Unit V

1. Data Communication Codes

A **Data Communication Character Code** is a set of standardized rules used to represent and transmit characters (like letters, digits, punctuation marks, and symbols) in communication systems. These codes enable different devices and systems to exchange information effectively by ensuring that transmitted data can be understood and correctly interpreted at the receiving end.

1. Data-Link Control Characters

These characters are specifically used to **control the flow of data** in a communication system. They help ensure that data blocks are transmitted and received in an orderly and structured manner. These characters are not printable; instead, they perform specific control functions.

Key Functions:

- Indicate the start or end of a transmission.
- Help receivers identify the boundaries of a message or data block.
- Facilitate error control and synchronization between sender and receiver.

Examples:

1. **STX (Start of Text)**:
    - Marks the beginning of a data block in communication.
    - Helps the receiver recognize when the actual content of a message starts.
2. **ETX (End of Text)**:
    - Indicates the end of a data block.
    - Helps the receiver know where the transmitted message ends.

Use Case: In protocols like **RS-232 serial communication**, STX and ETX are used to define the boundaries of data packets.

2. Graphic Characters

Graphic characters are **visible and printable characters** used to control the layout and formatting of text on screens, printers, or other output devices. These characters do not represent data content but influence how data is displayed or aligned.

Key Functions:

- Enhance the presentation of text for readability.
- Control spacing, alignment, and organization of text.

Examples:

1. **VT (Vertical Tab)**:
    - Moves the cursor or print head vertically to the next tab stop.
    - Often used in formatted text or reports to create vertical spacing.
2. **HT (Horizontal Tab)**:
    - Moves the cursor or print head horizontally to the next tab stop.
    - Used for creating consistent spacing between columns or text blocks.
3. **NL (New Line)**:
    - Moves the cursor or print head to the beginning of the next line.
    - Commonly used in text files to indicate line breaks.

Use Case: These characters are commonly used in programming, text editors, and printers for creating neatly formatted documents or output.

3. Alphanumeric Characters

Alphanumeric characters are the **content-bearing characters** that represent data like letters, numbers, and symbols. These characters are fundamental to human-readable communication and data storage.

Key Functions:

- Represent text and numerical data.
- Include punctuation and special symbols for creating structured content.
- Encode meaningful information for communication or processing.

Examples:

1. **Letters**:
    - Uppercase: `A, B, C... Z`
    - Lowercase: `a, b, c... z`
2. **Numbers**:
    - Digits: `0, 1, 2... 9`
3. **Punctuation Marks**:
    - Symbols like `, . ? ! : ; -`
4. **Special Characters**:
    - Characters like `@, #, $, &, *`

Use Case: Alphanumeric characters are the backbone of everyday data representation, such as writing text (in emails, documents) or displaying numerical data (prices, IDs, etc.).

Combined Role in Data Communication

- **Data-Link Control Characters** ensure the message is sent and received correctly.
- **Graphic Characters** enhance readability and presentation of the data.

- **Alphanumeric Characters** carry the actual information being communicated.

By combining these types of characters, data communication systems ensure **efficiency**, **accuracy**, and **clarity** in transmitting and presenting data.

Aspect	Data-Link Control Characters	Graphic Control Characters	Alphanumeric Characters
Purpose	To control the flow and management of data between sender and receiver.	To control the formatting and layout of text on output devices like monitors or printers.	To represent meaningful information like letters, numbers, and symbols.
Type of Character	Non-printable (used for communication protocol control).	Non-printable (used for layout and display control).	Printable (visible on output devices).
Role in Communication	Ensures orderly data transfer by marking the start, end, or control of a message.	Aligns, spaces, or organizes text presentation on a device.	Encodes content like text and numbers for human-readable communication.
Examples	- **STX (Start of Text)**: Marks the start of a message.	- **HT (Horizontal Tab)**: Adds horizontal spacing.	- Letters: `A`, `B`, `Z`, etc.
	- **ETX (End of Text)**: Marks the end of a message.	- **VT (Vertical Tab)**: Moves vertically in output.	- Numbers: `0`, `1`, `2`, etc.
	- **ACK (Acknowledgment)**: Confirms successful receipt of data.	- **NL (New Line)**: Moves to the beginning of a new line.	- Symbols: `!`, `@`, `#`, etc.
Visibility	Not visible to users; operates behind the scenes.	Not visible as characters but their effects (like spacing) are visible.	Always visible to users when printed or displayed.
Usage Context	Used in communication protocols to define or manage data transmission.	Used in text editors, printers, and systems for layout control.	Used in documents, messages, and databases to represent data content.
Examples in ASCII	ASCII codes 0–31, such as:	ASCII codes 9–13, such as:	ASCII codes 32–127, such as:
	- `STX` (Decimal: 2)	- `HT` (Decimal: 9)	- `A` (Decimal: 65), `1` (Decimal: 49), `!` (Decimal: 33)
	- `ETX` (Decimal: 3)	- `NL` (Decimal: 10)	
	- `ACK` (Decimal: 6)	- `VT` (Decimal: 11)	

Bar code

Barcodes: An Explanation

Barcodes are an essential tool for automated data capture and processing in many industries, enabling fast and accurate identification of items.

What is a Barcode?

A **barcode** is a series of black and white vertical bars or spaces used to encode information, typically in binary format (`1s` and `0s`). These codes can represent data about items like cost, inventory details, or product identification.

- The **black bars** and **white spaces** differ in width, representing binary data.
- They are commonly scanned using optical scanners or barcode readers, which detect light reflectivity to decode the information.

Components of a Barcode

1. **Start Field**:
    - A unique sequence of bars and spaces at the beginning of the barcode.
    - It allows the scanner to identify where the barcode starts.
2. **Data Characters**:
    - This section contains the encoded information, such as the item's identifier, price, or other data.
    - It is extracted and interpreted by the scanner.
3. **Check Characters**:
    - Used for error detection and verification to ensure the data was read correctly.
    - Helps validate the accuracy of the scanned data.
4. **Stop Field**:
    - Indicates the end of the barcode, helping the scanner recognize the conclusion of the data.

How Barcodes Work

1. A scanner emits light onto the barcode.
2. The black bars absorb the light, while the white spaces reflect it.
3. The scanner interprets the pattern of reflected light as an electrical signal and decodes it into binary information (`1s` and `0s`).
4. The decoded binary data is processed into meaningful information.

Types of Barcode Formats

Barcodes are classified into three primary formats based on the data they encode and their design:

1. **Discrete Code**:
    - Each character within the barcode is independent of the others.
    - Spaces (gaps) exist between the characters.
    - Example: **Code 39**.
2. **Continuous Code**:
    - No spaces or gaps exist between the characters.
    - The barcode is read as a continuous stream.
    - Example: **Universal Product Code (UPC)**.
3. **2D Code**:
    - Encodes data in two dimensions (horizontal and vertical).
    - Stores significantly more data compared to one-dimensional barcodes.
    - Typically, 2D barcodes store up to 1 kilobyte of data or more per symbol.
    - Example: **QR Codes**.

Applications of Barcodes

1. **Retail and Inventory Management**:
    - Track products and automate billing.
    - Example: Barcodes on groceries scanned at checkout.
2. **Shipping and Logistics**:
    - Used for package tracking and routing.
3. **Security and Identification**:
    - Secure access control systems.
4. **Healthcare**:
    - Barcode labels on medical records, prescriptions, and samples.

Advantages of Barcodes

- **Speed**: Scanning is faster than manual entry.
- **Accuracy**: Reduces human errors during data input.
- **Cost-Effectiveness**: Simple and inexpensive to implement.
- **Versatility**: Can be used in various industries.

==========

Error Control

Error control refers to techniques used in data communication systems to detect and correct errors that may occur during the transmission of data over a communication channel. Since communication channels are not perfect, errors can arise due to interference, noise, or signal degradation. Effective error control ensures that data reaches its destination accurately.

