Short Paper: The Softwater Modem – A Software Modem for Underwater Acoustic Communication

Brian Borowski and Dan Duchamp
Department of Computer Science
Stevens Institute of Technology
Castle Point on Hudson, Hoboken, NJ 07030
{bborowsk, djd}@cs.stevens.edu

ABSTRACT

The Softwater Modem is a software modem for underwater acoustic communication that enables users to run applications on the familiar sockets interface without any additional hardware except for transducers and associated amplification. A standard TCP or UDP transport protocol runs on top of IP, which via the Linux TUN driver, runs on top of custom datalink and PHY layers tailored specifically to the underwater channel. The datalink and PHY layers, written entirely in Java, use frequency-division multiple access (FDMA) with binary and 4-FSK (frequency shift keying) in any frequency band supported by the computer's sound card and can run at any bit rate supplied by the user. The transmitter sends a per-packet linear frequency modulated (LFM) chirp signal that the receiver uses for packet synchronization as well as channel estimation, with the option of applying impulse response estimates to channel equalization. Frames can contain up to 255 bytes and are encoded with Reed-Solomon (R-S) codes, for which the user can specify the number of parity bytes. The paper describes the architecture and performance of this system, which currently demonstrates two-way communication as well as real-time channel estimation.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Signal processing systems; C.2 [**Computer-Communication Networks**]: Data communications

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

Underwater acoustic modem, SDR, DSP, noncoherent FSK, inverse filter

1. INTRODUCTION

There is no such thing as a typical underwater acoustic communication channel [1]. The large variation in channel conditions among different locations – especially the difference between deep water and shallow water – suggest that vastly different communication parameters (modulation technique, frequency band, frame length, error correction methods, etc.) would be optimal for different locations. Existing acoustic modems are implemented at least partly with custom hardware and paired with a fixed-point or floating-point DSP [2, 3]. Such solutions typically offer a limited choice of operating parameters. Especially in the case of commercial products, modem parameters are often chosen based on worst-case channel assumptions in order to maximize the modem's utility, necessitating a series of products, each tailored to specific environments that vary in depth, link distance, and expected severity of multipath [4]. While logical, this is an unfortunate development because flexible, optimized communication is especially important in the bandwidth-limited underwater environment.

Accordingly, we have built an all-software acoustic modem for underwater operation. The only necessary hardware components are the transducers, amplifiers, and cables. All other tasks are performed either by software or hardware ordinarily found in any PC-like platform; e.g., A/D conversion is performed by the sound card. The modem is able to sense and adapt to its environment on a very short time scale. In particular, a "sounding signal" precedes every packet and is used by the receiver to compute and apply the channel's inverse impulse response to the modulated data signal that follows the sounding signal. In this way the receiver mitigates channel distortion on a packet-by-packet basis. The modem allows the user to alter its functionality by editing a text file containing name/value pairs. More than a dozen options are supported, including base frequency, number of carriers, symbols per second, number of parity bytes to be used within the R-S code, and the payload size.

Besides being able to adapt to channel conditions, a software modem offers the advantage of being far less costly than current hardware devices – such as the Benthos 013424 LF (9-14 kHz) omni-directional modem at $8800/pair as of April 2009 – and of being easily configurable. Modem parameters can be selected to match the environment, thereby avoiding worst case assumptions and making communication more efficient.

An additional advantage of our software architecture is that it supports TCP/IP based communication. While TCP/IP is not optimal for the underwater channel, applications can use the popular sockets interface and run unaltered on top of our modem layer, any number simultaneously. The effect is as if an Ethernet had been replaced by a (much slower) acoustic channel.

2. RELATED WORK

There are several underwater acoustic modems in existence, either as commercial products [2, 4, 5, 6] or research efforts [3, 7], all of which make use of custom hardware to varying degrees. The purely software defined approach to packet radio in Sound-

WUWNet'09, November 3, 2009, Berkeley, CA, USA.

modem [8] offers several modulation techniques but did not work well in our underwater communication experiment. With J-QAM [9], high data rates have been achieved by using a PC sound card with RF transmission, though the phase-coherent detection methods necessary for quadrature amplitude modulation (QAM) modulation work well only in vertical underwater acoustic channels with little multipath distortion [10]. GNU Radio [11] is probably the largest, most flexible software defined radio (SDR) platform to date, but it is designed for use with RF technology and is in a constant state of change. The prototype from UCSB [12] combines DSP techniques, hardware-software integration, and network protocols, but operates at a fixed rate of 161 bps. Hwang's design [13] uses a PC sound card with MATLAB as an SDR orthogonal frequency-division multiplexing (OFDM) communication system; however, the system cannot operate in real time. Furthermore, the code optimizations performed at UCONN [14] have still not been able to produce a real-time DSP-based OFDM receiver. Since each system has limitations, we focus on the implementation of a flexible software modem that performs well in various types of underwater acoustic channels while requiring a reasonable amount of processing power, such as that found in an average laptop PC.

