Introduction

- We study CN as a stack of 5 layers:
  Layer 5: Application Layer
  Layer 4: Transport Layer
  Layer 3: Network Layer
  Layer 2: Data Link Layer.
  Layer 1: Physical Layer.

This is known as TCP/IP stack. Each layer provides service to the layer above it and takes service from layer below it.

* IP Addresses

- Each IP address is divided into two parts:
  (1) Network ID
  (2) Host ID.

- IPv4 consists of 4 bytes, i.e. 32-bits.

Ex. 8 bits for N/W id.
So, # networks $= 2^8$
hosts $= 2^{32-8} = 2^{24} - 2$
(broadcast & host id)

This addressing scheme are rigid and inflexible. This only allows for 256 networks.

• Classful Addressing :

	Leftmost bits
(1) Class A: N/w bits = 8	0
(2) Class B: N/w bits = 16	10
(3) Class C: N/w bits = 24	110
(4) Class D:	1110
(5) Class E:	11110

Class A:

Range: 0.0.0.0 to 127.255.255.255

Class B:

Range: 128.0.0.0 to 191.255.255.255

Class C:

Range: 192.0.0.0 to 223.255.255.255

• Classless Addressing:

Number of network bits (prefix length) is not implicit, we have to explicitly specify that.

Ex. 12.24.34.127/21

Means that 21 bits are used for n/w prefix & 32-21 = 11 bits are used for host id.

So. n/w id: 12.24.00100010.127

So. first addr. of n/w: 12.24.32.0

last addr of n/w: 12.24.39.255

Ex. 220.8.24.255/25

Id: 220.8.24.11111111./25

∴ First Addr: 220.8.24.128.
Last Addr: 220.8.24.255.

Ex. B 167.199.170.82/27

Id: 167.199.170.010,10010./27

∴ First addr: 167.199.170.64
Last addr: 167.199.170.95
host addrs = 32 - 2 = 30.

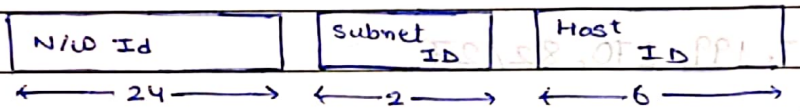
- The first address is used for identifying the network and hence can't be used for host addressing.
- The last address is used for broadcasting within the network and hence can't be used for addressing a host. Called as Direct Broadcast Address (DBA)
- Any host within a n/w can send an IP packet to IP 255.255.255.255 to broadcast within the n/w.
- Similarly 0.0.0.0 is an unusable address and is used within a host (unusable for routing)

255.255.255.255 is called "Limited Broadcast Address". It is limited, since the packets to this addr. are never fwded across a router. For DBroadcastAddr, the packet can be sent from other n/ws to be broadcasted in the n/w.

* Subnetting

Dividing a network into smaller networks is called as subnetting.

Ex. Divide the following n/w in four networks:
192.168.9.0/24.



Sub1: 192.168.9.0/26 to 192.168.9.63/26
Sub2: 192.168.9.64/26 to 192.168.9.127/26
Sub3: 192.168.9.128/26 to 192.168.9.191/26
Sub4: 192.168.9.192/26 to 192.168.9.255

Sub1: N/W ID: 192.168.9.0
Broadcast ID: 192.168.9.63
:

No. of hosts in each subnet = $2^6 - 2 = 62$

Ex. Divide such that each n/w has 10 hosts.
192.168.9.0/24.

We have 8 bits for hostid currently.
With 2 bits: $2^2 - 2 = 2$ hosts
With 3 bits: $2^3 - 2 = 6$ hosts
With 4 bits: $2^4 - 2 = 14$ hosts.
∴ 4 bits will be kept for hostid.
4 bits will be used for subnetid.

Ex. Say 192.168.9.0/24 is divided into four subnets & an external router forwards a packet to the n/w, then how to know if the packet is for subnet 1 or for the n/w itself?

N/w Id: 192.168.9.0
N/w id for subnet 1: 192.168.9.0

In this case, the packet is delivered to the n/w not the subnet.

Similarly, if dest^n IP is 192.168.9.255 then it would be broadcasted in whole n/w, not just the last subnet.

Ex. 14.24.74.0/24. Create 3 subnets with 10, 60 & 120 hosts respectively.

We'll go in decreasing order of #addresses always 120 > 60 > 10.

Currently 8 bits for host id.

Subnet 1: 120 hosts

14.24.74.0 _____ # hosts = $2^7 - 2 = 126$.

addr: 14.24.74.0/25 to 14.24.74.127/25

Subnet 2: 60 hosts

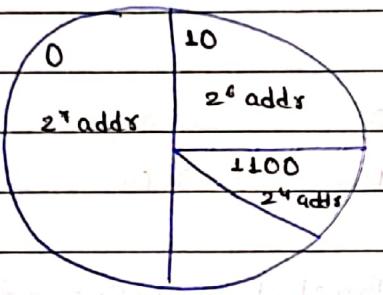
14.24.74.10 _____ # hosts = $2^6 - 2 = 62$

addr: 14.24.74.128/26 to 14.24.74.191/26

Subnet 8 : 10 hosts

14.24.74.1100_____ , #hosts = $2^4 - 2 = 14$.

addr: 14.24.74.192/28 to 14.24.74.207/28



Ex. 172.16.0.0/16, into subnets with 500, 200, 100, 62 & 22 hosts.

We have 16 host bits.

S1: 500 hosts

172.16.0000000.1_____ , #hosts = $2^9 - 2 = 510$.

Range: 172.16.0.0/23 to 172.16.1.255/23

S2: 200 hosts.

172.16.00000010._____ , #hosts = $2^8 - 2 = 254$

Range: 172.16.2.0/24 to 172.16.2.255/24.

S3: 100 hosts

172.16.00000011.0_____ , #hosts = $2^7 - 2 = 126$.

Range: 172.16.3.0/25 to 172.16.3.127/25

S4: 62 hosts

172.16.3.10_____ , #hosts = $2^6 - 2 = 62$.

Range: 172.16.3.128/26 to 172.16.3.191/26

S5 : 22 hosts

172.16.3.1010_____ #hosts = $2^5 - 2 = 30$.

Range : 172.16.3.192/27 to 172.16.3.223/27.

* Forwarding of IP Packeting

Every router stores a few mandatory fields in its routing table :

(1) Network ID
(2) Prefix Length
(3) Output Interface

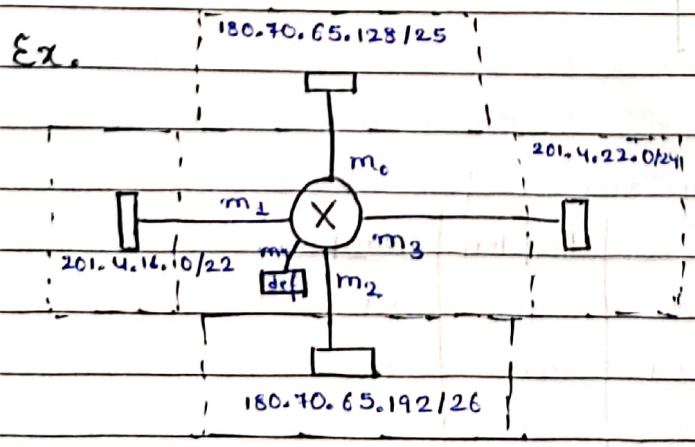
When an IP packet comes to router say 'A', then the router in descending order of Prefix Length does the following :

(1) Takes bitwise AND of 'A' with n/w mask.

A' ← A & 111....1000..0

(2) Compares A' with network ID.
(3) If matched, forward the packet to the respective interface.
(4) Else repeat for next row.

Ex.



S No.	N/W ID	Pref. len.	O/P Intf.
1	180.70.65.192	26	m_2
2	180.70.65.128	25	m_0
3	201.4.22.0	24	m_3
4	201.4.16.0	22	m_1
5	default	32	m_4

If no match is found router forwards the packet on the default interface.

(1) Destⁿ IP: 201.422.128.

S1: (201.4.22.128 & 1111....1000..0)
←26→

This doesn't match SNo1.

S2: (201.4.22.128 & 11....1000..0)
←25→

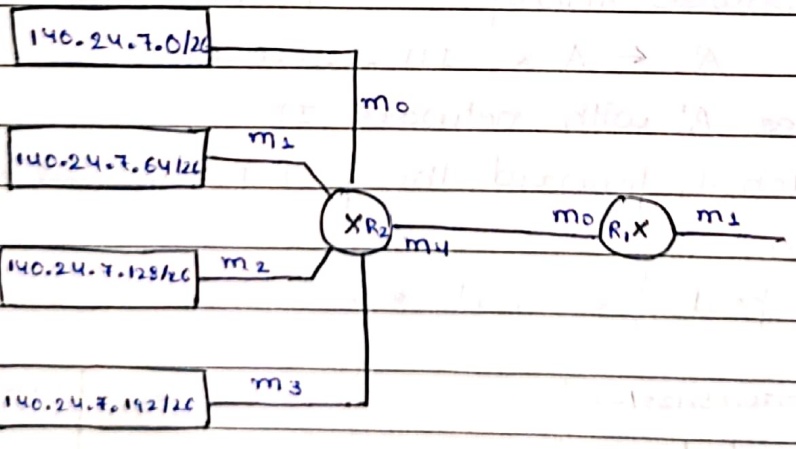
This doesn't match SNo2.

S3: (201.4.22.128 & 11....10.00)
←24→

This matches SNo.3.

∴ Fwd to m_3.

Ex.



If we create routing table of R_1 we'll see:

Sno.	N/w Id	Pref len	Intf.
1	140.24.7.0	26	m_0
2	140.24.7.64	26	m_0
3	140.24.7.128	26	m_0
4	140.24.7.192	26	m_0
5	Default	-	m_1

We can see redundancy with m_0 interface. We can combine the m_0 entry in one entry, as,

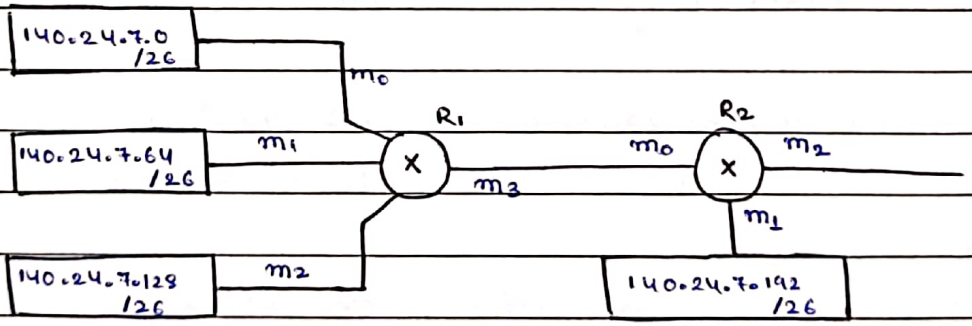
$140.24.7.0/24 \rightarrow m_0$.

This is called Supernetting.

S No.	N/W ID	Pref Len	Intf.
1	140.24.7.0	24	m_0
2	Default	-	m_1

Supernetting doesn't actually merge organizations, it is just reducing redundancy for a particular router. (Router R_1 in this case).

Ex.



R_2: S No.	N/W ID	Pref. Len.	Outp. Intf.
1	140.24.7.192	26	m_1
2	140.24.7.0	24	m_0
3	Default	-	m_2

at R_2

Say a packet comes with destn IP: 140.24.7.193. It matches both entry 1 & 2 upon applying respective mask.
The packet is forwarded to longest prefix matched.
So, packet is forwarded to m_1.

Network Layering & N/w Delays

* **Protocol:** A set of rules agreed upon & followed by two or more devices.

* **TCP/IP Stack**
  - L5: Application Layer
  - L4: Transport Layer
  - L3: Network Layer
  - L2: Data Link Layer
  - L1: Physical Layer.

The responsibilities and functionality of networking is divided among the five layers.
It can be assumed that b/w the same layers of two diff. stacks, there is a logical connection.

- Endpoint devices such as mobiles/laptops, etc. (basically hosts) operate on all five layers.

- Routers operate only upto layer 3 i.e upto N/w layer only.

* **Bandwidth:** The bandwidth of a channel is the amount of the data that can be sent in a given amount of time. (bits per second or bytes per μsec, etc)

Throughput: The total amount of data transferred over a certain amount of time.

BW is the ideal speed of transfer, throughput is the actual speed.

Ex. A n/w with bandwidth of 10 Mbps, can pass only an avg. of 12,000 frames per minute, with each frame carrying an avg 10^4 bits. What is the throughput?

