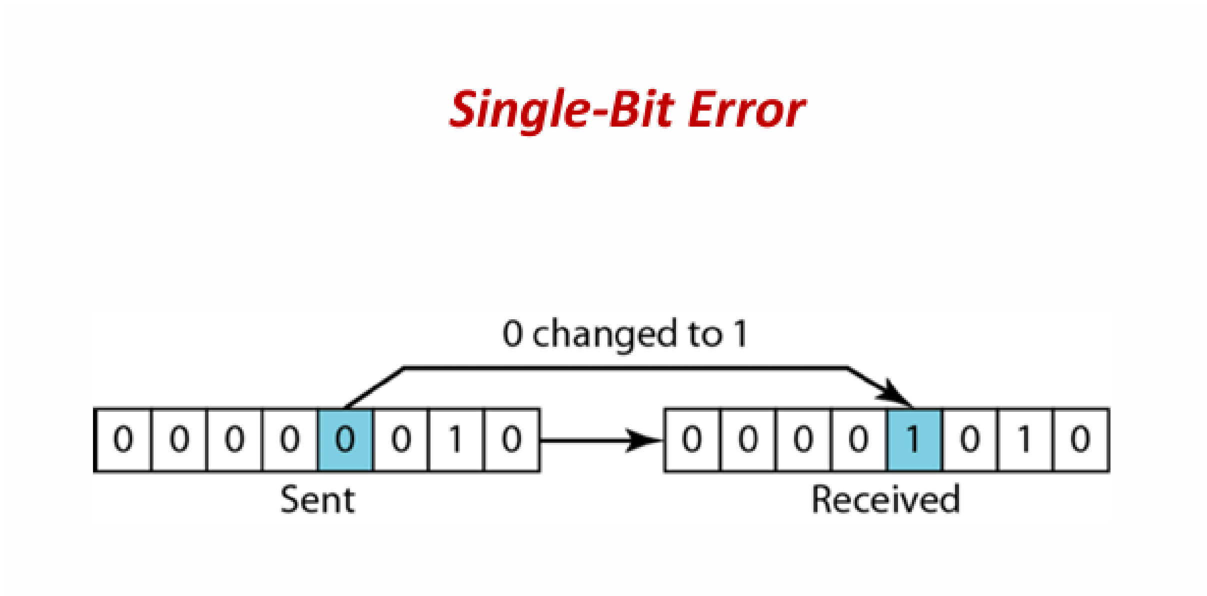
CN Module 2

Data Link and MAC Layer

Types of Errors in Data Transmission

1. **Interference**
   - **Definition**: Interference refers to any unwanted signals or noise that can distort the original signal being transmitted. This can happen due to various factors, such as electromagnetic interference from other devices.
   - **Impact**: It can change the shape of the signal, potentially leading to misinterpretation of the data.
2. **Single-Bit Error**
   - **Definition**: A single-bit error occurs when only one bit in a data unit (like a byte, character, or packet) is flipped. For example, a bit that was originally 0 changes to 1, or a 1 changes to 0.
   - **Example**: If you have the binary data 10110010 and a single-bit error occurs, it might change to 10110000, where the second-to-last bit has flipped.

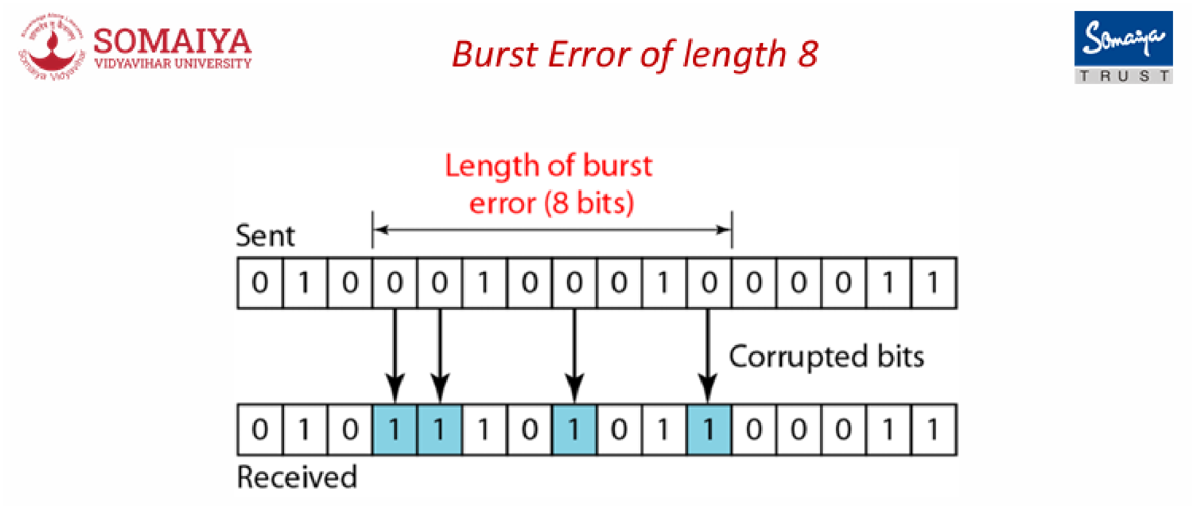


3. **Burst Error**
   - **Definition**: A burst error occurs when two or more bits in a data unit are changed. This means multiple bits are affected, but they do not necessarily have to be consecutive.

- **Example**: For the data 10110010, a burst error might change it to 10111100, where bits in positions 5 and 6 have changed. The term "burst" refers to the sequence of corrupted bits.
- **Measurement**: The length of a burst error is measured from the first corrupted bit to the last corrupted bit. For example, if bits 3 to 5 are affected, the burst length would be 3 bits.

Summary

- **Single-Bit Error**: Only one bit is affected.
- **Burst Error**: Multiple bits are affected, and the impact can span across the data unit.



Redundancy in Error Detection and Correction

Redundancy in Error Detection and Correction refers to techniques used in data communication to ensure that information is transmitted accurately and reliably, even in the presence of errors.

1. **Central Concept**:
   - Redundancy is fundamental in ensuring the reliability of data transmission. It involves adding extra information to the original data so that errors can be detected or corrected.
2. **Purpose of Redundant Bits**:
   - When data is transmitted over a network, it's susceptible to errors due to interference, noise, or other factors. To address this, redundant bits are sent along with the actual data.

   - These extra bits help the receiver determine whether the data has been transmitted correctly and, if necessary, how to correct any errors.

3. **How It Works**:
   - **Sender's Role**: The sender adds redundant bits to the data before transmission. This could be in the form of checksums, parity bits, or more sophisticated methods like CRC (Cyclic Redundancy Check).
   - **Receiver's Role**: Upon receiving the data, the receiver checks the redundant bits against the data. If discrepancies are found, the receiver can either request a retransmission or use the redundant information to correct the error, depending on the error correction scheme in use.

Examples of Redundancy Techniques

- **Parity Bits**: Adding a single bit to indicate whether the number of 1s in the data is even or odd.
- **Checksums**: Calculating a value based on the data and sending it along; the receiver performs the same calculation to check for errors.
- **CRC**: A more complex method that uses polynomial division to detect changes to raw data.

Detection Versus Correction

Detection versus correction refers to the difference between identifying errors in transmitted data and actively correcting those errors. While error detection simply identifies the presence of an error, error correction requires determining the specific nature and location of the errors to fix them.

1. **Correction is More Difficult**:
   - Correcting errors is inherently more complex than merely detecting them. This complexity arises because correction requires more detailed information about the errors.
2. **Error Detection**:
   - In error detection, the process is straightforward: it determines whether an error has occurred. The result is a binary answer—either "yes" (an error is present) or "no" (the data is correct).
3. **Error Correction**:
   - Error correction goes further by not only identifying that an error exists but also needing to know the specifics: how many bits are corrupted and where those corrupted bits are located within the message. This additional information is essential for accurately fixing the data.

Forward Error Correction Versus Retransmission

Forward Error Correction (FEC) and retransmission are two primary methods for managing errors in data communication. FEC enables the receiver to correct errors using additional data bits without needing to resend the original message, while retransmission requires the sender to resend the data when an error is detected.

1. **Two Main Methods of Error Correction**:
   - The two main methods for correcting errors in transmitted data are:
     - **Forward Error Correction (FEC)**
     - **Retransmission**
2. **Forward Error Correction (FEC)**:
   - In FEC, the receiver attempts to reconstruct the original message using the redundant bits included with the data. This allows the receiver to correct certain errors independently, making the process more efficient, especially in environments where resending data would be costly or time-consuming.
3. **Retransmission**:
   - In this approach, when the receiver detects an error, it sends a request to the sender to resend the entire message. This method ensures that the data is accurate but can introduce delays, particularly if errors are frequent.

Coding

Coding is the process of adding extra bits to data to help detect and correct errors during transmission. Different coding schemes create links between the original data bits and the extra bits.

1. **Redundancy**:
   - Redundancy is added through coding schemes that create a connection between the data and the redundant bits, helping to ensure accuracy.
2. **Receiver's Role**:
   - The receiver checks these connections to find and fix any errors that may have occurred during transmission.
3. **Types of Coding**:
   - **Block Coding**: This divides data into fixed-size blocks and adds redundant bits to each block.
   - **Convolutional Coding**: This processes data continuously, adding redundancy based on previous bits.

Coding

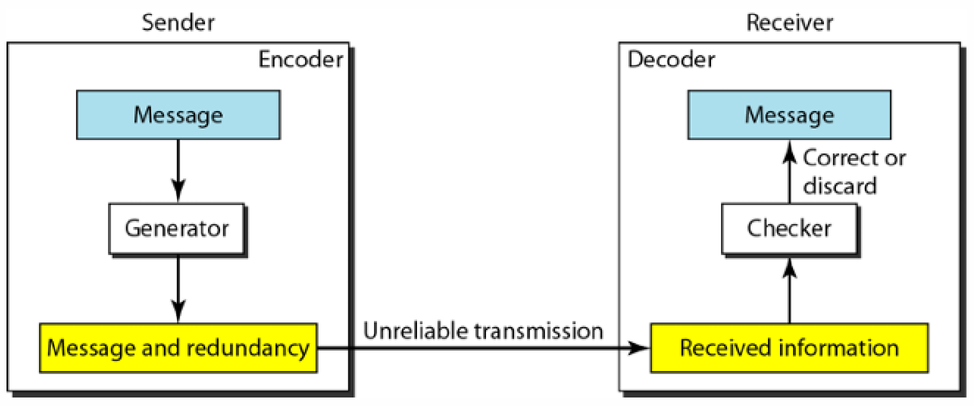


***Fig**: The structure of encoder and decoder*

Block Coding

Block coding is a method of error detection and correction where the message is divided into fixed-size blocks, allowing redundancy to be added systematically.

1. **Division into Blocks**:
   - The original message is split into blocks, each containing k bits. These blocks are referred to as datawords.
2. **Adding Redundant Bits**:
   - For each dataword, r redundant bits are added, making the total length of each block n=k+r
3. **Resulting Codewords**:
   - The new blocks, which include both the original data and the redundant bits, are called codewords. This structure helps the receiver detect and correct errors effectively.

Example

In this example, we assume k=2 (k is the number of data bits) and n=3 (n is the total number of bits, including redundancy). Each dataword is encoded into a codeword by adding 1 redundant bit.

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

Cases of Received Codewords:

- **Case 1**: The receiver receives 011.
    - This is a valid codeword. The receiver successfully extracts the dataword 01.
- **Case 2**: The receiver receives 111.
    - This is not a valid codeword, so it is discarded. The receiver cannot extract any data from it.
- **Case 3**: The receiver receives 000.
    - This is a valid codeword, but it incorrectly maps to the dataword 00. The receiver cannot detect the error because two bits have changed, making the error undetectable.

Why **Case 2** is invalid ?
Let's break it down step-by-step to clarify why 111 is considered invalid.

Key Concepts of Block Coding

1. **Datawords and Codewords**:
    - **Datawords**: The original messages we want to send. For example, 00, 01, 10, and 11.
    - **Codewords**: The messages sent, which include both data and redundant bits. For this example, we can have:
        - Dataword 00 → Codeword 000
        - Dataword 01 → Codeword 011
        - Dataword 10 → Codeword 101
        - Dataword 11 → Codeword 110 (this is an example; we may not have defined this yet).
2. **Redundant Bit**:
    - We add 1 redundant bit to help detect errors. So, if we have 2 bits of data, our codeword is 3 bits long (2 data + 1 redundant).

Received Codeword: 111

1. **What Happens When 111 is Received**:
    - The receiver looks at 111 and checks if it matches any of the valid codewords that were defined based on the datawords.
2. **Checking Validity**:

  - In this example, let's assume the valid codewords based on our previous definitions are:
    - 000 (for 00)
    - 011 (for 01)
    - 101 (for 10)
    - 110 (for 11)
3. **Is 111 a Valid Codeword?**:
  - The received 111 does not match any of the predefined valid codewords.
  - If the coding scheme does not include 111 as a valid mapping for any dataword, then 111 is considered invalid.

