CS378: Computer Networks Lab Fall 2024
Lab 2: Hands-On PHY Layer Design Document

Bathala Shashank(22b1041), Koyyana Suhas(22b1072),
Dayyala John Joseph(22b1063), Harideep Gandu(22b0987)

September 1, 2024

1 Encoding Technique

1.1 Hamming Code

Using the given input bit string (**a**) we can convert it into a new bit string (**b**) which comprises of the data bits (D) and parity bits (P) using **Hamming code**. Let the length of **a** be n, then the length of **b** will be n + $[\log_2 n]$ + 2 (where [.] indicates Greatest Integer Function). The new string **b** is generated by filling the data bits of **a** in the non-exponential (w.r.t 2) positions of **b** and the parity bits of **a** in the exponential positions (w.r.t 2 i.e, 1, 2, 4, 8, ...) of **b** which means **b** will be of the form $P_1P_2D_3P_4D_5D_6D_7P_8...$, where $D_3 = a_1, D_5 = a_2, D_6 = a_3$, ... the data bits are placed continuously at the non-exponential positions in **b**. Whereas a parity bit P_j (where $j = 2^i$) indicates the even parity of the string $D_{i_1}D_{i_2}D_{i_3}...$ and so on, where $i_1, i_2, i_3, ...$ indicates the positions in **b** where each i_r has the i^{th} bit set to 1.
Even Parity: If the number of 1's are even then the parity is 0, otherwise 1.

$$P_j = D_{i_1} \oplus D_{i_2} \oplus D_{i_3}...$$

For example, $P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12}$ as the 2^{nd} bit for 5, 6, 7, 12 is set to 1.

1.2 Preamble + Codeword = Encoded string

Preamble: It indicates the length of the input data string only. From the given problem statement as $n \leq 20$ and the integer 20 is of 5 bits, so we always give the length of the input string in 5 bits.
Codeword: It comprises of Input string (**a**), parity bit (p), Hamming code (**b**). The parity bit indicates the even parity of the string **b** i.e, if the the number of 1's in the string **b** are even then p = 0, otherwise p = 1.

$$Codeword = a + p + b$$

where + indicates concatenation. The reason behind the addition of p and **a** can be seen in the error detection/ correction part. The upcoming discussion comprises only of the transmitted string with atmost 2 bits flipped.

2 Error Detection & Correction

2.1 Number of bit flips

If the total number of 1's in (b + p) are even i.e, even parity of (b + p) is 0 then there might be definitely either 0 (or) 2 flips in (b + p), since before flipping itself (original string) we have the even parity of (b + p) as 0 (p $\oplus$ even-parity of b = p $\oplus$ p = 0). If the even parity of (b + p) is 1 then definitely there is one bit flip in (b + p). For correction it can be classified into two different cases.

2.2 Zero/Two Flips

We try to check whether parity bits tally with that of data bits in b (from the Hamming code method) since we know that parity bits always occupy the exponential positions in b we can easily identify parity, data bits in b at the receiver.
Case-(i): If the parity bits and data bits tally each other according to the Hamming code method then definitely there are no flips in b which implies 2 flips should occur in a. Using original Hamming code (b) we can extract the original input string and original transmitted message.
Case-(ii): If the parity bits and data bits doesn't tally each other according to the Hamming code method then definitely there are 2 flips in (b + p) which implies there are no flips in a. Hence we have the original input string, using this we can extract the original Hamming code (**b**) as well as its parity p.

2.3 One Flip

Here also we try to check whether parity bits tally with that of data bits in b (from the Hamming code method). We can have two different cases here.
Case-(i): If the parity bits and data bits tally each other then definitely there will be no flip in b which implies p is flipped. From the original Hamming code (b) we can extract the original input string and original transmitted message.
Case-(ii): If the parity bits and data bits doesn't tally each other according to the Hamming code method then definitely there will be one bit flip in b. Now we compare the parity bits in b (at the receiver) $...P_8P_4P_2P_1$ and the new parity bits from the data bits in b using Hamming code method $...P_8^*P_4^*P_2^*P_1^*$. If we have a difference between P_{2^j} and $P_{2^j}^*$ then definitely the j^{th} bit of i is set to 1 (where i is the position in b where there is a bit flip) since i falls in the P_{2^j} category (as i has j^{th} bit set to 1) and when i^{th} position in b gets flipped then P_{2^j} also gets flipped. Hence i can be calculated by,

$$i = (...P_8P_4P_2P_1) \oplus (...P_8^*P_4^*P_2^*P_1^*)$$

After evaluating i we can extract the original Hamming code (b) by flipping the i^{th} position in b at the receiver. From the corrected Hamming code (b) we can extract the original input string and original transmitted message.

3 Reception

3.1 Frequency Shift Keying

We are using the implementation of **Frequency Shift Keying (FSK)** communication system using **PyAudio** for the transmission of encoded bit string. **Frequency Shift Keying** is a digital modulation technique where information is transmitted by varying the frequency of a carrier wave between discrete values. In **binary FSK**, two distinct frequencies represent binary digits: one frequency for '0' and another for '1'. The carrier wave's frequency shifts according to the bit being transmitted, allowing the signal to convey digital data.

3.2 Sender Code-Receiver Code

The sender code generates an **FSK signal** based on evaluated encoded bit sequence and transmits it via audio output. Generates sinusoidal tones for bits '0' and '1' and concatenates these tones based on the bit sequence.
The receiver code captures the transmitted audio signal, performs frequency analysis, and decodes the bit sequence. **Epsilon** is used to identify the presence of specific frequencies in the received signal (there might be some difference in frequencies). Captures audio data in chunks and appends it to a buffer and concatenates buffer data to form the complete audio signal.

4 Changes in Design : #'s

Due to some calibration issues in the **Laptop's Microphone**, **Surroundings** there might some problem in sound detection. For example the device may detect some chunks of 0's, 1's even without the sound produced from the sender, then there will be a problem for detection of the transmitted message. To overcome that we are adding few (3-4) **hashes** before and after the transmitted message at the sender and # has a frequency different from 0 and 1. Hence the receiver is written in such a way that it listens and stores only the bit message between two #'s so that we can ensure that our bit message is **clean** (receives the message accurately).