Lec-1 : OSI Model & TCP/IP Stack

- OSI Model : An open system is a set of protocols that allows any two different systems to communicate regardless of their underlying architecture.

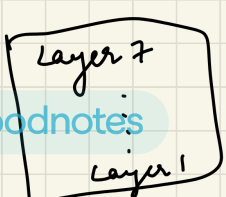
  Protocol : A protocol is a set of rules that govern data communication

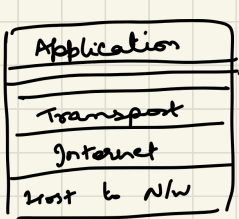
-

Layer 7	Application
Layer 6	Presentation
Layer 5	Session
Layer 4	Transport
Layer 3	Network
Layer 2	Data Link
Layer 1	Physical

- Key points of OSI model : ① Description of networking subsystem
  ② OSI Model is just a guideline
  ③ Layer is a complete logical functionality of a networking subsystem
  ④ Each layer is associated with a fn
  ⑤ fn of layers do not overlap across

- TCP/IP Reference model : multiple layers.





- OSI Reference model

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

Theoretical OSI Model

TCP/IP Stack

5	Application
4	Transport
3	Network
2	Data Link
1	Physical

Practical OSI model

- OSI Layer functions:

Application : supporting N/w applications , FTP, SMTP, HTTP, Ping, Aarogya Setu

Transport : process - process data transfer , TCP, UDP

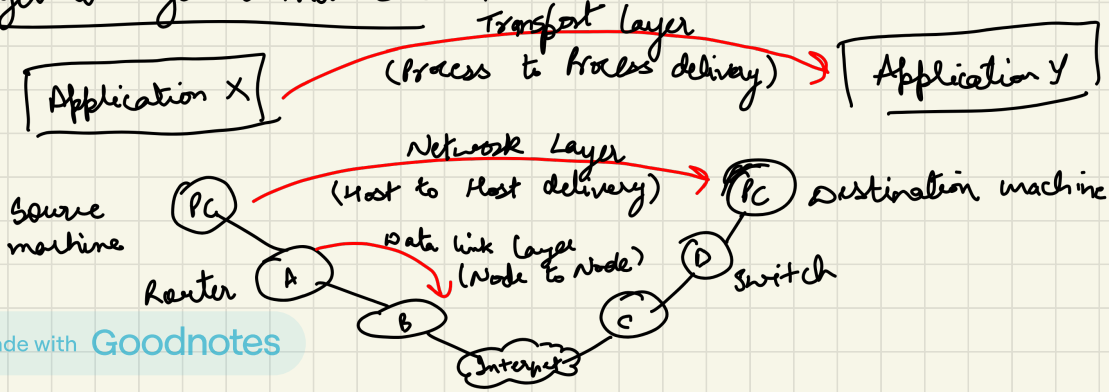
Network : routing of datagram from source to destination , IP, routing protocols

Data Link : data transfer b/w neighbouring N/w elements , Ethernet, 802.11 (wifi)

Physical : bits "on the wire"

No layer is aware of layer above it or below it
layers are isolated, with non-overlapping responsibilities

- Layer to Layer communication:



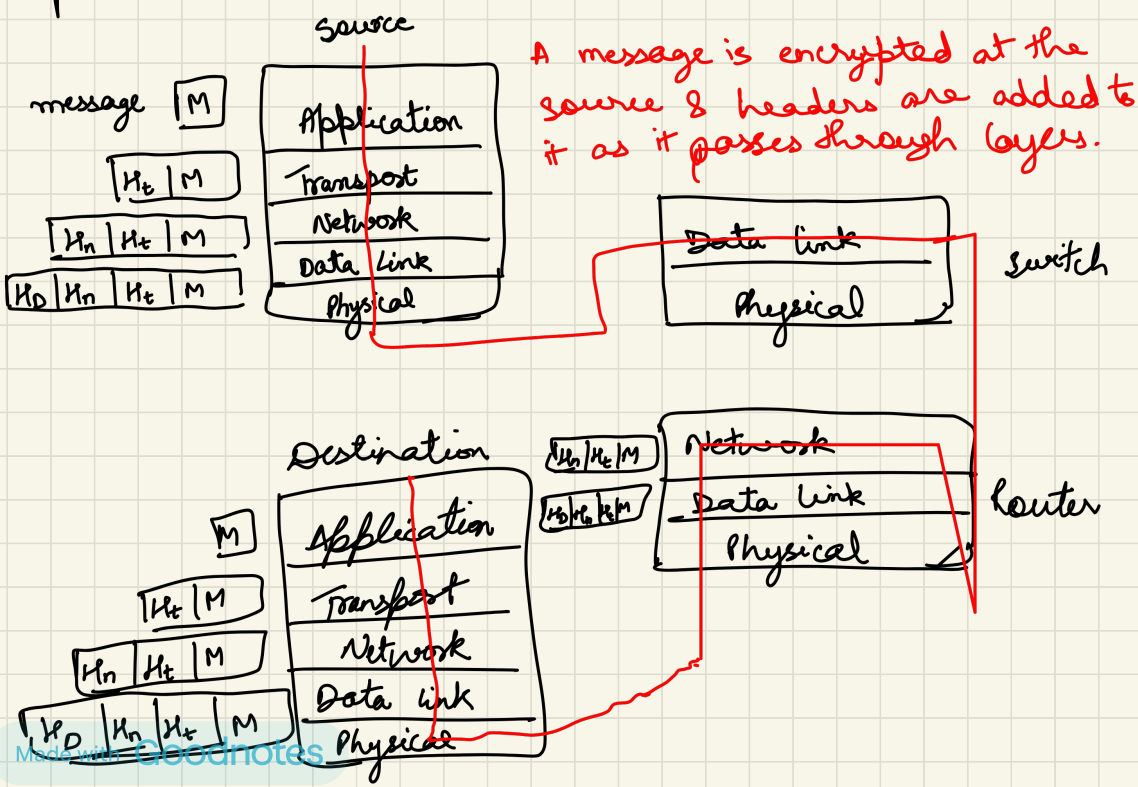
- Layer functions :

Physical : To transmit bits ; to provide mech. & electrical specification
Data link : To organize bits to frames ; to provide node-to-node delivery
Network : To move packets from source to destination, to provide networking
Transport : To provide end-to-end msg delivery & error recovery
Application : To allow access to n/w resources
To establish, manage, & terminate sessions
To translate, encrypt & compress data

* TCP/IP Stack real world analogy

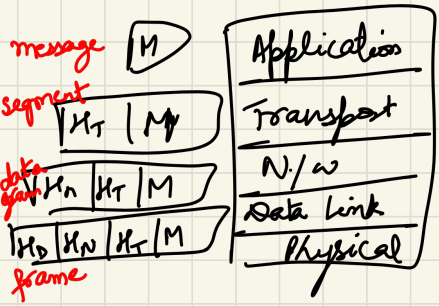
- Data Encapsulation & Decapsulation :

• Encapsulation

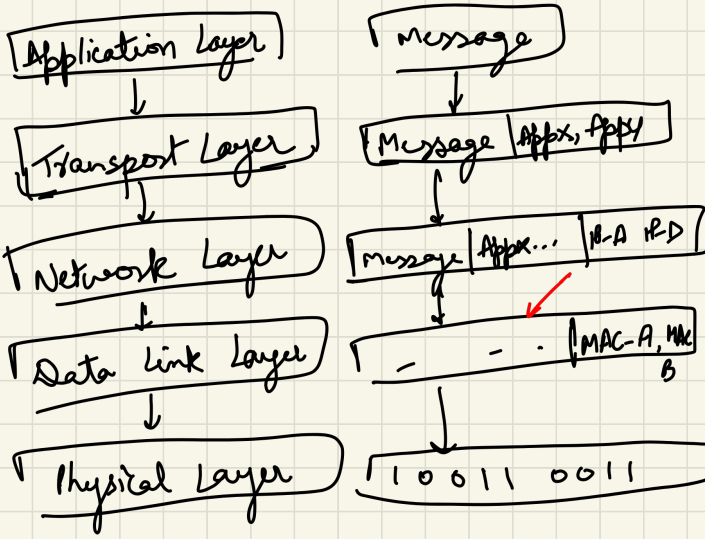


- On sending machine

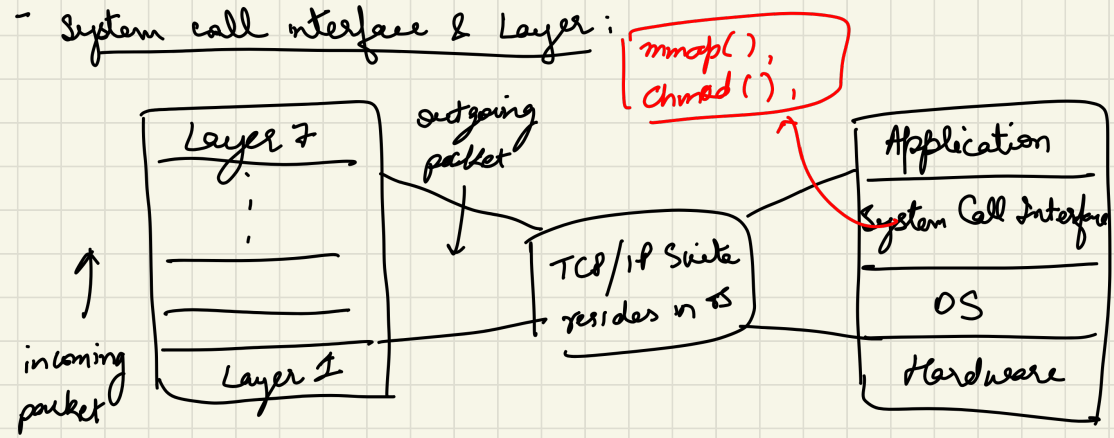
source



On receiving machine

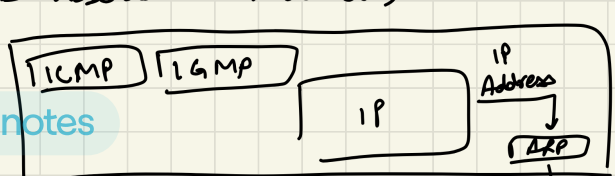


- System call interface & Layer:

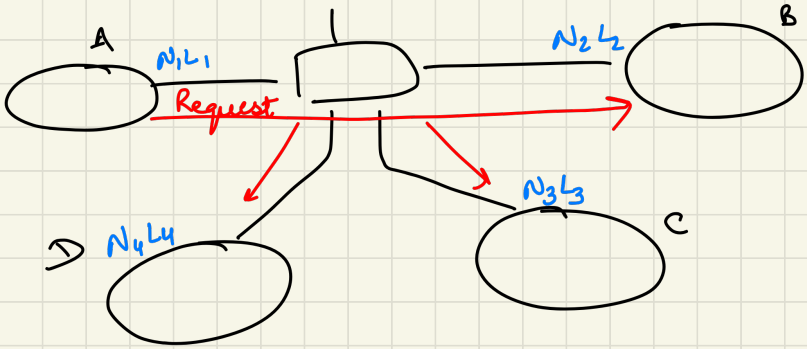


LEC 2 : Transport Layer & Protocols

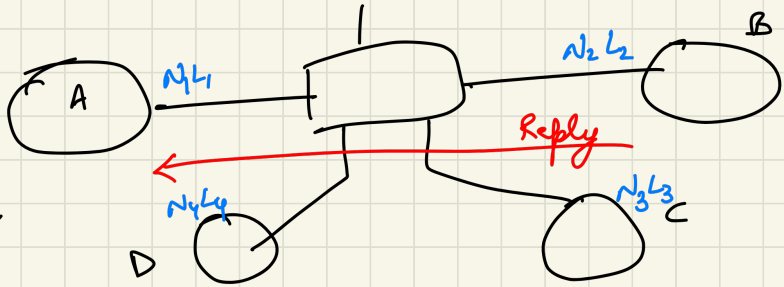
- ARP (Address Resolution Protocol)



- ARP operation



Request:
Looking for link-layer address of a node with IP address N_2



Reply:
I am the node & my link-layer is L_2

- ARP request is broadcast.
  ARP reply is unicast.

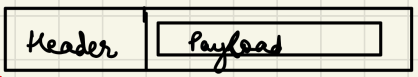
- **ARP Packet format:**

0	8	16	32
Hardware Type		Protocol Type	
Hardware Length	Protocol length	Operation Request: 1, Reply 2	
Source hardware address			
Source protocol address			
Destination hardware address			
Destination protocol address			

- IPv4 Protocol : Packets used by IP are called datagram,

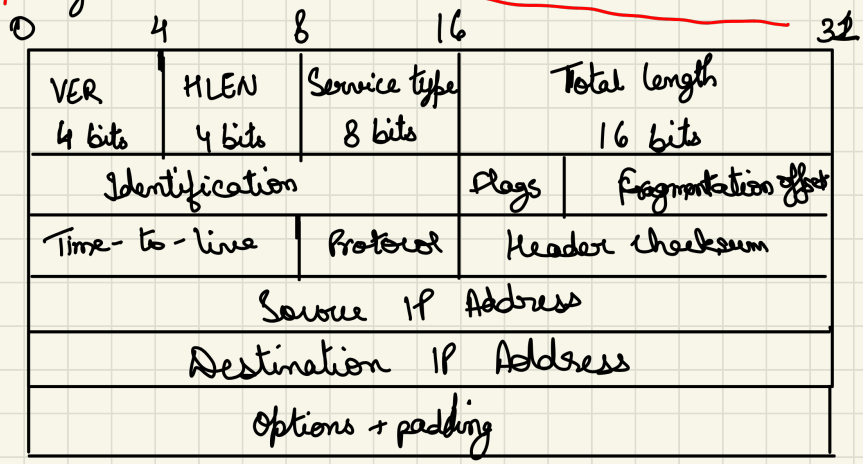
A datagram is a variable-length packet : header & payload

Header is 20 to 60 bytes in length & contain info. about routing & delivery.



IP datagram

20 - 60 bytes



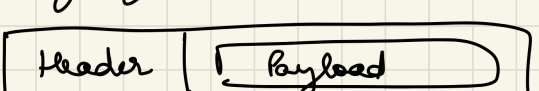
VER : version number

HLEN : header length

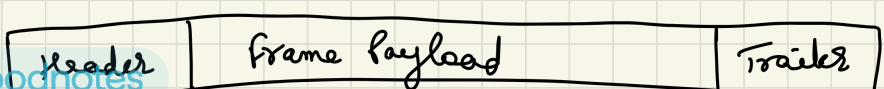
byte : 8 bits

- Fragmentation offset :

A datagram can travel through diff. networks.
Each router decapsulates IP datagram from the frame it receives, processes it & then encapsulates it in another frame.
The format & size of frame received (sent) depends on the protocol used by the physical network through which frame has just travelled (is going to travel).



