16-01-2025

Internet

↳ A networks of networks
  ↳ Each n/w may be under an autonomous organization.
  ↳ N/w of a large no. of heterogeneous n/ws.
    (wireless, fiber, copper, satellite, sensor, etc.)

↳ A giant network formed by
  - access networks (network edge)
  - core network (backbone of the internet).

↳ Millions of connected computing devices, running network apps.
  ↳ hosts = end systems.
  ↳ communication links → fiber, copper, radio, satellite.
    ↳ Transmission rate = Bandwidth
  ↳ routers : forward packets (chunk of data).

↳ Network software
  ↳ Implementation of applications; core n/w software and protocols.

↳ Protocols (control) : sending, receiving of messages.
  ↳ e.g., TCP, IP, HTTP, Skype, Ethernet.

↳ Internet standards
  ↳ RFC: Request for comments
  ↳ IETF: Internet Engineering Task Force
  ↳ IEEE standards.

↳ A n/w consists of:
  - Hosts, comm. links, protocols, standards and routers.
  - Hardware and software.

22-01-2025

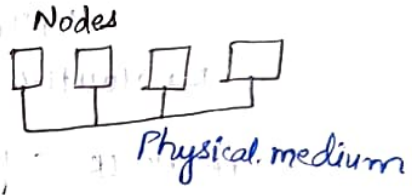
Network Topologies

↳ Org. of entities that form a computer network.
↳ Pattern of interconnection of elements of a computer n/w.
↳ How the hosts, routers and links are connected.

Eg. Bus
star
string
Extended star
Ring
Tree
Mesh → Full
Mesh → Partial

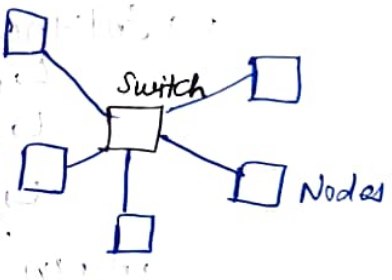
Bus:

↳ Nodes connected to broadcast medium.
↳ Eg. Ethernet (with coaxial cable)
↳ No. of links required: 1 to N-1
↳ Popular for: LANs
Cable TV n/w
Broadcast network (e.g., wireless n/w)



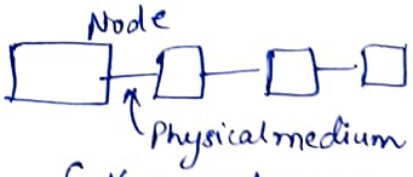
Star:

↳ All nodes connected to a central switch.
↳ A spoke and hub technology.
↳ Eg. Ethernet with twisted pair cable.
↳ No. of links required: N



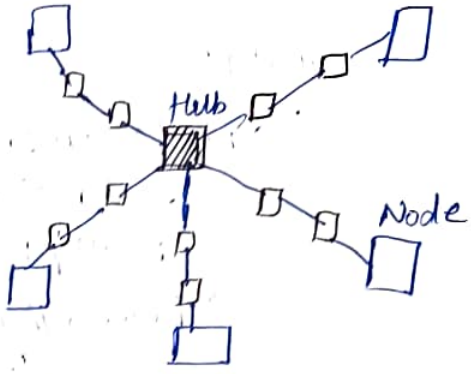
String (linear chain):

↳ Nodes connected in linear fashion.
↳ Nodes help forward the traffic.
↳ No. of links required: N-1
↳ Popular for:
- Relay n/w
- wireless or wired



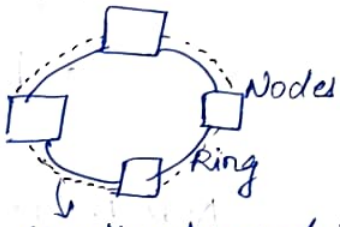
(others nodes may not get comms.)

Extended star:



↳ Modified star topology
↳ Hub and nodes connected through relays/string topology.
Eg: Ethernet with twisted pair cable.
↳ No. of links required: $N(M+1)$ (N: nodes, M: no. of relays per node)
↳ Popular for: • MANs
• Extended LANs.

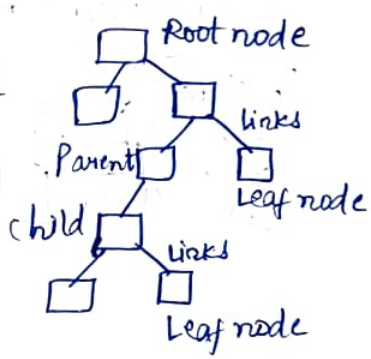
Ring:



↳ Nodes connected by links to complete ring.
↳ No. of links required:
Single ring → N
Dual ring → 2N
↳ Popular for optical ~~LA~~ MANs.

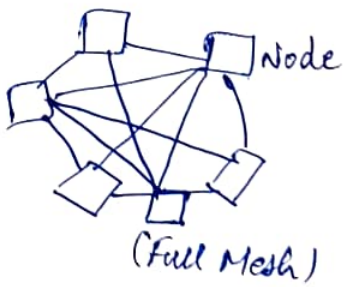
if a link breaks/fails then also it works

Tree:



↳ Point-to-point links are used to create a topology such that there is a root node, intermediate nodes and leaf nodes in ~~the~~ n/w.
↳ Only one route b/w 2 nodes.
↳ Maximally Acyclic (no loop).
↳ Minimal connectivity ⇝ every edge is critical for the n/w.
↳ No. of links required: $N-1$
↳ Popular for traditional telecomm. n/w and cellular backbone n/w.

Mesh:

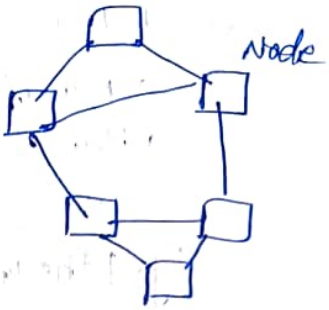


↳ Multiple paths b/w source-destination pair, unlike tree.

Full Mesh:

↳ A single hop link to every distination node.
↳ ~~Single~~ Multiple single or multi hop links b/w source-destination pairs.
↳ Highly redundant, but not scalable.
↳ Total links: $N \times (N-1)/2$

Partial Mesh:

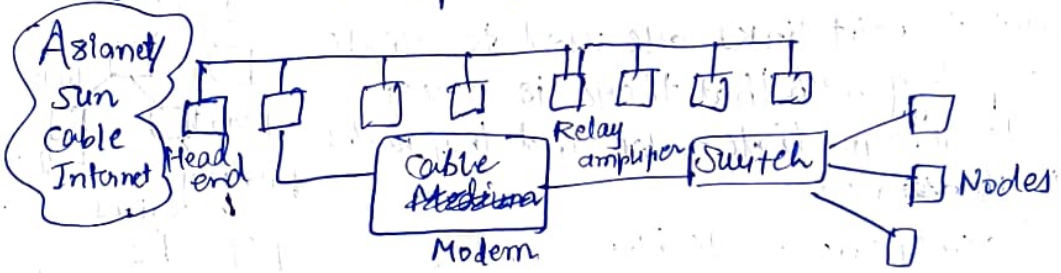


- Node doesn't have all the nodes as its neighbours and there exists multiple paths to destination node.
- Primarily uses multihop comm. to reach destination nodes.
- Both scalable and redundant.
- No. of rings required: $(N-1)$ to $N \times (N-1)/2$
- Popular for internet backbone & in wireless mesh n/w.

Real Network Topology

- Combination of multiple topology types.

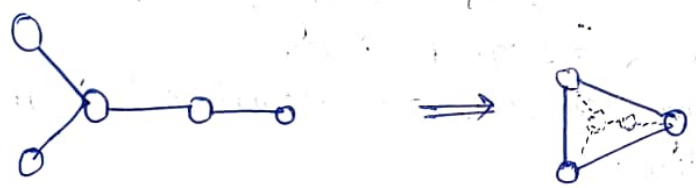
Eg. Home n/w topology



24-01-2025

Physical Topology	Logical Topology
• Refers to the physical connectivity, layout, pattern or organization of computers within a network.	• Refers to the abstract representation of how data flows and is transmitted within a n/w, independent of its physical topology.
• Depicts physical layout. Eg. Bus medium	• Depicts logistics of n/w concerned with transmission of data. Eg. star topology overlaid on Bus.
• Can be modified at higher cost.	• Can be modified at lower cost.
• Actual links and edges concerned with transmission of packets b/w nodes.	• High level representation of data or packets flow.
• Physical connection of n/w.	• Data path followed of the n/w.

Eg.



Communication Requirements :

① Information
② Energy to transmit.
③ A medium to transmit
④ A set of rules.

Protocol

↳ Defines format, order of messages sent and received among network entities, and actions taken on message transmission/receipt.

Protocol Design choices :

① Text Commands Based protocol :

↳ Protocol commands in human readable text, e.g., POST, GET, SET, RESET, ON, OFF.
↳ Simple, human readable, easy to diagnose, convenient.
↳ High bandwidth overhead (e.g., RESET takes ~40 bits).
↳ Usage: Application layer protocols where B/W constraint is not severe. Eg. HTTP, DNS

② Binary commands based protocol

↳ Protocol commands in binary coded form (2 bit each), eg, POST: 01, GET: 11.
↳ Bandwidth and storage efficient, machine readable.
↳ complex for humans, complex to program, Full n-command set takes only 2n bits.
↳ Usage: lower layer, proprietary protocols, protocols for BW constrained. Eg SNMP.

29-01-2025

③ Stateless Protocols: (e.g. Ethernet)

↳ End-nodes don't maintain the state of session/communication.
↳ Simple and efficient (light weight)
↳ Complex protocols with many states not possible.
↳ Eg. B/w constrained devices (DNS, SNMP, light, simple and low overhead protocol).

④ Stateful Protocol: (e.g. TCP) → Transmission Control Protocol.

↳ End nodes maintain the state at all times.
↳ Permit complex protocols with many states.
↳ Actions based on states can be taken.
↳ Storage/computing resources are required to maintain state information.
↳ Not scalable for large servers.
↳ Eg. where certain preferential treatment & quality of services are needed.

Network Core

↳ A mesh (topology) of interconnected routers.
↳ Belong to multiple organizations (autonomous systems).
↳ Span various countries or continents.
↳ Data transferred through internet → Techniques:

① Circuit switching → dedicated circuit per call → telephone net

② Message switching → Large sized messages (~KBs) sent through store-and-forward strategy.

③ Packet-switching
- Packets: much smaller messages
- Data sent through net in discrete small 'chunks'.

④ Cell switching
↳ Data sent through much smaller chunks than packets over virtual circuits.

Circuit Switching :

- End-end resources reserved for "call".
- Not shared with anyone else.
- call setup required.
- Guaranteed performance.

30-01-2025

- If the max. limit is reached (for a node), no further calls can be made. : Delimitation.
- Network resources (eg. bandwidth) divided into 'pieces' which can be shared among users.
- If the dedicated allocated BW is not used, it will be wasted.

BW of Telephone line :

Audible range : 400Hz – 4KHz

Sampling without loss of info. (Nyquist criteria) :

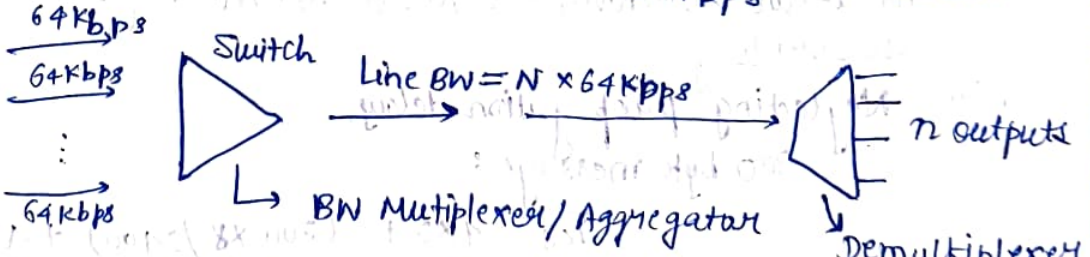
↳ 4KHz × 2 = 8KHz sampling

↓

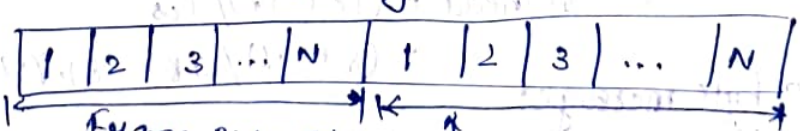
Digital info. (ADC of 8bits)

Total BW of the line = 8KHz × 8bits = 64Kbps.

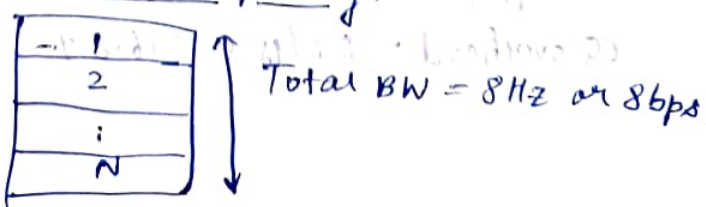
If line BW = 1Mbps, we can have $\frac{1\,Mbps}{64\,Kbps} = 16$ users.



Time division multiplexing :



Frequency division multiplexing :

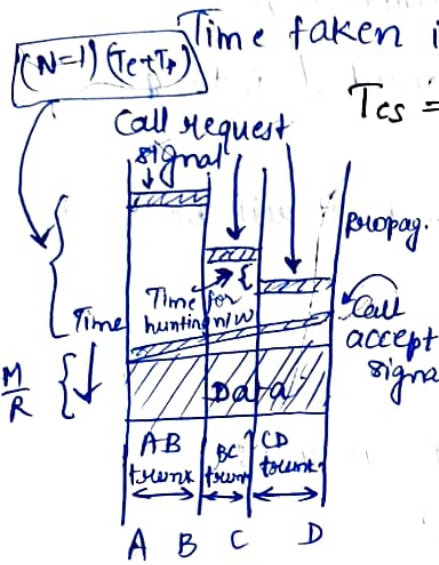


No. of users supported $= N = \frac{B_I}{B_u}$ → Total link capacity; → BW/user

Delay in Circuit Switched Network:

Three phases in circuit switching:

(i) Connection establishment (T_e)
(ii) Data transmission (T_t)
(iii) Connection release (T_r)



Time taken in data transfer, $T_{cs} = T_e + T_t + T_r$

$T_{cs} = (N-1)T_e + M/R + (N-1)T_r + (N-1)T_p \times 4$ (for all requests)

$(N-1)T_p \times 4$ → propagation delay

N: No. of nodes
M: length of data
R: Link BW
T_e: Conn establishment delay
T_r: Conn release delay

Eg. A c.s. connection involves 5 switching nodes. Each node takes about 2 sec and 0.2 sec. to establish and release conn; respectively. If the data transfer rate is 2400 bps, compute the data transfer time and circuit switching overhead for messages that are 300 and 3000 bytes long.

Soln: Neglecting propagation delay,.

300 byte message:

$T_{cs} = 4 \times 2 + (300 \times 8 / 2400) + 4 \times 0.2$
$= 9.8$ sec.

CS overhead $= (T_e + T_r)/T_{cs}$
$= (8 + 0.8)/9.8 = 89.7\%$.

3000 byte message:

$T_{cs} = 4 \times 2 + (3000 \times 8/2400) + 4 \times 0.2$
$= 18.8$ sec.

CS overhead $= 8.8/18.8 = 46.8\%$

Pros and Cons of Circuit Switching

Pros:

- Dedicated resource
- Guaranteed delay
- Guaranteed BW
- Good performance in terms of the quality of service.

Cons:

- BW utilization is not good.
  - If you do not use reserved BW, that will be wasted.

Good.

- for large blocks of data.
- when there is low connection setup time.
- when there is light network load.
- for time sensitive traffic (traffic that cannot be delayed).

Store and Forward Switching

- Switching nodes have the abilities to store user messages and forward as when links are available.
- Each node has a processor and a storage.
- No end-to-end physical connection is set up.
- choices : Message switching
  Packet switching
  Cell switching.

Receive user messages and store → Check messages for error and carry out error recovery → Determine destination address from message → Choose an appropriate destination link towards destination Based on routing mechanism.

Forward the message to the next node on the chosen link. ←

Message switching

↳ Uses data blocks

↳ Each link is shared for various message
(No end-to-end reservation of circuit)

↳ No connection setup and tear down processes.
↳ More efficient compared to CS.

↳ Requires additional control information.

End-to-end delay
$= (N-1)[(L+H)/R]$.

(no queuing delay)

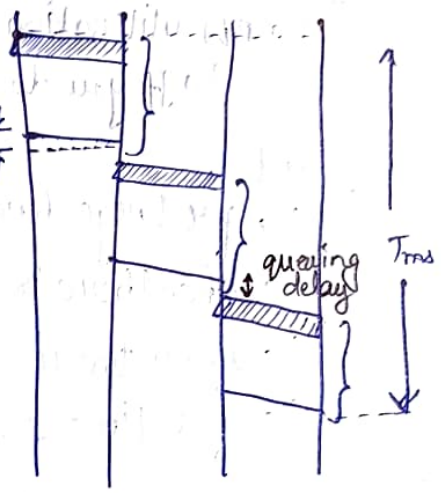
L → message (bits)
H → header (bits)
R → link BW

→ with D_q queuing delay,

$(N-1)[D_q + (L+H)/R]$

→ with p.d. D_p,

$(N-1)[D_p + D_q + (L+H)/R]$



Eg:
- 5 switching nodes
- Data transfer rate = 2400 bps
- Data size = 3000 bytes
- Each message require 50 byte control info.

Estimate time taken for data transfer and overhead for (packet)

(a) message switching.

Soln: $T_{ms} = (N-1) \times 3050 \times \frac{8}{2400} = 40.68$ [Neglecting q.d & p.d]

% MS overhead $= \frac{(N-1) \times 50 \times \frac{8}{2400}}{40.68} = 1.6\%$.

(b) packet switching.

[packet length = 500 bytes, D_q negligible]

Soln: $T_{ps} = (N-1)D_p + [N_p(L_p+H)]/R + (N-2)[D_s + D_{pp}]$

$N_p = 3000/500 = 6$

$\therefore T_{ps} = \overset{\approx 0}{D_q} + 6 \times 8 \times (500+50)/2400 + 3 \times [8 \times (500+50)/2400]$

$= 16.5\,s$

% PS overhead $= \left([(N-2)+N_p] \times 50 \times \frac{8}{2400} \right) / 16.5 = 9\%$

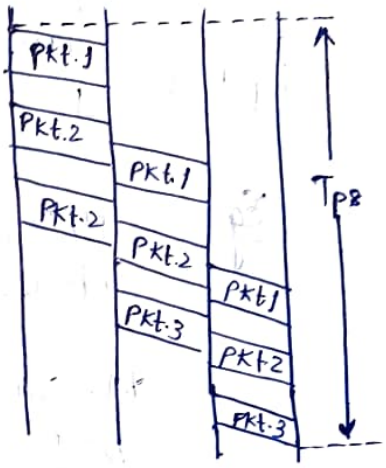
Packet Switching

↳ Uses much smaller data block.

↳ Probability of error is high with long messages.

↳ Efficiency decreases with high error as the entire message has to be retransmitted.

↳ Message switching requires more resources.



Neglecting queing delay,

$$T_{ps} = (N-1)D_p + \frac{[N_p(L_p+H)]}{R} + (N-2)[D_s + D_{pp}],$$

D_p: link p.d.

pkt transmission delay

D_s: S&F delay (includes pkt. txn time)

D_{pp}: packet processing delay

L, L_p: length of pkt., message

N_p: No. of pkts. $= L/L_p$

R: transmission rate of the link

H: overhead of pkt.

N: no. of nodes in path.

② Neglecting D_q & D_{pp},

$$\frac{L(L_p+H)}{RL_p} + (N-2)\frac{[L_p+H]}{R}, \text{ where } D_s = \frac{(L_p+H)}{R}, \frac{L}{L_p} = N_p.$$

$$= \frac{LH}{RL_p} + \frac{(N-2)L_p}{R}$$

$$\frac{d}{dL_p} \Rightarrow \left(\frac{LH}{R}\right)\left(-\frac{1}{L_p^2}\right) + \frac{(N-2)}{R} = 0$$

$$\Rightarrow L_p^2 = LH/(N-2)$$

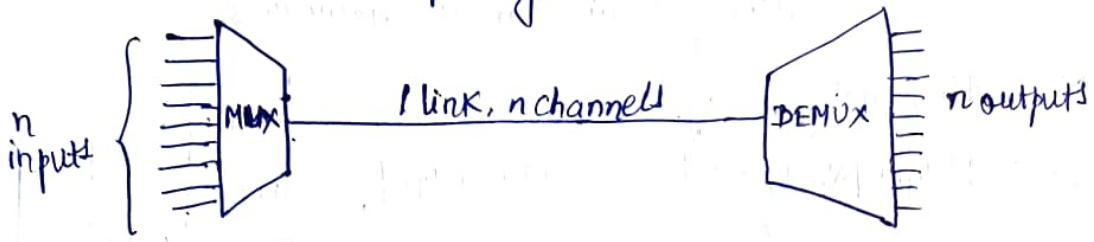
$$\Rightarrow \boxed{L_p = \sqrt{\frac{LH}{N-2}}}$$

Packet switching : statistical Multiplexing

↳ 95% of the n/w is not used (remain silent), due to reserved BW.

↳ B/W shared on demand.

↳ TDM: Each host gets same slot in revolving TDM frame. (Time Division Multiplexing)



Eg: 1 Mb/s link
Each user : 100 kp/s when 'active'
active 10% of time

Ckt. Switching:

$N = \frac{B_T}{B_U} = \frac{1\,Mbps}{100\,kpps} = 10$ users

Pkt. switching:

With 35 users, probability > 10 active at same time is less than 0.004

$$\approx 1 - \sum_{n=0}^{10} {}^{35}C_n \cdot p^n (1-p)^{35-n}$$

↳ Great for bursty data.
↳ Better res. sharing.
↳ Better link utilization
↳ Simpler, no call setup.
↳ Excessive congestion: packet delay & loss.
↳ Protocol needed for reliable data transfer, congestion control.

Cell switching over virtual circuits

↳ call setup, teardown for each call before data can flow
↳ Each packet carries virtual channel (VC) identifier (not destination host address).
↳ Each router on source-dest. path maintains 'state' for each passing connection.
↳ link, router resources (BW, buffers) may be allocated to VC (dedicated resources = predictable service).

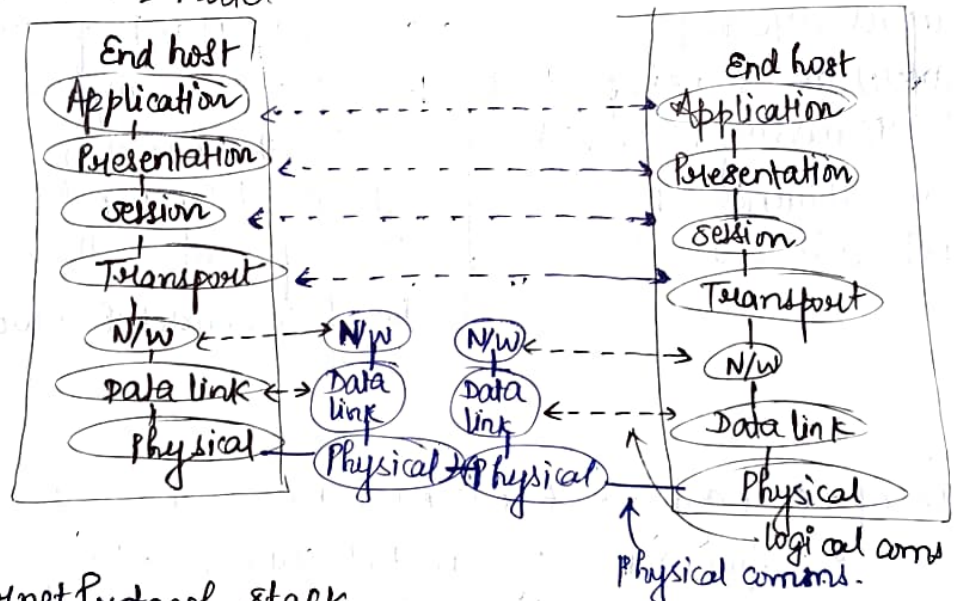
12-02-2025

Internet Protocol "Layers"

Layer N+1
Layer N
Layer N-1

↳ Layered protocol stack
↳ Dividing n/w into layers.
Let each layer provide a set of functions to the layer above it.
↳ Dealing with complex systems, explicit structure allows identification, relationship of complex system's pieces.
↳ Layer reference model for n/ws.
↳ Modularization eases maintenance, updating of system.
↳ Change of implementation of layer's service transparent to rest of system.

→ ISO-OSI Model

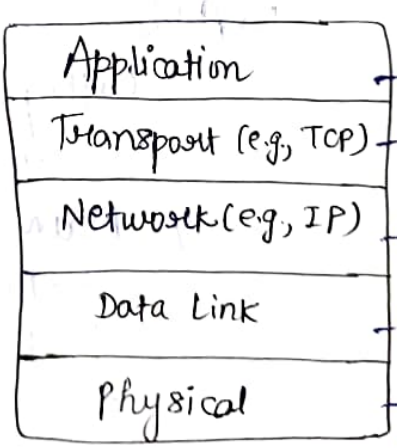


Internet Protocol stack

Application: Supporting n/w applications
↳ FTP, SMTP, HTTP
↳ Simple Mail Transfer Protocol

Transport: Process-process data transfer
↳ TCP (Transmission control Protocol)
↳ UDP (User Datagram Protocol)

Network: Routing of diagrams from source to destination.
↳ IP, routing protocols



[Stack of layers (Protocol stack)]

Data link : Data transfer b/w neighbouring n/w elements.
↳ PPP, Ethernet.

Physical : Bits "on the wire".

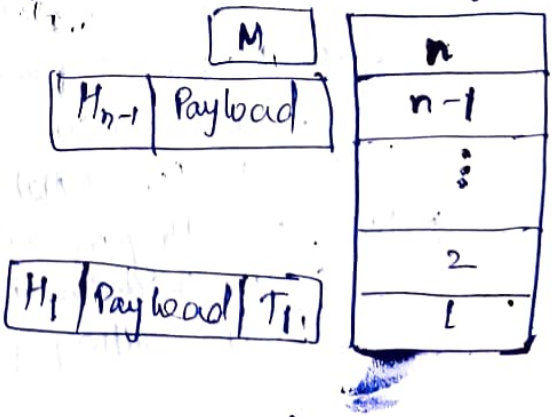
ISO-OSI	TCP/IP (Internet Protocol Stack)
• A ref. model than a successful architecture. • Model is defined before any prototypes existed. • No working systems existed while modeling. • Eg: X.25 • Some layers are not essential (session layer) and some important functions are missing (security). • Design is influenced by administrative bodies.	• A successful architecture. • A model is retrofitted to the popular TCP/IP protocol suite. • No real model was intended in the original software version, but it was later split b/w TCP & IP, closer to model. Eg. Internet • Some important functions are not defined (eg, security) and constant tweaking is done. • Influenced by popular Internet Tech. Development Culture. (Rough consensus & a working code).

Encapsulation

↳ Process of • packaging higher layer data by a lower layer.
• confinement of a packet within a larger packet.
• containing payload data in a packet.

↳ Adds certain control information to the data by the lower layer.
↳ Helps proper processing of packets at the lower layer and its peer layer.

↳ 2/3 Main parts :
- Header (control info.)
- Payload (data)
- Trailer (control info.)



Network Software Reality → Not strictly layer-wise. → Implemented as set of functions in the OS kernel.

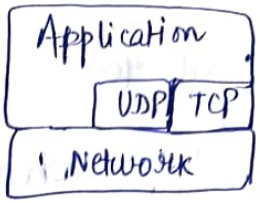
HTTP — Application } Application software

TCP, UDP — Transport } App. soft. } Software (Kernel modules)

IP — Network

802.11, 802.3 — Datalink

DSSS/OFDM, Ethernet — PHY } Hardware



Application layer Protocols

drives → HTTP (HyperText Transfer Protocol)

→ WWW : collection of cooperating servers and clients speaking HTTP.

HTTP

- Text-based protocol
  - Readable text commands
- Defines how the communication b/w a web browser and web-server can take place.
- Stateless (but new methods are added on to gather state of user information, e.g., cookies).
- Two kinds of messages used:
  - REQUEST
  - RESPONSE
- General message form:

```
START_LINE <CRLF>
MESSAGE_HEADER<CRLF>
<CRLF>
MESSAGE-BODY<CRLF>
```

- A webpage contains one or more objects: html, JPEG files, etc.
  - Located by URLs (for requesting) (Uniform/Universal Resource Locators)

HTTP session:

HTTP client initiates TCP connection to HTTP server (process) at host (www.iist.ac.in on port 80)

↓

HTTP server at host waiting for TCP connection at port 80, accepts connection, notifying client.

