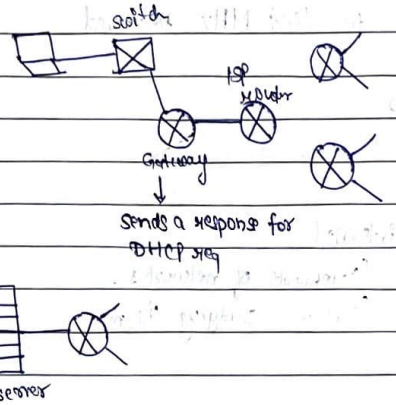
Look at Network adaptors in laptop

29/1/24.

Application	HTTP, DHCP, DNS, FTP
Transport	TCP/UPD
Network	IP, RIP, BGP, ARP
Data Link	MAC
Physical	

TCP/IP stack.



→ If your machine has 2 n/w adaptors → machine will have 2 IP addresses → MAC Addr.
↳ Given by manufac. of device
↳ Called Physical address.

MAC addr → 48 bit long (Hexadecimal format)
Looks like: 1G:FA:3B:8C:1G:15 → 6 bytes or 48 bits

⇒ If there are 3 Network Interface cards → machine will have 3 MAC Addrs.

⇒ For broadcasting a message to all the systems:
we use: FF:FF:FF:FF:FF:FF

⇒ IP Addr v/s MAC Addr
- Name v/s Institute Id
  - MAC
  - IP ↳ School Id, Clg Id, Company Id

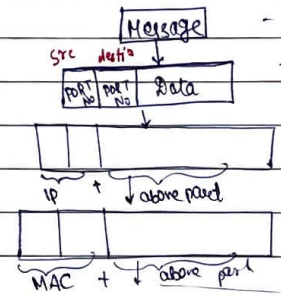
⇒ IP Addr changes with change in network connected or even change if used for long time with same network.
But MAC Addr always remains same

* 68.44.5.1 → 32 bit long. → 0–255 range for each [??]

⇒ PORT No:

https://www.google.com
↳ Port no: 443. (Fixed)

⇒



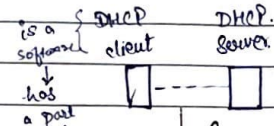
Book Pg 78 (5e)

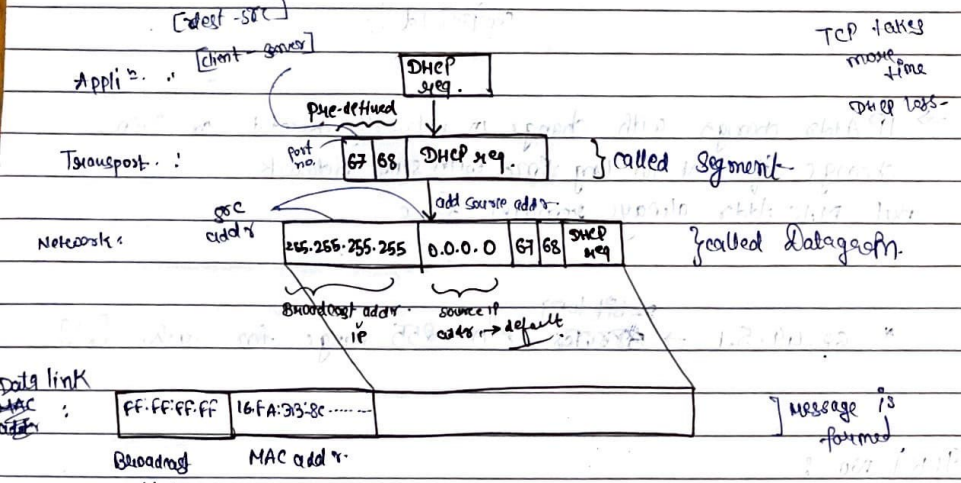
DHCP :

Laptop wants to connect to cisco.google.com using ethernet

↓

DHCP is required.





TCP takes more time

DHCP less

→ This all is happening in route

→

Port no : unique no. given to appli^n.

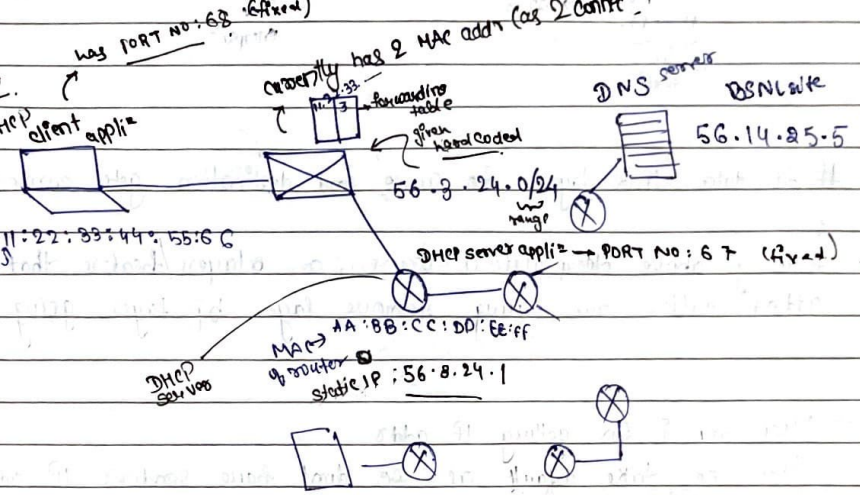
(1.)

