Data Error Detection and Correction

Networks must be able to transfer data from one device to another with acceptable accuracy. Any time data are transmitted from one node to the next, they can become corrupted in passage. Errors can occur during data transmission. From a simple blip to a massive outage, transmitted data is susceptible to many types of noise and errors.

Kinds of Noise in Data Transmissions (White, 2015)

- **White noise (Thermal noise or Gaussian noise)** is always present to some degree in transmission media and electronic devices and is dependent on the temperature of the medium. In this noise, when the temperature increases, the level of noise increases because of the increased activity of the electrons in the electronic device.
- **Impulse noise (Noise spike)** is a noncontinuous noise and one of the most difficult errors to detect because it can occur randomly. Typically, the noise is an analog burst of energy. If the impulse spike interferes with an analog signal, removing it without affecting the original signal can be difficult.
- **Crosstalk** is an unwanted coupling between two (2) different signal paths. This unwanted coupling can be electrical, as might occur between two (2) sets of twisted pair wire (as in a phone line), or it can be electromagnetic (as when unwanted signals are picked up by microwave antennas).
- **Echo** is the reflective feedback of a transmitted signal as the signal moves through a medium. It is usually a signal bouncing back from the end of a cable and creating an echo.
- **Jitter** is the result of small timing irregularities that become magnified during the transmission of digital signals as the signals are passed from one device to another. To put it another way, when a digital signal is being transmitted, the rises and falls of the signal can start to shift or become blurry and, thus, produce jitter.
- **Attenuation** is the continuous loss of a signal's strength as it travels through a medium. It is not necessarily a form of error but can indirectly lead to an increase in errors affecting the transmitted signal.

Types of Errors (Forouzan, 2013)

Whenever bits flow from one point to another, they are subject to unpredictable changes because of **interference.** This interference can change the shape of the signal.

- A ***single-bit error*** indicates that only one (1) bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.
- A ***burst error*** indicates that two (2) or more bits in the data unit have changed from 1 to 0 or from 0 to 1. A burst error is more likely to occur than a single-bit error because the duration of the noise signal is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits.
  - For example, if we are sending data at 1 kbps, a noise of 1/100 second can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10,000 bits.

Redundancy (Forouzan, 2013)

The central concept in detecting or correcting errors is **redundancy**. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

Error Detection (Neso Academy, 2020)

Error detection can be performed in the data link layer. When a device creates a frame of data at the data link layer, it inserts some type of error detection code. When the frame arrives at the next device in the transmission sequence, the receiver extracts the error-detection code and applies it to the data frame. Then, the data frame is reconstructed and sent to the next device in the transmission sequence.

- **Parity Check (Vertical Redundancy Check) -** A simple method of error detection is by adding redundant bits called parity bits to each character.
  - This method is called parity checking and is commonly used for ASCII characters where seven (7) bits are used for actual character encoding, and the eighth bit is for parity.
  - It comes in two (2) basic forms: even parity and odd parity. The basic concept of parity checking is that a bit is added to a string of bits to create either even parity or odd parity.
  - With **even parity,** the 0 or 1 added to the string produces an even number of binary 1s.
  - With **odd parity,** the 0 or 1 added to the string produces an odd number of binary 1s.

Example: Find each character block with its parity bit, if the condition is Even Parity is set to '0' and Odd Parity is set to '1'. Use the string "AIMHigh".

	Letter	ASCII Binary Equivalent	Number of Bits	Parity	Parity Bit Set	ASCII Binary with Parity Bit
1	**A**	1000001	2	Even	**0**	1000001**0**
2	**I**	1001001	3	Odd	**1**	1001001**1**
3	**M**	1001101	4	Even	**0**	1001101**0**
4	**H**	1001000	2	Even	**0**	1001000**0**
5	**i**	1101001	4	Even	**0**	1101001**0**
6	**g**	1100111	5	Odd	**1**	1100111**1**
7	**h**	1101000	3	Odd	**1**	1101000**1**

- **Two-Dimensional Parity (Longitudinal Redundancy Check)** - In the two-dimensional parity check, blocks of data are organized as a two-dimensional array. This method increases the likelihood of detecting burst errors.
  - Specifically, each row of the array is a data block that is to be transmitted.
  - A parity bit is appended to each column based on if even or odd parity is used.