Why It's Discarded

- Since 111 does not correspond to a valid codeword based on our definitions, the receiver has no way to know what dataword it represents. Therefore, it discards 111.

So where did
Dataword 00 → Codeword 000
Dataword 01 → Codeword 011
Dataword 10 → Codeword 101
Dataword 11 → Codeword 110 (this is an example; we may not have defined this yet).
This come from ?

Dataword 00 → Codeword 000
Dataword 01 → Codeword 011
Dataword 10 → Codeword 101
Dataword 11 → Codeword 110 (this is an example; we may not have defined this yet).

This is predefined.

Yes, the mapping of datawords to codewords is typically predefined in a coding scheme.

Explanation

1. **Predefined Mapping**:
  - In block coding, the relationship between datawords and codewords is established before transmission. This means that for each possible dataword, there is a specific codeword that corresponds to it.
  - For example:

        - **Dataword 00** corresponds to **Codeword 000**
        - **Dataword 01** corresponds to **Codeword 011**
        - **Dataword 10** corresponds to **Codeword 101**
        - **Dataword 11** corresponds to **Codeword 110**

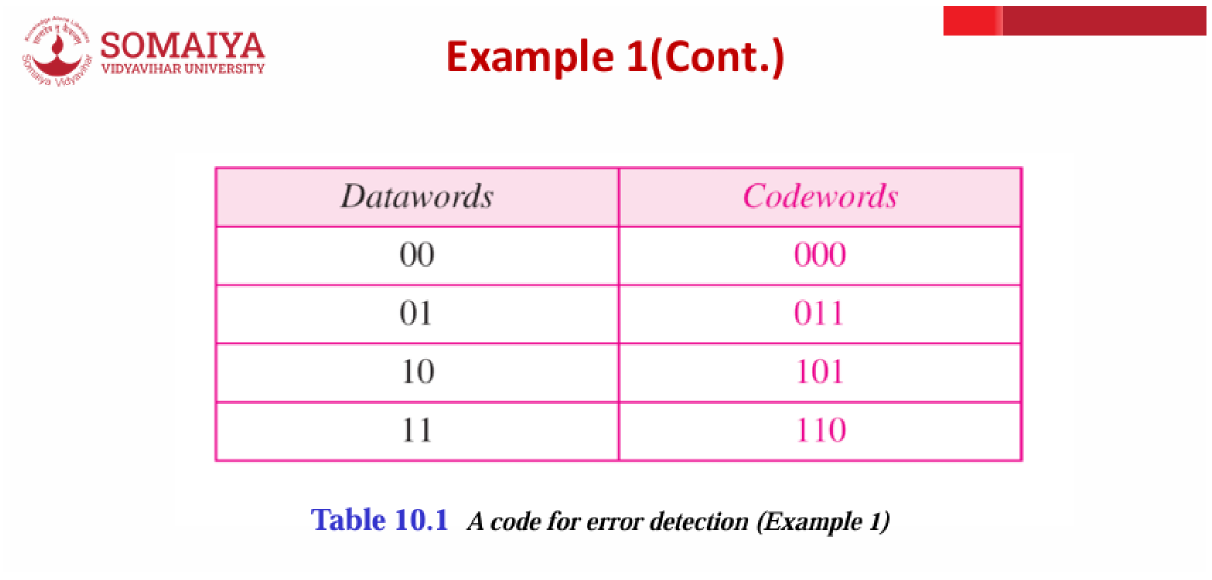
2. **Purpose**:
    - This predefined mapping allows the receiver to know exactly what to expect. When the receiver gets a codeword, it can check against this mapping to determine if the received codeword is valid.
3. **Consistency**:
    - The coding scheme must be consistent. If the mapping is defined in a certain way, both the sender and the receiver must adhere to that mapping for effective communication and error detection.

Q) In the above example we added 1 redundant bit Why we added 1 redundant bit Why not 2 redundant bits? Is their a formuale to add redundant bits?

A) Why Add 1 Redundant Bit?

In block coding, the decision on how many redundant bits to add depends on the required error detection and correction capabilities.

1. **Adding 1 Redundant Bit**:
    - Adding 1 redundant bit can allow for simple error detection. For instance, with a single parity bit, the system can detect if an odd number of bits are in error but cannot correct them.
2. **Why Not 2 Redundant Bits?**:
    - Adding 2 redundant bits could provide additional error detection or correction capabilities, but it would also increase the total number of bits needed for each codeword. This may not be necessary if the system is designed to only detect single-bit errors.
    - The choice of how many bits to add balances the need for error handling with the efficiency of data transmission.


SOMAIYA VIDYAVIHAR UNIVERSITY

Example 1(Cont.)

Datawords	Codewords
00	000
01	011
10	101
11	110

Table 10.1 *A code for error detection (Example 1)*

Error Correction

Error correction involves techniques that not only detect errors in transmitted data but also allow for the actual correction of those errors. This process is more challenging than simply detecting errors.

1. **Complexity**:
   - Error correction is significantly more complex than error detection. While detection merely indicates whether an error has occurred (yes or no), correction requires identifying the exact nature and location of the error.
2. **More Redundant Bits**:
   - To effectively correct errors, we typically need to add more redundant bits compared to what is required for error detection. This is because the additional bits help convey more information about the possible errors.
3. **Checker Functions**:
   - The mechanisms used for error correction involve more sophisticated algorithms and processes than those used for error detection. These algorithms analyze the received data, assess the redundancy, and determine how to correct any errors detected.

Hamming Distance

Hamming Distance is a key concept in error control coding that measures how different two binary words (of the same length) are. It helps determine the error detection and correction capabilities of a coding scheme.

1. **Definition**:

- The Hamming distance between two binary words is the number of positions at which the corresponding bits differ. It is calculated using the XOR operation, which yields a 1 for each position where the bits differ.

2. **Notation**:
   - The Hamming distance between two words x and y is represented as d(x,y)d(x, y)d(x,y).

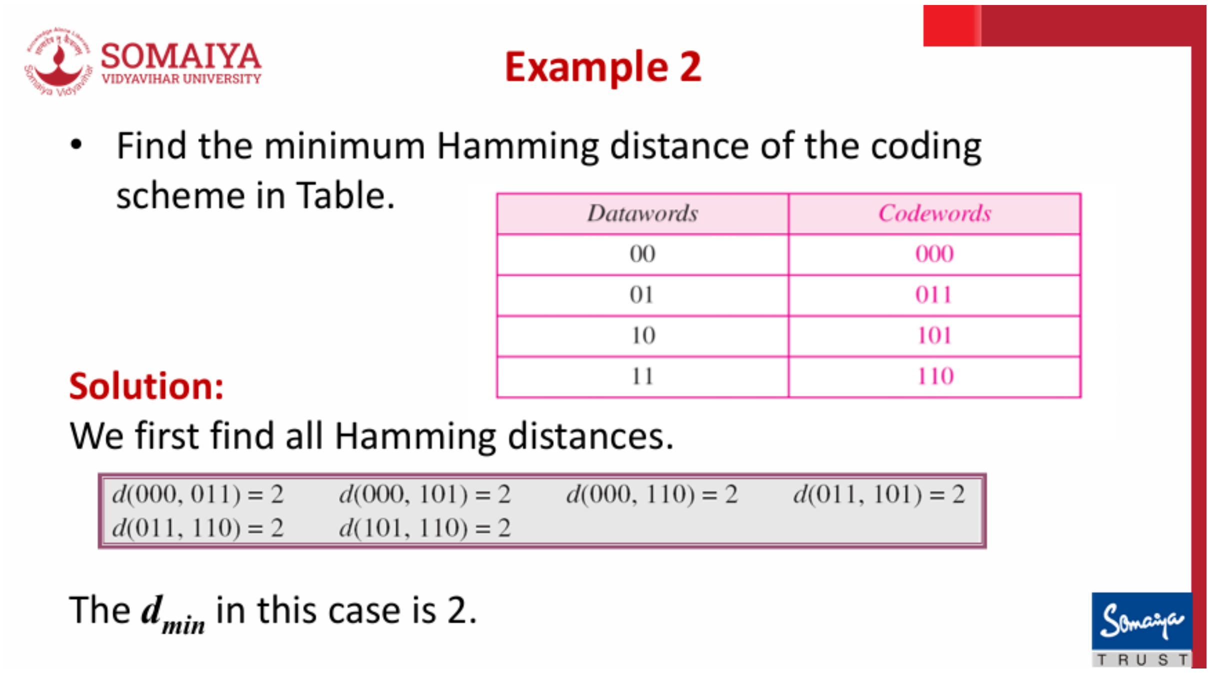
Examples

1. **Example 1**: Calculate d(000,011)
   - Compare the bits:
     - 000
     - 011
   - Differences:
     - First position: 0 vs 0 → no difference
     - Second position: 0 vs 1 → difference (1)
     - Third position: 0 vs 1 → difference (1)
   - Total differences: **2**
   - Thus, d(000,011) =2.
2. **Example 2**: Calculate d(10101,11110)
   - Compare the bits:
     - 10101
     - 11110
   - Differences:
     - First position: 1 vs 1 → no difference
     - Second position: 0 vs 1 → difference (1)
     - Third position: 1 vs 1 → no difference
     - Fourth position: 0 vs 1 → difference (1)
     - Fifth position: 1 vs 0 → difference (1)
   - Total differences: **3**
   - Thus, d(10101,11110)= 3.

Minimum Hamming Distance

The minimum Hamming distance is a crucial metric used in coding theory to evaluate the effectiveness of a code in detecting and correcting errors.

1. **Definition**:
   - The minimum Hamming distance, denoted as d_{min}, is the smallest Hamming distance between any pair of codewords in a given set of codewords.
2. **Importance**:
   - d_{min} , helps determine the error detection and correction capabilities of a code:
     - **Error Detection**: A code can detect up to dmin−1d_{\text{min}} - 1dmin−1 errors. For example, if d_{min}=3 the code can detect up to 2 errors.
     - **Error Correction**: A code can correct up to ⌊(dmin−1)/2⌋ errors. Using the previous example, it could correct 1 error.
3. **Designing Codes**:
   - When designing a coding scheme, ensuring a higher minimum Hamming distance is often a goal, as it enhances the code's ability to detect and correct errors effectively.



SOMAIYA VIDYAVIHAR UNIVERSITY

Example 2

- Find the minimum Hamming distance of the coding scheme in Table.

Datawords	Codewords
00	000
01	011
10	101
11	110

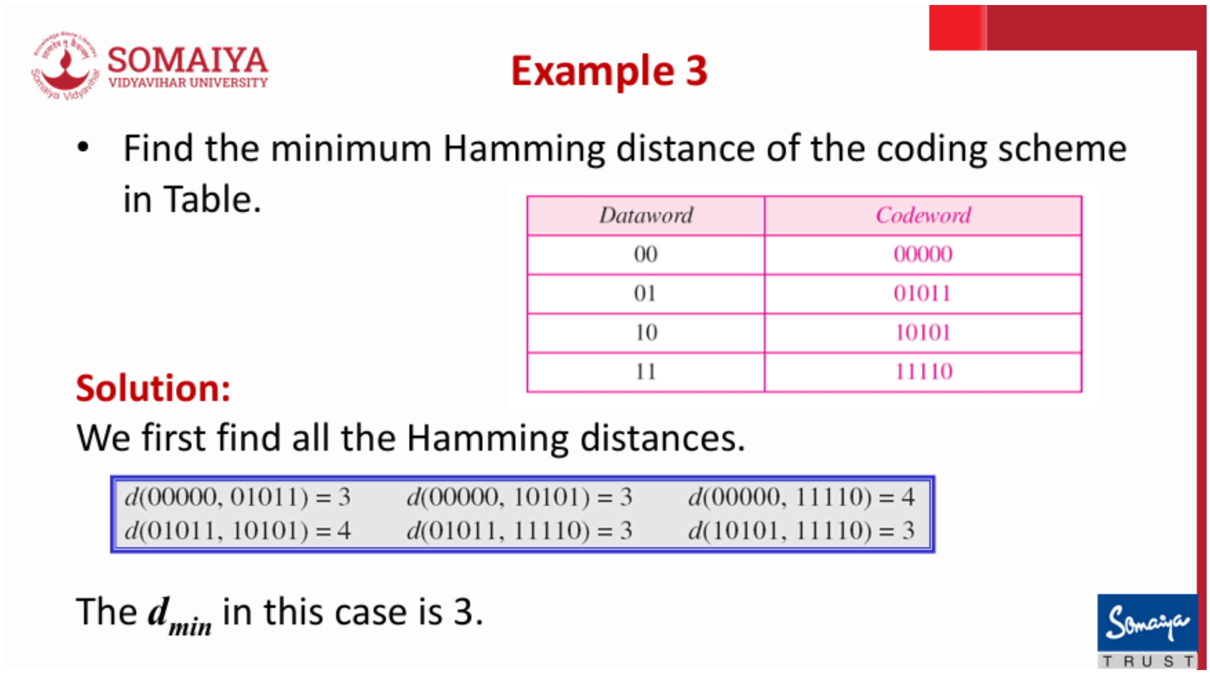
Solution:

We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Somaiya TRUST



Example 3

- Find the minimum Hamming distance of the coding scheme in Table.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Solution:

We first find all the Hamming distances.

$$d(00000, 01011) = 3 \qquad d(00000, 10101) = 3 \qquad d(00000, 11110) = 4$$
$$d(01011, 10101) = 4 \qquad d(01011, 11110) = 3 \qquad d(10101, 11110) = 3$$

The d_{min} in this case is 3.