Sources of Errors

Errors in data communication are caused by:

1. **Natural Sources**:
    - Lightning
    - Solar flares
2. **Man-Made Sources**:
    - Motors
    - Generators
    - Power lines
    - Fluorescent lights

These interferences disrupt the communication medium, introducing errors into the transmitted data.

Types of Transmission Errors

Transmission errors are generally classified into three categories:

1. Single-Bit Errors

- **Definition**: When only one bit within a data string is corrupted (flipped from `0` to `1` or `1` to `0`).
- **Impact**:
    - Only a single character or part of the transmitted message is affected.
    - Common in channels with low noise levels.

2. Multiple-Bit Errors

- **Definition**: When two or more non-consecutive bits in a data string are in error.
- **Impact**:
    - Affects multiple parts of the transmitted message.
    - Can lead to loss of significant portions of information.
    - Occurs in channels with higher levels of interference or noise.

3. Burst Errors

- **Definition**: When two or more consecutive bits within a data string are corrupted.
- **Impact**:
    - Affects several bits or characters at once.
    - Often caused by prolonged interference, such as power surges or signal fades.
    - More challenging to detect and correct due to the concentration of errors.

Error Detection in Data Communication

Error detection is the process of monitoring transmitted data to identify whether transmission errors have occurred. These techniques do not correct errors or pinpoint which bits are incorrect; they merely indicate that an error has happened. The primary purpose of error detection is to prevent undetected errors from affecting the communication process

Purpose and Need for Error Detection

- Errors in data transmission can arise from interference, noise, or other disruptions in the communication medium.
- Error detection ensures the reliability and integrity of transmitted data by signaling the presence of errors so that corrective measures can be taken (e.g., retransmission).

Techniques of Error Detection

The most commonly used error-detection techniques include:

1)Redundancy in Error Detection

Definition:
Redundancy is a method used in error detection where extra data is transmitted alongside the original message. This additional data provides a way for the receiver to verify the accuracy of the received information. Redundancy can involve sending either multiple copies of a single character (character redundancy) or the entire message (message redundancy).

Types of Redundancy

1. Character Redundancy

- **Explanation:**
  In character redundancy, each character in the data stream is transmitted **twice**. The receiver compares the two copies of each character to ensure they are identical.
- **How it Works:**
  - At the **sender's end**, each character (e.g., 'A') is transmitted twice (e.g., 'AA').
  - At the **receiver's end**, the two received characters are compared:
    - **If they match**, the transmission is assumed to be error-free.
    - **If they don’t match**, an error is detected, and the sender may be asked to retransmit the data.
- **Example:**
  - **Data Sent:** A, B, C (transmitted as AA, BB, CC).
  - **Data Received:** If the receiver gets AA, BB, CC, it assumes the transmission is correct.
  - **Error Scenario:** If the receiver gets AA, BC, CC, the mismatch in 'B' and 'C' will indicate an error.

Use Cases of Character Redundancy:

- It is useful in systems where single-character errors are more common, such as noisy communication lines or simple data streams.
- Example applications include basic control systems or legacy communication systems.

---

2. Message Redundancy

- **Explanation:**
  In message redundancy, the **entire message** is sent multiple times instead of just individual characters. The receiver compares the repeated copies of the message to identify errors.
- **How it Works:**
  - At the **sender's end**, the same message is transmitted multiple times (e.g., 3 copies of the message).
  - At the **receiver's end**, all the received messages are compared:
    - **If they are identical**, the transmission is considered error-free.
    - **If there is a mismatch**, an error is detected.
- **Example:**
  - **Data Sent:** "HELLO" (transmitted 3 times as "HELLO", "HELLO", "HELLO").
  - **Data Received:** If the receiver gets "HELLO", "HELLO", "HELLO", it assumes the transmission is correct.
  - **Error Scenario:** If the receiver gets "HELLO", "HELO", "HELLO", the mismatched second message identifies an error.

Use Cases of Message Redundancy:

- Used in systems where longer messages are sent and higher reliability is required.
- It is particularly beneficial for short but critical messages in applications like:
  - Emergency communications.
  - Remote control systems.
  - Sensor-based transmissions.

Key Characteristics of Redundancy

- **Error Detection Only:**
  Redundancy methods detect errors but do not correct them. If errors are found, retransmission is required.
- **Resource Usage:**
  Sending additional characters or entire messages increases **bandwidth usage** and transmission time.
- **Reliability:**
  The more copies transmitted, the greater the likelihood of detecting errors.

Advantages

1. **Simplicity:**
   - Easy to implement and understand.
2. **High Reliability:**
   - Works effectively for detecting errors in systems prone to noise or interference.
3. **Real-Time Feedback:**
   - Errors can be detected as soon as data is received.

Limitations

1. **Bandwidth Overhead:**
   - Doubling or tripling the data sent increases the bandwidth requirement.
2. **Time Delay:**
   - Repeated transmissions consume time, especially for large messages.
3. **No Error Correction:**
   - Redundancy only detects errors; it cannot fix them. Retransmission is needed to correct errors.

Comparison: Character vs. Message Redundancy

Feature	Character Redundancy	Message Redundancy
Unit of Redundancy	Individual characters	Entire message
Data Overhead	Less (duplicates only characters)	Higher (duplicates full message)
Error Detection	Detects single-character errors	Detects message-level errors
Use Case	Simple, low-speed systems	Critical and high-reliability data transmission

Echo Checking (Echoplex) in Error Detection

Definition: Echo checking, also known as **echoplex**, is a simple and effective error-detection technique used in real-time data communication systems. It ensures the accuracy of transmitted data by verifying that what is received at the destination matches the data sent from the source. This is done by "echoing" or retransmitting the received data back to the sender.

How Echo Checking Works

1. **Data Entry at Sender Side:**

    - The sender (usually a human operator) types data into a terminal or system.
2. **Transmission of Data:**
    - The typed characters are transmitted to the receiving terminal.
3. **Echo Back (Verification Step):**
    - Once the receiving terminal gets the data, it retransmits (or echoes) the exact same data back to the sender.
4. **Comparison at Sender Side:**
    - The sender's system compares the echoed data with the original input:
        - **If the echoed data matches** the original input: The sender assumes the transmission was successful.
        - **If the echoed data does not match** the original input: An error is detected, and the sender requests retransmission.

Example Scenario

Suppose a user types the word **"HELLO"**:

1. **Sender Side:**
    - The word **"HELLO"** is typed on the sender's keyboard and transmitted character by character.
2. **Receiving Side:**
    - The receiver gets each character (H, E, L, L, O) and sends it back to the sender for verification.
3. **Verification:**
    - If the echoed characters match what was typed, the communication is confirmed error-free.
    - If the echoed characters differ (e.g., **"HELLO"** is received as **"HELO"** due to an error), the sender identifies a discrepancy and resends the data.

Key Characteristics of Echo Checking

1. **Full-Duplex Communication:**
    - Echo checking requires a full-duplex communication channel where data can flow in both directions simultaneously.
2. **Immediate Verification:**
    - Errors are detected as soon as the character is echoed back, making it a real-time process.
3. **Simple Implementation:**
    - Since each character is verified individually, the process is straightforward and easy to implement.
4. **Human Involvement:**
    - It is often used in systems where human operators are involved, such as older computer terminals or point-of-sale systems.

Advantages

1. **Immediate Error Detection:**
    - Errors are caught as soon as they occur, reducing the chances of corrupted data being processed.
2. **Ease of Use:**
    - Simple to understand and implement in low-speed communication systems.
3. **Reliability:**
    - Effective in detecting errors caused by noise or interference in the transmission medium.

Limitations

1. **Inefficiency:**
    - Echo checking can double the time required for data transmission because every character must be sent and then echoed back.
2. **Not Suitable for High-Speed Systems:**
    - In modern high-speed networks, this method is too slow and inefficient.
3. **Limited to Detection:**
    - Echo checking detects errors but does not correct them. It relies on retransmission to fix errors.

