DATA LINK LAYER

- Introduction
- Block coding
- Cyclic codes
- Checksum
- framing
- flow & error control
- Noiseless channels
- Noisy channels
- HDLC
- Point to point protocols

⇒ Introduction:

- Networks must be able to transfer data from one device to another device with acceptable accuracy.
- For most applications, a system must guarantee that the data received are identical to the data transmitted.
- Any time data are transmitted from one node to the next, they can become corrupted in passage.
- Many factors can alter one or more bits of message.
- Some applications require a mechanism for detecting & correcting errors.

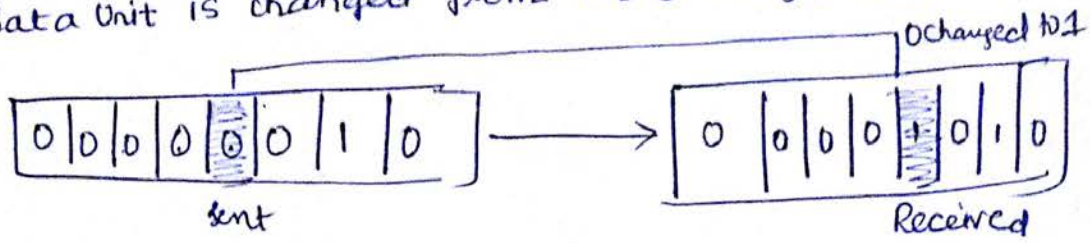
Types of errors → Single bit error
→ burst error

whenever bits flow from one point to another, they are subject to unpredictable changes because of interference.

- This interference can change the shape of the signal.
- In a single bit error, a 0 is changed to a 1 or a 1 to a 0.
- In a burst error, multiple bits are changed.

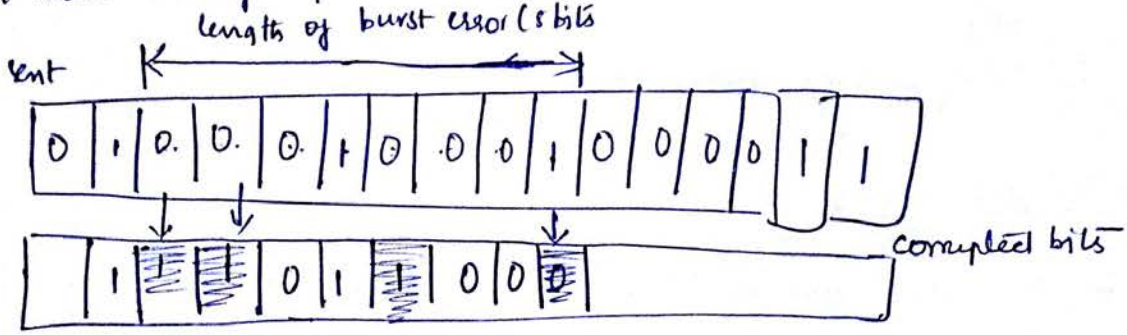
* Single-Bit Error:

↳ The term single-bit error means that only 1 bit of a given data unit is changed from 1 to 0 or from 0 to 1



* Burst Error:

The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1



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

Detection Versus Correction.

↳ The correction of errors is more difficult than detection.

↳ In error detection, we look only to see if any error has occured. The Answer is simple yes or no.

↳ In error correction, we need to know the exact number of bits that are corrupted & more importantly, their location in the message. The no. of errors & the size of the message are important factors.

Forward Error Correction Vems Retransmission.

→ Two methods of error correction

1) Forward Error Correction
2) Retransmission.

↳ Forward Error Correction is the process in which the receiver tries to guess the message by using redundant bits. it is possible if the number of errors is small.

↳ Retransmission is the technique in which the receiver detects the occurrence of an error and asks the sender to resend the message. Resending is repeated until a message arrives that the receiver believes is error free.

Coding.

↳ Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship b/w the redundant bits & the actual data bits.

↳ The receiver checks the relationships b/w the two sets of bits to detect or correct the errors.

↳ The ratio of redundant bits to the data bits and the robustness of the process are important factors in any coding scheme.

↳ We divide coding schemes into two broad categories

↓ ↓

block coding convolution coding.

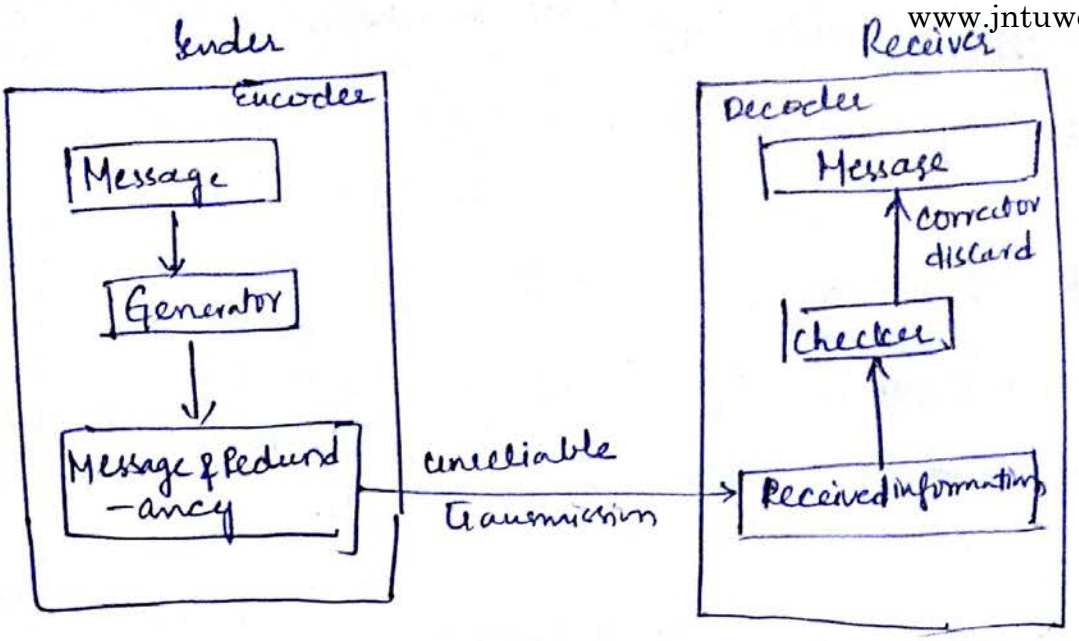


Fig: Structure of Encoder & Decoder.

⇒ Block Coding:

↳ In Block coding, we divide our message into blocks, each of k bits, called datawords.

↳ redundant bits 'r' are added to each block to make the length $n = k + r$.

↳ The resulting n-bit blocks are called codewords.

↳ We have a set of datawords, each of size k, & a set of codewords, each of size of n.

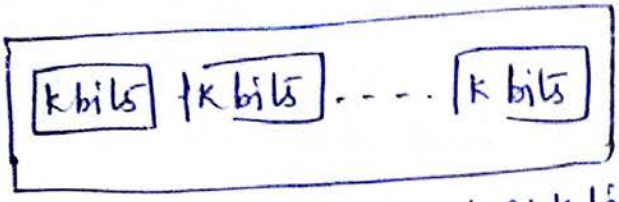
↳ with k bits we can create a combination of 2^k datawords;

↳ with n bits we can create a combination of 2^n codewords.

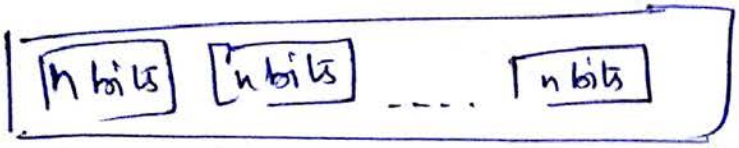
↳ Since $n > k$, the number of possible codewords is larger than the number of possible datawords.

↳ The block coding process is one-to-one; the same dataword is always encoded as the same codeword.

↳ This means that we have $2^n - 2^k$ codewords that are not used. We call these codewords invalid or illegal.



2^k Datawords, each of k bits



2^n codewords, each of n bits (only 2^k of them are valid)

Fig: Datawords & codewords in block coding.

Error detection:

↳ If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords
2. The original codeword has changed to an invalid one.

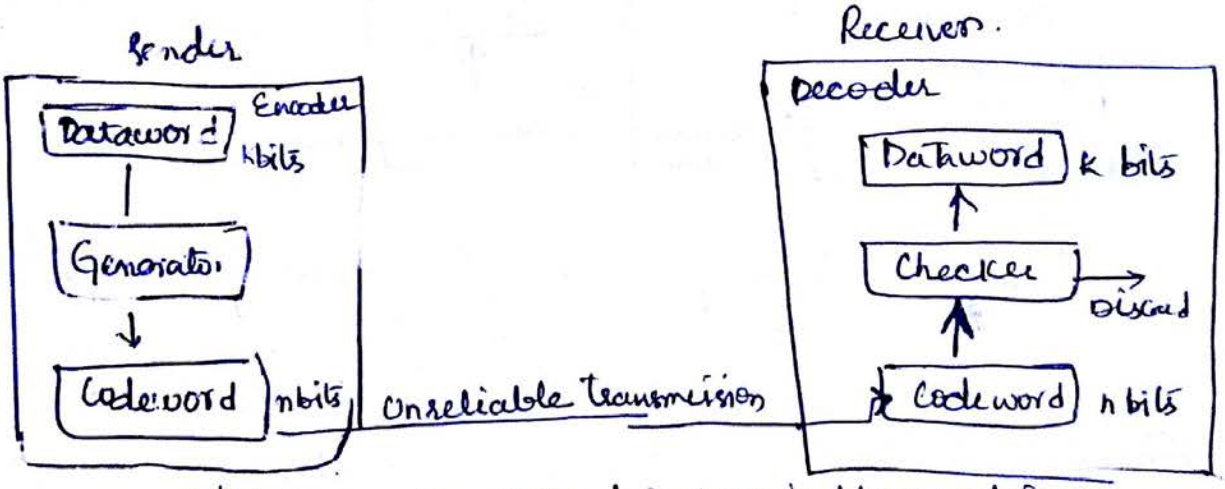


Fig: Process for error detection in block coding

↳ The sender creates codewords out of datawords by using a generator that applies the rules & procedures of encoding. Each codeword sent to the receiver may change during transmission.

If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword is ~~the~~ extracted for use.

If the received codeword is not valid, it is detected. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected. This type of coding can detect only single errors. Two or more errors may remain undetected.

Error Correction:

↳ Error correction is much more difficult than error detection.

↳ In error detection, the receiver needs to know only that the received codeword is invalid, in error correction the receiver needs to find the original codeword sent,

↳ There is need for more redundant bits for error correction than for error detection.

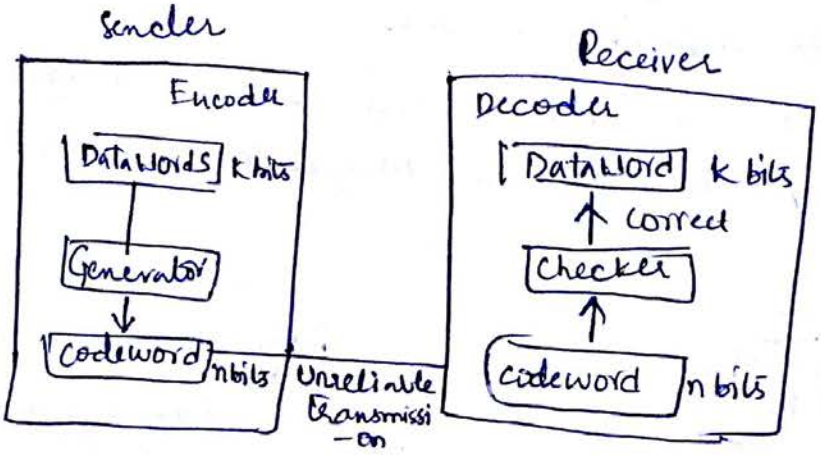


Fig: Structure of encoder & decoder in error correction

Hamming Distance:

↳ One of the central concept in coding for error control is the idea of Hamming distance.

↳ The Hamming distance between two words (of the same size), is the number of differences b/w the corresponding bits. We show the Hamming distance b/w two words x & y as d(x,y).

↳ The Hamming distance b/w two pair of words.

d(000, 011) is 2 because 000 ⊕ 011 is 011

Minimum Hamming distance:

↳ The concept of the Hamming distance is the central point in dealing with error detection & correction codes, the measure-ment that is used for designing a code is the minimum Hamming distance.

↳ In a set of words, the minimum Hamming distance is the smallest hamming distance b/w all possible pairs.

↳ We use d_{min} to define the minimum Hamming distance in a coding scheme.

↳ TO find this value, the Hamming distances between all words & select the smallest one.

The minimum hamming distance of the coding scheme

Dataword	Codeword
00	000[illegible]
01	[illegible]011
10	101[illegible]
11	[illegible]110

↳ Hamming distances

d(00000, 01011) = 3 d(00000, 10101) = 3 d(00000, 11110) = 4

d(01011, 10101) = 4 d(01011, 11110) = 3 d(10101, 11110) = 3

The $d_{min} = 3$

Three Parameters

↳ Any coding scheme needs to have atleast three parameters the codeword size n, the dataword size k, & the minimum Hamming distance d_{min}.

A coding scheme C is written as C(n, k) with a seperate expression for d_{min}.

↳ coding scheme C(3,2) with $d_{min} = 2$.

Hamming distance & Errors

↳ Relationship b/w errors occuring during transmission & Hamming distance.

↳ when a codeword is corrupted during transmission, the Hamming distance b/w the sent & received codewords is the number of bits affected by the error.

↳ The Hamming distance b/w the received codeword & the sent codeword is the number of bits that are corrupted during transmission.

↳ If codeword 00000 is sent & 01101 is received, 3 bits are in error & the hamming distance b/w the two is d(00000, 01101) = 3.

Minimum Distance for Error Detection.

↳ TO guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s+1$.

Minimum Distance for Error Correction

↳ TO guarantee the correction of up-to t errors in all cases, the minimum hamming distance in a block code must be $d_{min} = 2t+1$. [t = radius]

1.3 cyclic codes:

Cyclic codes are special linear Block codes with one extra proper-ty.

↳ In cyclic codes, if a codeword is cyclically shifted/rotated, then the result is another codeword.

i.e, Consider two words (a_0 to a_6) & (b_0 to b_6).