Minimum Hamming Distance for Error Detection and Correction

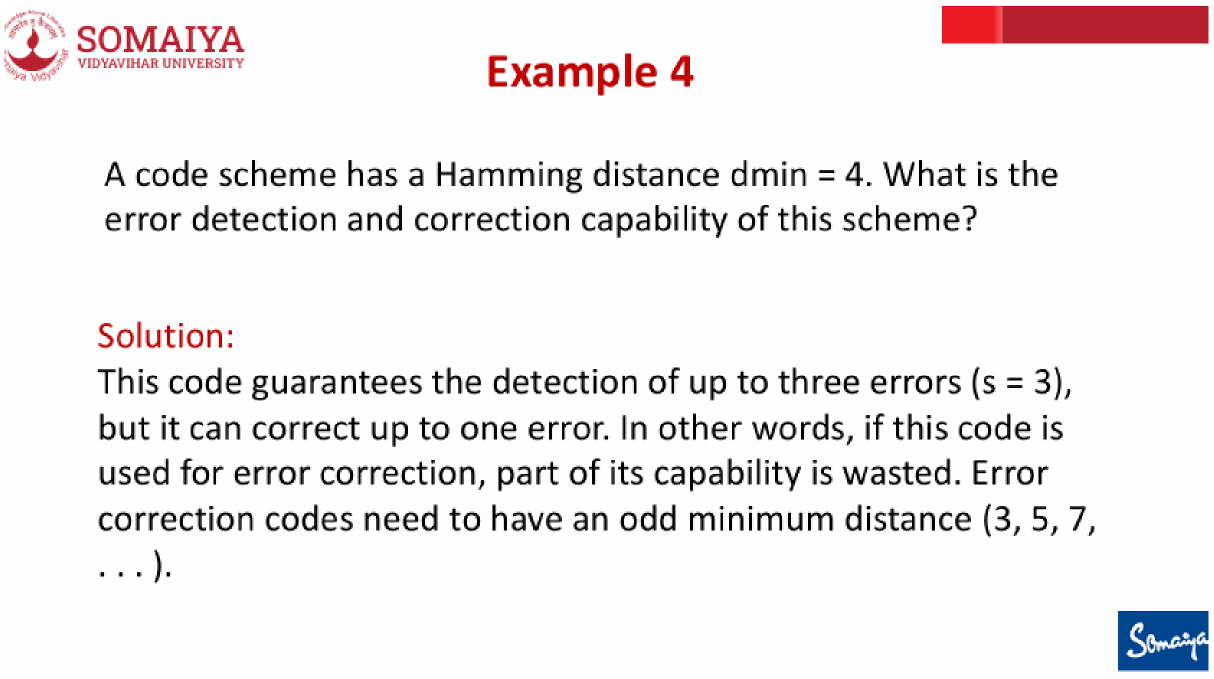
1. **Error Detection**:
   - To guarantee the detection of up to s errors in a code, the minimum Hamming distance (d_{min}) must be

     d_{min} =S+1
   - This means that if the minimum Hamming distance is S+1, the code can detect any combination of s errors. For example, if you want to detect up to 2 errors, the minimum Hamming distance should be at least 3.
2. **Error Correction**:
   - To ensure the correction of up to t errors, the minimum Hamming distance must satisfy:

     d_{min} =2t+1 "t" here is error
   - This formula indicates that for the code to correct t errors, the minimum Hamming distance needs to be at least 2t+1. For instance, to correct 1 error, d_{min} must be at least 3; to correct 2 errors, it must be at least 5.

Summary

- **Detection**: d_{min} =s+1
- **Correction**: d_{min} =2t+1



Example 4

A code scheme has a Hamming distance dmin = 4. What is the error detection and correction capability of this scheme?

Solution:

This code guarantees the detection of up to three errors (s = 3), but it can correct up to one error. In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes need to have an odd minimum distance (3, 5, 7, . . .).

Linear Block Codes

Linear block codes are a specific type of error-correcting code with distinct properties that make them easier to analyze and implement.

1. **Definition**:
   - A linear block code is characterized by the property that the exclusive OR (XOR) of any two valid codewords results in another valid codeword. This property is often referred to as being "closed under addition modulo-2."
2. **Minimum Distance for Linear Block Codes**:
   - The minimum Hamming distance (d_{min}) in linear block codes is defined as the number of 1s in the nonzero valid codeword that has the smallest number of 1s.
   - This means you look for the valid codeword with the fewest 1s (which indicates the smallest Hamming weight), and that number determines the minimum distance.

Key Points

- **Closure Property**: If C1 & C2 are valid codewords, then C1 $\oplus$ C2 (the XOR operation) is also a valid codeword.
- **Minimum Hamming Distance**: d_{min} is crucial for determining the code's error detection and correction capabilities.

There are two types of Linear Block Code

1) Simple Parity-Check Code

2) Hamming Codes

1) **Simple Parity-Check Code**

1. **Structure**:
    - A k-bit dataword is transformed into an n-bit codeword, where n=k+1. This means one additional bit is added to the original data.
2. **Parity Bit**:
    - The extra bit, known as the parity bit, is chosen to ensure that the total number of 1s in the codeword is either even (for even parity) or odd (for odd parity). This helps in detecting errors during transmission.
3. **Minimum Hamming Distance**:
    - The minimum Hamming distance for the simple parity-check code is d_{min} =2. This indicates:
        - **Error Detection**: The code can detect single-bit errors because if one bit is flipped, the parity changes.
        - **Error Correction**: It cannot correct any errors since it doesn't provide enough information to identify which bit was erroneous.

Summary

- **Dataword Length**: k
- **Codeword Length**: n=k+1
- **Parity**: Ensures even/odd total 1s.
- **Error Detection**: Can detect single-bit errors.
- **Error Correction**: Cannot correct any errors

Parity Check Code

Parity checking is a simple error detection technique used in data transmission. The idea is to add an extra bit (the parity bit) to a dataword to ensure that the total number of 1s in the resulting codeword is even (for even parity) or odd (for odd parity).

Example: Dataword 1011

- **Dataword**: 1011

- **Codeword**: 10111 (the extra parity bit ensures that the total number of 1s is even)

Cases Explained

1. **No Error**:
   - **Received Codeword**: 10111
   - **Syndrome**: 0 (indicating no errors detected)
   - **Dataword Created**: 1011 (correctly decoded)
2. **Single-Bit Error (first bit)**:
   - **Received Codeword**: 10011 (a1 changed)
   - **Syndrome**: 1 (indicating an error)
   - **Dataword Created**: None (error detected, cannot decode)
3. **Single-Bit Error (fourth bit)**:
   - **Received Codeword**: 10110 (r0 changed)
   - **Syndrome**: 1 (indicating an error)
   - **Dataword Created**: None (error detected, cannot decode)
4. **Two-Bit Error**:
   - **Received Codeword**: 00110 (r0 and a3 changed)
   - **Syndrome**: 0 (errors cancel each other out, appearing as no error)
   - **Dataword Created**: 0011 (incorrect dataword produced)
   - **Key Point**: The parity-check decoder fails to detect an even number of errors, leading to incorrect decoding.
5. **Three-Bit Error**:
   - **Received Codeword**: 01011 (a3, a2, and a1 changed)
   - **Syndrome**: 1 (indicating an error)
   - **Dataword Created**: None (error detected, cannot decode)

Summary

- **Detection**: Parity check can only detect an **odd** number of errors. If an even number of errors occur, the parity appears correct, leading to undetected errors.
- **Single-bit errors** can easily be identified, while **even numbers of errors** cannot be accurately detected.

Here we have taken taken Parity as 1 but we can take parity as 0 also.

Q) What is Syndrome?

A) **Key Points About Syndrome**

1. **Calculation**: The syndrome is typically calculated using a method that involves comparing the received codeword against the expected codeword or using a specific error-detection algorithm. For parity checks, the syndrome can be determined by examining the parity bits.
2. **Interpretation**:
   - A **syndrome of 0** usually indicates that no errors were detected in the received codeword.
   - A **non-zero syndrome** indicates that an error has occurred. The value of the syndrome can help identify the location of the error, especially in more sophisticated error-correcting codes.
3. **Application in Codes**: In more complex error correction schemes (like Hamming codes), the syndrome is used to identify which bit (or bits) may have been flipped during transmission, allowing for correction.

Example

In the case of the parity check code you mentioned earlier:

- If the received codeword has an even number of 1s, the syndrome might be calculated as 0 (no error).
- If it has an odd number of 1s, the syndrome would be non-zero, indicating that an error has occurred.

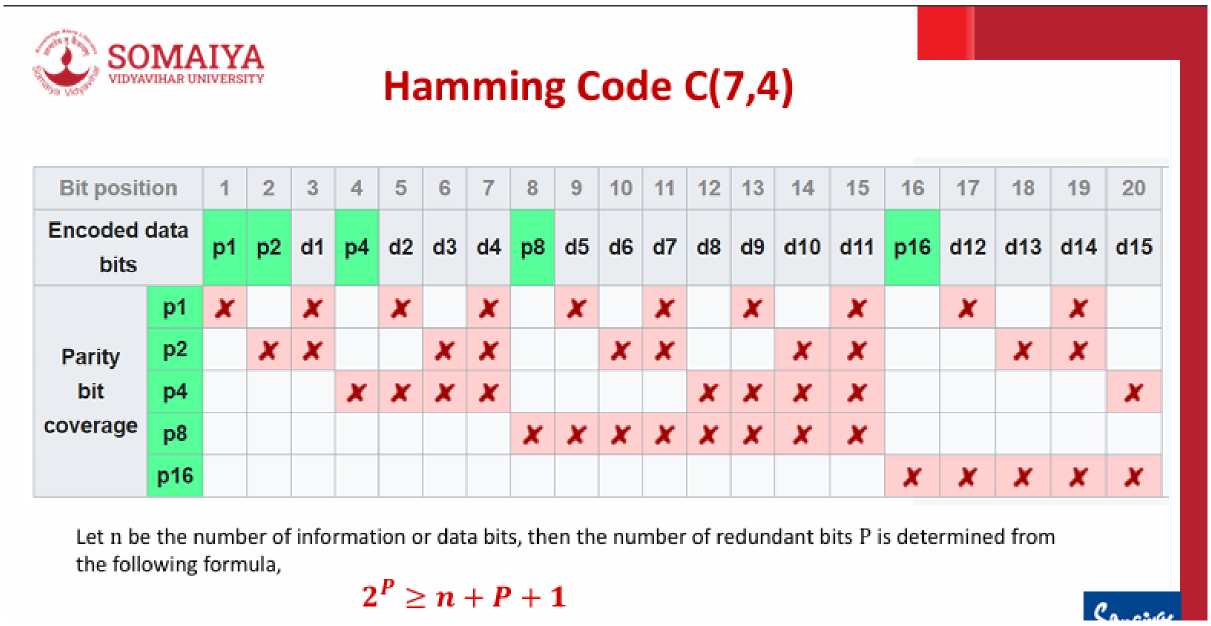
Q) What is a1 and r0 here ?

A) Data Bits (a bits): These are the original bits from the dataword. In your example of the dataword 101110111011:

- a3: 1 (the first bit)
- a2: 0 (the second bit)
- a1: 1 (the third bit)
- a0: 1 (the fourth bit)

Parity Bit (r bits): These are bits added for error detection. The parity bit ensures that the total number of 1s in the codeword meets the parity condition (even or odd).

- r0: The parity bit you add to the dataword to create the codeword. In your example of the codeword 10111, r0 is the fifth bit.



Hamming Code C(7,4)

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗	
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗	
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗
	p8								✗	✗	✗	✗	✗	✗	✗	✗					
	p16																✗	✗	✗	✗	✗

Let n be the number of information or data bits, then the number of redundant bits P is determined from the following formula,

$$2^P \geq n + P + 1$$

The image you've shared explains **Hamming Code C(7,4)** and how it works in terms of bit positions and parity bit coverage. I'll break this down for you.