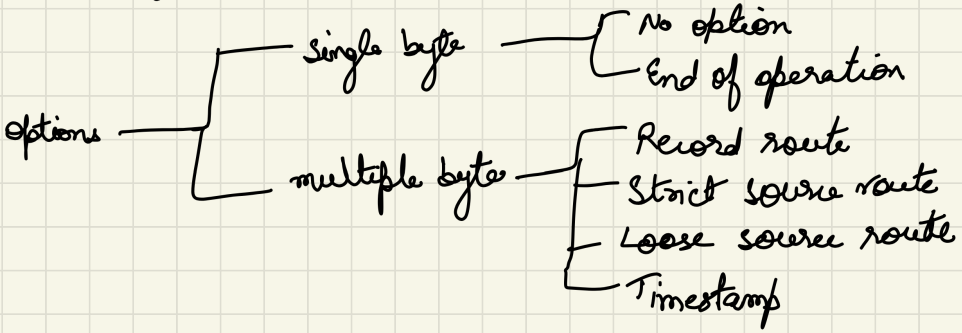
IP datagram



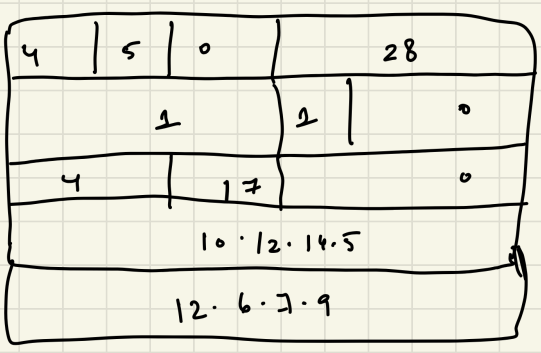
frame

- Header of IPv4 is made of two parts : fixed & variable.
(20 bytes) (of max. 40 bytes)

- Taxonomy Of options in IPv4 : ✱ ✱ ✱ ✱ ✱ ✱ ✱



- Checksum Calculation in IPv4 :



Converting decimal to hexadecimal :

0 — 0
1 — 1
⋮
9 — 9
10 — A
11 — B
12 — C
13 — D
14 — E
15 — F

10.12.14.5 = 0A.0C.0E.05

(17 = 11 (when base is 16))

{ formula for checksum = One's complement (Total length + Source IP + destination IP)

4, 5, 0 → 4 5 00
28 → 00 1C
1 → 00 01
4 & 17 → 04 11
0 → 00 00

10.12 → 0A 0C
14.5 → 0E 05
12.6 → 0C 06
7.9 → 07 09
+
Sum

Sum = 7 4 4 E

checksum = 8 B B 1

```
  4500
  001C
  0001
  0000
  0411
  0000
  0A0C
  0E05
  0C06
  0709
  ----
  744E
```

ones	tens	hundreds	thousands
12	2	5	3
1	1	4	4
1	1	10	7
12	4	14	7
5		12	
6		7	
9		52	
46	14		17

carry: 1, 19, 25

$16^2 3 + 4$

134

$32 + 14$

$16^1 2$ E

2E

⇒ Sum = 744E

744E =

$\overline{E} = \overline{14} = 15 - 14 = 1$

$\overline{4} = 15 - 4 = 11 = B$

$\overline{7} = 15 - 7 = 8$

⇒ checksum = $\overline{744E}$ = 8B B1

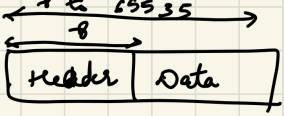
- Transport Layer protocols :

Responsibilities : process-to-process comm. using port no's. Port no's provide end to end addresses at the transport layer & allow multiplexing & demultiplexing at this layer, just as IP address do at the N/w layer.

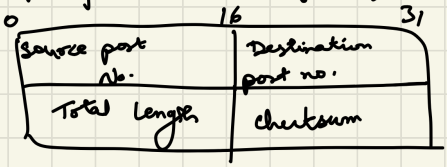
Types : UDP
TCP
SCTP

- UDP : (User Datagram Protocol)

i) UDP is a connectionless, unreliable transport protocol.
ii) UDP is a simple protocol using a min^m of overhead.
iii) UDP packets are called user datagrams, have a fixed sized header of 8 bytes made of 4 fields. (each of 2 bytes)



UDP user datagram



Header format

Total length needs to be less than 65535 as the UDP user datagram is stored in an IP datagram with total length of 65535 bytes.

TCP supports unicasting only.

TCP must discard duplicate data

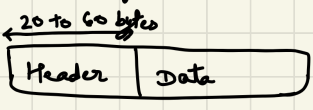
- TCP : (Transmission Control Protocol)

overhead !

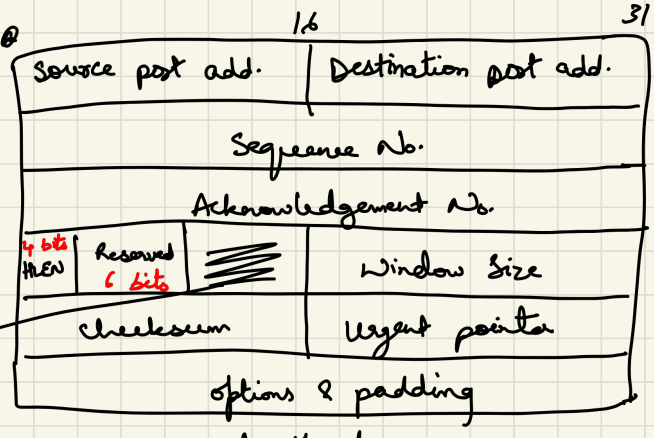
i) TCP is connection oriented, reliable protocol.
ii) TCP explicitly defines connection establishment, data transfer & teardown phases to provide a connection oriented service.
iii) TCP uses a combination of GBN (Go Back N) and SR (selective repeat) for providing reliable connection.

*** Sending & receiving buffers ***

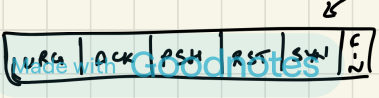
- TCP segment :



a) Segment



b) Header

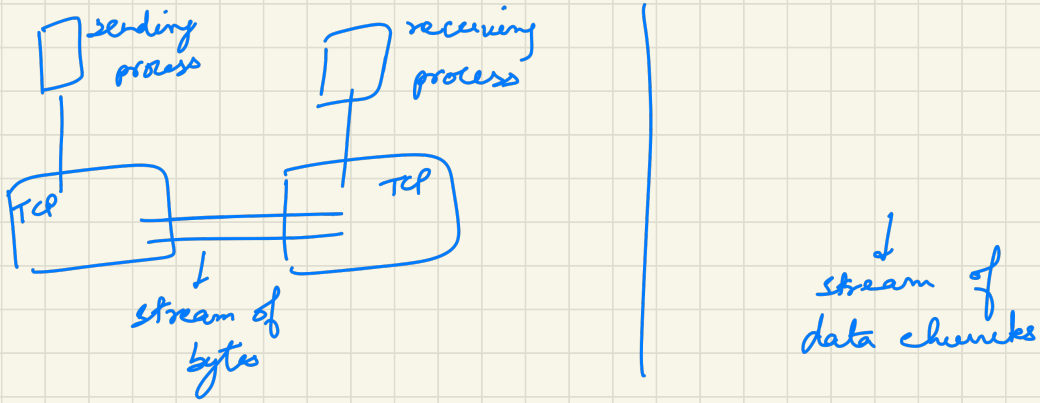


URG : urgent pointer is valid.
ACK : Acknowledgement is valid.
PSH : Request for push.
SYN : Synchronize sequence no's.
FIN : Terminate the connection.
RST : Reset the connection.

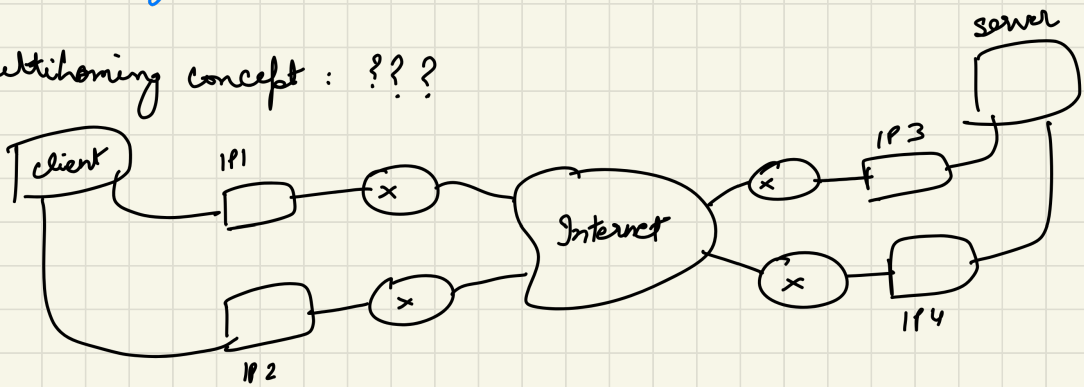
UAPRSF

- SCTP Protocol : (Stream Control Transmission Protocol)

i) New protocol designed to combine some features of UDP & TCP to create multimedia communication.



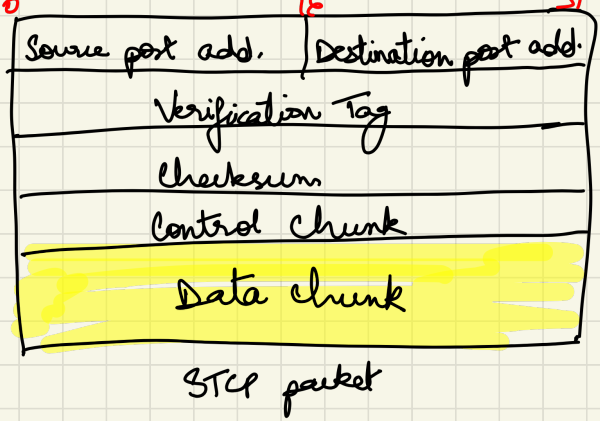
- Multihoming concept : ???



- SCTP features : i) Transmission sequence No. (TSN)
ii) Stream Identifier (SI)
iii) Stream Sequence No (SSN)

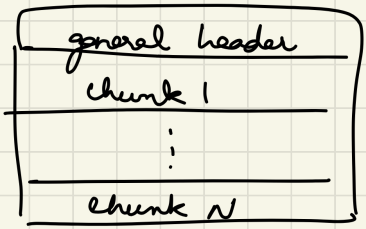
- Comparison b/w TCP and SCTP packet :

TCP segment



SCTP packet

general format of SCTP packet :



↓

two types of chunks

Control chunks → controls & maintains the association

Data chunks → carries user data

lecture 14 : (Domain Name Server - DNS)

- DNS helps in mapping website name to IP address & IP address to website name.

- Website domain name ⟶ IP address

This is functionality of DNS.

www.google.com is website hosted on server whose IP is 8.8.8.8

- IP address ⟶ website domain name

this is reverse functionality of DNS.

(DNS is like phonebook for internet)

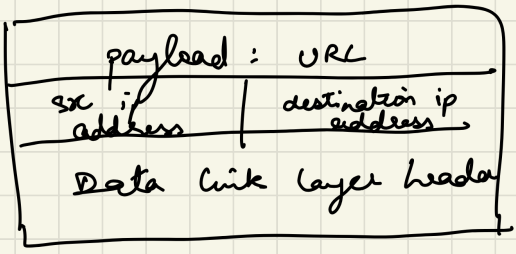
{ computers don't understand URLs. They use IP for routing. }
{ This is what DNS enables.

DNS is a distributed Database. Why is it not centralized.

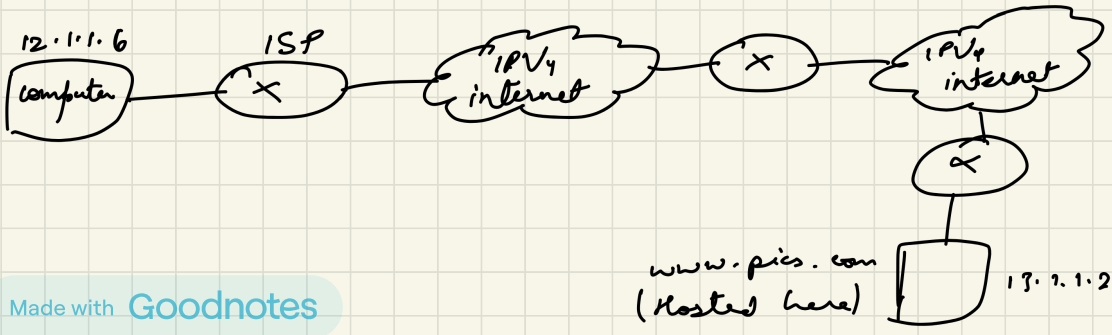
- single pt. of failure
- traffic volume
- maintainence

- DNS Architecture (a decentralised system):

Packet generated

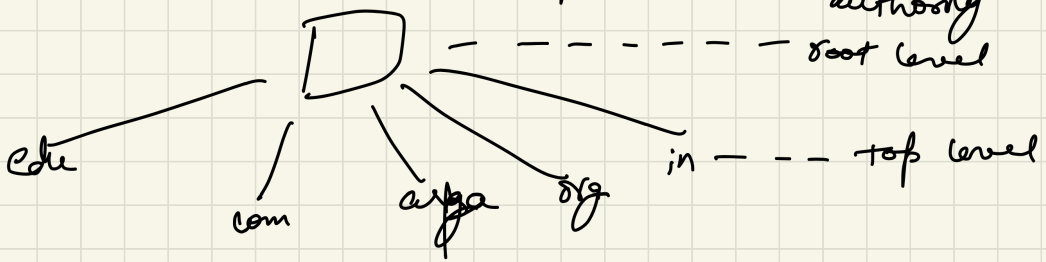


when you type www.mypic.com/index/flowers.jpg



- DNS is an application layer protocol.

- DNS system architecture — Root level } managed by
Top level } central authority



- DNS is a distributed system — Load balancing
fault tolerance
Redundancy
(multiple IPs for one hostname)

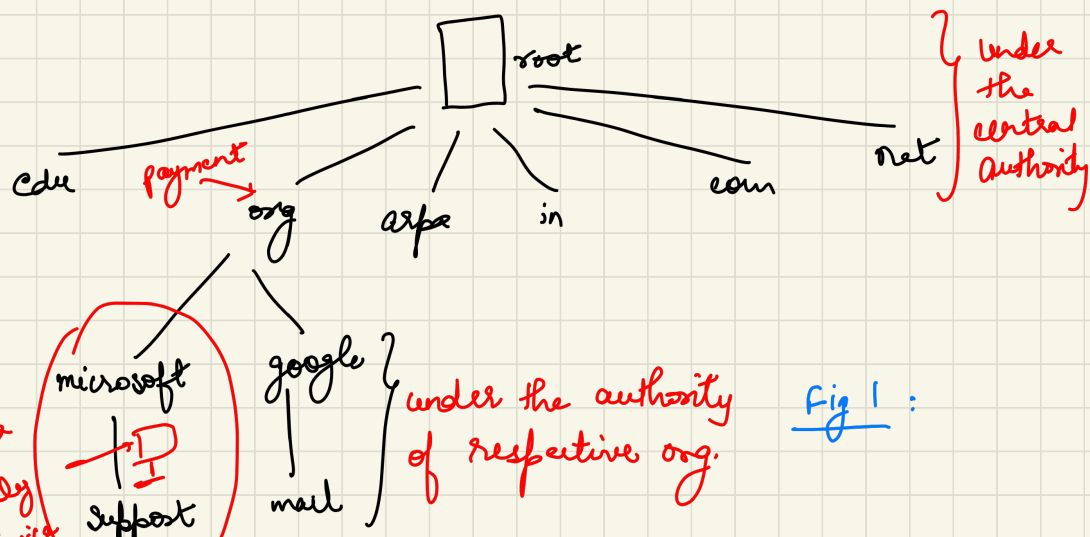
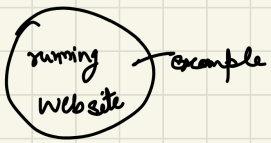


Fig 1 :

- servers higher in hierarchy know how to reach directly lower ones in the heirarchy.
- DNS at leaves of trees contains records of IP mapping them to hostname.

