Measurement-based Underwater Acoustic Physical Layer Simulation

Brian Borowski and Dan Duchamp
Stevens Institute of Technology
Department of Computer Science
Castle Point on Hudson
Hoboken, NJ 07030 USA

Abstract – **Simulation is important in the study of underwater networking because of the difficulty and expense of performing experiments. Underwater acoustic propagation is influenced by a wide variety of environmental factors, rendering analytic models complex, inaccurate, or both. Therefore, simulations based on models are of uncertain utility. In contrast, this simulator uses measured impulse response, CTD, noise, and transmission loss data in an effort to more realistically simulate the channel. The application layer generates data packets whose modulated waveforms are "mixed" with the channel's properties and sent to a receiver implemented fully in software, where the simulated bit error rate (BER) is measured. For a shallow time-invariant test channel, this process results in a simulated BER that is, on average, within 3.34% of the true BER.**

I. Introduction

Work on underwater networking is hampered by the difficulty and expense of performing experiments. One alternative is analytical modeling. Unfortunately, underwater propagation of acoustic waves is influenced by a wide variety of environmental factors, rendering analytic models complex, inaccurate, or both. An illustration is provided by Figures 1-3 in Guerra et al. [1], which show large differences in signal propagation in the same body of water during winter and summer and how both measurements differ drastically from predictions made by a standard model. Because of the difficulty of experimentation and the uncertainty of current state-of-the-art models, simulation is a vital tool in the study of underwater networking.

Work in underwater acoustic simulation is just beginning, and has so far developed primarily in two directions. One line of work focuses on network issues. Network simulators [2, 3] are commonly based on ns-2 [4] and use simple analytic models [5, 6] to develop an estimate of whether a packet traveling from node A to node B could be considered to be received based on the SNR or SINR computed by average-case formulas at B after accounting for the effects of attenuation, noise, and interference from other transmissions. The fidelity of this approach is limited by the accuracy of simple average-case models, which are typically derived from deep-ocean data. The second line of work includes physical layer simulators that seek to accurately model the effect of the surroundings. For example, Guerra et al. integrate the BELLHOP ray tracer with ns-2. Their ray tracing approach is driven by sound speed profile, bathymetric, and floor sediment data taken from well known databases that describe ocean points around the world [1]. In contrast, Xie et al. dismiss ray tracing as computationally impractical and use an approximation of the sound "contour" produced by the Monterey-Miami Parabolic Equation [7]. The goal of such physical modeling studies is also to compute SNR or SINR at reception points. Physical models are potentially more accurate than formulas that compute averages; however, physical models are much more computationally expensive and are, after all, still only models.

The work described in this paper presents a third approach: physical layer simulation that is driven by standard measurements taken from the environment of interest, including impulse response, CTD, noise, and transmission loss. The basic idea is to simulate an underwater channel based on its measurements in order to recreate the distortion that the channel would impose on a digital communication signal. Our approach does require in-water experiments, but only to take comparatively simple noise and impulse response measurements, not to gather detailed bathymetric and floor sediment data.

II. Simulator

Measurements form the basis of the simulation. CTD data yield the sound velocity profile of the channel, which is then used to calculate the propagation delay. Packets are converted into modulated waveforms which are convolved with the measured channel impulse response. Then recorded noise is added and a formula for transmission loss applied to approximate the SNR with which a packet is received. Finally, the distorted signal is demodulated and its BER is computed.

Figure 1 depicts the architecture of the simulator, which is written in a modular fashion using the OMNeT++ 4.0 discrete event simulator [8] and MATLAB so that any processing block can easily be extended or replaced. The application and link layers are implemented as objects within OMNeT++. While these layers are outside our current interest, they exist to form a complete network stack whose functionality could be extended in future work. The current application layer simply creates and receives messages. The transport layer has not been implemented. The link layer computes and checks checksums; there is no MAC protocol. The PHY layer and underwater acoustic channel model are implemented outside the OMNeT++ framework in MATLAB and exported as a shared library that OMNeT++ links in. OMNeT++ was chosen as the platform for our simulation because of its clean ar-

chitecture and relative ease of development. Other more sophisticated platforms, such as OPNET and ns-2, were considered and eliminated. Unlike these other platforms, OMNeT++ is only a discrete event simulator. It does not include default algorithms that the user must override, modify, or parameterize, and that may interact in ways that may not be intuitive. Since we use MATLAB to perform the heavy duty signal processing (modulation, convolution, and demodulation), a simple modular framework for each layer of the stack is all that we need.

