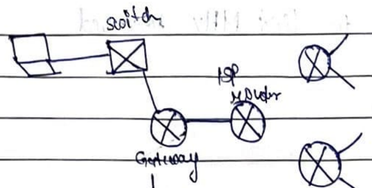
Look at Network adaptors in laptop

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

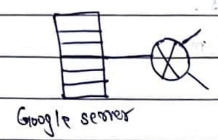
29/1/24.

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

TCP/IP stack.



↓
sends a response for DHCP req



→ 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 : 16 : FA : 3B : 8C : 16 : 15 → 6 bytes or 48 bits

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

⇒ 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

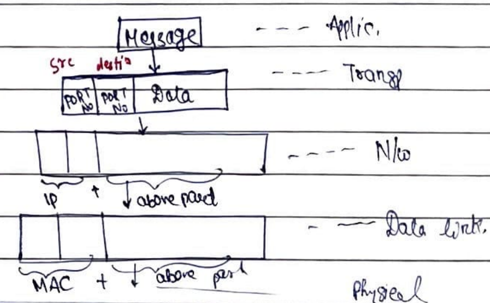
⇒ IP Addr v/s MAC Addr
- ~~Name~~ Name v/s Institute Id
  - Name → MAC
  - Institute Id → IP ↳ School Id, Clg Id, Company Id

⇒ 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)

⇒

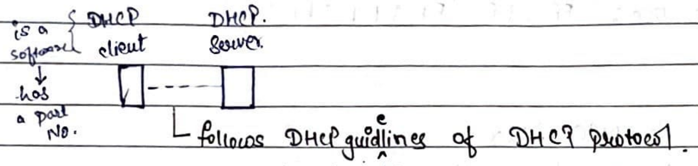


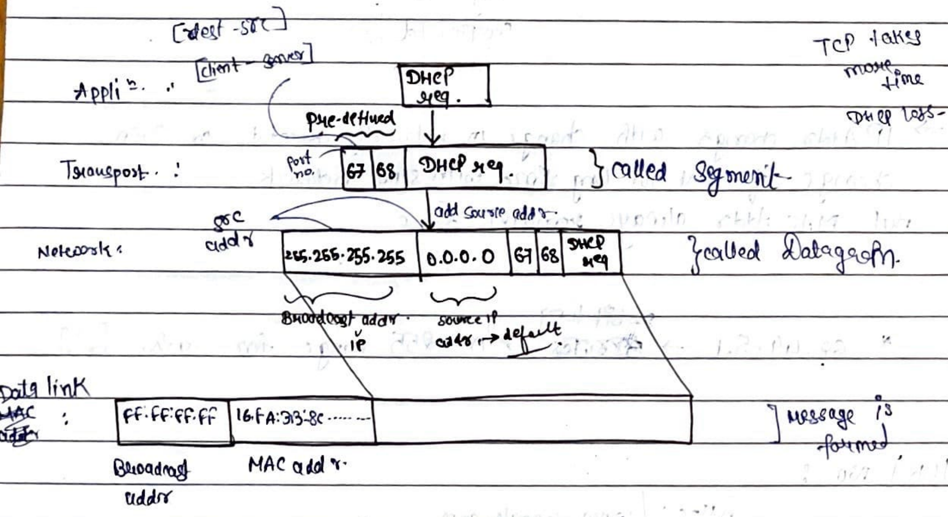
Book Pg 78 (5e)

Page No.:
Date:

DHCP :

Laptop wants to connect to www.google.com using ethernet
↓
DHCP is required.





→ This all is happening in route

→

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

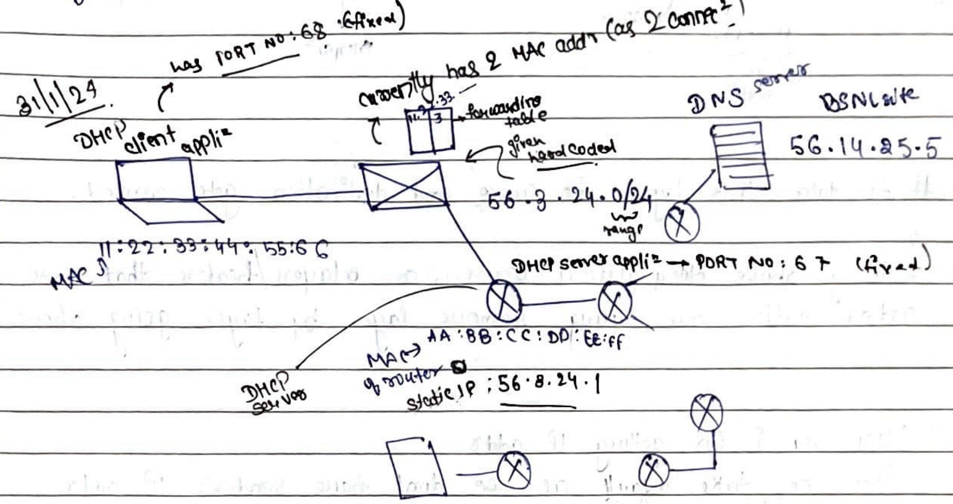
Page No.:
Date:

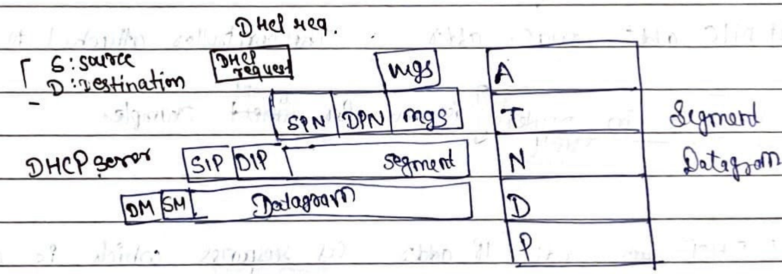
(1)

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

68.86.5.1 → of routers.

x.y.z.u



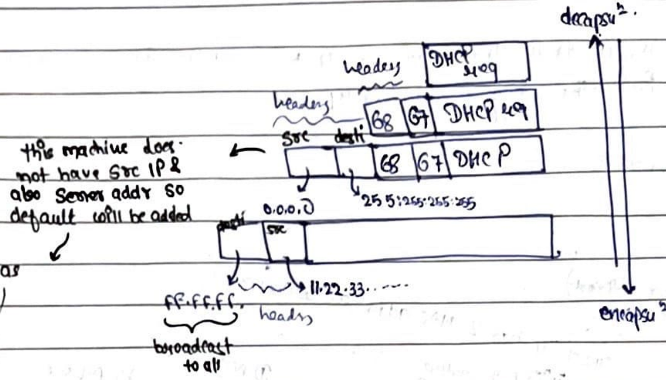


* We get MAC addr at NIC of each device
* IP addr is given by software by Network
* Server at which appli^n is running is called DHCP server.
* # No of NICs = no. of connec^n's device (router) has = No of MAC addrs
  DHCP servers provides IP addrs dynamically
  DNS server : Stores domain names

Adding addressing / headers → encapsulation

Removing headers → decapsulation.

Router is layer 3 device, ∴ High layer must be present



At data link layer ~~for~~ source 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 point (as its what we want), So at network layer in source we put default 0.0.0.0 IP addr.

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

$\boxed{\text{DHCP server is in router (itself)}}$ } assumed for current examples

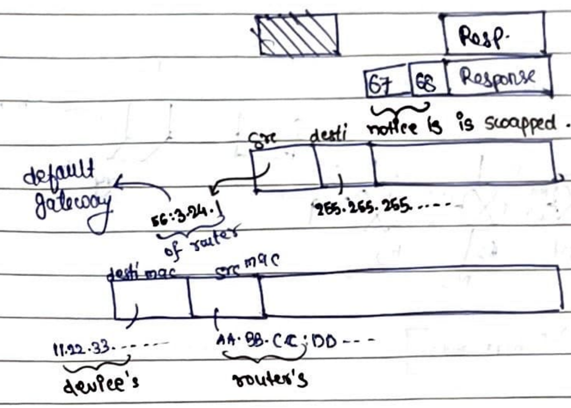
→ DHCP ~~res~~ sends IP addr as response which is available and adds some additional info

↳

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

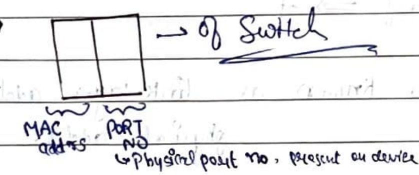
DHCP

Response : (Server → device)



↳ Now device have received DHCP response.

