DCNC Assignment 2

Noah Bakr | s4095646
Kyle Bonjour | s4003110
Sebastian Nagendran | s3949420

Task 1 [Noah]

Part 1 Student ID and Flowchart

The chosen student ID is '4095646'. To convert the ID to a 7-bit binary output, the mod 2 operation (% 2) is performed on each digit. The following flow chart explains the conversion process and the result is '0011000'.

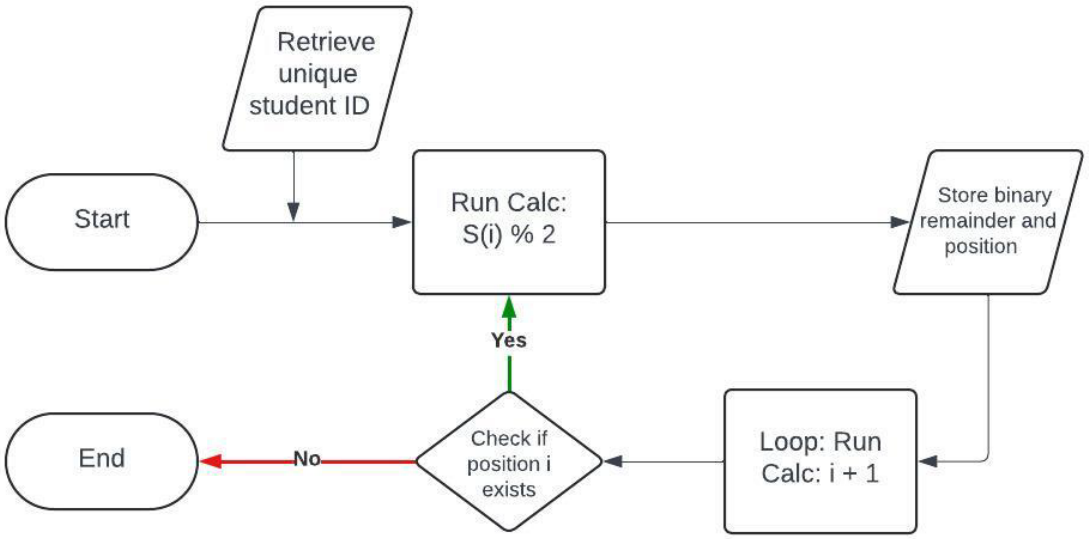

Figure 1.1, Flowchart presenting S(i) to D conversion

Part 2 Hamming Code

To ensure the data is transmitted correctly, with error tolerance in mind, the raw binary is converted to a Hamming Code binary number, which is 11 bits long.

How do we calculate this?

Firstly, we place our data bits (Dn) as follows: D7 D6 D5 D4 D3 D2 D1, which translates to 0011000. Next, the redundant bit placeholders (rn) must be placed in their respective positions. As the student ID binary is 7 bits long, we only require 4 redundant bits. Their number increases by the power of 2 and are placed by their power. For example, r4 is placed in the 4th position, inclusive to r1 and r2. As a result of the placement, the following table is created and now used as a guide: D7 D6 D5 r8 D4 D3 D2 r4 D1 r2 r1.

We start by calculating the first redundant bit (r1). The '1' refers to every second position from r1 so, we isolate those positions in focus, which can be presented like this: D7 […] D5 […] D4 […] D2 […] D1 […] r1. In this case, the isolated table will look like so: 0 […] 1 […] 1 […] 0 […] 0 […] r1. The next step is to count all the 1's in the focus. Since we are using the even parity bit rule, if there are an odd number of 1's present, r1 must be a '1' so that an even number of 1's are present. If there are already an even number of 1's, an additional '1' is not necessary, therefore r1 becomes a 0. In this case, there are two 1's, so r1 = 0: 0 […] 1 […] 1 […] 0 […] 0 […] 0.

We then calculate the second redundant bit (r2). The '2' refers to a '2 selected and 2 skipped' system, starting from r2 so, we isolate those positions in focus, which can be presented like this: D7 D6 […] […] D4 D3 […] […] D1 r2 […]. In this case, the isolated table will look like so: 0 0 […] […] 1 0 […] […] 0 r2 […]. As there are an odd number of 1's present in this focus (only a single 1 with 4 zeroes), r2 = 1, which makes the count of 1's even and is presented as 0 0 […] […] 1 0 […] […] 0 1 […].

We then calculate the third redundant bit (r4). The '4' refers to a '4 selected and 4 skipped' system, starting from r4 so, we isolate those positions in focus, which can be presented like this: […] […] […] […] D4 D3 D2 r4 […] […] […]. In this case, the isolated table will look like so: […] […] […] […] 1 0 0 r4 […] […] […]. As there are an odd number of 1's present in this focus (only a single 1 with 2 zeroes), r4 = 1, which makes the count of 1's even and is presented as […] […] […] […] 1 0 0 1 […] […] […].