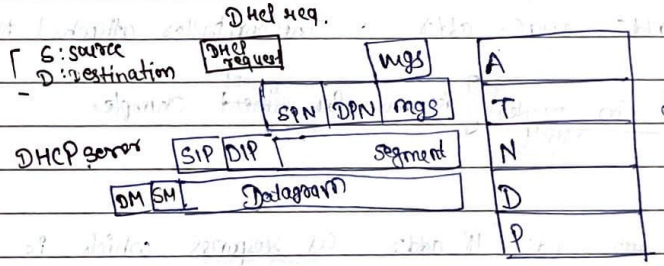
68.86.5.102 → the IP that the client should be assigned. This is selected from the given by ISP port

68.86.5.1 → of router.

x.y.z.u

31/1/24



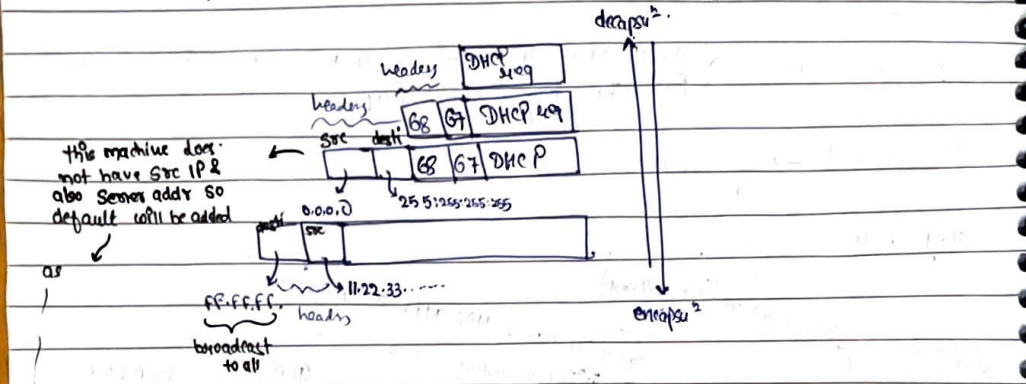


* We get MAC addr at NIC of each device
* IP addr is given by software by Network
* Server at which appli^n is running is called DHCP server.
* # No of NICs = no. of connn^s device (router) has. = No of MAC addrs
DHCP servers provides IP addrs dynamically
DNS server : Stores domain names

Adding addressing/headers → encapsulation

Removing headers → decapsulation.

Router is layer 3 device ; ↳ upto 3 layer must be present



At data link layer src and destination gets swapped.

Going above decapsulation occurs, so layers/headers that were added earlier are being remove layer by layer going above

↳ DHCP req is for getting IP addr

↳ so we take default as we dont have sender's IP addr at that port (as its what we want), So at network layer in source we put defaut 0.0.0.0 IP addr.

NIC adds mac. addr. or the controllers attached to it

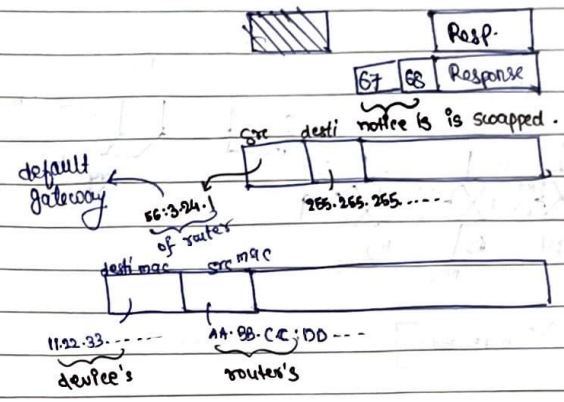
DHCP is in router (Servers itself) } assumed for current example

↳ DHCP sends IP addr as response which is available and adds some additional info

56.3.24.51	→ IP addr available to be returned.
56.3.24.1	→ IP addr of router through req has reached to DNS server
56.14.25.5	→ IP addr of DNS server

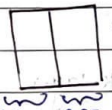
DHCP

Response : (Server → device)



↳ Now device have received DHCP response.

Forwarding Table consists of MAC addr's of Device, Router → of Switch



⇒ DHCP server v/s DNS server (Google it).

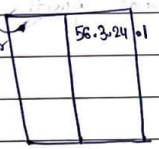
↓ assigns IP addr to device which asks for it.

~~asks~~ gives IP addr of website

⇒ Routing table?

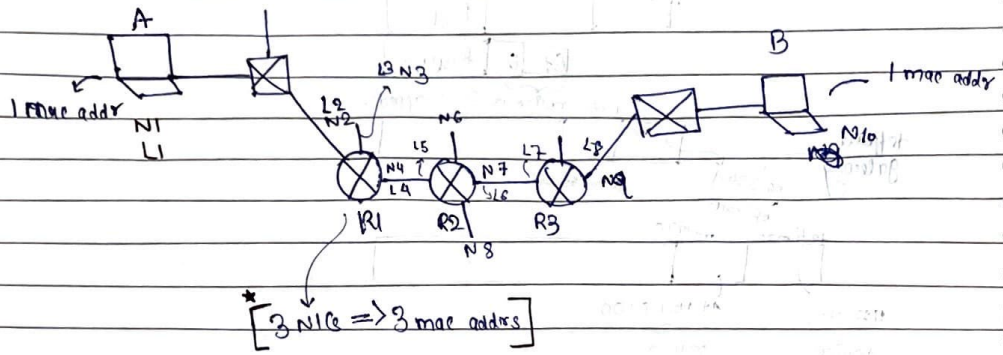
↓ for all addresses outside server

Router's fwd table



2/2/24

Look for this diagram in book.



* [3 NICs ⇒ 3 mac addrs]

How many IP addrs R2 has? ⇒ 4

mac-addresses are also known as link-layer addresses or physical addrs.

⇒ N1 → Network layer addr = IP Addr
L1 → Link layer addr = MAC-addr.

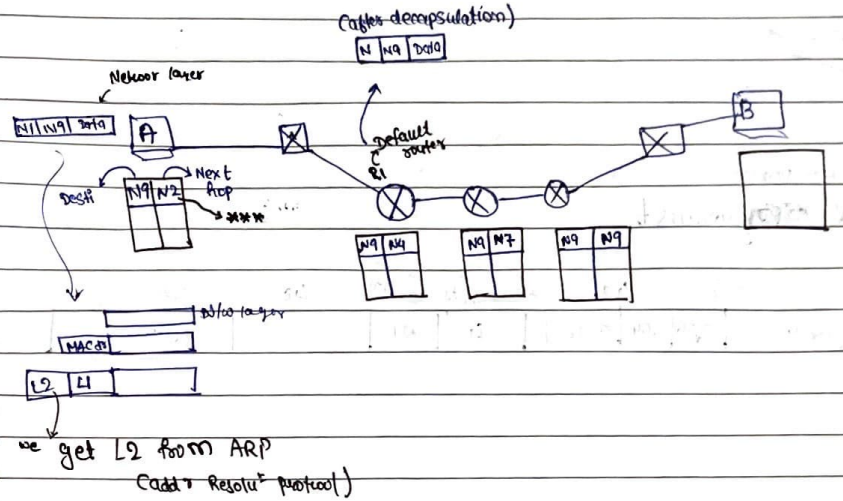
→ Switch - NO & IP mac addr ⇒ do not connect two diff devices
⇒ It only has ports.