↓

HTTP client sends HTTP request message (URL) into TCP connection socket.

↓

↓

Server receives request, forms response message and send into its TCP socket.

↓

Server closes TCP connection.

↓

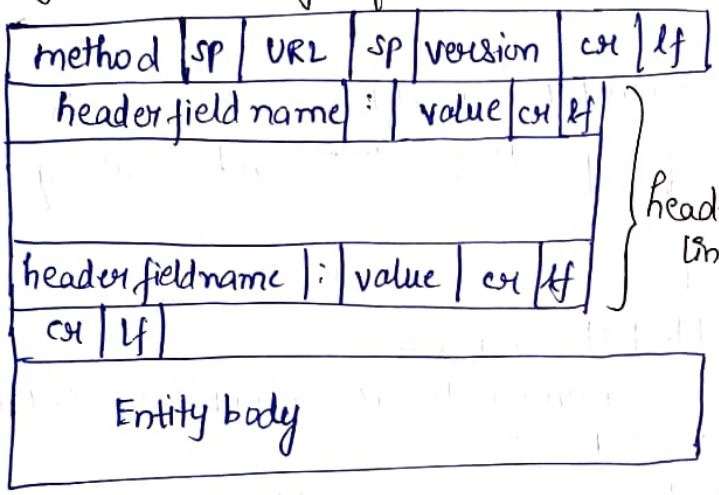
HTTP client receives response.

↓

Repeat for each object or other scripts.

19-02-2025

HTTP general message format:



HTTP Request operations

GET: Fetches a specific web object

HEAD: Fetch the status of the specific web object (eg, timestamp).

POST: Send information to the server (e.g, filled forms).

PUT: Store documents at the specific web URL (for uploading).

DELETE: Delete a specific URL (with sufficient permission).

TRACE: Loop back the message (for debugging, security verification).

CONNECT: ~~Connect~~ For use by proxy servers or for future purpose.

OPTIONS: Query certain options available with servers.

Response messages

CODE	Type	Examples
1xx	Informational	100 = server agrees to handle client's request
2xx	Success	200 = Request successful, 204 = no content present
3xx	Redirectional	301 = page moved (see location:), 304 = cached page still valid
4xx	client error	400 = Bad request, 403 = forbidden pages, 404 = page not found
5xx	server error	500 = Internal server error, 503 = try again later, 505 = HTTP version not supported.

Non-persistent HTTP

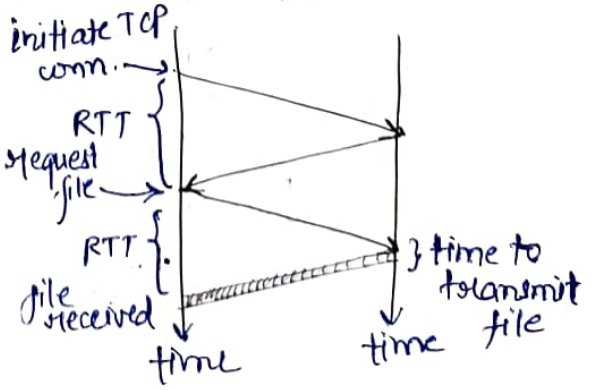
- → Uses separate TCP connections for each object.
- → High comms. overhead: too many control packets sent.
- → High computing resource overhead: too many sockets created & closed.
- → Increased response time (wait for a new conn. setup for every object).
- → Low scalibility (high server load).
- → Comm. b/w client & server can last longer, thus leading to large no of sockets throughout entire server (e.g. web based email).

Persistent HTTP

- → Allows a single TCP connection to persist over multiple object transfers.
- → Multiple request/response sessions can happen over the same socket; less comms. overhead.
- → Lower computing overhead: only fewer sockets to be maintained.
- → Quick response as socket is ready for communication.
- → High scalibility (low server load).
- → Socket can be a bottleneck? (Multiple sockets helpful in parallelizing comm.)
- → session timeout necessary.

RTT: Time for a small packet to travel from client to server and back. [Round Trip Time]

Response time:
Total = 2 RTT + transmit time



HTTP Proxy Servers

Web caching

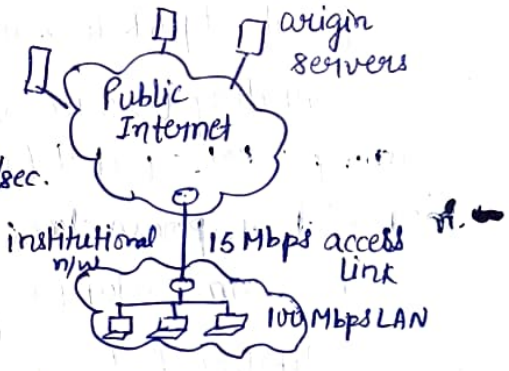
- ↳ Web cache acts as both client and server.
- ↳ Typically installed by ISP (University, company, residential ISP).
- ↳ Large array of web cache is deployed for content distribution networks.
- ↳ ~~Reduce~~ Reduce response time for client request. (reduce traffic).
- ↳ ~~Internet~~ dense with caches: Enables 'poor' content providers to effectively deliver content.

Browser ⟶ HTTP request ⟶ Proxy Server ⟶ HTTP Request ⟶ server
Browser ⟵ HTTP response ⟵ Proxy Server ⟵ HTTP response ⟵ server

Eg. Average object size = 1 M bits

Av. request rate from institution's browser to origin servers = 15/sec.

Delay from inst. router to any origin server & back to router = 2s.



∴ Utilization of LAN = $I = \frac{a \times L}{R} = \frac{(15/\text{sec})(1\ \text{Mbit})}{100\ \text{Mbps}} = 15\% \times 0.6$ ↳ after cache

Utilization on access link = $I = \frac{a \times L}{R} = \frac{(15/s)(1\ \text{M bits})}{15\ \text{Mbps}} = 100\% \times 0.6$

Total delay = Internet delay + access delay + LAN delay
= 2 sec + several minutes + m seconds.

↓ solution.

Increase BW of access link to 100 Mbps
↳ LAN utiliz. = 15 %.
A.L. utiliz. = 15%.
↳ costly upgrade

↓ solution

Cache (suppose hit rate = 0.4) → 40% requests satisfied immediately
↳ 60% by origin server.

Conditional GET:

↳ Don't send object if cache has up-to-date cached version.

Cache: Specify date of cached copy in HTTP request.

If-modified-since: <date>

Server: Response contains no object if cached copy is up-to-date.

HTTP/1.0 304 Not Modified.

State Maintenance in HTTP sessions

↳ HTTP: stateless protocol.

↳ Modified to do some key functions for session layer.

- ↳ To better serve connections & users.
- ↳ To provide differentiated services to users.
- ↳ To provide subscription services.
- ↳ To ensure security.

↳ Two main methods:

- Protocol endpoints: Maintain state at sender/receiver.
- Cookies: Carried in http messages can contain session state.

Protocol Endpoints:

↳ TCP connection state (within a single session)

↳ IP address (across multiple sessions).

Cookies

↳ State information maintained by storing small text files.

↳ Server can request to store text file or info. in the client's machine's harddisk.

↳ For subsequent visits, the client web browser transmits the information back to the server.

↳ ~~Four~~ steps/locations for cookies:

- Cookie header line of HTTP response message.
- Cookie header line in HTTP ~~response~~ request message.
- Cookie file kept on user's host, managed by user's browser.
- Backend database at the server website.

↳ Size limit: 4K (255 characters)

↳ Privacy concern: tracking user, stored without permission, etc.

Non-persistent (Session) Cookies	Persistent cookies
→ Valid only for a session → Expire either on session logout, session timeout or on browser closure. → Useful for login credentials. → To ensure user privacy. → Cookie stealing can be avoided.	→ Last forever (until erased or expired). → Usually set for many years. → User preferences, language, profile presentation, etc., can be personalized without loggin in. → Can track a user ↳ during his entire life. ↳ for his behavioral patterns.

20-02-2025

→ Local shared objects (LSO)

↳ Cookies stored by flash-plugins and other softwares.

↳ Much larger in size (100 KB compared to 4KB)

↳ Can remain in machine forever.

↳ Can carry pretty much any information about you/machine.

↳ not managed by the browsers.

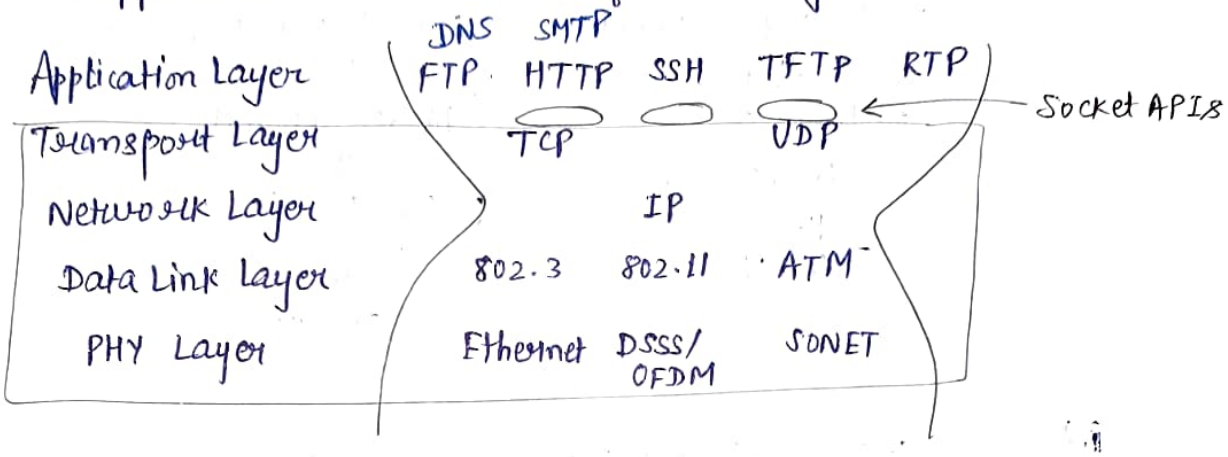
#Buttons

#Invisible objects!

Network Software APIs (Socket APIs)

↳ Usually part of the OS.

↳ Applications use N/W APIs for accessing n/w services.



End-to-end data transfer sessions

↳ Connection-oriented

↳ Connection-less

Connection oriented data transfer sessions:

↳ sets up a connection b/w the end points.

↳ Maintains data structures, variables, memory and other resources at both sender and receiver.

↳ Clears the connection after transfer.

Connection-less data transfer sessions:

↳ Sends and receives data without explicit connection setup.

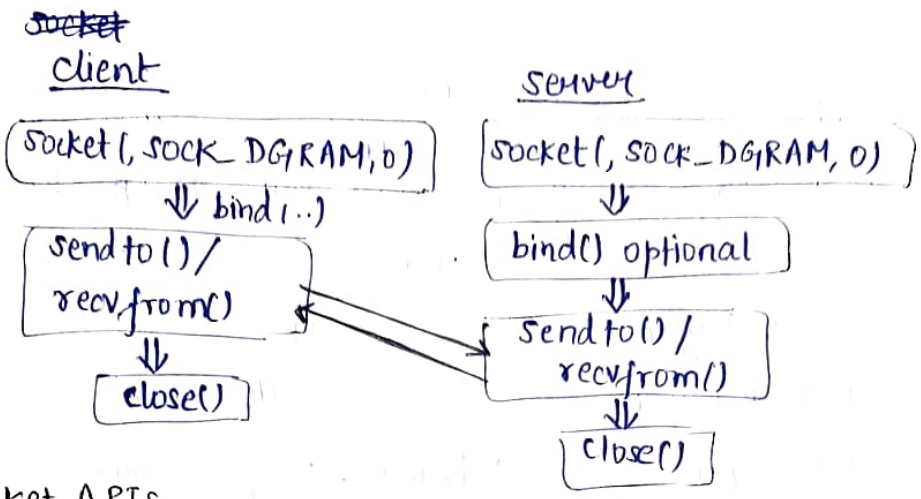
↳ No connection related state variables maintained at sender/receiver.

↳ Much lighter in implementation complexity than connection-oriented protocols.

↳ Some connection end-point variables need to be maintained, eg, socket, common buffers, etc.

Eg. User Datagram Protocol (UDP).

Connection-less transport session:



Main socket APIs

```
int socket (int domain, int type, int protocol)
int bind (int socketfd, struct socketaddr* addr, int addr_len)
int listen (int socketfd, int backlog)
int accept (int socket, struct socketaddr* addr, int addr_len)
int connect (int socket, struct socketaddr *addr, int addr_len)
int send (int socket, char* message, int msg_len, int flags)
int recv (int socket, char* buffer, int buf_len, int flags)
int select (int n, fd_set, *readfds, fd_set * writefds,
            fd_set * exceptfds, struct timeval * timeout)
int close (int socket)
```

int socket (int domain, int type, int protocol)

domain: Specifies comm. family domain
- ↳ helps to have single socket() api for a no of protocol families.
- ↳ selects the protocol family which will be used for comm, also called address family (AF_xxxx in unix systems).
  - ↳ PF_INET for Internet IPV4; PF_INET6 for IPV6.
  - ↳ PF_UNIX/PF_LOCAL for local comm. using Unix pipes.
  - ↳ PF_PACKET for direct n/w access.
    - ↳ Packet sockets are used to receive or send raw packets at the device driver (OSI Layer 2) level.
    - ↳ allow user to implement protocol modules in user space on top of physical layer.

protocol: defines protocol used for comm.
↳ 0 or UNSPEC for a single protocol.
↳ usually unused; defined by domain and type parameters.
Eg PF_INET and PF_STREAM implies the use of TCP.
PF_INET and SOCK_DGRAM defines the use of UDP.
↳ Many protocols may exist → a particular protocol must be specified.

Return value:
↳ on success, a file descriptor for new socket is returned.
↳ on error, -1 is returned, and errno is set appropriately.

type: (usually defined in <sys/socket.h> file)
↳ defines type of comm., end-to-end comm. ~~types~~ semantics.
↳ SOCK_STREAM → provides sequenced, reliable, two-way, conn.-based byte streams (usually for TCP-like).
↳ SOCK_DGRAM → supports datagrams (connectionless, unreliable messages of a fixed max^m length) (usually for UDP like protocols).
↳ SOCK_RAW → provides raw n/w protocol access (used in association with PF_RAW (or old PF_PACKET) protocol family domains.
↳ SOCK_RDM → provides reliable datagram layer that does not guarantee ordering.

→ send(), sendto(), and sendmsg()
↳ To transmit a message to another socket.
↳ send(): used only when socket is in a connected state.

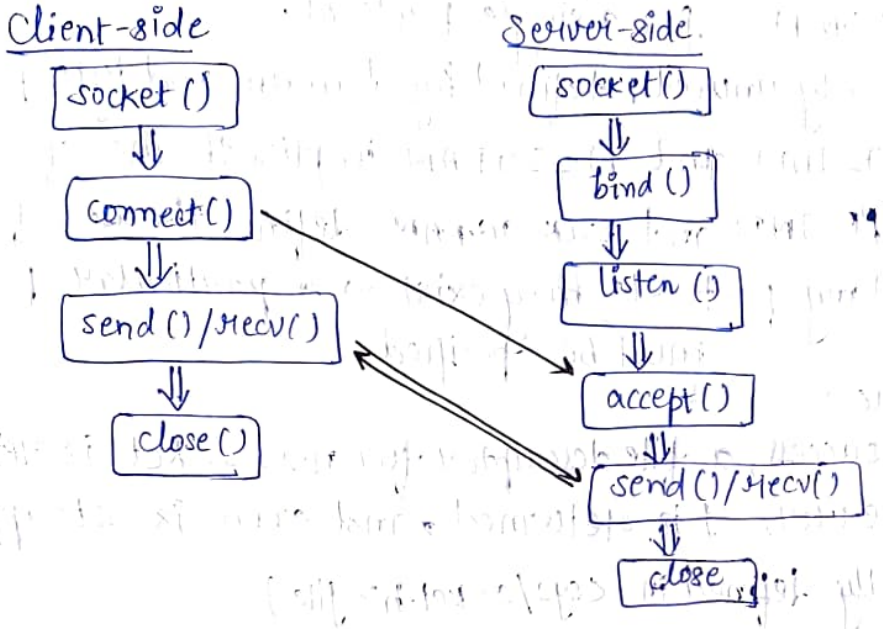
ssize_t send (int socket_fd, const void* buf, size_t len, int flags);
socket_fd: socket on which send is to be carried out
buf: carries data to be sent
len: length of message
flags: special control signals (eg. bitwise OR of MSG_DONTROUTE, MSG_MORE, MSG_OOB msgs.)

↳ Returns EAGAIN in non-blocking mode.
↳ Return values: No. of characters sent, on success.
↳ -1 & errno is set, on error.

Preferred for UDP like

ssize_t sendto (int socketfd, const void * buf, size_t len, int flags, const struct sockaddr * to, socklen_t tolen);
ssize_t sendmsg (int socketfd, const struct msghdr *msg, int flags);

Connection-oriented (TCP) session



Bit

↳ A single information digit: 0/1

↳ It has many dimensions:

↳ Electrical → signal, Energy. } covered by Digital communications

↳ Temporal

↳ Physical

Electrical (Signal) Dimension:

↳ Amplitude, frequency, phase of the signal represented by a bit.

Energy dimension:

↳ Electrical power related representation of a bit.

↳ Power in dBm used for transmission.

↳ Contributes to Signal to Noise Ratio (SNR).

Temporal dimension:

↳ Time duration of a bit.

R: bits per second (bps) capacity of a channel

Duration of a bit $= 1/R$.

Transmission time required for an L-bit packet $= \boxed{\frac{L}{R}}$ seconds.

Physical dimension:

↳ Length of a bit

Channel ⟶ R bits/sec.

Propagation speed or velocity of signal over a medium = V

$V = 3 \times 10^8$ m/s for radio/optical fiber.

$V = 2 \times 10^8$ m/s for typical copper cables.

Data Rate = R bits/sec.

R = 1 Mbps (for IEEE 802.11b long range links)

∴ Length of a bit $= \boxed{V/R}$

Eg $V/R = \frac{3 \times 10^8 \text{ m/s}}{1 \text{ Mbps}} = 300$ m/b → a single bit is 300 meters long!

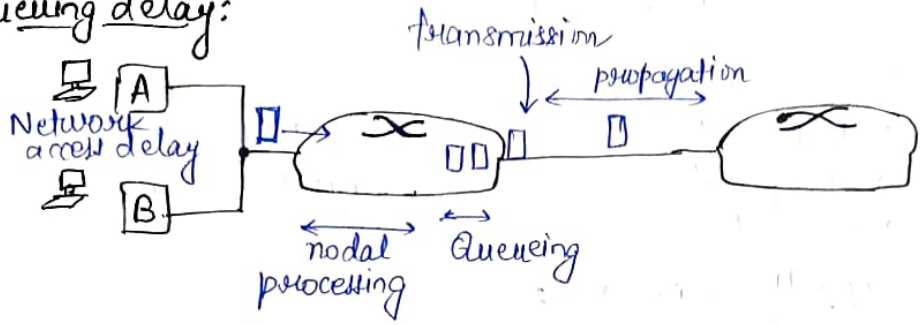
→ Multiple bits are combined to make a symbol.

Symbol length $= \frac{V}{R/K}$, K: no. of bits/symbol

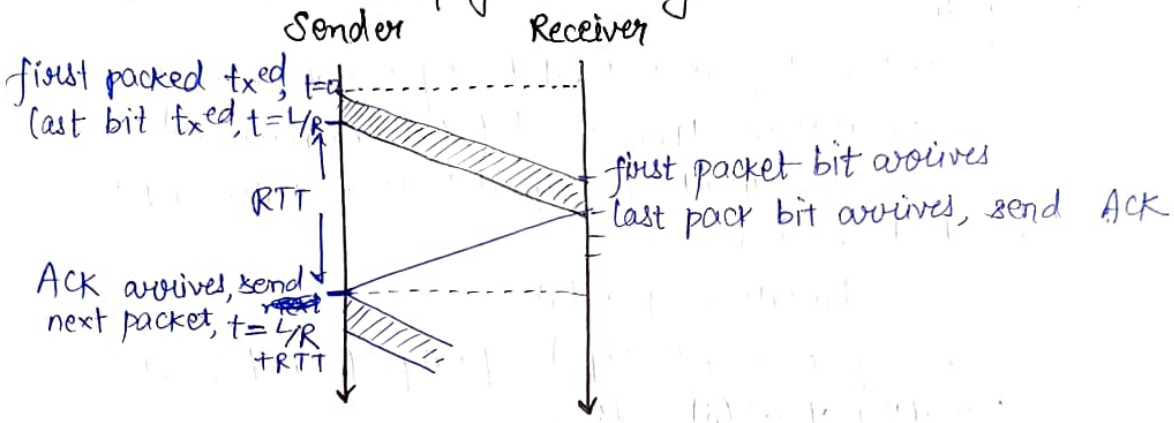
Symbol rate $= R/K$ bps.

Sources of Delay at a router

Queuing delay:

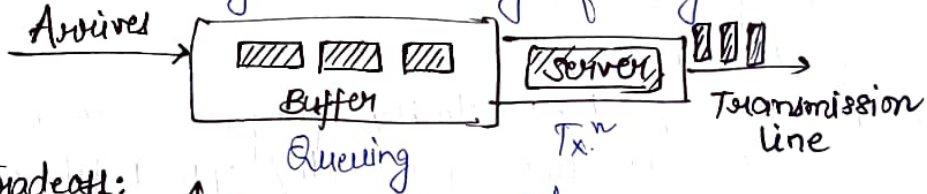


Transmission and propagation delay

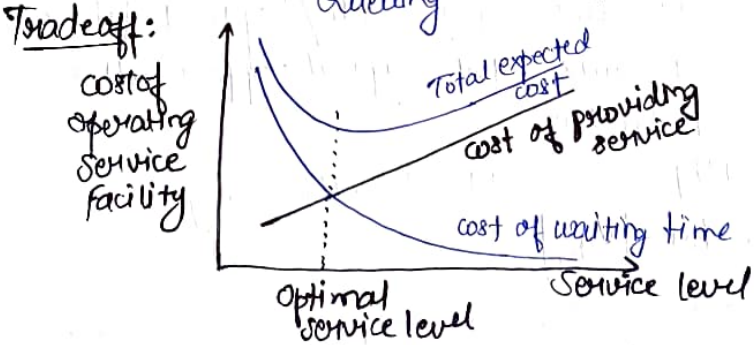


Queuing Theory

↳ Helps :- Finding the best level of service
- Analytical modelling of delay.



Tradeoff:



Main characteristics of a Queuing System:

↳ Arrival characteristics
↳ Queuing discipline
↳ Service characteristics

Arrival Characteristics

→ Arrival distribution : Random or fixed
  ↳ characterised by measuring time b/w consecutive arrivals, or arrival rate.
  ↳ Poisson distribution is often used for random arrivals.
→ Arrival population : Either infinite or limited.

Random Arrival characteristics :

↳ Poisson distribution.
  ↳ Known mean / expected arrival rate.

λ : average arrival rate per time unit.

$P(x)$: probability of exactly x arrivals occuring during one time period.

$$P(x) = \frac{e^{-\lambda} \cdot \lambda^x}{x!}$$

→ Arrival rate (# arrivals) in each discrete time period is independent.
→ Arrival rate → 0 as time interval → 0.

Queuing Discipline

→ Queuing Service discipline : method of serving
  ↳ Usually FIFO (First In First Out)
  ↳ Random, priority-based, etc.
→ Length of Queue : No. of items in the queue.
  ↳ Max possible queue length
    ↳ either limited or unlimited (practically limited).

Service Characteristics

→ No. of servers.
→ Distribution of service at the server
  ↳ Behaviour of time taken for serving one arrival.
    ↳ Fixed or Random (exponential dist.) service time.

μ : average service time
t : length of service time ($t \geq 0$)
$P(t)$: probability of service time greater than t.

$$P(t) = e^{-\mu t}.$$

Kendall's Queuing Notation : A/S/c/B/N/D

A : Arrival distribution.
- ↳ M : Poisson
- ↳ D : deterministic
- ↳ G : general

S : Service time distribution
- ↳ M : exponential
- ↳ D : deterministic
- ↳ G : general

C : No. of servers (1 or more)

B : No. of buffers / seats
- ↳ Finite or infinite

N : calling population
- ↳ Finite or infinite

D : Service discipline
- ↳ FCFS, Random, Priority or other disciplines.

Eg ① Simple system (M/M/1) → Poisson arrival, exp. service, one server, infinite buffer size, ∞ population, FCFS service → Eg Customer service desk in a store
M/M/1/∞/∞/FCFS

② Multiple server (M/M/s) → s servers → Eg Call center with multiple (s) ~~servers~~ executives
M/M/S/∞/∞/FCFS

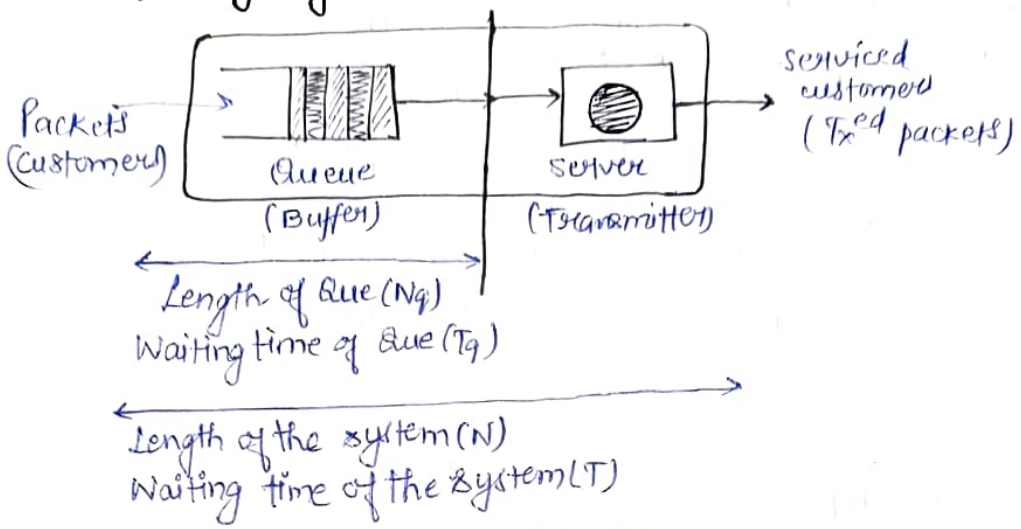
③ Constant service M/D/1 → fixed time service → Eg Automated car wash (every car takes a fixed time)

④ General service M/G/1 → General time service → Eg Electronic repair shop

⑤ Limited population M/M/s/∞/N → Limited servers, limited ~~buffer~~ population, ∞ buffer → Eg. A telephone exchange with only limited subscribers.

Analysis of Delay at a router

M/M/1 queuing system :

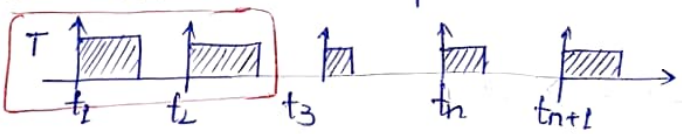


λ : Mean packet arrival rate (No. of packets per second)

- Arrival rate is Poisson distributed (with mean λ)

$$P(n) = \frac{e^{-\lambda T}(\lambda T)^n}{n!}, \quad n = 0, 1, \ldots$$

- Inter-arrival time is exponential (Markovian)



$\Delta t_1 = t_2 - t_1$

$\Delta t_n = t_{n+1} - t_n$

$$P(\Delta t_n < T) = 1 - e^{-\lambda T}, \quad T \geq 0$$

$$p(\Delta t_n) = \lambda e^{-\lambda \Delta t_n}, \quad \Delta t_n \geq 0.$$

μ : mean service ~~time~~ rate (Packets tx^ed per unit time) [Packet tx rate] $[\mu = R/L]$

- service time is exponentially distributed with parameter μ.
  - Depends on the length of the packet and the tx rate of link
  - Tx time of a packet = L/R

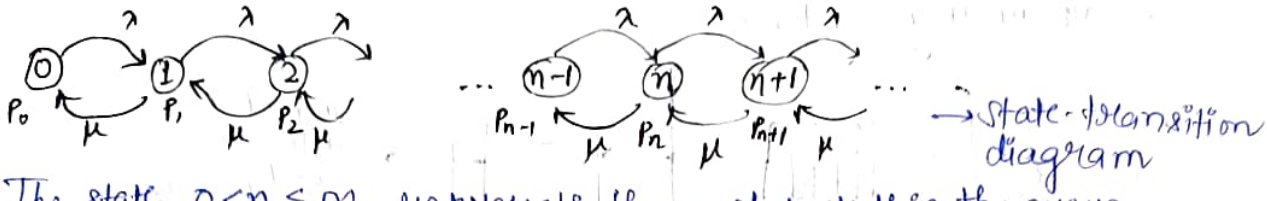
T_n : Transmission time of n^{th} packet

$$P(T_n \leq T) = 1 - e^{-\mu T}, \quad T \geq 0$$

Eg. $L = 1000$ bits, $R = 1$ Mbps

$L/R = 1$ ms, $\mu = R/L = 1000$ pps.

