Implementation and Evaluation of Multihop ARQ for Reliable Communications in Underwater Acoustic Networks

Alvin Valera   Pius W.Q. Lee   Hwee-Pink Tan   Huiguang Liang   Winston K.G. Seah
Institute for Infocomm Research (I^{2}R)
Agency For Science, Technology And Research (A*STAR)
1 Fusionopolis Way, #21-01 Connexis
Singapore 138632
Email: {acvalera,wqlee,hptan,hliang,winston}@i2r.a-star.edu.sg

***Abstract*—Underwater acoustic networking is an emerging technology platform for oceanographic data collection, pollution monitoring, offshore exploration and tactical surveillance applications. Design of reliable and efficient communications protocols is challenging due to the unique characteristics of underwater acoustic channels. In this paper, we present a modular and lightweight implementation of an opportunistic multihop automatic repeat request (ARQ) scheme in a real system. We evaluate the performance of the opportunistic ARQ using inexpensive underwater acoustic modems in a shallow underwater environment.**

I. Introduction

Underwater acoustic networking is an emerging technology platform for oceanographic data collection, pollution monitoring, offshore exploration and tactical surveillance applications. Underwater acoustic channels are characterized by severe bandwidth limitations, severe channel impairments due to multipath and fading, high and extremely variable propagation delay, high bit error rates and frequent connectivity interruptions due to shadow zones [1]–[4]. These characteristics pose difficult challenges in the design of reliable and efficient communications protocols.

Several automatic repeat request (ARQ) schemes have been proposed to improve the reliability of underwater acoustic communications. Recently, Tan et al [5] proposed an opportunistic ARQ scheme that uses knowledge of per-hop bit error rate (BER) to decide on whether to perform *implicit* or *explicit* acknowledgment. The scheme has been analytically shown to perform well compared with non-opportunistic ARQ schemes [5]. In this paper, we present an implementation the opportunistic ARQ in a real system. To enable the implementation of the opportunistic ARQ, we design and implement a network stack and software architecture that is suitable for challenged underwater acoustic networks. We validate the implementation by means of an experiment using inexpensive underwater acoustic modems.

The rest of the paper is organized as follows: Section II presents related work on reliable communications in underwater acoustic networks. Section III presents implementation design details of an underwater network stack, software architecture, and the opportunistic ARQ. Experimental setup and results from performance evaluation in a shallow underwater environment are presented in Section IV. Section V concludes the paper with a summary of the important findings and future work.

II. Related Work

To set the scene for this paper, we begin with a review of related work in the area of reliable communications in multi-hop underwater acoustic networks.

Several studies have explored the benefits of multi-hop underwater networks. Sozer [1] showed that the use of relays as opposed to direct communications minimized energy consumption. A study by Carbonelli and Mitra [6] showed that multi-hopping significantly improved signal detection in underwater acoustic networks. Studies by Stojanovic [7] and Zhang et al [8] further showed that multi-hopping can be used to improve achievable data rates. Given these positive results, it is therefore important to support multi-hop communications in underwater acoustic networks. This paper addresses this need as it presents a real-world implementation of an underwater network stack to supports multi-hop communications.

Compared with single-hop ARQ, ARQ mechanisms for multi-hop communications is much less studied. Wiemann et al [9] employed multi-hop end-to-end ARQ (e2e-ARQ) to handle node mobility in terrestrial wireless networks. Lott [10] proposed another multi-hop ARQ mechanism (M-ARQ) for terrestrial wireless networks by coupling a per-hop ARQ protocol with an e2e-ARQ. As these schemes are not suitable for underwater acoustic networks, Tan et al [5] proposed an opportunistic multi-hop ARQ that dynamically selects between implicit and explicit acknowledgment depending on the underwater channel conditions. This paper presents an real-world implementation of this scheme.

III. Implementation Design

A. Opportunistic Multihop ARQ

Before presenting the implementation design details, we briefly discuss the salient features of the opportunistic ARQ proposed by Tan et al [5].

