Contents


```
% ECE408 - Wireless Communications
% Jongoh (Andy) Jeong
% 802.11b WLAN Standard - Simulation Project
% Date: February 19, 2020
clear all; close all; clc;
```

IEEE 802.11b

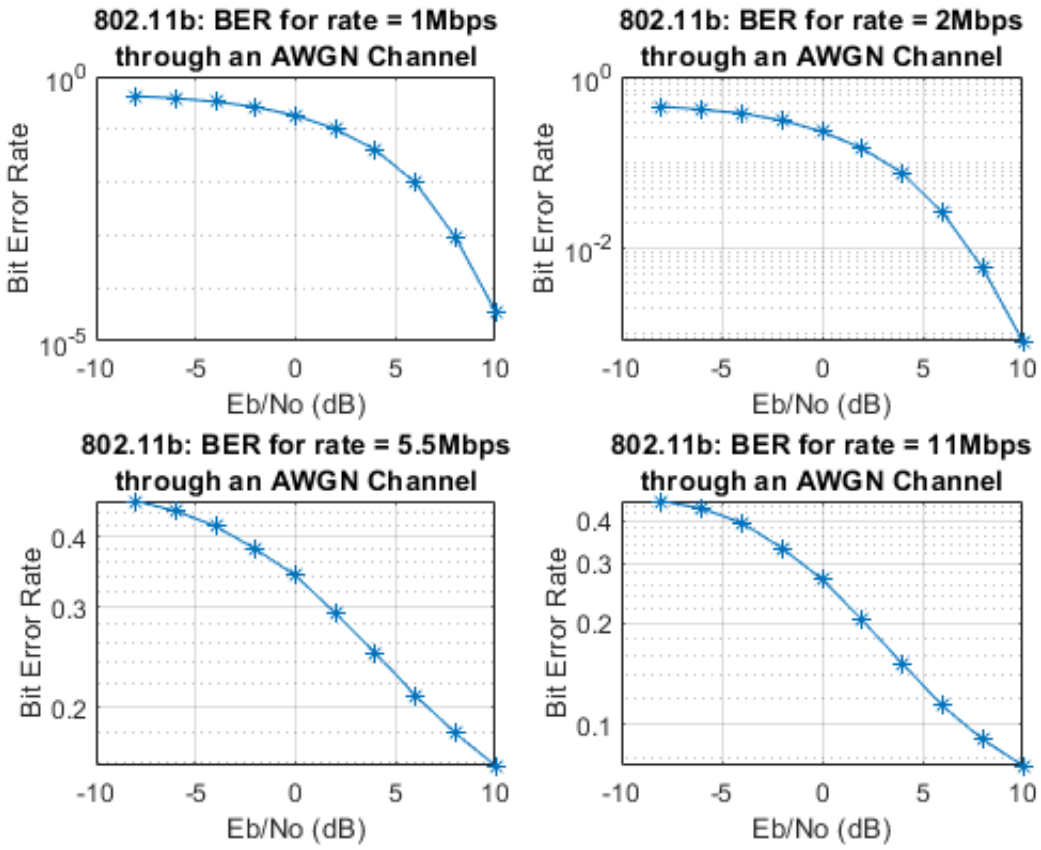
```
reset(RandStream.getGlobalStream);
rng default; % for reproducibility

% Simulation Parameters
snrVector = -8:2:10;
nIter = 10;
% min: 4, max: 8192 as per 802.11 standards
octetNumber = 1024; % number of octets
% samples per chip; samples will be up/down-sampled to this reference
spc = 8;

% call objects for modulation / demodulation with appropriate parameters
modFcns = {@(x, rate) ModSchemes.BarkerModulator(x,rate), ...
           @(x, rate) ModSchemes.BarkerModulator(x,rate), ...
           @(x, rate) ModSchemes.CCKModulator(x,rate), ...
           @(x, rate) ModSchemes.CCKModulator(x,rate)};
demodFcns = {@(x, rate) ModSchemes.BarkerDemodulator(x,rate), ...
           @(x, rate) ModSchemes.BarkerDemodulator(x,rate), ...
           @(x, rate) ModSchemes.CCKDemodulator(x,rate), ...