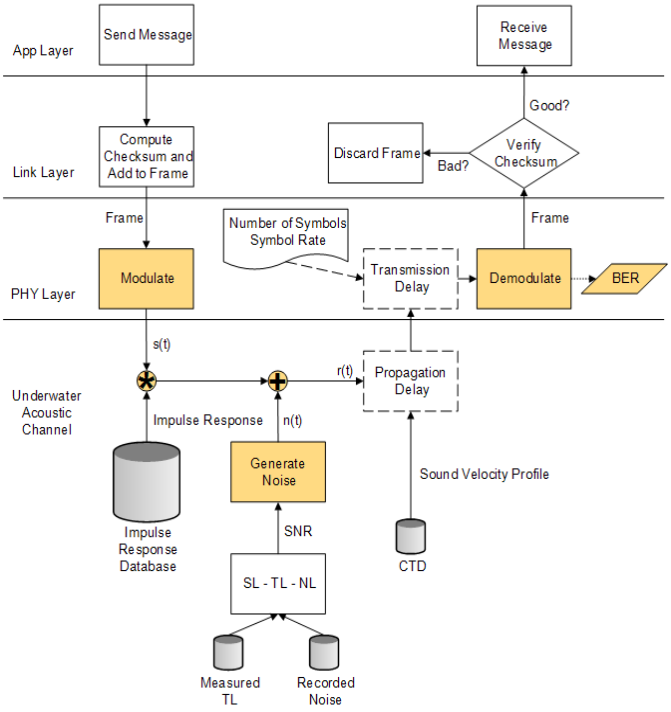

Figure 1. Architecture of OMNeT++ simulation for PHY layer and underwater acoustic channel. Areas in gray represent data from measurements. Areas in yellow are implemented in MATLAB.

Unlike other acoustic network simulators, our work includes a complete software modem, developed as part of our prior work [9]. The PHY layer transmitter and receiver code manipulates sampled signals just as in a real digital communication system, to show how a particular PHY implementation would work for the channel. This form of simulation accurately captures channel behavior while bypassing the need for expensive model-based ray tracing, and it results in more accurate BERs than formula-based, average-case BER computations based on SINR. Thanks to the embedded software modem, the simulator currently offers PSK and FSK modulation and three receiver techniques (a correlation receiver for PSK waveforms, and for FSK, a quadrature receiver or bandpass filters followed by envelope detectors). It is also possible to adjust the sampling rate, carrier frequency, and symbol rate as well as many other configuration parameters [9].

A. *Propagation and Transmission Delay*

The speed of sound in water is often estimated to be 1500 m/s; however, our simulation aims to be more accurate. The sound velocity profile is computed using CTD data and Medwin's expression for sound velocity in meters/second:

$$c = 1449.2 + 4.6T - 5.5 \times 10^{-2}T^2 + 2.9 \times 10^{-4}T^3 + (1.34 - 10^{-2}T)(S - 35) + 1.6 \times 10^{-2}D, \quad (1)$$

where D is the depth in meters, S is the salinity in parts per thousand (ppt), and T is the temperature in degrees Celsius [6].

The velocity values obtained over the water column are averaged. This average rate is then used to calculate the approximate propagation delay with the standard *time = distance / rate* formula. Transmission delay is the amount of time it takes to send all of the packet's bits into the channel. It is given by the formula $D_T = N / R$, where D_T is the transmission delay, N is the number of symbols, and R is the rate of transmission in symbols per second.

B. *Transmission Loss*

Transmission loss is based on a measurement taken in the shallow Hudson estuary [10]. The authors estimated the attenuation coefficient α at 0.058 dB/m between 10 – 80 kHz, yielding the following formula for transmission loss:

$$TL = 10\log(r) + \alpha r = 10\log(r) + 0.058r, \quad (2)$$

where r is the distance over which the loss is being calculated. The simulation uses this equation to estimate transmission loss, but any other method could be easily substituted.

C. *Noise*

Noise levels were recorded in the Hudson River estuary under different conditions ranging from ambient noise to a small speed boat to a large tug and barge [11]. The simulation chooses a noise level randomly for each packet that is sent. The simulation determines the average noise level in the frequency band of the modulated signal. For instance, binary FSK signaling at 1000 symbols per second typically requires 2 kHz of bandwidth for good performance with a noncoherent receiver. Therefore, if the carrier frequency were 50 kHz, the simulation would find the average noise level in dB from 49–51 kHz. Since the values are given in dB, they must converted to linear scale, averaged, and then converted back to dB scale, as in

$$dB_{avg} = 10\log_{10}\left((1/n)\sum_{1}^{n} 10^{(L/10)}\right), \quad (3)$$

where L is the level of a particular frequency component in dB.

D. *Modulation*

The simulation can generate binary FSK or PSK waveforms. With binary FSK, two tones are used. Since the FSK receiver implements noncoherent demodulation, the tones

must be separated by the number of Hz equal to the symbol rate. So, at 2000 bits per second, the tones must be separated by 2000 Hz. More formally, modulation index k is set to 1. The modulation index is defined by the formula

$$k = 2f_d/R, \tag{4}$$

where f_d is the frequency deviation in Hz (½ the separation between the two tones) and R is the data rate in symbols per second. As with other inexpensive noncoherent FSK systems, a modulation index of 1 is required to obtain reasonable receiver performance [12]. Continuous-phase FSK, or CPFSK, has been implemented as it is more bandwidth-efficient than traditional FSK that breaks phase at the start of each symbol. Figure 2 shows the MATLAB code to generate a CPFSK signal.

```
%% Performs CPFSK modulation.
%  data is vector of numberOfBits zeros and ones.
samplingRate = 200000;
symbolsPerSecond = 1000;
samplesPerBit = floor(samplingRate / ...
                      symbolsPerSecond);
carrierFreq = 80000;
fc = [(carrierFreq - symbolsPerSecond / 2) ...
      (carrierFreq + symbolsPerSecond / 2)];
txFSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    for j = 1:samplesPerBit
        t = t + 2 * pi * fc(data(i)+1) / ...
            samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        txFSK(j + samplesPerBit*(i-1)) = cos(t);
    end
end
```

Figure 2. MATLAB code to generate a CPFSK waveform.