↳ The bits are shifted as:

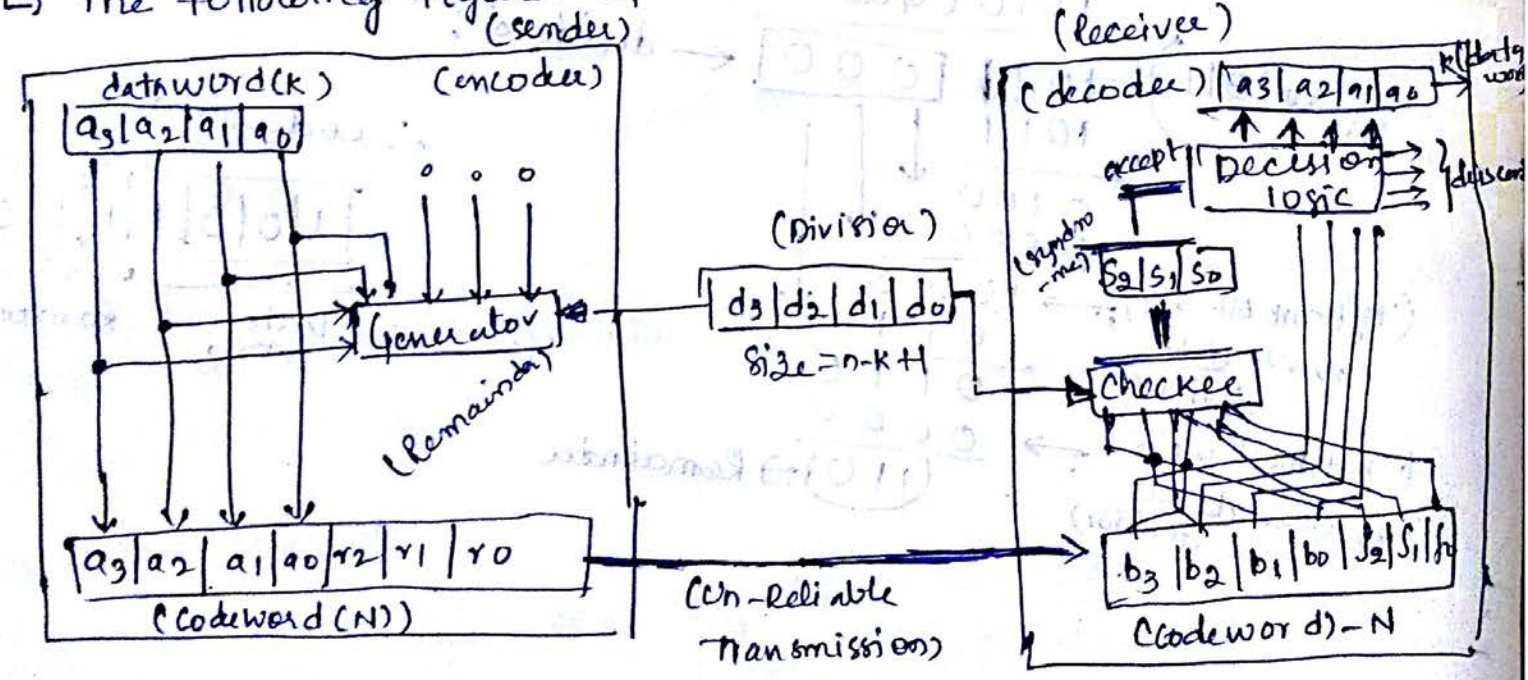
$b_1 = a_0$, $b_2 = a_1$; $b_3 = a_2$; $b_4 = a_3$; $b_5 = a_4$; $b_6 = a_5$; $b_0 = a_6$.

Cyclic redundant check (CRC)

↳ We can create cyclic codes to correct errors, using CRC.

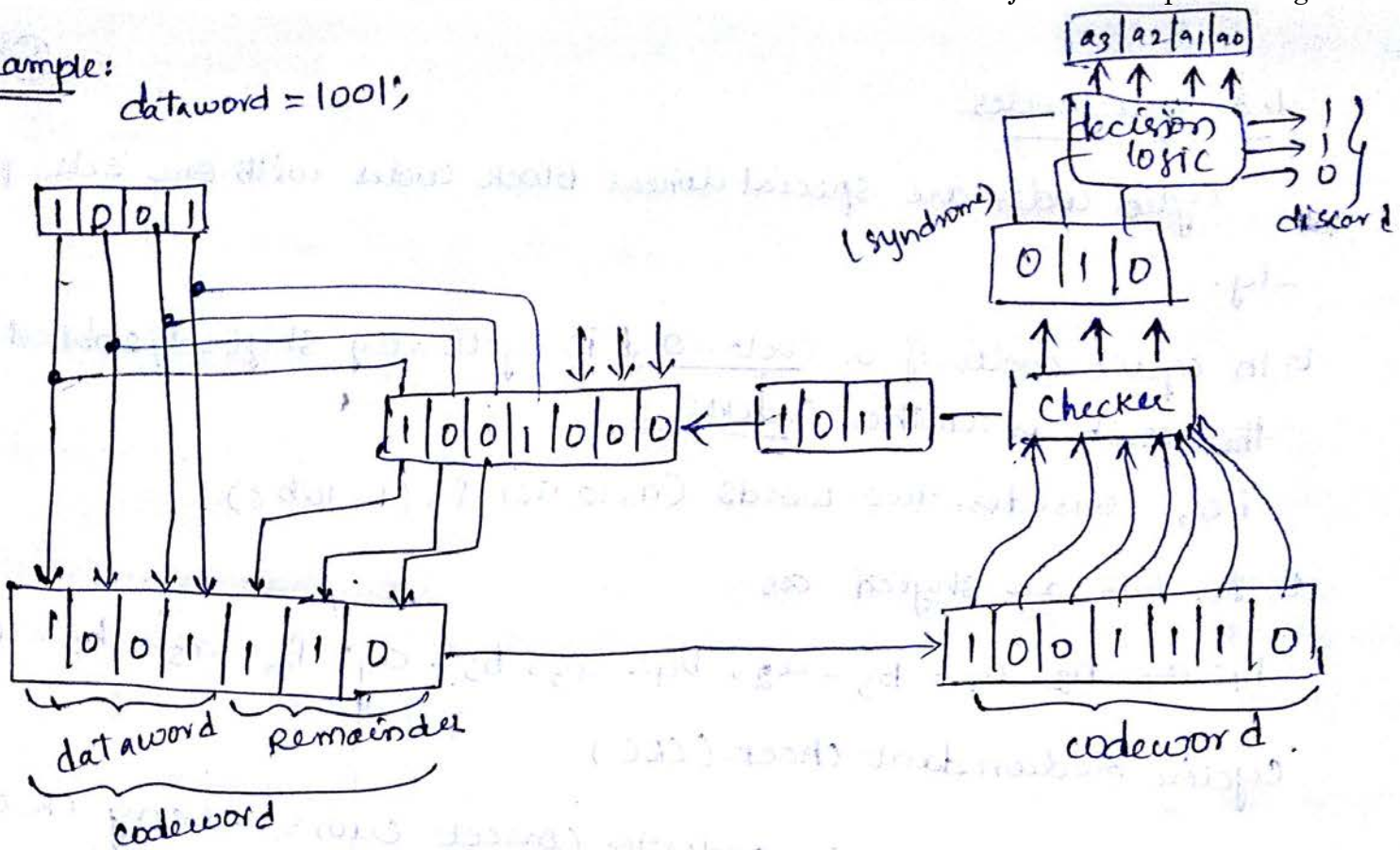
↳ CRC is popularly used in LAN's & WAN's.

↳ The following figure explains about the CRC encoder & decoder.



Example:

dataword = 1001;



CRC

division at encoder:

1010 (Quotient)

Divisor 1011) 1000 0 0 0 ← dividend:

1011

0100

0000

(leftmost bit '1'), so we use 1011 divisor → 1000

1011

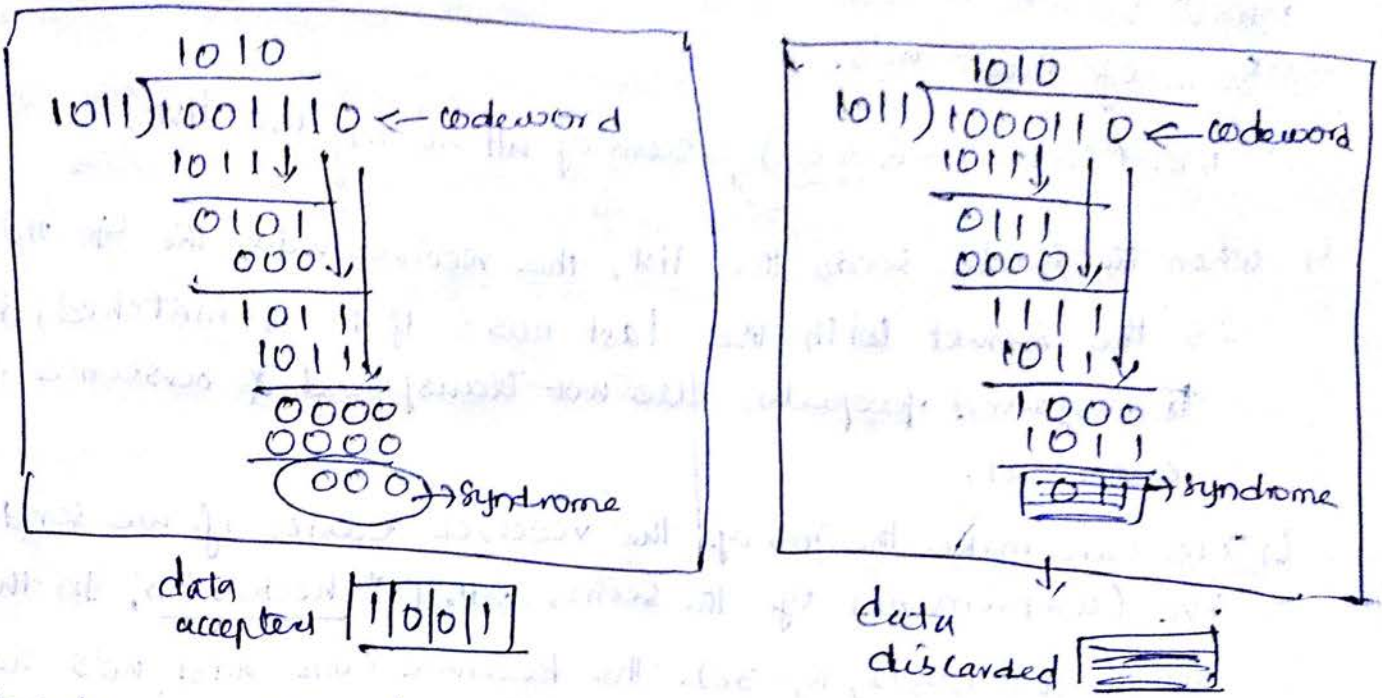
0110

(left most bit '0', so use 0000 divisor) → 0000

(110) ⇒ Remainder

∴ codeword =

* division in CRC decoder for two cases:



Note: In a cyclic code,

1. If $S(x) \neq 0$, one/more bits is corrupted.
2. If $S(x) = 0$, either
   - No bit is corrupted, or
   - Some bits are corrupted, but the decoder failed to detect them.

Advantage:

Cyclic codes are very popular because they detects single Bit/ Burst error errors

* Check Sum :-

- Checksum is also very popular error detection method.
- Checksum is not used in DLL, but used by another protocols/layers.
- Checksum is also based upon the concept of Redundancy.

Q: Consider a list of 4 numbers to be transferred from S–D what we do is, we send the list of 4 no's along with the sum of these no's.

i.e. (7, 11, 12, 6, 36) → sum of all no in the list.

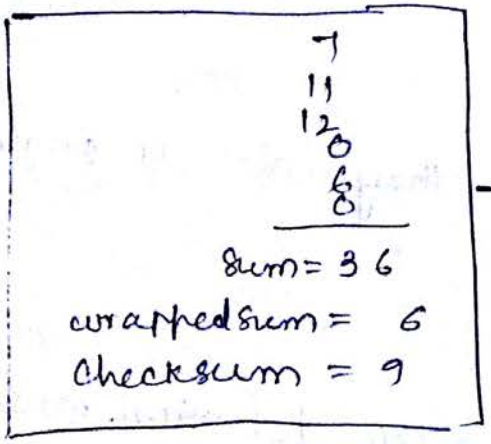
↳ when the sender sends this list, the receiver adds the 4 no's & compare -s the result with the last no's. If it is matched, data is transferred properly, else not transferred & assumes error is occeured.

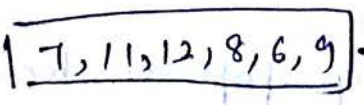
↳ We can make the job of the receiver easier if we send the negati -ve (complement) of the sum, called "Check Sum". In this case we send (7, 11, 12, 6, −36). The Receiver can add no's received (including check sum)

↳ If the result = 0, it assumes no error.

↳ If the result = 1, an error is occeured.

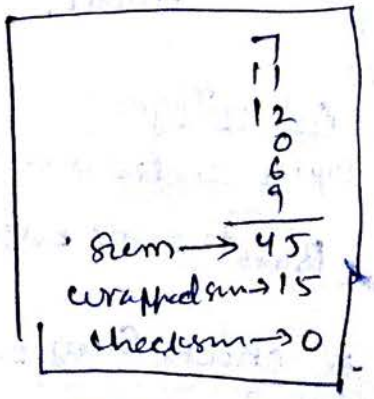
→ sender site

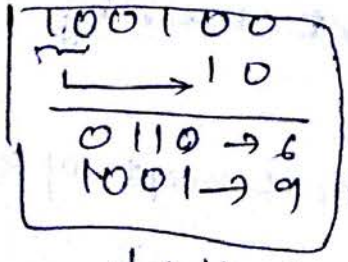




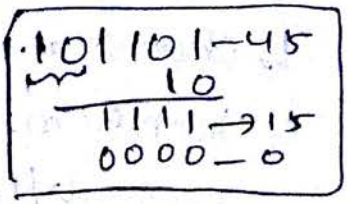
36
+ 9
9

Receiver site





details of wrapping & complementing



details of wrapping & complementing

Ex:- The sender needs to transfer the set {7, 11, 12, 0, 6} to the receiver using the error detection method checksum.

- The sender initiates the checksum = 0 & adds all data items = 36
- The data item (36) cannot be represented in 4-bits. So the extra (2) bits obtained will be wrapped & added with the sum to create the wrapped sum value = 6. (3+6=9).
- Then the sum is complemented, resulting in the checksum value 9 (15-6=9)
- The sender now sends six data items to the receiver including the checksum (9)
- The Receiver follows the same procedure as sender. It adds all the data items (including checksum): the result is 45.
- The sum is wrapped & becomes 15, the wrapped sum is complemented & becomes '0'.
- Since the value of checksum is "zero", this means that the data is not corrupted.
- The receiver drops the checksum & keeps the other data item
- If the checksum is not zero, the entire packet is dropped.

