each discovery cycle. Upon completion of the network discovery process, the master node has some knowledge of the network topology so that an optimized network routing can be determined.

The insights gained herewith are useful for the design and implementation of the discovery process for initializing an ad hoc Seaweb acoustic network of autonomous nodes, to be discussed in Chapter V.

III. THROUGH-WATER ACOUSTIC COMMUNICATIONS

This chapter provides an overview of the challenges posed by the physical ocean medium on acoustic communications. The discussion pertains to the Seaweb physical layer and the June 2008 trial environment where automated network discovery was first tested at sea.

A. SEAWEB PHYSICAL LAYER

The Seaweb physical layer is based on M-ary Frequency Shift Keying (MFSK) modulation of acoustic energy in the 9-14 kHz band [3]. MFSK uses multiple (M) frequencies, offset from the carrier frequency, to represent M different symbols, each containing n_b bits so that $M = 2^{n_b}$ [12]. A sample pulse train of a MFSK signal is shown in Figure 4.

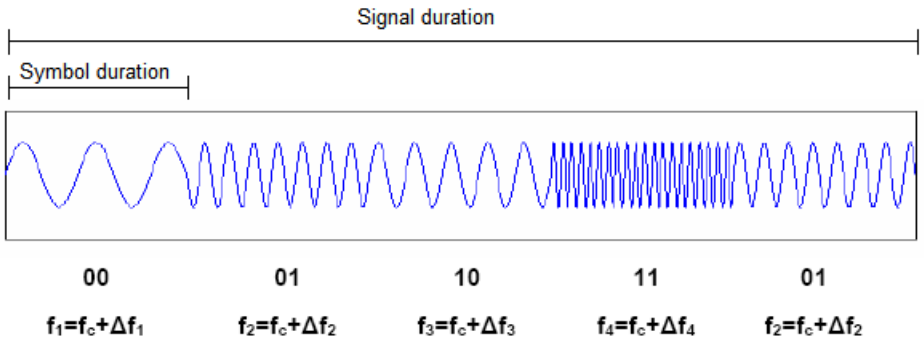


Figure 4. An example of 4-ary frequency shift keying using M = 4 frequencies to represent M = 4 symbols. Each frequency f_i is offset by a different amount Δf_i from the carrier frequency f_c [After 12].

Seaweb nodes are each equipped with a Teledyne Benthos underwater acoustic modem with a maximum source level (SL) of 186 dB re 1 µPa at 1 m. Varying degrees of forward error correction are employed to mitigate the high bit-error rate experienced in the acoustic channel.

B. HALIFAX TRIAL ENVIRONMENT

The June 2008 Seaweb ad hoc network discovery experiment (a part of the Unet 2008 trial) was conducted in St Margaret's Bay, Halifax, Nova Scotia, Canada.

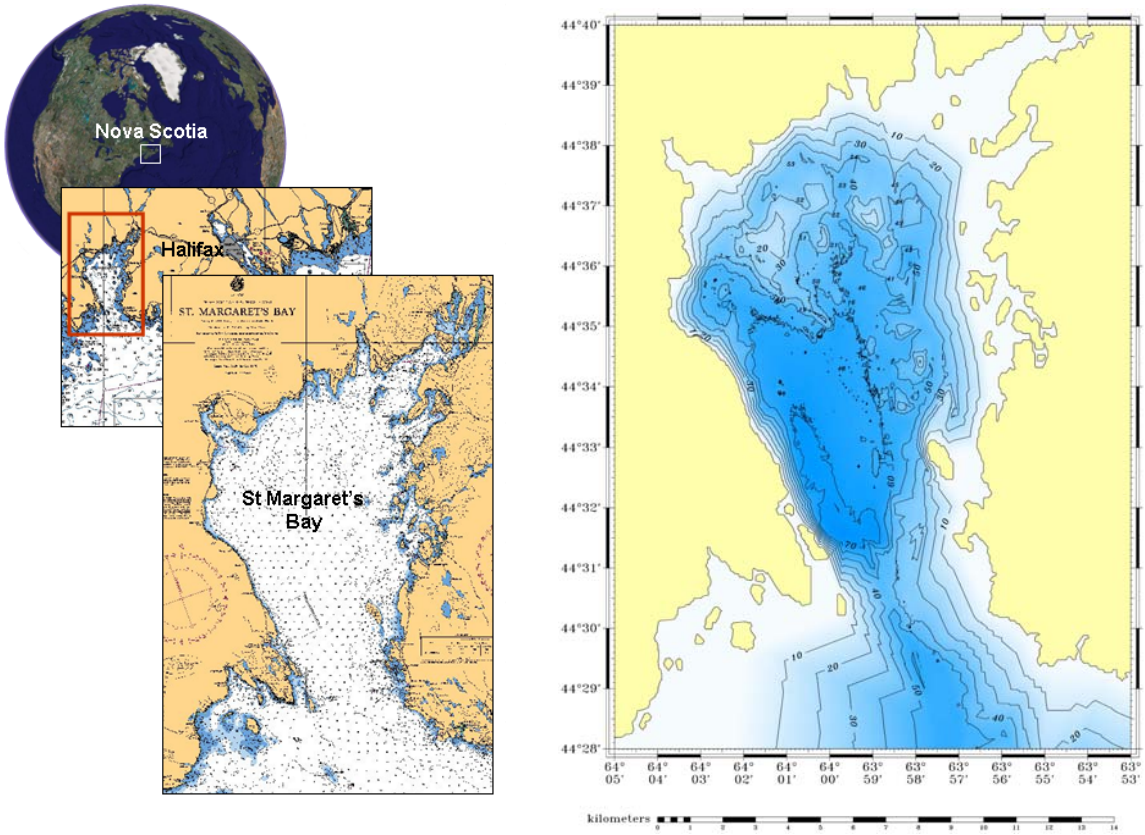


Figure 5. Location of June 2008 Seaweb ad hoc network discovery experiment (St. Margaret's Bay, Halifax, NS, Canada) and the bathymetry associated with the trial area.

The water depths in which Seaweb nodes were deployed varied from 30-70 m. Bottom type was generally sand and gravel. The surface wind speed ranged from 2 to 8 m/s during the trial. Shipping traffic was observed to be light.

C. THE COMMUNICATIONS CHANNEL

Acoustic communications bandwidth in the underwater environment is constrained by frequency-dependent transmission loss and non-white noise spectra. Communication is further challenged by multipath time spread, Doppler spread and highly variable propagation delay, five orders of magnitude larger than in radio frequency terrestrial channels. Together, these factors determine the temporal and spatial variability of the acoustic communications channel and make the bandwidth limited and dependent on both range and frequency [9].

1. Transmission Loss (TL)

Transmission loss arises from attenuation and geometric spreading [13]. Attenuation is primarily caused by absorption where acoustic signal energy is converted into heat [14, 15]. Absorption increases with distance and frequency. A plot of absorption coefficient as a function of frequency is given in Figure 6. For a Seaweb frequency range of 9-14 kHz, a value of 1 dB/km is a reasonable estimate for the attenuation coefficient.

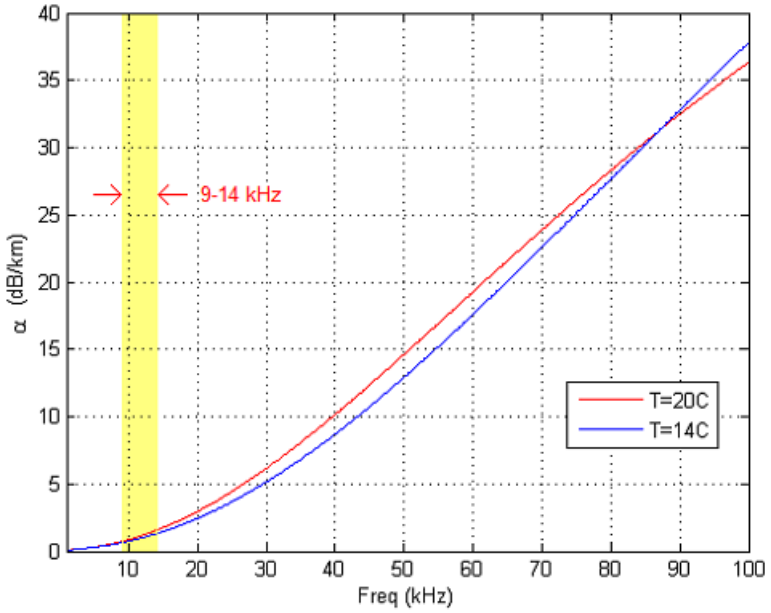


Figure 6. Attenuation coefficient α in dB/km versus transmission frequency in kHz based on Francois and Garrison [14, 15] for salinity S = 35 ppt, acidity pH = 8, and depth D = 50 m.

Transmission loss by geometric spreading refers to the spreading of acoustic energy due to wavefront expansion as sound travels away from the source. For the most part, it is independent of frequency and increases with propagation distance. Geometric spreading starts with spherical spreading close to the source, but channel boundaries may limit the propagation, absorbing some energy and reflecting or scattering the rest.

2. Noise Level (NL)

Noise in the ocean can be categorized into man-made noise and ambient noise. Man-made noise is mainly caused by machinery noise and shipping activity, while ambient noise is related to seismic and biological phenomena and movement of water including tides, currents, storms, wind, and rain.

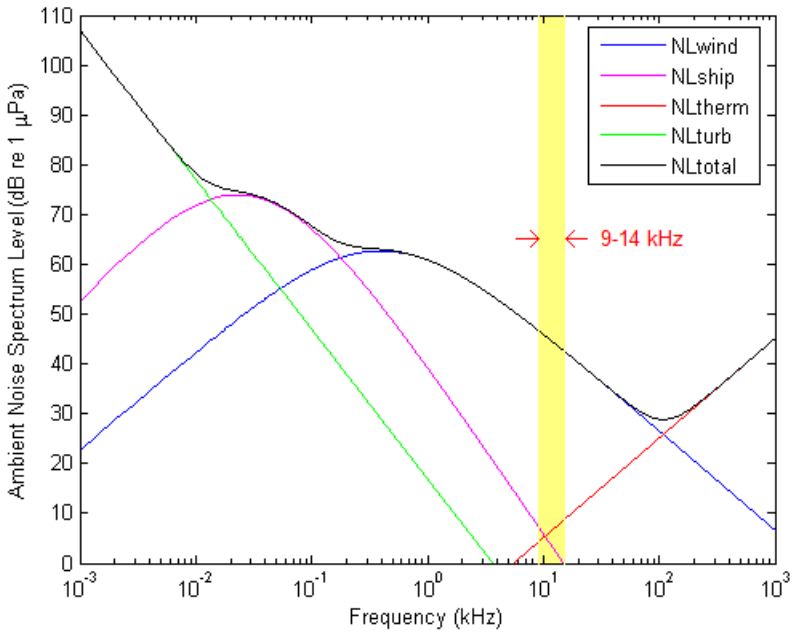


Figure 7. Noise spectrum level based on empirical formulae by Coates [After 12].

Coates [16] provided empirical formulae to estimate noise spectrum level (NSL) as a function of frequency. It can be observed from Figure 7 that different noise sources dominate different frequency bands, namely, turbulence (<10 Hz), shipping (10-200 Hz), wind (0.2-100 kHz) and thermal processes (>100 kHz).

For a Seaweb underwater acoustic network, wind noise is the main contributor to overall noise level. An increase in surface wind speed has a large effect on the noise spectrum level as illustrated in Figure 8, which in turn, may cause a drastic decrease in communications connectivity. Additionally, in-band noise levels experience episodic increases caused by passing boats and biological noise from shrimps, fish, and mammals.

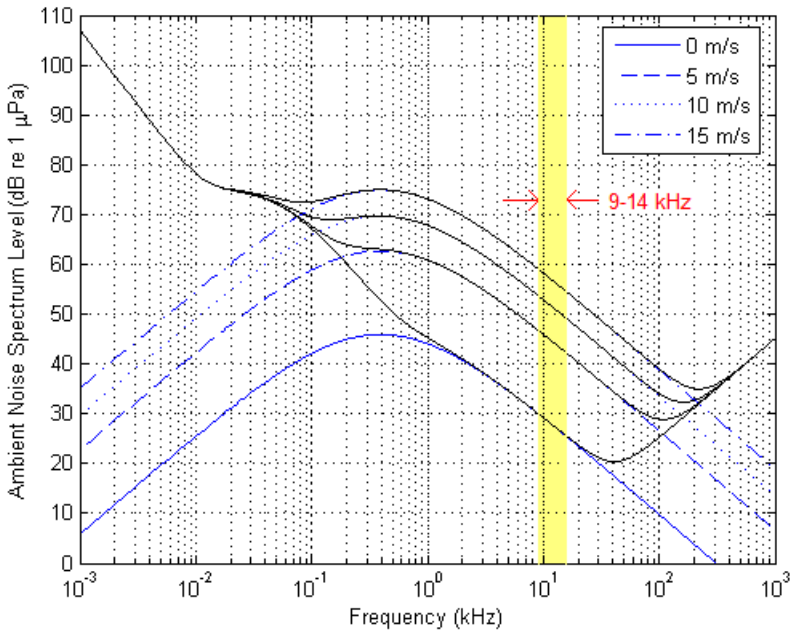
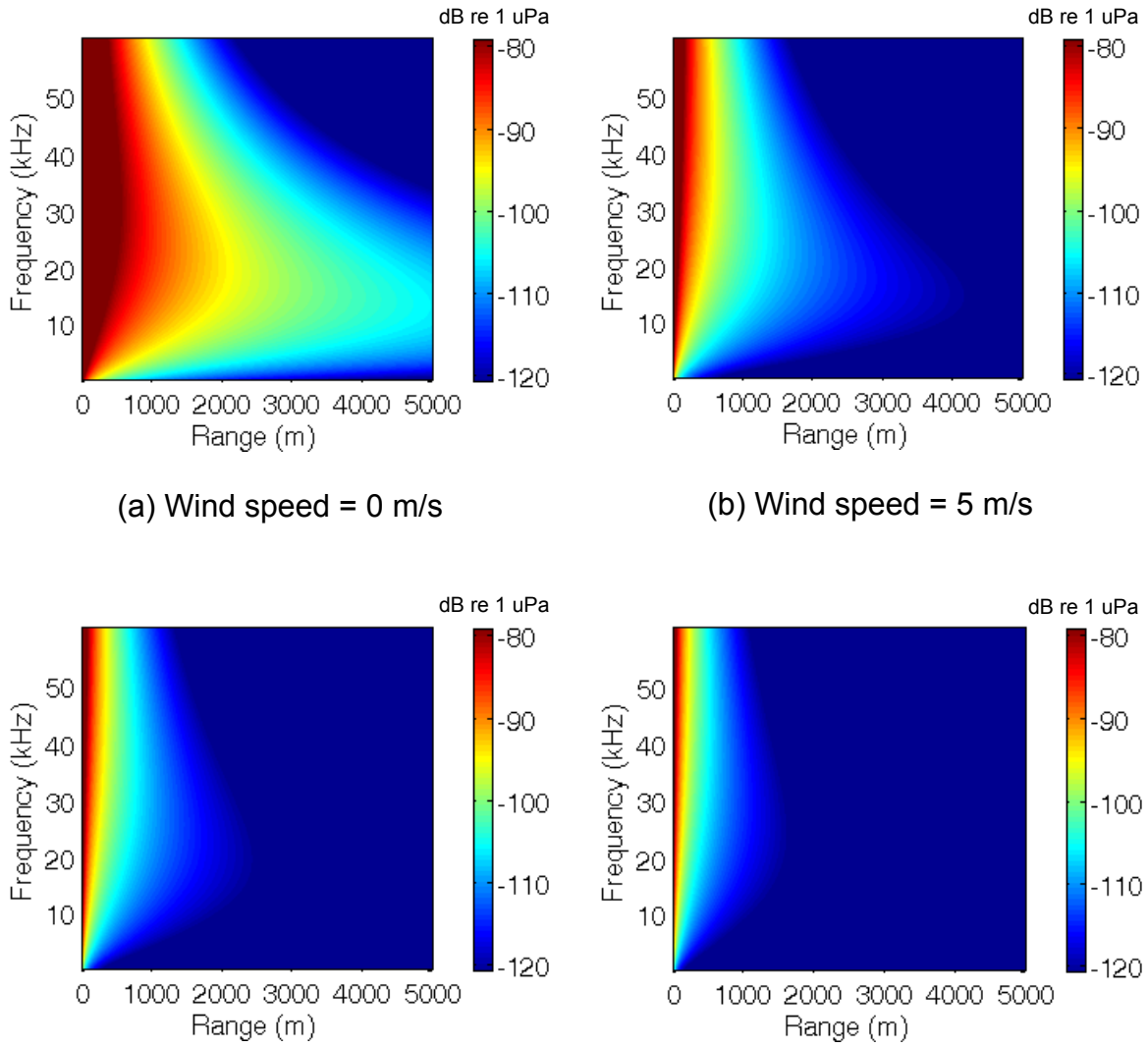


Figure 8. Effect of surface wind speed on noise spectrum level based on empirical formulae by Coates [After 12].

3. Channel SNR

The combined effects of transmission loss (TL) and noise level (NL) represent the gross channel impairment caused by environmental factors and is defined as the channel signal-to-noise ratio (SNR) [17]. It is both frequency and range dependent. Figure 9 shows the effect of varying wind speeds on channel SNR. It is a plot of $-(TL + NL)$ for a water depth of 50 m, temperature of 14 °C, salinity of 35 ppt, and light shipping traffic.



(a) Wind speed = 0 m/s

(b) Wind speed = 5 m/s

(c) Wind speed = 10 m/s

(d) Wind speed = 15 m/s

Figure 9. Effect of wind speed on $-(TL+NL)$ in dB re 1 μPa. For a wind speed below 5 m/s, an acoustic communication range of up to 4 km is possible. As wind speed increases, communication range drops drastically.

4. Multipath

Multipath propagation results in inter-symbol interference and thus may cause severe degradation of the received acoustic signal. Multipath response depends on the link geometry. Horizontal channels have a much larger multipath

spread than vertical channels. The extent of the spreading is highly dependent on depth and transmitter-to-receiver range.

The extent of multipath propagation within a given acoustic channel can be easily visualized with the help of ray traces and the channel impulse response. Figure 10 shows two sound-speed profiles collected near the gateway node in June 2008 Halifax trial environment.

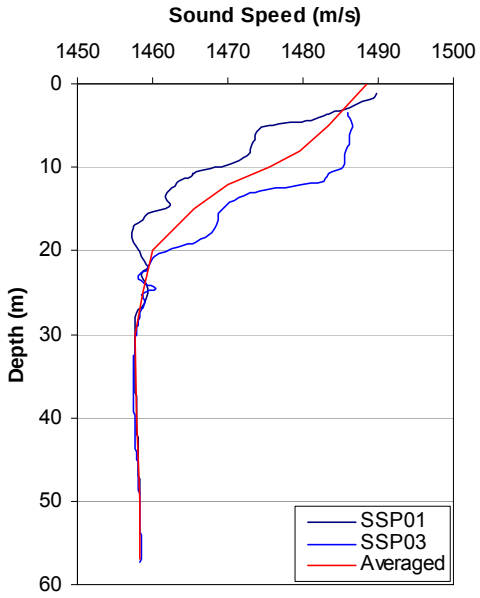
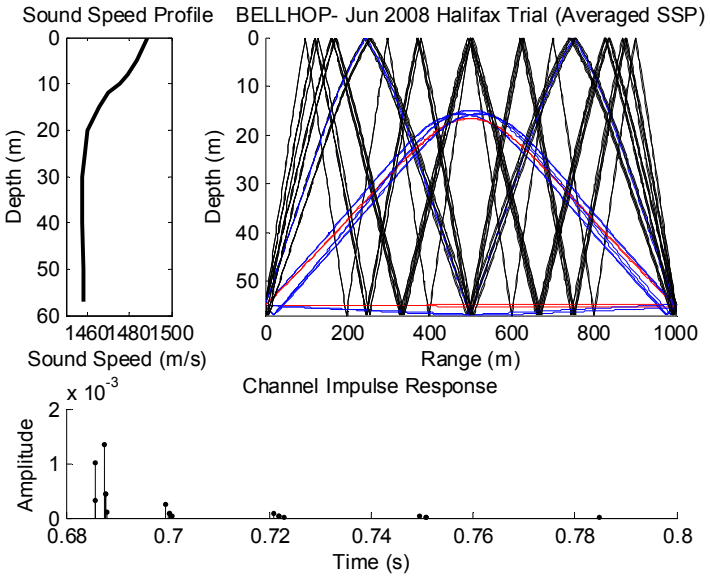
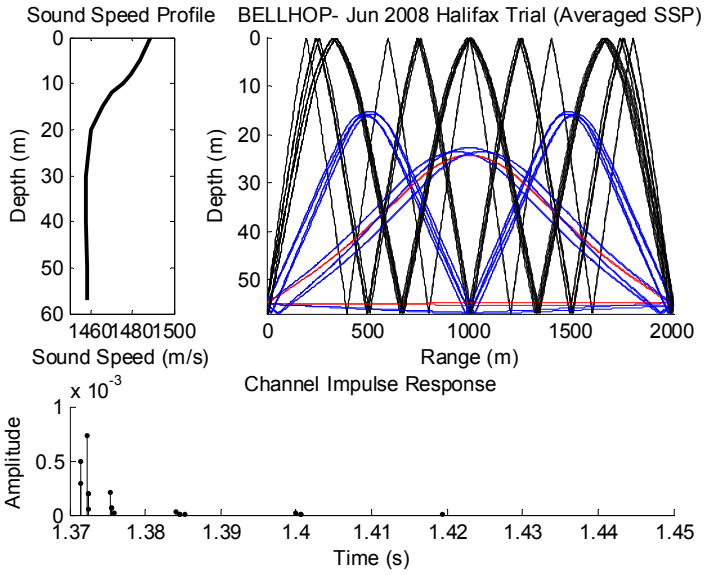


Figure 10. Sound-speed profiles from St Margaret's Bay (near the Seaweb network gateway node).

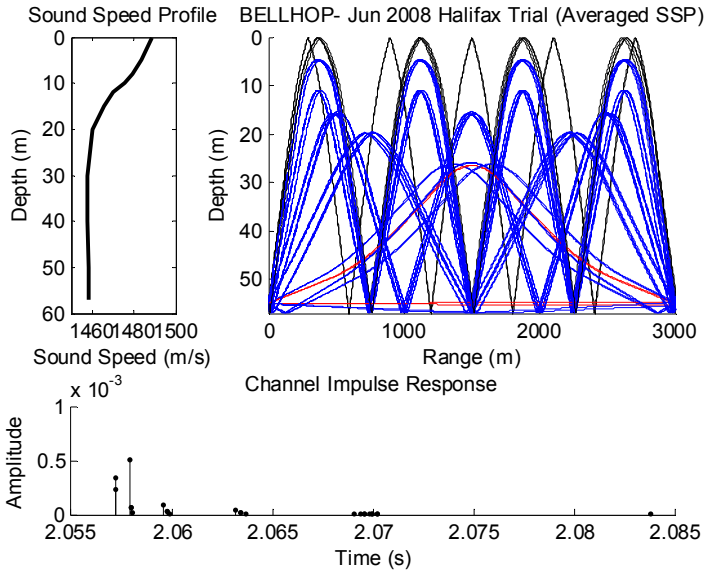
The average sound-speed profile is used to compute the eigenray traces and channel impulse response using code from Torres [18] that employs a Bellhop Gaussian beam tracing acoustic propagation model. Figure 11 depicts the eigenray traces and channel impulse response for the Halifax trial environment with a transmit frequency of 12 kHz, water depth of 57 m, source and receiver depths of 55 m, and varying source-to-receiver ranges of 1, 2, 3 and 4 km. The plots show a downward refracting channel that allows direct-path propagation. Multipath arrivals are the result of surface and bottom reflections.



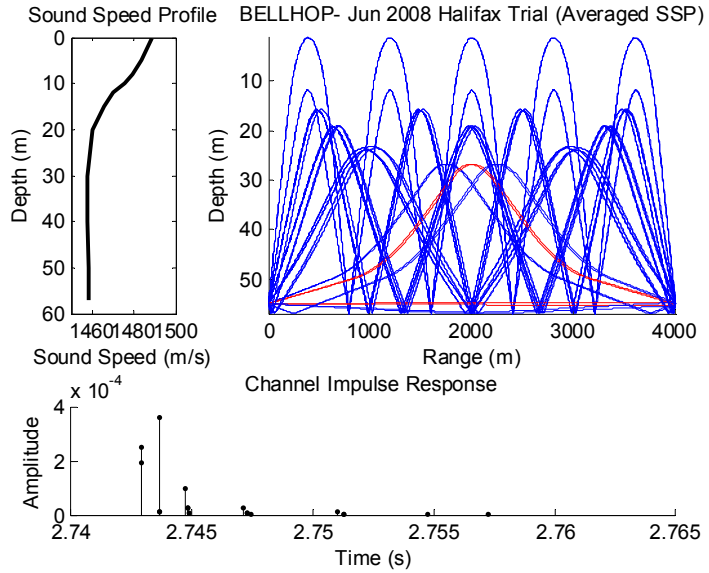
(a) Source-to-receiver range = 1 km



(b) Source-to-receiver range = 2 km



(c) Source-to-receiver range = 3 km



(d) Source-to-receiver range = 4 km

Figure 11. Bellhop eigenray traces for June 2008 Halifax trial depicting a downward refracting channel with multipath propagation. Direct-path arrivals are in red.

