UNIVERSITY OF CALIFORNIA

Los Angeles

Pushing the Limits of Wireless Sensor Networks - WSNs 20,000 Leagues Under the Sea

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Ruolin Fan

2017

ABSTRACT OF THE DISSERTATION

Pushing the Limits of Wireless Sensor Networks - WSNs 20,000 Leagues Under the Sea

by

Ruolin Fan
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2017
Professor Mario Gerla, Chair

In recent years, research in wireless sensor networks (WSNs) has made significant strides, facilitating a variety of applications in the Internet of Things, health, environmental monitoring, and military. Advances in wireless routing and delay tolerance design allow WSNs to achieve data acquisition in networking scenarios with intermittent link and time varying bandwidth. Creative power saving schemes were also developed to allow WSN nodes periodically power down. These advances allow researchers to deploy WSNs in a variety of challenging environments that have limited communication bandwidth with harsh channel dynamics, long delay, and high deployment costs. One such environment is under the ocean, where unconventional types of propagation medium such as acoustics and optics are the *only* viable types of communication medium.

This dissertation describes the design, implementation, and various other aspects of an underwater WSN system using software defined networking (SDN) principles that can be deployed on the ocean floor for purposes ranging from ocean exploration and oil drilling needs to search and rescue missions and military interests. The system consists of a flock of Autonomous Underwater Vehicles (AUVs) acting as mobile WS nodes and primarily relies on a centralized SDN controller doubling as the information hub and recharging station for the AUVs. We address systematic issues related to AUV deployment and networking, including power consumption, data transfer, and channel contention. To limit power consumption, we design two neighbor discovery methods that allow AUVs to turn off their radios and thus save power. For data transfer among nodes and channel contention, we explore the idea of delay-tolerant networking (DTN) and study various acoustic media access control schemes. We also develop a new underwater acoustic MAC protocol that maximizes throughput in good channel conditions and data transfer reliability in bad channel conditions. Simulations and experiments show that our protocol greatly outperforms existing underwater acoustic protocols in both speed and reliability. Along the process, we create and expand a public testbed allowing researchers to run underwater networking experiments in a water tank, in an emulator, and even in the open water for more realistic settings.

The dissertation of Ruolin Fan is approved.

Rajit Gadh

D. Stott Parker

Jack W. Carlyle

Mario Gerla, Committee Chair

University of California, Los Angeles

2017

To my parents . . .

who gave up a comfortable life in China

to emigrate to North America, enduring hardships

so that I may have a brighter future

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank God for bringing me to this point in my life, Who has been guiding me through every experience in my life to gain me for His purpose, to build me into His Body for His expression.

Next, I would like to thank Professor Mario Gerla, who advised me through these past five years and gave me encouragements to help me see through to the completion of this degree. Thanks also to Professor Kaigui Bian of Peking University for his guidance in our brief but successful collaboration over the summer of 2014.

Thanks to the UCLA Department of Computer Science for providing me with funding in my first year and many subsequent years of TA opportunities, and to the National Science Foundation (NSF) CISE Research Infrastructure (CRI), without whom the underwater project would not have been a possibility.

Thanks to all the good folks at the UCLA Marina Aquatic Center (MAC), Kristen Lockwood, Byron Pfeifer, etc., who amongst other things provided us equipment storage space as well as helping us with equipment deployment. Thanks also to everyone at the Marina del Rey Sheriff Station, Sergeant Brent Carlson and others, for accommodating all of our experiments in the marina.

Thanks to everyone in the UCLA Network Research Lab during my time there for providing me with support, both in research and in morale - Pengyuan Du, Josh Joy, Vince Rabsatt, Jihyoung Kim, Jorge Mena, Yu-Ting Yu (now at Qualcomm Research), You Lu (now at Google), etc., and the visitors, Li Wei of Michigan Tech University (I couldn't have done this without you!), Feng Zhou of Harbin Engineering University, Ciarán Mc Goldrick of Trinity College Dublin, and Torsten Braun of the University of Bern.

Last but not least, thanks to my parents, Kangmin Fan and Jie Shao, and to all the saints who are in Los Angeles, particularly those meeting in Hall 5. Thank you for your shepherding all these years!

VITA

2012	B.S. (Computer Science) Minor (Math), UCLA.
2012–2015	Teaching Assistant, Computer Science Department, UCLA.
	CS 111 Operating Systems
	CS31/32 Introduction to Computer Science
	CS33 Introduction to Computer Organization
	CS35L Software Construction Lab
2016–present	Research Assistant, Computer Science Department, UCLA.

CHAPTER 1

Introduction

Wireless sensors are increasingly becoming a ubiquitous tool as technology continues to advance in the 21st century. From everyday applications such as automatic doors and wireless health monitoring gadgets, to scientific research usage including tracking of animal movements and geological shifts, to even employments in military operations, wireless sensors have indeed become an indispensable, albeit oft overlooked, part of people's lives. These applications are only made possible by relentless efforts in the research community working on various aspects sensors, especially in the area of interconnectivity among a group or groups of sensors. Allowing many sensors to communicate with one another and deposit gathered information to the data sink.

More recently, significant efforts have been made to bring wireless sensors under water. The success of such an endeavor can have enormous benefits to humanity. The most obvious application of underwater sensor networks is oceanic exploration. Over 70% of the earth is covered with water, with less than 10% of it having been explored. There are great incentives to expand oceanic knowledge with regards to marine biology as well as oceanography. Placing a network of mobile sensors under the ocean can help explore ocean ridges, predict earthquakes and tsunamis, and track ocean life.

Subsea monitoring is another exciting application for underwater WSNs. Recently offshore oil drilling operations are becoming increasingly ubiquitous. These operations drill at the ocean floor 12,000 feet below the surface and have very complex setups, as seen in Figure 1.1. The combination of the complexity and the depth of deployment make monitoring every component, including the drill, highly crucial to ensure that everything is in working order and that spillages will not occur. For example, the BP oil spill disaster in 2010 that was

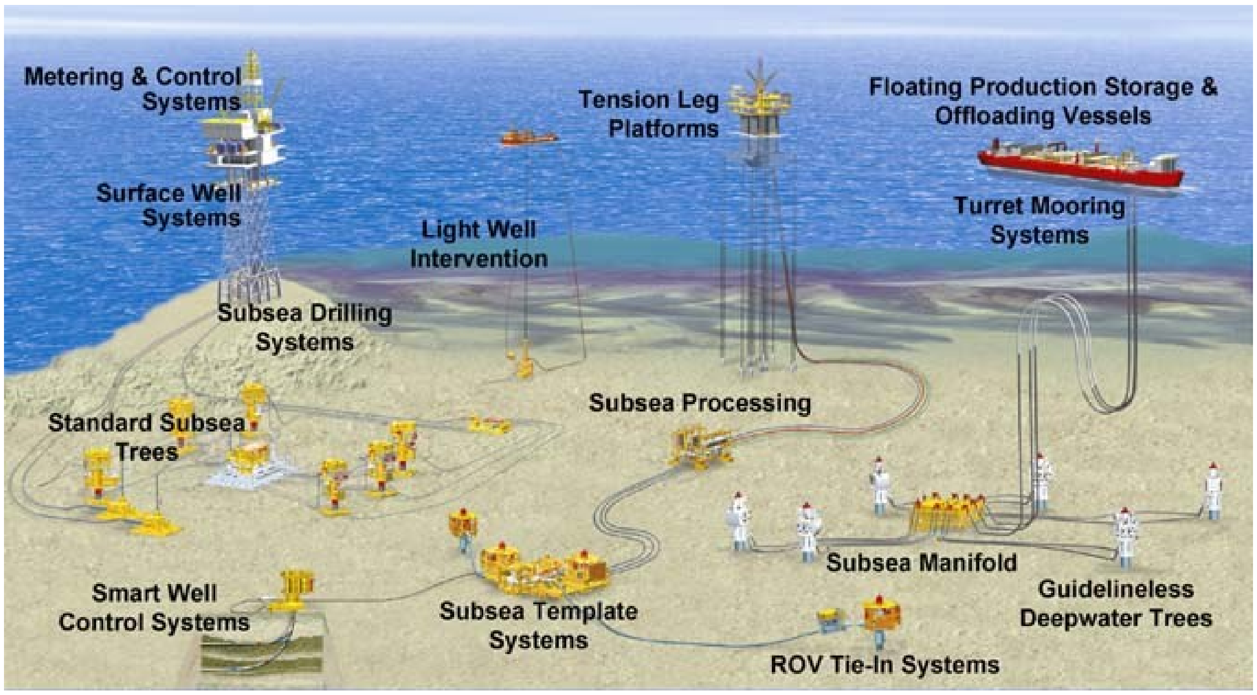


Figure 1.1: An underwater oil drilling system [FMC05]

one of the biggest spills in history and killed 11 people could have been averted had the crew detected the leaks sooner. This did not happen because the crew misinterpreted pressure data and was unable to infer the leak [Bly11]. Such would not be the case with a WSN actively monitoring for spills directly and reporting back to the surface constantly in real time. Similarly, communication cables that connect the continents to one another, the same cables used for the Internet, are deployed in the ocean. These cables require constant monitoring and regular maintenance because marine life and other natural actors will interact with them unexpectedly. Figure 1.2 depicts the layout of the communication cables on a world map on the left, showing the tall task of monitoring and up keeping every part of the cables. On the right side of the figure is an example of what may happen to one such underwater fiber optic cable - being bitten by a shark. All these factors make underwater WSN deployments extremely beneficial to monitor and possibly even automatically repair components of an offshore drilling operation or intercontinental cable line in the ocean. Permanently deployed WSNs would be capable, at the very least, of communicating detailed reports of the oil drill or the communication cables to their land-based headquarter. This gives great environmental

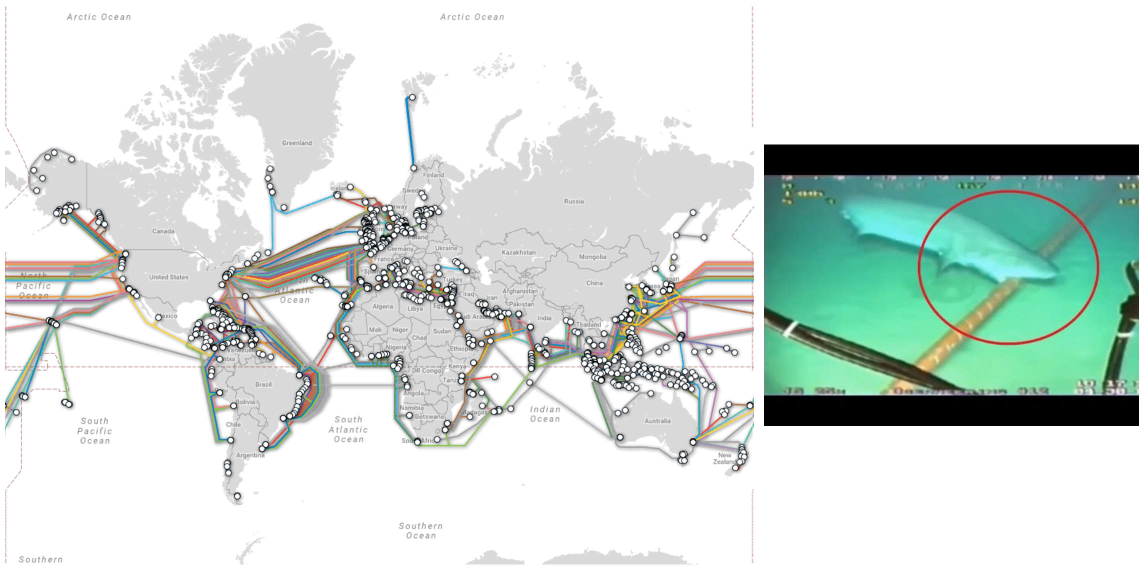


Figure 1.2: Left: communication cable laid under the ocean to connect the continents [Tel17]; right: a shark biting on one such cable [Lan14]

as well as financial incentives for underwater wireless communication technologies.

Finally, WSNs under the ocean can provide invaluable advantages in military operations, including search and rescue missions. As recently as 2014, the infamous MH370 disappeared over the ocean without a trace. To the general public, such disappearances in this day and age are simply inconceivable, but the missing aircraft was never found and efforts to recover it were eventually suspended. Employing WSNs can be of immense help to these efforts in curbing the costs and potentially increasing the speed of search. With all of these potential applications of underwater WSNs and more, the realization of such a technology will be full of impact.

1.1 Types of underwater communication medium

The main obstacle to wireless underwater communications is the high attenuation rate of most radio wave frequencies in salt water [SZT04]. Such phenomenon prevents the simple translation of traditional wireless methods in the air to under the ocean. This leads to considerations of other physical channel technologies, such as acoustics and optics.

1.1.1 The acoustic channel

Acoustics is the most prevalent method of communication method under water [PKL07]. Existing underwater acoustic radios are capable of transmitting at data rates up 500 Kbps, and some acoustic radios have transmission ranges of up to 30 km [CZ11]. Additionally, acoustic radios are generally omnidirectional and can pass through any non-absorbent objects, behaving similarly to radio waves in the air. Though the data rates leave more to be desired, the long propagation ranges and the ability to be omnidirectional are advantages for using acoustic radios. However, using acoustics as the physical medium comes with a huge side effect. The speed of sound stands at 343 m/s in the air and increases up to about 1,500 m/s in water, still less than 0.001% of the speed of light (about 300,000,000 m/s), the speed by which conventional wireless signals travel. As a result, underwater communications relying on the acoustic medium must deal with extremely high propagation delays, which leads to a multitude of other issues. One such issue is interference; conventional wireless technologies deal with this problem on the MAC layer using RTS-CTS packets [BDS94] with some limited success. In fact, even when packet collision occurs, the time it takes for retransmission is nearly instantaneous. Interference issues in underwater acoustics, on the other hand, are much more complicated. In order for the RTS-CTS packets to work as designed, they must be longer than the propagation delay to sufficiently cover the entire period of a potential collision. Since underwater acoustics have extraordinarily long delays, this collision-detection method becomes too cumbersome to be of much use [MS07]. In fact, due to the effects of the length of propagation delays in underwater acoustics and other differences between underwater and conventional networking, some have proposed completely redesigning the network stack to better serve the new scenarios, such as Aqua-net [PZC09].

1.1.2 The optical channel

Because of the clear limitations of underwater acoustics, much research is spent exploring other possible mediums of propagation, such as visible light, or underwater optics. Optical radios under water can transmit at data rates up to 2.28 Mbps [DR10], which is significantly faster than acoustic radios. Additionally, optical radios require less power and have very short propagation delay, since visible light travels at the speed of light. On the other hand, optical communications require line-of-sight between radios and are generally not omnidirectional. This results in very short ranges of transmission, up to about 100 meters [FBW10]. The unidirectional property of optical radios require relative localization among radios, presenting challenges, but at the same time also making duplex communication a possibility (ie. radios can send and receive packets at the same time). Finally, the line-of-sight requirement, though mainly seen as a disadvantage, can be valuable when covertness is a requirement, making optical radios ideal for covert military operations.

1.2 Related work

Most of the current efforts in underwater communications are very much focused on improving point-to-point throughput, regardless of the type of medium. On the acoustic front, [MS07], [PS06], [CW07], amongst others, proposed solutions in the MAC protocol to alleviate interference issues resulting from propagation delays. These ideas range from stricter regulating of transmission times (ie. having transmission slots, dictating the order of transmission, etc.) to modifying the MAC meta-packets (ie. changing the lengths of RTC-CTS packets, adding new meta-packets, etc.). The acoustic physical layer itself is also under scrutiny for improvement, as seen in [LZS07], [RDM07], and [LWB08]. The main goal here is to increase bit rates, the signal-to-noise ratio, and transmission distance. Other research related to underwater acoustics is mainly in the area of localization, which is useful not only for sensor mobility but also for optical communications. The basic technique for underwater localization is to use travel times of acoustic waves [CSL08][ASS00][Hah05]. In some cases, buoys equipped with GPS on the ocean surface may serve the same functions as GPS satellites to help localize the nodes below. Efforts improving the optical channel are more recent and are mainly done in the physical layer, a mix of signal processing and optical physics methods. [ABP10] gives insights for optical modulation; [AK09] suggests a way to avoid the line-of-sight requirement for optical radios by employing the total internal reflection (TIR) phenomenon.

Since Vasilescu et. al proposed [VKR05] in 2005, there have been very few, if any, work done on the overarching network system of underwater sensors. The system as described in [VKR05] consists of both static and mobile nodes. The majority of nodes are static sensors, with a few mobile nodes acting as data mules to transfer data optically to the sink. The nodes also use the acoustic channel, mainly for navigation purposes. Aqua-net [PZC09] describes an underwater network architecture, giving useful guidelines on application implementation and plugging in new protocols, but is more focused on the architecture of the network stack rather than the networking of a group of underwater nodes.

1.3 Roadmap of the dissertation

This dissertation summarizes our work in developing an overall working underwater system that addresses many systematic issues in AUV deployment and networking, such as channel contention, data transfer, and power consumption. This system will explore a variety of methods, including the standard acoustic approach, as well as an acoustic-optical combination approach for more speed and reliability. The rest of the dissertation is organized as follows:

Chapter 2 gives a detailed overview of our underwater WSN system and identifies some challenges related to it. Chapter 3 describes a geographically routed delay-tolerant network protocol for automobiles, whose principles can be adapted to the underwater environment. Chapter 4 gives two neighbor discovery algorithms that we can use to save power for the AUVs in our system. Chapter 5 describes a testbed that we developed to run better and more realistic underwater network experiments and the challenges we encountered along the way. Chapter 6 presents a MAC protocol in the acoustic channel that uses adaptive pipelining to increase throughput. Finally, we conclude with Chapter 7, which also gives an outline of future work for the underwater WSN system.

CHAPTER 2

The Software Defined Under-Water Network System

In this chapter we devise an underwater networking system using the software defined networking (SDN) approach that combines both the acoustic channel and the optical channel to allow each of their advantages to compensate for the other's lack, as outlined in the previous chapter. We use the SDN paradigm to build the system due to the centralized nature of communication in the underwater environment in general, where many exploratory AUVs send data back to a relatively few number of data sinks.

Software defined networking (SDN) is a relatively new networking paradigm aimed at flexibility and simplicity through high level abstractions by decoupling the data plane, where actual data are transmitted, from the control plane, where meta-data related to network functionality are transmitted. This is done by utilizing a centralized network controller that defines network behavior among all the other nodes. A comprehensive survey of ongoing research in the SDN area is provided in [KRE15].

This system is complete with a central network controller that principally handles routing and movement decisions for each AUV. The control plane is separated from the data plane through use of separate channels, and the modularity of the system components allow for flexible replacements of technologies including, but not limited to, propagation media and their associated MAC protocols, routing schemes, and applications depending on need. We implement a small-scale replica of our underwater SDN system in our remotely-accessible WaterCom[GMS15] testbed and, using it, we compare the performance of two state-of-the-art acoustic MAC protocols, Slotted FAMA [MS07] and UW- Aloha [PZC09] in a multi-hop scenario. Insofar as we know, this is the first comparison of underwater MAC protocols in a multi-hop scenario with physical modems.

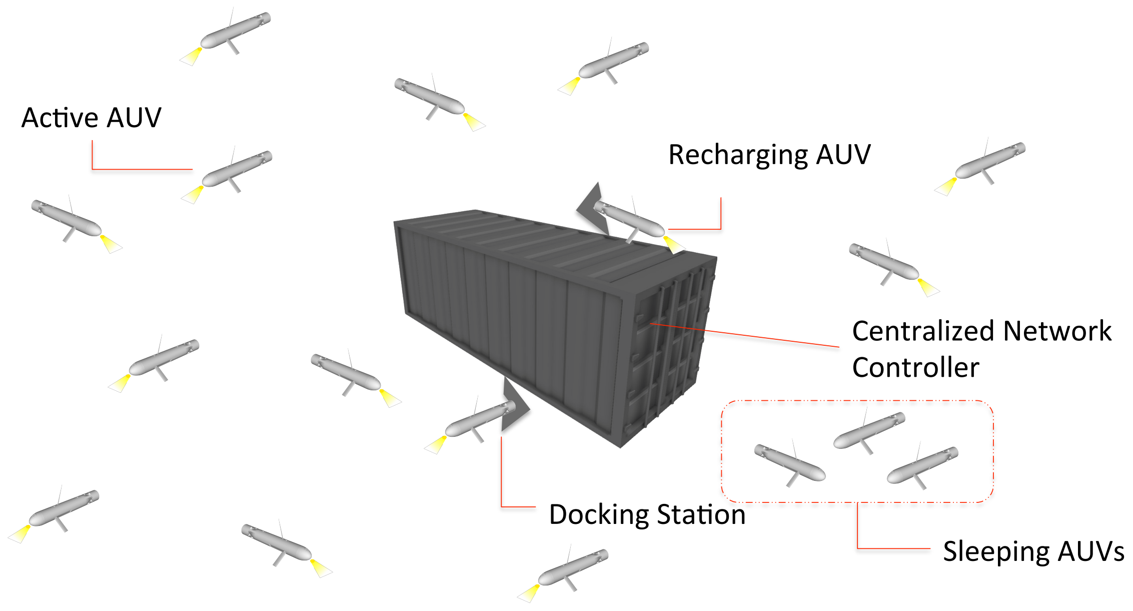


Figure 2.1: A diagram of our under-water SDN system. The network controller controls every aspect of each AUV, including their movements and routing information

2.1 System Overview

Our under-water SDN system is a network of mobile sensor nodes in the form of AUVs mainly confined to a specific area in the ocean where they will conduct their mission. At the center of the system is the static network controller doubling as a data sink for the mobile AUVs that can be placed on the ocean floor, close by or in the area of operation. This controller is charged with tasks such as controlling AUV movements to arrange the network topology, disseminating routing information, receiving exploration data collected locally by the AUVs, and acting as a recharge station for the AUVs. It may also act as a data gateway to the ocean surface through various means (wired link or other approaches). Figure 2.1 gives a visual representation of how such a system might work in practice.

2.1.1 Support for the acoustic PHY

One of the biggest challenges to any wireless sensor network is power conservation. The AUVs in our system, as mobile wireless sensors, are not exempt from this constraint. In order to save power, these AUVs can use lower-powered acoustic radios for transmission. However, this solution comes with trade-offs with respect to transmission range. While higher powered acoustic radios have transmission ranges in the kilometers, the power-saving radios that the AUVs have may only transmit at a range of hundreds of meters. As such, there is the need of multi-hop routing for both inter-AUV communication and AUV to data sink communication, to which we may apply SDN principles. The controller node, which supplies power to the AUVs, is able to use a higher powered acoustic radio that can reach all the AUVs with a single hop. Thus it can transmit control signals to all the AUVs to arrange the network topology, as well as disseminate routing information. To guarantee a high level of quality of service on the control plane, a specific acoustic channel is dedicated to these control signals so that there will be no interference from data packets. Once the AUVs receive these instructions they are able to arrange themselves and route data packets accordingly. With the controller knowing at all times the topology of the network and constantly updating all the AUVs with new routing information using long range acoustics, packets can be routed in the network as if the nodes are all stationery.

2.1.2 Support for the optical PHY

Our under-water SDN system is also capable of supporting the optical physical layer. Optical radios under water can transmit at data rates up to 2.28 Mbps [DR10], which is significantly faster than acoustic radios. Additionally, optical radios require less power and have very short propagation delay, since visible light travels at the speed of light. On the other hand, optical communications require line-of-sight between radios and are generally not omni-directional. This results in very short ranges of transmission, up to about 100 meters [FBW10], which is comparable with the power-saving acoustic radios. The uni-directional property of optical radios require relative localization among radios, presenting challenges, but at the same time also making duplex communication a possibility (ie. radios can send and receive packets at the same time). Finally, the line-of-sight requirement, though mainly seen as a disadvantage, can be valuable when covertness is a requirement, making optical radios ideal for covert military operations.

The tasks of the network controller in the optical PHY case are nearly identical to those in the acoustic PHY case. Here, the control plane continues to use the acoustic channel, but the data plane has now been moved to the optical one. Due to the need for localization for optical transmissions, the controller's instructions regarding each node's whereabouts become even more crucial. Once each AUV receive both routing information as well as the network topology information, they are able to direct their transmission beams toward the correct direction.

2.1.3 Traffic control

Since the majority of nodes in our network are mobile, the controller must give movement instructions as well as network communication instructions to each AUV to avoid physical AUV collisions. Moreover, because AUVs can recharge themselves close by the controller, it must also schedule AUV arrivals so that AUVs with higher priorities (AUVs that are soon to run out of power) can be served first. Other AUVs awaiting service are directed to "parking spaces" on the ocean floor. These AUVs may turn off their radios for the most part to save power, and can use WSN neighbor discovery methods similar to the one explored in Chapter 4 to reconnect with the SDN controller.

2.2 Implementation

We implemented parts of the acoustic version of this system in our WaterCom [GMS15] testbed. WaterCom allows anyone to set up and execute simple experiments remotely through our server reachable via the URL <apus.cs.ucla.edu>. Using a simple HTTP web

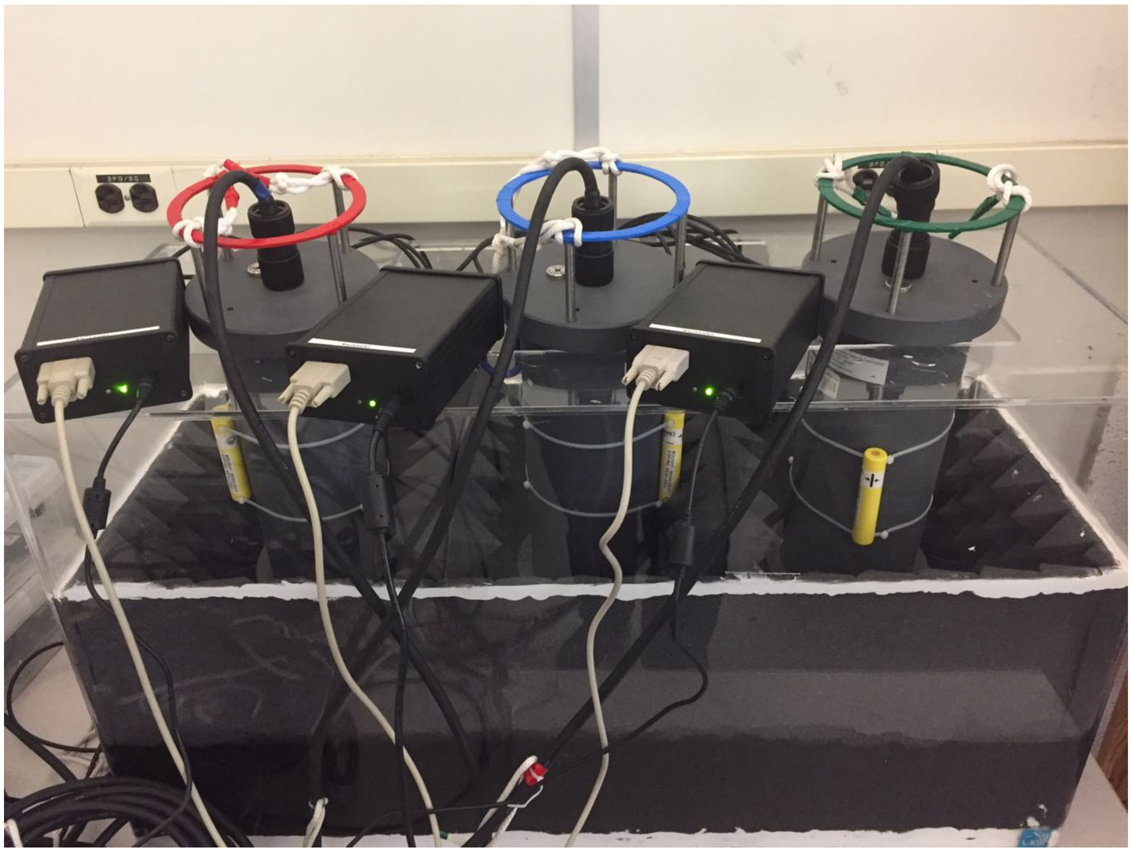


Figure 2.2: Our implementation of the under-water SDN scenario in the improved Water-Com [GMS15] testbed

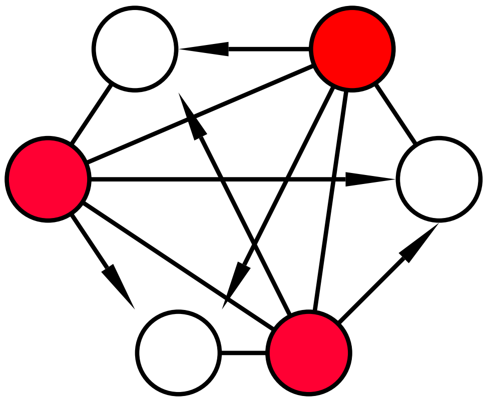


Figure 2.3: The resulting network topology in the testbed; the colored nodes represent the 13A models; arrow lines mean single-direction links

interface, one may submit a large variety of under-water communications tests remotely, which the server executes automatically. More complicated experiments may also be executed remotely, though these require SSH access.