Example: Find the LRC of the data block **'11100111' '11011101' '00111001' '10101001'**. Determine the data movement. Set condition: **odd parity to 1** and **even parity to 0**.

Step 1. Determine the LRC by counting parity bits by column.

1st Block	1	1	1	0	0	1	1	1
2nd Block	1	1	0	1	1	1	0	1
3rd Block	0	0	1	1	1	0	0	1
4th Block	1	0	1	0	1	0	0	1
Parity Bits	**3**	**2**	**3**	**2**	**3**	**2**	**1**	**4**
LRC	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**

Step 2. Append all data blocks together with the LRC.

$\xrightarrow{\hspace{6cm}}$

Direction of movement

10101010	**10101001**	**00111001**	**11011101**	**11100111**
LRC	4th Block	3rd Block	2nd Block	1st Block
	Data			

Note: Both end devices must agree with the LRC. To check the validity of each bit sent, recheck each column **(see Step 1)** with the agreed LRC to determine if data received has no error.

- **Arithmetic Checksum** - A checksum is a sequence of numbers and letters used to check data for errors. This sum is then added to the end of the message, and the message is transmitted to the receiving end.
  - Many higher-level protocols used on the Internet (such as TCP and IP) use a form of error detection in which the characters to be transmitted are "summed" together.
  - It has two (2) sides: **sender's side (checksum creation)** and **receiver's side (checksum validation)**

Example: Find the checksum and validate if there is no data corruption in the data block. Consider the data unit to be sent: 10011001111000100010010010000100

SENDER'S SIDE (CHECKSUM CREATION)

Step 1. Divide the data unit into 8 bits starting from right to left.

10011001 11100010 00100100 10000100

Step 2. Perform Binary Addition in all groups.

Carry			**1**	**1**	**1**	**1**	**1**		
			1	0	0	0	0	1	0	0
			0	0	1	0	0	1	0	0
			1	1	1	0	0	0	1	0
+			1	0	0	1	1	0	0	1
	1	**0**	0	0	1	0	0	0	1	1

Step 3. Add the 8-bit excess in the next line and get its 1's complement (binary inversion) to get the checksum.

	1	0	0	0	0	1	0	0
	0	0	1	0	0	1	0	0
	1	1	1	0	0	0	1	0
	1	0	0	1	1	0	0	1
	0	0	1	0	0	0	1	1
+							**1**	**0**
	0	0	1	0	0	1	0	1
1's complement (CHECKSUM)	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Step 4. Add the checksum to the leftmost part of the data unit.

11011010 10011001 11100010 00100100 10000100

RECEIVER'S SIDE (CHECKSUM VALIDATION)

Step 5. Perform Binary Addition in all data unit groups. Add the excess bits into itself.

Note: If the answers are all 1's, then there is no data transmission error.

		1								
Carry		**1**	**1**	**1**	**1**	**1**	**1**	**1**		
			1	1	0	1	1	0	1	0
			1	0	0	0	0	1	0	0
			0	0	1	0	0	1	0	0
			1	1	1	0	0	0	1	0
+			1	0	0	1	1	0	0	1
	1	**0**	1	1	1	1	1	1	0	1

	1								
Carry	**1**	**1**	**1**	**1**	**1**	**1**	**1**		
		1	1	0	1	1	0	1	0
		1	0	0	0	0	1	0	0
		0	0	1	0	0	1	0	0
		1	1	1	0	0	0	1	0
		1	0	0	1	1	0	0	1
		1	1	1	1	1	1	0	1
+								**1**	**0**
No Data Corruption		**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

- **Cyclic Redundancy Checksum** – This method treats the packet of data to be transmitted (the message) as a large polynomial, which adds 8 to 32 check bits to large data packets and yields an error-detection capability, approaching 100 percent.
  - Review the **XOR function** and **Binary Division**.
  - The remainder of the binary division is the CRC to be appended to the original message.