*** we got N9 from routing table ⇒ where it came from routing table?
↓
from DHCP response

Routing table
↳ contains entry of next device where data/packet is to be sent

say. N1 → N9 ⇒ at A :

Src	dest
N1	N9	Data



we get L2 from ARP
(Addr Resolution protocol)

at R1 : N9 is check if its equal to current IP or not
if not routing table is checked

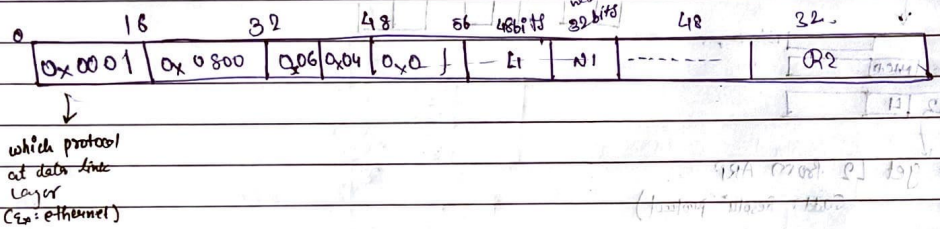
ARP (Address Resolution Protocol):

(1) Takes the addrs → creates packet

Shivay

→ ARP response

ARP request



which protocol at data link layer (e.g.: ethernet)

5/2/23

Data link Layer.

App
Xport
N/w
DL
Phy.

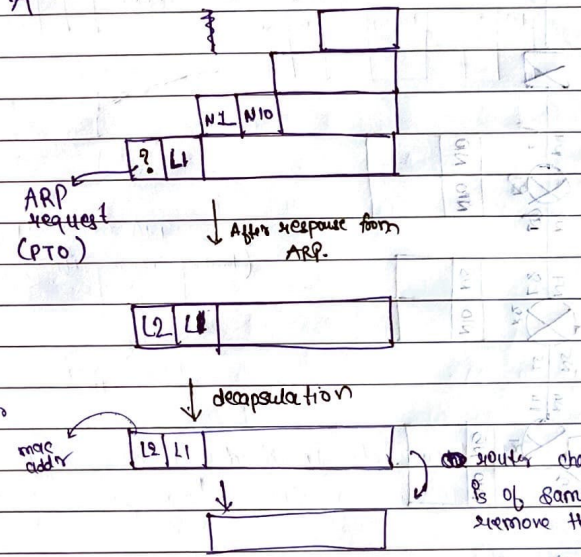
Xport: Port addr & DNS

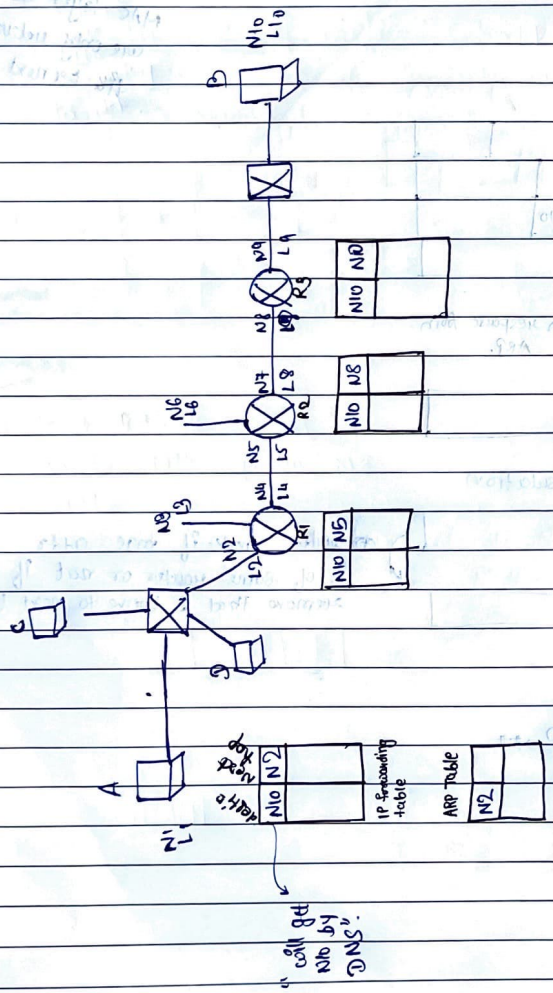
N/w: IP addr & domain name → IP addr

DL: MAC addr ARP: IP → MAC

MAC layer addr are only relevant till next hop.

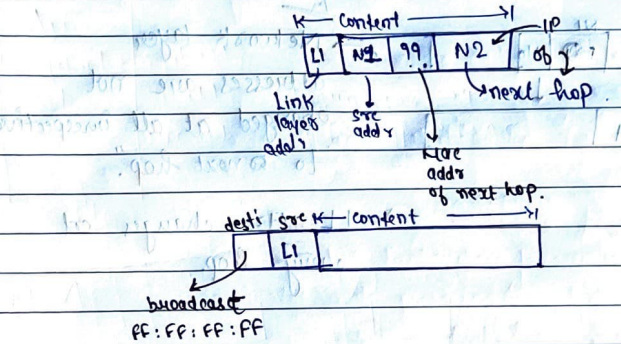
At A:





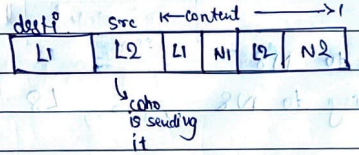
Returns mac addr. of next hop
we give IP addr of " in request.

ARP request sent by A :



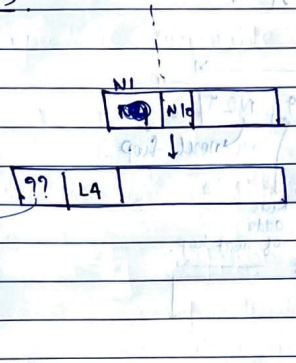
→ So now C & D will check & reject as req was not for them & then Router will send response.

Response



destin² pehle likhte hai due to efficiency reasons

(R1 → R2)



★ → "Network layer addresses are not changed at all irrespective to next hop."

↳ LL addrs changes at every hop.