$12 \times 10^3 \times 10^4$ bits per min.

$\Rightarrow$ 2×10^6 bps

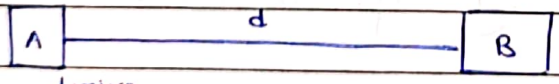
$\Rightarrow$ 2 Mbps.

* Latency

The network latency of a n/w is determined as:

Latency = Propagation Time + Transmission Time + Queing Delay + Processing Delay

- **Propagation Time:** The time to send a single bit from host A to host B.



Say d mtrs is distance b/w A & B, and speed of propagation is S mtrs/sec, then

$$PT = \frac{d \text{ mtrs}}{S \text{ mtrs/sec.}}$$

- **Transmission Time:** The time it takes for a sending host to put the data onto the channel (or link).

$$T_t = \frac{\text{Message Size}}{\text{Bandwidth}} = \frac{L}{B}$$

- **Queing Delay:** The time a packet/frame is queued for at a router.

* **Link Vs Channal**

A link is a single physical connection. On a link multiple channals can be there, divided by Time, freq., wavelength, etc.

Data Link Layer

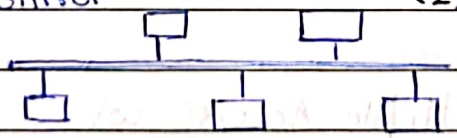
- Provides service to the Network layer. Encapsulates the IP packet.

The TCP/IP stack specifies that IP protocol be used at network layer, but it doesn't specify the data link layer protocol to be used. Generally Ethernet Protocol is used.

* **Functionalities:**
  (1) Framing
  (2) Flow Control
  (3) Error Control
  (4) Media Access Control.

Provides hop-to-hop service.

* In case of a broadcast link, i.e. multiple receivers & senders on same link, the Data Link layer is divided into two sublayers:
  (1) Data Link Control
  (2) Media Access Control



But in case of peer to peer link only Data Link control sublayer is needed.

MAC sublayer helps different sending hosts to avoid collisions by sharing the channel through different methodologies.

* Multiple Access Protocols

Random Access Protocols	Controlled Access Protocols	Channelization Protocols
(1) ALOHA	(1) Reservation	(1) FDMA
(2) CSMA	(2) Polling	(2) TDMA
(3) CSMA/CD	(3) Token Passing	(3) CDMA
(4) CSMA/CA		

- **Random Access:** All hosts can send. No host is superior to the other.

There are four questions a MAC protocol has to answer:
(1) When can a station access the medium?
(2) What can the station do if the media is busy?
(3) How can a station determine success/failure of transmission?
(4) What can a station do in case of collision?

* Carrier Sense Multiple Access w/ Collision Detection

Each host monitors the medium if it is busy or not. Only if it senses a free channel it will start transmission.

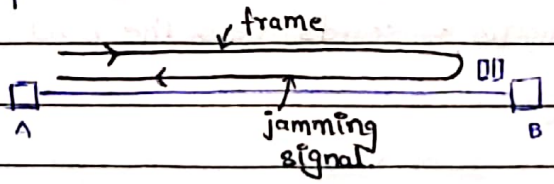
After a host transmits, it keeps sensing the channel to see if transmission was successful or not. A host can only sense the channel as long as its transmitting.

- Say A & B sense that the channel is free and start transmission.

After there is a collision a jamming signal is sent onto the channel so that all hosts are aware of the collision.

There is no system of acknowledgment in CSMA/CD.

Say that the collision happens 5 mins after A has transmitted, then the jamming signal would also take 5 mins to reach back to A.
But what if A has already sent the complete frame and stopped sensing the channel, then it wouldn't know about the collision.
Therefore the minimum size of frame transmitted by any host should be such that it takes long enough to transmit and in case of collision the jamming signal can reach back to host before transmission ends.



$$\therefore \quad T_t \geq 2 \cdot T_p$$

$$\Rightarrow \quad \frac{L_{min}}{BW} \geq 2 \cdot T_p$$

$$\Rightarrow \quad L_{min} \geq 2 \cdot T_p \cdot BW$$

$\therefore$ In CSMA/CD, $L_{min} \geq 2 \cdot T_p \cdot BW$

* **Backoff Algorithm:** After a collision happens, how long should a host wait before retransmitting the frame. Backoff algorithm reduces the chance of collision by choosing random amount of time to wait at each host.

Say A & B are involved in collision, and K is the count of collision.

- K = 1 :

A chooses from 0 to 2^{K-1} units of time to wait.

[0, 1]

K = 1

B chooses from 0 to 2^{K-1} units of time to wait.

[0, 1]

A	B	
0	0	← Collision
0	1	← A wins
1	0	← B wins
1	1	← Collision

P(Collision) = $2/4 = 1/2$.

Let's suppose A wins & sends the packet & wants to send next packet.

- K = 1

$[0, 2^0] = [0, 1]$

K = 2

$[0, 2^1] = [0, 2]$

A	B	
0	0	← C
	1	
	2	
1	0	
	1	← C
	2	

P (Collision) = $2/6 = 1/3$

P (A wins) = $3/6 = 1/2$

P (B wins) = $1/6 = 1/6$

and so on...

The probability of collision decreases over time.

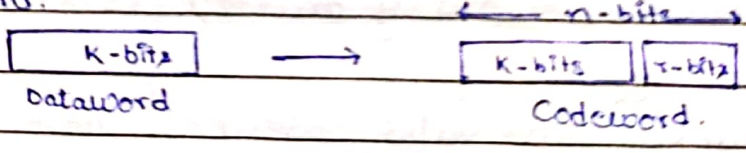
* Error Control

- There can be two types of errors:
  (1) Single bit Error
  (2) Burst Error.

- There are two steps of Error Control:
  (1) Error Detection
  (2) Error Correction.

- Given a dataword, the sender converts the dataword into a codeword which is then sent to the receiver.
  There are two types of encoding:
  (1) Block encoding
  (2) Convolution encoding.

- In block coding, bits are appended to the last of the dataword.



For a given k-bit dataword there is a unique codeword.
The detection of error on receiver side is dependent on the receiver knowing if the received codewords are valid or not.

Ex. $K = 4 \quad r = 2$

So, no. of valid codewords $= 2^K = 2^4 = 16$

no. of total codewords $= 2^n = 2^6 = 64$

$\therefore$ #invalid codewords $= 64 - 16 = 48$

The error would only be detected if the sent codeword gets changed to an invalid codeword. If a valid codeword changes to other valid codeword, the error can't be detected.

- **Hamming Distance:** The hamming distance b/w two n-bit binary strings is the number of positions in which they differ.

Ex. HD (10110, 01101) = 4

```
1 0 1 1 0
↕ ↕ ↕ ↕ ↕   ∴ 4
0 1 1 0 1
```

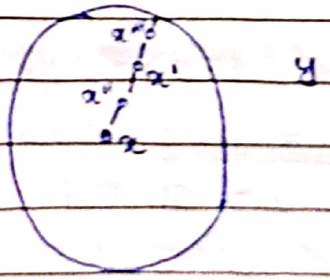
HD (x, y) = No. of 1's in (x ⊕ y)

- **Minimum HD for Error Detection in Block Code.**

Say we want to detect n-bit error b/w the sent and received codewords. then the minimum hamming distance b/w any two valid codewords must be (n+1).

This means for a sent valid codeword x to change into another valid codeword y on receiver's end at least (n+1) bits must toggle.

If the min. HD b/w two valid codewords were to be ≤ n, then n-bit errors couldn't be detected with 100% accuracy since x might change to some other valid codeword y, or invalid codeword x' within n-bits.



• **Linear Block Code:** An encoding scheme is linear blockcode if XOR of any two valid codewords results in another valid codeword.

• **Minimum HD for Error Detection in Linear BC**

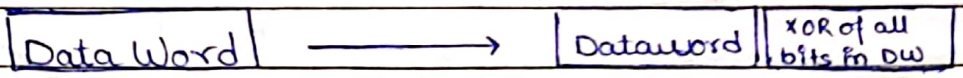
Since HD(x, y) = No. of 1s in (x ⊕ y) & for any x, y in a LBC, x ⊕ y is also a valid LBC, we can say, the min. HD in a LBC is the min. no. of 1s in any single codeword, except for all 0s codeword.

Ex. Suppose following LBC:

Dataword	Codeword	
00	000	#1s = 0
01	011	#1s = 2
10	101	#1s = 2
11	110	#1s = 2.

∴ Ignoring all 0s codeword min. HD for LBC given is 2.
∴ Upto 1 bit error can be detected by the given Block Code.

* • **Parity Check Code.**



Parity Check Code is Linear Block Code which can detect all odd bit errors.

- Cyclic Code: These are linear block codes, with the additional property that if a valid codeword is cyclically rotated the result is still a valid codeword.
  Ex. Cyclic Redundancy Code.

* Cyclic Redundancy Code:
There is a agreed upon divisor shared b/w sender and the receiver. The sender using the divisor generates the codeword for a given dataword. The receiver using the divisor checks for errors in the received codeword.

If divisor is of n-bits then n-1 bits are appended to the dataword.
At each step of CRC division XOR operation is performed.

Ex. Dataword: 1001 Divisor: 1011.

```
          1010
      ------------
1011 | 1001 000
       1011
       ----
       0100
       0000
       ----
       1000
       1011
       ----
       0110
       0000
       ----
        110
```

∴ Codeword = [Dataword][Remainder]
⇒ 1001 110

- On the receiver end same division is performed on the codeword.
  (1) If the remainder is non-zero, the codeword is corrupted.
  (2) If the remainder is all 0s, the codeword is valid, may or maynot be corrupted.

- We can convert the codeword from binary representation to polynomial representation.
  Ex. 00101010

$$0\cdot x^8 + 0\cdot x^7 + 1\cdot x^6 + 0\cdot x^5 + 1\cdot x^4 + 0\cdot x^3 + 1\cdot x^2 + 0\cdot x^1 + 0\cdot x^0$$
$$\Rightarrow x^6 + x^4 + x^2$$

- For CRC division both addition & subtraction of two binary numbers is same as XOR operation.

Ex. Dataword: $x^5 + x^3$ Divisor: $x^3 + x + 1$

$$\begin{array}{r|l} & x^3 + x \\ \hline x^3+x+1 & x^5 + x^3 \\ & x^5 + x^4 + x^3 \\ \hline & x^4 \\ & x^4 + x^2 + x \\ \hline & x^2 + x \end{array}$$

∴ Remainder: $x^2 + x$

So, Codeword: $x^5 + x^3 + x^2 + x$

• Analysing CRC.

Dataword: $d(x)$ Generator: $g(x)$

Codeword: $c(x)$ Error: $e(x)$

So, the received codeword is: $c(x) + e(x)$

Ex. $c(x) = x^6 + x^2$

$c(x) + e(x) = x^6 + x^5 + 1$

OR

CW = 01000100

Receiv = 01100001

∴ Error = 00100101

On the receiver end if we divide codeword $c(x)$ by divisor/generator $g(x)$ we get 0 as remainder.

$$\frac{\text{Received}}{g(x)} = \frac{c(x) + e(x)}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

So, $c(x)/g(x)$ would give 0 remainder. If for a non-zero $e(x)$, $e(x)/g(x)$ gives 0 remainder then such an $e(x)$ can't be detected as $c(x) + e(x)$ would be some other valid codeword.

(1) Single Bit Errors

To detect single bit errors what should $g(x)$ look like? W.K.T. for toggle of i^{th} bit, $e(x) = x^i$, so $g(x)$ should have at least two terms, i.e.

$$g(x) = x^k + x^l$$

Ex. Which of the following g(x) guarentees detection of all single bit errors:

✓(1) x^3+1

(2) x^3

(3) 1

(2) x^3 only guarentees single bit error detection for
$e(x) = x^2$ $e(x) = x$ $e(x) = 1$.
for $e(x) = x^i$, $i \geq 3$. $g(x) = x^3$ divides $e(x)$.
(3) 1 divides all $e(x)$.

(2) Two isolated single bit errors

$e(x) = x^j + x^i$

$\Rightarrow x^i (x^{j-i} + 1)$

Ex. Which of the following g(x) can detect two isolated single bit error:

(a) $x+1$

(b) x^4+1

✓(c) x^7+x^6+1

(a) No. $e(x) = x^5(x+1) = x^6+x^5$ is div. by $(x+1)$

(b) No. $e(x) = x^2(x^4+1) = x^6+x^2$ is div. by x^4+1

(c) Yes.

(3) Odd Number of Errors

$e(x) = x^k \ldots$ (odd no. of terms)

$(x+1)$ can never be factor of $e(x)$ since $e(x)$ has odd no. of terms. (XOR of odd no. of 1s is 1).