M/M/1 queue: steady state situation



→ State-transition diagram

The state, $0 \le n \le \infty$, represents the no. of packets in the queue.

- $n=0$: empty system (no waiting)
- $n=1$: one packet at present (being served)
- Any incoming packet will have to wait till the current packet is tx^{ed}.

In steady state,

$$\lambda P_0 = \mu P_1$$

$$\Rightarrow P_1 = f \cdot P_0 \quad , \quad f = \frac{\lambda}{\mu} ,$$

$$P_2 = f \cdot P_1$$

$$\Rightarrow P_2 = f^2 \cdot P_0 ,$$

$$P_3 = f^3 \cdot P_0$$

$$\vdots$$

$$\boxed{P_n = f^n \cdot P_0}$$

$$\Rightarrow \sum_{n=0}^{\infty} P_n = 1 = \sum_{n=0}^{\infty} f^n P_0$$

$$\frac{P_0}{1-f} = 1 \Rightarrow \boxed{P_0 = 1-f}, \quad f < 1.$$

Number of Packets in the System:

N: Mean no. of packets

↳ Helps in the design of buffer length.

$$N = E(n) = \sum_{n=0}^{\infty} n P_n = \sum_{n=0}^{\infty} n P_0 \cdot f^n$$

$$= \sum_{n=0}^{\infty} n (1-f) f^n = \boxed{\frac{f}{1-f}}, \text{ where } f = \frac{\lambda}{\mu}$$

$$\left[\text{as } \sum_{n=0}^{\infty} n f^n = \frac{f}{(1-f)^2}, \ f < 1 \right]$$

Packet Delay at a Router:

T: Waiting time for a packet at a router (Que + service)

Little's theorem: $T = \frac{N}{\lambda}$

$$= \frac{\rho}{(1-\rho)\lambda} = \frac{\lambda/\mu}{(1-\frac{\lambda}{\mu})\lambda} = \frac{\lambda}{(\mu-\lambda)\lambda} = \boxed{\frac{1}{\mu-\lambda}}$$

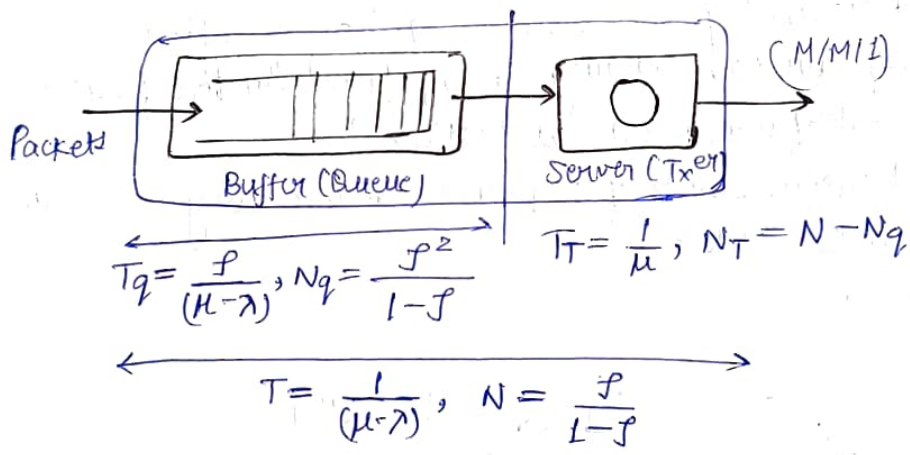
Packet Delay at the ~~Ro~~ Queue:

Waiting time for a packet at a router (Queue),

$$T_q = T - \frac{1}{\mu} = \frac{1}{(\mu-\lambda)} - \frac{1}{\mu} = \boxed{\frac{\rho}{\mu-\lambda}}$$

No. of packets in the queue,

$$N_q = \lambda T_q = \frac{\lambda \rho}{(\mu-\lambda)} = \boxed{\frac{\rho^2}{1-\rho}}$$



Service Rate in a Router:

Assume arrival rate, $\lambda = a$ pkts/sec.

Tx rate $= R$

Packet length $= L$ bits

service rate $= \mu = \frac{1}{\text{pkt. tx time}}$

$$= \frac{1}{L/R} = \frac{R}{L}$$

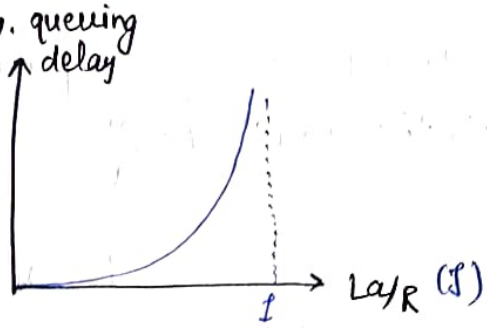
$$\rho = \frac{\lambda}{\mu} = \frac{aL}{R} = \frac{La}{R}$$

↳ Traffic intensity / utility factor / stability factor.

Queuing Delay:

R: Link BW (bps)
L: packet length (bits)
a: avg. packet arrival rate

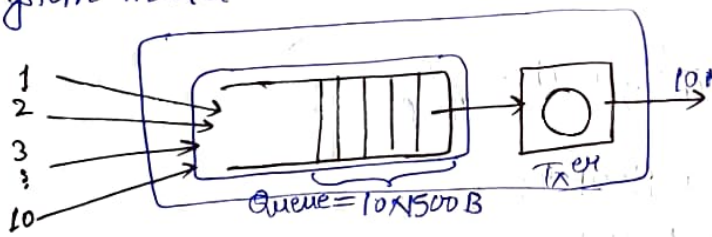
traffic intensity, utilization $= La/R$



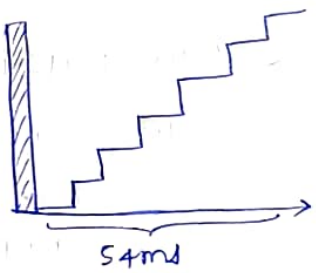
$La/R \sim 0$: avg. queuing delay small
$La/R \to 1$: delay become large
$La/R > 1$: more 'work' arriving than can be serviced, avg. delay infinite!

Eg. In an internet router with 10 input lines and one output line, assume each of ~~them~~ ten i/p lines simultaneously receive 1500 byte packet and place them in op line's buffer.
What is the ~~o/p~~ average queuing delay faced by the packets?
Assume all interfaces operate at 10 Mbps BW.
If such simultaneous bursts of 10 packets arrive every 100 milli seconds, will the system be stable?

Soln: System model

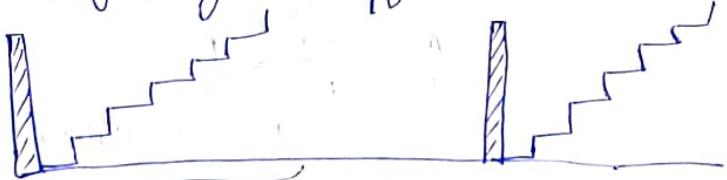


packet 1: 0 delay
packet 2: $L/R = \frac{1500 \times 8}{10\ \text{Mbps}} = 1.2$ ms



$$\text{Total delay} = \sum_{i=1}^{9} i \times (L/R) = 45\left(\frac{L}{R}\right) = 45 \times 1.2\text{ms} = 54\text{ms}$$

$$\text{Avg. delay} = \frac{54\text{ ms}}{10} = 5.4\text{ ms}$$



Time taken to finish txn $= 10 \times 1.2\text{ms} = 12\text{ms}$,
Period of repetition $= 100\text{ms}$ $\Rightarrow$ System is stable.

Instability scenario:

Eg. Consider a router with two 10 Mbps i/p lines having Poisson distributed arrival of packets with an avg. 150 packets per second. The router's o/p line is 1 Mbps. The packet length is exponentially distributed with mean 2500 bits.

Estimate the following:

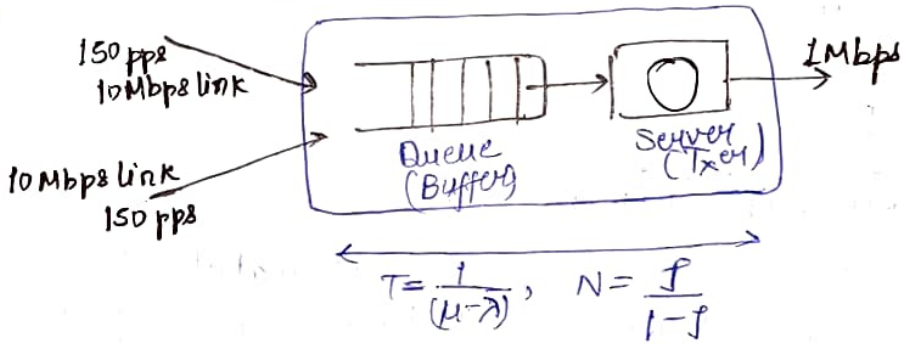
(i) what is the traffic intensity faced by the queuing system at the o/p link?

(ii) Is the system stable?

(iii) What is the avg. no. of packets in the system?

(iv) What is the avg. waiting time of packets?

Soln:



Mean packet length = 2500 bits

(i) $f = \lambda/\mu = \frac{(\lambda_1 + \lambda_2)}{\mu} = \frac{(150 + 150)}{(R/L)} = \frac{300}{(10^6/2500)} = \frac{300}{400} = 0.75$

(ii) Yes, $f < 1$, but moderately loaded.

(iii) $N = \frac{f}{1-f} = \frac{0.75}{0.25} = 3$

(iv) Avg. waiting time, $T = \frac{1}{(\mu - \lambda)} = \frac{1}{400 - 300} = 10$ ms

Impact of Higher Rates:

M/M/1 queue:

Arrival rate : λ increased to $K\lambda$, $K > 1$

Packet length distribution remains the same.

$\Rightarrow$ Txn rate is increased by the factor (same) $\rightarrow$ R to KR.

$\Rightarrow$ Avg. pkt. txn. time : L/R to $\frac{L}{KR}$, $1/\mu$ to $\frac{1}{K\mu}$

$\Rightarrow$ Ag. pkt. departure rate : $\frac{R}{L}$ to $\frac{KR}{L}$

μ to $K\mu$

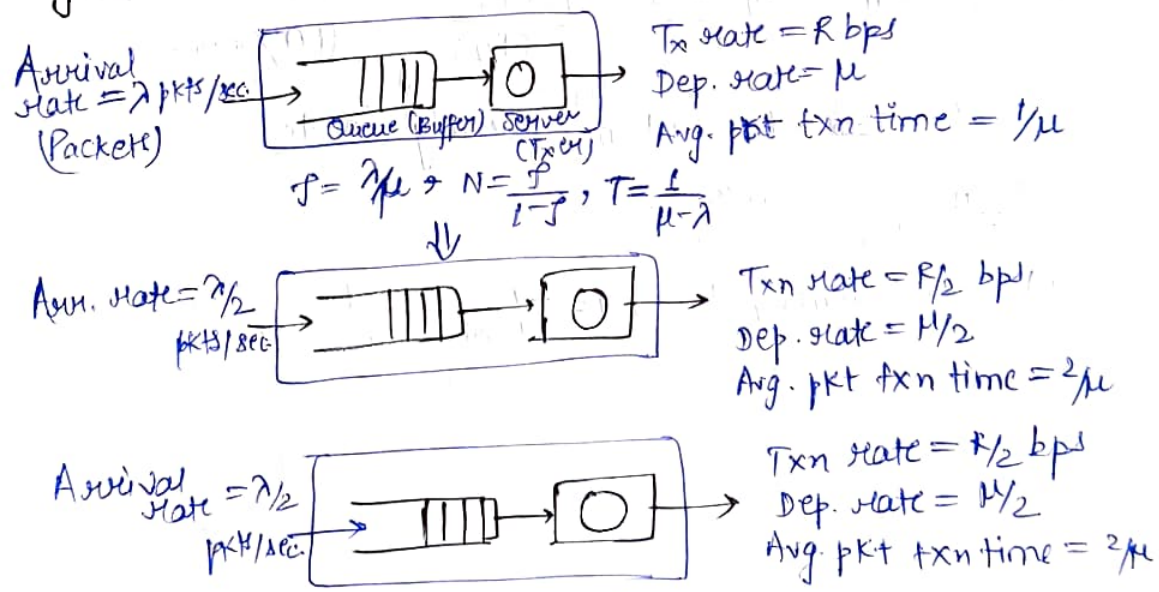
$\Rightarrow \rho = \frac{K\lambda}{K\mu} = \frac{\lambda}{\mu}$,

$N = \frac{\rho}{1-\rho}$,

$T = \frac{1}{(K\mu - K\lambda)} = \frac{1}{K(\mu-\lambda)}$

$\rightarrow$ A txn line K time as fast will accomodate K times as many packets/sec. at K times lower avg. delay per pkt.

Splitting B/W into smaller channels:



$\rho = \frac{\lambda/2}{\mu/2} = \frac{\lambda}{\mu}$, $N = \frac{\rho}{1-\rho}$

$T = \frac{1}{(\mu/2 - \lambda/2)} = \frac{2}{(\mu-\lambda)}$

Domain Name System

Steps in downloading a webpage:

① Extract hostname from URL
[https://www.google.com/index.html to www.google.com]

② Use DNS to translate www.google.com to IP address.
↳ Used for internet routing (e.g., 66.102.7.104).

③ Internet routers determine efficient path to 66.102.7.104.

④ Establish a TCP (socket) connection to the IP address.
↳ TCP handles n/w problems (drops, corruption, etc.).
↳ TCP layered on top of IP/Ethernet.

⑤ Protocol agreement for browser and server to speak HTTP.

⑥ Request using HTTP GET.

⑦ Receive the DATA.

Some Address types and their context:

→ Domain name (eg, www.google.com)
↳ Global, human readable
↳ Application layer

→ Transport layer
↳ Port Number (with or without IP address)

→ IP Address (eg. 66.102.7.104)
↳ Global, works across all networks.
↳ Network layer

→ Ethernet (eg, 08-00-2b-18-bc-65)
↳ local, works on a particular n/w.
↳ Link layer.

Application layer services : DNS

↳ Provides many services including name-to-address translation.
↳ IP addresses are required for Internet routers.
↳ Names can be used for routing, but with high inefficiency.

In original ARPANET,
↳ hosts.txt file contained host-to-address translation.
↳ A central server maintained and managed the changes in hosts.txt file.
↳ Hosts downloaded the hosts.txt file periodically.

DNS

↳ An important application layer protocol.

↳ Services:

→ Name to address resolution: Host name to IP address translation.

→ Name aliasing: Host name aliasing

→ Service aliasing: Mail services aliasing

→ Information services.

→ Performance optimization: Traffic load distribution, improving response time and improving scalibility.

→ Network security

① Name to Address Resolution:

↳ Translation from human-readable name to router-readable address.

eg Name server Lookup

```
# nslookup
> docs.google.com  ⟶ Domain Name
Server:  132.239.0.252
Address:  132.239.0.252 #53

Name : docs.google.com
Address : 132.239.51.6  ⟶ IP address
>
> server
Default server : 132.239.0.252  ⟶ Primary local DNS server
Address : 132.239.0.252 #53
Default server : 128.54.16.2  ⟶ Secondary local DNS server
Address : 128.54.16.2 #53
```

→ New alternative to nslookup: Domain Information Groper (dig) or host

② Host Name Aliasing:

↳ Translation from complex canonical names to simple mnemonic names.

eg cluster0028.asia.microsoft.com
Relay8951.cluster392.utopia.com

Non-authoritative answer:
docs.google.com canonical name = writely.l.google.com
↓ Mnemonic domain name
↓ Canonical domain name

③ Service Aliasing:
↳ Eg. mail service aliasing.
↳ IP address aliasing (reverse address lookup).
↳ SRV extensions (which port a particular service is running).
→ Mail services are typically not provided by the web server.
→ Web server is more popular than mail servers (www.microsoft.com)
↳ It is better served when associated with the same server name for both.
↳ somebody @ iist.ac.in is better than somebody @ receive-mailserver001.avio.iist.ac.in.
→ The DNS uses a special record to provide the domain name of the mail server associated with a domain name.
→ Helps multiple services such as web and mail using same domain name.

Features of DNS

① A hierarchical naming system
eg. iist.ac.in
② A distributed, federated and hierarchically organized server system.
③ An application layer protocol
- Simple, Query-Reply-based.
- Uses UDP (User Datagram Protocol) of the Transport layer.

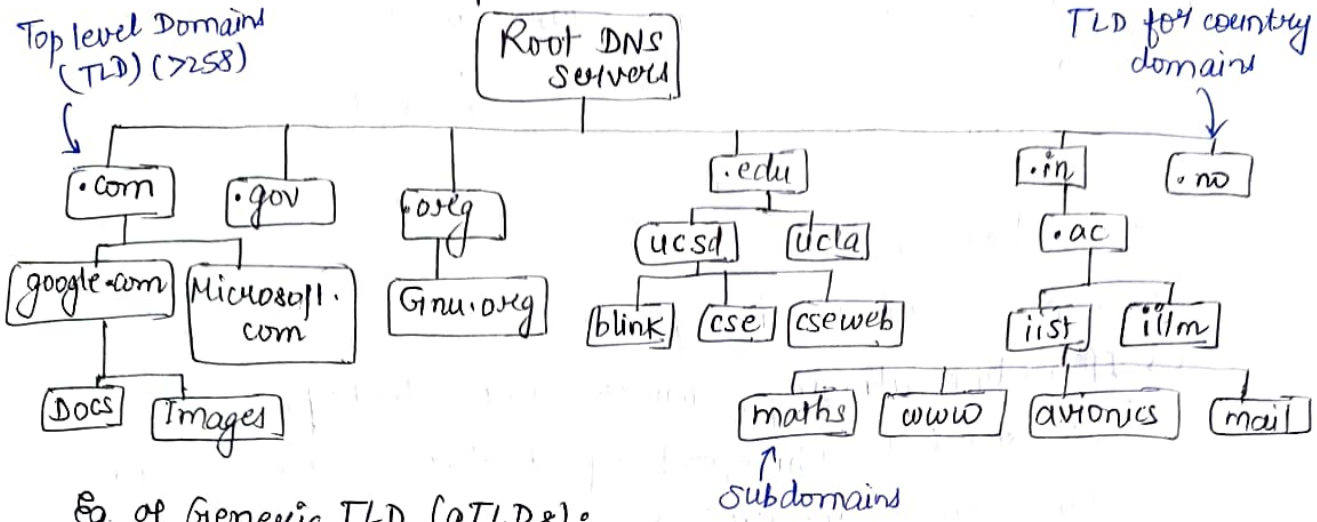
Hierarchical naming/server system: eg iist.ac.in

iist
- organization name (domain name) or subdomain
- under the control of domain (eg, images.google.com, mail.iist.ac.in)

ac
- Domain name for the category of organization

in
- Top level/country domain

→ 13 root domain servers
(10 in US, 2 in Europe, 1 in Asia)



Eg of Generic TLD (gTLDs):

- .aero → Aviation
- .asia → Asia
- .biz → business org.
- .cat → catalan language and culture
- .com → commercial
- .coop → co-operative org.
- .edu → education
- .gov → US government
- .info → open TLD
- .int → International org.
- .jobs → Jobs
- .mil → US Department of Defense
- .mobi → Mobile devices
- .museum → Museums
- .name → Personal
- .net → Networks
- .org → organizations
- .pro → credentialed professionals and related entities
- .tel → Publishing of contact data
- .travel → Travelling

→ Domain names are sold by domain name registrars.
↳ Very competitive business models.

→ Internet Corporation of Assigned Names and Numbers (ICANN)
↳ Responsible for accrediting various domain name registrars.

→ http://www.internic.net/ → For list of accredited registrars.

DNS Server Hierarchy

Root DNS (rDNS) servers:

- 13 of them (10 in US, 2 in Europe, 1 in Asia), each is a collection of servers.
- Do not maintain A records.
- Maintain records for locating Top level DNS servers.

Top level domain (TLD) DNS servers:

- Do not maintain A records.
- Responsible for the top level domains such as .com, .org, .edu, etc, and country-level domains like .us, .uk, .no, .jp, etc.
- More than 258 domains exist today.

Authoritative DNS server (aDNS):

- Provided by the organization which hosts the web servers and hosts.
- Maintain A record for many hosts (in some cases there maybe hierarchy of aDNS servers).
- Either hosted within the organization or through a DNS service provider.

Local DNS server (LDNS)

- Plays the intermediary role b/w client and the hierarchy of DNS servers
- Not part of the hierarchy of the DNS.

How DNS works?

A client calls resolver function:

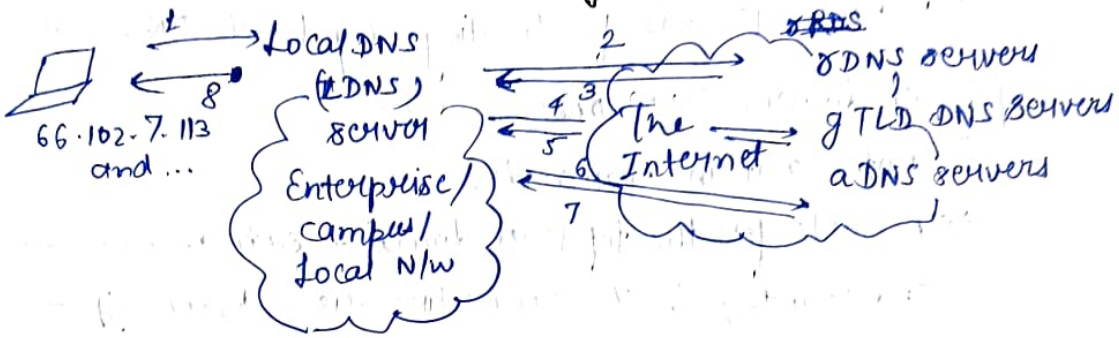
↳ A software library that implements the DNS client in an end-user machine.

↳ Resolver sends a DNS query message to its local DNS (LDNS) server.

↳ LDNS replies immediately if it has a cached information available.

↳ If LDNS does not have the IP address, then it tries:
- First the root DNS servers.
- Then the general TLD servers.
- Then one or more of the authoritative DNS servers.

⇒ Sequential and Iterative DNS querying



DNS APIs

→ For unix-based systems, the client calls

```
struct hostent * gethostbyname (const char * name);
```

↳ returns a struct of type hostent for the given host name.

↳ Name: char array containing the host name.

```
struct hostent * gethostbyaddr (const char *addr, int len, int type);
```

↳ used for reverse lookup (IP address to hostname).

```
struct hostent {
    char *h_name;  /* official name of host */
    char **h_aliases;  /* alias list */
    int h_addrtype;  /* host address type */
    int h_length;  /* length of address */
    char **h_addr_list;  /* list of addresses */
} // Declared in header file <netdb.in>.
```

A client-side code example :

```
...
int sfd;
struct sockaddr_un addr;
char host_name[256];
struct hostent * host_addr;
sprintf(host_name, "www.google.com"); // copies the name to the
                                      // host_name array
host_addr = gethostbyname(host_name);
if(!host_addr)
{
    fprint(stderr, "Unknown host: %s\n", host_name);
    exit(1);
}
sfd = socket(AF_UNIX, SOCK_STREAM, 0); // socket is opened for connection
                                       // establishment
if (sfd == -1) {
    perror("socket");
    exit(EXIT_FAILURE);
}
memset(&addr, 0, sizeof(struct sockaddr_un)); // clear structure
addr.sun_family = AF_UNIX;
strncpy(addr.sun_path, MY_SOCK_PATH, sizeof(addr.sun_path)-1);
```

Performance optimization of DNS :

- can be used for various performance optimization.
- Load balancing.
- Time-based name-to-address translation.
- Energy savings.
- can be used for providing security.
  - Moon computing → move the computing services to countries where the moon is (night falls, energy is cheap).
- Defending against cyber attacks.
- Highly customized contents.
  - Based on hardware, browsers and networks.

DNS-based Performance Optimization:

- Can be used for performance optimization.
  - ↳ Load balancing.
  - ↳ Response time improvement.
- Time-based name-to-address translation.
- Content-based name-to-address translation.
  - ↳ New servers over the Internet.
  - ↳ Content Distribution Network services.
- Defending against cyber attacks.
- Highly customized contents.

DNS Performance optimization services:

↳ Not originally intended → but very imp. today.

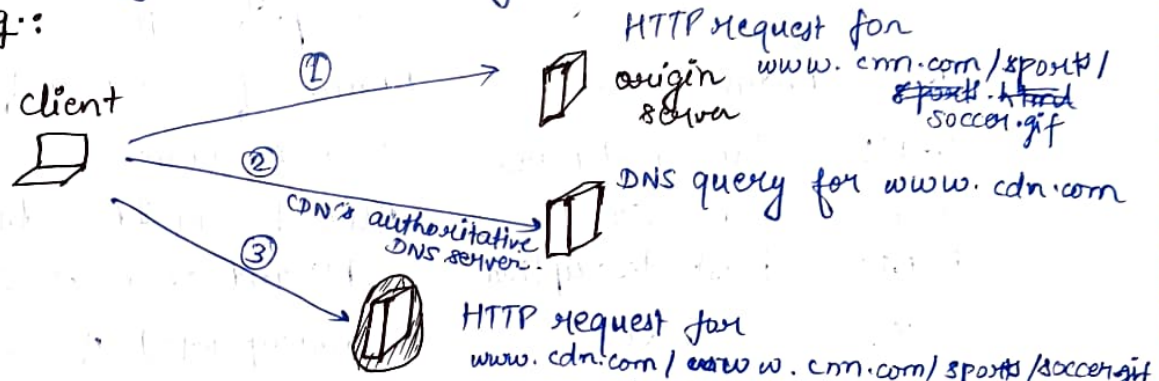
↳ Main performance optimization
- Load Distribution
- Provisioning Reliability
- Content Distribution network services.

↳ Load distribution
- DNS is used to provide load balancing among multiple servers that serve the same domain.
- A set of IP addresses is associated with a canonical name.
- When a client queries, DNS server replies with the entire set of IP addresses.
- Each client makes use of the first IP address in the set.
- The set of IP addresses is rotated with subsequent requests.
- Therefore, the load is almost equally balanced among the set of servers.

Content Distribution Network Services

→ DNS is effectively used to provide content dist. n/w services.
→ When a client queries for an address of a host, the DNS server provide the name of a CDN.
→ The CDN's DNS replies with the address of the web cache nearest to the client's location.
→ The client receives content from a nearby web cache than the main web server.
- Quick response
- Better load balancing
- High server scalability.

CDN eg.:



Origin server (www.cnn.com)
- Distributes HTML
- Replaces cnn.com/sports/soccer.gif with www.cdn.com/www.cnn.com/sports/soccer.gif

CDN company (cdn.com)
- Distributes gif files
- Uses its authoritative DNS server to route Redirect requests.

DNS for web server/service security:

→ Distributed Denial of Service Attacks can by prevented by CDNs.
→ All CDN service providers are now offering security services.
→ Most recent use of DNS: Security as a service.

Eg CDN companies: Rackspace CDN, Google Cloud CDN, Akamai.com, Swarmify, Limelight, Microsoft Azure CDN, Amazon cloud Front, cloud flare.

DNS resource record types:

→ A Resource Record (RR) is a basic data element in a DNS database.

→ DNS server stores different types of RRs.

→ Each RR corresponds to a set of information for a particular service DNS provides.

→ A record consists of multiple values that contains {Name, TTL, Class, Type, Value}.

Type	Meaning	Value
A	IP address of a host	32-bit integer.
MX	Mail Exchange	Priority, domain willing to accept e-mail
NS	Name server	Name of a server for this domain.
CNAME	Canonical Name	Domain name.
PTR	Pointer	Alias for an IP address.
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text.

DNS record types: Record: {Name, Value, Class, Type, TTL}

① Type A: Name = Hostname
Value = IP address
TTL = time to live

↳ Gives hostname-to-address resolution

Eg. {cseweb.ucsd.edu, 132.239.51.6, A, 19800}

② Type NS: Name = domain (eg., google.com)
Value = address of the authoritative DNS (aDNS) server.

↳ aDNS server Helps with resolution of hostnames in the domain.

③ Type CNAME: Name = hostname
Value = canonical host name for hostname

↳ Provides hostname aliasing service

Eg. {docs.google.com, writely.l.google.com, CNAME}

④ Type MX : Name = hostname
Value = canonical name of the mail server associated with the hostname.