The combined effects of transmission loss, noise, and multipath arrivals make communicating in the underwater acoustic channel a challenge. In order to ensure a reliable acoustic link between communicating nodes, there is a need for a robust link-layer mechanism.

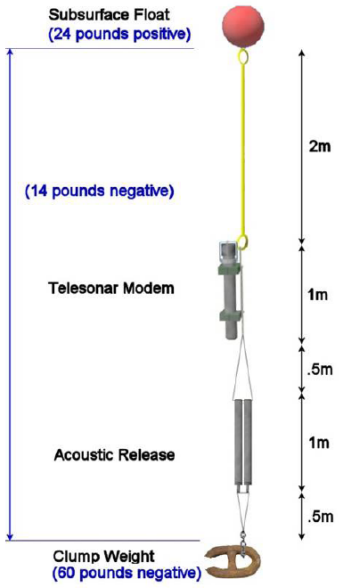
IV. SEAWEB ACOUSTIC NETWORK

This chapter discusses the Seaweb link-layer and network-layer mechanisms that play roles in the ad hoc network discovery process.

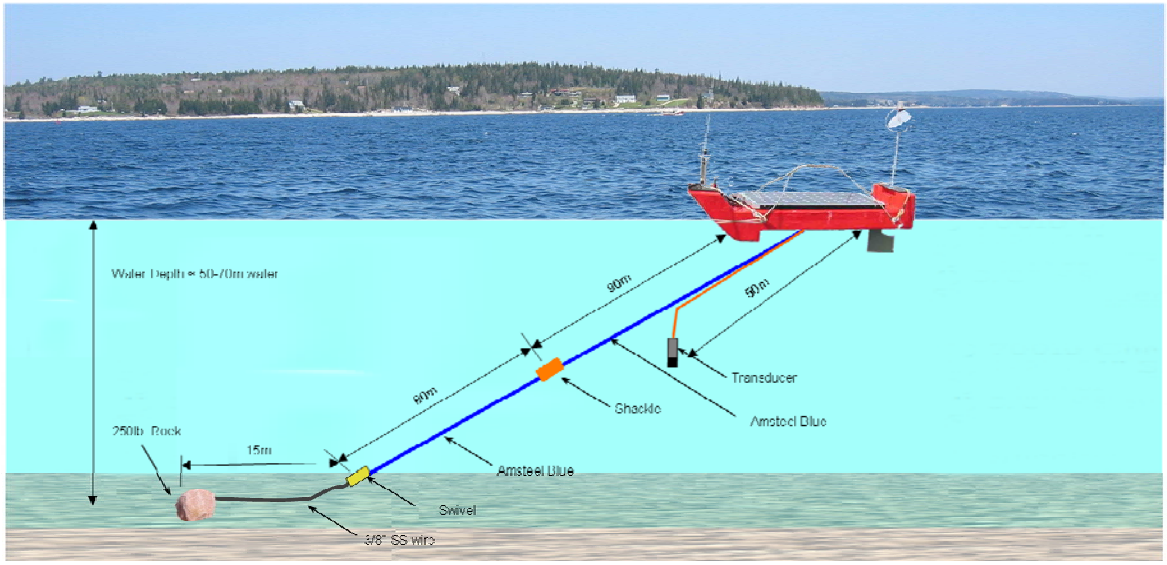
A. SYSTEM COMPONENTS

In the Unet 2008 sea trial, the Seaweb underwater acoustic network was made up of three components, namely, a set of telesonar repeater nodes, a racom gateway node, and a Seaweb server.

The telesonar repeater nodes were static nodes, each equipped with a commercial-off-the-shelf (COTS) Teledyne Benthos telesonar modem (ATM-885) loaded with proprietary US Navy Seaweb firmware. As discussed in Chapter III, the telesonar modems use half-duplex MFSK signaling in the 9-14 kHz band with variable amounts of forward error correction.



(a) Telesonar repeater node



(b) Racom gateway node

Figure 12. Schematics of a telesonar repeater node and the racom gateway node.

Seaweb gateway nodes provide an interface between the underwater network and command centers submerged, afloat, aloft, ashore and afar. The gateway node at Unet 2008 was a racom buoy. The racom buoy is a member of the underwater network and is equipped with a variety of communication links (Freewave line-of-sight packet radio, Airlink cellular modem, and Iridium satcom) to “reach back” to a command center.

A Seaweb server resides at the command center. It provides a user interface for the Seaweb operator to configure, monitor and manage Seaweb operations. It is also where incoming data packets from the submerged network can be fused as required by an application-layer protocol.

B. LINK LAYER

While the physical layer involves signaling schemes and error-correction coding, the Seaweb link layer is concerned with ensuring reliable node-to-node communications. This is achieved through the employment of compact 72-bit utility packets.

1. Handshaking Process

Establishment of a link between a pair of neighbor nodes is initiated by the *request-to-send* (RTS) and *clear-to-send* (CTS) utility packets as depicted in Figure 13. This handshaking process enables addressing, ranging, channel estimation, power control, and adaptive modulation [19]. Other link-layer features such as acknowledgements, range-dependent timers, retries, and *automatic repeat requests* (ARQ) further improve link reliability. Figure 13 further illustrates the *selective ARQ* (SRQ) link-layer mechanism for reliable transfer of large data files between neighboring nodes even when the physical layer suffers bit errors uncorrectable by forward error correction.

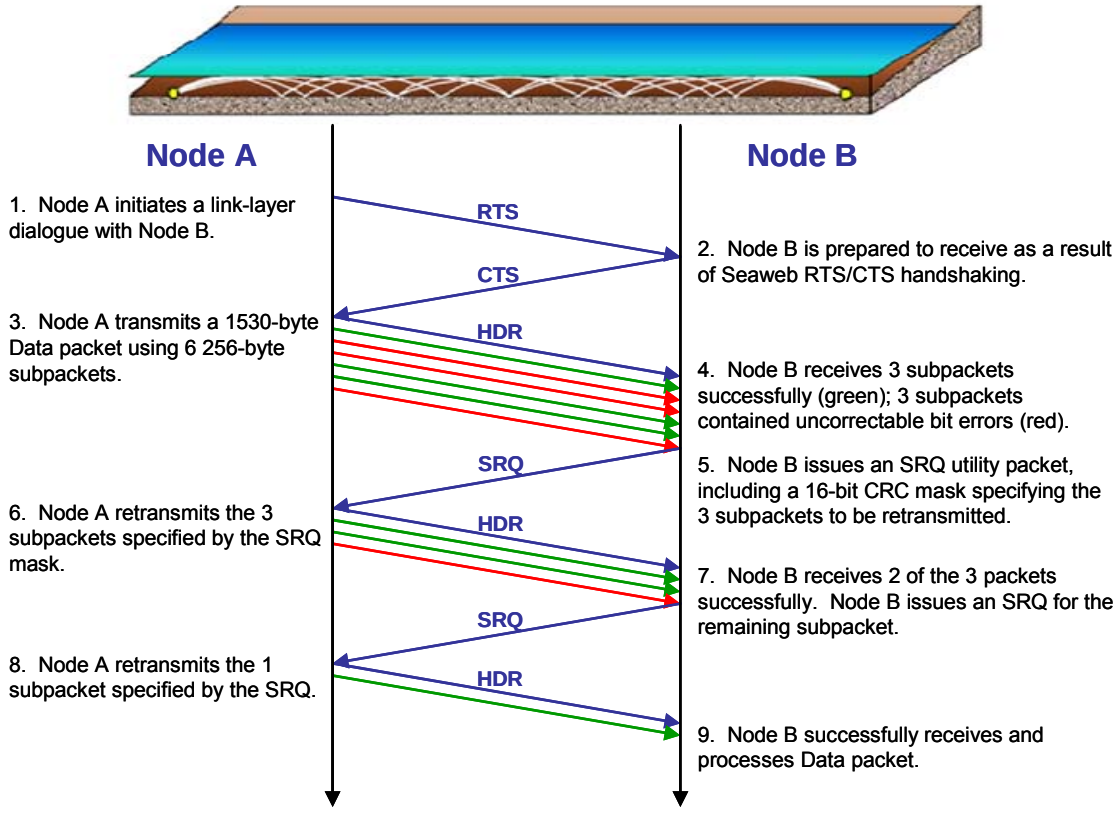


Figure 13. Seaweb link-layer SRQ mechanism. Blue arrows are Seaweb utility packets [After 20].

2. Node-to-Node Ranging and Broadcast Ping

Whenever a utility packet dialogue (such as RTS/CTS) takes place between a pair of nodes, the node-to-node range is incidentally calculated and incorporated into each node's link-layer neighbor table. The *ping* and *echo* utility packets are specifically designed to support neighbor discovery and node-to-node ranging. In order to achieve node-to-node ranging, the *ping* packet addresses the desired neighbor node [4].

When the address field of the *ping* utility packet is set to global address 0, its function becomes a *broadcast ping* and it elicits *echoes* from all neighboring nodes within earshot of this transmission. Upon receiving a *broadcast ping*, a replying node waits a certain amount of dwell time randomly chosen from a uniform distribution, whose parameters are specified in the *broadcast ping* utility packet. Randomizing the responses from neighboring nodes reduces the probability of echo collision at the eliciting node [21]. Figure 14 illustrates the process whereby a specific node is directed by the Seaweb server to issue a *broadcast ping*.

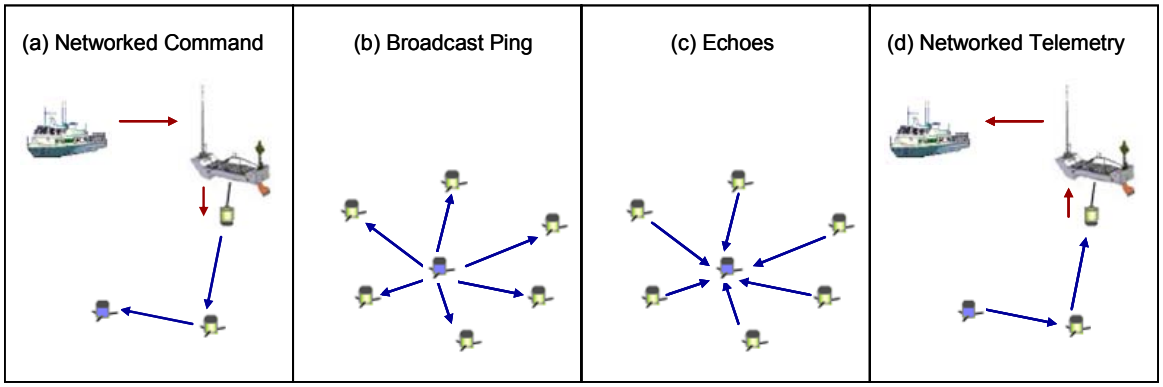


Figure 14. The *broadcast ping* process [After 21].

One-way travel time, and hence the node-to-node range (product of sound speed and one-way travel time), of any *ping* and *echo* pair of utility packets can be calculated without the need for any clock synchronization between the communicating nodes. Figure 15 illustrates this process.

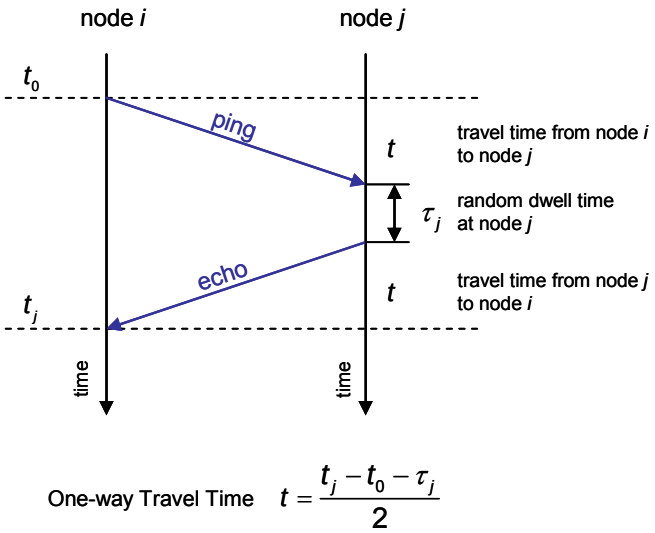


Figure 15. Seaweb node-to-node ranging process: node *i* transmit a *ping* utility packet to node *j*. Node *j* enters a random dwell time before replying with an *echo* utility packet. The dwell time is embedded in the *echo* reply from node *j* to node *i*. Upon receipt of the *echo* utility packet, node *i* computes the time elapsed between *ping* transmission and *echo* reply, and extracts the node *j* dwell time information. All time measurements are computed at node *i*. Thus, there is no need for clock synchronization [After 4].

The *broadcast ping* and node-to-node ranging features of the Seaweb link-layer protocols are exploited in the ad hoc network discovery process developed in the next chapter.

C. NETWORK LAYER

Network-layer protocols act across the network, serving to deliver communications from a source node to a destination node via some network route. Network-layer supervisory algorithms can be carried out at either an autonomous master node or at the Seaweb server [1].

1. Neighbor Tables and Routing Tables

Routing and navigation through Seaweb acoustic networks are made possible by embedded data structures distributed throughout the network. Each node maintains a local link-layer neighbor table containing information about adjacent nodes that are within a single hop range. In addition, each node stores

a local network-layer routing table indicating the neighbor nodes that have networked connectivity with the intended destination node [3].

In contrast to dynamic source routing [22] where the route is specified and explicitly declared in the network-layer header of a data packet, the routing table approach employed in Seaweb permits the network-layer header to contain only the source and destination addresses while relying on the distributed routing table to forward the data packet to the appropriate neighbor node en route to the destination.

At the Seaweb server, a global neighbor table and a global routing table are maintained to support network configurability.

2. Cellular Addressing

In keeping with the compact utility packet format of Seaweb, it was determined [23] that additional functionality was possible with the addition of just one more network-layer field in the utility packet, thus increasing the Seaweb utility packet size from 8 to 9 bytes. The additional field is a "cellular address" which enables delivery according to the established routing table to the cellular address whereupon a final network-layer link from the cellular address to the destination node is performed. Cellular addressing supports communications to a mobile address in the vicinity of a cellular node and may also be invoked for peer-to-peer communications or other network-layer addressing not already permitted by the routing tables [23].

3. Network Initialization

Seaweb network initialization is a process of populating the distributed local neighbor tables and local routing tables of the deployed Seaweb nodes so as to enable network connectivity. While neighbor tables can be populated through the use of the *ping* and *echo* utility packets, prior iterations of Seaweb networks have relied exclusively on the Seaweb operator to manually configure network routes from a master node (usually the gateway node) to all other nodes

in the network. Such an initialization process is unsuitable for a spontaneously deployed, ad hoc network of autonomous sensors where the number, addresses and deployed locations of nodes may not be known a priori to the operator.

The next chapter attempts to address this network initialization issue through the design of an ad hoc network discovery process that builds upon the existing link-layer and network-layer mechanisms of Seaweb.

THIS PAGE INTENTIONALLY LEFT BLANK

V. AD HOC NETWORK DISCOVERY PROCESS

Prior iterations of Seaweb networks have relied on the Seaweb operator to initialize and configure network routes from master node to all other nodes in the network. This was accomplished through the manual specification of the local routing tables distributed throughout the network.

This chapter proposes a network discovery process for initializing an ad hoc Seaweb acoustic network following spontaneous deployment of autonomous nodes, or where operator knowledge of the number of deployed nodes is limited or precluded. Such a network is expected to be capable of discovering member nodes post-deployment, and be able to auto-configure for networking purposes.

As briefly mentioned in Chapter II, this discovery process is centrally controlled by a node designated as the master node.

Upon completion of network discovery, there should be valid routes from the master node to all discovered nodes in the network and vice versa. Nodes needing to communicate with each other can use the master node as a hub.

A. TOPOLOGY SEARCH METHODS

Given a distribution of fixed nodes, there are essentially two basic search techniques for discovering directed paths (routes) from one specific root node (master node) to each node in the network – breadth-first search, and depth-first search [24].

1. Breadth-First Search

In breadth-first search, the algorithm starts at the master node and finds all the neighboring nodes. Then for each of those nearest nodes, it finds all their unexplored neighbor nodes, and so on, until there are no more unexplored nodes in the network.

2. Depth-First Search

In depth-first search, the algorithm starts at the master node and explores along a path as far as possible before backtracking one level up to search along the next path. Figure 16 illustrates the difference between the two search techniques.

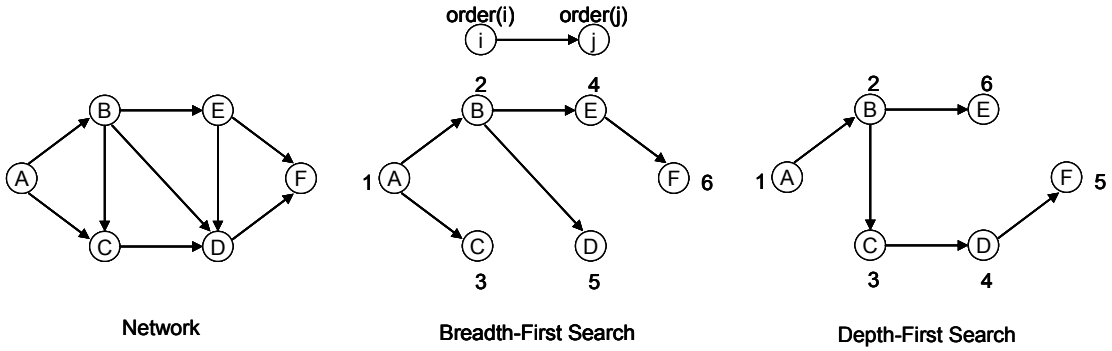


Figure 16. Two search techniques for a deterministic network – Breadth-first search vs Depth-first search [After 24].

3. Comparison

In breadth-first search, the path (route) from the master node to any node in the network is always *a shortest path* [24] in that it contains the fewest number of arcs (links) among all possible paths joining the two nodes. The same cannot be guaranteed when using the depth-first search technique, e.g., in Figure 16 the breadth-first search route from node A to node F contains three hops (A-B-E-F) whereas the depth-first search route contains four (A-B-C-D-F).

The runtime complexity of both search techniques is on the order of the sum of the number of nodes and the number of arcs in the entire network [24].

Given that the theoretical runtime complexities of the two search algorithms are the same, the breadth-first search technique will be adopted for the network discovery process since it is guaranteed to produce a route with the fewest number of hops from the master node to any other specified node.

B. CENTRAL CONTROL

Chapter II alluded to the use of a master node to initiate and control the network discovery process. Such a centrally controlled scheme allows the master node to actively monitor the progress of the discovery process as it proceeds. The master node is also able to proactively terminate the process if certain pre-determined conditions are met. Another advantage of central control is that only the master node's processing ability needs to be enhanced to handle the envisaged complexity of the ad hoc network discovery process, as opposed to upgrading all Seaweb nodes. The latter advantage has been important during the development and sea testing of this process, as software changes need only be applied at the master node. Existing Seaweb functionalities covered in Chapter IV, such as broadcast ping and its echo response, distributed neighbor tables and routing tables, are also exploited.

C. DESCRIPTION OF THE NETWORK DISCOVERY PROCESS

Since the master node centrally controls the network discovery process, all other nodes (branch nodes) in the network await instruction from the master before performing peer discovery or local routing table updates. Peer discovery involves each node in the network issuing broadcast pings to elicit echoes from its neighboring nodes. The branch node reports results back to the master node upon completion of each activity. The master node aggregates the received peer discovery data in a global neighbor table and ultimately decides how routing to each branch node should be configured. The routing tables are then distributed out to the branch nodes.

At the end of the discovery process, there should be valid routes from the master node to all discovered nodes in the network and vice versa. These routes may not be optimal for networked connectivity between any two given branch nodes, but they can always communicate with each other by using the master node as a hub [25].

Figure 17 represents a hypothetical node deployment for illustrating the ad hoc network discovery process.

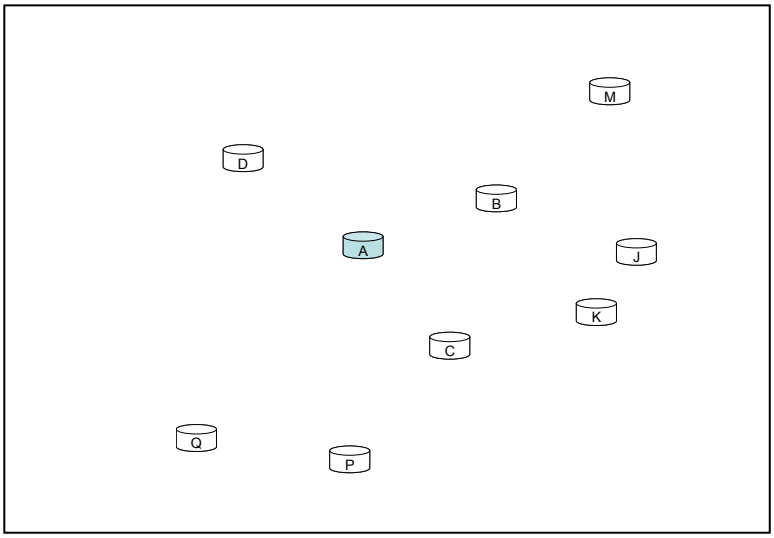


Figure 17. Hypothetical node deployment for illustrating the ad hoc network discovery process. Node A is the master node.

1. Master Node Discovery

When an arbitrarily designated master node receives a network discovery command from the Seaweb server, it issues a specified number of broadcast pings at a specified power level to discover its neighbors. The broadcast ping is repeated a user-specified number of times to mitigate the possibility of echo collisions, where the echoes from two or more neighbor nodes arrive at the master node at the same time. The results (neighboring node addresses and ranges) from these broadcast pings are aggregated in a global neighbor table that the master node uses to determine routing for the network. The global neighbor table resides only at the master node. The functionality of local neighbor table at each node, as described in Chapter IV, remains unchanged.

At the end of its immediate neighbor discovery, the master node's global neighbor table is filled with range and address data for all the branch nodes it discovered. Figure 18 depicts the nodes discovered by master node A.

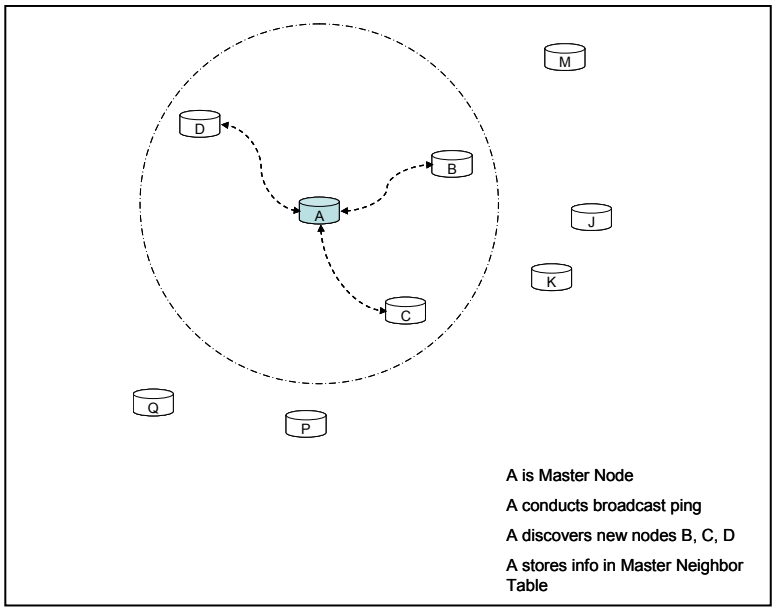


Figure 18. Master node A discovery. Node A conducts broadcast ping and discovers new nodes B, C, and D. Node A stores neighbor information in the master neighbor table.

2. Branch Node Discovery

Upon completion of master node discovery of its neighbors, the master node determines who are its immediate peers by applying a range cutoff, which is specified as an argument in the initial network discovery command. The master node then sequentially assigns bi-directional routes to these branch nodes, and commands each of them to perform peer discovery. The master node always operates on the nearest unprocessed branch node, establishing a route and then performing peer discovery.

Each branch node performing peer discovery of its neighbors also uses the same user-specified power level and performs the same specified number of broadcast pings. All discovery pings at branch nodes are explicitly commanded from the master node.

As each branch node discovers its neighbors, the results are sent back to the master node. As previously mentioned, the master node aggregates and stores the range and address information in a global neighbor table.

Figures 19-27 illustrate the rest of network discovery process as it cascades through the hypothetical network, alternating between routing and peer discovery, and following a breadth-first search scheme.