There are six OFDM acoustic modems in total in our testbed. Three of which are AquaSeNT AM-OFDM-13A models that can send out stronger signals with communication ranges up to 5 km, and the rest are the educational version OFDM models that can communicate up 150m. Transducers and hydrophones of all modems are placed in a small tank as shown in Figure 2.2. Next, we lined the bottom and four sides of our water tank with foam to attenuate the acoustic waves propagating along the tank, as well as the reflections. Finally, using the same foam material, we divided the tank into three compartments with a pair of strong and weak modems in each compartment. By applying these foam material, acoustic wave could be attenuated faster than usual, which will help us to customize the link topology. This allows us to create a topology where the educational modems must go through at least one stronger radio to reach any of the other radios. The graphical topology layout is seen in Figure 2.3. The colored nodes represent the 13A models capable of transmitting stronger signals. Due to compartmentalization, each smaller modem can only be heard by the modem in the same compartment as itself, though they can hear all the stronger modems transmitting.

The six modems are connected with the WaterCom server, via which we remotely control the modems. The underwater protocol stack SeaLinx [LPC13] is employed to provide the networking services for experiments. Different protocol modules can be loaded in transport layer, network layer and link layer of SeaLinx to compare their performances with different configurations.

In this configuration, the SDN controller is the server for simplicity's sake, which has a wired connection to each modem, representing the sensors. We justify this in that in a real deployment scenario, control packets are on a different channel with very little chances of loses due to interference. Thus the only real drawback is the failure to take into account the propagation time, which will not be manifest in a tank environment at any rate. The server as the network controller is still able to disseminate routing information to each

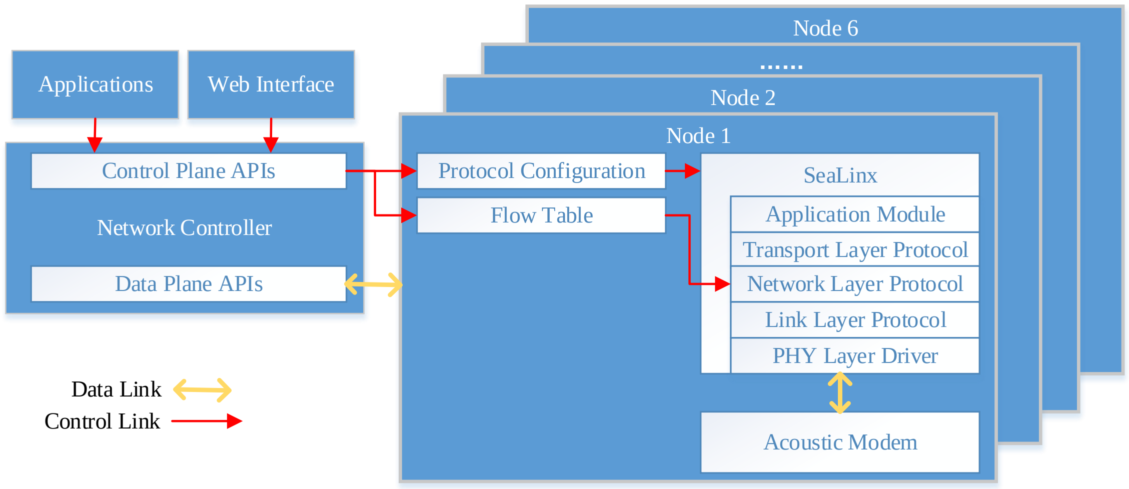


Figure 2.4: A diagram of the testbed in SDN terms

node, direct AUV movements in a simulated fashion by adjusting the sending power of each acoustic modem, and even swap out any of the protocols as needed. Figure 2.4 presents the separation of all the different functions of each component in our implementation of the underwater SDN system.

2.3 Comparing under-water acoustic MAC protocols in the under-water SDN testbed

With our under water SDN testbed, we compare the performance of two existing under water MAC protocols, Slotted FAMA [MS07] and UW-Aloha [PZC09]. Slotted FAMA is a underwater MAC protocol based on the FAMA protocol [FG95], which allows for carrier sensing via sending long enough RTS and CTS packets. Slotted FAMA improves upon FAMA by slotting transmission times, eliminating the asynchronous nature of the protocols and thus decreasing the lengths of the RTS and CTS packets to save power.

UW-Aloha is based directly on the pure Aloha protocol, where each node sends packets whenever needed, and retransmits collided packets later as required. However, the slow propagation speed of acoustics makes it impossible for nodes to determine whether a collision has

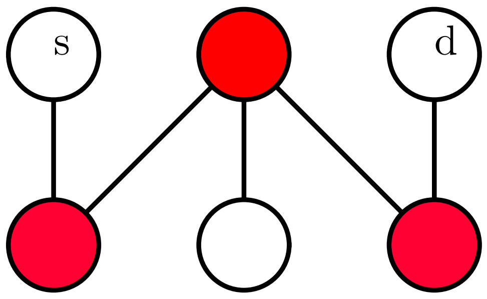


Figure 2.5: The network topology as used in the experiment; s denotes the source node, while d denotes the destination

occurred by the usual method of carrier sensing, especially when the transmission distances are long. UW-Aloha solves this problem by incorporating ACKs into the protocol to inform the senders of the transmission status.

We compare the two MAC protocols in an under-water multi-hop scenario using the physical setup from Section 2.2. To make things more interesting, we modify the network topology by omitting specific links from the topology by editing the routing table. First, we only use bi-directional links as potential routes, forcing each of the educational modems to both send and receive from the 13A model in the same compartment as itself. Next, we adjusted the transmission powers of the 13A models so that the ones on the ends of the tank cannot hear each other. The resulting topology is seen in Figure 2.5.

We dictate the far left educational modem to be the sending node, and the far right educational modem, the receiving node. A packet from the source must thus travel four hops to reach the destination, and all along the way, there are possible interferences among adjacent nodes. Under this condition we compare the overall throughput and packet delivery ratio of Slotted FAMA and UW-Aloha. For reference, we also test a third “dummy” MAC protocol that simply checks to see whether a given packet is for its corresponding node, and sends out packets as it receives them.

For all three protocols we run 5-minute test cases; for each test case we varying the length of each packet between 100 and 900 Bytes, and the packet generation rates between 0.3 and 1.2 packets per second. In our experiment, we find that all packets with the length of

900 Bytes experienced no throughput regardless of the protocol, and realized that the data packets cannot be properly decoded when the packet length is at 900 Bytes.

2.3.1 Custom protocol parameters and their impacts

The Slotted FAMA and UW-Aloha protocols have custom parameters that can be varied. Slotted FAMA has the parameter “guard time" to account for clock drifts among the different nodes, while UW-Aloha can vary its ACK timeout length. For Slotted FAMA, we only found that having too short of a guard time can negatively impact the packet delivery ratio, but otherwise it has very limited impacts at levels of 1000 and above. The ACK timeout length for UW-Aloha, on the other hand, is a different story. We vary the timeout between 10 and 40 seconds, and we see a consistent trend of the overall throughput due to fewer packets being sent out. This makes sense since the protocol is spending more time waiting for confirmation of packets received than generating more packets to transmit. Additionally, this leads to a higher packet delivery ratio on the MAC layer because there are fewer transmissions in general, thus less of a chance for collisions. However, the amount of throughput sacrificed does not justify this slightly higher packet delivery ratio.

To compare the three protocols, we choose the best performing set of transmissions according to the custom protocol parameter for each protocol. For Slotted FAMA, we use the dataset generated with guard time = 1000; for UW-Aloha, we use the dataset generated with ACK timeout = 100.

2.3.2 Comparing the MAC protocols amongst one another

We compare the packet delivery ratios and overall throughputs of the MAC protocols. We define packet delivery ratio here to be the number of packets received to the number of packets sent, both in the MAC layer. For example, packets dropped in the MAC layer before being sent out does not count. The overall throughput is the total amount of data received at the receiver over the 5-minute experiment period. As seen in Figure 2.6, it is clear that packet delivery ratio is negatively correlated with packet length for UW-Aloha

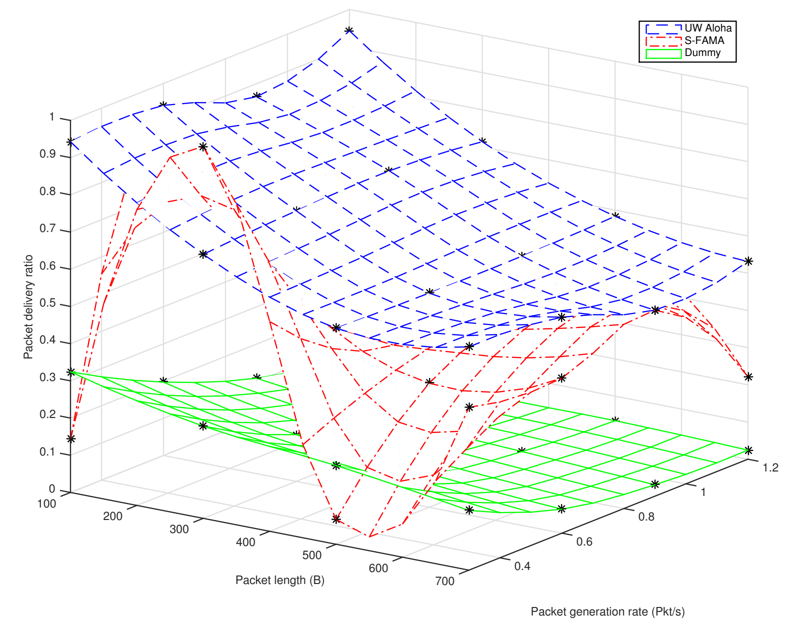


Figure 2.6: The packet delivery ratio of Slotted FAMA, UW-Aloha, and a dummy MAC protocol as influenced by packet length and packet generation rate

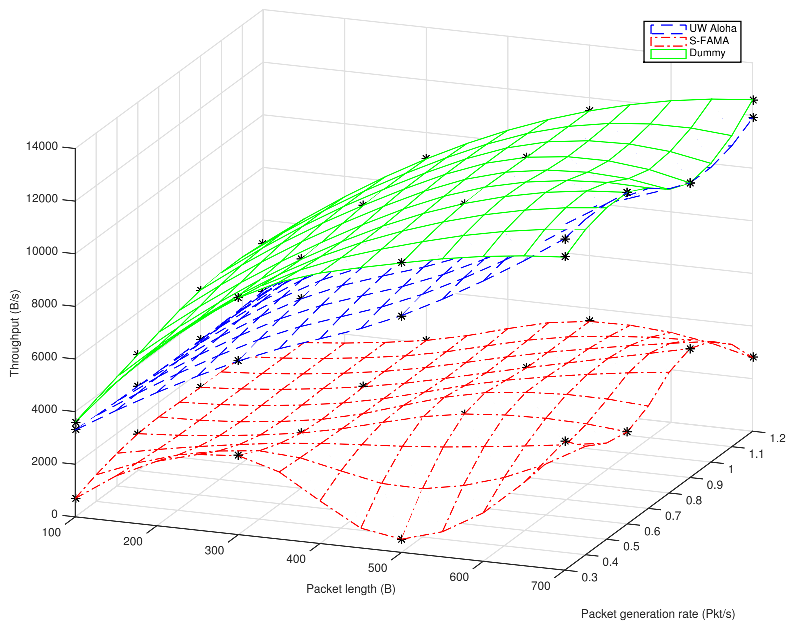


Figure 2.7: The overall throughput of Slotted FAMA, UW-Aloha, and a dummy MAC protocol as influenced by packet length and packet generation rate

and the dummy mac, while Slotted FAMA looks quite sporadic. On the other hand, the packet generation rate has a minimal effect on UW-Aloha, while negatively affecting the dummy protocol. This shows UW-Aloha's advantage in maintaining packet delivery ratio while increasing the throughput. In general, we observe that in terms of packet delivery ratio, UW-Aloha is more superior to Slotted FAMA, though Slotted FAMA is still better than the dummy protocol, which has no regulations at all.

Figure 2.7 shows how the throughput of each protocol is affected by packet length and packet generation rate. Unsurprisingly, all three protocols see a slight upward trend as the two parameters increase. In terms of comparison, the clear winner is the dummy protocol, since it does not wait for channel clearance or ACK signals, nor does it retransmit any packets but simply sends packets out as soon as it receives them. Interestingly, the high throughput of the dummy protocol actually attests to the performance of UW-Aloha, as it can reach throughput levels almost as high to the dummy protocol. Slotted FAMA has lowest throughput, making a huge trade-off in throughput for a slight gain in packet delivery ratio.

2.4 Conclusion

In this chapter we presented an underwater networking system for AUVs that leverages the SDN paradigm through use of a centralized network controller. Our design decouples the control and data planes by utilizing a dedicated channel for the controller to disseminate control packets to the AUVs. Using this design we were able to greatly simplify the task of AUV routing, even satisfying the stringent alignment requirements necessitated by the optical channel. We also implemented a small-scale replica of this underwater SDN system in our remotely-accessible testbed, which allows anybody in the world to develop new protocols and run experiments with ease. Using the testbed, we compared two existing underwater MAC protocols in a multi-hop network scenario. These experiments indicate that UW-Aloha gives high throughput whilst simultaneously guaranteeing relatively high packet delivery ratios.

CHAPTER 3

Routing in Delay Tolerant Networks

In under-water scenarios, localization is needed for navigation and point-to-point transmissions, especially when using the optical channel. Additionally, although acoustic signals have long enough ranges that a single-hop solution is usually sufficient, the short ranges but high transmission rates of optical beams make routing a necessity. Incidentally, the localization information can also be used for routing purposes at no extra cost. This chapter gives a geo-routing based protocol for VANETs on land, but the principle of it can be used under water as well.

Location-based routing algorithms are a class of routing algorithms that use locations of forwarding nodes to route packets. Unlike the more traditional link-state (LS) or distance-vector (DV) algorithms that rely on each node having some overarching information on the network topology [Huh04], location-based routing generally only require each node to have information on its immediate neighbors. This is a desirable property to have because it obviates the need for nodes to periodically flood the network with routing-related messages. Because of their scalability in this respect, these algorithms offer a very promising solution to the difficult problem of routing in vehicular ad-hoc networks (VANETs), where the network topology changes rapidly and bandwidth is limited [YMF06].

One of the first and the most well known of these routing protocols is the Greedy Perimeter Stateless Routing (GPSR) protocol [KK00], which sets the foundation for almost all subsequent location-based routing algorithms. In GPSR, each node in the network keeps a list of its immediate neighbors and their locations. Packets are thus routed based on *greedy forwarding* under normal circumstances, where the forwarding node transmits packets to the neighbor who is the closest geographically to the packets' destination. When no node closer to the destination than the current one can be found, the packet is said to have encountered a *local maximum* and must be routed around the perimeter via *perimeter forwarding*, which requires careful mapping of the nodes to form planar graphs so that routing loops can be avoided.

While the greedy forwarding method proposed by GPSR offers a great solution to routing in VANETs, the perimeter routing approach as a solution to the local maximum problem is inadequate [Lee10], as the highly mercurial network topology common to VANETs quickly outdates any planar graphs that may have been built from the nodes, much like routing tables in the DV and LS algorithms. GPCR (Greedy Perimeter Coordinator Routing) [LMF05], another location-based routing protocol that focuses more on urban VANETs, takes advantage of the urban layout that naturally forms a planar graph for its *recovery strategy* from a local maximum, and proposes the *restricted* greedy forwarding method, which prioritizes forwarding packets to urban cross-streets for better routing decision making.

To the end of avoiding local maximums and getting out of them in urban VANETs, many proposals are made [LHL07] [LLG09] [SLL04] [JSM07]. However, all of them make the assumption that despite mobility and the urban landscape, the network is always connected. Unfortunately, this is not the case for VANETs in reality, where **the network is often partitioned or disrupted due to mobility, giving no instant end to end routes**. According to [WBM07], network connections in VANETs are highly dependent upon inter-vehicle spacing, which is never constant, and that on freeways network disconnectivity generally heals on the order of seconds. In such situations, conventional geo-routing algorithms will cause large numbers of packets to be dropped. To address this problem, we propose RobustGeo, a location-based routing algorithm that takes the *store and forward* and *replication* approaches of delay-tolerant networks to explore the possibility of multiple paths to overcome such disruptions in VANET connectivity.

Although delay-tolerant networks (DTNs) were originally proposed for networks that experience very frequent and long periods of disconnectivity [JFP04], the DTN methods, with modification, can be used to deal with the shorter intermittencies in geo-routing with great benefits. One such method is the Data MULE (Mobile Ubiquitous LAN Extension)

[SRJ03] that exploits node mobility to deliver packets. While this on its own translates to unacceptable delay lengths in more conventional networks, we can, in the same spirit, rely on node mobility to recover from a disconnected scenario by saving the packet and actively looking for greedy forwarding routes at the same time. Another important aspect of DTNs is packet *replication*, as seen in Epidemic Routing [VB00] and Spray and Wait [SPR05]. The drawback to these methods is mainly in bandwidth overhead when too many packet replications occur. However, by using similar methods while limiting the number of replications, we can increase the packet delivery ratio and minimize delay.

With the combination of traditional geo-routing schemes and DTN approaches, RobustGeo indeed offers a more *robust* solution for geo-routing in urban VANETs. When the network connectivity is good, RobustGeo simply acts like any other state-of-the-art location-based routing algorithm. When the connectivity is intermittent, RobustGeo is not only able to withstand these disruptions and continue forwarding packets when the network heals, but also make use of the mobility of nearby neighboring nodes to even increase the healing rate (in the perspective of the packet stored by the node).

3.1 Design

RobustGeo combines the quick speed of geo-routing with the robustness of delay-tolerant networking. Under normal circumstances, greedy forwarding is used to quickly route packets to its destination. When, unavoidably, a local maximum situation arises, it can use a good recovery algorithm like the one suggested in [LLG09] to route around the perimeter while at the same time safeguard against complete disconnectivity by employing the delay-tolerant networking routing approach. In this approach, the nodes keep the packets in their disruption tolerant queue (DTQ), where the packets are opportunistically routed as available greedy forwarding neighbors are found. Periodically, nodes broadcast the packets inside their DTQ to their one-hop neighbors to explore other possible paths and increase the chances of finding a geo-routed path toward the destination.

3.1.1 Geo-routing

The geo-routing component of RobustGeo is compatible with all types of greedy-forwarding mechanism based on the one proposed in [KK00]. Generally, each node keeps a list of its immediate neighbors and looks for the neighbor geographically closest to the packet destination. Following these neighbors, the packets are forwarded greedily step by step until it reaches its destination. To combat challenges posed by urban grid layouts and high urban error rates, restrictive forwarding approaches as described in GPCR [LMF05], GpsrJ+ [LHL07], and TO-GO [LLG09] can be employed.

Likewise, RobustGeo can utilize any of the previously existing solutions for perimeter forwarding to attempt local maxima recovery.

3.1.2 Disruption-tolerance

The DTN component of RobustGeo exploits the mobile nature of VANETS. In such a highly mobile environment, it is beneficial to not give up on a local maximum node so quickly since topology can evolve quite rapidly. In many cases, a local maximum at one moment in time may very well no longer be a local maximum in the next simply because of mobility. Therefore unlike conventional geo-routing algorithms, when the perimeter forwarding phase is entered, RobustGeo routes a *replica* of the packet around the perimeter rather than the original packet itself. It then pushes the original packet into the node's DTQ, essentially turning the node into an “intelligent" data MULE [SRJ03] that looks for greedy forwarding neighbors and at the same time further packet replication by periodically broadcasting to its one-hop neighbors. In this way, RobustGeo recovers from a local maximum situation by entering DTN mode in addition to perimeter forwarding.

3.1.2.1 Perimeter forwarding with packet replication

When a node encounters a local maximum, it sends out a packet replica for perimeter forwarding, stores the original in its DTQ, and continues to actively try and look for nodes that are closer to the destination than the local maximum. If found, the node has successfully recovered greedy forwarding via DTN means.

Meanwhile, as the replicated packet is routed around the perimeter it records each node it visits as breadcrumbs. If a greedy path is recovered, a receipt can be sent back to the originator using the breadcrumb path so that the originator can remove the packet from its DTQ and abort the broadcasting count-down time. Otherwise, the perimeter-routed packet can continue to be routed by the right-hand rule until either a greedy route is found, or ends up back at the originator or reaches its TTL. In both of the latter cases the packet is discarded. Note that packet replication due to perimeter forwarding happens only once at the original local maximum node. This prevents the packet from replicating out of control as it is forwarded around the perimeter.

If the original packet and its replicas both manage to find a greedy forwarding path, two cases should be considered. In the first case, as illustrated in Figure 3.1, multiple different paths to the destination are found. This arguably improves the scenario, since having multiple paths to the destination causes the intermittent network to be more robust. In the second case (Figure 3.2), both packets are forwarded greedily onto the same path. While on the surface this may seem problematic, it is not a great cause for concern since in the unlikely event that this happens, the receiving node can simply drop one of the duplicates and continues to forward the packet onward. However, this does mean that the packets should be uniquely identified so that identical packets can be easily discovered. A way to realize this is to use a large sequence number in combination with the sender's identifier (like an IP address).

The decision to replicate the packet when doing perimeter forwarding is due to the fact that in urban scenarios, the local maximum situation often occurs because of complete network partitioning. When this is the case, no amount of perimeter forwarding can bring the packet to its destination. Fortunately, these situations are generally not permanent because of mobile nature of VANETs. An example of such an occurrence is when there are gaps in vehicular traffic caused by something as simple as a long red light. Traditional perimeter forwarding solutions in these situations will cause the packet to be dropped when it either

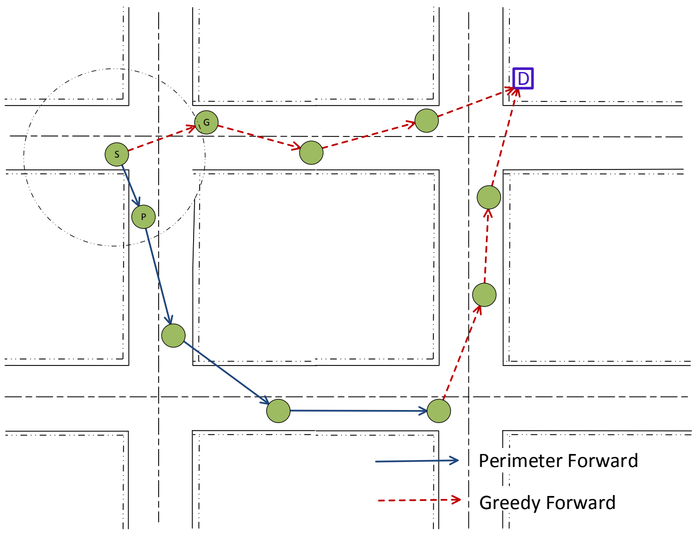


Figure 3.1: Greedy forwarding and perimeter forwarding finding two different paths. The greedy forwarding recipient node G moves into the source node S's range after S has perimeter forwarded the packet to node P.

reaches its TTL or end up back at its originator [KK00] [CLG10]. With RobustGeo, however, because the local maximum node keeps the perimeter forwarded packets, connectivity that is reestablished a little later can allow greedy forwarding to resume. Although this will cause the network to experience delays, the receiver is still able to receive the message eventually rather than experience complete disconnectivity.

3.1.2.2 Single-hop broadcasting to explore multiple paths

Packets that are not replicas of any previous broadcasts in the DTQ are scheduled to be one-hop broadcasted periodically to explore multiple paths, which potentially increases the delivery rate and decreases latency. Although the node is still continuing to look for greedy routes for the packet, once the packet was broadcasted, it essentially switches into "full DTN mode", relying more on node mobility to recover the next greedy routing path.

When a node receives a broadcasted packet, it first makes sure that it does not already

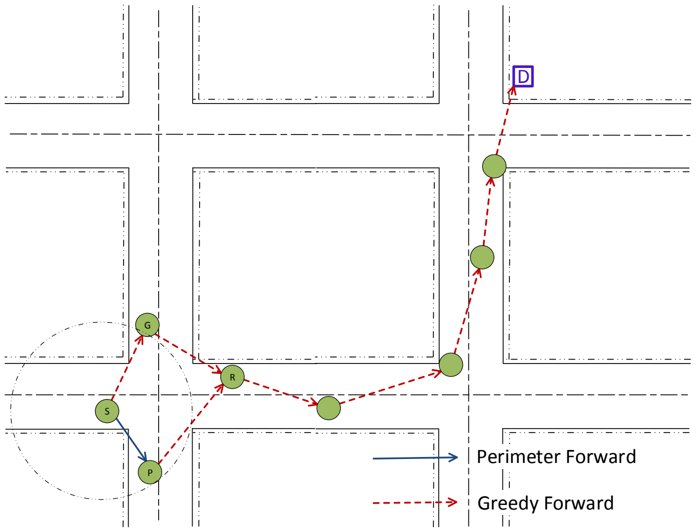


Figure 3.2: Greedy forwarding and perimeter forwarding finding the same path. Once again, the greedy forwarding recipient node G moves into the source node S's range a moment after perimeter routing is done by S. The two identical packets both pass through node R and the latter one is dropped

have the packet in its DTQ. Then it adds the packet into its DTQ, where the node can repeatedly attempt to greedy forward it.

The single-hop broadcasting technique is beneficial in situations where node density is sparser, taking advantage of the replication effects in delay-tolerant networking [Zha06]. In RobustGeo, we choose to replicate the packets using single-hop broadcast because it has a high replication-to-transmission ratio in that a packet can reach all the neighboring nodes with just a single transmission.

Since packet replication increases bandwidth usage, it is important to consider the length of the period in between broadcasts. Set too long, the node can lose opportunities to send to neighbors who are good data MULE candidates; set too short, the network would be saturated with replicated packets. With this in mind, we configure the period to be 6 seconds, which we believe is short enough to not miss most of the potential neighbors, yet long enough that it does not overload the network (see Section 3.2). We believe that most of the potential neighbors would not be missed because vehicles rarely travel faster than 20m/s (about 45mph) in an urban area. Taking incoming traffic into account, the relative speed can be up to 40m/s. Therefore 6 seconds is equivalent to 240m, still within most broadcasting ranges. Additionally, we set this as an adjustable parameter depending on needs. For example, if the target scenario is a very sparse network, then it would be beneficial to make this period shorter. In all, due to the relatively long broadcasting period, momentary interruptions in the network will not trigger one-hop broadcasts.

To further limit the number of packet replicas, a packet that was once received via broadcasting can never be broadcasted again. This is easily done by marking a single bit in the packet header and checking that bit whenever a packet is about to be put into broadcast mode. Finally, when the first copy of the packet, be it a replica or the original, reaches its destination, the destination can send out an *active receipt* [HA06] to clean up all of the packet replicas that are still in the network.

3.1.2.3 Scheduling

When a local maximum situation occurs, the node stores the packet into its DTQ, and packets bound for other destinations with available next-hop nodes continue to be forwarded normally. Every time the node receives a new beacon, be it from any of its existing neighbors or from a new neighbor, the node would check its DTQ to see if any of the packets can now be routed.

The DTQ is similar in function to a normal FIFO queue, in that packets which are enqueued earlier generally get dequeued earlier as well. However, the DTQ is not necessarily dequeued from the head all the time. Whenever a node receives a new beacon from its neighbor, potentially marking a new route, RobustGeo attempts to dequeue from the DTQ by attempting to greedily route to the destination every packet in the queue, beginning with the packet at the head. When a route is found for a packet, that packet is dequeued to be sent out. This speeds up average packet delivery time and keeps the queue length short.