- Hosting your own website



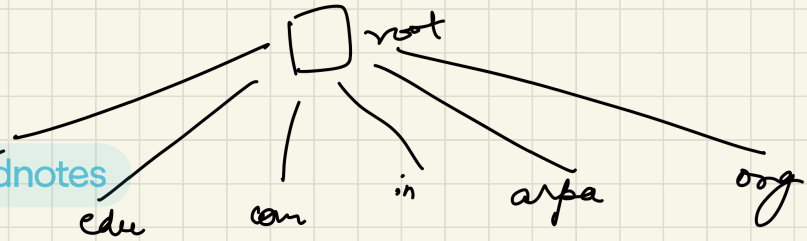
www.example.edu

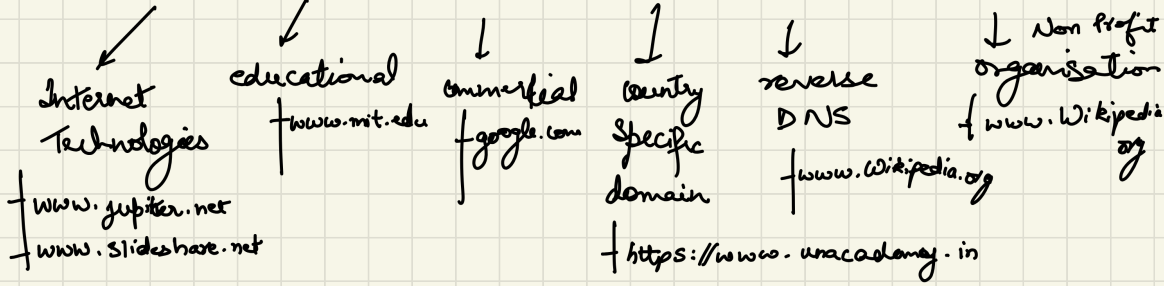
- Website domain name: constructed of individual words called literals. These literals are actually names of DNS servers in the path of DNS tree from root towards leaf of DNS tree.

- In fig 1: www.customer.support.microsoft.com.

. — represents root server
.com — TLDs (Top level Domain)
.microsoft — Microsoft's DNS server
.support — " sub " "
.customer — Microsoft's server which is hosting actual website.

- Top level DNS servers classification:



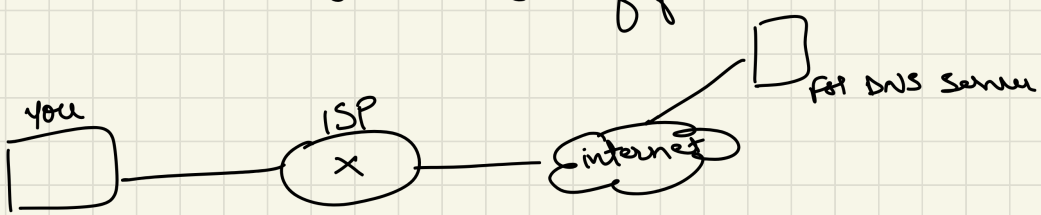


- First Hop DNS server :

fully qualified domain name = Host Name + Domain Name

- our computer is assigned the address of 1st DNS server to obtain IP address of domain name of website we are trying to access.

- If 1st DNS has ip address of host server which host the website it returns the address of our computer otherwise it queries the root DNS server on our behalf for websites IP Address



- 1st DNS server caches the recently accessed website's ip address in its local DNS cache so that in future our computer can obtain the IP address of recently accessed website, quickly

our computer also has its own local DNS cache.

- Recursive DNS query : FQDN → IP Address

i) you type out "www.customer.support.microsoft.com".
DNS query : what is IP address of FQDN " - - - ".

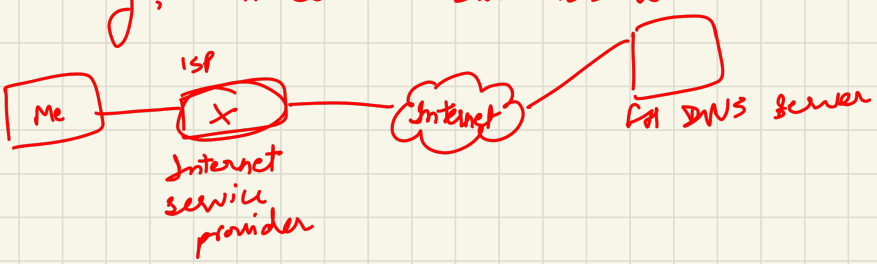
ii) your local server lookups in the local cache to find the ip address of domain name "- - - ". If it did not find, it sends a recursive DNS query to DNS resolver.

First Hop DNS Server :

FH DNS server → statically / manually / DHCP
our comp. ask for FH DNS first to access IP address of a domain name

if (FH DNS has ip address of host server) → returns to comp
else (it queries the root DNS server)

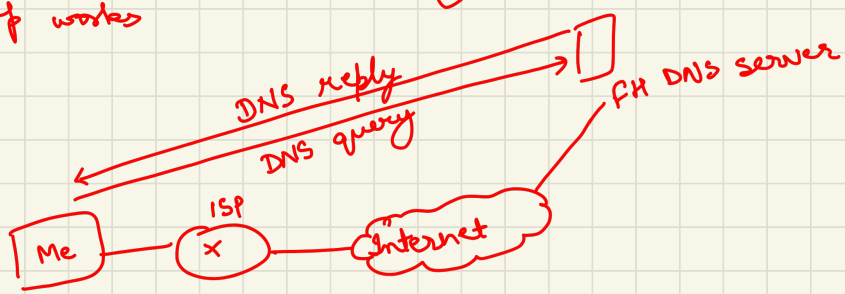
That is why, FH server = DNS resolver



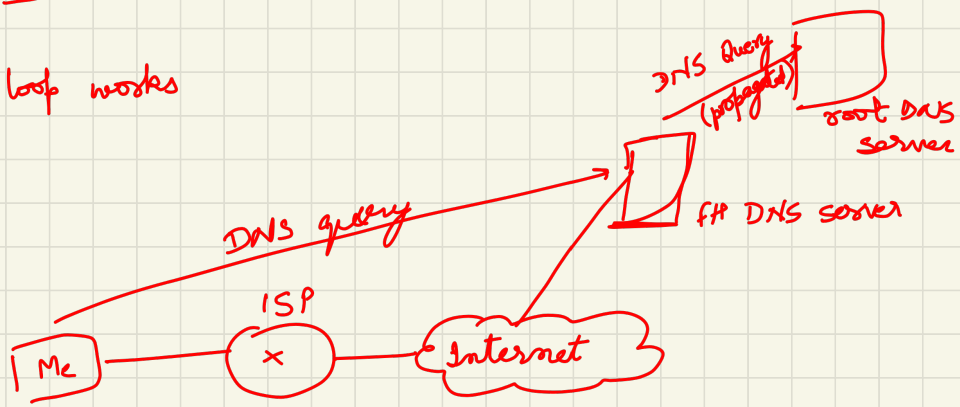
FH DNS server caches recently accessed website's ip address in its local DNS cache so that in future our computer can obtain the IP address of recently accessed website quickly from FH DNS only

our computer also has its own local DNS cache.

when 'if' loop works

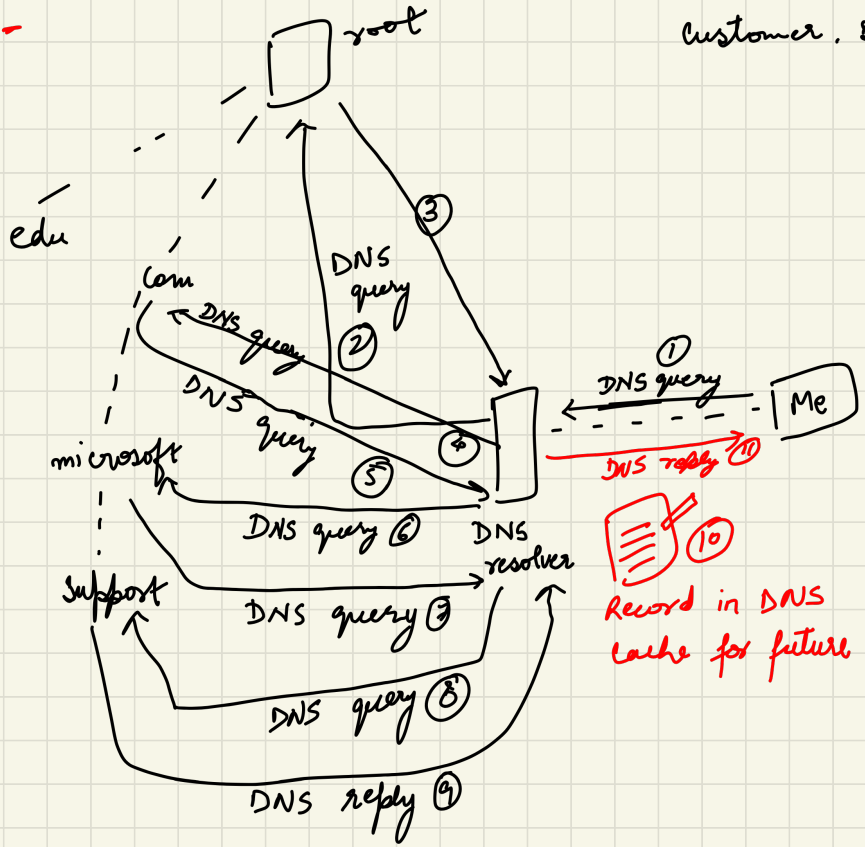


when 'else' loop works



first our computer looks up its local DNS cache. if ip not found our computer sends the DNS query to FH DNS server. if ip not found FH DNS server propagate DNS query to root DNS server.

customer.support.microsoft.com

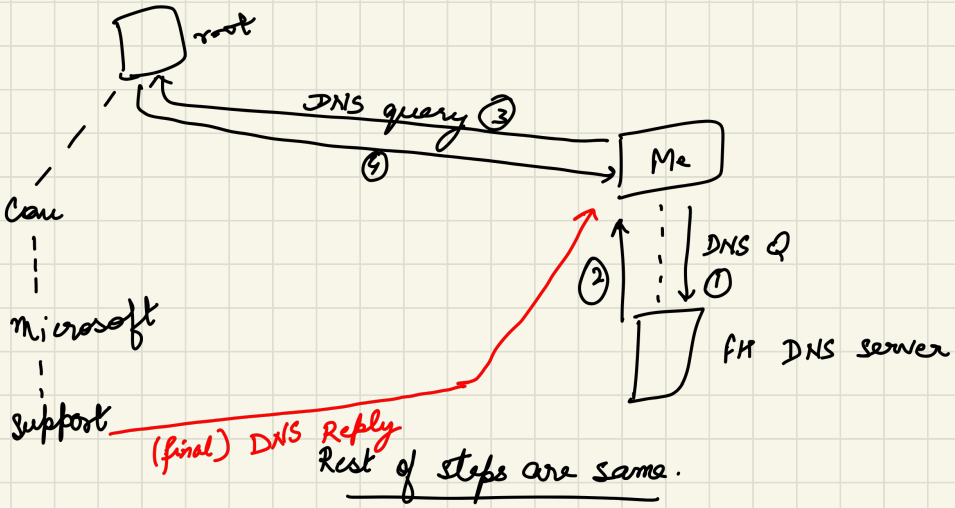


3. Root DNS servers are iterative servers i.e. they always reply back with referral answer to DNS query. They do not query any other DNS servers to get definite answer. Root DNS server checks that DNS query is for .com domain. It DNS with ip address of .com TLD server.

- root & TLD servers reply with referral answer to DNS resolver

- Iterative DNS query :

Rather than sending queries recursively to get a definite response, a new query is sent to the TLD server referred to in the previous response.



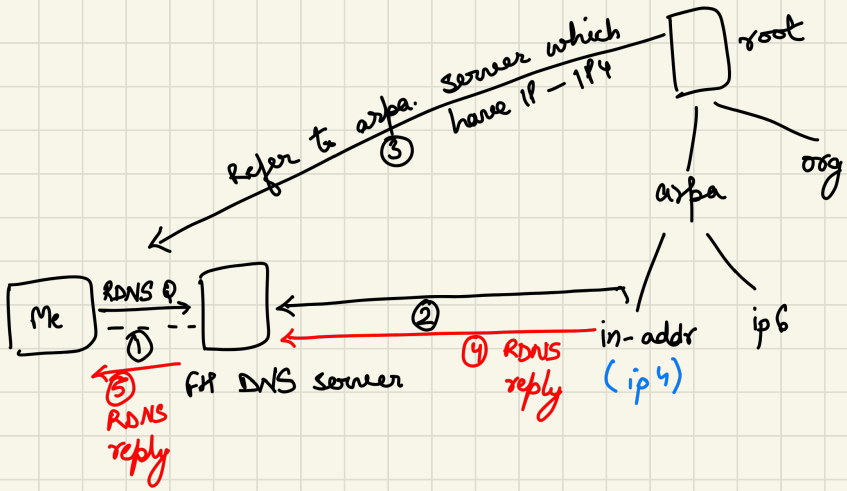
Rest of steps are same.

Recursive DNS query	Iterative DNS query
i) DNS resolver does all the work.	DNS client has to do all the work.
ii) DNS RESOLVER sends query to subsequent DNS server based on referral ans.	DNS CLIENT — ..
iii) DNS client recieves a definite ans: either DNS resolution or error msg.	DNS client receives a referral ans or error msg.
iv) Burden is on resolver.	burden is on DNS client.

- Reverse DNS Query :

if we have to look for domain name of IP address 212.132.1.2

1) Reverse the IP address : 2.1.132.212
2) append labels : in-addr.arpa.

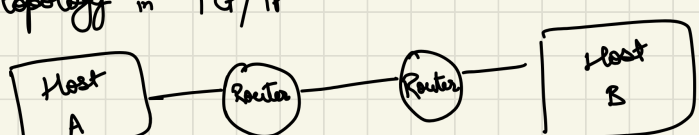


Reverse DNS is opposite of normal DNS lookup.
RDNS lookup can be both iterative or recursive.
arpa is TLD server responsible for handling all RDNS queries.
in-addr for ipV4 is 2nd level server to provide definite ans. to DNS Q.
ip6 DNS is ipV6 counter part of in-addr DNS.
RDNS is used in troubleshooting / traceroute commands etc.

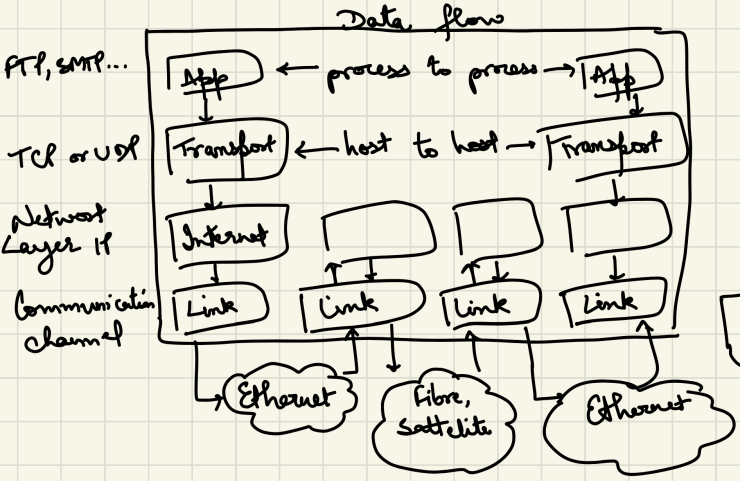
Lecture-4 (Intro to Socket Programming)

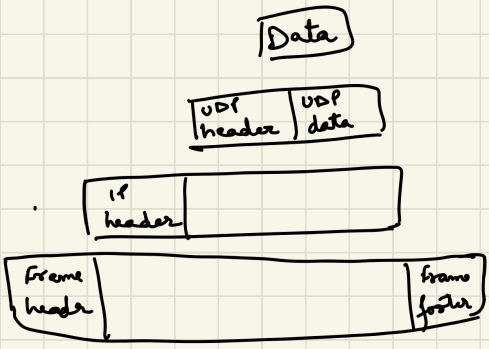
- TCP/IP provides end to end connectivity specifying how data should be formatted, addressed, transmitted, routed.
- fig 3 (diagram)

Network topology in TCP/IP



Data flow in TCP/IP





- TCP VS UDP :

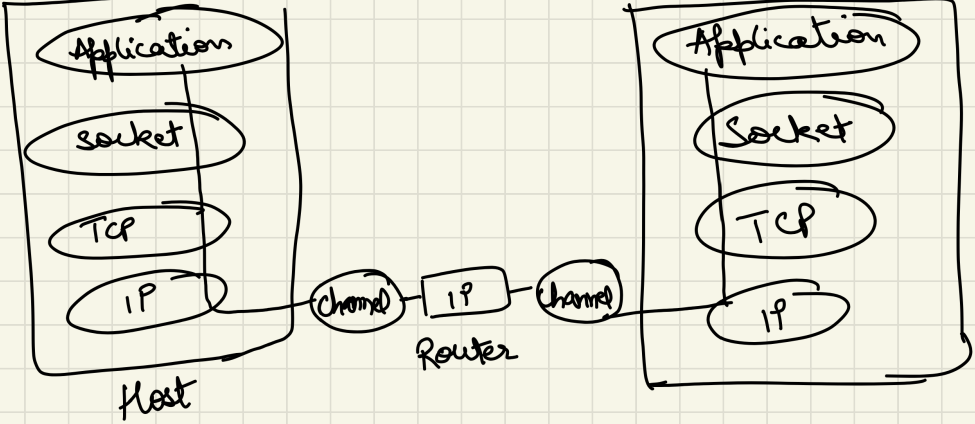
i) Both use port no's.

ii) UDP — user datagram protocol
connectionless
out of order transmission, duplicates possible

iii) TCP — transmission control protocol
Connection oriented, byte stream channel
bidirectional

- TCP is used for large data capacity & UDP is used for single query - reply actions.

