Computer Networks (CSC305)

Course Outline:

√Overview of Data Communication and Networking

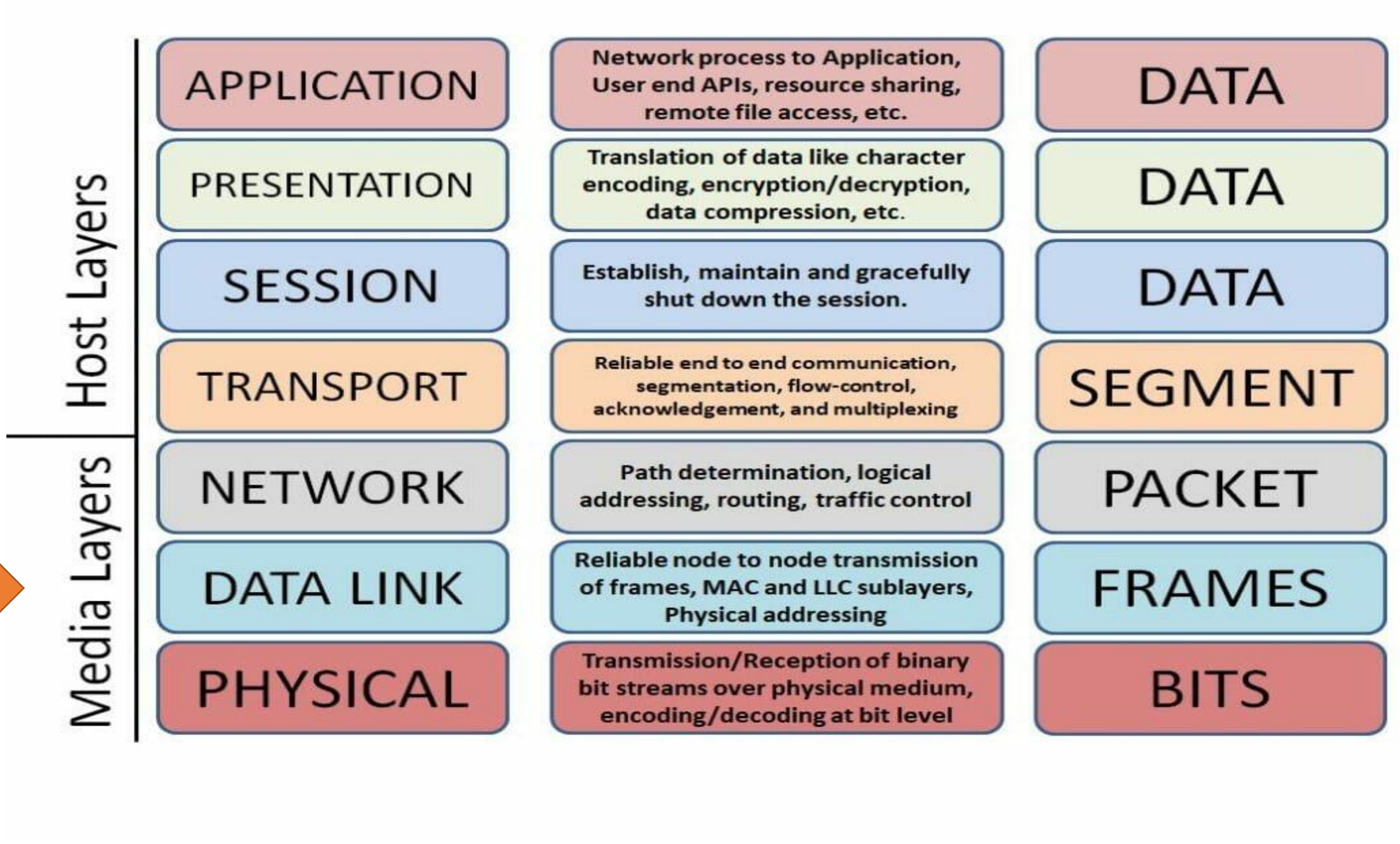
√Physical Layer

√Data Link Layer

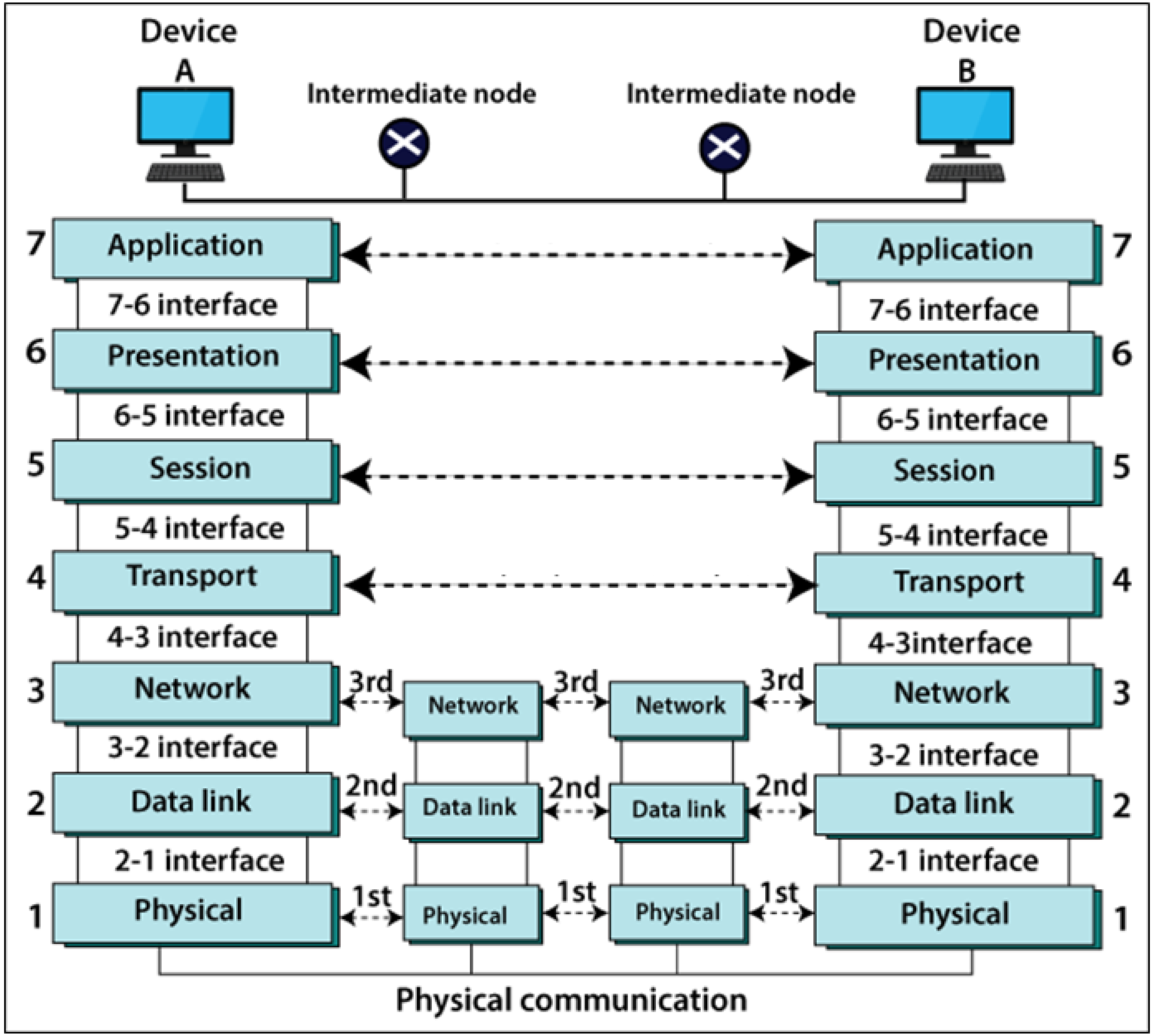
- Logical Link Control (LLC)
- Medium Access Control (MAC)

- Network Layer
- Transport Layer
- Application Layer

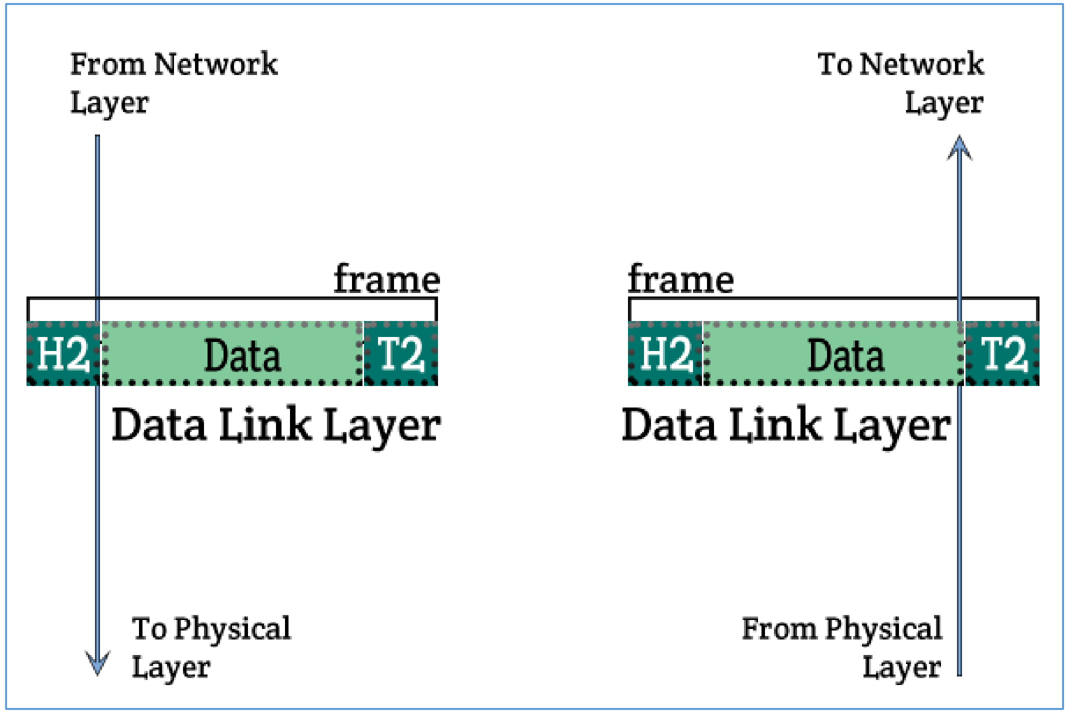
OSI Reference Model

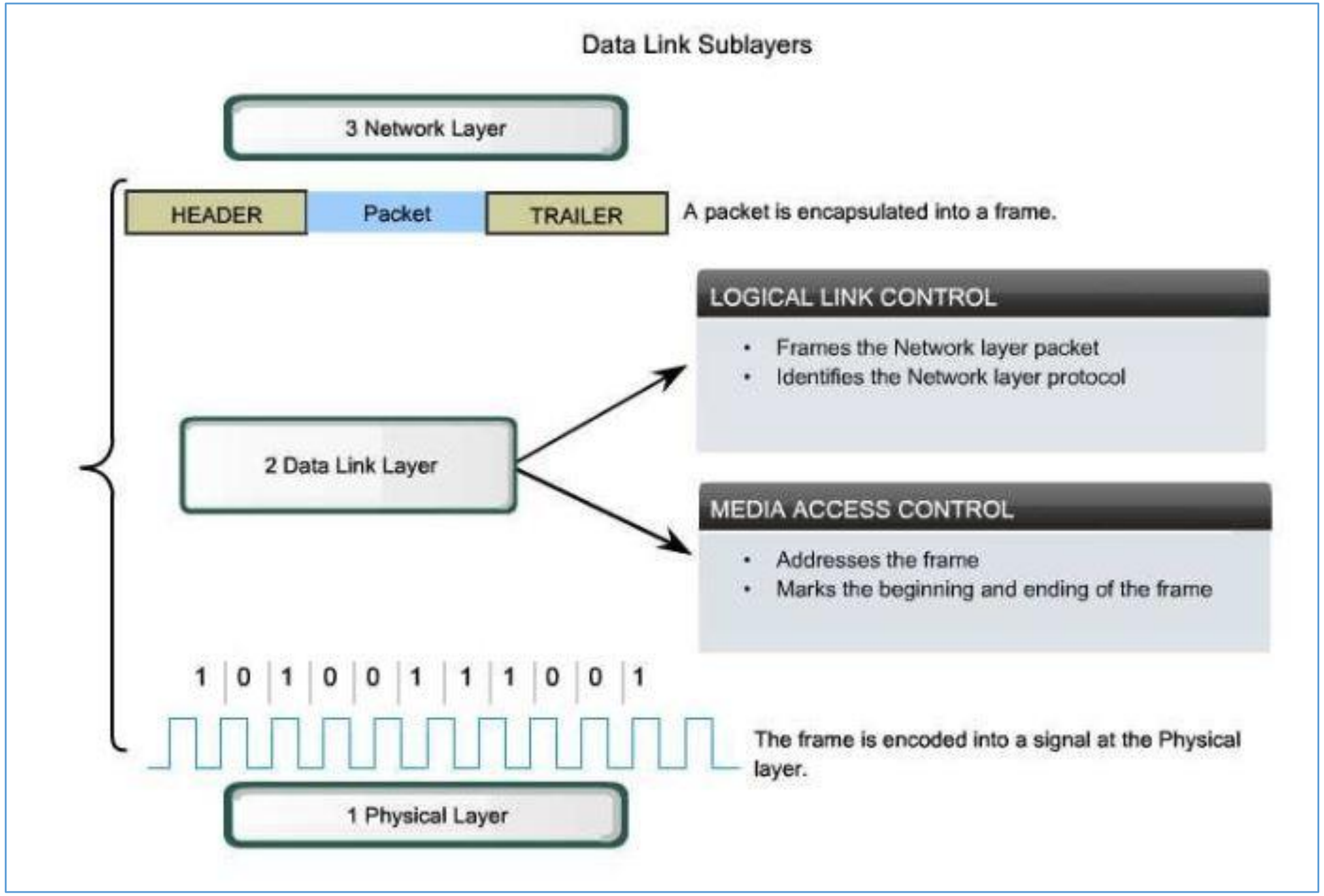


OSI Reference Model



Data Link Layer



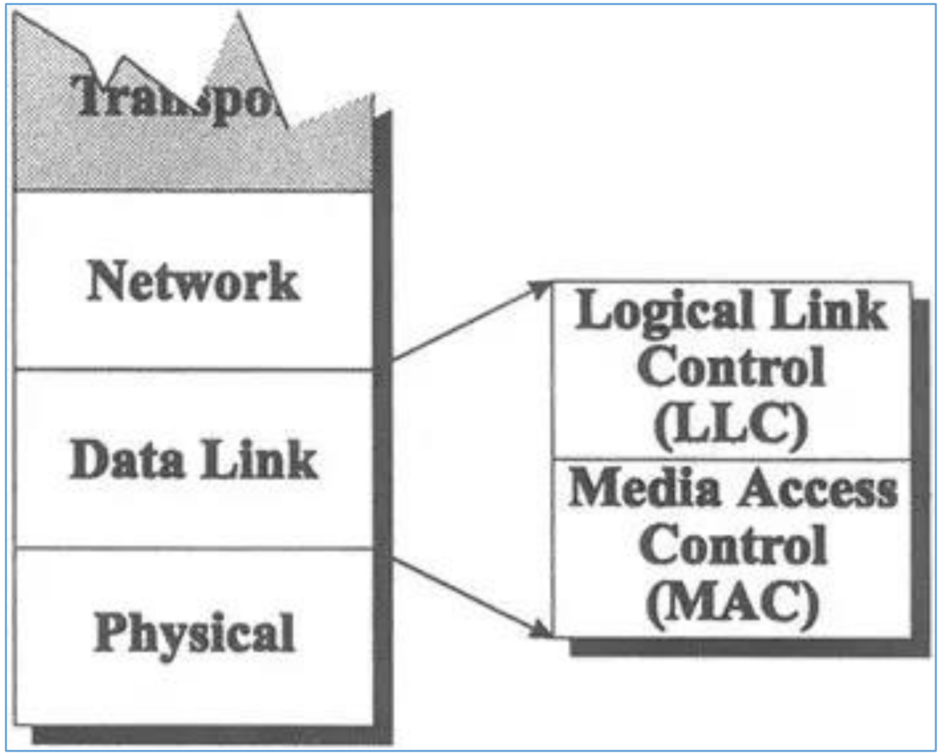


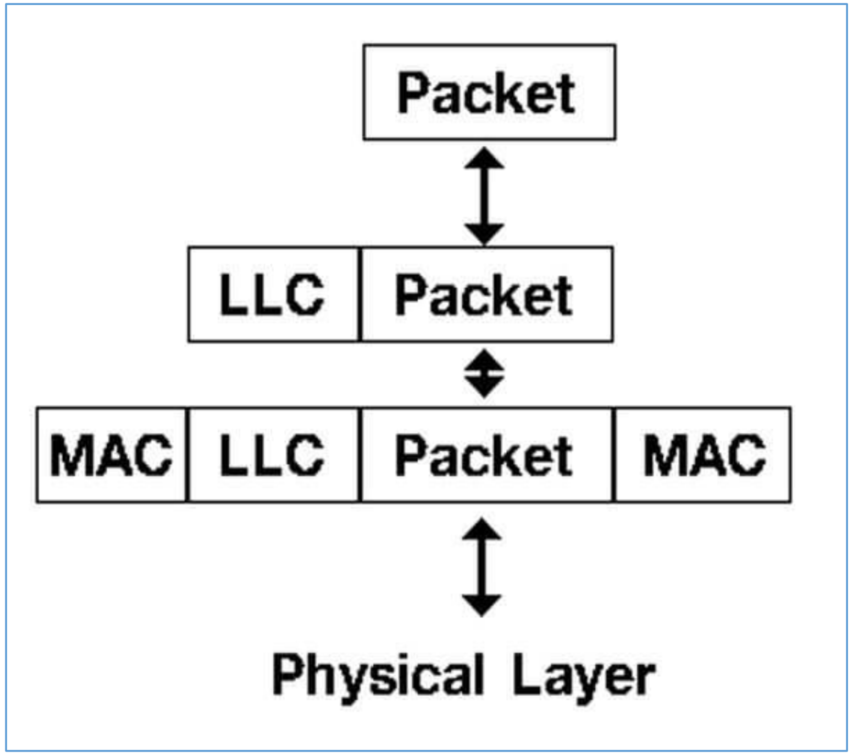
Responsible for **NODE-TO-NODE Communication**.

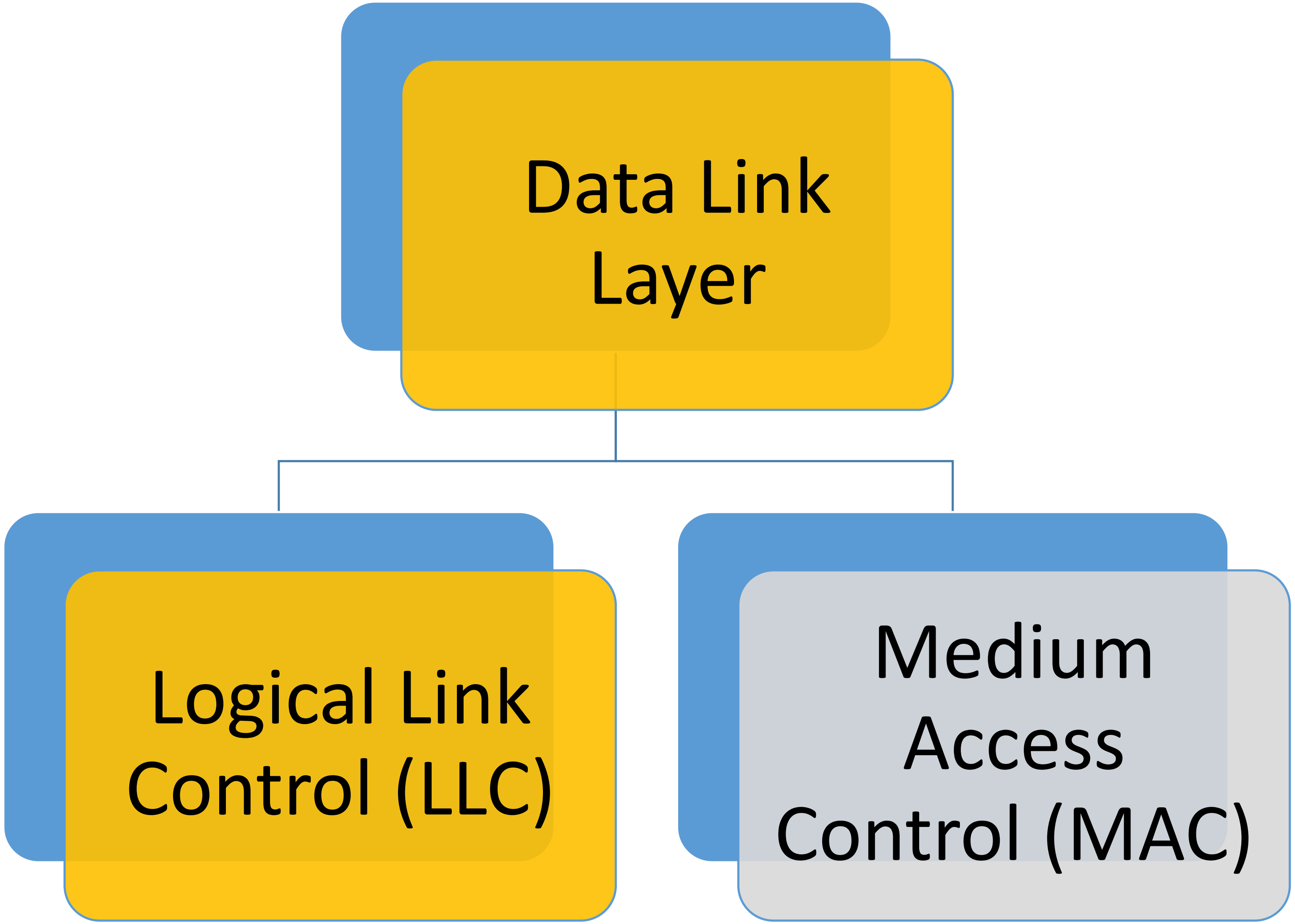
The data-link layer of the source host needs only to encapsulate, the data-link layer of the destination host needs to de-capsulate, but each intermediate node needs to both encapsulate and de-capsulate.

Data Link Layer

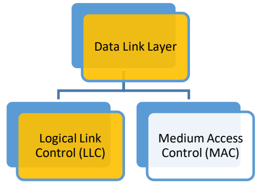
It has two sub-layers:






Data Link Layer
Logical Link Control (LLC)
Medium Access Control (MAC)

Data Link Layer | Logical Link Control (LLC)

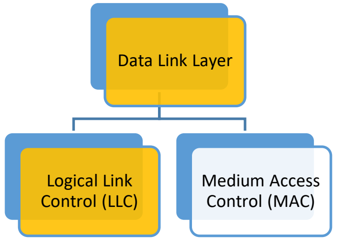


Deals with procedures for communication between two adjacent nodes (node-to-node communication) – no matter whether the link is dedicated or broadcast.

- **Framing**
  - Character Count
  - Character Stuffing
  - Bit Stuffing
- **Flow Control**
  - Stop-and-Wait
  - Sliding Window
- **Error Control**
  - Stop-and-Wait ARQ
  - Go-back-n ARQ
  - Selective-reject ARQ

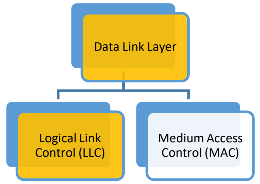
- **Error Detection and Correction**
  - Types of Errors
  - Detection
  - Correction
- **Protocol**
  - High-Level Data Link Control (HDLC)

Data Link Layer | Logical Link Control (LLC)



- **Framing**
  - **Character Stuffing**
  - **Bit Stuffing**
  - **Character Count**
  - **Physical Layer Coding Violation**
- Flow Control
  - Stop-and-Wait
  - Sliding Window
- Error Control
  - Stop-and-Wait ARQ
  - Go-back-n ARQ
  - Selective-reject ARQ
- Error Detection and Correction
  - Types of Errors
  - Detection
  - Correction
- Protocol
  - High-Level Data Link Control (HDLC)

Logical Link Control (LLC) | Framing



The physical layer provides bit synchronization to ensure that the sender and receiver use *the same bit durations and timing*.

The data-link layer needs to pack bits into frames, so that *each frame is distinguishable from another*.

Framing separates a message from source to a destination by adding a sender address and a destination address.

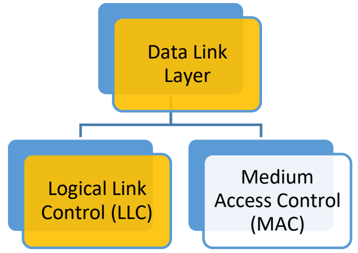
Can we put whole message in one frame?

- Flow and error control will become **inefficient for large frame**.
- Even a single-bit error would require the **re-transmission of the whole frame**.

Frames can be of fixed or variable size:

- In fixed-size framing, there is **no need for defining the boundaries of the frames**.
- In variable-size framing, need a way **to define** *the end of one frame and beginning of the next*.

Logical Link Control (LLC) | Framing



A <u>frame</u> is composed of <u>**four fields**</u>:

- **Kind:** tells whether the frame contains <u>data or control information</u>.
- **Seq:** tells about the <u>sequence number of the frame</u>.
- **Ack:** tells about the <u>acknowledgement of a frame</u>.

Control Information

- **Info:** <u>Only used in case of data frame</u>. It contains **a single packet**.

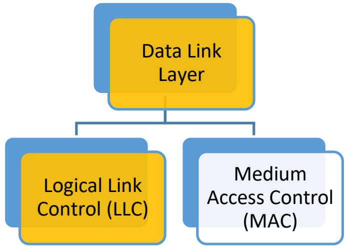
Actual Data

*The packet <u>from the network layer</u> is passed <u>to the data link layer</u> for the inclusion in to '**info**' field of an <u>outgoing frame</u>.*

When <u>the frame arrives at the destination</u>, <u>the data link layer extracts the packet from the frame</u> and <u>passes the packet to the network layer</u>.

The data link layer <u>**break the bit stream into discrete frames**</u>. <u>*This process is more difficult in case of variable-size framing*</u>.

Framing | Character Count



Methods to mark the START and END of each frame are:

Character Count

Uses a field in the header to specify **the number of characters in the frame**.

LOVE HATE MAKE A FUN

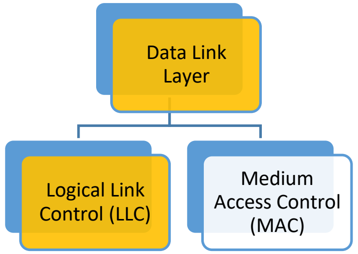
5	L	O	V	E	5	H	A	T	E	5	M	A	K	E	2	A	4	F	U	N

When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is.

Problem: Count can be garbled by a transmission error.

The character count method is **rarely used anymore**.

Framing | Character or Byte Stuffing



Character Count

Character or Byte Stuffing

This framing method gets around the problem of resynchronization after an error by having **each frame start and end with special bytes**.

To separate one frame from the next, **an 8-bit (1-byte) FLAG** is added at the beginning and the end of a frame.

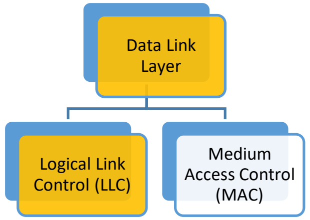
FLAG	**Header**	Payload Field (Data from Network Layer)	**Trailer**	FLAG

The **FLAG**, composed of protocol-dependent special characters, signals the start or end of a frame.

If the receiver ever loses synchronization, it can just search for the FLAG byte to find the end of the current frame.

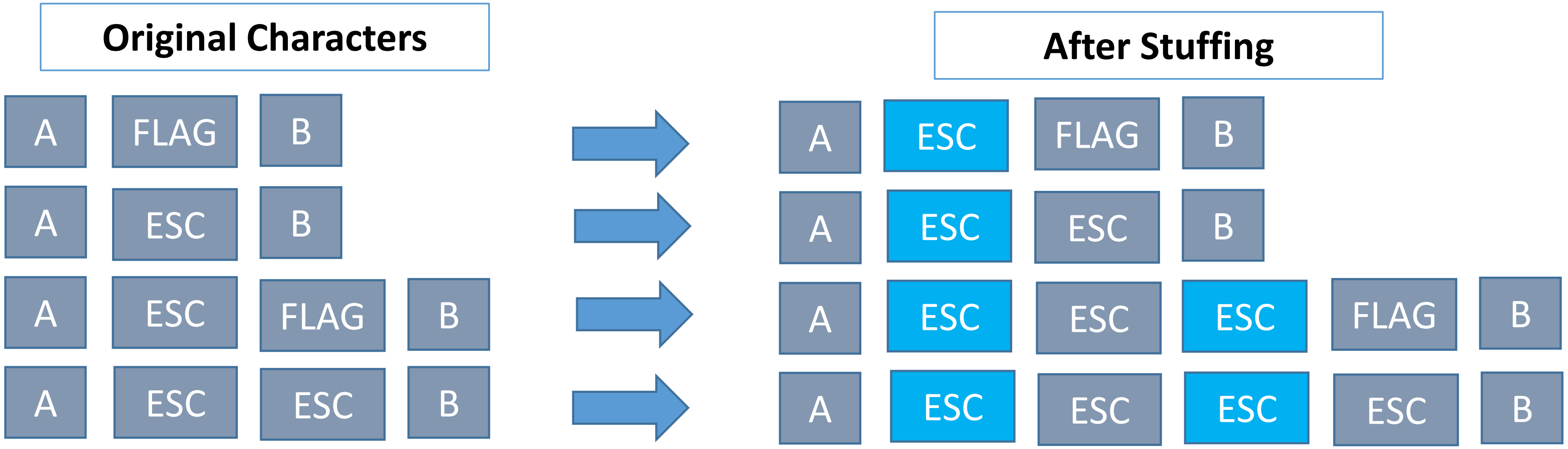
Two consecutive FLAG bytes indicate the end of one frame and start of the next one.