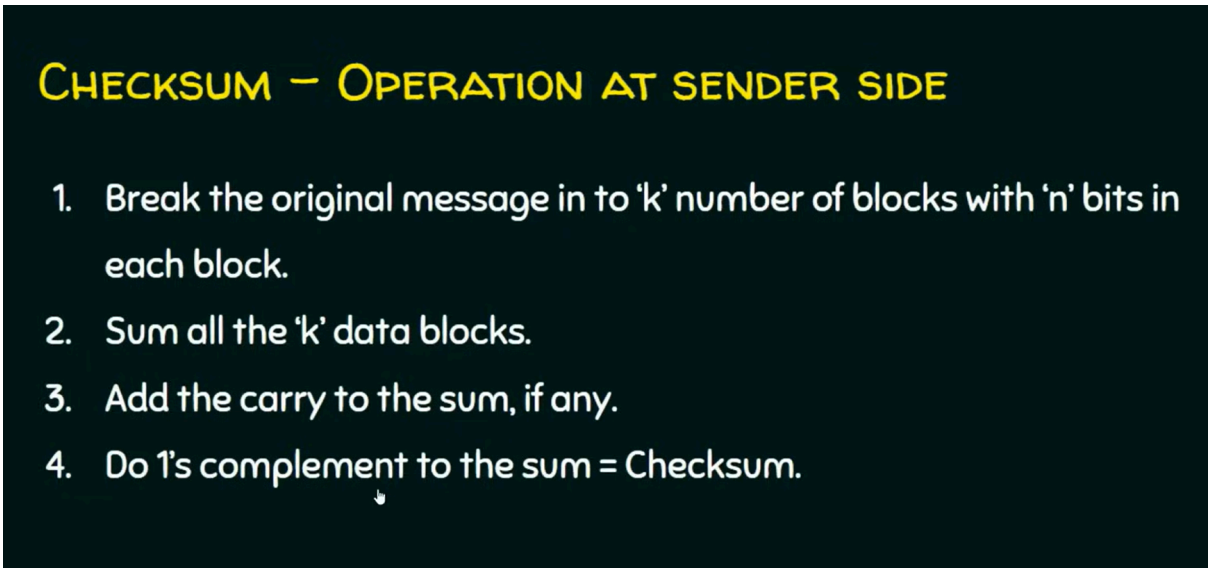
Applications

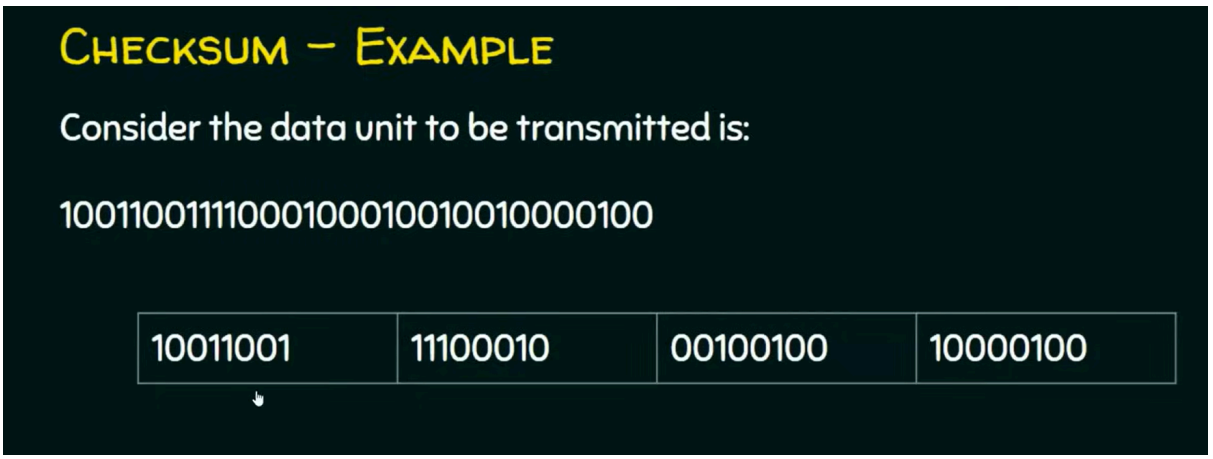
- **Older Communication Systems:**
    - Often used in legacy systems with low data rates.
- **Point-of-Sale (POS) Systems:**
    - Ensures the accuracy of transactions by verifying data input.
- **Simple Terminals:**
    - Used in systems where humans input data manually and real-time feedback is essential.

Echo checking is a fundamental technique for error detection in simpler communication environments, emphasizing reliability and ease of use. However, it has largely been replaced by more advanced error detection and correction methods in modern communication systems.

    -

Check Sum's (Wrote for time pass lite)





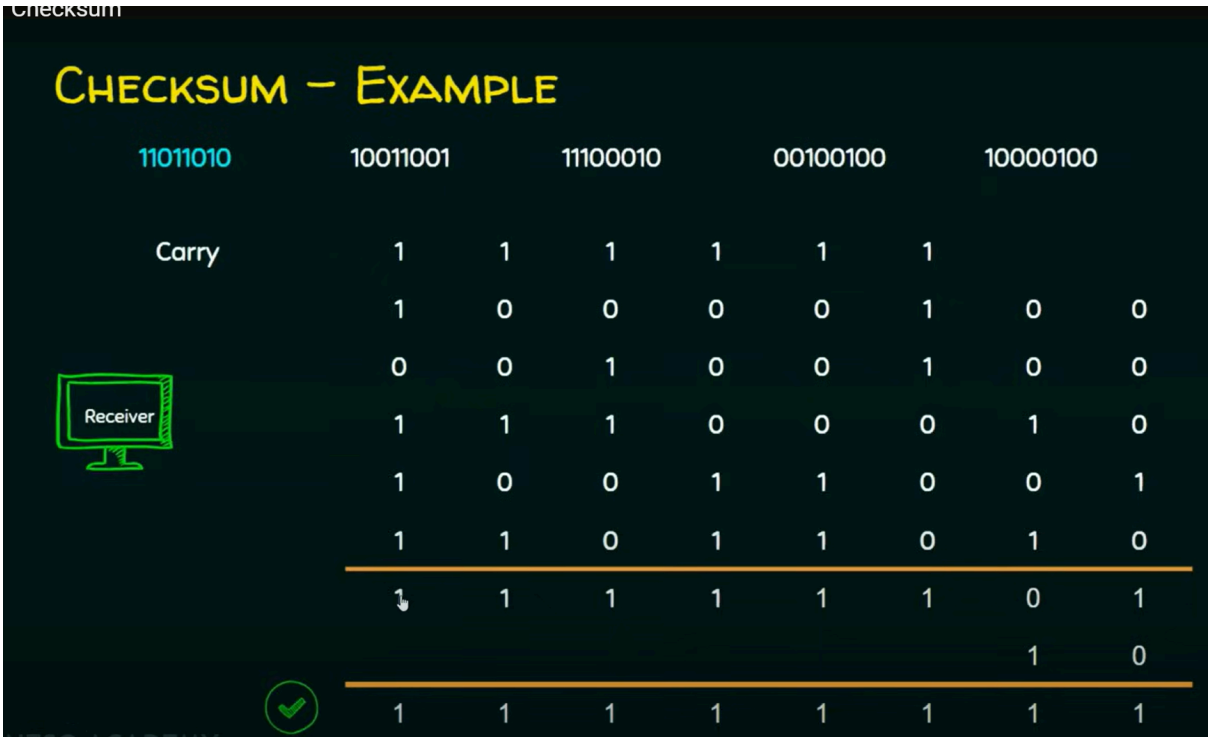
Checksum – Example

	10011001	11100010	00100100	10000100
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			
	0 0 1 0 0 0 1 1			
				1 0
	0 0 1 0 0 1 0 1			
1's Complement	1 1 0 1 1 0 1 0			

After we perform the ones' complement,

Checksum – Operation at receiver side

- ★ Collect all the data blocks including the checksum.
- ★ Sum all the data blocks and checksum
- ★ If the result is all 1's, ACCEPT; Else, REJECT.