- Berkley Sockets : known as sockets, an abstraction through which application may send & receive data

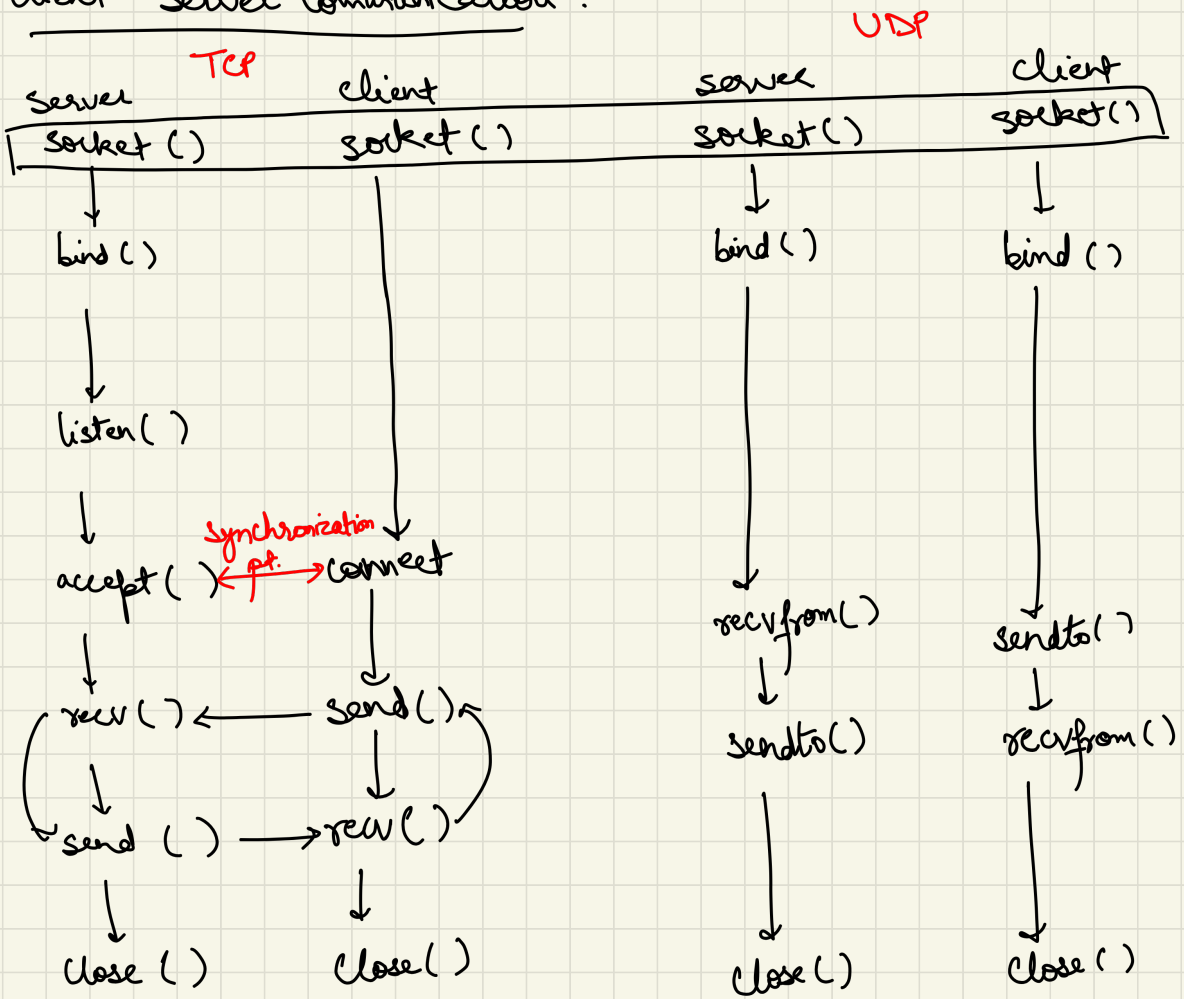


- sockets are uniquely identified by:
  An internet address
  An end to end protocol (eg. UDP, TCP)
  A port number
- sockets extend conventional UNIX I/O facilities:
  i) file descriptors for N/w communication
  ii) Extend the read & write system calls.

Client	Server
Active socket	passive socket
initiates the comm., must know the address & port of the server	passively waits for and responds to client

- socket : create a new communication endpoint
  bind : attatch a local address to a socket
  listen : announce willingness to accept connections
  accept : block caller until a connection request arrives
  connect : actively attempt to establish a connection

Client - Server Communication :



- receive : receive some data over connection
  send : send some data over connection
  close : release the connection

- socket creation in C : Socket()

① int sockid = socket (family, type, protocol)

sockid : socket descriptor

family : integer, communication domain, eg

PF_INET, IPv4 protocol,

protocol : usually set 0 (IPPROTO_TCP, IPPROTO_UDP)

type : communication type

SOCK_STREAM, SOCK_DGRAM

```
struct sockaddr_in {
unsigned short sin_family ;
unsigned short sin_port ;
struct in_addr sin_addr ;
char sin_zero[8];
}

struct in_addr {
unsigned long s_addr;
}
```

(Address port = 16 bits)

(Internet Address = 32 bits)

```
struct sockaddr {
unsigned short sa_family;
char sa_data[14];
}
```

(ii) int status = bind (sockid , & addrport , size);

addrport = generally set to INADDR_ANY i.e. choose any incoming interface

size : (in bytes) size of the addrport structure

Bind can be skipped for both types of sockets.

Datagram socket :

No need for binding while sending

The OS finds a port each time the socket sends a packet

If receiving, need to bind

Stream socket :

No need to bind while sending, ∵ destination determined during connection setup

(iii) `int status = listen (sockid, queueLimit);`

queueLimit : no. of active participants that can wait for connection

listen ( ) is non-blocking

- The listening socket (sockid) → not used for sending & receiving
  → used by server only as way to get new sockets

(iv) `int status = connect (sockid, &foreignAddr, addrlen);`

foreignAddr : address of passive participant

connect ( ) is blocking.

(v) `int s = accept (sockid, &clientAddr, &addrlen)`

new socket (used for data-transfer)

must be set appropriately before call, adjusted upon return

is blocking : waits for connection before returning
dequeues the next connection on the queue for socket

(vi) int count = send (sockid, msg, msglen, flags);

Exchanging data with TCP

special options

(vii) int count = recv (sockid, recvBuf, buflen, flags);

calls in (vi) & (vii) are blocking, returns only after data is sent/received.

(viii) int count = sendto (sockid, msg, msglen, flags, &foreignAddr, addrlen)

(ix) int count = recvfrom (sockid, recvbuf, buflen, flags, &clientAddr, addrlen)

↑ exchanging data with UDP

calls are blocking here too

(x) int status = close (sockid)
frees up the port used by socket

Lec 5 (Byte ordering & Byte manipulation fn):

- a 16 bit integer made up of 2 bytes can be stored in two ways:

(i) Little endian : A A+1

Lower byte is stored on starting address A and higher order byte is stored on next address A+1

(ii) Big endian : higher order byte is stored on starting adder A and lower order byte is stored on next address A+1.

Network Byte order : To allow machine with diff. byte order conventions, the internet protocols specify a canonical byte order convention for data transmitted over network.

Make sure data in sin_addr & sin_port in sockaddr_in is in Network byte order.

- Byte ordering functions : htons , htonl , ntohs , ntohl
  16 bit (2 byte) 32 bit (4 byte) 16 bit 32 bit

htons = host to network short
htonl = host to network long

on little endian machines, the code will change the values around to network byte order.

on big endian machines, no code is inserted since none is needed, the functions are defined as NULL.

Program to determine Host order:

```
#include<stdio.h>

int main(){
union{ushort s; char c[sizeof(s)];}un;

un.s = 0x102;
if(sizeof(short)==2){
if(un.c[0]==1 && un.c[1]=2){printf("big endian");}
elseif(un.c[0]==2 && un.c[1]==1){printf("little endian");}
else{printf("unknown");
}
else{printf("size: ",sizeof(short));}
exit(0);
}
```

s = 0x102, therefore

if c = {0x02, 0x01} → little endian

c = {0x01, 0x02} → Big endian

Byte Manipulation functions:

- inet_aton()
- inet_ntoa()
- inet_addr()

These are the functions to convert Internet addresses b/w ASCII strings and network byte ordered binary values.

(i) int inet_aton (const char* strptr, struct in_addr *addrptr)

f" converts specified string in Internet standard dot notation to a network address and stores the address in the structure provided.

The converted address is stored in Network Byte order.

It returns 1 if string and 0 on error.

(ii) in_addr_t inet_addr (const char* strptr)

f" converts string to integer value

(iii) char * inet_ntoa (struct in_addr inaddr)

f" converts specified internet host address to string in internet standard dot notation

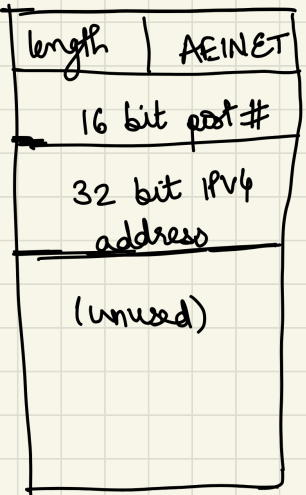
IPv6 Address Structure :

- IPv6 family → AF_INET6
  IPv4 family → AF_INET
- The SIN6_LEN constant defines the length member for socket address structures.

- sin6_flowinfo member is divided into two fields:
  i) low order 20 bits are flow label
  ii) high order 12 bits are reserved

IPv4

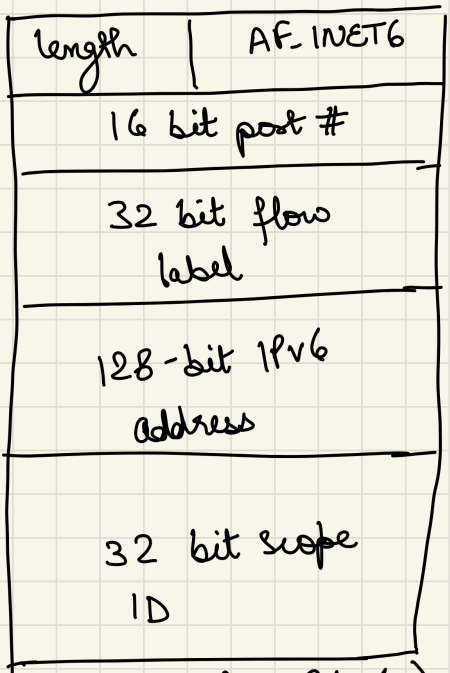
sockaddr_in{}



fixed-length (16 bytes)

IPv6

sockaddr_in6{}



fixed length (28 bytes)

```
struct sockaddr_in6 server_add;
int sockid = socket (AF_INET6, SOCK_STREAM, IPPROTO_TCP);
server_add.sin6_port = htons(SERVER_PORT);
server_add.sin6_family = AF_INET6;
server_add.sin6_addr = in6addr_any;
bind (sockid, (struct sockaddr_in*)& server_add, sizeof(server_add));
```

inet_pton (AF_INET6, "::1", &server_addr.sin6_addr)
the above might be used to assign loopback address only

Lecture 6 (socket option):

int getsockopt (int sockid, int level, int optname, void *optval, socklen_t *optlen);

int setsockopt (int sockid, int level, int optname, const void* optval, socklen_t optlen);

sockid = open socket descriptor

level = code in system to interpret the option (generic, ipv4, ipv6, TCP) SOL_SOCKET

optval = pointer to variable from which the new value of option is fetched by setsockopt or into which current value of option is stored by getsockopt.

optlen = the size of option variable

- Two type of functions are used:

(i) binary: options that enable or disable a certain feature

(ii) options that fetch & return specific values that we can either set or examine.

Why do we need socket options?

In TCP, connected socket is not returned to server by accept until three way handshake is completed by TCP layer.

We must set that option for listening socket to ensure that one of these socket options is set for connected socket when three way handshake completes.

1. Generic Socket option :

i) SO_BROADCAST : Enable or disable the ability of process to send broadcast messages.

only for UDP, not for TCP/SCTP as broadcast can't be done on connection based transport protocol.

ii) SO_DEBUG : When enabled for TCP socket, Kernel keeps track of detailed info. about all packets sent or received by TCP. (only for TCP)

iii) SO_DONTROUTE : outgoing packets are to bypass normal routing mechanisms of underlying protocol

This option is used by routing daemons (routed & gated) to bypass the routing table and force a packet to be sent out a particular interface.

iv) SO_ERROR : when error occurs on a socket, the protocol module in Berkley derived kernel sets a variable so_error for that socket.

Process can obtain value of so_error by fetching SO_ERROR socket option

v) SO_KEEPALIVE : when Keep alive option is set for TCP and no data has been exchanged across the socket in either direction for 2 hours. TCP automatically sends a keep alive probe to peer.

Peer response :

MSS = max^m Segment size
↓
max^m size of data that can be sent in a single segment

① ACK : everything ok

② RST : Tell the local tcp that the connection to peer host has crashed and rebooted. Socket's pending error is set to ECONNRESET and socket is closed.

③ No response: Berkeley derived TCPs send 8 additional probes, 75 seconds apart, trying to elicit a response. TCP will give up if there is no response within 11 min & 15 sec after sending first probe. The socket's pending error is set to ETIMEDOUT.

vi) SO_LINGER :

```
struct linger {
    int l_onoff;
    int l_linger;
};
```

- l_onoff = 0, option is turned off.
- l_onoff = nonzero and l_linger = 0, TCP aborts the connection when it is closed and discards any remaining data in the send buffer.
- l_onoff = nonzero and l_linger ≠ 0, process waits until remaining data is sent or the linger time expires.

If the socket has been set nonblocking it will not wait for close to complete, even if linger time is nonzero.

SO_LINGER DIAGRAM

- A way to know that peer application has read the data ?

- SO_RCVBUF , SO_SNDBUF
- SO_RCVLOWAT, SO_SNDLOWAT
- SO_RCVTIMEO, SO_SNDTIMEO
- SO_REUSEADDR, SO_REUSEPORT
- SO_TYPE
- SO_USELOOPBACK

2. IPv4 socket option :

level = IPPROTO_IP

(i) IP_HDRINCL : if this option is set for raw IP socket, we must build our IP header for all datagram that we send to raw socket.

(normally Kernel does so)

(ii) IP_OPTIONS : Setting this option allows us to set IP options in the IPv4 header

requires the knowledge of the format of IP options in IP header.

(iii) IP_RECVDSTADDR : This socket option cause the destination IP address of received UDP datagram to be return as ancillary data by recvmsg.

(iv) IP_RECVIF : returns index as ancillary data by recvmsg (index of interface on which UDP datagram is received)

(v) IP_TOS : set type of service (TOS) field in IP header for a TCP, UDP or SCTP socket.

If we call getsockopt for this option, the current value that would be placed in TOS (type of service) field in the IP header is returned.

(vi) IP_TTL : we can set and fetch default TTL that the system will use for unicast packets sent on a given socket.