What is Hamming Code C(7,4)?

Hamming codes are used for **error detection and correction**. In the case of C(7,4), it means that:

- There are **7 bits** in total, which include both data bits and parity (redundant) bits.
- Of these 7 bits, **4 bits** are actual **data bits** (d1, d2, d3, d4), and the other 3 bits are **parity bits** (p1, p2, p4) that help in error detection and correction.

Key Concepts:

1. **d_{min} = 3**:
   - This means the minimum Hamming distance between any two codewords is 3.
   - It allows the detection of up to 2-bit errors and the correction of a 1-bit error.
2. **The Hamming Code Setup:**
   - The image shows how **parity bits** (p1, p2, p4, p8, p16) are calculated and positioned in relation to the **data bits** (d1, d2, d3, etc.).

Bit Positions and Parity Calculation:

Each parity bit (p1, p2, p4, etc.) is responsible for checking specific data bits. This is done through **XOR operations**, where the parity bit is calculated based on the bits it oversees.

- **p1** (parity bit 1) checks bits: 1, 3, 5, 7, 9, 11, 13, 15.

- **p2** (parity bit 2) checks bits: 2, 3, 6, 7, 10, 11, 14, 15.
- **p4** (parity bit 4) checks bits: 4, 5, 6, 7, 12, 13, 14, 15.
- **p8** (parity bit 8) checks bits: 8, 9, 10, 11, 12, 13, 14, 15.
- **p16** (parity bit 16) checks bits beyond what is shown in the diagram.

The Formula:

The formula provided in the image:

$$2^P \geq n+P+1$$

Where:

- **P** is the number of parity bits,
- **n** is the number of data bits.

This formula is used to determine the number of parity bits required for a given number of data bits.

For example, if we have **4 data bits (n = 4)**, we need at least **3 parity bits** to satisfy the equation:

$$2^3 \geq 4+3+1$$

Which gives us a Hamming code with **7 total bits** (4 data bits + 3 parity bits).

Example:

If the data bits are **d1 = 1, d2 = 0, d3 = 1, d4 = 1**, the parity bits are calculated based on the bits they cover using XOR.

- Calculate **p1** based on bits {d1, d2, d4...}.
- Calculate **p2** based on bits {d2, d3...}, and so on.

Cyclic Codes:

Cyclic codes are a specific type of **linear block codes** used for error detection and correction, especially in digital communication and storage systems. Their key characteristic is that **shifting a codeword cyclically (rotating) results in another valid codeword**.

Key Concepts of Cyclic Codes:

1. **Linear Block Codes**:
   - Cyclic codes are a subset of linear block codes. A **linear block code** means that any linear combination of valid codewords also results in a valid codeword.

2. **Cyclic Shifts**:
    - If you take a valid **codeword** and **cyclically shift** (rotate) its bits to the left or right, the result is still a valid codeword. For example, if 1101 is a codeword, shifting it could give 1011, which is also a valid codeword.

Cyclic Redundancy Check (CRC):

CRC is an important **subset of cyclic codes** widely used for **error detection** in networks, particularly in:

- **LANs (Local Area Networks)**
- **WANs (Wide Area Networks)**

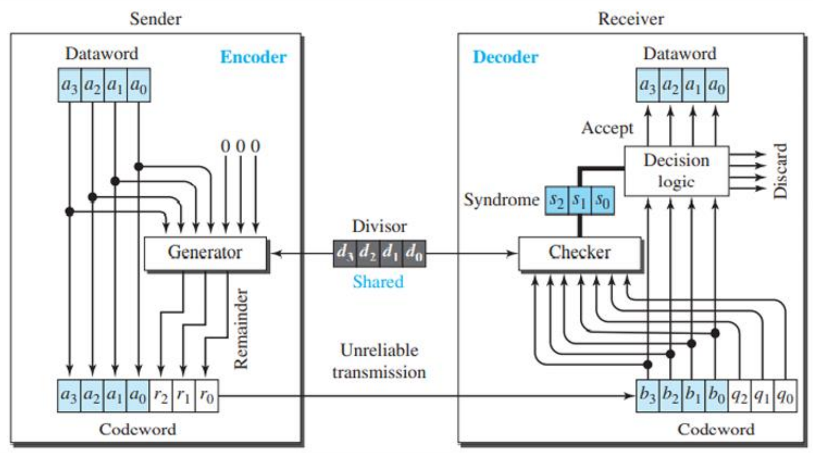
CRC is used to detect accidental changes or corruption in transmitted data. Here's how it works:

1. **Generator Polynomial**:
    - A specific **polynomial** is agreed upon by the sender and receiver. This polynomial defines how the data should be processed.
2. **CRC Algorithm**:
    - The sender computes the **CRC checksum** by dividing the data bits by the generator polynomial, appending the remainder (CRC) to the data before sending it.
    - The receiver performs the same division. If the remainder is zero, the data is considered correct. If not, an error has occurred in the transmission.

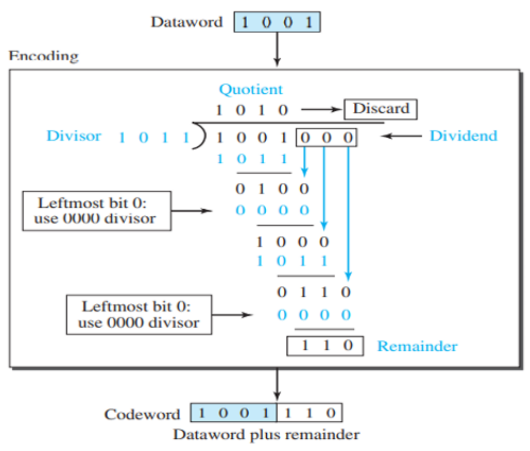
Why Use CRC?

- **Efficiency**: CRC is computationally efficient, making it a popular choice for real-time systems like networking.
- **Accuracy**: CRC can detect common types of errors, including burst errors, where multiple bits are corrupted in a sequence.

CRC Code encoder and decoder

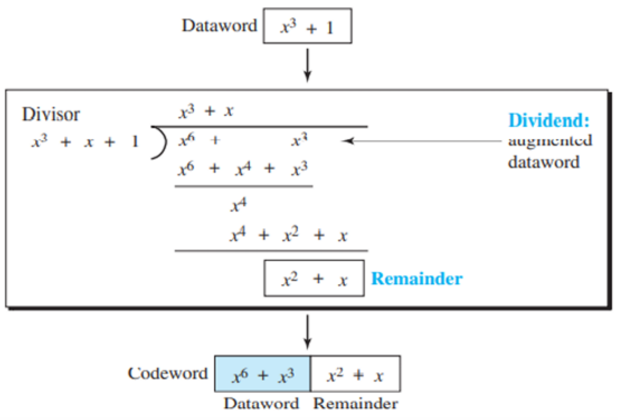


Division in CRC encoder

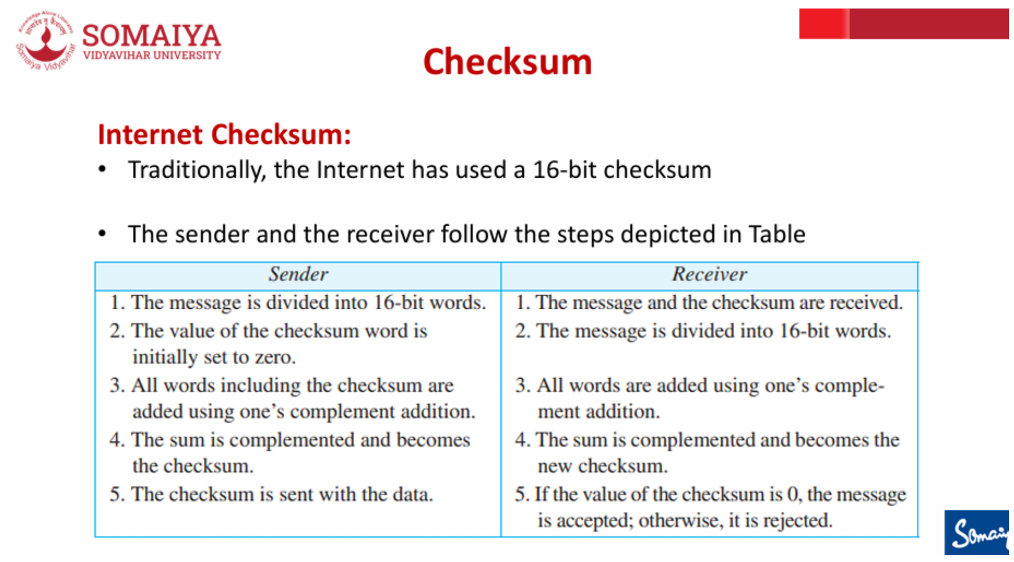


Note:
Multiply: AND
Subtract: XOR

CRC Division using polynomial



The divisor in a cyclic code is normally called the generator polynomial or simply the generator



Checksum

Internet Checksum:

- Traditionally, the Internet has used a 16-bit checksum
- The sender and the receiver follow the steps depicted in Table

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Data Link Layer

Data Link Layer is Divided into two sub layers:-

1) Logical Link Control (LLC Sublayer)

2) Media Access Control (MAC Sublayer)

1. Logical Link Control (LLC)

- **Frame Synchronization**: Makes sure that the sender and receiver know where a message starts and ends.
- **Flow Control**: Keeps the sender from sending too much data at once so the receiver can keep up.
- **Error Checking**: Checks if any mistakes happened during the data transfer and can request a resend if needed.

2. Media Access Control (MAC)

- **Moving Data**: Responsible for sending data packets from one device to another over the network.
- **Sharing the Channel**: Manages how multiple devices use the same network space to avoid collisions (when two devices try to send data at the same time).

Summary

- **LLC** makes sure messages are clear and received correctly.
- **MAC** sends the messages and ensures that devices share the network without interference.

Framing

Framing is the process of packaging bits into manageable units called frames. Each frame needs to be identifiable so that the receiving device knows where one frame ends and another begins. Think of it like sending letters in envelopes—each letter (frame) is distinct and separated by the envelope (delimiter).

Types of Framing

1. **Fixed Size Framing**:
    - Each frame has a set size. The size itself acts as a delimiter.
    - Example: If every frame is always 100 bytes, the receiver knows to read exactly 100 bytes for each frame.
2. **Variable Size Framing**:
    - Frames can be of different sizes, so you need a way to mark where one frame ends and the next begins. There are two main methods:
    - **Character (Byte) Oriented Framing**:
        - Here, data is divided into 8-bit characters (bytes).
        - Each frame has a header (info about the frame) and a trailer (used for error checking), and both are multiples of 8 bits.
        - To mark the beginning and end of a frame, special flags (like a specific byte value) are added.
    - **Bit Oriented Framing**:
        - This method works at the bit level and is less common in simple systems. It involves using bits to define the start and end of frames.

Character (Byte) Oriented Framing

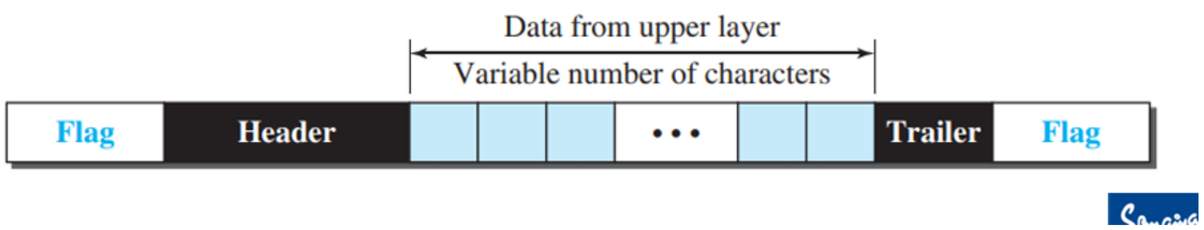
- In this method:
    - Data is structured as 8-bit characters.
    - Frames are identified by placing a flag at the start and end, helping the receiver know when a frame begins and ends.