Framing | Character or Byte Stuffing



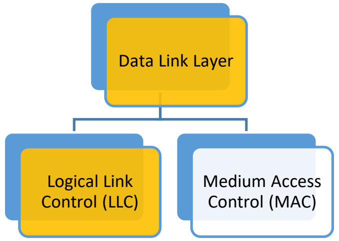
Problem: **<u>The FLAG bytes bit pattern may occurs in the data</u>**. *If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks <u>it has reached the end of the frame</u>.*

Solution: <u>A Character or byte-stuffing strategy was added to character-oriented framing.</u> A **special byte (ESC)** is <u>added to the data section of the frame when there is a character with the same pattern as the flag</u>.



Whenever the receiver encounters the ESC character, <u>it removes it from the data section</u> and <u>treats the next character as data, not as a delimiting flag</u>.

Framing | Bit Stuffing



Character Count | Character or Byte Stuffing

Bit Stuffing

The byte-stuffing framing is closely tied to the use of 8-bit characters (only ASCII).

Each frame begins and ends with a special bit pattern (01111110).

01111110	**Header**	Payload Field (data from Network Layer	**Trailer**	01111110

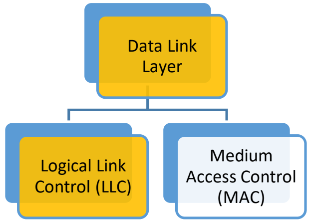
Whenever the sender's data link layer encounters five consecutive 1s in the data, **it automatically stuffs a 0 bit into the outgoing bit stream** – This is called **BIT STUFFING**.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, **it automatically de-stuffs the 0 bit**.

Original Data	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0			
Outgoing Data	0	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	0	0	1	0

Even if there is a 0 after five 1s, still stuff a 0. The 0 will be removed by the receiver.

Framing | Physical Layer Coding Violations



Physical Layer Coding Violations

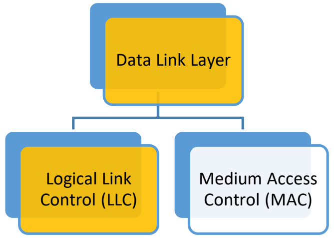
This method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy.

For example, some LANs encode 1 bit of data by using 2 physical bits.

Normally, a **1 bit is a high-low pair** and **0 bit is a low-high pair**.

The combinations high-high and low-low are not used for data.

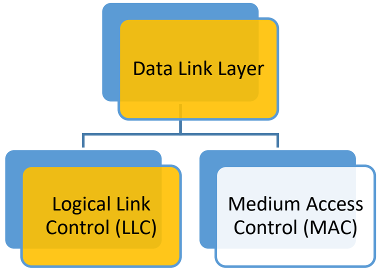
Data Link Layer | Logical Link Control (LLC)



- Framing
  - Character Count
  - Character Stuffing
  - Bit Stuffing
  - Physical Layer Coding Violation
- **Flow Control**
  - **Stop-and-Wait**
  - **Sliding Window**
- Error Control
  - Stop-and-Wait ARQ
  - Go-back-n ARQ
  - Selective-reject ARQ

- Error Detection and Correction
  - Types of Errors
  - Detection
  - Correction
- Protocol
  - High-Level Data Link Control (HDLC)

Logical Link Control (LLC) | Flow Control



Set of procedure to tell the sender that **how much data it can transmit before it must wait for an acknowledgement from the receiver**.

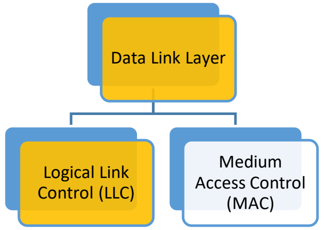
Each receiving device has its **limited speed of processing data** and has a **limited amount of memory (buffer)** to store such data.

The receiving device must be able to inform the sending device **before those limits are reached and to request that the sending device send fewer frames or stop temporarily**.

Two methods have been developed **to control the flow of data** across communication links:

- **Stop-and-Wait**
- **Sliding Window**

Logical Link Control (LLC) | Flow Control



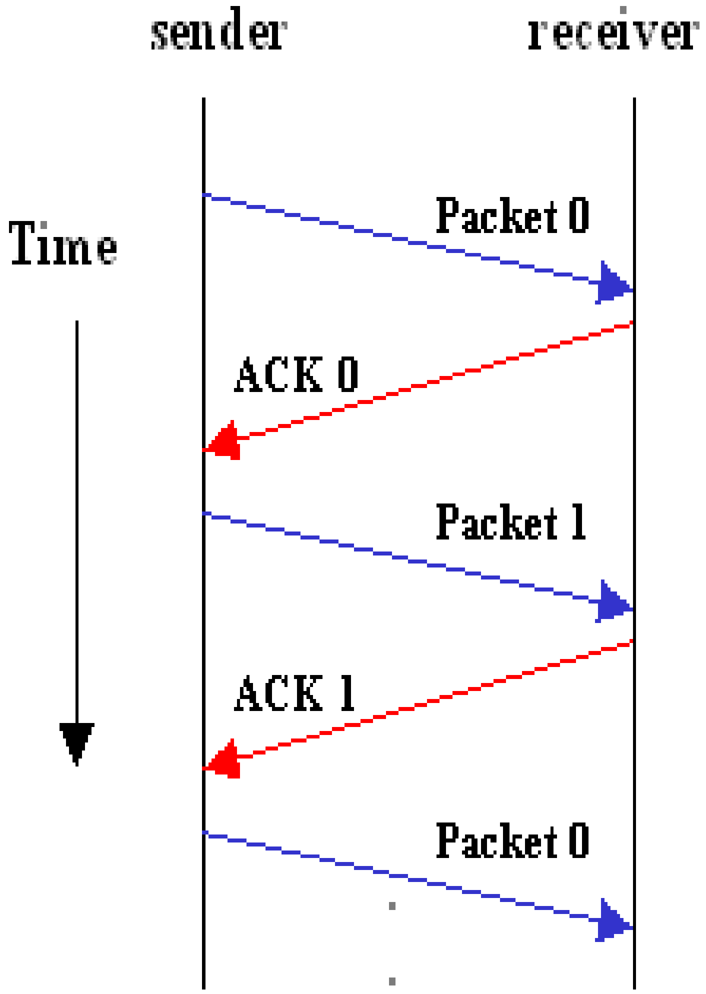
- **Stop-and-Wait Protocol**

The sender **waits for an acknowledgement after every frame it sends**.

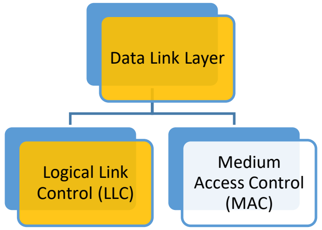
Next frame is sent **on receipt of an acknowledgement** of the previously sent frame.

Advantage: **Simplicity**. Each frame is checked and acknowledged before the next frame is sent.

Disadvantage: **Inefficiency**. Protocol is slow.



Flow Control | Stop-and-Wait Protocol



Calculation for an Efficiency/Utilization of Link:

Let us determine the **maximum potential efficiency** of a **Half-Duplex Point-to-Point Link**.

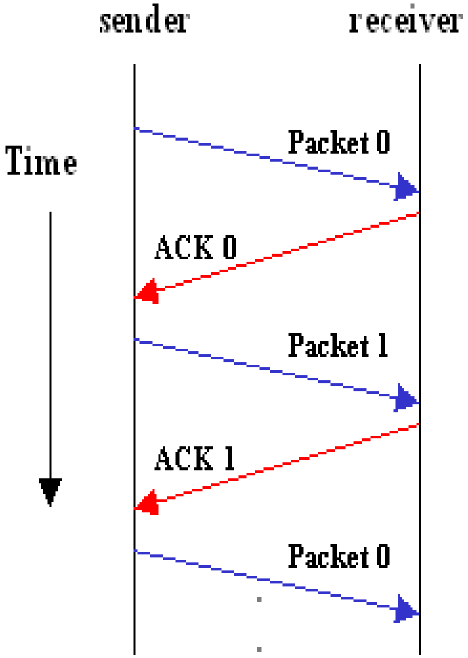
Suppose that a long message is to be sent as a sequence of frames F_1, F_2, …..,F_n from station S_1 to S_2 in the following fashion:

- Station S_1 sends F_1
- Station S_2 sends an ACK
- Station S_1 sends F_2
- Station S_2 sends an ACK

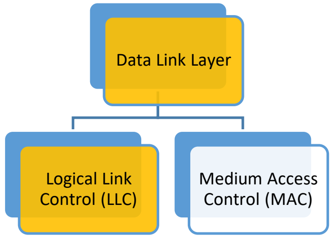
.

.

- Station S_1 sends F_n
- Station S_2 sends an ACK



Flow Control | Stop-and-Wait Protocol



The total time (T) to send the data can be expressed as:

$\boldsymbol{T = n * T_F}$, where T_F is the time to send one frame and receive an ACK.

T_F can be expressed as:

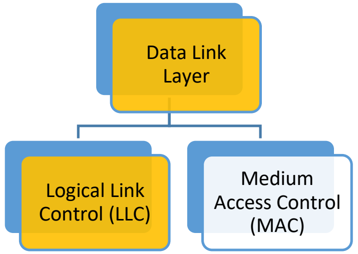
$$\mathbf{T_F = \; t_{prop} + t_{frame} + t_{proc} + t_{prop} + t_{ack} + t_{proc}}$$

where t_{prop} is propagation time from S_1 to S_2 or from S_2 to S_1.
t_{frame} is time to transmit a frame (time for the transmitter to send out all the bits of the frame).
t_{proc} is processing time at each station to react to an incoming event.
t_{ack} is time to transmit an ACK.

Assume, t_{proc} is relatively negligible.
t_{ack} is also very small, as the ACK frame is very small compared to the data frame.

$$\mathbf{T = n * \left(2 * t_{prop} + \; t_{frame}\right)}$$

Flow Control | Stop-and-Wait Protocol



$$\boxed{\mathbf{T = n * \left(2 * t_{prop} + \; t_{frame}\right)}}$$

<u>Only $(n * \; t_{frame})$ is actually spent transmitting data</u> and the rest is overhead.

The maximum possible utilization or efficiency of the link is:

$$\boxed{U = \frac{n * t_{frame}}{n * \left(2 * t_{prop} + t_{frame}\right)}}$$

$$\boxed{\boldsymbol{U = \frac{1}{1 + 2a} \qquad where\ a = \frac{t_{prop}}{t_{frame}}}}$$

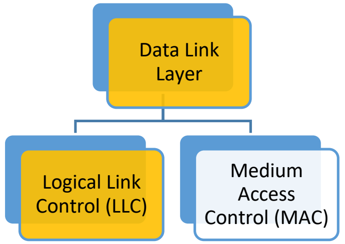
$$\boxed{\mathbf{t_{prop}} = \frac{\textbf{Distance of the Link}}{\textbf{Velocity of Propagation}} = \frac{\mathbf{d}}{\mathbf{V}}}$$

'V' is 3×10^8 m/sec in air or space and is 0.67 times the speed of light in cable.

$$\boxed{\mathbf{t_{frame}} = \frac{\textbf{Length of the frame}}{\textbf{Data Rate}} = \frac{\mathbf{L}}{\mathbf{R}}}$$

$$\boxed{\textbf{Therefore, } \boldsymbol{a = \frac{R * d}{V * L}}}$$

Flow Control | Stop-and-Wait Protocol



$$\mathbf{U} = \frac{\mathbf{1}}{\mathbf{1+2a}}, \qquad \text{where } \mathbf{a} = \frac{\mathbf{t_{prop}}}{\mathbf{t_{frame}}} = \frac{\mathbf{R * d}}{\mathbf{V * L}}$$

Case I: a < 1

Frame Size > Link Length

Propagation Time < Transmission Time

Transmission started at t_0.

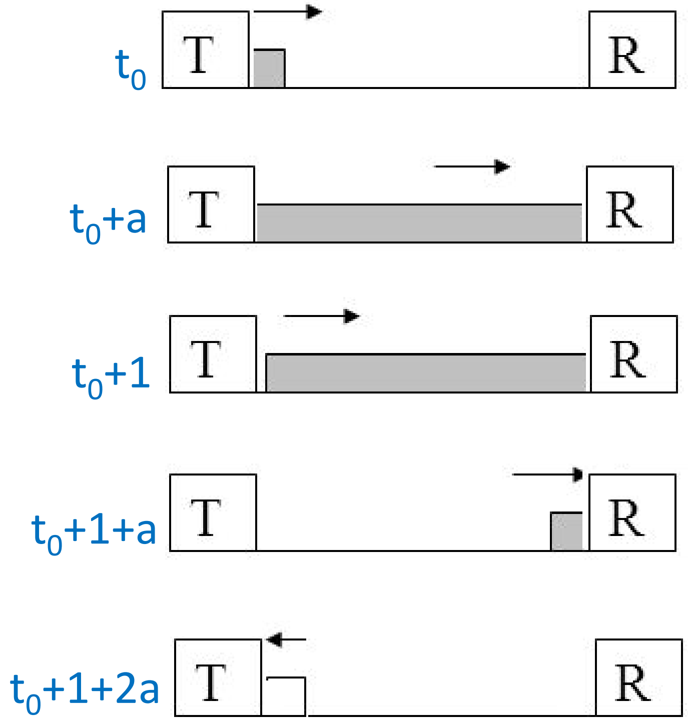
Acknowledgement arrives back to sender at t_0+1+2a.

Total time elapsed = (t_0+1+2a) - t_0

= (1+2a)

Total Transmission time = 1

$$\boldsymbol{Utilization} = \frac{\mathbf{1}}{\mathbf{1+2}\boldsymbol{a}}$$



Transmission started at t_0.

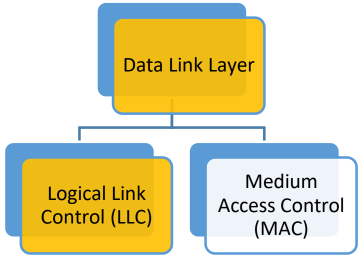
First bits of the frame arrived at the receiver before the source has completed the transmission of the frame.

Total Transmission time = 1

Last bit of the frame arrived at the receiver.

Acknowledgement arrives back to sender at t_0+1+2a.

Flow Control | Stop-and-Wait Protocol



$$\mathbf{U} = \frac{\mathbf{1}}{\mathbf{1+2a}}, \qquad \text{where } \mathbf{a} = \frac{\mathbf{t_{prop}}}{\mathbf{t_{frame}}} = \frac{\mathbf{R*d}}{\mathbf{V*L}}$$

Case II: a > 1

Frame Size < Link Length

Propagation Time > Transmission Time

Transmission started at t_0.

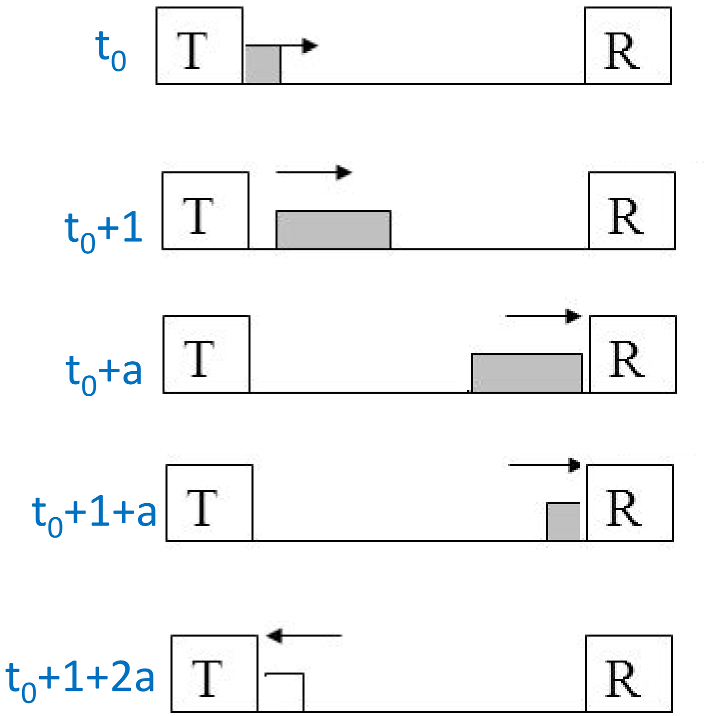
Acknowledgement arrives back to sender at t_0+1+2a.

Total time elapsed = (t_0+1+2a) - t_0

= (1+2a)

Total Transmission time = 1

$$\textbf{Utilization} = \frac{\mathbf{1}}{\mathbf{1+2a}}$$



Transmission started at t_0.

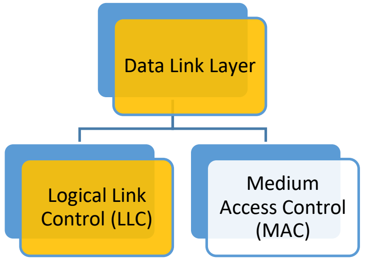
Total Transmission time = 1

First bits of the frame arrived at the receiver.

Frame completely arrived at the receiver.

Acknowledgement arrives back to sender at t_0+1+2a.

Logical Link Control (LLC) | Flow Control



- **Sliding Window Protocol**

There is a need for transmitting data in both directions (**Full Duplex**).

Options:

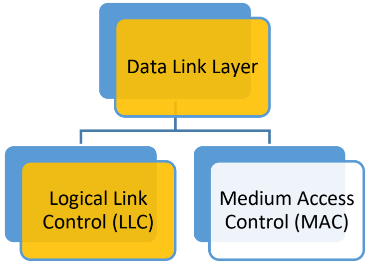
- Have **two separate communication channels** and use each one for simplex data traffic (in different direction).

  - There will be **two separate physical circuits** each with forward and reverse channel.
  - Bandwidth of the reverse channel is almost **entirely wasted**.
  - The user will **PAY for two circuits** and will **USE only the capacity of one**.

- Use **same circuit for data in both directions**.

  - **Data frames** from A to B will inter-mixed with the **ACK frames** from A to B.
  - ‘**kind**’ field of the header will tell receiver about the type of frame arrived.

Flow Control | **Sliding Window Protocol**



Options:

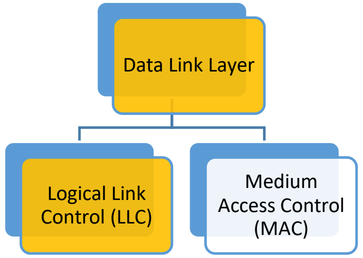
- Have **two separate communication channels** and use each one for simplex data traffic (in different direction).

- Use **same circuit for data in both directions**.

- When a data frame arrives, instead of immediately sending a separate control frame, the **receiver restrains itself** and **waits until the network layer passes it the next packet**.

  - The ACK is attached to the outgoing data frame (by setting '**ack**' field in header).
  - The **ACK gets a free ride** *on the next outgoing data frame*. This is called **PIGGYBACKING**.
  - **Better use of the available channel bandwidth.**

Flow Control | Sliding Window Protocol



The sender can **transmit several frames** **before needing an ACK**.

The receiver acknowledges only some of the frames, using **a single ACK** *to confirm the **receipt of multiple data frames***.

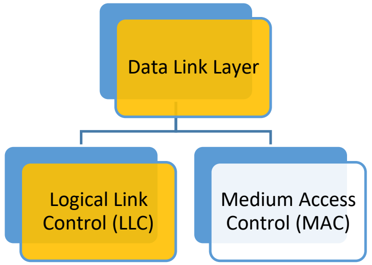
The **sliding window** refers to **imaginary boxes at both the sender and the receiver**. *This window can hold frames at either end* and *provides the upper limit on the number of frames* that can be transmitted **before requiring an ACK**.

Frames may be acknowledged at any point without waiting for the window to fill up and may be transmitted as long as the window is not yet full.

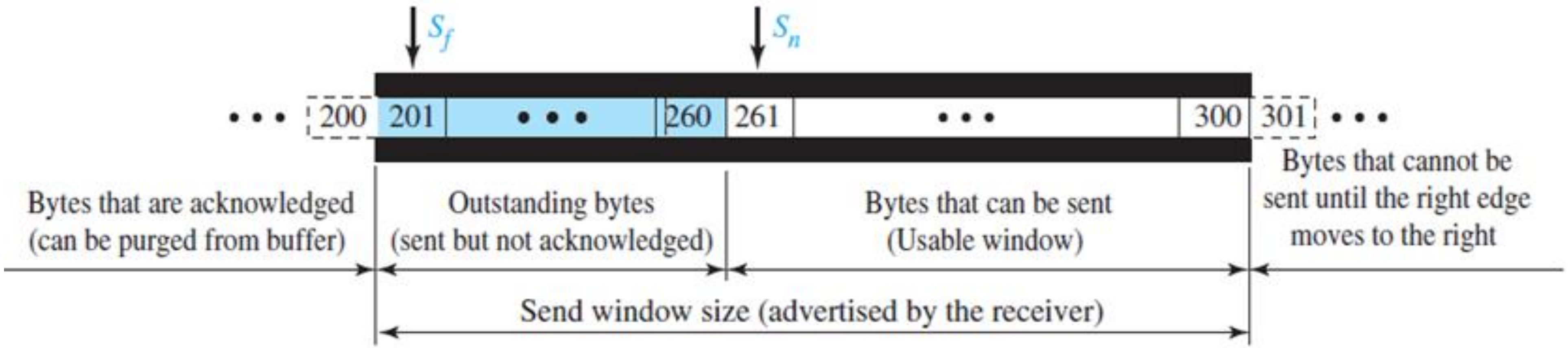
To keep track of which frames have been transmitted and which received, sliding window introduces **an identification scheme based on the size of the window**. **The frames are numbered modulo-n**, which means they are numbered from 0 to n-1.

When the receiver sends an ACK, **it includes the number of the next frame it expects to receive**.