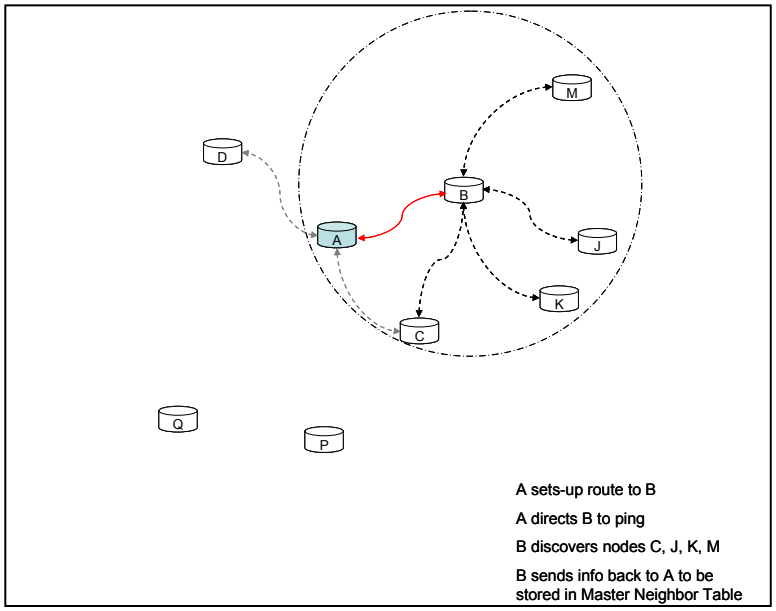


Figure 19. Master node establishes route (red arrow) to its nearest neighbor node B and directs it to perform peer discovery.

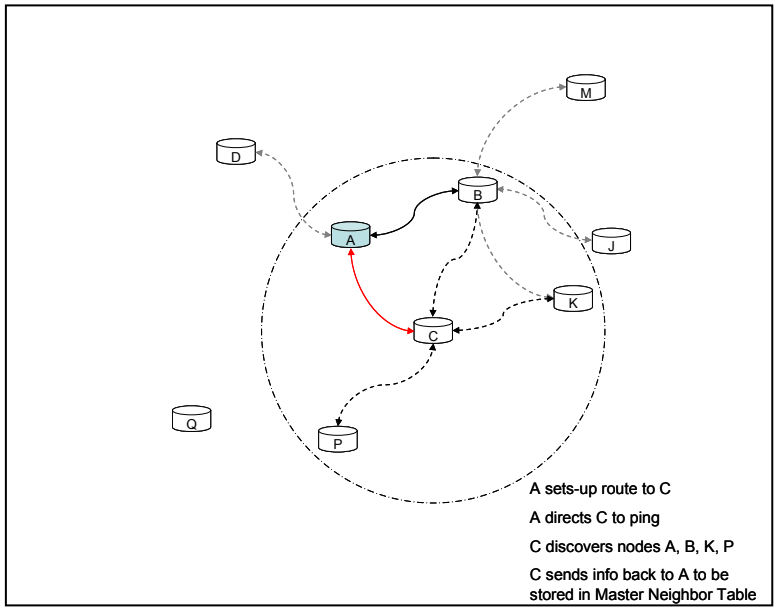


Figure 20. Master node establishes new route to next nearest node C and directs it to perform peer discovery.

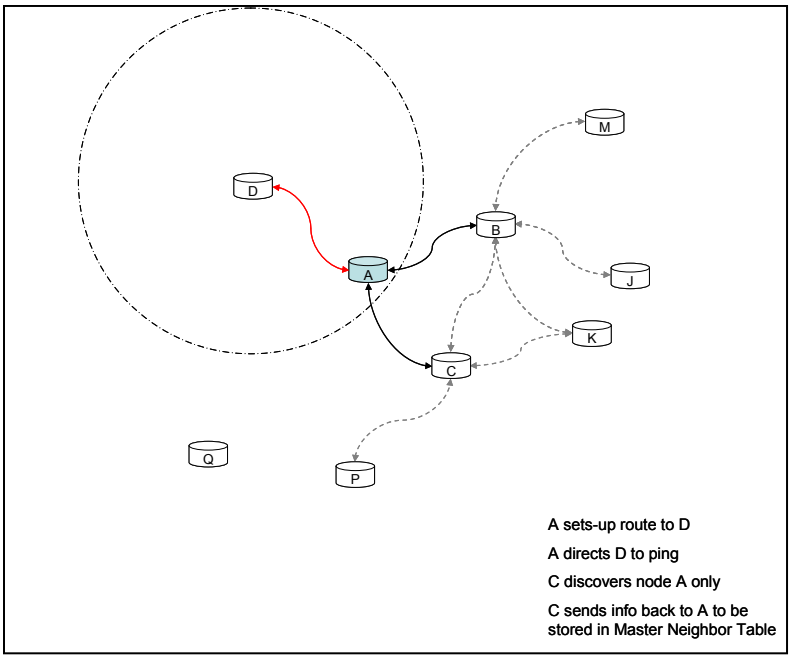


Figure 21. Master node establishes new route to node D and directs it to perform peer discovery.

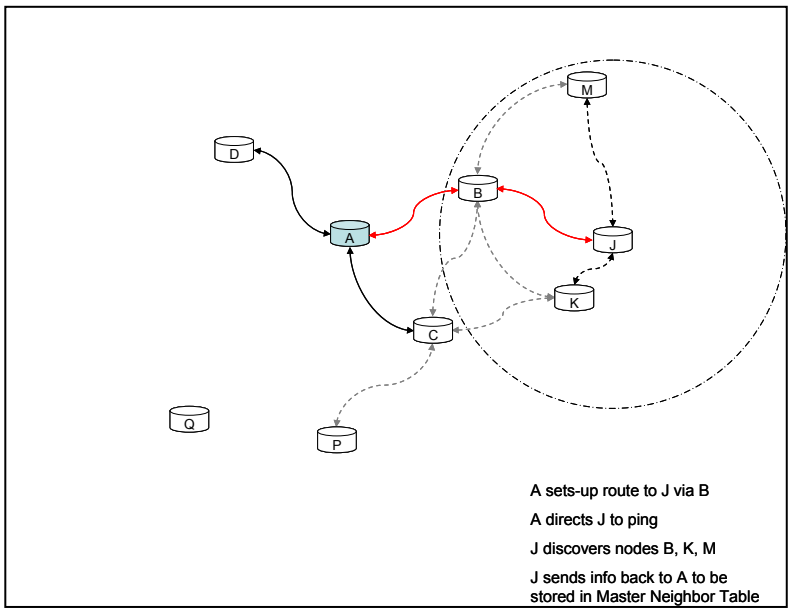


Figure 22. Master node establishes new route to next nearest node J and directs it to perform peer discovery. Route to J goes through node B.

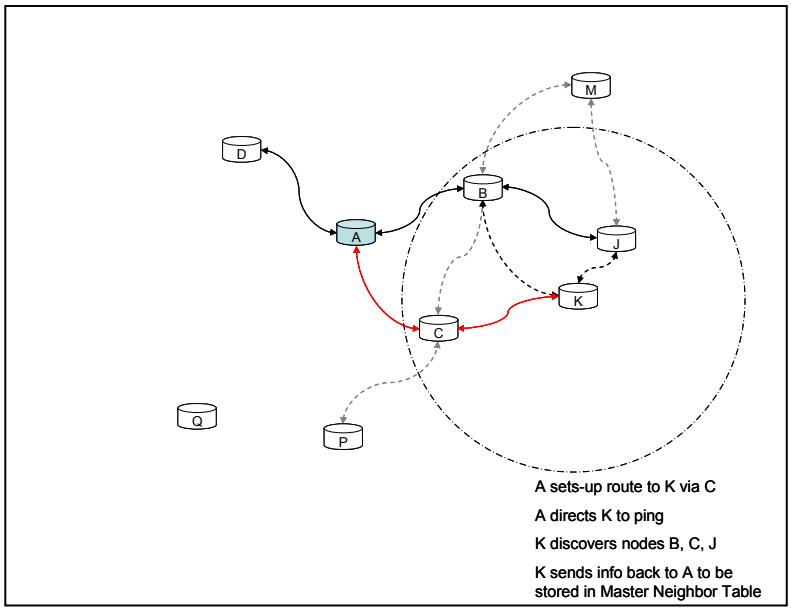


Figure 23. Master node establishes new route to next nearest node K and directs it to perform peer discovery. Route to node K goes through node C instead of node B, based on the route cost evaluation function.

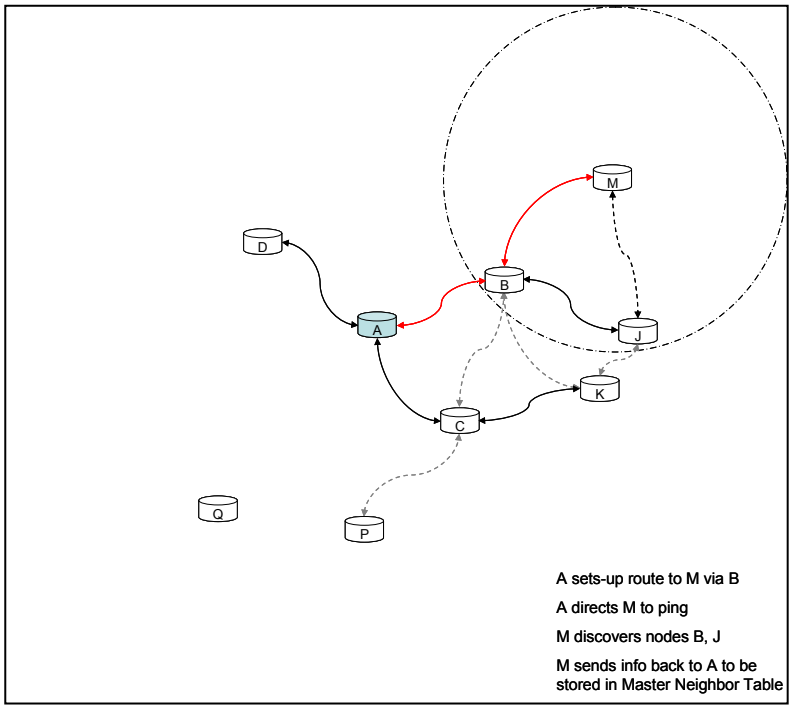


Figure 24. Master node establishes new route to node M via node B, and directs it to perform peer discovery.

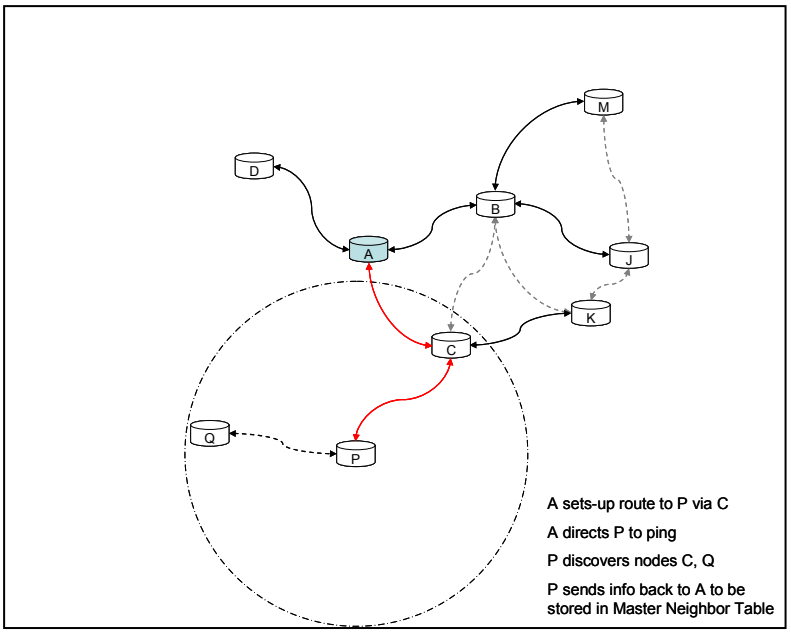


Figure 25. Master node establishes new route to node P via node C, and directs it to perform peer discovery.

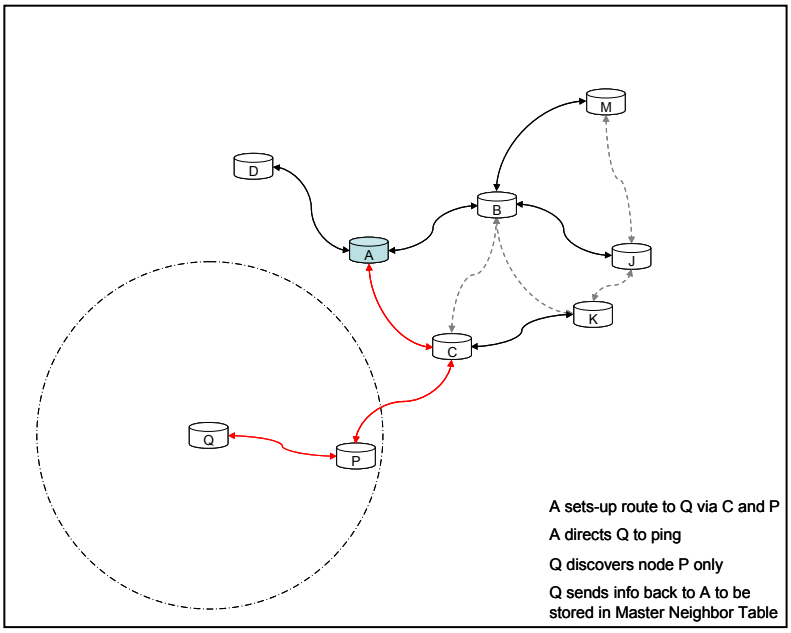


Figure 26. Master node establishes new route to node Q via nodes C and P, and directs it to perform peer discovery. No more new nodes are discovered and the network discovery process terminates.

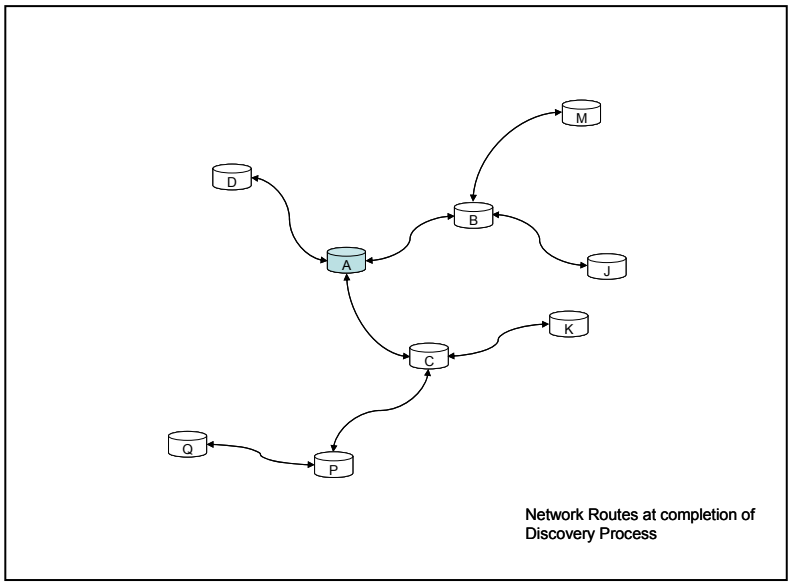


Figure 27. Resultant bi-directional routes from master node to all discovered nodes in the network upon completion of the ad hoc network discovery process.

3. Master Node to Branch Node Routing

Routing from the master node to a branch node is done in tandem with the network discovery process. When broadcast ping results return to the master node from a branch node, the master examines the results to determine if the branch node can see any other nodes that the master is not yet aware of, i.e., newly discovered nodes. This essentially expands the network that the master node has knowledge of. In addition, the master node needs to examine the global neighbor table and establish routes to branch nodes, using a route cost evaluation function, as they are discovered. When selecting a branch node to be routed to, the master node always chooses the next nearest branch node it has knowledge of. Route establishment and assignment to a branch node is done before the master node commands it to perform peer discovery. Figures 19-27 in the preceding pages depict the route establishment process being executed in tandem with the network discovery process[1].

[1] The route establishment process is very similar to Ad hoc On-Demand Distance Vector (AODV) routing protocol except that AODV does not report the route back to the originator (master node) until the shortest path reaches the destination [29].

4. Route Selection

When presented with multiple routing options to a specific branch node, the master node uses the route cost evaluation function to choose the lowest cost route. For example, in Figure 23, when attempting to establish a route to node K, the master is presented with two routing options – A-B-K and A-C-K. The final route selection (A-C-K) is the route with the lowest cost given by the route cost evaluation function.

5. Route Evaluation

In determining the lowest cost route to a branch node in the network, a variety of route cost criteria and associated cost functions can be used – hop count, path length, message delivery latency, transmission security, power consumption, network longevity, and message delivery reliability [26]. This implementation of the ad hoc Seaweb network discovery process employs a cost function empirically derived with two principal factors taken into consideration – preferred hop range, and the number of hops taken to reach the specific branch node.

a. Route Cost Evaluation Function

The empirically derived route (path) cost function is given by

$$C_i = \sum_{j=1}^{h} \left[\left(\frac{r_j - r_p}{r_p} \right)^2 + \left(\frac{r_j}{r_p} \right) \right]$$

where C_i is the path cost from the master node to node i,

h is the number of hops in the path,

r_j is the range between nodes in the j^{th} hop of the path, and

r_p is the preferred hop range.

The first term in the above cost function corresponds to the penalty associated with the variance from a user-specified preferred hop range (described in more detail below), and the second term is the penalty associated with the distance of the hop.

For an *N*-node network, the corresponding total network cost is the sum of all the *N*-1 path costs in the network, normalized by a factor that is a function of the preferred hop range and the range cutoff, and is defined as

$$C_{NETWORK} = \sum_{i=1}^{N-1} \left[\left(\frac{r_p}{r_c} \right)^2 C_i \right]$$

where $C_{NETWORK}$ is the total network cost,

C_i is the lowest path cost from master node to node *i*,

$N-1$ is the number of paths in the *N*-node network,

r_p is the preferred hop range, and

r_c is the range cutoff.

b. Preferred Hop Range

The preferred hop range (r_p) is a user-specified range value that roughly corresponds to the desired length of node-to-node links for the network. A small preferred hop range, relative to the range cutoff, causes the routing algorithm to choose a route containing more short-distance hops, which is a desired routing strategy if transmission security or power consumption were used as a route cost criterion [26]. On the other hand, a larger preferred hop range causes the routing algorithm to choose a route containing more direct, long-distance hops which reflects a routing strategy with hop count, path length, or latency as a cost criterion. When reliability is the primary criterion, the preferred hop range should represent the most desirable node-to-node range given the prevailing propagation conditions, expected noise levels, and performance expectations for the network.

c. Range Cutoff

Range cutoff (r_c) is a user-specified range beyond which any node discovered will not be considered as an immediate neighbor. Such a node must be reached via hops from a nearer neighbor. Range cutoff corresponds to the largest acoustic communications range supported by the propagation conditions of the channel. Obviously, the lower bound on range cutoff must be the shortest node-to-node spacing in the network; otherwise, no nodes will be discovered by the master node. A good estimate of the range cutoff can be obtained from the channel SNR plots illustrated in Figure 9 of Chapter III. For a wind speed of less than 5 m/s, a range cutoff of 4 km is chosen.

d. Handicapped Nodes

Certain nodes that are at an acoustic disadvantage, e.g., on the other side of a thermocline or at a different depth than most other nodes, can be flagged as handicapped nodes. When designated as a handicapped node, the preferred hop range associated with immediate links to and from that node is halved. The net result is that the route cost evaluation function favors a shorter hop to and from the handicapped node. In most Seaweb deployments, the master node is the racom gateway buoy with its transducer at a shallower depth than the rest of the nodes in the network. Thus, the master node is usually identified as a handicapped node.

6. Runtime Complexity

During the network discovery process, each node is directed to perform peer discovery once. In addition, as each peer discovery result is returned to the master node, the master processes the existing aggregated information in the master neighbor table in order to determine routing to the next node. In an *N*-node network, peer discovery is performed exactly *N* times. In a fully connected scenario where all the *N* nodes are interconnected such that each node has *N*-1 neighbors, the master node needs to process *N*-1 sets of neighbor information each time in order determine routing to the next node. Therefore, the theoretical

runtime complexity of the network discovery process for an *N*-node network is on the order of $O(N(N-1))$ or approximately $O(N^2)$.

7. Frequency of Network Discovery

Variability in underwater acoustic propagation and noise requires consideration of channel availability when specifying the preferred hop range. For example, if 90% channel availability is desired, the preferred hop range should correspond to a range meeting or exceeding 90% statistical availability of an adequate receiver signal-to-noise ratio (SNR) for the time-varying channel. To mitigate longer-term degradation of the channel, the network discovery process needs to be repeated at a regular interval so as to update existing network routes in a manner that adapts to the prevailing channel conditions in the evolving medium. Adaptation may be achieved by adjusting the preferred hop range in a feed-back manner according to recent performance statistics. Network discovery may also be event-triggered, such as when member nodes suffer outages by battery depletion or when new nodes are introduced to the network. The discovery update should fully utilize existing neighbor data along with accumulated performance statistics for each link. The frequency of periodic network discovery update is largely dependent on the long-term rate of change in the acoustic channel (as indicated by variations in the sound-speed profiles and in the ambient noise levels) where the nodes are deployed. Although this thesis does not specifically address the frequency with which network discovery needs to be repeated, it is practical to conduct a network discovery update on the order of once per day.

D. SIMULATION

The ad hoc network discovery process is implemented in C language for simulation purposes. Graphing and plotting of simulation results are done in MATLAB. A program flowchart is given in Figure 28 and the program source code is appended in Appendix A.

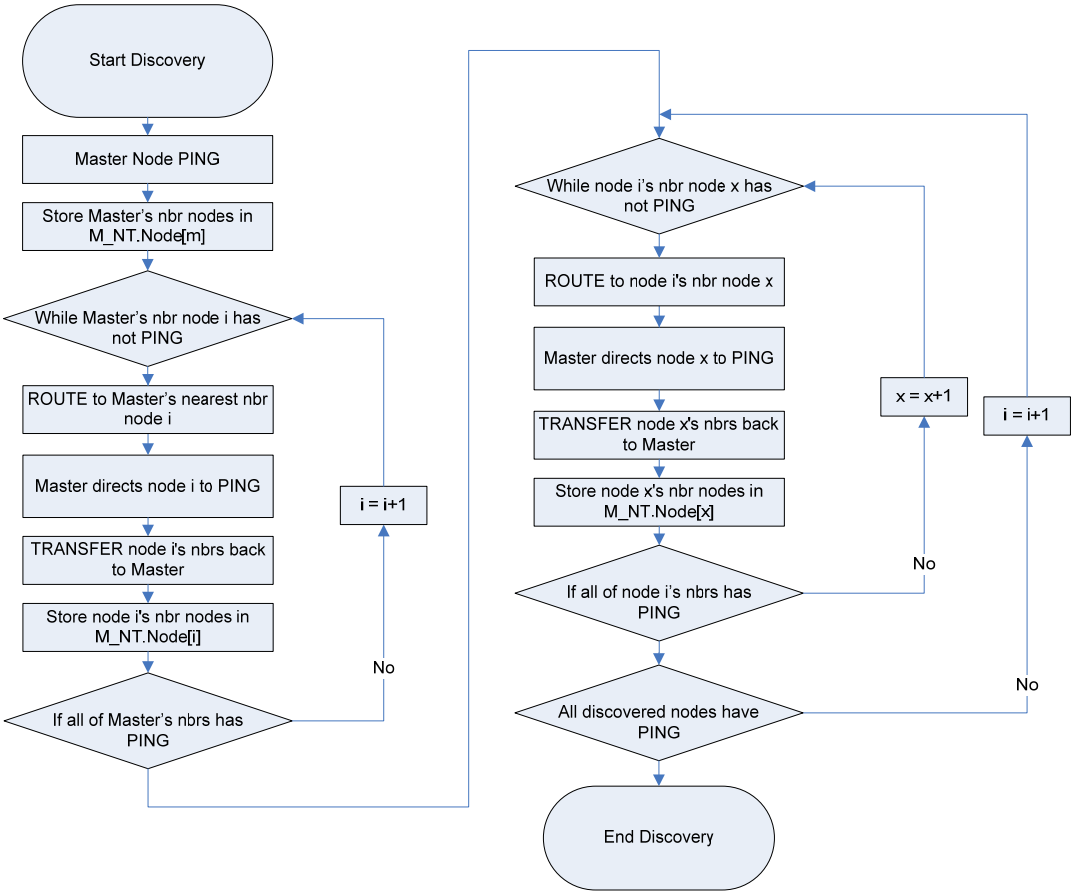
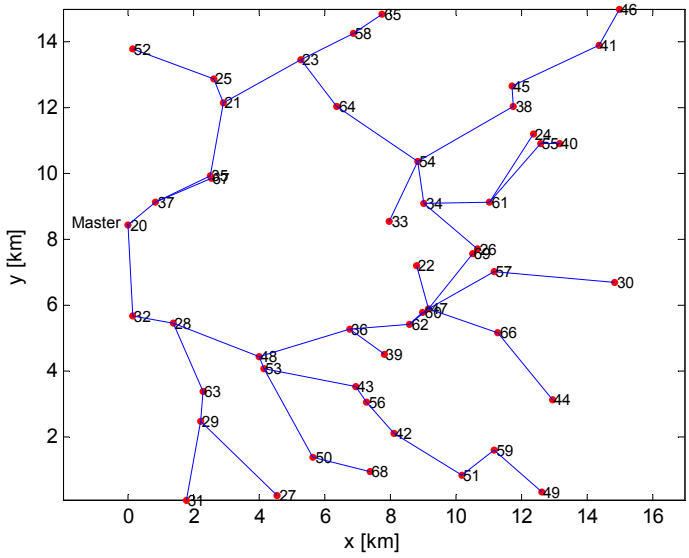
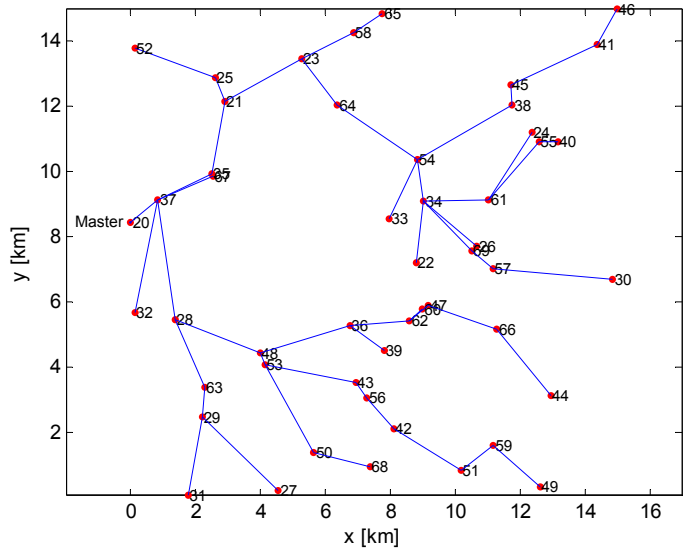


Figure 28. Program flowchart for the network discovery process implemented in C language.