Listen to reading

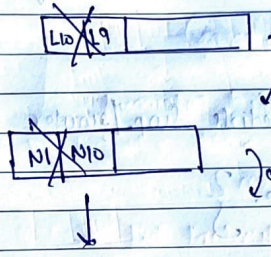
→ check Flod table

→ to go to N10 go to N8

need LL addr corresponding to N8 , i.e., L8

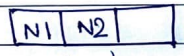
⇒ final

when reached ..



→ What if we dont have data LL ?

↓ if there's no mac, we take IP addr of next hop from flod table & change the destination

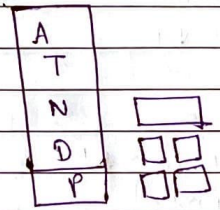


we lost final destina addr, N10 so its not possible.

Data link layer is responsible for creating a frame

(1) Framing :
↳ done at each intermediate hop./router.

⊛ { No. of DLL = No. of interface. }
↳



End device

ii) Error control
iii) Flow control
iv) Congestion control.

7/2/24

Data link Layer.

Error Control

1 0 0 1 1 1 0 0 0

→ During transmission, bits can be changed or corrupted.

- If single bit is changed : Single bit error
- If more than one bit changed : Burst error.
  ↳ Noise is affecting to more than one bit.

Error detection vs Error Correction :

Error detection

⊛ Block coding.
↳ Generic method for error detection

→ Divide message in smaller blocks of size k

block = k + (r) → redundant bit.

code word = k + r

dataword → codeword

check codeword → valid → No error
↳ Invalid → Error detected

→ Take 2 bit data word :
↳ 2^2 combin?

00	000	10	101
01	011	11	110

redundant bit

length of codeword = 3
↳ 8 combi : there 4 out of which 4 is valid

dataword ↳ Codeword

At receiver's end :

→ Received → 000 (No error)
↓
check for valid codeword
↓
After validation
↓
remove redundant bit &
Sends remaining two bits (00)
to send/feed to upper layer

→ If bit error is found (000 → 010)
↳ Received : (Say) 010
↓
Not one of the valid codeword
↓
Not sent to ~~valid~~ upper layer

→ If 2 bits changed (000 → 101)
↳ Received → 101
↓
but is one of the valid even if there exists error
↓
2 bit error can't be detected
↓
To determine : "Hamming distance"

Hamming distance :
↳ How many bits differ in two words given

d(010, 110) = 1

Take XOR → 010
⊕ 110
→ 100

~~Distance~~ Hamming distance b/w any two codewords is 2

If hamming distance = S+1
then No of errors detected = S

* Redundant bit must have some relation with data word, they are not randomly added.

Parity bit :

code words	
000	101
011	110

} Here even parity is considered
even 1's

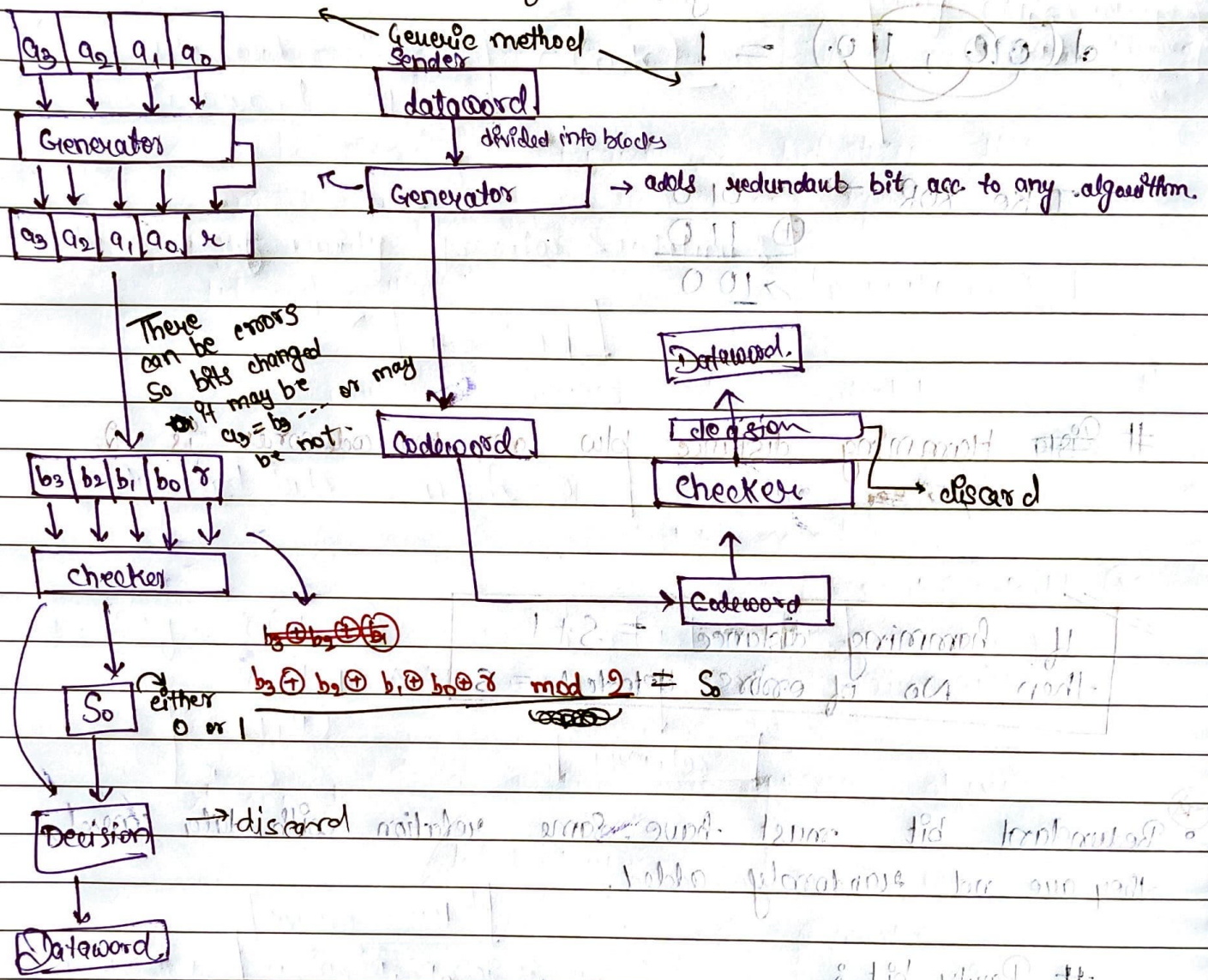
M T W T F S S

Page No.:

Date:

Cydic Redundancy check (CRC)

↳ Most commonly used in lans & WANs.