Internet checksum:

- The Internet is using a 16-bit Check sum. The checksum is calculated as:

Sender site:

- The message is divided into 16-bit words.
- The value of the checksum word is set to '0'
- All the words including the checksum are added using 1's complement addition.
- The sum is complemented & becomes the checksum.

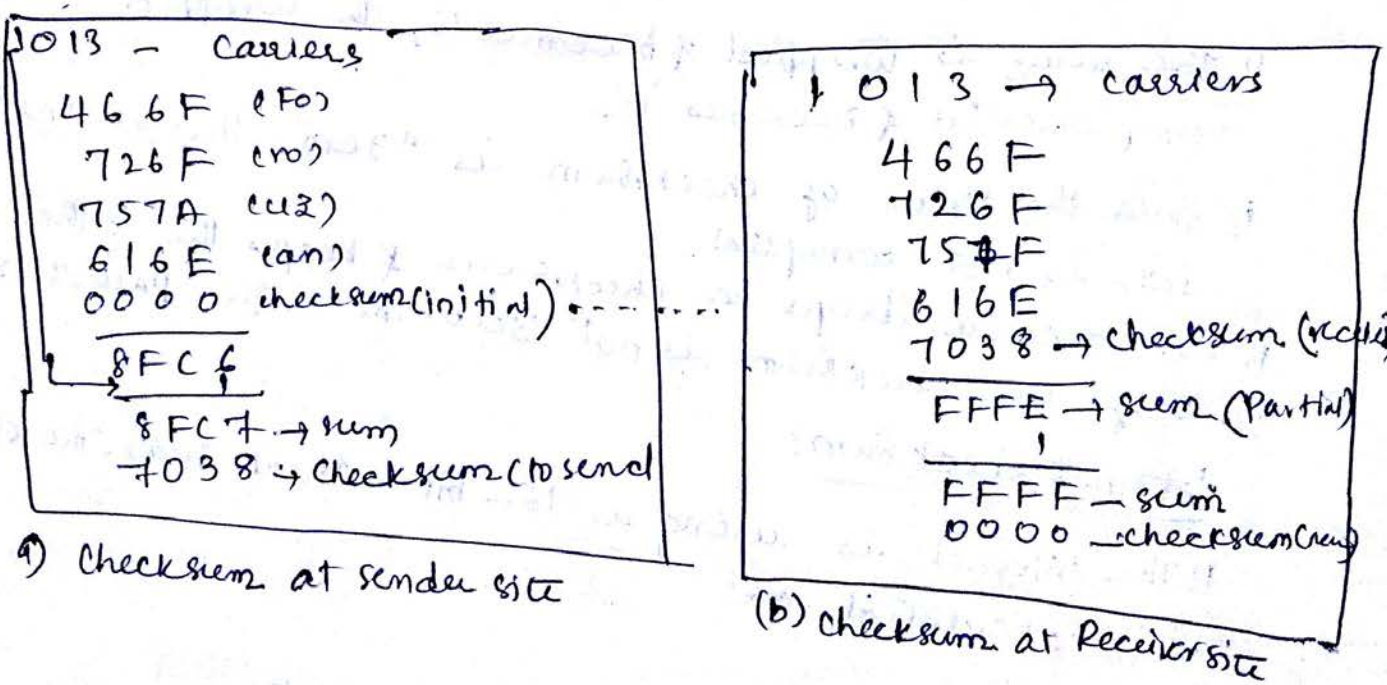
- The checksum is sent with the data.

Receiver site:

- The message (including checksum) is divided into 16-bit words.
- All words are added using 1's complement addition.
- The sum is complemented & becomes the new checksum.
- If the value of checksum = 0, the msg is accepted, otherwise rejected.

example:

Let us calculate the checksum for a text of 8 characters ("Forouzan"). The text needs to be divided into 2 bytes (16-bit) word.



(a) Checksum at sender site

(b) Checksum at Receiver site

Performance:

- The performance of checksum is not strong as the CRC in error-checking capability.
- The tendency in the internet, particularly in designing new protocols, is to replace the checksum with a CRC.

* Data link Control:

The DLL is divided into 2 parts/sublayers:

① Data link control, and

② Media Access Ctrl (MAC)

↳ The data link control (DLC)/(LLC) deals with the design & procedures for communication b/w two adjacent nodes (node-to-node communication).

↳ The functions of the DLC are:

- framing
- flow & error control
- S/w implemented protocols (that provide smooth & reliable transmission of frames b/w nodes)

* Framing

↳ The DLL, needs to pack bits into frames, and each frame is distinguishable from another.

↳ Framing is very important while data transmission because, when a message is carried in one very large frame, even a single-bit error would require the retransmission of the whole message. when a message is divided into smaller frames, a single-bit error affects only that small frame.

↳ 2 types of framing.

i) fixed-size framing, &
ii) variable-size framing.

① fixed-size framing:

frames can be fixed/variable in size. In fixed-sized framing there is no need for ~~delimiters~~ defining the boundaries of the frames the size itself can be used as a delimiter.

Ex:- of fixed size framing is ATM wide-Area Network, which uses frames of fixed size called cells.
(ATM LANS)

ii) Variable Size-framing:

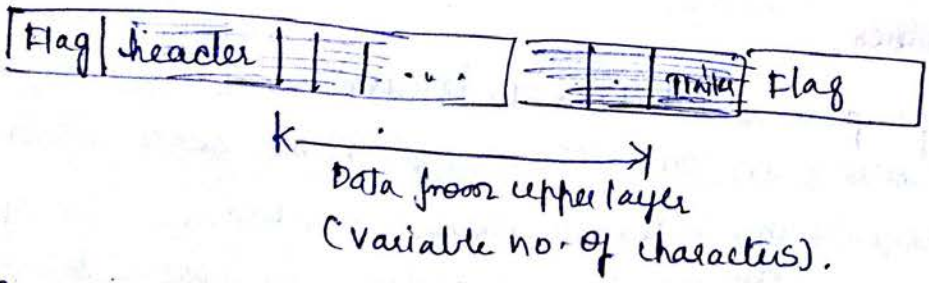
Used in LAN's. In Variable size framing, we need a way to define the end of the frame & the beginning of the next.

There are 2 approaches used for Variable-size framing.

a) character-oriented protocols, and
b) bit-oriented protocols

(a) Character-oriented protocols:

The frame format in character-oriented protocol is



Flag: (8-bits) are used to tell the start & end of the frame.

Header: It carries the source & destination address & other control information.

Trailer: It carries error correction/detection redundant bits.

Character-oriented framing was popular, when only text was exchanged by DLL.

If we send the data like audio/video/graphs, there will be a chance of this data to be matched with the flag data, so, who

happens is the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame so, to fix this problem, a byte-stuffing strategy is added to character-oriented data.

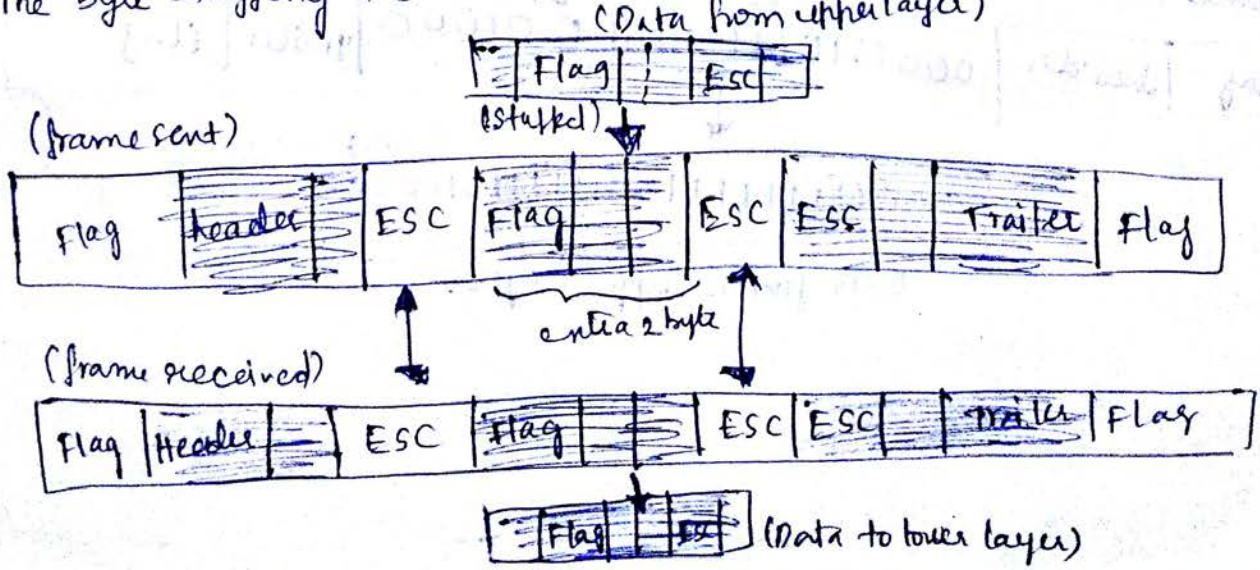
Note:

↳ In Byte stuffing / character stuffing, a special byte is added to the data section of the frame when there is a character with the same Pattern as the "flag". The data section is stuffed with an extra byte.
This byte is called as "Escape character (ESC)".

↳ whenever, the receiver encounters the ESC character, it removes it from the data section & treats the next character as data, not a delimiting flag.

↳ If the text contains 'ESC' character, then the extra one 'ESC' character is added to show that the second one is part of the text.

↳ The byte stuffing is shown as:



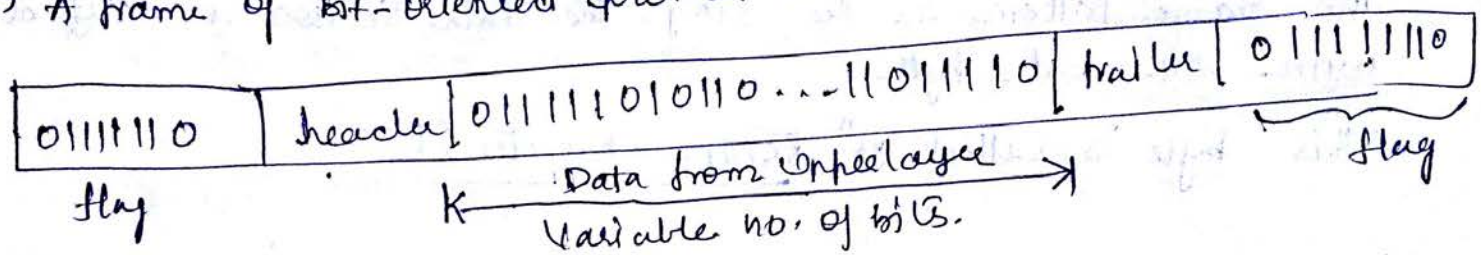
Note:

Byte-Stuffing is the process of adding 1 extra byte whenever there is a flag / escape character in the text.

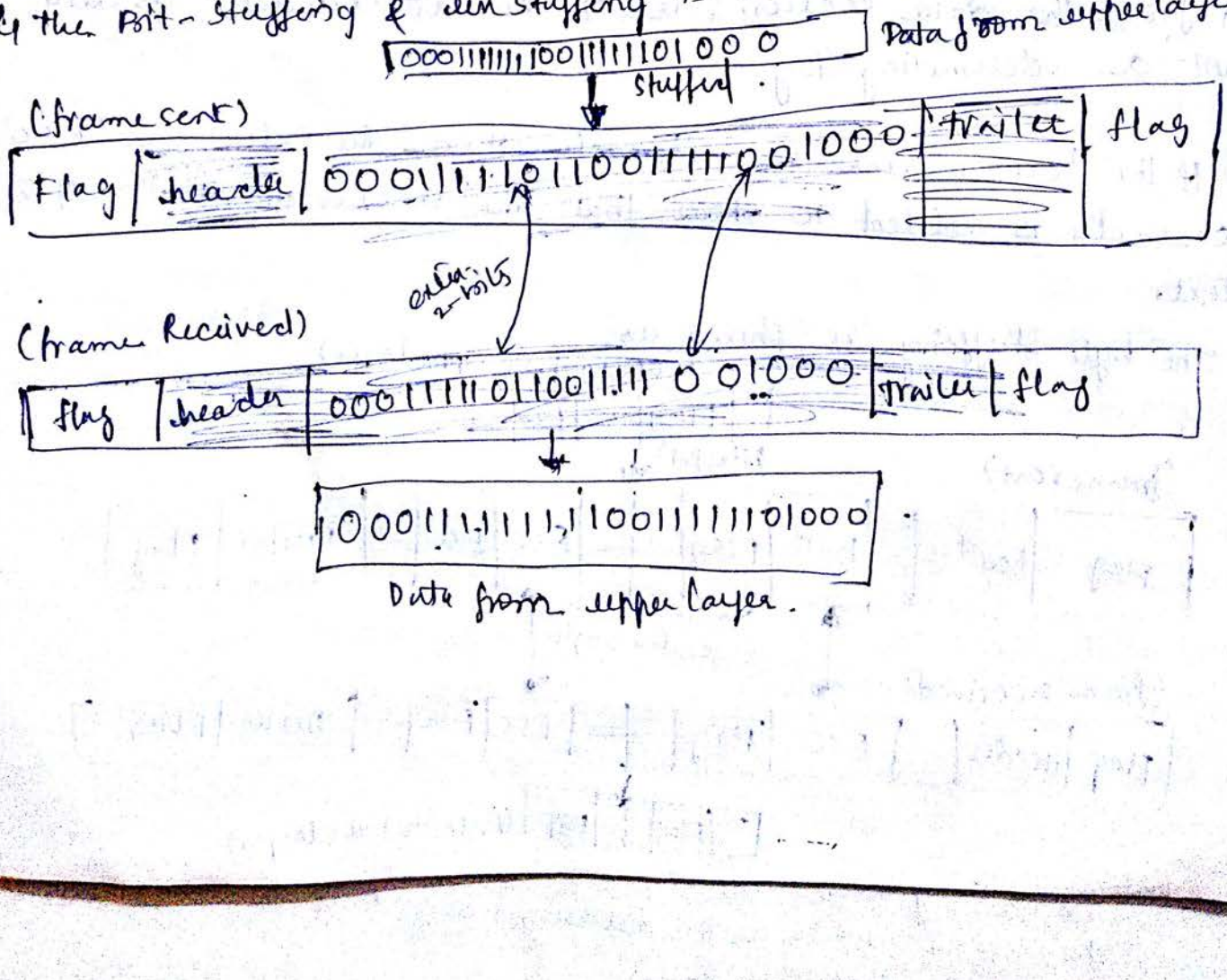
(b) Bit-Oriented protocols:

It is a process of adding 1 extra '0' whenever five consecutive 1's follow a '0' in the data, so that the receiver does not mistake the pattern "01111110" for a flag.

