Ad Hoc Networks 34 (2015) 252–264

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Software-defined underwater acoustic networking platform and its applications

Dustin Torres [a], Jonathan Friedman [a], Thomas Schmid [b], Mani B. Srivastava [a], Youngtae Noh [c,*], Mario Gerla [d]

[a] Electrical Engineering Department, University of California at Los Angeles (UCLA), United States
[b] Electrical and Computer Engineering Department at the University of Utah, United States
[c] Department of Computer Science and Information Engineering, Inha University, South Korea
[d] Computer Science, University of California at Los Angeles (UCLA), United States

ARTICLE INFO

Article history:
Received 2 March 2014
Received in revised form 19 December 2014
Accepted 21 January 2015
Available online 31 January 2015

Keywords:
Underwater acoustic networking
Software defined radio
Time synchronization
Channel allocation protocols

ABSTRACT

As underwater communications adopt acoustics as the primary modality, we are confronting several unique challenges such as highly limited bandwidth, severe fading, and long propagation delay. To cope with these, many MAC protocols and PHY layer techniques have been proposed. In this paper, we present a research platform that allows developers to easily implement and compare their protocols in an underwater network and configure them at runtime. We have built our platform using widely supported software that has been successfully used in terrestrial radio and network development. The flexibility of development tools such as software defined radio, TinyOS, and Linux have provided the ability for rapid growth in the community. Our platform adapts some of these tools to work well with the underwater environment while maintaining flexibility, ultimately providing an end-to-end networking approach for underwater acoustic development. To show its applicability, we further implement and evaluate channel allocation and time synchronization protocols on our platform.

1. Introduction

The underwater medium presents many challenges for digital communication. There is limited available bandwidth and high bit error rates caused by multipath, fading, and long propagation delay. The speed of sound is five orders of magnitude less than that of terrestrial radio, which can make it hard for underwater networks to synchronize, exchange data, update routes, and communicate efficiently.

Due to the severity of multipath underwater, a receiving node might be at a point where there is little energy in the received signal making it hard to receive without errors [1]. These spots of destructive multipath interference vary with time due to the movement of water, and make it quite hard to even have a static network topology. Along with multipath, other unfavorable characteristics such as ambient noise, bubbles, surface scattering, and slow propagation speed make developing underwater networks a difficult task [1].

Furthermore, the ocean varies significantly both temporally and spatially, which makes it challenging to create a model of a "typical" underwater acoustic channel [1]. Since there is no "typical" model, there is no single network architecture that works best in all situations. Therefore, depending on the current characteristics of the channel, there is a variable amount of bandwidth, various noise

* Corresponding author. Tel.: +82-32-860-7445.
E-mail address: ytnoh@inha.ac.kr (Y. Noh).

http://dx.doi.org/10.1016/j.adhoc.2015.01.010
1570-8705/

sources in different frequency bands, varying inter-symbol interference (ISI) depending on the water depth, and many other characteristics under consideration. Even if the channel noise is known, attenuation still depends on both distance and frequency, along with space and time varying multipath [2].

We present a software defined Underwater Acoustic Networking plaTform (UANT) to aid development of underwater acoustic networks. A software defined system can help to address the constantly changing underwater acoustic channel by way of reconfigurability. A flexible approach allows for system parameters at all layers to be easily modified without the need for specialized hardware. UANT uses GNU Radio, a software defined radio framework, to achieve configurability at the physical layer. TinyOS has been widely adopted for the use on sensor network platforms and provides a full network stack. We adapted these widely supported tools that have proven effective prototyping, development, and implementation for terrestrial networks to be used in UANT.

Since the characteristics of the underwater acoustic channel cannot be properly modeled with a static configuration, it is important to be able to change the properties of an acoustic modem at run time. UANT has the flexibility of software defined radios and the advantages of the network layers of TinyOS and Linux, ultimately providing and end-to-end network for easy underwater development from the physical to application layer. This paper significantly enhances our earlier work [3] as follows:

- We add Applications and protocol implementation on UANT section to show UANT's applicability and discuss channel allocation protocol's detailed description (Section 4).
- We implement and evaluate channel allocation protocol and time synchronization protocol (THSL) on our proposed underwater acoustic networking platform (Section 6.2 and 6.3 respectively).

2. Background

Software defined techniques have been of interest in recent years not only for terrestrial radios but also acoustics and ultrasonics [4]. Jones et al. described some of the benefits to the use of software define radio for underwater use [5]. They talked about the possibility to improve underwater acoustic performance by using methods that have worked well with terrestrial radios such as Cognitive radio via software defined techniques.

Sözer and Stojanovic developed rModem [6] to achieve a similar goal of having a configurable acoustic modem. However, the rModem is a standalone board that uses an FPGA and DSP. Furthermore, rModem focuses on lower layers and does not provide an end-to-end networking environment, making it hard to study the impact on actual applications. We hope to supplement platforms such as the rModem by aiding research in MAC and PHY layer protocols that can ultimately be implemented on standalone nodes to create an underwater network.

2.1. GNU Radio and USRP

Software defined radio has received a lot of attention most notably in the research community. The ability to use software to modulate and manipulate the received and transmitted signals allows for rapid development without the need or cost of specialized hardware. GNU Radio [7], one of the most popular SDR frameworks, is comprised of a flow graph and signal processing blocks. The signal processing blocks are written in C++ and act as the "heavy lifters" whereas the flow graph is setup in Python in order to move data from one block to the next. In this way many modulation schemes can be created using standard C++ blocks (already included in GNU Radio) and connecting them together in a flow graph. There is a large community of users who have contributed to this open source project, both signal processing blocks as well as various applications. The many contributions have led to a large library of modulation schemes including GMSK, PSK, QAM, CPM, OFDM, and more.

We use GNU Radio along with the Universal Software Radio Peripheral version 1 (USRP1) to achieve underwater acoustic communication. The USRP created by Ettus Research [8], is a radio front-end that is commonly used with GNU Radio. Although the option of using a sound card provides a low cost solution, the USRP offers a wider frequency range as well as more dedicated hardware. The USRP has a total of 4 ADCs and 4 DACs allowing for up to 16 MHz of bandwidth each way, which is proficient for the underwater acoustic channel. We have modified an example application used for digital communications bundled with GNU Radio to use the USRP as a NIC via the TUN/TAP interface. This allows for wireless communications between two computers using software defined radio that can run networking applications over TCP.

2.2. TinyOS and TOSSIM

TinyOS is a widely used sensor network operating system created at University of California, Berkeley and meant for sensor nodes requiring concurrency and flexibility while being limited to resource constraints [9]. TinyOS is implemented in the NesC language, which supports the concurrency model needed for sensor networks. TinyOS is widely used both in commercial applications as well as in academics for research purposes. There is a large user base who constantly contribute to the open source project, which allows UANT to always benefit from the newest protocols. For instance, Washington University contributed a MAC Layer Architecture [10] to provide the MAC layer with many commonly needed functions for MAC protocols.