Key Concepts of Character-Oriented Framing:

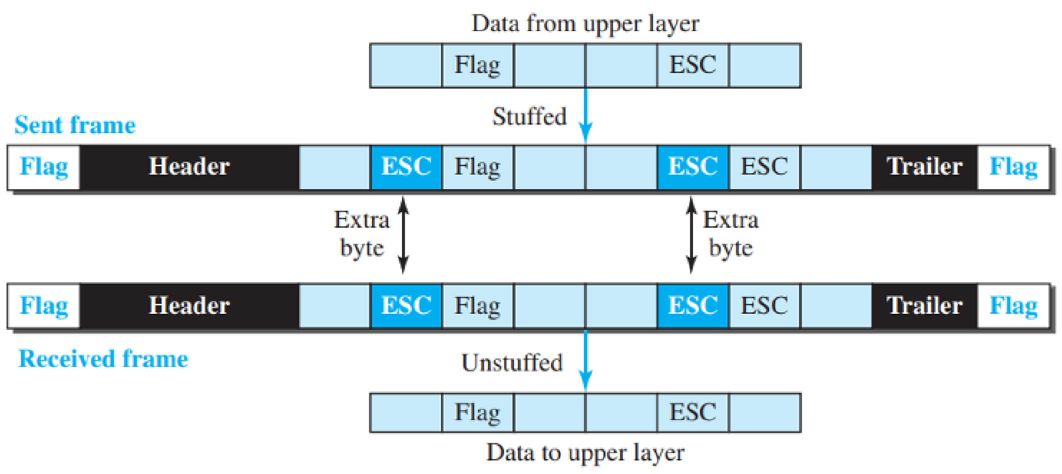
1. **Data in 8-bit Characters**:
    - In character-oriented framing, the data is sent in units of **8-bit characters**. Each character represents a byte of data.
2. **Header and Trailer**:

- **Header** and **Trailer** are also constructed from multiples of 8-bit characters. These contain important control information that helps in managing the data and ensuring its correct delivery.
- The header usually contains control information such as the source and destination addresses, while the trailer might contain error-checking information (like a checksum).

3. **Flags**:
   - To clearly mark the beginning and end of a frame, a special **flag** character is added at both the **start** and the **end** of the frame.
   - This helps the receiver know where a frame starts and where it ends, ensuring proper separation between frames.
   - Flags are used especially in protocols such as **PPP (Point-to-Point Protocol)** and **HDLC (High-Level Data Link Control)**.

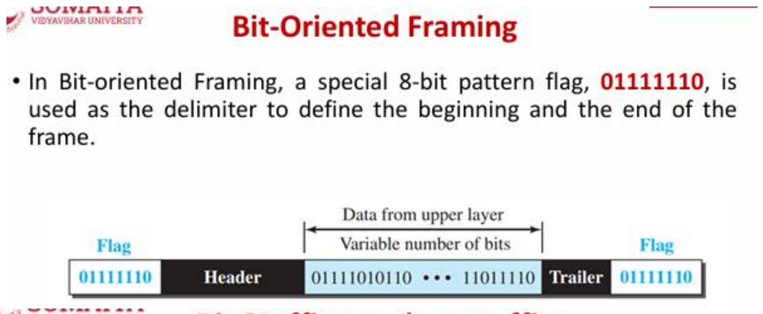


Byte Stuffing and Unstuffing: Byte stuffing is the process of adding one extra byte whenever there is a flag or escape character in the text.



Summary

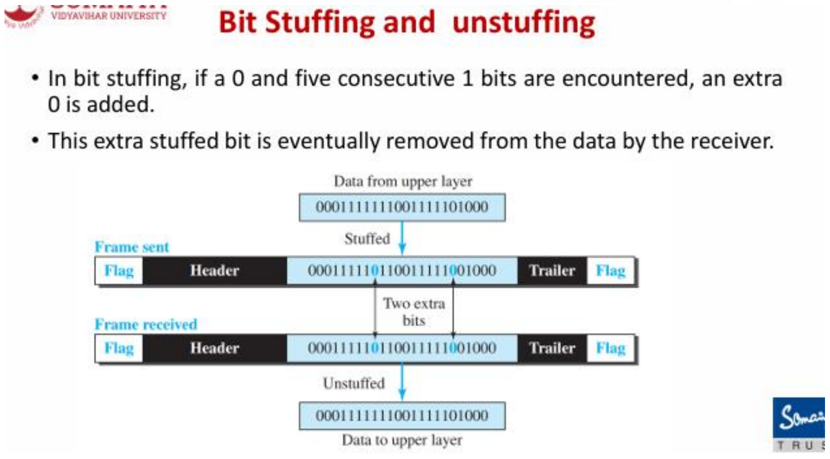
- **Byte Stuffing**: Adds an escape byte before special characters in the data to prevent misinterpretation.
- **Unstuffing**: The receiver removes the escape bytes, restoring the original data.



Bit-Oriented Framing

- In Bit-oriented Framing, a special 8-bit pattern flag, **01111110**, is used as the delimiter to define the beginning and the end of the frame.

Data from upper layer
Variable number of bits
Flag
Flag
01111110
Header
01111010110 ••• 11011110
Trailer
01111110



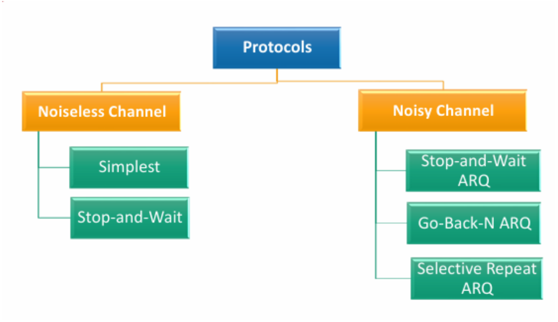
Bit Stuffing and unstuffing

- In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added.
- This extra stuffed bit is eventually removed from the data by the receiver.

Data from upper layer
0001111111001111101000
Stuffed
Frame sent
Flag
Header
000111110110011111001000
Trailer
Flag
Two extra bits
Frame received
Flag
Header
000111110110011111001000
Trailer
Flag
Unstuffed
0001111111001111101000
Data to upper layer

Flow Control and Error Control

- Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
- Flow of data must not allow to overwhelm the receiver.
- Receiving device have limited speed at which it can process incoming data and limited amount of memory to store incoming data.
- Error control is both error detection and correction.
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.



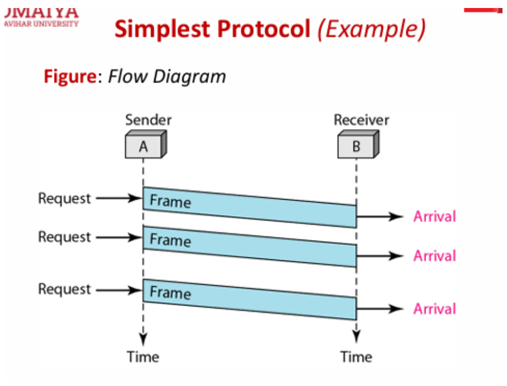
Noiseless Channel

Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.

- Simplest Protocol
- Stop-and-Wait Protocol

1)Simplest Protocol

- Cannot be used in real life
- No flow and error control
- Receiver can handle any frame with negligible processing time.
- Receiver can never be overwhelmed with incoming frames.

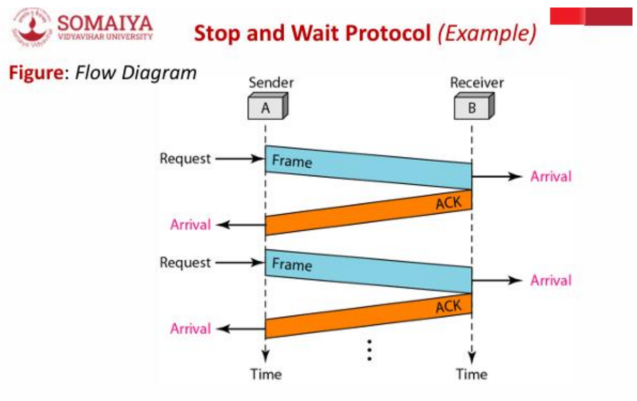


• The sender sends a sequence of frames without even thinking about the receiver.

• To send three frames, three events occur at the sender site and three events at the receiver site.

Stop and Wait Protocol

- If frames arrive faster than they can be processed, frames must be stored until their use.
- To prevent receiver from becoming overwhelmed, receiver needs to tell sender to slow down (ACK).
- Sender sends one frame, stops until it receives confirmation from receiver and then send next frame



• The sender sends one frame and waits for feedback from the receiver.

• When the ACK arrives, the sender sends the next frame.

Noisy Channel

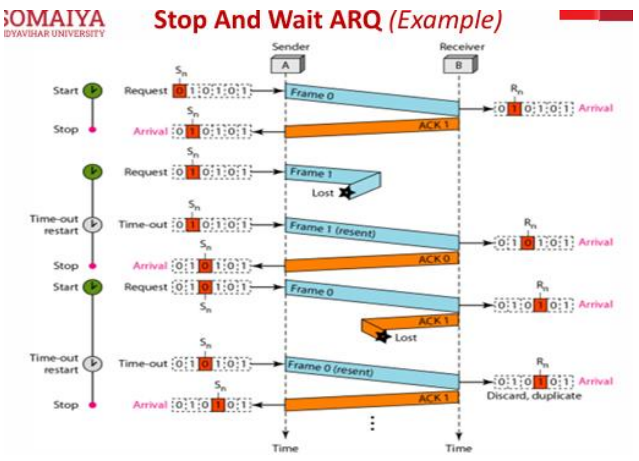
Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We discuss three protocols in this section that use error control.

- Stop-and-Wait Automatic Repeat Request
- Go-Back-N Automatic Repeat Request
- Selective Repeat Automatic Repeat Request

• Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.

• In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.

• In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.



Acknowledgment is always sent of the next frame

We will only use Frame 0 or Frame 1

And

Only use Acknowledgment 0 or Acknowledgment 1

Becaz the sender can only send 1 frame and receiver can receive only 1 frame.

So, total 2 frames

So we use only 0 and 1

If Frame 0 is sent Acknowledgment 1 is given

If Frame 1 is sent Acknowledgment 0 is given

• Very inefficient if channel is thick (large bandwidth)and long (round trip delay is long).

• The product of these two is called the bandwidth-delay product (volume of the pipe in bits).

• The bandwidth-delay product is a measure of the number of bits we can send out of our system while waiting for ACK from the receiver

Go-Back-N ARQ

Here the size of sender if 2^m-1 where m is the size of the sequence number field in bits

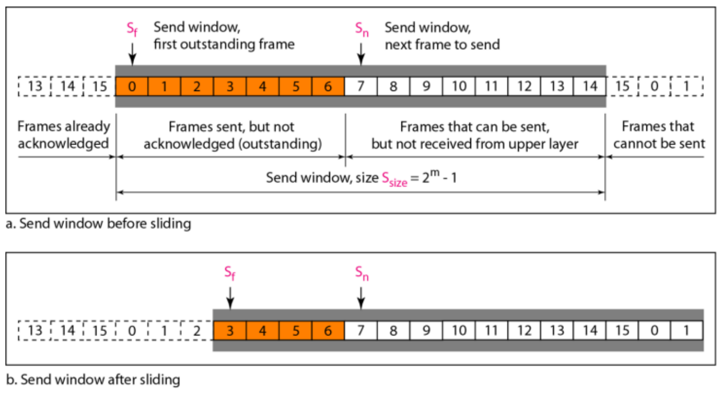
Size of receiver if 1

• Sliding window is an abstract concept that defines the range of sequence numbers.

• The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.

• The window at any time divides the possible sequence numbers into four regions.

Send Window for Go-Back-N ARQ:



S_f Send window, first outstanding frame
S_n Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$
a. Send window before sliding
S_f
S_n
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
b. Send window after sliding

Send Window :

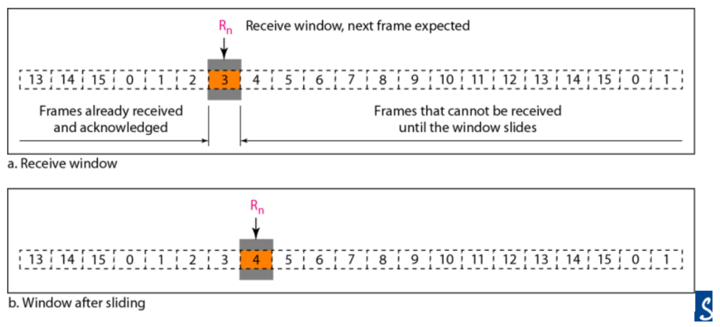
• Sliding window is an abstract concept that defines the range of sequence numbers.

• The send window can slide one or more slots when a valid acknowledgment arrives.