BSD uses default of 64 for both TCP and UDP sockets.

(3) TCP socket options :

Level = IPPROTO_TCP

(i) TCP_KEEPALIVE : specifies the idle time in second for connection before TCP starts sending the keepalive probe.

Default time is 2 hours. This option is effective only when so_keepalive is enabled.

(ii) TCP_MAXRT : specifies amount of time before a connection is broken once TCP starts retransmitting data.

- 0 : use default
- 1 : retransmit forever
- positive value : rounded up to next transmission time

(iii) TCP_MAXSEG : option that allows us to fetch or set MSS for a TCP connection.

The value returned is the max^m amount of data that our TCP will send to the other end.

It is the MSS announced by other end with its SYN unless our TCP chooses to use a smaller value than the peer's announced MSS.

(iv) TCP_NODELAY : This option disables TCP's Nagle algorithm.

Nagle Algorithm : If the given connection has outstanding data, then no small packets will be sent on connection in response to a user write operation until existing data is acknowledged.

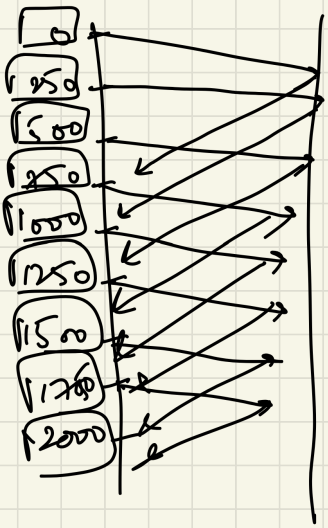
small packet is any packet smaller than MSS.

- Default enabled
- Reduces no. of small packet on WAN

```
if (avail_data & windowsize > MSS) { send }
else {
        if (unconfirmed data) { queue }
        else { send }
    }
```

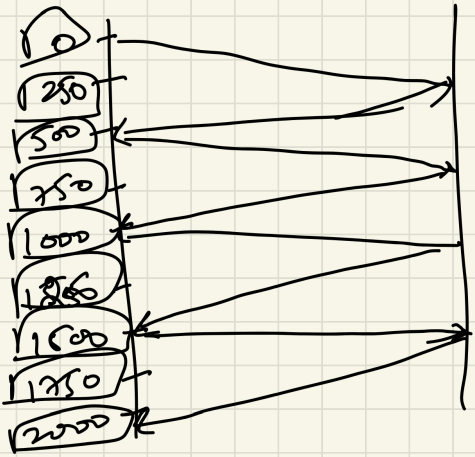
Nagle algo disabled

you send a packet & receive acknowledgement each time but if the MSS = 1750 then you won't be able to send all the packets.



Nagle algo enabled

But here you check the conditions mentioned above in the code & if the window size 500, then you can send multiple packets together & then will be able to send all the packets, in lesser no. of trips



Lecture 7 (select system call & I/O concurrency)

- for servicing multiple clients, there are two main approaches:

(i) forking with fork()
(ii) selecting with select()

fork() creates a new process to handle each incoming client. (highly inefficient due to high overhead due to context switching)

select() causes a single processor to handle all incoming clients without having to spawn separate child "server handlers".

- I/O multiplexing is used when a client has to deal with multiple descriptors. A server that handles both TCP & UDP.

- Need of multiplexing:

(1) when client is handling multiple descriptors
(2) client handling multiple sockets at same time
(3) If a server handles both TCP & UDP

- Significance of select function:

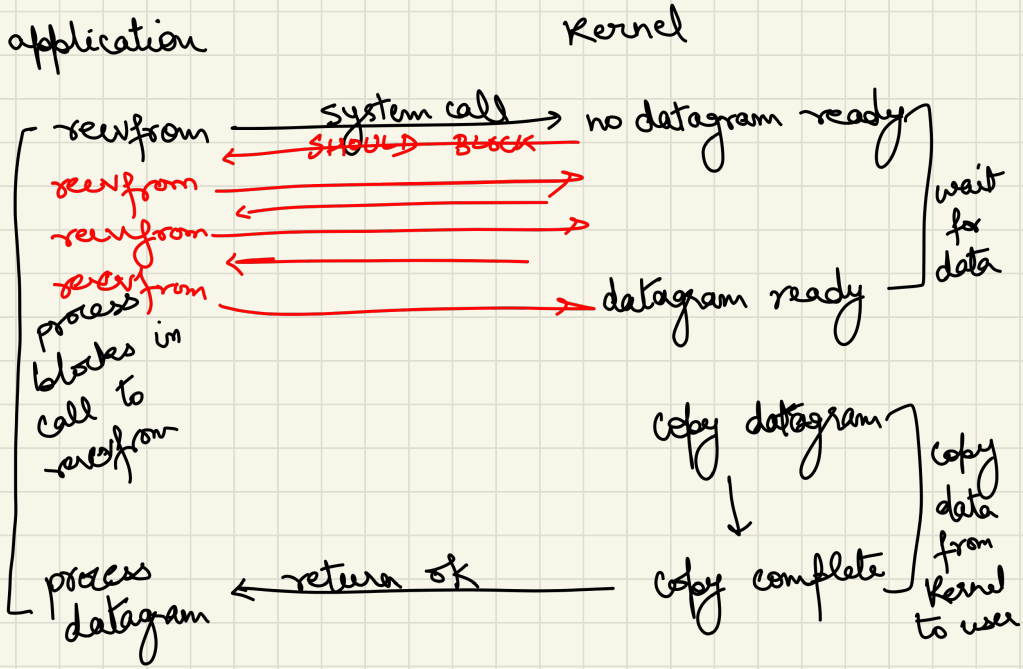
we encountered a problem when the client was blocked

in a call to fgets (on standard input) and the server process was killed.

The server TCP correctly sent a FIN to client TCP but since client process was blocked reading from standar input, it never saw EOF until it read from socket.

This is the capability of I/O multiplexing, provided by the select function.

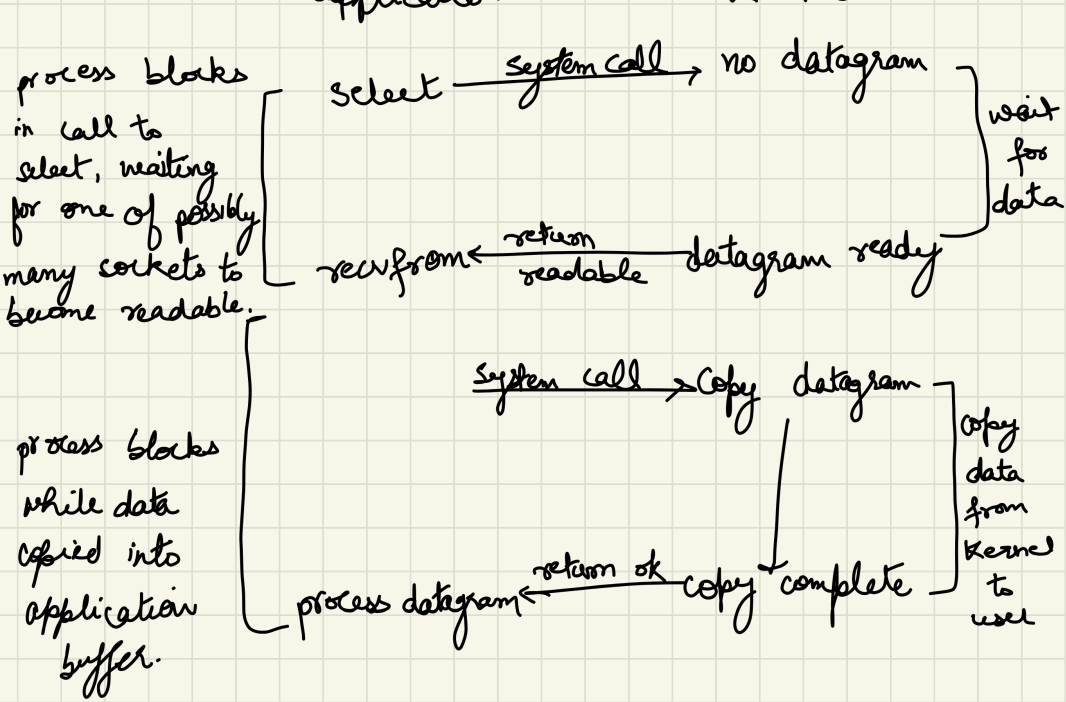
Blocking I/O : (Non Blocking)



select() :

int select (int maxfd, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval *timeout);

select ():



select system allows us to set blocking I/O on a set of descriptors

for eg. we can ask select to notify us when data is available for reading on either STDIN or a TCP socket.

The select system call provides a way for a single server to wait until a set of network connections has data available for reading.

maxfd = highest number assigned to a descriptor
readset = set of descriptors we want to read from
writeset = " " " " " " write to
exceptset = " " " " " " watch for errors
timeout = max^m time select should wait

- select will return if any of the descriptors in readset & writeset are ready. (number of descriptors is returned)
- FD_SET (int fd, fd-set * set)
  FD_ISSET (int fd, fd-set * set)
- setting timeval :

① setting timeout to 0, select() times out immediately
② setting timeout to NULL, select() will never time out and will block indefinitely until file descriptor is modified.
③ To ignore a particular file descriptor set, set it to NULL,
select (max, &readfds, NULL, NULL, NULL)

- struct timeval :

```
struct timeval {
    long tv_sec;
    long tv_usec;
};
```

```
struct timeval max = {1, 0};
```

using select() :

1. create fd_set.
2. clear the whole set with FD_ZERO.
3. Add each descriptor you want to watch using FD_SET.
4. Call select.
5. when select returns, use FD_ISSET to see if I/O is possible on each descriptor.

Operations :

void FD_ZERO (fd_set *fdset)

void FD_CLR (int fd, fd_set *fdset) — to remove

void FD_SET (" ") — to add

int FD_ISSET (" ") — to check

How to write the program for select system call ?

server side :

1. Create TCP i.e. listening socket.
2. Create UDP.
3. Bind both socket to server address.
4. Initialize descriptor for select & calculate max^m of 2 descriptors for which we will wait.
5. Call select & the ready descriptor.

TCP
bind for TCP
UDP
bind TCP & UDP
initialize
& calculate max
call select

⑥ Handle the new connection if ready descriptor is of TCP or receive datagram if ready descriptor is UDP.

UDP Client :

create UDP socket.
send message to server
wait until response from server is received.
close socket descriptor and exit.

TCP Client:

create a TCP socket.
call connect to establish connection with server.
when connection is accepted write msg to server.
read response from server.
close the socket.

server program :

```
int main() {
// create tcp socket
int tcp_id = socket (AF_INET, SOCK_STREAM, 0);
bzero (&server_addr, sizeof (serveraddr));
server_addr.sin_family      = AF_INET
server_addr.sin_port        = htons (port)
server_addr.sin_addr.s_addr = htonl (INADDR_ANY)
// bind the tcp socket
bind ( tcp_id, (struct sockaddr*) &serveraddr, sizeof(serveraddr),
```

```
listen (tcp_id, 10)

// create udp sockid
int udp_id = socket (AF_INET, SOCK_DGRAM, 0);
bind (udp_id, (struct sockaddr*) &servaddr, sizeof(servaddr));

// clear the descriptor set
FD_ZERO(&rset)

// get maxfd
maxfdp1 = max (tcp_id, udp_id) + 1;

for ( ;; ){
  FD_SET (tcp_id, &rset);
  FD_SET (udp_id, &rset);

  int nready = select (maxfdp1, &rset, NULL, NULL, NULL);

  if (FD_ISSET(tcp_id, &rset){
    len = sizeof (cliaddr)
  connfd = accept (listenfd, (struct sockaddr)* client_addr.
                   &len);
```

PYQ

Q1 Chatbot using TCP

Sol^n Client side :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <string.h>
#include <netdb.h>

int main() {

struct sockaddr_in servraddr;
char sendline[100], receiveline[100];

int sockid = socket (AF_INET, SOCK_STREAM, 0);
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl (INADDR_ANY);
serveraddr.sin_port = htons (2200);

connect (sockid, (struct sockaddr_in*) servraddr, sizeof
                                          (servaddr));

while (1) {
fgets (sendline, 100, stdin);
```

code

```
send (sockid, sendline, 100, 0);
if (strcmp (sendline, "exit", 4) == 0){ break; }
else {
recv (sockid, receiveline, 100, 0);
if (strcmp (receiveline, "exit", 4) ==0) { break;}
else {
  printf ("from server: %s", receiveline);
  bzero (receiveline, 100);
  }
 }
}
```

bind, listen, accept, recv, send

Server side :

```
# header files
int main ( ) {
 struct sockaddr_in serveraddr, clientaddr;
 char sendline [100], data [100]; int comm_sock;
 int sockid = socket (AF_INET, SOCK_STREAM, 0); [ ] code;
 bind (sockid, (struct sockaddr - in*) & serveraddr,
                              sizeof (serveraddr));
```

```
listen (sockid, 100);
while (1) {
comm_sock = accept (sockid, (struct sockaddr_in)NULL, NULL);
 while (1) {
 bzero (data, 100); → to ensure that any previously received data is cleared out
 recv (comm_sock, data, 100, 0);
 if ( strcmp(data, "exit", 4) == 0){
    break;
  }
 else {
 printf ( " received data : %s", data);
 fgets (sendline, 100, stdin); → Allows user to type a message to send back to client
 send (comm_sock, sendline, 100, 0) ;
 if (strcmp(sendline, "exit", 4) == 0){
 break;
 }
 }
}
```

q1 b I) listen → f" that announces willingness to accept connections

accept → blocks caller until connection request arrives

wait → f" used to wait until one or more sockets are ready to receive data
It returns as soon as atleast one socket has data available to be received.

connect → actively establishes connection

q2 # header files

Client

```
int main() {
int sockid = socket( ) ;  char str[100];
struct sockaddr_in serveraddr;
   ___
   ___
   ___
printf ("enter the string: ")
scanf (" %. [^\n]s", str );
Connect ( ) ;
int len = strlen ( str );
send ( sockid , str , len , 0 );
recv (                    );
```

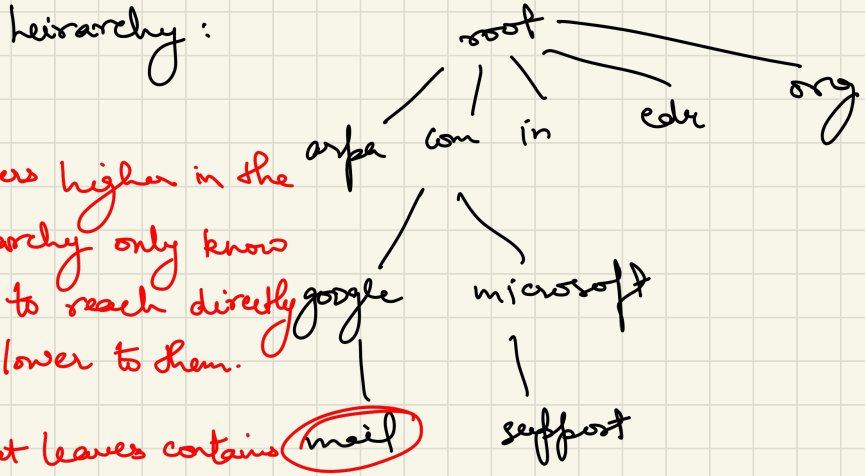
Server

