Mitigating Intersymbol Interference through Equalization

Xiangling Kong, Jeffrey Shih, and Xiaowen Zhang

***Abstract*—This paper discusses the procedures involved in mitigating intersymbol interference of multipath channels through adaptive equalizers and reduction of bit error rate through convolutional error encoding. A combination of both techniques is enough to reduce the bit error rate down to the target of 10^{-6}.**

I. Introduction

One common problem faced by transmission systems is distortion due to multipath channels. A multipath channel is a channel where a delayed symbol arrives at the receiver (due to different paths) and interferes with symbols received along other paths. This is especially problematic in wireless communications as reflections of a transmitted signal off of any surface will create intersymbol interference and distort the signal. In this simulation, intersymbol interference is modeled as the convolution of the original system with a channel representing the amplitude of each delayed signal. Noise is assumed to be AWGN at 12 dB SNR. There are two primary objectives: mitigate intersymbol interference in order to achieve less than 10^{-6} bit error rate and maximize bit rate while preserving the error rate.

II. Main Results

There are a total of three parts that must be completed. First, we must match the performance of 4 QAM and 16 QAM with the theoretical error rate for 0 to 20 dB SNR without intersymbol interference. Second, we must achieve a bit error rate of less than 10^{-4} at 12 dB SNR over the provided moderate ISI channel described by the vector [1 .2 .4]. Third, we must improve the bit error rate to be less than 10^{-6} at 12 dB SNR while maximizing the bit rate through the same moderate channel.

A. Matching Theoretical Error Rate

A template was provided to simulate BPSK through the three provided channels. In order to match the error rate for 4 QAM or 16 QAM, a correction factor must be added to the channel of $10 \cdot \log_{10}(k)$ where k is $\log_2$ of the QAM number. Applying this correction factor to the channel and changing all the 2 (for BPSK) to 4 or 16 (for 4 QAM or 16 QAM) yields the expected graph as shown in **Figure 1** and **Figure 2**. For 4 QAM, the graph seems to cut off abruptly after 10 dB SNR. This is because after 10 dB SNR, it was very hard to get any errors using 4 QAM even after running 1000 iterations to attempt to get one error.

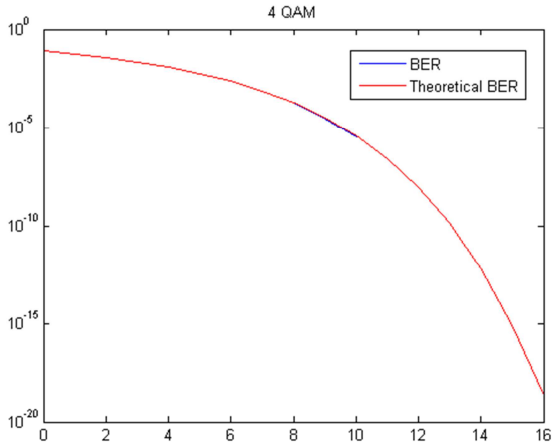

Fig. 1. Simulated BER vs. Theoritical BER for 4 QAM through an AWGN channel without intersymbol interference.

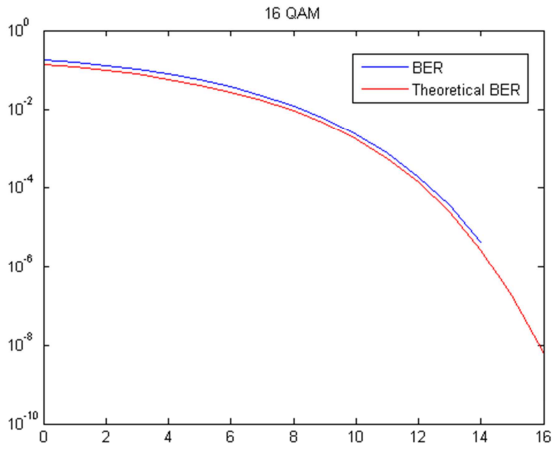

Fig. 2. Simulated BER vs. Theoritical BER for 16 QAM through an AWGN channel without intersymbol interference.