Generally TinyOS is compiled for a sensor network platform, with a network stack in accordance with the specific radio being used. However to gain the benefits of using TinyOS in UANT we have chosen to use TOSSIM [11], which simulates sensor network nodes on a PC. TOSSIM was created in order to support compiling TinyOS component graphs into a simulation for a PC. It utilizes a discrete event queue, reimplements some of the sensor nodes hardware, models the radio and ADC of a mote, and most importantly for UANT, uses communication services for external

programs to communicate with the running simulation over TCP sockets [11]. TOSSIM fully supports the TinyOS toolchain making it effortless to transition from simulation to real network, or vice versa. TOSSIM executes in a similar fashion to a real mote, with each simulation having a notion of a virtual clock running at 4 MHz. TOSSIM uses interrupts similar to that of a mote, however when an event fires, the simulation calls an interrupt handler in a hardware abstraction component. In UANT, TOSSIM is not used as a simulation framework but rather the real time running nodes, this is explained in greater detail in the next section.

3. System architecture

UANT has been created as a platform to allow testing of new protocols and modulation schemes on a fully functional underwater network. We used open source software (GNU Radio and TinyOS) that is widely supported and has been in use for many years. TinyOS applications are able to be incorporated into UANT simply by changing the configuration file to match the needs of the specific application. The running application can be easily monitored in Linux either by monitoring the TOSSIM simulation through currently supported techniques such as debug statements, or packets being forwarded over a TCP socket where the raw bytes of transmitted or received packets can be viewed, or both. Along with running TinyOS applications, it is possible to assign an IP addresses to the TOSSIM node in order to inject TCP/IP packets and use a wide variety of Linux applications to send data between nodes. UANT uses TOSSIM to run TinyOS applications and components on a PC in real-time fashion rather than being used as a simulator. This is similar to EmTos [12] where a wrapper around a TinyOS application is used to run on an EmStar node. It is important to note that each node in UANT is running in a different TOSSIM simulation. This allows the ability for every node to run a unique and different application, which is not possible in a single TOSSIM simulation.

Although the non-negligible latency introduced from samples traveling from the hardware frontend to software can make it difficult to develop a MAC layer for SDR, the long propagation delay of the underwater channel makes it possible to ignore this latency. Nychis et al. [13] used "split-functionality" in order to take advantage of the minimal latency on the USRP, yet still maintaining the control of the data flow in the host CPU. Yang et al. [14] recently demonstrated the benefits of parallelization in embedded realizations on SDR platforms in order to achieve a high degree of flexibility while satisfying the hard real-time constraints. As another branch of recent work, Truong et al. analyzed the sources of the latency and quantified them by using both analytical and experimental methods [15,16].

However, with the slow propagation delay of the underwater acoustic medium, the added bus latency between the USRP and the MAC layer implemented on the host machine can be considered negligible. The speed of sound underwater is 1500 m/s, so for two nodes that are 1.5 km apart the propagation delay is approximately one second. In [17], Schmid et al. measured SDR latency and characterized the round trip time to and from the USRP to be between 3 ms and 26.5 ms. The 26.5 ms measurement included host processing of the 802.15.4 protocol, while 3 ms is the theoretical latency to and from the host assuming no computation is done on the host machine. Although in terrestrial radio this latency makes some MAC protocols such as TDMA hard to implement, for underwater acoustics this is.3% of the propagation delay for nodes 1.5 km apart.

3.1. Architecture

UANT provides reconfigurability at the physical layer, the MAC layer, and the application layer. With control over these layers, we aim to provide a way to implement and compare proposed underwater MAC protocols and PHY layer techniques in the presence of a network running applications from Linux or TinyOS. Although UANT must be run on a PC, we tried to keep in mind that the end goal for much of the current underwater acoustic communication research is geared toward underwater sensor networks (USN) and underwater autonomous vehicles (UAV).

3.1.1. Overview

The flow of data through UANT can be seen in Fig. 1. To receive packets, an acoustic signal containing data is first heard by the transducer and passed through to the USRP, which will sample and down convert the data. The data continues over USB to software where GNU Radio first passes the data through an FFT filter. The streaming samples now represent the complex baseband signal in the frequency domain needed for demodulation. Depending on the modulation scheme that is being used the data will be processed accordingly. The output of the demodulation blocks is a string of bits which will be fed to a message queue in GNU Radio.

GNU Radio is started with a Python script, which establishes the flow graph of C++ blocks for the data to travel through. In UANT, the same Python script that is used to start the flow graph is also used to communicate with TinyOS. Data in the message queue that has been demodulated and ready to enter TinyOS must first be packed into a TinyOS serial packet with a one byte modification. If the incoming packet has a broadcast address it is changed to be the address of the currently running node ID on the host machine, which is necessary due to the limitations of the SerialActiveMessages of TinyOS. The packet is then sent over a socket to TOSSIM where it first enters the MAC layer. Depending on what type of message has been received, it will be properly routed through the TOSSIM simulation, and possibly to Linux via a virtual Ethernet card if required.

3.1.2. UANT: physical layer

GNU Radio has been widely adopted by the SDR community. There has been much advancement in terrestrial radios with the introduction of software defined radio. We have chosen to use GNU Radio for UANT in order to try and bring its success for fast prototyping and development to the underwater environment. GNU Radio offers many signal processing blocks that are required for basic and advanced modulation techniques. There have already

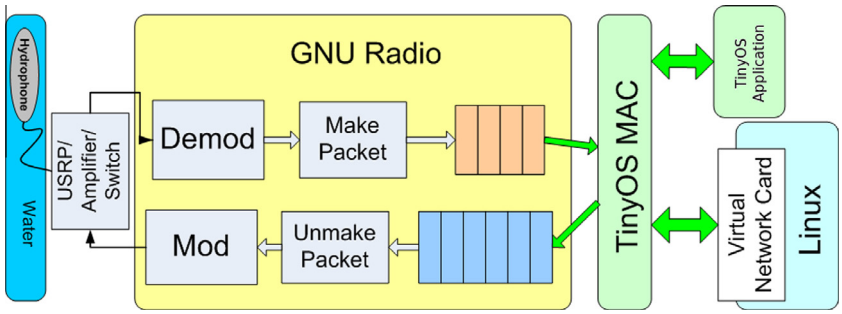


Fig. 1. UANT system architecture.

been many modulation schemes implemented in GNU Radio using these standard signal processing blocks. We have incorporated much of the prior work that has been done with GNU Radio including these modulation schemes as well as some of the example applications. Specifically we modified the tunnel example file that allows the USRP boards to be assigned IP addresses and use Linux applications. Modifications that were made include removing the simplified version of a single CSMA MAC, allowed for slower bit rates that could be desired in a harsh underwater channel, and changed the traffic source and sink from Linux to TinyOS. Keeping the modifications of the GNU Radio example to a minimum allows UANT to use any future expansion to the digital communication examples to be effortlessly incorporated into our system.

Although currently GNU Radio is too computationally expensive for embedded implementations, we preferred the ability for rapid prototyping and test of PHY and MAC even if it means that our host must be more powerful than some lower end platforms such as motes. If development of network protocols leads to increased performance on UANT it can ultimately be ported to the platform that is used in the USN or UAV, or directly transferred if TinyOS or Linux is used by the system. In addition, we later show that even with a more capable host UANT is still field-deployable and has been tested in different real world environments. Historically much of the development for GNU Radio has not considered energy savings which is essential for USN; however, there has been some recent work in this direction. Nychis et al. presented a power saving method for fast pattern detection in order to only pass the received packet over USB if it is in fact intended for the listening node [13].