To prevent a large number of packets in the DTQ from suddenly flooding the network,

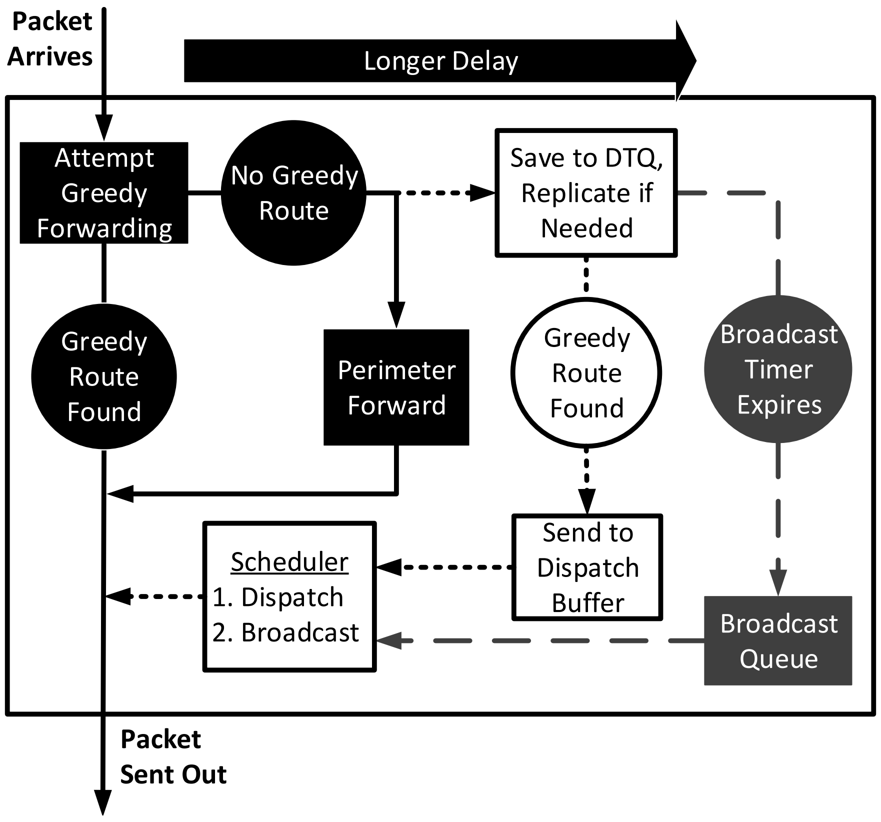


Figure 3.3: RobustGeo processing a packet. Normal packets use all paths as necessary; Broadcasted packet replicas do not traverse the long-dashed grey path; perimeter forwarded packet replicas only stay on the solid black path

RobustGeo allows only one packet to be sent out for a set period of time. For example, it may be decided that the backlogged packets should only take up about 1Mbps of the overall bandwidth, so a maximum of 1 packet can be sent out every millisecond. However, since the beaconing period is much longer than that, a dispatch buffer is introduced to manage the send rate. Therefore, the DTQ dequeues all eligible packets into the dispatch buffer up to a maximum of $send\ rate * beaconing\ period$ packets each beaconing period. Then at every $\frac{1}{send\ rate}$ period, RobustGeo sends out a packet from the dispatch buffer.

Packets marked for one-hop broadcasting are not sent out immediately either. Rather, RobustGeo adds the packet into a separate broadcasting queue and continues to look through its DTQ for other packets to be greedily forwarded. Additionally, the packet marked for broadcasting still remains in the DTQ for more future attempts at greedy forwarding recovery. If the recovery attempt is successful, the packet is removed from both the DTQ and the broadcasting queue. On the other hand, if the packet was broadcasted, it is removed from the broadcast queue only but kept in the DTQ, with the broadcasting timer reset to begin a new period of broadcasting.

Having two queues with different priorities, we employ our dispatch buffer to enforce these priorities. The broadcasting packets are never added to the dispatch buffer. Instead, the dequeuing method for the dispatch buffer dequeues a packet from the broadcast queue only if the dispatch buffer is empty. As a result, packets are only broadcasted when there are no more geo-routed packets to be sent out in the beaconing period. Figure 3.3 gives an overall view of the components of RobustGeo working together inside a node upon a packet arrival.

3.2 Analysis

Packet replication is a tricky parameter in ad hoc networks. In a lossy and especially delay tolerant environment, replication can provide substantial benefits in that it increases the delivery rate and decreases delay [Zha06]. Unfortunately, replication is a double-edged sword, as it also introduces bandwidth overhead into the network. Therefore as a protocol that stresses replication, it is important to see just how much overhead replication brings into the network.

In RobustGeo, packets are only replicated when a local maximum is reached. When the node is attempting to route the packet via perimeter routing, it introduces a single packet replica. If the recovery time is long, the packet can be replicated many times due to periodic broadcasting. The overall number of packets replicated as the result of a single local maximum is therefore

$$rep_1 = mn\lfloor \frac{t}{\pi} \rfloor + 1, \tag{3.1}$$

Where m is the number of packets in the network that experience local maximum recovery exactly once, t is the average local maximum recovery time in seconds; π is the broadcasting period of each node, and n is the number of neighbors that receive the broadcasted packet for the first time.

However, not every replicated packet ends up finding a greedy route. Packets that do not find alternate paths are ultimately dropped as they time out, and perimeter-routed replicas that find a greedy route cancel out its original. For this reason, we assign probabilities to the replicated packets to indicate that they actually remain in the network.

Suppose that each broadcasted packet finds an alternative route with probability P_b, and the perimeter-routed packet finds an alternative route without being able to inform its originator with probability P_p, then the number of replicated packets produced from a single local maximum recovery with these probabilities take into account is

$$rep_1 = mn\lfloor \frac{t}{\pi} \rfloor P_b + P_p. \tag{3.2}$$

Multiplying P_b to the first term takes into account that the only replicated packets that remain are those who are forwarded to neighbors that eventually find a greedy geo-forwarded route. The value 1 from Equation 3.1 is replaced with P_p to take into account that the packet replicated from perimeter routing ends up remaining in the network at the probability P_p.

If the distance between the sender and receiver is long, a packet may experience multiple cases of local maximum. Each time it happens, more packets are replicated. In RobustGeo, no packet replicas from broadcasting can be broadcasted again, but they may still replicate via perimeter- routing. We now calculate the number of replicas produced from the second time that m packets encounter local maximum recovery:

$$rep_2 = mn\lfloor \frac{t}{\pi} \rfloor P_b + P_p(mn\lfloor \frac{t}{\pi} \rfloor P_b) + P_p + P_p^2. \tag{3.3}$$

The first term, $mn\lfloor \frac{t}{\pi} \rfloor P_b$, is the number of packet produced from broadcasting the original packet. The second term, $P_p(mn\lfloor \frac{t}{\pi} \rfloor P_b)$, is the probabilistic number of packets produced from broadcasting the perimeter-routed packet. P_p is the replica from perimeter-routing the

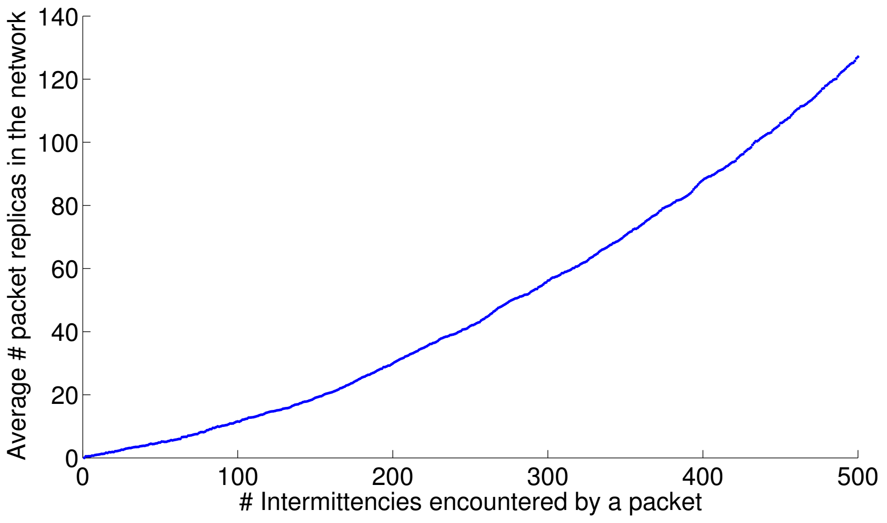


Figure 3.4: The average number of packet replicas in the network vs. the number of intermittencies encountered by a packet, where the intermittency length t is modeled as a Poisson random variable with $\lambda = 3$. All cases of intermittencies are independent of one another.

current original packet, and P_p^2 is the replica from perimeter-routing the replicated packet produced from the first time that perimeter-routing was performed.

Since P_p is a probability with value between 0 and 1, as it gets higher powers it becomes more negligible and is disregarded. Further, each time the packet is perimeter-routed, a new *broadcastable* packet is replicated from perimeter routing with probability P_p. The next time that the local maximum is encountered, all of these replicated packets are eligible for broadcasting. Therefore, the total number of replica packets in the network as m original packets are routed through K local maximums is approximately

$$rep \approx \sum_{k=1}^{K} mn\lfloor\frac{t}{\pi}\rfloor P_b(1 + (k-1)P_p) + P_p. \tag{3.4}$$

Figure 3.4 shows the number of replicas a single packet accumulates as it is routed in the network. It describes an urban scenario with short but frequent intermittencies. In this scenario, the recovery time is assumed to take 3 seconds on average, and the broadcasting period is 6 seconds. We also assume that the node has on average 10 new neighbors to broadcast to each time that it broadcasts. We also set P_p to be 0.01, a very low number because we assume that for the majority of times, the local maximum occurs due to brief network partitions and so it fails. For a similar reason, we set P_b to be 0.2 because we assume that most of the paths found through broadcasting are not unique. Because the broadcast rate is a floor function, we assume that the length of recovery is roughly a Poisson distribution with $\lambda = 3$[1] (for the 3 seconds of average recovery time) so that the results are not just summations of P_p.

We see that the number of packet replicates increase slowly with the number of intermittencies it encounters. At 100 intermittencies, the average number of replicas is less than 5 per packet. Thus packet replication is manageable.

3.3 Evaluation

We evaluate RobustGeo's performance by simulating three scenarios using the NS3 simulator, with the following parameters:

- PhyMode: OFDM 36Mbps
- Propagation Delay Model: Constant Speed
- Propagation Loss Model: Friis
- WiFi Standard: 802.11a
- Wifi Mac Type: Adhoc Wifi Mac
- Transport Protocol: UDP
- Application: CBR Client-Server Pair (20Kbps)
- Simulation Time: 200s

We begin with an artificially set-up scenario mainly to showcase the major features of RobustGeo, followed by two progressively more realistic scenarios, first with a realistically

[1]we can use a discrete pmf in this case since the output values we are concerned with are discrete

simulated mobility model running on an actual map of Washington D.C, then with an actual trace of taxi cabs in the San Francisco area. We believe that these three scenarios give a clear view of the superiority of RobustGeo over the other three competing routing algorithms.

To highlight the effectiveness of RobustGeo in intermittent situations, we treat all cases of local maximum in the simulation as network partitioning by not allowing perimeter routing in any of the algorithms we evaluate. For each scenario, we compare RobustGeo against *pure geo-routing*, which simply drops the packet that it cannot immediately route, *geo-routing with a DTQ* that improves delay tolerance but without packet replication, similar in behavior to GeoDTN+Nav [CLG10], and *geo-routing with controlled flooding*, where nodes can only pass along the packet to 5 unique neighbors before dropping the packet (Epidemic Routing [VB00] with restrictions).

We base our comparisons on the following metrics: packet delivery rate, average delay, and overall traffic per packet received. While packet delivery rates and average delay are obvious metrics to measure, we believe that the overall traffic per packet received is equally important because packet replication is a major feature of RobustGeo. Finally, using data gathered from the San Francisco cab trace, we analyze the number of intermittencies packets generally encounter in the network, as well as the number of times each packet is broadcasted, to further cement our claim that RobustGeo is scalable in terms of packet replication.

We begin by testing a synthetic scenario as illustrated in Figure 3.5. This 5-part system consists of groups of three static nodes and two groups of mobile nodes, with a total of 11 nodes. The two clusters of static nodes on the left form a stable connection, while the single group of static nodes on the right relies on the two mobile groups to receive messages. As the mobile nodes move around there are short intervals they form a bridge of network connection between the source and the destination. Most of the time, however, the network on the left side is completely partitioned off from the right. We believe that this is a scenario best fit for RobustGeo since the source is able to reach its stably connected neighbor via periodic broadcasting and thus make use of both routes as each become available. Such a path cannot be realized with perimeter routing because when the mobile node on the slant is connected to the static relay, it is not connected with anything else, so all of the perimeter

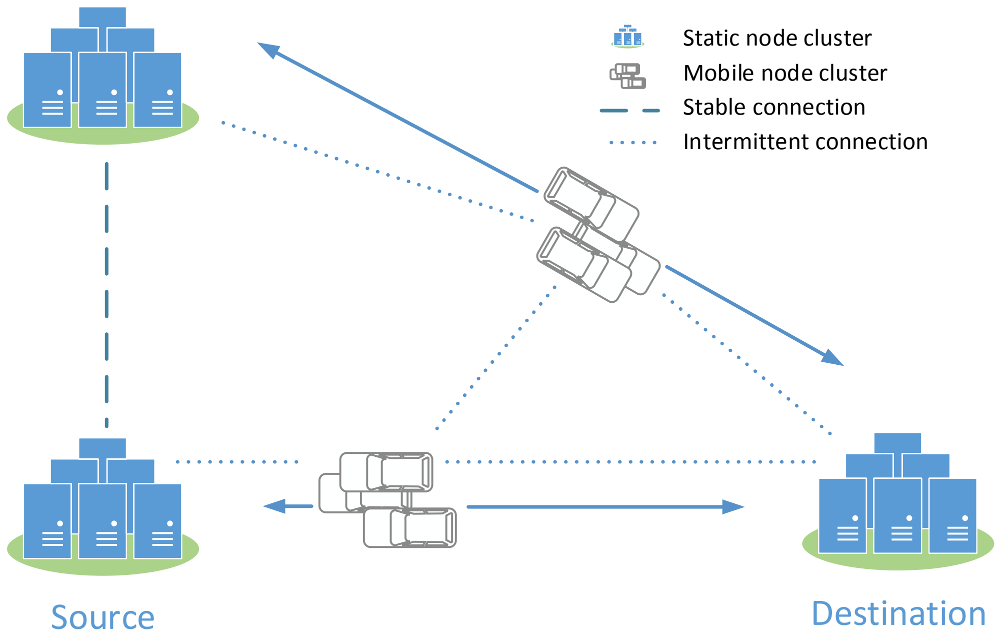
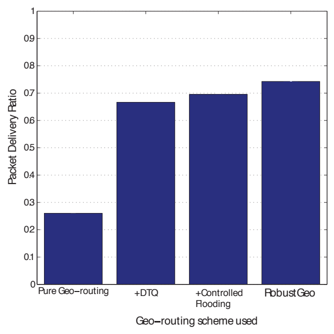


Figure 3.5: 5 groups of nodes in an intermittent situation; arrows denote node mobility, and dotted lines denote network connection

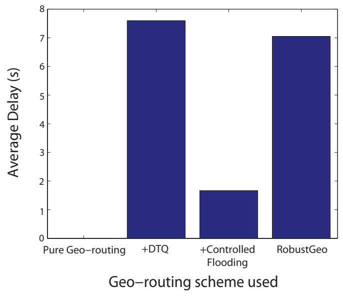
routed packets would be dropped.

We repeat our simulation 10 times with different seeds for each of the routing algorithms in this scenario. In figure 3.6, we see that as expected, the pure geo-routing method performs poorly because the sender can only transmit to the receiver when the two mobile clusters line up to form the transmission bridge. For the other three schemes, while RobustGeo performs marginally better than the rest in terms of packet delivery, it generated the most packets. We attribute this to the small number of nodes in the network. Controlled flooding only sends packets to another node if the packet is brand new to the node. With only 11 nodes in a highly partitioned system, there is not as much chance for sending packets in such a fashion. RobustGeo, on the other hand, broadcasts periodically regardless of who is around.

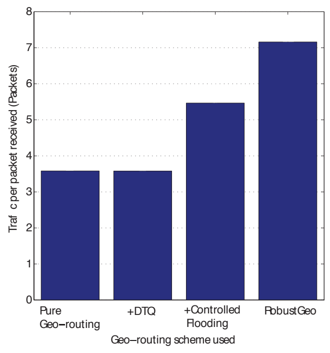
We also simulated a more realistic environment by downloading a 2000 m by 2000 m map of the Washington, DC area made available by the US Census Bureau's TIGER database, and simulating mobility on the map using the Intelligent Driver Model with Intersection



(a) Packet delivery ratios



(b) Average delay (seconds)

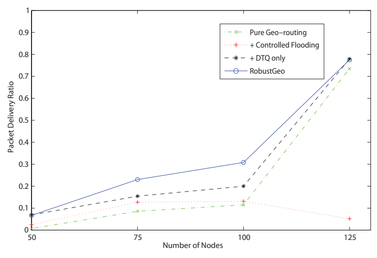


(c) Total traffic generated per packet received

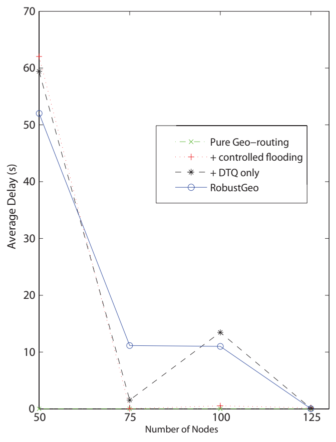
Figure 3.6: Results for the 5-group artificial scenario

Management by VanetMobisim [HFB06]. This model is complete with intersections and stop light rules to simulate realistic vehicular traffic. We generate 4 scenarios by varying the number of nodes between 50 and 125. In each scenario, we randomly place a static node on the map as the destination (server) and choose a random mobile node as the source (client).

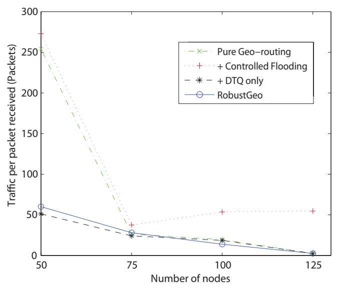
As Figure 3.7 shows, in the very sparse scenario of only 50 nodes, pure geo-routing completely breaks down and has a packet delivery ratio of 0. Having the DTQ and using RobustGeo marginally increases the delivery ratio, but it is still under 10%. As the nodes become denser, the protocols generally trend upwards in their packet delivery ratios. Something quite unexpected is that geo-routing + controlled flooding did considerably worse than pure geo-routing when the number of nodes increased to 125. We attribute this to the fact that each time a node cannot find a greedy forwarding path for a packet, the packet is sent out at least 5 times by that single node alone and multiplied by each recipient 5 more times. This generates a large amount of traffic, taking up available bandwidth and causing interference for nearby nodes. This suspicion is confirmed by the traffic line plot in Figure 3.7. In all, from the VanetMobisim scenarios we see that RobustGeo is indeed the most robust of all four routing schemes compared, having the highest delivery ratio for every situation. In terms of delay, geo-routing + DTQ did better in the 75-node case, but RobustGeo did better in all other cases. Additionally, we see that as predicted in Section 3.2, the replication



(a) Packet delivery ratios

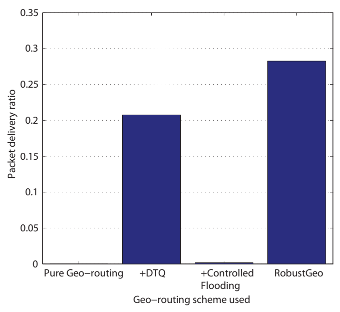


(b) Average delay (seconds)

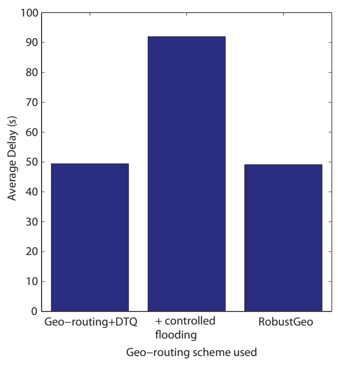


(c) Overall traffic generated per packet received

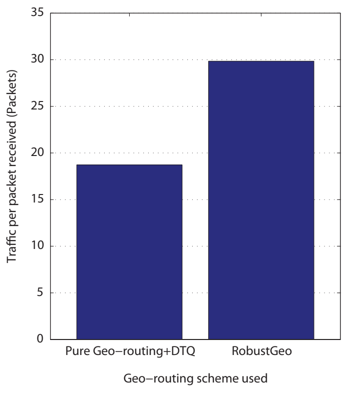
Figure 3.7: Results for the realistically simulated mobility in Washington, DC using Vanet-Mobisim and TIGER maps



(a) Packet delivery ratios



(b) Average delay (seconds)



(c) Overall traffic generated per packet received

Figure 3.8: Results for the real life San Francisco taxicab simulation scenario

overhead of RobustGeo is a manageable one, since it has only a slightly higher traffic to packet received ratio (about 10% or less) than the geo-routing + DTQ scheme that does not have replication.

Finally, we used real traces of an extremely intermittent network to further compare the four geo-routing schemes. In this scenario, we use actual mobility traces of taxi cabs in San Francisco downloaded from Crawdad [PSG09]. We run the simulation in a 5700m by 6600m area with 116 nodes moving for 200 seconds and again place a static node on the map to be the receiver. In this scenario, the nodes are extremely sparse because only cab movements are recorded in the trace file. This means that the network's period of disconnection is likely longer than its connectivity.

As Figure 3.8 shows, the pure geo-routing approach completely breaks down, with a receiving ratio of 0. Geo-routing + DTQ and RobustGeo does much better, with RobustGeo almost achieving a 30% packet delivery ratio. Once again, the geo-routing + controlled flooding approach fails, with a less than 1% delivery rate. We again attribute this to its high delay time. This means that had the simulation time been longer, the geo-routing + controlled flooding approach can likely a more acceptable packet delivery rate at the cost of even higher delay times. We choose to show the traffic per packet delivered metric for only pure geo-routing + DTQ and RobustGeo because the other two approaches yields incredibly high numbers (infinity and about 2500). As expected, RobustGeo generated 58% more traffic per packet delivered than geo-routing + DTQ. At the same time, it also has a 36% higher delivery ratio with a very slightly lower average delay. We believe that this is a good tradeoff as packet delivery ratios should be prioritized over bandwidth conservation.

Since the San Francisco cab scenario is very realistic and at the same time gives an extremely intermittent network, we use it to analyze disconnectivity and broadcast frequencies. Table 3.1 shows that in the given environment, the majority of packets experience disconnectivity 3 or fewer times, with only 8 packets encountering more than 4 intermittencies. Meanwhile, Figure 3.9 shows that about 80% of packets get broadcasted 6 times or fewer. This confirms the fact that packet replication will not spiral out of control.

# intermittencies	# packets
1	1434
2	1063
3	981
4	214
5	3
6	5

Table 3.1: The number of packets that encounter each frequency of intermittencies in the SF Taxicab scenario

We also postulate that the majority of packets experiencing higher frequencies of broadcast do not end up at the destination. If true, this can be used as a good metric to decide how long the packet should remain in a node's DTQ before being dropped.

3.4 Related Work

Several other works have also explored the possibility of using delay-tolerant networks that exploit location knowledge in VANETs. GeOpps [LM07] is one such DTN protocol. GeOpps uses the vehicle's navigation systems to find the *Nearest Point* (NP) to the message's destination that the vehicle will travel to, assuming that the vehicle has a predetermined destination of its own. While on the way, the vehicle will periodically broadcast the message's destination to its one-hop neighbors. The neighbors that receive the destination location calculates their Minimum Estimated Time of Delivery (METD) for the message based on their own calculated NP, and replies to the message carrier. The message carrier makes the decision to pass the along to its neighbor that has the shortest METD, and the process is repeated. While GeOpps' relying on a “smarter" way of choosing what are essentially Data MULEs can increase data dissemination speed, its high dependence on vehicular movements to carry the message to its destination causes it to experience much higher delays than greedy forwarding, since network propagation is much faster than physical node movements.

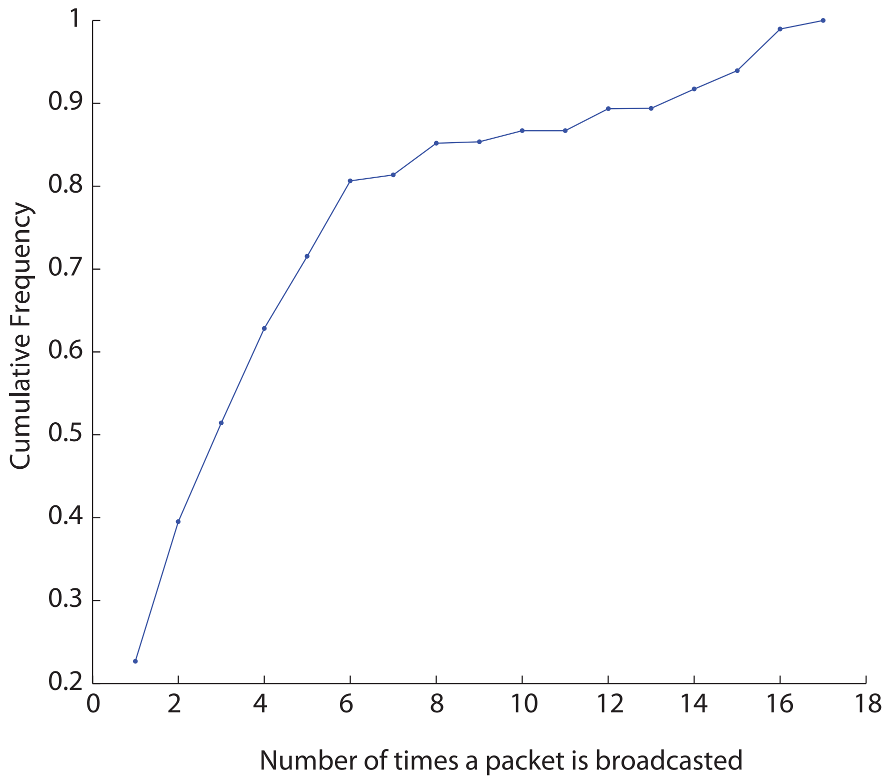


Figure 3.9: The cdf of packet broadcast rate in the SF Taxicab scenario

GeoSpray [SRF14] is another VANET routing solution that combines geographical knowledge with DTN ideas. Inspired by GeOpps, GeoSpray works in a similar fashion in that it takes into account the neighbors' METD and forwards the message to the ones with small METDs. However, they differ in that GeoSpray utilizes the concept of the spray phase in the binary Spray and Wait [SPR05] method where it utilizes controlled packet replication to explore multiple paths and decrease delay. This is similar to RobustGeo's singe hop broadcast phase, which also makes use of replication. However, just like GeOpps, GeoSpray suffers in a similar fashion in that it cannot take advantage of nodes who can otherwise be valid relay nodes in greedy forwarding, simply because they have a low METD value.

The work that is the most similar to RobustGeo is GeoDTN+Nav [CLG10]. GeoDTN+Nav fully incorporates the DTN paradigm into existing works on location-based routing protocols by operating with three modes - *restricted greedy forwarding*, *perimeter forwarding*, and *DTN*. The added DTN mode of GeoDTN+Nav allows packets to still be routed to the destination with a DTN approach in the event that the network is segmented due to sparse node topologies. The default greedy forward strategy for GeoDTN+Nav is the same as the one used in GPCR, and the default recovery mode is the same as that of VCLCR [LCW08], a location-based routing protocol that can detect routing loops in perimeter forwarding otherwise missed by GPCR. Finally, it uses the simple store-and-forward approach in DTN mode until greedy forwarding can be recovered.

GeoDTN+Nav switches from perimeter forwarding mode into DTN mode by having each node on the perimeter path calculate the *switching scores* of its neighbors to see if they are good candidates for carry-forward MULEs. These scores are based on three metrics - the probability of network disconnectivity, the neighbor's delivery quality in DTN mode, and the neighbors' travel direction. If the score is high enough, the packet is passed to the node's neighbor and DTN mode commences. In the DTN mode, the receiving node stores the packet and relies upon its own mobility to carry the packet until restricted greedy forwarding can once again be used to forward the packet to its destination.

GeoDTN+Nav differs from RobustGeo mainly in that the DTN method employed in GeoDTN+Nav uses a single-forwarding based algorithm, which generally has low reliability in packet delivery [Zha06]. For example, GeoDTN uses the neighbor's current traveling direction to determine its switching score. Unfortunately, vehicles' traveling directions are not constant. A change in direction by the vehicle after receiving the packet means that the packet would be actually brought *away* from the destination and thus increasing delays. Moreover, if no “good" neighbors are found within the packet's TTL, it is dropped. RobustGeo performs better in these situations because it explores the possibility of limited message replication by having the local maximum node keep a copy of the packet in the recovery phase, allowing multiple nodes to look for greedy paths toward the destination. Additionally, GeoDTN+Nav is very much focused upon the method with which to switch into the DTN mode from perimeter mode, whereas RobustGeo is designed so that unless the packet is immediately forwarded, it is, in some ways, always both in geo-routing mode *and* in DTN mode. RobustGeo is able to do this because it includes a simple scheduling system to dispatch packets opportunistically. Having such a feature allows RobustGeo to have much greater flexibility and thus be more robust.

In the underwater environment, there have also been no few attempts at leveraging location information for routing purposes. VBF [XCL06], RDBF [LYG14], and DBR [YSC08] are all examples of a geo-routing protocol for underwater wireless sensors that neither takes into account the void nor complete disconnectivities in the network. DBR is interesting in that its primary goal is to route packets from sensors deeper in the ocean to the data sink located near the surface. It accomplishes this by using water pressure to determine the depth of each relating sensor node to allow for geo-routing. However, the inability of these protocols to deal with local maximums is a great hindrance to their usefulness. VAPR [NLW13] and HydroCast [NLL16] address the shortcomings of DBR by providing a solution. While also using water pressure to route packets to the surface, VAPR seeks to avoid local maxima nodes by the way of beacons, allowing each node to know up to two hops of pressure information and thus avoid routing to the local maxima nodes. On the other hand, HydroCast recovers from the local maxima nodes by finding recovery paths using the flooding approach, similar to how RobustGeo deals with network disconnectivities. In any case, none of these routing protocols have a delay-tolerant feature, meaning that any temporary link breakages in a mobile AUV environment will cause the packets to be dropped.