Figures 29, 30 and 31, respectively, show simulation results for a network of 50 nodes randomly distributed in a 15 km by 15 km area to illustrate the effects of handicapped node, range cutoff, and preferred hop range on the resultant network routes. Node 20 is the master node from which network discovery is initiated. Simulation parameters held constant are: single broadcast ping at each node, and a ping power level corresponding to a reliable communications range of 4 km.

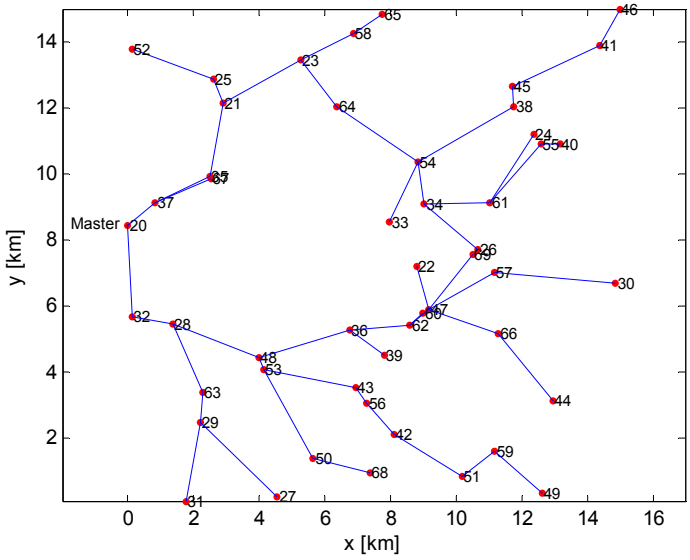


(a) Without handicap at master node 20

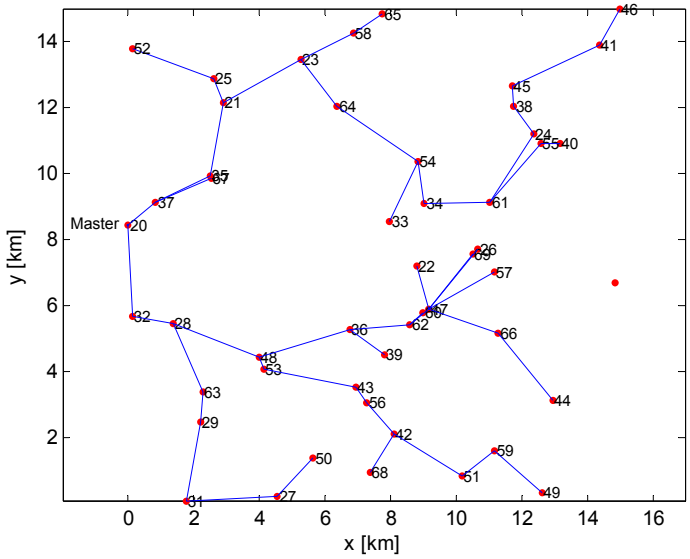


(b) With handicap at master node 20

Figure 29. Simulation results illustrating the effect of a handicapped master node on the resultant network routes. Range cutoff r_c = 4 km, preferred hop range r_p = 1 km, without handicap (top) and with handicap (bottom) at master node. Shorter hops are favored at the handicapped node.

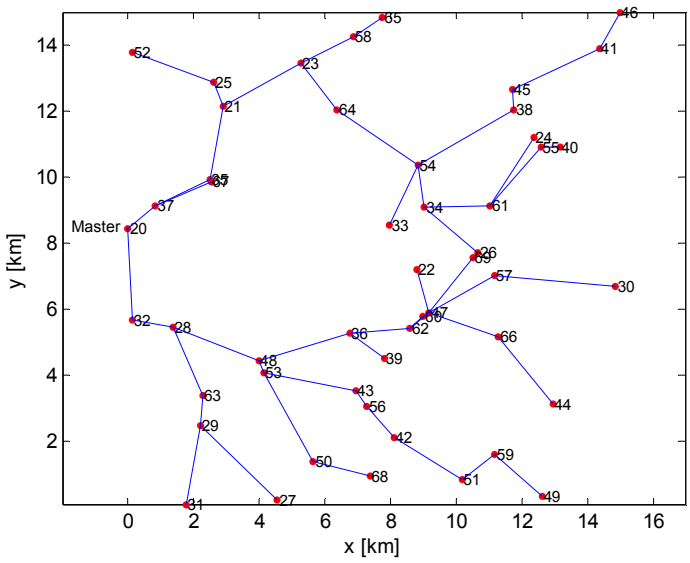


(a) Range cutoff r_c = 4 km

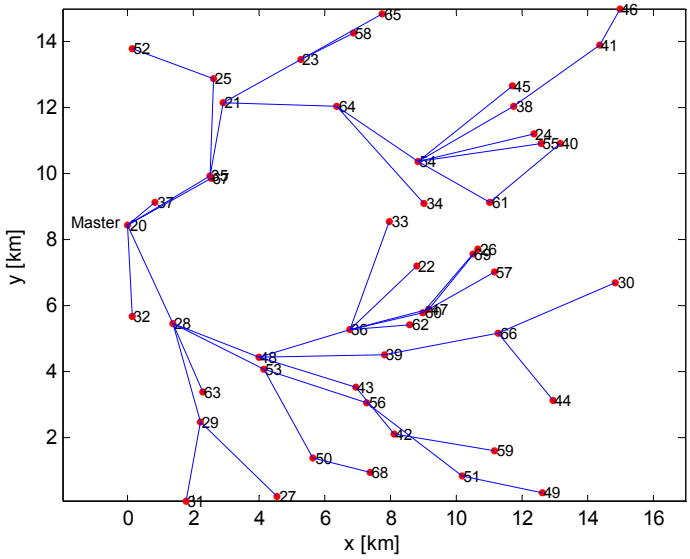


(b) Range cutoff r_c = 3 km

Figure 30. Simulation results illustrating the effect of range cutoff on the resultant network routes. Range cutoff r_c = 4 km (top) and r_c = 3 km (bottom), preferred hop range r_p = 1 km, without handicap at master node. One node is not discovered when the range cutoff is 3 km.



(a) Preferred hop range r_p = 1 km



(b) Preferred hop range r_p = 3 km

Figure 31. Simulation results illustrating the effect of preferred hop range on the resultant network routes. Range cutoff r_c = 4 km, preferred hop range r_p = 1 km (top) and r_p = 3 km (bottom), without handicap at master node. A larger r_p results in routes with more direct, long-distance hops.

VI. SEA TRIAL RESULTS AND FOLLOW-ON ANALYSIS

In parallel with the design and simulation of the ad hoc network discovery process at the Naval Postgraduate School, Teledyne Benthos was contracted to implement the same network discovery scheme as a firmware upgrade for existing Seaweb modems.

A Seaweb ad hoc network discovery experiment was conducted in June 2008 in St Margaret's Bay, Halifax, Nova Scotia, Canada. The experiment was conducted as part of the Unet 2008 sea trial. Refer to Chapter II for a description of the location and trial environment.

A. TRIAL SETUP

A total of 19 nodes were utilized for the purpose of the network discovery experiment. Table 1 lists the GPS coordinates of the deployed nodes and Figure 32 depicts the location of these nodes. Node 3 is the racom gateway buoy and it is also the master node from which network discovery is initiated. While any node can be designated as the master, use of the racom gateway node permits constant monitoring of the discovery process, the cost function evaluations, and the global routing tables. The Seaweb server resided onboard a Canadian Forces auxiliary vessel (CFAV Quest) with multiple radio communication links to the racom buoy.

Node ID	**Position** (ddmm.mmm)		**Node ID**	**Position** (ddmm.mmm)	
3	4435.609N	6359.712W	43	4436.713N	6358.396W
16	4435.400N	6359.500W	44	4437.072N	6358.393W
19	4435.279N	6400.633W	45	4437.347N	6358.483W
20	4435.870N	6359.810W	46	4435.639N	6359.253W
21	4436.350N	6359.900W	48	4434.302N	6359.727W
22	4436.850N	6359.860W	50	4435.747N	6400.316W
23	4437.340N	6359.710W	51	4436.468N	6400.629W
24	4437.810N	6359.440W	52	4437.097N	6400.689W
41	4435.790N	6358.580W	53	4437.694N	6400.904W
42	4436.270N	6358.170W			

Table 1. GPS coordinates of 19 nodes involved in June 2008 Seaweb ad hoc network discovery experiment.

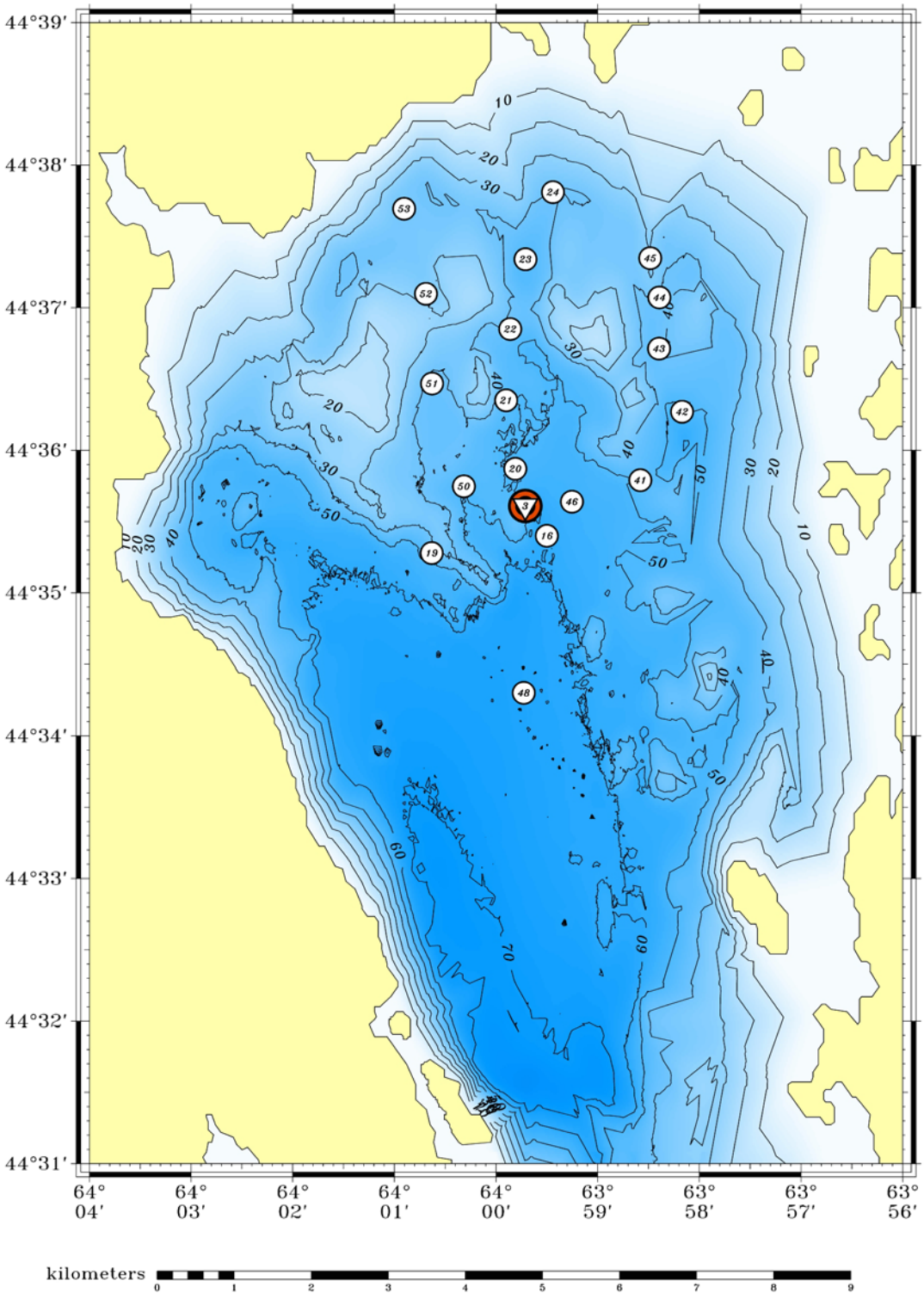


Figure 32. Location of 19 Seaweb nodes involved in the ad hoc network discovery trial. Node 3 is the racom gateway buoy and the master node.

Figure 33 shows photographs taken from the actual trial depicting the three components of a Seaweb network – a Seaweb server, a racom gateway buoy, and a telesonar repeater node. Refer to Chapter IV for a more detailed description of these components, including deployment configurations given in Figure 12.

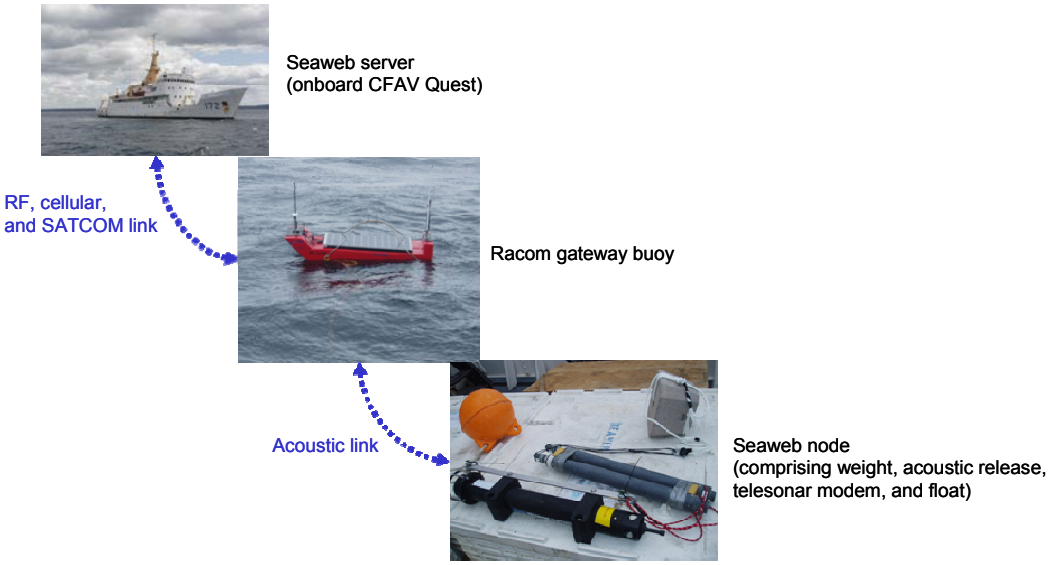


Figure 33. Three components of a Seaweb network – Seaweb server, racom gateway buoy, and repeater node.

B. TRIAL RESULTS

The ad hoc network discovery experiment was conducted over several days using varying numbers of Seaweb nodes. For the purpose of analysis and comparison in this chapter, only the 24 Jun 2008 trial results involving all 19 nodes are presented. The network discovery parameters are listed in Table 2.

Parameter	Value
Master Node	Node 3
No. of Broadcast Ping	1 per peer discovery
Ping Power Level	4 km (equivalent)
Range Cutoff	4 km
Preferred Hop Range	1 km
Handicapped Nodes	Node 3

Table 2. Network discovery parameters used on 24 June 2008.

Upon receipt of the ad hoc network discovery command and the user-specified parameters from the Seaweb server, the master node initiated network discovery in accordance with the process described in Chapter V. The resultant network routes are presented in Figure 34.

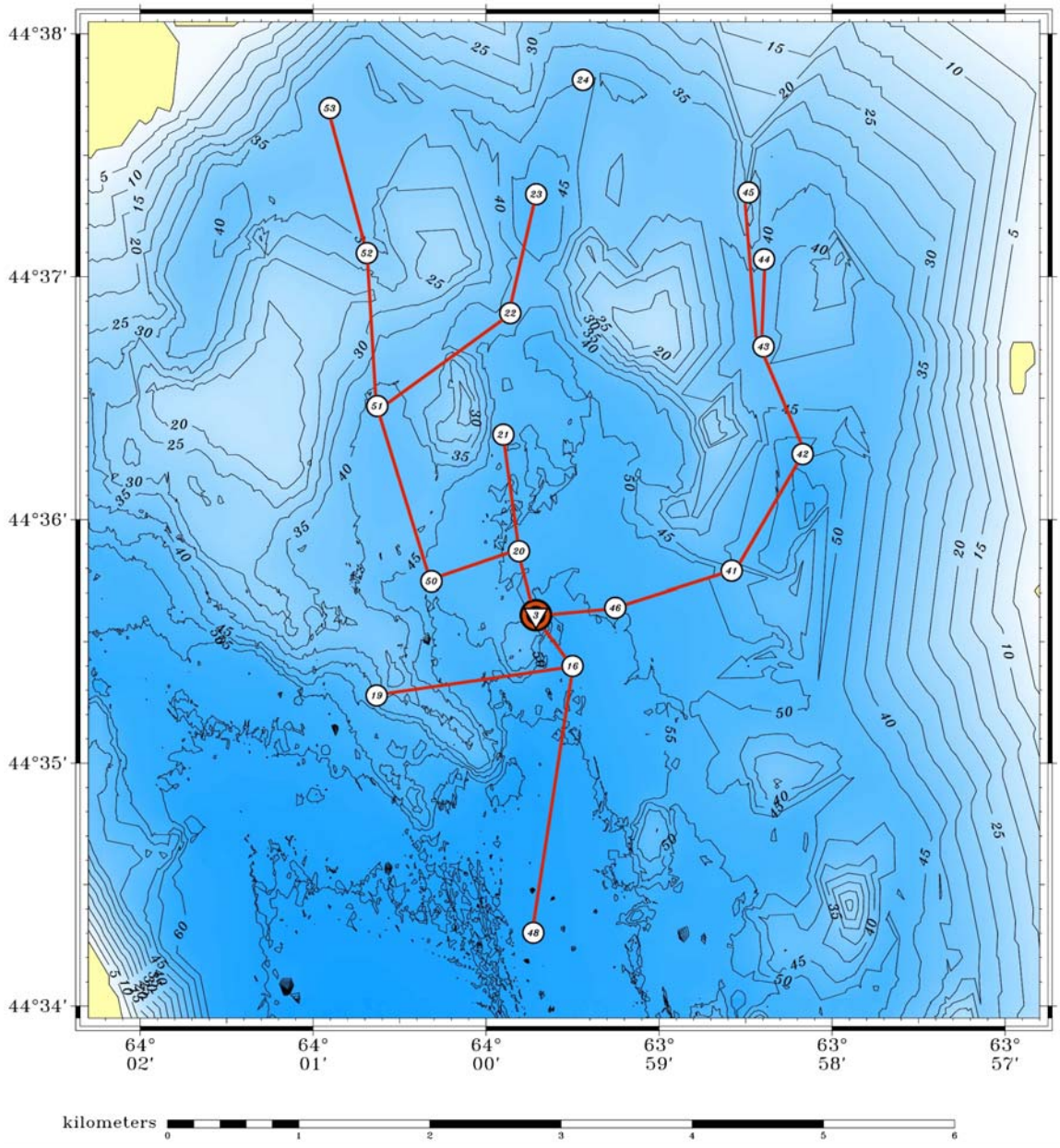


Figure 34. Resultant network routes upon completion of network discovery.

C. SIMULATION RESULTS

Using the node deployment positions in Table 1 and the network discovery parameters in Table 2, the discovery process is simulated using the previously described computer model. The node GPS coordinates are converted into a Cartesian coordinate 2-D plane [27] and depth variations associated with the deployed nodes are neglected. The simulation assumes an environment with perfect communications connectivity, without any temporal or spatial variation in the acoustic channel. The resultant network routes produced by the simulation are shown in Figure 35.

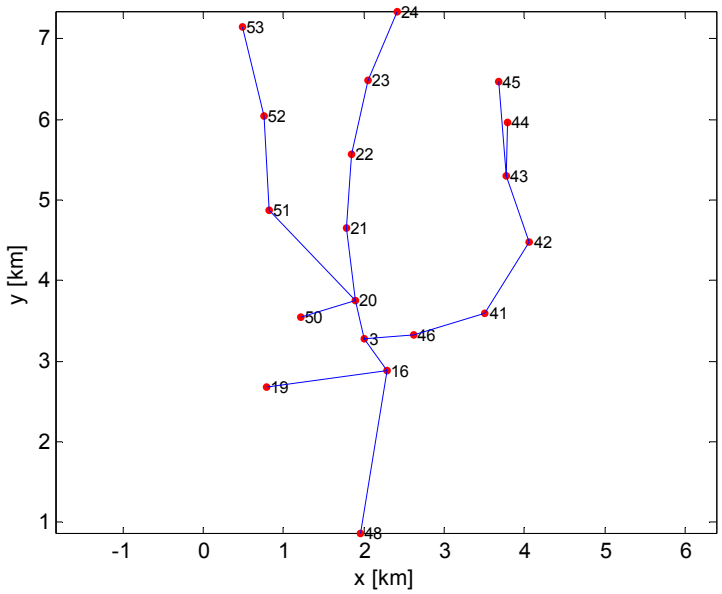


Figure 35. Resultant network routes obtained from simulation using 24 June 2008 trial coordinates and parameters.

D. COMPARISON AND ANALYSIS

A comparison between Figures 34 and 35 reveals that the simulation closely mirrors results from the sea trial, with three exceptions at nodes 24, 22 and 51 respectively.

Node 24 was the only node not found during the network discovery sea trial. Simulation indicates that it should have been discovered and routed via node 23. Closer examination of the trial log reveals that node 24 did not respond

to any of the ping messages from its neighboring nodes. Node 24 was essentially unreachable (non-participative) during the trial.

Node 22 was routed via node 51 during the sea trial, whereas simulation indicates that it should have been routed via node 21 which was nearer and situated to its south. Examination of the trial log reveals that node 21 did not discover node 22 during its peer discovery. In fact, node 21 did not discover any nodes other than those situated to its south (nodes 20 and 3). Node 22 was only discovered at a later stage during node 51's peer discovery, hence the resultant route via node 51. One possible explanation to node 21's inability to discover node 22 may lie in the bathymetry of the area where node 21 was deployed. Bathymetry contours in Figure 34 show that there was a steep underwater cliff within 50 m to the northwest of node 21's deployed position. This may have obscured acoustic communications to the west and north of node 21, thus preventing it from discovering node 22 during peer discovery.

Node 51 was routed via node 50 upon completion of the network discovery sea trial. Simulation indicates that node 51 should have been routed directly via node 20. The trial log reveals that node 20 did discover node 51 during its peer discovery and routing to node 51 should have been as suggested by simulation, i.e., via node 20. However, in trying to establish and distribute the routing table to node 51, there was a loss of communications connectivity between node 20 and node 51. As a result, node 51 was routed via another node it had connectivity with, i.e., node 50.

The total network costs ($C_{NETWORK}$) associated with the simulation routes and the less optimal sea trial routes are 3.27 and 3.59, respectively. The path cost to node 24 is not considered in the total network cost for both cases.