Cyclic Redundancy Check (CRC) - Explanation

Cyclic Redundancy Check (CRC) is a method used in computer networks and data storage systems for detecting errors in transmitted data. It uses a mathematical algorithm, specifically **binary division**, to check if any errors have occurred during transmission.

Steps for Error Detection Using CRC:

1. **Appending Zeros to Data**:
    - The original data is extended by appending a string of **n zeros** (where **n** is the number of bits in the divisor minus one). This extended data will be divided by the divisor in the next step.
    - Example: If the data is `11100` and the divisor is `1001` (which is 4 bits long), we append **3 zeros** to the data. The new data becomes `11100000`.
2. **Binary Division (Modulo-2 Division)**:
    - The extended data (`11100000`) is divided by the **divisor** (`1001`) using **binary division**. This process is also known as **modulo-2 division**. In binary division, the division is performed using XOR (exclusive OR) instead of regular subtraction.
3. **Generate the CRC Remainder**:
    - The result of the division is the **remainder**, which is called the **CRC remainder**.
    - In this case, after dividing `11100000` by `1001`, the remainder is `111`.
4. **Replace Zeros with CRC Remainder**:

The CRC remainder (`111`) replaces the zeros that were appended to the original data. The final data unit sent across the network becomes:
Copy code
`11100111`

- This is the **transmitted data**.

Error Detection at the Receiver End (CRC Checker):

1. **Receiving the Data**:
   - The receiver gets the transmitted data (`11100111`), which includes both the original data and the CRC remainder.
2. **Performing the Same Division**:
   - The receiver divides the received data (`11100111`) by the same **divisor** (`1001`) using binary division (modulo-2 division).
   - If the division results in a **remainder of 0**, it indicates that the data was transmitted correctly without any errors.
3. **Accept or Discard the Data**:
   - **No Errors (Remainder = 0)**: If the remainder after division is 0, it means the data is **error-free** and can be accepted by the receiver.
   - **Errors Detected (Remainder ≠ 0)**: If the remainder is non-zero, it means there was an **error in the transmission**, and the data is **discarded**.

Example of CRC Error Detection:

Let's go through an example based on the description you provided.

- **Original Data**: `11100`
- **Divisor**: `1001`

CRC Generation (Sender's Side):

1. **Append Zeros**:
   - Append 3 zeros (one less than the length of the divisor) to the data:
   - `11100` becomes `11100000`.
2. **Modulo-2 Division**:

Perform binary division of `11100000` by `1001`:
scss
Copy code
`11100000 ÷ 1001 = 111 (remainder)`

   -
3. **Final Transmitted Data**:
   - Replace the zeros with the CRC remainder:
   - `11100` + `111` = **`11100111`**

- The final transmitted data is `11100111`.

CRC Checking (Receiver's Side):

1. **Receive the Data**:
   - The receiver gets `11100111`.
2. **Perform Modulo-2 Division**:

Divide `11100111` by `1001`:
scss
Copy code
```
11100111 ÷ 1001 = 0 (remainder)
```

   -
3. **Check the Remainder**:
   - Since the remainder is `0`, the data is **error-free** and is accepted by the receiver.

Example with Error:

Let's say an error occurs during transmission, and the received data is `11101111` (instead of `11100111`).

- **Modulo-2 Division**:

Divide `11101111` by `1001`:
scss
Copy code
```
11101111 ÷ 1001 = 101 (remainder)
```

   -
- **Error Detection**:
   - Since the remainder is `101`, which is non-zero, it means there was an **error** in the transmission. The receiver will discard this data and request a retransmission.

Key Points of CRC:

1. **Efficiency**: CRC is efficient and can quickly detect errors in transmitted data using simple mathematical operations (XOR).
2. **Error Detection**: CRC is particularly good at detecting common types of errors, such as single-bit errors and burst errors.
3. **Reliability**: CRC is widely used in communication protocols (e.g., Ethernet, Wi-Fi, USB) and storage systems (e.g., hard drives, CDs) to ensure data integrity.

Conclusion:

CRC is a powerful error detection technique that helps ensure that data is transmitted correctly by checking for errors at the receiver's end using a mathematical division method. If the remainder from the division is zero, the data is assumed to be error-free. If the remainder is non-zero, an error is detected, and the data is discarded. This method is highly effective for detecting a wide range of transmission error

==========

Error Correction

Error Correction Explained

Error correction is a critical process in data communications and computing that ensures accurate transmission of data, even when errors occur during the transmission. Its primary goal is to detect and correct errors in a message so that the received data is identical to the original data sent.

Why Error Correction is Important:

When data is transmitted over a network or storage medium, it can be corrupted due to factors like electrical interference, noise, or hardware failures. Error correction ensures that these errors do not affect the accuracy of the received data, maintaining the reliability of communications and storage systems.

Types of Errors:

1. **Single-bit Errors:**
   Only one bit in a data block is incorrect. These errors are usually easier to detect and correct.
2. **Burst Errors:**
   Multiple bits in a block are corrupted, often due to noise over a short time. Burst errors are more difficult to handle than single-bit errors.

Techniques for Error Correction:

1. Error-Detecting Codes:

These methods help the receiver detect if an error has occurred but do not correct the error. Examples include:

- **Parity bits:** A simple technique that checks whether the number of 1s in a binary sequence is even or odd.
- **Cyclic Redundancy Check (CRC):** Used in network communications to verify data integrity.
- **Checksums:** Summation of data bits to detect errors.

Action: When an error is detected, the receiver requests the sender to resend the data.

2. Error-Correcting Codes (ECC):

These codes detect and correct errors automatically without needing retransmission. Common types include:

- **Hamming Codes:** Can detect and correct single-bit errors.
- **Reed-Solomon Codes:** Used in CDs, DVDs, and QR codes to correct burst errors.
- **Forward Error Correction (FEC):** Adds redundant data to the message so that the receiver can correct errors without retransmitting.

Two Main Modern Methods of Error Correction:

1. **Retransmission (ARQ - Automatic Repeat reQuest):**
   If an error is detected, the receiver requests the sender to retransmit the corrupted data. This method is reliable but can be inefficient if errors occur frequently.
2. **Forward Error Correction (FEC):**
   Redundant data is sent along with the original message. If an error occurs, the receiver uses this extra information to detect and correct errors. This method is faster because it avoids the need for retransmission, making it useful in real-time communications.

Conclusion:

Error correction ensures the reliability of data transmission by detecting and correcting errors. It is essential for various applications, including computer networks, storage devices, and telecommunications, where data integrity is crucial. Different methods are used depending on the system's requirements, balancing complexity, speed, and reliability.

Retransmission Explained in Detail

Retransmission is an error correction method used in data communications to ensure reliable data transfer. When an error is detected in a received message, the receiver requests the sender to resend (retransmit) the message or the corrupted portion of it. This technique is particularly effective in ensuring data integrity in networks where errors might occur due to interference, signal degradation, or other factors.

How Retransmission Works:

1. **Data Transmission:**
   The sender transmits a data message or frame to the receiver.
2. **Error Detection:**
   The receiver checks the received message for errors using error-detecting codes such as:
   - **Parity bits**
   - **Checksums**
   - **Cyclic Redundancy Check (CRC)**
3. **Error Handling:**
   - **No Errors:** If the message is received correctly, the receiver sends an acknowledgment (ACK) back to the sender.
   - **Errors Detected:** If errors are found, the receiver sends a negative acknowledgment (NACK) or does not send any acknowledgment, prompting the sender to retransmit the message.