The MATLAB code to generate a BPSK signal, shown in Figure 3, is quite similar in structure to the code for FSK. Note that t is reset to 0 before generating the next symbol. This step is necessary so that any carrier frequency can be used and successfully demodulated by a simple correlation receiver. If this step were not present, the symbol rate would need to evenly divide the carrier frequency, which in turn would need to evenly divide the sampling rate so that there would be an even number of periods of the sinusoid in each symbol and, therefore, phase transitions would occur only 180° apart at $y = 0$.

```
%% Performs PSK modulation.
%  data is vector of numberOfBits zeros and ones.
samplingRate = 200000;
symbolsPerSecond = 1000;
samplesPerBit = floor(samplingRate / ...
                      symbolsPerSecond);
carrierFreq = 80000;
txPSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    offset = samplesPerBit * (i-1);
    for j = 1:samplesPerBit
        t = t + 2 * pi * carrierFreq / samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        if (data(i) == 0)
            txPSK(j + offset) = -cos(t);
        else
            txPSK(j + offset) = cos(t);
        end
    end
    t = 0.0;
end
```

Figure 3. MATLAB code to generate a PSK waveform.

E. Channel Emulation

```
%% Extracts impulse estimates to individual wav
%  files.
seconds = 0.005;
len = seconds * samplingRate;
impulseResponse = zeros(numOfImpulseResponses, len);
for i = 1:numOfImpulseResponses
    snip = recordedSignal( ...
       (i-1)*referenceSamples+1:i*referenceSamples);
    temp = fftshift(real( ...
        xcorr(snip, conj(referenceSignal))));
    [maxValue maxIndex] = max(temp);
    earlyPeakIndex = find(temp(max( ...
      maxIndex - 0.0005*samplingRate, 1):end) > ...
      0.25*maxValue);
    [mainPeak mainPeakIndex] = max(temp(max( ...
       maxIndex - 0.0005*samplingRate, 1):end));
    offset = mainPeakIndex - earlyPeakIndex;
    temp = temp(maxIndex - offset:...
                maxIndex - offset + len);
    impulseResponse(i,:) = temp(1:len);
end
[maxVal maxIndex] = max(max(abs(impulseResponse)));
impulseResponse = 0.98 * impulseResponse / maxVal;
for i = 1:numOfImpulseResponses
    filename = sprintf('IR_200m/IR_%d', i);
    wavwrite(impulseResponse(i,:), samplingRate, ...
             filename);
end
```

Figure 4. MATLAB code to extract impulse estimates to individual wav files.

Impulse response estimates form the heart of the simulation. The database shown in Figure 1 is implemented as sets of wav files. The code in Figure 4 shows how the impulse estimates have been extracted. There are two noteworthy aspects of this code segment. The first is that each impulse response must be "cropped" to contain only significant multipath components. Any value that is within 6 dB (0.25) of the strongest arrival's intensity are included in the estimate, as it contains relevant information about the delay spread and frequency response of the channel at that moment. The second interesting point is that the impulse responses can be normalized only after all of them have been extracted. This way, the amplitudes of the estimates obtained during a deep fade are not artificially increased to equal those of estimates obtained during periods with high SNR. While this feature is not exploited in the current simulation, it is beneficial to create the database with

measurements that are as accurate as possible for future extensions.

When a frame is passed through the simulated channel, it is convolved with one randomly chosen impulse response estimate. As the lengths of the input vectors grow, the execution time of convolution performed in the time domain grows quadratically. Therefore, to improve the performance of the simulation for long data frames and/or delay spreads, FFT convolution is implemented. Convolution in the time domain corresponds to multiplication in the frequency domain. Thus, in FFT convolution the input signal is transformed into the frequency domain using FFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the inverse FFT [13]. The complexity of FFT convolution is $O(n \log n)$, a big improvement over $O(n^2)$.

After the convolution routine is finished executing, noise is added to the signal so that the resulting SNR = SL – TL – NL, where SL is the user-specified source level, TL is the transmission loss described in Section II.B, and NL is a randomly chosen noise level described in Section II.C. The noise added to the signal is AWGN. This is a reasonable simplification, since the simulation is concerned only with noise in the frequency band of the data transmission, which occupies a small amount of space on the acoustic spectrum.

F. Demodulation

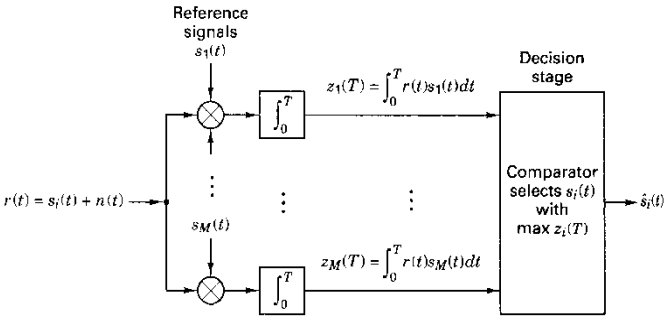

Figure 5. Correlation receiver with M reference signals. Drawing from [15].