∴ $g(x)$ must have $(x+1)$ as factor to detect odd no. of errors.

* Flow Control

When there is disparity b/w the speed with which the sender sends data & receiver receives data, then flow control methods allow for sender ~~& receiver~~ to adjust its speed.

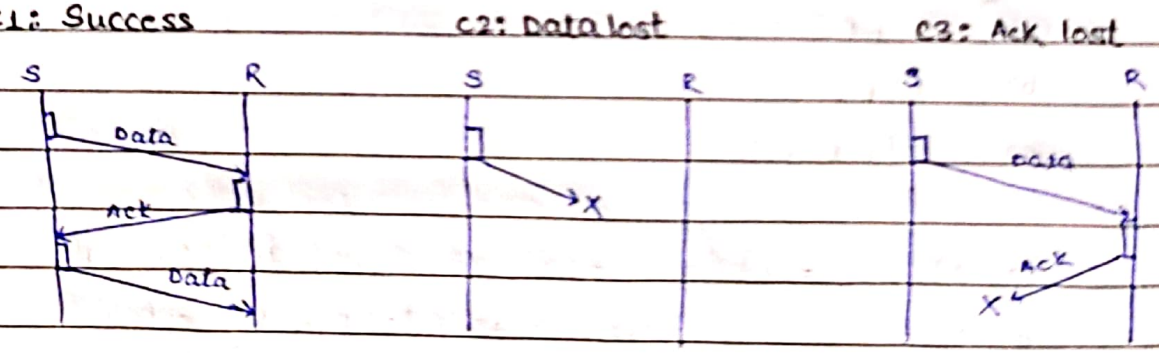
* Stop - And - Wait

Sender sends a packet and starts a timer. If an acknowledgement is received before the timer expires, sender moves on, else sender resends the packet and restarts the timer.

There can be two cases for not receiving acknowledgment :

(1) Data isn't received by receiver.

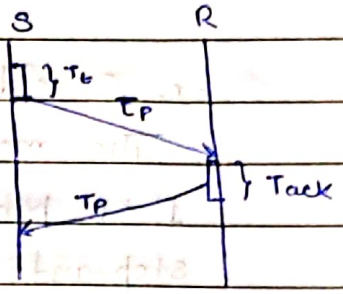
(2) Acknowledgment is lost.



- There can also be case where data is delivered but ~~ack~~ is corrupted and hence Ack isn't sent.

- Only 2 seq. no. are needed: 0 & 1. Ack 1 means sender would send Packet with Seq. No. 1.

- Total Delay = $T_t + T_p + T_{track} + T_p$
  $\Rightarrow T_t + 2T_p + T_{ack}$



- **Efficiency:** Total time sender is engaged in sending the packet upon total delay.

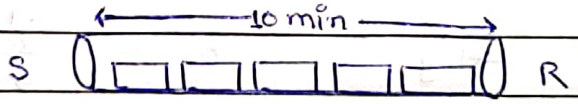
$$\eta = \frac{T_t}{T_t + 2T_p + T_{ack}}$$

(∵ Sender is only engaged for the transmission time).

$$\eta = \frac{1}{1 + 2\,T_p/T_t} = \frac{1}{1+2a} \quad \text{where } a = \frac{T_p}{T_t}$$

a tells the no. of packets that can fill the channel.

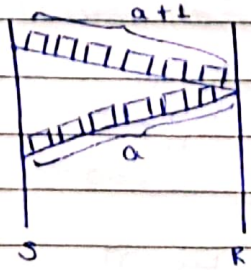
Ex. Say $T_p = 10$ min. $T_t = 2$ min.



So, this shows that if we take 2 min to put a packet on channel we can fill it with 5 packets.

We can define efficiency as such: No. of packets sent upon no. of packets that can be sent.

In Stop-And-Wait we are sending single packet, while we can send $2a+1$ packets.



$a+1+a = 2a+1$.

(a+1, since ack will be sent only • when 1st packet is fully received).

Ex. Consider a 40 Kb/s n/w link connecting earth to moon. The moon is about 1.5 light-second from earth. 1 KB packets are sent over this link. ~~Find~~ using stop-and-wait protocol. What data transfer rate can be achieved? What is the utilization of the link?

$Bw = 40$ Kbps $\quad L = 1\,KB = 8\,Kb.$

$T_t = \frac{8\,Kb}{40\,Kb/s}$ $\quad T_p = 1.5$ sec.

$\Rightarrow \frac{1}{5}$ sec.

$\Rightarrow 0.2$ sec.

$\therefore$ Total delay $= T_t + 2T_p = 3.2$ sec.

$$\text{Throughput} = \frac{L}{\text{delay}} = \frac{8\,Kb}{3.2\,sec} = 2.5\,Kbps$$

$$\text{Efficiency} = \frac{T_t}{\text{Delay}} = \frac{0.2}{3.2} = \frac{1}{16} = 6.25\,\%$$

- **Throughput:** Amount of data transferred upon time taken.

$$Th. = \frac{L}{T_t + 2T_p}$$

$$\Rightarrow \frac{L/T_t}{1 + 2\frac{T_p}{T_t}}$$

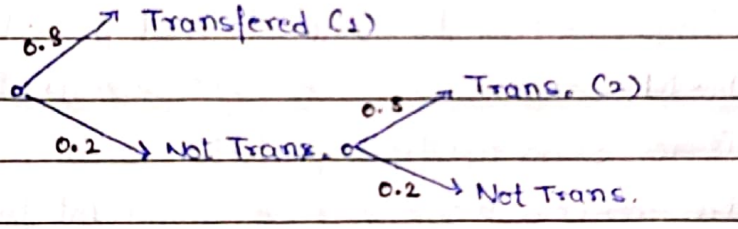
$$\Rightarrow Bw \times \eta$$

$$\therefore Th. = Bw \times \eta$$

Throughput = $\eta \times Bw$.

Hence throughput is the effective Bandwidth (rate of transfer).

Ex. On a wireless link the prob. of packet error is 0.2. A stop-and-wait protocol is used to transfer data across the link. The channel is independent for each transmission. What is the avg. number of transmissions required to transfer 100 packets?



$$E(\text{No. of transfer}) = P(\text{Trans}) \times 1 + P(\text{Not Trans})\,[1 + E(\text{No. of Trans})]$$

$$\Rightarrow \quad E(x) = 0.8 + 0.2\,(1 + E(x))$$

$$\Rightarrow \quad 0.8\,E(x) = 1$$

$$\Rightarrow \quad E(x) = \frac{1}{0.8} = \frac{10}{8}$$

So, for 100 packets $= 100 \times \frac{10}{8} = 125$

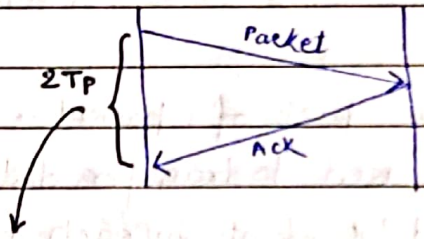
* Bandwidth Delay Product: Gives the amount of data that can fill the channel, in one Round Trip time.

$$BW\text{-}D = Bw \times 2T_p$$

$$\Rightarrow BW \times RTT$$

* Sliding Window Protocol

In stop-and-wait protocol we send a packet and wait for it to be acknowledgeded before sending another packet.



In this duration no work is being done while multiple packets couldve been sent.

Sliding-Window protocols allow for multiple packets to be sent before receiving acknowledgement of first packet.

(1) Go-Back-N (2) Selective Repeat

The idea is to send multiple packets within 2Tp duration, because the acknowledgement for the first packet arrives after 2Tp time.

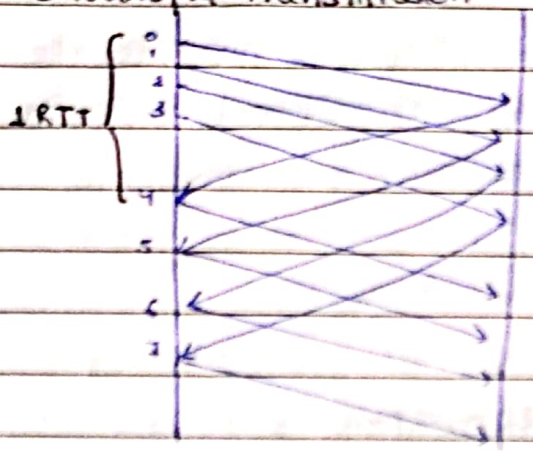
* Go-Back-N

- The sender has a "window" of N packets which can be sent without receiving acknowledgment. Receiver window is always of size 1.
- Cumulative Acknowledgement: If Ack. is received for packet number n, it means that all packets upto packet (n-1) are received successfuly.

For each sent packet, a timer is started.

Ex. N=4. Packets 0 1 2 3 4 5 6 7 8 9.

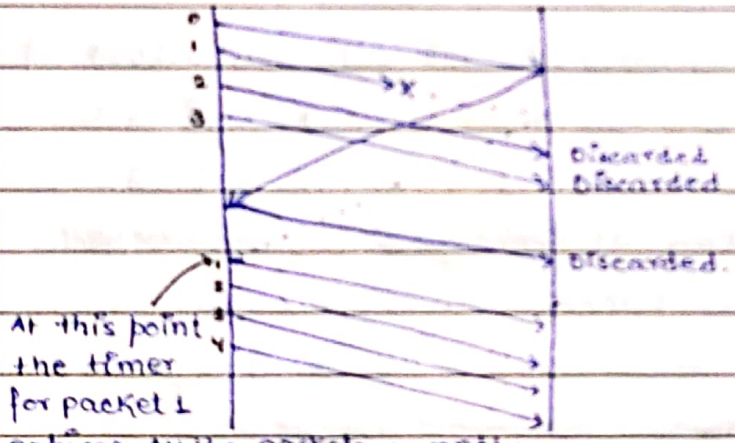
C1: Successful Transmission



There is no data loss in this case.

C2: Packet loss.

Suppose packet 1 gets lost.



when packet 1 isn't received, receiver discards all the subsequent packets and sends Ack for packet 1 only. (silently Discard)

C3: Ack loss.

Say Ack 2 gets lost.

In this case the sender's timer would expire while waiting for Ack & sender would resend, all packets in the window.

Basically, the receiver expects packets to come one at a time & in order (of seq. no.) & sends ack in the same sequence.

c4: Ack is delayed

Say Ack2 gets delivered to sender before Ack 1. This allows the sender to move window over from 0 & 1 & assume them to be delivered. This is infact the case. Since receiver only generates Ack in a sequence, packet 0 & 1 must have be received before sending Ack 2.

- Sequence Number & Window Size.

Suppose we have 4 unique seq. no. & window size is also 4.

[0 1 2 3] 0 1 2 3

say all four packet in window get delivered but Ack of all four get lost. In this case timer expires & the sender resends all packets: 0, 1, 2, 3. But since receiver already acked all four packets its expecting to receive a new packet with sequence number 0, but gets an already received packet with seq. no. 0 & receiver accepts it.
This is wrong.

∴ window size < 2^m

where m is no. of bits for seq. no. field.

- If a packet is corrupted, receiver silently discards the packet.

* Selective Repeat

- Sender and Receiver window size are same and are equal to $\frac{2^m}{2} = 2^{m-1}$

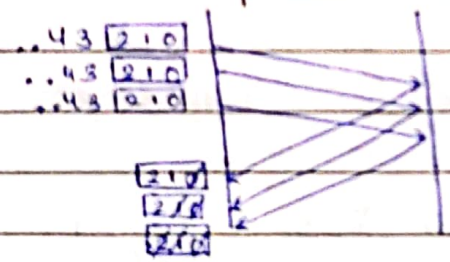
  where m is the no. of bits for sequence number field.

- Acknowledgement is individual. Ack n means the n^{th} packet has been received.

- A timer is started for each individual packet.

Ex. N=3 Packets: 0, 1, 2, 3, 4, 5, 6...

C1: Successful Transmission.



C2: Packet lost

Suppose packet 1 is lost, then packet 2 is still accepted and saved in received buffer. When timer for packet 1 expires only packet 1 is resent, not all the packets in the window.

C3: Ack is lost

Suppose Ack 1 is lost. When Ack 0 is received, window slides. When Ack 2 is received, 2 is marked sent. Since Ack for packet 1 hasn't been received, window doesn't slide on Ack 2. When timer for packet 1 expires only packet 1 is resent. On receiving Ack 1 window immediately slides to [3, 4, 5].

C3: Data is delivered delayed

Suppose packet 2 delivers before 1. Receiver accepts out of order packet 2 & mark it received and sends ACK 2. When p1 is received Ack 1 is sent and sender slides its window.