Types of Retransmission Protocols:

1. **Automatic Repeat reQuest (ARQ):**
   ARQ is a family of error-control protocols that automatically handle retransmission. The main types of ARQ are:
   - **Stop-and-Wait ARQ:**
     - The sender transmits one frame and waits for an acknowledgment before sending the next frame.
     - **Pros:** Simple to implement.
     - **Cons:** Inefficient for long-distance or high-latency networks because the sender remains idle while waiting.
   - **Go-Back-N ARQ:**
     - The sender can transmit multiple frames before waiting for acknowledgment but maintains a sliding window. If an error is detected, the sender retransmits all frames from the erroneous one onwards.
     - **Pros:** Better utilization of the communication channel.
     - **Cons:** If an error occurs, all subsequent frames must be retransmitted, even if they were received correctly.
   - **Selective Repeat ARQ:**
     - Similar to Go-Back-N but retransmits only the frames that contain errors.
     - **Pros:** More efficient than Go-Back-N because only erroneous frames are retransmitted.
     - **Cons:** Requires more complex buffering at both the sender and receiver.

Key Concepts in Retransmission:

- **Acknowledgment (ACK):**
  A signal sent by the receiver to indicate that the message has been received successfully.
- **Negative Acknowledgment (NACK):**
  Indicates that an error was detected and the message needs to be retransmitted.
- **Timeout Mechanism:**
  If the sender does not receive an acknowledgment within a specific time frame, it assumes the message was lost or corrupted and retransmits it.
- **Sequence Numbers:**
  These are used to keep track of transmitted and received frames, ensuring that data is reconstructed correctly at the receiver's end, especially in cases of retransmission.

Advantages of Retransmission:

- **Reliable Data Transfer:** Ensures that errors do not affect the final message, providing accurate communication.
- **Simple Implementation:** Easier to implement compared to some error-correcting codes.
- **Flexible:** Effective in both wired and wireless communication systems.

Disadvantages of Retransmission:

- **Latency:** Retransmission introduces delays, which can be problematic in real-time applications such as video streaming or voice calls.
- **Increased Traffic:** Retransmitting messages increases the network load, especially if errors occur frequently.
- **Efficiency Issues:** Protocols like Stop-and-Wait ARQ can be inefficient in high-latency networks.

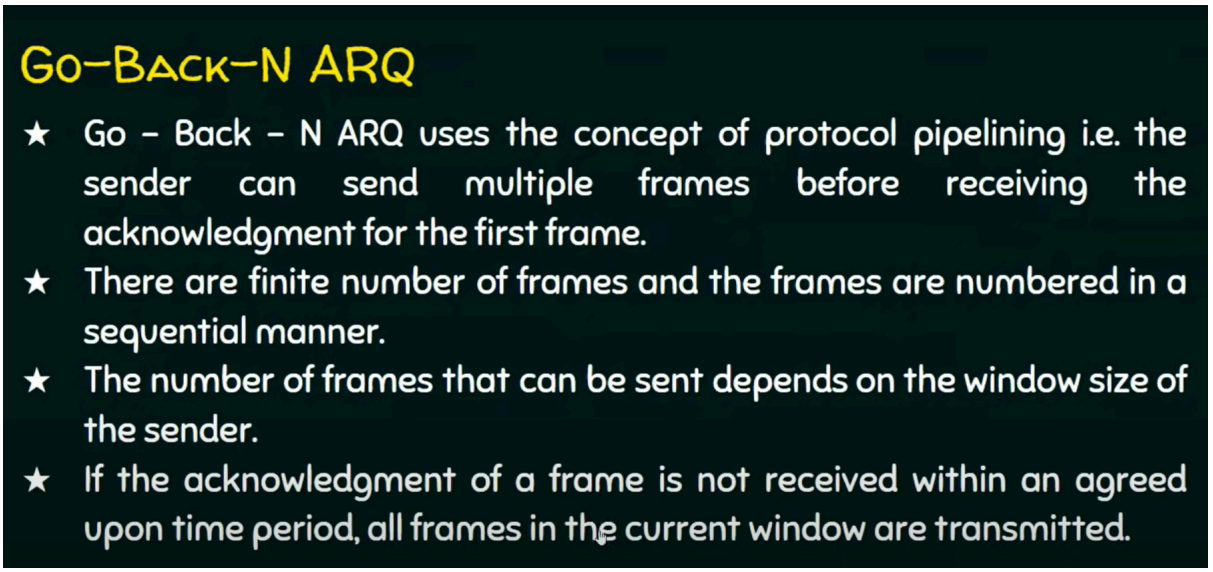
Applications of Retransmission:

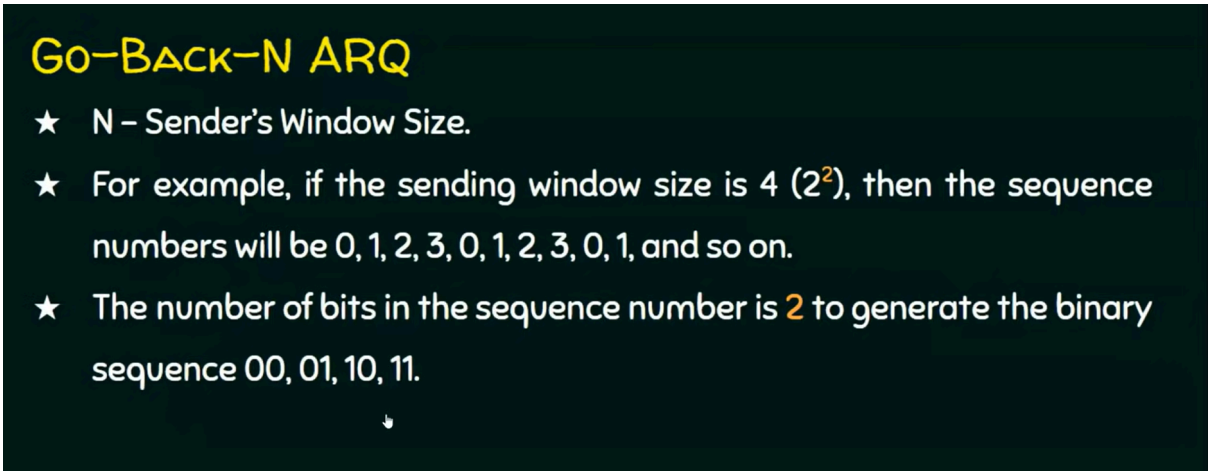
- **Computer Networks:** Used in protocols such as TCP (Transmission Control Protocol) to ensure reliable delivery of packets.
- **Wireless Communication:** Helps mitigate the impact of signal interference.
- **Satellite Communication:** Essential due to the high error rates over long distances.

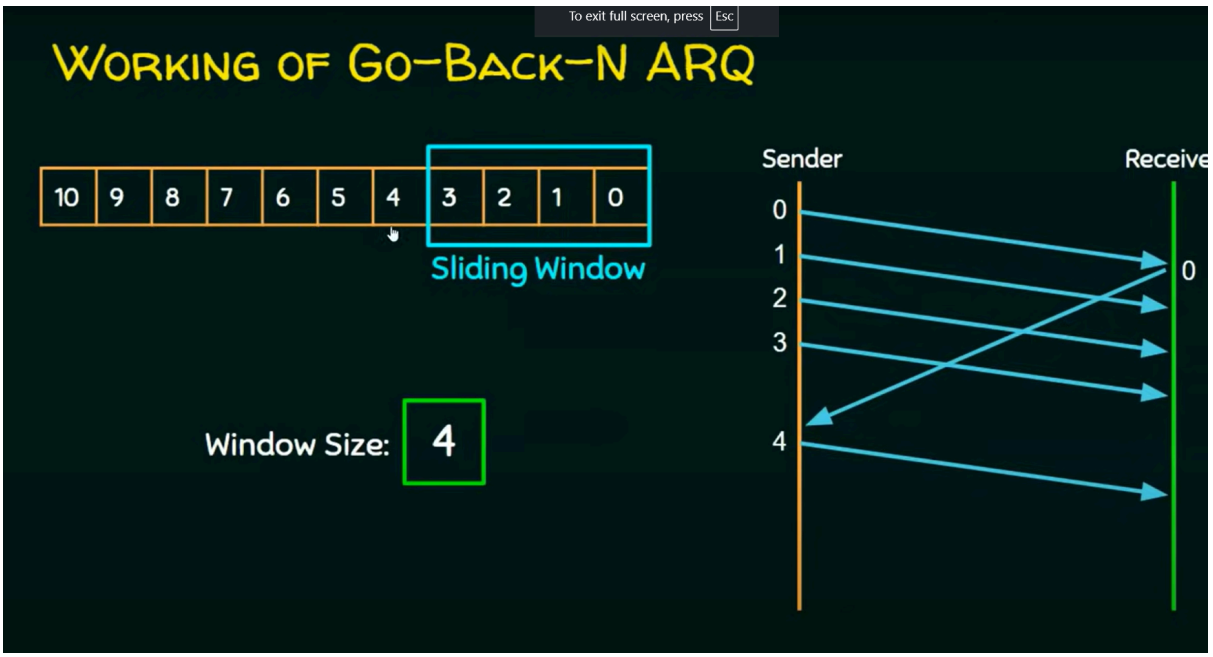
Conclusion:

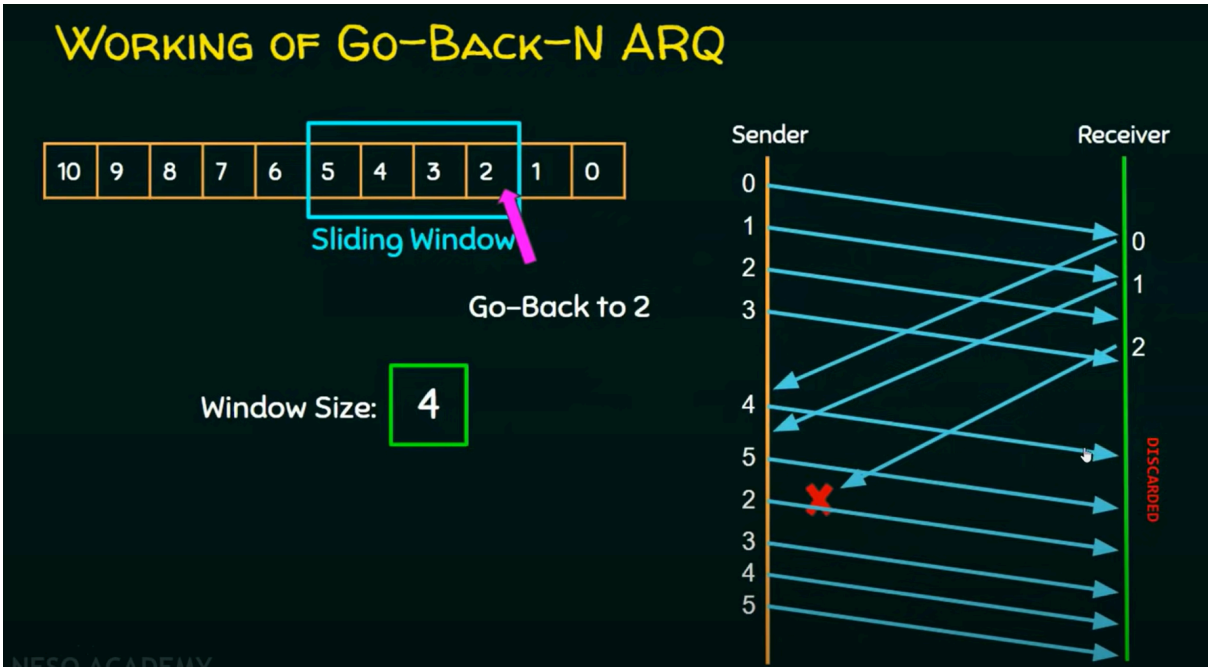
Retransmission is a crucial error correction mechanism in data communications. By allowing the receiver to request the retransmission of erroneous data, it ensures reliable and accurate data transfer. Although it introduces some latency and overhead, its simplicity and effectiveness make it widely used in various communication protocols, ensuring data integrity even in unreliable networks.