3.1.3. UANT: MAC layer

One aspect of GNU Radio that makes it inherently hard for compatibility with higher layer protocols is the use of flow graphs. We have chosen to use TinyOS to implement the MAC layer and transition from GNU Radio flow graphs to a packet based system. Mandke et al. used Click to develop a MAC layer for SDR however all applications had to be run in Linux [18]. The reason for using TinyOS in UANT is twofold. First and foremost, much of the motivation to advance underwater acoustic communication comes from the attempt to further underwater sensor networks. USN have limited resource constraints and are the perfect candidate to run TinyOS since it is made specifically for this class of device. UANT is designed for research purposes specifically on different MAC and PHY layers, but using TinyOS will allow for an easier transition to an underwater sensor network node. The second reason we chose TinyOS comes from the fact that there is a large contributing base of users. This allows for UANT to benefit from development of protocols and algorithms for MAC and higher layers that are currently implemented as well as being developed for TinyOS.

As previously mentioned, GNU Radio provides a message queue which is able to make the transition from a streaming flow of data to a packet which highly facilitates the implementation of a MAC layer. In UANT the MAC layer sits closest to GNU Radio, in this way any packet that is sent to TOSSIM over the serial connection originating at the transducer will first pass through the MAC. A block diagram of the TOSSIM simulation can be seen in Fig. 2. If the packet has been determined to be a control packet, it will not proceed further than the MAC layer, and will be acted upon immediately, for instance to send an ACK, CTS, or any other packet needed for the current protocol in use. However, if the packet is for the running application, it will be sent straight through the MAC and control layers to the application. The received message is not queued here, it is left up to the application to queue received messages if needed. Messages that are to go beyond the MAC layer are only queued if they are intended for Linux via the virtual Ethernet card.

3.1.4. UANT: application layer

UANT not only provides flexibility at the MAC and PHY layer, but also gives two options on where to run applications. TinyOS applications that can currently run in TOSSIM, can be easily integrated into UANT. The configuration file must be wired properly in order to correctly connect the applications components, while the application code itself will change very minimally if at all. Using TinyOS in UANT for applications allows to develop in an environment similar to what could potentially run on an USN. These applications along with the MAC layer could be directly implemented on a sensor node running TinyOS in the future. For more complex applications we have also provided the ability for Linux to be used. In UANT if this option is enabled, a virtual Ethernet card will be established. All IP addresses of the network must

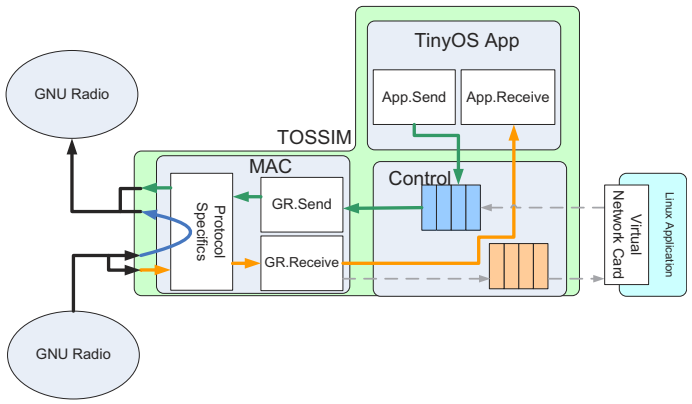


Fig. 2. TOSSIM simulation on node.

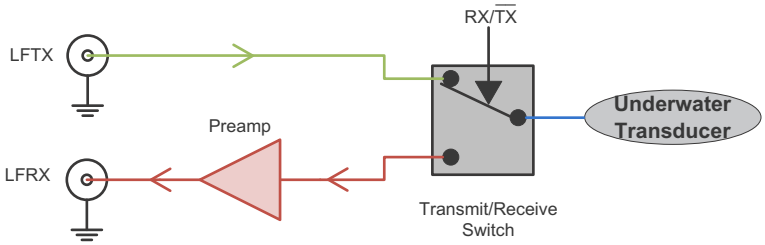


Fig. 3. A simplified schematic of preamplifier/switch board showing bidirectional nature of using single underwater transducer and a switch controlled by USRP. Transmit path (green), receive path (red), and shared transmit and receive path (blue) are shown. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

have the same network ID and the possible number of unique addresses for the network is 254. Applications such as ping, file transfer, and even media streaming applications could be possible with the use of proper physical layer modulation schemes that allow for the bandwidth needed. Not only does this allow for Linux applications to be used, but also for network monitoring tools that have been created such as wireshark [19] to be used to help characterize and monitor the links, since Linux sees UANT as a NIC.

It is possible to run both Linux and TinyOS applications simultaneously. This is accomplished using a control block that sits in TinyOS between the MAC and the application. A FIFO queue is kept (of configurable size) which allows packets from TinyOS and Linux to be multiplexed for more flexibility. When the MAC is ready to send a packet from a higher layer, it will simply signal for the first message in the queue. Any pending packets in the queue will be sent out to GNU Radio and through the water.

4. Applications and protocol implementation on UANT

4.1. Linux and TinyOS tools

When Linux applications such as ping and Netperf are chosen to be used then UANT benefits from Linux's TCP/IP stack. This is accomplished by using a TUN driver. A Python script is used to help establish the virtual Ethernet card and manage packets between Linux and TinyOS. Once the Python script has been started, an IP address can be assigned to the TinyOS node. The IP address that is manually assigned must have the same network address as all other nodes and the last byte must correspond to the same node ID used for the TOSSIM simulation. One current limitation in UANT regarding packets originating from Linux is the maximum transmission unit (MTU) must be set below 247 bytes. The reason for this is the packet size from Linux plus the TinyOS header size must be less than 256 bytes in order to be used with TinyOS packet protocols. The good news is UANT assigns the appropriate IP address with the MTU size that is set by the user.

Once the TinyOS node has been assigned an IP address Linux views UANT as a network card. The advantage to this approach is that UANT can now provide a full networking solution for underwater acoustics using the TCP/IP stack of Linux. As we later show this allows for application performance to be studied as a function of configuring the lower layers including MAC and PHY. If TinyOS applications are preferred than the many networking protocols that are available and used for sensor networks can be easily incorporated into UANT.

4.2. Channel allocation protocol

Baldo et al. found that allocating a higher frequency channel to a nearby node and lower frequency channel to

a distant node will be ideal with respect to the interference range of the transmitters [20]. However, this has not been tested and verified with a protocol yet. To verify this, we designed and implemented a channel allocation protocol by using a dedicated control channel to establish the channel of communication. After agreeing on the channel of communication, the sender and receiver initiate their data communications on that channel. Other nodes mark that channel as taken and use other idle channels to communicate thus lowering the rate of interference. By doing so, we have achieved lower interference and higher throughput at the same time, which are promising for underwater communications. The next two subsections explain the control packet and channel mask used by our protocol.