In basic stop-and-wait (SAW) ARQ, a sender waits for an acknowledgment (ACK) from the receiver after it completes its transmission of a data packet. If no ACK is received within the timeout period, the sender re-transmits the data packet. The proposed multihop ARQ modifies the basic SAW operation with the use of *implicit* and *explicit ACKs*.

Implicit Acknowledgment: Due to the isotropic nature of the underwater acoustic channel, a data packet forwarded by node $j-1$ to node j might be overheard by $j-1$ once j forwards the packet to the next hop node $j+1$. This overhearing of the data packet can serve as an implicit ACK from j to $j-1$.

Explicit Acknowledgment: This type of acknowledgment is piggy-backed in data packets. For example, when node j forwards a data packet to node $j+1$ (the data packet need not be necessarily from $j-1$), it piggy-backs an ACK (or several ACKs) for $j-1$.

The multihop ARQ uses knowledge of per-hop bit error rates (BER) for determining whether to use implicit or explicit acknowledgment. The scheme proposes two types of thresholds: (i) latency threshold (p_l^*); and (ii) energy-efficiency threshold (p_e^*). When the per-hop BER is greater than p_e^*, then explicit ACK is used. When the per-hop BER is less than p_l^*, implicit ACK is used. If the per-hop BER lies between p_l^* and p_e^*, either explicit or implicit ACK is used, depending on the objective or priority of the system. If the objective is to improve latency, explicit ACK should be used; otherwise, implicit ACK should be used for better energy-efficiency.

B. Protocol States

Figure 1 shows the protocol state transition diagram. In diagram, the decision process on whether to use implicit or explicit ACK when the per-hop BER is between p_l^* and p_e^* is not shown. In such cases, the state may either transition from IDLE to IMP or EXP, depending on the system objective. The important protocol transitions are as follows:

- When node j receives a data packet and it is the sink, the protocol transitions from IDLE to ACK-SEND state. An ACK is immediately sent to the packet sender and the protocol immediately transitions to IDLE.
- When node j receives a data packet, it is not the sink or source and the per-hop BER is less than the threshold p_l^*, then the protocol transitions from IDLE to IMP state. The protocol does not add anything onto the received data packet but sets the ACK-WAIT timeout period to IMP_TIMEOUT. The protocol immediately transitions to the DATA-SEND state where the packet is relayed to the next hop.
- When node j receives a data packet, it is not the sink or source and the per-hop BER is greater than the threshold p_e^*, the protocol transitions from IDLE to EXP state. An ACK is piggy-backed onto the received data packet and sets the ACK-WAIT timeout period to EXP_TIMEOUT. The protocol immediately transitions to the DATA-SEND state where the packet is relayed to the next hop.
- At the ACK-WAIT state, node j is essentially waiting for an acknowledgment for a packet it just transmitted. If acknowledgment is received, the protocol transitions to IDLE. If no acknowledgment is received within the timeout period, the protocol transitions to DATA-SEND again provided that the number of retries is less than RETRY_LIMIT. If no acknowledgment is received within the timeout period and the number of retries is greater than or equal to RETRY_LIMIT, then the protocol transitions to the QUEUE-CLEANUP state.