C4: Ack is delayed

Suppose ~~p~~Ack 2 is delivered before Ack 1. Then P2 is marked sent, but window doesn't slide. When Ack 1 is received then window slides over to [3,4,5].

C5: Packet is Corrupt

Silent discard works. Optionally a NegAck can be sent so that sender doesn't ~~send~~ wait for timer to expire for resend.

- Sequence Number & Window Size.

Suppose we have $m = 2$, so no. of unique seq. nos. is $2^2 = 4$ & let's suppose window size is $> 2^{\frac{m}{2}}$. Lets say $2^{\frac{m}{2}} + 1 = 3$.

[0 1 2] 3 0 1 2 3 0 1 ...

Suppose all 3 packets: 0, 1, 2 are received but Ack is lost. Receiver now expects: 0 1 2 [3 0 1] 2 3, since it already sent Ack. Sender resends packet 0, 1, 2 again, but receiver accepts packet 0 & 1 as new packets & discards packet 2.

So, window size can't be $> 2^m/2$.

$$\therefore \quad N \leq 2^{m-1}$$

- In general for any protocol:

Sender Window + Recv. Window $\leq 2^m$

$$\boxed{N_S + N_R \leq 2^m}$$

$\therefore$ There should be atleast $N_S + N_R$ unique seq. nos.

* Sliding Window (Contd.)

- Efficiency:

$$\eta = \frac{N_S \times T_t}{T_t + 2T_p}$$

- Throughput:

$$Th. = \frac{N_S \times L}{T_t + 2T_p}$$

$$\Rightarrow \frac{N_S \times L/T_t \times T_t}{T_t + 2T_t}$$

$$\Rightarrow N_S \times B_W \times \eta$$

$\therefore$ Throughput $= N_S \times B_W \times \eta$

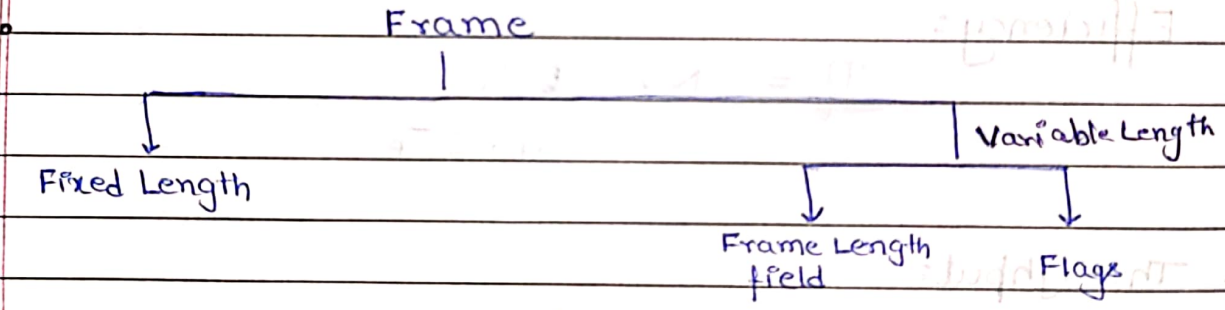
* Framing

On a link each frame is just a sequence of bits. How does the receiver know on which bit frame starts and on which bit frame ends?

Ex. 01100110101011010001

There is no way to know which part is which frame, since frame might be of variable length.

Framing is a way to mark the beginning and ending of a frame.

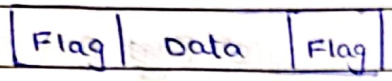


Fixed length frame waste lot of space if frames are very small in size.

Some protocols such as Ethernet use Frame Length field of fixed size at the very beginning of the frame to indicate the frame size to receiver.

Flags are added to the beginning and end of each frame, to differentiate them.

Flags



- Bit Stuffing
- Byte Stuffing

The problem with adding flag at beginning and end of data is that the flag might also be present in data itself.

Ex. Flag = 101 Data = 001101001010101

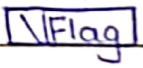
101 001101001010101 101

So receiver would think this as start.

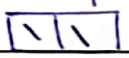
And this as end.

To solve this problem we have two ways.

(1) Byte Stuffing (Character stuffing): Wherever in the data, there occurs flag, it is escaped.



But what if data itself has \ escape char?
We escape the escape char as well:



Ex. Flag: $ Data: a0z1_$e\$95

Escaped Data: a0z1_\$e\\\$95

Frame: $a0z1\$e\\\$95$

(2) Bit Stuffing: If data contains flag pattern, we just stuff one bit.

The flag is generally of fixed pattern: 01111110.

The idea is to stuff a '0' whenever in data 0 followed by five 1s is seen.

Ex. Data: 00101011111101011111011111110

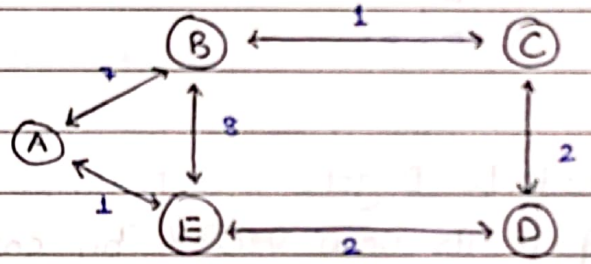
Stuffed: 001010111111001011110111111010

Network Layer

* **Routing Protocols:** Protocols used to populate the Routing Table for routers in a network. These information is then used by the routers for forwarding packets through the network.

The network is thought of as a graph with router as vertices & weighted edges b/w them. The edge weights can be a varity of things, such as number of hops, propagation time, etc.

Ex.



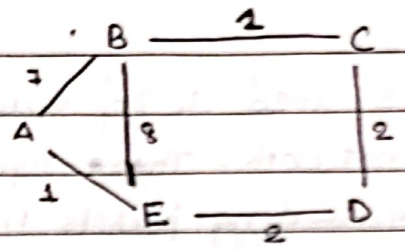
There are two popular routing algorithms:
(1) Distance Vector Routing
(2) Link State Routing.

* Distance Vector Routing

Initially each router has a distance vector, which is just a mapping b/w router id & distance to the router. The routers then shares this vector with the neighbours.
The process of sharing and updating repeats until all routers converge.

Note: Distance Vectors don't carry next hop info.

Ex.



Initially:

A:

A	0
B	7
C	∞
D	∞
E	1

B:

A	7
B	0
C	1
D	∞
E	8

and so on.

After this, say at node D. D gets the Distance Vector from E & C. D will compute its new vector by comparing values from E & C DVectors.

E:

A	1
B	8
C	∞
D	2
E	0

C:

A	∞
B	1
C	0
D	2
E	∞

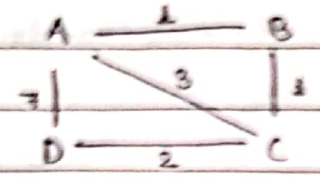
D:

A	min(D→E+E→A, D→C+C→A)
B	min(D→E+E→B, D→C+C→B)
C	2
D	0
E	2

∴ D:

		Next Hop
A	3	E
B	3	C
C	2	C
D	0	–
E	2	E

Ex.



Initial:

A:		
A	0	
B	1	
C	3	
D	7	

B:		
A	1	
B	0	
C	1	
D	∞	

C:		
A	3	
B	1	
C	0	
D	2	

D:		
A	7	
B	∞	
C	2	
D	0	

Round 1:

For A

		Nbr
A	0	-
B	1	B
C	2	B
D	5	BC

For B

		Nbr
A	1	A
B	0	-
C	1	C
D	3	C

For C

		Nbr
A	3	AB
B	1	B
C	0	-
D	2	D

For D

		Nbr
A	5	AC
B	3	C
C	2	C
D	0	-

Round 2:

A:			
A	0	-	
B	1	B	
C	2	B	
D	4	B	

B:			
A	1	A	
B	0	-	
C	1	C	
D	3	C	

C:			
A	2	B	
B	1	B	
C	0	-	
D	2	D	

D:			
A	4	C	
B	3	C	
C	2	C	
D	0	-	

Round 3:

A:			
A	0	-	
B	1	B	
C	2	B	
D	4	B	

B:			
A	1	A	
B	0	-	
C	1	C	
D	3	C	

C:			
A	2	B	
B	1	B	
C	0	-	
D	2	D	

D:			
A	4	C	
B	3	C	
C	2	C	
D	0	-	

- **Count-To-Infinity Problem:** Good news travels fast, bad news travels slow.

Ex. Suppose initially this is the case:

A B —1— C

∴ Dist. to A for A, B, C resp.: 0, ∞, ∞.

Let's say link is established b/w A & B:

A —1— B —1— C

Round 1: Dist. to A: (A, B, C) = (0, 1, ∞)

Round 2: —" —: (A, B, C) = (0, 1, 2)

So, good news travelled fast.

Now suppose link to A goes down:

A B —1— C

After this B passes its DV to C & C passes its DV to B.

R1: (A, B, C) = (0, 1+2, 1+1) = (0, 3, 2).
(B→C, C→A)

R2: (A, B, C) = (0, 1+2, 1+3) = (0, 3, 4)
(C→B, B→A)

R3: (A, B, C) = (0, 1+4, 1+3) = (0, 5, 4)
(B→C, C→A)

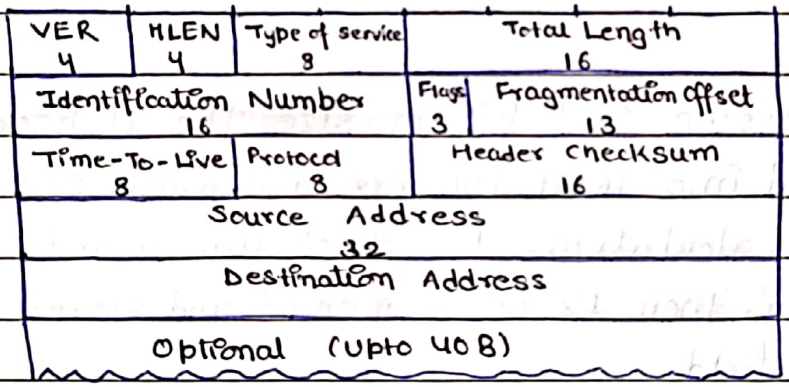
⋮

So eventually the values go to infinity for B & C, but very slowly.

There is a simple solution where C while sharing DV, it also shares next hop info, and vice-versa.
B seeing that C goes to A through B only doesn't use that info.

* Link State Routing: Pending

* IP Header



- VER: Only two versions got popular. IPv4 & IPv6.
- HLEN: No. of bytes in header. This can range from 20B to 60B. Since field is of 4 bits only, we divide Header size by 4, i.e. 20B to 60B → 5 to 15
- Type of Service: Describes the type of service according to RFC 791.
- Total Length: (Header + Data) length.
  Range: 20B to $(2^{16}-1)$ B
  (Min Header Size)
- Time-To-Live: The number of hops a packet can jump before reaching its destination. Prevents n/w loops and misrouting. Each hop decrements while sending out.

If packet arrives at destn with TTL 0, the destn accepts it. Intermediatte router discards such packet.

- Protocol: Used to signify which higher level protocol is being used, such as. TCP, UDP, ICMP, etc.

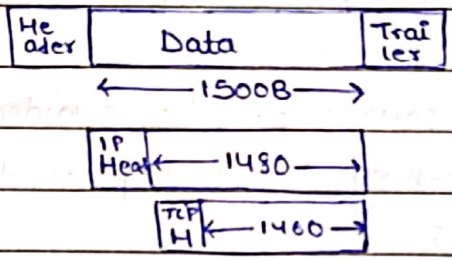
- Header Checksum: Used to store checksum of IP header. Needs recalculation each time IP header is modified intentionally, such as TTL decrement.
It is not a security measure. IP is an unreliable protocol.

Since checksum is 16 bit in size the IP header is grouped into 16 bit integers and added. During checksum calculation, the checksum field is all 0s. The sum is then 1's complemented and stored in the checksum field.

- Options & Padding: Used for nlw testing and debugging purposes, such as:
(1) Source routing
(2) Record Route
or just padding.

* Fragmentation

We know that an IP packet (Header + Data) can have size in range 20B to $2^{16}-1$, but there exist data link layer protocol which allow for much smaller IP packet (or any data) to be carried in a frame. Ethernet for example allows only 1500B.