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

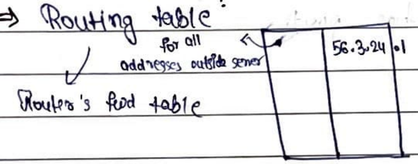


⇒ DHCP server v/s DNS server (Google it).

DHCP server: assigns IP addr to device which asks for it.

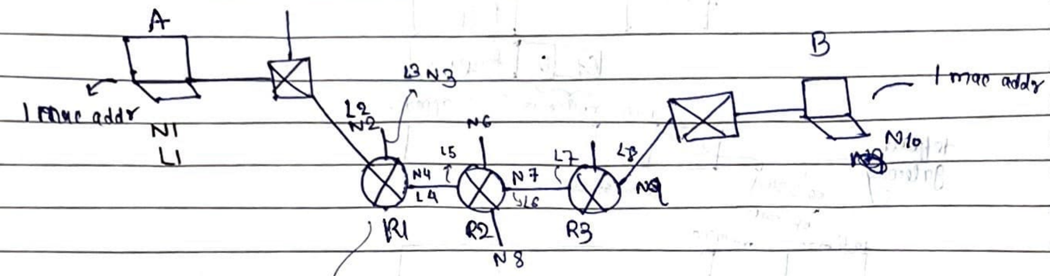
DNS server: ~~assig~~ gives IP addr of website

⇒ Routing table?



2/2/24 .

Look for this diagram in book.



* [3 NIC ⇒ 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 mac addr & IP ⇒ do not connect two diff devices
⇒ It only has ports.

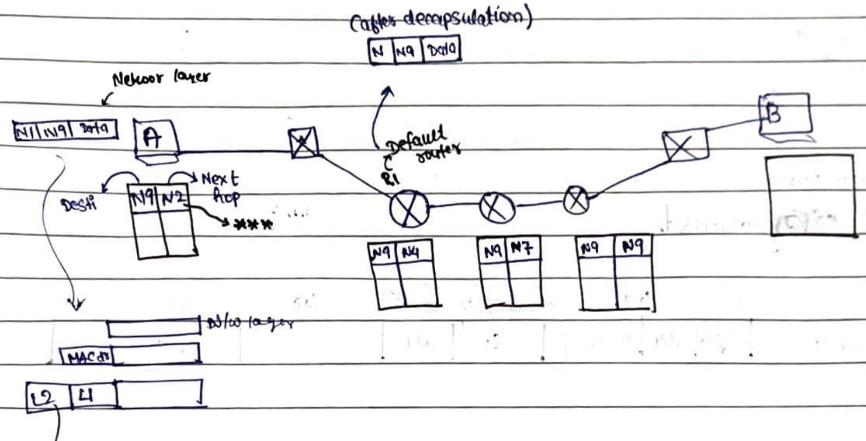
*** we got N2 from routing table ⇒ where it came from routing table?
↓
from DHCP response.

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 Resolu^n 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

Shivang

→ ARP response
ARP ~~reply~~ request

0	16	32	48	56	48 bits	next 32 bits	48	32
0x0001	0x0800	0x06	0x04	0x01	L1	N1	------	R2

↓ which protocol at data link layer (Ex: ethernet)

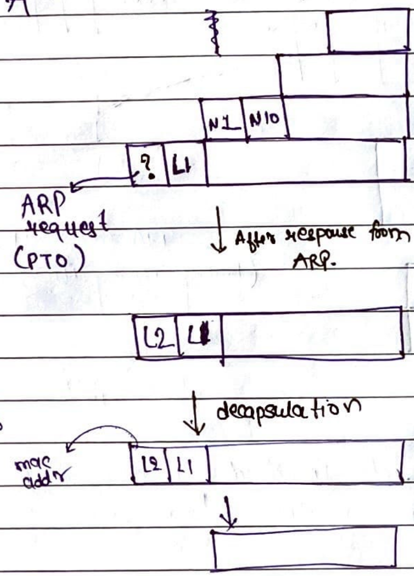
5/2/23

Data Link Layer.

App		
Xport	Port addr	DNS
N/W	IP addr	domain name → IP addr
DL	MAC addr	ARP : IP → MAC
Phy		

MAC layer addr are only relevant till next hop.