→ CRC :

data coword = k
Codeword = n
] C(n,k).

CRC : C(7,4) ...
Parity : C(7,6) → Always difference of one
C(5,4)

- In case of parity generator ⇒ relation ?
  → n = k + 1
  for parity ⊛. r = 1

redundand bits = n − k = r] For CRC r = n − k

Ex : say ~~k=4, r=3~~ for [a3 a2 a1 a0]
[C(7,4)
r = 3

→ Generator

divisor : r+1

d3 d2 d1 d0 | a3 a2 a1 a0 r2 r1 r0
known

Remaindr r2 r1 r0 ⇒ bits equal to r bits

#) dataword k = 20 bits
codeword n = 25 bits
} CRC ⇒ r = 5
divisor = 5+1 = 6
bits in remainder = 5 = r

Ex : 10111 0000

known 1001 | 10111 0000

1 0 1 1 1
1001 | 1011100000
⊕ 1001↓↓
00101
⊕ 1001↓
1100
⊕ 1001
1010
1001
011 → Remainder is actually the redundant bit

For modulo 2 ⇒
Subs ≡ add ≡ XOR

Code coord to be transmitted : "10111000"

↓

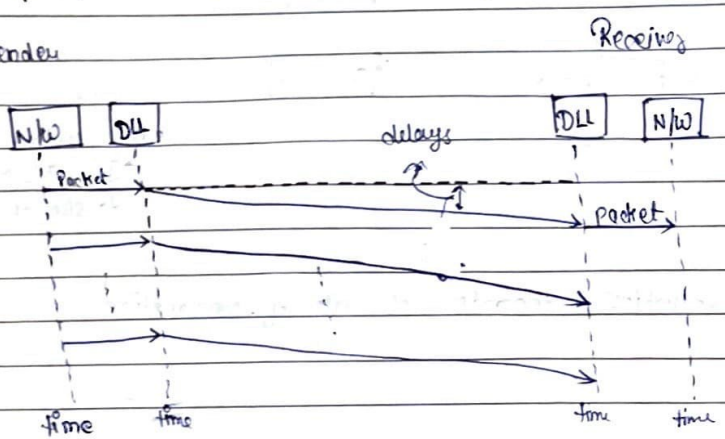
Received at receiver end.

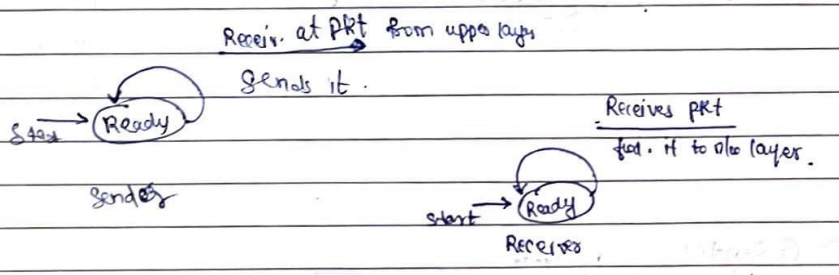
↓

Divided by same divisor → 1001

if remainder = 0 → Correct code
remainder ≠ 0 → Error is there.

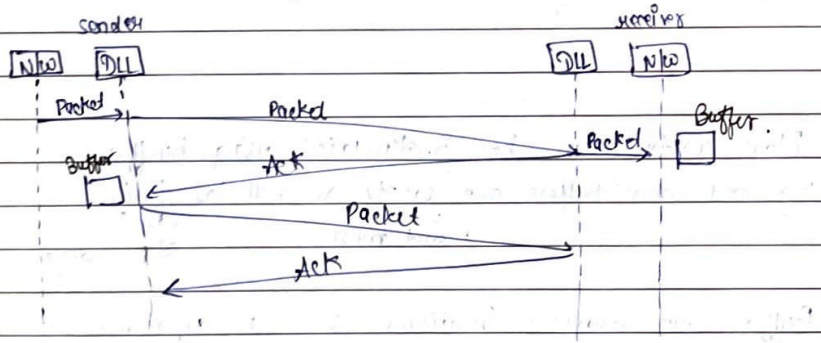
Simple protocol : no errors in path / lost msg.





- Here its always ready to accept the packet

Stop and Wait Protocol.



{ [event] / [Action taken] }

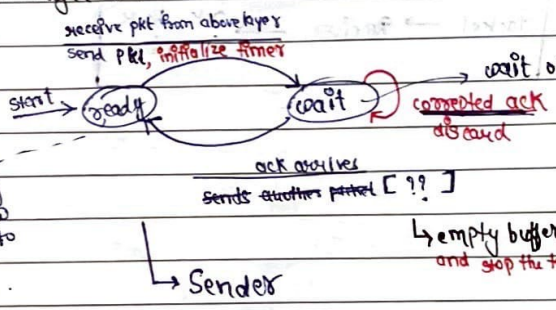
Ack: Acknowledgement → No errors detected

Next packet is sent only when ack. of ~~packet~~ previous is received

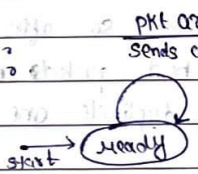
14/2/23.

Flow & error control.

