1. One way of detecting errors is to transmit data as a block of n rows of k bits per row and add parity bits to each row and each column. The bit in the lower-right corner is a parity bit that checks its row and column. Will this scheme detect all single errors? Double errors? Triple errors? Show that this scheme cannot detect some four-bit errors (Provide enough explanations). Show that this scheme can correct all single bit error. Show that this scheme cannot correct some two-bit errors.

Single Errors: Always detect AND correct the bit error. See picture below:

\*\*NOTE: ASSUME EVEN PARITY

\*\*Red numbers represent bit that was received incorrectly.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |

When you change one bit, that will violate the parity bit of the column and row (highlighted in yellow). The intersection of the column and row violation highlights the ***exact*** bit that caused the violation and thus can be corrected.

Double Bit Errors: Not able to detect some bit errors. Also unable to correct some 2-bit errors. See examples below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |

Here the row parity is valid, so it did not detect that a bit error occurred for the bit string: 000011 (trailing 1 bit is parity bit). However, it did detect that some error occurred because there was a violation for the column parity bits (highlighted in yellow). In this case, it would not be able to correct the bit error because there is no intersection between row and column parity violation that would highlight ***exactly*** where the incorrect bit is.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |

Highlights example where you can correct two bit errors that occur. This is because row and column parity violation exist, therefore you can find intersection that highlights where incorrect bit is.

Three Errors: Can detect some bit errors but not all. Also cannot correct some 3-bit errors. Same concept as double errors where one either a row or column parity could still be valid despite the bit string being different than original. See example below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |

* Only detected the third bit string as invalid but considered the second bit string as valid.
* Cannot correct because there’s no intersection between row and column that determines where the bit error occurred.

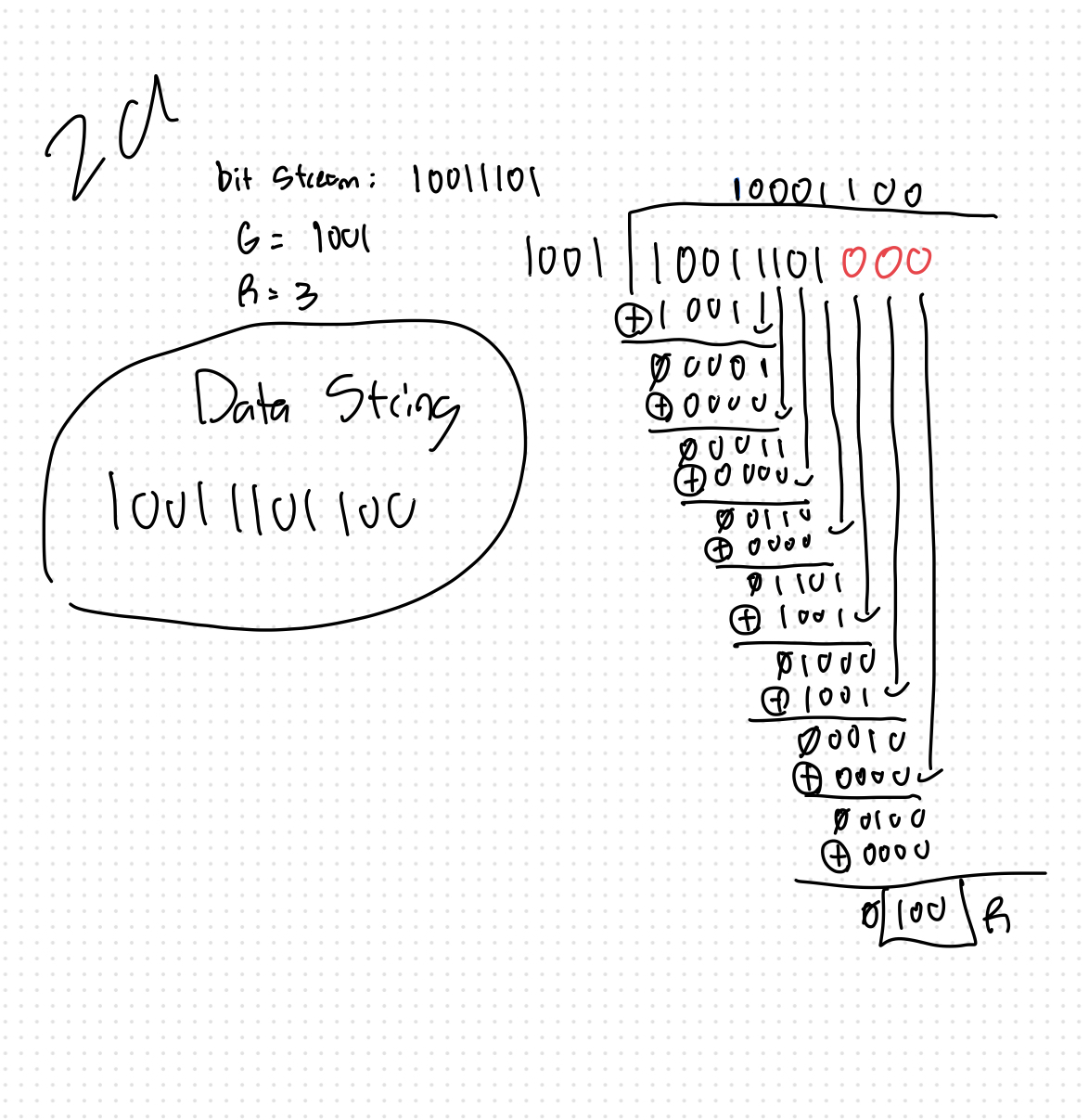
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |

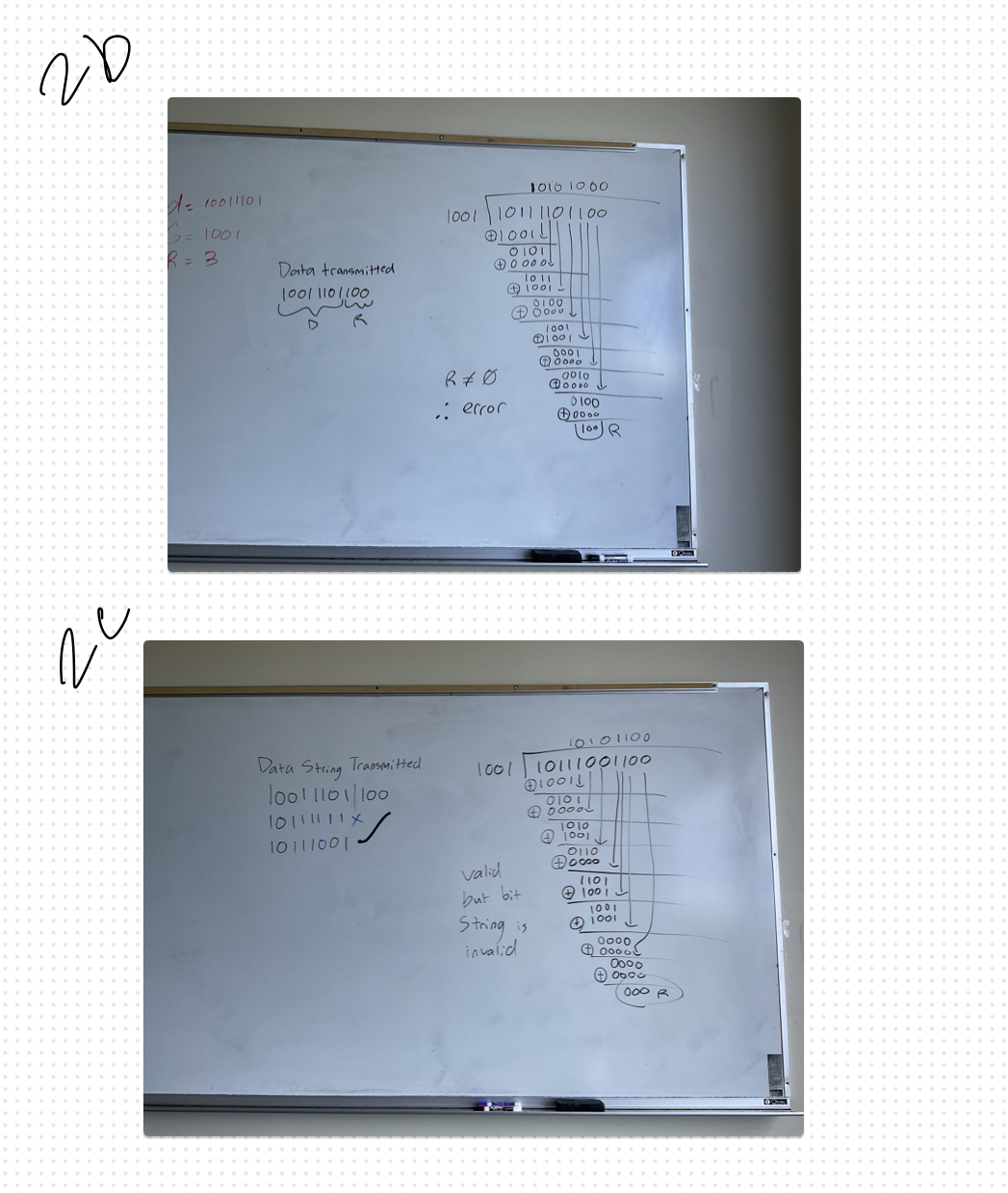
* Represents a case where all bit errors are detected and can be corrected.

4 Errors: Cannot detect some four-bit errors. In this case all parity for row and column is valid therefore it is received as a valid string despite having incorrect bits. See example below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |

2.