↳ A frame of bit-oriented protocol is shown as:



↳ the Bit-stuffing & destuffing is shown as:



Flow & 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 acknowledgement.

"Error control" in the DLL is based on automatic repeat request, which is the retransmission of data.

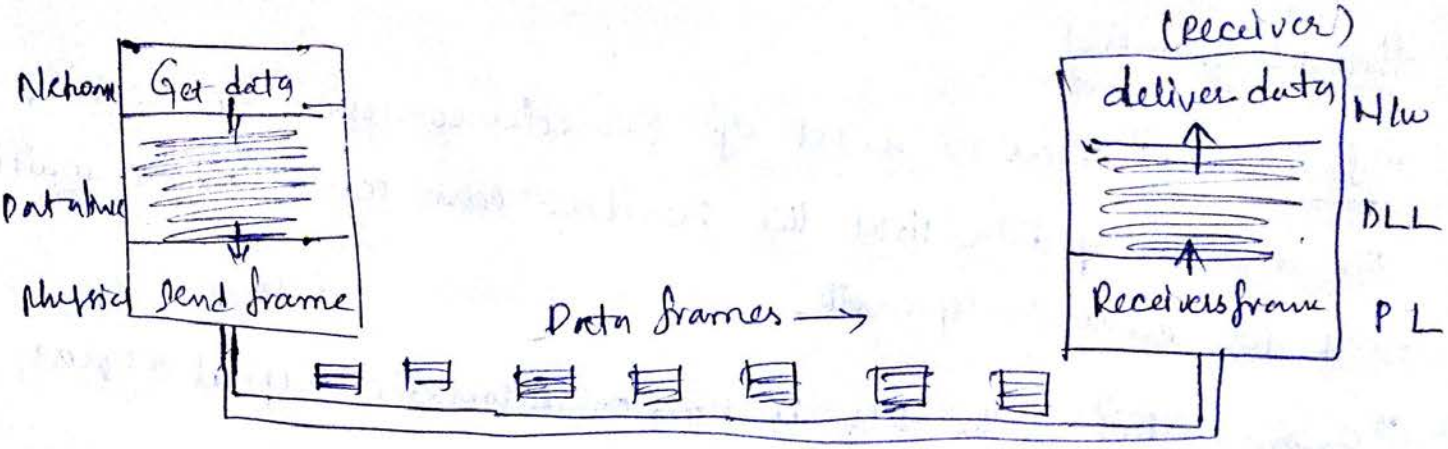
* **Protocols:**
  - → For Noiseless channels (Simplest, stop-and-wait)
  - → For Noisy channels
    - → Stop-& wait ARQ
    - → Go-Back-N-ARQ
    - → Selective Repeat ARQ.

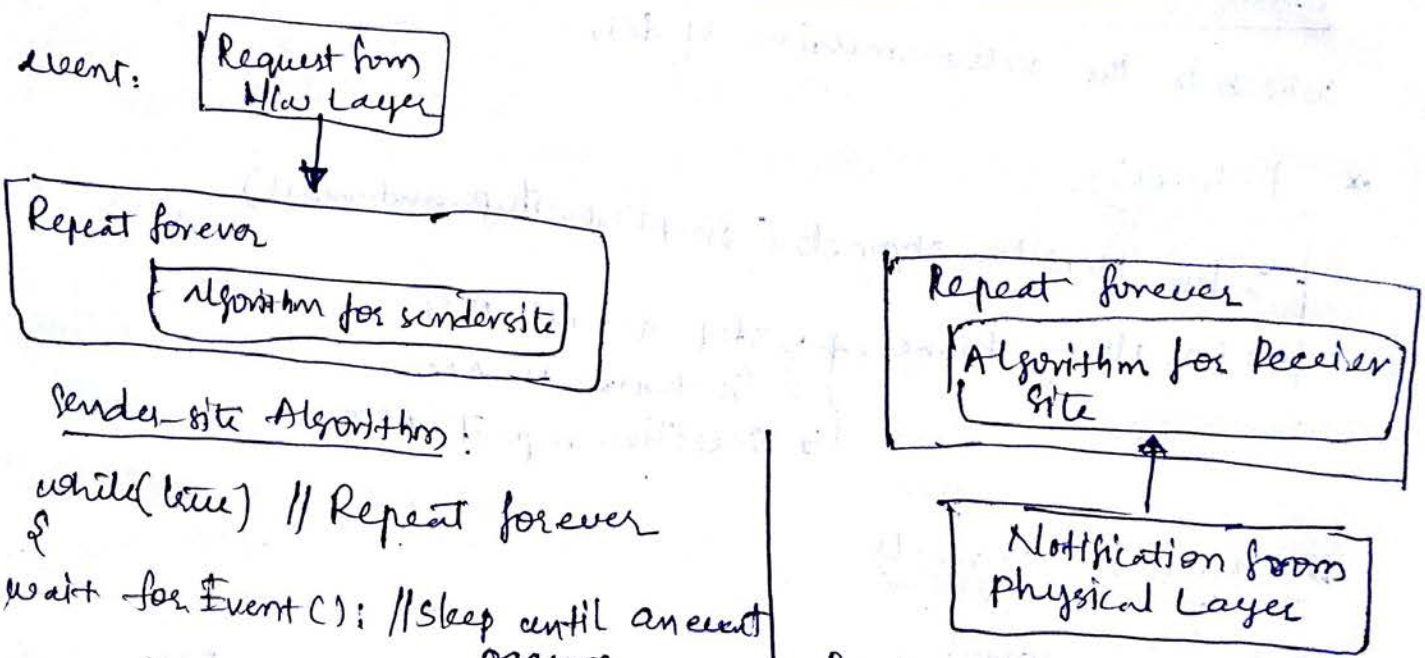
(*) Noiseless Channels

i) Simple protocol:

It is the protocol, which has no flow/error control.
It is unidirectional protocol in which data frames are travelling in only one direction, from sender to receiver.
The DLL of the receiver immediately removes the header from the frame & hands the packet to its N/w layer.

The design of the simplest protocol with no flow/error ctrl is:





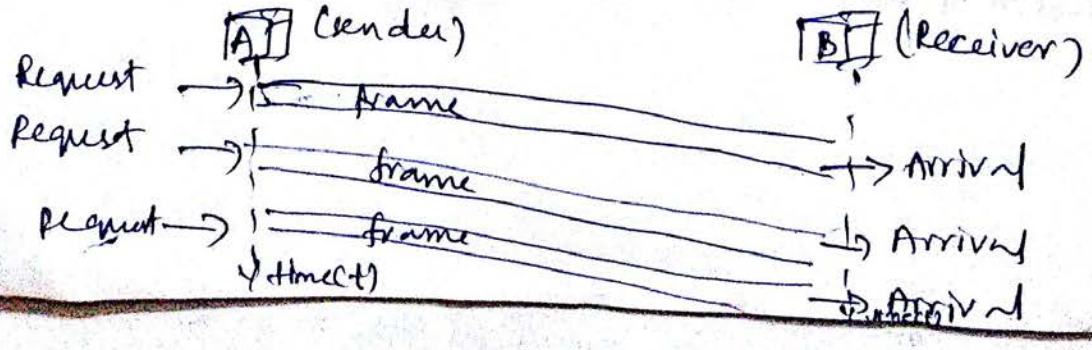
Sender-site Algorithm:

```
while(true) // Repeat forever
{
wait for Event(); //sleep until an event occurs
if (Event (Request ToSend)) //there is a packet to send
 {
Get Data();
make Frame();
send Frame(); // send Frame
}
```

Receiver site Algorithm:

```
while(true)
 {
wait for Event();
If (Event (Arrival Notification))
 {
Receive frame();
extract Data();
Deliver Data();
 }
}
```

⇒ the Flow-diagram is shown as:



1) Sender-site algorithm for stop-and-wait protocol:

```
while (true) // Repeat forever
CanSend = true // allow the first frame to go.
{
wait forEvent(); // sleep until an event occurs
if (Event (RequestToSend) AND CanSend)
  {
  GetData();
  makeFrame();
  sendFrame(); // send the data frame
  CanSend = false // cannot send until Ack arrives.
  }
wait ForEvent(); // sleep until an event occurs
if (Event (ArrivalNotification) // An Ack has arrived.
  {
  ReceiveFrame(); // Receive the Ack Frame
  CanSend = true;
  }
}
```

Receiver site - algorithm for stop-and-wait protocol.

```
while(true) // Repeat forever
{
wait ForEvent(); // sleep until an event occurs.
if (Event (ArrivalNotification)) // Data frame arrives
  {
  ReceiveFrame();
  ExtractData();
  DeliverData(); // deliver data to N.L
  SendFrame(); // Send an Ack frame
  }
}
```

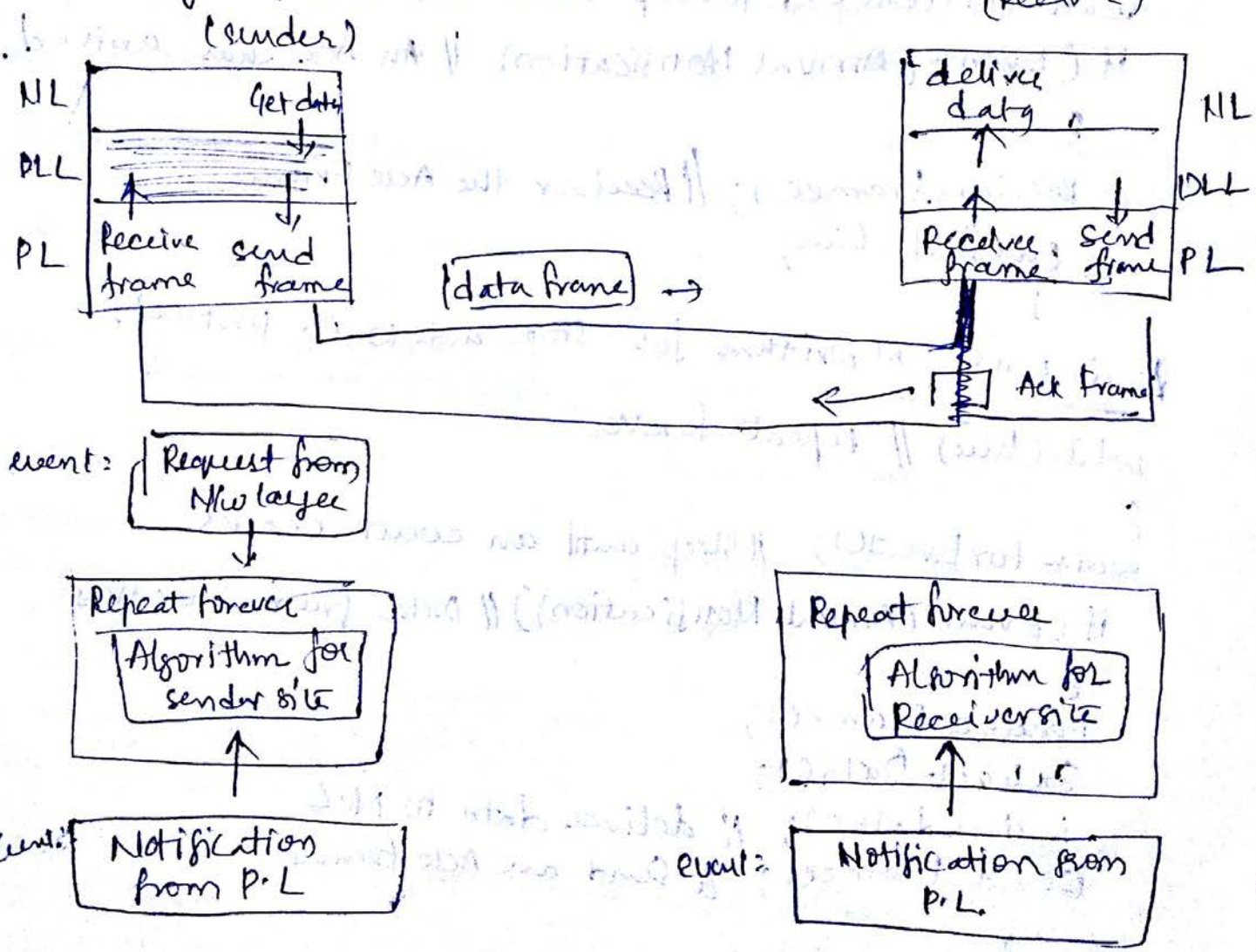
ii) Stop-and wait Protocol:

When the sender keeps on sending the frames, & the receiver is slow/ having many other frames to be received. In this case, the frames may be discarded / denial of service

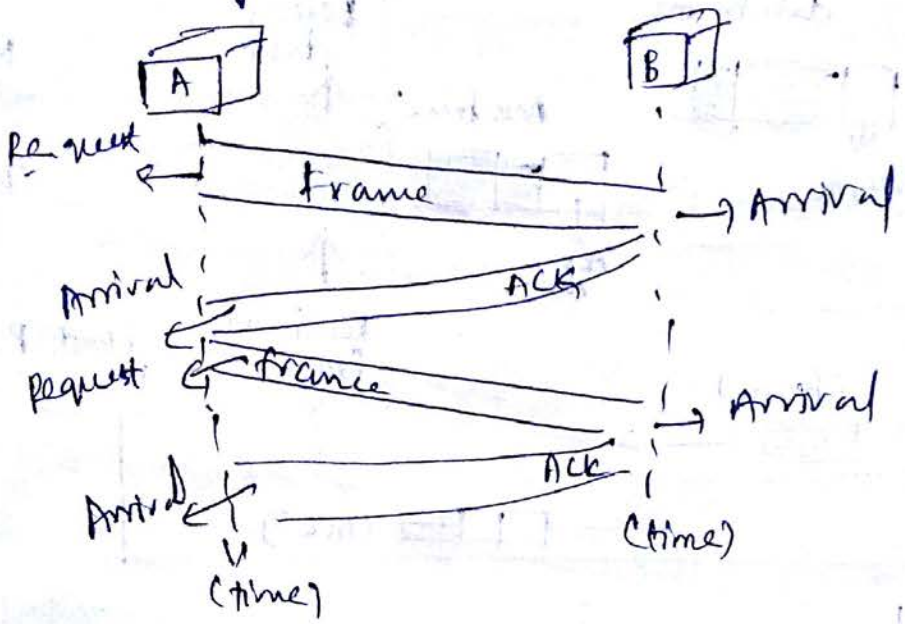
↳ To prevent receiver from over-helmed, stop-and-wait protocol is used, because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), & then sends the next frame.