Go back N

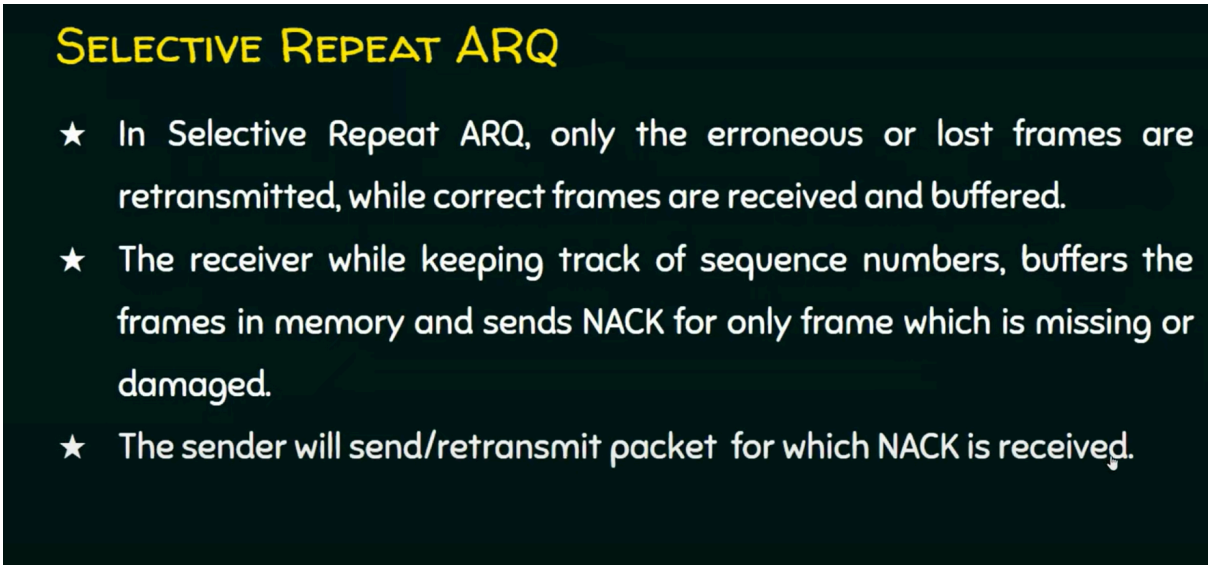


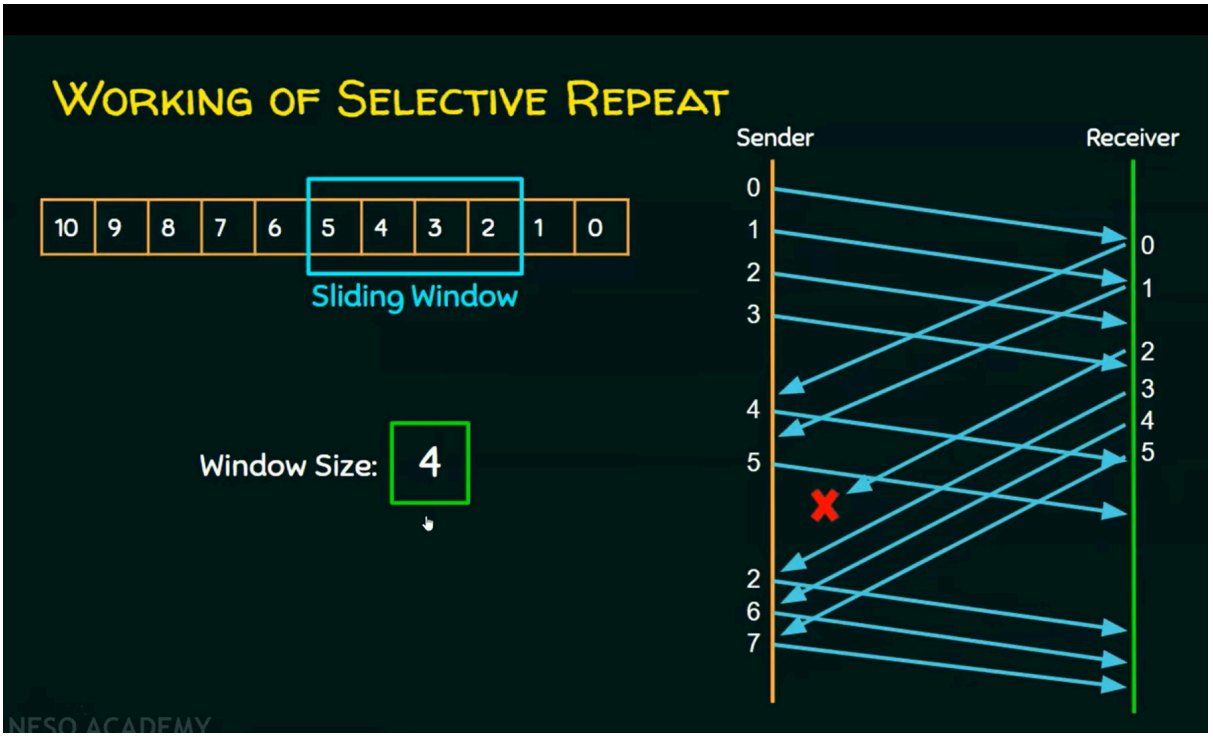



To exit full screen, press Esc
WORKING OF GO-BACK-N ARQ
10 9 8 7 6 5 4 3 2 1 0
Sliding Window
Window Size: 4
Sender
Receive
0 1 2 3 4
0

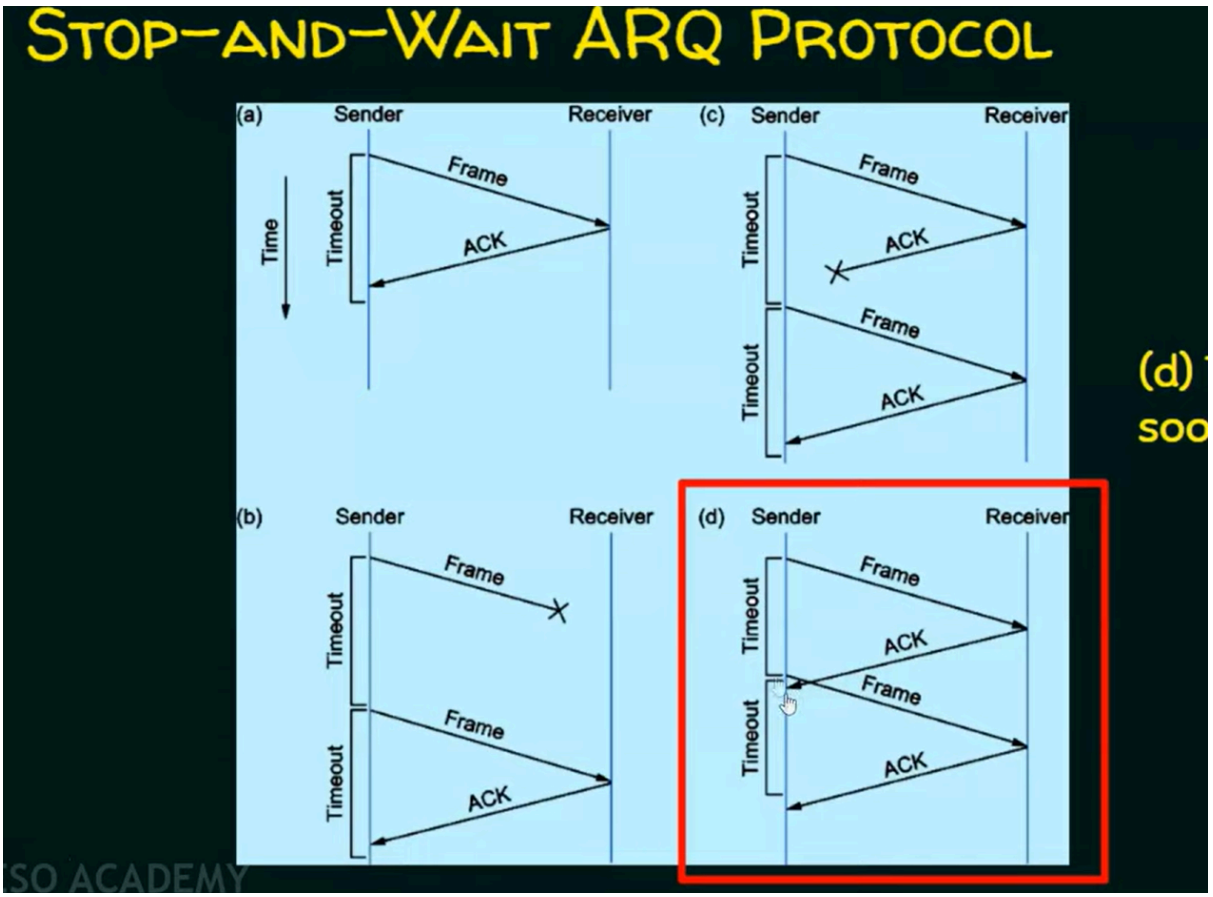

WORKING OF GO-BACK-N ARQ
10 9 8 7 6 5 4 3 2 1 0
Sliding Window
Go-Back to 2
Window Size: 4
Sender
Receiver
0 1 2 3 4 5 2 3 4 5
0 1 2
DISCARDED

Selective Arq





Stop and Wait Arq



character synchronization.

Asynchronous and Synchronous Serial Data Transfer

Serial data transfer refers to sending data one bit at a time over a communication line. The difference between **asynchronous** and **synchronous** transfer lies in how the sender and receiver synchronize their communication.

---

1. Asynchronous Serial Data Transfer

Definition:
In asynchronous transfer, data is sent one character or byte at a time, with no shared clock signal between the sender and receiver. Synchronization occurs at the start of each character using start and stop bits.

Key Characteristics:

1. **Start and Stop Bits**:
    - Each character is preceded by a **start bit** (logic 0) and followed by one or more **stop bits** (logic 1).
    - The start bit tells the receiver that a new character is being transmitted, and the stop bits ensure that the line returns to its idle state before the next character.
2. **Idle Line**:
    - When no data is being sent, the line remains in an **idle state** (logic 1).
    - This creates a gap between transmitted characters.
3. **Data Framing**:
    - Each character is framed independently with start and stop bits.
    - Optional **parity bits** can be used for basic error detection.
4. **Clock Independence**:
    - The transmitter and receiver operate on separate clocks.
    - Synchronization happens only during the transmission of each character, making timing errors (e.g., clock slippage) possible.
5. **Transmission Speed**:
    - Generally slower because of the additional start and stop bits.
    - Idle times between characters can further reduce efficiency.

Advantages:

- Simple to implement; no need for a shared clock.
- Suitable for low-speed communication over long distances.
- Works well for systems with inconsistent or unpredictable timing requirements.

Disadvantages:

- Less efficient due to the overhead of start/stop bits.
- Susceptible to clock slippage errors when transmitter and receiver clocks drift.

Use Cases:

- RS-232 communication (used in serial ports).
- Older modems and low-speed devices (e.g., keyboards, sensors).

---

2. Synchronous Serial Data Transfer

Definition:
In synchronous transfer, data is sent as a continuous stream, synchronized by a shared clock signal between the transmitter and receiver. The data is grouped into blocks or frames, eliminating the need for start/stop bits.

Key Characteristics:

1. **Shared Clock**:
    - Both the transmitter and receiver are synchronized using a **common clock signal**.
    - This ensures that data is sent and received at the same rate without requiring special framing.
2. **Data Framing**:
    - Data is sent in **blocks or frames**, not individual characters.
    - Each frame may include headers, data, and an error-checking mechanism (e.g., CRC).
3. **Continuous Data Stream**:
    - Data is transmitted without gaps, making synchronous transfer more efficient for high-speed communication.
4. **Higher Complexity**:
    - Requires more complex hardware to generate and synchronize the clock signal.
    - Ensuring precise timing is critical.
5. **Error Detection**:
    - Often employs robust error-detection methods like CRC or checksum for reliability.

Advantages:

- Higher efficiency and speed because there are no start/stop bits and minimal idle time.
- Ideal for high-speed communication where large amounts of data need to be transmitted.