```
# header files
int main() {
sockid = socket (      ) ; char str[100];
struct sockaddr_in server_addr;
==
==  ;
bind (                  );
listen (                );
comm_sock = accept  (  ) ;
recv (comm_sock , str , 100, 0 );
int i, j, temp ;
int len = strlen (str);
for (i=0 , j = len -i ; i<j ;  i++ , j--){
temp = str [i] ;
str[i] = str[j];
str[j] = temp;
}
send ( comm_sock, sizeof(str) , 0) ;
print ( " Reversed str : %s ", str );
return 0;
}
```

6) purpose of DNS : mapping of domain to IP name

Architecture of DNS : ① root level server
② top level servers (TLD)

hierarchy :



servers higher in the hierarchy only know how to reach directly ones lower to them.

DNS at leaves contains records of IPs of the server hosting the actual website.

root — root
.com — TLD
microsoft — microsoft's DNS server
support — " support " "
customer — microsoft's server which is hosting the actual website. Not a DNS.

customer.support.Microsoft.com

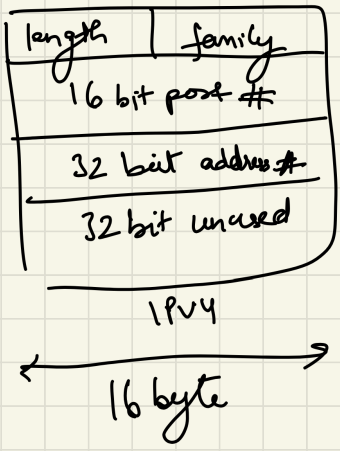
q 3 a)

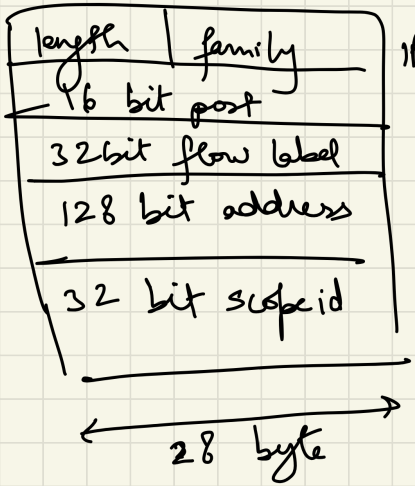
IPv4 : sin_family
sin_port
sin_addr
sin_zero

IPv6 : sin6_family
sin6_port
sin6_flowinfo
sin6_addr
sin6_scope_id : 4 byte field that specifies scope of the address

eg of IPv4 address : 192.168.0.1

IPv6 " : 2001:0db8:1333:4444:5555:6666:7777:8888





b) UDP

Source IP address	Destination IP address
Total length	checksum

TCP

Source IP	Dest. IP
Seq. No.	
Ack. No.	
HLEN | Reserved	flag | Window size
Checksum	Urgent ptr
option + padding	

U A R P S F
R C S S Y I
G K T H N N

FIN bit is used to release a connection

(a) Role of file descriptor?

Soln ① used to represent the socket as an open file in the OS
② It is an integer value, used to identify socket in the subsequent system calls.
③ creates a open communication endpoint

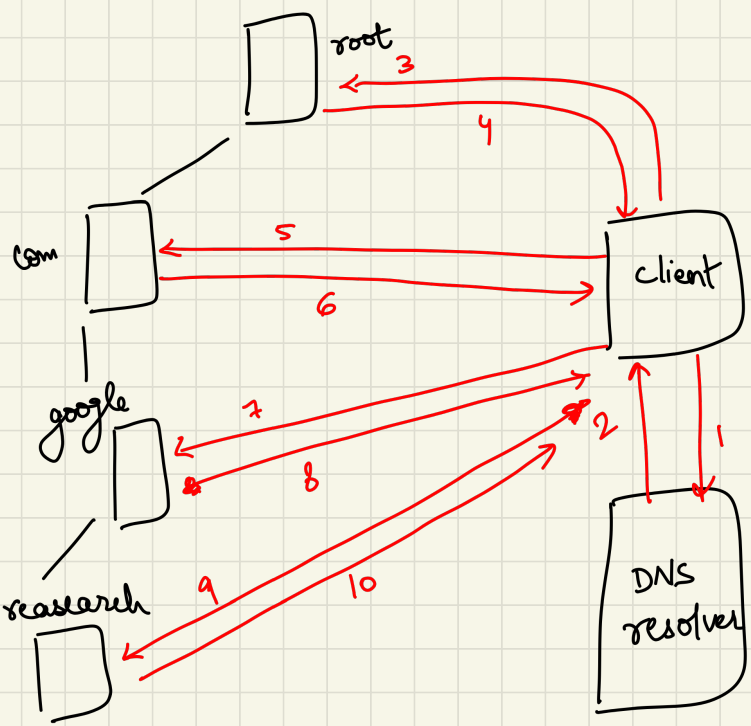
(a) Diff. b/w SCTP & TCP, UDP?

Soln SCTP combines features of both TCP & UDP and is used for multimedia comm. Can be used for multihoming too. It's main features are TSN, SSN, SI.

(2) steps to write recursive DNS query

"colab.research.google.com"

Soln



- Revise Reverse DNS query again !!

(q) 0532 | 0017 | 00000001 | 00000000 | 500207ff | 00000000

0532 → source IP
0017 → dest. IP
00000001 → seq.
00000000 → Ack. no.
500207ff → HLEN | Type of segment | Window size

in decimal

(5×256 + 3×16 + 2) = 1330
(16+7) = 23
1
0
(16×5) = 80
02
2047

(4) using getsockopt()

Sol^n

```
if (getsockopt(sockid, SOL_SOCKET, SO_BROADCAST,
                &optval, sizeof(optval)) < 0) {
    perror("socket broadcast failed");
    exit(EXIT_FAILURE);
}
```

(Q) UDP : B3260035001C001C

Source port no.	dest. port no.	total len	length of data
11	11 53	28	28

MULTICAST CODE (7)

(9) IP of size 1600 bytes
header of 30 byte
Max^m size of IP is 1400 byte
How would the IP be fragmented ?

Sol^n IP size = 1600, header size = 30, data size = 1570

MTU = 1400

No. of fragments = $\frac{1600}{1400} = 1.14 \approx 2$

fragment 1 : Packet size = 1400, byte range = 0-1399, MF = 1, Offset : 0

fragment 2 : = 200, 1400-1599, 0, = 1400/8

↓ more fragments

(2) int a = 340;
char *p;
p = (char*)&a;
printf("%d", *p);
printf("%c", *p);

Little endian machine

Sol" 340 = 10 10 10100 = 0000 0001 0101 0100

64 + 16 + 4 = 84

MSB LSB

output : 84
T

(2) max client = 2
one listening socket, multiple data comm. socket
(socket() TCP (accept)

listening & comm. socket ⟶ fd_set using select()

Sol" Server Side program :

header files
Max_client 2

```
int main ( ) {

int listenfd = socket (AF_INET, SOCK_STREAM,
                                    0);
    struct sockaddr_in serveraddr;

    serveraddr. sin_port = htons (2200);
    serveraddr. sin_family = AF_INET
    serveraddr. sin_addr. s_addr = INADDR_ANY

    bind ( listenfd, (struct sockaddr_in*)& serveraddr, sizeof(serveraddr));

    listen ( listenfd, max_client);
    fd_set rset;
    // Accept incoming connection and multiplex on same
      set fd_set
    while (1) {
     FD_zero ( &rset);

    FD_SET ( listenfd , rset);

    max_sd = listenfd + 1;

     for ( i = 0 to 2) {

        sd = data_socket [i]
        FD_SET (sd , & rset);
        }

    nready = select ( max_fd, & rset, NULL, NULL, NULL);
```

```
if ( FD_ISSET ( listenfd, &rset)) {

  int new_socket;

  struct sockaddr_in clientaddr;

  new_socket = accept ( listenfd, (struct sockaddr*)&client,
                        sizeof(clientaddr));

  for (i=0-2) {

    sd = data_socket[i];

    recv ( new_socket, data, 100, 0);

    send ( new_socket,

  }
```

AFTER MST !!

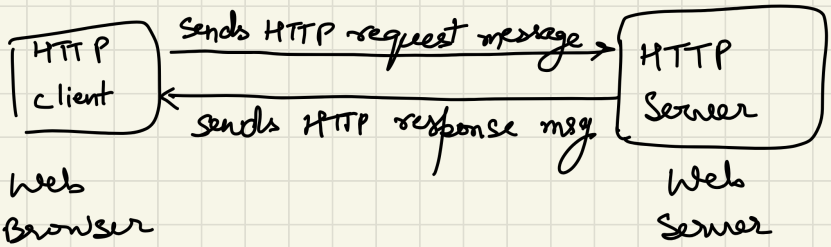
Lec-8 (HTTP server design and implementation)

- HTTP server is a software that understands URL and HTTP. (the protocol browser uses to view pages) (web addresses)

HTTP server can be accessed through domain names of the websites it stores and it delivers the content of these hosted websites to the end user's device.

HTTP (Hypertext transfer protocol)

- Interaction b/w server and web browser



- HTTP Client request :

when we want to connect to the server, we want some type of URL / address of website in the browser

To display the page, browser fetches the file index.html from a web server.

for eg:, www.example.com (Defaults : port 80, index.html, http protocol)

on entering the above link in the web browser, the web browser reconstructs the URL as :

http://www.example.com:80

- when browser sends HTTP request to a server, a HTTP header is sent.
(Let index.html be the default page displayed)

GET / index.html HTTP / 1.1

↑ method ↑ URL ↑ Protocol ↑ version

header

Host : www.example.com

User-Agent : Mozilla/5.0

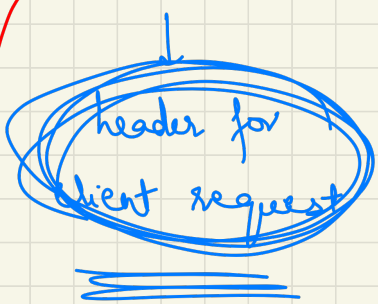
Accept : text/html, */*

Accept language : en-us

Accept charset : ISO-8859-1, utf-8

Connection : Keep-alive

HUAAAC



blank line

for POST and PUT method

Body (optional)

This HTTP request is received at server's end, which is processed by server.

- HTTP Server Response:

The client sent some headers to HTTP server and expects same in-return.

This is the HTTP response format the web browser is expecting from server.

HTTP Response - Read lines from socket

version → HTTP/1.1 | status → 200 | Status Message → OK

Date : Fri, 16 Mar 2018 17:36:27 GMT

Server : "YOUR SERVER NAME"

Content-Type : text/html ; charset = UTF-8

Content-Length : 1846

blank line

<?xml >

<!DOCTYPE html.... >

<html ... >

....

</html >

Header

DSCC

1

header for server response

Body

- HTTP Server Using Socket Programming :

Step 1 : Creating socket file descriptor

```
if ( (server_fd = socket (AF_INET, SOCK_STREAM, 0)) == 0){
    perror ("In socket");
    exit (EXIT_FAILURE);
}
```

Step 2 : Create a struct sockaddr_in address

```
#define PORT 80

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);
```

Step 3 : bind server_fd and address

```
if( bind(server_fd, (struct sockaddr*) &address,
    sizeof(address)) < 0) {
      perror("In bind");
      exit(EXIT_FAILURE);
    }
```

Step 4 : Listen on server_fd

```
if (listen(server_fd, 10) < 0) {
    perror("In listen");
    exit(EXIT_FAILURE);
  }
```

- Here 10 is queue_limit i.e. no. of active participants that can "wait" for a connection.

Step 5 : Accept client request using accept()

```
int addrlen = sizeof(address)

while(1) {

if (new_socket = accept(server_fd,(struct sockaddr *)&address,
                        (socklen_t*)&addrlen))<0){
```

used for further communication

```
perror("In accept");
exit(EXIT_FAILURE);

}
```

Step 6 : Read and print data received from client

```
while(1){

if ((new_socket = accept(server_fd, ...
.....

char buffer[30000] = {0}
valread = read(new_socket, buffer, 30000)
printf("%s\n", buffer);

}
```

Step 7 : Create HTTP Response in String

```
char * server_response = "HTTP/1.1 200 OK\n
Content-Type: text/html\n Content-Length: 72\n\n <!DOCTYPE
HTML><HTML><body><h1> Hello Thapar </h1></body></HTML>";

while (1) {

    - - -

}
```

Step 6 : Send Response to client and close the new socket

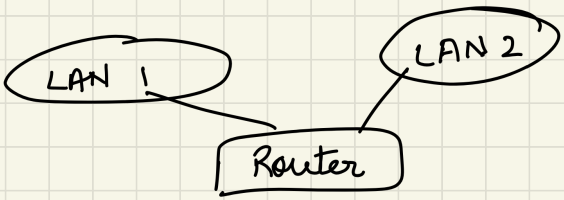
```
char * server_response = " ---- "