↳ Nothing but the data flow is controlled using stop-&-wait protocol,

↳ Design of stop & wait protocol is



↳ flow diagram is shown as:



Noisy channels:

iii) Stop-& wait ARQ (Automatic Repeat Request)

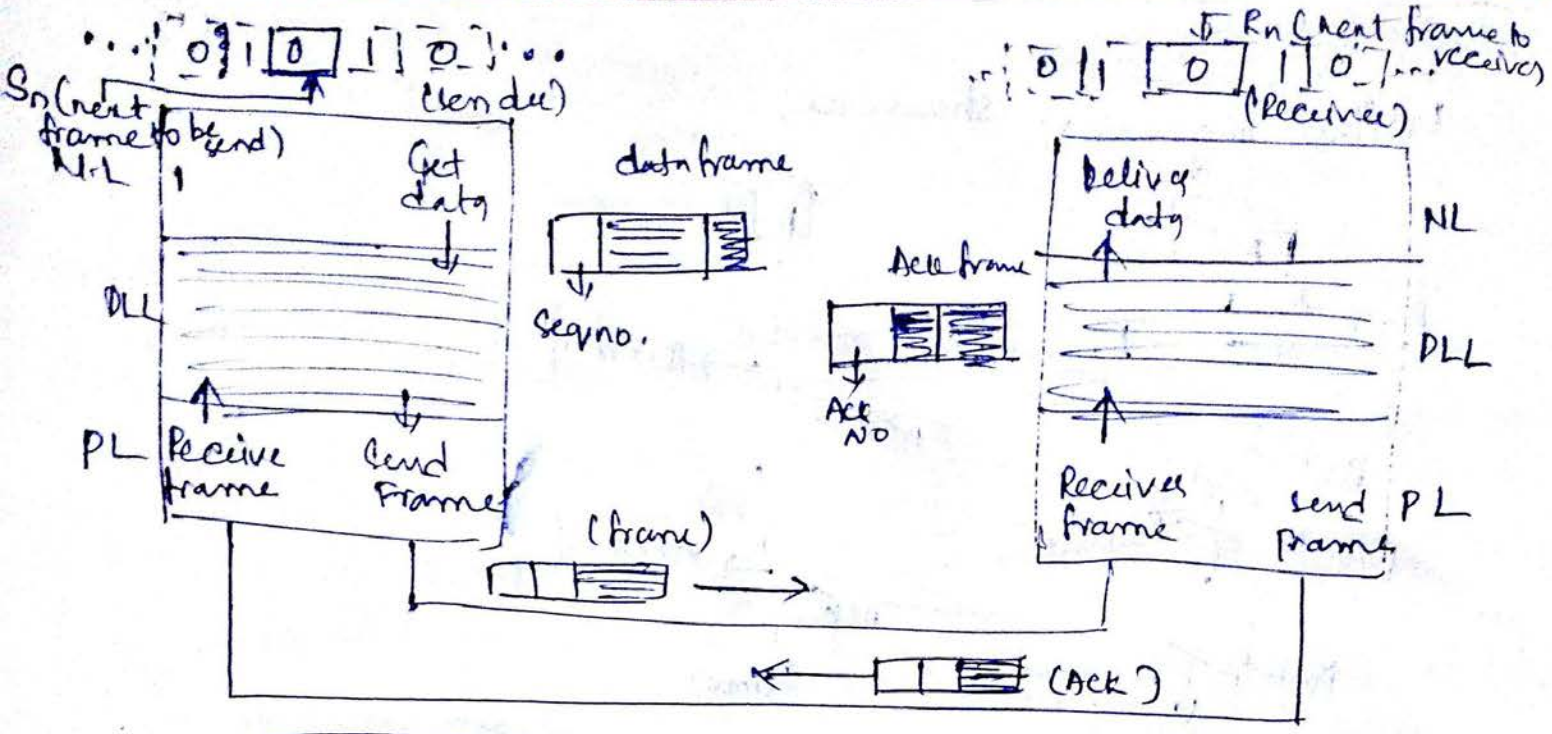
For simple stop & wait protocol (Noiseless), when we add error control mechanisms, then it becomes "Stop-&-wait ARQ"

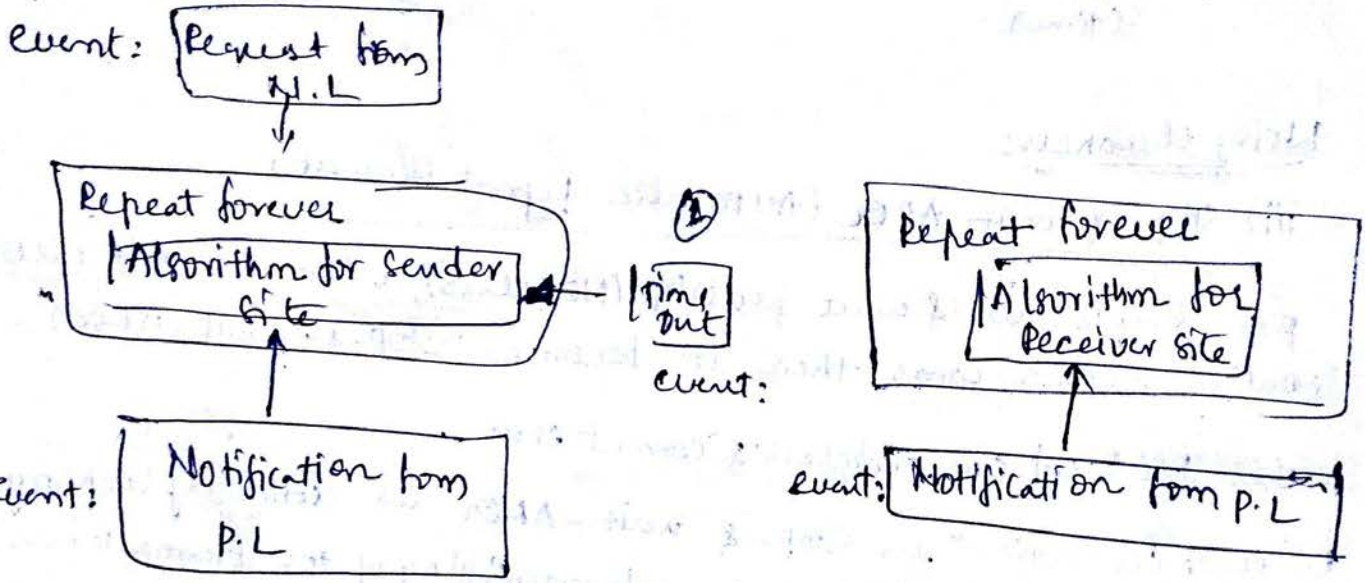
↳ This protocol can detect & correct errors.

↳ "error correction" in Stop-& wait-ARQ is done by keeping a copy of the sent frame & retransmitting of the frame when the timer expires.

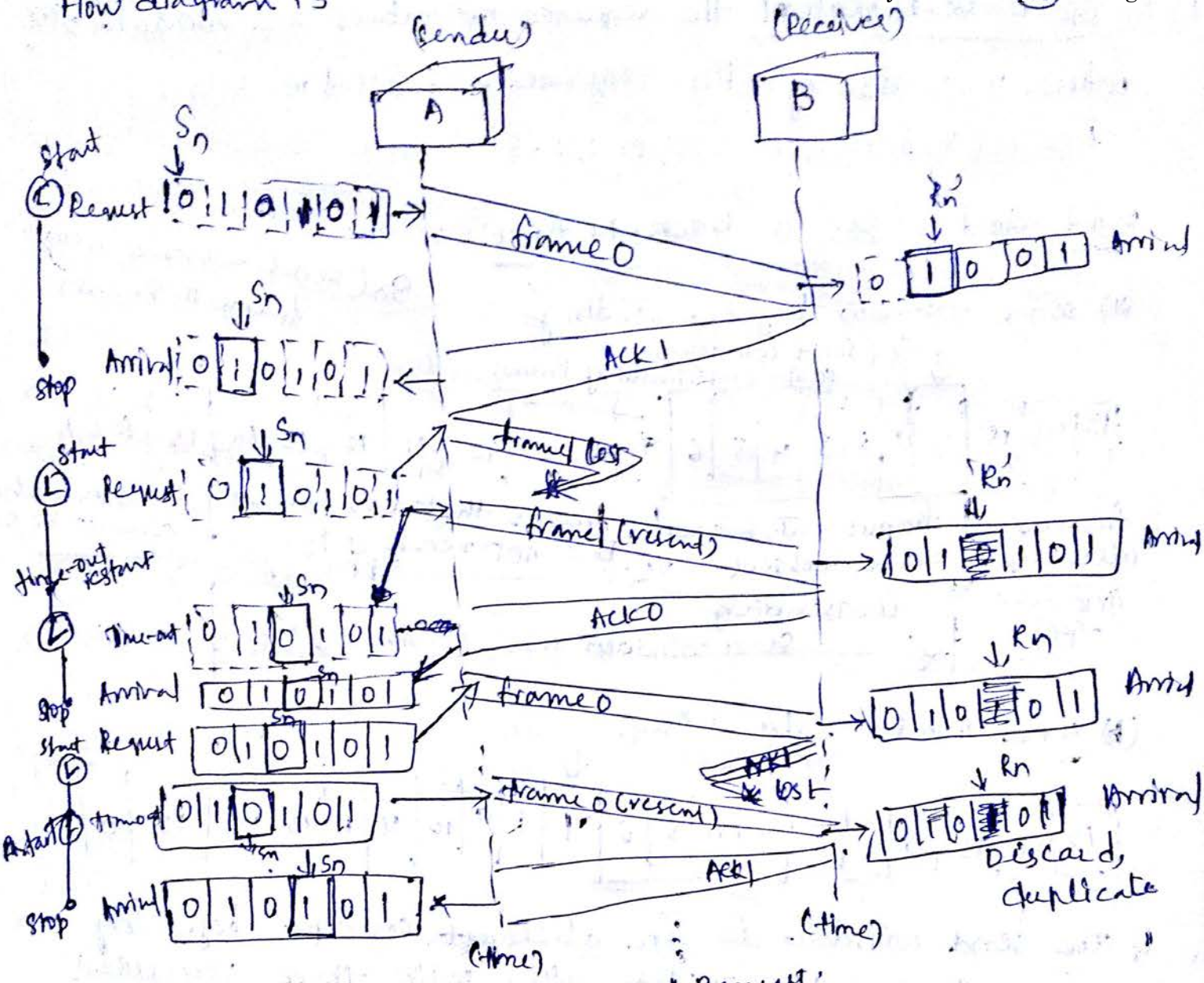
↳ In Stop-and-wait ARQ, we use sequence numbers to no-of frames. The sequence no's are based on modulo-2-arithemetatic.

↳ In stop-& wait ARQ, the acknowledgment no. always announces in modulo-2 arithematic the sequence number of the next frame expected.





Flow diagram is



iv) Go-Back-N Automatic Repeat Request:

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition instead of waiting for acknowledgement.

↳ So using Go-Back-N ARQ, we can achieve this:

↳ Using this protocol we can send several frames before receiving acknowledgements, we keep a copy of these frames until the acknowledgements arrive.

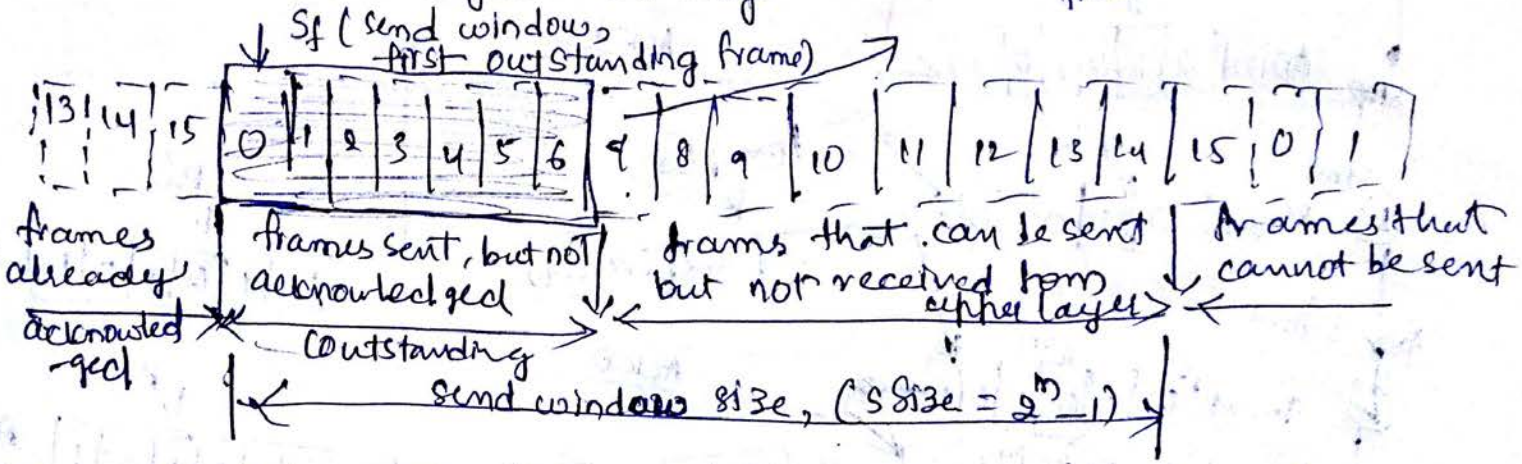
↳ Frames from sending station are numbered sequentially.

→ In Go-Back-N protocol, the sequence numbers are modulo 2^m, where m = size of the sequence no's field in bits.