           @(x, rate) ModSchemes.CCKDemodulator(x,rate)};

% supported data rates as per 802.11b standard
dataRates = [1, 2, 5.5, 11];
% bits per symbol for all data rates
BPSes = [1, 2, 4, 8];
% chip spreading rates for all data rates
chipSpreadLengths = [11, 11, 8, 8];

% scrambler initialization seed
% scramInit = 93;
% msgBinSc = wlanScramble(msgBin,scramInit);
% RxBits = wlanScramble(RxBitsDsc, scramInit);

BERVector = zeros(length(dataRates),length(snrVector));
fprintf('Simulation starting...\n'); tic;
for rate=1:length(dataRates) %correrssponding to 4 different rate options (1, 2, 5.5, 11 Mb
ps)
    fprintf('Data rate: %.1f Mbps\n', dataRates(rate));
    for i=1:length(snrVector)
        fprintf('SNR: %.1f\t', snrVector(i));
        totalbits = 0;
        nerror = 0;
        for iter = 1:nIter
            % adjust SNR
            sampRate = chipSpreadLengths(rate) * spc;
            snrdB = snrVector(i) + 10*log10(BPSes(rate)) - 10*log10(sampRate);
```

```
            % generate packet
            msgBin = randi([0 1],octetNumber*8,1);
            [preamble, header, mpdu] = generatePacket(msgBin, 1);

            % modulate
            preambleMod = ModSchemes.BarkerModulator(preamble',1);
            headerMod = ModSchemes.BarkerModulator(header',1);
            msgMod = modFcns{rate}(msgBin, dataRates(rate));

            % - Frame: pack a packet frame withou mpdu since preamble and header
            % each needs to be modulated separately as DSSS1M
            txFrame = [preambleMod', headerMod'];
            txNoisyFrame = awgn(txFrame, snrdB, 'measured');

            % - PSDU: upsample, pass through a pulse shaping filter
            [h, upsampledChips, chipFilterDelay] = Filter.PulseShapeFilter(msgMod, spc);
            samples = filter(h,1,upsampledChips);
            % pass samples through an AWGN channel with adjusted SNR
            txNoisy = awgn(samples, snrdB, 'measured');

            % filer back (and downsample), perfectly knowing impulse response of the filte
r
            filtTxSig = filter(h,1,txNoisy); % column vector
            [RxChips,bitDelay] = Filter.Receiver(filtTxSig, spc, chipFilterDelay, BPSes(ra
te), chipSpreadLengths(rate));

            % demodulate
            frameRx = ModSchemes.BarkerDemodulator(txNoisyFrame,1);

            % parse packet frame
            [preambleRx, headerRx] = parseFrame(frameRx');
            checked = checkFrame(preambleRx, headerRx);

            if checked == true
                rxBits = demodFcns{rate}(RxChips, dataRates(rate));