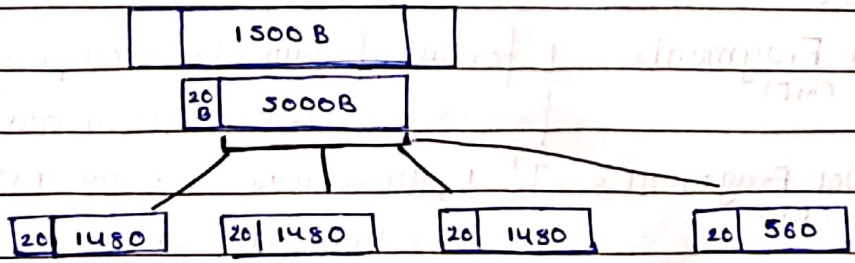
So in ethernet protocol only 1460B of Application data is allowed at max.
Here 1460B would be called Maximum Segment Size & 1500B would be called Maximum Transmission Unit.

- Maximum Transmission Unit: Amount of data allowed at maximum in a DL Layer frame. (without Header).

- Maximum Segment Size: Maximum Segment size (Application data) allowed, calculated using MTU.

- What if the IP data + header exceeds frame capacity? In this case the data would have to be partitioned.

Ex



This process is called Fragmentation.

- Since DataLink layer protocols are hop to hop.. the MSS b/w two links might change, so there may be fragmentation carried out by source as well as intermediatte routers.

In IPv6, fragmentation only happens at the source.

- Fragmentation at source can be avoided in IPv4 by Transport layer checking the DLL protocol used and adjusting its MSS.
This doesn't prevent fragmentation at intermediatte hops.

- Reassembly of all fragments happens only at the destination since fragments might go through different paths.

- IP Header fields:

(1) Identification Number: At the source before any fragmentation, the IP packet is assigned Id. No. Whenever this packet is fragmented the Id. No. is copied to each fragment so that at the destn all the fragments with same Id. No. are reassembled.

(2) Flags:

More Fragments (MF): 1 for all fragments except for last fragment. Last fragment has MF 0.

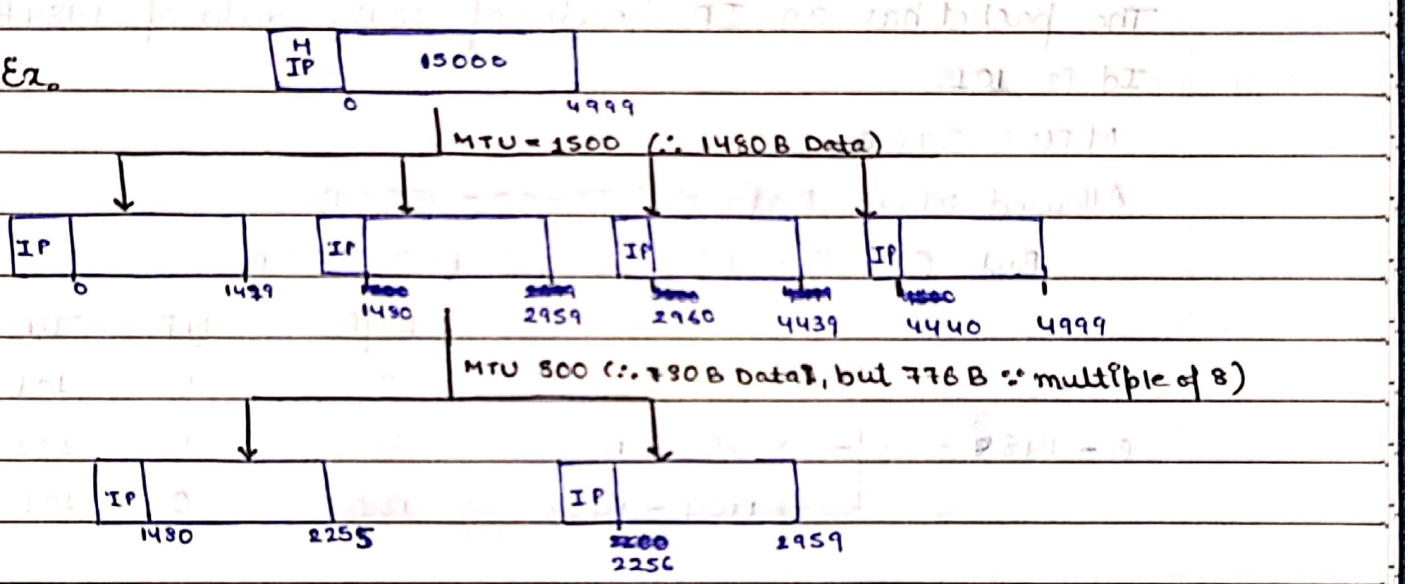
Do Not Fragment (DF): If 1, then whenever the MTU is smaller than IP packet size, router drops the package rather than frag.

and ICMP message is sent to Source.

Reserved.

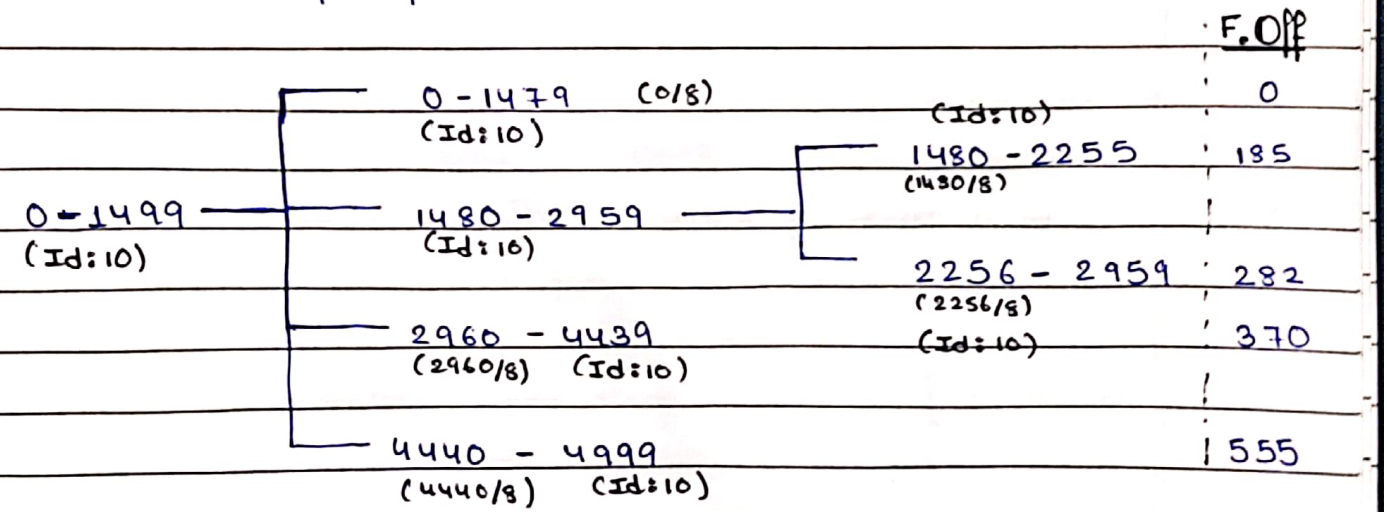
(3) Fragmentation Offset : This field is used to sequence the fragments of a single packet.

Ex.



So, Fragmentation offset is calculated as $\frac{\text{first byte index}}{8}$, because FO field is 13 bits & Total len is 16 bit field.

Since we do first byte index /8, the (MTU - IP Header size) must be multiple of 8, or we have to choose smaller value.



Based on the fragment offset the destn arranges the fragments and reassembles them before passing to upper layer.

Ex. Packet is to be forwarded to n/w with MTU of 576 B. The packet has an IP header of 20 B & data of 1484 B. Id is 101.

MTU = 576 B.

Allowed Max. Data = 576 - 20 = 556 B

But 556 % 8 ≠ 0, so we use 552 B

		FOff	MF	Id	Total len
0 - 1483 (3)	→ 0 - 551	0	1	101	572
	→ 552 - 1103	69	1	101	572
	→ 1104 - 1483	138	0	101	400.

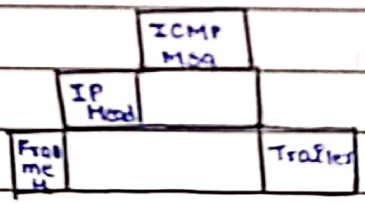
* Internet Control Message Protocol

IP is unreliable protocol. ICMP is an effort to try & provide some level of reliability although not completely.

- ICMP tries to answer:

(1) What happens if a router must discard datagram because:
   (a) Can't find a route to destination
   (b) OR TTL field has 0 value.

(2) What happens if dest^n must discard all received fragments because it didn't receive all frags. within time limit?

- ICMP reports errors to the source host only.
- ICMP also allows to check health of nodes with query messages.
- 

- Some important points about ICMP:

(1) No • ICMP error msg. is generated in response to a datagram carrying an ICMP error msg.

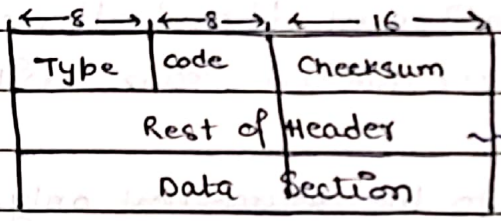
(2) No ICMP error msg would be generated for a fragment datagram if it isn't first fragment.

(3) No ICMP error message for datagram having multicast addr. or 127.0.0.1 or 0.0.0.0.

- Error Reporting Message: Report problem to source that a router or destⁿ host may encounter during processing of an IP packet.

- Query Message: Help get info about n/w such as link state or RTT, etc.

- Error Reporting Message.



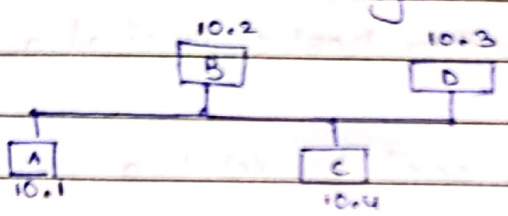
(1) Destination Unreachable.
(2) Source Quench
(3) Time Exceeded
(4) Parameter Problem
(5) Redirection

- Query Message.

(1) Echo message: Check reachability of some hop.
(2) Timestamp message: Get RTT.

* Address Resolution Protocol

- ARP helps a host within a LAN find the hardware address (MAC addr) using the IP address.



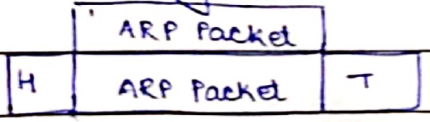
Say A wants to know the MAC addr of host with IP 10.3. It broadcasts an ARP message. D sees the message and unicasts ARP reply back to A with its MAC addr.

- ARP Packet

Hardware Type		Protocol Type
H/w Len	Protocol Len	Operation (1. Request 2. Reply)
Sender H/w Addr		
Sender protocol addr (Say IP Addr)		
Target H/w Addr		
Target Protocol addr (Say IP)		

In ARP request message the Target H/w addr is set to all 1s. In Response message target host replaces the field with its hardware addr.

ARP packet is directly put as data in DL layer frame.



- Usecases of ARP

(1) Sender of ARP packet is a host and wants to send a packet to another host in the n/w:
Use ARP to find another host's physical addr.

(2) The sender wants to send packet to a host in a diff. n/w:
- Sender looks at its routing table.
- Find the IP addr. of the nexthop (router).
- Use ARP to find m/c addr. of next hop.

(3) The sender is a router which received a packet to send to host in another n/w:
(Same as 2)

(4) The sender is a ~~the~~ router which wants to send a packet to host in same n/w:
(same as 1)

- In ARP request target Physical Addr. is set to all 1s for broadcast.

Transport Layer

- Transport Layer is required for process-to-process communication. TL provides service to Application Layer & receives service from Network Layer.

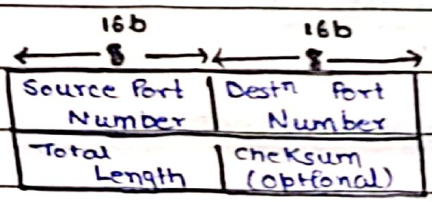
- To uniquely identify a processaron a host, 16 bit Port Numbers are used. The range of port numbers: 0 to $2^{16}-1$.
  Port Numbers 0 to 1023 are reserved for standard applications. Such as HTTP, FTP, etc.
  1024 to $2^{16}-1$ are ephermal port numbers.

- Responsibility of Transport Layer:
  (1) Reliability & Flow Control
  (2) Congestion Control.

* User Datagram Protocol

UDP is a connectionless, unreliable protocol for simple application to application communication.

- Header



Total length: 0 to $2^{16}-1$ B
Header is always 8 Byte
∴ Total length: 8 to $2^{16}-1$ B

* Transmission Control Protocol

- TCP is a full duplex, connection-oriented, reliable protocol. Both parties can send and receive at the same time.

- TCP is byte oriented protocol, i.e., each byte has an id., and TCP logically streams the bytes from one process to another.