B. Achieving 10^{-3} BER on a Moderate ISI Channel

After enabling the ISI channel, the bit error rate became almost constant for 4 QAM. A basic linear equalizer is applied with 8 weights using the least mean squares adaptive algorithm with a step size of 0.01. Using about 200 training symbols, it was possible to reduce the bit error rate to 10^{-3} at 12 dB SNR. A scatter plot is generated of the effects of the equalizer is shown in **Figure 3**. Without the equalizer, at each time, the receiver sees more than one symbol due to the arrival of delayed symbols. After training on 200 initial symbols, the equalizer is able to accurately undo the intersymbol interference.

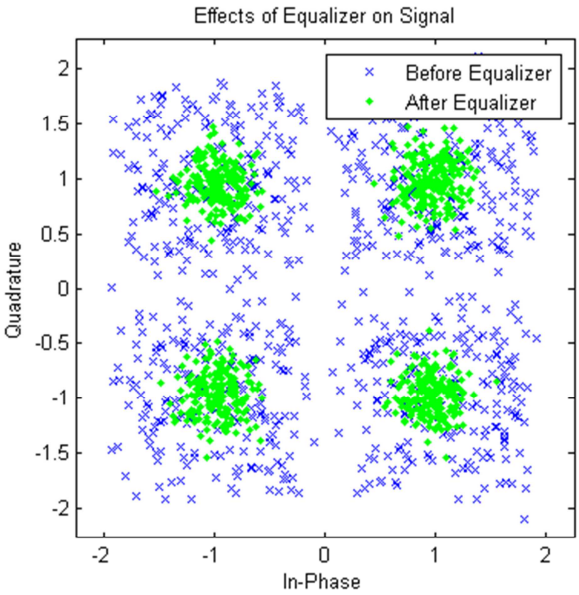

Fig. 3. Linear Equalizer with the Least Mean Squares adaptive algorithm on a moderate ISI channel.

C. Achieving 10^{-6} BER on a Moderate ISI Channel

Modifying and tweaking the parameters of the Equalizer did not greatly improve the BER. To bring the BER down to 10^{-6} a ½ convolutional encoder was used[1]. Since a convolutional encoder operates at the bit level, it must be applied before the bits are converted to symbols. Since a strong ½ encoder is used, this effectively cuts the bit rate in half as well. Using this scheme of combining a convolutional encoder and a linear equalizer, it was possible to reduce the bit error rate down to 10^{-6} with approximately 0.5 bits per symbol. Below are figures showing the BER plots for both moderate and severe ISI channels using BPSK and 4 QAM.

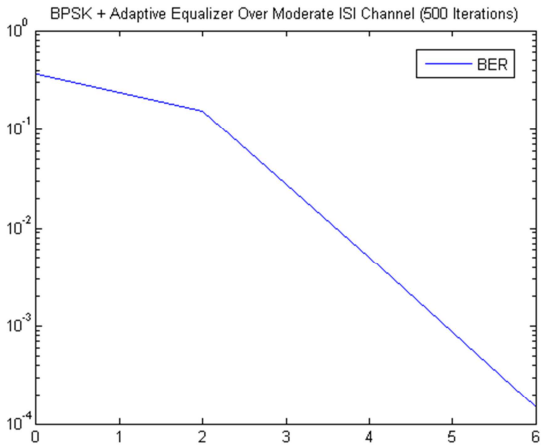

Fig. 4. BER for BPSK using convolutional encoding and adaptive equalizer over a moderate ISI channel with 500 iterations.

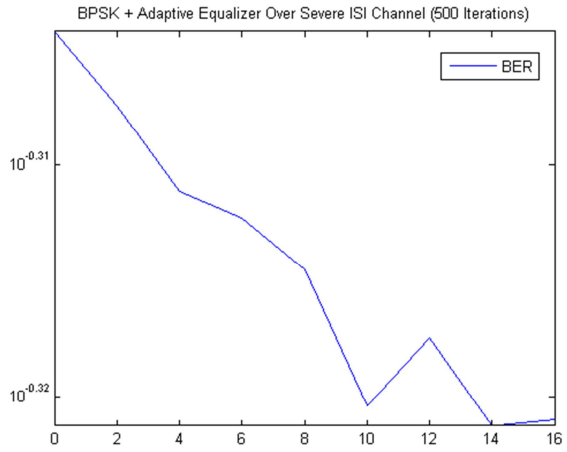

Fig. 5. BER for BPSK using convolutional encoding and adaptive equalizer over a severe ISI channel with 500 iterations.