The above comparison shows that the network discovery algorithm is a useful tool for accurate prediction of the resultant network routes in a perfect

connectivity environment. This raises an interesting question. How optimal are the resultant network routes obtained at the end of the ad hoc network discovery process?

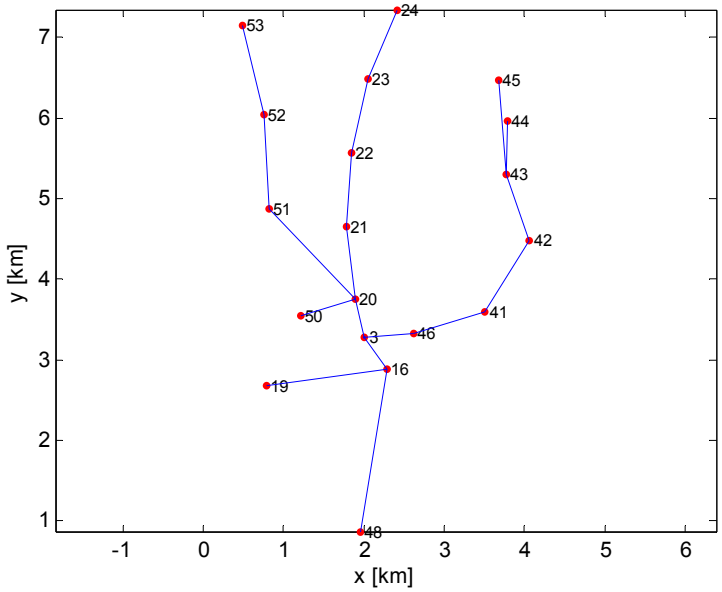
E. FOLLOW-ON ANALYSIS

1. Comparison with Dijkstra's Algorithm

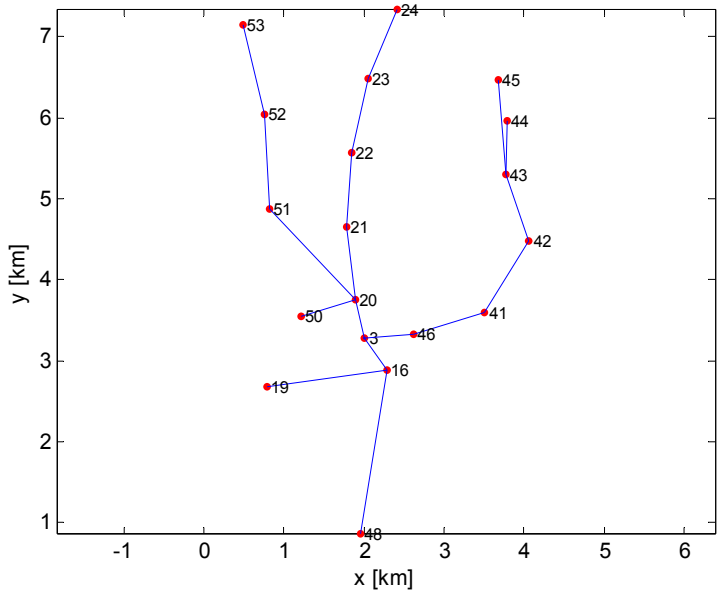
Dijkstra's algorithm finds the shortest paths from a given source node to all other nodes in a network by developing the paths in order of increasing path length. The algorithm proceeds in stages. By the k^{th} stage, the shortest paths to the k nodes closest to (least cost away from) the source node have been determined; these nodes are in a set T. At stage k+1, the node not in T that has the shortest path from the source node is added to T. As each node is added to T, its path from the source node is defined. The algorithm terminates when all nodes have been added to T [28].

The ad hoc network discovery process detailed in Chapter V and implemented for Seaweb bears a remarkable resemblance to Dijkstra's shortest path algorithm. The source node in our case is the master node and the nodes in the network are the branch nodes that are discovered as the network discovery process unfolds. Path lengths are the route costs calculated using the route cost evaluation function, and the set of nodes in T is the set of branch nodes with established routes and which have been directed by the master node to perform peer discovery.

For a given set of nodes and the set of link costs between connected nodes, Dijkstra's algorithm guarantees the best (lowest cost) path routes [24] from the master node to all discovered nodes in the network. In order to determine the optimality of the resultant network routes produced by the ad hoc network discovery process, there is a need to compare it against the corresponding results from the Dijkstra's algorithm. Figure 36 illustrates such a comparison, using the 24 June 2008 node coordinates and the corresponding network discovery parameters.



(a) Ad hoc discovery algorithm



(b) Dijkstra's algorithm

Figure 36. Comparison of results between simulation (top) and Dijkstra's algorithm (bottom). Both sets of resultant network routes are identical.

Both the simulation and Dijkstra's algorithm produce the same set of resultant network routes. This finding is not surprising since in the design of the ad hoc network discovery process, the master node always chooses the next nearest branch node to conduct peer discovery. Routing to that selected branch node is established prior to its execution of broadcast pings. Furthermore, neighbor information that is received at the master after each peer discovery are sorted based on increasing range before being aggregated into the master neighbor table. Such an implementation mirrors Dijkstra's concept of developing paths in order of increasing path length.

Therefore, it is concluded that the set of resultant network routes obtained at the end of the ad hoc network discovery process, in a perfect connectivity environment, is a set of optimal shortest (lowest cost) paths between the master node and all other discovered nodes in the network.

2. Cost Function Revision

Recall that the route cost evaluation function was defined in Chapter V as

$$C_i = \sum_{j=1}^{h} \left[\left(\frac{r_j - r_p}{r_p} \right)^2 + \left(\frac{r_j}{r_p} \right) \right]$$

where C_i is the path cost from the master node to node i,
h is the number of hops in the path,
r_j is the range between nodes in the j^{th} hop of the path, and
r_p is the preferred hop range.

In a more generic form, it can be rewritten with weighting coefficients (α, β) and exponent (γ) as study parameters

$$C_i = \sum_{j=1}^{h} \left[\alpha \left(\frac{r_j - r_p}{r_p} \right)^{\gamma} + \beta \left(\frac{r_j}{r_p} \right) \right]$$

The cost function is used to evaluate multiple paths from the master node to a specific branch node in order to identify the path with the lowest cost. The following section seeks to improve the empirically derived cost function by recognizing that complex routes are simply the sum of individual hops, as formulated in the cost function. Hence, the route selection problem may be reduced to a 3-node problem.

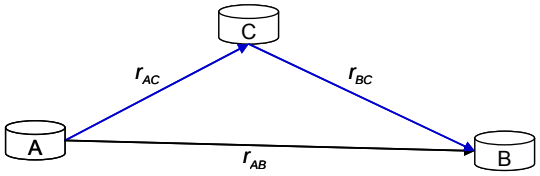


Figure 37. Schematic of the 3-node routing problem.

Given a source node A and a destination node B, an intermediate node C may be deployed with connections to both A and B. The routing algorithm would evaluate the route costs associated with direct path A-B (1 hop) and indirect path A-C-B (2 hops). Assuming no handicapping of nodes and a fixed user-specified preferred hop range (r_p) of 1 km, Figure 38 depicts the loci of positions for intermediate node C, within which a 2-hop route will be chosen over a 1-hop route, for three distances (r_{AB}) separating nodes A and B.

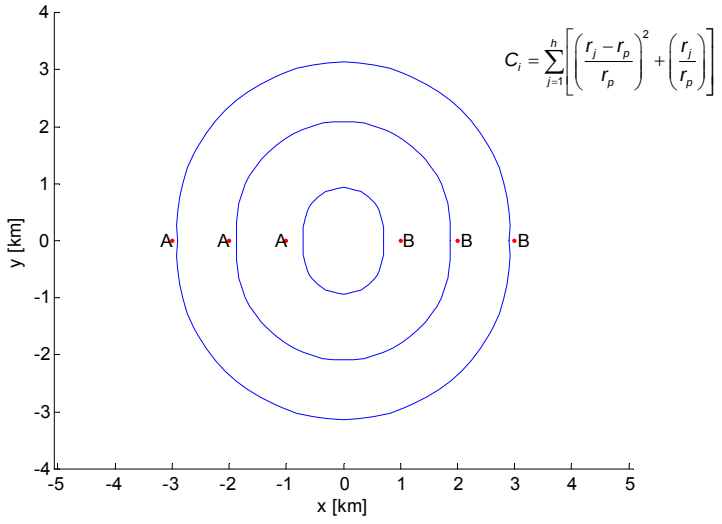


Figure 38. Loci of positions for node C within which node C will be chosen as the intermediate node for the route from node A to node B. r_p = 1 km, and r_{AB} = 2, 4, 6 km.

Notice that the locus of valid positions for node C to be chosen as an intermediate node using the current cost function is a vertical ellipse. This implies that if node C is deployed along the perpendicular bisector of line AB, there exists a higher chance of node C being accepted as a valid intermediate node that anywhere else between nodes A and B. Such an implication is undesired. Figures 39-41 are a study of the effect on the locus shape in response to adjustments to the cost function.

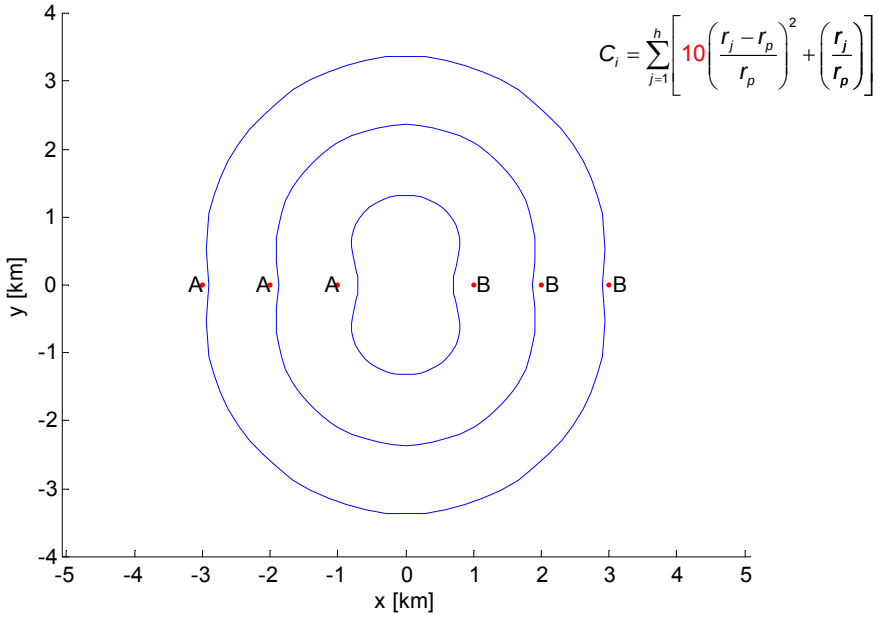


Figure 39. Effect of adding a coefficient ($\alpha = 10$) to the first term in the cost function.

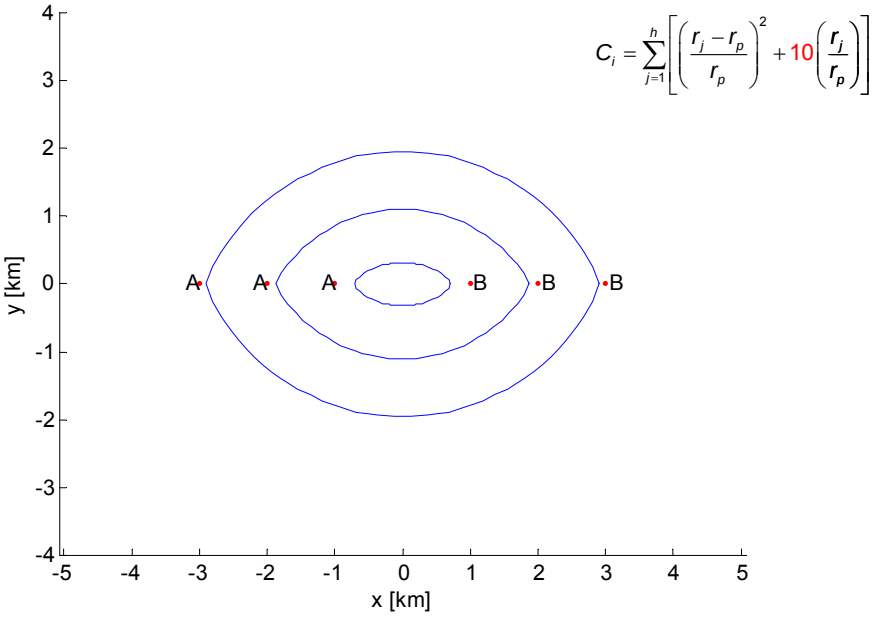


Figure 40. Effect of adding a coefficient ($\beta = 10$) to the second term in the cost function.

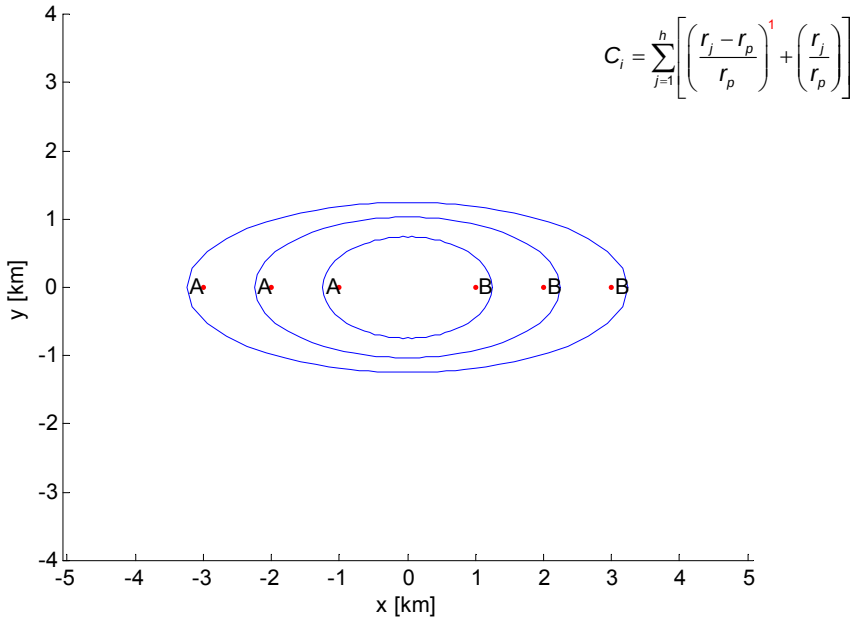


Figure 41. Effect of reducing the exponent ($\gamma = 1$) of the first term in the cost function.

It was subsequently determined that a cost function with a coefficient of 2 in the second term ($\beta = 2$) results in a locus of valid positions for node C that is circular, thus implying that all positions within a certain range from the mid-point of A and B have the same probability of being selected as a valid intermediate node. The corresponding loci of positions for node C is shown in Figure 42, and the revised route cost evaluation function for the ad hoc network discovery process is thus re-defined as

$$C_i = \sum_{j=1}^{h} \left[\left(\frac{r_j - r_p}{r_p} \right)^2 + 2 \left(\frac{r_j}{r_p} \right) \right]$$

where C_i is the path cost from the master node to node i,

h is the number of hops in the path,

r_j is the range between nodes in the j^{th} hop of the path, and

r_p is the preferred hop range.

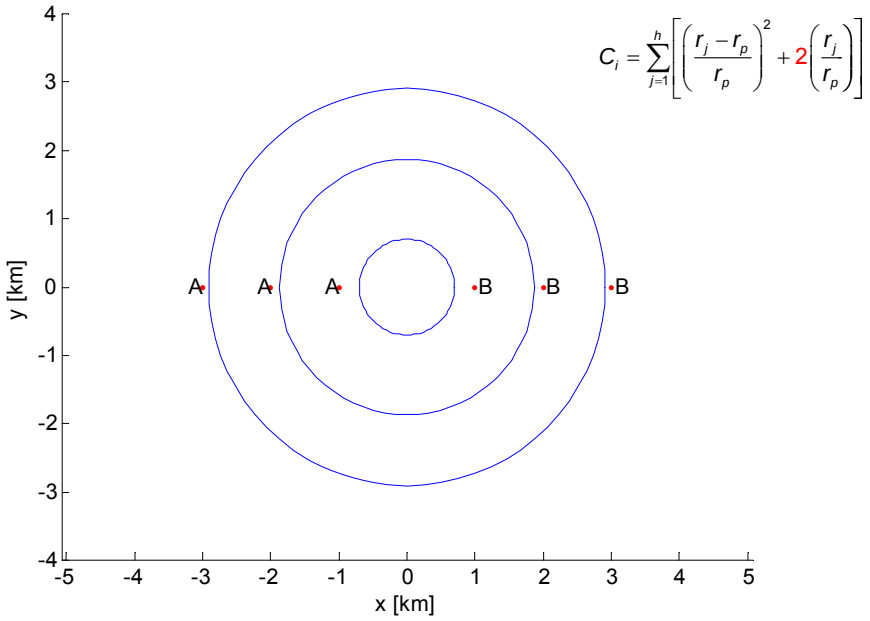


Figure 42. Loci of positions for node C to be chosen an intermediate node based on the revised cost function, keeping preferred hop range fixed at 1 km.

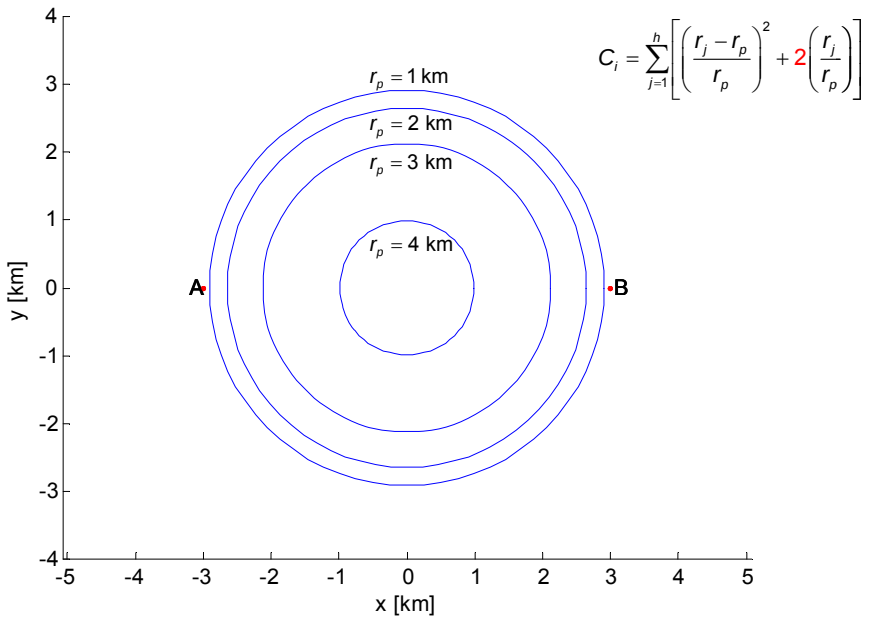


Figure 43. Loci of positions for node C as a function of preferred hop range (r_p), keeping distance between A and B fixed at 6 km.

Figure 43 presents the loci of positions for node C to be chosen as an intermediate node, as a function of the preferred hop range, keeping the range between nodes A and B constant. As the value of the preferred hop range increases, the locus shrinks, indicating that a direct long-distance hop (skipping node C as an intermediate node) is favored over a route comprising two short-distance hops, unless node C is deployed near to the mid-point of node A and node B.

3 Verification of the Revised Cost Function

The ad hoc network discovery process with the revised route cost evaluation function was sea-tested at Horten, Norway in September 2008. This time, the network consisted of only 9 nodes with a maximum node-to-node spacing of less than 1000 m. The average water depth was 15 m. The network discovery parameters used for the trial are listed in Table 3. The resultant network routes obtained upon completion of the discovery process and that from simulation are presented in Figures 44 and 45 respectively.

Parameter	Value
Master Node	Node 3
No. of Broadcast Ping	2 per peer discovery
Ping Power Level	1000 m (equivalent)
Range Cutoff	1000 m
Preferred Hop Range	150 m
Handicapped nodes	Node 3

Table 3. Network discovery parameters used on 24 September 2008.

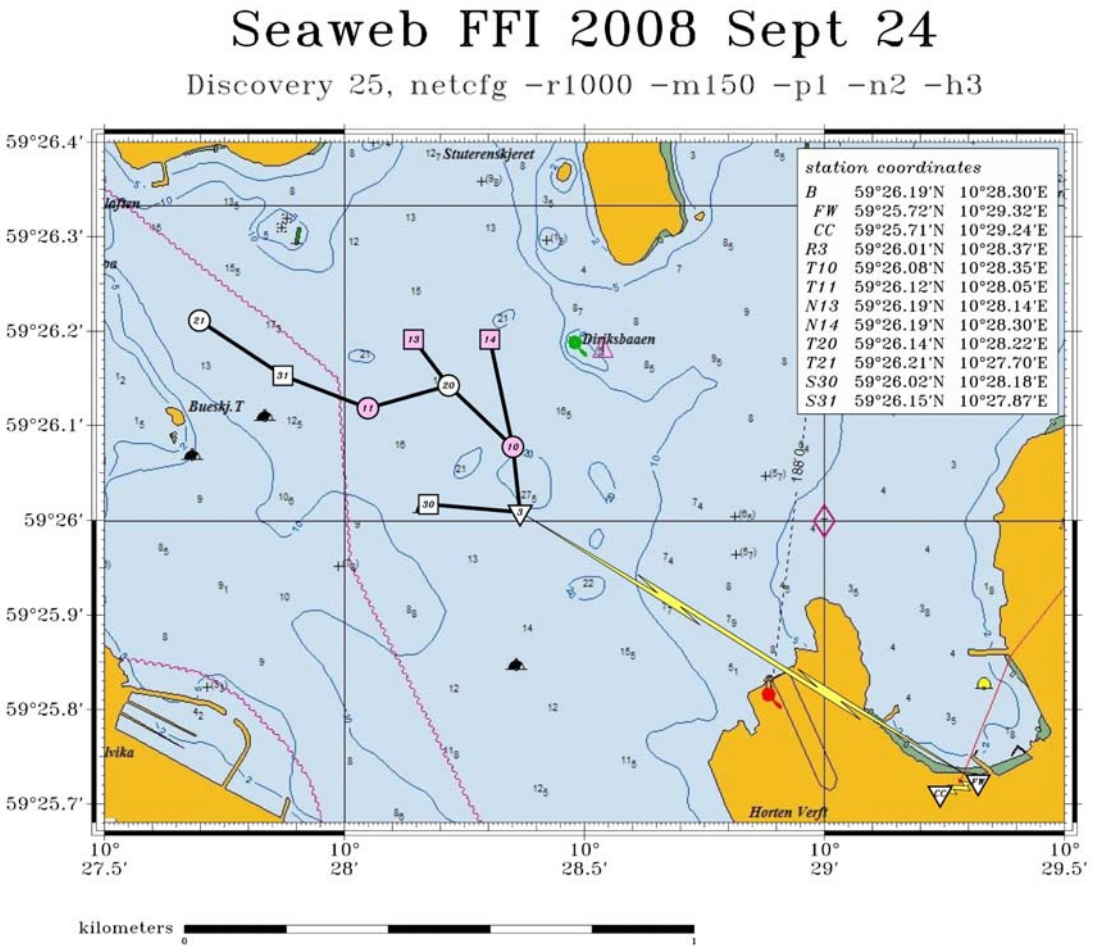
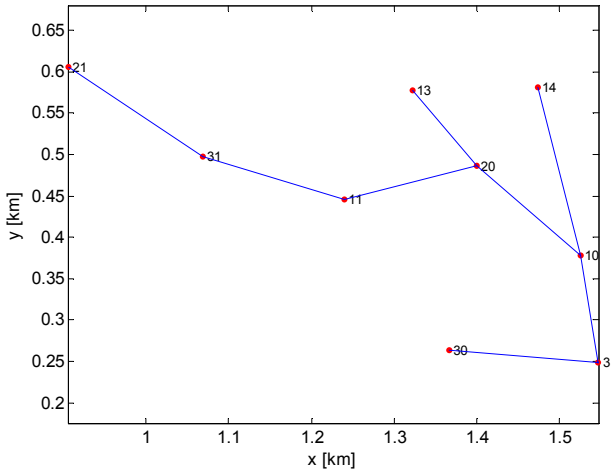


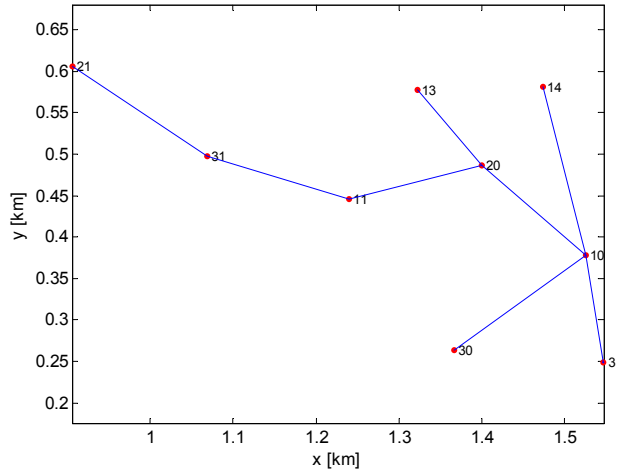
Figure 44. Resultant network routes from the September sea trial using revised cost function.



$$C_i = \sum_{j=1}^{h}\left[\left(\frac{r_j - r_p}{r_p}\right)^2 + 2\left(\frac{r_j}{r_p}\right)\right]$$

Figure 45. Simulation results using revised cost function and Sep trial coordinates.