Flow Control | Sliding Window Protocol



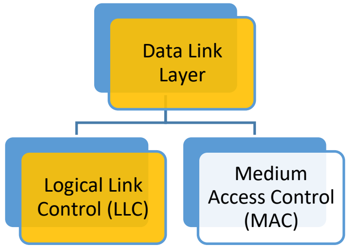
Sender Window



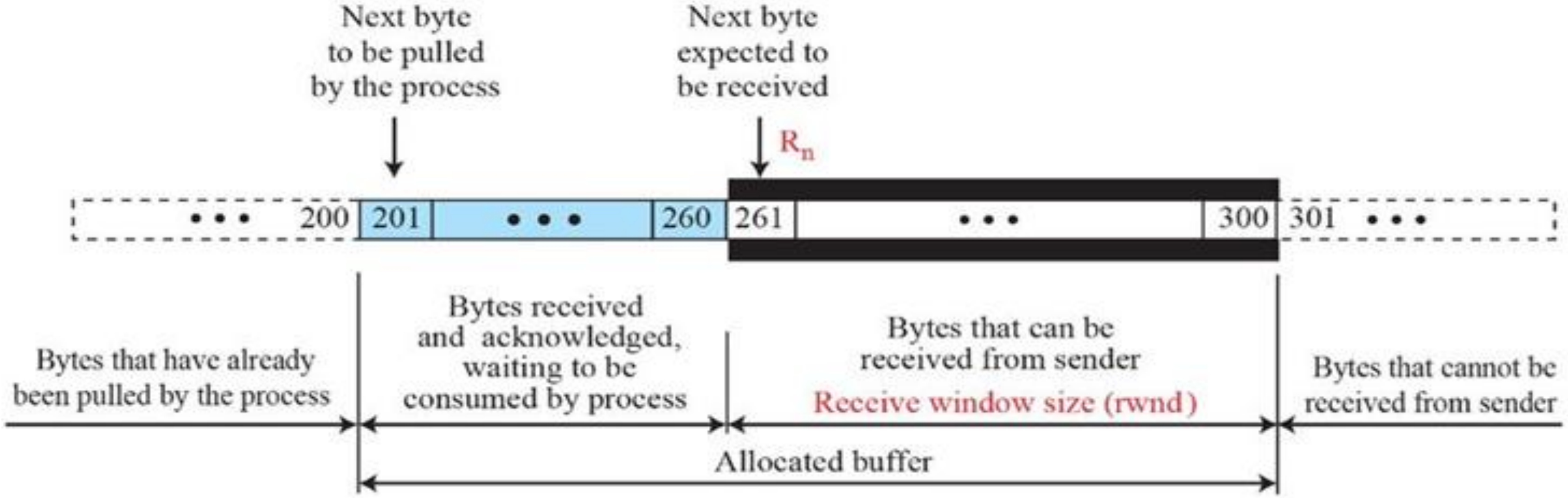
Send window

Opening, closing, and shrinking send window

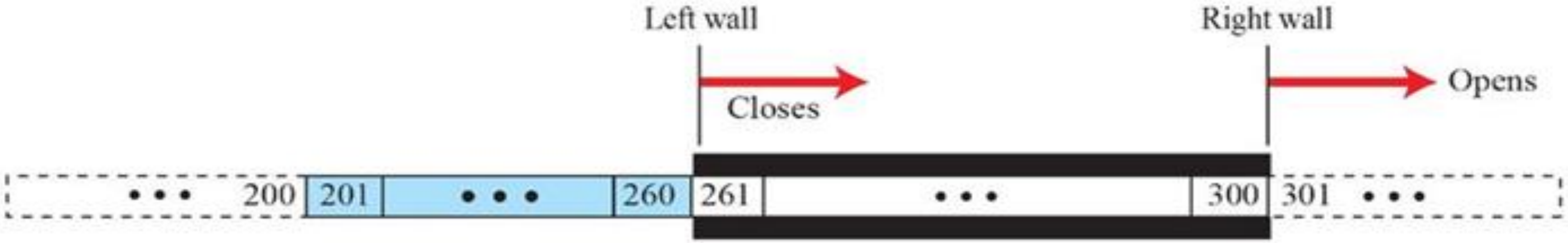
Flow Control | Sliding Window Protocol



Receiver Window

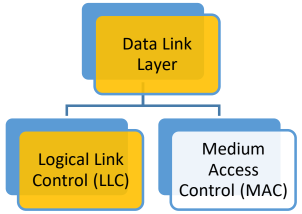


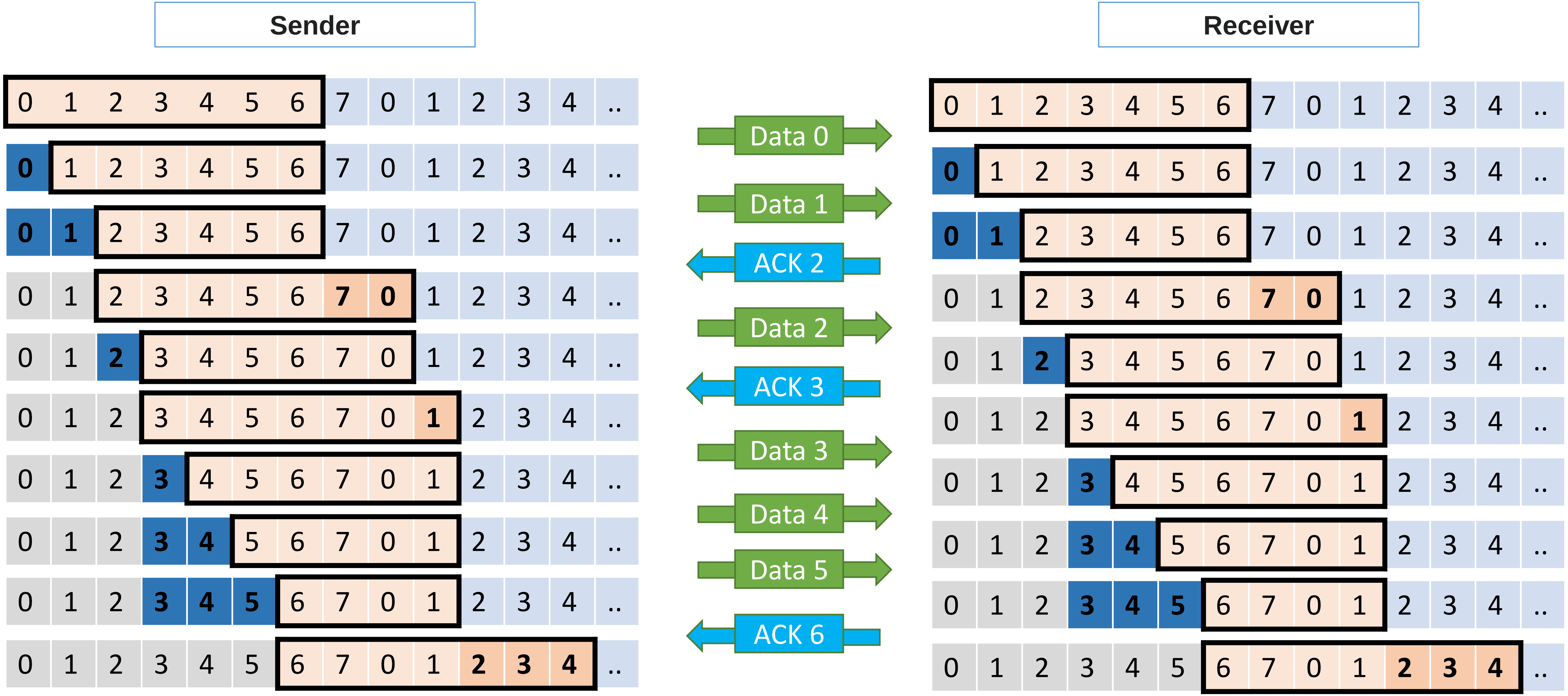
Receive window and allocated buffer



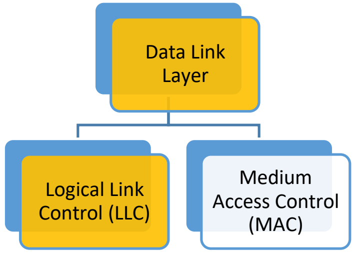
Opening and closing of receive window

Flow Control | Sliding Window Protocol



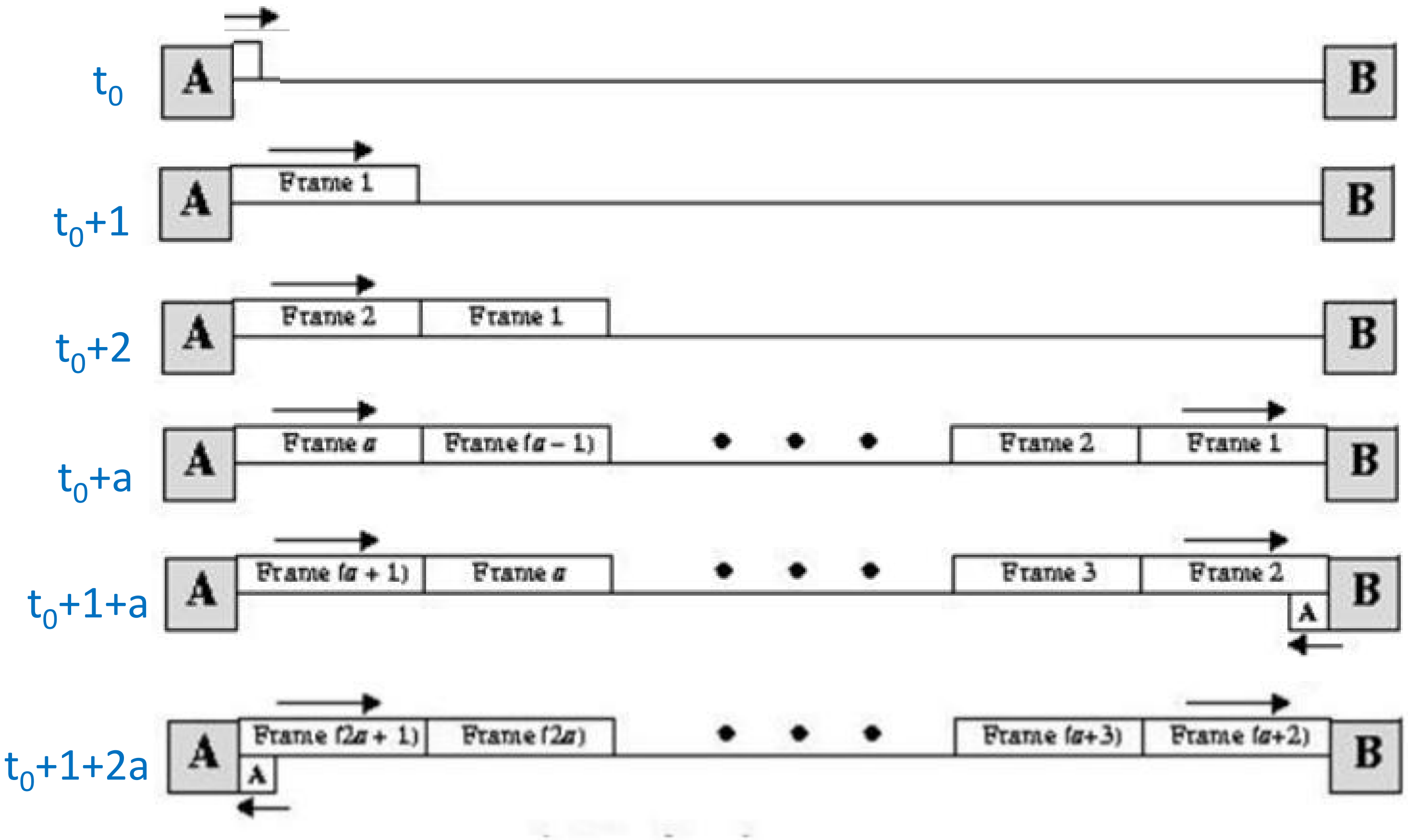


Flow Control | Sliding Window Protocol



Calculation for an efficiency of link:

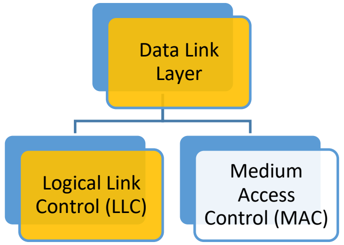
Case I: W >= 2a+1 where, $a = \frac{t_{prop}}{t_{frame}} = \frac{R * d}{V * L}$



Normalized Throughput/Efficiency is 1.

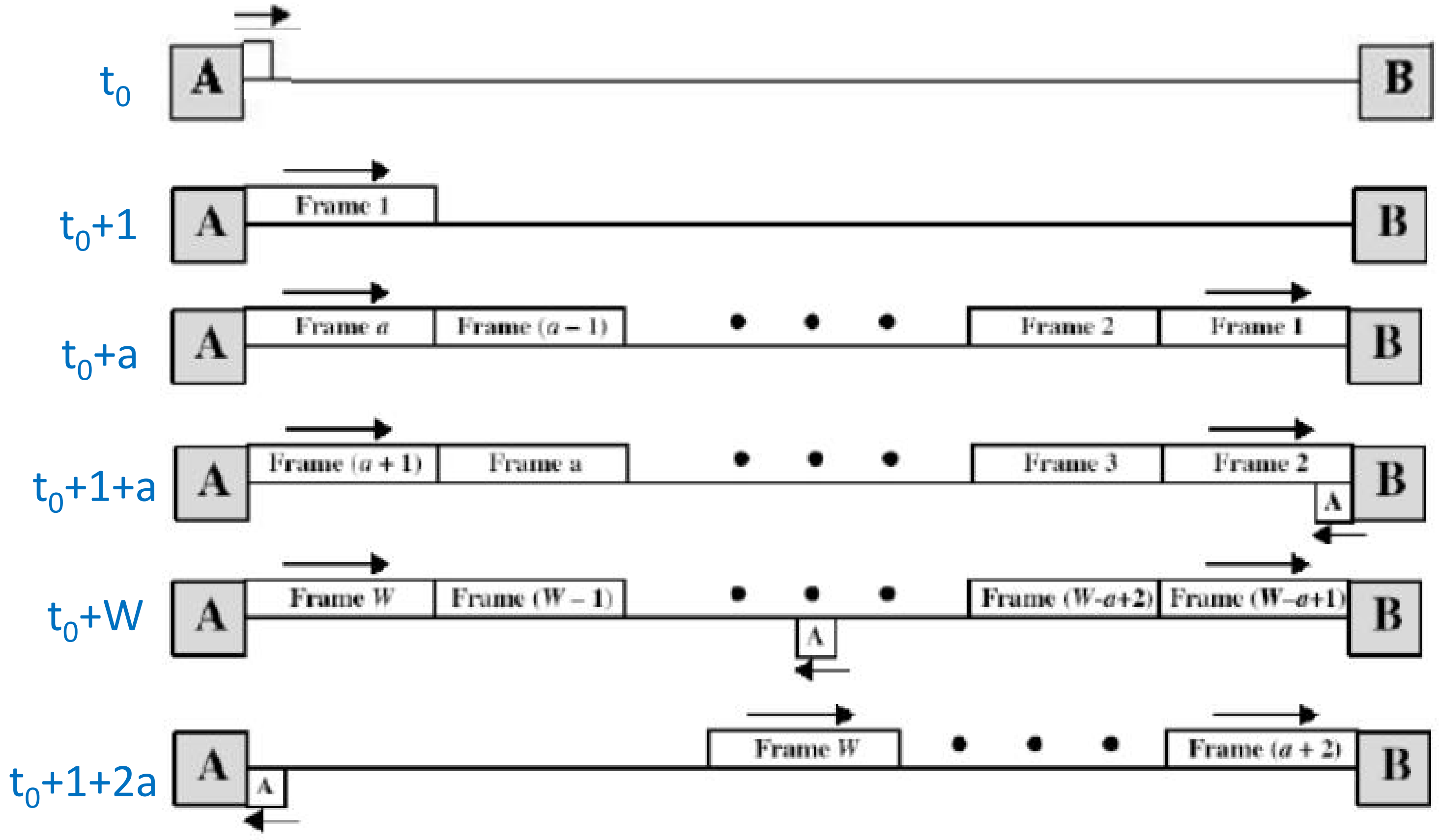
The ACK for frame 1 reaches 'A' before 'A' has exhausted its window. Thus, <u>'A' can transmit continuously with no pause</u>.

Flow Control | Sliding Window Protocol



Case II: W < 2a+1

where, $a = \dfrac{t_{prop}}{t_{frame}} = \dfrac{R * d}{V * L}$

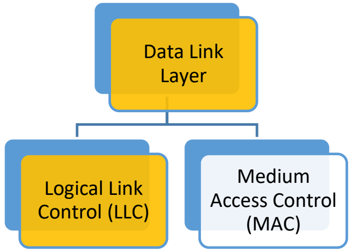


$$\text{Utilization} = \begin{cases} 1, & W \geq 2a+1 \\ \dfrac{W}{2a+1}, & W < 2a+1 \end{cases}$$

Normalized Throughput/Efficiency is ‘W’ time units out of a period of (2a+1) time units.

‘A’ exhausts its window at t=W and cannot send additional frames until t=2a+1.

Flow Control | Sliding Window Protocol

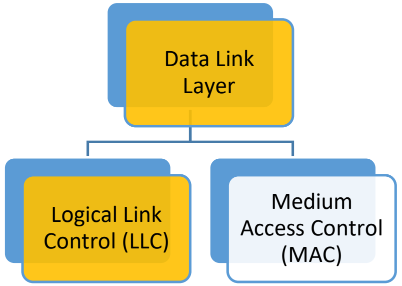


One Bit Sliding Window

Protocol with **a maximum window size of 1**.

Uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.

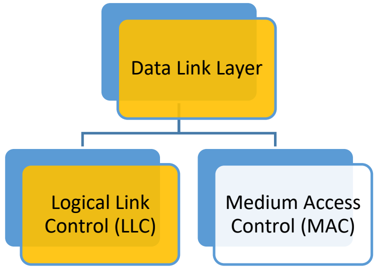
Data Link Layer | Logical Link Control (LLC)



- Framing
  - Character Count
  - Character Stuffing
  - Bit Stuffing
  - Physical Layer Coding Violation
- Flow Control
  - Stop-and-Wait
  - Sliding Window
- **Error Control**
  - **Stop-and-Wait ARQ**
  - **Go-back-n ARQ**
  - **Selective-reject ARQ**

- Error Detection and Correction
  - Types of Errors
  - Detection
  - Correction
- Protocol
  - High-Level Data Link Control (HDLC)

Logical Link Control (LLC) | Error Control



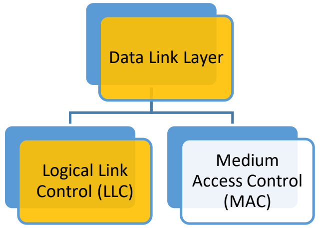
The underlying technology **at the physical layer** is **not fully reliable**, **there is a need to implement** error control at the data-link layer **to prevent the receiving node from delivering corrupted packets to the network layer**.

The term error control refers primarily to methods of **Error Detection and Retransmission**.

Feedback from the receiver is required to ensure reliable delivery of frames. The receiver is expected to send back special control frames bearing **positive or negative acknowledgements (ACK or NAK)** about the incoming frames.

When the sender transmits a frame, it also **starts a timer**. *The timer is **set to go off** after an interval long enough for the frame to reach the destination, be processed there, and have the ACK propagate back to the sender.*

Logical Link Control (LLC) | Error Control

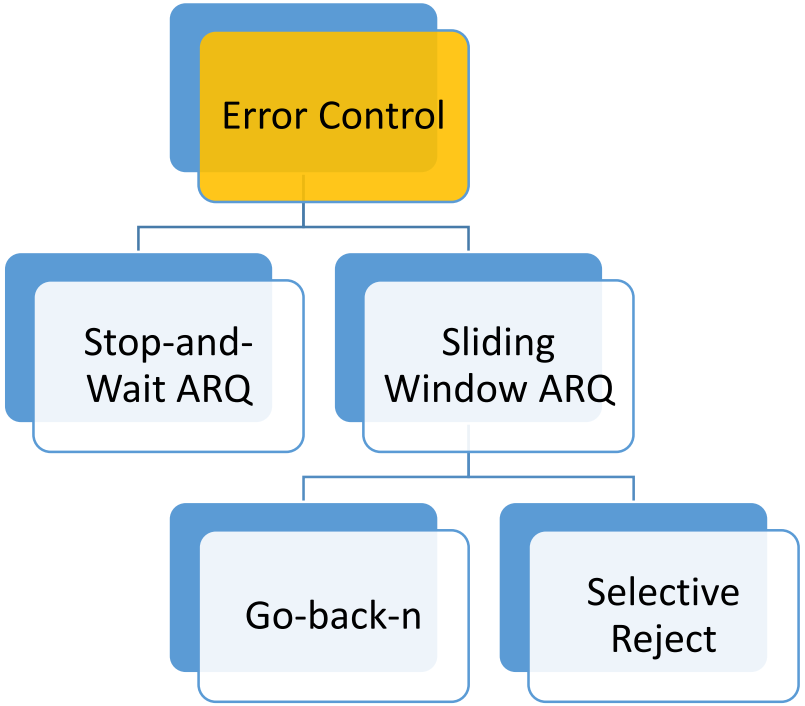


Error Detection and Retransmission

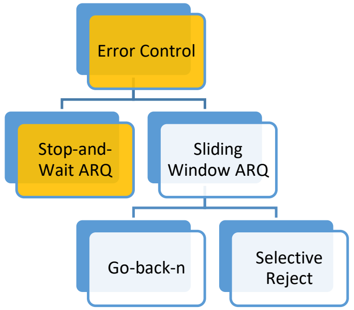
Anytime an error is detected in an exchange, **a negative acknowledgement (NAK) is returned** and **the specified frames are re-transmitted**. This process is called **AUTOMATIC REPEAT REQUEST (ARQ)**.

Receipt of the Damaged Frame (due to noise in transmission) will be treated as the **frame has been lost**.

The automatic re-transmission of lost frames, including **lost acknowledgement (ACK)** and **lost negative acknowledgement (NAK) frames**.



Logical Link Control (LLC) | Error Control



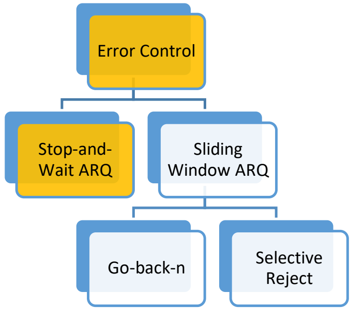
Stop-and-Wait ARQ

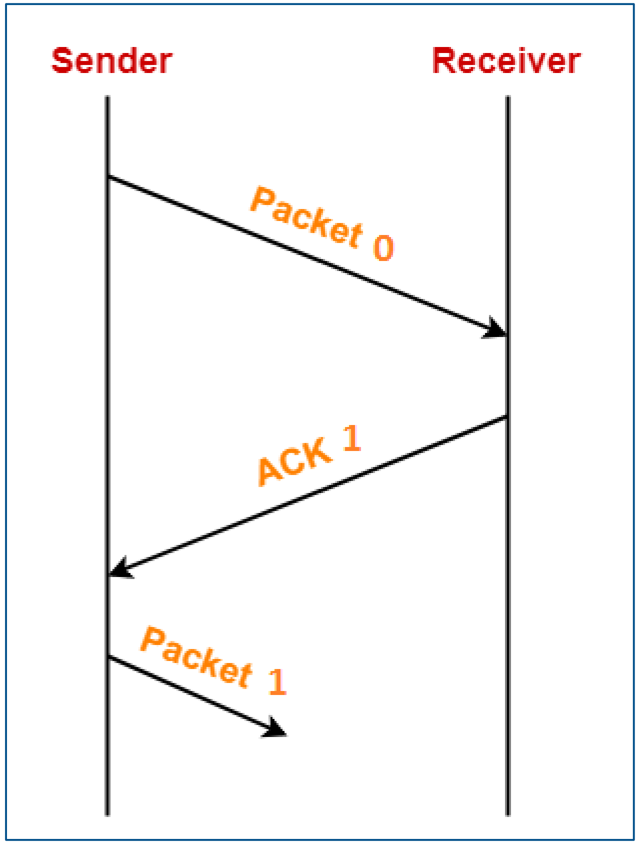
It is a form of Stop-and-Wait flow control extended to include **re-transmission of data** in case of lost or damaged frames.

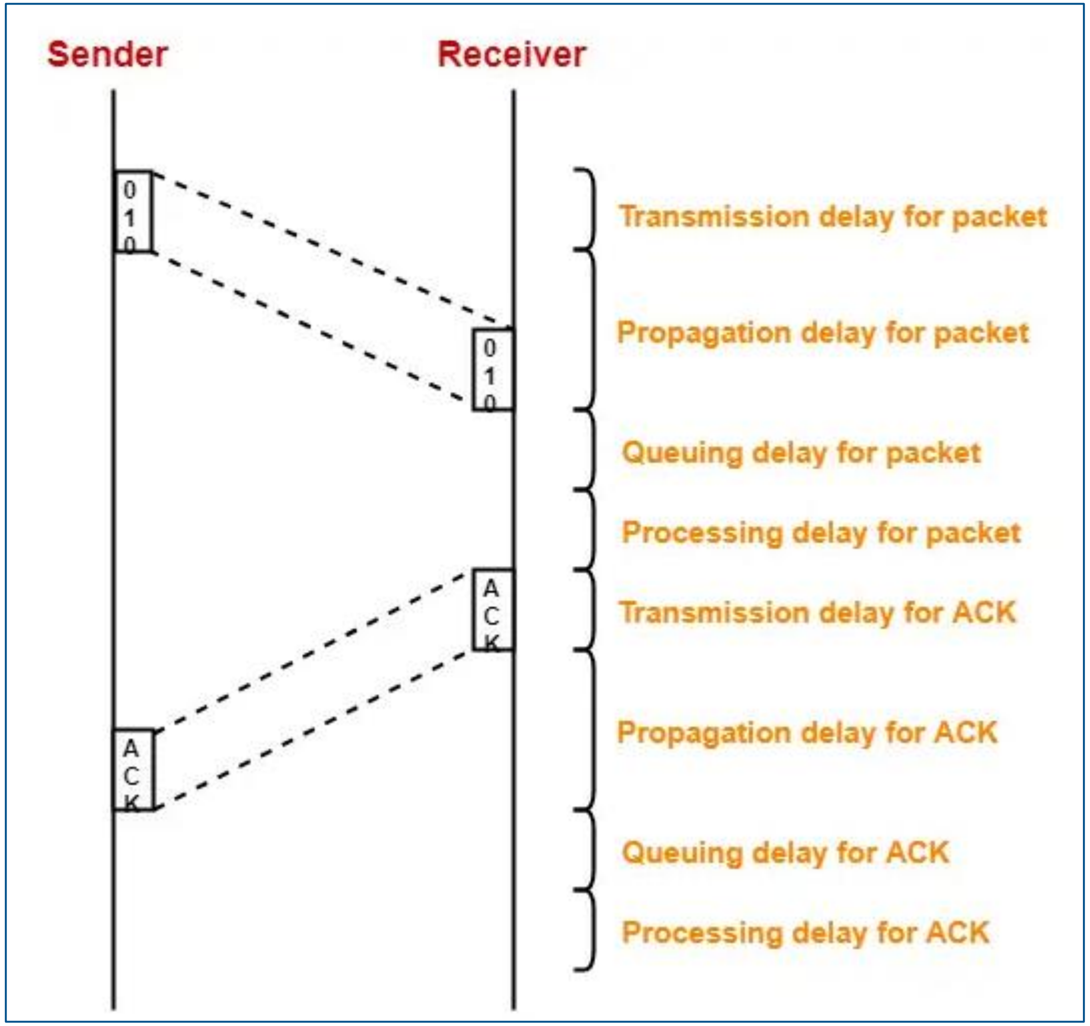
Four features are added to the basic Stop-and-Wait flow control mechanism**:**

- **The sending device keeps a copy of the last frame transmitted** until it receives an ACK for that frame.
- **For identification purposes**, both data frames and ACK frames are **numbered alternately 0 and 1**.
- **A NAK frame is returned** on receipt of the corrupted frame at the receiver end. **NAK frames are unnumbered**.
- The sending device is equipped with a timer. If an expected ACK is not received within an allotted time period, the sender assumes that the last data frame was lost in transit and **sends it again**.

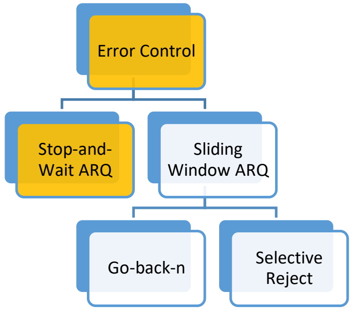
Error Control | Stop-and-Wait ARQ



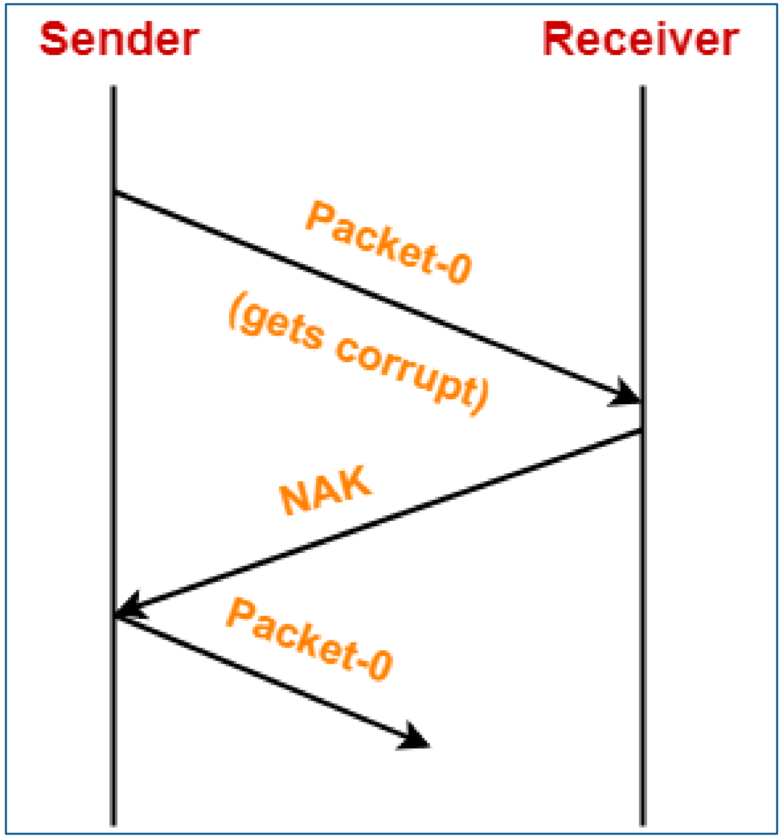




Error Control | Stop-and-Wait ARQ



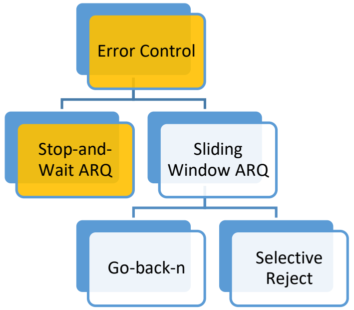
Damaged Frame



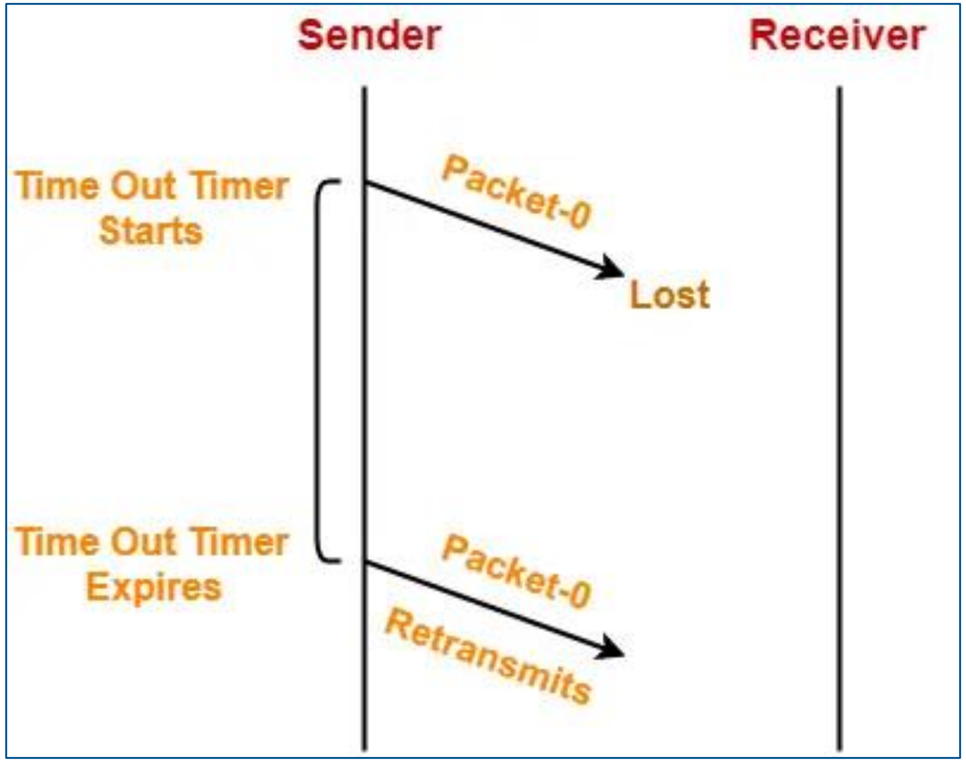
Frame is discovered by the receiver to contain an error

The sender re-transmit the frame on receipt of NAK

Error Control | Stop-and-Wait ARQ



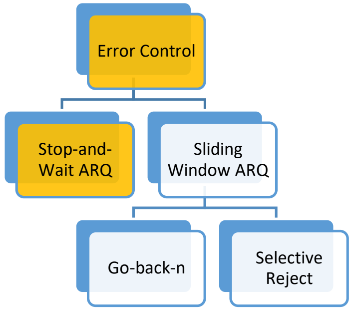
Lost Data Frame



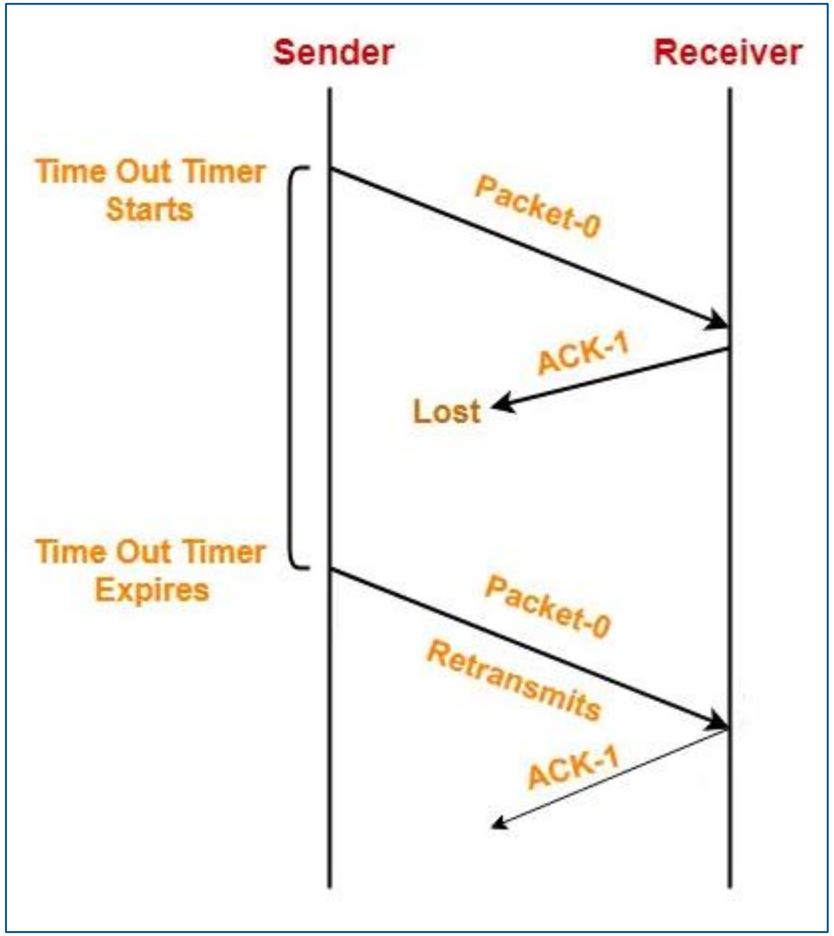
The sender waits for an ACK or NAK frame **until its timer goes off**, at which point it tries again.

