Networks Lab 3: Design Draft

Brian Mackwan(22b0413), Ekansh Ravi Shankar(22b1032),
Shravan S(22b1054), Ved Danait(22b1818)

October 2024

Chapter 1

Physical Layer

1.1 Transmission and Reception

1. We maintain a set of three frequencies corresponding to three input values : **0** – 1000, **1** – 4000, **2** – 500.

2. We use the input value "2" as a sort of buffer to separate adjacent bits. Our encoding is as follows :

   $$bit[i] \xrightarrow{encode} bit[i] + \text{‘2’}$$

   So for example, 10110 will become (12)(02)(12)(12)(02)

3. For each value in the encoded message, we play the corresponding frequency. For the sender, the **duration** for each value is **0.2** seconds. Meaning in a 0.4 second interval we can relay a single bit in the original message which has 0.2 seconds of the original message and 0.2 seconds of the buffer frequency.

4. In the receiver, we have tentatively kept the **duration** as **0.1** seconds. This decreases the probability of error since we break the tone into more windows, meaning that we have higher chances of catching each distinct tone.

5. We keep the **threshold** as **2 Hz** on either side of the corresponding frequencies.

6. We collect each and every discernible tone at the receiver which is within 2 Hz of any of our frequencies. We now need to post-process this string. This is simply done by iterating through the array and storing each value which is not the same as the previous value. Finally, just remove all '2's from the string to obtain the original message.

1.2 Error Correction

1. We use **Cyclic Redundancy Check** to correct the errors.
2. To perform CRC, we use an appropriate divisor polynomial, which when applied on messages with 20 or lesser bits, has a **Minimum Hamming Distance of 5** in the final message which is the initial message and the appended remainder.
3. Thus, when two error bits are introduced, we can simply brute force by **taking all possible pairs** as error bits, and flipping them and checking which one gives a **remainder of zero** when dividing by the divisor polynomial used.
4. This works because all the strings in our space have a minimum Hamming Distance of 5, and hence there will be a **unique pair which has to be flipped** to give the solution.
5. The **divisor polynomial** to be used is

   $$x^{10} + x^8 + x^6 + x^5 + x^4 + x + 1$$

   According to `https://users.ece.cmu.edu/~koopman/crc/` and local tests conducted, this polynomial works for all messages of sizes upto 20 bits, which means the final message transmitted will be upto **30 bits** (this is only the message and remainder), because the degree of the divisor is 10.

Chapter 2

MAC Layer

2.1 Design and Instructions to Run

In this lab, we shall be implementing CSMA - CA with acknowledgements for the MAC layer, and will be demonstrating using three laptops.
There is one identical file corresponding to each node, with only changes in the MAC ID variable in the file. There is also one input file corresponding to each node, where the input will be given. On each laptop, the command Get-Content input1.txt | python mac1.py has to be run for Windows, and python3 mac1.py < input1.txt for Linux. Here, 1 is the MAC ID and has to be changed accordingly for each node.

2.2 Message Encoding

1. Add the receiver and sender MAC IDs before the message. That is, m becomes $[R][S]m$, where [R] is the MAC ID in binary for the receiver, and [S] is that of the sender.

2. Perform CRC on this new message, with the key $x^5 + x^3 + x + 1$, and append the remainder. This key has been selected based on the fact that the message length to perform CRC on will be no more than 19 bits.

3. Now, perform bitstuffing on this new message, where a zero is added after 4 consecutive 1s, if found.

4. Finally, prepend three ”3s”, which are a different frequency to indicate the start of a message, and append ”011111” to indicate the end of a message.

We use a Return to Zero method of encoding, during transmission where we add a ”2” between every bit in the message, which is of a different frequency, so that we can identify the number of bits in succession. This has been explained in Chapter 1.

We append "011111" so that we know when a message ends and the node can stop listening, and we use CRC to detect whether the message has errors in it or not, in which case it can be retransmitted.
The acknowledgement is "4444", where the 4s are a different frequencies. This is so that a node does not confuse a message for an acknowledgement.
The frequencies used are '2': 500, '1': 4000, '0':1000, '3':750, '4': 2000.