The simulator can demodulate PSK and FSK signals. A correlation receiver is used to demodulate PSK waveforms. It works by dividing the incoming stream into small sections with the duration of an individual symbol. Each section is multiplied by the reference signals of the same length that represent the symbols '0' and '1'. The results are then summed over the symbol duration, and finally the comparator chooses the symbol whose sum was the greatest. Figure 5 shows the block diagram for a correlation receiver with M reference signals. Note that in the case of BPSK waveforms, only two reference signals are required. Figure 6 shows how to translate the block diagram into MATLAB code.

```
%% Demodulates PSK signal using a correlation
%  receiver.
t = 0:samplesPerBit-1;
psk0 = -cos(2 * pi * carrierFreq/samplingRate * t);
psk1 =  cos(2 * pi * carrierFreq/samplingRate * t);
rxPSK = zeros(1, numberOfBits);
for i = 1:numberOfBits
    rcv = packet((i-1)*samplesPerBit + 1: ...
                 i*samplesPerBit);
    zero = rcv .* psk0;
    one = rcv .* psk1;
    z0 = sum(zero);
    z1 = sum(one);
    rxPSK(i) = (z1 > z0);
end
```

Figure 6. MATLAB code implementing a correlation receiver for PSK signals.

While the PSK waveform that was transmitted originally contained '0' and '1' reference signals of $-\cos(2\pi ft)$ and $\cos(2\pi ft)$, where f is the carrier frequency and t is time, the phase of received waveform may be distorted so that transitions no longer occur at 0° and 180°. For instance, perhaps they now occur at 25° and 205°. In order to improve the performance of the correlation receiver, it would be necessary to estimate these offsets. A simple solution, the one implemented in the simulator, is to send training symbols of alternating ones and zeros at the beginning of the signal. The receiver can first try various offsets to find the one that maximizes the correlation of the known symbols. Since the model currently convolves a single impulse response with the data signal, this type of synchronization is adequate. However, when communicating through and modeling real time-variant channels, more sophisticated methods are necessary. A receiver that jointly performs carrier synchronization and fractionally spaced decision feedback equalization of the received signal, and whose parameters are adaptively adjusted using a combination of the RLS algorithm and second-order digital PLL, was shown to perform well in short-range shallow water of 2 nautical miles at 10 kbps [14]. Since the simulator is written in modular fashion, one could write the code for such a receiver in MATLAB and incorporate it into model with minimal effort.

Noncoherent detection of FSK waveforms is implemented via both the quadrature receiver and bandpass filters/envelope detectors. Each receiver is also able to employ a hard limiter [16] as the first stage of demodulation process. This procedure places a cap on high-amplitude samples and raises low-amplitude samples to the value of the cap. Utilization of a hard limiter helps equalize the signal so that each tone in the FSK signal has the same amplitude before it is passed to subsequent stages of the receiver. This is especially useful when working with transducers that do not have a flat frequency response. Two correlators are used for each tone, one for the in-phase (I) and quadrature (Q) channels. Since measurements of the signal's phase cannot be exploited, the receiver is merely an energy detector. By summing the squares of the correlation on each channel, the same values will be fed to the decision stage if either channel had full correlation while the other had none or if the incoming signal partially correlated with both references. Figure 7 shows the block diagram for a quadrature receiver; Figure 8 shows how to translate the block diagram into MATLAB code.