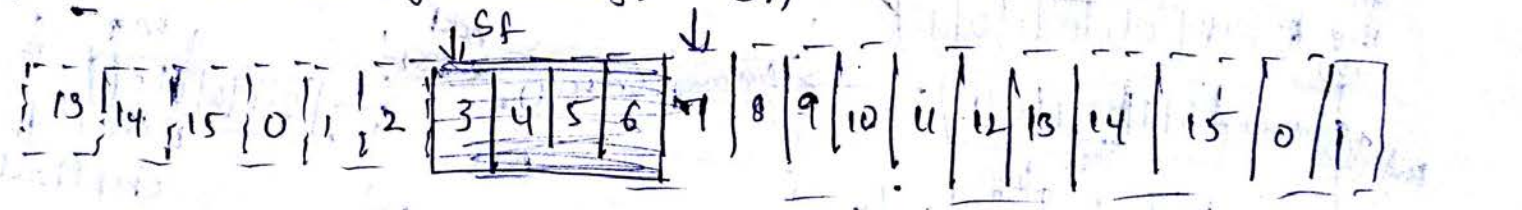
i.e 0, 1, 2, 3, 4, 5, 6, 7 ..., 15, 0, 1, 2, 3 ... 15

Send window for Go-Back-N ARQ:

(a) send window before sliding



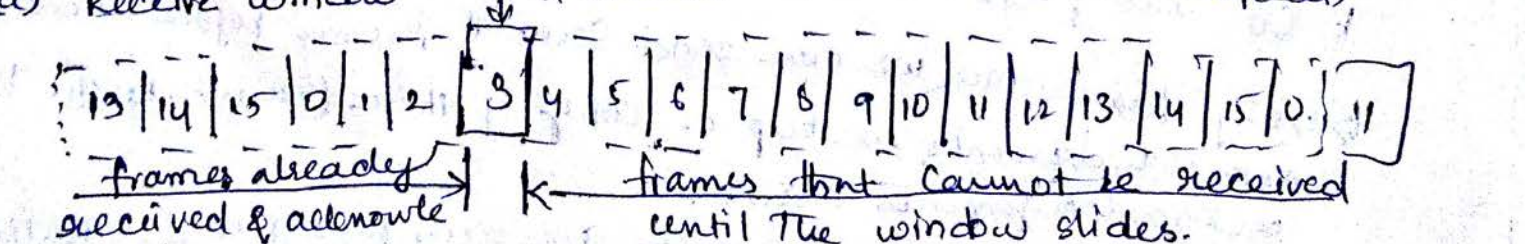
(b) send window after sliding:



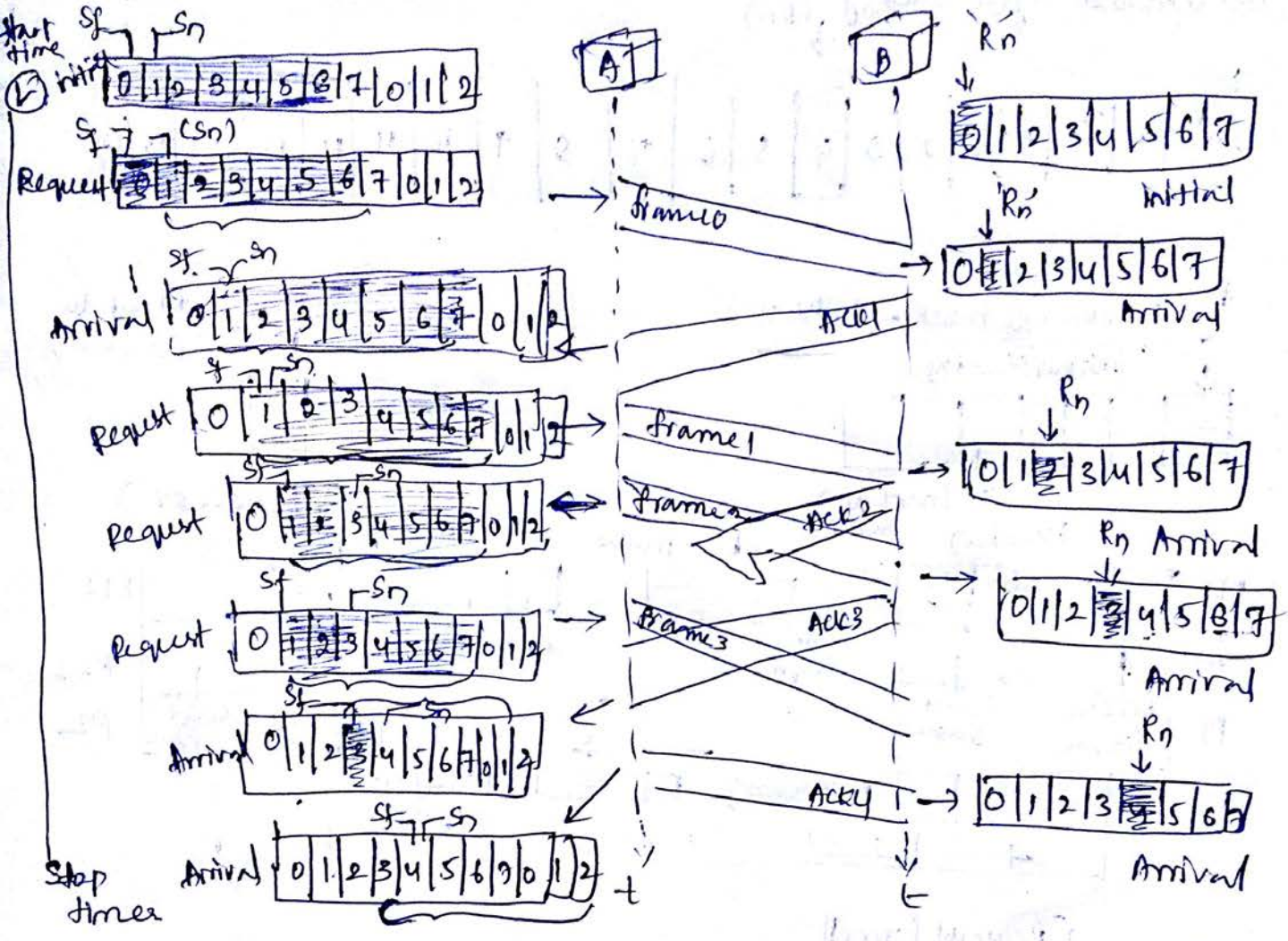
- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n & S_{size}
- The send window can slide one/more slots when a valid acknowledgement arrives.

Receive window for Go-Back-N ARQ.

(a) Receive window



Flow diagram

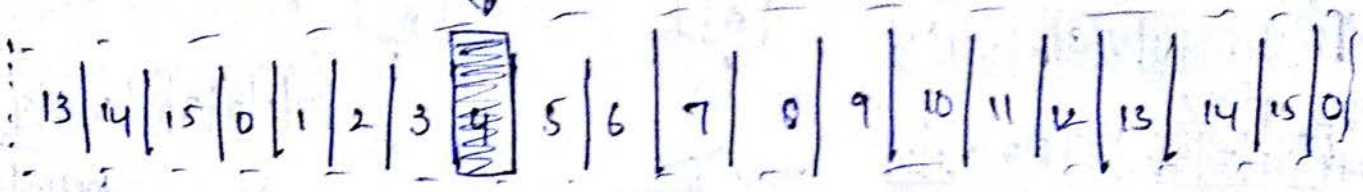


ii) Selective-Repeat Automatic Repeat Request:

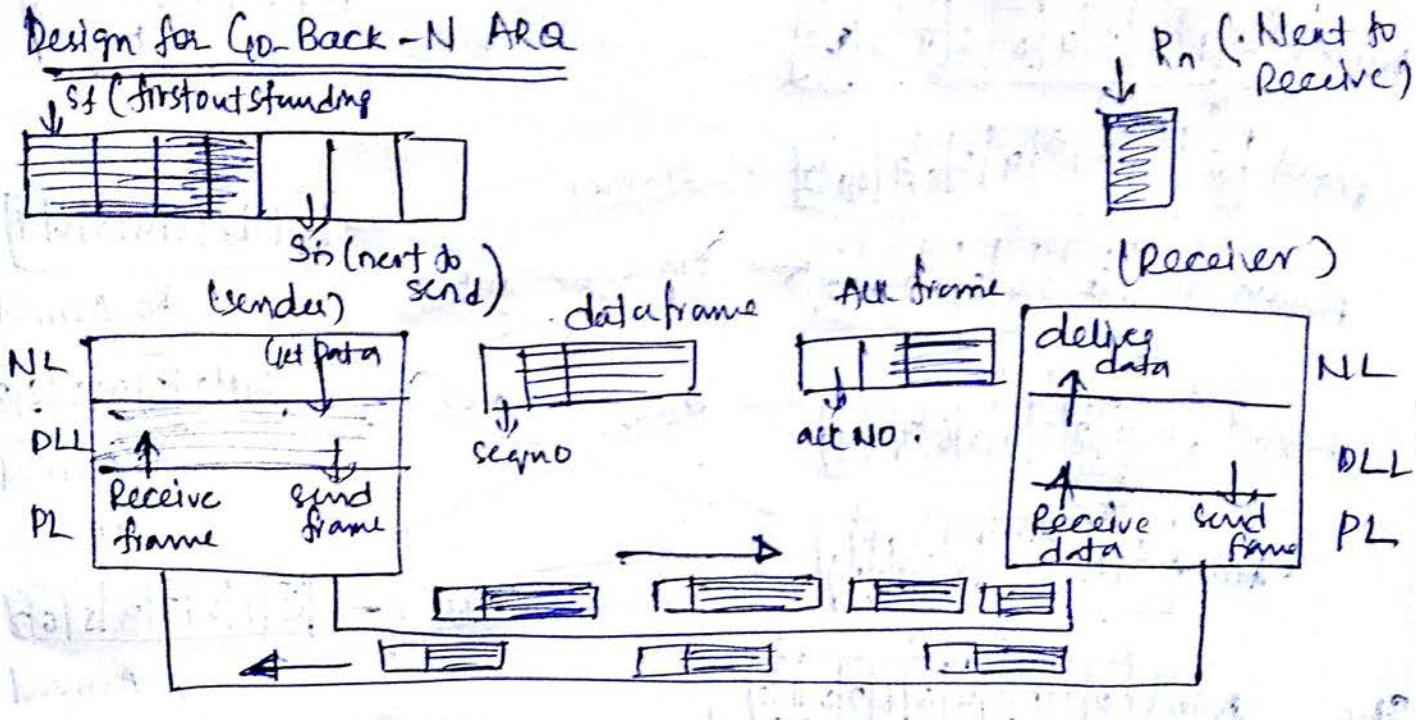
Go-Back-N ARQ is very inefficient for a noisy link.

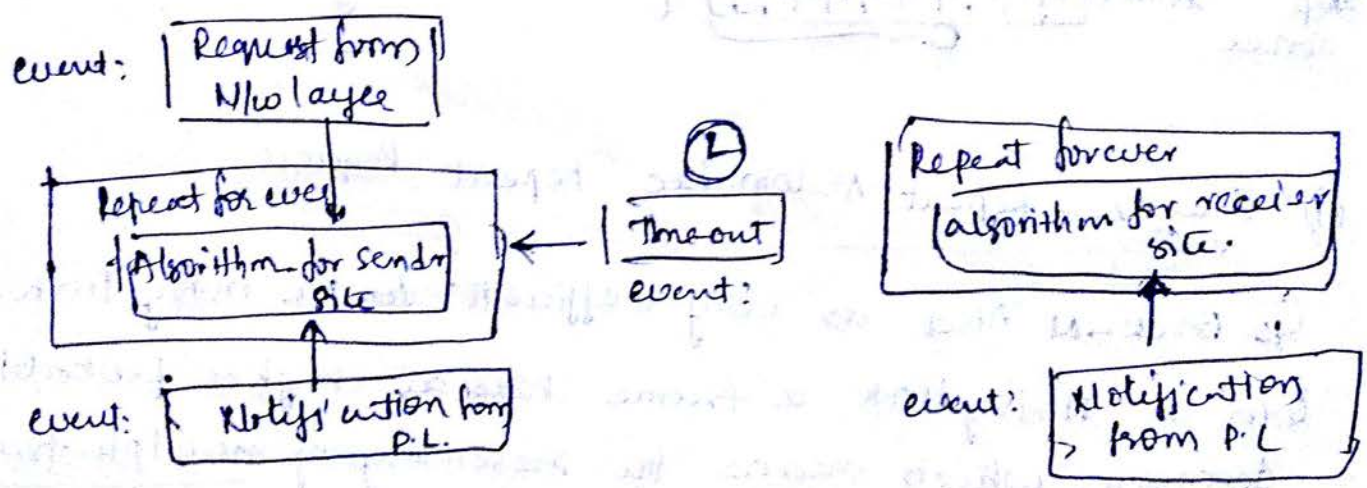
- In a Noisy link, a frame has a higher probability of damage, which means the resending of multiple frames.
- This resending uses up the b w & slows down the transmission.
- For noisy links, there is another mechanism that doesn't resend N Frames when just one frame is damaged; only the damaged frame is resent.

(b) window after sliding (Rn)



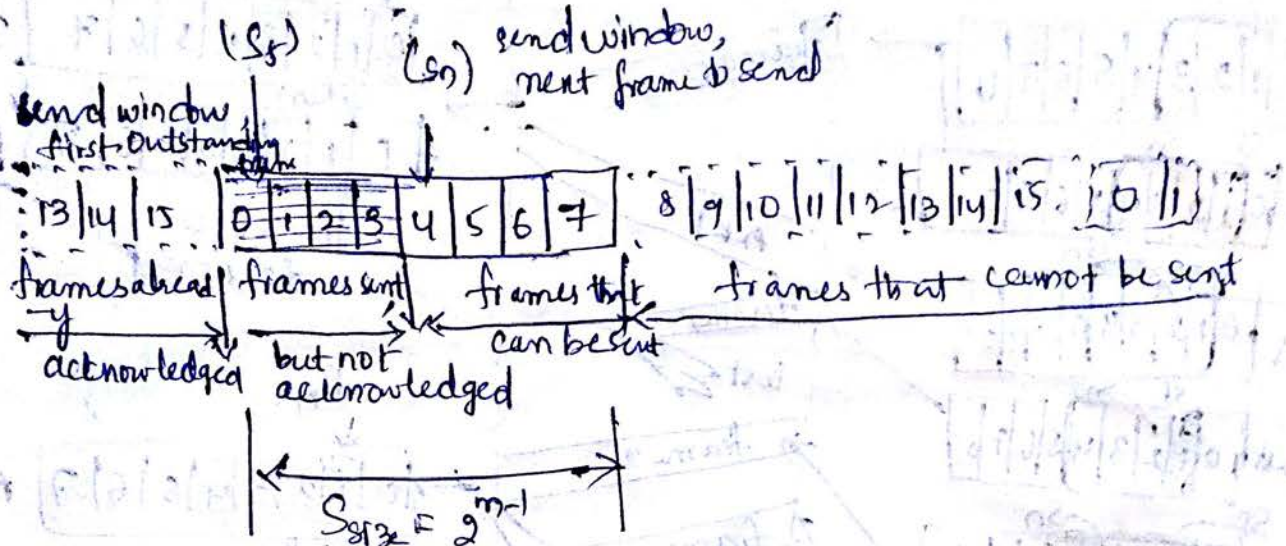
Design for Go-Back-N ARQ



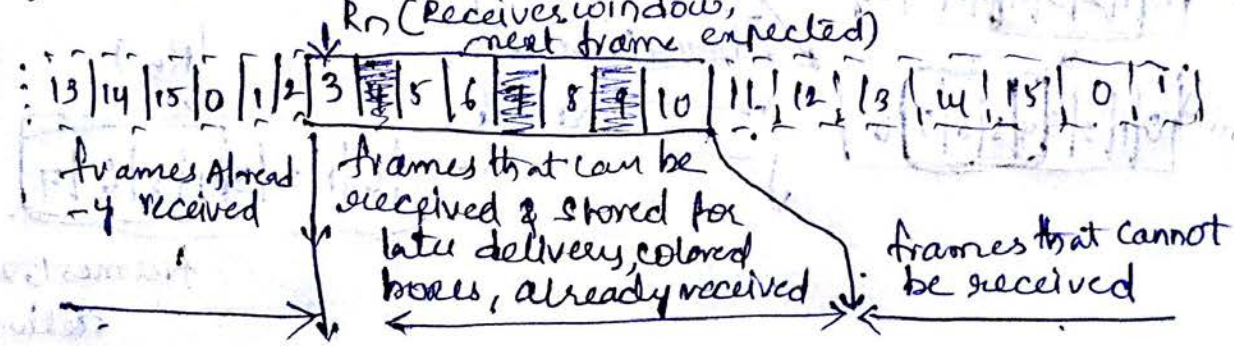


This mechanism is called selective Repeat (ARQ)