4.2.1. Control packet

The control packet used by the channel allocation protocol is a 7-byte control packet. We tried to keep it as small as possible so as to reduce interference on this control channel as all the nodes will be sharing this channel for the handshaking. It consists of source/destination information, a sequence number to suppress redundant packets, a channel mask indicating the list of available idle/taken channels from the sender or the selected channel from the receiver, a message type field indicating if it'sa RTS or CTS message and the length of the packet. The packet structure is shown in Fig. 4.

4.2.2. Channel mask

The channel mask used in channel allocation protocol is a sequence of zeros and ones. We are assuming eight channels that the nodes can communicate on. Channel 0 is reserved as a control channel and the remaining seven channels can be used for actual data transmission. Every node is maintaining a local channel mask indicating the list of idle and taken channels. Numeric 1 is used to indicate a idle channel and numeric 0 is used to indicate a taken channel. A typical channel mask will look like 11000010 representing that channel 1, 6, and 7 are available for communication.

4.2.3. Protocol description

Whenever a transmitter wants to communicate with another node, it sends a RTS (request to send) message and a channel mask through the control packet. On receiving RTS, the receiver perform logical AND operation on the sender's channel mask with its local channel mask to find the channels which are available at both the sender and the receiver. Then it uses the distance information gained from time synchronization (explained in subsequent sections) to select the channel of communication. It packs that information into the control packet and sends a CTS (clear to send) message to the sender. Sender on receiving the CTS, starts communicating with the receiver on the selected channel. A timer is set to fire for a period of packet length times the max transmissions permitted in case the packet is lost. Both sender and receiver hop back to the control channel after the transmission is done or the timer is fired whichever is earlier. Other nodes which have overheard this CTS will update their channel masks and mark the selected channel as taken. Also a timer is set as before and these nodes will reset the channel mask setting the channel as idle once the timer is fired. Handshaking, picking the channel and data transmission is shown in Fig. 5.

4.3. Time Sync: TSHL

TimeSync: It is useful for underwater acoustic networks to all have the same notion of time for time-stamping purposes and measuring distance each other. Syed et al. showed that clock offset and skew can be corrected in a reliable and efficient manner to achieve time synchronization for underwater acoustic networks using the Time Synchronization for High Latency (TSHL) protocol [21]. Using this protocol a leading transmitter will send out multiple time stamped beacons. All receiving nodes will calculate the difference between the received time stamp and the local time, compute a linear regression over all these values and find the slope of the line. Finally in the second phase offset is found using the skew compensated time. To verify its performance in a test bed, we have implemented this protocol on the UANT platform and evaluated its performance.

5. Implementation

We have implemented UANT on Linux boxes using the Intel core 2 quad Q8200 processor, with 4 GB of memory. We used USRP version 1.0 using the LFTX and LFRX daughterboards which have a range of DC-30 MHz, allowing for use in any underwater acoustic range. Typically with the low frequency daughterboards a separate transducer must be used for receiving and transmitting, since the daughterboards are independent. We have built a custom preamplifier board that also incorporates a switch in order to allow for one transducer per node, as well as amplify the received signal entering the USRP. We used RESON TC 4013 transducer for both transmitting and receiving, with a possible range of.1 Hz to 175 kHz.

The preamplifier that we built cut cost by using off the shelf ICs (LT6230 [22]. Furthermore only one transducer is needed rather than the usual two transducers required for the low frequency USRP daughterboards. The USRP allows control over several General-Purpose Inputs/Outputs (GPIOs) which we wired to a switch on the preamplifier board.

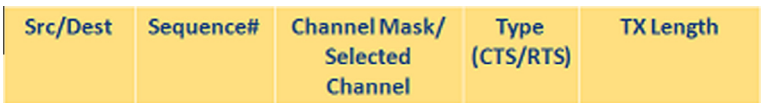


Fig. 4. Control packet.

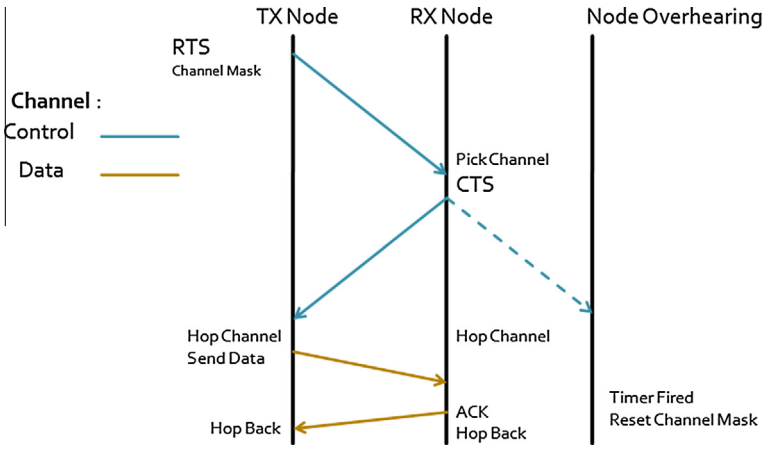


Fig. 5. Channel allocation protocol.

On one end of the Single Pole Double-Throw (SPDT) switch is the transducer being used both for transmitting and receiving. The receive side of the switch is fed into the preamplifier and finally out to the LFRX daughterboard. While the other signal path begins at the LFTX daughter board, goes through the switch and out the board to the transducer and through the water. A simplified schematic of this can be seen in Fig. 3.

The preamplifier that we built cut cost by using off the shelf ICs (LT6230 [22]. Furthermore only one transducer is needed rather than the usual two transducers required for the low frequency USRP daughterboards. The USRP allows control over several General-Purpose Inputs/Outputs (GPIOs) which we wired to a switch on the preamplifier board. On one end of the Single Pole Double-Throw (SPDT) switch is the transducer being used both for transmitting and receiving. The receive side of the switch is fed into the preamplifier and finally out to the LFRX daughterboard. While the other signal path begins at the LFTX daughter board, goes through the switch and out the board to the transducer and through the water. A simplified schematic of this can be seen in Fig. 3.