- TCP has three phases:
  (1) The two TCPs establish logical connection.
  (2) Data is exchanged in both direction
  (3) Connection is torn down.

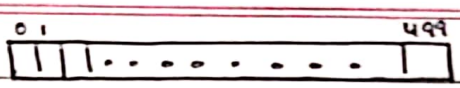
- TCP uses acknowledgment mechanism for reliable delivery.

- TCP numbers all bytes. These bytes are then packet into segments of n bytes each. The segment size can differ from segment to segment. Each segment is attached with TCP header.

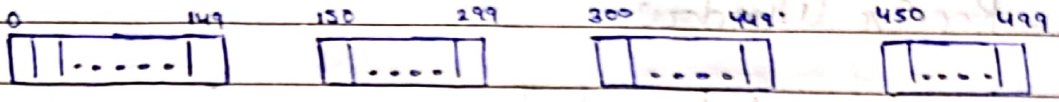
Sequence Number: This field in TCP header is used for sequencing acknowledgement, etc. After segments are created they are assigned a 32-bit sequence number as follows:

(1) The first segment is assigned a random sequence number in range 0 to $2^{32}-1$.

(2) The sequence no. of other segments is calculated as sequence number of previous segment + no. of bytes in previous segment.

Ex. 

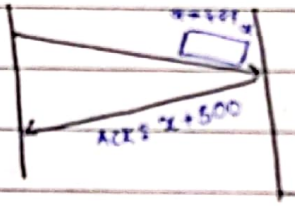
500 B.

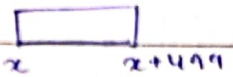


Seq. No	Random (0, $2^{32}-1$)	2152 + 150	2302 + 150	2452 + 150
	2152	2302	2452	2602

Acknowledgment Number: It is the number or index of next byte the receiver expects to receive.

Ex.





ACK: x+500

Acknowledgment in TCP is cumilative

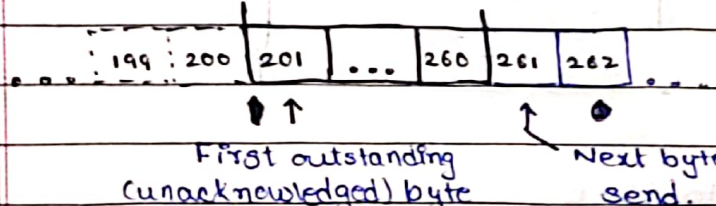
Ack No. is calculated with Initial Sequence No. x:

x + no. of bytes received ⇒ Index of next byte.

• Windows in TCP

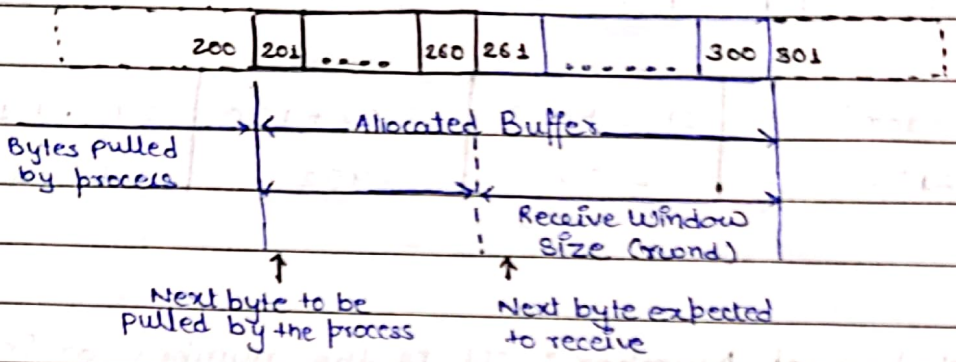
TCP is full-duplex & hence both parties have: Send Window & Receive Window.

Send Window:



So send window has two pointers.

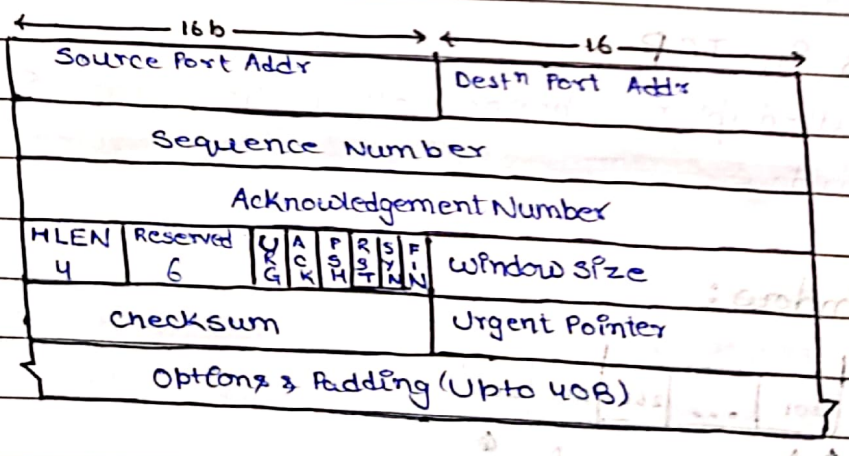
Receive Window



The receiver allocates some buffer of fixed size. The buffer has two pointer, one points to the already received byte which the process hasn't pulled yet. & the other points to the next byte expected to be received.

The effective receiver window size in bytes is calculated as bytes allocated to buffer - bytes outstanding to be pulled by process.

- TCP Header



Header size: 20B to 60 B.

Source Port Addr: Uniquely identifies sending process on a host

Destn Port Addr: Uniquely identifies receiving process on a host.

Sequence Number: Seq. No. of TCP Segment

Acknowledgement Number: Index of next byte expected to be received. Acknowledgment can be piggy-backed, i.e. it can be sent along with data if there's data to be sent.

HLEN: Scaled by dividing by 4.

20 to 60 → 5 to 15.

Flags:

(1) ACK: If set, this means Ack Number is valid & not garbage.

(2) RST: If set, the connection has to be reset.

(3) SYN: Set only for first packet. When set, it means the packet is sent for Synchronization purpose.

(4) FIN: Set only for last packet. When set, party wants to end connection.

(5) PSH: Push flag. Generally the receiving TCP waits for the process to pull the data. If the sender wants the receiving TCP to push the data to application process as soon as segment is received, it sets this flag.

(6) URG: When set, this means Urgent Pointer field is valid. The Urg. pointer is used to indicate that from segment start to the pointed byte the data is of high priority and should be treated as so.

Options

Window Size: The receiver using this field indicates the current capacity of allocated buffer. (rwnd = allocated size - outstanding size) so that sender doesn't send more than rwnd bytes.

Sometimes 16 bits are not enough to indicate a rwnd, so in options we have additional 14 bits for window size, so total 30 bits.

Checksum: It is calculated using both TCP Header, Data, and IP Header fields such as IP addr (src, destn).

* TCP Connection Establishment

Although TCP is two way full duplex protol, only one of the two parties can initiate the connection. The party which can initiate the connection is called TCP client & the other which waits for incoming connections is called TCP Server.

Suppose a client process wants to establish a connection with some server process. It first informs client TCP that it wants to establish connection

For the establishment of connection the client TCP sends the first msg informing server TCP that it wants to send some data. Since this is the first packet client → server, SYN flag is 1. This is just synchronization packet and the server may accept/reject the request, so just dummy data of 1 Byte is sent with Seq. No. x.

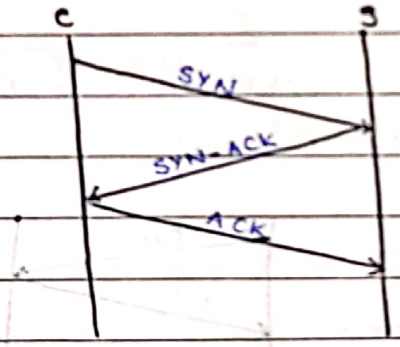
The server reply for the client request has ACK flag 1, since it is replying, also SYN flag is 1, since first packet server → client. Ack. No. is set to x+1, since 1B dummy data was sent.

Since server also wants to send some data to client. It also sets Seq. No. to some value y & adds one byte dummy data.

Since both these packets had 1B dummy data they both consumed 1 sequence no.

Upon receiving Server's reply client finally acknowledges receiving byte with seq. no. y & sets Ack flag 1 & Ack No. as y+1. This packet may be piggybacked with actual application data & can consume sequence number or it can just be pure acknowledgment and not consume sequence number.

While accepting (acknowledging) request to send data both side also advertise their Window Size (rwnd).



Ex.	SYN	ACK	Seq. No.	Ack No.	RWND	Consumes Seq No?
Cl. $\xrightarrow{SYN}$ Sv.	1	0	5000	–	–	Yes. 1B Dummy data.
Svr. $\xrightarrow{SYN/ACK}$ Cl.	1	1	8000	5001	3200	Yes. 1B Dummy data
Cl. $\xrightarrow{Ack}$ Svr	0	1	5001	8001	2100	Yes, if data is sent, no otherwise.

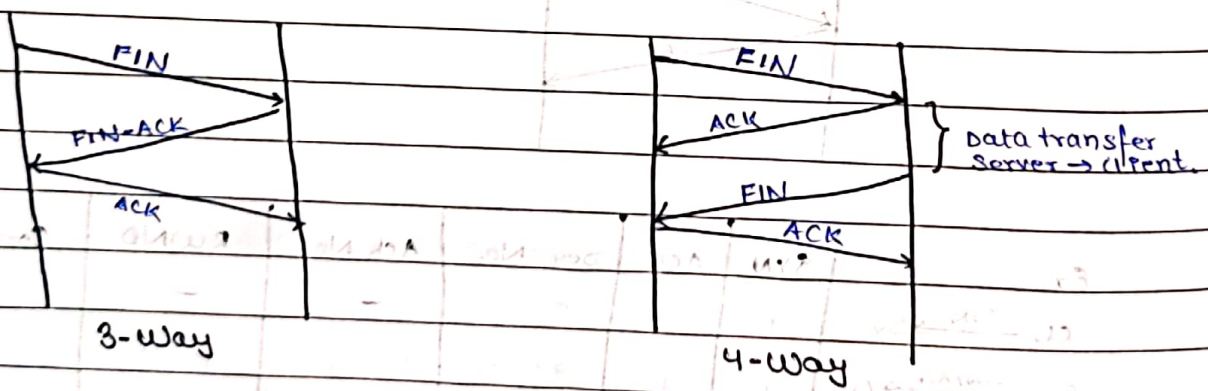
The first two packet consume sequence no. because they have to be acknowledged.
Last packet if contains no data doesn't require acknowledgment.

1. TCP Connection Termination

Termination can happen in one of two ways:

(1) 3 Way Handshaking (2) 4 way Handshaking

- Client when it wants to terminate connection sends FIN packet which is just the last packet with FIN flag 1. It may or maynot contain data but consumes one seq. no.
- The server may respond in one of two ways. If server also wants to terminate, it sends FIN+ACK packet. This packet may/maynot contain data. This also consumes one seq. no. If the server has data left to send it just sends Ack. If server sends FIN+ACK it is 3 way handshake & if ACK is sent, 4 way handshake.
- The client replies with Ack packet in 3 way Handshake. This packet also may/maynot contain data.