- Re-transmits the last data frame.
- Restarts its timer.
- Wait for an ACK.

Error Control | Stop-and-Wait ARQ



Lost Acknowledgement



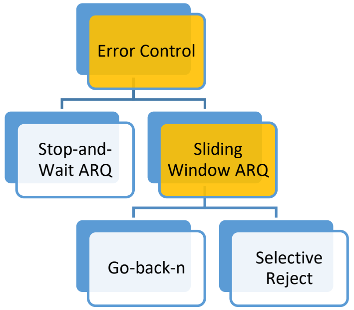
The sender waits for an ACK or NAK frame **until its timer goes off**, at which point it tries again.

- Re-transmits the last data frame.
- Restarts its timer.
- Wait for an ACK.

Discards duplicate packet (Packet-0) and resends ACK1.

If the lost frame was a **NAK**, accepts the new copy of packet (Packet-0) and **returns the appropriate ACK**.

Logical Link Control (LLC) | Error Control



Sliding Window ARQ

Three features are added to the basic Sliding Window flow control mechanism:

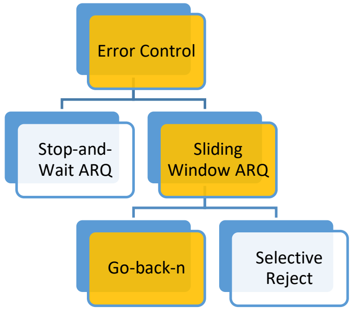
- The sending device **keeps copies of all transmitted frames** until they have been acknowledged.
- **Both ACK and NAK frames** **must be numbered for identification**. Data fames that are received without errors do not have to be acknowledged **individually**. However, every damaged frame must be acknowledged immediately.
- The sending device is equipped with **a timer** to enable it to handle lost acknowledgements (ACK/NAK).

n-1 frames may be sent before an ACK must be received.

If n-1 frames (*the size of the window*) are awaiting ACK, **the sender starts a timer and waits before sending any more**.

If the allotted time has run out with **no ACK**, the sender assumes that the **frames were not received** and **retransmit one or all of the frames depending on the protocol**.

Error Control | Sliding Window ARQ



Go-back-n ARQ

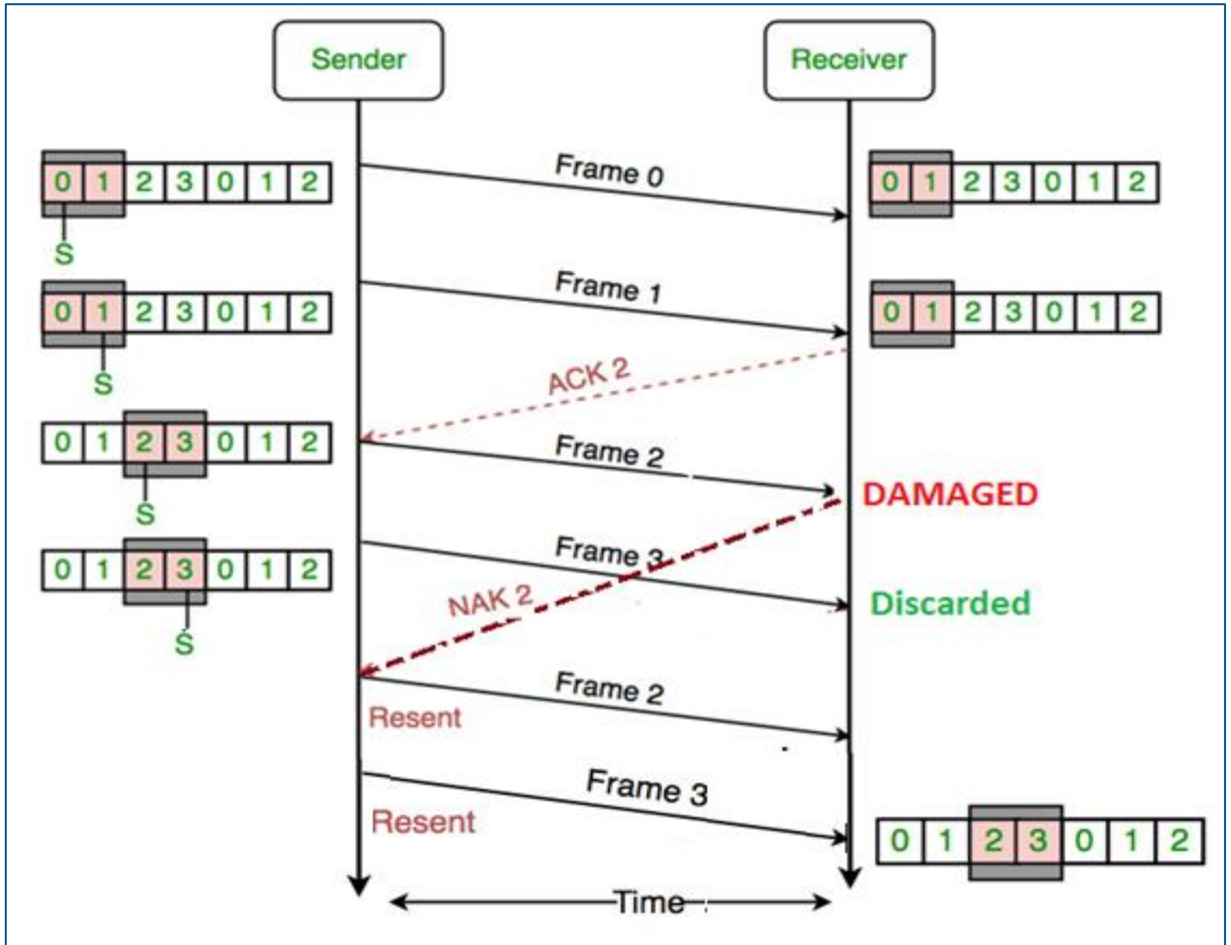
If one frame is lost or damaged, **all frames sent since the last frame acknowledged are retransmitted**.

Damaged Frame

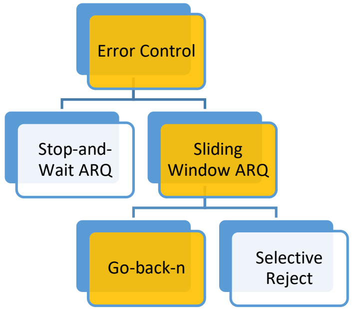
As soon as receivers recovers problem, it stops accepting subsequent frames until the damaged frame has been replaced correctly.

Receiver send NAK to the sender. Here, NAK2 means either the frame 2 is **damaged or missing**.

On the receipt of NAK#, **the sender retransmits all frames starting from #**.



Error Control | Sliding Window ARQ



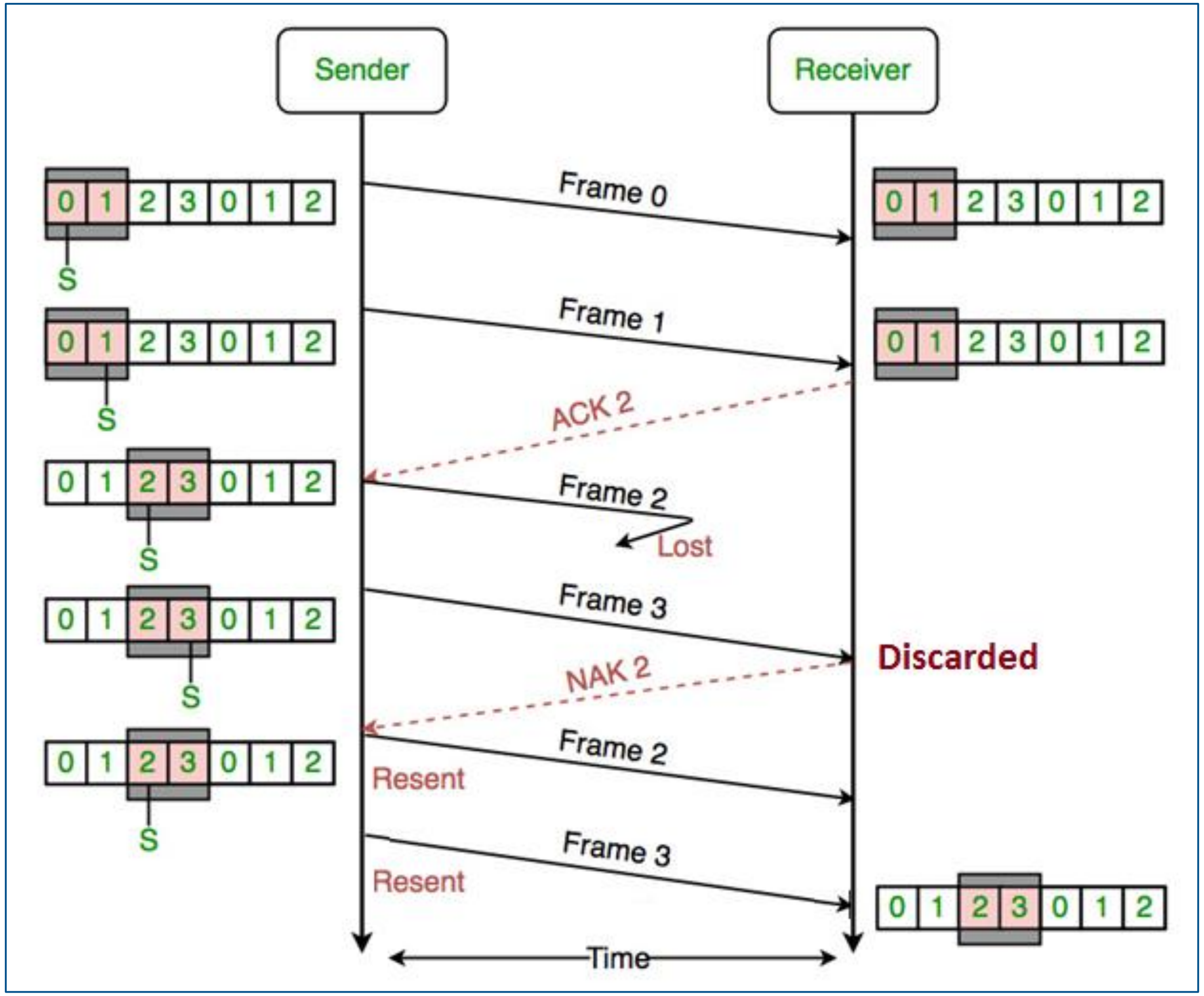
Go-back-n ARQ | Lost Data Frame

The data frames **must be transmitted sequentially**.

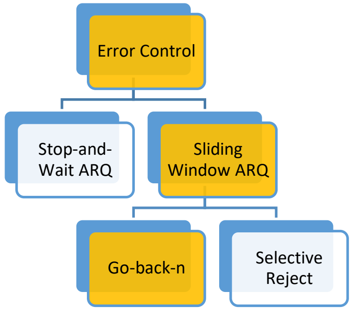
The receiver checks the identifying number on each frame, **discovers that one or more have been skipped**, and returns a NAK for the first missing frame.

NAK2 means either the frame 2 is damaged or missing.

On the receipt of NAK#, **the sender retransmits all frames starting from #.**



Error Control | Sliding Window ARQ



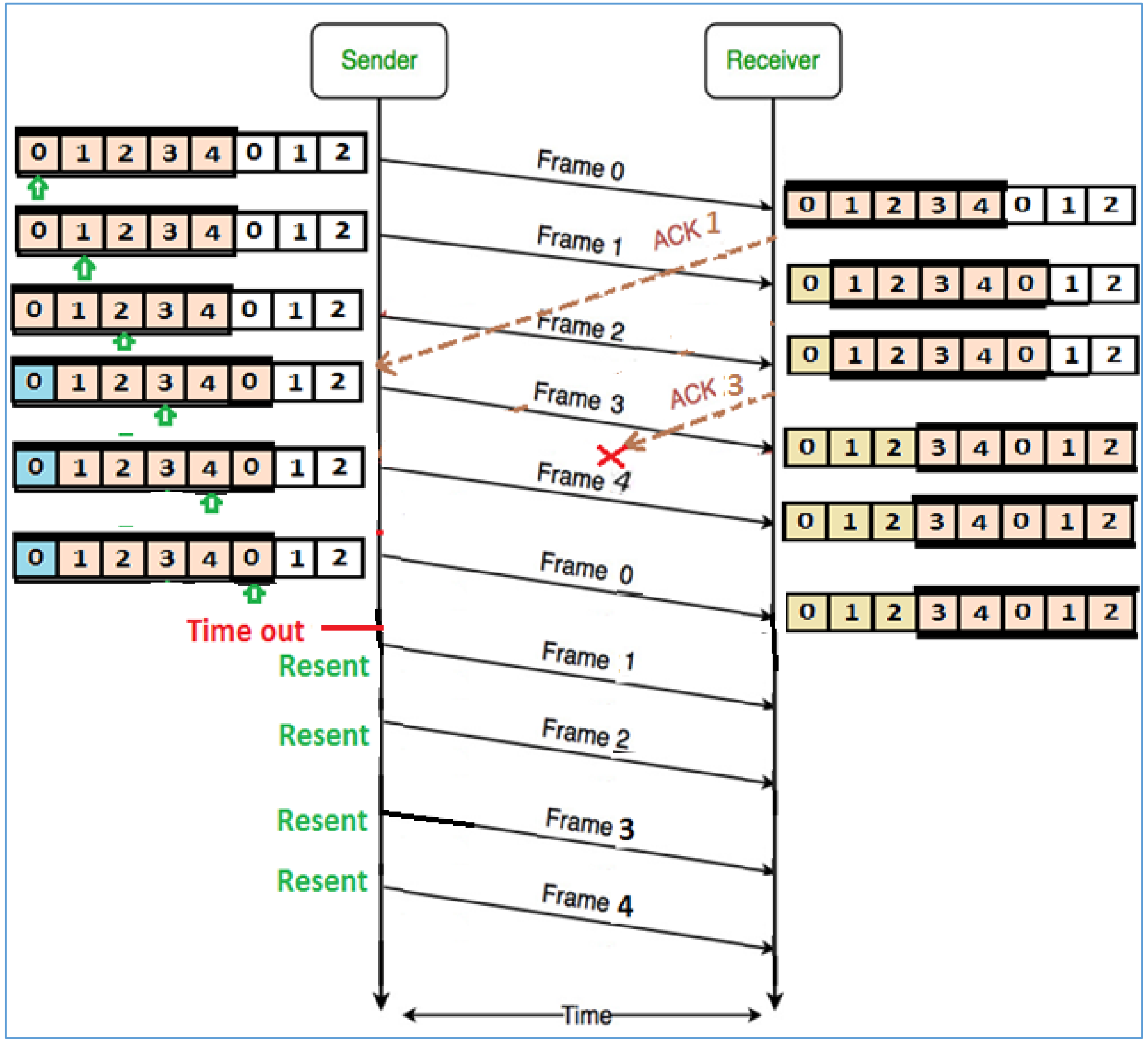
Go-back-n ARQ | Lost Acknowledgement

The sending device can send **as many frames as the window allows** before waiting for an ACK.

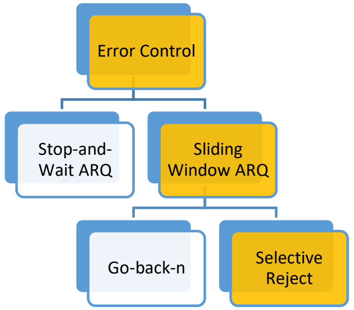
Once that limit has been reached or the sender has no more frames to send, **it must wait**.

The sender is equipped with a timer that begins counting whenever the **window capacity is reached**.

If an ACK has not been reached within the time limit, **the sender retransmits every frame transmitted since the last ACK.**



Error Control | Sliding Window ARQ



Selective-Reject ARQ

If a frame is corrupted in transit, **a NAK is returned** and **the frame is resent out of sequence**.

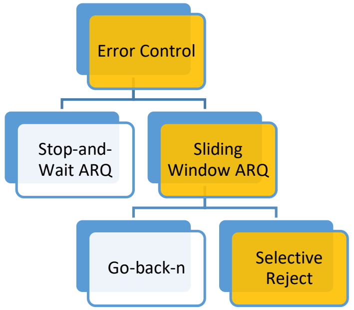
If one frame is lost, **only the specific damaged or lost frame is retransmitted**.

- The **receiving device** must be able to sort the frames it has and insert the retransmitted frame into its proper place in the sequence.
- The **sending device** must contain a searching mechanism that allows it to find and select only the requested frame for retransmission.
- A **buffer in the receiver** must keep all previously received out-of-sequence frames on hold.

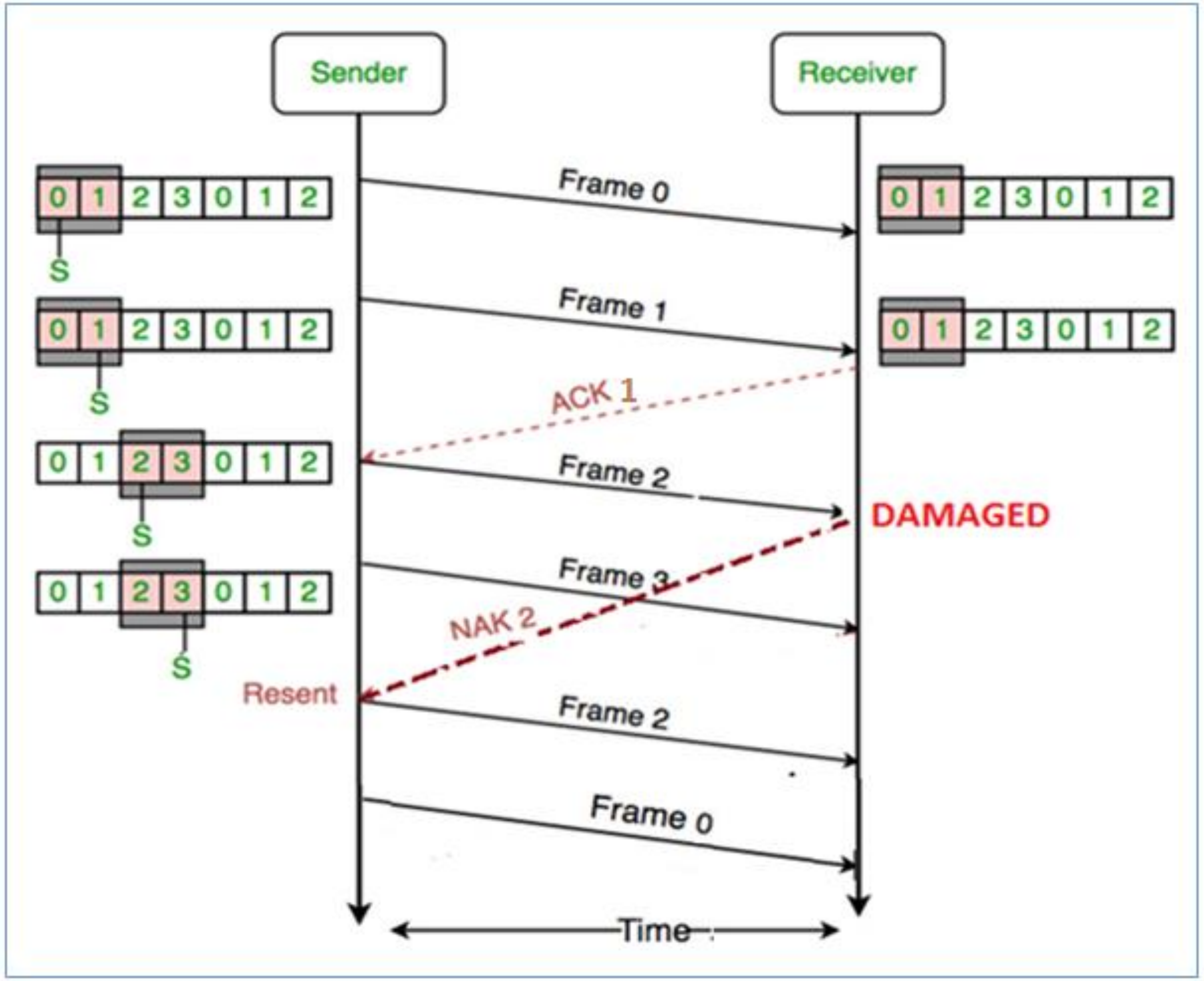
To aid selectivity, **ACK number must refer to the FRAME RECEIVED instead of the next frame expected**.

Due to above complexity in the process, **the recommended window size is less than or equal to (n+1)/2, where (n-1) is the Go-back-n window size**.

Error Control | Sliding Window ARQ



Selective-Reject ARQ | Damaged Frame

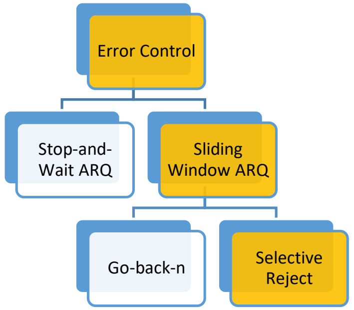


The damaged frame is resent out of sequence.

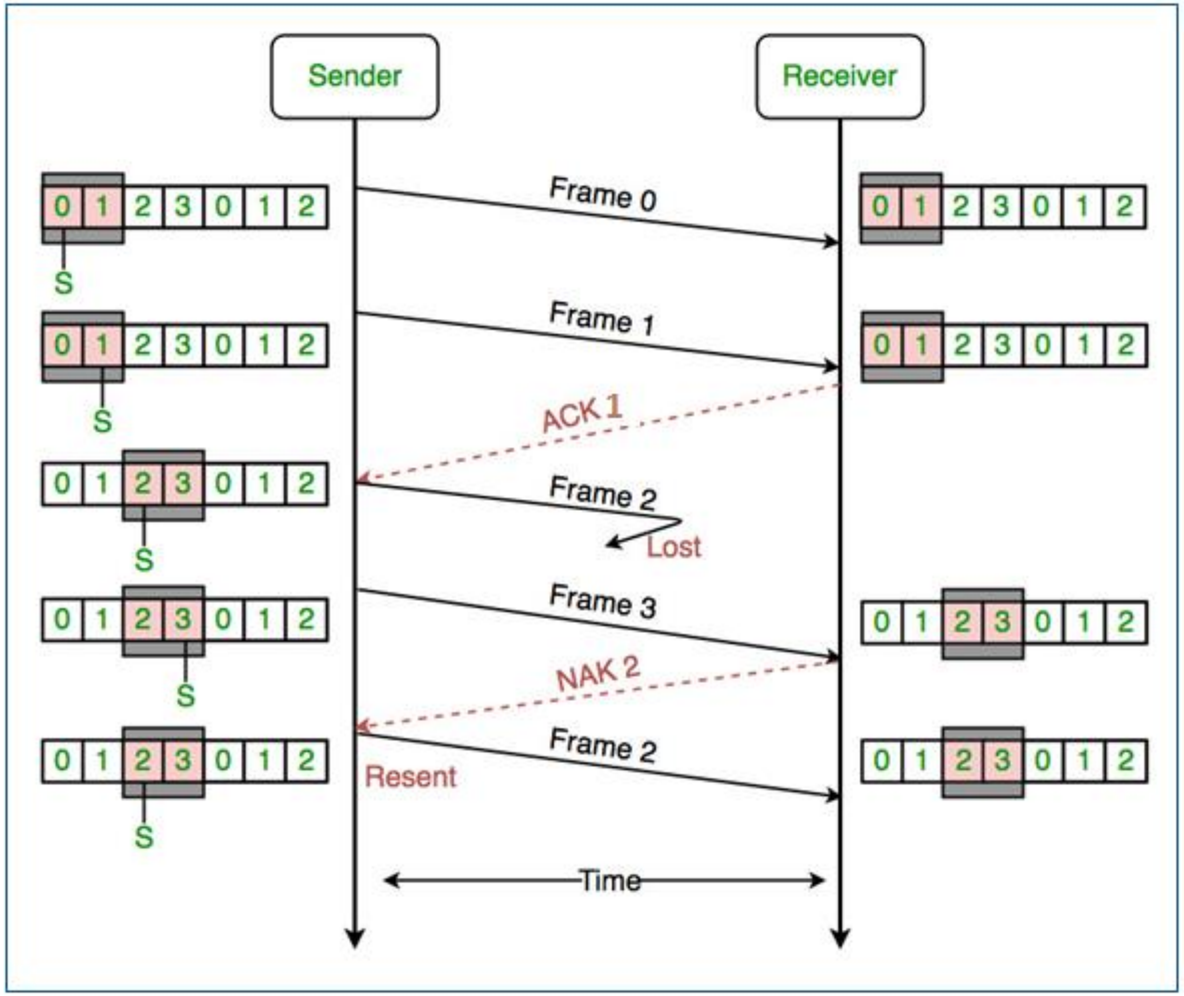
A NAK is returned.

Frames received after the damaged frame **cannot be acknowledged** until the damaged frames have been retransmitted.

Error Control | Sliding Window ARQ



Selective-Reject ARQ | Lost Data Frame



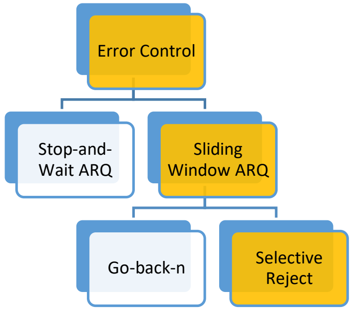
The lost frame is resent out of sequence.

If the lost frame was the LAST of the transmission, the receiver does nothing and the sender treats the silence like a lost of ACK.

Frames can be accepted out-of sequence BUT they cannot be **acknowledged out of sequence.**

A NAK is returned.

Error Control | Sliding Window ARQ



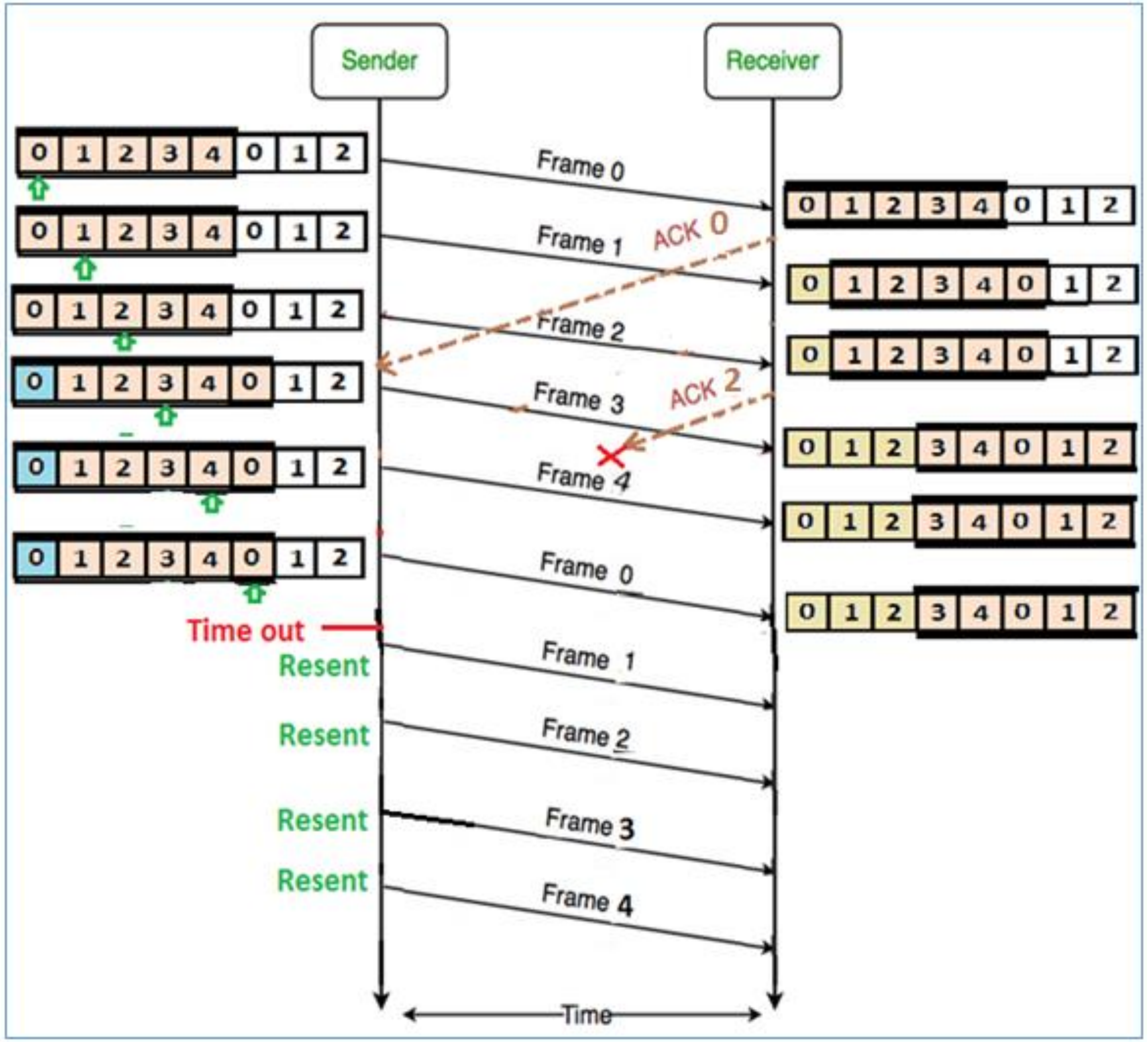
Selective-Reject ARQ | <u>Lost Acknowledgement</u>

Same as that of **<u>Go-back-n AR</u>**Q.