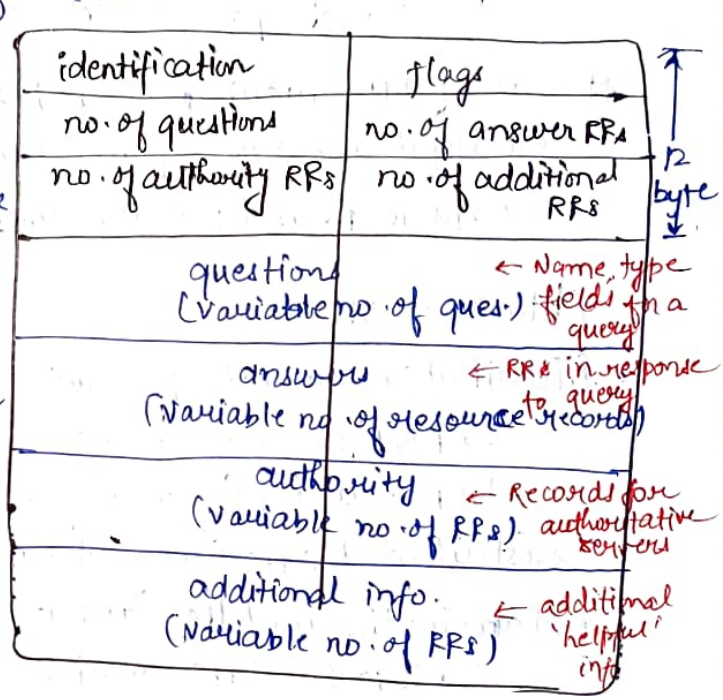
Eg. {iist.ac.in, mail.iist.ac.in, MX}

DNS protocol, messages :

DNS protocol : query and reply messages, both with same message format.

msg header
identification : 16 bit # for query, reply to query uses same #

flags :
- query (0) or reply (1)
- recursion desired
- recursion available
- reply is authoritative



Inserting records into DNS :

Eg. New startup (Network-Spacia)
↓
Register name at DNS registrar (network-spacia.com)
- Provide names, IP addresses of authoritative name server (primary & secondary)
- Registrar inserts two RRs into .com TLD server

{network-spacia.com, dns1.network-spacia.com, NS}
{dns1.network-spacia.com, 212.212.212.1, A}
↓
Create authoritative DNS server for network-spacia.com
- Type A records for www.network-spacia.com;
- Type MX records for mails for network-spacia.com

New Services based on DNS:

- Network security services → through DNS
- DNS censoring
  - ↳ Political and religious censoring of DNS services (eg. China).
- Many new (suspicious) Private DNS services
  - → Google Public DNS → free (privately owned)
  - → OpenDNS.com
    - ↳ not open source, open to querying from anyone

50% of DNS queries issued to root servers do not return successful answers [2013 study].
- ↳ Main Reason: malformed queries with invalid TLDs.

Attacks on DNS:

- Distributed Denial of service
  - ↳ B/W flooding or resource consumption
  - ↳ Flooding ping messages.
- Distributed DNS query attack.
- Man-in-the middle attack.
  - ↳ Capture DNS queries and generate bogus DNS replys.
- DNS poisoning
  - ↳ Erroneously populate the DNS caches by originating bogus replys.
- Reflection attacks against hosts by
  - ↳ Generate spoofed DNS queries and thus create large no. of replys.
  - ↳ Targetting a host with DNS replys.
  - ↳ Querys can be ANY? so that replys will be bigger packets.

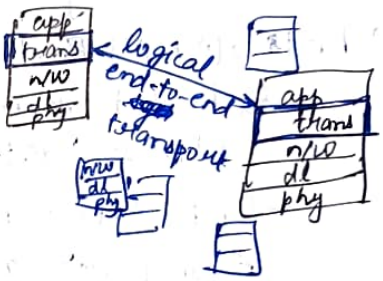
Transport Layer

Transport services and protocols

→ Provides logical comm. b/w app processes running on different hosts.

→ Transport protocols run in end systems.
- Send side: breaks app messages into segments, passes to network layer.
- receiver side: reassembles segments into messages, passes to app layer.
- B/w send and receive sides: controls n/w congestion, end-to-end flow control, reliability, QoS, etc.

→ More than one transport protocol available to apps.
Internet: TCP and UDP.



Transport Layer Services offered:

→ Multiplexing/demultiplexing
→ Network Bandwidth utilization
→ End-to-end (reliable or unreliable) data transfer.
→ Flow control.
→ Congestion control
→ End-to-end quality of service.

Internet transport-layer protocols:

TCP:
- Reliable end-to-end transmission
- In order delivery
- congestion control
- Flow control
- Connection-oriented
- Stream-oriented

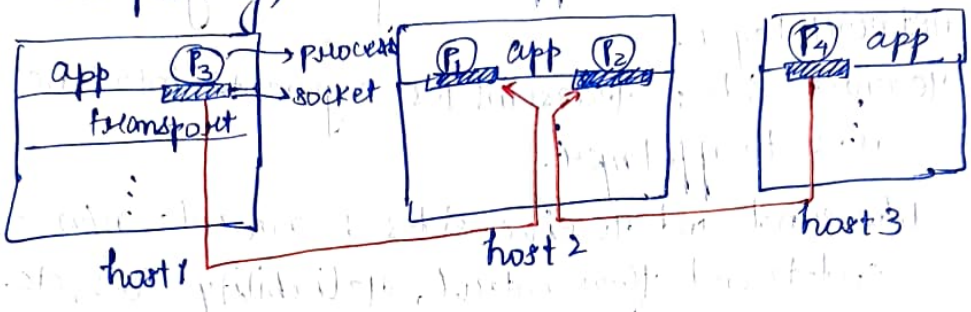
UDP:
- Unreliable (best effort)
- Not guaranteed for ordered delivery
- Message oriented
- connection-less

Services not available:
- delay guarantees
- bandwidth guarantees.

Multiplexing / Demultiplexing:

Demultiplexing at receiver host: Delivering received segments to correct socket.

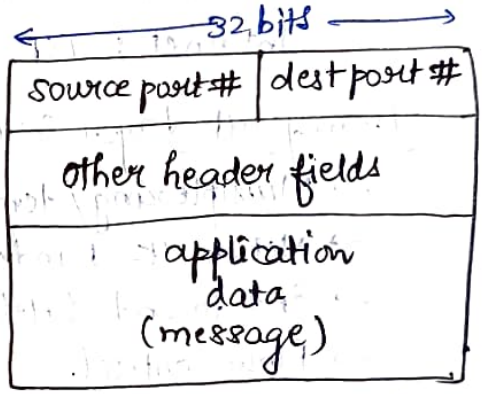
Multiplexing at sender host: Gathering data from multiple sockets, enveloping data with header (later used for demultiplexing).



How demultiplexing works?

→ Host receives IP datagrams.
- Each datagram has source IP address, dest. IP address.
- Each datagram carries 1 transport-layer segment
- Each segment has source, destination port no.

→ Host uses IP addresses & port no.s to direct segment to appropriate socket.



TCP/UDP segment format

Connectionless Demultiplexing:

→ When host receives UDP segment:
- checks dest. port no. in segment.
- directs UDP segment to socket with that port no.

→ IP datagrams with different source IP addresses and/or source port no.s directed to same socket.

→ Source port and source IP provides "return address".

UDP: User Datagram Protocol [RFC 768]

↳ Simple, "no frills", or "bare bones" Internet transport.

↳ "Best effort" service, UDP segments may be:
- lost
- delivered out of order to app.

↳ connectionless:
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- no congestion control: UDP can blast away as fast as desired.
- no handshaking b/w UDP sender, receiver.
- each UDP segment handled independently of others.

↳ Two popular apps for UDP:
- SNMP
- DNS

Connection-oriented demux

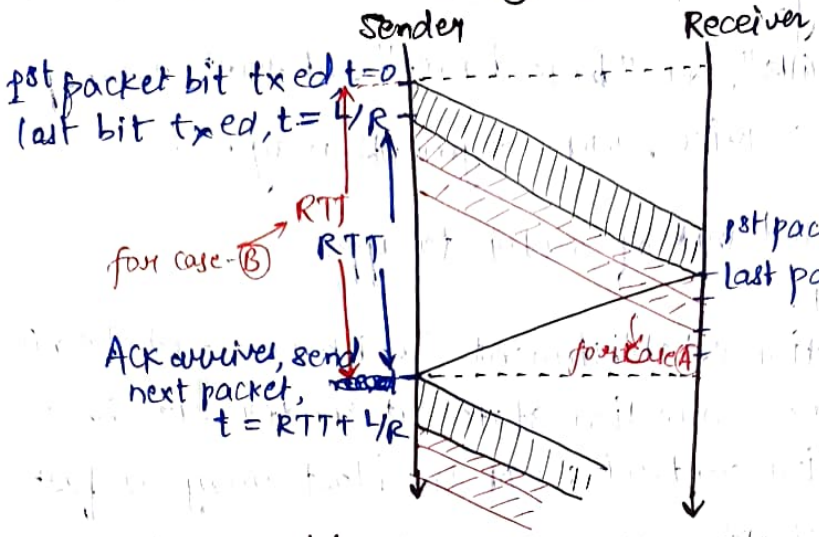
→ TCP connection is identified by 4-tuple.
- source IP address.
- source port no.
- dest. IP address.
- dest. port no.

→ Recv host uses all 4 values to direct segment to the appropriate socket.

→ server host may support many simultaneous TCP sockets:
- each socket identified by its own 4-tuple.

→ web servers have different sockets for each connecting client.
- non-persistent HTTP will have different socket for each request.

Network Bandwidth Utilization:



$$\text{Utilization} = \frac{L/R}{L/R + RTT}$$

$$= \frac{1500 \times 8 / 10^6}{1500 \times 8 / 10^6 + 2 \times [3600 \times 1000 / 3 \times 10^8]} = 4.7\%$$

Case-A: Increasing N/W B/W utilization

$$\text{Utilization} = \frac{3 \times L/R}{L/R + RTT} = \frac{3 \times 0.012\,s}{0.012\,s + 2 \times 0.12\,s} = 14\%$$

↑ by a factor of 3

Case-B: N/W B/W utilization with RTT estimation differently

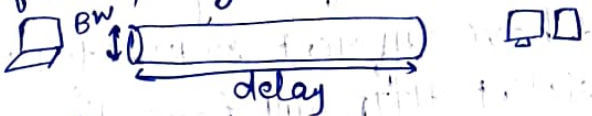
If RTT starts from $t=0$:

$$\text{Utilization} = \frac{L/R}{RTT}$$

How much can fill the pipe?

→ Txn rate optimization to achieve n/w capacity.

→ Bandwidth delay product (BDP) is the suitable metric for flow capacity.

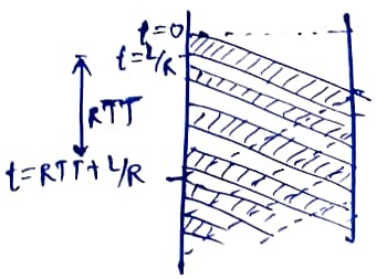


Eg: $R = 1\,Mbps$, $RTT = 1ms \Rightarrow BDP = 10^6 \times 10^{-3} = 10^3\,bits$

Filling the pipe operation: BDP

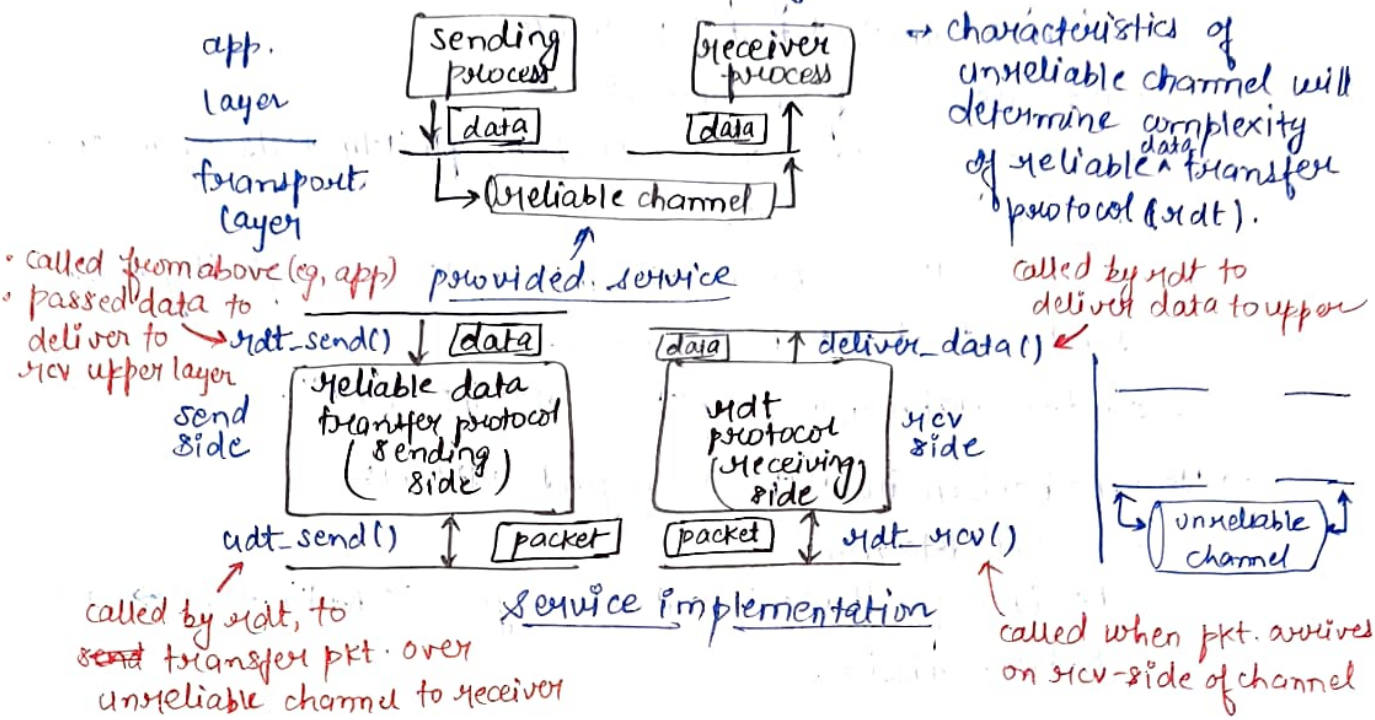
Case-A: # packets $= N = \frac{BDP}{\text{packet length}} + 1$

Case-B: # packets $= N = \frac{BDP}{\text{packet length}}$



Principles of Reliable Data Transfer:

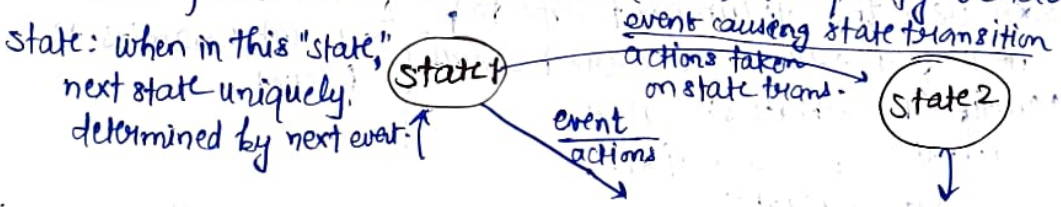
↳ Imp. in app., transport, link layers.



→ characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt).

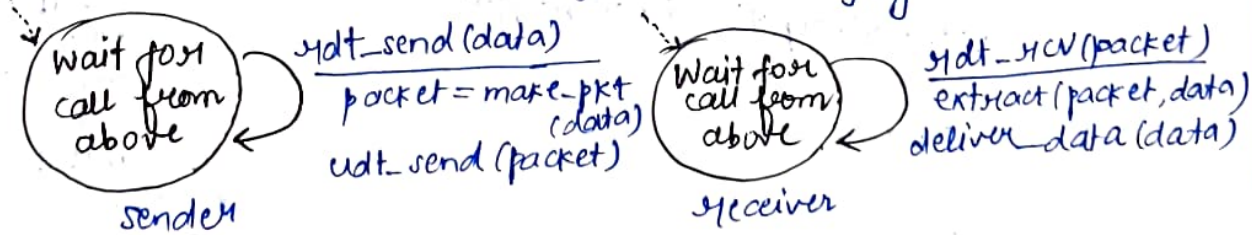
Reliable data transfer:

We will
- → Incrementally develop sender, receiver side of rdt protocol.
- → consider only unidirectional data transfer.
  - but control info. will flow on both dir^{n}s.
- → Use finite state machines (FSM) to specify sender, receiver.



Rdt 1.0: Reliable transfer over a reliable channel

- → Underlying channel perfectly reliable (no bit errors, no pkt loss).
- → separate FSMs for sender, receiver:
  - sender sends data to underlying channel.
  - receiver reads data from underlying channel.



Rdt 2.0 : Channel with bit errors

→ Underlying channel may flip bits in packet.
- checksum to detect bit errors.

→ How to recover from errors:
- acknowledgements (ACKs) : receiver explicitly tells sender that packet received OK.
- negative acknowledgements (NAKs) : receiver explicitly tells sender that pkt had errors.
- sender retransmits pkt on receipt of NAK.

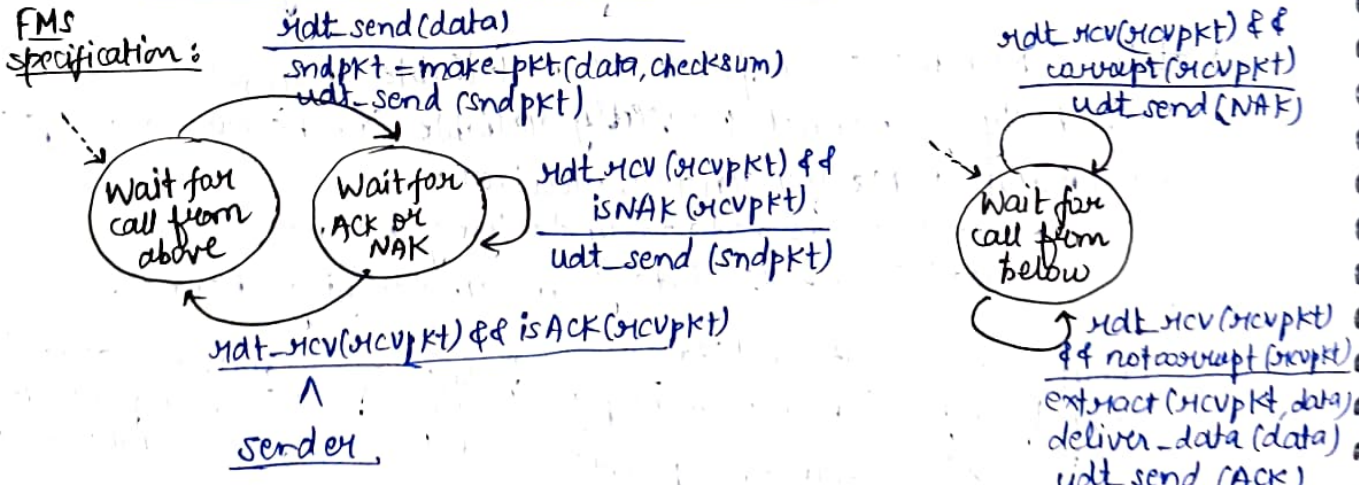
→ New mechanism in rdt 2.0 (beyond rdt 1.0):
- Error detection
- Receiver feedback : control msgs (ACK, NAK) rcvr → sender
- Retransmission.

Two approaches of error handling:

① Forward Error Correction (FEC)
- Txr adds redundant info. in the packet.
- eg checksum or CRC field.
- Rxr uses the redundant info. to check for errors or correct errors.
- Multiple possibilities: Error detection vs correction.

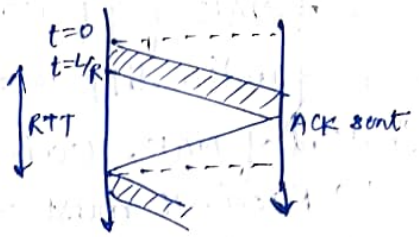
② Automatic Repeat reQuest (ARQ)
- Receiver detects error in a packet.
- Receiver sends a response to indicate the correctness of the packet.
- Txr retransmits the errored packet.

FMS specification:



Problem: stop and wait operation

$$U_{sender} = \frac{L/R}{RTT + L/R}$$



Another fatal flaw: if ACK/NAK corrupts.

→ sender doesn't know what happened at receiver.

→ ACK → NAK
- Retxn results in duplicate data
- data integrity violation.

→ NAK → ACK
- No retxn can result in data-loss.

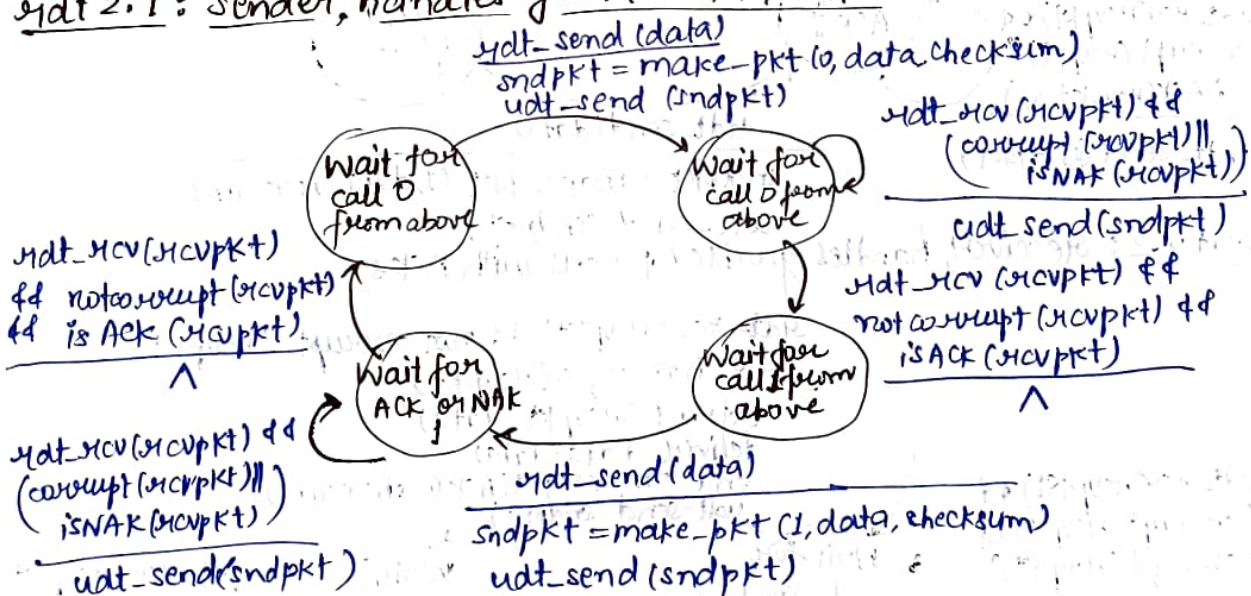
Handling ACK/NAK errors:

→ Sender retransmits current pkt if ACK/NAK garbled.

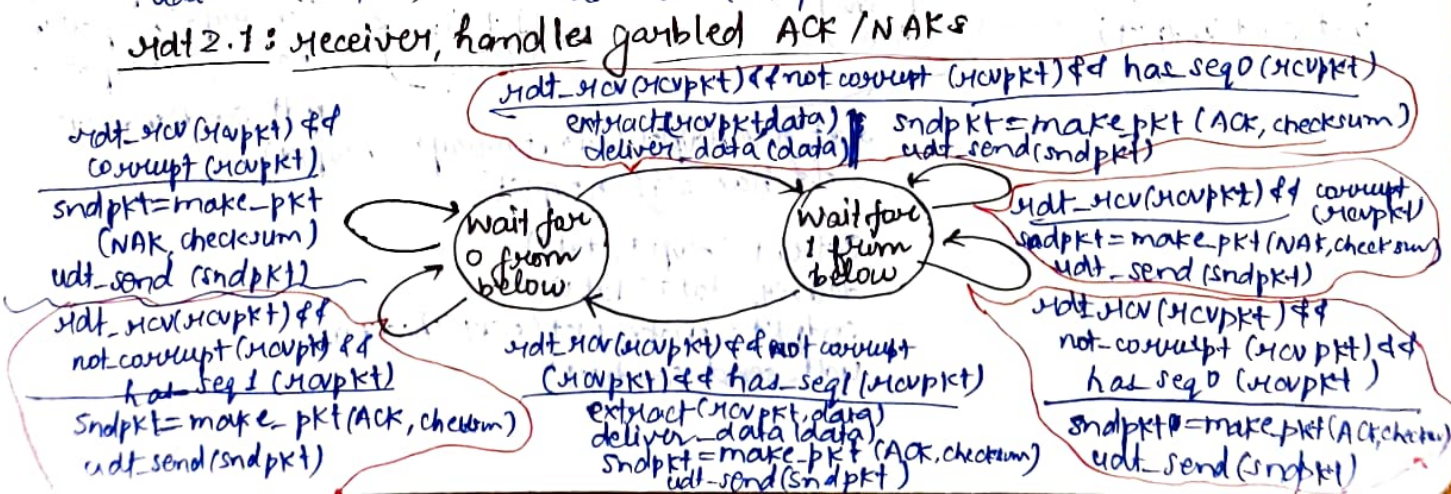
→ sender adds sequence no. to each pkt.

→ Rx should discard (doesn't deliver up) duplicate data.

rdt 2.1: sender, handles garbled ACK/NAKs



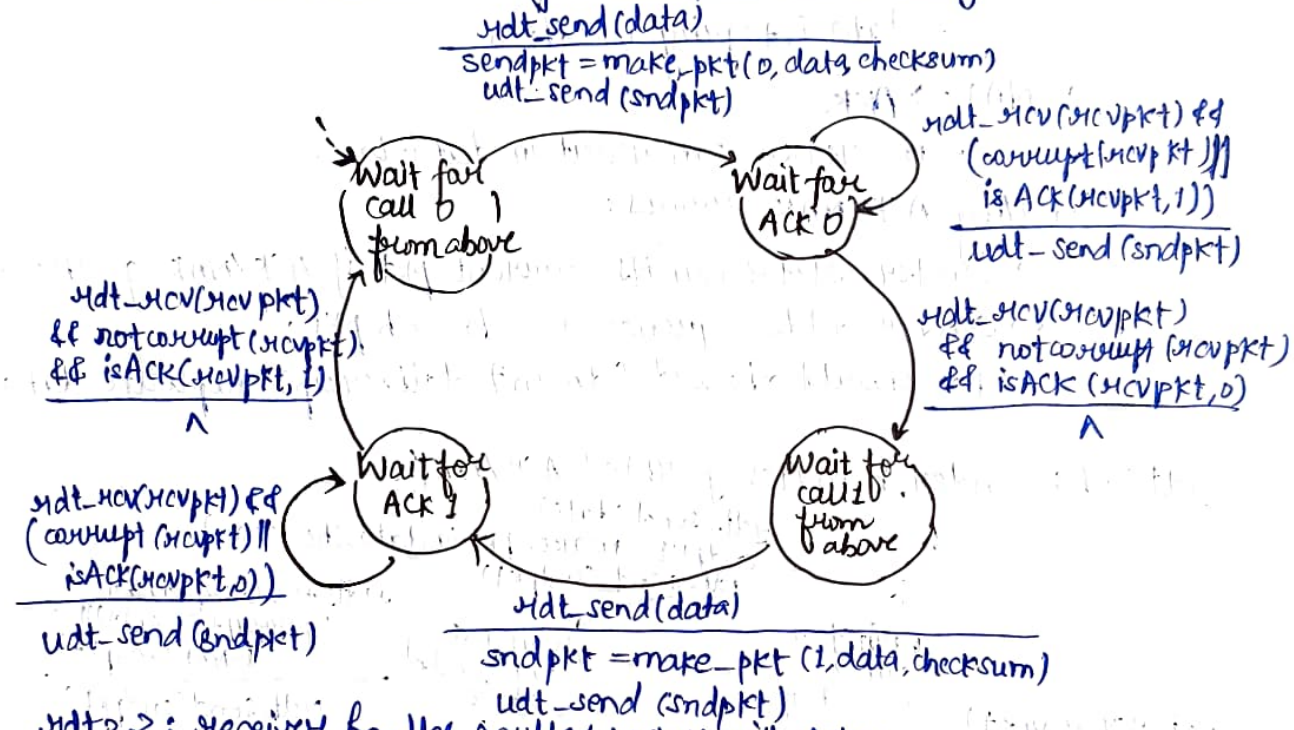
rdt 2.1: receiver, handles garbled ACK/NAKs



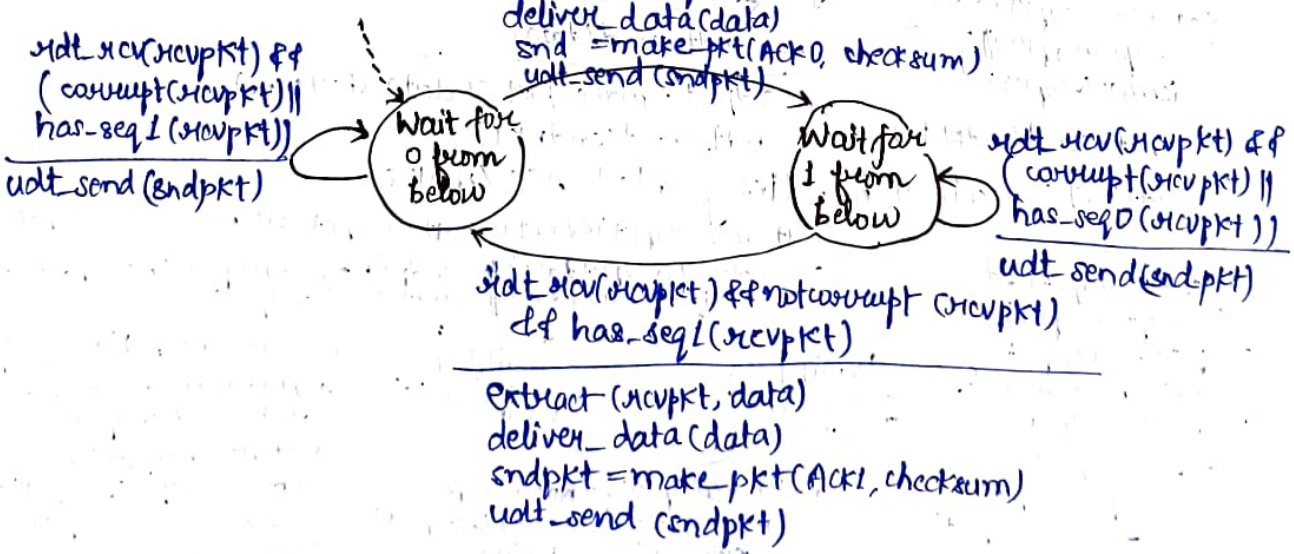
rdt 2.2 : A NAK-free protocol

- Same functionality as rdt 2.1, using ACKs only.
- Instead of NAK, receiver sends ACK for last pkt received OK.
  - Receiver must explicitly include # seq # of pkt being ACKed.
- Duplicate ACK at sender results in same action as NAK: retransmit current pkt.

rdt 2.2 : Sender, handles garbled packets with only ACKs.



rdt 2.2 : Receiver, handles garbled packets with ACKs.