Receive Window :

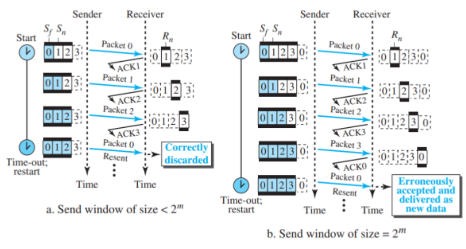
- The receive window is an abstract concept defining an imaginary box of size 1 with one single variable Rn .
- The window slides when a correct frame has arrived; sliding occurs one slot at a time

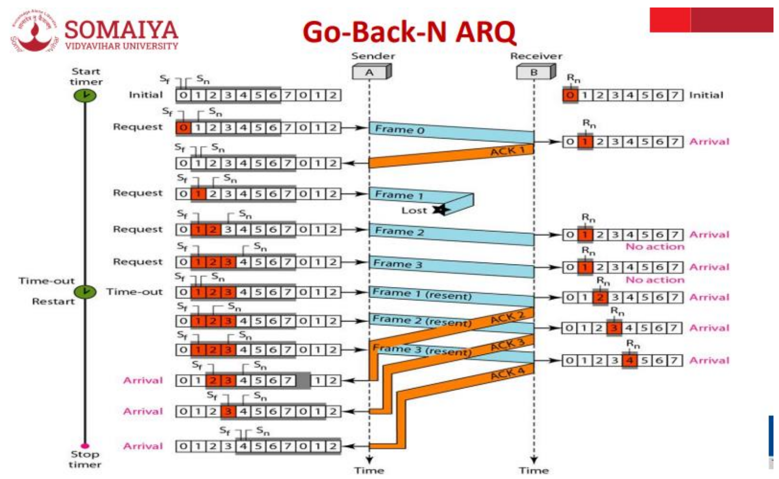
Receive Window for Go-Back-N ARQ:



NOTE:

- *In Go-Back-N ARQ, the size of the send window must be less than 2^m;*
- *the size of the receiver window is always 1.*





I am solving go back n arq I only have confusion in the window size concept I know that in the question in my exam They'll either give sequence number Or bits in sequence number Or directly window size So explain me pls

Go-Back-N ARQ Overview

In Go-Back-N ARQ, the sender can send multiple frames before needing an acknowledgment for the first one, but there’s a limit to how many unacknowledged frames can be in transit. This limit is known as the **window size**.

Understanding Window Size

1. **Window Size**:
   - The window size (often denoted as N) determines how many frames the sender can send without waiting for an acknowledgment.
   - If the window size is N, the sender can send frames numbered 0 to N−1 before needing an acknowledgment for the first frame.
2. **Sequence Numbers**:
   - Sequence numbers help identify each frame sent. For example, if the window size is N, the sequence numbers will typically range from 0 to N−1.
   - After reaching the maximum sequence number, the sender wraps around, meaning it can reuse numbers but must ensure that it does not confuse frames.
3. **Bits in Sequence Number**:
   - If you're given the number of bits for the sequence number, you can calculate the maximum window size.
   - For k bits, the number of unique sequence numbers is 2^k. The maximum window size is generally half of this, i.e., $2^{(k-1)}$, because the sender needs to differentiate between new frames and acknowledgments for old frames.

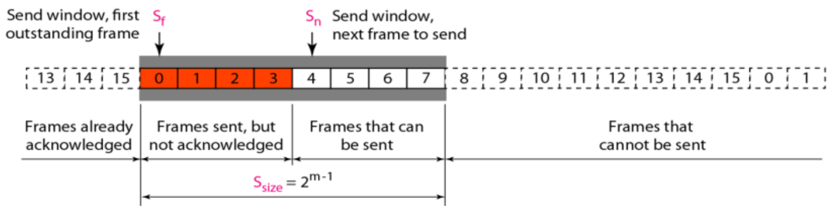
Calculation Examples

1. **Given Sequence Number**:
   - If the sequence numbers are 0,1,2,3,4 (which means you can send frames up to 4), the window size is 5 (0 to 4).
2. **Given Bits in Sequence Number**:
   - If you're given 333 bits for the sequence number:
     - Unique sequence numbers: $2^3=8$
     - Maximum window size: $2^{(3-1)}=4$
   - So, you can send 4 frames before needing an acknowledgment.
3. **Directly Given Window Size**:
   - If the problem states the window size is 4, you can send 4 frames without waiting for acknowledgments.

Selective Repeat ARQ

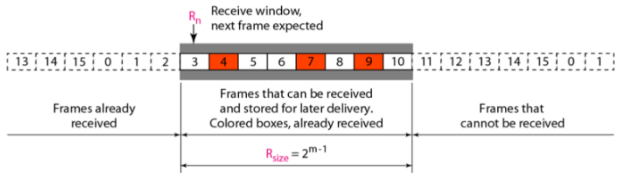
- Mechanism that does not resend N frames when just one frame is damaged; only the damaged frame is resent.
- More efficient for noisy links, but the processing at the receiver is more complex.
- the size of the send window is much smaller; it is 2^{m-1} .
- Sizes of the send window and receive window are the same.

Send window for Selective Repeat ARQ:



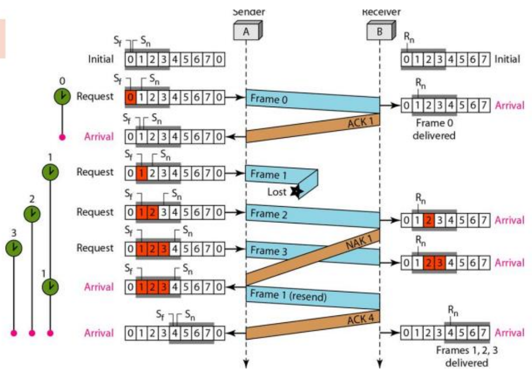
Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$

Receive window for Selective Repeat ARQ:



R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
$R_{size} = 2^{m-1}$

Flow Diagram:



HDLC

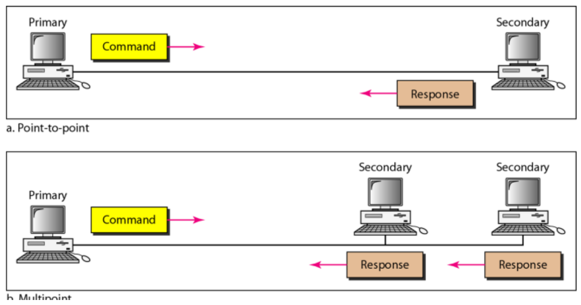
- **HDLC** is a protocol for communication between devices, using bits (1s and 0s).
- It ensures reliable communication with **ARQ** (Automatic Repeat reQuest). If something goes wrong, data is sent again.

It has two common transfer modes

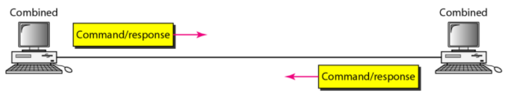
1. **Normal Response Mode (NRM)**
   - One device (the **primary**) controls communication.
   - The other devices (the **secondary**) respond only when asked by the primary.

SOMAIYA VIDYAVIHAR UNIVERSITY

Normal Response Mode

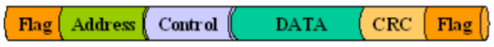


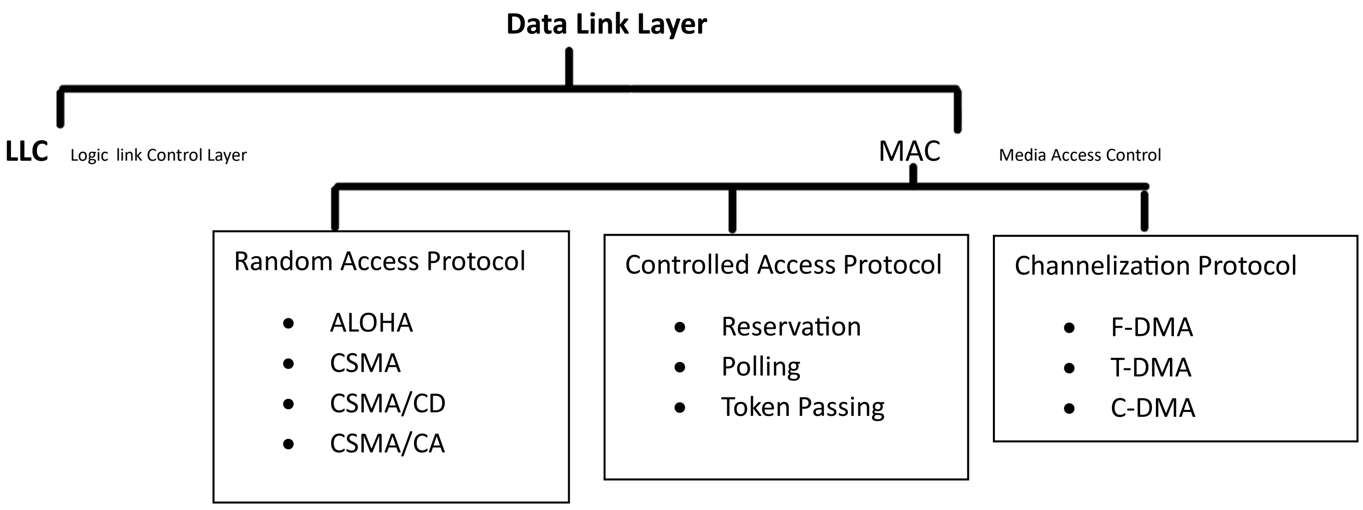
2. **Asynchronous Balanced Mode (ABM)**
   - Both devices are **equal** (no primary or secondary).
   - Both can send and receive data at any time.



SOMAIYA VIDYAVIHAR UNIVERSITY

HDLC frames





Multiple Access

Multiple access refers to the methods used to allow multiple devices to share the same communication channel or medium, like a network.

Types of Multiple Access Mechanisms

1. **Random Access**
   - Devices send data whenever they want, without waiting for permission.
   - There is a risk of **collision** (two devices sending at the same time).
   - Examples: **ALOHA**, **CSMA/CD** (Carrier Sense Multiple Access with Collision Detection).
2. **Controlled Access**
   - Devices must **ask for permission** before sending data.
   - A central device or protocol decides which device can send data.
   - Examples: **Polling**, **Token passing**.
3. **Channelization**
   - The communication channel is **split** into smaller channels.
   - Each device gets its own smaller channel (or time slot).
   - Examples: **TDMA** (Time Division Multiple Access), **FDMA** (Frequency Division Multiple Access), **CDMA** (Code Division Multiple Access).

Random Access

Random Access means that **no station** is assigned to control another station. Each station decides on its own when to send data.

- At any moment, a station that has data to send will use a specific protocol (rules) to decide if it should send its data or not.
- **Transmission is random**: Stations send data without coordination, and there's a possibility that more than one station tries to send at the same time.
- Since multiple stations **compete for access** to the communication medium, this is called **contention**.
- **Collisions** can happen when more than one station tries to send data at the same time. When this occurs, the data (frames) gets destroyed or corrupted, and they may need to be retransmitted

Aloha

ALOHA is a random access protocol used to manage how multiple stations share the same communication channel. When two stations send data at the same time, a **collision** occurs, and both data frames are destroyed or corrupted.

Aloha are of two types:-

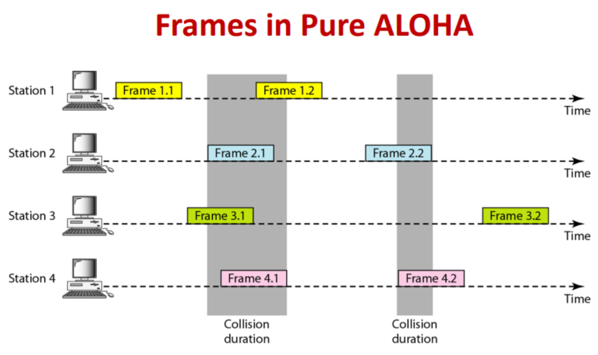
1) Pure ALOHA
2) Slotted ALOHA

Pure ALOHA

- In **Pure ALOHA**, stations send data (called "frames") whenever they have something to send, without any coordination.
- **Collision Risk**: Since there's only one channel, multiple stations can send data at the same time, causing collisions between their frames. Even if **one bit** from each frame overlaps, the whole frame is lost.

How Pure ALOHA Works:

1. **Resending**: If a station's frame is destroyed (due to a collision), it waits for an **acknowledgment** from the receiver. If no acknowledgment is received the station **resends** the frame.
2. **Timeout**: After sending a frame, a station waits for an acknowledgment. If it doesn't receive one before the timeout period, it **retransmits** the frame.
3. **Random Backoff Time**: After a collision and a timeout, each station waits for a **random amount of time** (known as **backoff time** or **TB**) before trying to resend the frame. This randomness helps reduce the chance of further collisions. [If all have a fixed timeout then again we will have collision]
4. **Maximum Retransmissions**: There is a limit on how many times a station can retry sending a frame. After exceeding the maximum number of retries (**Kmax**), the station gives up and tries later.