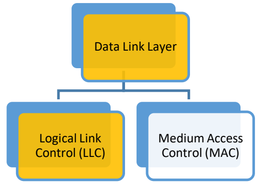
When the sending device reaches either the capacity of its window or the end of its transmission, **it sets a timer**.

If no ACK arrives in the time allotted, **the sender retransmits all of the frames that remain unacknowledged**.

In most cases, <u>the receiver will recognize any duplications and discards them</u>.



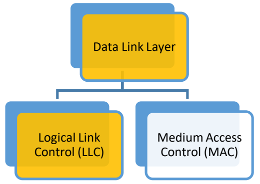
Data Link Layer | Logical Link Control (LLC)



- Framing
  - Character Count
  - Character Stuffing
  - Bit Stuffing
  - Physical Layer Coding Violation
- Flow Control
  - Stop-and-Wait
  - Sliding Window
- Error Control
  - Stop-and-Wait ARQ
  - Go-back-n ARQ
  - Selective-reject ARQ

- **Error Detection and Correction**
  - **Types of Errors**
  - **Detection**
  - **Correction**
- Protocol
  - High-Level Data Link Control (HDLC)

Logical Link Control (LLC) | Error Detection and Correction



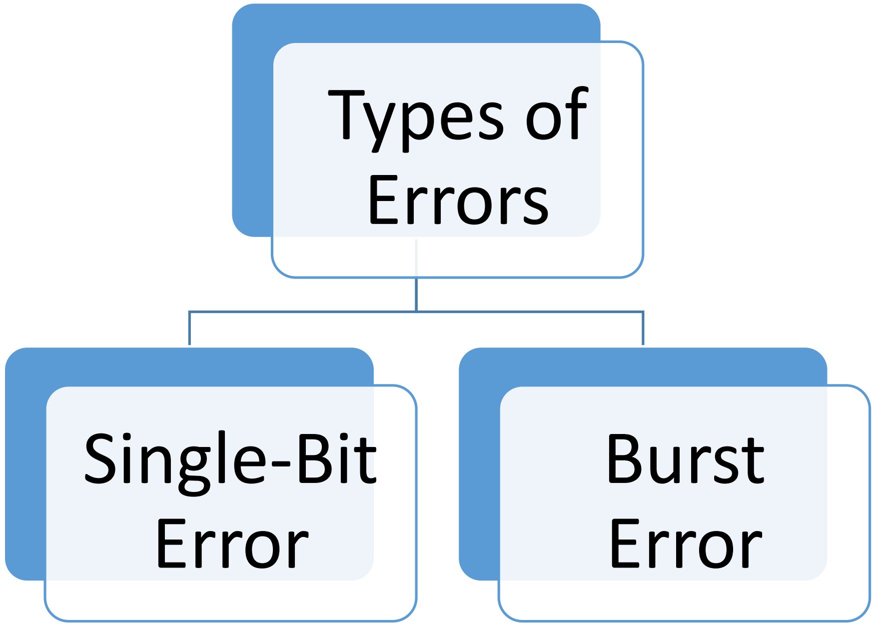
Data can be **corrupted** during transmission.

Networks must be able to transfer data from one device to another **with complete accuracy**.

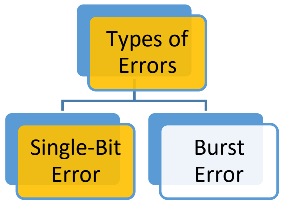
Reliable systems must have a mechanism for DETECTING AND CORRECTING ERRORS.

Whenever an **electromagnetic signal flows** from one point to another, it is subject to **unpredictable interference** from *heat, magnetism, and other forms of electricity*. This interference can change the shape or timing of the signal.

Error detection and correction are implemented either at the data link layer or the transport layer of the OSI reference model.

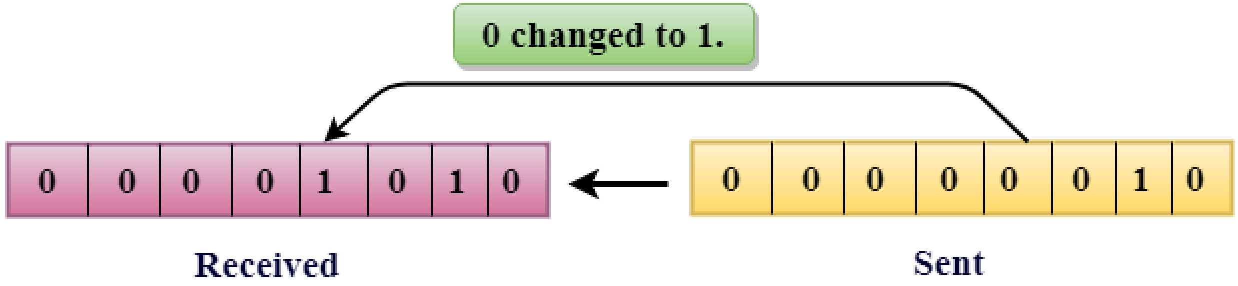


Error Detection and Correction | Types of Errors



Single-Bit Error

Only one bit of a given data unit (such as a byte, character, data unit, or packet) is changed from 1 to 0 or from 0 to 1.



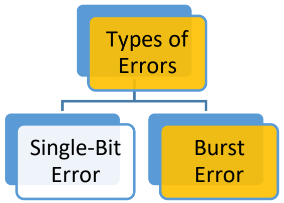
The least likely type of error in serial data transmission.

Sender sends data at **1 Mbps**.
Each bit **lasts only 1/1,000,000 second or 1μsec**.
The noise must have a duration of only 1μsec, **which is very rare**.

Happens if we are sending data using parallel transmission.

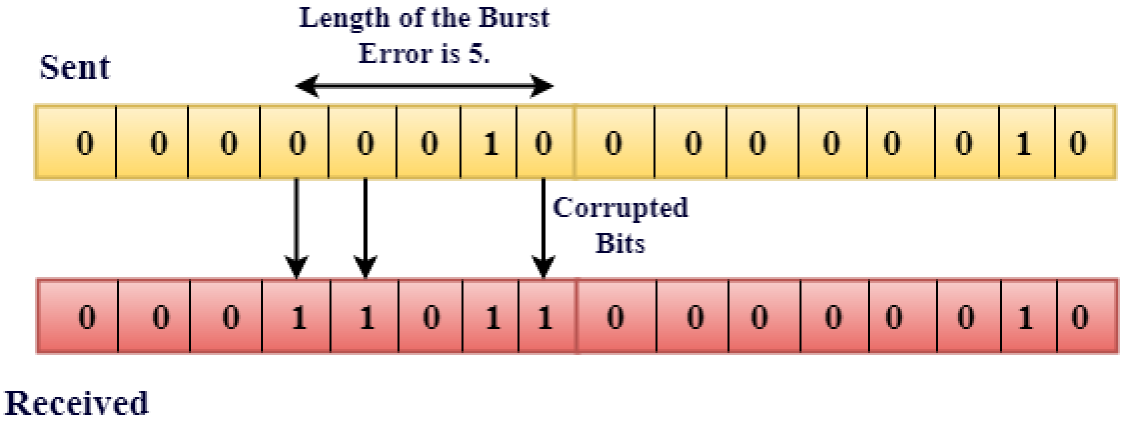
If one of the wires is noisy, one bit of each byte will be corrupted.

Error Detection and Correction | Types of Errors



Burst Error

Two or more bits in the data unit have changed from 1 to 0 or from 0 to 1.



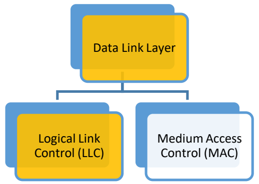
Burst error does not necessarily mean that the errors occurs in consecutive bits.

The length of the bursts is measured from the first corrupted bit to the last corrupted bit.

Burst error is most likely to happen in a serial transmission.

The duration of noise is **normally longer than the duration of a bit**.
When noise affects data, it affects a set of bits.
Number of bits affected depends on the **data rate and duration of noise**.

Logical Link Control (LLC) | Error Detection



The central concept in detecting or correcting errors is **REDUNDANCY**.

To be able to detect or correct errors, we need to send come extra bits with our data.

These redundant bits are added by the sender and removed by the receiver. **Their presence allows the receiver to detect or correct corrupted bits**.

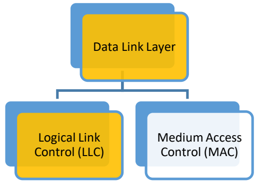
Send every data unit twice:

- Receiving device would then be able to do **a bit-for-bit comparison**.
- Any discrepancy would indicate an error and appropriate correction mechanism could be set in place.
- **Completely accurate system** but its would also be **insupportably SLOW.**
- **Transmission time will be Double** and also time to compare every unit bit by bit will **slow the system**.

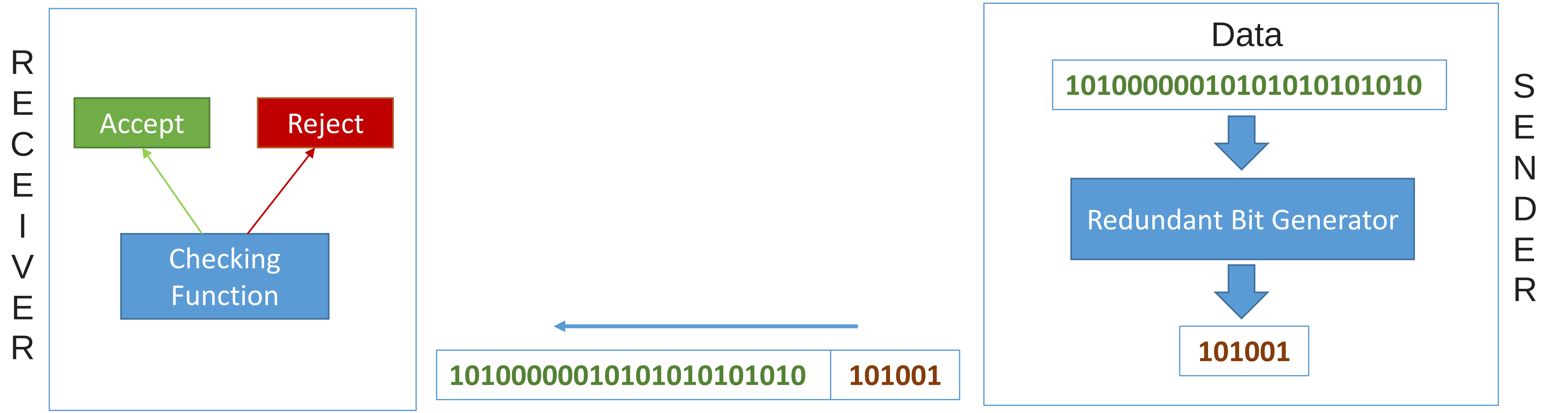
Including extra information is needed but need to add shorter group of bits.

Extra bits are **redundant to the information** and they are discarded as soon as the accuracy of the transmission has been determined.

Logical Link Control (LLC) | Error Detection



Redundancy Concept



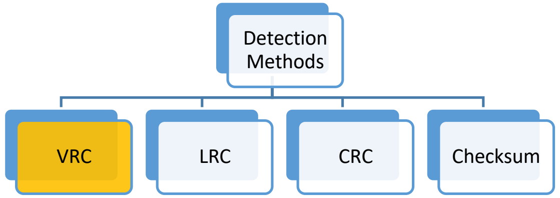
Types of redundancy checks used in data communication.

- Detection Methods
  - VRC
  - LRC
  - CRC
  - Checksum

Block Coding

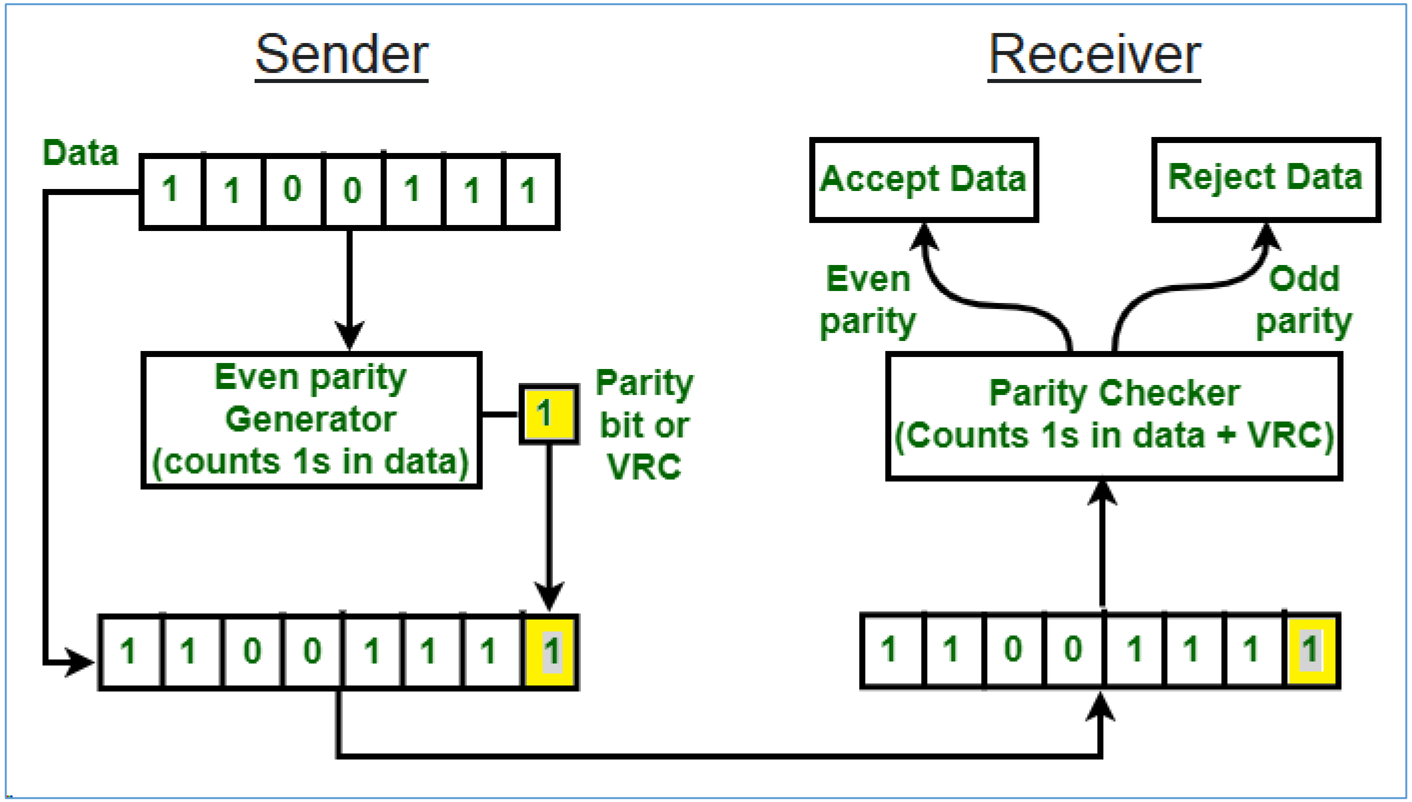
VRC: Vertical Redundancy Check
LRC: Longitudinal Redundancy Check
CRC: Cyclic Redundancy Check

Error Detection | Vertical Redundancy Check (VRC)



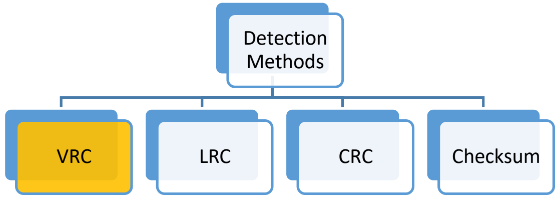
Other name is **Parity Check**.

A redundant bit (parity bit) is appended to every data unit so that the total number of 1s in the unit (including the parity bit) becomes EVEN (if even parity is being followed otherwise ODD).



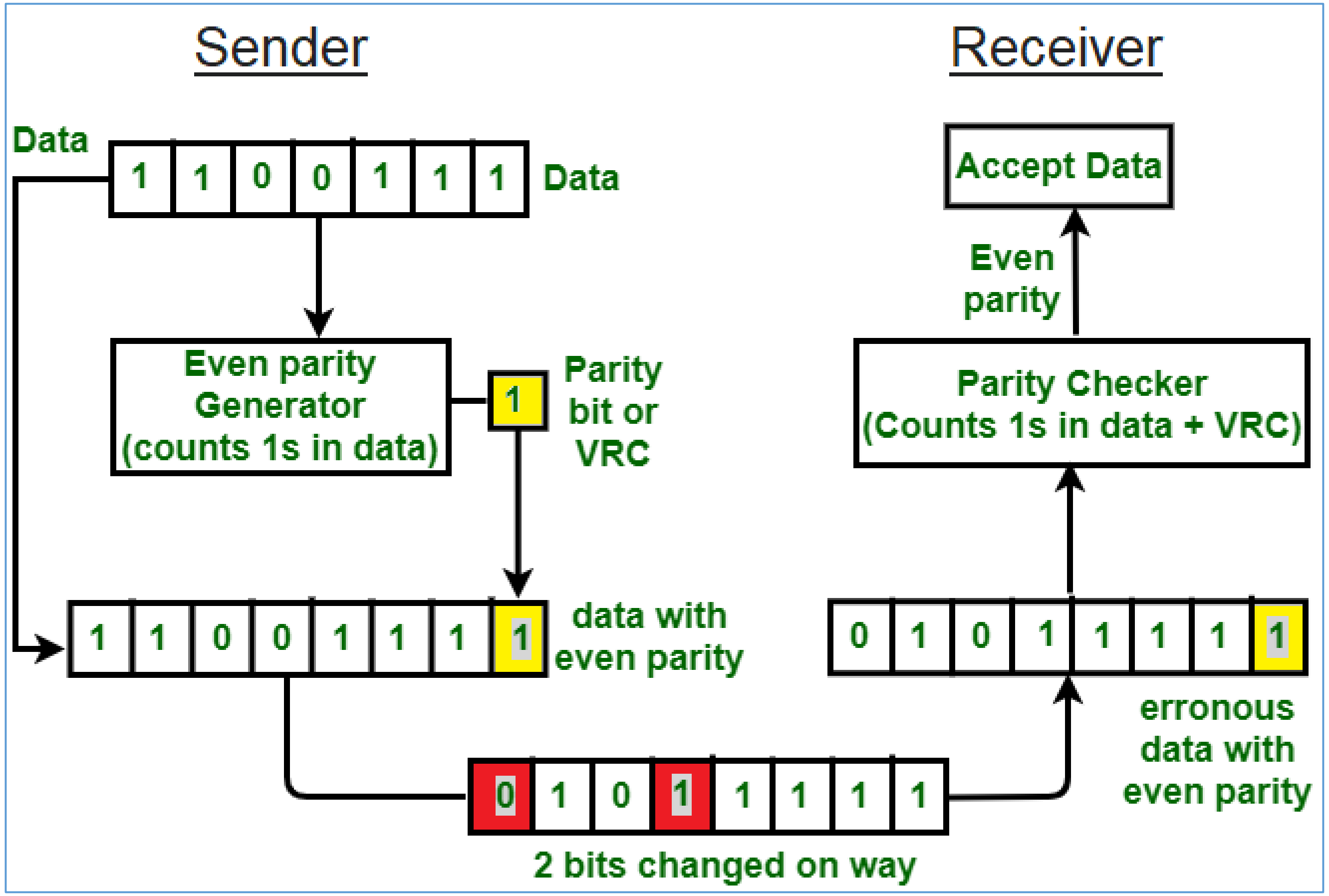
Most common and **least expensive mechanism** for error detection.

Error Detection | Vertical Redundancy Check (VRC)



What if the data unit has been changed in transit?

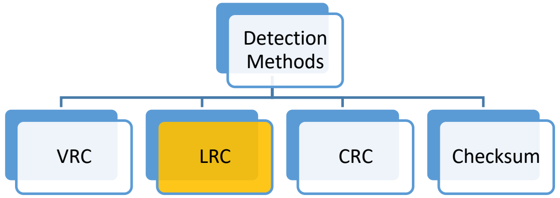
The receiver checks the parity at its end and it will know that an error has been introduced into data somewhere and therefore rejects the whole unit.



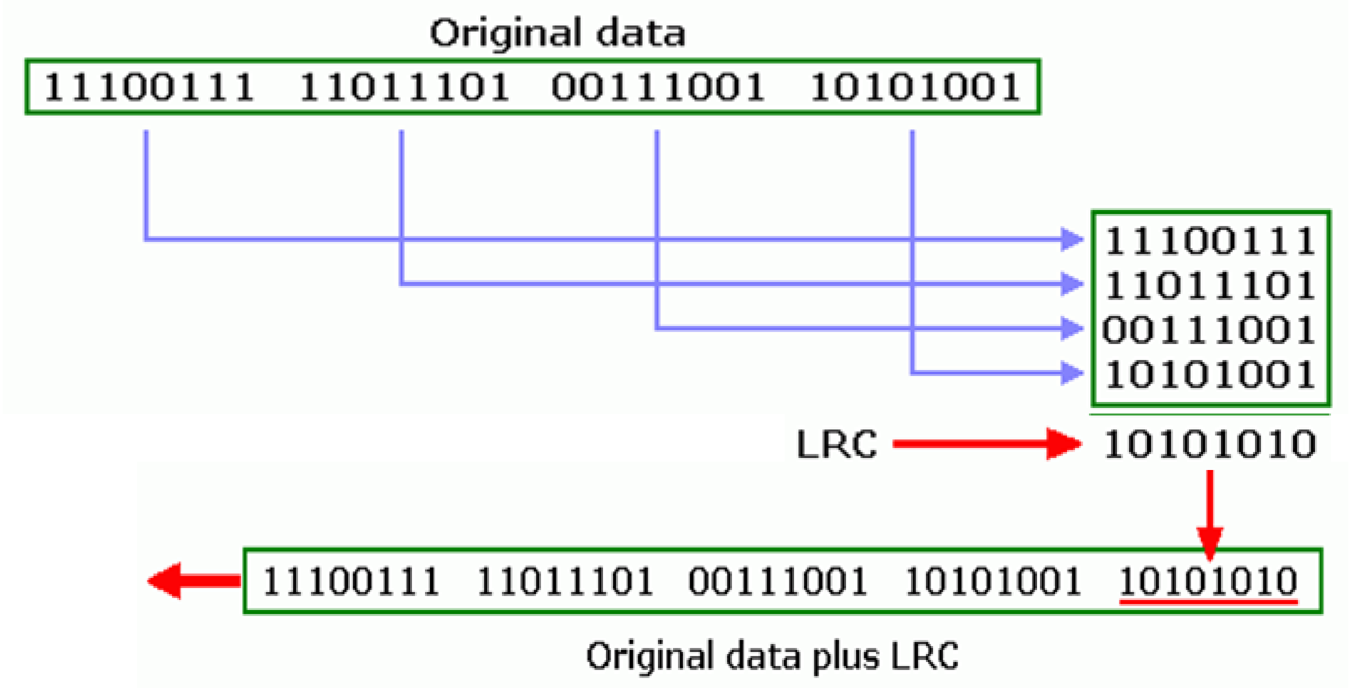
VRC can detect **all single-bit errors**.

It can **also detect burst errors** as long as the total number of bits changed is ODD.

Error Detection | Longitudinal Redundancy Check (LRC)

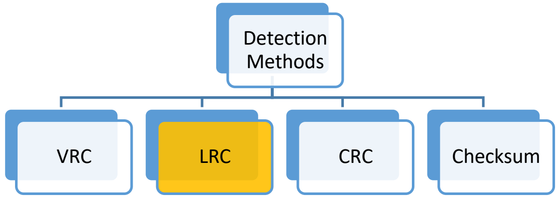


1. A block of bits is **organized in a table** (rows and columns).
2. **Calculate the parity bit for each column** and create a new row of same number of bits, which are the **parity bits for the whole block**.
3. Attach the parity bits block (redundant) to the original data and *send them to the receiver*.



LRC increases the likelihood of detecting burst errors.

Error Detection | Longitudinal Redundancy Check (LRC)



It ***two bits in one data unit are damaged*** and **two bits in exactly the same positions** in another data unit **are also damaged**, the **LRC checker will not detect an error**.

10101001	00111001	11011101	11100111

Original Data

1	0	1	0	1	0	0	1
0	0	1	1	1	0	0	1
1	1	0	1	1	1	0	1
1	1	1	0	0	1	1	1

1	0	1	0	1	0	1	0

LRC

Data sent to the Receiver **LRC**

10101001	00111001	11011101	11100111	10101010

Error introduced on Transit

10100011	00110011	11011101	11100111	10101010

LRC Calculation at the Receiver

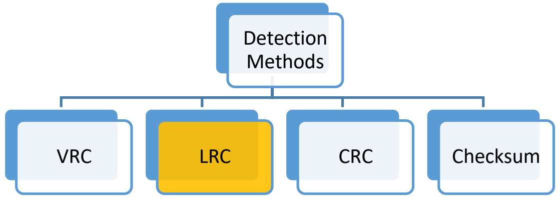
1	0	1	0	0	0	1	1
0	0	1	1	0	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	0	1	1	1

1	0	1	0	1	0	1	0

1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0

Comparison **MATCHED** **NO ERROR!**

Error Detection | VRC & LRC



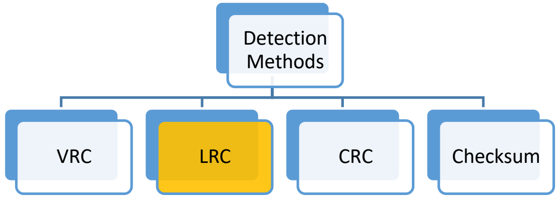
Two-dimensional Parity Code

	Data Blocks							VRC
	1	0	0	0	0	0	1	0
	0	1	0	0	0	0	1	0
	1	1	0	0	0	0	1	1
	0	0	1	0	0	0	1	0
	1	0	1	0	0	0	1	1
	0	1	1	0	0	0	1	1
	1	1	1	0	0	0	1	0
LRC	0	0	0	0	0	0	1	1

10000010	01000010	11000011	00100010	10100011	01100011	11100010	00000011

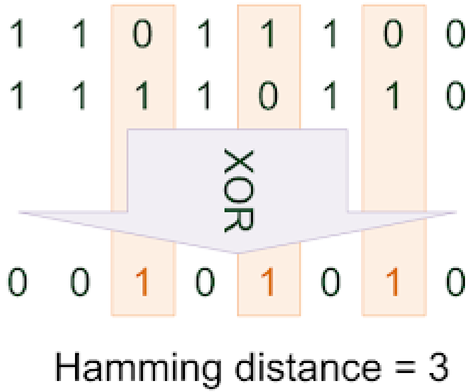
Data to be Sent

Error Detection | VRC & LRC



Hamming Distance

The hamming distance between two words is the <u>number of differences between corresponding bits</u>.



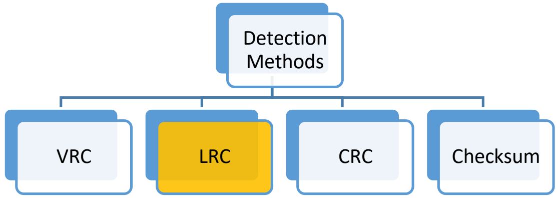
The **Minimum Hamming Distance** is the smallest Hamming distance between <u>all possible pairs of codewords</u>.