Rdt 3.0 : Channels with errors and packet loss

New assumption:

Underlying channel can also lose packets (data or ACKs)

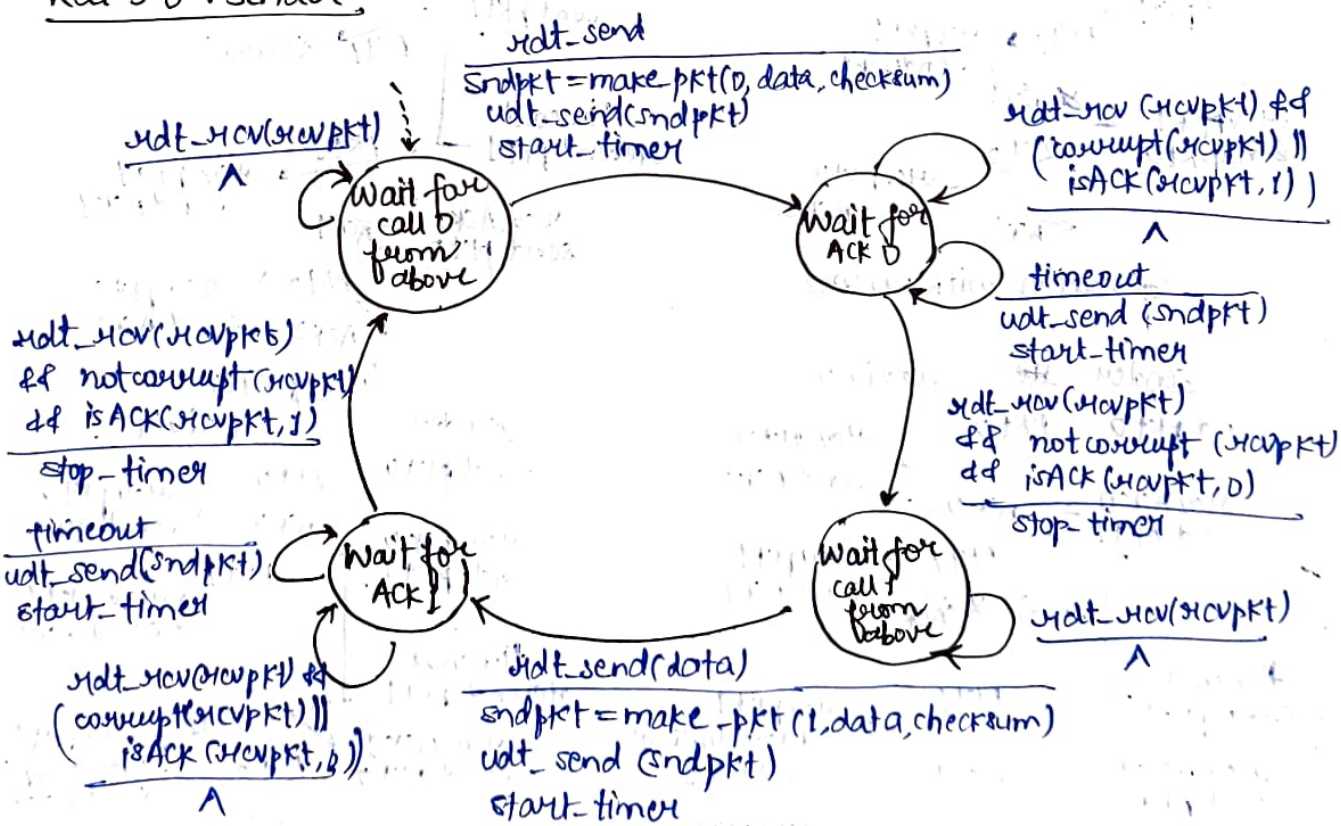
- checksum,
- sequence Number,
- ACKs and
- Retransmissions.

(will be of help, but not enough).

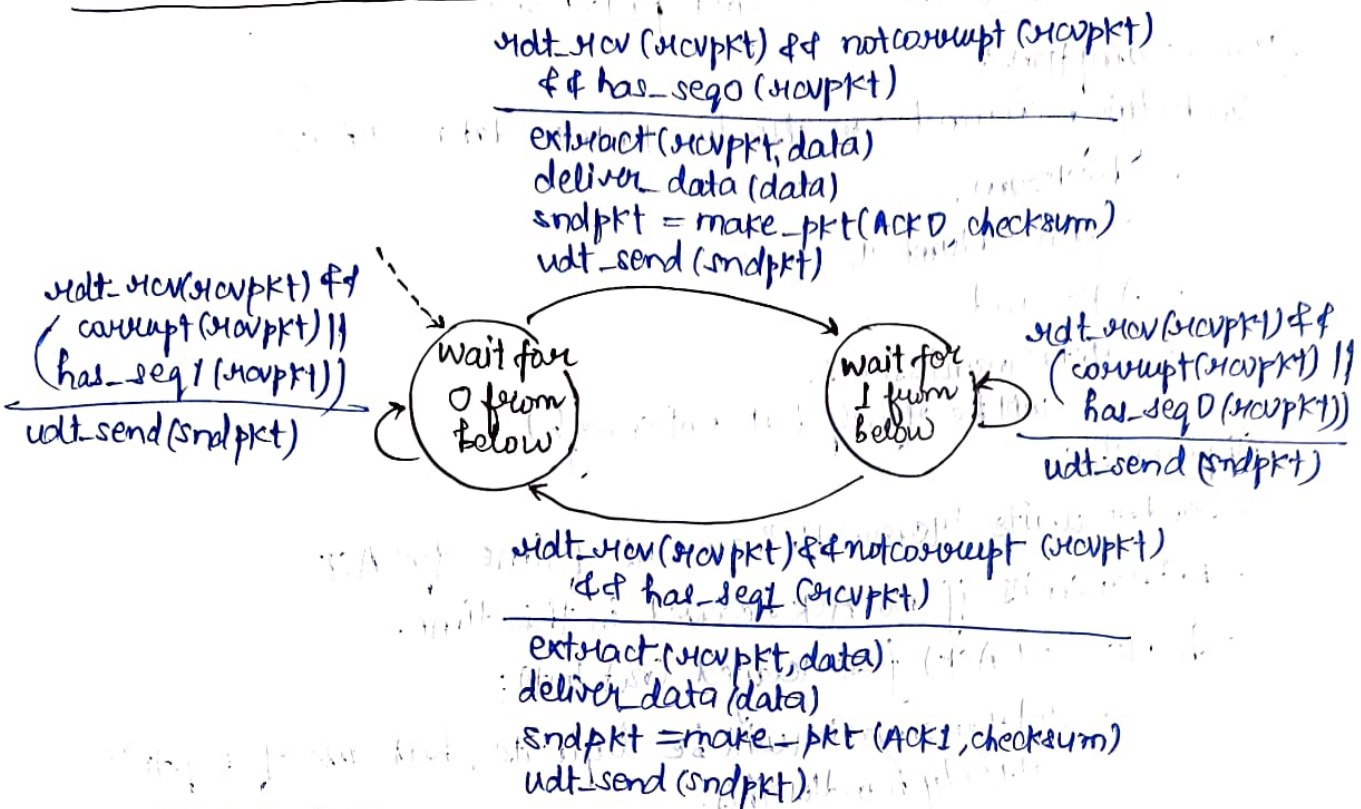
Approach:

- Sender waits "reasonable" amount of time for ACK
- Retransmits if no ACK received in this time.
- If pkt (or ACK) just delayed (not lost):
  - Retransmission will be duplicate, but use of seq.#'s already handles this.
  - Receiver must specify seq# of pkt being Acked.
- Requires countdown timer.

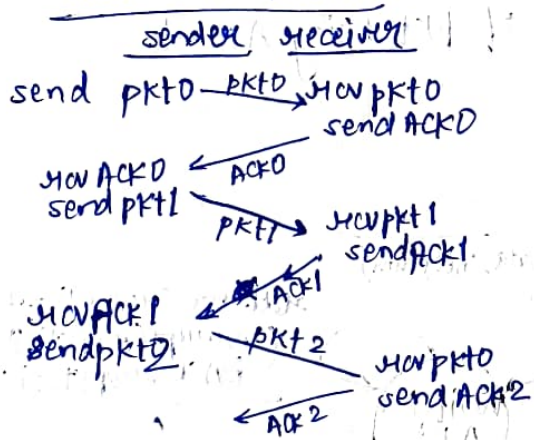
Rdt 3.0 : sender



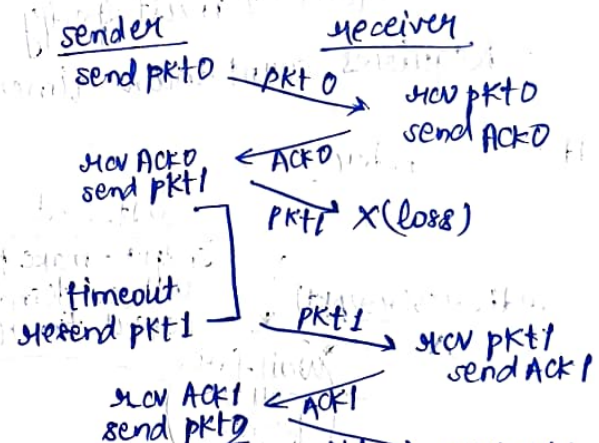
Receiver side FSM : Rdt 3.0



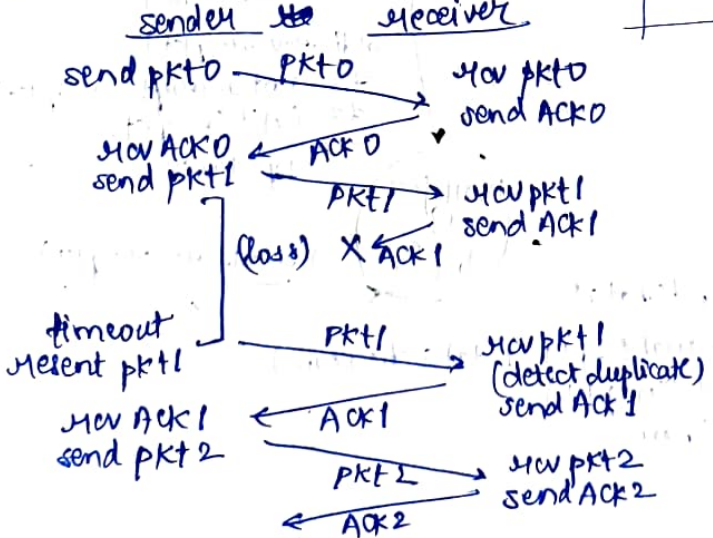
rdt3.0 in Action:



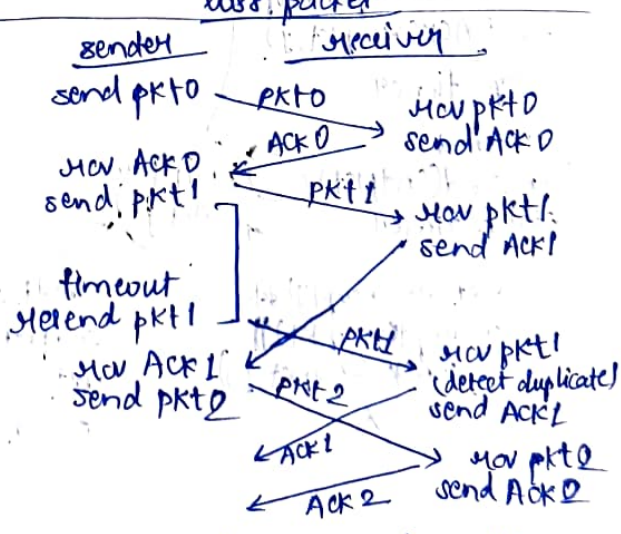
Operation with no loss



loss packet



lost ACK



premature timeout

Implementation of Finite State Machines

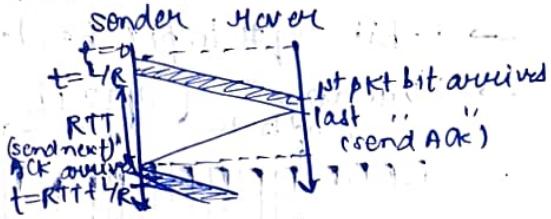
→ The conditional statements of any programming language (e.g., C)
- if-then-else
- switch-case
- A sequence of function calls
- A set of function pointers.

Reliable Transfers so far

↳ Reliable data transfer over unreliable data channels:
- K&R Rdt 1.0 : Reliable txn over reliable channel.
- K&R Rdt 2.0 : Reliable txn over DATA pkt corruption channel.
- K&R Rdt 2.1 : Reliable txn over DATA and ACK/NAK corruption channel.
- K&R Rdt 2.2 : Reliable txn over DATA/ACK corruption channel (NAK-free)
- K&R Rdt 3.0 : Reliable txn over DATA and ACK corruption/Loss channel.

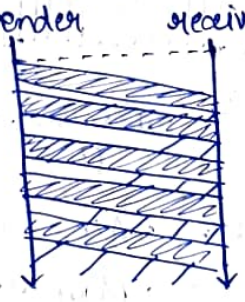
→ Rdt 2.0 – 3.0 suffers from low utilization.

$$U_{sender} = \frac{L/R}{RTT + L/R}$$



→ Filling the pipe operation: BDR

$$U_{sender} = \frac{N \times L/R}{RTT + L/R}$$



Sliding Window / Pipelined Transport Protocols

Go-back-N:

↳ Sender can ~~send~~ have up to N unacked packets in pipeline.

↳ Receiver only sends cumulative acks.

↳ Does not ack packet if there is a gap.

↳ Sender has timer for oldest unacked packet.

↳ If timer expires, retransmits all unacked packets.

Selective Repeat:

↳ Sender can have up to N unacked packets in pipeline.

↳ Rcvr sends individual ack for each packet.

↳ Sender maintains timer for each unacked packet.

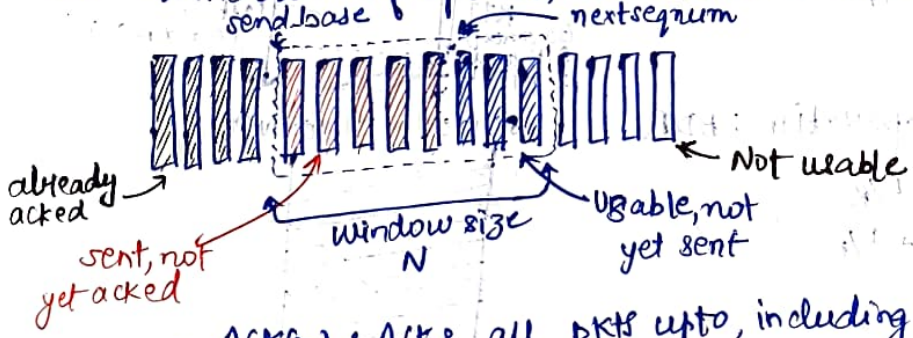
↳ When timer expires, retransmit only unacked packet.

Go-Back-N

Sender:

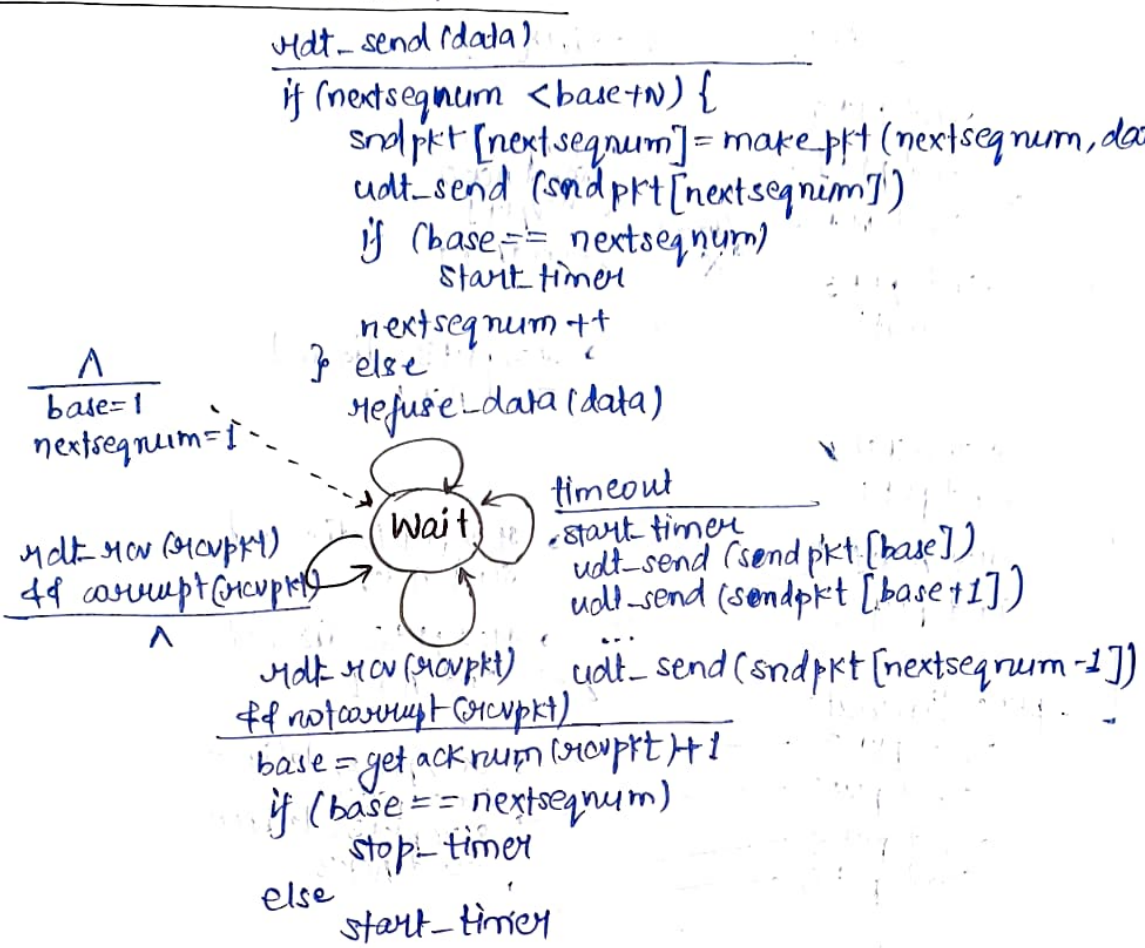
→ k-bit seq# in pkt header

→ "window" of upto N, consecutive unacked pkts allowed.

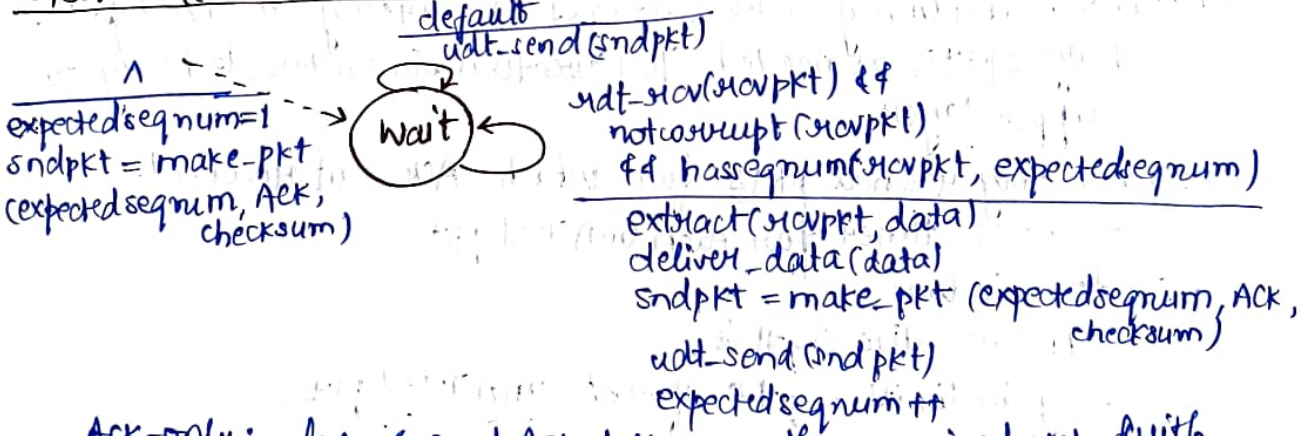


- ACK(n): Acks all pkts upto, including seq# n - "cumulative ACK"
  - ↳ May receive duplicate ACKs (see receiver)
- Timer for each in-flight pkt
- timeout(n): retransmit pkt n and all higher seq# pkts in window.

GBN: sender's extended FSM



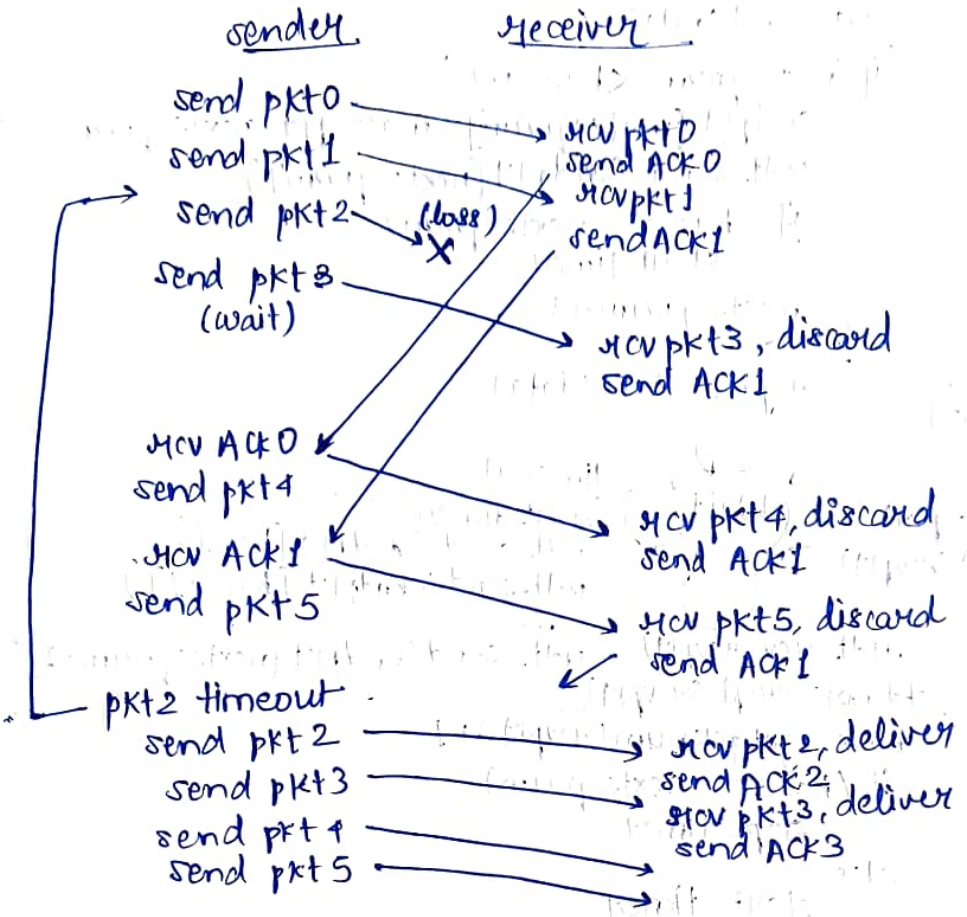
GBN: receiver extended FSM



ACK-only: always send ACK for correctly-received pkt with highest in-order seq#
- may generate duplicate ACKs
- need only remember 'expectedseqnum'

→ out-of-order pkt:
- discard (don't buffer) → no receiver buffering!
- Re-ACK pkt with highest in-order seq#

GBN: an example action



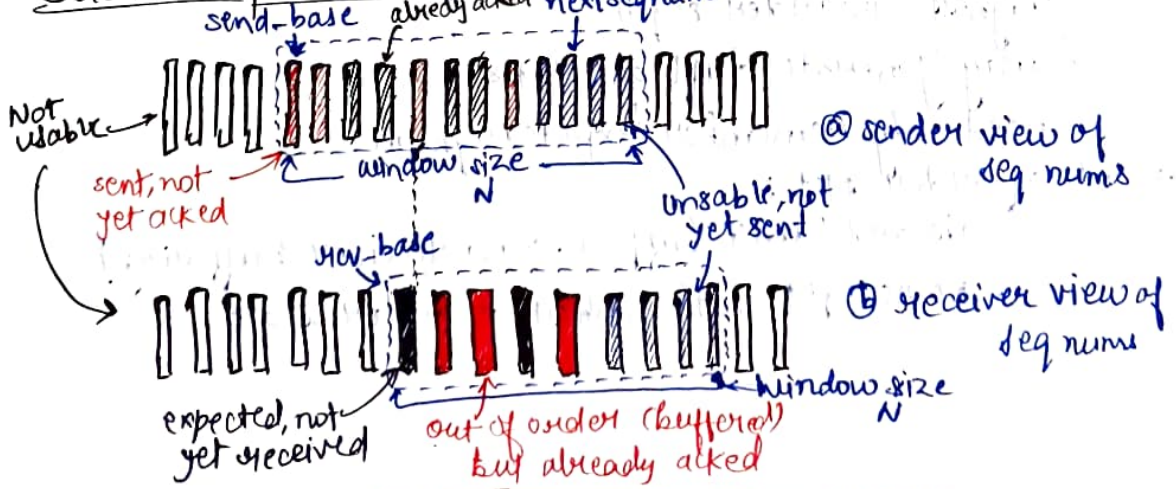
Selective Repeat sliding window protocol

↳ Receiver individually acknowledges all correctly received pkts.
- buffers pkts, as needed, for eventual in-order delivery to upper layer.

↳ Sender only resends pkts for which ACK not received.
- Sender timer for each unACKed pkt.

↳ Sender window
- N consecutive seq#'s
- again limits seq#s of sent, unACK'ed pkts.

Selective Repeat: sender, receiver windows



selective repeat:

sender:

data from above
- if next available seq# in window
  - send pkt
  - start timer
- else
  - reject the packet

timeout (i)
- resend pkt, i, restart timer

ACK (K) in [sendbase, sendbase+N]
- mark pkt K as received
- if K smallest unACKed pkt, advance window base to next unacked seq #.