3. SYSTEM ARCHITECTURE

3.1 Software architecture

The overall architecture of the system includes three layers of user space applications. The highest layer is the application itself, which can use either TCP or UDP. The lowest layer is the Java application that implements the functionality of an acoustic modem. Between the two is the tunnel relay application, which is responsible for passing IP datagrams between the network application and Java modem. Figure 1 depicts the overall architecture of the system and shows how the component applications are linked.

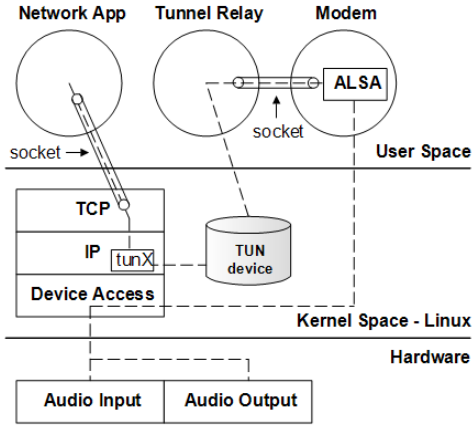

Figure 1: Software architecture

The tunnel relay application communicates with the Java modem via UDP, as sockets are the only form of inter-process communication (IPC) that works with Java. Both the tunnel relay application and Java modem bind to the local loopback IP address, but with different port numbers, to allow full-duplex communication between the two processes. Thus, the Java modem listens on one socket for datagrams from the tunX device that need to be transmitted acoustically, while it sends datagrams that have been received acoustically on the other socket where the tunnel relay program is listening.

3.2 Associated hardware

Two laptops have been used in the development of this system. Each has a dual-core Intel processor running at 2 GHz and 2 GB of memory. One has an ADI1981 codec running on top of the integrated Intel high-definition audio (HDA) controller. The sound card supports a maximum sampling rate of 48 kHz. The other has a Conexant CX20561 codec paired with an Intel HDA controller. This integrated sound card supports a maximum sampling rate of 192 kHz. Both systems were running Ubuntu Intrepid 8.10, ALSA 1.0.17, and Java 1.6.0 Update 13.

4. MODEM ARCHITECTURE

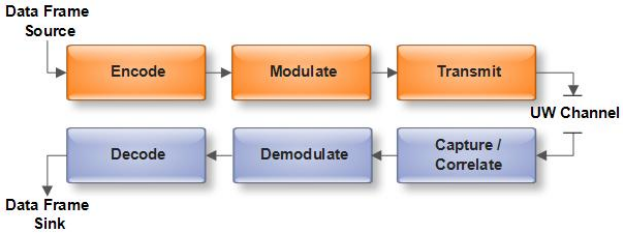

Figure 2: Processing blocks within Java modem

The modem portion of the system is written entirely in Java. It is divided into two main functions, transmit and receive, which execute in parallel. Both functions are implemented as a series of stages which are processed by threaded objects. Adjacent stages communicate via a thread-safe queue as shown in Figure 2.

4.1 Transmitter design

The transmitter consists of three stages – the source encoder, the modulator, and playback mechanism. The encoder reads incoming datagrams from the UDP socket attached to the tunnel relay application, wraps them in a frame header, and optionally applies R-S codes. The modulator converts the incoming byte-oriented data frame into symbols of 0s and 1s for binary FSK (or the 2-bit symbols 00, 01, 10, and 11 for 4-FSK) and then translates the symbols into the samples of the sine wave that correspond to the frequency representing a given symbol. The modulator also prepends an LFM chirp signal and guard time block to the beginning of a data frame for synchronization and channel estimation purposes at the receiver. Finally, the transmitter takes the buffered modulated signal and sends it to the sound card for playback. The transmitter can also optionally record each outgoing modulated data frame in a .wav file for future reference.