0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**

1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

Minimum Hamming Distance = 2

Error Detection | VRC & LRC



All possible 4 bits Datawords

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

All valid Codewords

0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**

1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

Even parity

Minimum Hamming distance of any pair of the possible valid codewords = 2

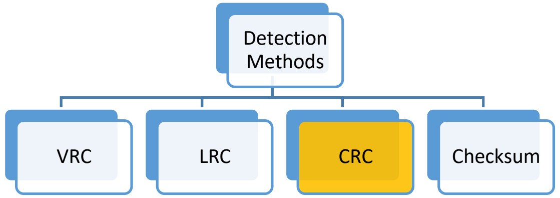
The system can detect an error of = 1 bit only

1	1	0	0	0	➡	0	1	0	0	0

1	1	0	0	0	➡	0	1	1	0	0

To guarantee the detection of up to 's' errors, the Minimum Hamming Distance in a block code must be (s+1).

Error Detection | Cyclic Redundancy Check (CRC)



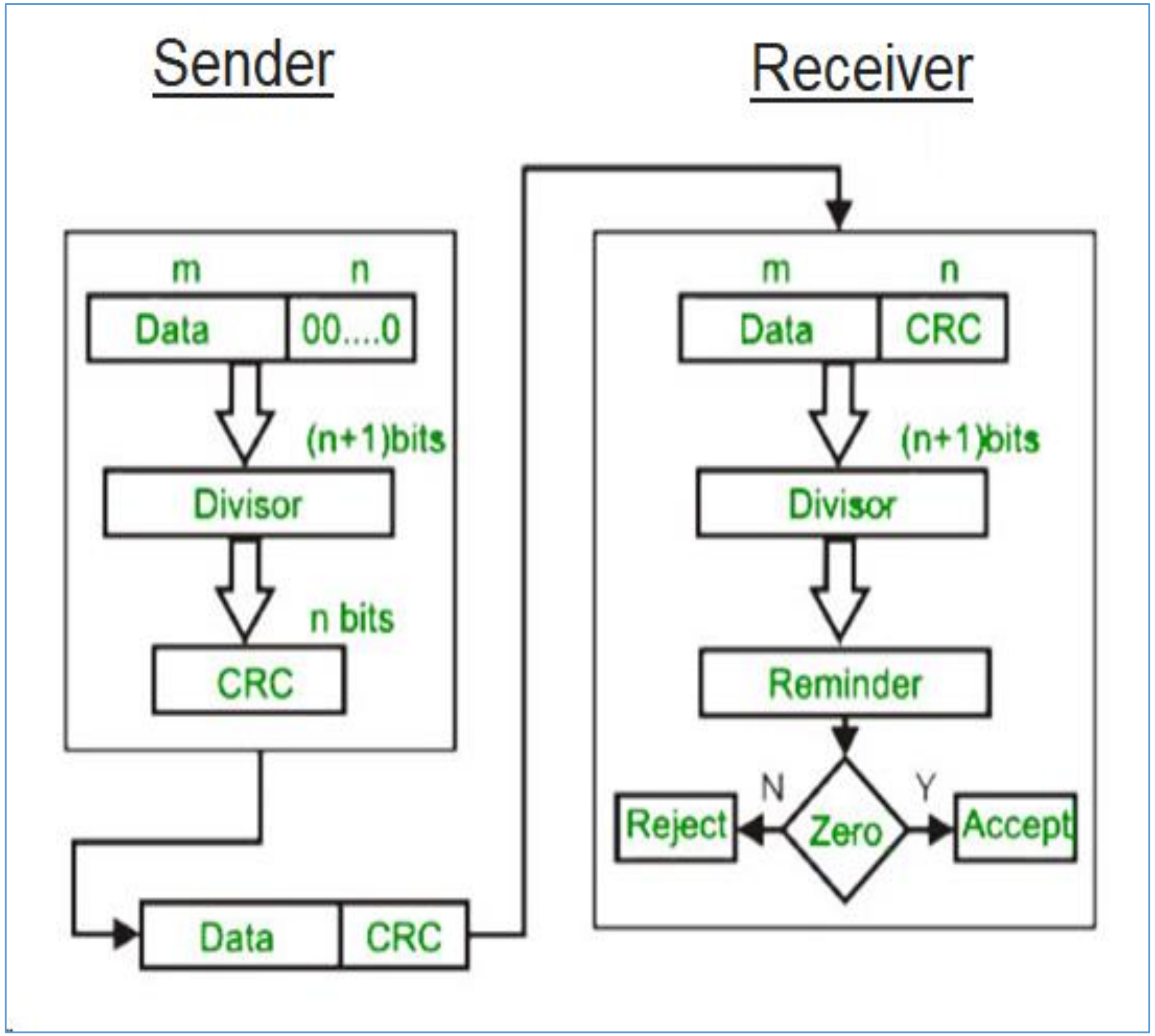
Most powerful of the redundancy check technique.

VRC and LRC are based on addition.

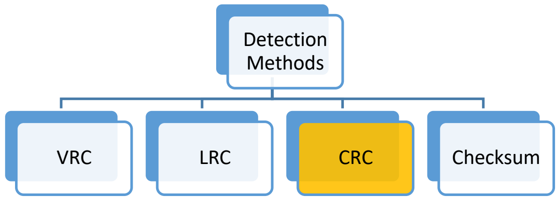
CRC is based on division.

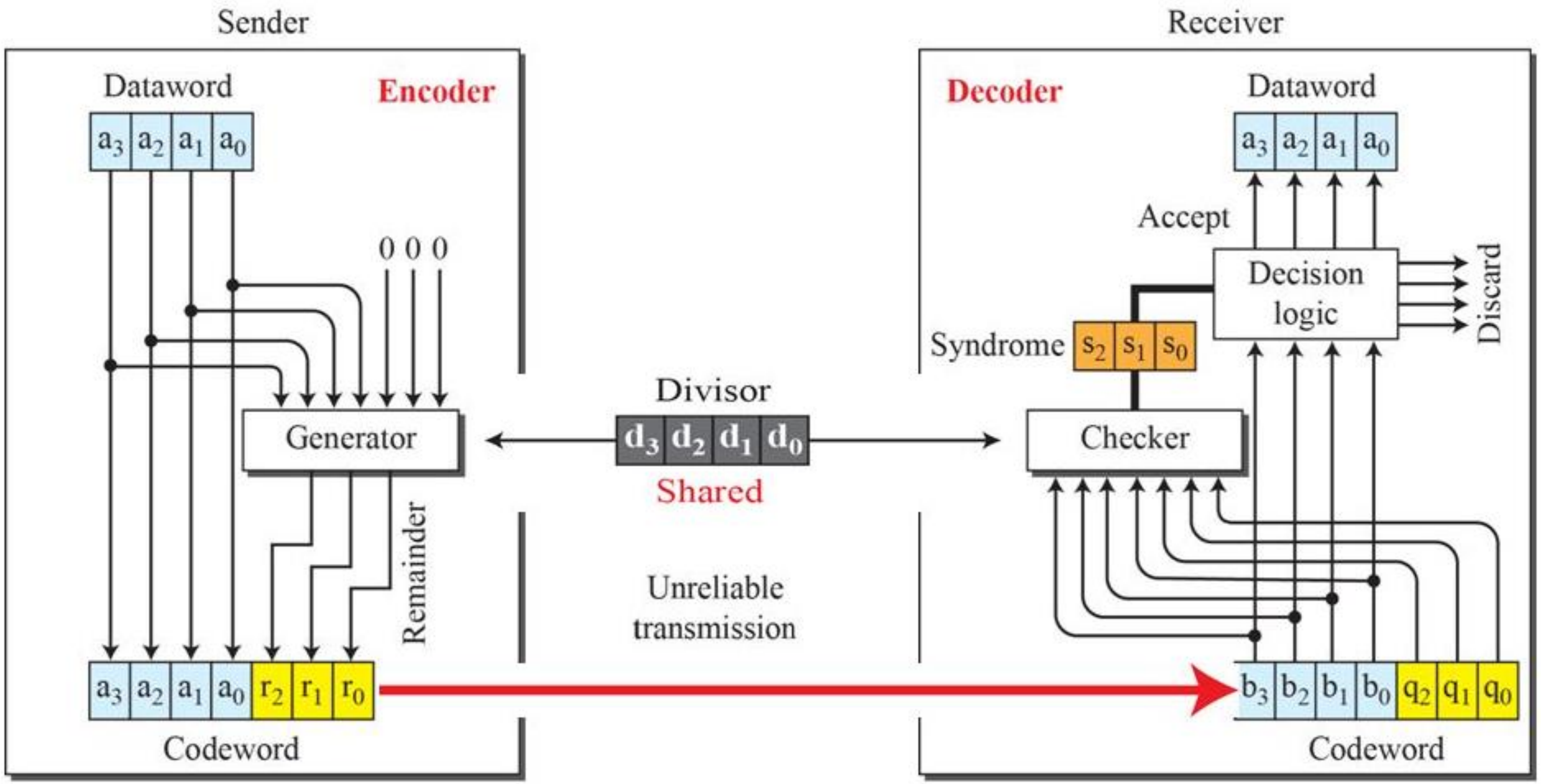
A sequence of redundant bits (called CRC remainder) is appended to the end of a data unit so that the **resulting data unit** becomes exactly divisible by a second, predetermined binary number.

At the receiver end, the incoming data unit is divided by the same number. If at this step there is no remainder, **the data unit assumed to be intact and is therefore accepted**. The remainder indicates that **the data unit has been damaged in transit and therefore must be rejected**.

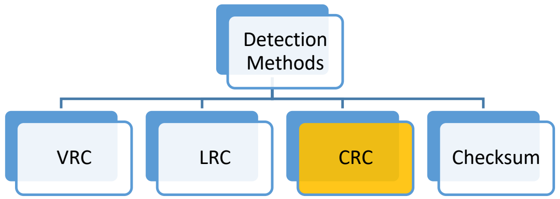


Error Detection | Cyclic Redundancy Check (CRC)





Error Detection | Cyclic Redundancy Check (CRC)



Original data unit is (**100100**).

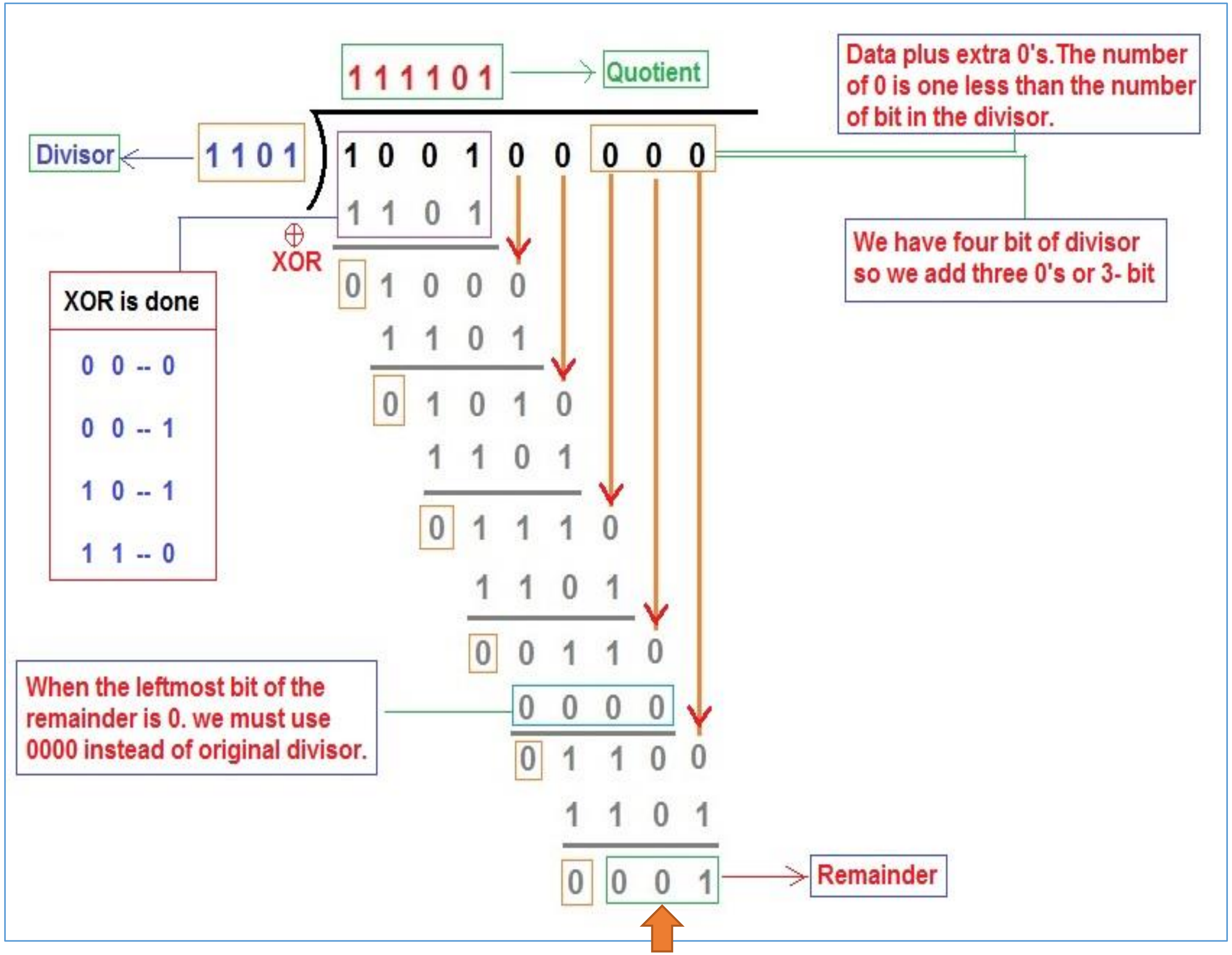
A string of ‘n’ 0s is *appended to the original data unit*.

The number ‘n’ is one less than the number of bits in the **pre-determined divisor (1101)**, which is **n+1 bits**.

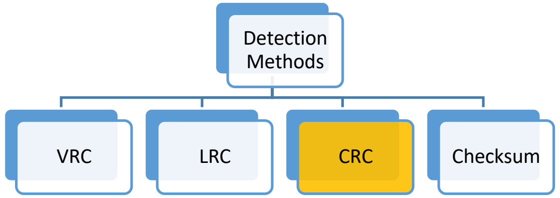
A CRC generator uses **modulo-2 division.**

Binary division of **elongated data unit** (**100100**000) and **divisor (1101)**. The remainder resulting from the division is **CRC (001)**.

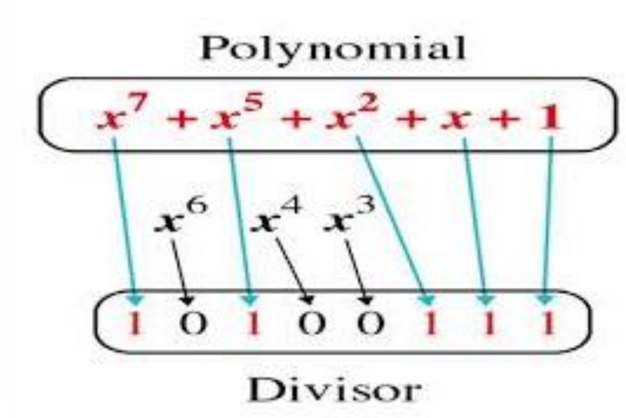
100100	001



Error Detection | Cyclic Redundancy Check (CRC)



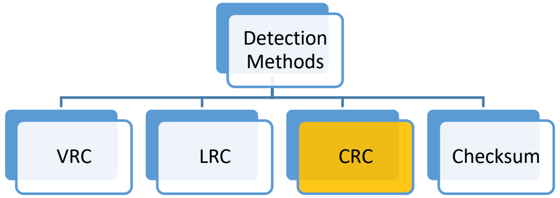
The **CRC divisor** is most often **represented as an** **POLYNOMIAL**.



The **Polynomial** should be selected to have at least the following properties:

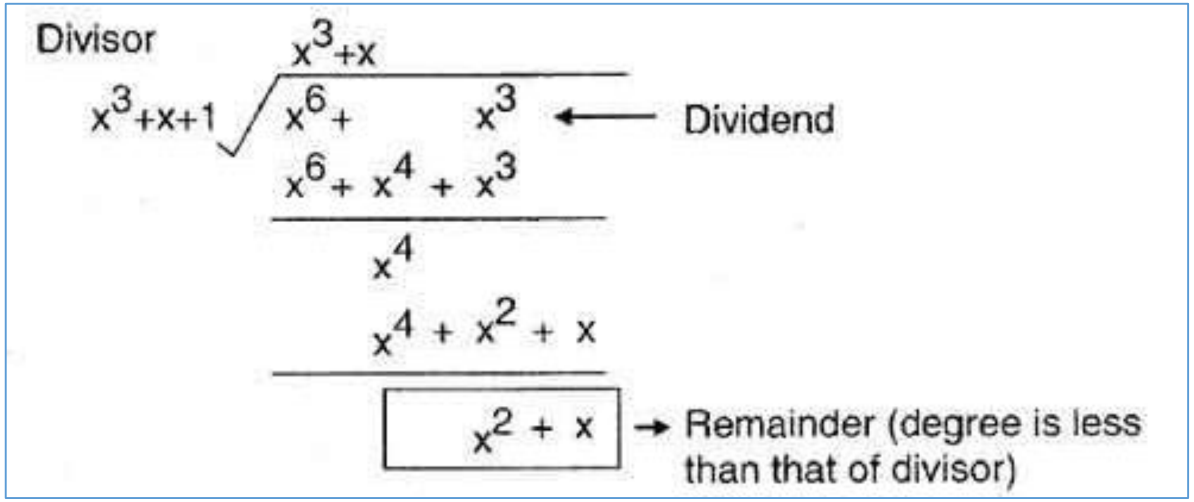
- It should **not be divisible by x.**
  - Guarantees that all burst errors of a length equal to the degree of the polynomial are detected.
- It should be **divisible by (x+1).**
  - Guarantees that all burst errors affecting an odd number of bits are detected.

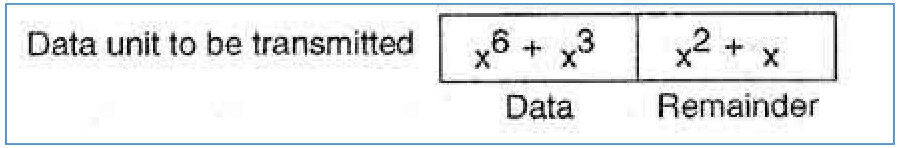
Error Detection | Cyclic Redundancy Check (CRC)



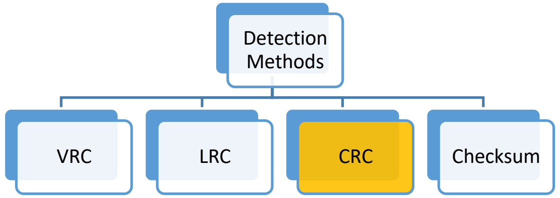
CRC Division using Polynomial:

Data	1001	$x^3 + 1$
Division	1011	$x^3 + x + 1$





Error Detection | Cyclic Redundancy Check (CRC)



CRC Analysis

Codeword Transmitted: $\boldsymbol{C(x)}$

Imagine a transmission error $(\boldsymbol{E(x)})$ occurs.

Codeword Received: $\boldsymbol{C(x) + E(x)}$

Each 1 bit in $E(x)$ corresponds to a bit that has been inverted.

If there are 'k' 1 bits in $E(x)$, 'k' single-bit errors have occurred.

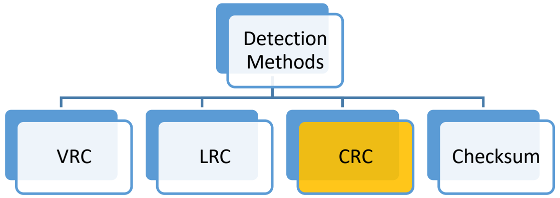
Those errors that happens to correspond to polynomials containing **$\boldsymbol{G(x)}$ as a factor** will slip by; all other errors will be caught.

On receipt of the data, it does the same modulo-2 division:

$$\frac{\boldsymbol{C(x) + E(x)}}{\boldsymbol{G(X)}} = \frac{\boldsymbol{C(x)}}{\boldsymbol{G(x)}} + \frac{\boldsymbol{E(x)}}{\boldsymbol{G(x)}}$$

If the remainder is all 0's, the CRC is dropped and the **data accepted**. Otherwise, the received stream of bits is discarded and data are resent.

Error Detection | Cyclic Redundancy Check (CRC)



Find the criteria that must be imposed on the generator ($G(x)$) **to detect the type of error to be detected**.

CASE – I: Single Bit Error

Data: 1001

Divisor: 1011

Divisor($G(x)$): $x^3 + x + 1$

Codeword Tx^ed: 1001110

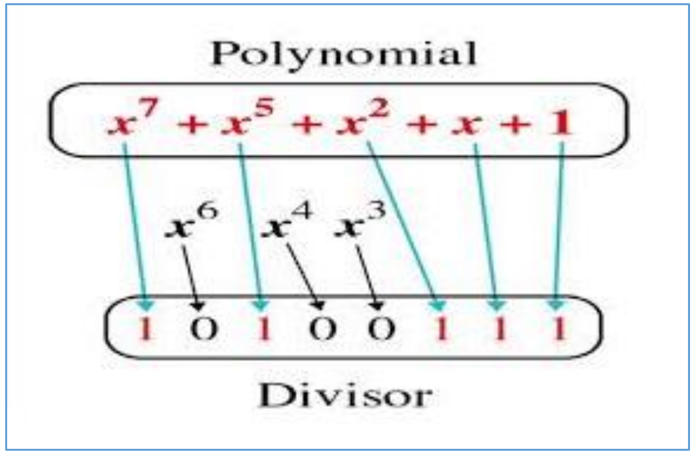
Codeword Rx^ed: 1011110

Error at x^4 bit.

Structure of $G(x)$ to guarantee the detection of a single bit error:

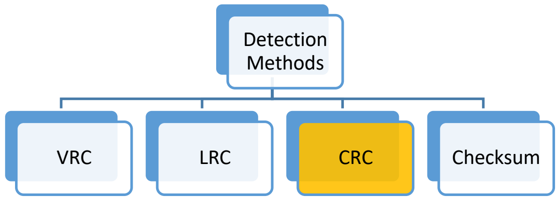
- If a single bit error (say $E(x) = x^i$) is caught, then $E(x)$ is not divisible by $G(x)$.

- If $G(x)$ has at least **two terms and the co-efficient of x^0 is not zero**, then $E(x)$ cannot be divided by $G(x)$.



Check for: $G(x)$ = (x+1) $G(x)$ = x^3

Error Detection | Cyclic Redundancy Check (CRC)



CASE – II: Two Isolated Single Bit Errors

Date: 1001

Divisor: 1011

Divisor($G(x)$): $x^3 + x + 1$

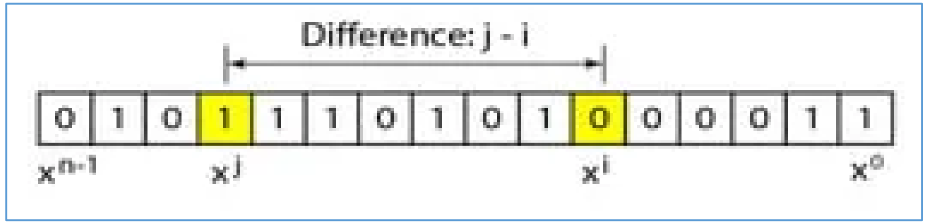
Codeword Tx[ed]: 1001110

Codeword Rx[ed] : 1101010

Error at $x^5 + x^2$ bit.

Under what conditions can this type of error be caught?

$$E(x) = x^j + x^i = x^i(x^{j-i} + 1)$$



- If $G(x)$ has **more than one term and one term is x^0**, it cannot divide x^i.
- If $G(x)$ is to divide $E(x)$, **it must divide $(x^{j-i} + 1)$**.

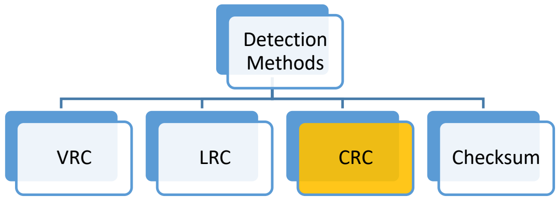
In other words, **$G(x)$ must not divide $(x^t + 1)$**, where ‘t’ is between 0 and n-1.

- ➢ t = 0 is meaningless.
- ➢ t = 1 is needed.

Means ‘t’ should be between 2 and n-1.

Check for: $G(x)$ = (x+1) | $G(x)$ = x^4+1 | $G(x)$ = x^7+x^6+1

Error Detection | Cyclic Redundancy Check (CRC)



CASE – III: Odd Numbers of Errors

A generator that **contains a factor of (x+1)** can **detect all odd-numbered errors**.

Example: $(x^4 + x^2 + x + 1)$ can catch all odd-numbered errors since it can be written as **a product of the two polynomials** $(x + 1)$ **and** $(x^3 + x^2 + 1)$.

Generator should not be only (x+1) - It **cannot catch** the two adjacent isolated errors.

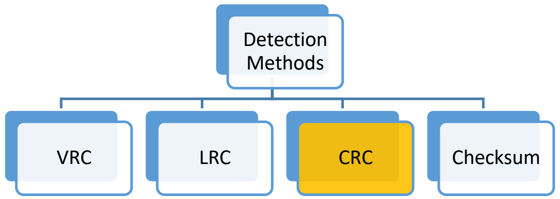
CASE – IV: Burst Errors

$$E(x) = (x^j + \cdots + x^i) = x^i * (x^{j-i} + \cdots + 1)$$

If the generator can detect a single error (minimum condition for a generator), then it cannot divide x^i.

$$\frac{x^{j-i} + \cdots + 1}{x^r + \cdots + 1}$$ **must not be zero.**

Error Detection | Cyclic Redundancy Check (CRC)



$\boxed{\dfrac{x^{j-i} + \cdots + 1}{x^r + \cdots + 1}}$ **must not be zero.**

If $(j - i) < r$:– **The remainder can never be zero.** $\boxed{\dfrac{x^3 + x^2 + 1}{x^4 + x^2 + 1}}$

All burst errors with length ≤ the number of check bits ‘r’ will be detected.

If $(j - i) = r$:– **In some rare cases, the syndrome is 0** and the error is undetected. $\boxed{\dfrac{x^3 + x^2 + 1}{x^3 + x^2 + 1}}$

The probability of undetected burst error of length (r+1) is: $\left({}^{1}/_{2}\right)^{r-1}$

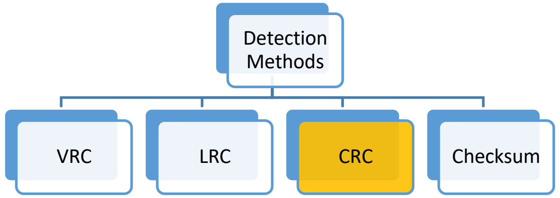
If $(j - i) > r$:– **The syndrome is 0** and the error is undetected. $\boxed{\dfrac{x^4 + x^2 + 1}{x^3 + x^2 + 1}}$

The probability of undetected burst error of length greater than (r+1) is : $\left({}^{1}/_{2}\right)^{r}$

Generator: $G(x) = (x^6 + 1)$

- Can detect all burst errors with a length **less than or equal to 6 bits**.
- **3 out of 100 burst errors** with length 7 will slip by.
- **16 out of 1000 burst errors** with length 8 or more will slip by.

Error Detection | Cyclic Redundancy Check (CRC)



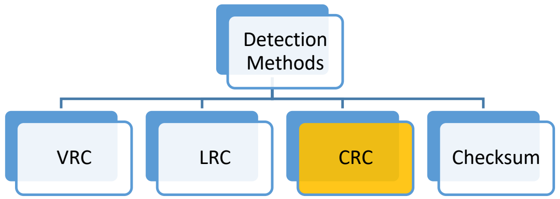
A good polynomial generator needs to have the following characteristics:

- It should have at least two terms.
- The co-efficient of the term x^0 should be 1.
- It should not divide $x^t + 1$, for t between 2 and $(n - 1)$.
- It should have the factor $(x + 1)$.

