Linsey Szabo
Computer Networks
Fall 2023

Project 1 Design Document

Decoder Design

1. Level 1: At this level, I undid the bit permutations accomplished by interleaving and returned the length of the message.
    a. First, I found the length of the message. From examining the transmitter, I found that the length was encoded using repetition and padded to be a length of 128. At the end, this length field is concatenated to the front of the output, so to start finding the length, I used splicing to get the first 128 bits of the output. I used NumPy functions to find where the first 1 occurred in this splice. After finding that position, I looped through that smaller segment. For every 3 1's I saw, I appended a 1 to my resulting binary string; for every 3 0's I saw, I appended a 0 to my resulting binary string. To finally find the length, I converted the binary string I had built up into an integer.
    b. To undo the interleaving, it was necessary to again examine the transmitter. The Interleave array contains 0, 4, …, 128. Then, looping through the bits in chunks of 128, the bits are permuted according to the indices in Interleave. Lastly, the message is padded. To mirror this procedure on the receiver side, I found the correct loop range and created the Interleave array just like the transmitter did. Then, I mapped the Interleave indices back to their original places in my result array. I removed the same amount of padding as added in the transmitter. Lastly, to convert the binary bits into the message, I placed the bits into arrays of 8 to call the NumPy packbits() function and finally took the chr() of each element to get the decoded message.
2. Level 2: To undo the turbo encoding and modulation at this step, I used soft Viterbi decoding and the demodulate() function from commpy. After getting my output from this, I fed it into my level_1() function to undo interleaving at the end. Again, I dealt with the length and message separately, because the length is not being turbo encoded.
    a. The length could simply be demodulated using the function from commpy. However, it was important to find the correct segment upon which to perform this. From examining the transmitter, we can see that the returned output has the preamble, then the length, and finally the message. Because the length of the output is cut in half after modulation, instead of indexing by 128, we index by 64 and then know that the length segment is bits 64-128 in the output. I then demodulated this segment.
    b. I implemented soft Viterbi decoding, because it is more robust to error correction, which will be important as the SNR is varied. I used a Viterbi_state class to define the transitions in the state diagram (trellis) and the current state (set to (0,0) at the start). To define the transitions, I used a dictionary that mapped a tuple of the current state and received bits to a tuple of the next state and output bit. I also wrote a function to find the distance between coordinates

by summing the square absolute differences between the x and y of each coordinate. Looping through each part of the message by looking at the real and imaginary part together, I examined the most probable path by calculating the distance between the received bits and possible next states. I then set the next state and output bit in the decoded message accordingly.

3. Level 3: To undo the OFDM signal generation, the most important thing here was examining the transmitter. It is clear that OFDM signal generation is being performed by taking the inverse Fourier transform. Therefore, to undo this, I took the Fourier transform in the same chunks that the transmitter did.
4. Level 4: To find where the preamble occurred, I first made the given preamble into the necessary format to compare to the transmitter output, and then I found the maximally correlated window of the transmitter output and the preamble and returned where this occurred to find where the zero padding began. From examining the transmitter, I saw that the zero padding at the beginning and ending of the packer were randomized at each transmission, and so this approach is necessary. To transform the preamble into the format of the transmitter output, I modulated it then took the inverse Fourier transform to essential perform modulation and OFDM signal generation. Finally, NumPy's correlate() and argmax() accomplished my described logic. To get the final message, I cut off the part of the transmitter message before my begin_zero_padding index and fed that into my level_3() function. Lastly, using the length returned from level_1(), I indexed into the final message to cut off the zero padding (added in the transmitter after examination).

Inferring Length of Packet

I inferred the length of the packet by decoding the repetitive encoding in level_1() and correctly locating the preamble in level_4().