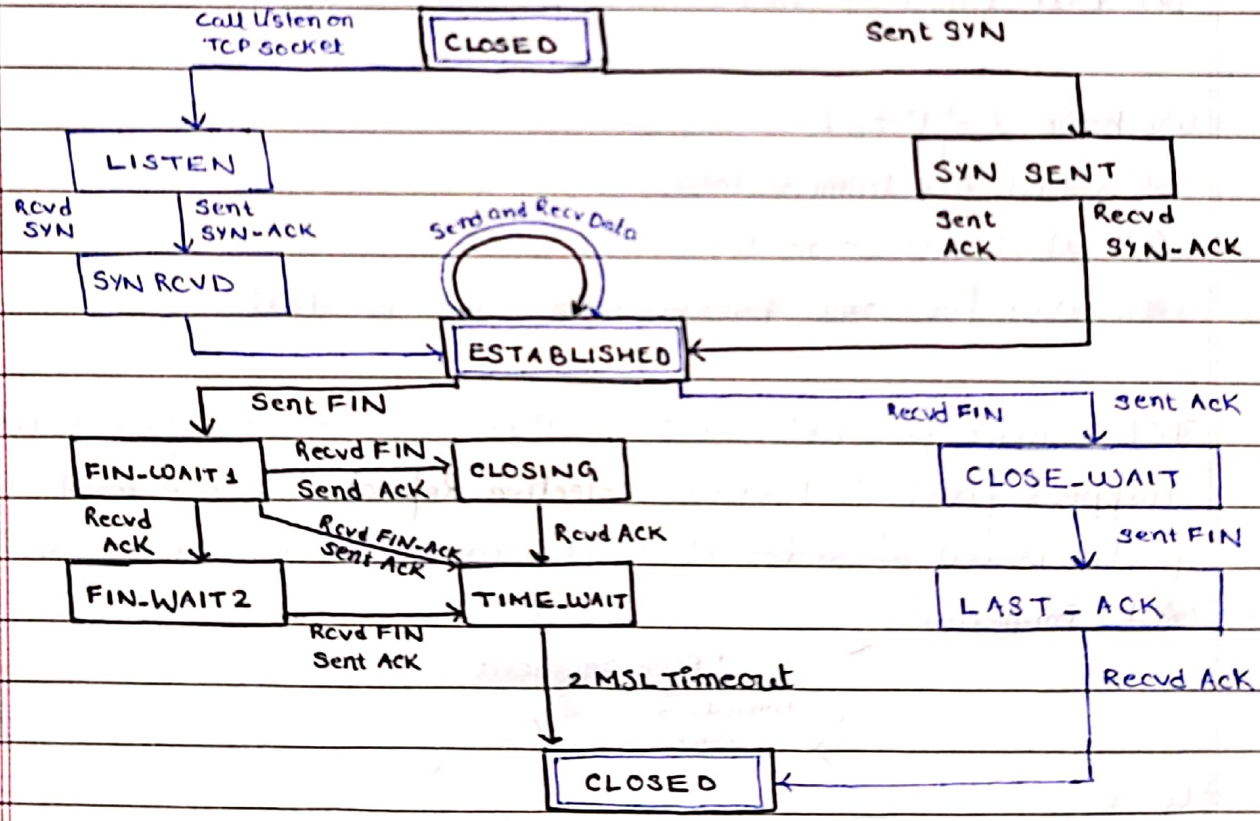


In 3-way FIN & FIN-Ack always consume 1 or more seq. no. depending on if they contain data. Ack maynot carry data & here maynot consume seq. no.

In 4 way, FIN, FIN (in sequence) would consume 1 or more seq. no. depending on if they carry data. The Ack from S→C may/maynot carry data & hence may/maynot consume seq. no. Ack from C→S can't carry data since connection is terminated, so doesn't consume sequence number.

- TCP States

→ Client
→ Server



Client is Active closing. Server is passive closing.

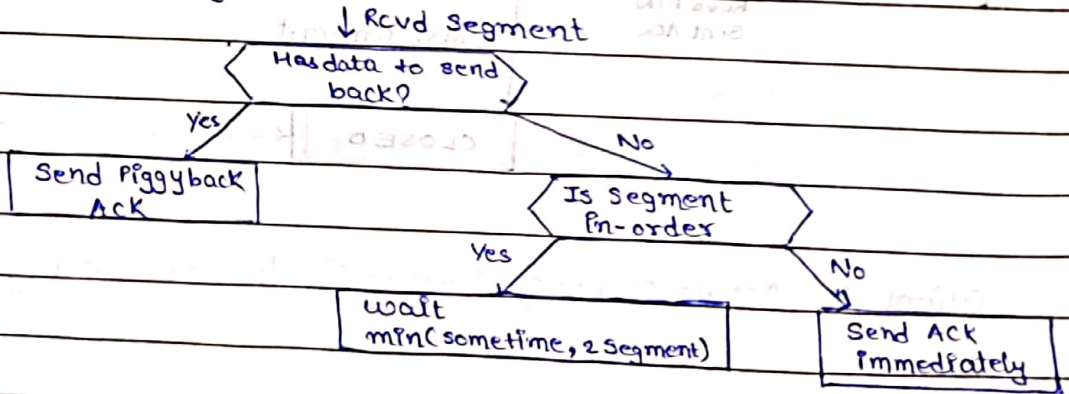
• Acknowledgement in TCP

Acknowledgements in TCP are cumulative and can be piggypacket.

When a receiver receives a segment it has to send ACK:
(1) If receiver has some data to send, it will piggyback the acknowledgment immediately, without any delay.
(2) But what if there is no packet to send back?

We have 3 options:
(1) Send Ack immediately.
(2) Wait for some time.
(3) Wait for some packets and send cumulative ack.

TCP waits for both: (1) Some time (2) for 2 segment, whichever happens first. TCP uses **Selective Repeat**. If the ~~out~~ recvd packet is out of order it can't cumulatively send Ack, so it sends Ack immediately.



- TCP receiver may receive data out of order and store in temporary buffer, but TCP is reliable protocol & hence delivers data application layer in-order only.

- Retransmission

- When a sent packet is lost (not delivered), the timer started for the packet expires and the packet is sent again.

- It might also happen that a packet gets lost but all other packets get delivered, and sender receives multiple Acks with Seq. no. of lost packet. After sender TCP receives 3 Duplicate Ack, i.e. 3 ack with seq. no. of lost packet it retransmits.

- The first scenario is: Retransmission on Timeout. Second is Retransmission on 3 Duplicate Acks.

- In second case the sender knows that although 1 packet is lost other (at least 3) packets got delivered.

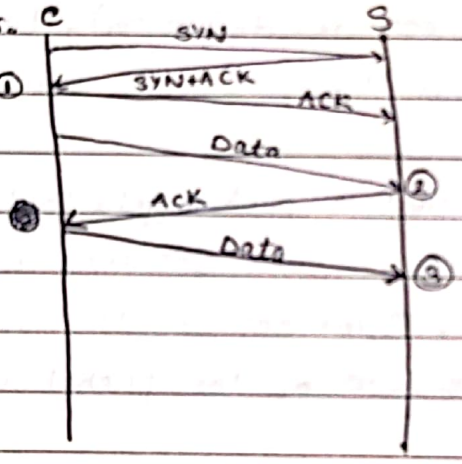
- Since the second case doesn't wait for long timeout it is also called Fast Retransmission.

- Timeout indicates that congestion (traffic) in the n/w is very high, since no Ack for the lost packet was received. Receiving 3 Ack indicates that congestion is high but not as much, since other packets got delivered.

• Flow Control

Flow Control in TCP is done with help of Window size of receiver (rwnd). The sender decides its send window based on the advertised rwnd.

Ex. Lets suppose client is the sender and server is the receiver.



① Receiver suppose in SYN+ACK packet header advertises x B window size to the sender, i.e.

$$rwnd = x.$$

Sender adjusts its send window accordingly.

② After receiving data, say t B, $(t \leq x)$, the receiver has t B outstanding data to be pulled by application process, so new rwnd, allocated buf. - outstanding size

$$rwnd = x - t$$

Sender again adjusts its send window.

③ Suppose b/w 2 & 3, k bytes $(k \leq t)$ got consumed by app. process

so, $rwnd = x - t + k$ (k freed.)

and so on.

Congestion Control

Congestion is traffic in the n/w. It can happen as a result of various problems, such as packet drops, TTL expires, link failures, etc.
Because of congestion, the sent packet might get lost or the acknowledgment may get dropped.
Since the IP protocol doesn't take care of n/w congestion explicitly, TCP does.

Congestion is detected by TCP based on two heuristics:
(1) Timeout: When a packet is sent, timer is started. If timer expires, its sign of congestion.
(2) 3 Duplicate ACKs: When three ACK packets have same Ack No.

Congestion is handled based on an algorithm with two phases:
(1) Slow Start: Exponential Increase
(2) Congestion Avoidance: Additive Increase.
The algorithm switches b/w the two phases in different situations.

(1) Slow Start

In this phase the congestion window (cwnd) starts with size 1 MSS.
There are two assumptions:
(1) Every packet size is equal to Max. Segment Size.

$$MSS = MTU - 20 - 20 \quad \text{(generally)}$$

(Min IP Header, Min TCP header)

(2) Every packet is individually acknowledged.

Congestion Window: Theoretical window size of which is adjusted based on Congestion control phase & n/w traffic. (cwnd)

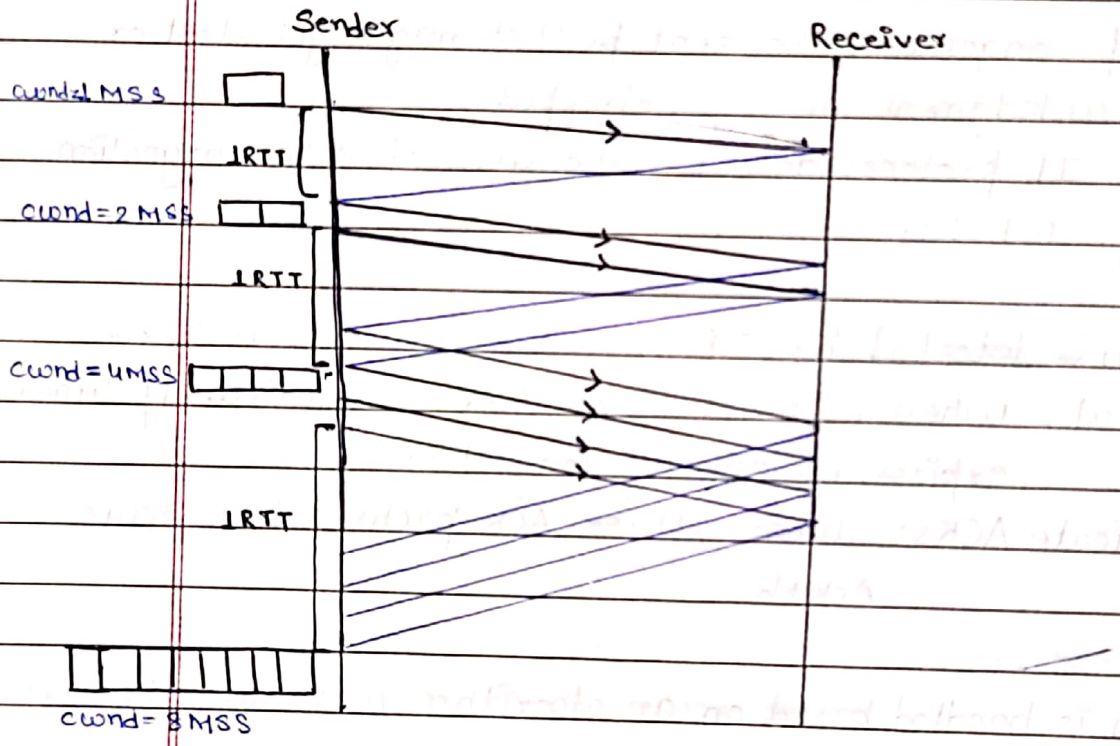
• cwnd is measured in MSS.

Therefore,

Send Window Size = min (rwnd, cwnd).

Slow Start phase says: If an Ack arrives,

cwnd = cwnd + 1 MSS



In 1 RTT : cwnd = 2 × cwnd.

So, after every Round Trip Time, i.e time it takes to deliver packet and receive acknowledgement, the cwnd doubles, in case of 1 Ack per packet.

The situation would be different for 1 Ack for 2 packets.

Initial: 1 MSS

Send packet. Receiver timeouts and sends Ack. ⇒ 2 MSS

Sends 2 packets. Receiver sends 1 Ack. ⇒ 3 MSS.

Sends 3 packets. Receiver sends 1 Ack & waits. ⇒ 4 MSS. If receiver timeouts it sends another Ack ⇒ 5 MSS. else sender sends another packet & cwnd = 5 MSS only.

The general assumption while solving problems is 1 Ack per packet.

(2) Congestion Avoidance.