At A:



router checks if mac addr is of same router or not if so, remove that & move to next layer

↳PTO →**

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

ARP request sent by A :

← content →

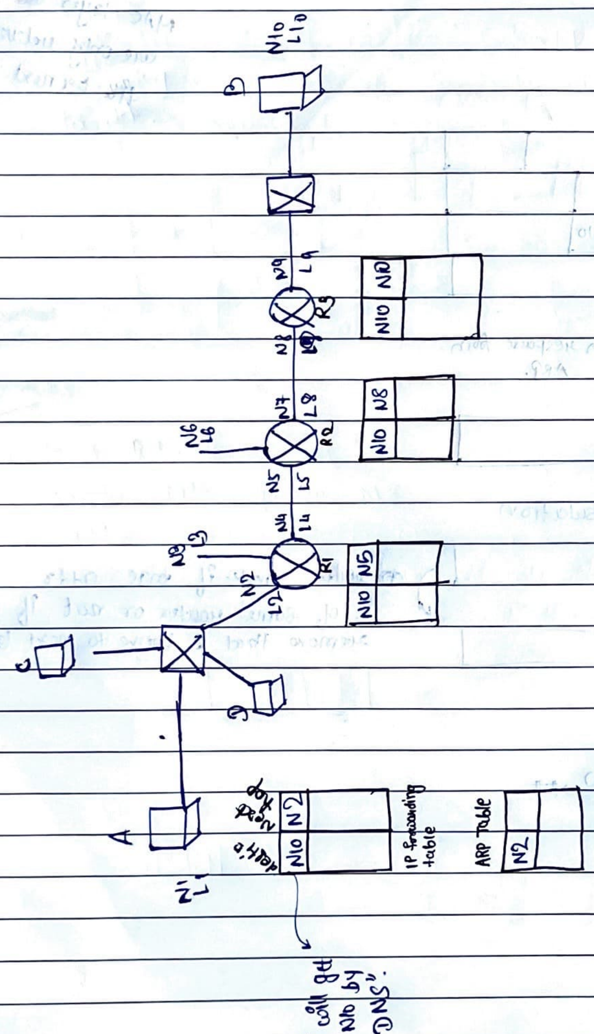
L1	N1	??	N2
Link layer addr	src addr	Mac addr of next hop	IP of next hop

desti	src	← content →
broadcast FF:FF:FF:FF	L1	

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

Response

desti	src	← content			→
L1	L2	L1	N1	L2	N2
	who is sending it				



destin^n pehle likhte hai
due to efficiency reasons

(R1 → R2)

N1	N10	

??	L4	

need mac addr corresponding to N4.

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

→ LL addrs changes at every hop.

listen to reading

→ check fwd table

→ to go to N10 go to N8

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

L8	L7	

⟹ final

L10	L9	---

when reached :-

~~L10~~	~~L9~~	

} check & remove

~~N1~~	~~N10~~	

} check if its meant for it self
if yes, remove & process

↓

Process.

→ What if we dont have data LL ?

N1 ←

N1	N10	

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

N1	N2	

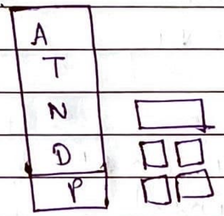
we lost final destina^n addr (N10), so
its not possible.

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.

Data link layer.

7/2/24

Error Control

100111000

→ 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 V/S 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 $n = k + r$

dataword → codeword

check code word → valid — No error

→ Invalid → Error detected

→ Take 2 bit data word :

↳ 2^2 combin^n

dataword	codeword	dataword	codeword
00	000	10	101
01	011	11	110

↳ redundant bit

length of code word = 3

↳ 8 combi^n out of which these 4 is valid

At receiver's end :

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

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

Hamming distance b/w any two codewords is 2

$\boxed{\text{If hamming distance} = S+1 \text{ then No of errors detected} = S}$