⇒ send window of selective Repeat ARQ is



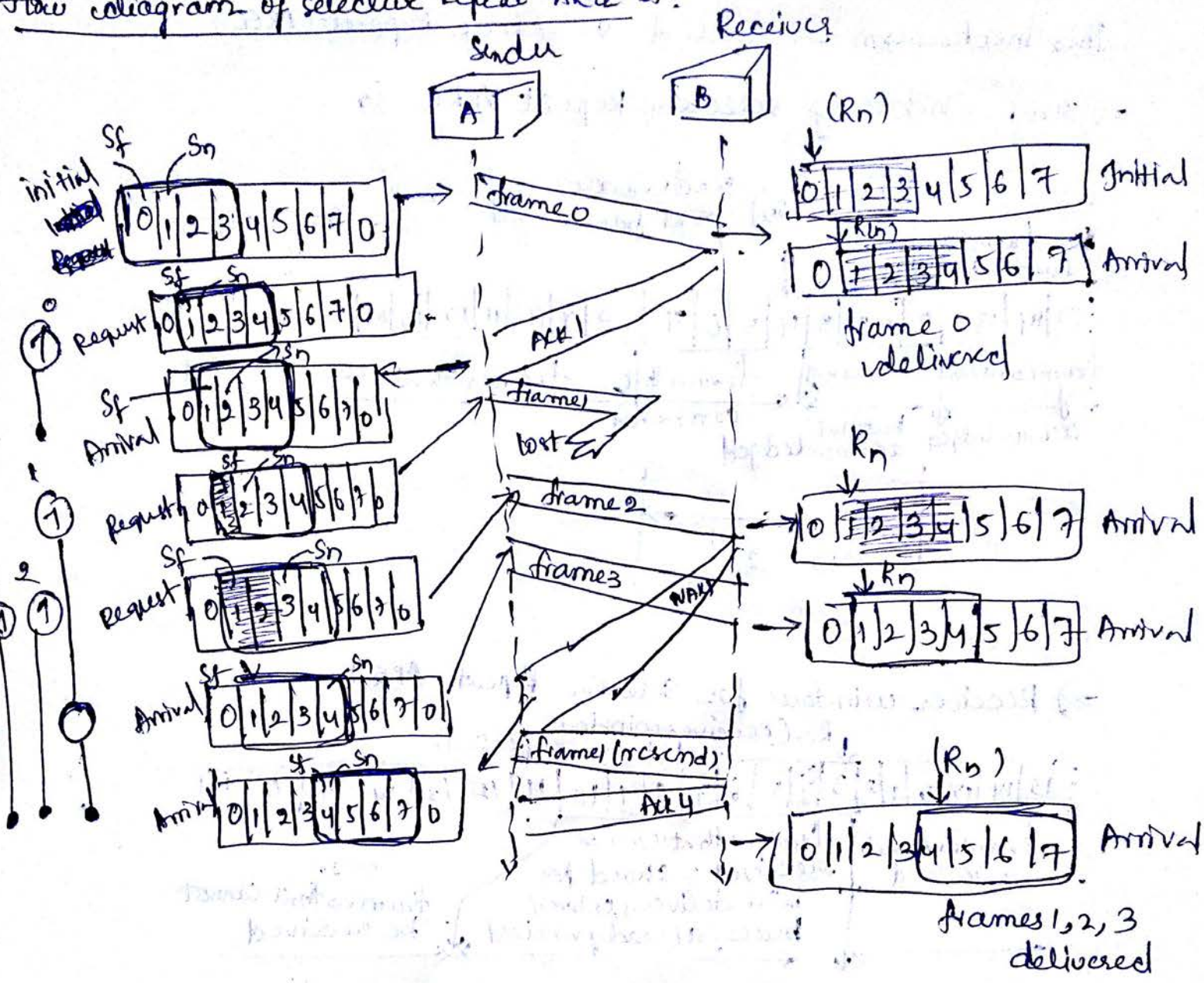
⇒ Receiver window for selective Repeat ARQ is:



⇒ The design of selective Repeat ARQ is similar to Go-Back-N ARQ:

(refer to the diagram of Go-Back-N ARQ)

Flow diagram of selective Repeat ARQ is:

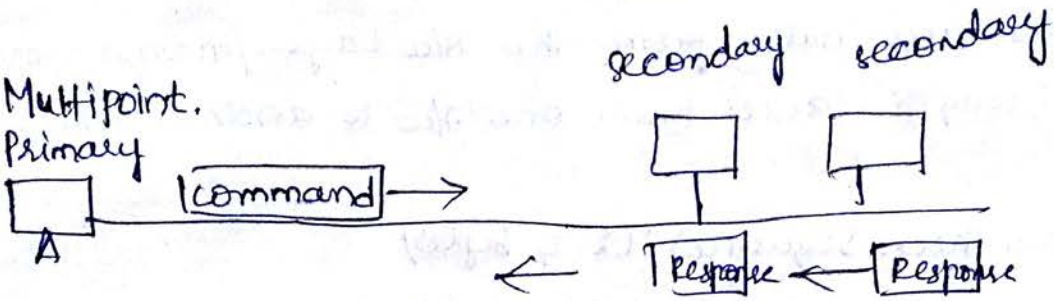


* HDLC (High-level Data link control)

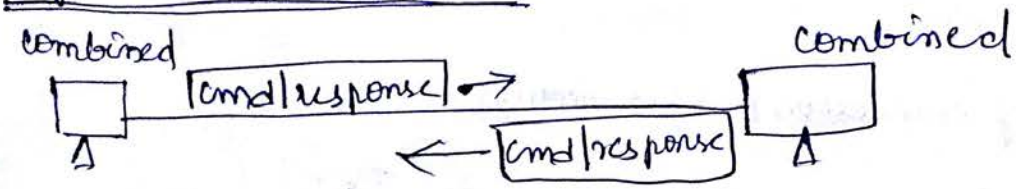
- HDLC is a bit oriented protocol for communication over point-to-point & multipoint links.
- HDLC provides two common transfer modes:
  1) Normal Response Mode (NRM) → Point-to-point / multi-point
  2) Asynchronous balanced mode (ABM)

(a) point-to-point

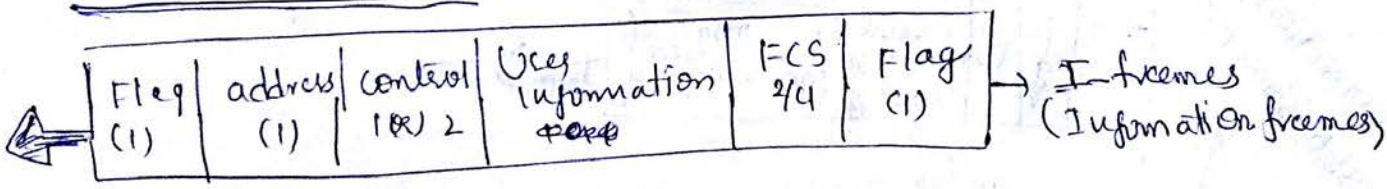
b) Multipoint.

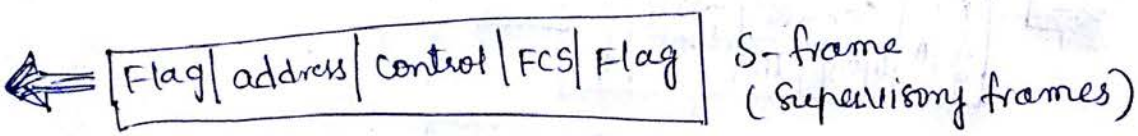


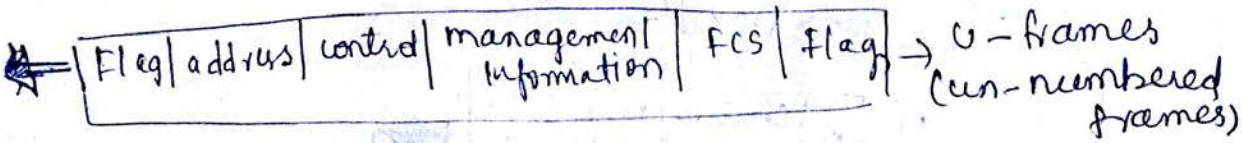
asynchronous Balanced mode



frame format of HDLC:







Flag-field: (8 bit)

Flag bit recognizes the start & end of a frame & serves as a synchronization pattern for the receiver.

address field: (1 byte or several bytes long)

contains to address (source) & from address (destination)

Control field: (1(or) 2 bytes)

Used for flow & control

Information field: (User/management)

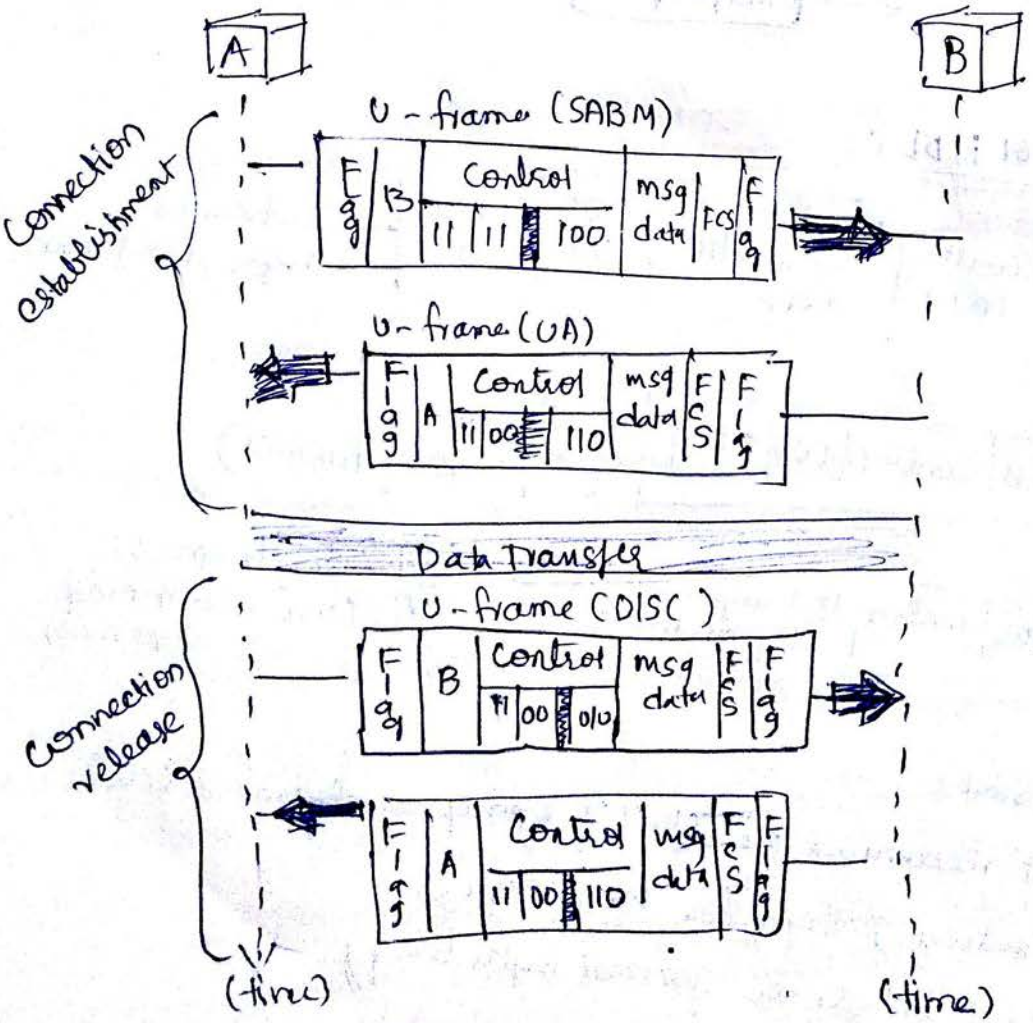
Contains the user data from the N/w Layer/management information. (length carries from one n/w to another N/w).

FCS (Frame check Sequence): (2-4 bytes)

It is error detection field (ITU-T CRC)

example:

Example of connection & disconnection:



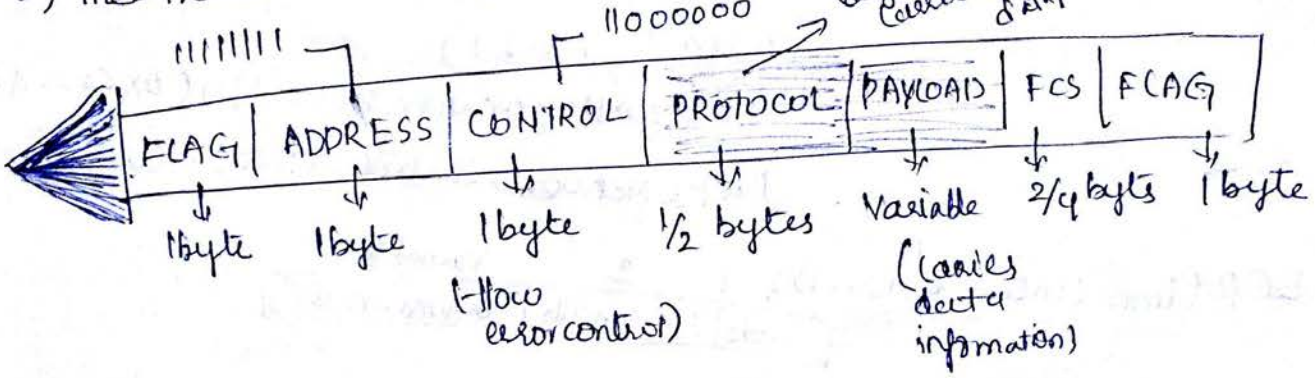
Point-to-Point Protocol:

↳ PPP is a byte-oriented protocol.

↳ One of the common protocols for point-to-point access is the point-to-point protocol.