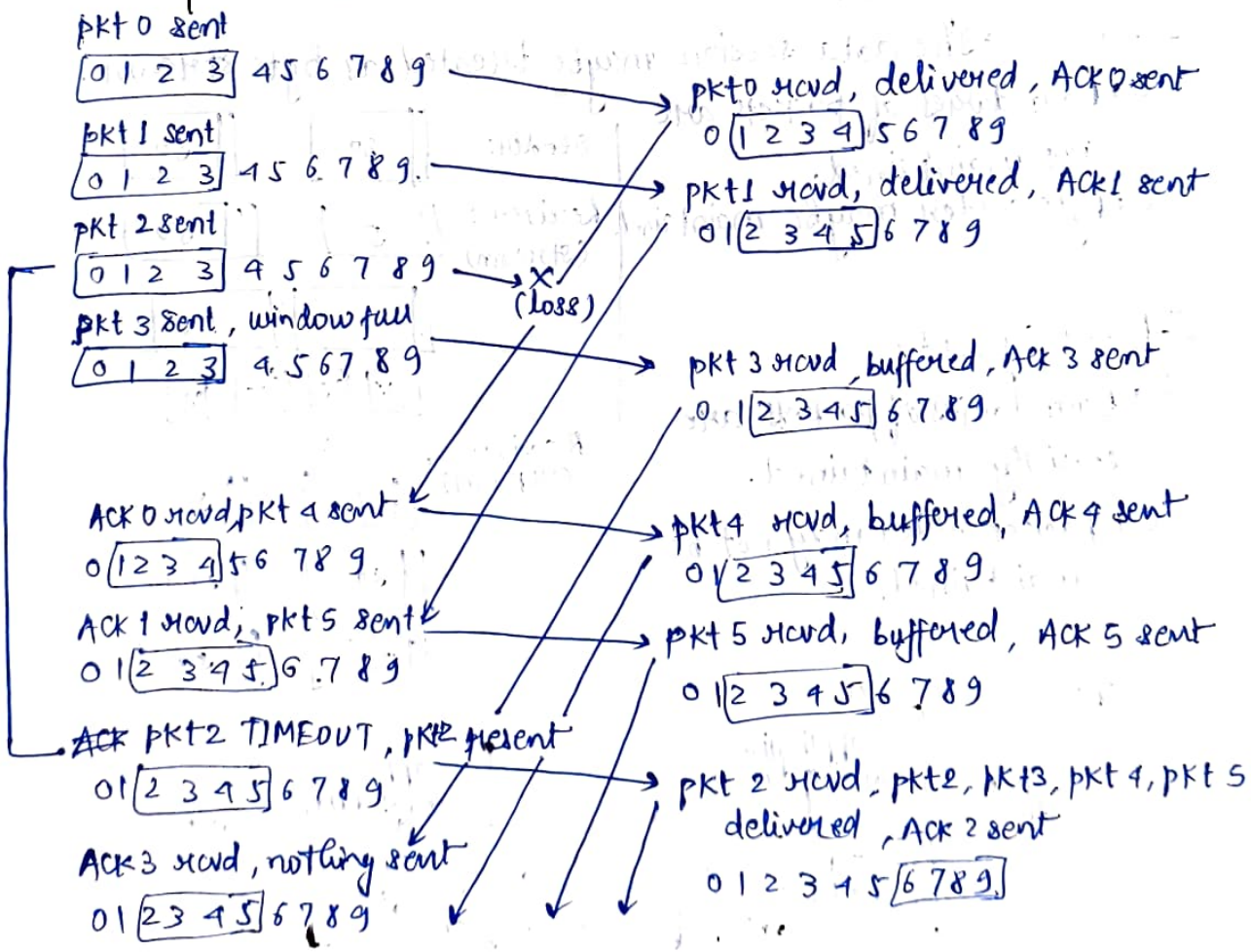
receiver:

pkt in in [rcvbase, rcvbase+N-1]
- send ACK(i)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet received pkt

pkt i in [rcvbase-N, rcvbase-1]
- ACK(i)

otherwise:
- ignore.

Selective repeat in action:



TCP : Transmission control Protocol

- RFCs : 793, 1122, 1323, 2018, 2581
- Point-to-point : one sender, one receiver
- Reliable, in-order, byte stream → no "message boundaries"
- Pipelined — TCP congestion and flow control set window size.
- Send & receive buffers.
- Full duplex data
  - bi-directional data flow in same connection
  - MSS : max^m segment size.
- Connected-oriented
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange.
- Flow controlled — sender will not overwhelm receiver.

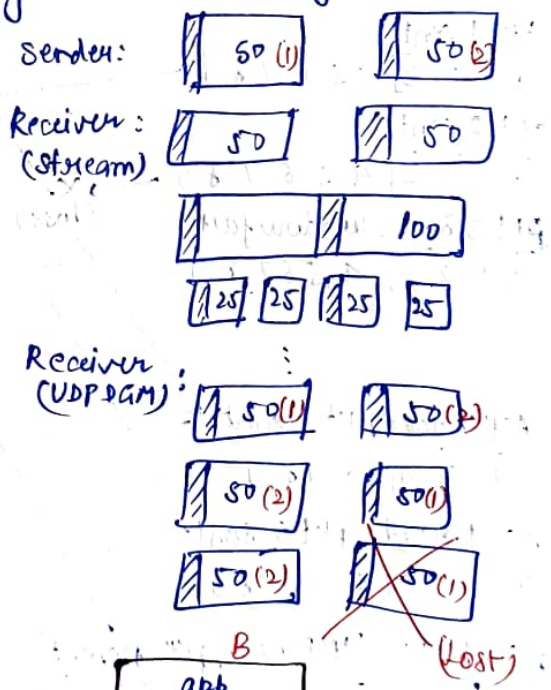
Stream vs Datagrams

Stream: → The data session maybe treated as byte stream.

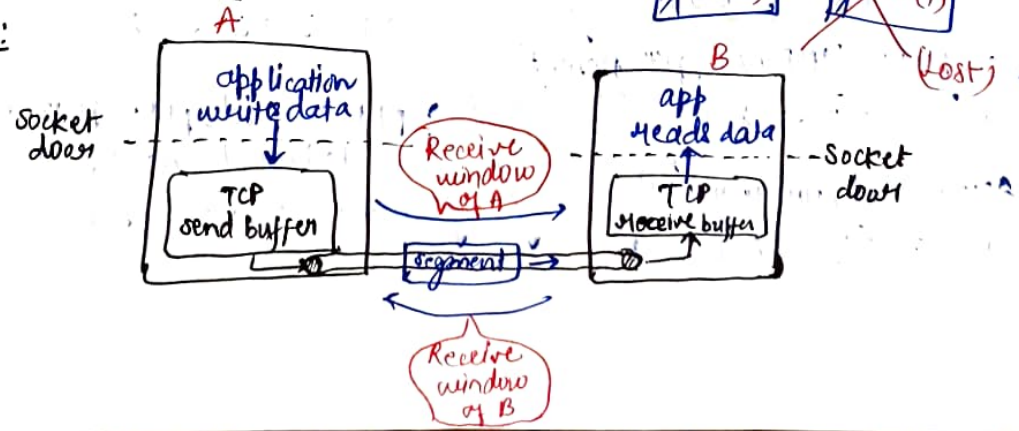
- Boundaries of packets are not maintained.
- Byte order maybe maintained

UDP Datagrams:

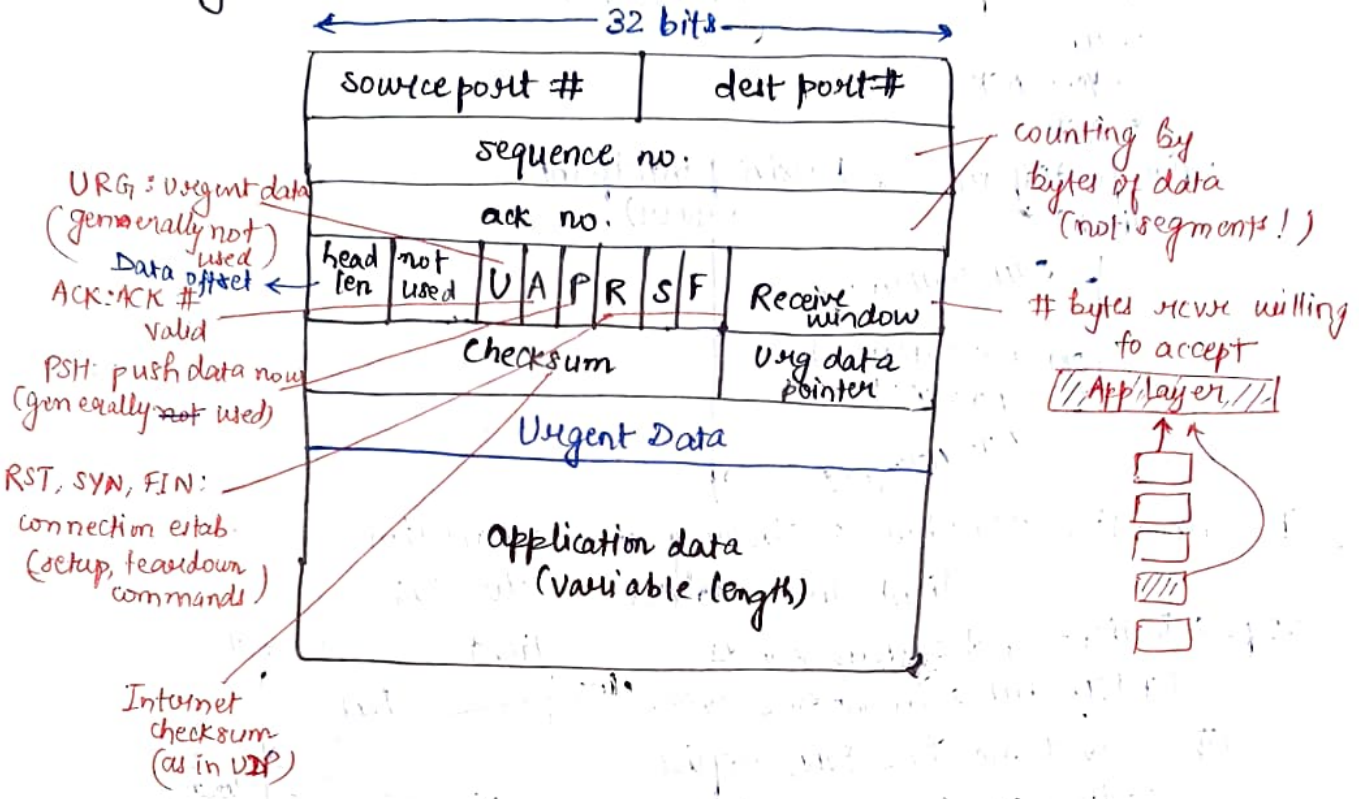
- Boundaries of packets are strictly maintained.
- Packet order may not be maintained.



TCP:



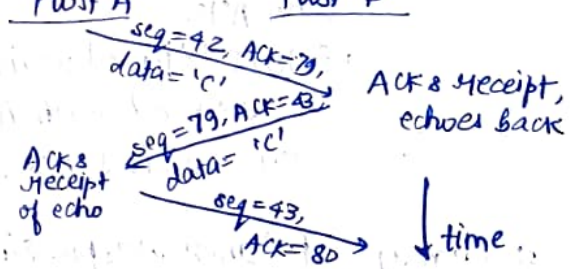
TCP segment structure :



Seq #: Byte stream no. of first byte in segment's data

ACKs: seq # of next byte expected from other side
↳ cumulative ACK.

Q) How receiver handles out-of-order segments?
↳ TCP spec. doesn't say,
- upto implementor.

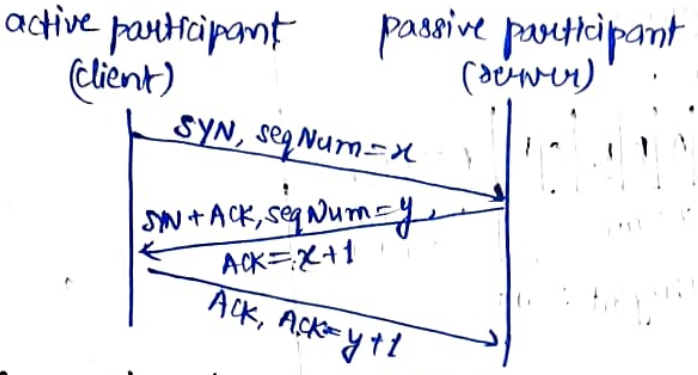


Flags:
- ACK: To mark relevance of ACK field.
- SYN-FIN: Connection setup and teardown.
- Urgent flag: To mark the presence of URGENT data.
  ↳ URGENT data: ssize_t send(int sockfd, const void *buf, size_t len, int flags);
- Push flag (PSH): To quickly deliver data without waiting for buffering.
  ↳ Use setsockopt() with TCP_NODELAY
- Reset flag: Reset the connection.

→ Flags can be used with MSG_OOB.

TCP connection setup : Three way handshake

- SYN
- SYN + ACK
- ACK



TCP Connection teardown: Closing a connection

↳ client closes socket : close();

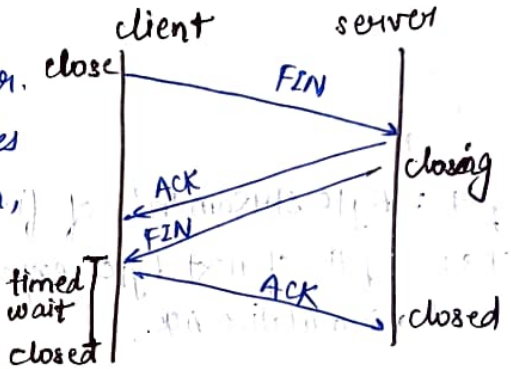
Steps : (i) client end system sends TCP FIN ctrl segment to server.

(ii) server receives FIN, replies with ACK. closes connection, sends FIN.

(iii) client receives FIN, replies with ACK.

↳ Enters "timed wait" – will respond with ACK to received FINs.

(iv) server, receives ACK. Connection closed.



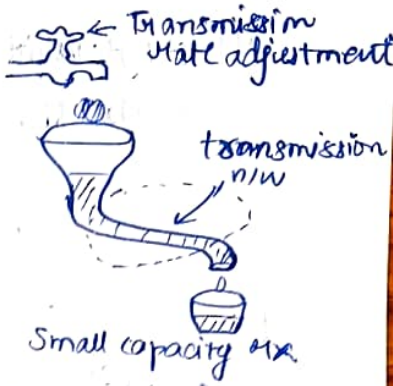
Transport layer responsibilities:

- Multiplexing / De-multiplexing
- Reliable / Unreliable delivery
- In order delivery
- BW utilization
- Message or byte stream service
- Flow control
- Congestion control.

Transport layer protocols → UDP, TCP

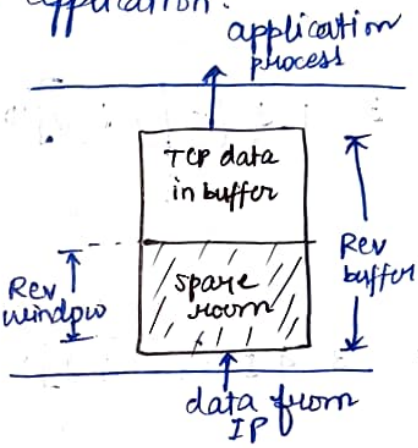
UDP →
- light-weight
- Mux-Demux provided
- Message based
- connection-less
- Full-duplex operation
- no reliability
- no in-order delivery
- no sliding window-based control of transmissions.

TCP Flow Control



↳ A fast sender and fast n/w feeding ↔ a constrained receiver.

→ Receiver side of TCP connection has a receive buffer → may be ~~un~~limited.
→ Application process may be slow at reading from buffer.
→ Too many connections at the same application.



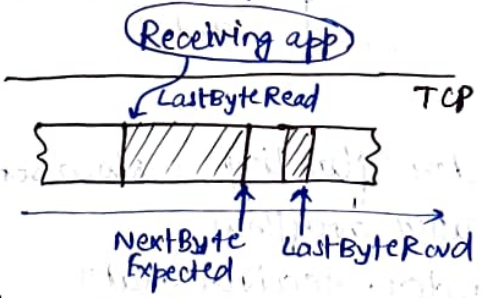
Flow control: sender won't overflow receiver's buffer by transmitting too much, too fast.

Speed matching service: Matching the send rate to receiving app's drain rate.

Sliding window

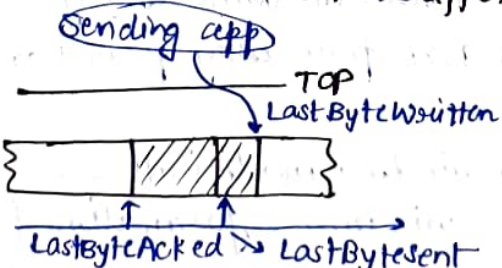
↳ Used originally for ensuring reliability as well as utilization of BW.
↳ can be simply modified to achieve flow control.
↳ Flow control info. is sent over TCP segment from receiver to transmitter.

Receiver:



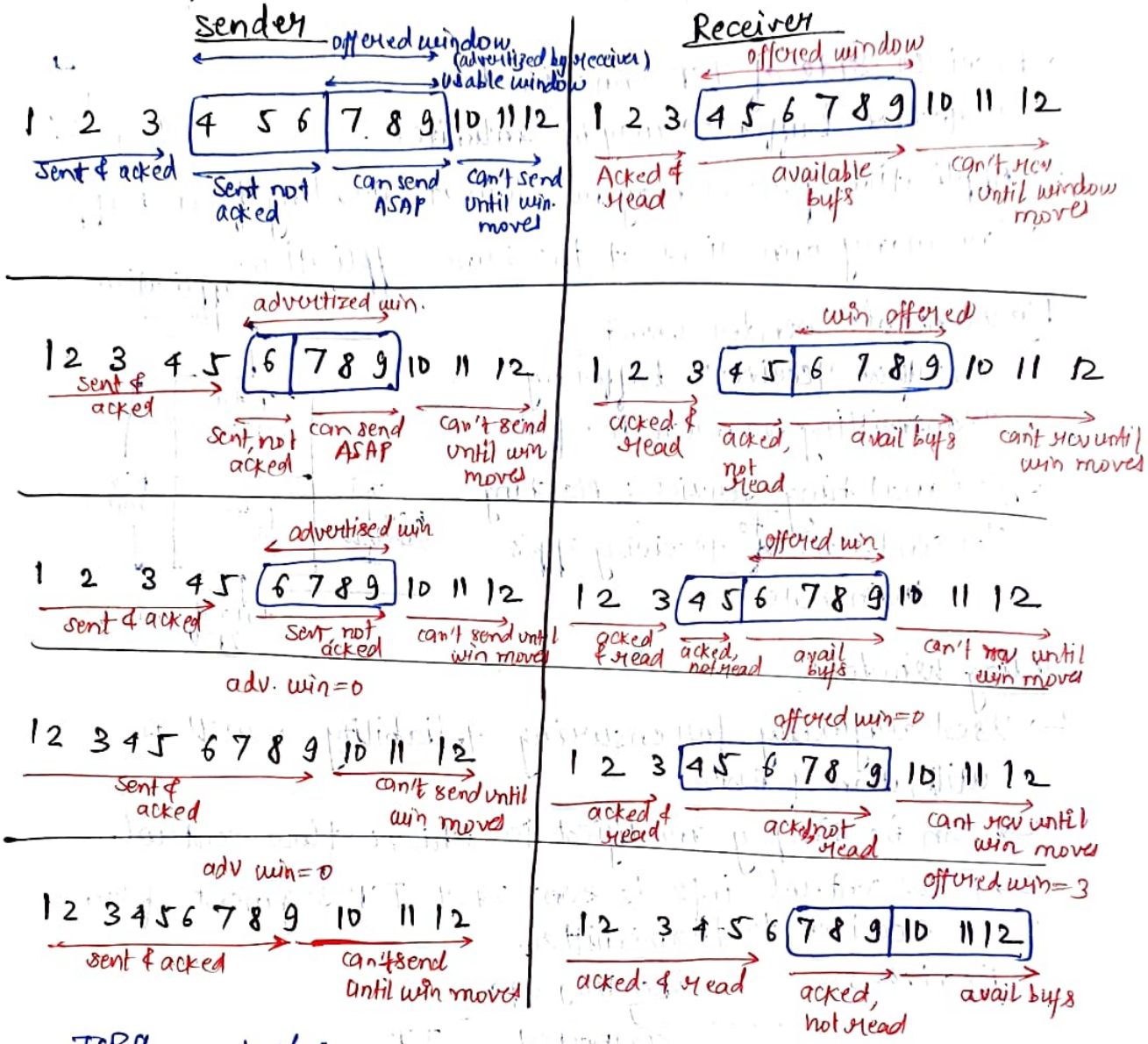
AdvertizedWindow = MaxRcvBuffer − [(NextByteExpected − 1) − LastByteRead]

Sender:



- LastByteSent − LastByteAcked ≤ AdvertizedWindow
- EffectiveWindow = AdvertizedWindow − (LastByteSent − LastByteAcked)
  - EffectiveWindow > 0 for transmitter to transmit.

• (LastByteWritten − LastByteAcked) + y > MaxSendBuffer
  − Else the application is prevented from sending more.



TCP flow control :

→ Avoid a fast sender flooding a slow ~~send~~ receiver.
  ↳ Avoids buffer overflow.
→ Matches the sender-receiver rates
  ↳ Don't result in unnecessary retxn.
  ↳ saves BW, processing power and energy.
→ Achieved by
  ↳ Sliding window modification.
  ↳ Receiver window information fedback to the sender.

TCP RTT estimation and timeout mechanism

↳ Shorter timeout → unnecessary retransmissions (premature timeout).
↳ Longer timeout → lower throughput → slow reⁿ to segment loss.
→ Poorer RTT estimat results in low throughput.

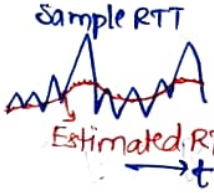
BDP = R × RTT

→ Timeout value: set longer than RTT.
↳ But RTT varies.

RTT estimation:

- sampleRTT → measured time from segment txn until ACK receipt.
  ↳ ignore retxn.
  ↳ will vary, want estimated RTT "smoother".
  ↳ Avg. several recent measurements; not just current sampleRTT.

EstimatedRTT = $(1-\alpha)$*EstimatedRTT + α*sampleRTT



↳ Exponential weighted moving average.
↳ Influence of past sample decreases exponentially fast.
↳ Typical value: $\alpha = 0.125$.

Setting the timeout:

↳ EstimatedRTT + safety margin
  ↳ large variation in EstimatedRTT → larger safety margin.
↳ First estimate how much sampleRTT deviates from EstimatedRTT:

DevRTT = $(1-\beta)$*DevRTT + β*|sampleRTT − EstimatedRTT|

(typically, $\beta = 0.25$)

Then, set timeout interval:

TimeoutInterval = EstimateRTT + 4*DevRTT

Congestion Control

Network congestion: When there is more data offered than the capacity of the network.

↳ Congestion results in deterioration of network performance.

Congestion Control: Control of txn rate of senders in order to prevent congestion from happening in a network.

Principles of Congestion Control:

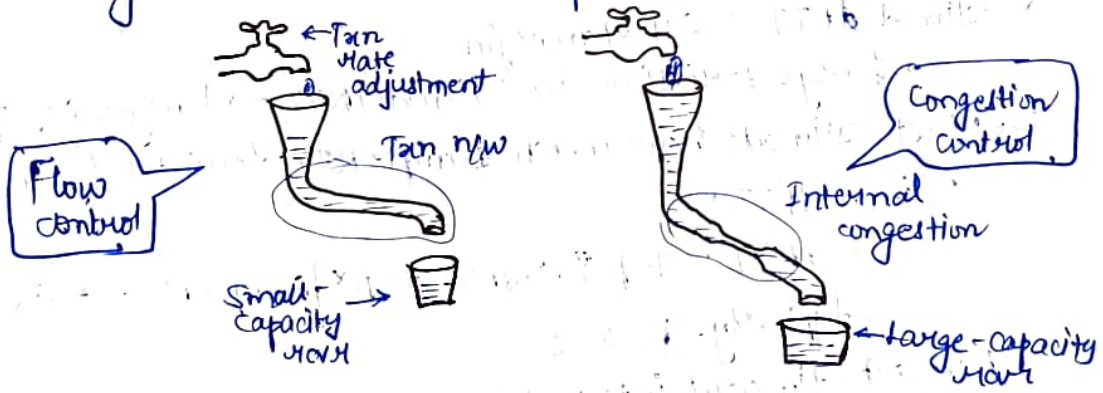
Congestion: Too many sources sending too much data too fast for n/w to handle.

Manifestations:

- Packet loss (buffer overflow at routers)
- Long delays (queuing in router buffers)
- BW waste (due to retxns)
- Energy waste (processing duplicate packets, retxns).

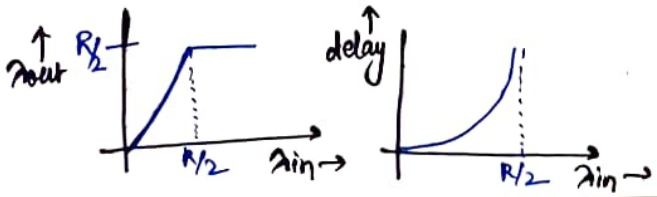
Flow control vs Congestion Control:

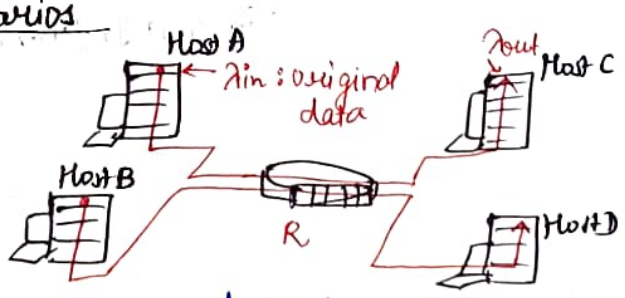
- Flow control is end-to-end session problem affecting only a sender receiver pair
- Congestion control is a n/w problem.



Causes/costs of congestion: scenarios

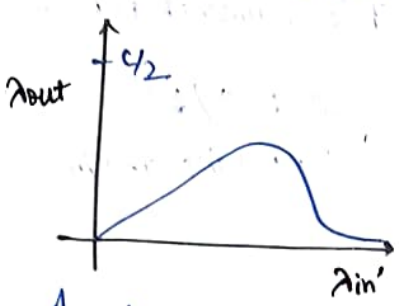
→ Two senders, 2 rcvrs.
→ One router, ∞ buffers.
→ No retxns.

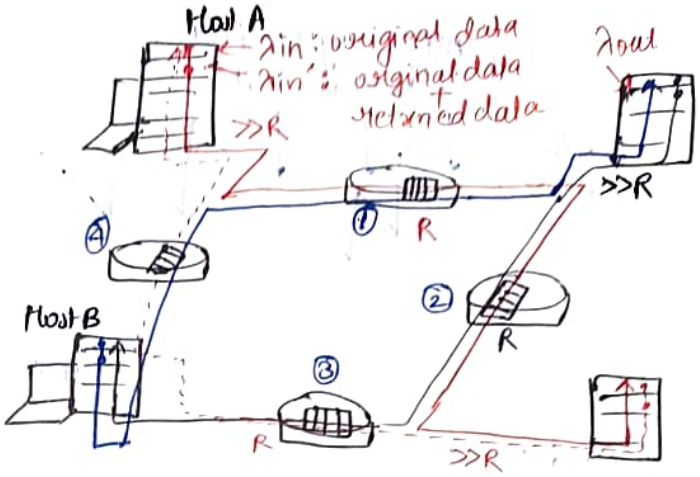




→ Large delay when congested.
→ Maxm achievable throughput.

→ 4 senders
→ Multihop paths
→ timeout/retransmit





Another cost of congestion:
↳ when pkt dropped, any upstream txn capacity used for that pkt was wasted.

Approaches: Towards Congestion Control

End-to-end congestion control:
→ Approach taken by TCP
→ No explicit feedback from n/w.
→ Congestion inferred from end-system observed loss, delay.

Network-assisted congestion control:
→ N/w devices help the sender nodes to recognize congestion.
→ Routers provide feedback to end systems.
- single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
- Explicit rate sender should sent at.

→ Not all routers may extend this help.

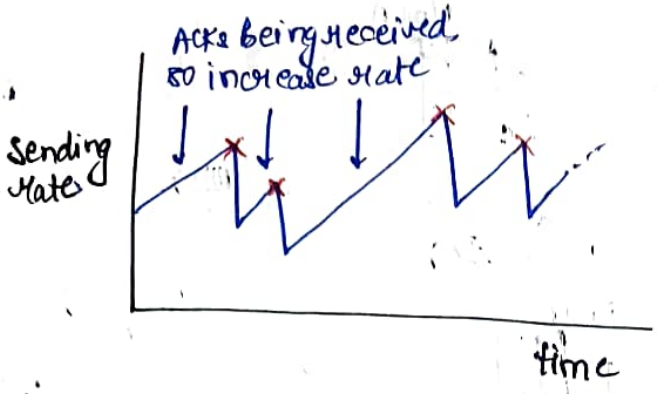
TCP congestion control:

→ TCP sender should transmit as fast as possible to get high throughput, without congesting the n/w.

How to find rate just below congestion level?

→ Each TCP sender sets its own rate, based on implicit feedback: (Decentralized)
- ACK - segment received (a good thing!), - n/w not congested, so increase sending rate.
- lost segment: assume loss due to congested n/w, so decrease sending rate.