                % compute number of error bits
                overlapSequenceMsgBin = msgBin(1:end-bitDelay);
                overlapSequenceRxBin = rxBits(bitDelay+1:end);
                % accumulate error and total bits over iterations
                nerror = nerror + sum(overlapSequenceMsgBin ~= overlapSequenceRxBin);
                totalbits = totalbits + (length(rxBits) - bitDelay);
            end
            fprintf('.');
        end
        fprintf('\n');
        BERVector(i, rate) = nerror/totalbits;
    end
end
toc;
fprintf('Simulation completed.\n');
```

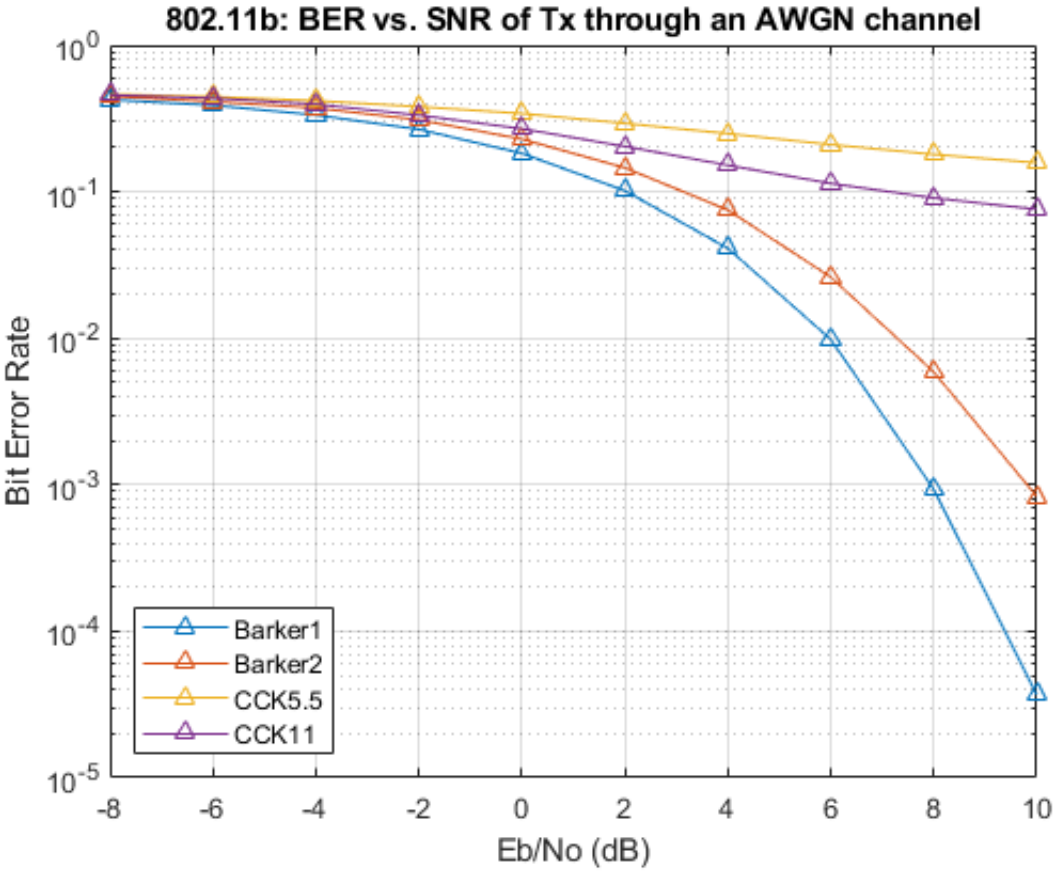
```
Simulation starting...
Data rate: 1.0 Mbps
SNR: -8.0       ..........
SNR: -6.0       ..........
SNR: -4.0       ..........
SNR: -2.0       ..........
SNR: 0.0        ..........
```

```
SNR: 2.0        ..........
SNR: 4.0        ..........
SNR: 6.0        ..........
SNR: 8.0        ..........
SNR: 10.0       ..........
Data rate: 2.0 Mbps
SNR: -8.0       ..........
SNR: -6.0       ..........
SNR: -4.0       ..........
SNR: -2.0       ..........
SNR: 0.0        ..........
SNR: 2.0        ..........
SNR: 4.0        ..........
SNR: 6.0        ..........
SNR: 8.0        ..........
SNR: 10.0       ..........
Data rate: 5.5 Mbps
SNR: -8.0       ..........
SNR: -6.0       ..........
SNR: -4.0       ..........
SNR: -2.0       ..........
SNR: 0.0        ..........
SNR: 2.0        ..........
SNR: 4.0        ..........
SNR: 6.0        ..........
SNR: 8.0        ..........
SNR: 10.0       ..........
Data rate: 11.0 Mbps
SNR: -8.0       ..........
SNR: -6.0       ..........
SNR: -4.0       ..........
SNR: -2.0       ..........
SNR: 0.0        ..........
SNR: 2.0        ..........
SNR: 4.0        ..........
SNR: 6.0        ..........
SNR: 8.0        ..........
SNR: 10.0       ..........
Elapsed time is 82.255146 seconds.
Simulation completed.
```

```
figure;
for rate=1:4
    subplot(2,2,rate);semilogy(snrVector,BERVector(:,rate),'*-');grid on;
    title_str = {['802.11b: BER for rate = ' num2str(dataRates(rate))...
                 'Mbps'] 'through an AWGN Channel'};
    title(title_str); xlabel('Eb/No (dB)'); ylabel('Bit Error Rate');
end
```




```
figure;
for rate=1:4
    semilogy(snrVector, BERVector(:,rate),'^-'); grid on; hold on;
end
hold off;
title('802.11b: BER vs. SNR of Tx through an AWGN channel');
legend('Barker1','Barker2','CCK5.5','CCK11','Location','SouthWest');
xlabel('Eb/No (dB)'); ylabel('Bit Error Rate');
```



Published with MATLAB® R2019b