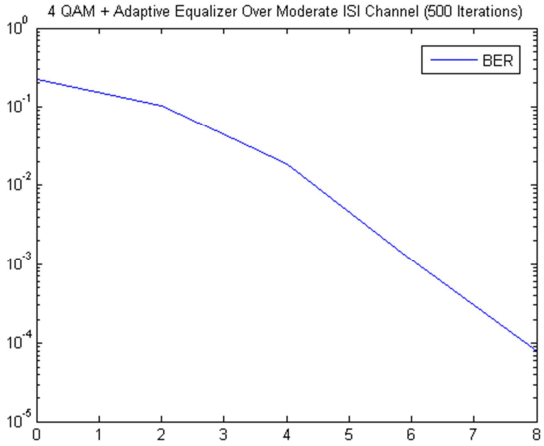

Fig. 6. BER for 4 QAM using convolutional encoding and adaptive equalizer over a moderate ISI channel with 500 iterations.

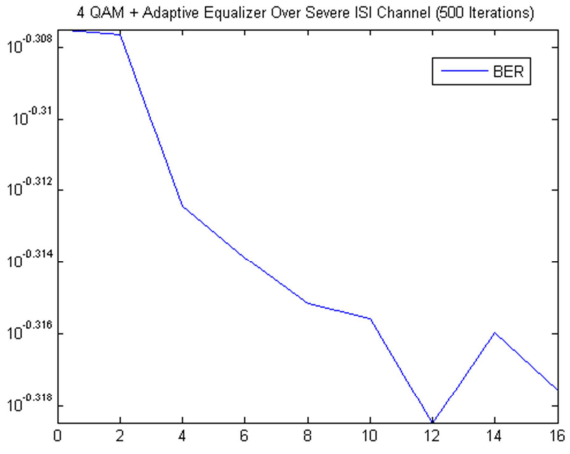

Fig. 7. BER for 4 QAM using convolutional encoding and adaptive equalizer over a severe ISI channel with 500 iterations.

In **Figure 4**, for BPSK, no errors were made beyond 6 dB SNR in the moderate ISI channel with 500 iterations and in **Figure 6**, for 4 QAM, no errors were made beyond 8 dB SNR. However, for both BPSK and 4 QAM, the severe channel

effects were irreversible under this system.

D. *Improving Bit Rate at 10^{-6} BER (Training Symbols)*

In order to improve the bit error rate, a variety of techniques were attempted. First, the number of training symbols was reduced from 200 down to 100. Anything below 100 resulted in an increase in BER beyond 10^{-6} and so 100 symbols were chosen as the minimum number of training symbols.

E. *Improving Bit Rate at 10^{-6} BER (Decision Feedback)*

Since having 100 training symbols is much higher than optimal, the linear equalizer was switched for a decision feedback equalizer. In addition, the algorithm was switched to recursive least mean squares since it has the fastest convergence rate. Under these conditions, the number of training symbols was able to be reduced to 25 while still keeping the 10^{-6} BER. This process was able to achieve 0.943 bits per symbol with a BER curve shown in **Figure 8**.

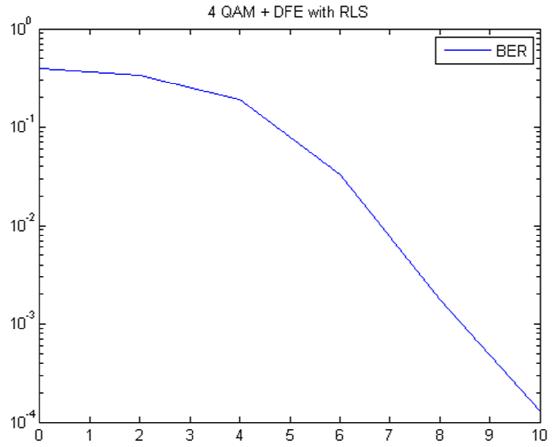

Fig. 8. BER for 4 QAM using decision feedback and recursive least mean squares algorithm over 1000 iterations.