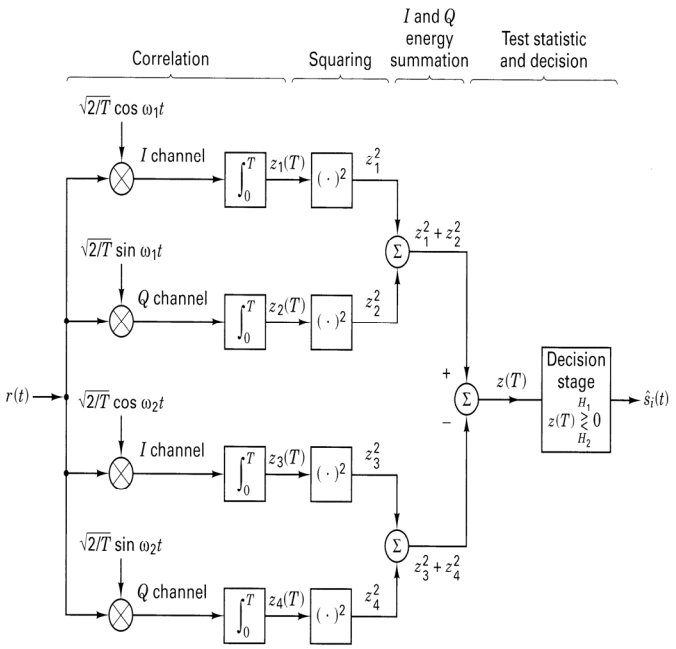

Figure 7. Quadrature receiver for non-coherent detection of FSK signals [15].

```
function [y] = hardlimit(x)
%HARDLIMIT limits the amplitude of a signal.
%   y(t) = g[x(t)] = {+1, 0, -1}
%          if x(t) {> 0, = 0, < 0}.
    y = 2 * ((x > 0) - 0.5);

%% Demodulates FSK signal using a quadrature
%  receiver.
fc = [(carrierFreq - symbolsPerSecond/2) ...
      (carrierFreq + symbolsPerSecond/2)];
t = 0:samplesPerBit-1;
cos0 = cos(2 * pi * fc(1)/samplingRate * t);
sin0 = sin(2 * pi * fc(1)/samplingRate * t);
cos1 = cos(2 * pi * fc(2)/samplingRate * t);
sin1 = sin(2 * pi * fc(2)/samplingRate * t);
rxFSK_q = zeros(1, numberOfBits);
for i = 1:numberOfBits
    if (useLimiter)
        rcv = hardlimit( ...
                packet((i-1)*samplesPerBit + 1: ...
                       i*samplesPerBit));
    else
        rcv = packet((i-1)*samplesPerBit + 1: ...
                     i*samplesPerBit);
    end
    Izero = rcv .* cos0;
    Qzero = rcv .* sin0;
    Ione = rcv .* cos1;
    Qone = rcv .* sin1;
    z1 = (sum(Izero))^2;
    z2 = (sum(Qzero))^2;
    z3 = (sum(Ione))^2;
    z4 = (sum(Qone))^2;
    energy0 = z1 + z2;
    energy1 = z3 + z4;
    rxFSK_q(i) = (energy1 > energy0);
end
```

Figure 8. MATLAB code for implementing a quadrature receiver for FSK signals.

The simulation also implements noncoherent FSK detection with bandpass filters followed by envelope detectors, as shown in Figure 9. When using binary FSK, the incoming signal is passed through two separate filters ($M = 2$ in Figure 9) to eliminate signals outside the band used by each tone. Second-order IIR filters based on the design in [13] are used. For a second-order filter, the best performance is obtained when the product BT is close to 1.0, where B is the -3dB bandwidth in Hz and T is the duration of a symbol [17]. Therefore, the filters have been designed to operate with bandwidth $1/T$ centered on the tones representing 0s and 1s.

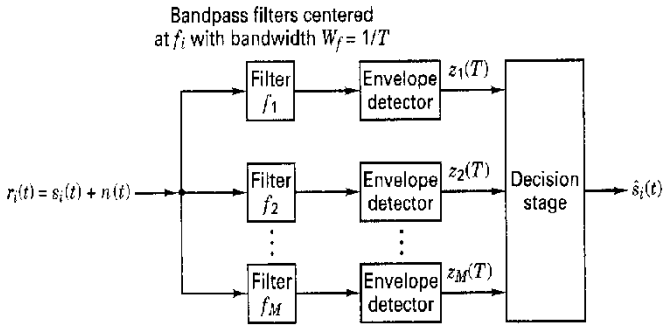

Figure 9. Noncoherent detection of FSK signals using bandpass filters and envelope detectors [15].

Upon completion of the filtering stage, the resulting signals are passed to the envelope detectors, each of which first applies the Hilbert transform to obtain the imaginary part x_i of a signal containing only real values x_r. The envelope is then obtained by taking the magnitude of the signal. Finally, for each symbol, the envelopes are summed over several samples, and the symbol corresponding to the greatest sum is outputted by the decision stage. Figure 10 shows how to translate the block diagram in Figure 9 into MATLAB code.

```
%% Demodulates FSK signal using a bandpass filters
%  and envelope detectors.
fc = [(carrierFreq - symbolsPerSecond/2) ...
      (carrierFreq + symbolsPerSecond/2)];
if (useLimiter)
    zeroSignal = abs(hilbert(filterSignal( ...
       hardlimit(packet), samplingRate, ...
       fc(1), symbolsPerSecond)));
    oneSignal = abs(hilbert(filterSignal( ...
       hardlimit(packet), samplingRate, ...
       fc(2), symbolsPerSecond)));
else
    zeroSignal = abs(hilbert(filterSignal( ...
       packet, samplingRate, fc(1), ...
       symbolsPerSecond)));
    oneSignal = abs(hilbert(filterSignal( ...
       packet, samplingRate, fc(2), ...
       symbolsPerSecond)));
end
diff = oneSignal - zeroSignal;

rxFSK_e = zeros(1, numberOfBits);
for i = 1:numberOfBits
    % Sample in second half of symbol to avoid ring
    % in IIR filter.
```

```
    rcv = sum(diff((i-1)*samplesPerBit + ...
          floor(samplesPerBit/2):i*samplesPerBit));
    rxFSK_e(i) = (rcv > 0);
end
```

Figure 10. MATLAB code implementing a receiver for FSK signals with bandpass filters and envelope detectors.

III. EVALUATION

In order to validate the measurement-based PHY layer simulator, test signals were transmitted through a time-invariant channel in a shallow test tank. The simulation would be perfect if frames transmitted through the channel have the same bit errors as when convolved with the impulse response of the channel by the simulator. The difference between the errors of the transmitted frame and the errors of the simulated frame provide a measure of the inaccuracy of the simulation approach for this channel.