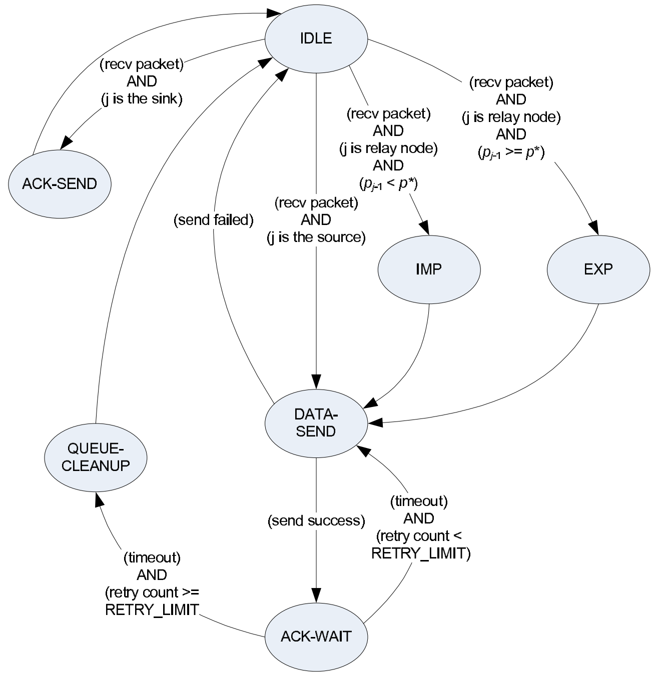
IDLE
(recv packet) AND (j is the sink)
(recv packet) AND (j is relay node) AND (p_{j-1} < p*)
(recv packet) AND (j is relay node) AND (p_{j-1} >= p*)
ACK-SEND
(send failed)
(recv packet) AND (j is the source)
IMP
EXP
DATA-SEND
QUEUE-CLEANUP
(send success)
(timeout) AND (retry count < RETRY_LIMIT)
(timeout) AND (retry count >= RETRY_LIMIT
ACK-WAIT

Fig. 1. Protocol state transition diagram

C. Underwater Network Stack and Software Architecture

While it is attractive to design an entirely new protocol stack for underwater networks, our strategy is to adapt the widely-accepted and widely-tested TCP/IP protocol stack. However, due to severely low capacity of underwater acoustic channels, we must minimize the overhead due to packet headers. To achieve this, we propose the inclusion of a component that will perform compression (and decompression) of upper layer packet headers.

Our proposed software architecture is shown in Figure 2. The component "Network Header Compression Adapter" is responsible for compressing (and decompressing) the network layer packet headers. For outgoing packets, it will "decapsulate" packets received from the network layer by stripping off the network header and replacing it with a lightweight underwater (UW) header. For incoming packets, it will strip off

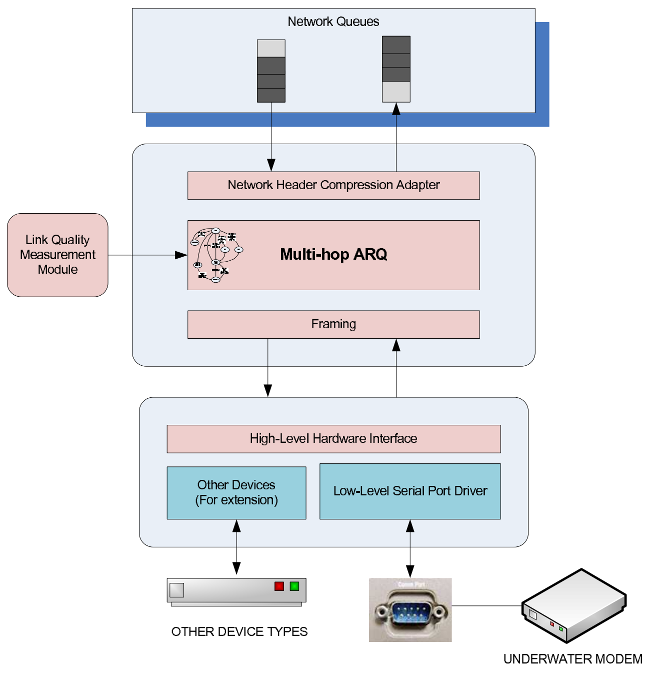
Network Queues
Network Header Compression Adapter
Link Quality Measurement Module
Multi-hop ARQ
Framing
High-Level Hardware Interface
Other Devices (For extension)
Low-Level Serial Port Driver
OTHER DEVICE TYPES
UNDERWATER MODEM

Fig. 2. Software implementation architecture

the UW header and construct the appropriate network header.