F. *Improving Bit Rate at 10^{-6} BER (Encoder Rate)*

A new trellis is added in order to attempt to improve the bit rate. Since a ½ encoder is a very aggressive coding scheme, the 2/3 encoder was attempted to see if the error rate would remain the same. Switching the trellis in the convolutional encoder for a 2/3 encoder yielded the same SNR at 12 dB but gave a much higher bit rate. **Figure 9** demonstrates that although the bit error rate increased from zero to about 3.1447 x 10^{-6} since the modification from part E, the 2/3 encoder was able to raise the bit rate to 1.272.

Since convolution encoding wasn't performing as well, turbo coding was used instead. Lowering the turbo encoding rate to ½ allowed us to use 64 QAM instead of 16 QAM which provided 2 bits extra to transmit information. This process overall was able to raise the bit rate to 1.839 while keeping the error down to below 10^{-6} (in this case, no errors were made even after 1000 iterations).

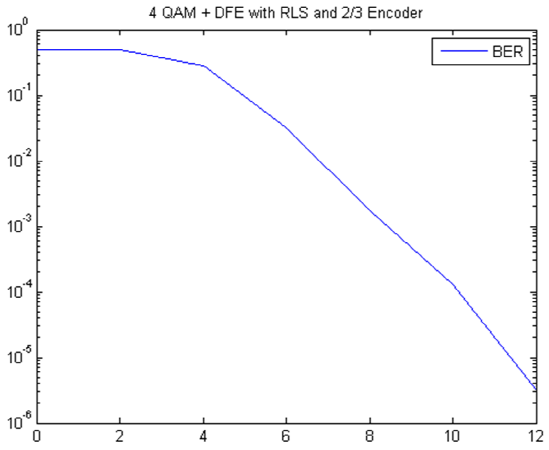

Fig. 9. BER for 4 QAM using decision feedback and recursive least mean squares algorithm over 500 iterations with 2/3 encoder.

III. Summary

In order to mitigate intersymbol interference, an equalizer must be used. To further reduce the bit error rate, an error encoder is required. However, the error encoder trades off bit error rate with bit rate and thus a compromise has to be reached. Combining an equalizer with an error encoder allows a system to resolve intersymbol interference as well as achieve low bit error rates.

Appendix

A. *Code for Correcting SNR*

```
% Project 2 BER Detector for 4 QAM and 16 QAM
format compact; clear all; close all; clc;

% Keene's Holy Constants
numIter = 1000;
nSym = 1000;
SNR_Vec = 0:1:16;
lenSNR = length(SNR_Vec);

% Xiangling's Holy Constants
M = 16;
k = log2(M);

berVec = zeros(numIter, lenSNR);
for i = 1:numIter
    bits = randi([0,1], 1, nSym * k);
    msg = bi2de(reshape(bits, k, length(bits) /
k).', 'left-msb')';
    for j = 1:lenSNR
        tx = qammod(msg, M);
        % Add Noise
        txNoise = awgn(tx, SNR_Vec(j) + 10*log10(k),
'measured');
        % Receive Data
        rxMSG = qamdemod(txNoise, M);
        rxBits = de2bi(rxMSG, 'left-msb');
        rxBits = reshape(rxBits.', numel(rxBits),
1).';
        [Discard, berVec(i,j)] = biterr(bits,
rxBits);
    end
end

% Display Plots
ber = mean(berVec, 1);
berTheory = berawgn(SNR_Vec, 'qam', M, 'nondiff');
```

```
figure
semilogy(SNR_Vec, ber);
hold on
semilogy(SNR_Vec, berTheory, 'r');
hold off
legend('BER', 'Theoretical BER');
title(strcat(num2str(M), ' QAM'));
```