Thirty composite test signals were generated and transmitted through the tank. Binary FSK and PSK signals were transmitted at five different bit rates to cover cases with and without channel-induced ISI. Communication was tested in three frequency bands – 7.5 kHz, 12.5 kHz, and 17.5 kHz – that evenly divided and collectively spanned the usable channel bandwidth. The sampling rate for all signals was 48 kHz over the channel bandwidth of 0-24 kHz.

TABLE I
BIT RATES TESTED AT EACH CARRIER FREQUENCY

Frequency (Hz)	Bit Rate (bps)				
7,500	250	500	1250	2500	3750
12,500	250	500	1250	2500	3125
17,500	250	500	1250	2500	3500

Each composite signal started with a 5-second LFM chirp over the entire channel bandwidth that was used to estimate the system's impulse response at the beginning of that particular test. Modulated waveforms followed the long chirp signal. Fifty duplicate packets, each beginning with a short chirp signal for synchronization purposes and separated from the next by 1 second of silence, were transmitted in all tests except for those running at only 250 bps, where because of the slow rate only twenty duplicate packets were transmitted. Table I summarizes the bit rates tested at each frequency. Only bit rates that even divided the carrier frequency were used in this experiment. Figures 11 and 12 show the time and frequency domain views of the recorded chirp signal and FSK-modulated packets during one of the fifteen tests.

The FSK-modulated packets were demodulated with the envelope detector and quadrature receiver, both with and without the hard limiter, as described above. Detection of PSK packets was accomplished via the correlation receiver. The BER of each packet was computed, and then the average BER for that trial was calculated.

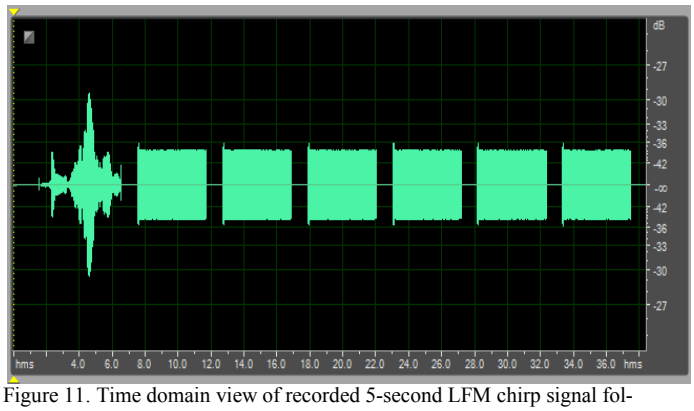

Figure 11. Time domain view of recorded 5-second LFM chirp signal followed by FSK-modulated packets at 500 bps with a 12.5.kHz carrier.

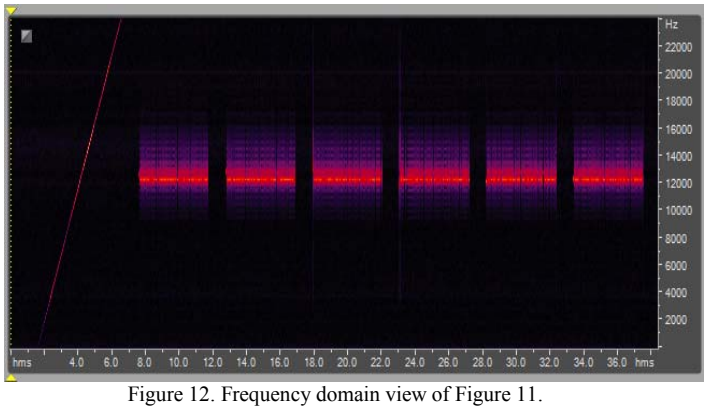

Figure 12. Frequency domain view of Figure 11.

The simulated BER was obtained by first estimating the channel's impulse response. The impulse response was obtained by cross-correlating the received signal with the complex conjugate of the reference signal, which in this case is the 5-second LFM chirp signal at start of each trial. The impulse response was then convolved with the signal containing the waveform of a single modulated packet. Note that only one packet is needed here, since only one impulse response estimate was obtained at the start of the test. In theory, because the tank is a time-invariant environment, the estimated impulse response should remain nearly constant over the duration of a trial.

Tables II through V summarize the comparison of BERs obtained via simulation versus actual transmission. The percent difference is defined merely as the difference between the two BERs. The standard formula for percent error cannot be applied, since it requires the measured and expected values to be greater than zero and there are several cases where the difference is zero. The average difference is approximately 3.34%. Zero-difference instances appear mostly at the slower bit rates, where the symbol duration easily exceeds the channel's delay spread. There are also a few cases of large discrepancies (greater than 20%) appearing in seemingly random individual tests.