The sea trial results indicate that the revised cost function is feasible and capable of producing a set of network routes that are optimal since it matches the simulation results, which are in turn a set of Dijkstra's shortest (lowest cost) paths as previously discussed. For the sake of comparison, the set of network routes from simulation with the original cost function is shown in Figure 46. Notice that the only difference is node 10 is not favored as an intermediate node for the route from master node 3 to node 30 when the revised cost function is used.



$$C_i = \sum_{j=1}^{h}\left[\left(\frac{r_j - r_p}{r_p}\right)^2 + \left(\frac{r_j}{r_p}\right)\right]$$

Figure 46. Simulation results using the original cost function and Sep trial coordinates.

VII. AN ALTERNATIVE NETWORK DISCOVERY PROCESS

The ad hoc network discovery process presented in the preceding two chapters worked well in an acoustic environment not severely affected by spatial and temporal variations in propagation conditions. The resultant network routes, obtained under such a favorable connectivity environment, are a set of Dijkstra's shortest paths from the master node to all discovered nodes in the network.

However, when there is a temporary loss of connectivity with one or more branch nodes, especially during route establishment or distribution of local routing tables, the resultant route to the affected node may not be the shortest (lowest cost) path. Such a scenario was encountered with node 51 during the June 2008 sea trial detailed in Chapter V. In fact, some resultant routes may be circuitous in nature.

In an attempt to address this issue and to refine the network discovery process, this chapter presents an alternative network discovery scheme that tries to gather as much neighborhood information as possible prior to establishing a route to a specific branch node. This alternative discovery scheme is based largely on the original implemented network discovery process. However, the alternative scheme exploits the network-layer feature of cellular addressing, introduced in Chapter IV.

A. DESCRIPTION

The functionalities of peer discovery (broadcast ping) and route establishment in the alternative scheme are unchanged from the original network discovery process detailed in Chapter V. The key difference is that the master node now uses cellular addressing to communicate with the neighbors of a routed branch node. The use of cellular addressing enables the master to direct these neighbor nodes to perform peer discovery an additional ply into the network, thereby increasing the master's knowledge of the network before a

routing decision to the next nearest branch node is made. This is in contrast to the original implemented process where a node is always routed to prior to being directed to perform a peer discovery.

Figures 47-55 illustrate the alternative discovery process, up to the point of completion when all the immediate neighbors of node A (tier 1 nodes) are routed.

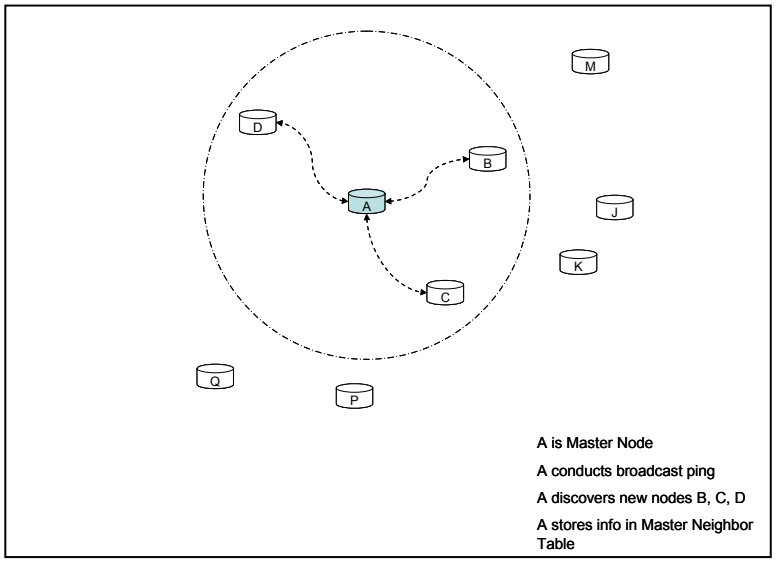


Figure 47. Master node A performs neighborhood discovery and finds nodes B, C, and D.

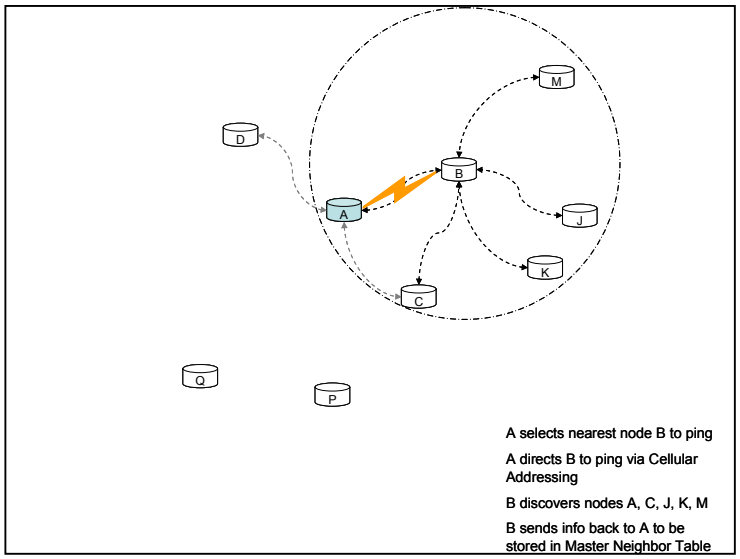


Figure 48. Master node A uses cellular addressing (orange lightning symbol) to direct nearest node B to perform peer discovery. Node B sends neighbor information back to master node utilizing the cellular address (node A).

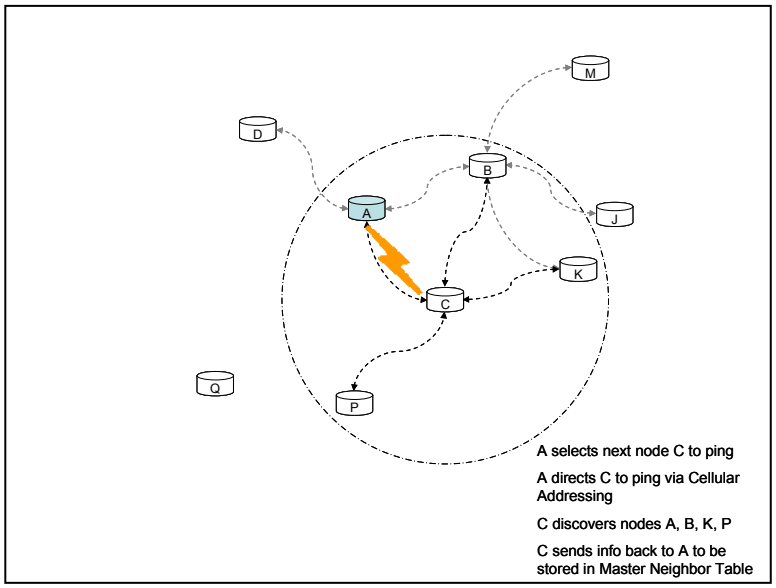


Figure 49. Master node A uses cellular addressing to direct next nearest node C to perform peer discovery. Master node expands its knowledge of network.

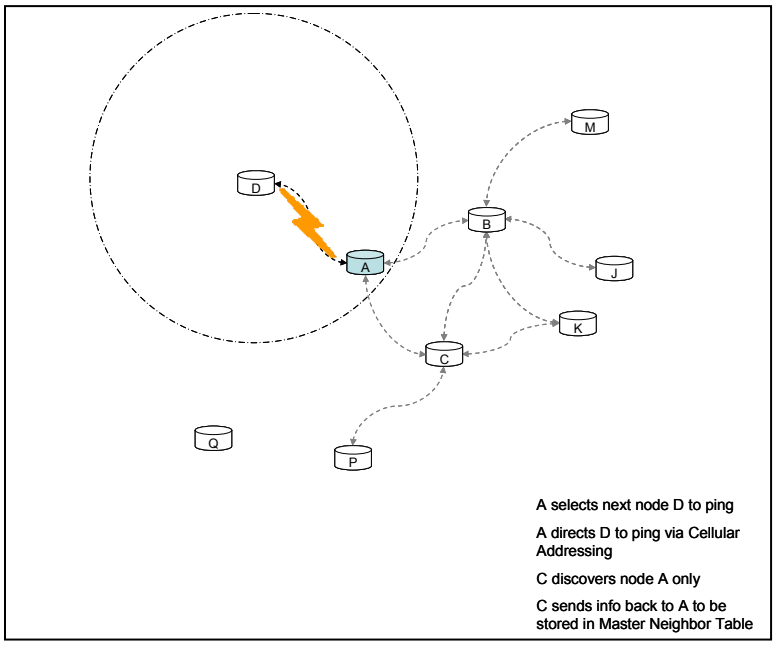


Figure 50. Master node A uses cellular addressing to direct next nearest node D to perform peer discovery. No other nodes are reachable via cellular address (node A).

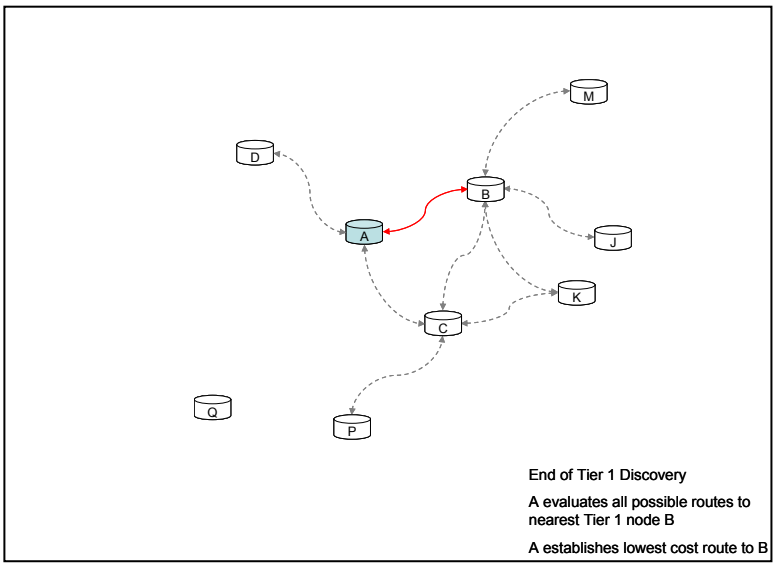


Figure 51. Master node examines master neighbor table to establish lowest cost route (red arrow) to nearest immediate neighbor node B.

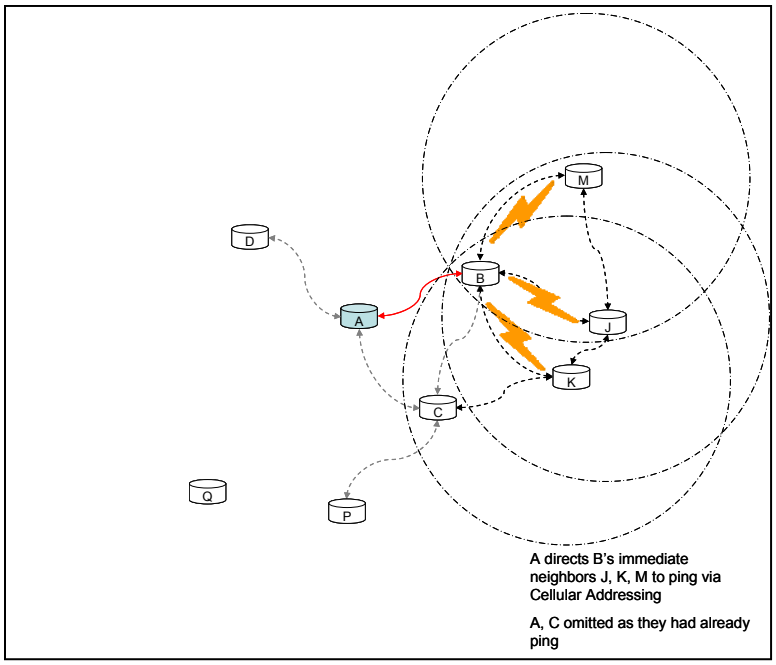


Figure 52. After routing to node B, master node A uses cellular address (node B) to direct node B's neighbors to conduct peer discovery one at a time. Master node's knowledge of the network is further expanded.

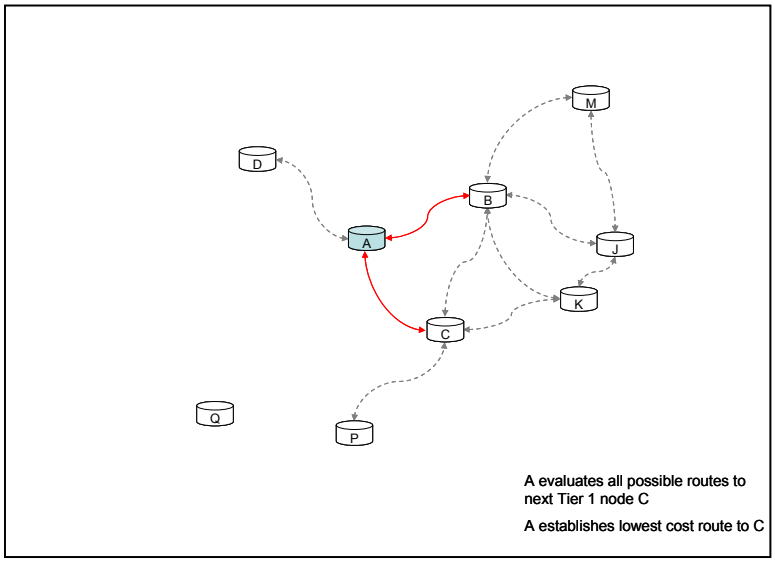


Figure 53. Master node A establishes route to its next nearest immediate neighbor node C.

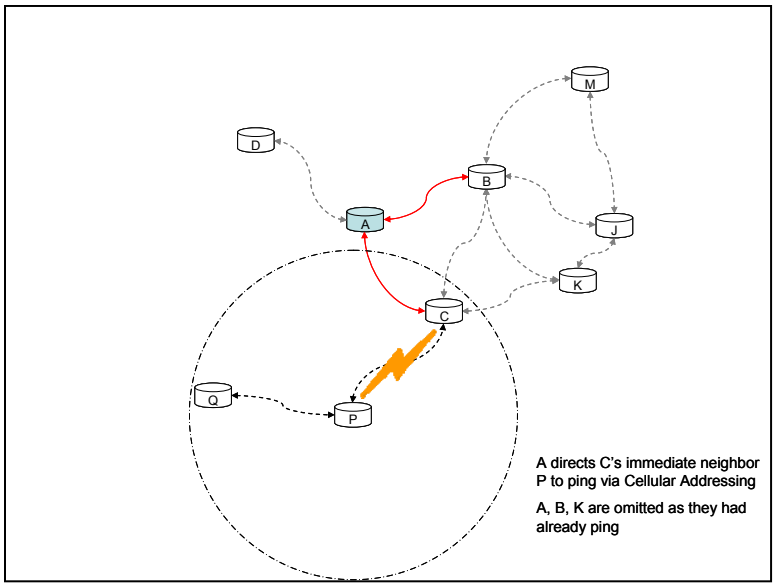


Figure 54. After routing to node C, master node A uses cellular address (node C) to direct node P to perform peer discovery. Node P found node Q. No other nodes that have not performed peer discovery are reachable via cellular address (node C).

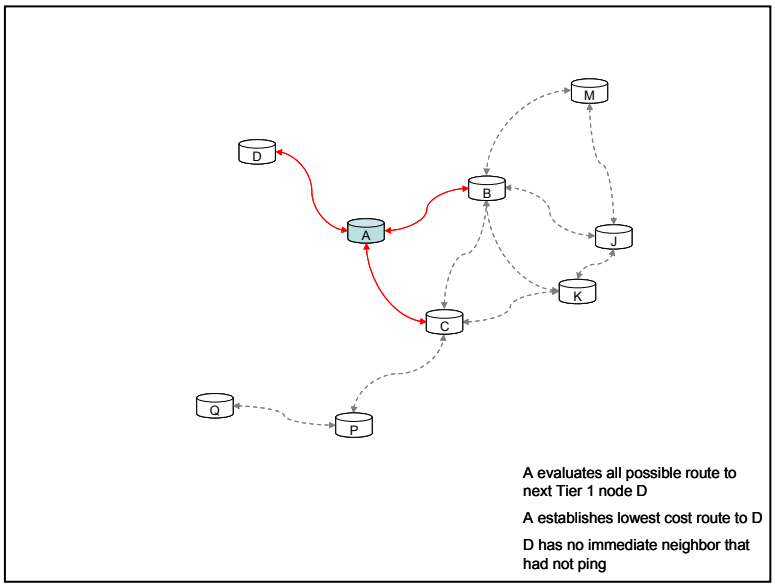


Figure 55. Master node establishes route to next nearest neighbor node D. Node D does not have any immediate neighbors that have not performed peer discovery.

Upon completion of routing to all its immediate neighbor nodes (as depicted in Figure 55), the master node proceeds to establish a route to the next nearest node (in this case node J), sets it as the cellular address, and the discovery cycle is repeated. The network discovery process terminates when no more new nodes are discovered and all discovered nodes have been routed.

B. COMPARISON WITH ORIGINAL DISCOVERY PROCESS

Evident from the preceding illustrations, the alternative discovery process allows the master node to acquire a more extensive knowledge of the network before a routing decision is made. Such an expanded knowledge of the network would be useful in the event of temporary loss of connectivity with one or more nodes as this would present more path options to the master node as it tries to establish a “next best” route.

In a perfect connectivity environment, both the original discovery process and the alternative discovery scheme produce the same resultant network routes. This is because in such an environment, all nodes within earshot of a broadcast ping would have responded and the master node’s knowledge of the network up to the range of the specific branch node that conducted peer discovery is the same in both cases. While the alternative discovery process presents the master node with more path options to the branch node, the shortest (lowest cost) path is always one that comprises intermediate nodes that lie between the master and that branch node. Therefore, expanded knowledge of the network beyond the range of that branch node, made possible by the alternative discovery process, does not serve to improve the shortest path routes in a perfect connectivity environment. Figure 56 shows the simulation results from both the discovery processes using June 2008 trial coordinates and the revised cost function.

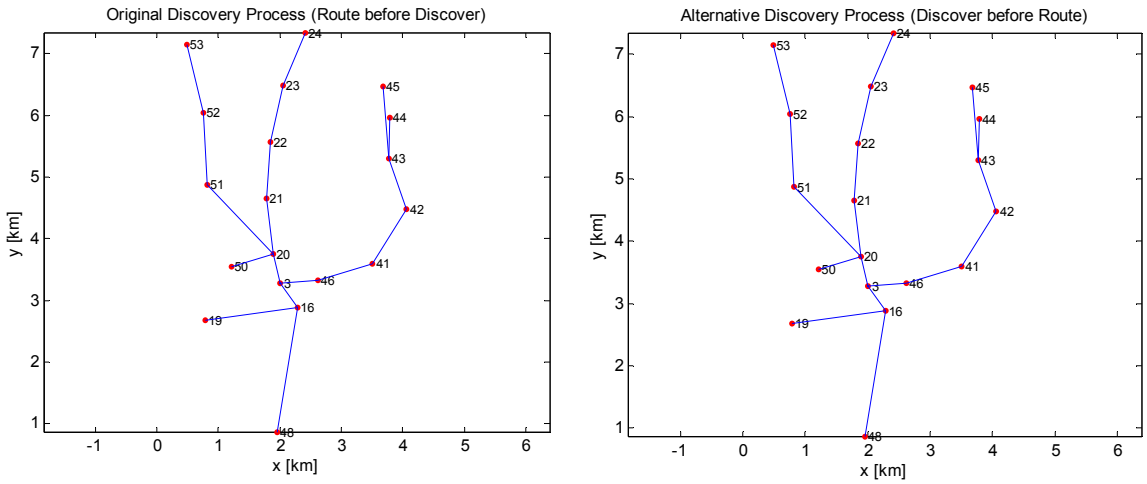


Figure 56. Simulation results from the original discovery process (left) are the same as that from the alternative discovery process (right) in a perfect connectivity environment.

In reality, the underwater acoustic environment is an imperfect connectivity channel. The benefit of having an expanded knowledge of the network before route establishment seems invaluable. Moreover, the acquisition of range data gives opportunity for localizing the nodes during the discovery process rather than afterwards.

Figure 57 presents the program flowchart of the alternative discovery process (discover before route) contrasted against the original implemented network discovery process (route before discover).

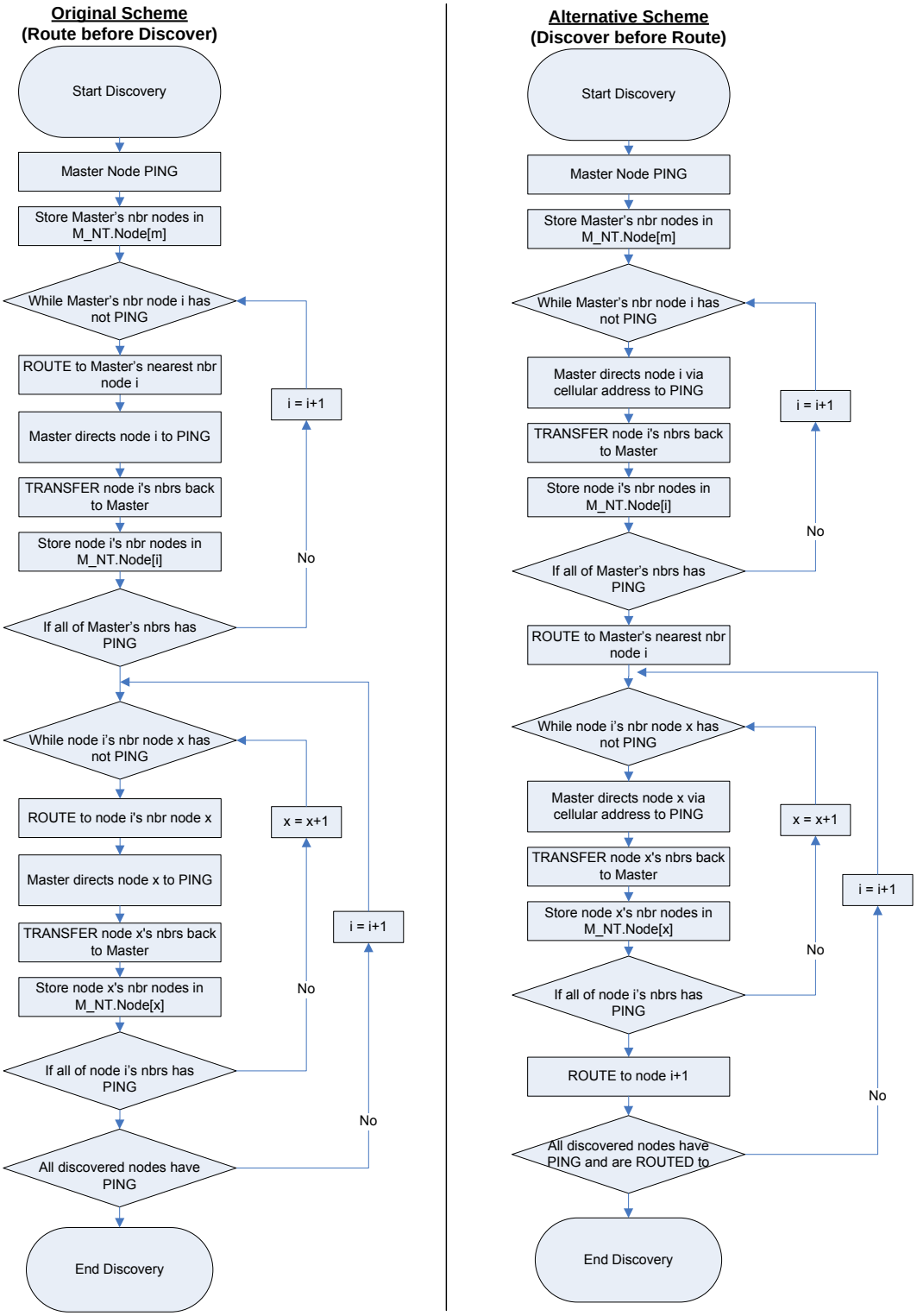


Figure 57. Comparison between the original and the alternative discovery schemes.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSIONS

A. SUMMARY

This thesis developed and implemented a Seaweb network discovery process in an effort to enable spontaneous deployment of ad hoc autonomous nodes capable of auto-configuration for networking purposes.