while (1) {

--- accept() ---
--- print client data --
write (new_socket, server_response, strlen (server_response));
printf (" -- Response sent to client --- \n");
close (new_socket);
}
```

Lecture-9 Virtual LANS

① What is LAN?



LAN = single broadcast domain = Subnet

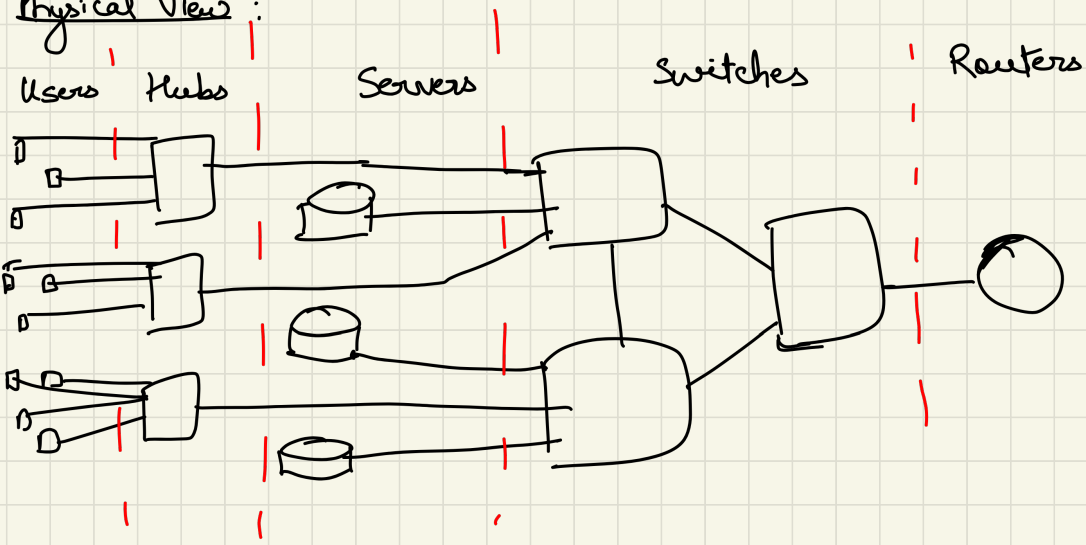
No routing b/w members of LAN

Routing required

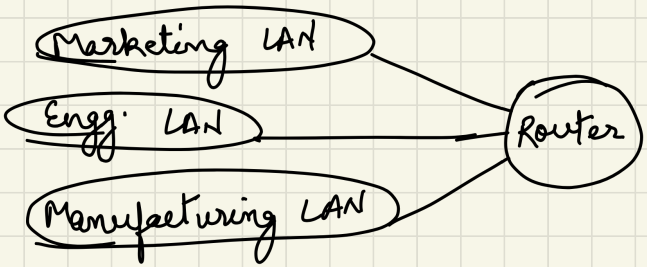
Immobility

Security

Physical View :



Logical View :



VLAN : they help in logical partitioning of switches (for making multiple LAN)

- A VLAN is a logical partition of a Layer 2 network.
- there can be multiple VLANs.
- each VLAN is a broadcast domain, usually with its own IP network.
- VLANs are mutually isolated and packets can only pass b/w them via router.
- partitioning of a Layer 2 network takes place inside Layer 2 device, usually via switch.
- the host grouped within a VLAN are unaware of the VLAN's existence.

Benefits of VLAN :

- security : broadcast msg of one VLAN is not going to disturb others

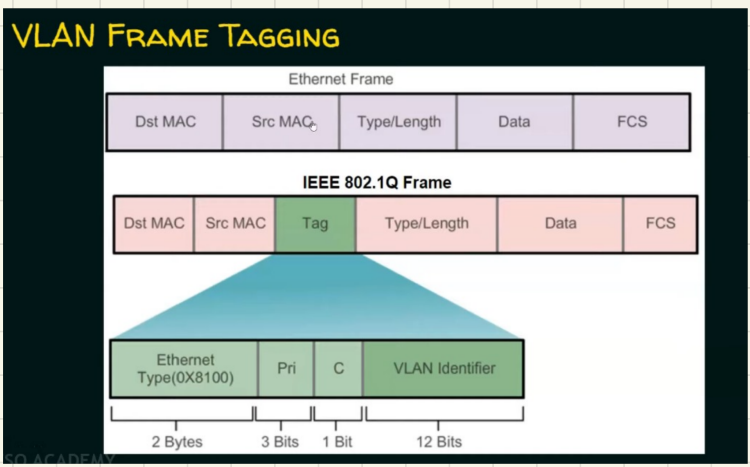
- Cost reduction : In place of purchasing multiple switches, we buy only required no. of switches & we are going to make partitions in the switch.
- Better performance : Max^m utilisation of the switch
- Shrink broadcast domain
- Improved IT staff efficiency
- Simpler project and application management

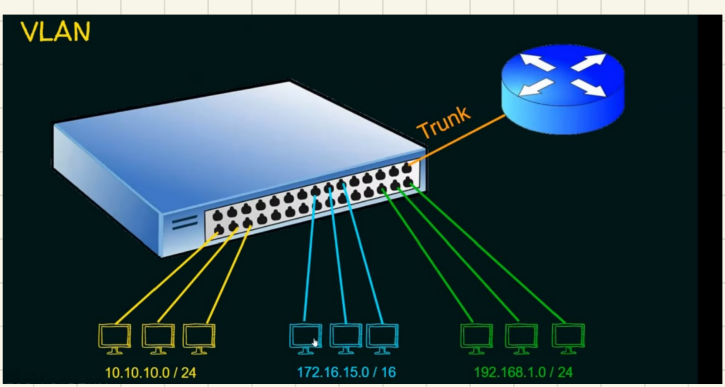
Types of VLAN : Data VLAN, Default VLAN, Native VLAN, Management VLAN, Voice VLAN

VLAN Tagging :

- Frame tagging is the process of adding a VLAN identification header to the frame.
- It is used to properly transmit multiple VLAN frames through a trunk link.
- Switches tag frames to identify the VLAN to that they belong. Different tagging protocols exist ; IEEE 802.1Q is an eg.
- The protocol defines str. of tagging header added to the frame.

- Switches add VLAN Tags to the frames before placing them into trunk links and remove tags before forwarding frames through non-trunk ports.
- When properly tagged, the frames can traverse any no. of switches via trunk links and still be forwarded within the correct VLAN destination.





Virtual LAN :

Virtual LAN = Broadcasts and multicast goes only to nodes in VLANs.

- LAN membership defined by network manager ⇒ virtual

Why VLAN ?

- Location independent (Marketing LAN can be all over the building)
- Users can move but not change LAN.
- Traffic b/w LANs is routed (keep all traffic on one LAN)
- Switch when you can, route when you must.
  (Do not VLAN over expensive WAN Links)
- Better security.

Types of VLAN :

- Layer-1 VLAN = Group of physical ports
  Layer-2 VLAN = group of MAC addresses
  Layer-3 VLAN = IP subnets

- Layer-1 VLANs :

- Also known as port switching
- can be used to provide security and isolation
- does not allow user mobility
- Moved user has a new subnet ⇒ new IP address
  ⇒ May go through a router to access the old server

- Layer-2 VLANs:

= LANs defined by a list of MAC addresses
- Provides full user movement
- client and server always on the same LAN regardless of location.
- Problem: Too many addresses need to be entered and managed
- Notebook PCs change docking stations

Alternative: Membership implied by MAC protocol field.
eg VLAN1 = IP, VLAN2 = ...

Ethernet

Dest. Address	Source Address	Protocol Type

802.3

Dest. Address	Source Address	Length	

AA	AA	03	Protocol Type	

- Layer-3 VLAN:

Dest. Addr	Source Address	Protocol Type	

IP Dest. Addr	IP Source Addr	

- Also known as virtual subnet

- VLAN membership implied by MAC-Layer protocol type field and subnet field 123.34.*.*
- VLAN configuration is learned by switches
- Stations do not belong to VLANs, packets do.
- Multiprotocol stations are put into multiple VLANs.

- Higher Layer VLANs :

- Different VLANs for different applications :

① FTP

② Multimedia

- Service based VLANs : All workstation using EMAIL server are on the EMAIL-VLAN, all workstations using employee database server are on HR-VLAN.
- IP multicast address based VLANs
- General policy based : VLAN membership can be based on a combination of incoming port, MAC address, subnet or higher layer info., time of day

- VLAN Tagging :

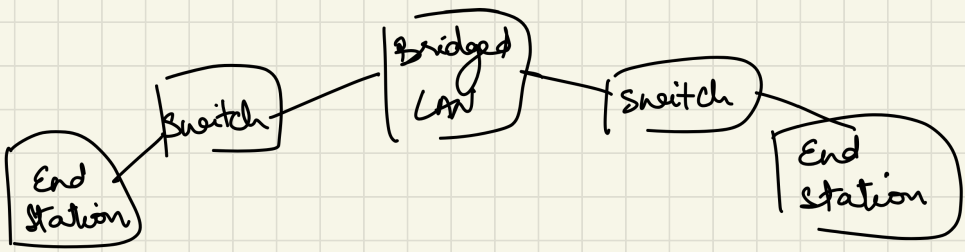
Dest. Addr.	Src Address	VLAN Tag	Prot. type

first switch adds tag containing VLAN id to all incoming packets.

Intermediate switches do not recompute the VLAN id.

Last switch removes tags from all outgoing packets.

Tag is not swapped at every hop like labels.



- IEEE 802.1Q (features)

Allows upto 4095 VLANs (0 to 4095 = 4096)

upward compatible with existing VLAN-unaware hubs & bridges

supports shared multimedia & switched LANs

upward compatible with existing VLAN aware hubs and bridges

Allows mixing legacy bridges and VLAN aware bridges

Retains plug and play mode of current LAN bridges

extends 802.1p priority mech. to priority based on

VLAN membership

VLAN based priority takes precedence over other priority considerations

Allows signalling priority info. on non-priority LANs (CSMA/CD)

Allows both local and universal MAC address

Operation with / without explicit VLAN header in the frame

Supports static and dynamic configurations for each VLAN

Allows intermixing different IEEE 802 MACs and FDDI

Allows signaling source routing info. on CSMA/CD LANs

each VLAN is a subset of a "single" physical spanning tree

Doesn't preclude feature extensions to multiple spanning trees

- Overlapping VLANs:

Multiple stations with same individual address

one station with multiple interface using same address

Restriction: One station or interface per VLAN

- Tagging Rules:

① On a given LAN segment for a given VLAN, all frames should be either implicitly or explicitly tagged

② Different VLANs on the same segment may use different options

Access links: contain VLAN unaware devices, All frames on access links are untagged

Hybrid links: contain both VLAN-aware and VLAN unaware devices

Tagged frame format:

Tag header:

TUCV

16 b	3b	1b	12
TPID	user priority	CFI	VLAN Id

Ethernet Frame:

6B	6B	4B	2B	0-30B		4B
DA	SA	Tag	PT	[RIF]	Data	FCS

802.3 frame:

6B	6B	4B	2B	0-30B		42-1478	4B
DA	SA	Tag	Length	[RIF]	LLC	Data	FCS

TPID = Tag protocol ID

CFI = Canonical format Indicator
- = Bit order of address info. in TR/FDDI frames
- = Presence/absence of RIF in 802.3/Ethernet frames

RIF = Routing info. field

Next ' Type: 01 = Transparent frame

DA = Destination address, SA = source address

PT = Protocol type, LLC = Logical Link Control

FCS = Frame check sequence

Lagest data size on 802.3 = 1470

- Communication b/w VLANs:

Need routers

can use 1-armed VLAN-aware router

VLAN-aware switches can route b/w VLANs

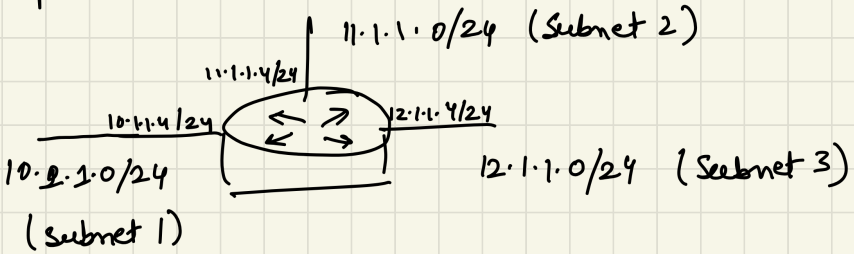
Such switches can be placed in the core, edges & everywhere

VLAN Benefits : Segmentation, Mobility, Reduced broadcast domain, Resolved thrashing

Lec-10 VLAN Routing (L3 Routing)

- VLANs are basically subnets.

- Router possession:



L3 is in possession of three subnets. It means, this router should receive the traffic for destination ip addresses = member of any subnets possessed by the router.

- Subnet interfaces: (To add more interfaces)

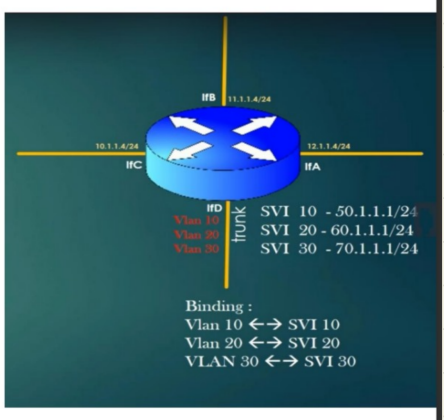
① Create logical interfaces and assign ip/mask to them.
② These logical interfaces are given special name as SVI. (switch virtual interfaces)
③ SVIs are not physical hardware but software based interface.
④ So we can create as many SVIs as I want.

Switch Virtual Interfaces (SVI)

- Let us create SVI10, SVI20, SVI30 on Router and assign ip/mask to them as shown
- We have also created one loopback interface LO1 with ip-100.1.1.1/32
- Now router is said to be in possession of following subnets:
  1. 11.1.1.0/24
  2. 10.1.1.0/24
  3. 12.1.1.0/24
  4. 50.1.1.0/24
  5. 60.1.1.0/24
  6. 70.1.1.0/24
  7. 100.1.1.1/32
- So using the concept of SVIs you can make a L3 router is possession of as many subnets as you want without attaching real physical interface/hardware.
- Now this router must receive the traffic with destination ip address =member of any subnet possessed by L3 router.
- So, in addition to real physical subnets this router would receive traffic destined for ip address 50.1.1.x, 60.1.1.x, 70.1.1.x or 100.1.1.1 (self).

L3 Router Configuration for VLAN forwarding

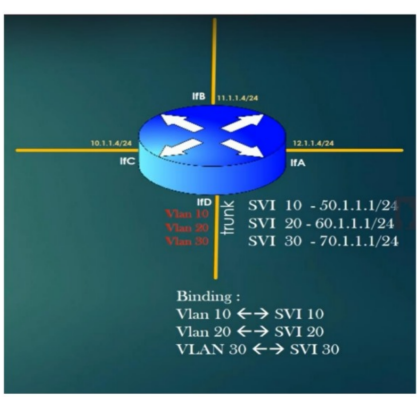
- Take one physical interface of a router, say ifD and configure it as a Trunk.
- Configure ifD to operate in VLAN 10,20 and 30.
- Bind SVI interfaces to corresponding VLANS, one to one mapping.
- By binding the SVIs-VLAN together, VLANS borrow the network id of SVI's.
- For example, VLAN 10 network id will be 50.1.1.0/24. All host machines present in VLAN 10 must be configured with ip address 50.1.1.x/24.



L3 Router Configuration for VLAN forwarding

- Steps for router-VLAN routing:

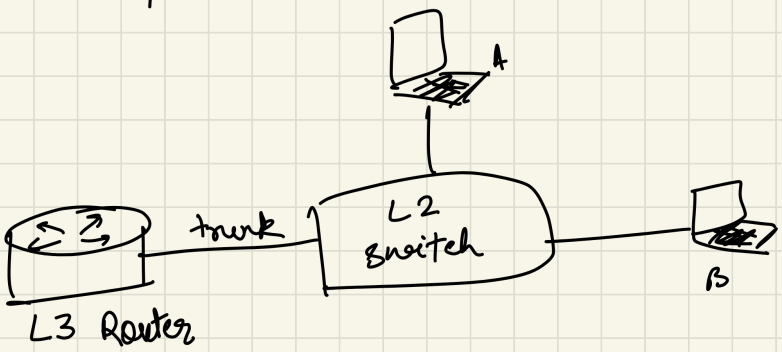
1. If router receives a traffic for destination = 60.1.1.10, it checks in which SVI the Destination ip address is a member of, in this case it is SVI 20 (60.1.1.10 lies in subnet 60.1.1.0/24)
2. Router then checks the VLAN bind to SVI 20, which is VLAN 20
3. Router tags the packet with VLAN id 20
