

Decoding the Binary Hamming Codes

Navid Azizi

February 19, 2003

1 Introduction

Hamming Codes are a family of $[n,k]$ linear block codes that have the following parameters

$$\begin{aligned}n &= 2^m - 1 \\k &= 2^m - m - 1\end{aligned}$$

2.1 Encoder

The encoder module has two functions. The *initialize_encoder* function creates an encoder for a particular hamming code. More specifically, it initializes the generator matrix for that code. The *encode* function then produces a codeword for the message.

2.2 Channel

The channel module also has two functions. The *initialize_channel* function sets the probability of bit-error for the channel, and the *receive* function returns the bit pattern after the codeword has been sent through a symmetric memoryless channel.

2.3 Decoder

The decoder module has two functions that are visible to the rest of the program. The *initialize_decoder* function creates a decoder for a particular hamming code. It creates the parity check matrix and a lookup table for syndromes. The *decode* function uses a series of local functions to obtain the syndrome, look up the error pattern, correct the received vector, and extract the data from the codeword.

2.4 Statistics

The statistics module used the sent and received data to compute the BER and FER for each communication. It stores these statistics in internal data structures. We can reset the data or have it printed with accessible functions.

3 Results

The C program described above was run originally with a probability of error, p , of 0.49 and the BER and FER was measured. The probability of error was reduced by 1.2 times in each successive experiment, until a probability of 0.0005 was reached.

Figure 1 shows the BER and FER for the [7, 4] Hamming Code. We see that in the log-log scale we obtain a straight line except when p nears 0.5 where the BER and FER start to saturate at 1 and 0.5 respectively.

The FER is always nearly a factor of 2.3 times larger than the BER. This is because at least 2 bit errors per frame are needed to obtain a frame error.

Figure 2 shows the BER and FER for the [15, 11] Hamming Code. We see that we see a very similar plot to the error-rates of the [7, 4] code. The difference between the BER and FER is now nearly 5. This is due to the larger block length, with there being the possibility of larger number of bit errors in each frame.