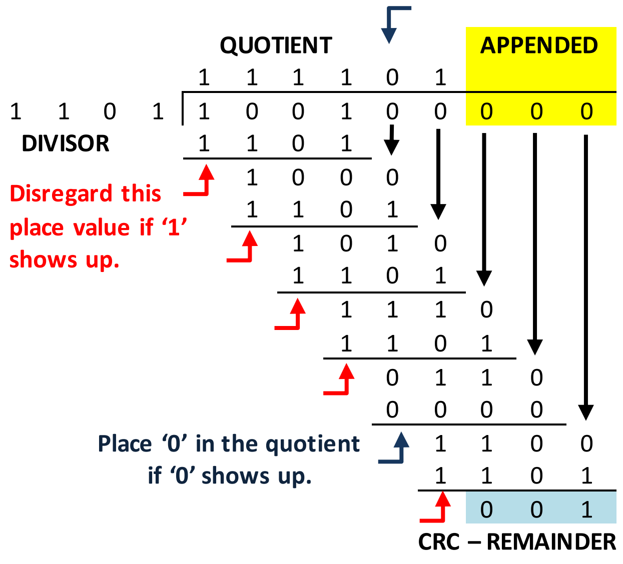
Example: Find the CRC for the data blocks 100100 with the divisor 1101.

SENDER'S SIDE (CHECKSUM CREATION)

Step 1: Identify the length of the divisor (L) and how many zeros to be appended using L-1. Append zeros in the data block.

Original Data block	100100
Divisor (L)	**1101**
L	**4**
Appended Zeros (L - 1)	**4 – 1 = 3 zeros**
Appended Data block	100100**000**

Step 2: Perform binary division. Append the remainder in the original data block.



Remainder - CRC	**001**
Appended CRC	100100**001**

RECEIVER'S SIDE (CHECKSUM VALIDATION)

Step 3. Perform binary division with the appended data block with the CRC.

```
                      1   1   1   1   0   1
  1   1   0   1   |   1   0   0   1   0   0   0   0   1
                      1   1   0   1   ↓   ↓   ↓   ↓   ↓
                      ---------------
                          1   0   0   0
                          1   1   0   1
                          ---------------
                              1   0   1   0
                              1   1   0   1
                              ---------------
                                  1   1   1   0
                                  1   1   0   1
                                  ---------------
                                      0   1   1   0   1
                                          1   1   0   1
                                          -------------
                                              0   0   0
```

Disregard this place value if '1' shows up.

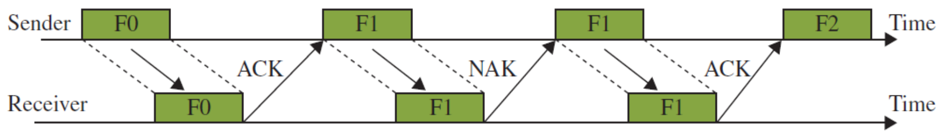
Place '0' in the quotient if '0' shows up.

No Data Corruption

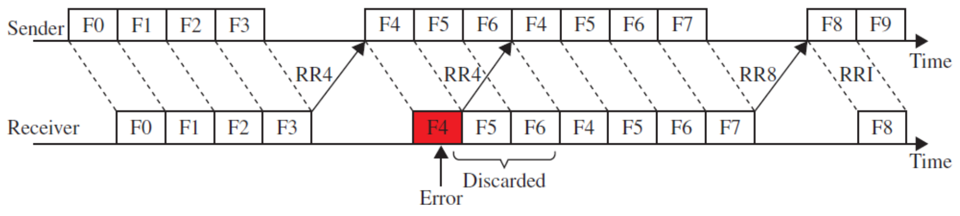
Error Correction (Ibe, 2018)