3.5 Underwater compatibility

RobustGeo is also compatible with underwater networking with AUVs. In particular, it works well with our SDN-controlled network system as mentioned in Chapter 2, which uses long-ranged acoustics for the control plane and short-ranged but high speed optics for the data plane. In exploration or monitoring scenarios with non-pre-programmed paths, the AUVs will periodically lose contact with one another on their optical channel and thus the path back to the data sink would be disrupted. However, each AUV can still learn from the central controller using the control channel the locations of all the other AUVs, as well as their movements. Armed with this information, each AUV is able to run a slightly modified version of RobustGeo routing with directed optical beams to send delay-tolerant data back to the controller. Underwater optical RobustGeo have two main differences from VANET RobustGeo. First, the optical modems are incapable of broadcasting because all transmissions are directional. Second, In the majority of time the data sink for all of the AUVs are the same. While the first difference places a restriction on routing, the second one greatly simplifies the protocol. As such, RobustGeo modified for underwater AUVs no longer has the broadcast phase. Instead, when what used to be the broadcast timer expires, the AUV can replicate its packets by transmitting to a nearby AUV, perhaps one that is the closest (other than itself) to the data sink. The fact that almost all packets are destined for the same node (the data sink) also simplifies the DTQ to a great extent, making it just a simple FIFO queue rather than a hash labeled queue.

3.6 Conclusion

In this chapter we presented a location-based routing algorithm, RobustGeo, which is designed to withstand network intermittencies common to urban VANETs. Though the primary setting is for urban VANETs, the major principles can be used for under-water communications. RobustGeo achieves this result by taking advantage of both the store-and- forward and packet replication strategies found in delay tolerant networks. With RobustGeo, we were willing to make the tradeoff of introducing extra bandwidth overhead into the network with packet replications in favor of more reliable delivery. Though our analysis and experimental evaluations, we showed that the tradeoff is a viable one. RobustGeo allows for connections in both reliable and intermittent networks, which are attributes of both VANETs and AUV networks.

CHAPTER 4

Neighbor Discovery in WSNs

Power-saving for underwater sensors is highly important because it is not only difficult to recharge the devices, but nearly impossible to recover a device that completely dies. One way to save power is to keep the radios of the sensors off, or in its sleeping state, for as long as possible and only become active when data needs to be transmit. Thus, each device can alternate between active and sleeping states by keeping its radio "ON" for only some of the time [FN01]. This is challenging to achieve because two nodes can communicate only when both of their radios are "ON" at the same time; and with clock drifts, having set times for all the nodes to wake up at the same time is not trivial. Since clock synchronization is difficult in a distributed system, neighbor discovery must be done asynchronously. Over the years, the asynchronous neighbor discovery problem has been widely studied [BTK12][DC08][JTH05][KLR10][ZHL12][ZHY12][ZHS03], and existing research mainly focused on satisfying the following three design requirements:

1. Guarantee neighbor discovery within a reasonable time frame;

2. Minimize the number of time slots for which the node is awake to save energy;

3. Match the nodes' awake-sleep schedules with their heterogeneous battery duty cycles as closely as possible to prolong overall battery lifetime[1].

Here we present two optimal neighbor discovery protocols, called *Hedis* (*HEterogeneous DIScovery* as a quorum based protocol) and *Todis* (*Triple-Odd based DIScovery* as a co-primality based protocol), that guarantee asynchronous neighbor discovery in a heteroge-

[1]A duty cycle is the percentage of one period in which a sensor/radio is active.

neous environment, meaning that each node could operate at a different duty cycle. Specifically, they optimize the duty cycle granularity in their respective protocol categories to support duty cycles in the form of $\frac{2}{n}$ and $\frac{3}{n}$ respectively, and n is an integer that help achieve *almost all* duty cycles smaller than one. We analytically compare these two protocols with existing state-of-the-art protocols to confirm their optimality in the support of duty cycles, and also compare them against each other as a comparison between the two general categories of neighbor discovery protocols (quorum vs. co-primality based protocols). Our results show that while the discovery latencies are similar for both protocols, Hedis as an optimal quorum based protocol matches actual duty cycles much more closely than Todis as a co-prime based protocol.

We formally define the problem as well as any necessary terms in section 4.1, and give a taxonomy of current research efforts in this area in section 4.2. In sections 4.3 and 4.4, we present our optimizations for the quorum based and co-primality based protocols respectively, and we evaluate them with simulations in section 4.5. Finally, we conclude with section 4.6.

4.1 Problem Formulation

Here we define the terms and variables used to formally describe the neighbor discovery problem and its solution, as well as state any assumptions used in devising our protocols.

Wake-up schedule. We consider a time-slotted wireless sensor network where each node is energy-constrained. The nodes follow a *neighbor discovery wake-up schedule* that defines the time pattern of when they need to wake up (or sleep), so that they can discover their respective neighbors in an energy-efficient manner.

Definition 1. The neighbor discovery *schedule* (or simply schedule) of a node a is a sequence $\mathbf{s}_a \triangleq \{s_a^t\}_{0\leq t<T_a}$ of period T_a and

$$s_a^t = \begin{cases} 0 & a \text{ sleeps in slot } t \\ 1 & a \text{ wakes up in slot } t \end{cases}.$$

Clock drift. We do not assume clock synchronization among nodes, therefore any two given nodes may have random clock drifts. We use the cyclic rotation of a neighbor discovery schedule to describe this phenomenon. For example, a clock drift by k slots of node a's schedule s_a is

$$rotate(\mathbf{s}_a, k) = \{r_a^t\}_{0 \leq t < T_a},$$

where $r_a^t = s_a^{(t+k) \bmod T_a}$.

Definition 2. The *duty cycle* δ_a of node a is the percentage of time slots in one period of the wake-up schedule where node a is active (node a wakes up), defined as

$$\delta_a = \frac{|\{0 \leq t < T_a : s_a^t = 1\}|}{T_a}.$$

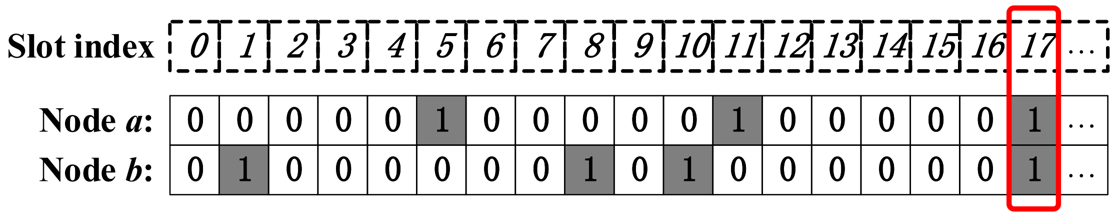
Neighbor discovery. Suppose two nodes a and b have schedules $\mathbf{s}_a$ and $\mathbf{s}_b$ of periods T_a and T_b, respectively. If $\exists t \in [0, lcm(T_a, T_b))$ such that $\mathbf{s}_a^t = \mathbf{s}_b^t = 1$ where $lcm(T_a, T_b)$ is the least common multiple of T_a and T_b, we say that:

- Nodes a and b can discover each other in slot t.
- Slot t is called the *discovery slot* between a and b.

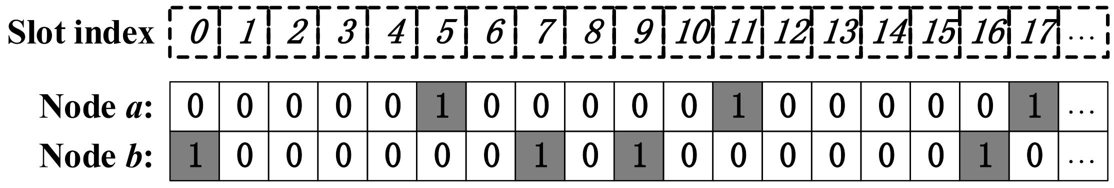
Figure 4.1 shows an example of two nodes with neighbor discovery schedules $\mathbf{s}_a = \{0, 0, 0, 0, 0, 1\}$ and $\mathbf{s}_b = \{0, 1, 0, 0, 0, 0, 0, 0, 1\}$, that have period lengths of $T_a = 6$ and $T_b = 9$ respectively. Node a is active on 1 slot within each period (6 slots) while node b is active on 2 slots within each period (9 slots). Thus the duty cycles of a and b are $d_a = \frac{1}{6} \approx 16.7\%$ and $d_b = \frac{2}{9} \approx 22.2\%$. In Figure 4.1a, We see that for every period of 18 slots ($lcm(T_a, T_b) = 18$), nodes a and b discover each other in slot 17. However, as illustrated in Figure 4.1b, when a one-slot clock drift occurs in node b, we have $rotate(\mathbf{s}_b, 1) = \{1, 0, 0, 0, 0, 0, 0, 1, 0\}$ and these two nodes can no longer discover each other.

4.2 A Taxonomy of Neighbor Discovery Protocols

In this section, we introduce a taxonomy of deterministic asynchronous neighbor discovery protocols. Through examining existing solutions to the neighbor discovery problem, we divide these protocols into two broad categories.


Slot index 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 …
Node *a*: 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 …
Node *b*: 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 …

(a) Without clock drift


Slot index 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 …
Node *a*: 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 …
Node *b*: 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 …

(b) Node b drifts by 1 time slot to the left

Figure 4.1: An example of neighbor discovery: Two neighbor discovery schedules are $\mathbf{s}_a = \{0, 0, 0, 0, 0, 1\}$ and $\mathbf{s}_b = \{0, 1, 0, 0, 0, 0, 0, 0, 1\}$. Without clock drift (4.1a), the two nodes can discover each other every 18 time slots since $lcm(T_a, T_b) = 18$. With clock drift (4.1b), neighbor discovery fails.

4.2.1 Why Deterministic Protocols

Many solutions have been proposed to solve the neighbor discovery problem. One of the earliest such solutions are the birthday protocols [MB01], which take upon a *probabilistic* approach to neighbor discovery. These protocols rely on the *birthday paradox*, which states that with as few as 23 people, the probability that two people have the same birthday exceeds $\frac{1}{2}$. As a non-deterministic protocol based upon probability, birthday protocols are heterogeneous and supports every duty cycle with the finest granularity. Following this, many more similar probabilistic protocols were also developed [VKT05][VTG09][ZVC11][ZL08]. However, due to their probabilistic nature, these protocols fail to provide a guaranteed upper bound for neighbor discovery latency, which means that there is a chance for two neighbors to never discover each other.

To combat this insufficiency, *deterministic* protocols with worst case bounds for neighbor discovery were developed. The earlier deterministic protocols such as [JTH05][THH03], and [LRC10] all use the quorum concept. However, while these protocols are effective in guaranteeing neighbor discovery, they are generally lacking in duty cycle support. For example, [THH03] and [JTH05] are homogeneous, meaning that they require all the nodes to have the same duty cycle. As a result, the co-primality based approach was developed with Disco [DC08] and U-Connect [KLR10], although U-Connect is in some ways a hybrid approach using elements from both the quorum and co-primality paradigms.

4.2.2 Quorum vs. Co-primality Based Protocols

The deterministic protocols for neighbor discovery can be largely classified into two major categories, *quorum* based protocols and *co-primality* based protocols, though they both work because of the CRT.

4.2.2.1 Quorum Based Protocols

Quorum based protocols take advantage of geometry in a 2-dimensional array.

Bounded discovery delay. In the most original protocols like [THH03], time is arranged into an $m \times m$ matrix. Every node then chooses a row and a column for which to wake up. This ensures that regardless of any clock drifts, any two nodes would be able to wake up at the same time slot at least twice every m^2 time slots, thus guaranteeing an upper bound for neighbor discovery. However, this method only works if every node happens to use the same duty cycle. Lai et al. [LRC10] improve upon this method by constructing *cyclic quorum system* and *grid quorum system* pairs, which allow for two different duty cycles to coexist and still ensure bounded neighbor discovery.

Example protocols. The current latest development in quorum based protocols is Searchlight [BTK12], which is able to support multiple duty cycles in the network. Searchlight essentially divides the duty cycle period into a $\frac{t}{2} \times t$ matrix, and uses a combination of *anchor* and *probing* slots to generate wake up patterns. At the beginning of every t time slots is an anchor slot, and a probing slot occurs at random slots between the anchor slots. With this technique, Searchlight [BTK12] shows that it is able to allow neighbor discovery among nodes with many different duty cycles.

4.2.2.2 Co-primality Based Protocols

A co-primality based neighbor discovery protocol is one in which

- Each node, say, node a, chooses a set of integers (not necessarily distinct) $N_a = \{n_1^a, n_2^a, n_3^a, \ldots, n_{|N_a|}^a\}$.
- For two distinct nodes a and b, N_a and N_b must satisfy the following *co-prime pair property.*

Definition 3. For two distinct nodes a and b under a co-primality based neighbor discovery protocol, there exists an integer in N_a that is co-prime to an integer in N_b—i.e., $\exists n_{i_0}^a \in N_a$ and $n_{j_0}^b \in N_b$ such that $n_{i_0}^a$ and n_{j0}^a are co-prime.

Node a's schedule $\mathbf{s}_a \triangleq \{s_a^t\}_{0 \le t < T_a}$ under this co-primality based protocol is

$$s_a^t = \begin{cases} 1 & t \,\text{is divisible by some}\; n_i^a \in N_a \\ 0 & \text{otherwise} \end{cases}.$$

The period length is $T_a = lcm(n_1^a, n_2^a, \ldots, n_{|N_a|}^a)$ and its duty cycle δ_a is

$$\delta_a = \sum_{1 \le i_1 \le |N_a|} \frac{1}{n_{i_1}^a} - \sum_{1 \le i_1 < i_2 \le |N_a|} \frac{1}{lcm(n_{i_1}^a, n_{i_2}^a)}$$

$$\cdots + (-1)^{|N_a|+1} \frac{1}{lcm(n_1^a, n_2^a, n_3^a, \cdots, n_{|N_a|}^a)}.$$

Bounded discovery delay. By the Chinese Remainder Theorem (CRT) [Nat00], we can obtain the following theorem.

Theorem 4. *A co-primality based neighbor discovery protocol can guarantee discovery for any two nodes for any amount clock drift if the associated integer sets of the nodes in this network satisfy the co-prime pair property. And the worst-case discovery delay is bounded by the product of the two smallest co-prime numbers, one from each set, i.e.:*

$$\min_{\gcd(n_i^a, n_j^b)=1, 1 \le i \le N_a, 1 \le j \le N_b} \{n_i^a \cdot n_j^b\}.$$

Suppose the clock of node a is d time slots ahead of that of node b, i.e., node b's t^{th} time slot is the $(t+d)^{th}$ time slot of node a, where d is the clock drift, the following congruence system w.r.t. t applies:

$$\begin{cases} t \equiv 0 & \pmod{p_i} \\ t + d \equiv 0 & \pmod{p_j} \end{cases}. \tag{4.1}$$

If t is a solution to Eq. (4.1), then node a will discover node b in node a's t-th time slot (i.e., node b's $(t+d)$-th time slot). By CRT, since p_i and p_j are co-prime, there exists a solution $t \equiv t_0 \pmod{p_i p_j}$.

Example protocols. Disco [DC08], as such a co-primality based protocol, ensures co-primality by only using prime numbers as possible parameters. In Disco, each node chooses two distinct primes to create its wake-up schedule. For example, node a chooses two distinct primes p_1 and p_2 and node b chooses p_3 and p_4. Node a is active (wakes up) in the t-th time slot iff t is divisible by either p_1 or p_2 while node b is active in the t-th time slot iff t is divisible by either p_3 or p_4. Therefore, Disco can guarantee neighbor discovery for any two nodes for any amount of clock drift with a bounded discovery delay of

$$\min_{\gcd(p_i,p_j)=1, i=1,2, j=3,4} \{p_i \cdot p_j\}.$$

Again, this delay is the product of the two smallest co-prime numbers following from the CRT.

U-Connect [KLR10] is a combination of Disco and the basic quorum based protocol in that it restricts the dimensions of the square quorum matrix to be a prime number. In this way, if the duty cycles of the nodes happen to be the same, neighbors would discover one another via the quorum method. On the other hand, if they are different, the numbers chosen would be co-prime to each other and thus enabling neighbor discovery by Theorem 4.

More comprehensive surveys on neighbor discovery can be found in [ACD09] and [GH12].

4.3 Hedis: Optimizing the Quorum based Protocols

Hedis is an asynchronous periodic slot-based neighbor discovery protocol where each node picks its anchor and probing slots according to the elements of a quorum that is carefully selected in an $(n-1)$ by n matrix.

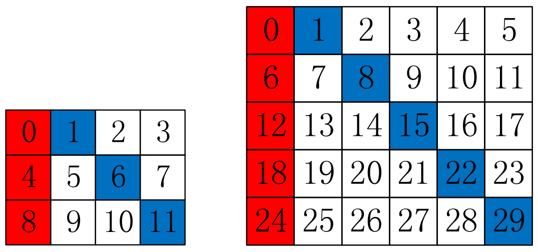
4.3.1 Design of the Hedis Schedule

For a node a that has a desired duty cycle δ, the period of its schedule under Hedis, $\mathbf{s}_a = \{s_a^t\}_{0 \le t < n(n-1)}$, consists of $n(n-1)$ time slots, where the integer n is chosen such that $\frac{2}{n}$ comes as close to δ as possible (and we call n the *parameter* of this node). Under Hedis, its schedule is

$$s_a^t = \begin{cases} 1 & t = ni, (n+1)i+1 \ (i = 0, 1, 2, \dots, n-2) \\ 0 & \text{otherwise} \end{cases},$$

where ni $(i = 0, 1, 2, \dots, n-2)$ denotes the index of an *anchor* slot and $(n+1)i+1$ denotes the index of a *probing* slot.

Figure 4.2 shows two example Hedis schedules when $n = 4, 6$, and the two schedules consist of of $n(n-1) = 12, 30$ time slots, respectively. Each grid in the figure represents a time slot, and the integer inside a grid denotes its slot index, e.g., the grid with 0 inside denotes the 0th time slot in the schedule (note that a schedule starts from the 0th time slot). The red and blue slots represent the anchor and probing slots, during which the node wakes up. When $n = 4$, the duty cycle is $2/4 = 50\%$. The full schedules are depicted in Figure 4.3, where the two nodes with different duty cycles can achieve successful neighbor discovery (overlap of colored slots between schedules of nodes a and b) for many times in every period. Next, we will show that Hedis can guarantee neighbor discovery for any two nodes of same-parity parameters (both odd or both even) with heterogeneous duty cycles for any amount of clock drift.



(a) n=4 (b) n=6

Figure 4.2: Two example Hedis schedules.

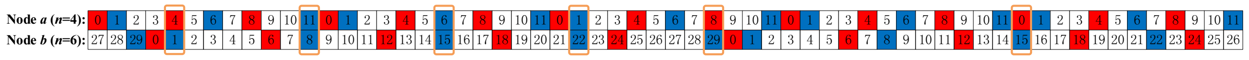


Figure 4.3: Node a's schedule is 3-slot ahead of node b's schedule. The overlapped colored slots between their schedules represent the successful neighbor discovery.

4.3.2 Bounded Discovery Delay under Hedis

Lemma 5. *Let m and n be positive integers. For any integers a and b, there exists an integer x such that*

$$x \equiv a \pmod{m} \tag{4.2}$$

and

$$x \equiv b \pmod{n} \tag{4.3}$$

if and only if

$$a \equiv b \pmod{gcd(m, n)}.$$

If x is a solution of congruences (4.2) and (4.3), then the integer y is also a solution if and only if

$$x \equiv y \pmod{lcm(m, n)}.$$

The proof can be found in [Nat00], on page 61, Theorem 2.9. By Lemma 5, we further establish the following theorem.

Theorem 6. *Hedis guarantees neighbor discovery within bounded latency for any two nodes with the same-parity parameters n and m, given any amount of clock drift between their schedules. The average discovery latency is $O(nm)$.*

Proof. Nodes a and b are two arbitrarily given nodes, whose parameters are n and m, respectively. The periods of the Hedis schedules of nodes a and b are $T_a = n(n-1)$ and $T_b = m(m-1)$, respectively. We use d to denote the clock drift.

Without loss of generality, we study the following system of congruences with respect to t:

$$\begin{cases} t \equiv ni + d, (n+1)i + 1 + d \pmod{n(n-1)} \\ t \equiv mj, (m+1)j + 1 \pmod{m(m-1)}, \end{cases} \tag{4.4}$$

where $i \in [0, n-2]$, $j \in [0, m-2]$.

$$t \equiv ni + d, (n+1)i + 1 + d \pmod{n(n-1)}$$

$(i \in [0, n-2])$ is true iff $\exists i \in [0, n-2]$ such that it is true, and the same meaning for

$$t \equiv mj, (m+1)j + 1 \pmod{m(m-1)}$$

$(j \in [0, m-2])$.

There are a number of nm pairs of simultaneous congruences, which we divide into 4 groups: anchor-anchor, anchor-probing, probing-anchor and probing-probing groups. E.g., the anchor-probing group denotes the case where an anchor slot of node a overlaps a probing slot of node b. Note that if we find a solution that meets the requirements of any one of these congruences, we obtain a solution to Eq. (4.4).

Group 1: anchor-anchor. Consider the following system of congruences

$$\begin{cases} t \equiv ni + d \pmod{n(n-1)} & i \in [0, n-2] \\ t \equiv mj \pmod{m(m-1)} & j \in [0, m-2] \end{cases},$$

which is equivalent to

$$\begin{cases} t \equiv d \pmod{n} \\ t \equiv 0 \pmod{m} \end{cases}. \tag{4.5}$$

By Lemma 5, Eq. (4.5) has a solution if and only if

$$\gcd(n, m) \mid d.$$

Group 2: anchor-probing. Consider the following system of congruences

$$\begin{cases} t \equiv ni + d \pmod{n(n-1)} & i \in [0, n-2] \\ t \equiv (m+1)j + 1 \pmod{m(m-1)} & j \in [0, m-2] \end{cases}, \tag{4.6}$$

which is equivalent to

$$\begin{cases} t \equiv d \pmod{n} \\ t \equiv (m+1)j + 1 \pmod{m(m-1)} & j \in [0, m-2] \end{cases}. \tag{4.7}$$

By Lemma 5, Eq. (4.7) has a solution if and only if $\gcd(n, m(m-1)) \mid (m+1)j + 1 - d$ for some integer $j \in [0, m-2]$, i.e., the congruence with respect to j

$$(m+1)j \equiv d - 1 \pmod{\gcd(n, m(m-1))} \tag{4.8}$$

has a solution.

Group 3: probing-anchor. Consider the following system of congruences

$$\begin{cases} t \equiv (n+1)i + 1 + d \pmod{n(n-1)} & i \in [0, n-2] \\ t \equiv mj \pmod{m(m-1)} & j \in [0, m-2] \end{cases}, \tag{4.9}$$

which is equivalent to

$$\begin{cases} t \equiv (n+1)i + 1 + d \pmod{n(n-1)} & i \in [0, n-2] \\ t \equiv 0 \pmod{m} \end{cases}. \tag{4.10}$$

By Lemma 5, Eq. 4.10 has a solution if and only if

$$\gcd(m, n(n-1)) \mid (n+1)i + 1 + d$$

for some integer $i \in [0, n-2]$, i.e., the congruence with respect to i

$$(n+1)i \equiv -d - 1 \quad (\text{mod } \gcd(m, n(n-1))$$

has a solution.

Group 4: probing-probing. Consider the following system of congruences

$$\begin{cases} t \equiv (n+1)i + 1 + d \pmod{n(n-1)} & i \in [0, n-2] \\ t \equiv (m+1)j + 1 \pmod{m(m-1)} & j \in [0, m-2] \end{cases}. \tag{4.11}$$

By Lemma 5, Eq. 4.11 has a solution if and only if

$$\gcd(n(n-1), m(m-1)) \mid (n+1)i - (m+1)j + d$$

for some integer $i \in [0, n-2]$ and $j \in [0, m-2]$.

Now we begin to prove this theorem by cases.

Case 1: If $m > n$, the congruence system of anchor-probing (Group 2) is true. *Proof:* If $m > n$, we have $m-1 \geq n \geq \gcd(n, m(m-1))$. And note that $\gcd(m+1, \gcd(n, m(m-1))) = \gcd(m+1, n, m(m-1)) = \gcd(m+1, 2, n) = 1$. This is because m and n are both odd or are both even. So one of $m+1$ and n are odd, and we have $\gcd(m+1, 2, n) = 1$. Therefore $(m+1)j$ ($j \in [0, m-2]$) runs over all congruence classes modulo $\gcd(n, m(m-1))$. Then Eq. (4.8) has at least $\lfloor (m-1)/\gcd(n, m(m-1)) \rfloor$ solutions and on average $(m-1)/\gcd(n, m(m-1))$ solutions. Hence Eq. (4.7) has at least $\lfloor (m-1)/\gcd(n, m(m-1)) \rfloor$ solutions and on average $(m-1)/\gcd(n, m(m-1))$ solutions modulo $lcm(n, m(m-1))$. Therefore, the average discovery latency is $\frac{lcm(n,m(m-1))}{(m-1)/\gcd(n,m(m-1))} = nm$.

Case 2: If $n > m$, the congruence system of probing-anchor (Group 3) is true. *Proof:* If $n > m$, similarly to case 1, we have Eq. (4.10) has at least $\lfloor (n-1)/\gcd(m, n(n-1)) \rfloor$ solutions and on average $(n-1)/\gcd(m, n(n-1))$. Hence Eq. (4.9) has at least $\lfloor (n-1)/\gcd(m, n(n-1)) \rfloor$ solutions and on average $(n-1)/\gcd(m, n(n-1))$ modulo $lcm(m, n(n-1))$. Therefore, the average discovery latency is nm.

Case 3. If $n = m$, we consider the result of d mod n. If $d \equiv 0 \pmod{n}$, then $\gcd(n, m) = n \mid d$, and thus the anchor-anchor case (Group 1) is true and the average discovery latency is $O(nm)$. Now we concentrate on the case where $d \not\equiv 0 \pmod{n}$. Since $n = m$, Eq. (4.8) becomes

$$(n+1)j \equiv d - 1 \pmod{n}.$$

Since $(n+1)j = nj + j \equiv j \pmod{n}$, this is equivalent to

$$d \equiv j + 1 \pmod{n}.$$

For $j \in [0, n-2]$, $j+1$ runs over$[1, n-1]$. Because $d \not\equiv 0 \pmod{n}$, there exists a $j \in [0, n-2]$ that satisfies Eq. (4.8), and therefore the anchor-probing case (Group 2) is true. Similarly, the probing-anchor case (Group 3) is also true. And it is easy to check that the average discovery latency is $O(n^2)$, i.e., $O(nm)$. $\square$

4.4 Todis: Optimizing Co-primality based Protocols

Now we optimize the asynchronous co-primality based protocols, and propose Todis that exploits properties of consecutive odd integers for achieving co-primality.

As a co-primality based protocol, Todis creates wake-up schedules for the nodes based on multiples of numbers that are co-prime to each other. This ensures that any two given nodes would be able to wake up at the same time by the co-prime pair property as illustrated in Section 4.2, thus succeeding in neighbor discovery. Recall that Disco [DC08] guarantees this by simply using prime numbers as parameters, which limits the variety of parameters to choose from. For two nodes a and b, we need to construct two sets of integers, N_a and N_b, that must satisfy the co-prime pair property. In our quest to find co-prime pairs, we observe that for two numbers to be co-prime, at least one of them must be odd. Thus, we explore the possibility of achieving co-primality using odd integers. We observe that given two odd integers a and b, if they are not co-prime, often times either "$a+2$ and b", or "a and $b+2$" is a co-prime pair. For example, if 15 and 21 are not co-prime, we are able to find that either "17 and 21", or "15 and 23" is a a co-prime pair. Following this logic, we design our Todis protocol using sets of consecutive integers.

4.4.1 Design of the Todis Schedule

4.4.1.1 Trying sets of two consecutive integers

First, we tried using two consecutive odd integers in N_a for each node a, i.e., $N_a = \{n, n+2\}$ where $n \geq 1$ and n is odd. Unfortunately, for given nodes a and b, there are instances where the sets N_a and N_b do not satisfy the co-prime pair property for very small numbers (i.e., less than 100). For example, when $N_a = \{33, 35\}$ and $N_b = \{75, 77\}$ and $\forall n_i^a \in N_a, n_j^b \in N_b$, we have $\gcd(n_i^a, n_j^b) > 1$.

4.4.1.2 Using sets of three consecutive integers in Todis

In Todis, we use three consecutive odd integers $n-2$, n and $n+2$ ($n \geq 3$) for constructing a wake-up schedule.

The co-prime pair property requires that at least one of the three consecutive odd integers that node a chooses (i.e., $n-2$, n and $n+2$) is co-prime w.r.t. one of the three integers that node b chooses (i.e., $m-2$, m and $m+2$).