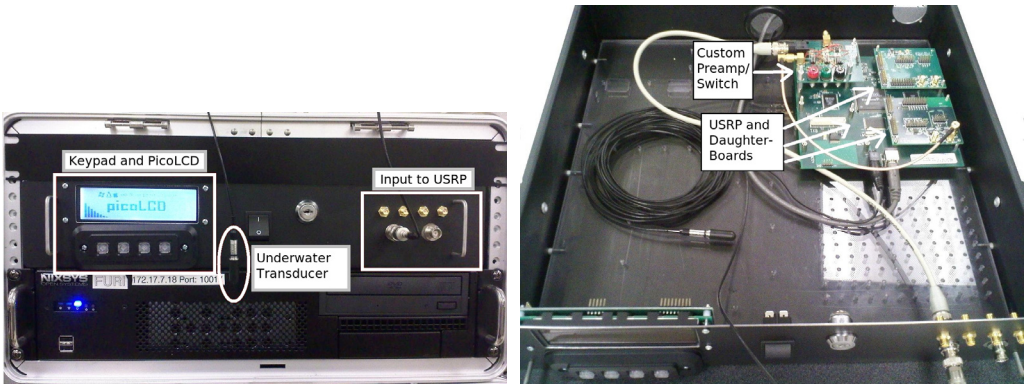
We have configured the GPIO to remain high unless the USRP is transmitting in which case it will be toggled low. After the transmission is completed the GPIO will again be high and the system will be in receive mode waiting for an ACK or any other packet to receive. This GPIO is controlled by the FPGA on the USRP which allows for very low latency switching between transmit and receive. The preamplifier can be powered from the USRP which ensures the need of only one power source for all components of UANT, making field deployments much easier. Powering the preamplifier from the USRP allows for a maximum voltage swing of 6.6 V. Using a high gain amplifier, we were able to correctly demodulate received signal less than 10 mV (before amplification) by varying the actual gain based on environmental conditions. We typically used values gain ratios of 10 or 100. As the water tank's limited size (48.25"L $\times$ 12.75"W $\times$ 21"H), we were able to use a gain value of 10 for most of our testing.

A deployable UANT system can be seen in Fig. 6. We have used rackmount servers (described above) enclosed in a rugged case for UANT to be taken into various terrain. A rackmount drawer was modified for UANT to be able to quickly access all connections to and from the USRP and the outside world. The USRP can be powered with any 5–6 V DC power supply allowing hard drive power connectors to power the USRP along with our amplifier boards used for acoustic communication. For field deployments a universal power supply can be used to power these UANT boxes.

For our MAC layer we currently implemented a basic Aloha protocol in TinyOS. The configurable parameters for this MAC are the minimum and maximum times for a backoff interval. If no ACK was received after data was sent, a retransmission is sent after a random time between the provided interval.

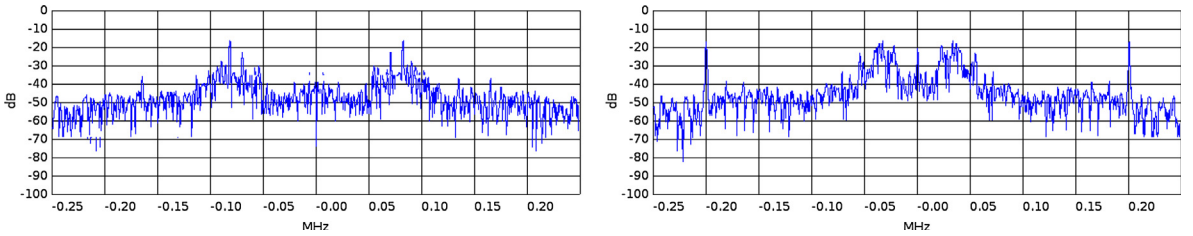
For the physical layer we have taken advantage of many of the configurable options that are provided for digital communication in GNU Radio. Transmission bit rate can be configured from 244 bits per second to 500 kilobits per second. We also varied the center frequency, and the limits are from .1 Hz up to 30 MHz which makes the real limiting factor the transducers (and the channel) being used which in our case was a maximum of 175 kHz.

Currently, due to transmission output power limitations we have been limited to the distance of our communications. The USRP has a maximum output voltage of two volts peak-to-peak and the LFTX daughterboard used provides approximately 2 mW (3 dBm) of output power. The underwater transducers being used have high input impedance at ideal acoustic frequencies requiring further power amplification for long range. For most of our testing we used high frequencies (100–175 kHz) because the impedance of the transducer is much lower allowing for higher transmit power. The gain of this added transmit power was more beneficial than the higher attenuation with distance that occurs at these frequencies. Even with the power limitation we have still successfully deployed UANT both in our lab tank as well as in a pool. The results



(a) Front view of UANT system (b) View inside drawer

Fig. 6. These two figures show a deployable UANT system (a) front view and (b) inside view of USRP, low frequency daughterboards and custom preamplifier and switch.



(a) FFT of tank setup. (b) FFT of larger pool.

Fig. 7. Frequency spectrum of tank (a) and pool (b). The two large noise sources (around 25 kHz in the tank, and 75 kHz in the pool) are the water pumps.

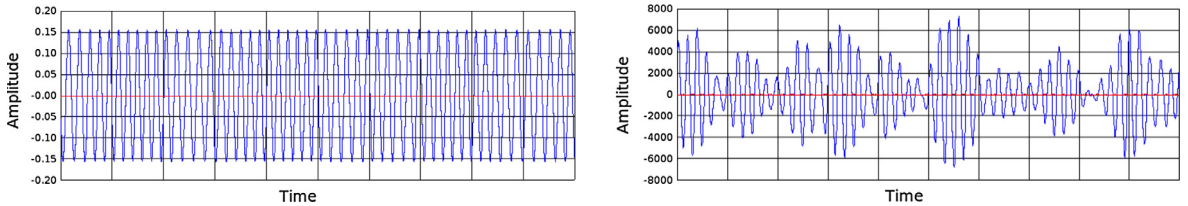
to follow were taken from testing in the 55 gallon tank setup (48.25"L $\times$ 12.75"W $\times$ 21"H) we had in our lab.

6. Evaluation

The need to have a deployment time configurable system is realized with Fig. 7. Depending on the location the optimal transmit center frequency can differ. The noise source of the lab tank was primarily the filtration system which was centered around 25 kHz. When we deployed UANT in the pool we found that the noise source was centered closer to 75 kHz.

Although it might be expected that a controlled environment would produce better results than a real world scenario this is not always the case. We found that our tank inside the lab was unfavorable for acoustic communication. The reason for the poor results can be seen in Fig. 8, where the multipath of the signal is quite apparent. The modulation for this test was not amplitude modulation but rather GMSK. The reason there is such a variation in the amplitude of the signal is multipath and inter-symbol interference, which is a result of the slow speed of propagation. The best remedy for the multipath in our lab tank is to lower the transmission speed, this allows more samples per symbol to be sent out to the USRP and received at the transducer.

One counterintuitive result we found in our lab tank testing was the effect of our water filtration system. Generally introducing a source of noise such as a filter that produces bubbles, acoustic noise, and surface waves, will degrade performance. Surprisingly this is not the case



(a) Original GMSK signal without multipath. (b) Effect of multipath on a GMSK signal.

Fig. 8. These two figures show the effect of multipath on a GMSK signal. (a) Shows the original signal, without multipath. Figure (b) is the result if the same signal gets transmitted through a tank.

and the explanation goes back to multipath. With the water standing still a transducer that is attempting to receive can be caught in a blind spot of destructive multipath interference such that almost no signal is seen. Although typically these blind spots vary with time and space due to moving reflective surfaces such as waves, in a controlled environment this is potentially not the case. Turning on the filter, although introducing noise, improved system performance by receiving up to twice the amount of correct packets because of the now time varying blind spots.