Standard Polynomial:

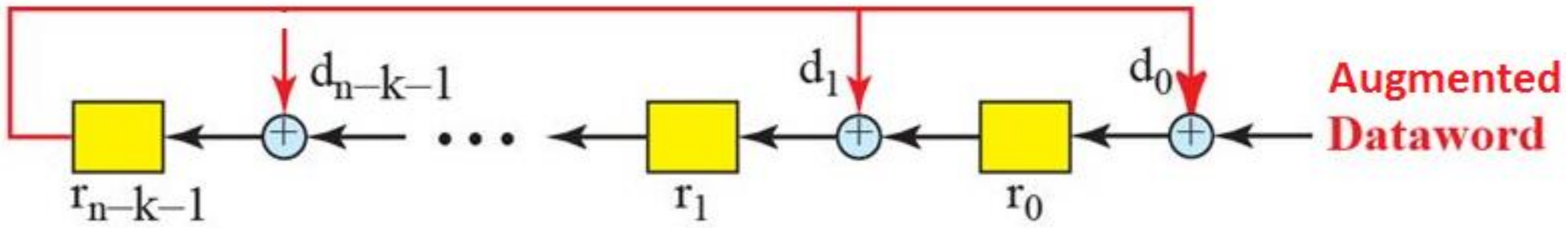
- CRC-8: $x^8 + x^2 + x + 1$
- CRC-10: $x^{10} + x^9 + x^5 + x^4 + x^2 + 1$
- CRC-16: $x^{16} + x^{12} + x^5 + 1$
- CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Error Detection | Cyclic Redundancy Check (CRC)

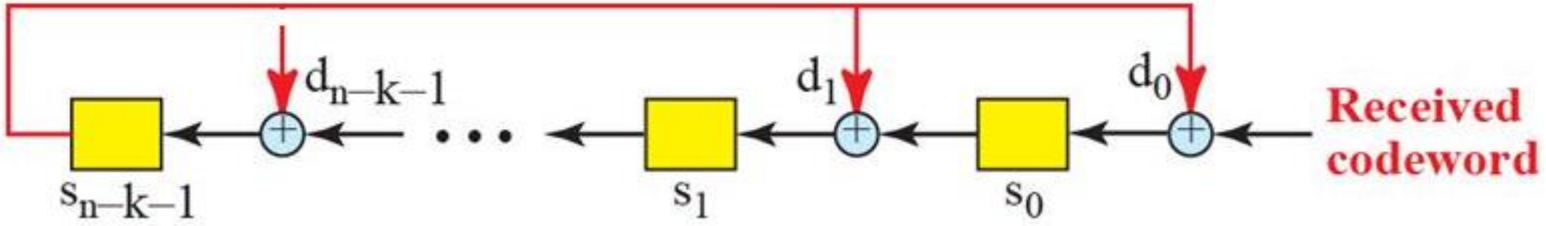


Hardware Implementation of CRC Encoder and Decoder

Encoder



Decoder



1-bit Shift Register (n-k)

XOR Device (n-k)

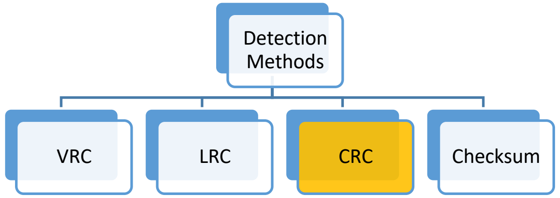
Dataword has ‘k’ bits.

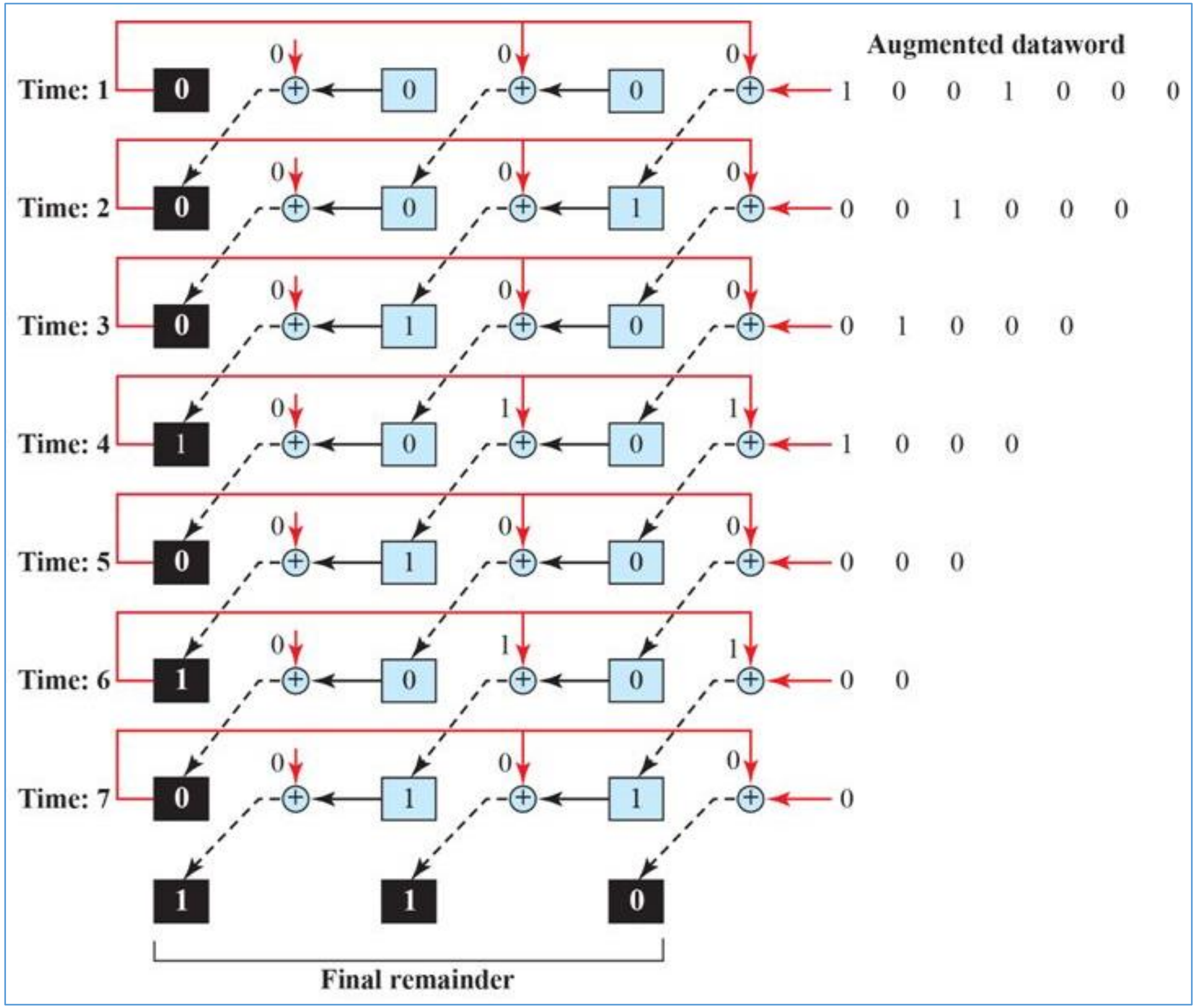
Codeword has ‘n’ bits.

CRC check has ‘n-k’ bits.

Divisor has ‘n-k+1’ bits.

Error Detection | Cyclic Redundancy Check (CRC)





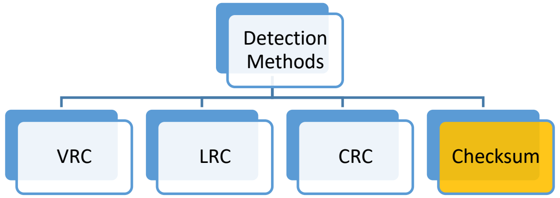
Assume that the **<u>remainder is originally all 0s</u>**.

<u>At each time click</u> (<u>arrival of 1 bit from an augmented Dataword</u>), repeat the following *two actions*:

1. Use the **<u>leftmost bit of the remainder</u>** to **make a decision about the divisor** (**0<u>11</u>** or **<u>000</u>**).

2. The **<u>other 2 bits of the remainder</u>** and the **<u>next bit from the augmented dataword</u>** (**total of 3 bits**) are **XORed with the 3-bit divisor** **<u>to create the next remainder</u>**.

Error Detection | Checksum



Error-detecting technique that can be applied to a message of any length.

Mostly used in **Network and Transport layer** rather than the data-link layer.

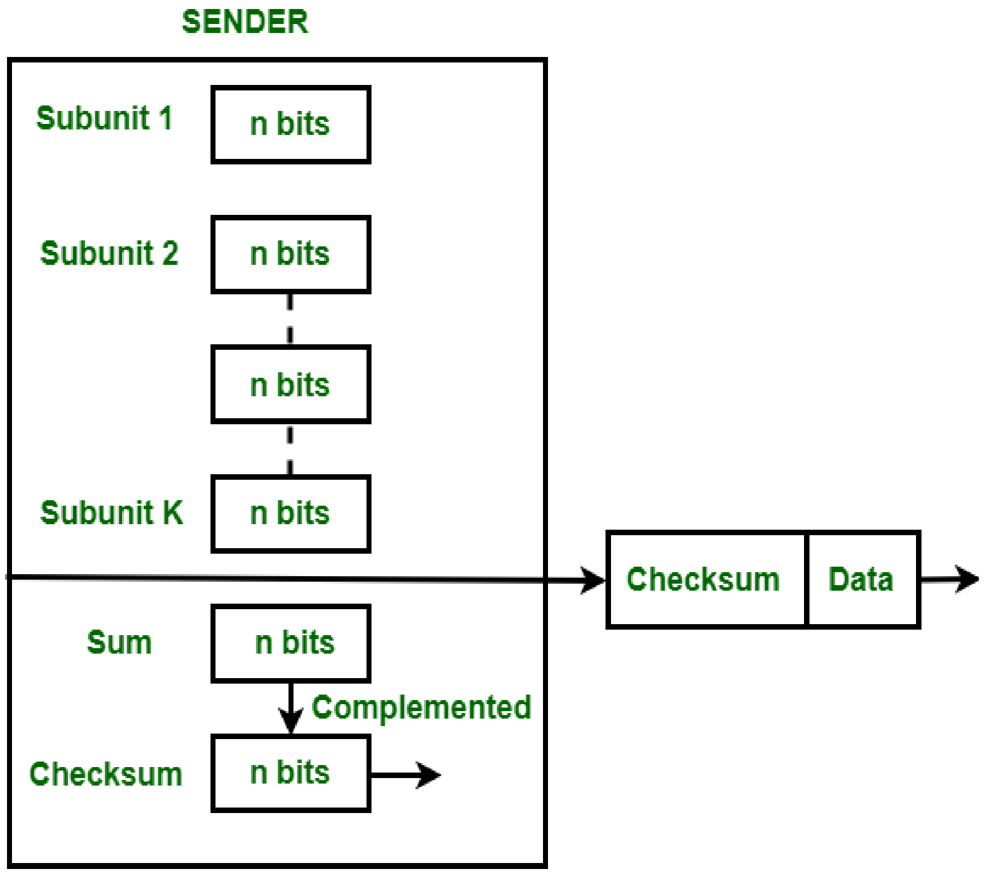
Checksum Generator: At Sender Side

Sub-divides the data unit into **equal segments of 'n' bits**.

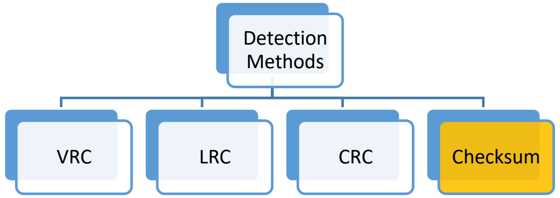
All segments are added together using **1's complement arithmetic** in such a way that the **total is also 'n' bits long**.

Total sum is then complemented and appended to the end of the original data as redundant bits, called the **checksum field**.

The extended data unit is transmitted across the network.



Error Detection | Checksum



Checksum Checker: At Receiver Side

Sub-divides the received data unit into **equal segments of ‘n’ bits**.

All segments are added together using **1’s complement arithmetic** in such a way that the **total is also ‘n’ bits long**.

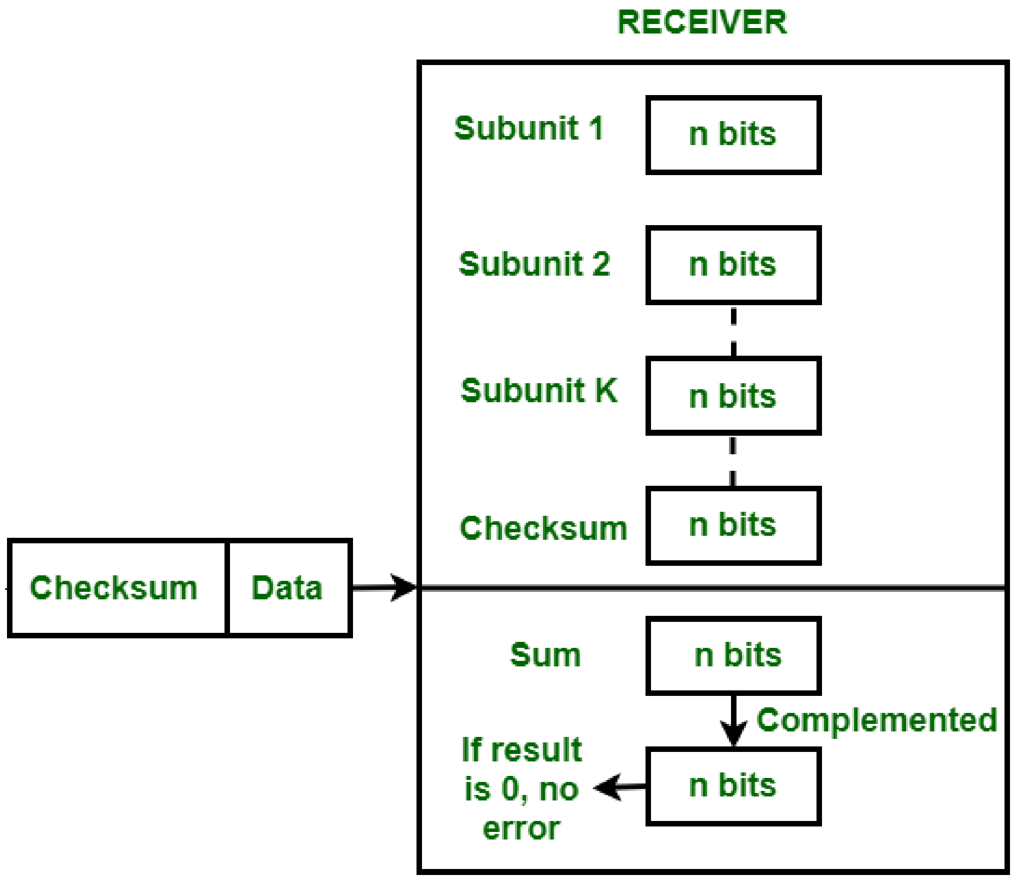
Total sum is then complemented.

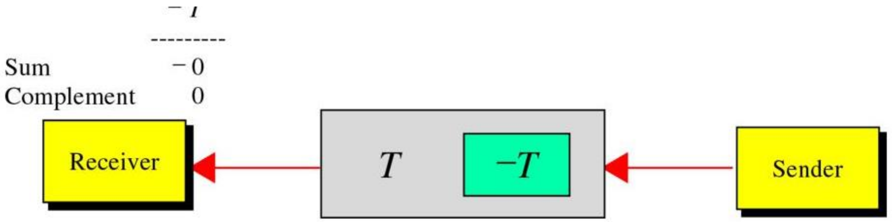
No error -> Checksum field should be **ZERO**.

The receiver accepts the packet.

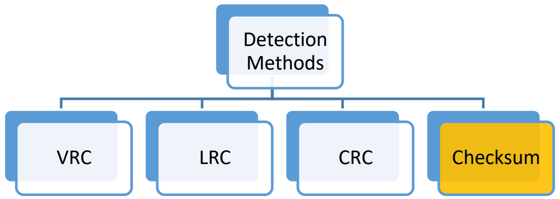
Error -> Checksum field should be **NON-ZERO**.

The receiver rejects the packet.





Error Detection | Checksum



Checksum Generator

Original Data

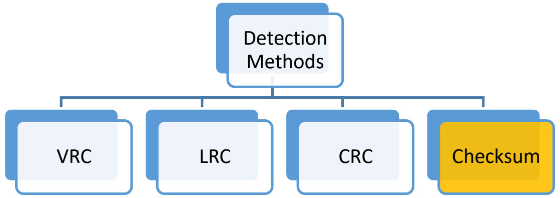
10011001	11100010	00100100	10000100
1	*2*	*3*	*4*

Transmitted Data

10011001	**11100010**	**00100100**	**10000100**	**11011010**
				Checksum

```
1       10011001
2       11100010
       ----------
       101111011
                1
       ----------
        01111100
3       00100100
       ----------
        10100000
4       10000100
       ----------
       100100100
                1
       ----------
Sum:     00100101
       ----------
CheckSum: 11011010
```

Error Detection | Checksum



Checksum Checker

Received Data

10011001	**11100010**	**00100100**	**10000100**	**11011010**

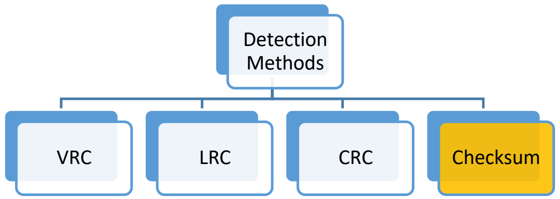
Final Data after dropping Redundant Bits

10011001	**11100010**	**00100100**	**10000100**

```
1            10011001
2            11100010
            ----------
            101111011
                    1
            ----------
             01111100
3            00100100
            ----------
             10100000
4            10000100
            ----------
            100100100
                    1
            ----------
             00100101
             11011010
            ----------
Sum:         11111111
Complement:  00000000
```

Indicates no ERROR

Error Detection | Checksum



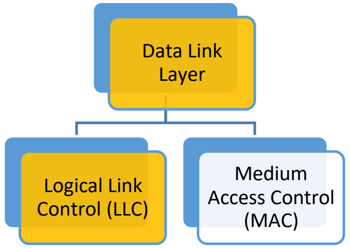
If the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the **sum and checksum remain the same**.

- Fletcher Checksum
- Adler Checksum

Weight each data items according to its position.

If the value of several words are incremented but **the sum and checksum do not change**, the **errors are not detected**.

Logical Link Control (LLC) | Error Correction



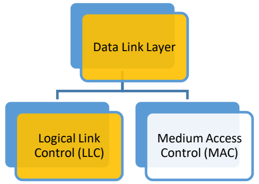
Error correction can be handled in two ways:

- When an error is discovered, the receiver can request the sender to retransmit the **entire data unit**.
- The receiver can use an error-correcting code, which **automatically corrects certain errors**.

Error correcting codes are **more sophisticated** than error-detection codes and **require more redundancy bits**.

Most error correction is limited to ***one, two*** or ***three-bit errors***.

Logical Link Control (LLC) | Error Correction



Single-Bit Error Correction

VRC can be used to detect a single-bit error in the received frame. BUT the **major concern** is to **locate the invalid bit**.

To correct a single-bit error in an ASCII character, the error correction code **must determine** which of the seven bits has changed.

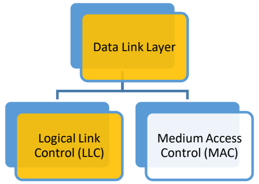
This requires **enough redundancy bits** to show all **eight states** (No Error, Error in position 1 to 7).

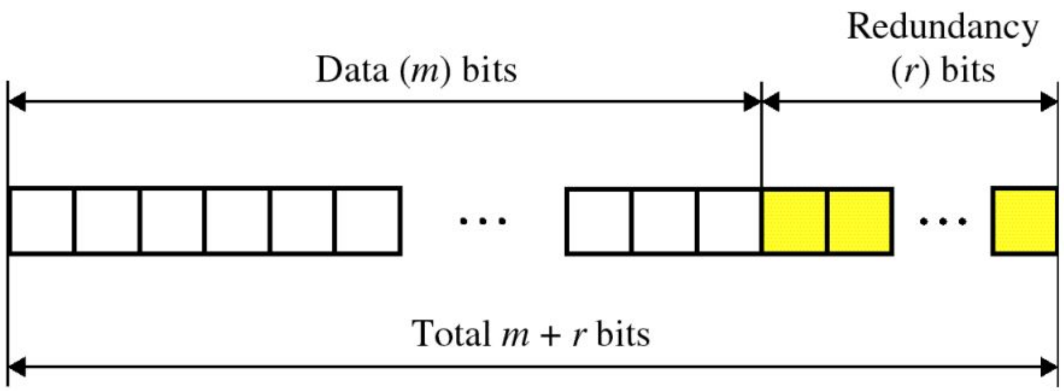
A **three-bit redundancy code** should be adequate and can therefore indicate the locations of eight different possibilities.

What if an error occurs in redundancy bits themselves?

Additional bits are necessary to cover all possible error locations.

Error Correction | Single-Bit Error Correction





If the **total number of bits in a transmittable unit is** $(m + r)$**,** then **'r' must be able to indicate** at least $(m + r + 1)$ **different states**.

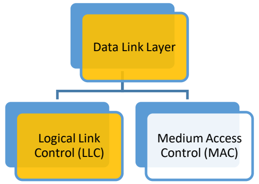
'r' bits can indicate 2^r different states. Therefore, $\mathbf{2^r \geq (m + r + 1)}$

If the value of **'m' is 7**, the smallest 'r' value that can satisfy the above equation will be **4**:

$$\mathbf{2^4 \geq (7 + 4 + 1)}$$

NUMBER OF DATA BITS (M)	NUMBER OF REDUNDANCY BITS (R)	TOTAL BITS (M+R)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

Error Correction | Single-Bit Error Correction



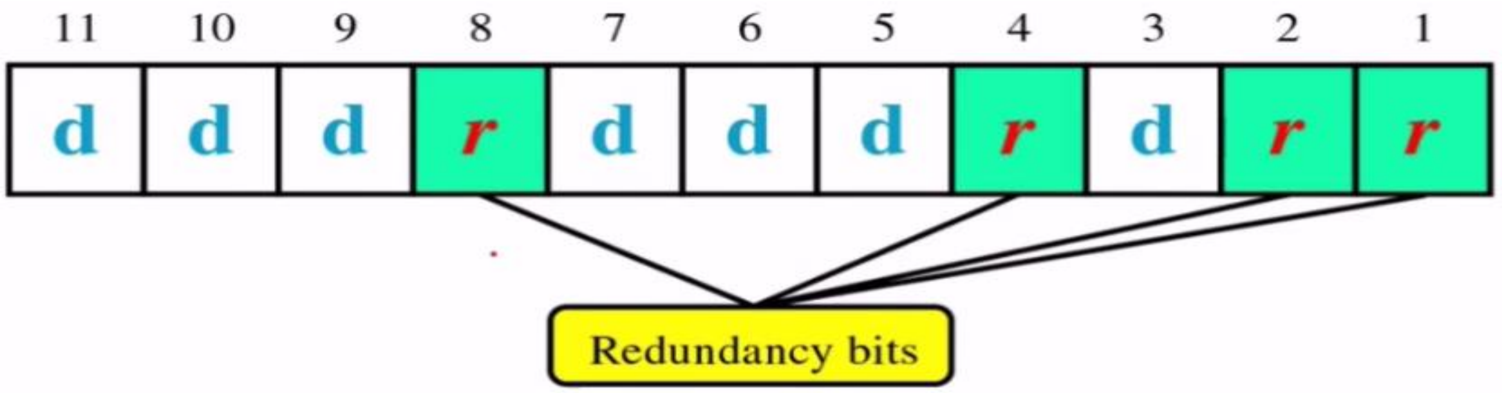
How do we manipulate those bits to discover which state has occurred?

Hamming Code

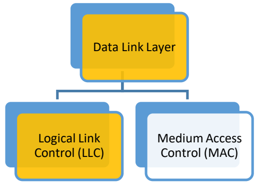
The Hamming code can be **applied to data units of any length** and uses the relationship between data and redundancy bits.

Example: A 7-bit ASCII code **requires 4 redundancy bits** that can be **added to the end of the data unit** or interspersed with the original data bits.

The **redundant bits** are placed in positions **1, 2, 4, and 8** (the positions in an 11-bit sequence that are **powers of 2**).



Error Correction | Single-Bit Error Correction



11	10	9	8	7	6	5	4	3	2	1
d	d	d	*r*	d	d	d	*r*	d	*r*	*r*
			4th				3rd		2nd	1st

Each 'r' bit is the **VRC bit for one combination of data units**. The combinations used to calculate each of the four 'r' values (r_1, r_2, r_3, r_4) for a 7-bit data sequence are as follows:

r_1	1	3	5	7	9	11
	0001	0011	0101	0111	1001	1011

Check for availability of **'1' at 1st place**.

r_4	4	5	6	7
	0100	0101	0101	0111

Check for availability of **'1' at 3rd place**.

r_2	2	3	6	7	10	11
	0010	0011	0110	0111	1010	1011

Check for availability of **'1' at 2nd place**.

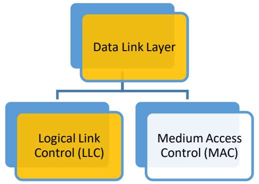
r_8	8	9	10	11
	1000	1001	1010	1011

Check for availability of **'1' at 4th place**.

Each data bit may be included in **more than one VRC calculation**.

Each of the 'm' bits is included in **at least two sets**, while the **'r' bits** are included in **only one**.

Error Correction | Single-Bit Error Correction



Original Data (m) = 1001101

1	0	0		1	1	0		1		
11	10	9	8	7	6	5	4	3	2	1

Adding r_1:

1	0	0		1	1	0		1		
11	10	9	8	7	6	5	4	3	2	1

$r_1 = 1$

Adding r_2:

1	0	0		1	1	0		1		1
11	10	9	8	7	6	5	4	3	2	1

$r_2 = 0$

Adding r_4:

1	0	0		1	1	0		1	0	1
11	10	9	8	7	6	5	4	3	2	1

$r_4 = 0$

Adding r_8:

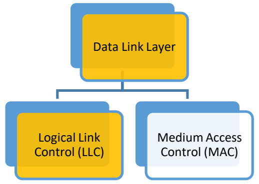
1	0	0		1	1	0	0	1	0	1
11	10	9	8	7	6	5	4	3	2	1

$r_8 = 1$

Code (m+r) = 10011100101

EVEN PARITY

Error Correction | Single-Bit Error Correction



Lets imagine that by the time the transmission (**10011100101**) is <u>received</u>, the number 7 bit has been changed from **1 to 0** (**10010100101**).

The receiver takes the transmission and **<u>recalculates four new VRCs</u>** using the <u>same sets of bits used by the sender plus the relevant parity (r) bit for each set</u>.

10010100101

10010100101	**r_1 = 1**
10010100101	**r_2 = 1**
10010100101	**r_4 = 1**
10010100101	**r_8 = 0**

0	**1**	**1**	**1**	= 7

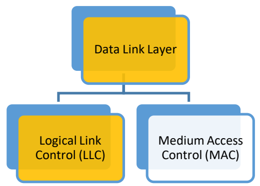
10010100101

Error at the location 7.

Receiver can **<u>reverse the 7th bit to correct the error</u>**.

10011100101

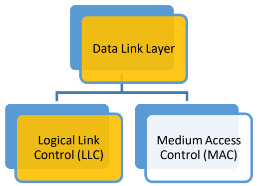
Data Link Layer | Logical Link Control (LLC)



- Framing
  - Character Count
  - Character Stuffing
  - Bit Stuffing
  - Physical Layer Coding Violation
- Flow Control
  - Stop-and-Wait
  - Sliding Window
- Error Control
  - Stop-and-Wait ARQ
  - Go-back-n ARQ
  - Selective-reject ARQ

- Error Detection and Correction
  - Types of Errors
  - Detection
  - Correction
- **Protocol**
  - **High-Level Data Link Control (HDLC)**

Logical Link Control (LLC) | High-Level Data Link Control Protocol



HDLC is s bit-oriented data link protocol designed to support both half duplex and full-duplex communication over **point-to-point** and **multi-point links**.

Station Type:

Primary Station – has complete control of the link. **Sends Command** to the secondary stations.

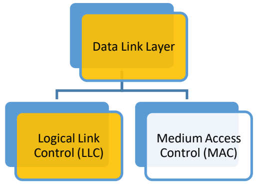
Secondary Station – **Respond** to the primary station.

Combined Station – can **Command and Respond**.

Configurations: *Relationship of hardware devices on a link*

- **Unbalanced Configuration** (also called a master/slave configuration) – one device is primary and the others are secondary.
- **Balanced Configuration** – both stations are combined type. The stations are linked by a single line that can be controlled by either station.
- **Symmetrical Configuration** – Each physical station on a link consists of **two logical stations** (one a primary and the other is secondary). Behaves like an unbalanced configuration **except that control of the link can shift between the two stations**.

High-Level Data Link Control Protocol



Modes of Communication:

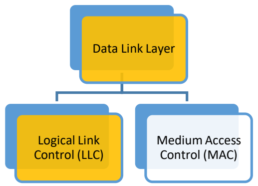
Normal Response Mode (NRM) – Refers to the standard primary-secondary relationship. **A secondary device must have permission** from the primary device before transmitting.

Asynchronous Balanced Mode (ABM) – **All stations are equal** and therefore only combined stations connected in point-to-point are used.