Bounded discovery delay in practical networks. Generally, the triples consisting of three consecutive odd integers can also fail to satisfy the required co-prime pair property, as seen in counterexamples shown by the CRT. However, the two smallest sequences of odd integers in these counterexamples are $N_a = \{1600023, 1600025, 1600027\}$ and $N_b = \{2046915, 2046917, 2046919\}$. Such integers are too large to be chosen for creating a "practical" duty cycle anyway. For example, an n value larger than 1600023 would imply a duty cycle smaller than δ_a of 0.00000187496. In practical applications, however, duty cycles are much greater than 0.00000187496. Therefore, any chosen sets N_a and N_b based on duty cycles would satisfy the co-prime pair property. By Theorem 4, Todis guarantees neighbor discovery with a delay bounded by

$$\min_{\gcd(n+i,m+i)=1, i=-2,0,2} \{n \cdot m\}.$$

A node a that has a desired duty cycle of δ may therefore choose an odd integer n such that

$$\frac{3(n^2-n-1)}{n(n^2-4)} \approx \frac{3}{n} = \hat{\delta}$$

is as close to δ as possible. We call n the *parameter* of node a.

Under Todis, its wake-up schedule is

$$s_a^t = \begin{cases} 1 & t \text{ is divisible by either } n-2,\ n, \text{ or } n+2 \\ 0 & \text{otherwise} \end{cases},$$

with a period length of $(n-2)n(n+2)$ and a duty cycle of

$$\frac{1}{n-2} + \frac{1}{n} + \frac{1}{n+2} - \frac{1}{(n-2)n} - \frac{1}{n(n+2)} - \frac{1}{(n-2)(n+2)}$$

$$+\frac{1}{(n-2)n(n+2)} = \frac{3(n^2-n-1)}{n(n^2-4)}.$$

Figure 4.4 shows the first 71 time slots under the Todis schedule when $n = 15$ (i.e., the node chooses 13, 15 and 17). Each grid in the figure represents a time slot, and the integer inside a grid denotes its slot index, e.g., the grid with 0 inside denotes the 0th time slot in the schedule (note that a schedule starts from the 0th time slot). The gray slots represent the active slots where the node wakes up. In this example, the duty cycle is $\frac{3\cdot(5^2-5-1)}{5\cdot(5^2-4)} \approx 18.9\%$.

4.4.2 Analysis of Duty Cycle Granularity

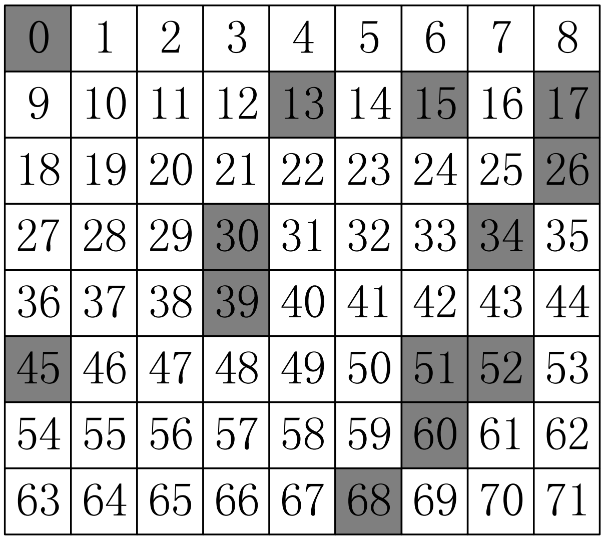
Now we discuss the granularity of Todis in matching any desired duty cycle in practical applications. Suppose node a's desired duty cycle is δ_a, the relative error $\epsilon(\delta_a)$ between δ_a and its approximation $\hat{\delta_a}$ is defined by

$$\epsilon(\delta_a) = \frac{\left|\hat{\delta}_a - \delta_a\right|}{\delta_a}. \tag{4.12}$$

We want to mathematically estimate the upper bound of ϵ given δ_a, which we denote as $\hat{\epsilon}(\delta_a)$.

In Todis, node a needs to choose an odd integer n_a such that $\frac{3(n_a^2-n_a-1)}{n_a(n_a^2-4)}$ lies closest to δ_a, i.e.,

$$n_a = \arg\min_{n \text{ odd}} \left|\frac{3(n^2-n-1)}{n(n^2-4)} - \delta_a\right|.$$



▨ 0	1	2	3	4	5	6	7	8
9	10	11	12	▨ 13	14	▨ 15	16	▨ 17
18	19	20	21	22	23	24	25	▨ 26
27	28	29	▨ 30	31	32	33	▨ 34	35
36	37	38	▨ 39	40	41	42	43	44
▨ 45	46	47	48	49	50	▨ 51	▨ 52	53
54	55	56	57	58	59	▨ 60	61	62
63	64	65	66	67	▨ 68	69	70	71

Figure 4.4: The first 71 time slots under the Todis schedule when $n = 15$ (i.e., the node chooses 13, 15 and 17). The node wakes up in time slots 0, 13, 15, 17, 26, 30, 34, 39, 45, 51, 52, 60, 68, ...

Thus, the best approximation of the desired duty cycle δ_a is

$$\hat{\delta}_a = \frac{3(n_a^2 - n_a - 1)}{n_a(n_a^2 - 4)} \equiv \min_{n \text{ odd}} |\frac{3(n^2 - n - 1)}{n(n^2 - 4)} - \delta_a|.$$

Let $f(2k-1)$ and $f(2k+1)$ be two consecutive supported duty cycles, where $f(n) = \frac{3(n^2-n-1)}{n(n^2-4)}$. Relative error ϵ reaches a local maximum at $\delta_a = \frac{f(2k-1)+f(2k+1)}{2}$. Thus we obtain a quartic equation with respect to k

$$16\delta_a k^4 - 24k^3 + (12 - 40\delta_a)k^2 + 36k + 9\delta_a - 9 = 0. \tag{4.13}$$

By Eq. (4.13), we can obtain a solution $k = k(\delta_a)$ in complex radicals (the other three solutions are discarded). Then we have

$$\epsilon(\delta_a) \le \hat{\epsilon}(\delta_a) \triangleq \frac{f(2k(\delta_a) - 1) - \delta_a}{\delta_a},$$

where $\hat{\epsilon}(\delta_a)$ is also a complex expression in radicals with respect to δ_a.

Note that $\epsilon(\delta_a) = \hat{\epsilon}(\delta_a)$ iff $\delta_a = \frac{3(n^2-n-1)}{n(n^2-4)}$ for some odd integer n. We illustrate $\hat{\epsilon}(\delta_a)$ in Figures 4.5 and 4.6 (see the "Estimation" lines), and we can observe that $\hat{\epsilon}(\delta_a)$ is a very tight upper bound for $\epsilon(\delta_a)$.

The upper bound function $\hat{\epsilon}(\delta_a)$ is an increasing function in $[0, 1)$. In practical applications, δ_a is smaller than 20%, and thus ϵ is upper bounded by 6.71%, which is a very small relative error. Moreover, ϵ drops below 3.34% when $\delta_a \leq 10\%$. Asymptotically,

$$\hat{\epsilon}(\delta_a) \simeq \frac{2\delta_a}{\sqrt{9 + 4\delta_a^2} + 3} \simeq \frac{1}{3}\delta_a$$

linearly approaches 0 as δ_a goes to 0. This property implies that the error decreases with the decline of the desired duty cycle.

4.5 Performance Evaluation

We compare Hedis and Todis against state-of-the-art neighbor discovery protocols of both the quorum based and the co-primality based varieties. These protocols include Disco [DC08] (co-primality based), Searchlight [BTK12] (quorum based), and U-Connect [KLR10] (a combination of both). We evaluate the performances of these protocols using two metrics, namely the discovery latency and the duty cycle granularity.

- In Disco, each node chooses a pair of primes p_1 and p_2 to support duty cycles of the form $\frac{1}{p_1} + \frac{1}{p_2}$, and the worst-case discovery latency is $\min\{p_1 p_3, p_1 p_4, p_2 p_3, p_2 p_4\}$.

- In U-Connect, each node wakes up 1 time slot every p time slots and wakes up $\frac{p+1}{2}$ time slots every p^2 time slots. Therefore U-Connect supports duty cycles of the form $\frac{3p+1}{2p^2}$, and has the worst-case discovery latency of $p_1 p_2$ if one node uses prime p_1 while another uses p_2. The dependence of Disco and U-Connect upon prime numbers greatly restricts their support of choices of duty cycle varieties.

- Searchlight requires that a node's parameter n_1 be a multiple or factor of its neighboring node's parameter n_2 to guarantee neighbor discovery. Therefore, in a network that implements Searchlight, the number that each node chooses must be a power-multiple of the smallest chosen number (i.e., 2, 4, 8, 16, or 3, 9, 27, 81, etc.), guaranteeing that any two nodes' numbers are multiples of each other. As a result, Searchlight only supports duty cycles of the form $\frac{2}{t^i}$, where t is an integer (i.e., the aforementioned smallest chosen number) and $i = 0, 1, 2, 3, \ldots$

Table 4.1: Comparison of Hedis and Todis with existing neighbor discovery protocols.

Protocol name	Parameter restriction	Average dis. delay	Supported duty cycles
Disco	prime p_1, p_2 p_3, p_4	$O(\min\{p_1p_3, p_1p_4, p_2p_3, p_2p_4\})$	$\frac{1}{p_1} + \frac{1}{p_2}$
U-Connect	prime p_1 , p_2	$O(p_1p_2)$	$\frac{3p_1+1}{2p_1^2}$
Searchlight	power-multiple of t_1 , t_2	$O(t_1t_2)$	$\frac{2}{t_1^i}$
Hedis	same parity n, m	$O(nm)$	$\frac{2}{n}$
Todis	odd n, m	$O(nm)$	$\frac{3(n^2-n-1)}{n(n^2-4)}$ $\approx \frac{3}{n}$

Table 4.1 gives an overall numerical comparison among these protocols. As the table shows, while the difference in discovery latency exists among these protocols, all of them perform on the order of the multiple of the principle parameters in the two participating nodes.

- Discovery latencies may be similar among the different protocols, because two nodes may choose similar parameters so as to match the desired duty cycle.
- In contrast, the metric of duty cycle granularity presents a different story. While all the parameters used in the protocols all have special restrictions due to protocol design, it is obvious that those for Hedis and Todis are the least stringent. For example, fewer than 2% of integers under 1000 are prime, while half of them are odd, giving Todis a much larger pool of numbers to choose from for its parameters as compared to Disco and U-Connect.

We confirm these numerical results using simulations. We measure the relative errors each of the aforementioned protocols yields at differing duty cycles, as well as their discovery

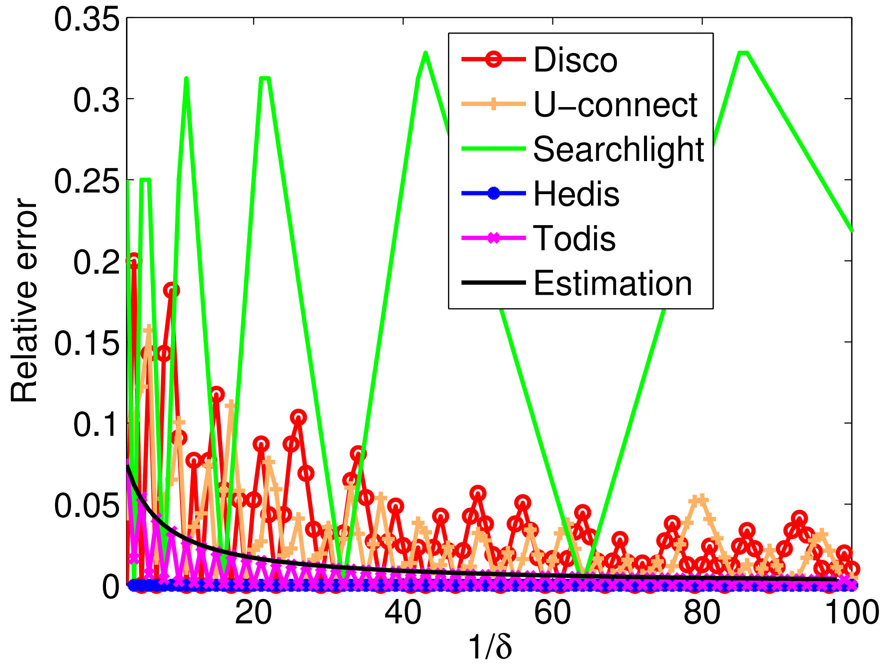


Figure 4.5: Relative error vs. small duty cycle δ. The "Estimation" line is the theoretical upper bound estimation of relative error induced by Todis (see Section 4.4).

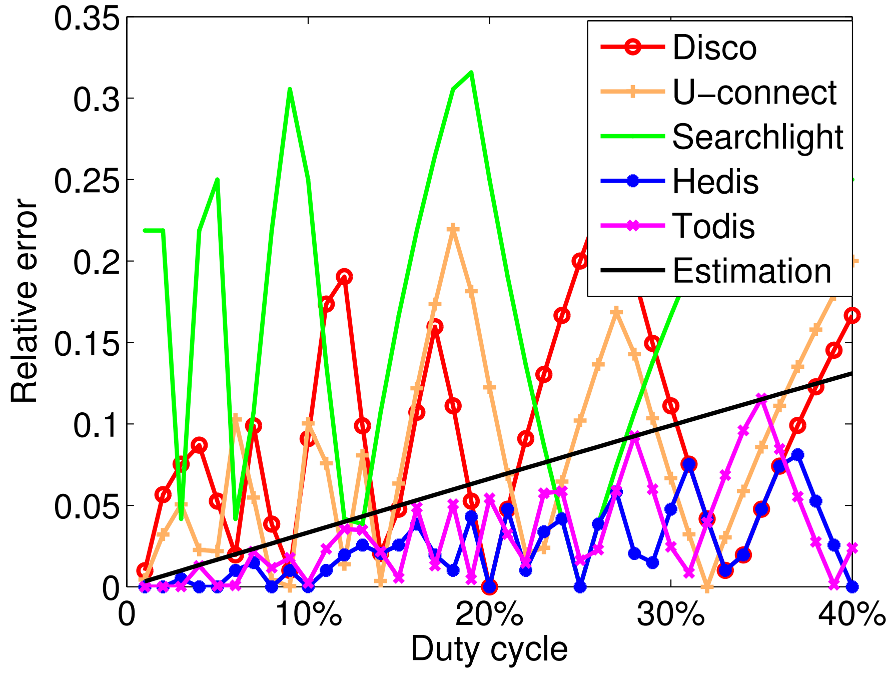


Figure 4.6: Relative error vs. large duty cycle δ. The "Estimation" line is the theoretical upper bound estimation of relative error induced by Todis (see Section 4.4).

latencies in node pairs operating at various duty cycles.

4.5.1 Duty Cycle Granularity

The first set of simulations comparatively studies the supported duty cycles. We study two groups of duty cycles:

1. Small duty cycles $1 \leq 1/\delta \leq 100$, i.e., $\delta = 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{100}$;
2. Equispaced large duty cycles $0 \leq \delta \leq 1$, i.e., $\delta = 0\%, 1\%, 2\%, 3\%, 4\%, \ldots, 100\%$.

We use the metric called *the relative error* (defined in Eq. 4.12) to quantify the capability of supporting each studied duty cycle, which is denoted as

$$\epsilon \triangleq |\delta' - \delta|/\delta,$$

where δ' is the closest duty cycle that is supported by each simulated protocol, w.r.t. δ. Note that a smaller ϵ implies that the protocol provides more choices for energy conservation with a finer granularity of duty cycle control. For Searchlight, we let the smallest duty cycle unit be 1/2 to allow the finest duty cycle granularity.

Figure 4.5 illustrates the results for small duty cycles, while Figure 4.6 shows those of large duty cycles. These results provide us the following insights:

- Searchlight is inferior to the other protocols in supporting various duty cycles because it requires the duty cycle to be $\frac{2}{t^i}$, where t is a fixed integer and $i = 1, 2, 3, \ldots$. In this simulation, we use $t = 2$ to give Searchlight support for the duty cycles $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$. The relative error increases significantly as the desired duty cycle deviates away from the supported duty cycles (e.g., in Figure 4.6, it has a peak at $37.5\% = \frac{1/2+1/4}{2}$, and 1/2 and 1/4 are supported duty cycles).
- The schedules in Disco and U-Connect are generated using prime numbers, which have a denser distribution than power-multiples. Thus, Disco and U-Connect perform better than Searchlight.

- Both Hedis and Todis greatly outperform all the other protocols, having very small relative errors. In fact, for small duty cycles, the relative errors from Hedis is nearly constantly zero (see Figure 4.5). On the other hand, although Todis also performs well, its error rate obviously increases much faster than Hedis as the duty cycle δ increases.

- The theoretical "Estimation" lines for Todis (see Figures 4.5 and 4.6) holds up well in that it follows the same pattern as Todis' actual error rates. This confirms our prior analysis in section 4.4 of Todis' duty cycle granularity, where we estimated the upper bound relative error for Todis.

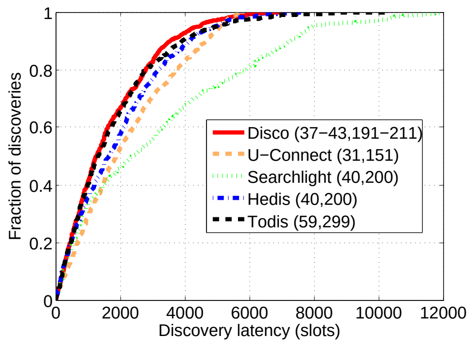


Figure 4.7: CDF of discovery latency when each pair of nodes operate at duty cycles 1% and 5%, respectively. Numbers in the parentheses indicate the parameters of each corresponding protocol.

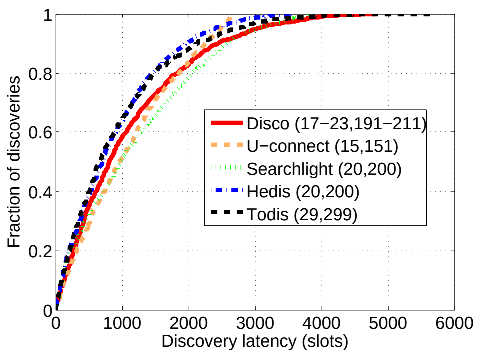


Figure 4.8: CDF of discovery latency when each pair of nodes operate at duty cycles 1% and 10%, respectively. Numbers in the parentheses indicate the parameters of each corresponding protocol

4.5.2 Discovery Latency

In this set of simulations, we study the discovery latencies of these protocols. For each simulation, we take 1000 independent pairs of nodes and assign various duty cycles. In two instances, we compare the protocols' performance in heterogeneous discovery scenarios. We assign duty cycles of 1% and 5% to each respective node in the node pair in the first instance (see Figure 4.7), and 1% and 10% in the second instance (see Figure 4.8). We also compare the performance of the protocols in two homogeneous discovery scenarios, with each node in

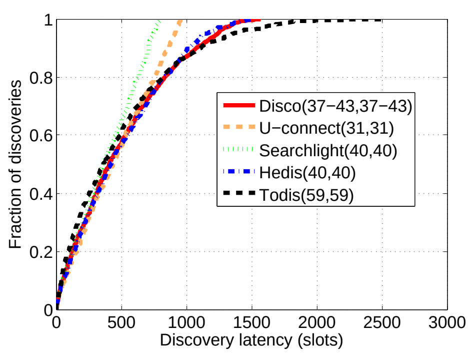


Figure 4.9: CDF of discovery latency when each pair of nodes operate at the same duty cycle of 5%. Numbers in the parentheses indicate the parameters of each corresponding protocol.

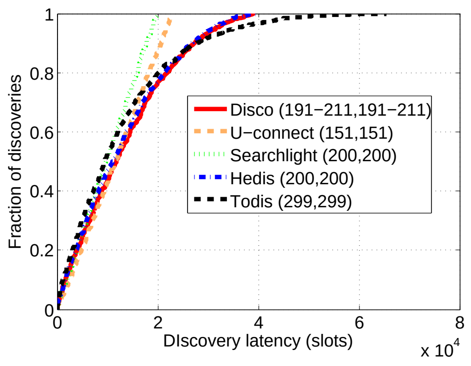


Figure 4.10: CDF of discovery latency when each pair of nodes operate at the same duty cycle of 1%. Numbers in the parentheses indicate the parameters of each corresponding protocol.

the node pair operating at the same duty cycles of 5% in the first scenario (see Figure 4.9) and 1% in the second one (see Figure 4.10).

Heterogeneous vs. homogeneous duty cycles. From these four cumulative distribution function graphs (CDFs), we see that overall, all of the protocols have comparative discovery latencies, with the odd exception of Searchlight in Figure 4.7. Nonetheless, it must be noted that all 5 protocols presented were eventually successful in neighbor discovery for 100% of the pairs tested. These CDFs also show that Hedis is one of the few protocols that consistently perform above average in both the heterogenous and homogeneous neighbor discovery cases. For example, Figures 4.9 and 4.10 indicate that Searchlight is the clear winner for discovery latency in the homogeneous case, but it does poorly in the heterogeneous cases, as seen in Figures 4.7 and 4.8.

In addition, we see that for up to 90% of the CDF, Hedis and Todis are both near top performers, but the protocol with one of the smallest latencies in reaching 100% of the CDF in every case is U-Connect. We attribute this to the fact that U-Connect uses smaller values as its parameters, thus having a smaller upper bound in the worst case.

Similarly, we attribute Todis' consistent long tail in each CDF scenario to its larger parameters. Therefore, although it can quickly allow nodes to discover each other in most cases, seen in its quickly reaching 90% in the CDFs, it has the longest latency in the worst-case scenarios.

Hedis vs. Todis. These various simulations show that Hedis and Todis optimize the duty cycle granularity in both the quorum based and the co-primality based neighbor discovery approaches, with Hedis having a finer granularity than Todis. Additionally, both protocols perform reasonably well in terms of discovery latency, with Todis having a larger worst case latency bound due to its larger parameters.

4.6 Conclusion

In this chapter we explored the current two main approaches of designing an asynchronous heterogeneous neighbor discovery protocol with guaranteed latency upper bounds—the quorum based and the co-primality based approaches. Using these two approaches we designed the Hedis and Todis neighbor discovery protocols, emphasizing on duty cycle granularity optimization for both. Hedis, as a quorum based protocol, forms a $(n-1) \times n$ matrix of time slots and uses the anchor- probing slot method to ensure neighbor discovery. Todis, as a co-primality based protocol, uses sets of three consecutive odd integers to ensure co-primality and thus ensures neighbor discovery due to CRT. In the design of both protocols we proved their capability in ensuring acceptable upper bounds in discovery latency. Through analytical comparisons as well as simulations, we confirmed the optimality of Hedis and Todis in duty cycle granularity among existing protocols. Hedis is able to support duty cycles in the form of $\frac{2}{n}$, while Todis can support duty cycles roughly in the form of $\frac{3}{n}$, allowing both protocols to effectively cover any practical duty cycle and thus prolong battery longevity.

We also showed in both our analysis and simulations that Hedis as a quorum based protocol is better than Todis as a co-primality based protocol in both duty cycle granularity and discovery latency, although differences by the latter metric are minor. By being able to support duty cycles at such a fine granularity while still guaranteeing an acceptable discovery latency bound, Hedis truly paves the way for neighbor discovery in under-water sensor networks.

CHAPTER 5

From Marina del Rey to the NS3 Simulator: The Underwater Network Testbed

In Chapter 2, we created a prototype implementation of an underwater SDN system in the WaterCom [GMS15] testbed, which consisted of many underwater acoustic modems networking with one another in a water tank. Although WaterCom at the time successfully supported our implementation and experiments, it had a number of deficiencies. First, the environment it simulated was unrealistic, especially with the number of modems present. The high node density, combined with acoustic reflections off of the tank itself, created an environment whose channel clarity was negatively correlated with transmission power, and whose propagation delay is practically nonexistent. Second, the close proximity of the nodes to one another, where the distances between any two of the nodes were less than one meter apart, meant that it was very difficult to create an idea of a network with a particular topology. The way we overcame this problem in Chapter 2 was by using a combination of less powerful modems and foam barriers in the tank, but the situation was not ideal. Finally, the testbed was limited in the number of nodes it could provide, based on the number of actual acoustic radios that were available.

In this chapter we make significant upgrades to WaterCom by bringing the acoustic modems from the water tank in the lab into Marina del Rey, which is a Pacific Ocean marina by West Los Angeles, to increase channel realism, and by adding an emulation mode that allows the user to use larger numbers of nodes at longer distances from one another. We also include an improved web user interface that allows researchers to remotely access and control our testbed, both for emulation and real devices, with an SSH terminal.

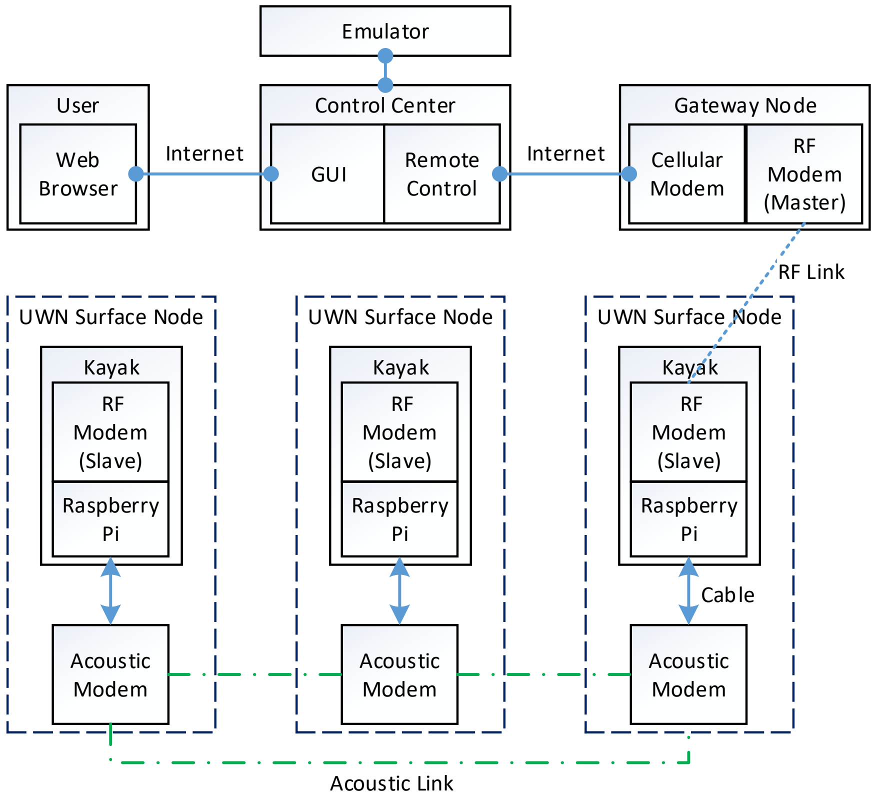


Figure 5.1: The overall architecture of our underwater network testbed.

5.1 General testbed architecture

Our testbed system gives the users two main options to run experiments. The first option is the Marina del Rey testbed, which uses three AquaSeNT AM-OFDM-13A acoustic radios deployed in the marina for more realistic channel models while limited by node count. The second option is emulation on the server using channel models from the NS3 simulator. While this experiment route is not as realistic, it allows the user to use a much larger number of nodes limited only to server capacity, which can be increased via relatively cheaper upgrades (compared to buying new modems and setting them up).