The number of errors and the size of the message are important factors in dealing with error correction. These mechanisms deal with *error correction by retransmission.* When an error occurs, the receiver will detect it and inform the source, and the source will have to retransmit the frame either on **positive acknowledgment (ACK)** or **negative acknowledgment (NAK)**. Three (3) error control schemes based on the ***automatic repeat request (ARQ)*** are used depending on how retransmission is done when errors occur:

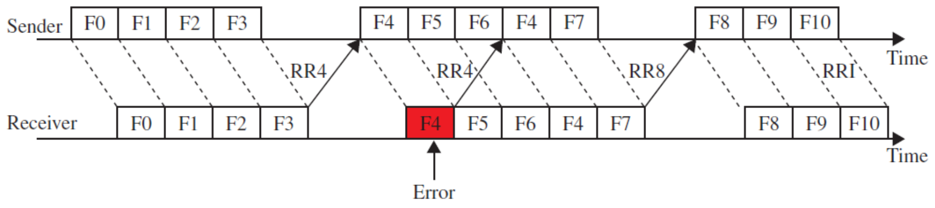
- **Stop-and-Wait ARQ -** is used to deal with errors that occur when the stop-and-wait flow control protocol is used.
  - Errors can occur in four ways:
    - The frame was corrupted in transit when going from source to sink.
    - The frame was OK, but the ACK was corrupted in transit.
    - The frame was lost in transit.
    - The ACK was lost in transit.
  - Under the stop-and-wait ARQ, the source sends a frame and waits for for a response, which can be an ACK or a NAK that can also be in the form of a timeout.
    - When a NAK is received, the source resends the frame and keeps resending it until it receives an ACK after the frame has been correctly received at the destination.
    - Some protocols permit a fixed maximum number of retransmissions. After the permission of retransmission is finished, the link is discarded and set to unusable.



- **Go-Back-*N* ARQ -** deals with errors that occur when the sliding window protocol is used.
  - When a NAK is received for a particular frame, the source resends that frame and all the frames that have been transmitted since that frame was sent as well as any new frames, provided the total number of frames being sent does not exceed *N*.



- **Selective Repeat ARQ.** Under this scheme, only the frame in error is retransmitted. The drawback is that the receiver must provide enough buffer to store the frames that were transmitted after the erroneous frame until the frame has been retransmitted.
  - The reason for this is that the destination must resequence the frames and deliver them in the same order that they appear at the source. Thus, until the errored frame has been retransmitted and correctly received, the frames that are not errored must be stored in the buffer at the receiver.



References:
Forouzan, B. (2013). *Data communications and network.* McGraw-Hill.
Ibe, O. (2018). *Fundamentals of data communication networks* (1st ed.). Wiley & Sons, Inc.
Kurose, F., & Ross, K. (2017). *Computer networking: A top-down approach* (7th ed.). Pearson.
Neso Academy. (2020, March 24). *Vertical redundancy check (VRC)* [Video]. YouTube. https://www.youtube.com/watch?v=UwERCzJv-y8
Neso Academy. (2020, March 25). *Longitudinal redundancy check (*LRC) [Video]. YouTube. https://www.youtube.com/watch?v=nNONvBsOtrE
Neso Academy. (2020, March 26). *Checksum* [Video]. YouTube. https://www.youtube.com/watch?v=AtVWnyDDaDI
Neso Academy. (2020, March 27). *Cyclic redundancy check (CRC) - Part 1* [Video]. YouTube. https://www.youtube.com/watch?v=A9g6rTMblz4
Neso Academy. (2020, March 28). *Cyclic redundancy check (CRC) - Part 2* [Video]. YouTube. https://www.youtube.com/watch?v=wQGwfBS3gpk
White, C. (2015). *Data communications and computer networks - A business user's approach.* Cengage