The "Multihop ARQ" component implements the opportunistic ARQ described in Section III-A. It uses link quality measurements obtained from a separate component called "Link Quality Measurement Module". Due to the modularity of the architecture, one can easily implement other ARQ schemes and integrate them into this architecture.

Addressing: To minimize the overhead due to addressing, node addresses will be 8 bits wide. The address 0xFF will be reserved for broadcast address. This results to an address range from 0x00 to 0xFE, or 254 addressable hosts. The network and data link layers will use the same address. Hence, no address resolution protocol will be required. To accomplish this addressing scheme without changing the IP layer, the component "Network Header Compression Adapter" will provide IP address spoofing.

Frame Types: The protocol implementation defines three types of frames: (i) *plain data frame* – a frame that contains data of up to 60 bytes; (ii) *data frame with piggybacked acknowledgment* – a frame that contains both data of up to 58 bytes and an acknowledgement for one data frame; and (iii) *acknowledgement frame* – a frame that contains acknowledgement for one data frame. Figure 3 shows the fields of the three frame formats. The meaning/purpose of the fields are as follows: (i) the 2-bit field ”V” refers to the protocol version; (ii) the 3-bit field ”R” is reserved for future use; (iii) the 1-byte sequence number is a sender-generated number to uniquely identify frames that the sender have sent out; (iv) the 1-byte sender address field contains the address of the sending node; (v) the 1-byte receiver field contains the address of the receiving node; (vi) the 1-byte data length indicates the length of the data field. For the 2nd and 3rd frame types, the fields

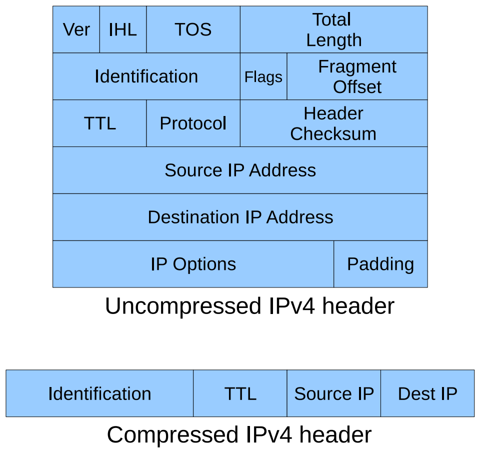

Fig. 4. Compressed and uncompressed IPv4 header.

Sequence Number of Data to ACK and Previous Hop of Data to ACK, which are both 1 byte long, essentially refer to the identity of the data frame being acknowledged.

D. Header Compression

One of the most difficult challenges in underwater acoustic networking is its severely low channel capacity [1]–[4]. Thus, protocols must ensure that packet overheads due to headers are minimized. The proposed architecture includes a header compression component that is designed to operate with an IPv4-based network layer. The advantage of this approach is that we can tap the rich array of applications written for IP-based networks.

Figure 4 shows the compressed and uncompressed IPv4 header. In this compression scheme, only those fields that cannot be calculated or mocked are included. The compressed header version is only five bytes long. The IP address only uses one byte, that is, the forth octet in the four-octet IPv4 address. Obviously, the compression scheme does not allow IP options. Furthermore, it does not include the protocol field. Hence, applications that will use this stack must directly inject packets above the IP layer. In Linux, this can be easily accomplished with the use of raw IP sockets.

E. Implementation Details

We implemented the underwater stack and all the relevant components in the Linux operating system (using kernel version 2.6). In this study, we used the 480 bps Aquacomm acoustic modems from DSPComm Pty Ltd as the hardware platform. Table I shows the detailed specifications of the modem while Figure 5 shows the modem's underwater casing and transducer. Each of the components (*i.e.*, multihop ARQ, framing component, network header compression adapter, and low level device driver) was written as a stand-alone loadable kernel module. The underwater stack provides a programming interface and hooks to easily implement and replace the individual components. For example, if we were to use acoustic modems from other vendors, we would only need to replace the low level device driver component.