The general overall architecture of our testbed is illustrated in Figure 5.1. Both the marina and emulation options are available remotely via our web interface that connects the user directly to the Control Center on the server. Once the users connect to the Control

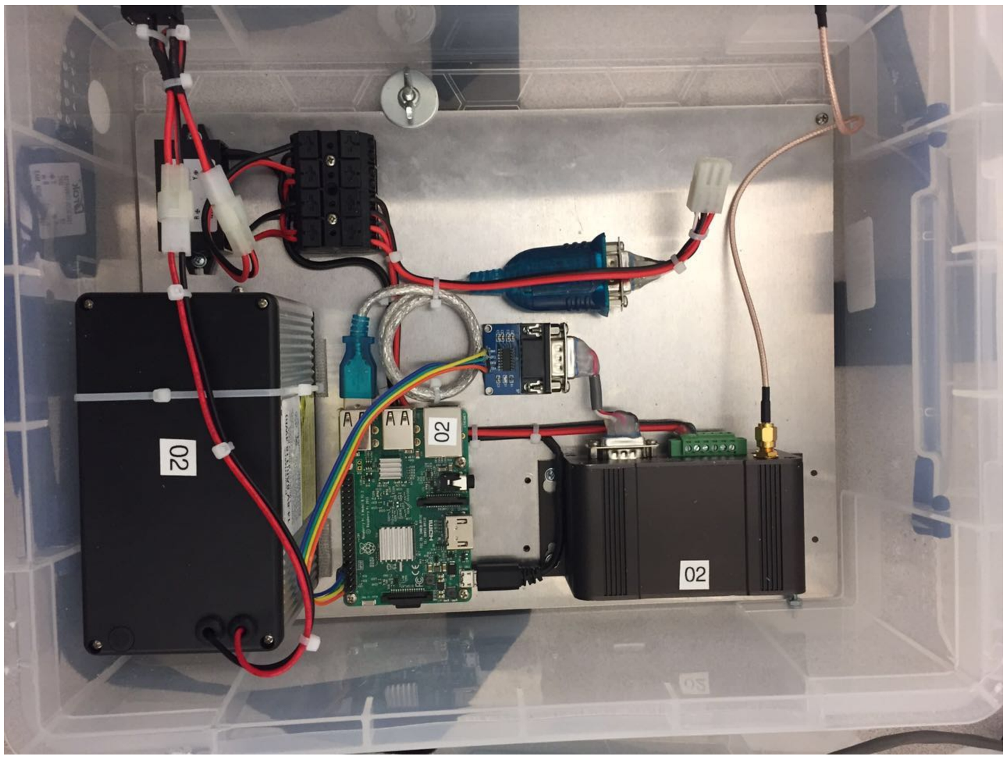


Figure 5.2: The local node controller box consisting of, from left to right, a 14.8V 8Ah (118.4Wh, 8A rate) battery, a Raspberry Pi, and a 900 MHz wireless modem

Center, they may choose to remain on the server to run an emulated experiment, or to make another connection from there to the cellular connected Gateway Node in Marina del Rey. The Gateway Node is the master controller for all of the experiment nodes in the testbed. Using long range RF links, it allows the users to set up protocols and experiment scripts on each of the experiment nodes, as well as execute and end all experiment instances.

Next, we will delve deeper into the implementations of both the Marina testbed and the underwater emulator on the server.

5.2 Implementation for Marina del Rey

The Marina del Rey testbed consists of 4 nodes numbered from 1 through 4. Node 1 is the special Gateway Node and it does not have an acoustic modem attached to it. Rather, it provides a platform for the user to control nodes 2 through 4 via the RF control channel.

5.2.1 The controller box

Each of the Marina nodes has a local controller box like the one shown in Figure 5.2, containing the three major components essential for node operation - a battery, a Raspberry Pi computer, and an RF modem. The battery is a 14.8V 8Ah (118.4Wh, 8A rate) battery capable of continuous operation for 48 hours for a single charge, it supplies power to everything on the node, including the Raspberry Pi, the RF modem, as well as the acoustic OFDM modem. The Raspberry Pi is the local controller for the node. Loaded with Ubuntu Linux, the Raspberry Pi runs the experiment scripts and the SeaLinx [LPC13] protocol stack that drives the acoustic modem. From the Gateway Node, users may access the Raspberry Pi on nodes 2 through 4 via the attached MicroHard$^{\text{TM}}$ n920-ENC modem, which is a spread-spectrum modem that operates at 900MHz with wireless link rates up to 230.4kbps, more than enough for terminal accessing each of the experiment nodes. To increase its range, we attach to each of these modems a 900 MHz 8 dBi Omnidirectional Antenna (visible in Figure 5.3) on the outside of the control box. The Gateway Node also has a cellular modem as an additional component, allowing users to access it from the Control Center in the lab.

5.2.2 Setup and deployment

For deployment into water, we use a child-sized kayak. The kayak acts both as a buoy from which our acoustic modem can hang and as a storage unit that holds all the necessary equipment for the node. We modify the kayak by mounting on it a wooden frame, which has the roles of supporting the antenna as mentioned in the previous section, stabilizing the kayak, and lodging the controller box in its place. The kayak is anchored in place from both ends to prevent itself from drifting in circles. The acoustic modem hangs from the back of the kayak on a rope tied to the wooden frame so that its cable would not need to support its weight. The complete setup of one of the experiment node is pictured in Figure 5.3, and a photo of a completely deployed experimental node is seen in Figure 5.4.

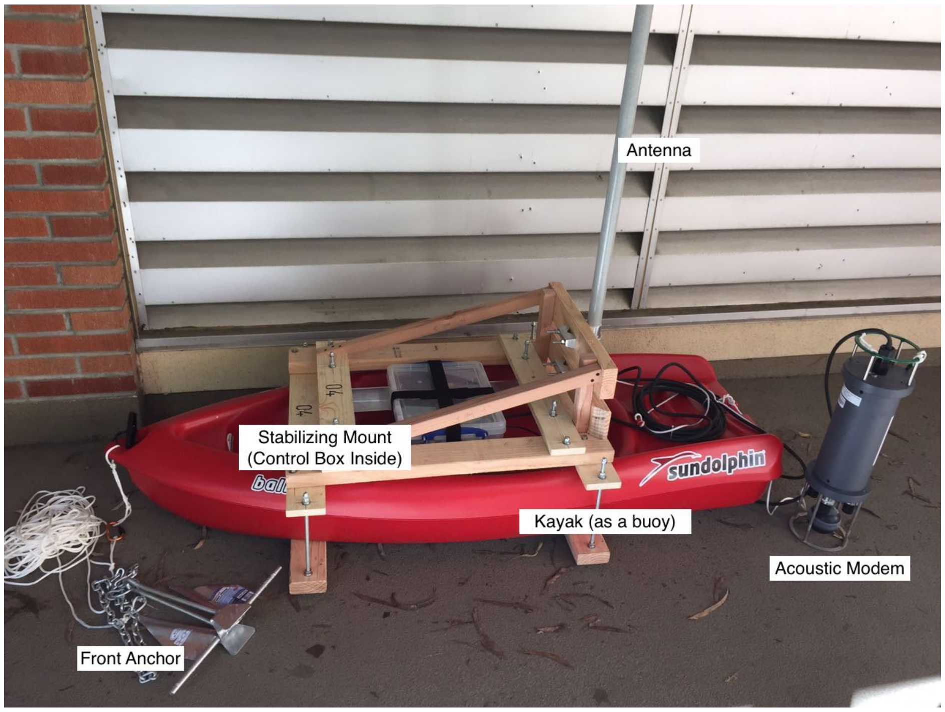


Figure 5.3: The complete setup of an experiment node (Node 4); an extra anchor is attached to the back of the kayak in deployment to prevent circular drifting around the front anchor

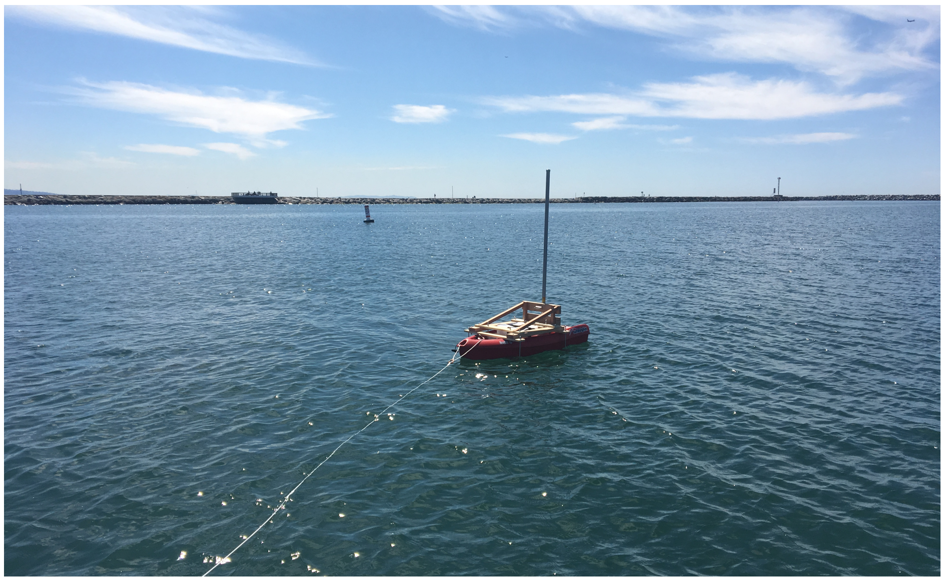


Figure 5.4: An experiment node being deployed in the marina in its anchoring phase

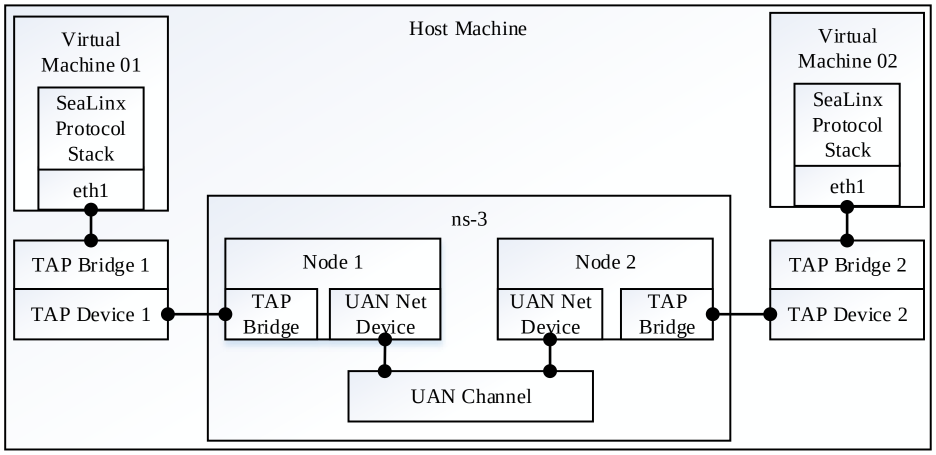


Figure 5.5: An example of an emulated system with two virtual underwater acoustic nodes connected to each other through NS3

5.3 Emulation using the NS3 Simulator

While the Marina del Rey testbed provides researchers with a realistic underwater networking environment, it limits them in the number of available nodes to use. For experiments requiring larger numbers of nodes, we provide an emulated solution on the server itself.

We use the emulation capabilities of the NS3 simulator, rather than simply implementing a simulator in NS3 because we would like to provide a flexible, unified platform that is consistent with our marina testbed. This is advantageous because it allows researchers to test their protocols by only making a single implementation in SeaLinx. Alternatively, if someone would like to test his or her protocol in both the marina testbed and the simulator, he or she would need to implement it once in SeaLinx, then again in NS3, which is costly in both time and effort.

Our emulator uses virtual machines capable of running the SeaLinx protocol stack and allow them to connect to one another via an NS3 simulated channel. In a sense, we replace the physical acoustic modem in a node with a simulated modem in NS3 with its channel module. This is possible because the SeaLinx protocol stack runs in application space rather than the modem so that researchers can more easily develop new protocols for it. The actual AquaSent modems themselves simply broadcast their messages and allow SeaLinx to resolve the network. Therefore emulation can be made possible by replacing the actual acoustic modem with any other device, virtual or not, to receive data from the “physical" layer of SeaLinx and send data to that layer. In our emulator, data sent out from SeaLinx are rerouted to a network device in the virtual machine. The device that the VM sees is actually a tap bridge created on the host machine, which is connected to a tap device, which is connected to a simulated NS3 node. Figure 5.5 gives an example set up of the emulated system that contains two underwater acoustic nodes.

There were two main challenges in realizing this setup. First, the SeaLinx physical layer communicates with the acoustic modem using operation commands specified by the modem. A lack of proper response from the modem is interpreted as error and can cause the physical layer to shut down. To fix this, we design a “dummy” physical layer, using the upper layer templates, that reads from and writes to a UDP socket. Since the real acoustic modem broadcasts all of its packets, our dummy physical layer sends all received packets to the broadcast IP address for the tap-bridge device connected to the VM. This way, all of the SeaLinx data packets can be encapsulated in IP packets and be transmitted among VMs in any networking method needed, and SeaLinx will continue to work as it does with an actual acoustic modem.

Another issue is the existing NS3 implementation itself. NS3 provides a TapBridge NetDevice that automatically handles connections to the system tap device on the host machine. To use emulation mode, users can simply create a TapBridge node on NS3 to handle connections from outside NS3, then another network node of their choice (depending on the kind of experiment) that is connected to the TapBridge node. The TapBridge node then forwards the traffic from outside NS3 into the regular network device to simulate the network. The underwater networking module for NS3 is the UAN Framework, providing the channel, PHY, MAC, and AUV models. Unfortunately, the two NS3 devices are incompatible with each other because NS3 TapBridge nodes use the standard 48 bit MAC address, while NS3 UAN nodes use a customized 8 bit MAC address. To make them compatible, we create an address translator module that is a part of the UAN NetDevice, which automatically generates a

UAN MAC address from existing standard MAC addresses and keeps that translation, so that when requesting the UAN node for its MAC address, the standard MAC address would be returned.

5.4 Conclusion

This chapter documents the creation of a better underwater experiment testbed over the existing WaterCom testbed. The new testbed both has a more realistic network environment, being in Marina del Rey, and includes an emulation mode that allows its users to run larger scale experiments. The work in improving our testbed is still ongoing, with plans to outfit our kayak with more sturdy materials so that they can eventually be deployed in the open ocean.

CHAPTER 6

Windowed Aloha, an Underwater MAC Protocol

Medium access remains a great challenge in underwater acoustic networks. Using our SDN testbed WaterCom, we previously compared two existing protocols, Slotted FAMA [MS07] and UW-Aloha [PZC09], in Chapter 2. That comparison showed that UW-Aloha soundly outperforms S-FAMA in busy channel conditions, where interference from the high number of transmitting nodes is rampant. This is due to the high overhead from the extra control packets (RTS and CTS) that S-FAMA employs, which also sets higher channel quality requirements for data transmission. On the other hand, under a clear channel with little interference, UW-Aloha's reliance on ACK packets for carrier sensing greatly limits its transmission rates. This chapter presents a new medium access control protocol, which we call "Windowed Underwater Aloha (WUW-Aloha)," that allows the sender to adaptively pipeline its packets based on channel conditions to maximize the rate of transmission according to the environment.

6.1 A comment on existing protocols: S-FAMA and UW-Aloha

Slotted FAMA uses channel reservation schemes such as RTS and CTS to minimize channel usage overlap and thus avoid packet collisions. Unfortunately, interference from multiple senders is not the only source of packet loss in an underwater environment. The underwater acoustic channels can be disrupted by a host of things, from ocean vessels passing by, to oceanic wildlife, to even large surface waves. Additionally, other environmental factors such as water temperature and salinity play a large role in the quality of the acoustic channel. As such, the underwater acoustic channel experiences periods of good and bad quality independent of data transmissions. For a pair of nodes relying on RTS and CTS for channel reservation under bad channel conditions, not only is it difficult for the nodes to complete the channel reservation handshake process, but a successful RTS-CTS handshake also does not ensure high packet transmission success rates.

To address this issue, Peng, Zheng et. al developed the underwater version of the simple Aloha protocol, UW-Aloha. UW-Aloha does away with channel reservation packets and encourages nodes to transmit packets at-will. It uses (the lack of) Acknowledgment packets from the receiver to detect packet collisions and trigger transmission back-offs. This design allows for nodes to communicate in bad acoustic channel conditions with minimal superfluous communication overhead. However, it also places limitations on overall transmission throughput, since every single packet must be acknowledged before the next one is sent. This is further exacerbated by high transmission delays, including propagation delays inherent to the acoustic channel and low bit-rates due to technological limitations for acoustic modems. As a result, UW-Aloha is unable to maximize channel usage under good channel conditions.

To illustrate this point, consider the single sender, single receiver case where each node is capable of transmitting at k pkts/s, and the transmission delay is d seconds. The optimal throughput is thus theoretically k pkts/s with an initial latency of d seconds, assuming continuous transmissions without control packet overhead. Under uw-aloha, however, it would take approximately $2(\frac{1}{k} + d)$ seconds to transmit a single packet from having to wait for the packet to arrive at the destination node, then for the destination node to transmit the ACK back. This yields maximum throughput of $\frac{k}{2(1+dk)}$ pkts/s under optimal conditions, which is less than 50% of the optimal throughput, and getting worse as the delay increases.

Furthermore, current usage cases for underwater networking is mainly exploratory information gathering, which generally requires a more sizable amount of data to be useful. For example, when an exploring AUV generates an image and transmits it to another node, the destination node is unable to interpret the image until it has received some percentage of the file. Therefore, it is more preferable for a node to receive a single chunk of usable data from one sender than to receive the same amount of data spread across many different senders, rendering the data unusable until some time later. Additionally, the data to be received later is not guaranteed, rendering much of the previously received packets useless.

We develop our Windowed Underwater Aloha (WUW-Aloha) protocol as a flexible solution to this problem. WUW-Aloha features an adaptive pipeline that expands and contracts according to channel conditions. Under good channel conditions, the sender is able to send increasingly more packets for every Acknowledgment packet received, thus reducing control packet overhead. At the same time, it is able to function robustly, as UW-Aloha does, under bad channel conditions, because it does not use RTS-CTS packets to reserve the channel. Additionally, we design WUW-Aloha to fit the likely usage case of underwater networking, encouraging complete transmissions of larger chunks of data from individual senders rather than spreading the available throughput amongst multiple senders.

6.2 Design

When transmitting data in the underwater environment, the network nodes tend to work in a more cooperative manner to achieve the same goal, namely to gather exploration data. As such, the packets transmitted among nodes are of two types - meta packets and data packets. Meta packets tend to be sparse, relaying control information such node movements and localization. These packets require very little throughput and function well enough with the UW-Aloha protocol. Data packets, on the other hand, stands to have much to gain with increased throughput, since they tend to come in large chunks that represent files such as images, text-format reports, and even video clips. In these usage cases, it makes more sense for the data sink to receive one large, interpretable data *chunk* from a single source at a time, rather than to receive a sparse set of individual packets not immediately interpretable from many different sources at the same time. The reason behind this is twofold. First, the traditional data transmission model often requires large bandwidth overheads from constant channel contention due to the data sources all attempting to transmit to the data sink at the same time. Second, obtaining interpretable data early allows the data sink to make decisions early. Some of these decisions may affect how and if further data transmissions are required, which can be game-changing in a bandwidth-limited environment such as the underwater acoustic channel. In short, granting each sender longer periods of channel occupancy to transmit entire files decreases channel contention overheads and allows for faster decision making at the data sink.

We design WUW-Aloha by modifying UW-Aloha to fit this data transfer model. Under WUW-Aloha, the sender node can send multiple packets in a pipelined fashion to the receiver for a single ACK using the principles of TCP congestion control. Just as each sender has a congestion window (cwnd) in TCP, WUW-Aloha grants each sender a *pipeline window (pwnd)*, which, like TCP's cwnd, experiences "slow start" and "congestion avoidance" phases. This allows a sender to quickly increase the number of packets it is able to pipeline to the receiver under optimal conditions to achieve near optimal throughput. Furthermore, WUW-Aloha's pwnd decreases when packets are lost and will shrink to 1 in the case of an ACK timeout (3 RTT), at which time the sender would, like any MAC protocol, use binary back-off to contend for the channel.

6.2.1 Channel contention and pipelining

WUW-Aloha allows senders to pipeline packets to the receiver through the pipeline window. Each time the sender transmits a packet, it is recorded in the pwnd, and the sender may fill up the entirety of its pwnd before waiting to hear back an ACK from the receiver. In this way, all the packets from the same sender pwnd are transmitted in a pipelined fashion. The size of the pwnd changes based on the ACK response from the receiver. WUW-Aloha initiates slow start with the size of its pwnd at 1, increasing exponentially each time a "full ACK" is heard from the receiver, until the pwnd reaches the threshold size. At this point it enters the conservative phase, where the pwnd size increases by 1 every time a "full ACK" is received by the sender. ACK packets feed the sender information on the conditions of the channel by informing it the number of packets received at the receiving node. A "full ACK" from the receiver indicates that all of the packets sent by the sender in the pipeline are received by the receiver. In the event that the sender receives a partial ACK, where the number of packets ACKed is fewer than the number of packets sent, the sender will decrease

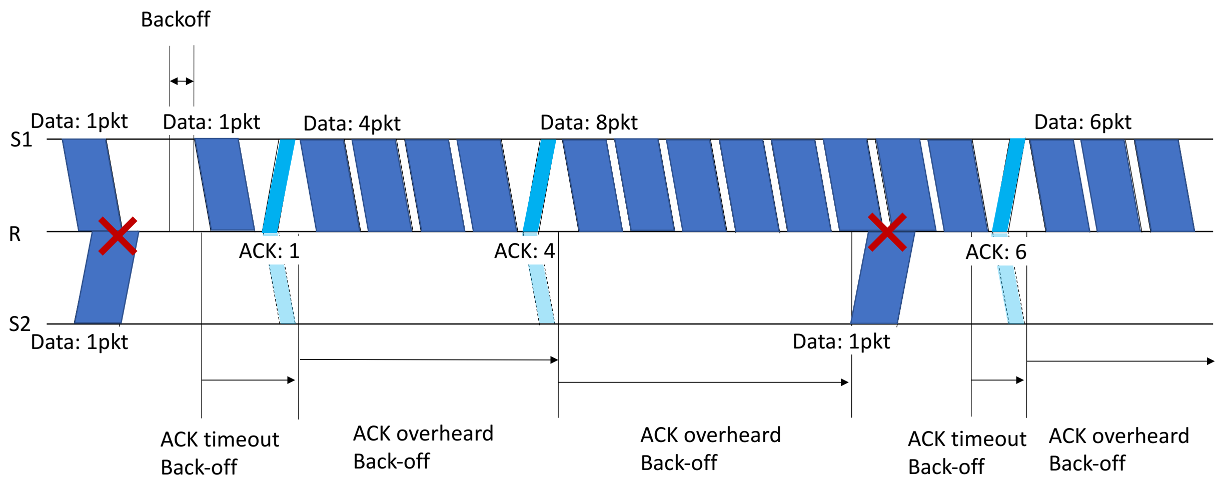


Figure 6.1: An illustration of 2 senders transmitting to a single receiver using WUW-Aloha. Sender 1 gets the channel and begins to pipeline packets to the Receiver, while Sender 2 yields until Sender 1 finishes

the size of its pwnd to be equal to the number of packets successfully transmitted. Finally, if the sender times out waiting for an ACK response, it assumes that the channel is now occupied, and will, therefore, shrink its pwnd down to 1, decrease the window threshold size to half of its previous pwnd size (only when pwnd > 10), and begin the back-off channel contention process.

When increasing the pipeline size, we consider the initial increase from size 1 as a special case. Upon the success of the first transmission, the pwnd size increases to 4 rather than doubling to 2. We design this case to address the fact that one packet can potentially interfere with three packets at the receiver end (the duration of a packet lengthens due to reflections in a channel, allowing it to overlap with up to three packets arriving from other senders). Increasing the pwnd size to 4 from 1 ensures that the receiver will receive one packet even in the event that a packet from a different sender arrives at the same time as the current sender's packets.

To minimize channel contention overhead and maximize pipelining, WUW-Aloha encourages data sources to transmit large chunks of data one at a time by prioritizing channel occupation for nodes that are already transmitting packets. Every packet includes the pwnd size between the sender and the receiver. When a sender overhears an ACK packet destined for a different sender, it compares the pwnd from the ACK packet to its own pwnd. If its pwnd size is smaller than the one carried in the ACK, it will defer the channel to the other sender by shrinking its own pwnd to 1 and backing off, allowing for better pipelining in the network. Similarly, the sender also backs off if it overhears data packets sent to the same receiver that it is trying to send to. To prevent one sender from taking up the channel for too long, a cross-layer approach is used, where the upper layers (such as the application layer) leave some time between data chunk transmissions. When WUW-Aloha times out receiving packets from the upper layers, it will shrink its pwnd down to 1 and get ready to enter channel contention when more packets arrive. Figure 6.1 gives a clear illustration of a normal case where one of two sending nodes gains access to the channel and begins a pipelined transmission to the receiver. The other sender backs off whenever it hears an ACK packet indicating that the other sender has a larger pwnd than itself.

6.2.2 ACKs and pipeline preservation

Because of interference against packets further down the pipeline, the receiver of a data packet cannot immediately send back an ACK. Therefore the receiver sends back ACKs if it receives the maximum number of packets in the pipeline window, or if it times out on receiving data packets from the sender. The issue with having a single ACK take care of many packets in a pipeline is the single point of failure. An unfortunate spontaneous loss of an ACK packet from the receiver can spell doom for a pipeline that delivers most of its packets. We alleviate this by allowing for a retransmission of the ACK by the receiver if it does not hear from the sender 1.5 RTT from the time of sending out its ACK. Note that this allows for the original ACK packet to arrive at the sender and the subsequent data packet from the sender to arrive at the receiver, with some leeway. It also happens to be half the time of the ACK timeout for the sender, allowing the retransmitted ACK to arrive at the sender before the sender needs to take channel contention measures.

Another issue at the receiver node is when, in the course of it being in a pipeline with

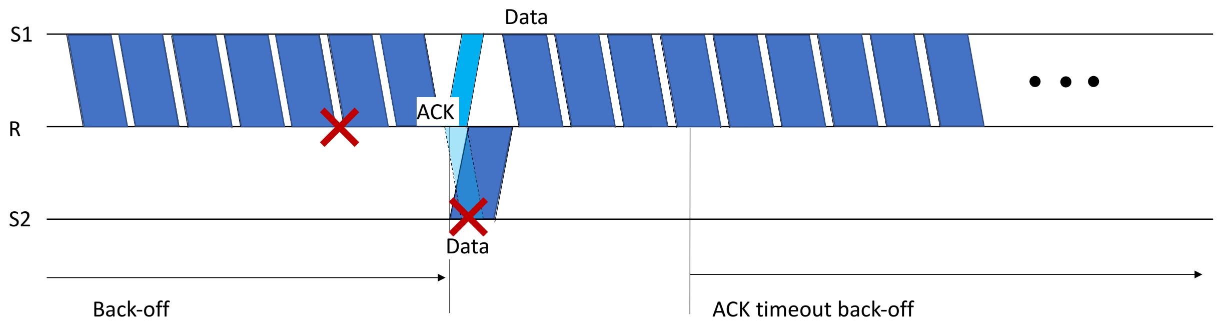


Figure 6.2: A receiver dealing with receiving a packet from a non-pipeline sender. Sender 2 transmits during the gap between pipelines and misses the ACK packet from the receiver. The Receiver receives the packet from Sender 2 but suppresses the ACK packet to preserve its pipeline with Sender 1. Eventually, Sender 2 backs off and gives a clear channel for Sender 1 to finish sending its chunk

one sender, it receives a packet from a different sender. This may occur between pipeline transmissions after the receiver has sent out its ACK packet and is waiting for the next packet from its pipelined sender. Although the senders can resolve the contention on their end through overhearing ACKs as mentioned in the previous section, the receiver must keep track of each sender with its corresponding pipeline and ACK accordingly. Additionally, it must make sure that its ACKs for one sender does not interfere with the incoming packets of a different sender. As a result, a receiving node under WUW-Aloha will only send ACK packets to a single sender at a given time period. When a receiver is pipelining with a sender, it will keep the packets from other senders, but not send back ACKs until the pipeline times out (3 RTT). This also has an added benefit of further decreasing the channel contention overhead. Figure 6.2 shows how a receiver handles such a case under WUW-Aloha.

6.2.3 Fairness

Fairness is another important issue with regard to MAC protocol design. For a simple Aloha or FAMA protocol, all nodes contend for the transmission medium randomly. If a node fails to gain access to the medium, it backs off for a random period of time before trying again.

To ensure fairness, the expected back-off time period decreases exponentially each time that the node fails to gain access to the medium. This allows all nodes to have an equal chance to transmit packets. WUW-Aloha employs something similar but requires the cooperation from the upper layers. Since WUW-Aloha mainly serves underwater exploratory sensors such as AUVs, the relationship among the nodes in terms of channel contention is a cooperative one, meaning that each node is willing to yield from the application layer. Additionally, because WUW-Aloha encourages packet transmission in chunks, each node may consequently be able to occupy the channel indefinitely after gaining it if it always has data to transmit. We combat this issue with cross-layer design, where the application layer is designated with a maximum number of packets that it can send out as a chunk (ie. maximum chunk size). After the application finishes transmitting a chunk, it will not attempt to transmit another packet for several minutes. This ensures that a different sender is able to acquire the channel and build up its pipeline with its receiver.

6.3 Protocol analysis

6.3.1 The advantage of transmitting data in packet chunks

Here we show the throughput advantages of transmitting data in chunks and the added benefits of pipelining enhancements.

Consider two scenarios where the acoustic channel is completely clear and all the sending nodes magically send data to the receiver without any collisions (ie. no time wasted on failed transmissions) and perfect pipelining at the receiver (ie. the receiver receives each packet one after another). We assume that the receiving nodes require an aggregation of k packets to interpret the data, and the duration of each packet is t seconds. In the first scenario, the data sink receives data from its senders on a per-packet basis, while in the second scenario, the data sink receives data from its senders on a per-k-packet basis.