6.1. Linux and TinyOS tools

6.1.1. Ping

UANT allows the ability to see application level performance as a factor of the tunable parameters at the PHY and MAC layer. Fig. 9 shows how changing the minimum time needed to wait for an ACK before retransmission affect the Linux ping application. The backoff interval is set to 400 ms for this experiment and all other parameters are constant. If the minimum time to wait is too small there will be many packet collisions mostly with the retransmission of data and the ACK packet itself, this leads to an increased number of retransmissions. On the other hand, if the minimum time before retransmission is too large than if a packet is not received the transmitter is waiting needlessly long to resend the data leading to low throughput and a high round trip time (RTT) even though very few retransmissions are needed. Looking at this plot it is clear that trade-offs can be achieved to either optimize for minimum transmissions, quickest RTT, or a function of both.

6.1.2. Netperf

Another application that was tested in UANT was Netperf. Netperf establishes a connection between a client and server and then sends TCP packets across the link. Note that we did not need to make changes to standard TCP implementation in our lab testing. However, default timeout values should be increased for a larger distance between nodes to avoid unnecessary timeouts. Fig. 10 shows the results from the Netperf testing, where each

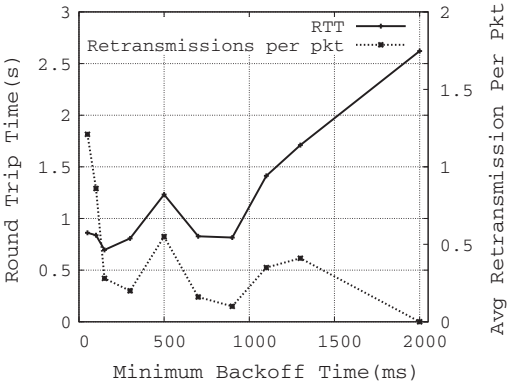


Fig. 9. Averaged over 50 pings for each data point with center frequency at 100 kHz and transmit rate at 5 kb/s. All backoff intervals (max–min) set to 400 ms.

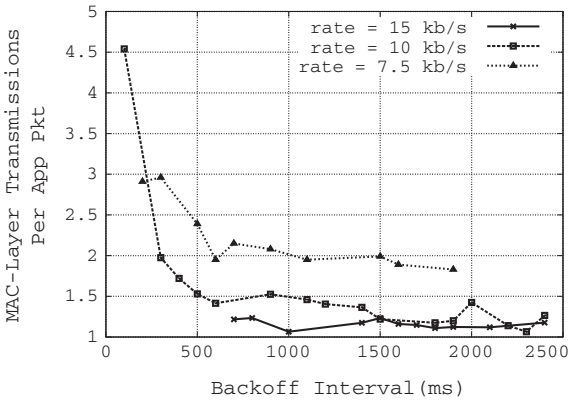


Fig. 10. Netperf performance while varying minimum Aloha backoff time at different transmission rates. Each data point collected over 300 s of communication.

data point was collected over approximately a 300 s interval. We varied the transmission rate at both the physical layer and MAC layer to see how that would affect performance. Regardless of the transmission rate if the Aloha minimum backoff interval was too low than it would cause many retransmissions. As the minimum backoff interval increased we found that the amount of retransmissions decreased since the nodes were not trying to retransmit to soon. As the transmission rate varied the amount of retransmissions needed also varied. The fastest rate of 15 kb/s lead to the least amount of retransmissions while the slowest rate that was tested (7.5 kb/s) resulted in the highest amount of retransmissions. This result brings up another advantage of having an easily reconfigurable system which allows to specify parameters at runtime.

6.1.3. A TinyOS application

We have also implemented an example TinyOS application that fires a periodic timer and transmits its count over the network to the other nodes. We were able to leave the application code unmodified while only changing the wiring configuration. The throughput of this application was much higher than the Linux applications that we tested for performance. One reason for this being that the overhead of a TCP/IP packet is much greater than the eight bit TinyOS message header that was used. Also there is added latency for packets from Linux to get to TinyOS

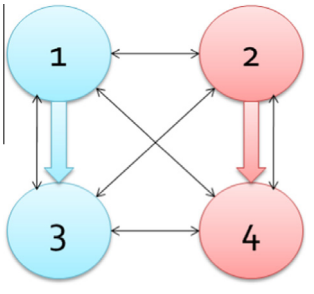


Fig. 11. Test topology for channel allocation protocol.

and then sent out, when compared to that of the application currently running in TOSSIM.

6.2. Channel allocation protocol

To evaluate our distributed channel allocation protocol, we deployed four nodes as shown in Fig. 11. In this topology, the network is fully connected and there are two sources and two sinks. This topology was chosen because we were limited to four nodes in our lab tank (i.e. severe multipath) and wanted to have simultaneous transmissions. Node one transmits to node three periodically while node two transmits to node four. A message is generated using the RadioCountToLeds TinyOS example application. This application periodically (every second by default) transmits a counter packet with different payload size to all of its neighbors.

6.2.1. Collision ratio

A plot of the collisions using the channel hop protocol can be seen in Fig. 12. We also plotted the collisions using a shared channel and a random access MAC–ALOHA. Although this is not a completely fair comparison since proposed channel allocation protocol does use more bandwidth, it is plotted for a reference. One important observation is that ALOHA without channel allocation increases the collision rate as the packet size increase. However, the channel allocation approach shows approximately fixed collision rate with the different packet size. The reason for this is that the collisions happen only on the control channel in proposed channel allocation protocol. Once the transmitter and receiver hop to the reserved channel, they can transmit data without collisions. This result leads to the conclusion that if the nodes are able to access the control channel sooner, collisions would be lesser. This can be seen by increasing control packet size, whose results omitted for sake of space limitations. *Packet size*: To verify this gain in a large-scale, we deployed 60 nodes in the network via TOSSIM. As there were more nodes deployed, the data channels need to be shared. This will increase collision rate as the channels were reused and packet size grew. This effect can be seen in the simulation results in Fig. 13. This result shows that with an increase in the number of channels, the collision ratio is reduced. Furthermore since it is a 60 node network and there are only four channels available, some nodes have to share channels to transmit data on and collisions may happen there. However, it is still significantly reduced from the case where all nodes share a common channel.

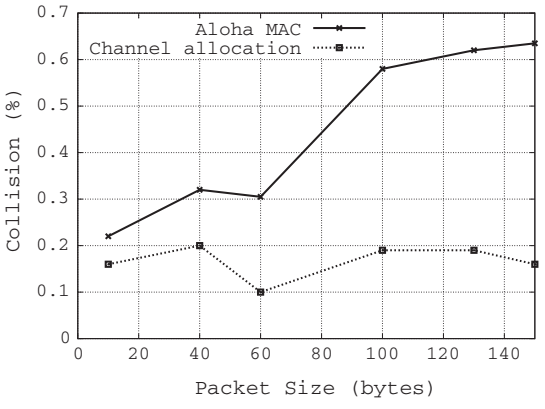


Fig. 12. Collision ratio as a function of packet size.