Vulnerable time = $2*T_{fr}$

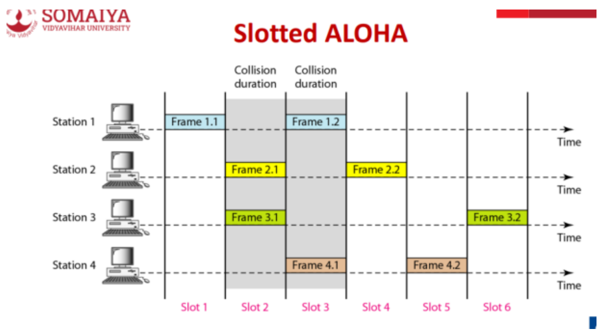
Throughput=Ge^{-2G}

G is the number of stations wish transmit in the same time.

Maximum Throughput will be =0.184 for G=0.5($^1/_2$).

Slotted ALOHA

- Slotted ALOHA was invented to improve the efficiency of pure ALOHA.
- Slotted ALOHA divide time into slots of Tfr and force the station to send only at the beginning of the time slot



Through Put of slotted ALOHA is

$S=Ge^{-G}$ where G is the average number of frames requested per frametime

The maximum Throughput

S_{max} =0.368 when G=1

Q) Calculate possible values of TB when stations on an ALOHA network are a maximum of 600 km apart; signal propagates at 3 x 10^8 m/s. [Pure ALOHA]

Ans) **Problem:**

You are given the following data:

- The **distance between two stations** in the ALOHA network is **600 km**.
- The **speed of the signal propagation** is **3 × 10^8 m/s** (which is the speed of light in a vacuum).

The goal is to **calculate the backoff time (TB)** for an ALOHA network when **stations are 600 km apart**.

Step 1: Calculate the Propagation Time (Tp)

The **propagation time (Tp)** is the time it takes for a signal to travel from one station to another. It is calculated using the formula

$$Tp = \frac{distance}{Speed\ of\ Signal}$$

In your case:

- Distance = **600 km** = **600 × 10^3 meters** (since 1 km = 1000 meters)
- Speed of signal = **3 × 10^8 meters/second**

Now, plug these values into the formula:

$$Tp = \frac{600 \times 10^3 m}{3 \times 10^8 m/s}$$

Tp = 2ms.

After a collision occurs in ALOHA, the station needs to wait for a random amount of time before retransmitting the data. This random wait time is referred to as **backoff time (TB)**. The idea is that this randomness helps avoid further collisions.

- The backoff time is calculated in **multiples of Tp**, because the station needs to wait for a random period before it retransmits.
- When **K = 1** (first retry attempt), the possible backoff times are:
  - **0 ms**: This means no delay, the station could try again immediately (i.e., no backoff).
  - **2 ms**: This corresponds to the **propagation time (Tp)** itself. So, after a collision, a station may wait for a time equivalent to the propagation delay before retrying.

Thus, when **K = 1**, the possible backoff times are:

TB∈{0 ms,2 ms}

When **K = 2** (second retry attempt), the backoff times increase. The possible values for backoff time are multiples of **Tp**:

- **0 ms**
- **2 ms**

   - **4 ms** (2 × Tp)
   - **6 ms** (3 × Tp)

So, when **K = 2**, the possible backoff times are:

TB∈{0 ms,2 ms,4 ms,6 ms}

Numerical on Pure ALOHA: Throughput

Q) A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the throughput if the system (all stations together) produces a. 1000 frames per second? b. 500 frames per second? c. 250 frames per second?

Ans) To calculate throughput for a **Pure ALOHA** system, we need to understand the following:

- **Frame transmission time** is the time it takes to send one frame on the channel.
- **G** is the average number of frames generated by the system per frame time (essentially how busy the channel is).
- **Throughput (S)** is the fraction of frames that are successfully transmitted, given the system's frame generation rate.
- The formula for throughput in **Pure ALOHA** is:

$$S=G\times e^{-2G}$$

Where:

- **G** is the traffic intensity (frames per frame time).
- The expression **e^{-2G}** accounts for the probability that a frame **does not** collide with another frame.

Given Information:

- **Frame size**: 200 bits
- **Channel speed**: 200 kbps (kilobits per second)

Frame transmission time = $\frac{Frame\ Size}{Channel\ Speed} = \frac{200\ bits}{200{,}000\ Bits} = 1ms$

a. 1000 frames per second

1. **Frame generation rate**: The system generates **1000 frames per second**.
   - Since the transmission time of each frame is **1 ms**, the system generates **1 frame per millisecond**.
   - So, **G = 1** (because 1 frame per millisecond = 1 frame per frame time).
2. **Calculate Throughput**:

$S = G \times e^{-2G}$

$= 1 \times e^{-2\times1} = 1 \times e^{-2} \approx 1 \times 0.1353 = 0.1353$

So, the throughput is about **13.5%** of the total frames sent.

3. **Number of successful frames**: The system creates **1000 frames per second**, and the throughput is **0.1353**.

Successful frames=1000×0.1353=135 frames

So, out of 1000 frames, approximately **135 frames** will successfully be transmitted.

b. 500 frames per second

1. **Frame generation rate**: The system generates **500 frames per second**, or **0.5 frames per millisecond**.
   - So, **G = 0.5**.
2. **Calculate Throughput**:

$S = G \times e^{-2G} = 0.5 \times e^{-2\times0.5} = 0.5 \times e^{-1} \approx 0.5 \times 0.3679 = 0.184$

So, the throughput is **18.4%** of the total frames sent.

3. **Number of successful frames**: The system creates **500 frames per second**, and the throughput is **0.184**.

Successful frames=500×0.184=92 frames

So, out of 500 frames, approximately **92 frames** will successfully be transmitted.

Note: This is the **maximum throughput case** for **Pure ALOHA**, as the throughput is highest at **G = 0.5**.

c. 250 frames per second

1. **Frame generation rate**: The system generates **250 frames per second**, or **0.25 frames per millisecond**.
   - So, **G = 0.25**.
2. **Calculate Throughput**:

$S = G \times e^{-2G} = 0.25 \times e^{-2\times0.25} = 0.25 \times e^{-0.5} \approx 0.25 \times 0.6065 = 0.1516$

So, the throughput is about **15.2%** of the total frames sent.

3. **Number of successful frames**: The system creates **250 frames per second**, and the throughput is **0.1516**.

Successful frames=250×0.1516=38 frames

So, out of 250 frames, approximately **38 frames** will successfully be transmitted.

Slotted ALOHA: Throughput Calculation

Q) A slotted ALOHA network transmits 200-bit frames using a shared channel with a 200-kbps bandwidth. Find the throughput if the system (all stations together) produces

a. 1000 frames per second.

b. 500 frames per second.

c. 250 frames per second

Ans) Frame size: 200 bits

- **Channel speed**: 200 kbps (kilobits per second)
- **Frame transmission time**:

Frame transmission time= $\frac{Frame\ Size}{Channel\ Speed}$ = $\frac{200\ bits}{200{,}000\ bps}$ = 1ms

In **Slotted ALOHA**, the throughput is calculated using the formula:

$S=G\times e^{-G}$

Where:

- **G** is the **average number of frames** the system attempts to send per time slot.
- **S** is the **throughput**, or the fraction of frames that are successfully transmitted.

Now, let's calculate the throughput for each case.

a. 1000 Frames per Second

1. **Frame generation rate**:
   The system produces **1000 frames per second.**
   Since each frame takes **1 ms** to transmit, the system generates **1 frame per millisecond**.
   So, **G = 1** (1 frame per millisecond).
2. **Throughput Calculation**:
   Using the formula for **Slotted ALOHA** throughput:

$S=G\times e^{-G} =1\times e^{-1} \approx 1\times 0.3679= 0.368$

So, the throughput is **36.8%** of the total frames sent.

3. **Successful Frames**:
   The system creates **1000 frames per second**, and the throughput is **0.368**.

Successful frames=1000×0.368=368 frames

So, out of **1000 frames**, approximately **368 frames** will be successfully transmitted.

b. 500 Frames per Second

1. **Frame generation rate**:
   The system generates **500 frames per second**.
   Since each frame takes **1 ms** to transmit, the system generates **0.5 frames per millisecond**.
   So, **G = 0.5** (0.5 frames per millisecond).
2. **Throughput Calculation**:
   Using the formula for **Slotted ALOHA** throughput:

$S = G \times e^{-G} = 0.5 \times e^{-0.5} \approx 0.5 \times 0.6065 = 0.303$

So, the throughput is **30.3%** of the total frames sent.

3. **Successful Frames**:
   The system creates **500 frames per second**, and the throughput is **0.303**.

Successful frames=$500 \times 0.303 = 151.5 \approx 151$ frames

So, out of **500 frames**, approximately **151 frames** will be successfully transmitted.

c. 250 Frames per Second

1. **Frame generation rate**:
   The system generates **250 frames per second**.
   Since each frame takes **1 ms** to transmit, the system generates **0.25 frames per millisecond**.
   So, **G = 0.25** (0.25 frames per millisecond).
2. **Throughput Calculation**:
   Using the formula for **Slotted ALOHA** throughput:

$S = G \times e^{-G} = 0.25 \times e^{-0.25} \approx 0.25 \times 0.7788 = 0.1947$

So, the throughput is approximately **19.5%** of the total frames sent.

3. **Successful Frames**:
   The system creates **250 frames per second**, and the throughput is **0.195**.

Successful frames=$250 \times 0.195 = 49$ frames

So, out of **250 frames**, approximately **49 frames** will be successfully transmitted.

CSMA

- Carrier Sense Multiple Access
- Reduce the possibility of collision, but cannot completely eliminate it.
- Carrier sense multiple access (CSMA) requires that each station first check the state of the medium before sending
- The possibility of collision still exists because of propagation delay when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and sense it.

- The possibility of collision still exists because of propagation delay; a station may sense the medium and find it idle, only because the firstbit sent by another station has not yet been received.

TYPES OF CSMA

1. 1-Persistent CSMA

2. P-Persistent CSMA

3. Non-Persistent CSMA

4. O-Persistent CSMA

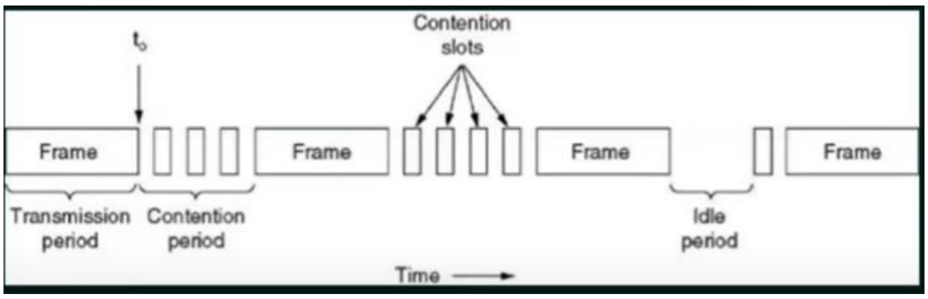
CSMA/CA (CSMA with Collision Avoidance)

CSMA/CD (CSMA with Collision Detection)

They are Solving technique's

1) CSMA/CA (CSMA with Collision Avoidance)

- If two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately. Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop transmitting as soon as the collision is detected.
- Quickly terminating damaged frames saves time and bandwidth.
- This protocol, known as CSMA/CD (CSMA with Collision Detection) is widely used on LANs in the MAC sublayer.
- This is widely used by Ethernet



- At the point marked t_0, a station has finished transmitting its frame. Any other station having a frame to send may now attempt to do so.
- If two or more stations decide to transmit simultaneously, there will be a collision.

- Collisions can be detected by looking at the power or pulse width of the received signal and comparing it to the transmitted signal.
- After a station detects a collision, it aborts its transmission, waits a random period of time, and then tries again, assuming that no other station has started transmitting in the meantime.
- Therefore, model for CSMA/CD will consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet.

2) **CSMA/CD (CSMA with Collision Detection)**

- Carrier-sense multiple access with collision avoidance (CSMA/CA) is a network multiple access method in which carrier sensing is used, but nodes attempt to avoid collisions by beginning transmission only after the channel is sensed to be "idle".
- It is particularly important for wireless networks, where the collision detection of the alternative CSMA/CD is not possible due to wireless transmitters desensing their receivers during packet transmission.
- CSMA/CA is unreliable due to the hidden node problem and exposed
- terminal problem.
- Solution: RTS/CTS exchange.
- CSMA/CA is a protocol that operates in the Data Link Layer (Layer 2) of the OSI model.
- CSMA/CA is good for wireless networks.
- CSMA/CA is unreliable due to the hidden node problem and exposed terminal problem. To solve the above problem we use RTS/CTS exchange.
- IEEE 802.11 Wi-Fi uses CSMA/CA.

Collisions are avoided through the use of CSMA/CA's three strategies:

o Inter-frame space

o Contention Window

o Acknowledgments

Difference Between ALOHA and CSMA/CD:

1. **Persistence (Medium Sensing)**:
   - **ALOHA**: No sensing before sending. Stations just transmit anytime.
   - **CSMA/CD**: Stations sense the channel before sending. They wait if the channel is busy.
2. **Frame Transmission**:
   - **ALOHA**: Transmits the whole frame first, then waits for an acknowledgment.