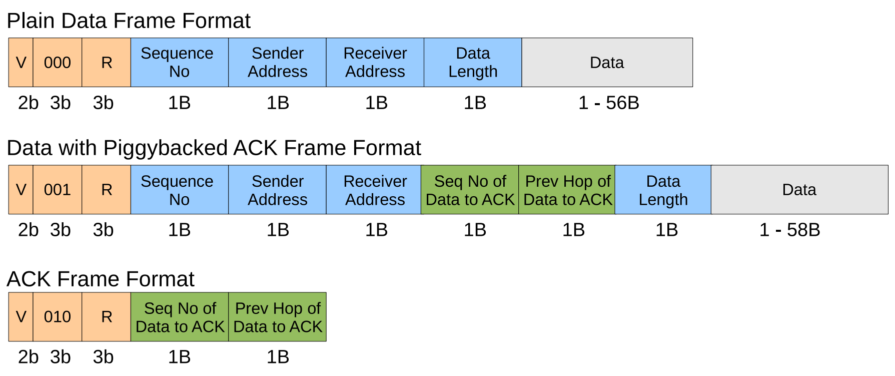

Fig. 3. Frame formats used by the proposed underwater network stack.

TABLE I
AQUACOMM ACOUSTIC MODEM SPECIFICATIONS

Parameter	Value
Data rate	480 bits per second
Bandwidth	Broadband operation 16 KHz to 30 KHz
Range	Up to 3 km
Modulation	Direct sequence spread spectrum / OFDM
Error detection	CRC-16 error detection
Host interface	Serial port (RS-232)

Fig. 5. Aquacomm acoustic modem casing and transducer (red)

Implementing the framing component and network header compression adapter was straight-forward and was easily accomplished. Implementing the low level device driver requires knowledge of the Aquacomm acoustic modem programming interface. In this case, the interface is in the form of RS-232 AT commands. We wrote a software development kit to encapsulate the commands supported by the modem and expose it to higher layers through the “High Level Hardware Interface” component. The SDK takes advantage of the mature TTY support in Linux to communicate with the modem through the serial port.

We encountered several challenges while implementing the multihop ARQ using the Aquacomm acoustic modems. Recall that the ARQ requires per-hop bit error rate (BER) threshold to determine which acknowledgement scheme is to be applied. As this particular model does not provide BER information, we slightly modified the threshold to use per-packet receive signal strength (RSS) instead as it is provided by the modem. We later discovered that the RSS values provided by the modem have very low correlation with the packet delivery ratio. Hence, RSS could not be used as an indicator of the link quality.

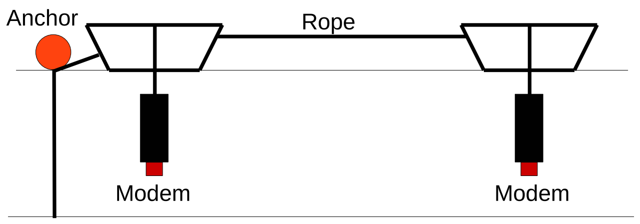

Fig. 6. Experimental setup for single-hop tests. The two boats are tied using a nylon rope. The modems are submerged at around 1 meters from the water surface.

IV. EVALUATION

A. Experimental Setup

We used the relatively inexpensive 480 bps Aquacomm acoustic modems (see Table I for the detailed specifications and Figure 5 for an image of the underwater casing and transducer) to evaluate the real-world performance of the opportunistic ARQ scheme. In this performance evaluation, we focus our attention first on the single-hop performance. We deployed two acoustic modems in an area that is approximately 3 square kilometers and with depths ranging from 4 to 9 meters. We fixed the inter-node distance to 100 m by tying the boats together as shown in Figure 6.

