Homework 1, CSE 461 Fall 2009

Instructor: Wetherall, Out: 9/30, Due: 10/14

Part A – Sending Bits over a Wire

Your job in this part is to write a program to decode the message in a signal and then use that program to explore design tradeoffs in network links. You must write in Java (start with the simple class that we give you) and you may do this part of the homework with a partner (and we recommend this).

At this stage of the course, our prototypical 461-net is a sending computer, a wire, and a receiving computer. The sender has encoded a message as a signal that has propagated across the wire. You have a record of the signal as seen at the receiver in "samples.txt". It is a sequence of voltage samples taken at a rate of one sample per bit. The sender encodes its message as follows:

1. It starts with a sequence of bits. The bits are a readable message, where each 8 bits is an ASCII character and the bits are sent in least-significant-bit order.
2. The bits are scrambled using the scrambler defined in Part 17.3.5.4 of the 802.11 (2007) standard initialized with all 1s. A local copy of the standard is available.
3. The bits are then protected with a Hamming (7,4) error correcting code. This code is not in your textbook, but you can learn about this code from Wikipedia or many other sources.
4. The bits are transmitted with an NRZ line coding that uses +1V as the high level and -1V as the low level.

Your first task is to decode the message. *Do this in stages for better mental health*. The file "Decoder.java" is a starting point that simply shows you how to read the samples. The file "samples-nrz.txt" has a message with steps 1 and 4 above only. The file "samples-scramble.txt" has steps 1, 2 and 4 only. The file "samples-hamming.txt" has steps 1, 3 and 4 only.

You will know your decoder works when you can read the message.

Question 1: Use your decoder to experimentally find out how well the encoding performs with and without the Hamming error correction code. To do this you will need to add different amounts of noise to the samples; more noise will produce more bit errors. (Decoder.java will get you started.) Hand in a graph that plots the bit error rate (BER) as a function of the signal-to-noise ratio (SNR) on a log-log scale.

Question 2: You are designing an encoding scheme for a wireless link. The bandwidth and transmit signal power of the link are fixed. The link must work for a range of SNR levels at the receiver. You have two encoding schemes available: with and without Hamming (7,4) error correction as plotted above. Describe when you would use each encoding and for what range of SNR your link works well.

Question 3: Why should getting any readable message out of your decoder in the testing scenario above imply that your decoder works? Give a rough line of reasoning in terms of error detection.

Turn in your code along with the answers to the above questions. We will not run your code to test it, but we will look at it to see what you did. Submit only the relevant portion, e.g., no pages of GUI code.

Extra Credit (purely for interest):

It is possible to build a better receiver for the same error correction scheme than you have probably built above by more cleverly decoding signals! Consider a repetition code in which a "1" is sent as the signal high-high and a "0" is sent as low-low. If we get high-low we know there was an error. However, we might still be able to correct the error. The noise might have pushed a low to a high in the first part of the signal or a high to a low in the second part of the signal. If one of the signals is close to 0V and the other is close to +1 or -1V then we probably know which sample is the more reliable indicator of the bit. (Draw signals and you will see.) Error decoding that uses the strength of the signal is called soft-decoding, while error decoding that maps directly to bits is called hard-decoding. Use these ideas to build a better Hamming (7,4) decoder. Hint: correlate the received signal with the possible codewords. How much better is this method?

Part B – Problems

1. P&D 1.13, 1.16, 1.23

2. The long-term average bit error rate of a link is 2^-30. Errors on the link are bursty. When there is an error, the probability that the burst is L bits long is 2^-L. The link carries 1KB packets that include a 16 bit CRC for error detection. Assume this CRC can detect all burst errors up to 15 bits and fail to detect burst errors of 16 or more bits with probability 2^-16. How good is this link? Provide a quantitative answer justified by rough calculations.

3. Why are N-bit CRCs preferable to an N-bit checksums in real systems? Hint: consider how a checksum can fail.

4. P&D 2.24, 2.47.