Disadvantages:

- Requires complex hardware for clock synchronization.
- Less flexible for irregular or sporadic data transmission.

Use Cases:

- Ethernet communication.
- SPI (Serial Peripheral Interface) and I²C (Inter-Integrated Circuit) protocols.
- USB (Universal Serial Bus) and other high-speed data transfer systems.

Comparison of Asynchronous and Synchronous Serial Data Formats

Aspect	Asynchronous	Synchronous
Clock Synchronization	Does **not require a shared clock**; synchronization is done at the start of each character using start and stop bits.	Requires a **shared clock signal** between transmitter and receiver for continuous synchronization.
Data Framing	Data is framed with a **start bit** (logic 0) and one or more **stop bits** (logic 1).	No start/stop bits are used; data is sent as a **continuous stream** in blocks or frames.
Efficiency	Less efficient due to the overhead of start/stop bits for each character.	More efficient since data is sent in continuous blocks with less overhead.
Speed	Typically slower because of idle line gaps and additional framing bits.	Faster because it transmits data continuously without pauses.
Error Handling	Optional parity bit for simple error detection; errors can occur due to clock slippage.	Often includes advanced error detection methods, like CRC (Cyclic Redundancy Check).
Complexity	Simpler hardware and setup; no shared clock needed.	More complex hardware and setup due to the need for clock synchronization.
Use Cases	Low-speed communication (e.g., RS-232, dumb terminals).	High-speed communication (e.g., Ethernet, SPI, I2C).

Data Transmission through Hardware

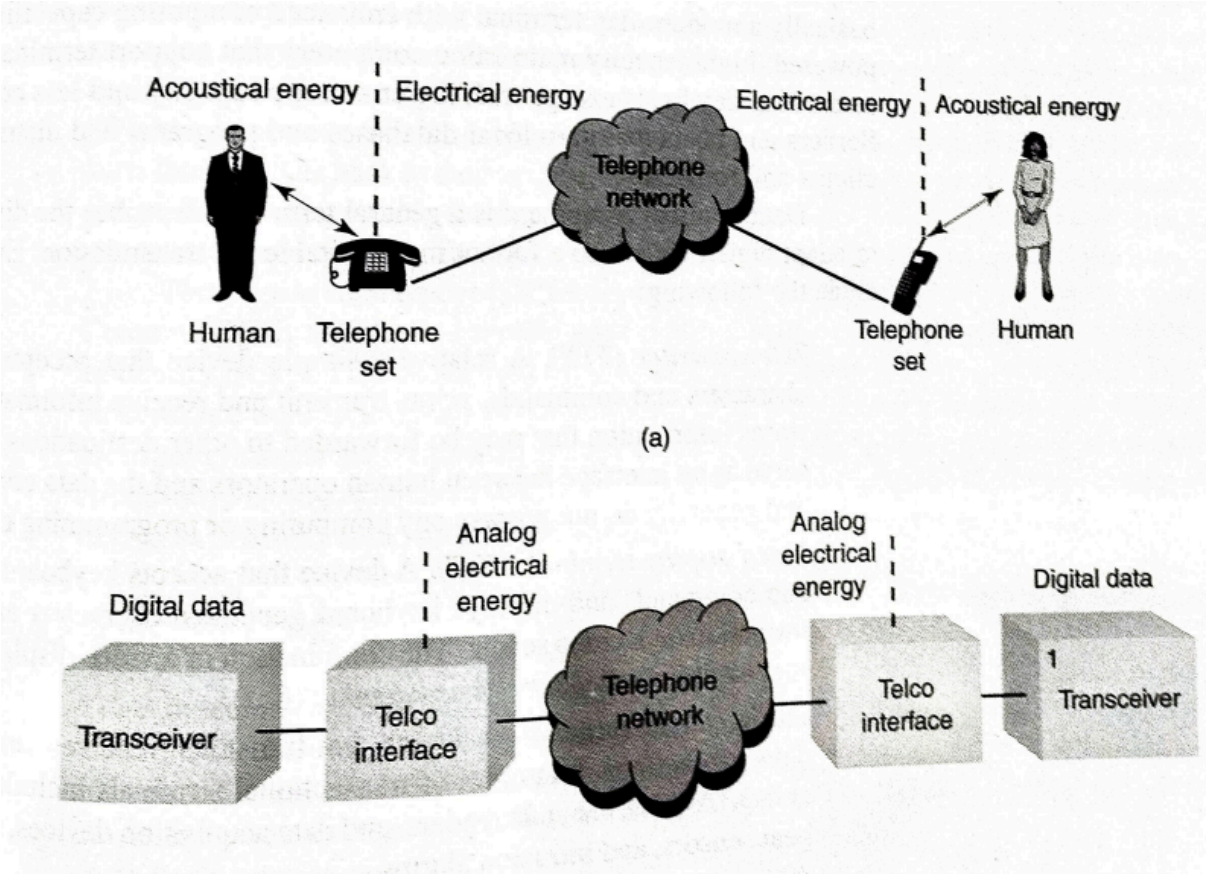
Human Speech Communication (Analog):

1. **Speaking:** When you talk on an old-fashioned phone, your voice (sound waves) is converted into electrical signals by the phone.
2. **Transmission:** These electrical signals travel through the telephone network.
3. **Listening:** At the other end, the signals are converted back into sound waves by the recipient's phone.

Computer Data Communication (Digital):

1. **Sending Data:** Your computer converts digital data into electrical signals.
2. **Transmission:** These signals travel through the telephone network.
3. **Receiving Data:** At the other end, the electrical signals are converted back into digital data by the recipient's device.

In short, both processes use the telephone network, but human speech is converted into sound waves, while computer data is converted into digital signals. This ensures effective communication through the same network.



Data Communication circuit

A **data communication circuit** refers to the path or medium used for transmitting data between devices or systems. It connects the sender (source) and the receiver, allowing them to exchange information. These circuits can vary in complexity, depending on the requirements of the data transmission. Here's a brief overview:

Types of Data Communication Circuits:

1. **Short-range Circuits**:
    - These are typically used for communication over short distances (e.g., a few feet to a few miles).
    - Common examples include **wired connections** such as **Ethernet cables**, **USB cables**, or **Wi-Fi** networks.

    - They are used in local area networks (LANs) or between devices within the same building or close proximity.
2. **Long-range Circuits**:
    - These circuits cover larger distances and are used for communication over cities, countries, or even continents.
    - Examples include **satellite communication** systems, **microwave transmission**, and **fiber optic cables**.
    - These circuits often involve more complex infrastructure, such as communication towers, satellites, or fiber optic networks.
3. **Wireless Circuits**:
    - Wireless communication circuits use radio frequencies to transmit data without physical connections (e.g., **Wi-Fi**, **Bluetooth**, **cellular networks**).
    - These circuits are essential for mobile and remote communication, allowing devices to connect to each other and the internet.
4. **Communication Medium**:
    - The circuit is established over a **transmission medium**, which can be **physical** (e.g., copper wire, fiber optic cables) or **wireless** (e.g., air, satellite link).
    - The quality and capacity of the medium influence the speed and reliability of data transmission.

Data Communication Elements:

1. **Sender**: The device that originates the message (e.g., a computer, sensor).
2. **Receiver**: The device that receives the message (e.g., another computer, server).
3. **Transmission Medium**: The physical path or channel through which the data is transmitted (e.g., cables, wireless signals).
4. **Protocol**: A set of rules that govern data exchange, ensuring the sender and receiver can understand and process the data correctly (e.g., TCP/IP).

Key Considerations:

- **Bandwidth**: The capacity of the communication circuit to carry data, usually measured in bits per second (bps).
- **Latency**: The delay in data transmission, which can affect real-time communication.
- **Error Control**: Mechanisms to detect and correct errors during data transmission, ensuring reliable communication.

In summary, data communication circuits play a critical role in modern communication by providing the infrastructure necessary to transmit data across various distances and between multiple devices.