(Today many of internet users who need to connect their home computers to the server of an internet services provider use PPP)

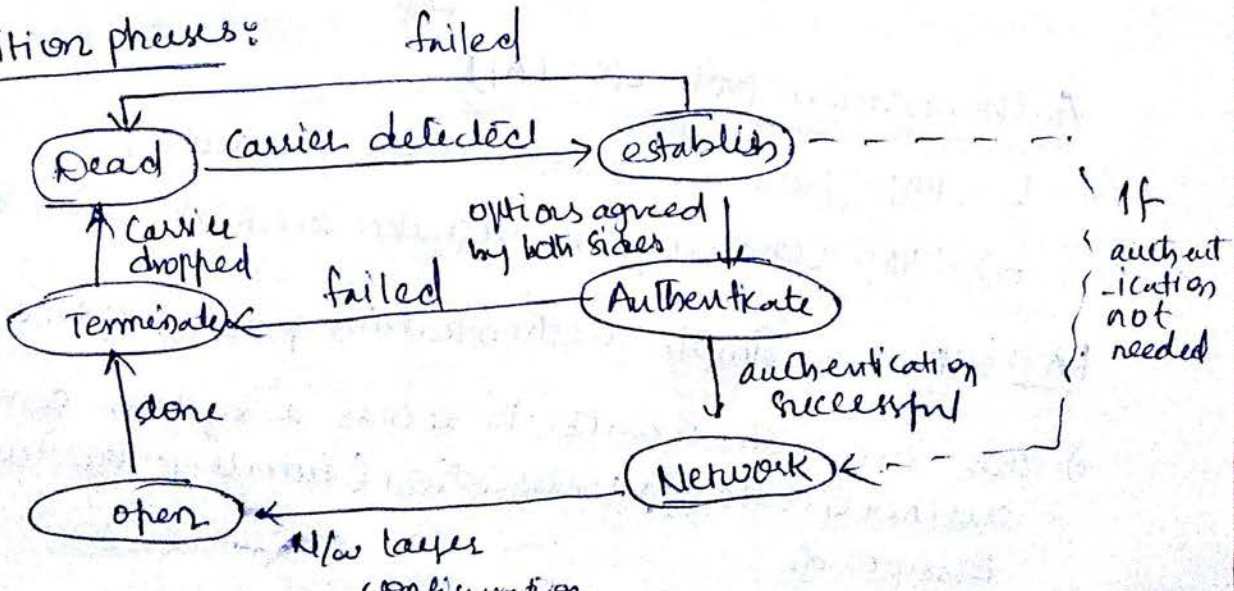
→ The frame format of PPP-protocol is



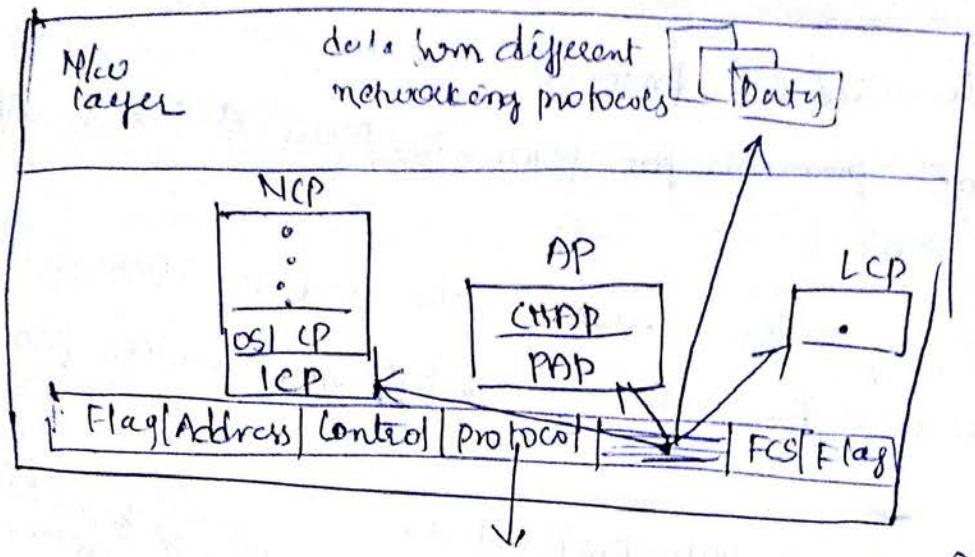
☆ Note:

PPP is a byte-oriented protocol, using byte stuffing with the escape byte 01111101.

Transition phases:



Multiplexing in PPP:



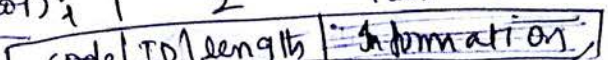
LCP (link control protocol) → (0xC021..)

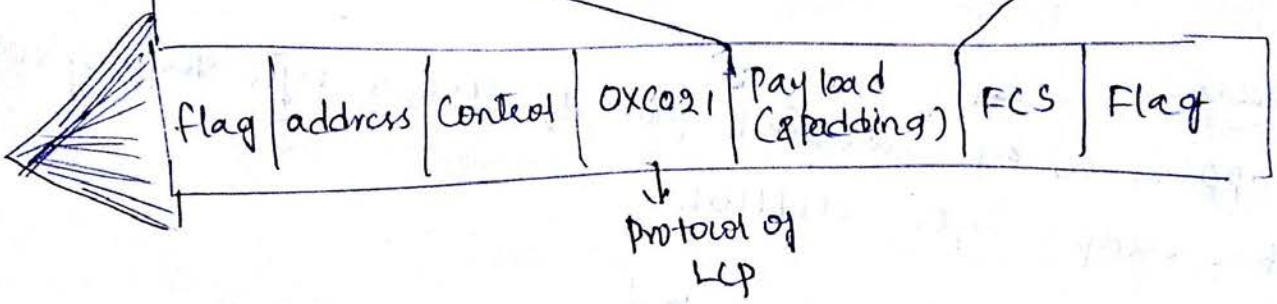
Data (0x8021)

AP (authentication protocol) → (0xC023 & 0xC223)

NCP (Network control protocol) → (0x8021 & ...)

LCP (link control protocol):





Authentication protocols: (AP)

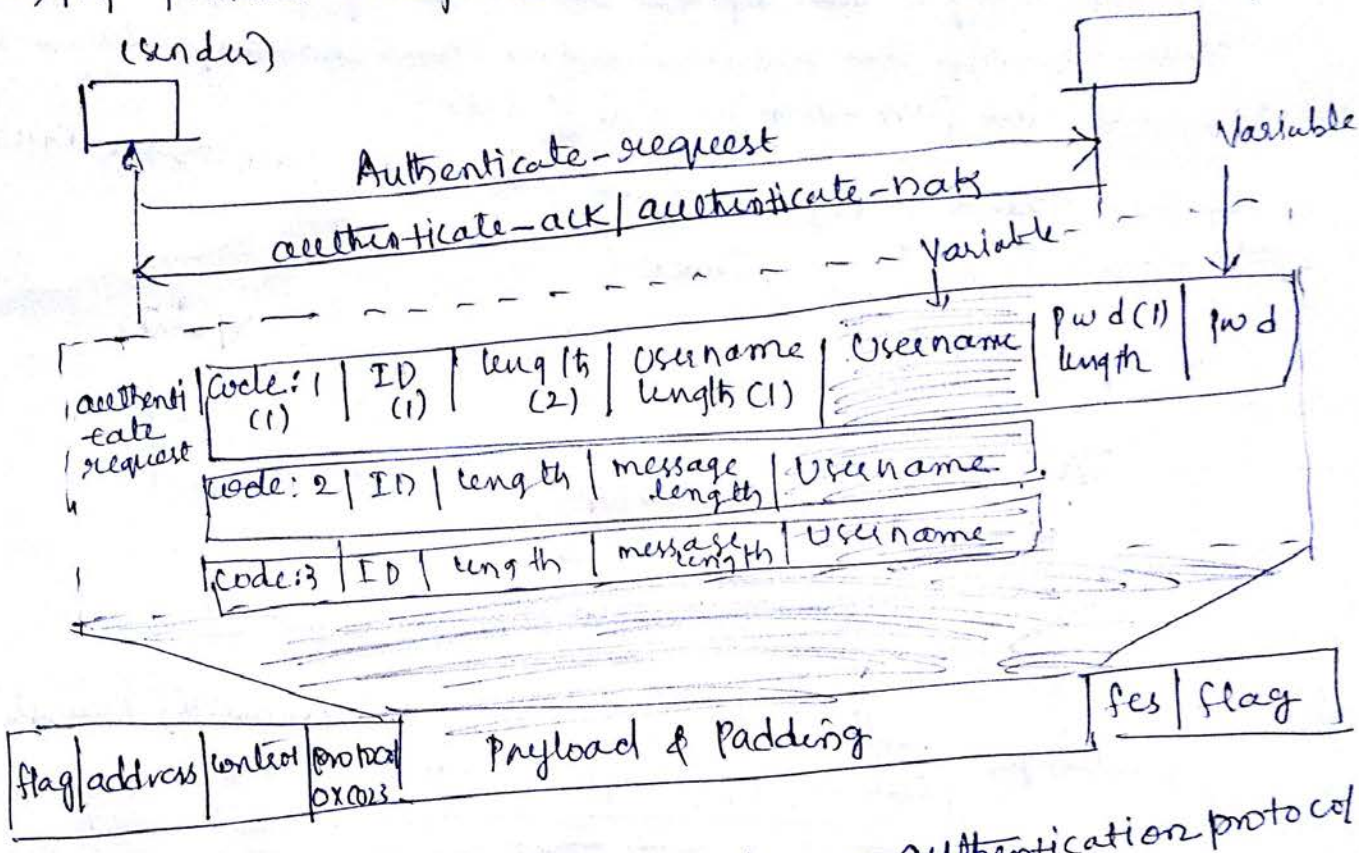
- PAP (password authentication protocols)
- CHAP (challenge handshake authentication protocols)

PAP: It is a simple authentication protocol of two steps:

1) The user who wants to access a system sends an authentication identification (usually the username) & a password.

2) the system checks the validity of the identification & password and either accepts/denies connection

⇒ PAP packets encapsulated in a PPP frame is shown as:



CHAP: It is a three-way hand shaking authentication protocol that provides greater security than PAP.

In this method, password is kept secret, it is never sent online.

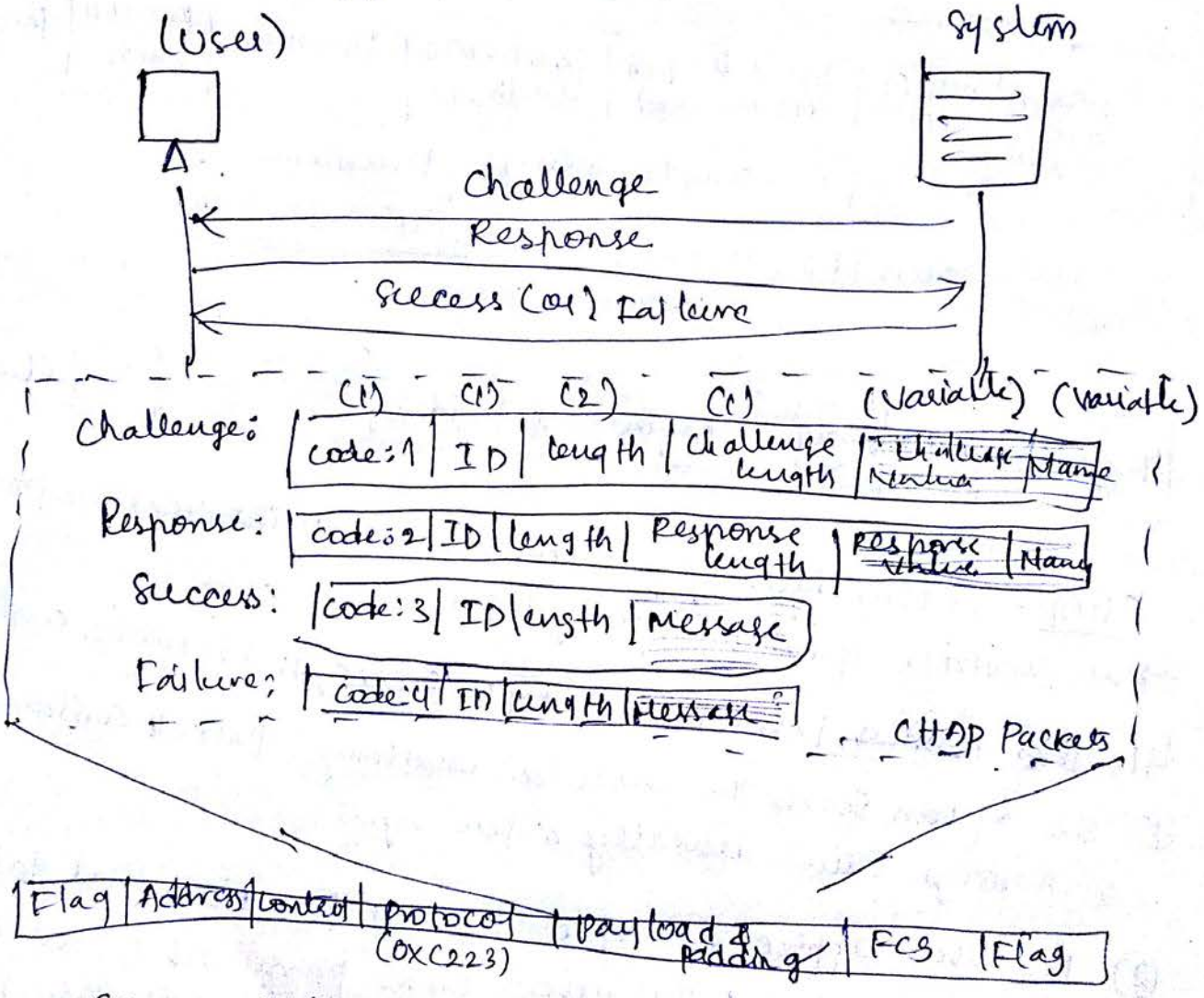
(1) the system sends the user a challenge packet containing a challenge value, usually a few bytes.

(2) The User applies a pre-defined function that takes the challenge value & the user's own password & creates a result. The user sends the result in the response packet to the systems.

(3) the system does the same. It applies the same function to the password of the user (known to the systems) &

& the challenge value to create a result. If the result created is the same as the result sent in the response packet, access is granted, otherwise, it is denied. CHAP is more secure than PAP, especially if the system continuously changes the challenge value. Even if the intruder learns the challenge value & the result, the password is still secret.

↳ The following figure shows how CHAP packets are encapsulated in a PPP frame.



(*) IPCP (Internet protocol control protocol)