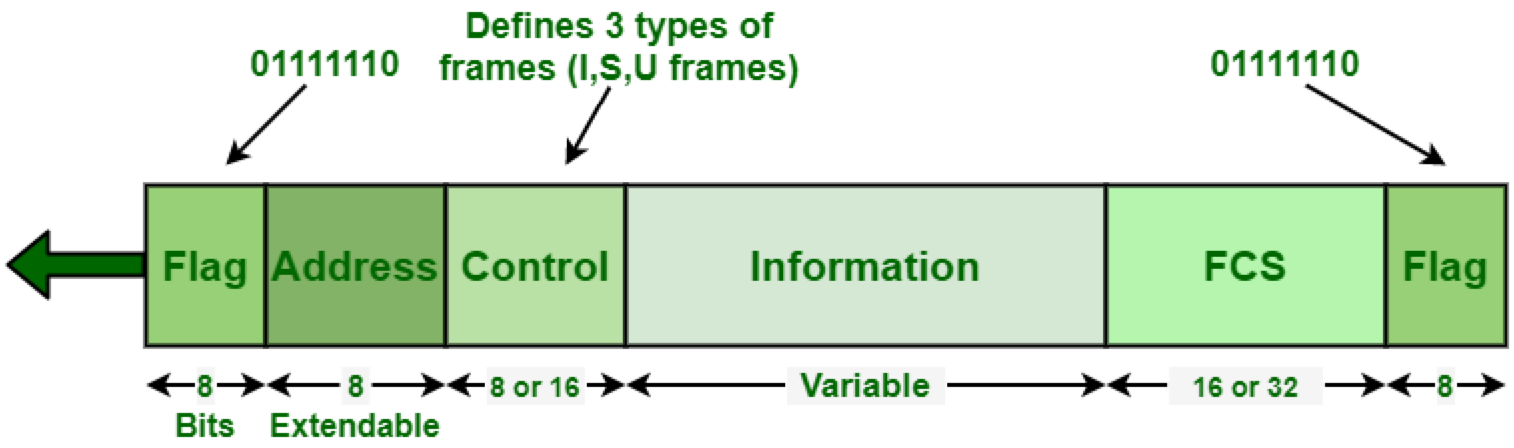
Asynchronous Response Mode (ARM) – **A secondary may initiate a transmission without permission** from the primary whenever the channel is idle.

	NRM	ABM	ARM
Station Type	Primary & Secondary	Combined	Primary & Secondary
Initiator	Primary	Any	Either
Configuration	Unbalanced	Balanced	Unbalanced

High-Level Data Link Control Protocol



Frame Structure:



FLAG: 01111110 (to identifies both the beginning & end of a frame and serves as a synchronization pattern for the receiver)

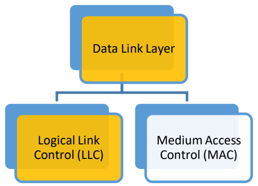
ADDRESS: Used to identify one of the terminals.

CONTROL: Used to share sequence numbers and acknowledgement of a frame.

DATA: Contain arbitrary information.

FCS (Frame Check Sequence): Used for error detection (2-byte or 4-byte standard CRC).

High-Level Data Link Control Protocol



ADDRESS: Used to identify one of the terminals.

If a **primary station creates a frame**, it contains a **to-address** in the ADDRESS field of a frame.

If a **secondary station creates a frame**, it contains a **from-address** in the ADDRESS field of a frame.

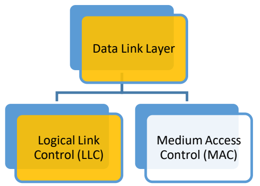
If the address field is only one byte, the **last bit is always a 1**.

7 bits	**1 bit**	
	1	**LSB**

If the address field is more than one byte, **all bytes but the last one will end with 0**; **only the last byte will end with 1**. Ending each intermediate byte with 0 **indicates to the receiver** that **there are more address bytes to come**.

For MAC Ethernet – 6 bytes address occur (48 bits – so six such address fields will be used).

High-Level Data Link Control Protocol



Types of Frames: Works as an **envelope for the transmission** of a different type of message.

- **Information (I-frames):** To transfer user data and control information of a different type of message.
- **Supervisory (S-frames):** To transfer only control information (**Error and Flow control**).
- **Unnumbered (U-frames):** Reserved for system management and managing link.

The **contents of the control field** for these three kinds are:

Flag	Address	Control	Information	FCS	Flag
		8			

I-Frame

0	Sequence	P/F	Next
1	3	1	3

S-Frame

1	0	Type	P/F	Next
1	1	2	1	3

U-Frame

1	1	Type	P/F	Modifier
1	1	**2**	1	**3**

The Sequence field is the **frame sequence number**.

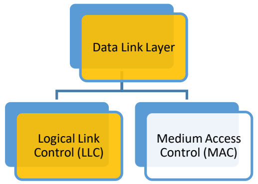
The Next field is a **piggybacked acknowledgement**.

The P/F field stands for **Poll/Final**.

The Type field is used to indicate different types of control message.

The protocol uses a sliding window, with a **3-bit sequence number**.

High-Level Data Link Control Protocol



The **P/F field** is a single-bit with a **dual purpose**. It has meaning only when it is set (bit = 1) and can mean **POLL** or **FINAL**.

- It means **POLL** when the frame is sent by a primary to a secondary (**address filed** contains **address of the receiver**).
- It means **FINAL** when the frame is sent by a secondary to a primary (**address filed** contains **address of the sender**). All the frames sent by the secondary, except the final one, have the **P/F bit set to P**. The **final one is set to F.**

Various kinds of Supervisory frames:

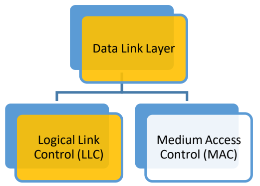
Type 0: Acknowledgement frame used to indicate the **next frame expected** (officially called **RECEIVE READY**). Value for Positive ACK is 00.

Type 1: Negative Acknowledgement frame (officially called REJECT). It is used to indicate that a transmission error has been detected. Value for Negative ACK is 01 (**Go-back-n**).

Type 2: **RECEIVE NOT REAY**. It acknowledges all frames up to but not including Next, just as RECEIVE READY, but it tells the sender to stop sending. Value for Positive ACK is 10.

Type 3: Negative Acknowledgement frame (officially called REJECT). It calls for retransmission of only the frame specified. Value for Negative ACK is 11 (**Selective Reject**).

High-Level Data Link Control Protocol



Normal: 8 Bits

I-Frame

0	Sequence	P/F	Next
1	3	1	3

S-Frame

1	0	Type	P/F	Next
1	1	2	1	3

U-Frame

1	1	Type	P/F	Modifier
1	1	2	1	3

Extendable: 16 Bits

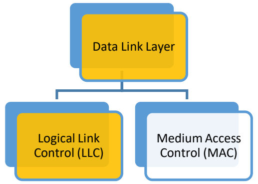
I-Frame

0	Sequence	P/F	Next
1	7	1	7

S-Frame

1	0	Type	0 0 0 0	P/F	Next
1	1	2	4	1	7

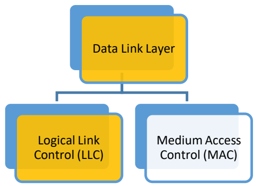
High-Level Data Link Control Protocol



DATA: Contain arbitrary information.

Contains the user’s data in an I-frame, network management information in a U-frame, and no information in a S-frame.

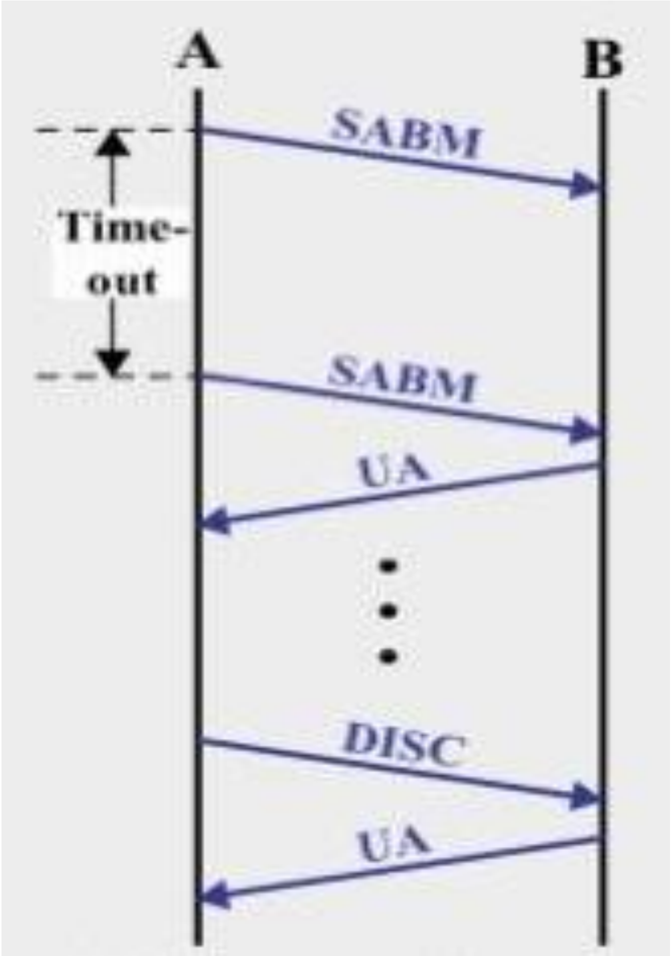
Its length can vary from one network to another but is always fixed within each network.

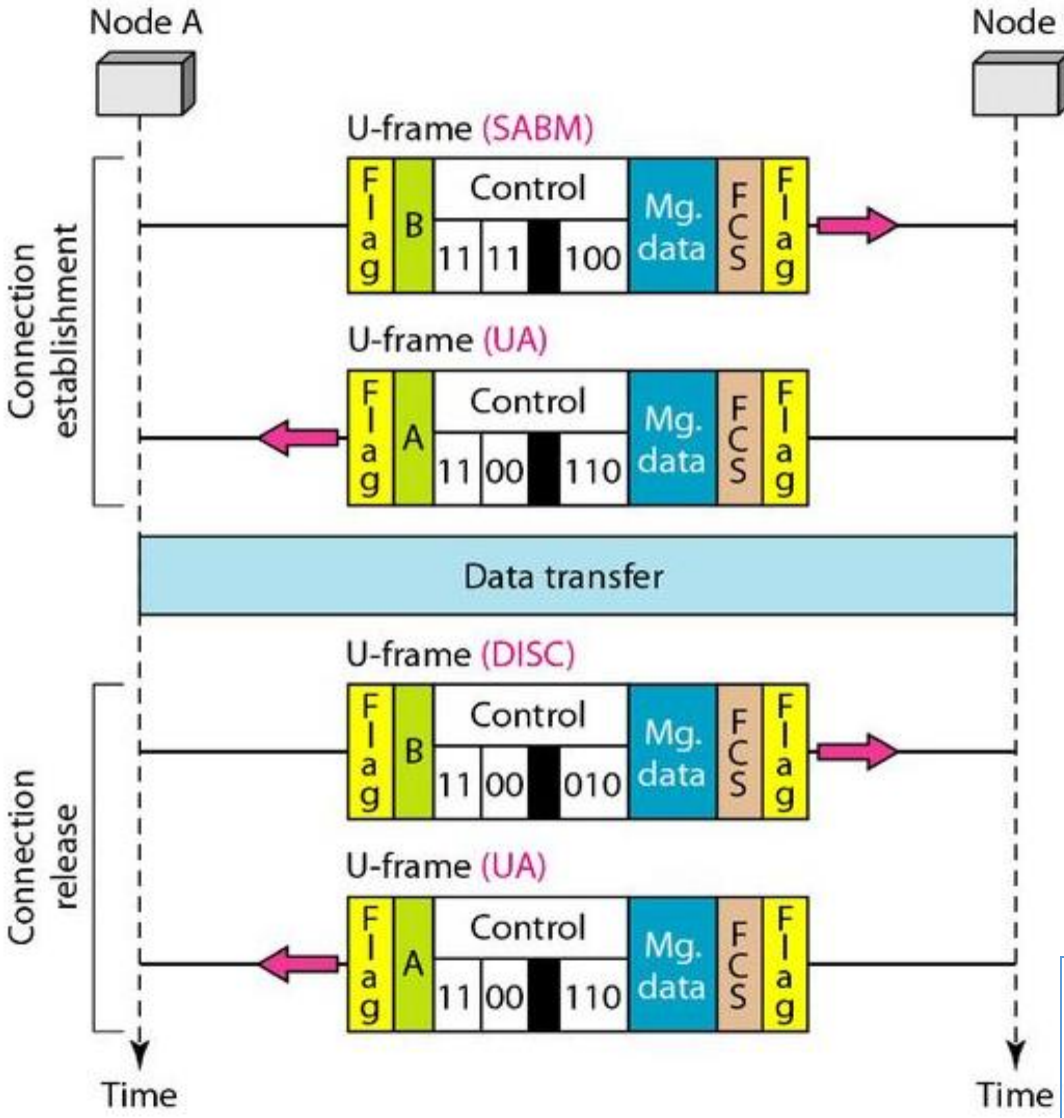


High-Level Data Link Control Protocol

EXAMPLE OF HLDC OPERATION:

Link Setup and Disconnect:





SABM: Set Asynchronous Balanced Mode

UA: Unnumbered Acknowledgement

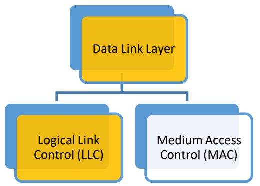
DISC: Disconnect

SABME: Set Asynchronous Balanced Extended Mode

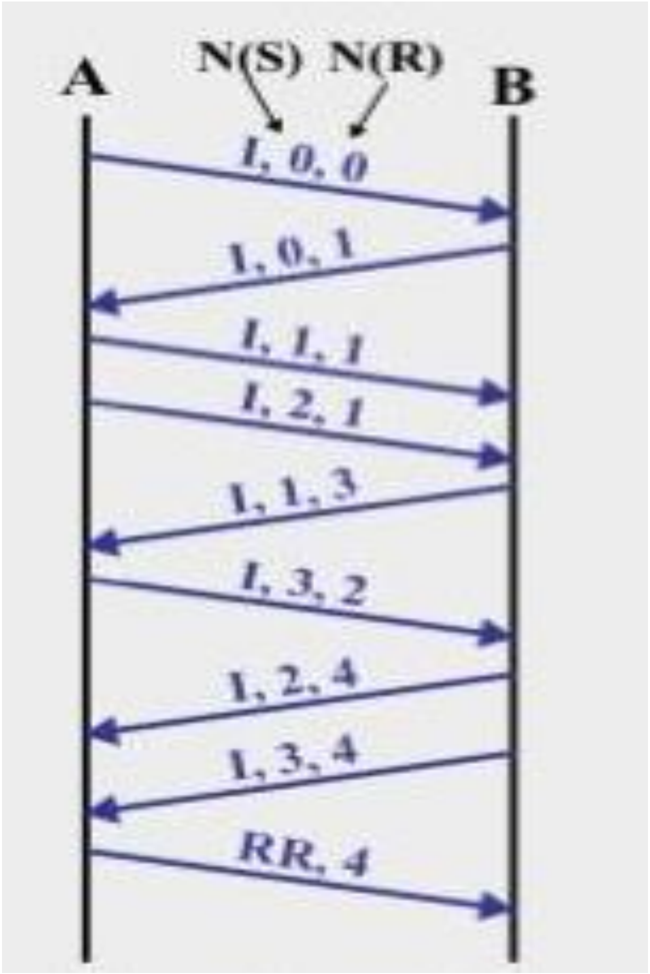
Sharing of: **U-Frames**

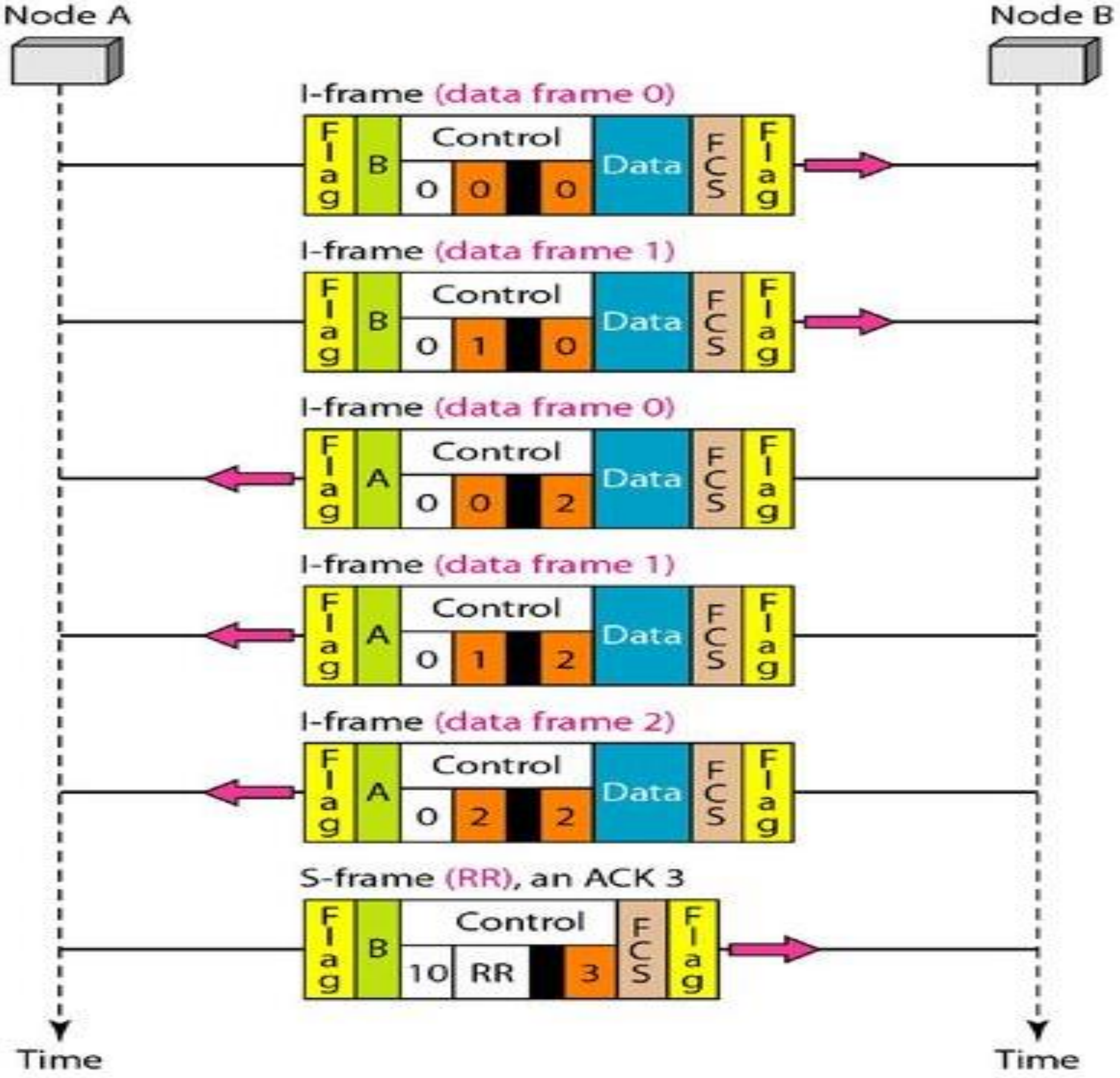
1	1	Type	P/F	Modifier
1	1	2	1	3

High-Level Data Link Control Protocol



Two-way Data Exchange:





RR: Receive Ready (Acknowledgement frame used to indicate the next frame expected). Type is 00.

Sharing of:

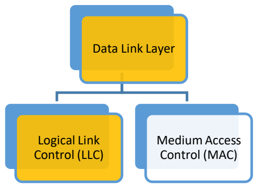
I-Frame

0	Sequence	P/F	Next
1	3	1	3

S-Frame

1	0	Type	P/F	Next
1	1	2	1	3

High-Level Data Link Control Protocol



Busy Condition:

Sharing of **I & S-Frames:**

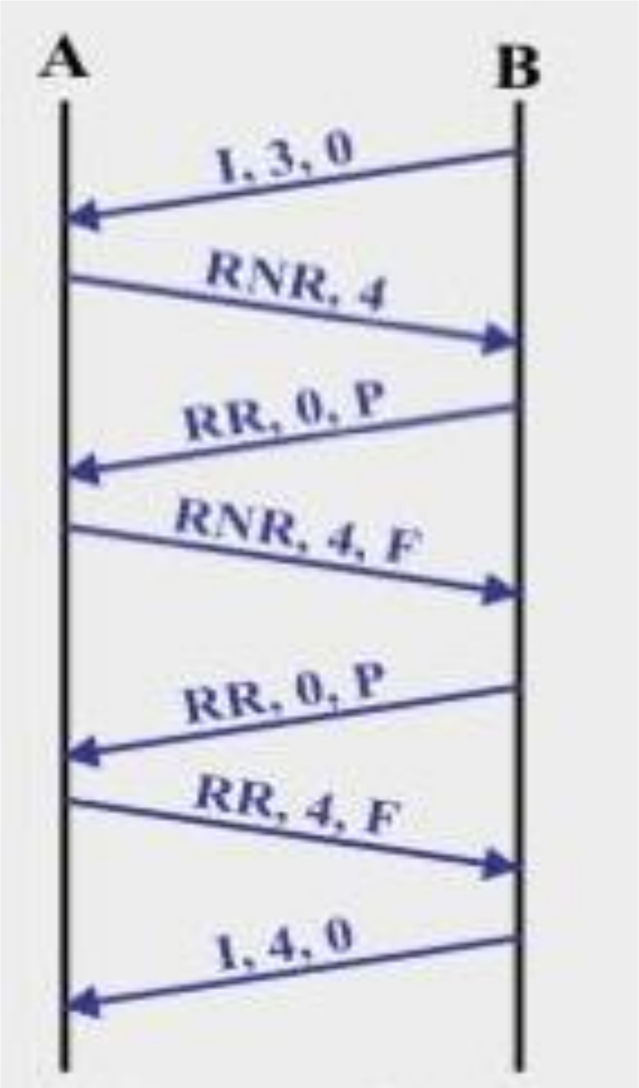
0	Sequence	P/F	Next
1	3	1	3

1	0	Type	P/F	Next
1	1	2	1	3

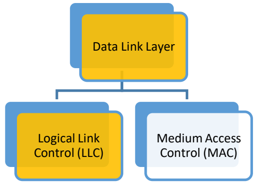
RR: Receive Ready (Acknowledgement frame used to indicate the next frame expected). Type is 00.

RNR: Receive NOT Ready (Positive Acknowledgement frame used to indicate not ready to receive). Type is 10.

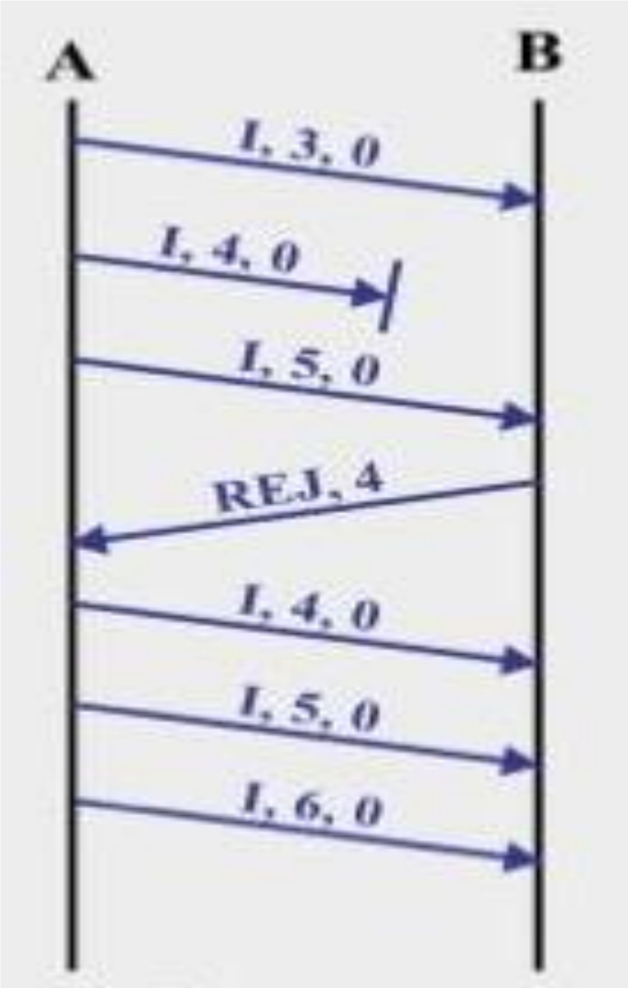
- **Station ‘A’ issues an RNR**, which requires ‘B’ to **halt transmission of I-frames**.
- Station ‘B’ **POLL the busy station ‘A’** at some periodic interval by sending an RR with the ‘P’ bit set.
- Station ‘A’ respond with **an RR or an RNR**.

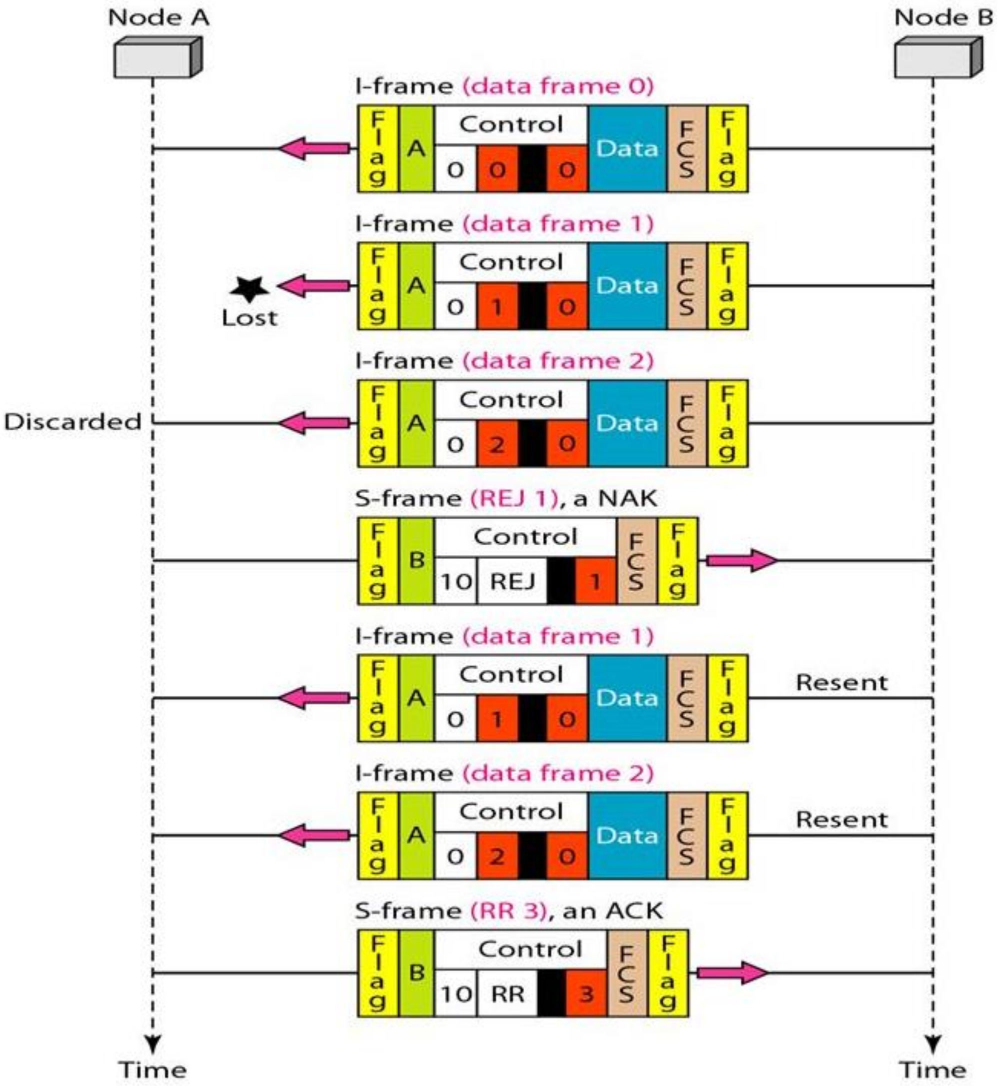


High-Level Data Link Control Protocol



Reject Recovery:





REJ: Negative Acknowledgement frame. It is used to indicate that a transmission error has been detected. Value for Negative ACK is 01 **(Go-back-n)**.

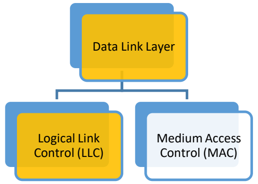
RR: Receive Ready (Acknowledgement frame used to indicate the next frame expected). Type is 00.

Sharing of:

I-Frame	0	Sequence	P/F	Next
	1	3	1	3

S-Frame	1	0	Type	P/F	Next
	1	1	2	1	3

High-Level Data Link Control Protocol



Timeout Recovery:

Sharing of **I & S-Frames:**

0	Sequence	P/F	Next
1	3	1	3

1	0	Type	P/F	Next
1	1	2	1	3

Lets assume **I-frame no 3** is the **LAST FRAME in sequence**.

- **I-frame no 3** of station 'A' is **lost on transit**.

Recovery action will initiate after timeout period.

- Station 'A' will **send RR** (Receive Ready - Acknowledgement frame used to indicate the **next frame expected**) with a P bit set (**demands a response**).

- Station 'B' will **send RR** (Receive Ready - Acknowledgement frame used to indicate the **next frame expected**) with an F bit set (**demands a response**).