In both cases, the data sink receives all the packets after nkt seconds, but there is a difference in the amount of time before the sink receives the first chunk of interpretable data. In the second case, since the packets are received in chunks of k packets, it would only take kt seconds. On the other hand, it would take $(n(k-1)+1)t$ seconds for the first case, which is the amount of time it takes for all n nodes to transmit $k-1$ packets plus the amount of time for the first node to transmit its last packet. Thus, the average time for the sink to receive an interpretable data chunk from each of its senders is

$$\begin{aligned}\frac{\sum_{i=1}^{n}(n(k-1)+i)t}{n} &= \frac{t\sum_{i=1}^{n}(n(k-1)+i)}{n} \\ &= t(n(k-1)+\frac{(n+1)}{2}).\end{aligned} \tag{6.1}$$

In contrast, the average time to receive an interpretable data chunk in the second case is

$$\begin{aligned}\frac{\sum_{i=1}^{n} ikt}{n} &= \frac{\frac{(n+1)n}{2}kt}{n} \\ &= \frac{n+1}{2}kt.\end{aligned} \tag{6.2}$$

To have some idea for comparison between the two time-durations, we take the ratio of equation 6.2 to 6.1, which we call α. We get

$$\alpha = \frac{\frac{k(n+1)}{2}}{(n(k-1)+\frac{(n+1)}{2})} = \frac{k(n+1)}{2n(k-1)+(n+1)}. \tag{6.3}$$

Figure 6.3 gives a graphical illustration of the average amount of time saved through sending by chunks. When 5 nodes are sending chunks of 10 packets, sending packets in whole chunks saves almost $\frac{1}{3}$ of the time as compared to sending in a packet-by-packet basis. This is significant because various delays for packet transmission in the underwater acoustic channel add up to be on the order of seconds. For example, a 1000-byte packet has a duration of over 4 seconds for a stable transmission (ie. with adequate guard times, etc.), not taking into account other hardware limitations and transmission delay.

Moreover, these two scenarios assume equal overall transmission time for both cases, which is far from realistic due to the difficulty of pipelining in the second scenario and the

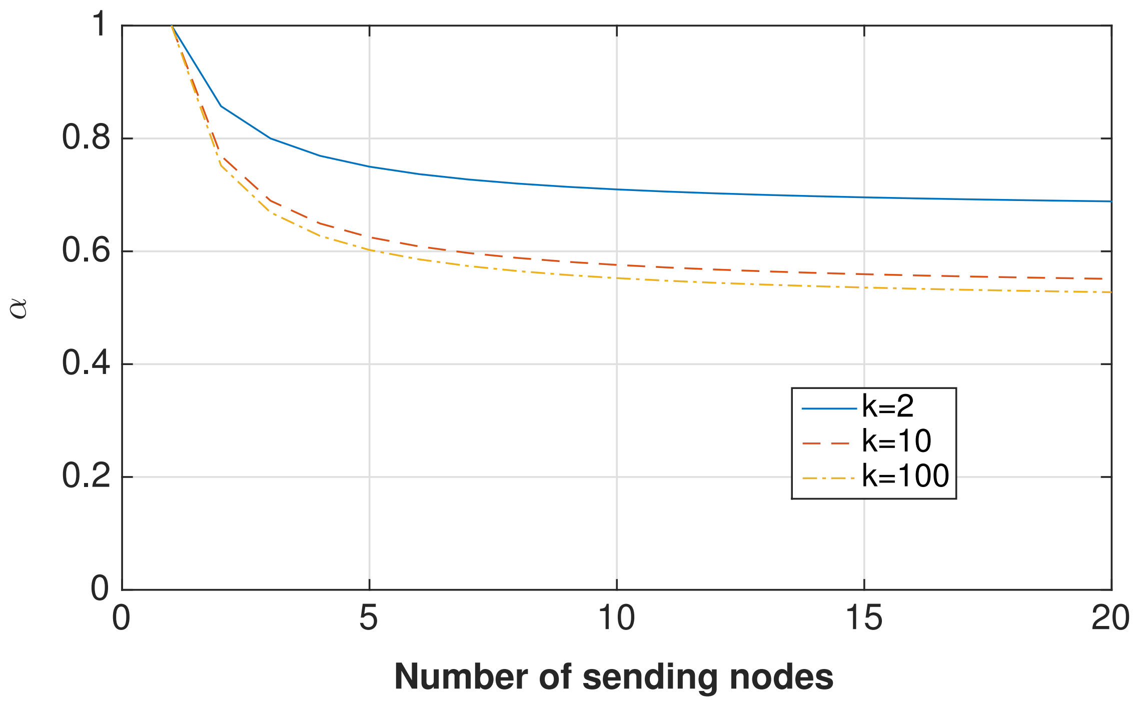


Figure 6.3: The relationship between the ratio α (as defined in Equation 6.3) and the number of transmitting nodes in the network, where each node sends out a chunk of k packets

required overhead associated with frequent context switches, such as the increased number of ACK, packets, the possible need for RTS and CTS packets, and extra time needed for channel contention.

6.3.2 A comparison of control packet overhead

Nodes using the WUW-Aloha protocol send packets in chunks and use pipelining to reduce packet overhead. In a perfectly clear channel, the sender exponentially increases the pipeline size until the threshold T is reached, whereby the pipeline size increases by one for each ACK. The relationship between the pwnd sizes, the number of data packets sent, and the number of ACK packets needed, is as follows:

Transmission phase	# pwnd Size	Data Pkts	# ACK Pkts
First packet	1	1	1
Exp. growth up to T	w	$2w-3$	$log_2 w$
Linear growth after T	$T+i, i=1,2,3..$	$2T-3+\frac{i(2T+1+i)}{2}$	$log_2 T+i$

The number of ACK packets required compared to the number of data packets sent is few. For example, if the threshold is 16, transmitting 100 packets successfully would require $i = 4$ (since $2 \times 16 - 3 + \frac{4(2\times16+1+4)}{2} = 103$) and thus only $log_2 16 + 4 = 8$ ACK packets are needed. This compares very favorably to UW-Aloha, which requires an ACK packet to be transmitted for every data packet sent. Therefore with UW-Aloha, 100 ACK packets are needed when 100 data packets are transmitted.

The other protocol to consider here is Slotted FAMA, which allow for "burst" transmissions, which are equivalent to a fixed pwnd. However, S-FAMA also relies upon RTS and CTS packets for every data transmission burst, which is similar in size to ACK packets and causes the same amount of delay. Thus, for S-FAMA, transmitting k data packets would require $3\lceil\frac{k}{T}\rceil$ control packets, where T represents the transmission burst size. This means that for a sender to send 100 data packets under SFAMA with a burst size of 16 (equivalent to the WUW-Aloha's pwnd threshold being 16), $3\lceil\frac{100}{16}\rceil = 21$ control packets are required.

Figure 6.4 gives a more comprehensible illustration of the amount of control packet overhead required for the three protocols as the number of data packets increases. In a clear channel, the number of control packets increases the slowest in WUW-Aloha and the fastest in UW-Aloha. Although control packets take significantly less time than data packets to transmit, each one of them nonetheless has at least 1 to 2 seconds of duration, and transmission delay is constant regardless of packet size.

6.3.3 Dealing with bad channels and channel contention

In a real environment with multiple senders, the channel is likely to be noisy. This affects the three protocols differently in both the average chunk completion time and overhead delay. Nodes using WUW-Aloha would require more control packets because of its adaptive pwnd.

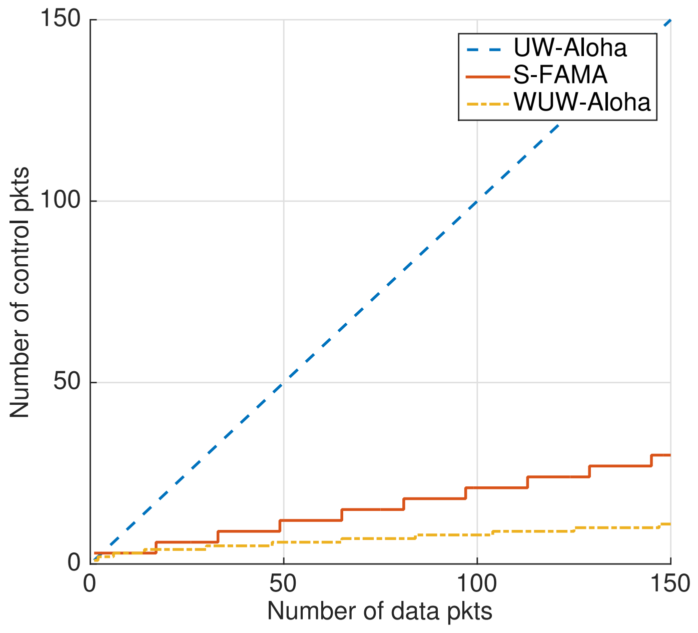


Figure 6.4: The number of control packets required compared to the number of data packets transmitted for each protocol. The burst size for S-FAMA and the pwnd threshold for WUW-Aloha are both 16.

A moderate amount of noise in the channel would cause the pwnd to increase slower to the threshold value, and likely prevent it from increasing past it. This results in the protocol requiring about $\lceil \frac{k}{T} \rceil$ ACK packets for k data packets, which is still an acceptable number. At the same time, the back-off mechanisms in the protocol allow the node that is sending data to the receiver to continue filling up the pwnd and pipeline packets to the receiver until the chunk is complete. As such, WUW-Aloha thrives when the channel is clear to moderately clear. When the channel quality is extremely poor, however, WUW-Aloha will not be able to effectively employ pipelines due to the large loss of data packets and possibly ACK packets. This in turn causes a breakdown in its efforts to send data by chunks and it would default to something similar to UW-Aloha.

Channel quality effects on UW-Aloha minimally, because UW-Aloha requires an ACK packet for each data transmission even in a clear channel. When the channel is poor, UW-Aloha's throughput decreases simply from packet loss. The nature of UW-Aloha also does not encourage chunk transmissions, though the senders are unlikely to transmit in the extreme case as illustrated by equation 6.1.

Finally, S-FAMA performs well in a clear channel with many senders because of the RTS-CTS channel reservation. However, after each successful packet burst, the sender backs off to allow a different sender to send packets for fairness. This guarantees channel contention after every T packets are transmitted, which adds significant overhead, especially when multiple RTS packets are sent in the same time slot causing all sending nodes to back off from overhearing the RTS packets. This problem is exacerbated when environmental noises cause poor channel quality because it can prevent the RTS and CTS packets from being received. Furthermore, the number of packets included in a burst is fixed in S-FAMA, so in the event that the RTS-CTS handshake is successful in a poor channel, the burst will experience very high loss rates.

6.4 Related works

Over the years many other MAC protocols have been proposed in addition to the already-mentioned UW-Aloha and S-FAMA protocols. For the Aloha-based protocol, there is PDT-ALOHA [AK08], which seeks primarily to overcome propagation delay issues present in the simple Slotted ALOHA protocol by introducing guard bands at time slots. ALOHA-CA and ALOHA-AN [CSC07] are two more similar protocols. They take advantage of propagation delays in the underwater environment and make use of overheard packet information to determine whether to transmit or to back off. Interestingly, our WUW-Aloha protocol leverages the same technique in that it checks for the destination of the overheard packets as well as the pipeline window size to determine whether to back off or to send a new packet.

DOTS [NLH14] and M-FAMA [HNL13] are two underwater MAC protocols that use location information of the underwater nodes in addition to the long propagation delay to allow for concurrent transmission among nodes. In a way, this is using pipelines but in a different meaning. Rather than establishing a pipeline between a sender-receiver pair, DOTS and M-FAMA allows many senders to “pipeline" with one another to send single data packets to the receiver based on expected propagation delay inferred from locations. However, this is difficult to implement in a real system because of the relative unpredictability of the underwater channel and arbitrary positioning of nodes.

Similar to S-FAMA, there are many other protocols that use hand shaking to reserve the channel and minimize packet collisions. DACAP [PS07] is one such example, which saves power just as S-FAMA does, but sees an improved throughput. DACAP works by using the RTS-CTS handshake, with an extra warning message from the receiver when needed. The warning message is sent before the sender begins to send its data packets if the receiver overhears other packets that will cause interference on its end. MACA-MN [CSC08] implements handshakes in a packet train, allowing a sender to establish handshakes with multiple neighbors using one RTS packet. However, this does not fit the underwater usage scenario very well, since in both search and monitoring applications, the situation is for the most part multiple senders with a single receiver.

In addition to these, many other underwater acoustic MAC protocols exist. A comprehensive survey of almost all currently existing protocols is found in [CMC14], which not only lists the protocols but also groups them by different attributes. For example, there are “contention-free MAC protocols", which includes FDMA, TDMA, and CDMA techniques, and “contention-based MAC protocols". Contention-based protocols are further subdivided into random access protocols and handshaking protocols. Random access protocols include various ALOHA and CSMA protocols, whereas handshaking protocols focus on channel reservation methods to minimize interference. Handshaking protocols can also be done either on a single channel or by using multiple channels. Some examples of multiple channel handshaking protocols include RCAMAC [TR08], COPE-MAC [PZZ10], and UMIMO-MAC [KM11].

6.5 Evaluation

We evaluate the WUW-Aloha protocol by comparing it against the S-FAMA and UW-Aloha protocols in our state-of-the-art testbed as described in chapter 5, using both our Marina del Rey testbed and emulator to compare the three protocols.

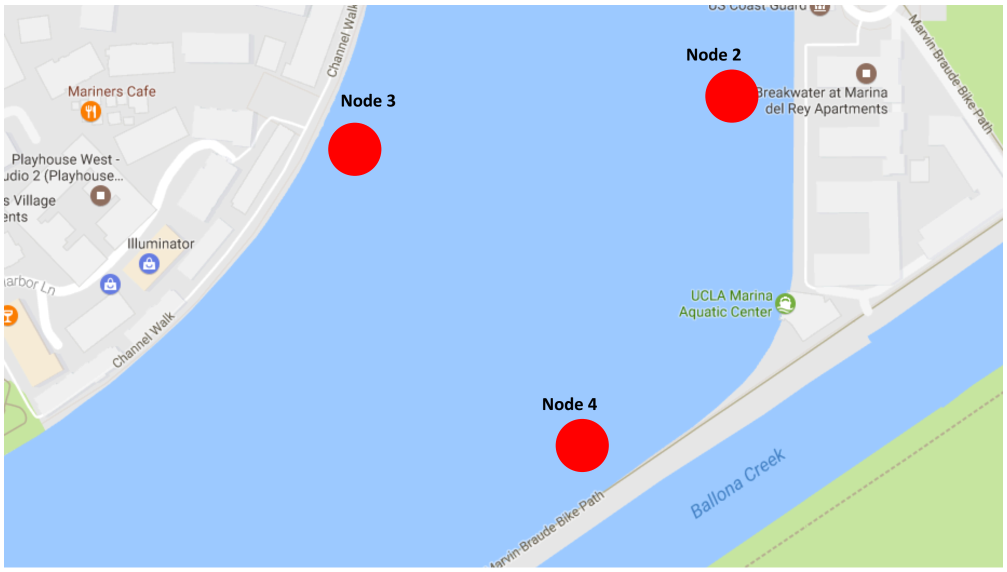

Node 2
Node 3
Node 4
Mariners Cafe
Playhouse West - Studio 2 (Playhouse...
Illuminator
Channel Walk
Breakwater at Marina del Rey Apartments
UCLA Marina Aquatic Center
Marvin Braude Bike Path
Ballona Creek
US Coast Guard

Figure 6.5: The deployment topology of our acoustic modems - Nodes 2 and 4 transmit data to Node 3

6.5.1 Performance in Marina del Rey, with real modems

For the Marina del Rey testbed, we position the three OFDM acoustic radios in such a way that two nodes are about the same distance from the third node. The first two nodes are the data transmitters while the third node receives. This simulates an exploration scenario of one controller/data sink with two AUVs that transmit exploratory data to it. Figure 6.5 shows how the three modems are deployed in the marina.

The transmission settings of the modems are as follows:

- Modulation Mode (Maximum Data per block): Mode 3 (122 B/Blk)
- Number of blocks: 10
- Guard time between blocks: 150 ms
- Actual Packet Size: 1152 B

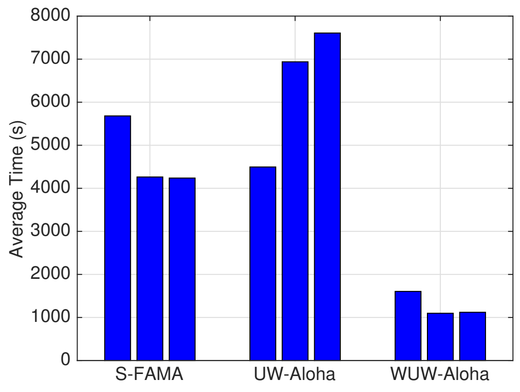
These settings, along with the delay inherent to the OFDM modem, such as modulation delay, warm up time, etc., mean that it takes a little more than 4 seconds for the modem to send out a packet.

The settings for each of the three tested protocols are as follows:

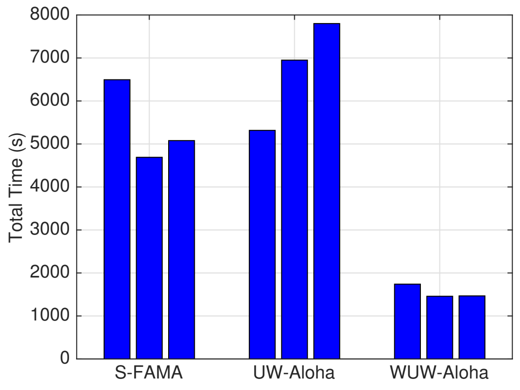
Protocol	Parameter	Value
UW-Aloha	ACK Timeout (s)	25
	Max Retransmissions	3
S-FAMA	Max Queue Length	200
	Max Burst Rate	16
WUW-Aloha	ACK Timeout (s)	30
	Pipeline Timeout (s)	30
	Initial Pipeline Threshold	16

We leave UW-Aloha to its default settings, with the ACK timeout at 25 seconds and allowing for 3 retransmissions. For S-FAMA, we set a large queue length because we send a large number of packets continuously, and S-FAMA is not able to catch up. We also set the maximum burst rate for S-FAMA equal to the initial pipeline threshold for WUW-Aloha for a fairer comparison, allowing S-FAMA to send out packets in bursts similar in size to the pipeline used by WUW-Aloha. The ACK timeout for WUW-Aloha is slightly longer than that of UW-Aloha. This is because the consequences for an ACK timeout are more severe in WUW-Aloha, where a potentially large pipeline can be shrunken down to 1. The longer ACK timeout gives the receiver a chance to retransmit the ACK packet in the event of an ACK packet loss. Finally, we set the pipeline timeout to be the same as the ACK timeout, so that the receiver can immediately switch over to a different sender once the original sender experiences an ACK timeout and must back off before being able to send again.

We send data in chunks of 150 packets from Nodes 2 and 4 to Node 3 using the three different protocols and record the time required for the first chunk from each of the two nodes to finish sending. Between chunks, there is a larger gap of packets from the application layer to simulate that a large piece of data has been sent out, and the AUV is now collecting another large piece of data. We start all three of the nodes at the same time, meaning that the senders will always begin with channel contention. This ensures that the experiment fully tests the channel contention aspects of each protocol. We run the experiment with



(a) The average time spent for each of the two sending node to finish transmitting its packets



(b) The total time spent for both sending nodes to finish transmitting their packets

Figure 6.6: Transmission times for two nodes to send 150 packets each to the receiver, 3 trials for each protocol

each protocol 3 times and measure the amount of time the senders take to finish their transmissions, the packet delivery ratio, as well as the amount of control packet overhead received on the receiving node throughout the course of the transmission.

Figure 6.6 shows the amount of time required for the nodes to finish transmitting data to the receiver. Each bar represents a trial with that protocol. S-FAMA and UW-Aloha are comparable to each other, though S-FAMA performs slightly better on average. Our WUW-Aloha protocol outperforms both S-FAMA and UW-Aloha by a large margin. The sending nodes in every WUW-Aloha trial took less than half the amount of time as the sending nodes in all of the other two protocol trials. S-FAMA outperforming UW-Aloha in our experiment is contrary to our results from Chapter 2. We attribute this to two reasons - 1) The environment at the Marina, with less signal reflection and fewer transmitting nodes, means that the channel is clearer. This allows S-FAMA to take advantage of its RTS and CTS packets and successfully send bursts of packets, which decreases the control packet overhead. 2) The two experiments use completely different data stream models. In the experiments from Chapter 2, data were generated and sent out one packet at a time randomly in a Poisson distribution. This decreased the chances of packet collisions and thus increased the speed and success rates of UW-Aloha. At the same time, it neutralized the advantages of packet bursting in S-FAMA, effectively forcing sending nodes to use RTS and CTS every time they need to transmit a single packet.

To showcase the effectiveness of sending packets in chunks, we graph the transmission time in two ways. Figure 6.6a shows the average amount of time for each node to finish transmitting a chunk, while 6.6b is the total amount of time taken for both nodes to finish transmitting, which is effectively the longer of the two individual transmission times. The percent difference between the two times for each trial run is recorded as follows:

Protocol	% difference between average time and total time
S-FAMA 1	12.51%
S-FAMA 2	9.09%
S-FAMA 3	16.54%
UW-Aloha 1	15.44%
UW-Aloha 2	0.16%
UW-Aloha 3	2.48%
WUW-Aloha 1	7.72%
WUW-Aloha 2	24.70%
WUW-Aloha 3	23.60%

In general, WUW-Aloha sees the biggest percent difference between average time and total time, which is unsurprising because nodes transmit data in chunks under WUW-Aloha. Trial 1 of WUW-Aloha is an outlier because bad channel conditions periodically prompted the each of the transmitting nodes to drop from the pipeline and allow the other node to take over. In normal channel conditions, however, one can expect each node to receive packets in over 20% less time as the total time using WUW-Aloha. For the other two protocols, S-FAMA generally sees a greater difference because of its packet burst feature.

We also record the packet delivery ratios of the three protocols in Figure 6.7. While the three protocols are similar in performance, UW-Aloha is visibly the worst performing

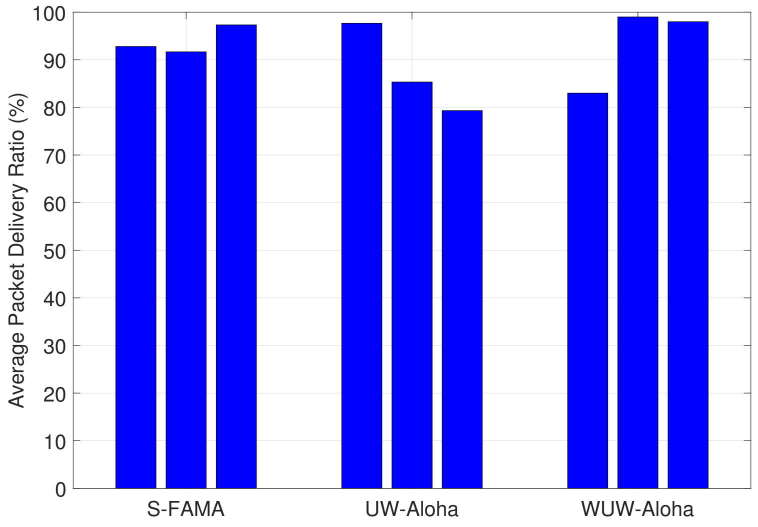


Figure 6.7: The packet delivery ratio for two nodes to send 150 packets to the receiver, 3 trials for each protocol

protocol. We again attribute this to the fact that our new packet generation model, which generates many packets continuously, causes more collisions for UW-Aloha. Between WUW-Aloha and S-FAMA, the packet delivery ratios are more similar. Averaging all three trials, S-FAMA closely edges out WUW-Aloha, but two instances of WUW-Aloha outperform all three trials of S-FAMA, while the other one performs very poorly. Once again, the outlier is trial 1 of WUW-Aloha, with the packet delivery ratio of just over 80%. This actually confirms the issues with channel quality during this trial as discussed earlier. Note that these channel quality issues are not due to packet collisions, but environmental factors, such as water temperature, waves, and engine noise from water vessels. In any case, S-FAMA, UW-Aloha, and WUW-Aloha all have reasonably good packet delivery ratios in the real underwater environment. However, since WUW-Aloha has much higher throughput, it is the obvious choice when sending large data chunks in the underwater environment.

We also aggregate the number of control packets transmitted compared to the number of data packets received. For every data packet received at the receiver node, we count the number of corresponding control packets sent out, as well as any other corresponding control packets that it receives. We choose trials 3, 1, and 2 for protocols S-FAMA, UW-

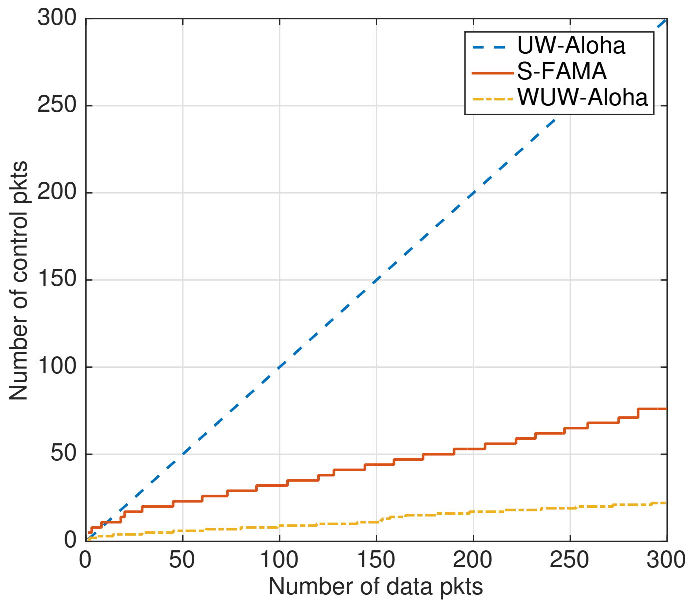


Figure 6.8: The number of control packets transmitted compared to the number of data packets received at the receiving node for each protocol in the experiment

Aloha, and WUW-Aloha respectively, which have the highest packet delivery ratios for each protocol and plot them in Figure 6.8. These results match up nicely with our earlier analysis and actually looks very similar to Figure 6.4. One major underlying difference is that the analysis section considers all packets in a single sender, single receiver system with perfect channel qualities, while this is simply a snapshot of packet aggregation from the receiver, with two different senders, in the real, albeit imperfect, underwater environment. The receiver for UW-Aloha sends an ACK packet for every data packet received (including possibly duplicate packets, which we do not take into consideration here), resulting in the exact same number of control packets sent as the number of data packets received. The S-FAMA and WUW-Aloha protocols also bear strong similarities to their counterparts from Figure 6.4, though not exactly the same. The number of control packets for S-FAMA increases slightly faster for each data packet received than the theoretical results. We attribute this to having 2 senders that contend for the channel with control packets, as well as imperfect channel conditions that cause data packet losses. Both of these factors increase the control to data packet ratio. WUW-Aloha is interesting at the small section after the 150th data packet. From there the

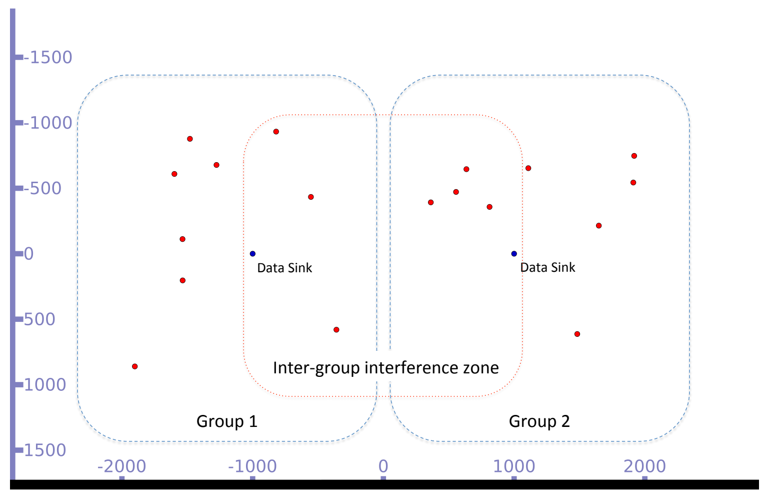


Figure 6.9: The NS3 emulation topology: 2 groups of 10 nodes, the blue nodes are the data sinks for each group; red nodes in the "inter-group interference zone" will interfere with one another as well as with the blue data sink nodes of the opposite group

growth pattern from the control packets mimics the beginning, having a slightly faster initial growth that slows down. This is the effect of transmitting data in chunks, where the first node has finished its packet transmissions for its chunk and the second node is beginning to build its pipeline. In all, these results validate our analysis of the control packet overheads of each protocol and how they impact the rates of data transmission.