TABLE III
OVERALL COMPARISON OF BERS OBTAINED WITH TRANSMISSION VERSUS SIMULATION

Overall % Difference			
Average	Min	Max	Std. Dev.
3.34	0.00	33.31	8.21

TABLE III
COMPARISON OF BERS OBTAINED WITH TRANSMISSION VERSUS SIMULATION, GROUPED BY CARRIER FREQUENCY

	% Difference			
Frequency (Hz)	Average	Min	Max	Std. Dev.
7,500	1.74	0.00	12.47	3.35
12,500	3.66	0.00	33.31	8.16
17,500	4.61	0.00	27.69	8.11

TABLE IV
COMPARISON OF BERS OBTAINED WITH TRANSMISSION VERSUS SIMULATION, GROUPED BY THE TYPE OF MODULATION-DEMODULATION

			% Difference			
Mod	Demod	Implementation	Avg	Min	Max	Std Dev
FSK	Envelope Detector	Amplitude Comp.	2.60	0.00	20.16	5.22
		Hard Limiter	3.34	0.00	27.69	7.28
	Quadrature Receiver	Default	4.35	0.00	24.41	7.36
		Hard Limiter	2.89	0.00	24.31	6.14
PSK	Correlator	N/A	3.51	0.00	33.31	9.01

TABLE V
COMPARISON OF BERS OBTAINED WITH TRANSMISSION VERSUS SIMULATION, GROUPED BY BIT RATE

	% Difference			
Bit Rate (bps)	Average	Min	Max	Std. Dev.
250	0.00	0.00	0.00	0.00
500	2.03	0.00	24.41	6.29
1250	7.50	0.00	27.69	9.72
2500	3.91	0.00	13.25	4.21
>3000	3.25	0.00	33.31	8.41

The amount of inaccuracy increases with the carrier frequency. Since there should be no correlation between frequency and inaccuracy, we investigated our observation. The frequency response of the impulse response estimate obtained at the start of each test can be computed using an FFT algorithm, which is accomplished easily with MATLAB's *freqz* function. Figure 13 shows the impulse response obtained during one of the tests. Figure 14 shows the frequency and phase response of the system derived from the impulse response in Figure 13. Over the course of the entire experiment, the impulse response and corresponding frequency and phase responses remained nearly constant. Upon viewing Figure 14 it is clear why there are more errors at 12.5 kHz and 17.5 kHz than at 7.5 kHz. There are dips of more than 20 dB in the frequency response centered on 12.7 kHz and 17.5 kHz, which naturally give rise to drastic changes in the phase response at those frequencies. While the impulse response estimates do capture these properties, it is clear that they are not perfectly accurate, as the simulated bit error rates deviate from the measured values.

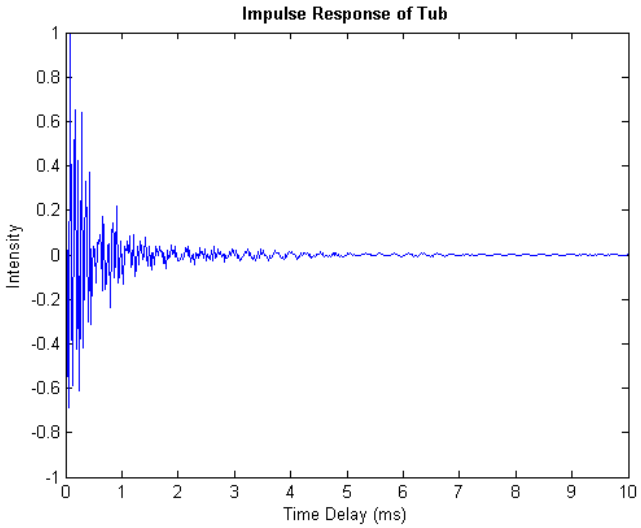

Figure 13. Impulse response of test tank during validation experiment.

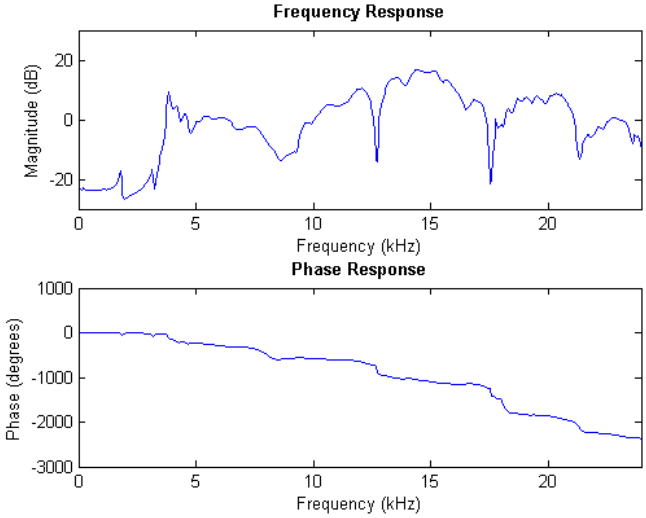

Figure 14. Frequency and phase response of test tank, derived from impulse response in Figure 13.