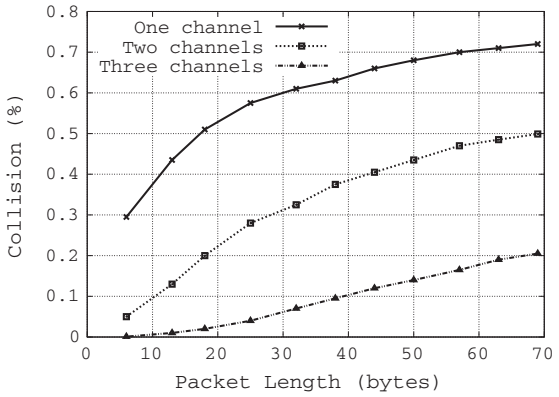


Fig. 13. 60-Node network with different packet sizes.

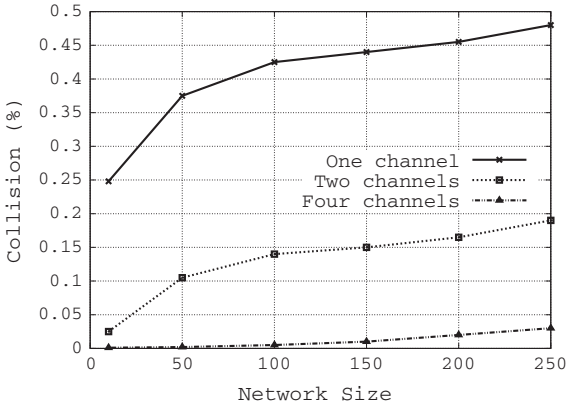


Fig. 14. A fixed packet size with different network sizes.

6.2.2. Network size

We also simulated the channel allocation protocol while varying the number of nodes in a network. In this simulation, we fixed the packet size of 12-byte. As shown in Fig. 14, it shows the global tendency that collision rates increase as the number of nodes in the network increase. However, using multiple channels significantly reduce the collision rates in this scenario.

6.3. Time synchronization

To evaluate THSL's performance, we were taken 20 Beacons using 10-s intervals on UANT. In addition, all our tests were run using the millisecond timer as opposed to the microsecond timer. We had the opportunity to use the more precise microsecond timer; however, since our errors were on the range on milliseconds, we did not see a point

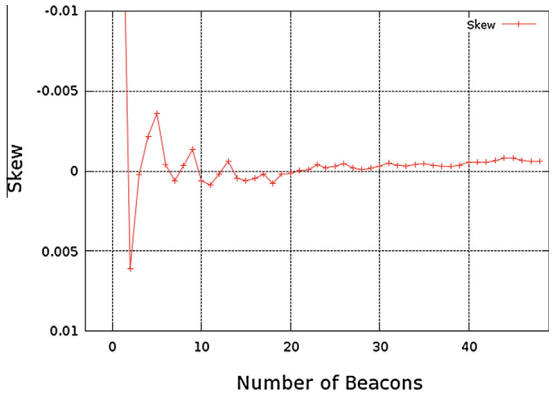


Fig. 15. Number of beacons vs. skew.

in placing additional stress on the system by using microsecond timers.

6.3.1. Number of beacons

Shown in Fig. 15, the nodes share the same notion of time after 7 beacons are sent the skew between nodes converges. We also noticed that using a software defined radio introduces non-deterministic latency, so the accuracy of the skew and the offset of two nodes will increase with a dedicated hardware solution even if fewer beacons are used.

6.3.2. Beacon intervals

Fig. 16 indicates that when the beacon interval is not long enough, the interference from multiple beacon messages (i.e., multipath) serve to disrupt reliable communication between the beacon and the local node. Due to clock drift that is apparent in all oscillators, even after nodes have been time synchronized, they will eventually drift apart. This leads to the need for periodic re-synchronization.

6.3.3. Factors of non-linearity

There were a few possible reasons that affect the time synchronization results. One was that our time-stamping was done at a very high level in our system, at the application layer. This is a limitation with our system. Perhaps with a switch to a real-time pre-emptive kernel, the non-determinism can be reduced. However, even switching to a better kernel might not change the fact that there is some inherent non-determinism in our system. The packet will spend some non-deterministic time going from the application layer all the way down to the MAC layer, where ideally, is where we should time-stamp the packet. At the time of the developing the system, there was no easy way to have timestamps inserted into the data stream at the hardware level. The only way to insert the time-stamp at the appropriate location was when forming the packet at the higher layers. This restriction could be alleviated by adding custom blocks in the USRP FPGA offloading some of the software radio processing to hardware in exchange for easy configurability. The USRP driver now offers an option to timestamp received samples coming from the radio. This allows an easy way to get an precise timestamp of a received packet. However, to inject a timestamp for a transmitted packet would still require modifications to the USRP FPGA. By making improvements on the receiver side will help, but not eliminate the error introduced.

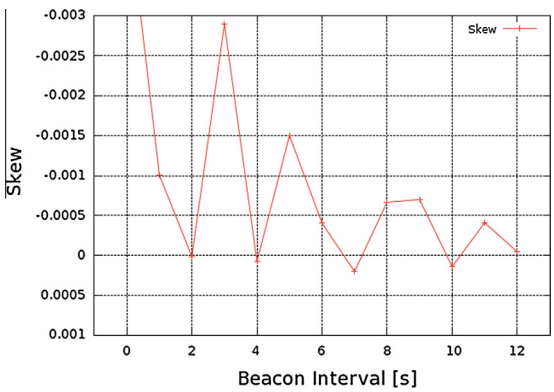


Fig. 16. Beacon intervals vs. skew.

While very cheap oscillators tend to have a drift of 30–50 ppm (ppm), many underwater ranging solutions use more precise clocks that are temperature compensated that can achieve accuracies of less than 1 ppm [23]. Two nodes with 50 ppm clocks can accumulate a maximum error of 50 ms in approximately 8.3 min, while the clock used by Eustice et al. in [23] will accumulate 2 ms of error in just under 14 h. Therefore depending on the nodes hardware, re-synchronization rates can vary dramatically but are still feasible with limited overhead.

In addition to an accurate clock source being used to reduce overhead of re-synchronization, time stamp information of beacons can be piggybacked in the header of a data transfer from the node with the reference clock. In this way when a node is receiving data it can also perform the linear regression and update the values of skew and offset. Since phase two of TSHL requires one packet from the receiving node to be sent back to the transmitter, this information can be appended to the acknowledgment that is sent after the data transfer.

7. Conclusion

The challenges of the underwater acoustic channel are great, requiring the need for flexibility during system development. We have presented UANT as a complete end-to-end networking platform for underwater acoustic communication. UANT is designed for field-deployment, geared toward fast prototyping and testing of PHY and MAC layer schemes. It is implemented using open source software, facilitating further extension. We have demonstrated a Linux application running on UANT and evaluated channel allocation and time synchronization protocols on UANT.

Although we have applied and evaluated protocols and algorithms along with implementing a few of our own, many are geared toward over the air or wired communication. In the future we plan to implement recently proposed underwater MAC and Routing protocols recently available. Allowing the protocols to be compared using UANT with either a TinyOS or Linux application to benchmark performance will help distinguish strengths of each solution as

compared with one another. We plan to further study the effect of changing PHY and MAC layer parameters.