State diagram : (finite state machine (FSM))



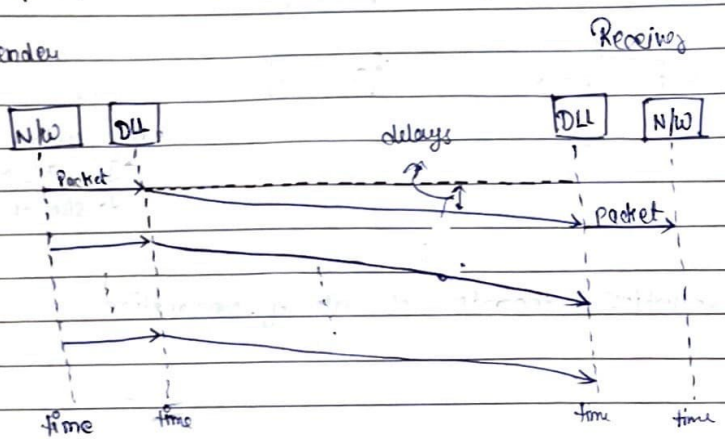
For receiver :

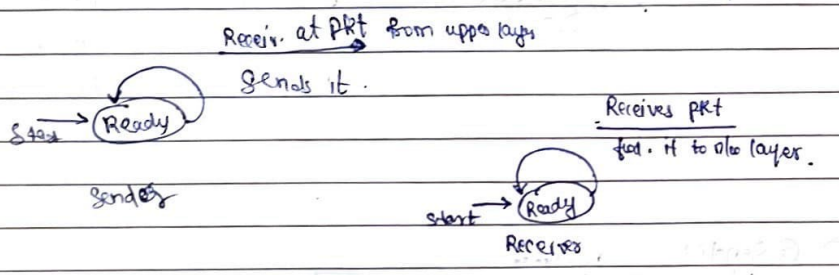


~~# Stop and wait Protocol~~

Packet is kept in buffer unfill ack. is carried

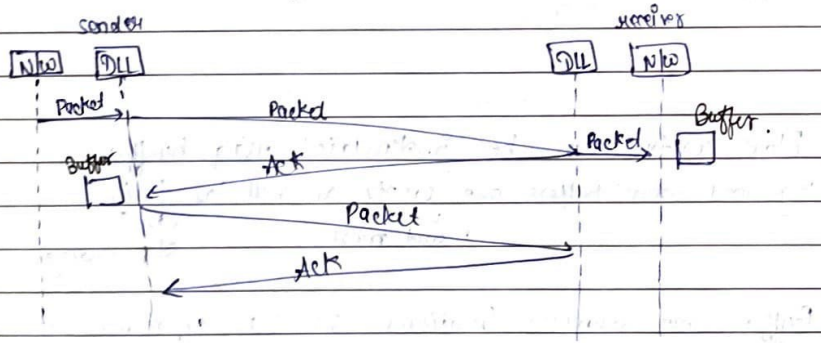
Simple protocol : no errors in path / lost msg.





- Here its always ready to accept the packet

Stop and Wait Protocol.



{ [event] / [Action taken] }

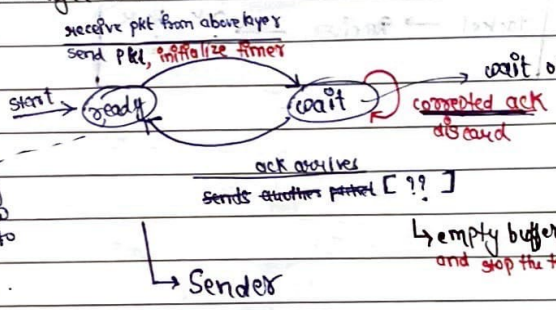
Ack: Acknowledgement → No errors detected

Next packet is sent only when ack. of ~~packet~~ previous is received

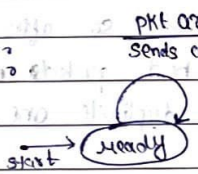
14/2/23.

Flow & error control.

State diagram : (finite state machine (FSM))



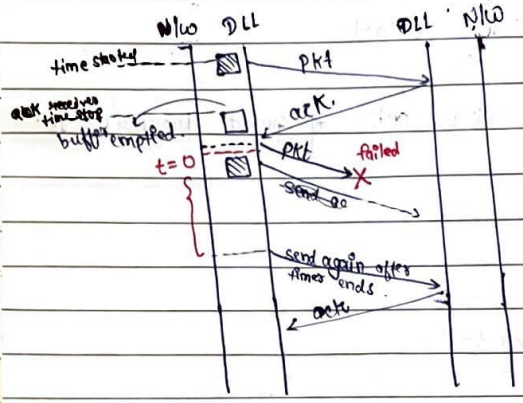
For receiver :



~~# Stop and wait Protocol~~

Packet is kept in buffer unfill ack. is carried

Packet got lost



an timer is initialized

Round trip time : [Packet → Receiver → ack. back to sender.]

Ack. got lost.

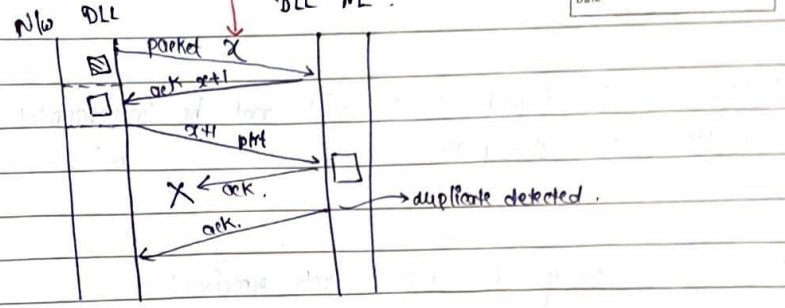
Packet send → but ack didn't received, so after time expired sender again sends to receiver but packet was already there at receiver, so we got a duplicate one. This should not happen

↳ We have a Sequence no. to check for duplicate.

↳ if duplicate received with same seq. no., it has been dropped & ack is sent.

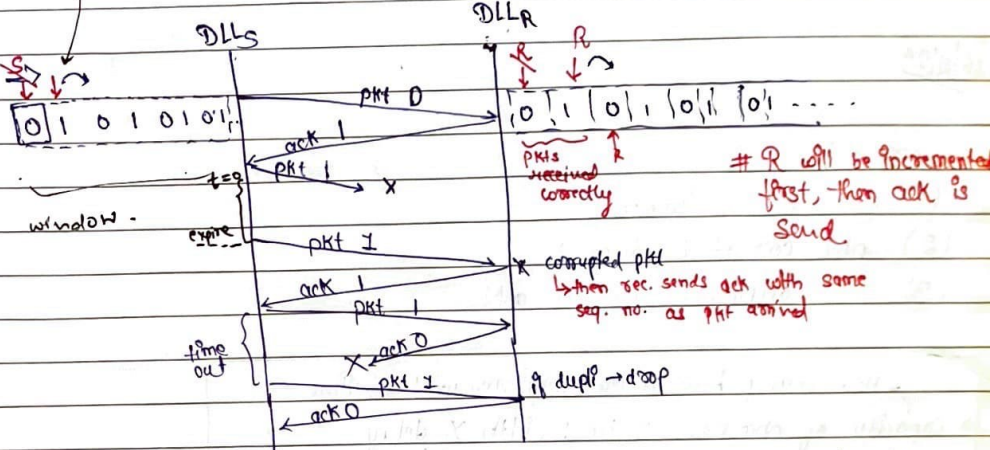
→ 1 bit is utilized for seq. no.

seq. no.