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

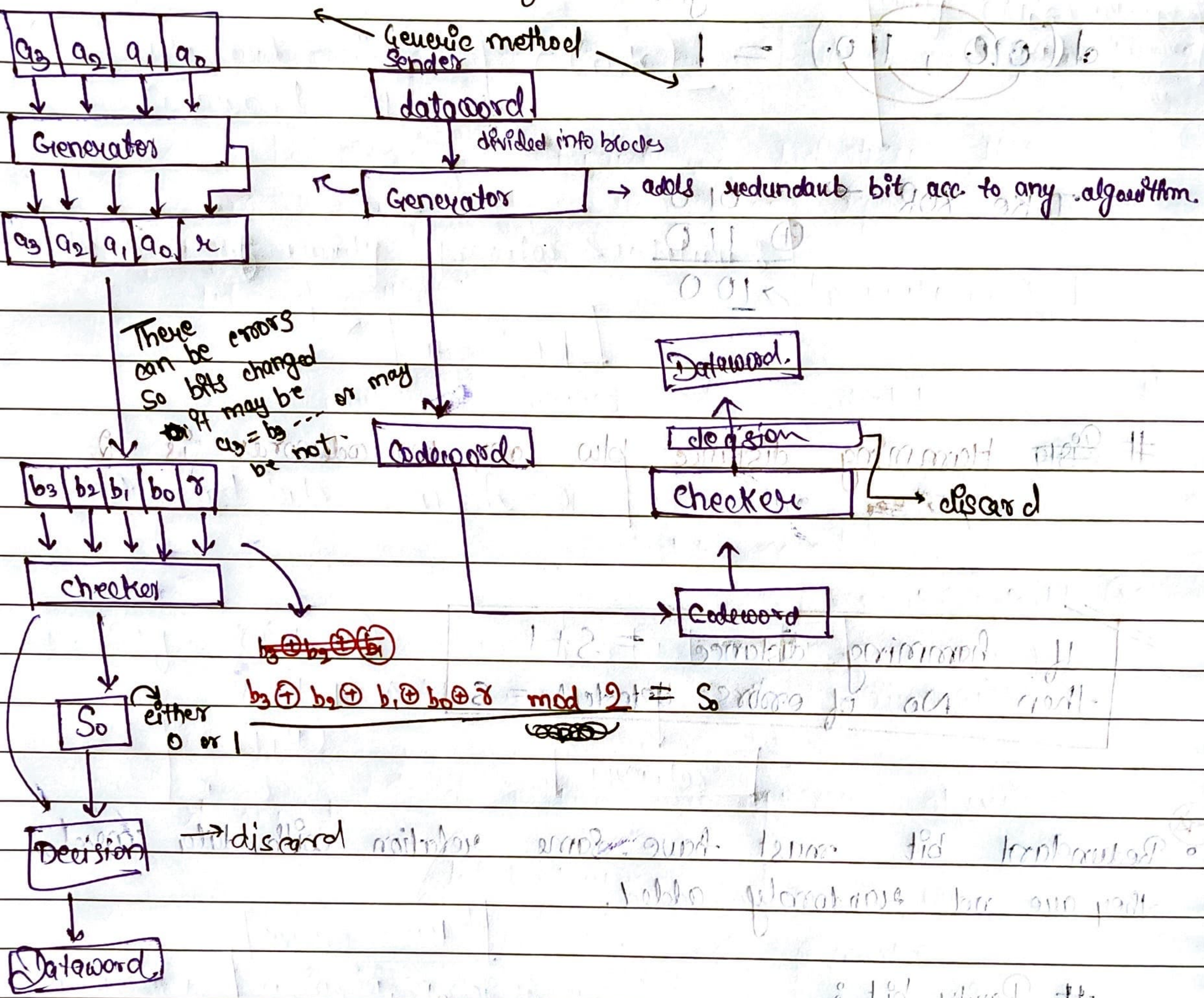
Parity bit :

code words	
000	101
011	110

} Here even parity is considered
even 1's

Cyclic Redundancy check (CRC)

↳ Most commonly used in lans & WANs.



→ CRC :

dataword = 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 :
  ↳ $\boxed{n = k+1}$

for parity r = 1

redundand bits = n − k = r

For CRC r = n−k

Ex : say for $a_3 \mid a_2 \mid a_1 \mid a_0$

C(7,4)
r = 3

Generator

divisor : r+1

$d_3\ d_2\ d_1\ d_0$ | $a_3\ a_2\ a_1\ a_0\ r_2\ r_1\ r_0$

known

Remaindr $r_2\ r_1\ r_0$ ⇒ 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

```
           101011
     1001 ) 10111000
          ⊕ 1001
            001010
          ⊕   1001
              1100
          ⊕   1001
               1010
               1001
                011
```

known

For modulo 2 ⇒
Subs ≡ add ≡ XOR

011 → Remainder is actually the redundant bit

Code word to be transmitted : "10111 0001"
↓
Received at receiver end.
↓
Divided by same divisor → 1001

if remainder = 0 → Correct code

remainder ≠ 0 → Error is there.