BW probing: Increase txn rate on receipt of ACK, until eventually loss occurs, then decrease txn rate.
↳ continue to increase on ACK, decrease on loss (since available BW is changing, depending on other connections in n/w).



X – loss, so decrease rate

→ TCP's sawtooth behavior.

How fast to ↑/↓ ?
↳ Details to follow.

cwnd:

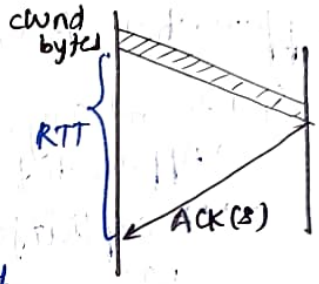
→ Sender limits rate by limiting no. of unACKed bytes in pipeline:

LastByteSent – LastByteAcked ≤ cwnd

→ cwnd: differs from rwnd.

→ sender limited by min(cwnd, rwnd)

Roughly, $\boxed{\text{Tx rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}}$



→ cwnd is dynamic window, function of perceived n/w congestion (or perceived available BW).

→ Three phases for typical TCP congestion control:

ACK received: increase cwnd

(i) slow start phase: increase exponentially fast (despite name) at connection start, or following timeout.

(ii) congestion avoidance: increase linearly.

segment lost: reduce cwnd

(iii) congestion control:

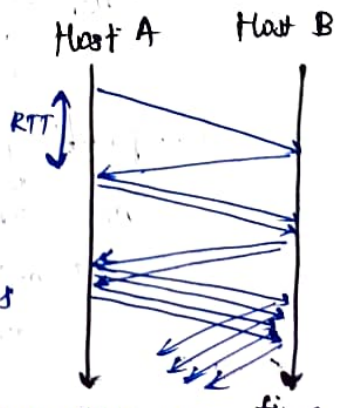
– timeout: no response from receiver
↳ cut cwnd to 1!

TCP slow start:

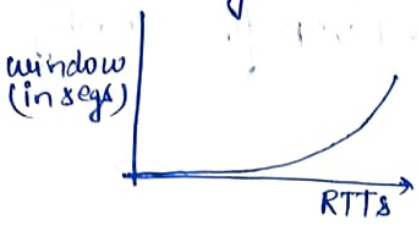
→ when connection begins, cwnd = 1 MSS.

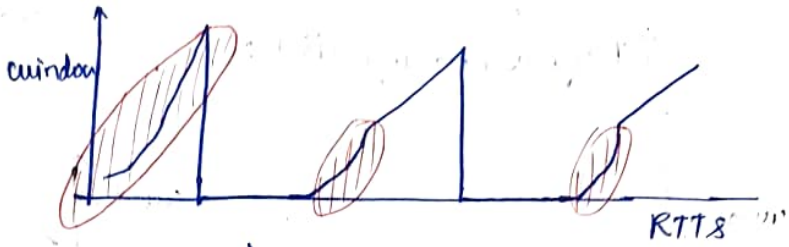
→ Available BW may be >> MSS/RTT
– desirable to quickly ramp up to respectable rate.

→ Increase rate exponentially until first loss event or when threshold reached.
– double cwnd every RTT
– done by incrementing cwnd by 1 for every ACK received.



Exponential growth:





TCP: Congestion Avoidance → slowstart-threshold

When cwnd >> ssthresh, grow cwnd linearly.

- ↑ cwnd by 1 MSS per RTT
- approach possible congestion slower than in slowstart.
- Implementation:

cwnd = cwnd + MSS/cwnd for each ACK received.

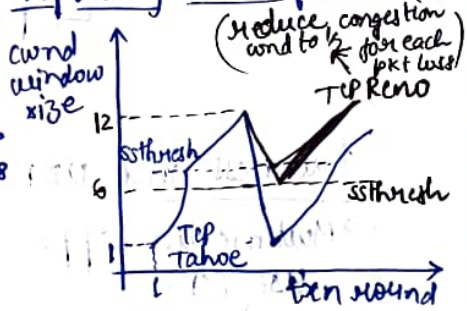
AIMD: Additive Increase Multiplicative Decrease

→ ACKs: ↑ cwnd by 1 MSS. per RTT: additive increase

→ loss: cut cwnd in half (non-timeout-detected loss): multiplicative decrease.

Linear growth:

Popular "flavours" of TCP:

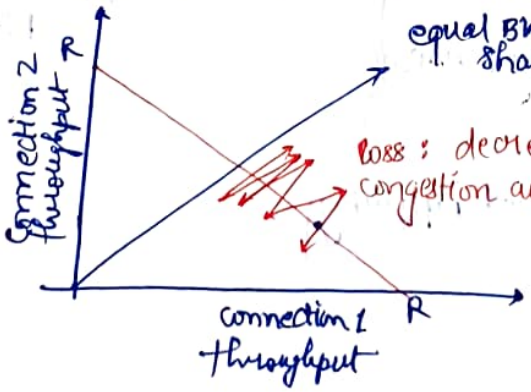


Fairness of TCP: TCP is fair.

Fairness goal: If K TCP sessions share same bottleneck link of BW R, each should have avg. rate of R/K.

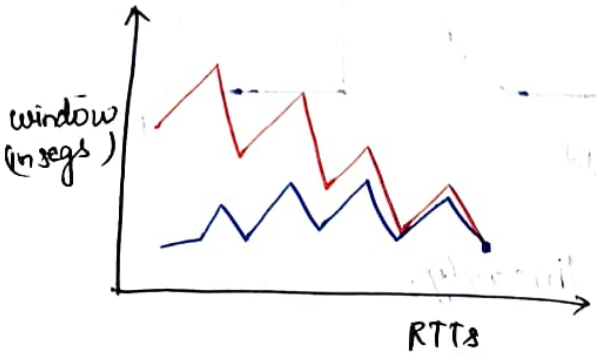
TCP: 2 competing sessions:

- Additive ↑ gives slope of 1, as throughput ↑.
- Multiplicative ↓ decreases throughput proportionally.

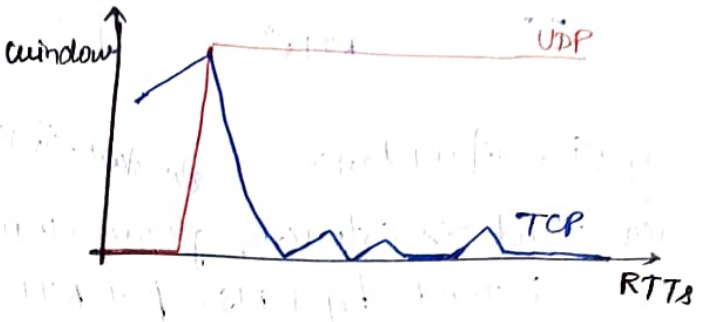


BW sharing with TCP :

2 TCP flows sharing a link :



TCP and UDP flows sharing a link.



Measure of Fairness :

① Standard deviation

② Jain's fairness index

$$f(x_1, x_2, x_3, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

↳ varies b/w 0 and 1.

↳ For a fair system : $x_1 = x_2 = x_3 = \ldots = x_n$

$$\Rightarrow f(\ldots) = \frac{(nx)^2}{n(nx^2)} = 1.$$

Fairness and UDP :

→ Multimedia apps often do not use TCP.
- do not want rate throttled by congestion control.

→ ~~Route~~ Instead use UDP :
- pump audio/video at constant, tolerate packet loss.

→ Today :
- Popular streaming services use TCP.
- N/ws are faster.
- TCP works well with buffering.

Fairness and parallel TCP conn.

→ Nothing prevents app from opening parallel connections b/w 2 hosts.

→ Web browsers do this.

Eg. Link of rate R supporting 9 conn.
- new app asks for 1 TCP, gets rate R/10.
- new app asks for 11 TCPs, gets R/2!

When and when not to design a protocol for fairness?

Fairness is not essential for:
- Proprietary protocols
- Protocols used in limited n/w segments.
- Protocols that do not competing with public n/w traffic.
- Protocols that work over a single link (satellite and GS).
- Protocols working over dedicated channels.

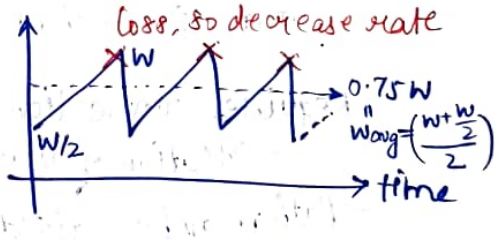
Fairness is important for:
- Public protocols used by large volume of users.
- Protocols in public n/w s.
- Protocols that compete with other traffic.
- Protocols that share resources with other protocols.
- Protocols that work over shared channels.

TCP throughput

Let W be window size when loss occurs ⇒ Throughput = W/RTT.
- Just ~~after~~ after loss, window drops to $W/2$, throughput to $W/2RTT$.
- Average throughput $= 0.75 \frac{W}{RTT}$

Estimation:

Assuming in a cycle, 1 pkt is lost.



$$\frac{W}{2} + \left(\frac{W}{2}+1\right) + \ldots + W$$

$$= \sum_{n=0}^{W/2} \left(\frac{W}{2}+n\right)$$

$$= \left(\frac{W}{2}+1\right)\frac{W}{2} + \sum_{n=0}^{W/2} n$$

$$= \left(\frac{W}{2}+1\right)\frac{W}{2} + \frac{W/2\,(W/2+1)}{2} = \frac{W^2}{4} + \frac{W}{2} + \frac{W^2}{8} + \frac{W}{4} = \frac{3W^2}{8} + \frac{3}{4}W$$

⇒ Loss rate, $\boxed{L = \frac{1}{\frac{3}{8}W^2 + \frac{3}{8}W}} \approx \frac{1}{\frac{3}{8}W^2}$ (since $\frac{3}{8}W^2 >> \frac{3}{4}W$)

$$\Rightarrow \boxed{W \approx \sqrt{\frac{8}{3L}}}$$

$$\therefore \text{Throughput} = 0.75\,W/RTT = \frac{3}{4}\sqrt{\frac{8}{3L}}\,\frac{MSS}{RTT} = \frac{1.22}{RTT}\,\frac{MSS}{\sqrt{L}}$$

Eg 1500 byte segments, RTT = 100ms, wants 10 Gbps throughput
⇒ Requires window size, W = 83,333 in-flight segments
⇒ $L = 2 \times 10^{-10}$ → very small
⇒ New TCP version required (for high speed)

Network Layer

→ Transport segment from sending to receiving host.
→ On sending side, encapsulates segments into datagrams.
→ On receiving side, delivers segments to transport layer.
→ N/w layer protocols in every host, router.
→ Router examines header fields in all IP datagrams passing through it.

Two key Network-Layer Functions:

① **Routing:** Determine route taken by packets from source to dest.
↳ Routing algorithms.

② **Forwarding:** Move packets from router's i/p to appropriate router o/p.

Network Layer Protocols

Connection oriented (ATM)	Connectionless (IP)
- Requires connection setup. - Creation of virtual circuits. - Path may be fixed for the life time of a connection. - Packets may follow the same path during a connection - Routing (switching) is simplified but connection setup is complex.	- Requires no connection setup. - Transfers datagrams over dynamic paths. - Packets belong to the same transport layer session may follow different paths. - Routing is complex.

Connection and connection-less service:

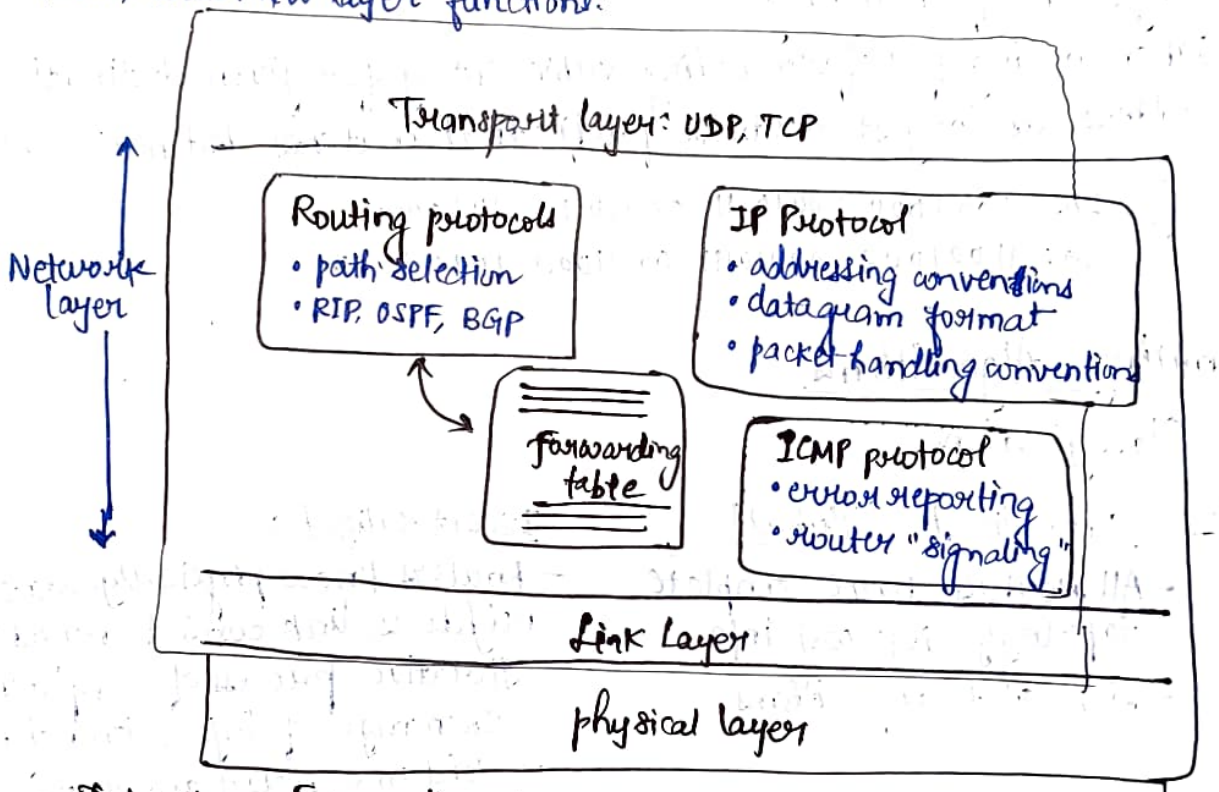
Analogous to the transport-layer services, but:
- service : host-to-host
- no choice : n/w provides one or the other.
- Implementation: in n/w core.

Datagram (connection-less) networks

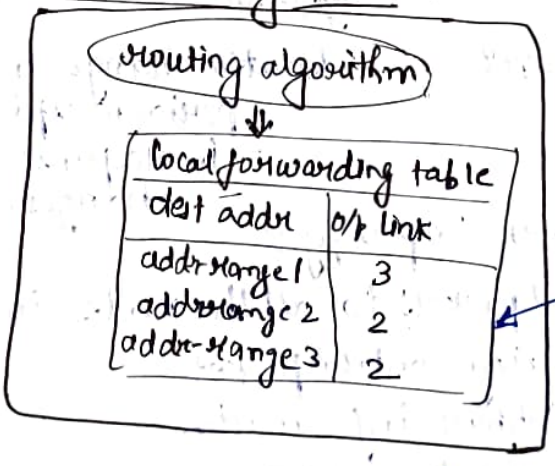
↳ Very popular and growing.
→ No call setup at n/w layer.
→ Routers: no state about end-to-end connections.
- No network-level concept of "connection".
→ Packets forwarded using destination host address.
- Packets b/w same source-dest. pair may take different paths.

Internet N/w Layer:

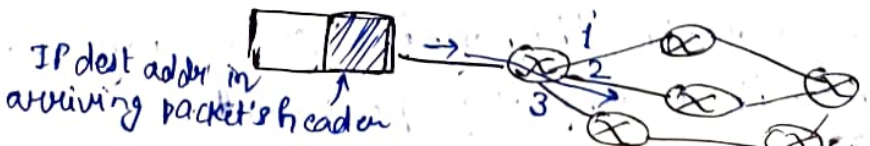
Host, router n/w layer functions:



Datagram Forwarding table:



4 billion IP addresses, so rather than list individual dest. address, list range of addresses (aggregate table entries)



Destination Address Range	Link interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
otherwise	2

Longest Prefix Matching:

When looking for forwarding table entry for given destination address, use longest address prefix that matches destination address.

Eg. DA: 11001000 00010111 00010110 10100001

DA: 11001000 00010111 00011000 10101010

Routing Algorithms

Classification:

A Centralized (Global):
- All routers have complete topology, link cost info
- Link state algorithms

Decentralized:
- Routers know physically-connected neighbors, link costs to neighbors.
- Iterative process of computation, exchange of info with neighbors.
- Distance vector algorithms.

B Static:
- Routes change slowly over time.

Dynamic:
- Routes change more quickly.
  - periodic update
  - in response to link cost changes.

How to build the Network Topology?

→ By exchanging control packets to all the nodes in the n/w.
- Link state routing control packets.
- containing neighbor information.
- Flooded throughout the n/w.

→ Every node receives the neighbor info of every other node.
- Using this, a complete graph is built.

→ Advantages: • optimal shortest path routing
• Global info of the entire n/w available with each node.

→ Disadvantages: • Too much control packet overhead.
• Useful for smaller no. of nodes.

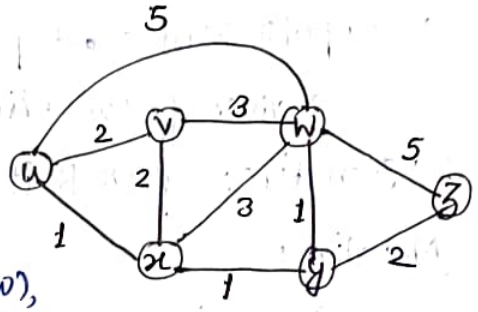
Graph Abstraction of a Network

Graph: $G=(N,E)$

N = set of routers = $\{u, v, w, x, y, z\}$

E = set of links

$= \{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$



↳ Useful in other n/w contexts.

Eg. P2P, where N is set of peers and E is set of TCP conn.

$c(x, x')$ = cost of link (x, x')

Eg. $c(w,z)=5$

↳ could always be 1, or inversely related to BW, or inversely related to congestion.

Cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{p-1}, x_p)$.

~~Rough~~ Routing Algorithm: finds least-cost path.

A Link-state Routing Algorithm : Dijkstra's Algorithm

↳ Net topology, link costs known to all nodes.
- accomplished via "link state broadcast".
- all nodes have same info.

↳ computes least cost path from one node (source) to all other nodes.
↳ Gives forwarding table for that node.

↳ Iterative: after k iterations, know least cost path to k dest's.

Notation:

$c(x,y)$: link cost from node x to y
$= \infty$ if not direct neighbours.

$D(v)$: current value of cost of path from source to dest. v.

$p(v)$: predecessor node along path from source to v.

N': set of nodes whose least cost path definitely known.

Dijkstra's Shortest Path Algorithm:

Initialization:
$N' = \{u\}$
for all nodes v
if v adjacent to u
then $D(v) = c(u,v)$
else $D(v) = \infty$

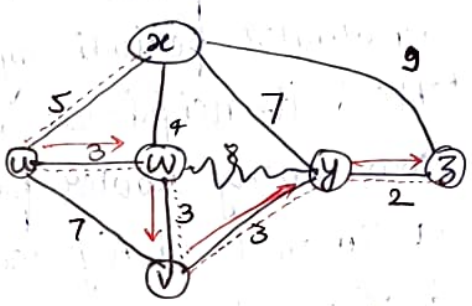
} Initialization part phase

Loop:
find w not in N' such that $D(w)$ is a minimum
add w to N'
update $D(v)$ for all v adjacent to w and not in N':
$D(v) = \min(D(v), D(w) + c(w,v))$
// new cost to v is either old cost to v or known
// shortest path cost to w plus cost from w to v
until all nodes in N'

} Shortest path finding algorithm

Eg.

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwx	6, w			11, w	14, x
3	uwxv				9, v	14, x
4	uwxvy					11, y
5	uwxvyz					



Notes:

- Construct shortest path tree by tracing predecessor nodes.
- Ties can exist (can be broken arbitrarily).
- Path : u–y : 9
  v–y, w–v, u–w
  u–w–v–y

How to represent a Graph:

↳ Simplest way is Adjacency Matrix.
↳ Linked lists, cursor approach, etc.

	U	V	W	X	Y	Z
U	0	7	3	5	∝	∝
V	7	0	3	∝	3	∝
W	3	3	0	4	8	∝
X	5	∝	4	0	7	9
Y	∝	3	8	7	0	2
Z	∝	∝	∝	9	2	0

Distance Vector Routing Approach

→ First routing approach used in ARPANET and Internet.

→ Fully distributed and hop-by-hop routing.

→ Each router maintains a vector (table) indicating its perceived cost (eg; distance or delay) to each router in the network.

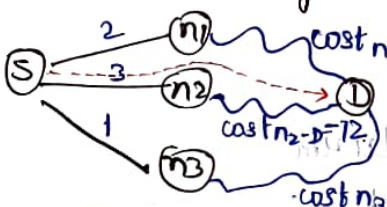
- cost depends on routing metric (hop count, delay, etc.)

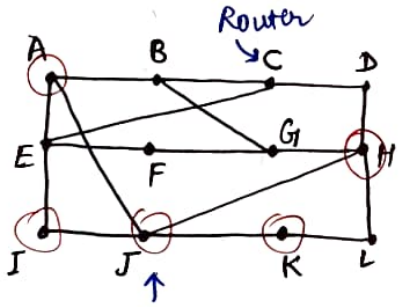
→ Each router periodically or aperiodically send its routing table to its neighbors; also receives one from each neighbor.

→ Now, every router performs an iterative estimation (Bellman-Ford Algorithm) of costs to each of the routers in the network.

$$cost_{S \to D} = \min_{K \in Neighbors} [cost_{S \to K} + cost_{K \to D}]$$

Eg





To	A	I	H	K	Line
A	0	24	20	21	A
B	12	36	31	28	A
C	25	18	19	36	I
D	40	27	8	24	H
E	14	7	30	22	I
F	23	20	19	40	I
G	18	31	6	31	H
H	17	20	0	19	H
I	21	0	14	22	I
J	9	11	7	10	—
K	29	22	22	0	K
L	29	33	9	9	K
	JA	JI	JH	JK	
	delay is 8	delay is 10	delay 14 12	delay 14 6	

J-A : 8
J-I : 10
J-H : 12
J-K : 6

J-A-G : 8 + 18
J-H-G : 12 + 6
J-I-G : 10 + 31
J-K-G : 6 + 31

New routing table for J

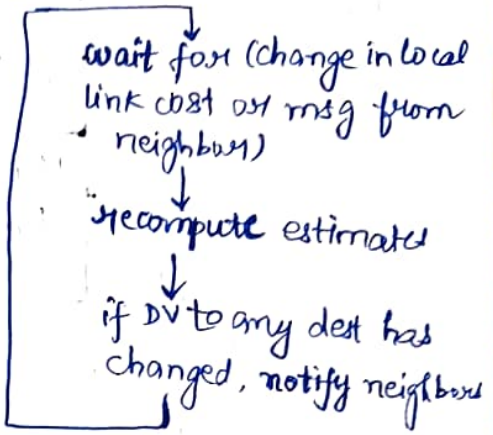
Vectors received from J's 4 neighbors

Distance vector Algorithm:

↳ Iterative, asynchronous : each node iteration called by:
- local link cost change
- DV update message from neighbour

↳ Distributed:
- each node notifies neighbours only when its DV changes.
  - neighbors then notify their neighbours if ~~neig~~ necessary.

Each node:

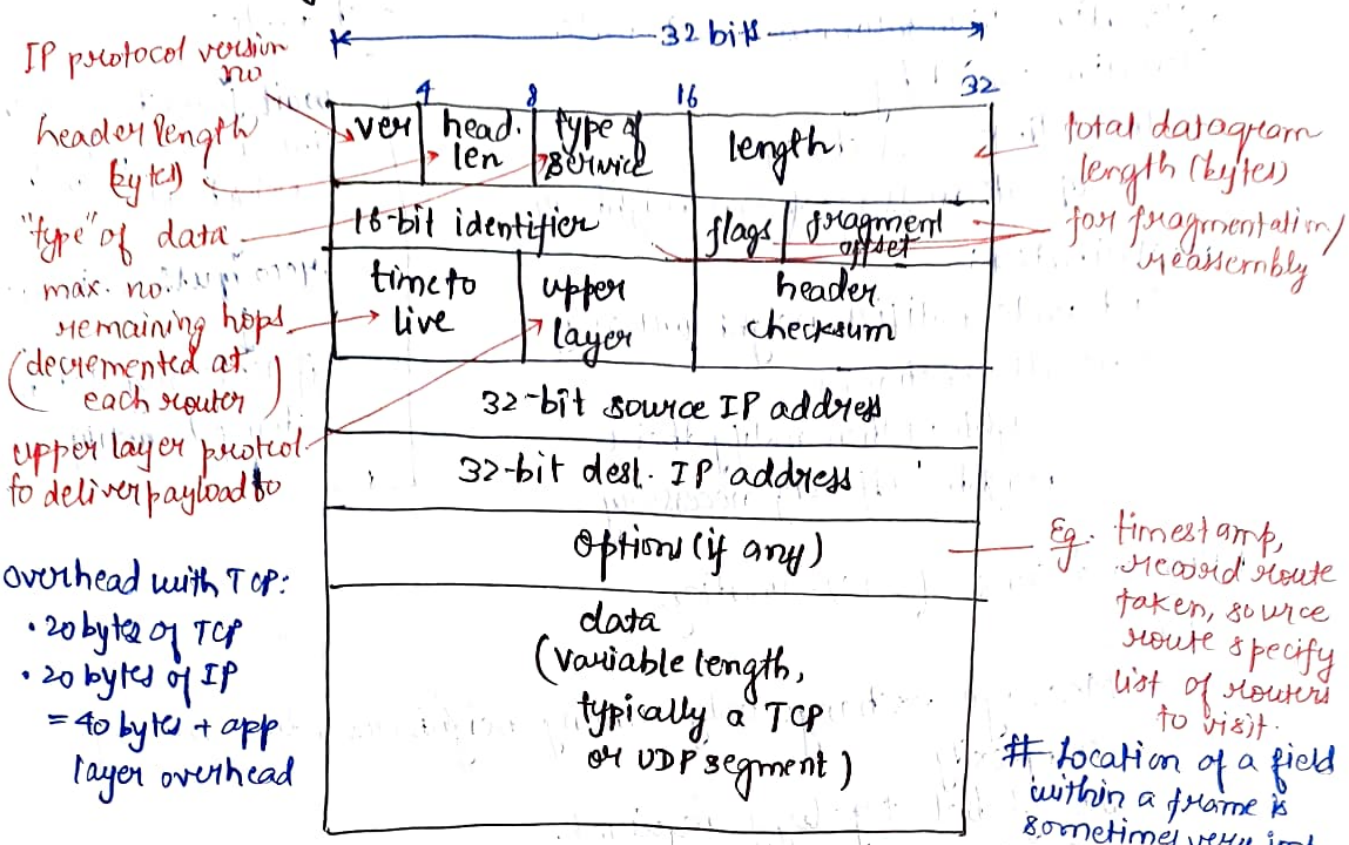


Pros and cons of DV-routing:

Pros:
- Message exchange only among neighbors ; low message complexity.
- No global topology required.
- simple and effective.

Cons:
- Non-optimal routing arising out of slow convergence.
- link cost change may need aperiodic exchanges in addition to periodic DV updates.
- count to infinity problem and routing loop formation.

IPv4 datagram format:



overhead with TCP:
- 20 bytes of TCP
- 20 bytes of IP
= 40 bytes + app layer overhead

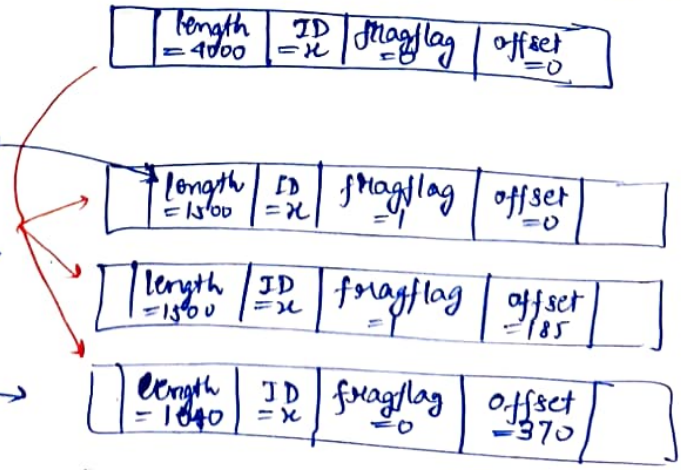
location of a field within a frame is sometimes very imp.