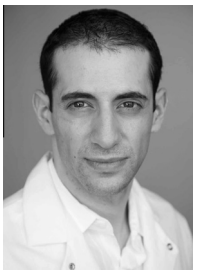
Acknowledgment

This work was supported in part by the US National Science Foundation under Grant No. 1205757.

References

[1] J. Preisig, Acoustic propagation considerations for underwater acoustic communications network development, SIGMOBILE Mob. Comput. Commun. Rev. 11 (4) (2007) 2–10.
[2] E. Sozer, M. Stojanovic, J. Proakis, Underwater acoustic networks, IEEE J. Ocean. Eng. 25 (1) (2000) 72–83.
[3] D. Torres, J. Friedman, T. Schmid, M.B. Srivastava, Software-defined underwater acoustic networking platform, in: WUWNet, 2009.
[4] G.E. Santagati, T. Melodia, Sonar inside your body: prototyping ultrasonic intra-body sensor networks, in: INFOCOM, Toronto, Canada, 2014.
[5] E. Jones, The application of software radio techniques to underwater acoustic communications, 2007, pp. 1–6.
[6] E.M. Sözer, M. Stojanovic, Reconfigurable acoustic modem for underwater sensor networks, in: WUWNet, 2006.
[7] E. Blossom, Exploring GNU Radio. <http://www.gnu.org/software/gnuradio>.
[8] USRP Brochure, April 2009. <http://www.ettus.com/>.
[9] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, E.B.M. Welsh, D. Culler, in: Ambient Intelligence, Springer, Berlin Heidelberg, 2005, pp. 115–148.
[10] K. Klues, G. Hackmann, O. Chipara, C. Lu, A component-based architecture for power-efficient media access control in wireless sensor networks, in: SenSys, 2007.
[11] P. Levis, N. Lee, M. Welsh, D. Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, in: SenSys, 2003.
[12] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, T. Schoellhammer, A system for simulation, emulation, and deployment of heterogeneous sensor networks, in: SenSys.
[13] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, P. Steenkiste, Enabling MAC protocol implementations on software-defined radios, in: NSDI, 2009.
[14] L. Yang, Z. Zhang, W. Hou, B.Y. Zhao, H. Zheng, Papyrus: a software platform for distributed dynamic spectrum sharing using SDRS, SIGCOMM Comput, Commun. Rev. 41 (1) (2011).
[15] N. Truong, Y.-J. Suh, C. Yu, Latency analysis in GNU radio/USRP-based software radio platforms, in: Military Communications Conference, MILCOM 2013 – 2013 IEEE, 2013.
[16] N. Truong, C. Yu, Investigating latency in GNU software radio with USRP embedded series SDR platform, in: Eighth International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA), 2013, 2013.
[17] T. Schmid, O. Sekkat, M.B. Srivastava, An experimental study of network performance impact of increased latency in software defined radios, in: WinTECH, 2007.
[18] K. Mandke, S.-H. Choi, G. Kim, R. Grant, R. Daniels, W. Kim, R. Heath, S. Nettles, Early results on hydra: a flexible mac/phy multihop testbed, in: VTC, 2007.
[19] Wireshark network protocol analyzer. <http://www.wireshark.org/>.
[20] N. Baldo, P. Casari, P. Casciaro, M. Zorzi, Effective heuristics for flexible spectrum access in underwater acoustic networks, in: OCEANS 2008, 2008.
[21] A.A. Syed, J. Heidemann, Time synchronization for high latency acoustic networks, in: Infocom, 2006.
[22] LT6230. <http://www.linear.com/product/LT6230>.
[23] R. Eustice, L. Whitcomb, H. Singh, M. Grund, Recent advances in synchronous-clock one-way-travel-time acoustic navigation, in: OCEANS.

Dustin Torres is a firmware engineer for Intel Corporation. He received his B.S. in computer engineering from University of California, Irvine in 2008 and his M.S in electrical engineering from University of California, Los Angeles in 2010. His research interests include wireless sensor networks and underwater wireless communication.

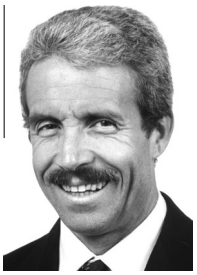


Jonathan Friedman is the CEO and cofounder of GetScale, Inc., a company that develops technologies to improve yield in Chinese factories. He received his B.S. in computer engineering from the Georgia Institute of Technology (Georgia Tech) and his M.S. and Ph.D. in electrical engineering from the University of California, Los Angeles (UCLA). He is a visiting scholar in the Department of Neuroscience at UCLA where his research interests include automated failure mode analysis, process controls, and sensor systems for medical and industrial environments.

Thomas Schmid is an Adjunct Assistant Professor in the Electrical and Computer Engineering Department at the University of Utah, and the CEO and co-founder of RF Ranging, Inc.. His research interests are in wireless, embedded, and networked systems. He was previously a CI Fellows Post Doctoral Scholar at the University of Michigan, Ann Arbor. He received his PhD in Electrical Engineering from the University of California, Los Angeles in 2009, and was selected to receive the UCLA Elecrical Engineering Department's 2009–2010 Outstanding Doctor of Philosophy Award for his dissertation. His research interests involve the hardware-software boundary and its impact on energy consumption, including software radios, large scale sensing system architectures, and networking, with a focus on wireless embedded systems.

Mani B. Srivastava (Ph.D. 1992, Berkeley is on the faculty at UCLA where he is associated with the EE Department with a joint appointment in the CS Department. His research is broadly in the area of networked human-cyber-physical systems, and spans problems across the entire spectrum of applications, architectures, algorithms, and technologies. His current interests include issues of energy efficiency, privacy & security, data quality, and variability in the context of systems and applications for mHealth and sustainable buildings. He is a Fellow of the IEEE.

YoungTae Noh is a Software Engineer at Cisco Systems, Inc. Prior to joining Cisco Systems, he received his B.S. in computer science from Chosun University in 2005, an M.S. degree in Information and Communication from Gwangju Institute of Science Technology (GIST) in 2007, and a Ph.D. in computer science at University of California, Los Angeles (UCLA) in 2012. His research areas include data center networking, wireless networking, future Internet, and mobile/pervasive computing.



Mario Gerla is a Professor in the Computer Science at UCLA. He holds an Engineering degree from Politecnico di Milano, Italy and the Ph.D. degree from UCLA. He became IEEE Fellow in 2002. At UCLA, he was part of the team that developed the early ARPANET protocols under the guidance of Prof. Leonard Kleinrock. At Network Analysis Corporation, New York, from 1973 to 1976, he helped transfer ARPANET technology to Government and Commercial Networks. He joined the UCLA Faculty in 1976. At UCLA he has designed and implemented network protocols including ad hoc wireless clustering, multicast (ODMRP and CodeCast) and Internet transport (TCP Westwood). He has lead the $12 M, 6 year ONR MINUTEMAN project, designing the next generation scalable airborne Internet for tactical and homeland defense scenarios. He is now leading two advanced wireless network projects under ARMY and IBM funding. His team is developing a Vehicular Testbed for safe navigation, urban sensing and intelligent transport. A parallel research activity explores personal communications for cooperative, networked medical monitoring (see www.cs.ucla.edu/NRL for recent publications).