Stop & wait is ↴

Sliding window protocol



if S=0 & R=0 → feed pkt to upper layers and slide the window & R side

pkt send with seq. no S & ack. arrived with seq. no. S+1

Photo 14/2 if ack = S+1 → window is slided at S and

Read book

Wrong note

photo

⇒ If timeout happens (pkt send failed) → send pkt with same seq. no S

corrupted ack → Ack has CRC error.

If corrupted pkt is received, then receiver sends ack. with same seq. no. as pkt arrived

If CRC is found → R will not be incremented, and ack with same seq no.

S++ ⇒ only if correct ack received

R++ ⇒ only if pkt with correct (expected) seq. no. received.

16/2/24

error :
(1) pkt - lost : timeout
(2) pkt corrupted : discard
(3) " duplicate : discard, ack

* Capacity of channel : Bandwidth (How many bits it can carry per unit of time) × delay × RTT (Round trip time)

Q) Given bandwidth (Mb channel) = 10 Mbps, packet size = 2000 bits, delay (RTT) = 20 ms
What is the capacity of channel or how much bits it carried during the RTT

⇒ Capacity = $10 \times 10^6 \times 20 \times 10^{-3}$ (2000)
$= 2 \times 10^5$

Utilization % = $\frac{2000}{2 \times 10^5} \times 100$

= 1 % → we require more utilization, so need another protocol.

Go-Back-N protocol :

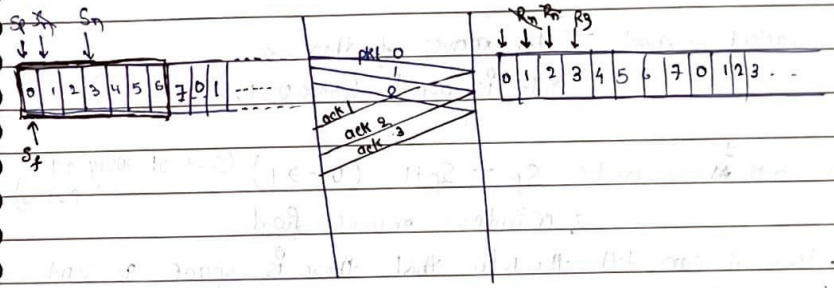
→ Assume m bits for the seq. no. ⇒ Range : $0 - 2^m - 1$

→ Let m = 3 ⇒ 0 - 7

Sender Window size = $2^m - 1$ (in Stop n wait it was 1)

↳ these many seq. no. are visible
↳ " " bits can be sent to the receiver

for receiver window size = 1.



Packets are coming from N/w layer
Sf moves only when ack will arrive

Sf
Sn = Sf + window size

→ One pkt arrives ⇒ pkt no. = 0
two " " ⇒ Sn moves fwd : 3 pkts had arrived
Sf wont moves unless ack arrives
When Sf is inc → window will also slide by that inc.
(if Sf +1 ⇒ window +1)

19/2/23

⇒ Initially S_f , $S_n = 0$.

If all buffer is full → S_n is at end (here at 6)
& S_f is still at 0 (no ack came till now) & packs not sent)

need to check $S_{n+1} \doteq (S_f + W_s) \% 2^m$
↳ If this is equals to window size, then sender will go to blocked state & no packets will be accepted.

⇒ Packet send :

	Receivers end
p0 →	
p1 →	
⋮	
p6 →	

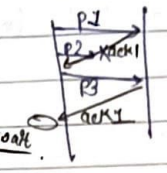
→ When packet received → R_n moves further & ack is send (ack 0, 1, 2, ~ 6)

→ When ack received : $S_f \to S_f + 1$ (0 → 1) (S_f++ at every ack arrives)
& window moves fwd.
& Now it can tell the N/w that there is space so send to packet to be sent

⇒

If Errors occur :

(1) Say if packet 2 didn't got sent



→ S_f will not get inc. as ack for p2 is not received

→ And if p3 arrives → It'll be "out of turn" packet so it cant accept it & p3 will be discarded and ack 1 will be sent (Recei → Send)

→ As sender get ack1 again (for out of turn ack), sender will wait.

Also a timer has also started, when p2 was not send and sender is waiting, after time has expired & all pkts have not been acknowledged, then the packets will be sent again.

those will be resent jya thi error aya hoy if p3 - error } these two resent
p4 ✓

⇒ Time is mostly $2 \times t_{propagation delay}$

(2) # If Packets are corrupted :

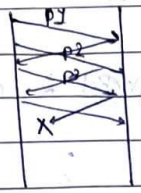
Receiver will discard the packet and all the packets after this will now be out of turn packet, so all those also will get discarded

And after time expired → Packets will be resend.

those only jya thi error aya
(say pack 1, 2 ✓
pack-3 → corrupted ✗ } 3&4 resend
4 ✗

Rn wont be inc only when pkt arrives correctly

③ If ack is lost (not reached)



ack 1 ✓

ack 2 → doesn't reach

ack 3 → reached. → this means packets 0, 1, 2 are successfully and correctly received & start pointing to 3. & sf will move further assuming 0, 1, 2 are reaching and ignoring failure of ack 2.

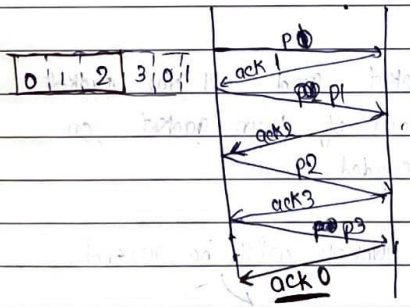
If last ack didn't received & all previous ✓, then after timeout, last pack sent again

{☆ Ack is cummulative}

#listen recording

④ Duplicate packet?

⇒ Why window size 2^{m-1} or lesser (what if more than $2^m - 1$?)



0 1 2 3 0 1 2 - - -

Receiver

★ Duplicate is not possible → will accept duplicate

21/2/24.