We used the default modem settings for all the experiments. We used *nemesis* [11], an IP packet injection program, to generate constant bit rate traffic at 1 packet every 10 seconds. We varied the packet size from 16 bytes to 32 bytes. Although the modem supports up to 64 bytes of packet size, we did not use it as the performance using this parameter was extremely inconsistent due to severe channel conditions. For the opportunistic ARQ, we set the ACK timeout to three seconds and limited the data packet retransmissions to three. If no ACK is received after four transmissions (one transmission plus three retransmissions), the data packet is dropped.

B. Results

We now present our results obtained from the experiments. Figures 7 and 8 show the packet delivery ratio (PDR) and end-to-end delay, respectively, of the opportunistic ARQ scheme compared with the broadcast (no ARQ or reliability). The

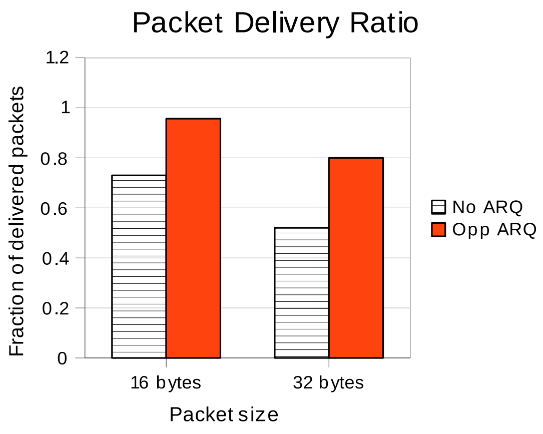

Fig. 7. Packet delivery ratio

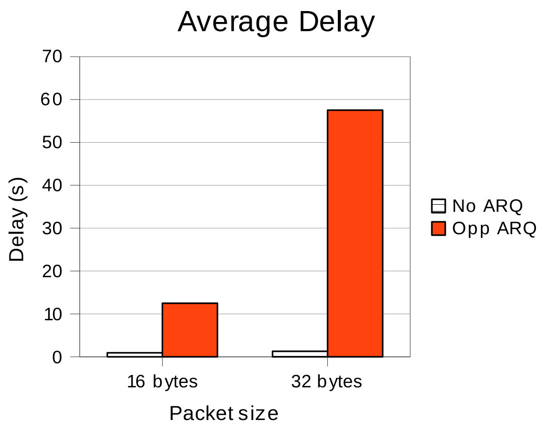

Fig. 8. End-to-end delay

PDR results show that the opportunistic ARQ scheme provides significant improvement on the reliability of the channel. Without ARQ, PDR is at most 73% and 52% for 16 bytes and 32 bytes, respectively. With the opportunistic ARQ, PDR increase by more than 20% for 16 bytes and almost 30% for 32 bytes.

However, the use of ARQ to provide improvement in data delivery ratio has a negative impact on the delay. In both 16 bytes and 32 bytes, the delay when no ARQ is employed is very low. At 16 bytes, the delay is around one second while at 32 bytes, the delay is around 1.3 seconds. When the opportunistic ARQ is employed, the delay shows a dramatic increase. At 16 bytes, the delay increases to 12.5 seconds while at 32 bytes, the delay increases to 58 seconds. These results are not surprising given the fact that the opportunistic ARQ utilizes packet retransmissions.

We elaborate on why the delay can significantly increase when the opportunistic ARQ is deployed. Recall that in these experiments, we limited the retransmissions to three for the opportunistic ARQ. Intuitively, the maximum delay using the opportunistic ARQ is approximately four times the timeout when no ARQ is deployed (around 12 seconds). Note however that the traffic is being generated at a faster pace, namely, one packet every ten seconds. What happens is that when the network layer receives a packet and the ARQ is still in the process of retransmissions (i.e., not in the IDLE state), the packet is simply queued at the network layer. Once ARQ completes the process and returns to the IDLE state, it processes the next packet from the network layer queue. With this kind of processing, it is obvious that packets may accumulate at the network layer queue. Thus, in extremely poor channel conditions, the packets can experience high delay due to both queuing delay and retransmissions, with the former contributing a significant fraction in the 32 byte case.