2.3 Algorithm

The following is the algorithm followed for CSMA - CA

1. initialise K to 0
2. We first check if the audio channel is free by listening to if any active transmissions are being made. If yes, proceed to the next step
3. We wait for 30 iterations in which the channel is empty after which we generate a random time using exponential backoff, and if it is we move on to the next step, and if not, we go back to the previous step
4. For exponential backoff, we have kept an exponential upper bound (u) and exponential cutoff (c). We choose a random time R between 1 and $min(u2^K, c)$ and then wait for R iterations and then transmit if channel is empty
5. Wait for a time-out. If ACK is received, then success otherwise increases K to K+1 and repeat from Step 2
6. The above exponential backoff is implemented till we reach c, beyond which we can choose to continue without a backoff

This algorithm keeps listening on the channel and checks if the new message is meant for it. If it is, then it also sends an acknowledgement back. This works because in smaller systems the randomisation ensures that even in case of a collision we will not receive the proper acknowledgement message and retransmission will occur.

2.4 States Used

The states for each node that have been used in our algorithmare as follows

1. CS : Carrier Sensing - In this state we will keep track of the message on the channel, and decide whether it has errors, or it is meant for you, or someone else. If it is not meant for you, then move to CB, otherwise, acknowledge it by moving to BSACK or SACK. If the channel is empty and you have a message to send, you will try sending it in AM, using exponential backoff.

2. CB : Channel Busy - The Node has sensed someone else's transmissions and it stays silent for 120 transmissions and then moving to CS.

3. WACK : Wait Acknowledgement - The Node has sent a message and waits for an acknowledgement from receiver for 120 iterations. After 120 iterations, if it still has not received a response it goes to the state CS and the cycle starts again after exponential backoff. If it receives an acknowledgement, then it pops the message off of the message queue and goes to the state CS.

4. SACK : Send Acknowledgement - The Node has to send an acknowledgement to the send after receiving the message. Once this is done it goes back to the state CS.

5. BWACK : Broadcast Wait Acknowledgement - This is similar to WACK, but waits for two acknowledgements. That is, after hearing one acknowledgement, it moves to state WACK. This can be generalised to n nodes by waiting for $n-2$ acknowledgements and then moving to WACK.

6. BSACK : Broadcast Send Acknowledgement - Here, you send an acknowledgement if you are the minimum node to receive a message, otherwise you move to a state called WASACK, where you wait for an acknowledgement, and then send your acknowledgement. This is also generalisable to n nodes, by keeping track of the last node that has sent an acknowledgement.

7. AM : Attempt Message - Here, you wait for R iterations, defined in the algorithm, and if you do not hear anything, you play your message and move to BWACK or WACK depending on whether it is a broadcast message or not. If you hear something, you will reset and move back to CS state, and try sending again.

2.5 References

https://www.geeksforgeeks.org/carrier-sense-multiple-access-csma/

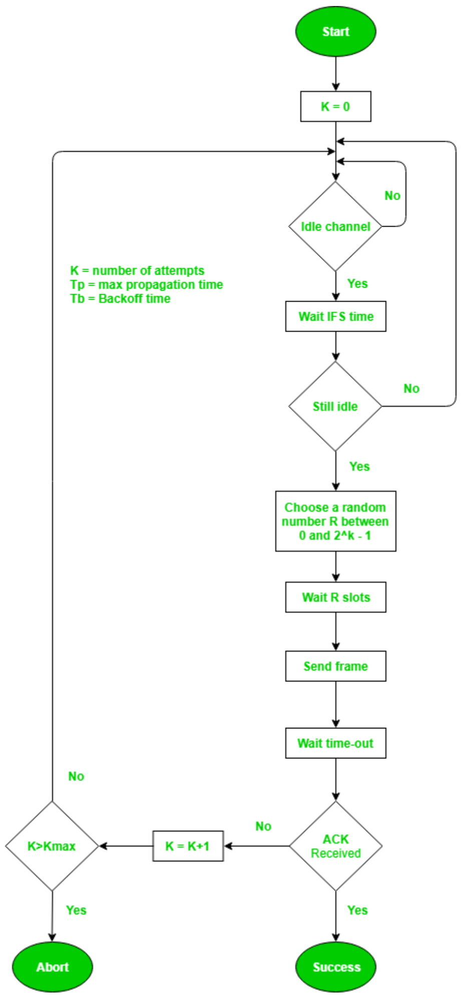

Figure 2.1: CSMA Image