② Given, B/w = 100 Mbps, dist b/w S & R = 10000 km;
avg pkt size = 100000 bits; Speed of propagation = 2×10^8 m/s.

Find max. Speed of send & recei window?

→ determine capacity of channel = bandwidth × delay (prop. delay)

$= (1000) \times \frac{100000 \times 10^3}{2 \times 10^8}$

$= 5000$ Mbits. $\rightarrow \frac{1}{20}$ sec.

Prop. delay in one direct = $\frac{10^4 \times 10^3}{2 \times 10^8} = \frac{1}{20}$ sec

#

Round trip time (delay) = $\frac{1}{10}$ sec

capacity = b/w × total delay

$= 100 \times \frac{1}{10}$

$= 10$ Mb

$= 10 \times 10^6$ bits.

no. of packets = $\frac{10^7}{10^5} = 100$ packets.

[window size = 100] - for sender.

actual window size = 100
Max. " " = 128
Seq. no. acc. to m = 0 - 127
actual seq no = 0 - 99

Seq. no ⇒ 0 - 99

↳ 7 bits req. to accomodate this seq. no.

so [m = 7]

↳ Max window size ⇒ 0 - 127 (seq no.)

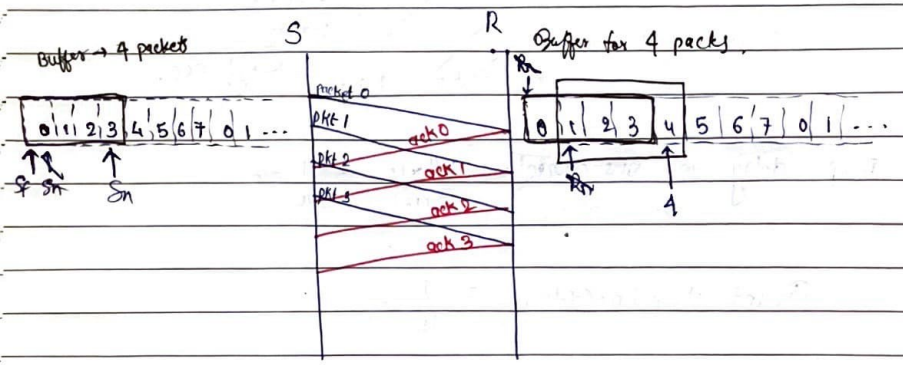
100 = actual

④ Selective Repeat (SR):

↳ Sender can send > 1 packet
↳ Here receiver can receives > 1 packets.

→ m bits
max window size = ~~2^m~~ 2^{m-1} $\quad W_S \le 2^{m-1}$

$\boxed{W_S = W_R = 2^{m-1}}$



→ Say 4 packets receive at S end ⇒ Sn++ 4 times & $S_n = 3$

⇒ $S_n = W_S$ → can't accept more packets

⇒ S_f ⇒ for outstanding packet

⇒ All 4 packs will be sent with seq no 0, 1, 2, 3

⇒ At R end
↳ gets packet → check → valid packet → ~~[illegible]~~
Send ack 0. first then do Rn++ and window size ~~[illegible]~~ moves forward.
whenever gets seq no = Rn

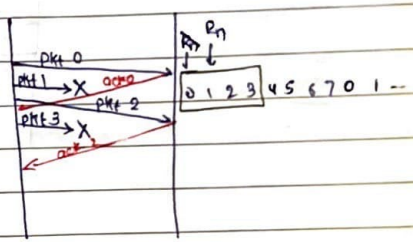
If ack no (received at Sender's end) $= S_f$, then slide window forward at Sender's end as we need not to maintain copy of S_f++

No copy is maintained at both S & R end

⑤ Errors

① If packets are lost :

pk0 ✓ & ack 0 ✓
↳ ~~slide~~ Rn++
↳ Window fixed → No copy maintained at Rec. end



→ pkt 1 not received
pkt 2 received → so copy of it is maintained & ack 2 is send

→ When ack 2 received at Sender's end, it will mark ack 2 as acknowledged.

⇒ Now Sender's end has idea that 1 & 3 are outstanded.
⇒ When timer ends only outstanded packets which are not marked ack. at Sender end will be resend from S's end not 0 & 2.

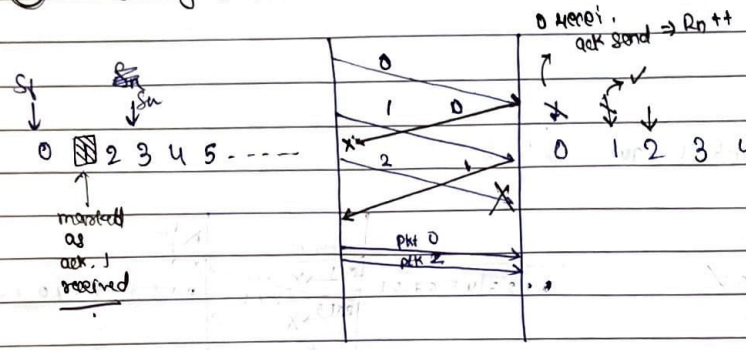
Whenever at packet is ~~sent~~ received, it mark it as received & its ack is send.
↳ say 0, 1, 2 → X but 3 ✓ ⇒ mark seq no 3 ✓ & send ack 3
& after time out rest will be send ~~again~~ again.

→ S_f is always the first unack packet

M T W T F S S
Page No.:
Date:

(*) → R_n ++ & window slides fwd only when consecutive packets arrives at receivers end.

(2) Ack gets lost.



After above situation ⇒ $S_n = 3$
$S_f = 0$
$R_n = 2$.

Send all outstanding pkt ⇒ pkt 0 ✓
pkt 2 ✓

pkt 0 received → duplicate → discarded
(why?? how??)

Since R_n has inc. to 2 so ack0 will never be sent again

28/2/24

M T W T F S S
Page No.:
Date:

Cummulative ⇒ ack of n means ack of prev all.

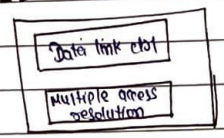
DLL has:
(1) framing ✓
(2) flow and error control ✓
(3) Media access control.

(*) Media Access control : → its all about reducing the chances of collision

listen before talk : ⇒ collision.

• choose a random time to wait if someone else is talking.

DLL



→ Network Protocols to detect and manage collisions

(*) Carrier Sense Multiple Access :

→ Busy
→ Idle.