The network discovery process builds upon existing Seaweb link-layer and network-layer features. Neighborhood discovery is conducted in tandem with network routing under the centralized control of a master node. Simulation and at-sea trial data indicate that the implemented discovery process is feasible, and in the absence of loss of connectivity, the resultant network routes obtained upon completion of the discovery process are a set of optimal Dijkstra's shortest (lowest cost) paths from the master node to all discovered nodes in the network. Refinements to the route cost evaluation function were identified and tested at sea.

An alternative discovery scheme aimed at expanding the master node's knowledge of the network before any route establishment is also discussed. It is believed that the alternative discovery process is more robust in an environment affected by temporal and spatial variations in the acoustic channel.

B. RECOMMENDATIONS FOR FUTURE WORK

1. Alternative Network Discovery Process

The simulation results presented in this thesis assume an environment with perfect acoustic communication connectivity. In order to quantify the expected advantage afforded by the alternative network discovery scheme (discover before route) over the original implemented discovery process (route before discover), there is a need to simulate an environment where the probability associated with loss of communication connectivity is modeled.

Results from such a simulation can subsequently inform the actual implementation of the alternative discovery process.

2. Node Localization

Ad hoc network initialization can sometimes entail the need for node localization. Conceptually, the master node needs at least three different sets of node-to-node ranges in order to triangulate a particular node, relative to a local coordinate system. Such a localization algorithm can be incorporated to work in tandem with the network discovery process. As peer discovery ripples through the network and the global neighbor table is populated, the master node can process the node-to-node ranges and localize discovered nodes.

3. Route Optimization

The network routes obtained upon completion of the network discovery process are a set of optimal (lowest cost) bi-directional routes from the master node to all discovered nodes in the network. However, these routes are not optimized between any two arbitrary branch nodes trying to communicate with each other. A peer-to-peer route optimization is required. The neighborhood information contained in the global neighbor table provides the necessary data to initiate a route optimization algorithm. Once peer-to-peer routes are optimized, the master node may update the local routing tables of all branch nodes.

4. Quickening the Discovery Process

The network discovery process developed in this thesis follows an exhaustive and sequential approach. As such, the execution of this process is time-consuming with duration on the order of N^2. There is opportunity for speeding up the process with adaptive tuning of timers associated with the broadcast ping processes.

APPENDIX A NETWORK DISCOVERY SOURCE CODE

```
// Ad Hoc Network Discovery Code (ver 13 Aug 08)
// Ong Chee Wei

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 50                // No. of nodes
#define A 15                // Area A km by A km
#define cutoff_rng 4        // Cutoff range (km) for nodes to be considered immediate nbrs
#define ping_pwr_lvl 8      // Broadcast ping power level (1 to 8). Ranges defined in Ping function
#define P_echo_fail 0.0     // Probability (0 to 1) of echo failure in response to each ping
#define no_of_pings 1       // User defined no. of broadcast pings (1 or 2)
#define metric 1            // User defined metric (km) corresponding to preferred hop length
#define handicap_val 0.5    // User defined handicap value (0.5 or 1) at master node (RACOM buoy)
#define C1 1                // New Cost Fn weight associated with 1st term
#define C2 2                // New Cost Fn weight associated with 2nd term
#define E1 2                // New Cost Fn exponent associated with 1st term

struct posn {
       float x;             // node position
       float y;
       char address;        // pre-configured unique 8 bit node address
       char srl;
       };
struct posn g_posn[N];      // Global position structure

float g_rng[N][N];          // True Range lookup table b/w nodes

struct nbr {
       float range;
       char address;
       };

struct Nbr_table {
       struct posn myposn;
       char myaddress;
       char found_by;
       int rt[N];               // local routing vector
       struct nbr mynbr[N];
       };
struct Nbr_table Node[N];       // Node Neighbour Table

struct m_nbr {
       char address;
       float range1;
       float range2;
       float filtered_range;
       };

struct M_Nbr_table {
       char address;
       struct m_nbr nbr_node[N];
       };
struct M_Nbr_table M_NT_row[N];  // Master Neighbour Table

int M_Routing_table[N][N];       // Master Routing Table

int ping(int m, int ping_node, int pwr_lvl);   // Ping function declaration
void transfer(int m, int fr, int to);          // Tranfer function declaration
int aggregate(int m, int ping_node);           // Aggregate function declaration
void determine_route(int m, int ping_node);    // Determine Route function declaration
void distribute_route(int m, int ping_node);   // Distribute Route function declaration

float min_route_cost[N];         // min route cost table

/* Main Program */
main(){

       FILE *out0, *out1, *out2;
       out0=fopen("log.out", "w");
       out1=fopen("g_posn.out", "w");
       out2=fopen("rng_table.out", "w");
       FILE *out5;
```

```
        out5=fopen("graph.out","w");

        int node_i, node_j;       // counters

//-------------------------------------------------
        /* Global position to setup N nodes in A km by A km area */
        for (node_i=0; node_i<N; node_i++){
            g_posn[node_i].x = (float)rand()/RAND_MAX*A;
            g_posn[node_i].y = (float)rand()/RAND_MAX*A;
            g_posn[node_i].address = 20 + node_i;       // node address starts from 20

            Node[node_i].myaddress = g_posn[node_i].address;   // each node knows its address

            fprintf(out1,   "%d\t   %f\t   %f\t   %d\n",   node_i,   g_posn[node_i].x,   g_posn[node_i].y,
g_posn[node_i].address);
            }
        fclose(out1);

//-------------------------------------------------
//        /* Global position of N nodes read-in from input0.in file */
//        FILE *in1;
//        in1=fopen("input0.in", "r");
//
//        for (node_i=0; node_i<N; node_i++){
//            fscanf(in1,"%f %f %d", &g_posn[node_i].x, &g_posn[node_i].y, &g_posn[node_i].srl);
//            g_posn[node_i].address = 20 + node_i;       // node address starts from 20
//            Node[node_i].myaddress = g_posn[node_i].address;   // each node knows its address
//                fprintf(out1,  "%d\t  %f\t  %f\t  %d\n",  node_i,  g_posn[node_i].x,  g_posn[node_i].y,
g_posn[node_i].address);
//            }
//        fclose(in1);
//        fclose(out1);
//-------------------------------------------------

        /* Compute true range between nodes i and j (stored in look-up table)*/
        for (node_i=0; node_i<N; node_i++){
            for (node_j=0; node_j<N; node_j++){
                g_rng[node_i][node_j]    =    sqrt((g_posn[node_i].x-g_posn[node_j].x)*(g_posn[node_i].x-
g_posn[node_j].x)
                                          +      (g_posn[node_i].y-g_posn[node_j].y)*(g_posn[node_i].y-
g_posn[node_j].y));
                fprintf(out2, "g_rng[%d][%d]\t %f\n", node_i, node_j, g_rng[node_i][node_j]);
                }
            }
        fclose(out2);

////////*** Network Discovery ***/////////
        int m, p, i, j;                         // m = Master node index, p = ping node index
        int nodes_found=0, total_nodes_found=1, new_nodes_found=0;
        int initialise=0, ping_tries;

        FILE *out3;
        out3=fopen("M_NT.out", "w");

        for (i=0; i<total_nodes_found; i++){
            if (initialise == 0){               // Master node initialisation
               m=0;                             // Master node index set to 0
               p = m;                           // p = index of Ping Node
               initialise = 1;
               }
            else p = M_NT_row[i].address - 20;              // index of subsequent Ping Node

            determine_route(m, p);                          // Master determines route to Ping Node

            distribute_route(m, p);                         // Master updates route to Ping Node

            for (ping_tries=0; ping_tries<no_of_pings; ping_tries++){
                nodes_found = ping(m, p, ping_pwr_lvl);     // User-defined No. of Discovery Pings
                if (nodes_found > 0)
                   transfer(m, p, m);                       // Ping node sends nbr data to M_NT
                }

            new_nodes_found = aggregate(m, p);              // Master aggregates data and expands M_NT

            total_nodes_found = total_nodes_found + new_nodes_found;
            }

        printf("\nNo. of nodes NOT discovered = %d\n\n", N-total_nodes_found);
////////*** End Network Discovery ***/////////
```

```
//-----------------------------------------------
       /* Output M_NT */
       for (i=0; i<total_nodes_found; i++){
           j=0;
           fprintf(out3, "%d\t %d\t %f\t %f\t %f\n", M_NT_row[i].address,
                  M_NT_row[i].nbr_node[j].address, M_NT_row[i].nbr_node[j].filtered_range,
                  M_NT_row[i].nbr_node[j].range1, M_NT_row[i].nbr_node[j].range2);
           for (j=1; j<N; j++){
               if (M_NT_row[i].nbr_node[j].address != 0){
                   fprintf(out3, "%d\t %d\t %f\t %f\t %f\n", M_NT_row[i].address,
                          M_NT_row[i].nbr_node[j].address, M_NT_row[i].nbr_node[j].filtered_range,
                          M_NT_row[i].nbr_node[j].range1, M_NT_row[i].nbr_node[j].range2);
                  }
                  else break;
               }
           }

       fprintf(out0,"\nNo. of nodes NOT discovered = %d\n\n\n", N-total_nodes_found);

       fclose(out3);
       fclose(out0);
       fclose(out5);

       /* Output local routing vectors */
       FILE *out6;
       out6=fopen("M_RT.out","w");
       for (i=0; i<N; i++){
           fprintf(out6,"Node %d\t", Node[i].myaddress);
           for (j=0; j<N; j++){
               fprintf(out6,"%d\t", Node[i].rt[j]);
               }
           fprintf(out6,"\n");
           }
       fclose(out6);

       /* Output min route cost */
       FILE *out7;
       out7=fopen("min_route_cost.out","w");
       for (i=0; i<N; i++){
           fprintf(out7,"%d\t %f\n", Node[i].myaddress, min_route_cost[i]);
           }
       fclose(out7);
//-----------------------------------------------

       system("PAUSE");
       }

////////////////////////////////////////////////////////////////////////////////////////////////////

/* Ping function definition */
int ping (int m, int ping_node_idx, int pwr_lvl){
       FILE *out0;
       out0=fopen("log.out", "a");
       int i, j, hit = 0;
       int exist, k = 0;                  // k = index to Nbr table
       float power[8] = {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0};    // 8 ping ranges
       float ping_rng = power[pwr_lvl-1];

       int node_idx = m, broadcast_ping = 0, route_node;

       if (node_idx == ping_node_idx) broadcast_ping = 1;            // Master Node Ping
       else {
            while (node_idx != ping_node_idx){                       // forward the ping command
                  route_node = Node[node_idx].rt[ping_node_idx];
                  node_idx = route_node - 20;                        // index of route node
                  if (node_idx == ping_node_idx) broadcast_ping = 1;
                  }
            }

       /* Broadcast Ping */
       float P_rand;

       if (broadcast_ping == 1){

       for (i=0; i<N; i++){
           if (i!=node_idx){
           P_rand = (float)rand()/RAND_MAX;   // generate random echo return probability at each node
           if (g_rng[node_idx][i] <= ping_rng){
```

```
            if (P_rand >= P_echo_fail){        // proceed with echo in response to broadcast ping
               for (j=0; j<N; j++){            // check if node i exists in Nbr table
                   if (Node[node_idx].mynbr[j].address == g_posn[i].address){
                      exist = 1;
                      Node[node_idx].mynbr[j].range = g_rng[node_idx][i];    // retain the latest range
                      break;
                      }
                   else exist = 0;
                   }
               if (exist == 0){               // node i is a new node
                  for (j=0; j<N; j++){
                      if (Node[node_idx].mynbr[j].address == 0){
                         k = j;
                         break;
                         }
                      }
                  Node[node_idx].mynbr[k].range = g_rng[node_idx][i];
                  Node[node_idx].mynbr[k].address = Node[i].myaddress;

                  fprintf(out0,"Pwr lvl %d\t Node %d\t discovered Node %d\t at Range = %f\n",
                          pwr_lvl, Node[node_idx].myaddress, Node[node_idx].mynbr[k].address,
                          Node[node_idx].mynbr[k].range);
                  }  // end of if
               hit = hit + 1;                 // no. of neighbours found
               }   // end of if
               }
            }
            }  // end of for
         return(hit);                         // return no. of neighbours discovered
         }  // end of if
         }

////////////////////////////////////////////////////////////////////////////////////////////////////

/* Transfer neighbour data from Ping node to Master node to be stored in M_NT function definition */
void transfer (int m, int fr, int to){
         int i, j, k;
         struct Nbr_table temp_NT[N];
         int idx, route_node, nodes_found, exist;

         if (fr == m){
            printf("%d ", Node[fr].myaddress);
            temp_NT[to] = Node[m];                          // at Master node
            }
         else {
              idx = fr;
              temp_NT[fr] = Node[fr];
              while (idx != to){
              printf("%d ", Node[fr].myaddress);
                    route_node = Node[idx].rt[to];          // reverse path transfer
                    idx = route_node - 20;
                    temp_NT[idx] = temp_NT[fr];
                    fr = idx;
                    }
              if (idx == to) temp_NT[to] = temp_NT[idx];    // at Master node
              }
         printf("%d\n", Node[to].myaddress);


         for (i=0; i<N; i++){
             if (temp_NT[to].mynbr[i].address == 0){
                nodes_found = i;                // determine no. of nodes found by ping node
                break;
                }
             }
         for (i=0; i<N; i++){
             if (M_NT_row[i].address == temp_NT[to].myaddress){
                k = i;                          // row index in M_NT corresponding to ping node
                break;
                }
             }
         for (i=0; i<nodes_found; i++){
             for (j=0; j<N; j++){               // check if node i exists in M_NT_row[k]
                 if (M_NT_row[k].nbr_node[j].address == temp_NT[to].mynbr[i].address){
                    exist = 1;
                    M_NT_row[k].nbr_node[j].range2 = temp_NT[to].mynbr[i].range;
                    break;
                    }
                 else exist = 0;
```

```
            }
        if (exist == 0){                // node i is a new node in M_NT_row[k]
           for (j=0; j<N; j++){
               if (M_NT_row[k].nbr_node[j].address == 0)    // find empty row
                  break;
               }
           M_NT_row[k].nbr_node[j].address = temp_NT[to].mynbr[i].address;
           M_NT_row[k].nbr_node[j].range1 = temp_NT[to].mynbr[i].range;
           }
        }
    }

////////////////////////////////////////////////////////////////////////////////////////////////

/* Master node aggregates data and expand M_NT function definition */
int aggregate (int m, int ping_node_idx){
    int i, j, k, exist, nodes_found, new_nodes = 0;
    float range, range1, range2;

    for (i=0; i<N; i++){
        if (M_NT_row[i].address == Node[ping_node_idx].myaddress){
           k = i;               // row index k in M_NT corresponding to ping node
           break;
           }
        }
    for (i=0; i<N; i++){
        if (M_NT_row[k].nbr_node[i].address == 0){
           nodes_found = i;     // determines no. of nodes found by ping node
           break;
           }
        }

    for (i=0; i<nodes_found; i++){        // Master node aggregates ranges & checks for new node
        range1 = M_NT_row[k].nbr_node[i].range1;
        range2 = M_NT_row[k].nbr_node[i].range2;
        if (range1 != 0 && range2 == 0) range = range1;       // rules to filter 2 ranges
        if (range1 == 0 && range2 != 0) range = range2;
        if (range1 != 0 && range2 != 0) range = (range1 + range2) * 0.5;
        M_NT_row[k].nbr_node[i].filtered_range = range;
        }

    struct m_nbr Temp;
    for (i=0; i<nodes_found; i++){         // Bubble Sort according to range
        for (j=0; j<nodes_found-i-1; j++){
            if (M_NT_row[k].nbr_node[j].filtered_range > M_NT_row[k].nbr_node[j+1].filtered_range){
               Temp = M_NT_row[k].nbr_node[j];
               M_NT_row[k].nbr_node[j] = M_NT_row[k].nbr_node[j+1];
               M_NT_row[k].nbr_node[j+1] = Temp;
               }
            }
        }

    for (i=0; i<nodes_found; i++){
        for (j=0; j<N; j++){
            if (M_NT_row[k].nbr_node[i].address == M_NT_row[j].address){
               exist = 1;
               break;
               }
            else exist = 0;
            }
        if (exist == 0){            // M_NT_row[k].nbr_node[i] is a new node
                                    // apply cutoff range to determine immediate nbrs
           if (M_NT_row[k].nbr_node[i].filtered_range <= cutoff_rng){
              for (j=0; j<N; j++){
                  if (M_NT_row[j].address == 0)     // find empty row in M_NT
                     break;
                  }
              M_NT_row[j].address = M_NT_row[k].nbr_node[i].address;  // expand M_NT
              new_nodes++;
              }
           }
        }
    return(new_nodes);
    }

////////////////////////////////////////////////////////////////////////////////////////////////

/* Master node determines route to Ping Node (node_idx) function definition */
void determine_route (int m, int node_idx){
```

```
        int i, j, k, exist;
        FILE *out5;
        out5=fopen("graph.out","a");

        M_Routing_table[node_idx][node_idx] = Node[node_idx].myaddress;

        if (node_idx == m){         // initialise M_NT for Master node
           M_NT_row[0].address = Node[m].myaddress;
           Node[m].found_by = Node[m].myaddress;

           fprintf(out5,"%d\t%f\t%f\t%d\t%d\n", node_idx, g_posn[node_idx].x, g_posn[node_idx].y,
                  Node[node_idx].myaddress, Node[node_idx].found_by);

           return;
           }

        for (i=0; i<N; i++){
            if (M_NT_row[i].address == Node[node_idx].myaddress){
               k = i;               // row index k in M_NT corresponding to node index
               break;
               }
            }


        if (M_Routing_table[m][node_idx] != 0)   // check if route exist
           exist = 1;
        else exist = 0;

        int idx, route_node, low_route_node, r, s, found;
        float route_value, low_route_value=10000;

        if (exist == 0){
           for (i=0; i<k; i++){
               found = 0;
               for (j=0; j<N; j++){
                   if (M_NT_row[i].nbr_node[j].address == 0) break;
                   if (M_NT_row[i].nbr_node[j].address == Node[node_idx].myaddress){
                      if (M_NT_row[i].nbr_node[j].filtered_range <= cutoff_rng){
                         found = 1;
                         idx = M_NT_row[i].address - 20;

                         float metr;
                         if (idx == m) metr = handicap_val*metric;    // handicap master node
                         else metr = metric;

                         route_value = (C1*pow(((M_NT_row[i].nbr_node[j].filtered_range - metr)/metr),
E1) + C2*M_NT_row[i].nbr_node[j].filtered_range/metr);

                         while (idx != m){
                               route_node = M_Routing_table[idx][m];

                               for (r=0; r<N; r++){
                                   if (M_NT_row[r].address == route_node)
                                      break;        // row index r in M_NT corresponding to route_node
                                   }
                               for (s=0; s<N; s++){
                                   if (M_NT_row[r].nbr_node[s].address == Node[idx].myaddress)
                                      break;        // nbr index s in M_NT_row[r] corresponding to idx
                                   }

                               idx = route_node - 20;

                               if (idx == m) metr = handicap_val*metric;    // handicap master node
                               else metr = metric;

                               route_value    +=    (C1*pow(((M_NT_row[r].nbr_node[s].filtered_range    -
metr)/metr), E1) + C2*M_NT_row[r].nbr_node[s].filtered_range/metr);

                               }    // end while
                         }
                      break;
                      }
                   }  // end for

               if (found == 1){
                  if (route_value < low_route_value){    // select lowest cost route
                     low_route_value = route_value;
                     low_route_node = M_NT_row[i].address;
                     min_route_cost[node_idx] = low_route_value;
```

```
                    }
                  }
              }   // end for

           Node[node_idx].found_by = low_route_node;
           M_Routing_table[node_idx][low_route_node-20] = low_route_node;

           fprintf(out5,"%d\t%f\t%f\t%d\t%d\n", node_idx, g_posn[node_idx].x, g_posn[node_idx].y,
                   Node[node_idx].myaddress, Node[node_idx].found_by);

           int idx1=low_route_node-20, idx2=node_idx;

           while (idx2 != m){          // update M_Routing_table
                 M_Routing_table[idx2][m] = Node[idx1].myaddress;
                 M_Routing_table[idx1][node_idx] = Node[idx2].myaddress;
                 idx2 = idx1;
                 idx1 = Node[idx1].found_by - 20;
                 }    // end while
           }   // end if
       }

////////////////////////////////////////////////////////////////////////////////////////////////////
//////

/* Master distribute route to Ping Node (update local routing table) function definition */
void distribute_route (int m, int node_idx){
       int j, idx1=m, idx2;

       if (node_idx == m){
          for (j=0; j<N; j++)
              Node[m].rt[j] = M_Routing_table[m][j];
          return;
          }

       while (idx1 != node_idx){
             idx2 = M_Routing_table[idx1][node_idx] - 20;
             for (j=0; j<N; j++){
                 Node[idx1].rt[j] = M_Routing_table[idx1][j];
                 Node[idx2].rt[j] = M_Routing_table[idx2][j];
                 }
             idx1 = idx2;
             }
       }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B ALTERNATIVE NETWORK DISCOVERY SOURCE CODE

```
// Alternative Network Discovery Code (ver 15 Aug 08)
// Ong Chee Wei

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 50                // No. of nodes
#define A 15                // Area A km by A km
#define cutoff_rng 4        // Cutoff range (km) for nodes to be considered immediate nbrs
#define ping_pwr_lvl 8      // Broadcast ping power level (1 to 8). Ranges defined in Ping function
#define P_echo_fail 0.0     // Probability (0 to 1) of echo failure in response to each ping
#define no_of_pings 1       // User defined no. of broadcast pings (1 or 2)
#define metric 1            // User defined metric (km) corresponding to preferred hop length
#define handicap_val 0.5    // User defined handicap value (0.5 or 1) at master node (RACOM buoy)
#define C1 1                // New Cost Fn weight associated with 1st term
#define C2 2                // New Cost Fn weight associated with 2nd term
#define E1 2                // New Cost Fn exponent associated with 1st term

struct posn {
       float x;             // node position
       float y;
       char address;        // pre-configured unique 8 bit node address
       char srl;
       };
struct posn g_posn[N];      // Global position structure

float g_rng[N][N];          // True Range lookup table b/w nodes

struct nbr {
       float range;
       char address;
       };

struct Nbr_table {
       struct posn myposn;
       char myaddress;
       char found_by;
       int rt[N];               // local routing vector
       struct nbr mynbr[N];
       };
struct Nbr_table Node[N];       // Node Neighbour Table

struct m_nbr {
       char address;
       float range1;
       float range2;
       float filtered_range;
       };

struct M_Nbr_table {
       char address;
       struct m_nbr nbr_node[N];
       };
struct M_Nbr_table M_NT_row[N];  // Master Neighbour Table

int M_Routing_table[N][N];       // Master Routing Table

int ping(int m, int ping_node, int pwr_lvl, int cell);       // Ping function declaration
void transfer(int m, int fr, int to, int cell);              // Tranfer function declaration
int aggregate(int m, int ping_node);                         // Aggregate function declaration
void determine_route(int m, int ping_node, int ping_ptr);    // Determine Route function declaration
void distribute_route(int m, int ping_node);                 // Distribute Route function declaration

float min_route_cost[N];         // min route cost table

/* Main Program */
main(){

       FILE *out0, *out1, *out2;
       out0=fopen("log.out", "w");
       out1=fopen("g_posn.out", "w");
```

```
        out2=fopen("rng_table.out", "w");
        FILE *out5;
        out5=fopen("graph.out","w");

        int node_i, node_j;       // counters

//-------------------------------------------------
        /* Global position to setup N nodes in A km by A km area */
        for (node_i=0; node_i<N; node_i++){
            g_posn[node_i].x = (float)rand()/RAND_MAX*A;
            g_posn[node_i].y = (float)rand()/RAND_MAX*A;
            g_posn[node_i].address = 20 + node_i;       // node address starts from 20

            Node[node_i].myaddress = g_posn[node_i].address;   // each node knows its address

            fprintf(out1,   "%d\t   %f\t   %f\t   %d\n",   node_i,   g_posn[node_i].x,   g_posn[node_i].y,
g_posn[node_i].address);
            }
        fclose(out1);

//-------------------------------------------------
//       /* Global position of N nodes read-in from input.in file */
//       FILE *in1;
//       in1=fopen("input0.in", "r");
//
//       for (node_i=0; node_i<N; node_i++){
//           fscanf(in1,"%f %f %d", &g_posn[node_i].x, &g_posn[node_i].y, &g_posn[node_i].srl);
//           g_posn[node_i].address = 20 + node_i;       // node address starts from 20
//           Node[node_i].myaddress = g_posn[node_i].address;   // each node knows its address
//                fprintf(out1,  "%d\t  %f\t  %f\t  %d\n",  node_i,  g_posn[node_i].x,  g_posn[node_i].y,
g_posn[node_i].address);
//           }
//       fclose(in1);
//       fclose(out1);
//-------------------------------------------------