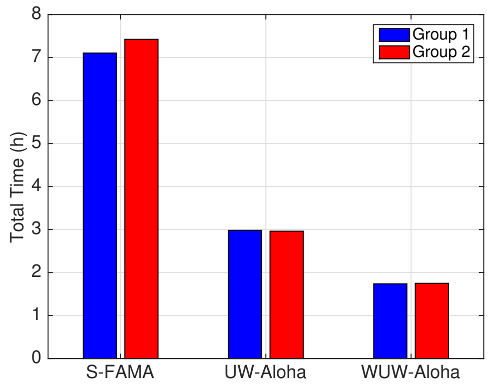
6.5.2 Performance with more nodes in the emulator

To see how our protocol performs when more nodes are present, we emulate a scenario with 20 nodes, each capable of guaranteeing transmission for up to 1.5km. These twenty nodes are divided into two different groups of ten nodes, in which one of the nodes is the data sink and the other nine are the transmitters. This simulates a topology of two groups of sensors that monitor an underwater infrastructure, such as a large oil drilling operation or as part of a costal defense system. The two data sinks for the two groups are 2km away and thus will not interfere with each other. The transmitting nodes for each data sink are placed in a random fashion within a 1km radius from their corresponding data sink. This means that certain transmitting nodes from different groups will interfere with each other, as well as with the data sink of the other group. Figure 6.9 shows the emulation topology.

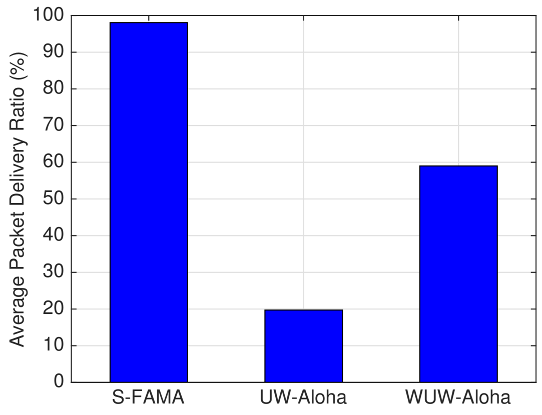
Since the provided NS3 UanNetDevice does not come with OFDM radio implementations, we use the FSK modulation model. However, we set the simulated modem speed equal to the speed of our AquaSeNt acoustic radios. The following are the NS3 simulation parameters used to run the emulation.

- Modem modulation: FSK BPSK
- Data Rate: 2264 bps
- Center Frequency: 24 kHz
- Bandwidth: 6 kHz
- Channel spreading coefficient: 1.5 (scale between 1-2)
- Receiving threshold: 5 dB
- Transmitting power: 140 dB
- Wind speed: 4.5 m/s
- Ship noise: 0.5 (scale between 0-1)

Similar to the Marina testbed experiment, the nodes transmit packets in chunks to the data sink, this time in groups of 100 packets rather than 150. Because there are now two data sinks, inter-group interference becomes a factor, where nodes located between the two data sinks can overhear ACK packets from the opposite data sink, as well as interfere with packets sent to the opposite data sink. Additionally, more transmitting nodes mean more data to be transmitted and more interference, even with a single data sink. Once again, we compare the three MAC protocols, UW-Aloha, S-FAMA, and WUW-Aloha under this scenario.



(a) The total time spent for all the nodes in each group of 10 nodes to finish transmitting their packets



(b) The average packet delivery rate for each protocol

Figure 6.10: Emulation results for two groups of 20 nodes

Figure 6.10 show the total time required for each of the group to finish as well as the average packet delivery ratio for each protocol. While S-FAMA boasts a near-perfect average packet delivery ratio, both groups took over 7 hours for all the nodes to finish transmitting their first data chunks. This is due to the over-cautiousness of the protocol, where oftentimes all of the nodes are backing off due to overhearing one another's RTS packets. Coming in at 3 hours, UW-Aloha is significantly faster, but it also has a packet delivery ratio of mere 20%. This is the result of having too many simultaneous transmissions at a high volume, and though UW-Aloha has re-transmission mechanics, it gives up after three tries. Finally, our WUW-Aloha performs the fastest, coming in at less than 2 hours with a much higher packet delivery ratio of close to 60%. We attribute this to two main reasons. First, WUW-Aloha encourages pipelining, greatly increasing the throughput since the greater distance between nodes cause longer transmission delays. In addition, WUW-Aloha dictates that nodes back off upon overhearing all ACK packets and any data packets meant for the same destination. This allows it to achieve a similar, though far less stringent, effect of S-FAMA's RTS-CTS handshake, giving the pipelining nodes a clearer channel to transmit. The performances of S-FAMA and WUW-Aloha show a clear trade-off between high packet delivery ratio and packet transmission rates. While S-FAMA strives for a near perfect packet delivery ratio by

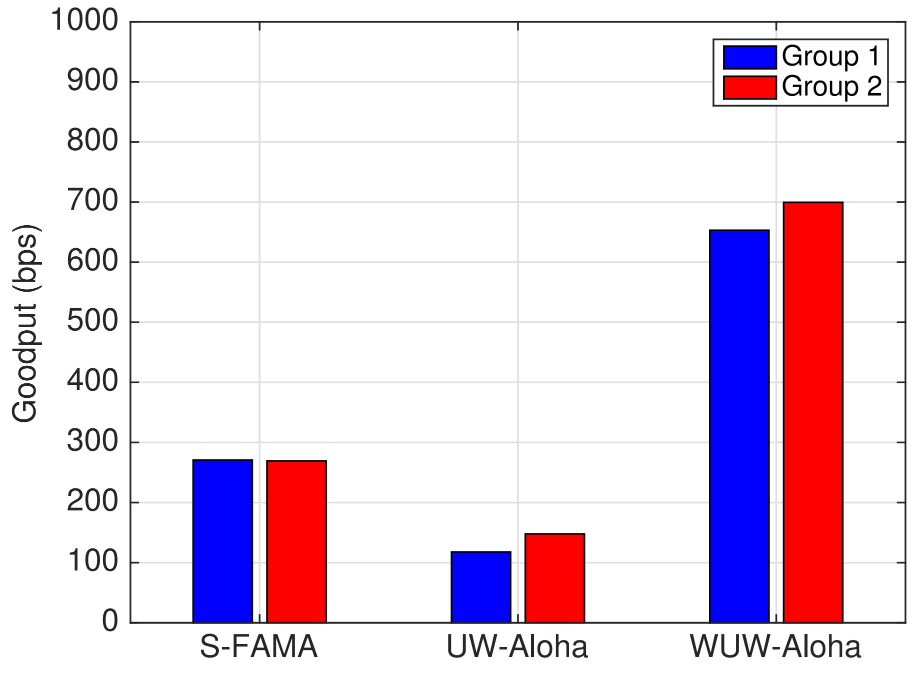


Figure 6.11: The resulting overall goodput in each of the 10-node groups

placing heavy restrictions on when a node may transmit its packet, it sacrifices too much potential throughput to achieve it, ultimately making it impractical because of its slow speed. On the other hand, WUW-Aloha is designed to maximize throughput as its first priority, with reliable transfer added as an enhancement to increase throughput. As a result, it is able to take a small fraction of the time as S-FAMA while still keeping an acceptable packet delivery ratio.

To have a clearer picture of the effects how each protocol performed taking into account both their throughput and packet delivery ratio, we also plot their goodput in Figure 6.11. Goodput measures the throughput of useful data only, thus combining the two metrics above into one. As expected, WUW-Aloha is once again the clear winner among the three protocols, with both groups' goodput more than triple those of UW-Aloha and more than doubling those of S-FAMA. UW-Aloha's performance compared to S-FAMA shows that though it allows the nodes to finish transmitting faster, the amount of sacrifice made on transmission reliability causes it to ultimately underperform compared to S-FAMA.

6.6 Conclusion

In this chapter, we presented a new underwater MAC protocol, Windowed Underwater Aloha, which uses pipelines and transmitting data in chunks to increase data throughput. WUW-Aloha borrows the idea of adaptive window sizes from the TCP transport protocol to decrease the amount of control packet overhead for performance gains. Our experiments comparing it against the two existing S-FAMA and UW-Aloha protocols proved its superiority over both, more than doubling either protocol in data throughput and at the same time maintaining a high data delivery rate. By incorporating WUW-Aloha as their MAC protocol, AUV networks will see great improvements, with the ability to reliably transmit data at higher speeds.

CHAPTER 7

Conclusion and Future Work

At the beginning of this dissertation, we introduced a full model of an AUV networking system. This system functions by employing a centralized SDN controller that also doubles as an underwater information sink. While underwater optical modem hardware technologies are still maturing, our system anticipates for the future and can take full advantage of an acousto-optic hybrid system, by using the longer ranged, omnidirectional acoustic channel as the SDN control plane and the faster but shorter ranged optic channel as the SDN data plane.

We then explore several issues related to this system in isolation. First, we address delay-tolerant routing that will be required for almost all underwater networks due to bandwidth and latency constraints present in underwater channels. Although the geo-routing DTN solution presented in Chapter 3 is focused on RF networking with street vehicles, we have outlined similarities between a VANET and our AUV network system. Moreover, we presented an easy way to apply RobustGeo principles to our underwater network system, where directional transmission technologies such as optics that also require localization can benefit from such a delay tolerant routing scheme.

To increase battery efficiency in deployed nodes in an underwater WSN, we addressed power conservation methods for wireless sensors in Chapter 4. Wireless sensors spend a substantial amount of their power on data transmissions, even when keeping the radios idle. Neighbor discovery techniques allow sensors to periodically turn off their radios to save power, while still guaranteeing eventual transmission success. We discussed our design of two protocols to facilitate neighbor discovery among wireless sensor nodes and showed their effectiveness in closely matching any granularity of battery duty cycles while guaranteeing discovery in bounded time. These protocols allow WSNs and AUVs to be deployed inside the ocean for longer periods of time without needing to recharge.

Following this, we described our new underwater networking testbed in Chapter 5. This testbed is a huge upgrade to the previous WaterCom testbed, which consisted of six underwater acoustic OFDM modems in a water tank. Not only does the new testbed bring the OFDM modems into Marina Del Rey for more accurate channel conditions, but it also has an extra emulation mode, allowing users to run experiments with more nodes using the NS3 simulated channel.

Finally, we presented, in Chapter 6, a new MAC protocol called Windowed Underwater Aloha for underwater acoustic radios that uses adaptive pipelines to increase throughput without sacrificing much reliability. WUW-Aloha's adaptive pipelines work in a similar way as its inspiration, the TCP congestion window, except they do not halve each time a packet is lost. Rather, they follow a more moderate approach and decrease their sizes to match the number of packets previously transmitted successfully. We also discussed the advantages of single nodes sending packets in chunks over many nodes taking turns sending one packet at a time. Using the newly developed testbed, we compared WUW-Aloha against two other underwater MAC protocols, UW-Aloha and S-FAMA. According to results, WUW-Aloha is more than twice as fast as both UW-Aloha and S-FAMA, while having comparable packet delivery rates in normal channel conditions in Marina Del Rey. Such a MAC protocol greatly advances the feasibility of underwater networking using acoustics, allowing nodes to transmit data reliably at much higher speeds.

In all, we designed a complete software-defined underwater wireless sensor network system with prototypical implementations. Along the way, we addressed many issues, proving the feasibility of such a system. Our system provides an adequate blueprint for any underwater WSN implementations, whether it is for AUVs or stationary sensors, using pure acoustics or acousto-optic hybrid. In this way, our system has the required flexibility to pave the way for the future of underwater communications.

For future work, we are continuing to improve our testbed by outfitting the kayaks with more sturdy materials so that they can be deployable in the open ocean. In preparation for this, we have already purchased the necessary materials such as underwater pingers and raw materials. On the protocols front, more work can be done in the optical channel relating to MAC and routing protocols. With the possible development of an underwater optical channel of NS3, tests for RobustGeo underwater should be conducted. These would then lead to a full implementation of the network system based on SDN, which would be immensely useful for any underwater exploration endeavor.

With the proper integration of these technologies, we are confident that the dream of wireless connectivity under the ocean will be realized, bringing humanity another step closer to the future of oceanic exploration.

REFERENCES

[ABP10] Davide Anguita, Davide Brizzolara, and Giancarlo Parodi. "VHDL modules and circuits for underwater optical wireless communication systems." *WSEAS Trans. Commun*, **9**(9):525–552, 2010.

[ACD09] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. "Energy conservation in wireless sensor networks: A survey." *Ad Hoc Networks*, **7**(3):537–568, 2009.

[AK08] Joon Ahn and Bhaskar Krishnamachari. "Performance of a propagation delay tolerant ALOHA protocol for underwater wireless networks." In *International Conference on Distributed Computing in Sensor Systems*, pp. 1–16. Springer, 2008.

[AK09] Shlomi Arnon and Debbie Kedar. "Non-line-of-sight underwater optical wireless communication network." *JOSA A*, **26**(3):530–539, 2009.

[ASS00] Thomas C Austin, Roger P Stokey, and Kenneth M Sharp. "PARADIGM: a buoy-based system for auv navigation and tracking." In *OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 2, pp. 935–938. IEEE, 2000.

[BDS94] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. "MACAW: a media access protocol for wireless LAN's." In *ACM SIGCOMM Computer Communication Review*, volume 24, pp. 212–225. ACM, 1994.

[Bly11] Mark Bly. *Deepwater Horizon accident investigation report.* Diane Publishing, 2011.

[BTK12] Mehedi Bakht, Matt Trower, and Robin Hilary Kravets. "Searchlight: won't you be my neighbor?" In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pp. 185–196. ACM, 2012.

[CLG10] Pei-Chun Cheng, Kevin C. Lee, Mario Gerla, and Jérôme Härri. "GeoDTN+Nav: Geographic DTN Routing with Navigator Prediction for Urban Vehicular Environments." *Mob. Netw. Appl.*, **15**(1):61–82, February 2010.

[CMC14] Keyu Chen, Maode Ma, En Cheng, Fei Yuan, and Wei Su. "A survey on MAC protocols for underwater wireless sensor networks." *IEEE Communications Surveys & Tutorials*, **16**(3):1433–1447, 2014.

[CSC07] Nitthita Chirdchoo, W-S Soh, and Kee Chaing Chua. "Aloha-based MAC protocols with collision avoidance for underwater acoustic networks." In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 2271–2275. IEEE, 2007.

[CSC08] Nitthita Chirdchoo, Wee-Seng Soh, and Kee Chaing Chua. "MACA-MN: A MACA-based MAC protocol for underwater acoustic networks with packet train for multiple neighbors." In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pp. 46–50. IEEE, 2008.

[CSL08] Xiuzhen Cheng, Haining Shu, Qilian Liang, and David Hung-Chang Du. "Silent positioning in underwater acoustic sensor networks." *Vehicular Technology, IEEE Transactions on*, **57**(3):1756–1766, 2008.

[CW07] Yin-Jun Chen and Hao-Li Wang. "Ordered CSMA: a collision-free MAC protocol for underwater acoustic networks." In *OCEANS 2007*, pp. 1–6. IEEE, 2007.

[CZ11] Paolo Casari and Michele Zorzi. "Protocol design issues in underwater acoustic networks." *Computer Communications*, **34**(17):2013–2025, 2011.

[DC08] Prabal Dutta and David Culler. "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications." In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pp. 71–84. ACM, 2008.

[DR10] Marek Doniec and Daniela Rus. "BiDirectional optical communication with AquaOptical II." In *Communication Systems (ICCS), 2010 IEEE International Conference on*, pp. 390–394. IEEE, 2010.

[FBW10] N Farr, A Bowen, J Ware, C Pontbriand, and M Tivey. "An integrated, underwater optical/acoustic communications system." In *OCEANS 2010 IEEE-Sydney*, pp. 1–6. IEEE, 2010.

[FG95] Chane L Fullmer and JJ Garcia-Luna-Aceves. *Floor acquisition multiple access (FAMA) for packet-radio networks*, volume 25. ACM, 1995.

[FMC05] FMC Technologies. "Offshore Capabilities." `https://www.sec.gov/Archives/edgar/data/1135152/000119312505181482/dex991.htm`, 2005.

[FN01] Laura Marie Feeney and Martin Nilsson. "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment." In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pp. 1548–1557. IEEE, 2001.

[GH12] Valerie Galluzzi and Ted Herman. "Survey: discovery in wireless sensor networks." *International Journal of Distributed Sensor Networks*, **2012**, 2012.

[GMS15] Ciarán Mc Goldrick, Mark Matney, Enrique Segura, Youngtae Noh, and Mario Gerla. "WaterCom: A Multilevel, Multipurpose Underwater Communications Test Platform." In *Proceedings of the 10th International Conference on Underwater Networks & Systems*, p. 14. ACM, 2015.

[HA06] Khaled A Harras and Kevin C Almeroth. "Transport layer issues in delay tolerant mobile networks." In *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, pp. 463–475. Springer, 2006.

[Hah05] Matthew J Hahn. "Undersea navigation via a distributed acoustic communications network." Technical report, DTIC Document, 2005.

[HFB06] J. Härri, F. Filali, C. Bonnet, and Marco Fiore. "VanetMobiSim: Generating Realistic Mobility Patterns for VANETs." In *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks*, VANET '06, pp. 96–97, New York, NY, USA, 2006. ACM.

[HNL13] Seongwon Han, Youngtae Noh, Uichin Lee, and Mario Gerla. "M-FAMA: A multi-session MAC protocol for reliable underwater acoustic streams." In *INFOCOM, 2013 Proceedings IEEE*, pp. 665–673. IEEE, 2013.

[Huh04] Aleksandr Huhtonen. "Comparing AODV and OLSR routing protocols." *Telecommunications Software and Multimedia*, pp. 1–9, 2004.

[JFP04] Sushant Jain, Kevin Fall, and Rabin Patra. "Routing in a Delay Tolerant Network." In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pp. 145–158, New York, NY, USA, 2004. ACM.

[JSM07] Moez Jerbi, S-M Senouci, Rabah Meraihi, and Yacine Ghamri-Doudane. "An improved vehicular ad hoc routing protocol for city environments." In *Communications, 2007. ICC'07. IEEE International Conference on*, pp. 3972–3979. IEEE, 2007.

[JTH05] Jehn-Ruey Jiang, Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Hwang Lai. "Quorum-based asynchronous power-saving protocols for IEEE 802.11 ad hoc networks." *Mobile Networks and Applications*, **10**(1-2):169–181, 2005.

[KK00] Brad Karp and Hsiang-Tsung Kung. "GPSR: Greedy perimeter stateless routing for wireless networks." In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 243–254. ACM, 2000.

[KLR10] Arvind Kandhalu, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. "U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol." In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 350–361. ACM, 2010.

[KM11] Li-Chung Kuo and Tommaso Melodia. "Distributed medium access control strategies for MIMO underwater acoustic networking." *IEEE Transactions on Wireless Communications*, **10**(8):2523–2533, 2011.

[KRE15] Diego Kreutz, Fernando MV Ramos, P Esteves Verissimo, C Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. "Software-defined networking: A comprehensive survey." *Proceedings of the IEEE*, **103**(1):14–76, 2015.

[Lan14] Lanca VideosHD. "Shark Bites Fiber Optic Cables Undersea 15.8.2014. [YouTube video]." `https://www.youtube.com/watch?v=XMxkRh7sx84`, August 2014.

[LCW08] Kevin C Lee, Pei-Chun Cheng, Jui-Ting Weng, Lung-Chih Tung, and Mario Gerla. "VCLCR: a practical geographic routing protocol in urban scenarios." *UCLA Computer Science Department, Tech. Rep. TR080009*, 2008.

[Lee10] Kevin C Lee. *Geographic routing in vehicular ad hoc networks.* University of California at Los Angeles, 2010.

[LHL07] Kevin C Lee, Jér Ome Häerri, Uichin Lee, and Mario Gerl. "Enhanced perimeter routing for geographic forwarding protocols in urban vehicular scenarios." In *Globecom Workshops, 2007 IEEE*, pp. 1–10. IEEE, 2007.

[LLG09] Kevin C Lee, Uichin Lee, and Mario Gerla. "TO-GO: TOpology-assist geo-opportunistic routing in urban vehicular grids." In *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*, pp. 11–18. IEEE, 2009.

[LM07] Ilias Leontiadis and Cecilia Mascolo. "Geopps: Geographical opportunistic routing for vehicular networks." In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pp. 1–6. IEEE, 2007.

[LMF05] Christian Lochert, Martin Mauve, Holger Füßler, and Hannes Hartenstein. "Geographic routing in city scenarios." *ACM SIGMOBILE Mobile Computing and Communications Review*, **9**(1):69–72, 2005.

[LPC13] Son N Le, Zheng Peng, Jun-Hong Cui, Hao Zhou, and Janny Liao. "Sealinx: A multi-instance protocol stack architecture for underwater networking." In *Proceedings of the Eighth ACM International Conference on Underwater Networks and Systems*, p. 46. ACM, 2013.

[LRC10] Shouwen Lai, Binoy Ravindran, and Hyeonjoong Cho. "Heterogenous quorum-based wake-up scheduling in wireless sensor networks." *Computers, IEEE Transactions on*, **59**(11):1562–1575, 2010.

[LWB08] G. Leus, P. van Walree, J. Boschma, C. Fanciullacci, H. Gerritsen, and P. Tusoni. "Covert underwater communications with multiband OFDM." In *OCEANS 2008*, pp. 1–8, Sept 2008.

[LYG14] Zonglin Li, Nianmin Yao, and Qin Gao. "Relative distance based forwarding protocol for underwater wireless networks." *International Journal of Distributed Sensor Networks*, **10**(2):173089, 2014.

[LZS07] Baosheng Li, Shengli Zhou, M. Stojanovic, L. Freitag, Jie Huang, and P. Willett. "MIMO-OFDM Over An Underwater Acoustic Channel." In *OCEANS 2007*, pp. 1–6, Sept 2007.

[MB01] Michael J McGlynn and Steven A Borbash. "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks." In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pp. 137–145. ACM, 2001.

[MS07] Marcal Molins and Milica Stojanovic. "Slotted FAMA: a MAC protocol for underwater acoustic networks." In *OCEANS 2006-Asia Pacific*, pp. 1–7. IEEE, 2007.

[Nat00] Melvyn B Nathanson. *Elementary methods in number theory*, volume 195. Springer, 2000.

[NLH14] Youngtae Noh, Uichin Lee, Seongwon Han, Paul Wang, Dustin Torres, Jinwhan Kim, and Mario Gerla. "DOTS: A propagation delay-aware opportunistic MAC protocol for mobile underwater networks." *IEEE Transactions on Mobile Computing*, **13**(4):766–782, 2014.

[NLL16] Youngtae Noh, Uichin Lee, Saewoom Lee, Paul Wang, Luiz FM Vieira, Jun-Hong Cui, Mario Gerla, and Kiseon Kim. "Hydrocast: pressure routing for underwater sensor networks." *IEEE Transactions on Vehicular Technology*, **65**(1):333–347, 2016.

[NLW13] Youngtae Noh, Uichin Lee, Paul Wang, Brian Sung Chul Choi, and Mario Gerla. "VAPR: void-aware pressure routing for underwater sensor networks." *IEEE Transactions on Mobile Computing*, **12**(5):895–908, 2013.

[PKL07] Jim Partan, Jim Kurose, and Brian Neil Levine. "A survey of practical issues in underwater networks." *ACM SIGMOBILE Mobile Computing and Communications Review*, **11**(4):23–33, 2007.

[PS06] Borja Peleato and Milica Stojanovic. "A MAC protocol for ad-hoc underwater acoustic sensor networks." In *Proceedings of the 1st ACM international workshop on Underwater networks*, pp. 113–115. ACM, 2006.

[PS07] Borja Peleato and Milica Stojanovic. "Distance aware collision avoidance protocol for ad-hoc underwater acoustic sensor networks." *IEEE Communications Letters*, **11**(12), 2007.

[PSG09] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. "CRAWDAD data set epfl/mobility (v. 2009-02-24)." Downloaded from http://crawdad.org/epfl/mobility/, February 2009.

[PZC09] Zheng Peng, Zhong Zhou, Jun-Hong Cui, and Zhijie Jerry Shi. "Aqua-Net: An underwater sensor network architecture: Design, implementation, and initial testing." In *OCEANS 2009, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*, pp. 1–8. IEEE, 2009.

[PZZ10] Zheng Peng, Yibo Zhu, Zhong Zhou, Zheng Guo, and Jun-Hong Cui. "COPE-MAC: a contention-based medium access control protocol with parallel reservation for underwater acoustic networks." In *OCEANS 2010 IEEE-Sydney*, pp. 1–10. IEEE, 2010.

[RDM07] S. Roy, T.M. Duman, V. McDonald, and J.G. Proakis. "High-Rate Communication for Underwater Acoustic Channels Using Multiple Transmitters and Space - Time Coding: Receiver Structures and Experimental Results." *Oceanic Engineering, IEEE Journal of*, **32**(3):663–688, July 2007.

[SLL04] Boon-Chong Seet, Genping Liu, Bu-Sung Lee, Chuan-Heng Foh, Kai-Juan Wong, and Keok-Kee Lee. "A-STAR: A mobile ad hoc routing strategy for metropolis vehicular communications." In *NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, pp. 989–999. Springer, 2004.

[SPR05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. "Spray and wait: an efficient routing scheme for intermittently connected mobile networks." In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pp. 252–259. ACM, 2005.

[SRF14] Vasco NGJ Soares, Joel JPC Rodrigues, and Farid Farahmand. "GeoSpray: A geographic routing protocol for vehicular delay-tolerant networks." *Information Fusion*, **15**:102–113, 2014.

[SRJ03] Rahul C Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. "Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks." *Ad Hoc Networks*, **1**(2):215–233, 2003.

[SZT04] Felix Schill, Uwe R Zimmer, and Jochen Trumpf. "Visible spectrum optical communication and distance sensing for underwater applications." In *Proceedings of ACRA*, volume 2004, pp. 1–8, 2004.

[Tel17] TeleGeography. "Submarine Cable Map." `http://www.submarinecablemap.com`, 2017.

[THH03] Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Yueng Hsieh. "Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks." *Computer Networks*, **43**(3):317–337, 2003.

[TR08] Leonard T Tracy and Sumit Roy. "A reservation MAC protocol for ad-hoc underwater acoustic sensor networks." In *Proceedings of the third ACM international workshop on Underwater Networks*, pp. 95–98. ACM, 2008.

[VB00] Amin Vahdat, David Becker, et al. "Epidemic routing for partially connected ad hoc networks." Technical report, Technical Report CS-200006, Duke University, 2000.

[VKR05] Iuliu Vasilescu, Keith Kotay, Daniela Rus, Matthew Dunbabin, and Peter Corke. "Data collection, storage, and retrieval with an underwater sensor network." In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 154–165. ACM, 2005.

[VKT05] S. Vasudevan, J. Kurose, and D. Towsley. "On neighbor discovery in wireless networks with directional antennas." In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pp. 2502–2512 vol. 4, March 2005.

[VTG09] Sudarshan Vasudevan, Donald Towsley, Dennis Goeckel, and Ramin Khalili. "Neighbor Discovery in Wireless Networks and the Coupon Collector's Problem." In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, pp. 181–192, New York, NY, USA, 2009. ACM.

[WBM07] N. Wisitpongphan, Fan Bai, P. Mudalige, and O.K. Tonguz. "On the Routing Problem in Disconnected Vehicular Ad-hoc Networks." In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 2291–2295, May 2007.

[XCL06] Peng Xie, Jun-Hong Cui, and Li Lao. "VBF: vector-based forwarding protocol for underwater sensor networks." In *Networking*, volume 3976, pp. 1216–1221. Springer, 2006.

[YMF06] Saleh Yousefi, Mahmoud Siadat Mousavi, and Mahmood Fathy. "Vehicular ad hoc networks (VANETs): challenges and perspectives." In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pp. 761–766. IEEE, 2006.

[YSC08] Hai Yan, Zhijie Shi, and Jun-Hong Cui. "DBR: depth-based routing for underwater sensor networks." *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pp. 72–86, 2008.

[Zha06] Zhensheng Zhang. "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges." *Communications Surveys & Tutorials, IEEE*, **8**(1):24–37, 2006.

[ZHL12] Desheng Zhang, Tian He, Yunhuai Liu, Yu Gu, Fan Ye, Raghu K. Ganti, and Hui Lei. "Acc: Generic On-demand Accelerations for Neighbor Discovery in Mobile Applications." In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pp. 169–182, New York, NY, USA, 2012. ACM.

[ZHS03] Rong Zheng, Jennifer C. Hou, and Lui Sha. "Asynchronous Wakeup for Ad Hoc Networks." In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, pp. 35–45, New York, NY, USA, 2003. ACM.

[ZHY12] Desheng Zhang, Tian He, Fan Ye, R.K. Ganti, and Hui Lei. "EQS: Neighbor Discovery and Rendezvous Maintenance with Extended Quorum System for Mobile Sensing Applications." In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pp. 72–81, June 2012.

[ZL08] Zhensheng Zhang and Bo Li. "Neighbor discovery in mobile ad hoc self-configuring networks with directional antennas: algorithms and comparisons." *Wireless Communications, IEEE Transactions on*, **7**(5):1540–1549, May 2008.

[ZVC11] Wei Zeng, Sudarshan Vasudevan, Xian Chen, Bing Wang, Alexander Russell, and Wei Wei. "Neighbor Discovery in Wireless Networks with Multipacket Reception." In *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc*

Networking and Computing, MobiHoc '11, pp. 3:1–3:10, New York, NY, USA, 2011. ACM.