Finally, we calculate the fourth redundant bit (r8). The '8' refers to a '8 selected and 8 skipped' system, starting from r8 so, we isolate those positions in focus, which can be presented like this: D7 D6 D5 r8 […] […] […] […] […] […] […]. In this case, the isolated table will look like so: 0 0 1 r8 […] […] […] […] […] […] […]. As there are an odd number of 1's present in this focus (only a single 1 with 2 zeroes), r8 = 1, which makes the count of 1's even and is presented as 0 0 1 1 […] […] […] […] […] […] […].

Final Step - Stacking

To combine these focused views and calculations into a single binary with Hamming Code, we 'stack' the duplicate numbers alongside the unique bits. This concept is shown in the table below. For example, the redundant bits are unique as they only appear once, while the data bits appear multiple times. To ensure this process has been done correctly, all stacked bits should be the same. The Hammed Code result is '00111001010'.

INDEX	11	10	9	8	7	6	5	4	3	2	1
Encoded	d7	d6	d5	r8	d4	d3	d2	r4	d1	r2	r1
r1	0		1		1		0		0		0
r2	0	0			1	0			0	1	
r4					1	0	0	1			
r8	0	0	1	1							
HC	0	0	1	1	1	0	0	1	0	1	0

Part 3 Error Detection and Correction

When sending the student ID as a piece of binary with Hamming Code, errors could be present. When a piece of binary is received, the receiver must use the Hamming Code parity bits to check for any errors and resolve them. When scanning individual digits, the receiver uses the redundant bits as a guide, from r8 to r1. In this case, the receiver has received '10111001010'

[**Step 1**] As a visual representation, the focused values are highlighted. The receiver first scans over all digits that relate to the 8th redundant bit, these values are '1011', with the leftmost '1' being assigned to r8, shown here: 10111001010. As the even parity bit rule is being followed, we can see that there is an error in the 8th redundant bit section (odd number of 1's).

[**Step 2**] With that error in mind, the receiver then scans over all digits that relate to the 4th redundant bit, these values are '1001', with the leftmost '1' being assigned to r4, shown here: 10111001010. As the even parity bit rule is being followed, we can see that there is no error in the 4th redundant bit section. This parity bit can be dismissed.

[**Step 3**] The receiver then scans over all digits that relate to the 2nd redundant bit, these values are '001001', with the leftmost '1' being assigned to r2, shown here: 10111001010. As the even parity bit rule is being followed, we can see that there is an error in the 2nd redundant bit section (odd number of 1's).

[**Step 4**] The receiver then scans over all digits that relate to the 1st redundant bit, these values are '111000', with the leftmost '1' being assigned to r1, shown here: 10111001010. As the even parity bit rule is being followed, we can see that there is an error in the 1st redundant bit section (odd number of 1's).

[**Step 5**] To locate the position of the error, we recall the redundant bits with errors. These were: r8, r2, and r1. An addition calculation is performed: 8 + 2 + 1 = 11. As a result, the error is in position 11, seen here: '10111001010', to solve the error, we switch the '1' to a '0', like so: '00111001010'.

[**Step 6**] As a final measure, the receiver scans through each redundant bit section to ensure the error is solved:

- 00111001010 = satisfies even parity bit of r_8.
- 10111001010= satisfies even parity bit of r_4.
- 10111001010= satisfies even parity bit of r_2.
- 10111001010= satisfies even parity bit of r_1.

Task 2 [Sebastian]

2.1, List 3-5 steps to explain how the frame check sequence (FCS) is used for error detection. Draw a figure to show how the receiver checks the error (4 marks).

[Step 1, Sender data setup] The sender will have a message, for example if the message is a 6-bit message it could be M(x) = 100100. There will also be a pattern that is agreed upon by both the sender and the receiver, for example it could be a 4-bit pattern, therefore P(x) = 1101. As the pattern is a 4-bit value the CRC value will then be a 3-bit message as the CRC value is 1 less bit than the pattern. As the CRC value is 3 bits, 3 zeros are padded onto the end of the 6-bit message making it 9-bits, therefore it is F(x)=100100000.

[Step 2, Sender calculation] The pattern is used as a divisor to divide F(x)=100100000, the result is then divided by the divisor again, this process is repeated until there is a final remainder, it should be noted that if a remainder during the process has a leading 0 then the divisor's bits will turn to 0s for that specific calculation. If the final remainder is not 3 bits, it will be padded with 0s until it is 3 bits, for example if the final remainder is 1 it will become 001.

[Step 3, Data sent from sender to receiver] The original 6-bit message is then padded with the CRC value, which is the final remainder of the sender calculations, and is then sent to the receiver

[Step 4, Receiver calculation] Then this 9-bit data is divided by the divisor on the receiver's end, the procedure is the same as explained in step 2, this process continues until there is a final remainder.