4.2 Receiver design

The receiver consists of three stages as well – the correlator, demodulator, and decoder. The correlator continually reads blocks of samples from the sound card. The block itself must be longer than the chirp signal, so that the chirp signal is guaranteed to fit within two consecutive blocks. For every incoming block, the correlator concatenates it with the previous block before using cross-correlation to detect the start of a frame. If a frame is detected, the exact number of samples within the frame is buffered, and then placed on the queue for the demodulator to pick up.

The demodulator converts an acoustic signal into a bit stream of 0s and 1s. It optionally takes the impulse response obtained during frame detection, inverts it with the Levinson-Durbin algorithm [15], and convolves it with the signal as a means of performing channel equalization on a packet-by-packet basis. Regardless of whether inverse filtering is applied, the demodulator

bandpass filters the signal, holds a tournament to see which of the carriers has the strongest signal over the duration of a symbol, and outputs the corresponding symbol (0 or 1 for binary FSK; 00, 01, 10, 11 for 4-FSK). It also optionally computes the SNR for the frame before placing it on the queue for the decoder to pick up. At the user's request, the demodulator can also record each incoming unprocessed frame and demodulated frame in separate .wav files and each impulse response and inverse impulse response in a .csv file for post-processing.

The decoder applies R-S codes to an incoming data frame and reports if no errors were detected, if errors were found and corrected, or if errors were found but could not be corrected. The decoder also verifies the CRC in the header before extracting the frame payload, or IP datagram, and sending it out to the TUN device via the UDP socket.

5. FRAME FORMAT

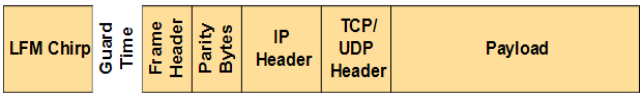

Figure 3: Format of data frame

Figure 3 depicts the frame format. Each frame begins with a LFM chirp signal followed by a block of silence known as the guard time. The 4-byte frame header format is extremely simple, containing only two fields – a 16-bit CRC and 16-bit length attribute. All other information relevant for communication is contained in the headers for the IP and the transport layer, whether TCP or UDP. User data appears after all headers and fills the remaining bytes of the frame up to the 255-byte limit.

If R-S codes are enabled, the parity bytes appear after the frame header and before the IP header. In comparison with the frame sizes supported by the Micro-modem [3], 255 is a reasonable limit and should be more than adequate for harsh underwater channels requiring use of noncoherent FSK demodulation.

6. SIGNAL PROCESSING

In order to maintain orthogonality between the tones, the modulation index is set to 1 so that the distance between the tones is equivalent to R Hz, where R is the data rate in symbols per second.

The LFM chirp signal that precedes a frame is generated as an array of floats. The chirp signal covers only the frequency band required by the data modulation. Incoming samples are cross-correlated with the reference chirp signal, which generates an estimate of the channel's impulse response. When cross-correlation yields a value that exceeds the user-defined threshold, a frame is deemed present and its samples are stored in a buffer.

Once all the samples have arrived, the receiver passes the data buffer, impulse response estimate, and most recently computed power spectral density of noise to the demodulator block for processing. The demodulator is comprised of a series of stages that implement noncoherent FSK detection.

The first stage, which serves as a means of channel equalization on a frame-by-frame basis, takes the impulse response estimate and inverts it by means of the Levinson-Durbin algorithm. As long as the channel remains fairly constant over the duration of a data frame and SNR > 12 dB to prevent noisy impulse response estimates, this method works to mitigate intersymbol interference (ISI) and problems with the frequency response of the transducers.