Packet size shows a significant effect on the PDR and delay. Without ARQ, the higher packet size of 32 bytes has 20% less PDR. With ARQ, the higher packet size has 15% less PDR. These result is expected since a bigger packet has a higher probability of being corrupted and therefore not received at the network layer. In terms of delay, the results are consistent, with the bigger packet size having larger delays than the smaller packet size.

V. Conclusion and Future Work

Underwater acoustic networks is an emerging technology platform for oceanographic data collection, pollution monitoring, offshore exploration and tactical surveillance applications. Design of reliable and efficient communications protocols is challenging due to the unique characteristics of underwater acoustic channels. In this paper, we presented a modular and lightweight implementation of an opportunistic multihop ARQ in a real system. We also designed and implemented a lightweight, flexible and extensible network stack that is suitable for challenged underwater acoustic networks. We evaluated the performance of the ARQ using inexpensive underwater acoustic modems in a shallow environment and demonstrated that the opportunistic ARQ can provide significant improvement in terms of data delivery ratio. The disadvantage is an increase in end-to-end delay due to queuing and retransmissions.

The single-hop results presented in this paper are important to verify the stability and functional correctness of the protocol implementation. We have sufficiently demonstrated that the implemented ARQ is indeed stable and functionally correct. As the implemented opportunistic ARQ scheme is targeted for multihop networks, we are currently conducting experiments in an underwater multihop network.

Acknowledgment

The authors would like to thank Mr. Lim Kok Tong and the staff of Public Utilities Board (PUB) Pandan Reservoir Office for allowing us to use their facilities and also providing us with logistical support during our field experiments.

References

[1] E. Sozer, M. Stojanovic, and J. Proakis, “Underwater acoustic networks,” *IEEE Journal of Oceanic Engineering*, vol. 25, no. 1, pp. 72–83, Jan. 2000.

[2] I. F. Akyildiz, D. Pompili, and T. Melodia, “Underwater acoustic sensor networks: research challenges,” *Ad Hoc Networks (Elsevier)*, vol. 3, pp. 257–279, 2005.

[3] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li, “Research challenges and applications for underwater sensor networking,” in *Proc. IEEE WCNC 2006*. Las Vegas, Nevada, USA: IEEE, April 2006, pp. 228–235.

[4] J. Preisig, “Acoustic propagation considerations for underwater acoustic communications network development,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 11, no. 4, pp. 2–10, 2007.

[5] H.-P. Tan, W. K. Seah, and L. Doyle, “A multi-hop arq protocol for underwater acoustic networks,” in *Proc. OCEANS 2007 - Europe*, Jun. 2007.

[6] C. Carbonelli and U. Mitra, “Cooperative multihop communication for underwater acoustic networks,” in *Proc. ACM WUWNet 2006*. New York, NY, USA: ACM, 2006, pp. 97–100.

[7] M. Stojanovic, “Capacity of a relay acoustic channel,” in *Proc. OCEANS 2007*, 29 Sept–4 Oct 2007, pp. 1–7.

[8] W. Zhang, M. Stojanovic, and U. Mitra, “Analysis of a simple multihop underwater acoustic network,” in *Proc. ACM WuWNeT 2008*. New York, NY, USA: ACM, 2008, pp. 3–10.

[9] H. Wiemann, M. Meyer, R. Ludwig, and P. Chang, “A novel multi-hop arq concept,” in *Proc. IEEE VTC 2005-Spring*, vol. 5, May-1 June 2005, pp. 3097–3101.

[10] M. Lott, “Arq for multi-hop networks,” in *Proc. IEEE VTC-2005-Fall*, vol. 3, Sep. 2005, pp. 1708–1712.

[11] J. Nathan, “Nemesis – packet injection tool suite,” http://nemesis.sourceforge.net.