4. Router forwards the packet out of all local physical interfaces which are operating in VLAN 20, in this case interface ifD only.



⑧ Diagram for VLAN forwarding example

Lec-11 Inter-VLAN Routing :

Problem: How will host A talk to host B ?



- Traditional L3 Routing :

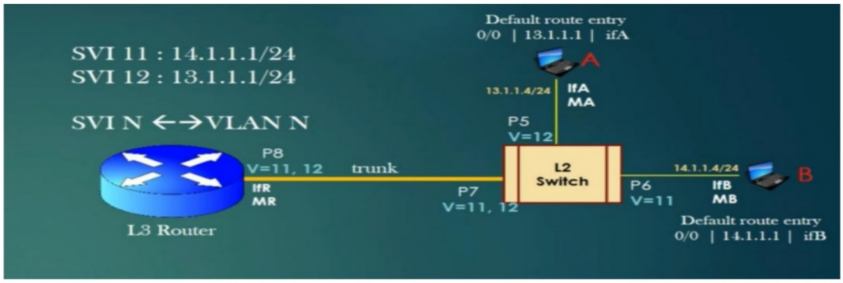
Whenever a host machine A needs to communicate with host machine B present in a remote subnet, A sends frame to its gateway router using default entry in its routing table.

To do so, host machine A must know the MAC address of default gateway IP

or it must have ARP entry 13.1.1.1 ←→ MR2 in its ARP cache. Once A has this ARP mapping, A can send a frame to L3 gateway router using dest MAC = MR2 in ethernet header.

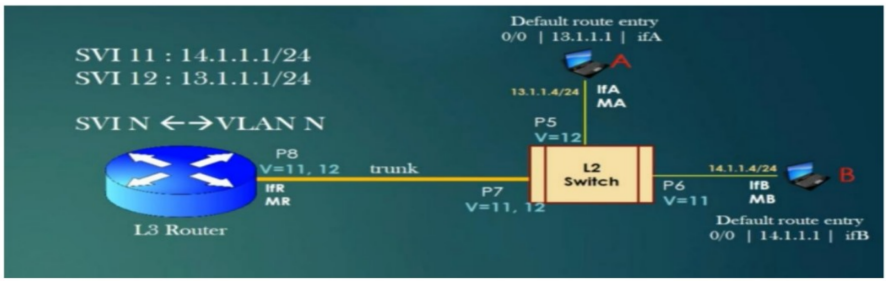
- Inter VLAN Routing :

Default Route



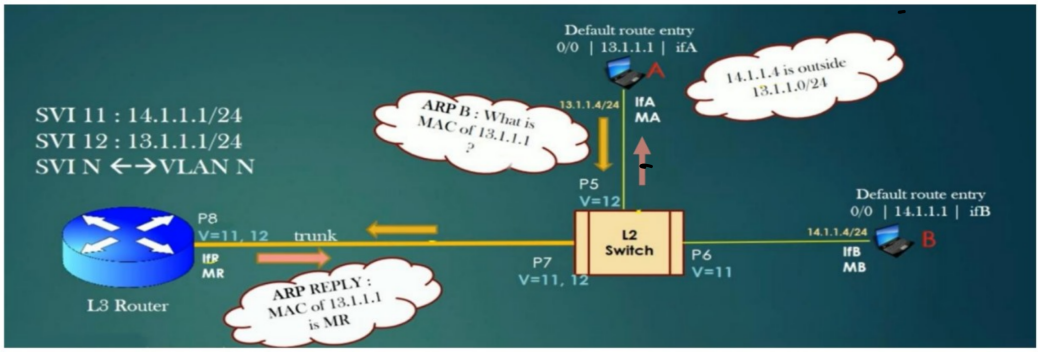
- To support inter-vlan routing, every host machine in the network is configured with a default route
- Thus default route of host machine= IP address of SVI interface which is bind to VLAN on which host machine is present.
- Thus A's default route = 13.1.1.1 because A is in VLAN 12, and VLAN 12 binds with SVI 12.
  - B's default route = 14.1.1.1 because B is in VLAN 11, and VLAN 11 binds with SVI 11
  - Default route on host A's routing table is installed as: 0.0.0.0/0 / 13.1.1.1 / ifA
- Whenever the host needs to send a frame to destination machine which is outside its own subnet (vlan), host machine use a default route.

ARP Resolution for Default Routes



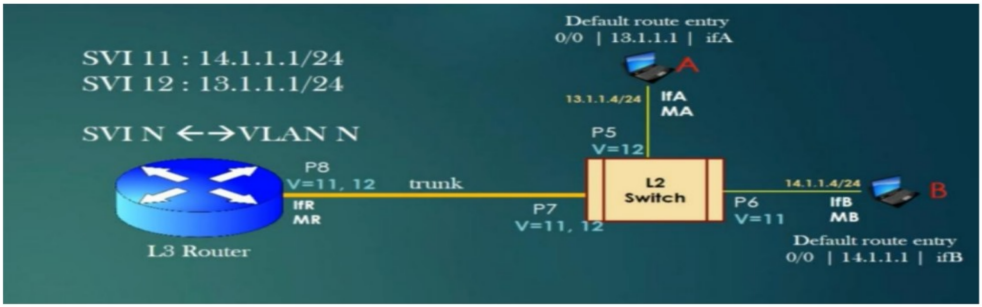
- Whose MAC address L3 router would return if it receives ARP Broadcast message for ARP resolution of IP= IP address of SVI Interface?
- For example, Host Machine A issues ARP Broadcast message to know MAC for default ip=13.1.1.1
- 13.1.1.1 is the IP of SVI 12, but SVI 12 is a logical interface, then whose MAC should L3 router must return in ARP reply?
- ANS: MAC of physical interface on which ARP Broadcast message is received .
- In this case, MAC=MR will be returned in ARP reply.
- Thus L3 router returns MAC=MR in ARP reply for IP address = SVI's ip address (14.1.1.1 or 13.1.1.1) since these are the ip addresses of one of router's local interface (however logical interface) of L3 router.

Inter VLAN Routing Steps



- Steps: from source host machine A to L3 gateway router
  1. A finds the B's ip=14.1.1.4 belongs to remote subnet.
  2. A decides to send data using default route which states that gateway ip is 13.1.1.1, outgoing interface is ifA.
  3. To send data to gateway router, A needs MAC address of gateway router having IP 13.1.1.1
  4. A checks its ARP cache to resolve ARP for default gateway ip 13.1.1.1
  5. A's ARP cache is empty, A launches ARP B message out of ifA to know MAC for 13.1.1.1
  6. L3 router returns MAC MR in ARP reply to A.

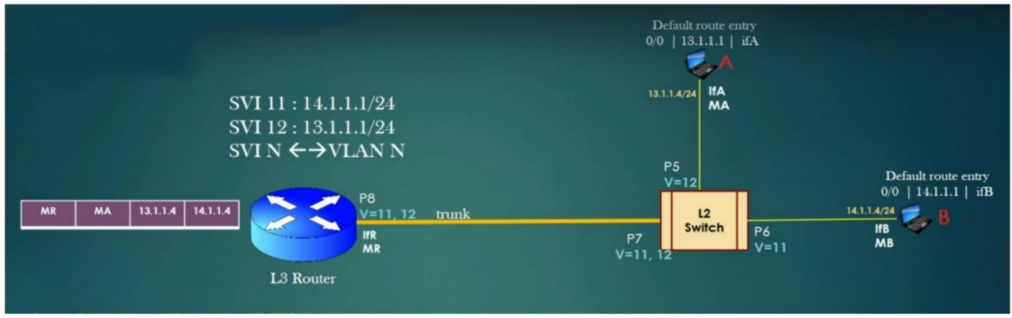
Inter VLAN Routing Steps



Steps : from gateway L3 router to destination host B.

10. The frame received by L3 router is tagged with VLAN 12.
11. Router checks the dst ip 14.1.1.4 address in frame belongs to SVI 11's network id
12. Router finds the VLAN bound to SVI 11 –in this case VLAN 11
13. The router interfaces operating in VLAN 11 is P8 (ifR)

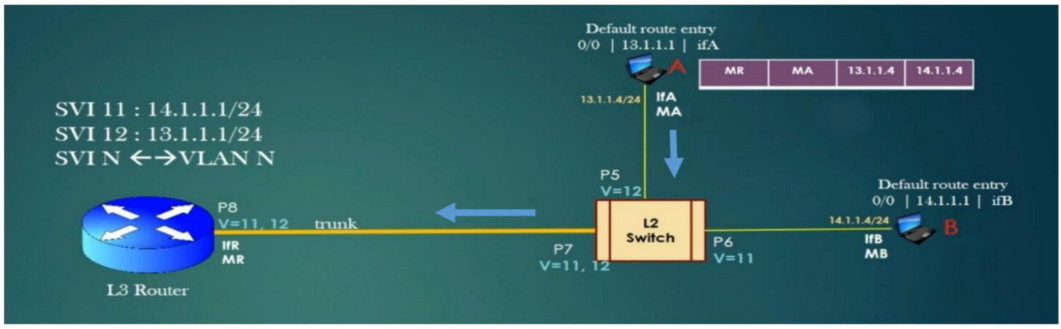
Inter VLAN Routing Steps



Steps : from gateway L3 router to destination host B.

14. Router prepares the frame: Dst MAC: MB, Src MAC: MR, Src IP: 13.1.1.4, Dst IP: 14.1.1.4
    If router do not know the Dst MAC MB then it launches ARP broadcast message for Dst ip=14.1.1.4 on all interfaces operating in VLAN 11.
15. Router retags the frame from VLAN 12 to VLAN 11
16. Router sends out frame on interface P8
17. Host B receives the untagged frame.

Inter VLAN Routing Steps



- Steps: from source host machine A to L3 gateway router
  7. A prepares the Frame: Dst MAC: MR, Src MAC: MA, Src IP: 13.1.1.4, Dst IP: 14.1.1.4
  8. This frame is received by L3 router only, and now L3 router needs to forward the frame to host machine whose IP is 14.1.1.4
  9. Now rest of the steps are same as that of L3 router-VLAN routing which we learnt in previous Module.

Lec-12 L2 & L3 Routing:

- Network layer is focused on getting packets from source to the destination, routing error handling and congestion control. It's functionalities :

Addressing, Packeting, Routing, Inter Networking

addressing : maintains address at frame header of both src & dest.

Packeting : performed by internet protocol. The network layer converts packets from its upper layer.

Routing : (Most imp.) layer choose the most relevant and best path for data transmission from src to dest

Inter-N/w : It delivers a logical connection across multiple devices

- Routing / forwarding - process to deliver packet by choosing an optimal path from one network to another

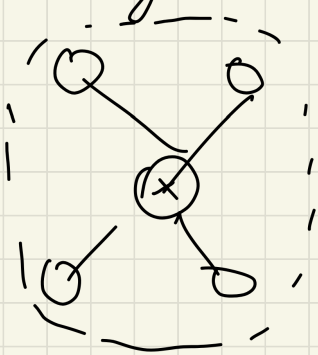
i) Routing decides in which direction to send the packets
ii) " is process of population routing or forwarding table
iii) Routers exchange messages about networks they can reach.

forwarding requires host & destination to have a routing table

- forwarding is process of sending a packet on its way (forwarding means to place the packet in its route to its destination)

- Router: It is a device for forwarding packets b/w computer network.

When a packet arrives at a Router, it examines destination IP address of a received packet & make routing decisions accordingly.



Routers use routing tables to determine out which interface the packet will be sent.

Each router's routing table is unique & stored in RAM of device.

Routing table: Table used to determine where data packets traveling over an Internet Protocol (IP) network will be directed.

Entries of IP Routing Table: Network ID, Subnet Mask, Next Hop, Outgoing Interface, Metric

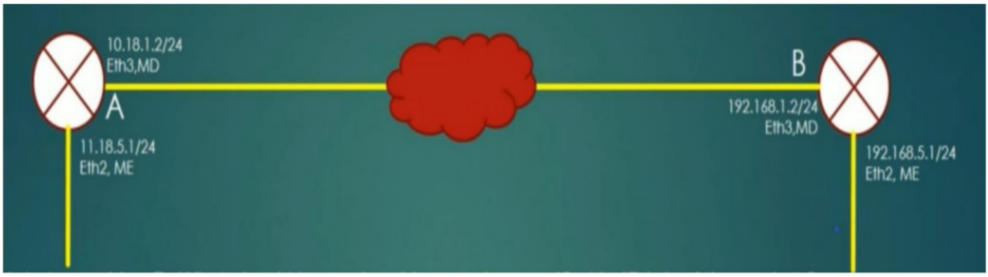
routing metric — hop count, delay, bandwidth, load, ...

forwarding techniques — N/w specific, Host, ...

Lec-13 Loopback Interface:

Loopback Interfaces

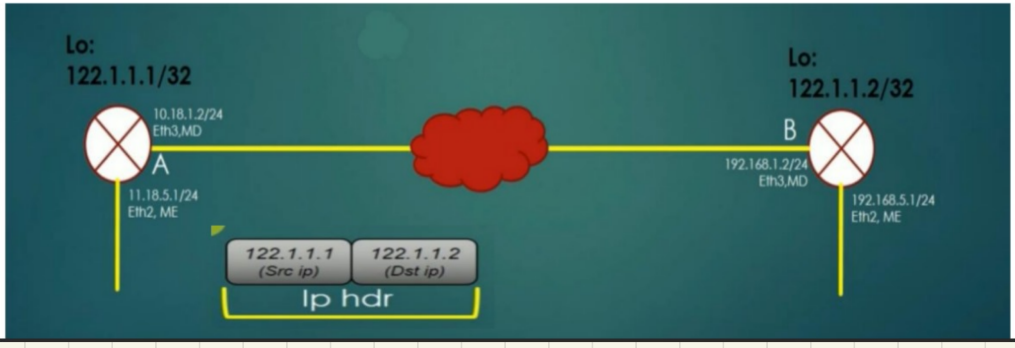
- When we send a packet destined to a remote machine, what is the IP address we should specify as source and destination IP address?



- Ans is: A can specify source IP = IP of any local interface and destination IP = IP of any local interface of B
- And lets call it IP-RULE
- Thus any combination of source IP 10.18.1.2 or 11.18.5.1 and destination IP 192.168.1.2 or 192.168.5.1 would lead to same result. That is packet would be delivered to B by network layer.

Loopback Interfaces

- There is a need to come up with special IP addresses which identify the L3 router as a device in the network and should be unique.
- Loopback addresses help achieve this goal.
- Loopback addresses are used to represent the identity of the networking device.

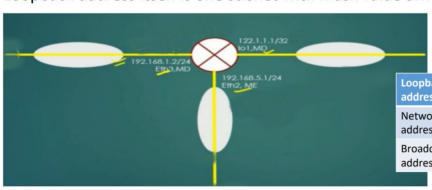


Properties of loopback interface

- only interfaces have ip addresses and not machines. We need ip addresses to identify machines uniquely so we use loopback interface.
  loopback interface is not hardware, but ~~software~~.
- 'A' sends the packet to 'B' using loopback interface by assigning source IP = ip of loopback interface of A
  dest IP = ip of loopback interface of B
- B consumes it ~~out~~ of the loopback concept but not physical

Properties of loopback interface

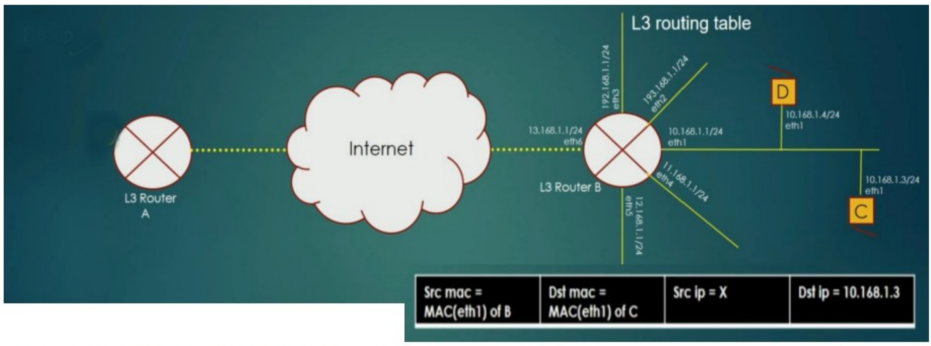
- View loopback interface as just another interface.
- Loopback address itself is one subnet with mask value always 32.



Loopback address	122.1.1.1/32
Network address	122.1.1.1/32
Broadcast address	122.1.1.1/32

- Loopback subnet (A.B.C.D/32) is always a local subnet for the local router and remote subnet for any other network device.

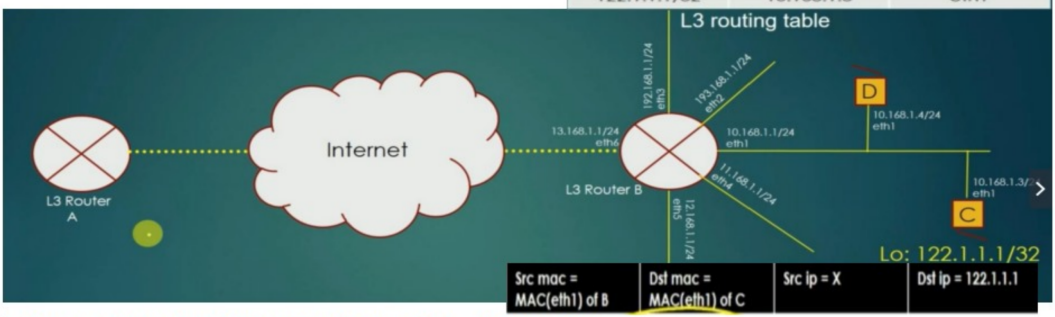
Layer 3 Routing Using Loopback IP Addresses



Destination IP address in the packet = 10.168.1.3

Destination IP address falls in the local subnet of router B, therefore router B propagates the packet to the destination C via L2 routing.

Destination Subnet	Gateway Ip	OIF
10.168.1.0/24	-	-
122.1.1.1/32	10.168.1.3	eth1



- Destination IP address in the packet = 122.1.1.1
- Dst IP address falls in the subnet (122.1.1.1/32) which is remote subnet wrt router B, therefore router B propagates the packet further to the destination C via L3 routing.
- If dest address is loopback address of the dest machine, packet never undergoes L2 routing.

Layer 3 Routing Using Loopback IP Addresses

- Machines D,E and F are present in the same local subnet 12.1.1.0/24
- D,E and F can communicate with each other using Destination IP addresses as 12.1.1.x using pure L2 routing.
- Ex: D can send a packet to E using Dst. IP address as 12.1.1.3

Layer 3 Routing Using Loopback IP Addresses

- But can D send a packet to E using Dst. IP =122.1.1.1?
- 122.1.1.1 do not falls in any local subnet of D.
- Therefore, D is trying to send a packet to remote subnet.
- Hence, D can send a packet to E using only L3 routing and for that D should have following L3 routing entry.

L3 entry:	122.1.1.1/32	122.1.1.3	IFD

- If Dst address is loopback address of dst machine, packet never undergoes L2 routing even if communication machine are physically present in same subnet.