B. *Code for Most Bit Rate at 10^{-6} Bit Error Rate*

```
% Project 2
format compact; clear all; close all; clc;

% Keene's Holy Constants
numIter = 1000;
nSym = 1000;
SNR_Vec = 12;
lenSNR = length(SNR_Vec);

% Xiangling's Holy Constants
M = 64;
k = log2(M);
train = 25;
% t = poly2trellis(7, [171 133], 171);
% t = poly2trellis([5 4], [23 35 0; 0 5 13]);
t = poly2trellis(4, [13 15], 13);

intrlvrIndices = randperm(round(nSym * k / 3));
hEnc = comm.TurboEncoder('TrellisStructure', t,
'InterleaverIndices', intrlvrIndices);
hDec = comm.TurboDecoder('TrellisStructure', t,
'NumIterations', 9, 'InterleaverIndices',
intrlvrIndices);
hEMod =
comm.RectangularQAMModulator('ModulationOrder', M,
...
    'BitInput', true);
hDMod =
comm.RectangularQAMDemodulator('ModulationOrder', M,
...
    'BitOutput', true, ...
    'DecisionMethod', 'Log-likelihood ratio');

% Pick a Channel
% chan = 1; % No Channel (Aww Yes Best Channel)
chan = [1 .2 .4]; % Moderate ISI (Looks Decent)
% chan = [0.227 0.460 0.688 0.460 0.227]; % Severe
ISI (Looks Nasty. Ewww)

% Danger! Do Not Uncomment Below
%
==================================================
====================
% ts = 1/1000;
% chan = rayleighchan(ts, 1);
% chan.pathDelays = [0 ts 2*ts];
% chan.AvgPathGaindB = [0 5 10];
% chan.StoreHistory = 1;
%
==================================================
====================

berVec = zeros(numIter, lenSNR);
brVec = zeros(numIter, lenSNR);
hw = waitbar(0, 'Please wait while your computer is
heating up...');
for i = 1:numIter
    txBits = randi([0,1], 1, round(nSym * k / 3));
    txTurbo = step(hEnc, txBits.');
    tx = step(hEMod, txTurbo).';
    % Pick a Channel
    if isequal(chan, 1)
        txFiltered = tx;
    elseif isa(chan, 'channel.rayleigh')
        reset(chan);
        txFiltered = filter(chan, tx);
    else
        txFiltered = filter(chan, 1, tx);
    end
    for j = 1:lenSNR
        % Add Noise
        txFilteredAndNoise = awgn(txFiltered,
SNR_Vec(j) + 10*log10(k * 1 / 3), 'measured');
        % Equalizer
        eq1 = dfe(3, 3, rls(0.99));
        eq1.SigConst = qammod(0:M-1, M);
        RefTap = 1;
        eq1.RefTap = RefTap;
        [rxEqualized, rxDetected] = equalize(eq1,
txFilteredAndNoise, tx(1:train));
        rxEqualized = [rxEqualized(RefTap:end)
zeros(1, RefTap-1)];
        rxTurbo = step(hDMod, rxEqualized.');
        % Plot signals.
        %h = scatterplot(txFilteredAndNoise, 1,
train, 'bx'); hold on;
        %scatterplot(rxEqualized, 1, train, 'g.',
h);
        %legend('Before Equalizer', 'After
Equalizer'); hold off;
        % Decode
        rxBits = step(hDec, -rxTurbo).';
        txBits_1 = txBits((train + 1) * k : end-
RefTap-5);
        rxBits_1 = rxBits((train + 1) * k : end-
RefTap-5);
        [Discard, berVec(i,j)] = biterr(txBits_1,
rxBits_1);
        % Count Bits
        brVec(i,j) = length(txBits_1);
        % Beautiful Wait Bar
        waitbar((i * lenSNR + j) / (numIter *
lenSNR), hw);
    end
end
close(hw);

ber = mean(berVec, 1);
br = mean(brVec, 1);
BER_AT_12_SNR = ber(1)
BIT_RATE_AT_12_SNR = br(1)
```

REFERENCES

[1] Using the Communications Toolbox [Online]. Available: http://www-rohan.sdsu.edu/doc/matlab/toolbox/comm/tutor132.html