There is no obvious conclusion to draw about the discrepancies between the simulated and measured values grouped by the modulation-demodulation routine. The average difference is within a few percent for all tests, though PSK demodulation

does possess the worst-case value of the experiment. There is, however, a trend seen with increasing bit rate. The simulator is quite accurate for slow data rates, where the symbol duration exceeds the delay spread of the channel. At 250 bps and 500 bps, except for the FSK trial at 12.5 kHz bps with the default quadrature receiver, the simulated and measured bit error rates are identical. For this particular case, it seems that the simulator exaggerates the dip in the frequency response at 12.7 kHz, since it yields a BER of about 24% while the BER for real packets was actually 0%. When using the impulse response, the tone representing a '1' is so much lower in amplitude than the tone representing a '0' that incorrect correlation produces a bit error in favor of '0' 50% of the time a '1' should be present, resulting in the simulated BER of 24%. Note that the particular sequence of bits used in this experiment contains a 50.4/49.6 mixture of zeros and ones, respectively. As an aside, this one case demonstrates the value of the hard limiter for FSK-based receivers. At higher bit rates, the simulator's accuracy decreases. It produces the worst results at 1250 bps, when the symbol duration is roughly equal to the delay spread of the channel.

In conclusion, using impulse response measurements to simulate a time-invariant channel is very accurate when the symbol duration exceeds the multipath spread of the channel and no ISI exists. The accuracy drops slightly when the symbol time is less than the channel's delay spread. This method, however, becomes error-prone when the symbol duration is very close to the length of the delay spread, since a small discrepancy between the estimated and actual channel responses can lead to very different results at the receiver. The simulator seems to perform well for both FSK and PSK signals transmitted at various carrier frequencies, though it works best when the system has linear frequency and phase response. Unfortunately, we were unable to test the simulator's accuracy for time-variant channels such as the Hudson River estuary. The model might be too simplistic to generate accurate BERs without the inclusion of varying impulse response estimates and fading.

IV. CONCLUSION

Using impulse response measurements to simulate a time-invariant channel is very accurate when the symbol duration exceeds the multipath spread of the channel and no ISI exists. The accuracy drops slightly when the symbol time is less than the channel's delay spread. This method, however, becomes error-prone when the symbol duration is very close to the length of the delay spread, since a small discrepancy between the estimated and actual channel responses can lead to very different results at the receiver. The simulator performs well for both FSK and PSK signals transmitted at various carrier frequencies.

REFERENCES

[1] F. Guerra, P. Casari, and M. Zorzi, "World Ocean Simulation System (WOSS): A Simulation Tool for Underwater Networks," in Proc. WUWNet'09, Nov. 2009.

[2] A. F. Harris III and M. Zorzi, "Modeling the Underwater Acoustic Channel in ns2," NSTools '07, Nantes, France, Oct. 2007.

[3] P. Xie et al., "Aqua-Sim: An NS-2 Based Simulator for Underwater Sensor Networks," in Proc. MTS/IEEE Oceans, Biloxi, Oct. 2009.

[4] The Network Simulator – ns-2, Accessed online: http://nsnam.isi.edu/nsnam/index.php/User_Information, May 2010.

[5] L. M. Brekhovskikh and Y. P. Lysanov, *Fundamentals of Ocean Acoustics,* Springer 1982.

[6] R.J. Urick, *Principles of Underwater Sound*, 3rd ed., McGraw-Hill, 1996.

[7] G. Xie, J. Gibson, and L. Diaz-Gonzalez, "Incorporating Realistic Acoustic Propagation Models in Simulation of Underwater Acoustic Networks: A Statistical Approach," in Proc. MTS/IEEE Oceans, Sept. 2006.

[8] *OMNeT++ Community Site*, Accessed online: http://www.omnetpp.org, Mar. 2010.

[9] B. Borowski and D. Duchamp, "The Softwater Modem – A Software Modem for Underwater Acoustic Communication," in Proc. WUWNet'09, Nov. 2009.

[10] H.-S. Roh, A. Sutin, and B. Bunin, "Determination of acoustic attenuation in the Hudson River Estuary by means of ship noise observation," JASA Express Letters, May 2008.

[11] B. Borowski, A. Sutin, H.-S. Roh, and B. Bunin, "Passive Acoustic Threat Detection in Estuarine Environments," in Proc. of SPIE Vol. 6945, March 2008.

[12] R. Marrow, *Bluetooth Operation and Use*, McGraw-Hill Professional, 2002.

[13] S. Smith, *Digital Signal Processing, A Practical Guide for Engineers and Scientists*, Newnes, 2003.

[14] M. Stojanovic, J.A. Catipovic, and J.G. Proaxis, "Phase-Coherent Digital Communications for Underwater Acoustic Channels," Trans. IEEE Journ. Oceanic Engineering, 19(1), Jan. 1994.

[15] B. Sklar, *Digital Communications: Fundamentals and Applications*, 2nd ed., Prentice Hall, 2001.

[16] J. Jones, "Hard-Limiting of Two Signals in Random Noise," IEEE Trans. on Information Theory, 9(1):34-42, Jan. 1963.

[17] Watkins-Johnson Company, "FSK: Signals and Demodulation, technotes," 7(5), Sept./Oct. 1980.

ACKNOWLEDGMENT

This work was supported in part by a Stevens Institute of Technology Stanley Graduate Fellowship. The authors express their thanks to the personnel of the Stevens Maritime Security Laboratory, who provided vital help in taking and interpreting measurements in the Hudson.