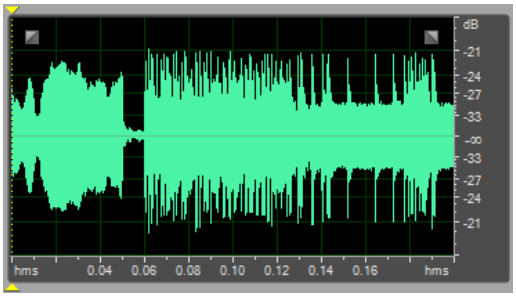

Figure 4: Unequalized reception of data frame

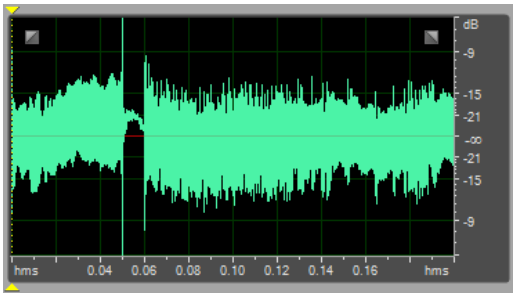

Figure 5: Equalized reception of data frame

The inverse impulse response is then convolved with all the samples in the frame via FFT convolution. Figure 5 displays the equalized time-domain view of the unequalized data frame shown in Figure 4. Spectral splatter associated with rapidly turning the transmitter on and off becomes more evident after applying the inverse filter, with large spikes appearing at the beginning and end of the acoustical signals. While spectral splatter does not degrade performance in a point-to-point system, it is something to consider when other devices begin sharing the channel, especially one that is bandwidth-limited. Since R-S codes perform well when errors occur in bursts, they have been built into the system as a means of combating spectral splatter from neighboring devices.

After packet synchronization, noncoherent detection begins with bandpass filtering. When using binary FSK, the incoming signal is passed through two separate second-order infinite impulse response (IIR) filters (four in 4-FSK) to eliminate signals outside the band corresponding to the tones representing bits. The best performance is obtained when the product BT is close to 1.0, where B is the -3dB bandwidth of the filter in Hz and T is the duration of a symbol [16].

Upon completion of the filtering stage, the resulting signals are passed to the envelope detection stage, which first applies the Hilbert transform to create the analytic signal from the input and then computes its absolute value. Finally, for each symbol in the frame, the demodulator compares the envelopes, choosing the symbol that corresponds to the tone with the larger amplitude.

7. PERFORMANCE

Table 1 lists the running times in milliseconds of various methods within the Java code measured with JRat [17] on three different platforms. The values shown are averages computed over 5 frames. Each frame consisted of a 4-byte frame header, 16 parity bytes, and 128 bytes of payload, for a total of 1184 bits, which at 1000 bits/second takes 1184 ms to transmit. The total frame transmission time was 1184 ms plus 50 ms for the chirp signal and a 10 ms guard time: a total of 1244 ms. The time to encode the data with R-S codes and modulate the entire block is longest on

the T500, taking 96 ms, or 7.72% of the total frame transmission time. Note the desktop processor is overclocked to 3.0 GHz.

Table 1: Processing time of subroutines

	Desktop Q6600 OC	Laptop T60p	Laptop T500
Transmit			
a. Modulate	8.00	12.00	13.40
b. Encode Reed-Solomon	9.33	74.40	82.60
Sum (a:b)	17.33	86.40	96.00
Frame duration	1244.00	1244.00	1244.00
Comp Time/Signal Length	1.39 %	6.95 %	7.72 %
Receive			
c. Cross-correlation	2.36	5.03	4.46
Block length	85.33	85.33	85.33
Comp Time/Signal Length	2.77%	5.89%	5.23%
Demodulate			
d. Levinson-Durbin	3.40	3.80	5.33
e. FFT convolution	29.80	65.00	43.83
f. Bandpass filtering	2.60	3.60	4.16
g. Envelope detection	61.60	117.60	84.50
h. Normalizer	1.60	3.70	1.50
i. Comparator	0.40	2.00	0.33
j. Bit Decision	0.40	1.60	0.50
k. Decode Reed-Solomon	1.33	21.40	17.40
l. Write 2 wav files	2.00	3.40	2.60
m. Write IR data to csv file	16.33	55.40	36.00
Sum (d:m)	119.46	277.50	196.15
Frame duration	1244.00	1244.00	1244.00
Comp Time/Signal Length	9.60 %	22.31%	15.77%

Data from the sound card is read in blocks of 4096 samples, which correspond to approximately 85.33 ms. Two blocks are concatenated before performing the cross-correlation operation. Performing cross-correlation on 8192 samples on the slowest platform consumes 5.03 ms, 5.89% of the duration of an incoming block of audio samples.

The Levinson-Durbin algorithm, which is $O(n^2)$, operates on 480 samples of guard time (10 ms sampled at 48 kHz). The slowest platform takes 5.33 ms. We also tested a 20 ms guard time, where matrix inversion took 13.50 ms on average. For small numbers of samples, the computation time for a Java implementation of Levinson-Durbin matrix inversion seems quite practical.