        /* Compute true range between nodes i and j (stored in look-up table)*/
        for (node_i=0; node_i<N; node_i++){
            for (node_j=0; node_j<N; node_j++){
                g_rng[node_i][node_j]    =    sqrt((g_posn[node_i].x-g_posn[node_j].x)*(g_posn[node_i].x-
g_posn[node_j].x)
                                          +      (g_posn[node_i].y-g_posn[node_j].y)*(g_posn[node_i].y-
g_posn[node_j].y));
                fprintf(out2, "g_rng[%d][%d]\t %f\n", node_i, node_j, g_rng[node_i][node_j]);
                }
            }
        fclose(out2);

////////*** Network Discovery ***/////////
        int m, p, r, i, j;                  // m = Master node index, p = ping node index, r = route
node index
        int nodes_found=0, total_nodes_found=1, new_nodes_found[N];
        int initialise=0, ping_tries, ping_ptr, cell_idx;

        FILE *out3;
        out3=fopen("M_NT.out", "w");

        for (i=0; i<total_nodes_found; i++){
            if (initialise == 0){               // Master node initialisation
               m = 0;                           // Master node index set to 0
               p = m;                           // p = index of Ping Node
               r = m;                           // r = index of node to be routed to
               cell_idx = m;
               ping_ptr = 0;
               initialise = 1;
               }
            else r = M_NT_row[i].address - 20;              // index of subsequent Route Node

            determine_route(m, r, ping_ptr);                // Master determines route to Route Node

            distribute_route(m, r);                         // Master updates route to Route Node

            if (p == m){
            for (ping_tries=0; ping_tries<no_of_pings; ping_tries++){
                nodes_found = ping(m, p, ping_pwr_lvl, cell_idx);
                if (nodes_found > 0)
                   transfer(m, p, m, cell_idx);             // Ping node sends nbr data to M_NT
                }
            new_nodes_found[ping_ptr] = aggregate(m, p);    // Master aggregates data and expands M_NT
```

```
            total_nodes_found = total_nodes_found + new_nodes_found[i];
            ping_ptr++;
            }

            cell_idx = M_NT_row[i].address-20;
            for (j=0; j<new_nodes_found[i]; j++){
                if (ping_ptr >= N) break;
                p = M_NT_row[ping_ptr].address - 20;
                for (ping_tries=0; ping_tries<no_of_pings; ping_tries++){
                    nodes_found = ping(m, p, ping_pwr_lvl, cell_idx);
                    if (nodes_found > 0)
                       transfer(m, p, m, cell_idx);
                    }
                new_nodes_found[ping_ptr] = aggregate(m, p);
                total_nodes_found = total_nodes_found + new_nodes_found[ping_ptr];
                ping_ptr++;
                }
            }

        printf("\nNo. of nodes NOT discovered = %d\n\n", N-total_nodes_found);
////////*** End Network Discovery ***/////////

//---------------------------------------------
        /* Output M_NT */
        for (i=0; i<total_nodes_found; i++){
            j=0;
            fprintf(out3, "%d\t %d\t %f\t %f\t %f\n", M_NT_row[i].address,
                   M_NT_row[i].nbr_node[j].address, M_NT_row[i].nbr_node[j].filtered_range,
                   M_NT_row[i].nbr_node[j].range1, M_NT_row[i].nbr_node[j].range2);
            for (j=1; j<N; j++){
                if (M_NT_row[i].nbr_node[j].address != 0){
                    fprintf(out3, "%d\t %d\t %f\t %f\t %f\n", M_NT_row[i].address,
                            M_NT_row[i].nbr_node[j].address, M_NT_row[i].nbr_node[j].filtered_range,
                            M_NT_row[i].nbr_node[j].range1, M_NT_row[i].nbr_node[j].range2);
                   }
                   else break;
                }
            }

        fprintf(out0,"\nNo. of nodes NOT discovered = %d\n\n\n", N-total_nodes_found);

        fclose(out3);
        fclose(out0);
        fclose(out5);

        /* Output local routing vectors */
        FILE *out6;
        out6=fopen("M_RT.out","w");
        for (i=0; i<N; i++){
            fprintf(out6,"Node %d\t", Node[i].myaddress);
            for (j=0; j<N; j++){
                fprintf(out6,"%d\t", Node[i].rt[j]);
                }
            fprintf(out6,"\n");
            }
        fclose(out6);

        /* Output min route cost */
        FILE *out7;
        out7=fopen("min_route_cost.out","w");
        for (i=0; i<N; i++){
            fprintf(out7,"%d\t %f\n", Node[i].myaddress, min_route_cost[i]);
            }
        fclose(out7);
//-----------------------------------------------

        system("PAUSE");
        }

////////////////////////////////////////////////////////////////////////////////////////////////////

/* Ping function definition */
int ping (int m, int ping_node_idx, int pwr_lvl, int cell_idx){
        FILE *out0;
        out0=fopen("log.out", "a");
        int i, j, hit = 0;
        int exist, k = 0;                // k = index to Nbr table
        float power[8] = {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0};    // 8 ping ranges
        float ping_rng = power[pwr_lvl-1];
```

```
        int node_idx = m, broadcast_ping = 0, route_node;

        if (node_idx == ping_node_idx) broadcast_ping = 1;            // Master Node Ping
        else {
             while (node_idx != ping_node_idx){                       // forward the ping command
                   route_node = Node[node_idx].rt[cell_idx];
                   node_idx = route_node - 20;                        // index of route node
                   if (node_idx == cell_idx){
                      node_idx = ping_node_idx;                       // Cellular addressing
                      broadcast_ping = 1;
                      }
                   }
             }

        /* Broadcast Ping */
        float P_rand;

        if (broadcast_ping == 1){

        for (i=0; i<N; i++){
            if (i!=node_idx){
            P_rand = (float)rand()/RAND_MAX;   // generate random echo return probability at each node
            if (g_rng[node_idx][i] <= ping_rng){
            if (P_rand >= P_echo_fail){        // proceed with echo in response to broadcast ping
               for (j=0; j<N; j++){            // check if node i exists in Nbr table
                   if (Node[node_idx].mynbr[j].address == g_posn[i].address){
                      exist = 1;
                      Node[node_idx].mynbr[j].range = g_rng[node_idx][i];    // retain the latest range
                      break;
                      }
                   else exist = 0;
                   }
               if (exist == 0){               // node i is a new node
                  for (j=0; j<N; j++){
                      if (Node[node_idx].mynbr[j].address == 0){
                         k = j;
                         break;
                         }
                      }
                  Node[node_idx].mynbr[k].range = g_rng[node_idx][i];
                  Node[node_idx].mynbr[k].address = Node[i].myaddress;

                  fprintf(out0,"Pwr lvl %d\t Node %d\t discovered Node %d\t at Range = %f\n",
                          pwr_lvl, Node[node_idx].myaddress, Node[node_idx].mynbr[k].address,
                          Node[node_idx].mynbr[k].range);
                  }  // end of if
               hit = hit + 1;                 // no. of neighbours found
               }   // end of if
               }
            }
            }   // end of for
        return(hit);                          // return no. of neighbours discovered
        }    // end of if
        }

////////////////////////////////////////////////////////////////////////////////////////////////////

/* Transfer neighbour data from Ping node to Master node to be stored in M_NT function definition */
void transfer (int m, int fr, int to, int cell_idx){
        int i, j, k;
        struct Nbr_table temp_NT[N];
        int idx, route_node, nodes_found, exist;

        if (fr == m){
           printf("%d ", Node[fr].myaddress);
           temp_NT[to] = Node[m];         // at Master node
           }
        else {
             idx = fr;
             temp_NT[fr] = Node[fr];
             printf("%d ", Node[fr].myaddress);
             idx = cell_idx;
             temp_NT[idx] = temp_NT[fr];
             fr = idx;
             while (idx != to){
             printf("%d ", Node[fr].myaddress);
                   route_node = Node[idx].rt[to];          // reverse path transfer
                   idx = route_node - 20;
```

```
                temp_NT[idx] = temp_NT[fr];
                fr = idx;
                }
             if (idx == to) temp_NT[to] = temp_NT[idx];     // at Master node
             }
        printf("%d\n", Node[to].myaddress);


        for (i=0; i<N; i++){
            if (temp_NT[to].mynbr[i].address == 0){
               nodes_found = i;              // determine no. of nodes found by ping node
               break;
               }
            }
        for (i=0; i<N; i++){
            if (M_NT_row[i].address == temp_NT[to].myaddress){
               k = i;                        // row index in M_NT corresponding to ping node
               break;
               }
            }
        for (i=0; i<nodes_found; i++){
            for (j=0; j<N; j++){             // check if node i exists in M_NT_row[k]
                if (M_NT_row[k].nbr_node[j].address == temp_NT[to].mynbr[i].address){
                   exist = 1;
                   M_NT_row[k].nbr_node[j].range2 = temp_NT[to].mynbr[i].range;
                   break;
                   }
                else exist = 0;
                }
            if (exist == 0){                 // node i is a new node in M_NT_row[k]
               for (j=0; j<N; j++){
                   if (M_NT_row[k].nbr_node[j].address == 0)    // find empty row
                      break;
                   }
               M_NT_row[k].nbr_node[j].address = temp_NT[to].mynbr[i].address;
               M_NT_row[k].nbr_node[j].range1 = temp_NT[to].mynbr[i].range;
               }
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////

/* Master node aggregates data and expand M_NT function definition */
int aggregate (int m, int ping_node_idx){
        int i, j, k, exist, nodes_found, new_nodes = 0;
        float range, range1, range2;

        for (i=0; i<N; i++){
            if (M_NT_row[i].address == Node[ping_node_idx].myaddress){
               k = i;                // row index k in M_NT corresponding to ping node
               break;
               }
            }
        for (i=0; i<N; i++){
            if (M_NT_row[k].nbr_node[i].address == 0){
               nodes_found = i;      // determines no. of nodes found by ping node
               break;
               }
            }

        for (i=0; i<nodes_found; i++){        // Master node aggregates ranges & checks for new node
            range1 = M_NT_row[k].nbr_node[i].range1;
            range2 = M_NT_row[k].nbr_node[i].range2;
            if (range1 != 0 && range2 == 0) range = range1;       // rules to filter 2 ranges
            if (range1 == 0 && range2 != 0) range = range2;
            if (range1 != 0 && range2 != 0) range = (range1 + range2) * 0.5;
            M_NT_row[k].nbr_node[i].filtered_range = range;
            }

        struct m_nbr Temp;
        for (i=0; i<nodes_found; i++){         // Bubble Sort according to range
            for (j=0; j<nodes_found-i-1; j++){
                if (M_NT_row[k].nbr_node[j].filtered_range > M_NT_row[k].nbr_node[j+1].filtered_range){
                   Temp = M_NT_row[k].nbr_node[j];
                   M_NT_row[k].nbr_node[j] = M_NT_row[k].nbr_node[j+1];
                   M_NT_row[k].nbr_node[j+1] = Temp;
                   }
                }
            }
```

```
        for (i=0; i<nodes_found; i++){
            for (j=0; j<N; j++){
                if (M_NT_row[k].nbr_node[i].address == M_NT_row[j].address){
                   exist = 1;
                   break;
                   }
                else exist = 0;
                }
            if (exist == 0){            // M_NT_row[k].nbr_node[i] is a new node
                                        // apply cutoff range to determine immediate nbrs
               if (M_NT_row[k].nbr_node[i].filtered_range <= cutoff_rng){
                  for (j=0; j<N; j++){
                      if (M_NT_row[j].address == 0)     // find empty row in M_NT
                         break;
                      }
                  M_NT_row[j].address = M_NT_row[k].nbr_node[i].address;  // expand M_NT
                  new_nodes++;

                  M_Routing_table[M_NT_row[j].address-20][m]=Node[ping_node_idx].myaddress;
                  }
               }
            }
        return(new_nodes);
        }

////////////////////////////////////////////////////////////////////////////////////////////////

/* Master node determines route to Ping Node (node_idx) function definition */
void determine_route (int m, int node_idx, int ping_ptr){
        int i, j, k=ping_ptr, exist;
        FILE *out5;
        out5=fopen("graph.out","a");

        M_Routing_table[node_idx][node_idx] = Node[node_idx].myaddress;

        if (node_idx == m){         // initialise M_NT for Master node
           M_NT_row[0].address = Node[m].myaddress;
           Node[m].found_by = Node[m].myaddress;

           fprintf(out5,"%d\t%f\t%f\t%d\t%d\n", node_idx, g_posn[node_idx].x, g_posn[node_idx].y,
                  Node[node_idx].myaddress, Node[node_idx].found_by);

           return;
           }

        if (M_Routing_table[m][node_idx] != 0)   // check if route exist
           exist = 1;
        else exist = 0;

        int idx, route_node, low_route_node, r, s, found, found_route_node=0;
        float route_value, low_route_value=10000;

        if (exist == 0){
           for (i=0; i<k; i++){
               found = 0;
               for (j=0; j<N; j++){
                   if (M_NT_row[i].nbr_node[j].address == 0) break;
                   if (M_NT_row[i].nbr_node[j].address == Node[node_idx].myaddress){
                      if (M_NT_row[i].nbr_node[j].filtered_range <= cutoff_rng){
                         found = 1;
                         idx = M_NT_row[i].address - 20;

                         float metr;
                         if (idx == m) metr = handicap_val*metric;    // handicap master node
                         else metr = metric;

                         route_value  =  C1*pow(((M_NT_row[i].nbr_node[j].filtered_range  -  metr)/metr),
E1) + C2*M_NT_row[i].nbr_node[j].filtered_range/metr;

                         while (idx != m){
                                 route_node = M_Routing_table[idx][m];
                                 for (r=0; r<N; r++){
                                     if (M_NT_row[r].address == route_node)
                                        break;        // row index r in M_NT of route_node
                                     }
                                 for (s=0; s<N; s++){
                                     if (M_NT_row[r].nbr_node[s].address == Node[idx].myaddress)
                                        break;        // nbr index s in M_NT_row[r] of idx
```

```
                                    }

                                idx = route_node - 20;

                                if (idx == m) metr = handicap_val*metric;    // handicap master node
                                else metr = metric;

                                route_value    +=    C1*pow(((M_NT_row[r].nbr_node[s].filtered_range    -
metr)/metr), E1) + C2*M_NT_row[r].nbr_node[s].filtered_range/metr;

                                }    // end while
                            }
                        break;
                        }
                    }  // end for

                if (found == 1){
                   if (route_value < low_route_value){    // select lowest cost route
                      low_route_value = route_value;
                      low_route_node = M_NT_row[i].address;
                      min_route_cost[node_idx] = low_route_value;
                      }
                   }

                }   // end for

             Node[node_idx].found_by = low_route_node;
             M_Routing_table[node_idx][low_route_node-20] = low_route_node;

             fprintf(out5,"%d\t%f\t%f\t%d\t%d\n", node_idx, g_posn[node_idx].x, g_posn[node_idx].y,
                     Node[node_idx].myaddress, Node[node_idx].found_by);

             int idx1=low_route_node-20, idx2=node_idx;

             while (idx2 != m){          // update M_Routing_table
                   M_Routing_table[idx2][m] = Node[idx1].myaddress;
                   M_Routing_table[idx1][node_idx] = Node[idx2].myaddress;
                   idx2 = idx1;
                   idx1 = Node[idx1].found_by - 20;
                   }    // end while
             }    // end if (exist == 0)
          }

//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

/* Master distribute route to Ping Node (update local routing table) function definition */
void distribute_route (int m, int node_idx){
          int j, idx1=m, idx2;

          if (node_idx == m){
             for (j=0; j<N; j++)
                 Node[m].rt[j] = M_Routing_table[m][j];
             return;
             }

          while (idx1 != node_idx){
                idx2 = M_Routing_table[idx1][node_idx] - 20;
                for (j=0; j<N; j++){
                    Node[idx1].rt[j] = M_Routing_table[idx1][j];
                    Node[idx2].rt[j] = M_Routing_table[idx2][j];
                    }
                idx1 = idx2;
                }
          }
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] J. A. Rice, R. K. Creber, C. L. Fletcher, P. A. Baxley, K. E. Rogers, and D. C. Davison, “Seaweb Undersea Acoustic Nets,” in *Biennial Review 2001*, SSC San Diego Technical Document TD 3117, pp. 234-250, August 2001.

[2] E. M. Sozer, J. G. Proakis, J. A. Rice, and M. Stojanovic, “Shallow-Water Acoustic Networks,” *Encyclopedia of Telecommunications*, Wiley-Interscience, 2003.

[3] J. A. Rice, “Seaweb Acoustic Communication and Navigation Networks,” in *Proceedings of the International Conference on Underwater Acoustic Measurements: Technologies and Results*, Heraklion, Crete, *Greece*, June 2005.

[4] M. H. Hahn, “Undersea Navigation via a Distributed Acoustic Communications Network,” M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, June 2005.

[5] C. L. Fletcher, J. A. Rice, R. K. Creber, and D. L. Codiga, “Undersea Acoustic Network Operations Through a Database-Oriented Server/Client Interface,” in *Proc. IEEE Oceans 2001*, pp. 2071-2075, November 2001.

[6] C. L. Fletcher, J. A. Rice, and R. K. Creber, “Operator Access to Acoustically Networked Undersea Systems through the Seaweb Server,” in *Proc. IEEE Oceans 2003*, pp. 1-5, September 2003.

[7] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, “A Review of Routing Protocols for Mobile Ad Hoc Networks,” *Ad Hoc Networks (Elsevier)*, vol. 2, pp. 1-22, Jan 2004.

[8] K. Akkaya and M. Younis, “A Survey on Routing Protocols for Wireless Sensor Networks,” *Ad Hoc Networks (Elsevier)*, vol. 3(3), pp. 325-349, May 2005.

[9] I. F. Akyildiz, D. Pompili, and T. Melodia, “State-of-the-Art in Protocol Research for Underwater Acoustic Sensor Networks,” in *Proc. of the 1st ACM International Workshop on Underwater Networks* , pp. 7-16, 2006.

[10] A. K. Othman, A. E. Adams, and C. C. Tsimenidis, “Node Discovery Protocol and Localization for Distributed Underwater Acoustic Networks,” in *International Conference on Internet and Web Applications and Services*,” pp. 93-98, Feb 2006.

[11] G. Xie and J. H. Gibson, “A Network Layer Protocol for UANs to Address Propagation Delay Induced Performance Limitations,” in *Proc. IEEE Oceans 2001*, pp. 2087-2094, November 2001.

[12] B. A. Kerstens, “A Study of the Seastar Underwater Acoustic Local Area Network Concept,” M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, December 2007.

[13] R. J. Urick, *Principles of Underwater Sound*, McGraw-Hill, 1983.

[14] R. E. Francois and G. R. Garrison, “Sound Absorption based on Ocean Measurements: Part I: Pure Water and Magnesium Sulfate Contributions,” *Journal of the Acoustical Society of America*, vol. 72, no. 3, pp. 896-907, 1982.

[15] R. E. Francois and G. R. Garrison, “Sound Absorption based on Ocean Measurements: Part II: Boric Acid Contribution and Equation for Total Absorption,” *Journal of the Acoustical Society of America*, vol. 72, no. 6, pp. 1879-1890, 1982.

[16] R. F. W. Coates, *Underwater Acoustic Systems*, New York: Halsted Press, 1989.

[17] J. T. Hansen, “Link Budget Analysis for Undersea Acoustic Signaling,” M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, June 2002.

[18] J. C. Torres, “Modeling of High-Frequency Acoustic Propagation in Shallow Water,” M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, June 2007.

[19] J. A. Rice, V. K. McDonald, M. D. Green, and D. Porta, “ Adaptive Modulation for Undersea Acoustic Telemetry,” *Sea Technology*, vol. 40, no.5, pp. 29-36, May 1999.

[20] J. M. Kalscheuer, “A Selective Automatic Repeat Request Protocol for Undersea Acoustic Links,” M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, June 2004.

[21] S. P. Ouimet, M. J. Hahn, and J. A. Rice, “Undersea Communication Network as a UUV Navigation Aid,” in *Proc. IEEE Oceans 2005*, vol. 3, pp. 2485-2490, 2005.

[22] L. M. Ni and Y. Liu, “Communication Protocols,” in *Wireless Ad Hoc Networking: Personal-Area, Local-Area, and Sensory-Area Networks*, Auerbach Publications, 2007, pp. 25-63.

[23] H. A. Kriewaldt, “Communications Performance of an Undersea Acoustic Wide-Area Network,” M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, March 2006.

[24] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[25] R. Pinelli, and M. Coryer, “Seaweb Ad Hoc Discovery Process Using Teledyne Benthos Acoustic Modems – Functional Description,” Teledyne Benthos, North Falmouth, MA, USA, Tech. Rep., July 2008.

[26] D. J. Grimmet, “Message Routing Criteria for Undersea Acoustic Communications Networks,” in *Proc. IEEE Oceans 2007 - Europe*, pp. 1-6, 2007.

[27] “Decimal Degree to UTM Conversion using WGS 84 as Datum,” http://www.whimbrel.com/deg_to_utm3.html, accessed July 2008.

[28] W. Stallings, *Data and Computer Communications*, Prentice Hall, 2007.

[29] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing,” IETF RFC 3561, 2003.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Joseph A. Rice
   Naval Postgraduate School
   Monterey, California

4. Professor John C. McEachen
   Naval Postgraduate School
   Monterey, California

5. Professor Daphne Kapolka
   Naval Postgraduate School
   Monterey, California

6. RADM (Ret) Ray Jones
   Naval Postgraduate School
   Monterey, California

7. RADM (Ret) Rick Williams
   Naval Postgraduate School
   Monterey, California

8. Professor Matt Carlyle
   Naval Postgraduate School
   Monterey, California

9. Professor Don Brutzman
   Naval Postgraduate School
   Monterey, California

10. CAPT (Ret) Jeffrey Kline
    Naval Postgraduate School
    Monterey, California

11. Wendy Walsh
    Naval Postgraduate School
    Monterey, California

12. Bill Marn
SPAWAR Systems Center Pacific
San Diego, California

13. Chris Fletcher
SPAWAR Systems Center Pacific
San Diego, California

14. Bob Creber
SPAWAR Systems Center Pacific
San Diego, California

15. Doug Grimmett
SPAWAR Systems Center Pacific
San Diego, California

16. Lonnie Hamme
SPAWAR Systems Center Pacific
San Diego, California

17. Mike Blue
SPAWAR Systems Center Pacific
San Diego, California

18. Paul Grendron
SPAWAR Systems Center Pacific
San Diego, California

19. Mark Gillcrist
SPAWAR Systems Center Pacific
San Diego, California

20. William Macha
SPAWAR Systems Center Pacific
San Diego, California

21. Vincent McDonald
SPAWAR Systems Center Pacific
San Diego, California

22. Drew Mitchell
Naval Surface Warfare Center
Panama City, Florida

23. Joel Peak
Naval Surface Warfare Center
Panama City, Florida

24. Jody Wood-Putnam
Naval Surface Warfare Center
Panama City, Florida

25. Anthony Matthews
Naval Surface Warfare Center
Panama City, Florida

26. David E. Everhart
Naval Surface Warfare Center
Panama City, Florida

27. Dale Green
Teledyne Benthos, Inc
North Falmouth, Massachusetts

28. Rob Pinelli
Teledyne Benthos, Inc
North Falmouth, Massachusetts

29. Mike Coryer
Teledyne Benthos, Inc
North Falmouth, Massachusetts

30. Ken Scussel
Teledyne Benthos, Inc
North Falmouth, Massachusetts

31. Tom Drake
ONR 321CG
Arlington, Virginia

32. Dana Hesse
ONR 321MS
Arlington, Virginia

33. Bob Headrick
ONR 321OA
Arlington, Virginia

34. Dave Johnson
ONR 321
Arlington, Virginia

35. Tom Swean
ONR 321OE
Arlington, Virginia

36. Mike Wardlaw
ONR 321MS
Arlington, Virginia

37. Mike Traweek
ONR 321MS
Arlington, Virginia

38. CDR Patrick Lafontant
NAVSEA PMS NSW
Washington, District of Columbia

39. Mike Wood
Naval Special Warfare Group 3
Coronado, California

40. CDR Brad Mills
Naval Special Warfare Group 3
Coronado, California

41. Robert Mabry
US Special Operations Command
Tampa, Florida

42. Garry Heard
DRDC Atlantic
Dartmouth, Nova Scotia, Canada

43. Svein Haavik
Norwegian Defence Research Establishment (FFI)
Oslo, Norway

44. Roald Otnes
Norwegian Defence Research Establishment (FFI)
Horten, Norway

45. Tor Knudsen
Norwegian Defence Research Establishment (FFI)
Horten, Norway

46. Roger Birkeland
Norwegian Defence Research Establishment (FFI)
Horten, Norway

47. Ed Franchi
Naval Research Laboratory
Washington D.C.

48. RADM (Ret) W. G. Ellis
Pentagon
Washington D.C.

49. David Hughes
NATO Undersea Research Centre
La Spezia, Italy

50. Randy Unger
OASD Homeland Defense
Washington D.C.

51. LCDR Bjorn Kerstens
Defence Materiel Organisation
The Hague, Netherlands

52. LTC Ong Chee Wei
Republic of Singapore Navy
Singapore