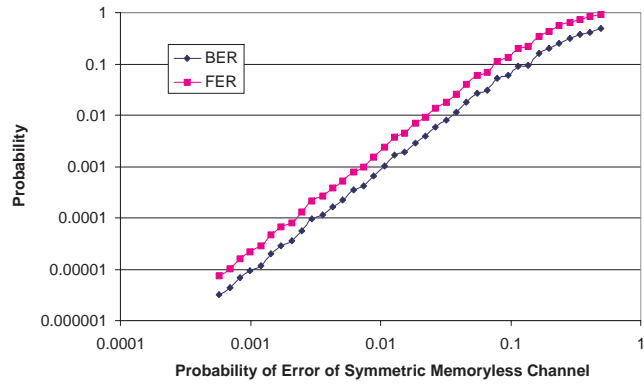


Figure 1: BER and FER for the [7, 4] Hamming code

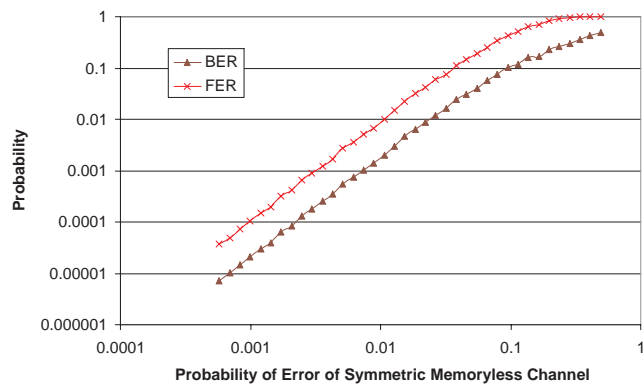


Figure 2: BER and FER for the [15, 11] Hamming code

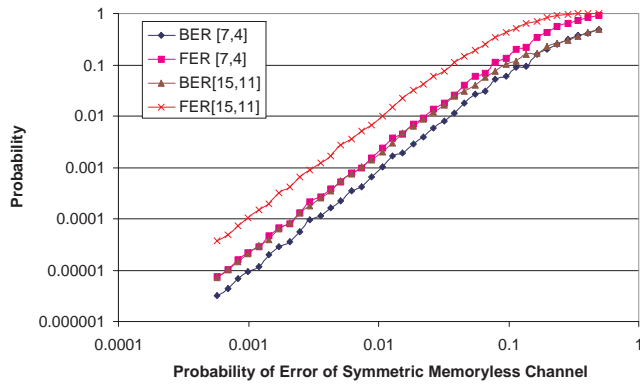


Figure 3: Comparing the [7, 4] and [15, 11] Hamming Codes

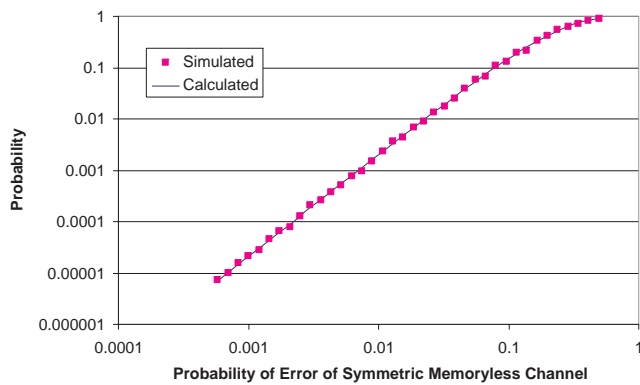


Figure 4: Simulated FER versus theory for the [7, 4] Hamming code

Figure 3 shows a comparison between the BER and FER for the [7, 4] and [15, 11] codes. We see that the [7, 4] code outperforms the [15, 11] since they are both single-error correcting codes and the [7, 4] code is a lower-rate code.

3.1 Theory versus Simulation

As seen in Figures 4 and 5 the simulation results follow closely the theoretical FER that was developed in Section 1.

Figures 6 and 7 show the histogram of the the percent error in the simulated FER results versus the FER obtained from the theory. It is seen that the [7, 4] percent error has a double hump distribution, and that is is skewed towards overestimating the FER. The mean and standard deviation in the percent error for the [7, 4] code is 2.2% and 7.0% respectively.

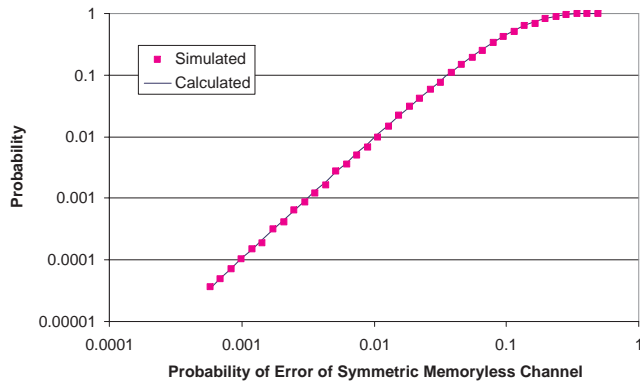


Figure 5: Simulated FER versus theory for the [15, 11] Hamming code

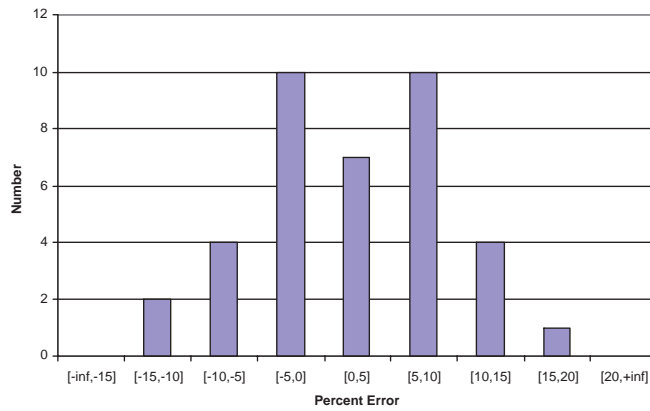


Figure 6: Histogram of % error of simulated FER versus theory for the [7, 4] Hamming code

For the [15, 11] code, the percent error histogram is skewed. The simulation results that overestimate the FER, overestimate it by small amounts (upto 5%), while the results that are underestimated, can range upto a 15% underestimation. The mean and standard deviation in the percent error for the [15, 11] Hamming code is -1.3% and 4.3% respectively.

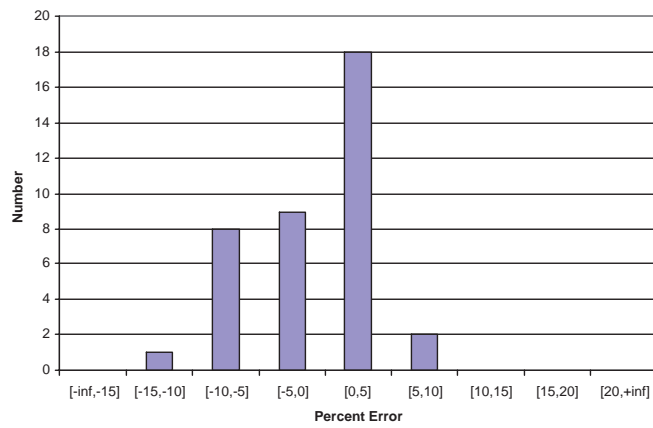


Figure 7: Histogram of % error of simulated FER versus theory for the [15, 11] Hamming code