- **CSMA/CD**: Transmits while continuously checking for collisions. If a collision happens, it stops and retries.

3. **Jamming Signal**:
   - **ALOHA**: No jamming signal.
   - **CSMA/CD**: Sends a jamming signal if a collision is detected to notify other stations.

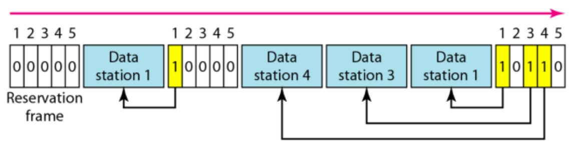
Controlled Access

1. In controlled access, the stations consult one another to find which station has the right to send.
2. A station must be authorized by someone (e.g., other stations) before transmitting
3. Three common methods:
   - Reservation
   - Polling
   - Token passing

Reservation

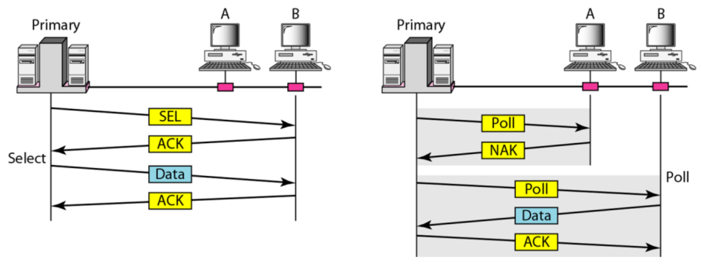
Each station needs to make a reservation before sending data.

If there are N stations in the system, there are exactly N reservation minislots in the reservation frame. [As shown in the fig.]



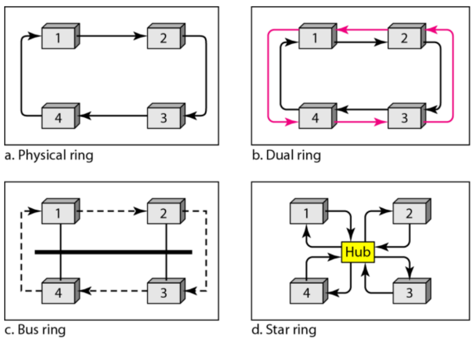
Polling

- Polling works with topologies in which one device is designated as a primary station and the other devices are secondary stations.
- All data exchanges must be made through the primary device.
- It is up to the primary device to determine which device is allowed to use the channel
- Primary device - always the initiator of a session
- The select function is used whenever the primary device has something to send.
- The poll function is used by the primary device to solicit transmissions from the secondary devices



Token Passing

- A station is authorized to send data when it receives a special frame called a token.
- Here there is no master node.
- A small, special-purpose frame known as a token is exchanged among the nodes in some fixed order.
- When a node receives a token, it holds onto the token only if it has
- some frames to transmit; otherwise, it immediately forwards the token to the next node.
- If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node.
- Token passing is decentralized and highly efficient. But it has problems as well.
- For example, the failure of one node can crash the entire channel. Or if a node accidentally neglects to release the token, then some recovery procedure must be invoked to get the token back in circulation.



Channelization

- Similar to multiplexing.
- Multiplexing in computer networking means multiple signals are combined together thus travel simultaneously in a shared medium.
- Three schemes :-

  o Frequency-Division Multiple Access (FDMA)

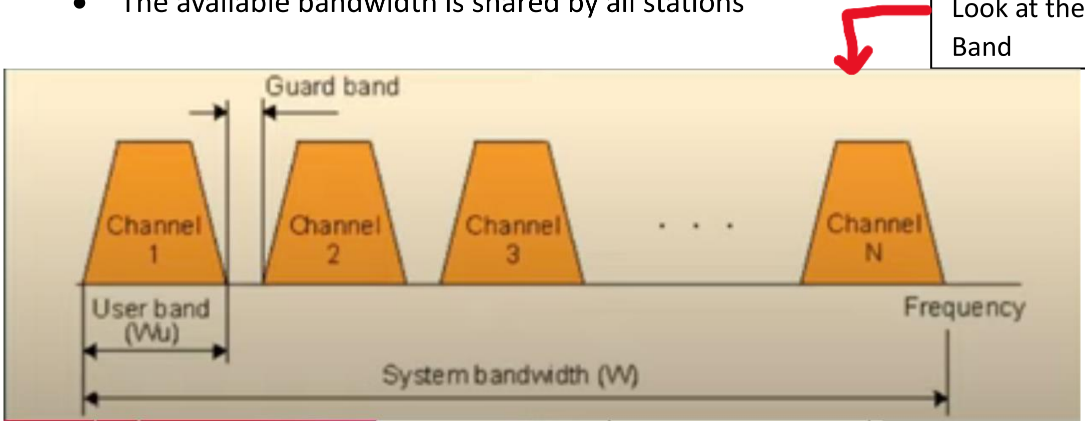
  o Time-Division Multiple Access (TDMA)

  o Code-Division Multiple Access (CDMA)

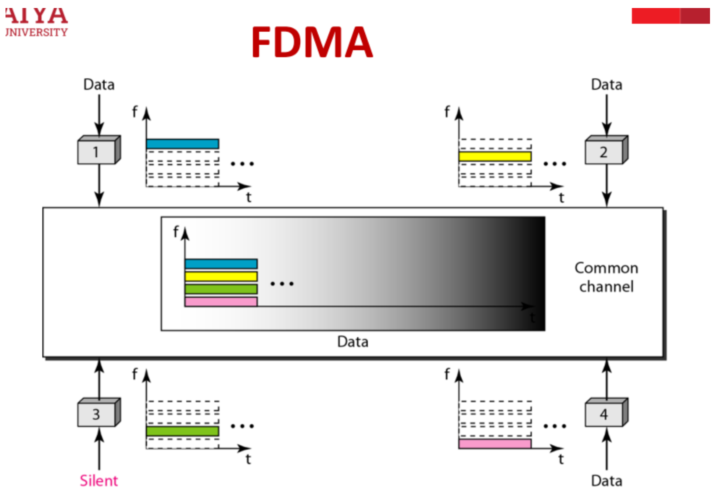
FDMA

- In FDMA, the available bandwidth of the common channel is divided into bands that are separated by guard bands.
- The available bandwidth is shared by all stations

Look at the Guard Band



- FDM is a physical layer multiplexing technique, while FDMA is a data link layer access method.
- Using FDM to allow multiple users to utilize the same bandwidth is called FDMA.
- FDM uses a physical multiplexer, while FDMA does not.

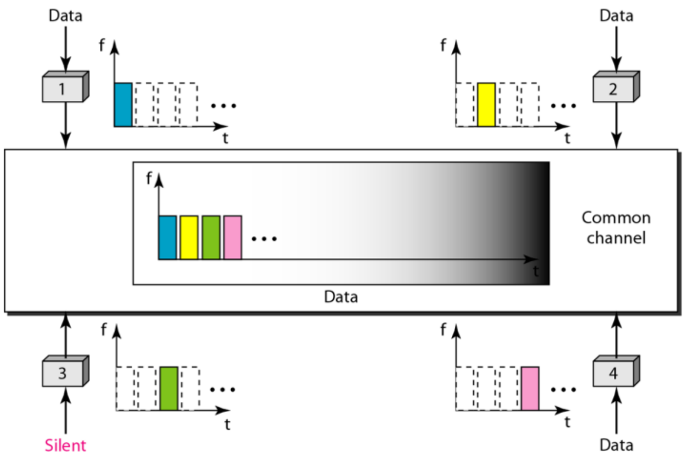


There are 4 stations.
Each sending some data.

There is a guard Band between all of them.

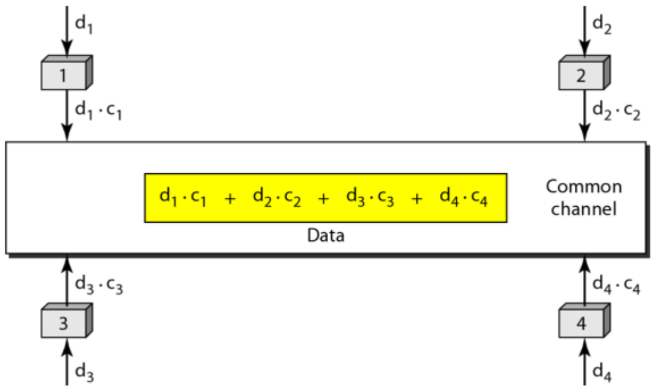
TDMA

- In TDMA, the bandwidth is just one channel that is time shared between different stations.
- The entire bandwidth is just one channel.
- Stations share the capacity of the channel in time.



CDMA

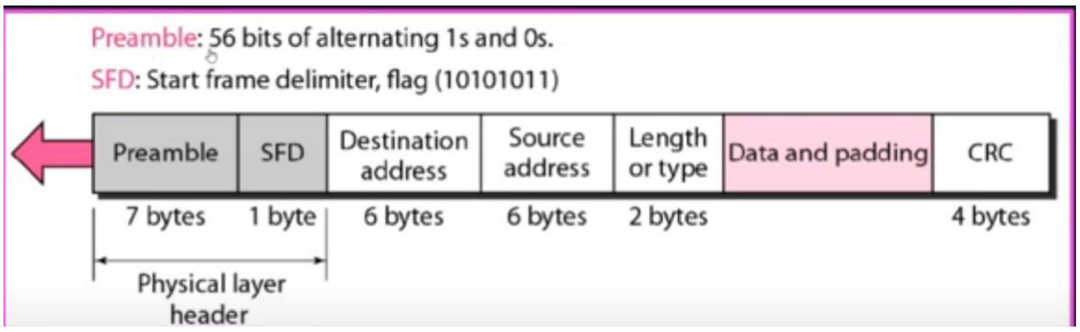
- In CDMA, one channel carries all transmissions simultaneously.
- CDMA differs from FDMA because only one channel occupies the entire bandwidth of the link.
- It differs from TDMA because all stations can send data simultaneously; there is no time sharing.



https://www.youtube.com/watch?v=n9-VQvIFvFc&t=29s

Watch this video.

Ethernet



In Preamble we have 56 bits of alternating 1,s and 0.

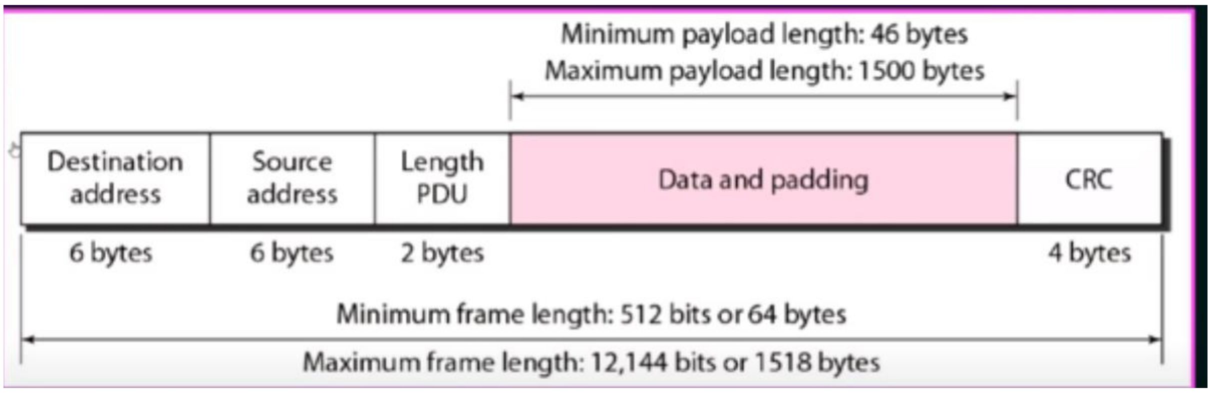
SFD has 10101011

The last 11 in SFD is used to tell that not destination will start.

Why Preamble and SFD are like this ?

Becaz they are used for synchronization

Min & Max of Length of Ethernet Frame


Minimum payload length: 46 bytes
Maximum payload length: 1500 bytes
Destination address
Source address
Length PDU
Data and padding
CRC
6 bytes
6 bytes
2 bytes
4 bytes
Minimum frame length: 512 bits or 64 bytes
Maximum frame length: 12,144 bits or 1518 bytes

As u can see we have ignored Preamble and SFD.

Minimum Frame Length = Minimum Payload Length + Destination address Size + Source address Size + Length PDU Size + CRC

= 46 + 6+6+2+4

= 64 Bytes

Maximum Frame Length = MaximumPayload Length + Destination address Size + Source address Size + Length PDU Size + CRC

= 1500+6+6+2+4

= 1218 Bytes