[Step 5, Error detection] If the final remainder is zero then the data unit is correct, however a non-zero remainder indicates to us that the transmitted data is corrupted and therefore an error is detected, and the sender will send the data again, this is the CRC base 2 method.

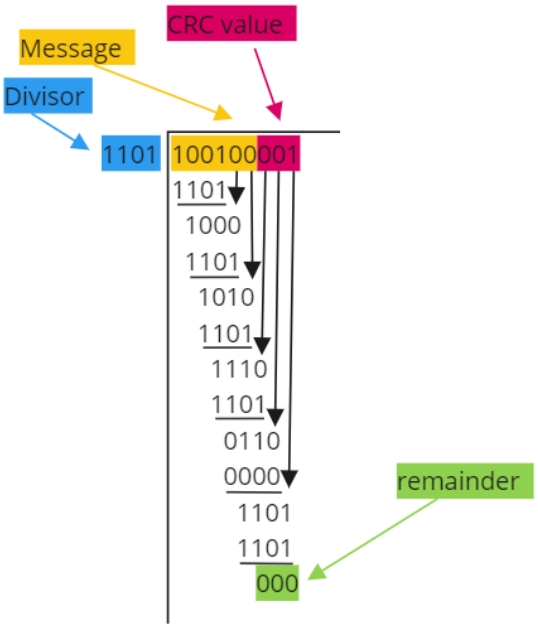

Figure 2.1, showing how the receiver checks the error

2.2, Which layer of the TCP/IP model associates with the FCS? Based on the figure you created in Task 2.1, explain how to ensure integrity in that layer? (3 marks).

The frame check sequence (FCS) is associated with the Link layer of the TCP/IP model, as this layer handles the physical and logical transmission of data between devices which are on the same network segment. One of the aspects of data transmission that this layer is responsible for is error detection. These reasons can be attributed as to why the frame check sequence is associated with the link layer of the TCP/IP model.

Integrity is ensured at this layer by the cyclic redundancy check (CRC), as the calculation, explained in task 2.1 above, results in a remainder. In the case where the remainder is zero, it is assumed that this data is valid and correct, however if the value is a non-zero remainder, then it can be assumed that the data has been corrupted and therefore an error has occurred and the sender will send the data to the receiver again. This is precisely how integrity is ensured at this layer.

2.3, Give an example to discuss how to trade-off the reliable data transmission and minimize latency when selecting the Transport layer protocols (3 marks).

The transport layer protocols to compare when examining the trade-off between reliable data transmission and minimizing latency are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). UDP minimizes latency as it has less delay than TCP, this is because UDP has does not require acknowledgment when sending and accepting data, therefore it is able to continuously send through data which will minimize latency. UDP also has less latency than TCP because TCP checks all the data and requires acknowledgement, therefore it is a more reliable form of data transmission, but as there are more checks this leads to higher latency. This can also be noticed in their headers, as UDP has a fixed-length header which holds less bytes and is faster and more efficient than TCP's variable-length header, which is slower, holds more bytes, but is more reliable.

An example of how to trade off the reliable data transmission and minimize latency would be using UDP for live streaming, as opposed to TCP. Live streaming requires minimal latency and doesn't need very reliable data transmission as it views small packet loss as acceptable if that means latency minimization is prioritized, which is why UDP would be used as it would prioritize minimizing latency over providing reliable transmission of data.

Task 3 [Kyle]

	Stop-and-Wait	**Go-Back-N ARQ**	**Selective Repeat ARQ**
Mechanism	The sender sends one frame and waits for the ACK before continuing.	The sender sends multiple frames before needing ACK. The receiver then sends ACK.	The sender sends multiple frames and waits for a single ACK for single frames. If a frame is incorrect then only that frame is sent again.
Resources Required	Low Buffer Size (Only one frame at a time)	Moderate Buffer Size (Stores up to N frames)	High Buffer Size (Buffers for the sender and receiver)
Advantages	- Simple Implementation. - Small amount of Requirements.	Efficient use of bandwidth with a moderate amount of frames within transit.	- Very Efficient use of bandwidth. - Only error frames are retransmitted, saving bandwidth.
Disadvantages	- High wait time so ACK is needed after each frame. - Not efficient for high bandwidth or high latency networks.	- Can lead to unnecessary retransmissions if an error occurs - Needs Larger Buffers compared to Stop and Wait	- Implementation is Complex due to the need for buffering and handling individual ACKs and retransmissions.

References

- **BasuMallick C (2022), TCP vs. UDP: Understanding 10 Key Differences, Spiceworks, accessed 12 June 2024. https://www.spiceworks.com/tech/networking/articles/tcp-vs-udp/#:~:text=To%20begin%20with%2C%20TCP%20uses,can%20have%20only%20eight%20bytes.**
- **Patadia Y (2024), TCP vs. UDP: Optimising Video Streaming Performance, Gumlet, accessed 12 June 2024. https://www.gumlet.com/learn/tcp-vs-udp/**