3. Consider the delay of pure ALOHA vs slotted ALOHA at low load. Which do you prefer? Explain.

🡪 I would take pure ALOHA over slotted ALOHA at low load because pure ALOHA works best / most efficient when there’s little traffic. The reason being is that it doesn’t need to worry about syncing with other nodes or limiting the amount of data it can send on the channel. With little traffic then there’s less chances of collision therefore pure ALOHA could be just as efficient for sending data.

4. Give an example that there exists starvation using CSMA / CD algorithm. Starvation means that some PCs want to send data but they wait for infinite time.

🡪 services like DropBox or apps where you share / send files to a shared server / drive. If two nodes excessively use that channel compared to other nodes to send files then a node might wait for an infinite amount of time waiting for one of those nodes to stop sending. This is especially true for cases like Wireless LANs where it might take even longer for a node to detect if a channel is now empty due to the amount of time it takes to propagate and/or transmit a file across the link to the channel.

5.

A 🡪 B: No other communications possible because A can communicate with all other stations.

B 🡪 A THEN B 🡪 B (through A) and/or B 🡪 D (through A).

B 🡪 C THEN B 🡪 D

6.

A sends a frame to C.

Steps:

* A sends out frame to S1
* S1 receives frame from A at Port 1 and records it. (A | 1)
* S1 broadcasts message to all other ports except 1
  + Note: S2 receives a frame and records A at Port 4 (A | 4)
* C receives frame from A and acknowledges.
* C sends to S1.
* S1 records acknowledgement from C at Port 3 (C | 3).

Table:

S1

|  |  |
| --- | --- |
| A | 1 |
| C | 3 |

S2

|  |  |
| --- | --- |
| A | 4 |

E sends a frame to F

Steps:

* E sends out frame to S2
  + In reality, F receives frame because they are part of same port and linked via Ethernet. F sends out acknowledgement to S2
* S2 records E at Port 2. (E | 2)
* S2 records F at Port 2. (F | 2)
* S2 broadcasts to all other ports
* S1 receives message from E at Port 4 (E | 4) and records it.

Table:

S1

|  |  |
| --- | --- |
| A | 1 |
| C | 3 |
| E | 4 |

S2

|  |  |
| --- | --- |
| A | 4 |
| E | 2 |
| F | 2 |

F sends a frame to E

Steps:

* F sends out message to S2
  + E receives message because it’s part of same port and linked via ethernet
* S2 receives message from F.
* S2 checks table and sees that E is on Port 2 and does nothing
* E receives message and sends acknowledgement.
* S2 checks table and sees that F is on Port 2 and does nothing

Table:

S1

|  |  |
| --- | --- |
| A | 1 |
| C | 3 |
| E | 4 |

S2

|  |  |
| --- | --- |
| A | 4 |
| E | 2 |
| F | 2 |

G sends a frame to E

Steps:

* G sends out frame to S2
* S2 receives message and records G at Port 3 (G | 3)
* S2 sees that E is on port 2 and broadcasts message to Port 2
* E receives messages and acknowledges.
* S2 receives acknowledgement and broadcasts to Port 3

Table:

S1

|  |  |
| --- | --- |
| A | 1 |
| C | 3 |
| E | 4 |

S2

|  |  |
| --- | --- |
| A | 4 |
| E | 2 |
| F | 2 |
| G | 3 |

D sends a frame to A

Steps:

* D sends out frame to S2.
* S2 receives message and records D at Port 1. (D | 1)
* S2 then broadcasts message to all other ports except 1.
* S1 receives D’s message from Port 4 and records it in switch table (D | 4)
* S1 sees that A is on Port 1 then broadcasts message to Port 1
* A receives message and acknowledges.
* S1 receives acknowledgement then broadcasts message to Port 4
* S2 receives acknowledgement then broadcasts message to Port 1.

Table:

S1

|  |  |
| --- | --- |
| A | 1 |
| C | 3 |
| E | 4 |
| D | 4 |

S2

|  |  |
| --- | --- |
| A | 4 |
| E | 2 |
| F | 2 |
| G | 3 |
| D | 1 |

B sends a frame to F

Steps:

* B sends out message to S1.
* S1 gets message and records B at Port 2 (B | 2)
* S1 then broadcasts message to all other ports except Port 2
* S2 receives message and records B at Port 4 (B | 4)
* S2 then sees that F is on Port 2 and broadcasts message to Port 2
* F receives message and sends acknowledgement to S2
* S2 then broadcasts message to Port 4
* S1 then receives acknowledgement and records F at Port 4
* S1 then broadcasts acknowledgement to Port 2

Table:

S1

|  |  |
| --- | --- |
| A | 1 |
| C | 3 |
| E | 4 |
| D | 4 |
| B | 2 |
| F | 4 |

S2

|  |  |
| --- | --- |
| A | 4 |
| E | 2 |
| F | 2 |
| G | 3 |
| D | 1 |
| B | 4 |