The two functions that consistently consume the most time on each system are convolution and envelope detection. This is not surprising, as each requires computing the forward and inverse FFT of 56,832 samples. Since the worst-case demodulation time is already about 22% of the duration of the data frame with binary FSK (and more with 4-FSK), it might be beneficial to investigate other options, including faster FFT implementations and even the quadrature receiver for noncoherent energy detection.

8. CONCLUSION

We have described the implementation of an open source all-software acoustic modem that can handle packets of up to 255 bytes in real time on stock hardware. The modem is parameterized so that the user can adapt it to different operating conditions. Parameters include modulation technique, symbol rate, frame size, and ECC overhead. Additionally, the modem performs real-time, per-packet channel characterization by preceding each frame with a chirp signal that enables the receiver to compute the channel impulse response within the relevant band and to compute and apply an inverse filter for equalization.

9. ACKNOWLEDGMENTS

The authors thank Bruce McNair and Theodore Kamakaris for the many discussions about practical aspects of building and measuring the performance of digital communication systems.

10. REFERENCES

[1] J. Preisig, "Acoustic Propagation Consideration for Underwater Acoustic Communications Network Development," Proc. *WUWNet'06*, Los Angeles, Sept. 2006.

[2] Benthos, "Telesonar Underwater Acoustic Modems Product Catalog 2009," URL=http://www.benthos.com/pdf/Modem%20Telesonar%20Product%20Catalog%20LR.pdf, Accessed Apr. 2009.

[3] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball, "The WHOI Micro-Modem: An Acoustic Communications and Navigation System for Multiple Platforms," Proc. IEEE/MTS OCEANS Conf. Exhib., Washington, D.C., Sept. 2005.

[4] LinkQuest, "Underwater Acoustic Modem Models," URL=http://www.link-quest.com/html/uwm_hr.pdf, Accessed Apr. 2009.

[5] DSPComm, "AquaComm: Underwater wireless modem," URL=http://www.dspcomm.com/products_aquacomm.html, Accessed Apr. 2009.

[6] Tritech, "AM-300 Acoustic Modem," URL=http://www.tritech.co.uk/products/datasheets/am-300-acoustic-modem.pdf, Accessed Apr. 2009.

[7] E. Sözer and M. Stojanovic, "Reconfigurable Acoustic Modem for Underwater Sensor Networks," Proc. *WUWNet'06*, Los Angeles, Sept. 2006.

[8] T. Sailer, "Soundmodem on Modern Operating Systems (2000)," URL=http://www.baycom.org/~tom/ham/dcc2000/soundmodem.pdf, Accessed Apr. 2009.

[9] J. Olds, "J-QAM: A QAM soundcard modem," URL=http://homepages.paradise.net.nz/peterfr2/QAM.htm, Accessed Apr. 2009.

[10] C. Pelekanakis, M. Stojanovic, and L. Freitag, "High Rate Acoustic Link for Underwater Video Transmission," Proc. OCEANS 2003, Vol. 2, pp. 1091–1097, Sept. 2003.

[11] "GNU Radio: the gnu software radio," URL=http://gnuradio.org/trac/wiki, Accessed Apr. 2009.

[12] T. Fu, D. Doonan, C. Utley, R. Iltis, R. Kastner, and H. Lee, "Design and Development of a Software-Defined Underwater Acoustic Modem for Sensor Networks for Environmental and Ecological Research," Proc. OCEANS 2006, Sept. 2006.

[13] J.-K. Hwang, "Innovative communication design lab based on PC sound card and Matlab: a software-defined-radio OFDM modem example," Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Vol. 3, pp. 761-4, Apr. 2003.

[14] H. Yan, S. Zhou, Z. J. Shi, and B. Li, "A DSP Implementation of OFDM Acoustic Modem," Proc. *WUWNet'07*, Montreal, Sept. 2007.

[15] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Fourth Edition, Pearson: Upper Saddle River, 2007.

[16] Watkins-Johnson Company, "FSK: Signals and Demodulation, tech-notes," Vol. 7, No. 5, Sept./Oct. 1980.

[17] JRat, The Java Runtime Analysis Toolkit. URL=http://jrat.sourceforge.net, Accessed Apr. 2009.