In this phase, cwnd is increased once Ack for all the packets in current window arrive, i.e. in 1 RTT

$$cwnd = cwnd + 1$$

After 1 Ack arrives:

$$cwnd = cwnd + \frac{1}{cwnd}$$

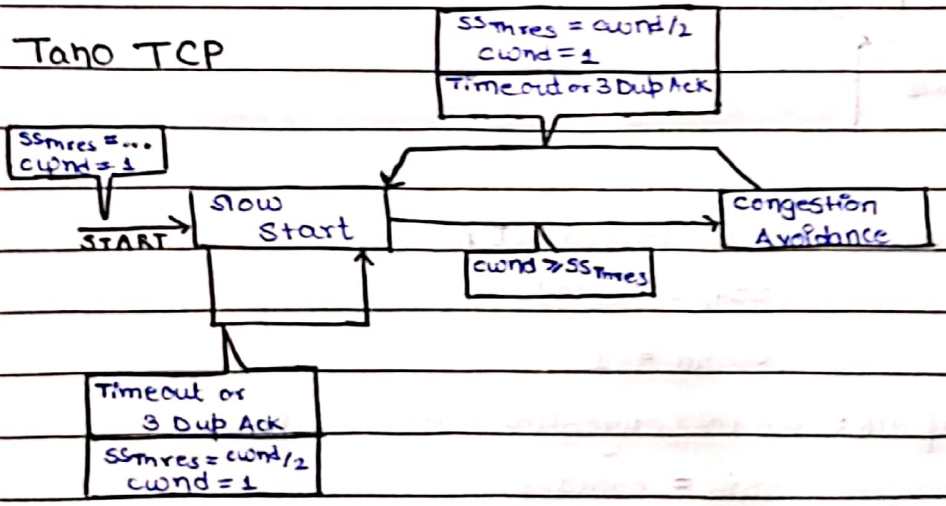
SlowStart Threshold : The size of cwnd, which when reached or exceeded, we move from Slow Start to Congestion Avoidance phase.

i.e. SS $\xrightarrow[cwnd \geq SS_{thres}]{}$ CA

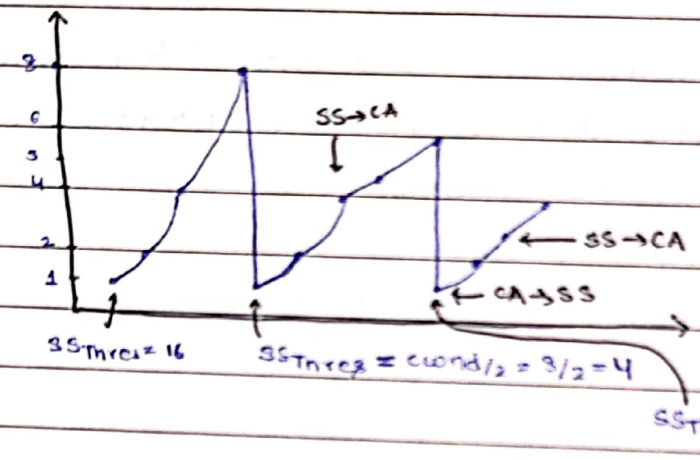
Congestion can occur in either of the phases due to either of the two problems: Timeout or 3 Dup Ack.

The process of handling the congestion depends on implementation.

Tahoe TCP

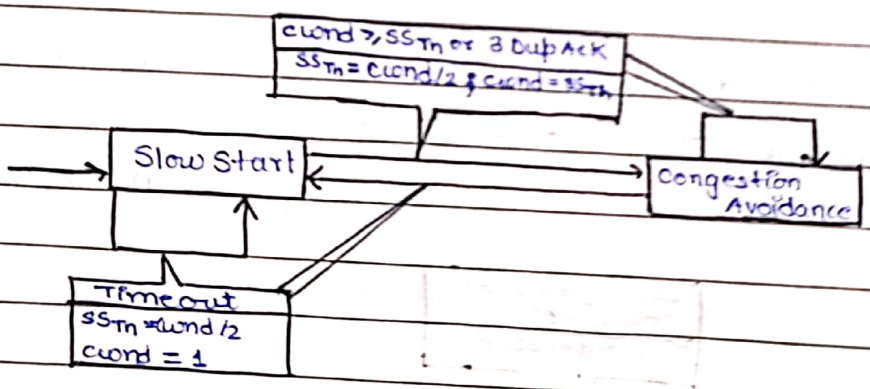


Ex. TCP starts data transfer with $SS_{Thres} = 16$ MSS. TCP begins at slow start phase with cwnd = 1. Timeout occurs after 3rd RTT, and after 7th RTT.



Taho TCP doesn't differentiate b/w Timeout & 3 dupAck.

Implementation 2



i.e. (1) Timeout : Go to Slow Start phase.

$SS_{Th} = cwnd/2$

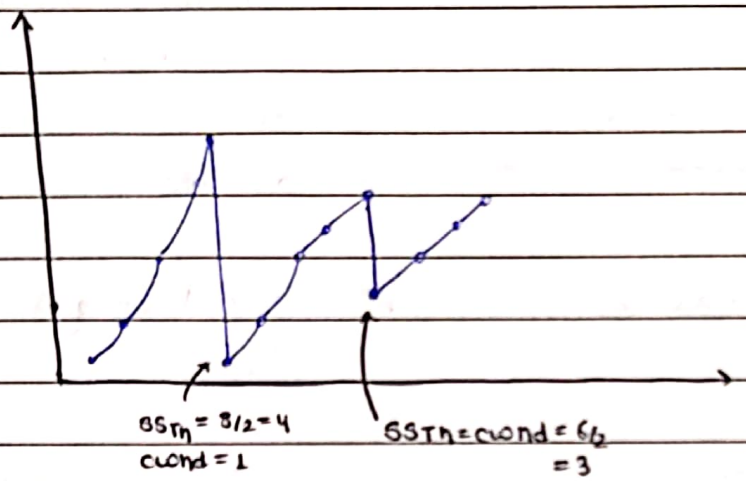
$cwnd = 1$

(2) 3 DupAck : Go to Congestion Avoidance phase.

$SS_{Th} = cwnd/2$

$cwnd = cwnd/2$

Ex. Previous example with Impl. 2.



- TCP Timers: There are four TCP timers of importance:
  - Retransmission Timeout Timer: Generally set based on RTT.
    - Measured RTT: Just measure by sending segment. (Rm)
    - Smoothed RTT: $R_s(0) = R_m(0)$
      $R_s(t) = \alpha R_m(t) + (1-\alpha) R_s(t-1)$
      {α is generally $1/8$}
  - Time Wait Timer: Started after closing conn. from client side. Generally 2MSS.
  - KeepAlive Timer: Generally 2hrs. Used by server TCP.
  - Persistent Timer: Used in case rwnd becomes zero.

Application Layer

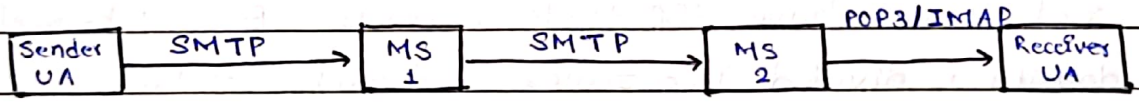
* Email System

Email system is asynchronous, i.e., both users don't need to be online at the same time.

User Agents: Application programs which can send a email.

Mail Server: Server which sends and receives mail using various protocols.

User agents might go offline, Mail Servers never goes down & that's why they act as intermediary b/w two user agents.



UAs are connected to different Mail Servers, say one may be connected to Gmail MS and other to Yahoo MS.

- **Simple Mail Transfer Protocol:** Used to send mail from sender UA to its mail server and one mail server to another. Uses TCP/25.

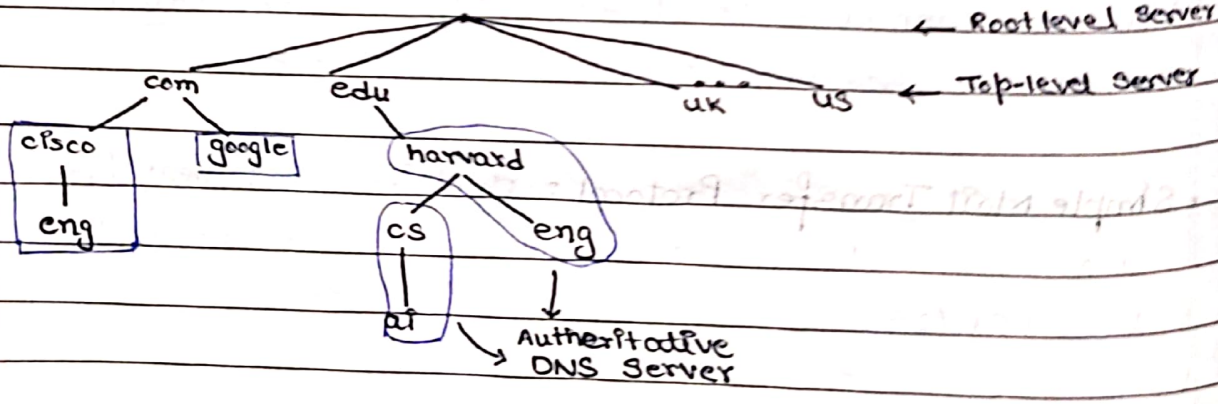
- **POP3 / IMAP:** These are pull protocols and are used to retreive mails from MS to receiver UA. POP3 moves mails from MS to UA. Uses TCP/110. POP3 is obsolete. IMAP copies mail. Uses TCP/143

* Domain Name System

Over the internet hosts are identified using IP addr. but this is hard to remember. Hosts are thus identified using "hostname", and DNS is used to map hostname to IP address.

- Services provided by DNS:
  (1) Hostname to IP translation
  (2) Host aliasing
  (3) Mail server aliasing
  (4) Load Balancing.
  etc.

- There are total 13 root level servers in world. Each root server have 250 top level domains. Each top-level domain is divided into zones, and each zone has a authoratative DNS server.



- Root DNS Server: Hold the entry of root node, i.e <top-level domain name, Top-level server IP> pairs. These are 13 in the world.

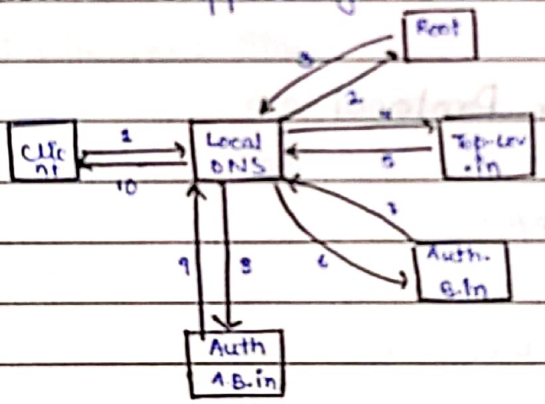
- **Top-Level Server:** Has mapping for Toplevel domains. Total 250 top-level domains.

- **Authoritative DNS Server:** It holds IP addr of all hostnames in the zone, it may also hold IP for other zones.

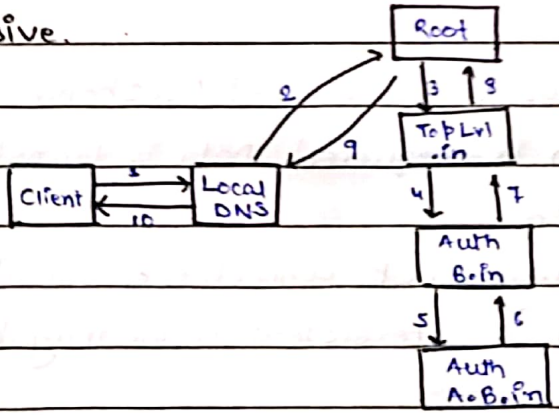
- **Local DNS Server:** The DNS server clients connect to on startup. If it doesn't know some IP for a given hostname, it might query the root level server. (DNS Resolver)

- Query might happen in one of two ways:

(1) **Iterative:** Suppose you want A.B.in



(2) **Recursive.**



* File Transfer Protocol

- Used for transfer large files.
- Uses TCP/20 & TCP/21.
- Control Channel: TCP/20. Used for exchange of control info. Only one per connection.
- Data Channel: TCP/21. Used for data transfer. Opened a new connection per file.

- In Active Mode FTP both server & client might initiate connection, in passive mode only client can.

* Hypertext Transfer Protocol.

- The server uses TCP/80.
- There are two versions:
  (1) HTTP 1.0 (Non-Persistent)
  (2) HTTP 1.1 (Persistent by default)

- Stateless protocol.
- Non-Persistence: For each request/response cycle a new TCP connection is required. Data is transferred until EOF is reached.
- Persistence: On a single TCP connection multiple request/response cycles may run. Persistent conn. may be pipelined or non-pipelined.
  Pipelined req. means that once you get base page & you need to get multiple linked files like images, js, css, multiple request can be sent in parallel to the server.

* Dynamic Host Configuration Protocol

- When computer boots up, it doesn't know its IP addr, subnet mask, etc.
  One way is to manually enter IP addr on the host.
  DHCP helps to do this in automatic way.

- DHCP provides : IP addr, default gateway addr, DNS addr, network mask, etc.

- DHCP works in four phases: DORA process.

(1) DHCP Discover: Since client is unaware of any IP, it broadcasts this message over the n/w, with destn: 255.255.255.255, and its MAC addr.

(2) DHCP Offer: One or more DHCP server unicasts the offer with IP addr & other info, along with a lease time to the host using the destn MAC of the host & src MAC addr as its own.

(3) DHCP Request: The host chooses from the offers, and broadcasts the accepted offer so other DHCP servers know that their offer was rejected.

(4) DHCP Ack: The DHCP server whose offer is accepted sends acknowledgment to the host as unicast message.