IP fragmentation, reassembly:

→ MTU (Max. Transfer Unit):
- N/w links have ~~highest~~ largest possible link-level frame.
- different link types, different MTUs.

→ Large IP datagram is "fragmented" within n/w layer.
- One datagram becomes several datagrams.
- Reassembled only at final destination.
- IP header bits used to identify, order related fragments.

Eg. 4000 byte datagram
MTV = 1500 bytes



IPv6 : motivation

↳ 32 bits IPv4 address space soon to be completely allocated.
↳ header format helps speed processing / forwarding.
↳ header changes to facilitate QoS.

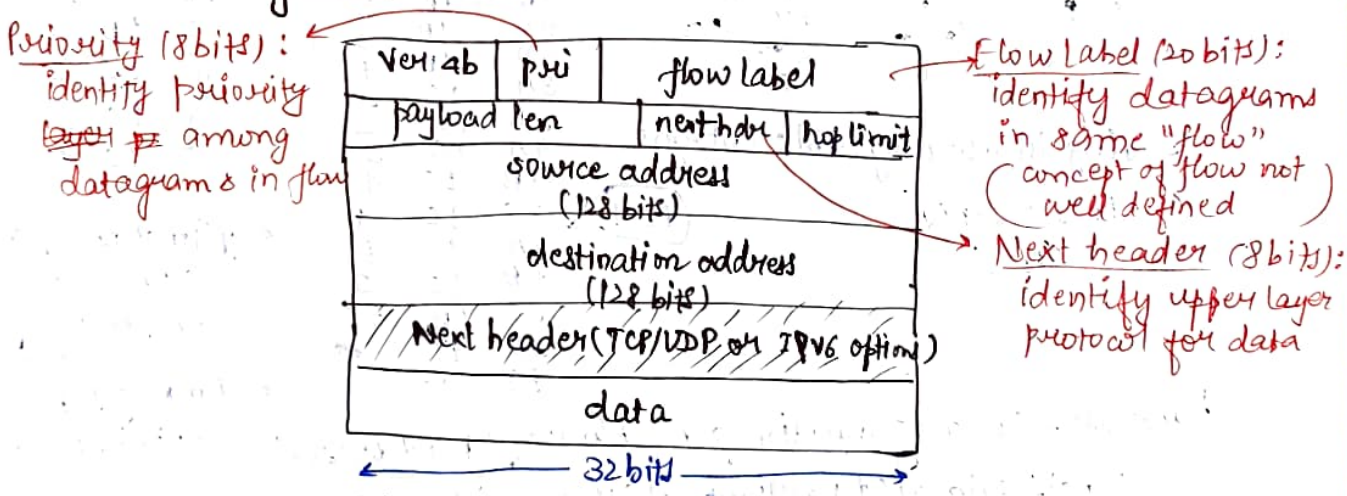
IPv6 datagram format :

↳ fixed-length 40 byte header
↳ no fragmentation allowed.

IPv4 and IPv6 header comparison :

Property	IPv4	IPv6	Benefit for IPv6
Address length	32 bits	128 bits	2^{128} addresses
Simplicity of header	less	better	less complex processing
Header length	variable	fixed	high speed packet processing is possible
Payload length	variable	variable	N/A
Options	Part of IPv4 header	Part of the next header	Fast processing for pkts without next header

IPv6 datagram format :



Hierarchical addressing and routing :

→ Routers are organized into hierarchies
- different regions and hierarchies of regions.

→ The destination (IP) address may have hierarchy and/or region info in addition to dest. host.

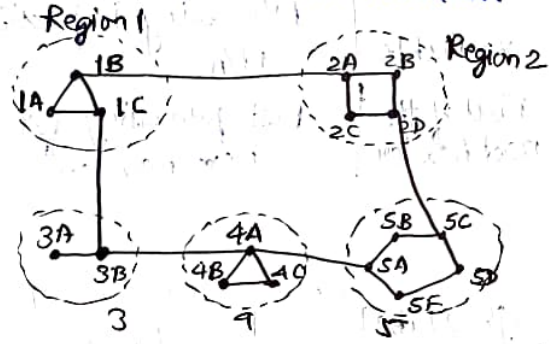
→ Each router knows how to route packets to a dest. within its own region.
- whereas it knows only how to reach other regions.
- Global state info within regions, local state across regions.

Two kinds of addressing:

① Hierarchical addressing:
- Address allocated from a hierarchical space.
- Host (IP) address has two parts:
  ① Network part (subnet part)
  ② Host part.

② Flat addressing: # (port no.)
↳ Address allocated from a flat address space.
- MAC address.

Eg



Saving in routing table entry is

$$\frac{(N-1-e\ln N)}{(N-1)}$$

(N-1)

Full table for 1A

Dest.	Line	Hops
1A	—	—
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

Hierarchical table for 1A

Dest.	Line	Hops
1A	—	—
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

- optimal no. of levels in a hierarchy is $\ln N$, N: no. of routers.
- Routing table entries: $e \ln N$

How does an organization get a block of IP addresses?

↳ Gets allocated portions of its provider ISP's address space

ISP's block: 11001000 00010111 00010000 00000000 200.23.16.0/20

Org 0: 11001000 00010111 00010000 00000000 200.23.16.0/23

Org 1: 11001000 00010111 00010010 00000000 200.23.18.0/23

⋮

Org 7: 11001000 00010111 00011110 00000000 20.23.30.0/23

subnet part | host part

IP addressing : CIDR (Classless InterDomain Routing)

↳ Subnet portion of address of arbitrary length.

↳ Address format : a.b.c.d/x, where x : no. of bits in subnet portion of address.

Subnets :

↳ A segment of a n/w that share a common n/w address part.

↳ Device interfaces with same subnet part of IP address.

↳ can physically reach each other without intervening router.

IP addr: - subnet part : high order bits

- host-part : low order bits.

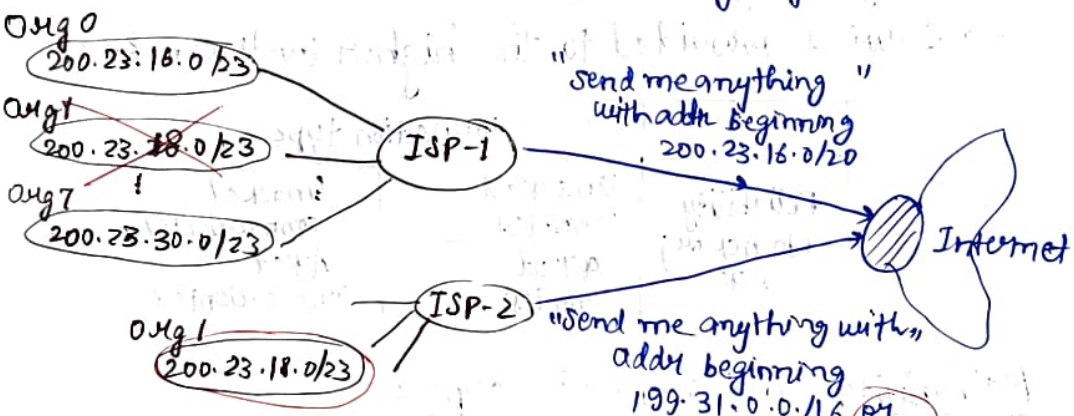
Eg. IP addr : 172.20.42.55

Netmask : 255.255.255.0

Result : 255.255.255.255

Hierarchical addressing:

Route aggregation : Hierarchical addressing allows efficient advertisements of routing information.



ISP-2 has a more specific route to org 1

Link Layer

→ Host and routers are nodes.
→ comm. channels that connect adjacent nodes along comm path are links (wired, wireless, LANs).
→ layer-2 packet is a frame, encapsulates datagram.

Data Link layer:

↳ Has responsibility of transferring datagram from one node to physically adjacent node over a link.
↳ Primary objectives:
- to provide a well-defined service interface to the n/w layer.
- link layer transmission (mostly covered in digital comm)
- link level error control (detection/correction)
- link level flow control (similar to transport flow control)
- channel access control (MAC)
- framing.

→ Services provided to the higher levels can be:

	connection type	
Reliability (No Ack or) Ack	Unacked conn-less	Unacked conn-oriented
	Acked conn-less	Acked conn-oriented

Reliability and connection type:

Reliability:

① Unacked services:
- Transmit and forget
- Higher layers are responsible for reliable delivery.
- Some applications may not need reliable delivery.
  - eg. voice over IP
  - Delivery time is more important: delay is worse than loss.

② Acknowledged services:
- Receiver replies with an ACK packet upon successful receipt of a pkt.
- sender retransmits if it either receives a negative ACK or when it does not receive an ACK within a specified time duration.

Connection-type:

① connection-oriented:
- Logical conn. setup and release operations.
- state variables maintained.
- Expensive (in terms of resources) and complex.

② Connectionless:
- No conn. setup are required.
- No state info is maintained.
- Inexpensive and light weight protocol implementation.

Application scenarios

→ A conn-oriented service is chosen if:
- sophistication of services is required
  eg→ call admission control, guaranteed services, in-order delivery, quality of service.
- channel is not reliable.
- we have enough resources.

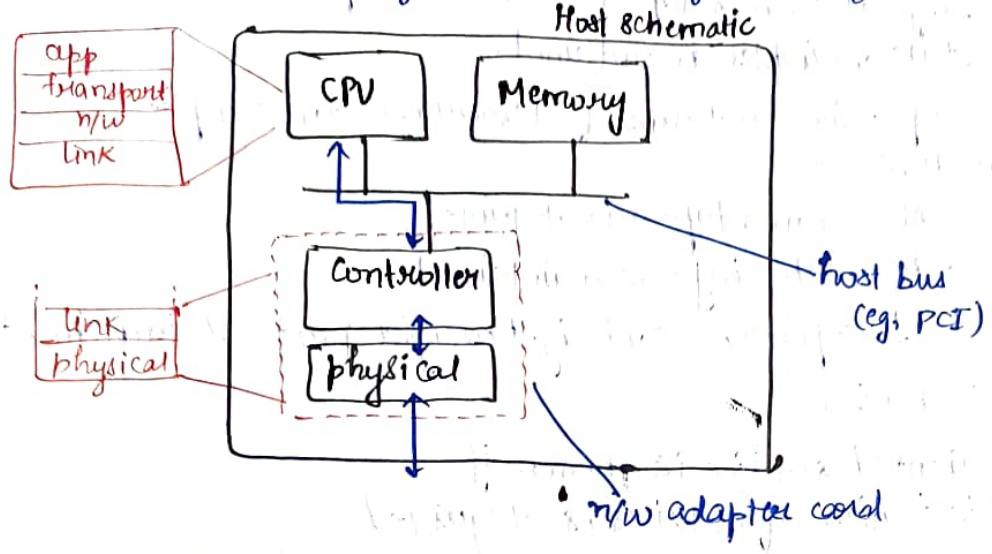
→ A conn-less service is preferred when:
- we need only best-effort service.
- channel is highly reliable.
- if we have resource constraints.
- if we need a light-weight implementation.

DLL Implementation:

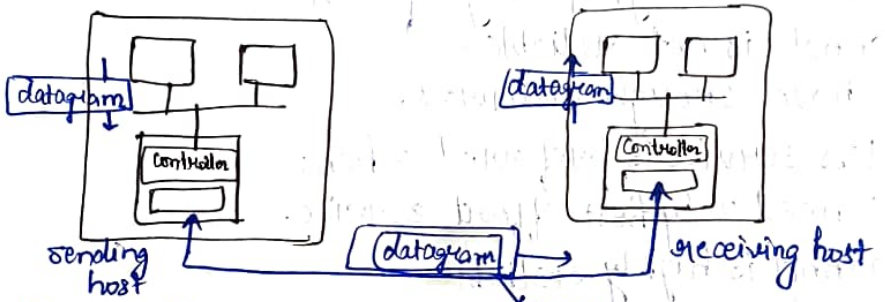
→ Higher layer implementations:
- Ordinary systems
  ↳ Higher layers are implemented fully in software.
    - kernel space
    - User space.
- Sophisticated high capacity systems.
  ↳ Either in software or hardware.
  ↳ Preferably in hardware accelerated TCP/IP engines are used for data centers/high capacity servers.

→ Implemented in:
- each and every host
- in adaptor (aka Network interface card or NIC)
  ↳ Ethernet card, PCMCI card, 802.11 card
  ↳ implements link, physical layer.

- attaches into host's system buses.
- combination of hardware, software, firmware.



Network Interface Cards (Adaptors) Communicating:



Sending side:
- encapsulates datagram in frame.
- adds error checking bits, rdt, flow control, etc.

Receiving side:
- looks for errors, rdt, flow control, etc.
- extracts datagram, passes to upper layer at receiving side.

Sub layers in DLL:

In shared channels, DLL has two sub layers:

① Logical Link control (LLC) sublayer (eg, IEEE $802_{2.2}$)
- Responsible for framing, error control and flow control.
- Multiplexing packets for txns.
- De-multiplexing packets when receiving.

② Medium Access Control (MAC) sub layer (eg, IEEE 802.3)
- channel access policy and txn opportunity.

Medium / Link / Channel Classification:

① Dedicated
- dedicated for one sender-receiver pair
- point-to-point
- wired, wireless or optical.
- sender can transmit as and when it wants.

② Shared
- the txn medium is shared among a no. of nodes.
- can be a broadcast channel.
- can be a ring-like physical layer.
- can be either wired, wireless or even optical.
- sender needs to contend with other nodes to obtain a txn opportunity.
  - Complex MAC protocols are necessary.

~~DLC Responsibility~~

Types of Links:

① Dedicated (point-to-point):
- PPP for dial-up access.
- point-to-point link b/w Ethernet switch and host.

② Broadcast (shared wire or medium):
- old-fashioned Ethernet
- upstream HFC
- 802.11 wireless LAN.

A simple classification:

① Static channel Allocation, channel Partitioning or Contention Free MAC protocols:
- share channel efficiently and fairly at high load.
- inefficient at low load: delay in channel access, 1/N BW allocated even if only 1 active load.

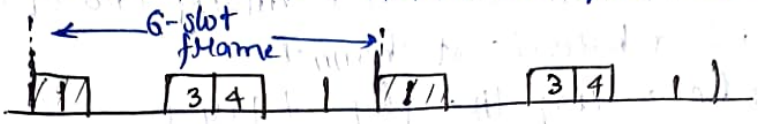
② Dynamic or random access MAC protocols:
- efficient at low load: single node can fully utilize channel.
- high load: collision overhead.

③ Token based or "taking turns" protocols:
- work for best of both worlds!

Contention-Free MAC protocols:

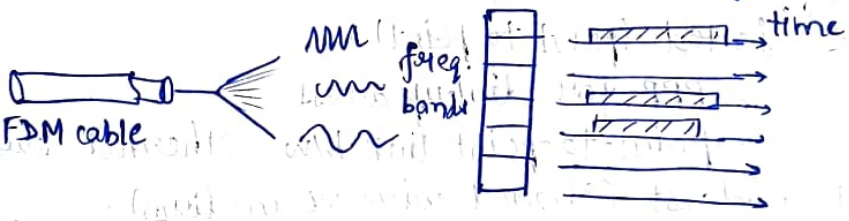
TDMA: Time division multiple access
- access to channel in "rounds" (or round-robin).
- each station gets fixed length slot (length = pkt txn time) in each round.
- unused slots go idle.

Eg. 6 station LAN; 1,3,4 have pkt; slots 2,5,6 idle.



FDMA: Frequency division multiple access
- channel spectrum divided into frequency bands.
- each station assigned fixed frequency band.
- unused txn time in freq. bands go idle.

Eg. 6-station LAN; 1,3,4 have pkt; frequency bands 2,5,6 idle.



Random Access Protocols

→ When node has packets to send
- transmit at full channel data rate R.
- no a priori coordination among nodes.

→ Two or more transmitting nodes → collision.

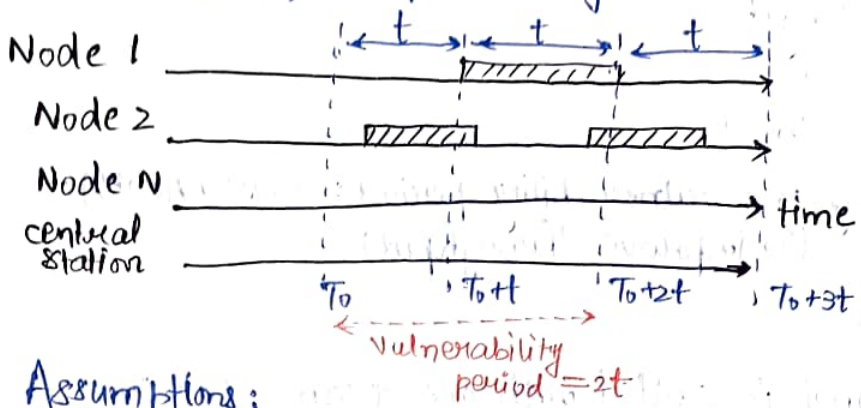
→ Random access MAC protocol specifies:
- how to detect collisions.
- how to recover from collisions (eg, via delayed txns).

↳ Eg. ALOHA, slotted ALOHA, CSMA, CSMA/CD, CSMA/CA.

ALOHA (meaning Hello)
↳ Came into operation in 1970.
↳ A node transmits when it has data to transmit.
↳ collision can be detected.
↳ Expensive
↳ Depends on the size of the n/w.

→ Vulnerability period of a given packet decides the probability of collision.

↳ Here, 2 x packet length is the vulnerability period.



Assumptions:

- Infinite population.
- Frame rate, $0 < N_F < 1$.
- Frame rate, $N_F > 1$ refers to extremely high load where channel cannot handle the traffic.
- Poisson Arrival (including new and returns) with G arrivals per frame time ($G \geq N_F$).

→ Throughput (info transferred over the system):

offered load x Probability of success = $G \times P_0$,

P_0 : probability that the frame does not suffer collision.

→ A frame will not be colloided, if no txn takes place within the vulnerable period (2t).

→ Probability of k frames generated during a given frame time is $P_k = \frac{G^k e^{-G}}{k!}$.

→ Probability of no other frames txned, $P_0 = e^{-G}$

→ Probability that no other frames are generated in vulnerable period, $P_0 = e^{-2G}$.

Throughput (S) = $G \times P_0 = G e^{-2G}$.

↳ Max when $G = 0.5$

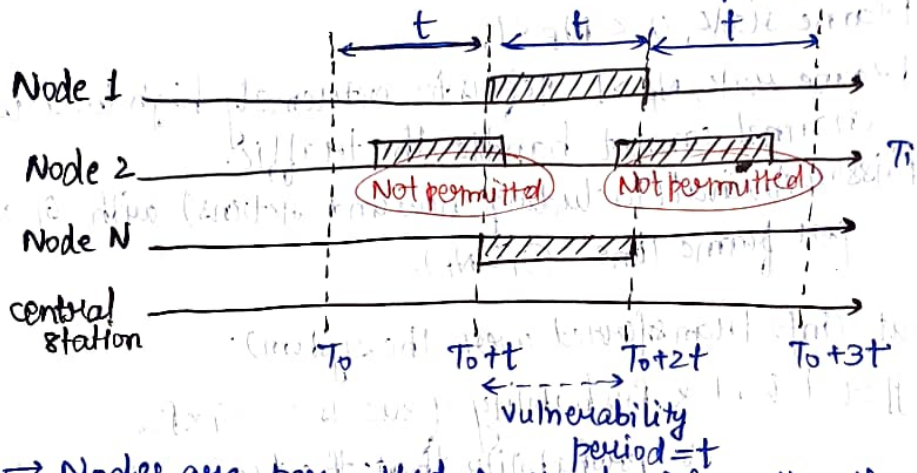
↳ Max throughput = $1/2e = 0.184 = 18.4\%$.

→ Factors affecting the throughput:
- vulnerability period
- Random txn
- packet length
- No. of nodes
- Traffic load.

→ Try reducing the vulnerability period: Slotted ALOHA
(To improve throughput)

Slotted ALOHA

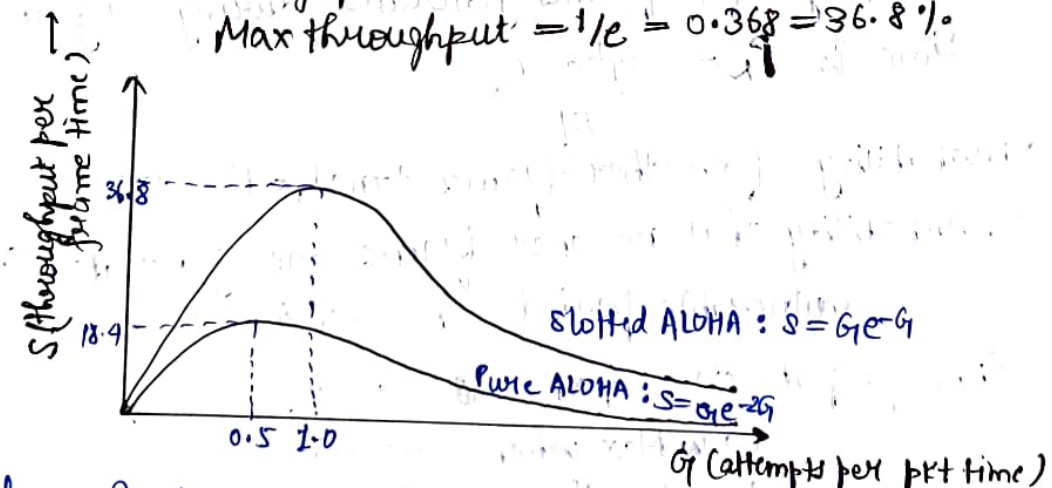
↳ The channel is slotted with one frame per slot.
↳ Central station initiates a periodic slot signal.



→ Nodes are permitted to start right after the signal.
→ The probability of no other txn in vulnerability period is $P_0 = e^{-G}$.

Throughput $(S) = G \times P_0 = G e^{-G}$

Max throughput $= 1/e = 0.368 = 36.8\%$



Slotted ALOHA: Probability of collision $= 1 - P_0 = 1 - e^{-G}$

Expected no. of retxns $= eG$

Eg. A group of N stations share a 56 Kbps ALOHA channel. Each station outputs 1000 bit frame on average of once every 100 sec, even if the previous one has not yet been sent (eg, the stations can buffer out going frames). What is the max^m value of N?

Soln: Each station: 1 frame/100 sec. = 1000 bit/100 s = 10 bps
With ALOHA, efficiency = 0.184 ⇒ 56 Kbps × 0.184 = 10.3 Kbps
∵ Each node requires 10 bps,
So $N = \frac{10.3 \text{ Kbps}}{10 \text{ bps}} = 1030$ nodes.

↳ At very light load, ALOHA is better than slotted ALOHA, as nodes need not wait till the beginning of the next slot.

Eg. Ten thousand airline ~~tickets~~ reservation stations are competing for the use of a single slotted ALOHA channel. The avg. station makes 18 requests/hour. A slot is 125 μsec. What is the approximate total channel load, G?

Soln: N = 10000
Avg. reqs/hour = 18 = 0.005/s.
∴ Total channel offered load = 10000 × 0.005/s
= 0.0625 /time slot.

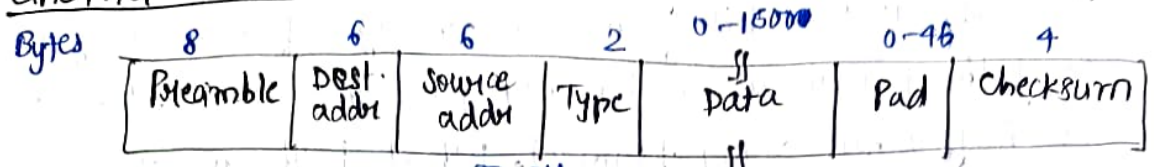
Carrier sense Multiple Access with collision Detection (CSMA/CD)

↳ Carrier sensing, deferral as in CSMA.
- collisions detected within short time.
- colliding txns aborted, reducing channel wastage.

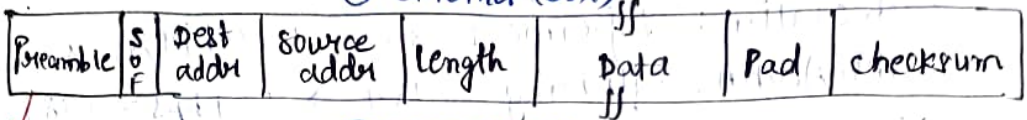
→ collision detection:
- easy in wired LANs: measure signal strengths, compare trned, received signals.
- difficult in wireless LANs: received signal strength overwhelmed by local txn ~~signal~~ strength.

[Human analogy: polite conversationalist]

Ethernet Frame Format:



(a) Ethernet (DIX)



(b) IEEE 802.3.

Preamble: Sequence of 10101010
↳ Manchester coding of this pattern produces a 10-MHz square wave for 6.4 μsec to allow the receiver's clock to synch with sender's

#DIX stands for consortium formed by DEC, Intel and Xerox.

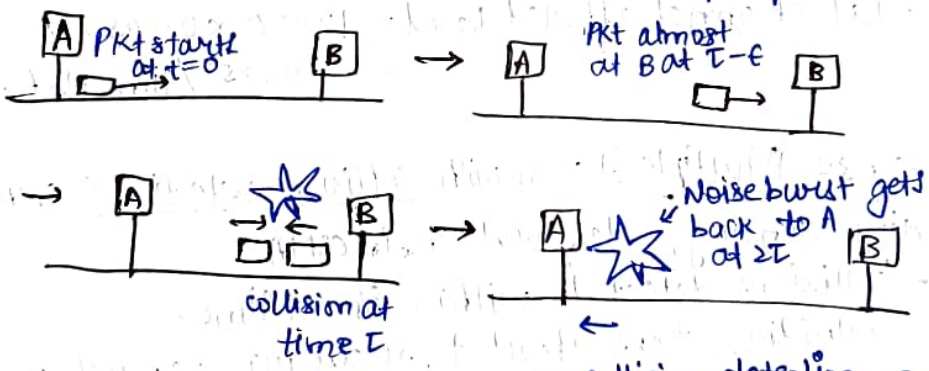
SOF: 10101011

SA/SD: 6 bytes (3 bytes of org ID, 3 bytes org provided addr)

Type: Kind of data in the frame (0x0800 for IPv4 packet)
↳ Any no. ≤ 0x600 (1536) can be interpreted as length, and any no. ≥ 0x600 ↳ Type

Wireshark cannot capture full DLL Header without additional hardware/driver support.
— MadWiFi driver in Linux can help capture the WiFi header

e.g.



↳ collision detection can take as long as 2T.

→ A given node senses the channel before txn.
↳ If the channel is not free, it waits for the channel to be idle for at least 96 bits.
→ If the channel is free for 96 bit duration, it transmits.
↳ When transmitting, signal on channel is checked.
↳ If the rcved signal differs from txned signal: event of collision.
↳ Transmits a jam signal for 48 bits duration and stops txn.
↳ Backs off exponentially for another txn.

↳ **Binary exponential back-off**

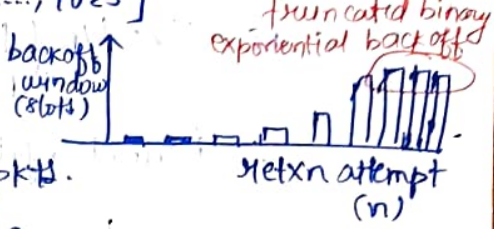
↳ A given node backs-off a uniform random no. (M) of slots (512 bit width) b/w $[0, 2^m - 1]$ where $m = \min(n, 10)$, n = retxn count.

$n=1$: $M = [0,1]$

$n=2$: $M = [0,1,2,3]$

$n=10$: $M = [0, 1, 2, \ldots, 1023]$

→ The back-off window variation.



→ Min^m frame length is 64 bytes.

- Padding used to fill shorter pkts.

→ Min^m pkt. length should provide a txn time $\geq 2 \times T_{prop}$

- with a min^m pkt size of 64 bytes, the Ethernet (10Mbps) can have a max^m length of 2500 m (including a max^m of 4 repeaters).
- Min^m packet duration = 51.2 μsec.

Performance Analysis of CSMA/CD:

Assumptions: • N stations.

• p: probability that each station transmits during a contention slot.

step ①: Probability that some node successfully acquires channel in a given slot $= A = Np(1-p)^{N-1}$

A is max when $p = 1/N$

where $A \to 1/e$ as $N \to \infty$.

step-②: Probability that contention interval has exactly j slots $= (1-A)^j A$

Expected no. of slots per contention (geometric distribution)

$$= \sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}$$

1 fail | 2 fail | 3 fail | j fail | Success

backoff win. exponentially ↑ on an avg with retxns.

Each slot has a duration of $2t$.

Data txn time $= T_x$.

Expected contention utilization $= \frac{2t}{A}$

Channel efficiency (utilization) $= \frac{T_x}{T_x + 2t/A}$

As $T_x = L/R$ (L: frame length, R: BW),

$\therefore$ Channel efficiency $= \frac{1}{1 + 2tR/AL}$