Ch - 1

→ Access Networks

1. Cable Based: Freq Div Multiplexing.
2. Digital Subscriber Line Access Multiplexer: data → internet voice → telephone net both at diff freq over same DSL line.
3. WLAN: like Wifi router.
4. Wide Area Cellular Access Network: provided by mobile, cellular network operator (4G/5G)
5. Home Network: Cable/DSL modem, router + wired Ethernet / Wifi access point.
6. Institutional / Entreprise network
7. Data Centre Networks: high bandwidth links

→ Radio comm. types WLAN (Wifi), Wide Area, Bluetooth, satellite, terrestrial microwave

→ Sending / Receiving

R = transmission rate / link capacity / link bandwidth

L = size of packet Trans. Delay = L/R

- Forwarding ≡ Switching (Local Action)

- Routing (Global action, routing algos)
- # hops = # routers b/w src and dest^n
- Packet switching: store & forward + queuing
- Circuit switching: FDM and TDM
- $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$
  $= L/R \quad = d/s$
- traffic intensity $= \frac{La}{R}$
  - 0 queing delay small
  - ~1 large
  - >1 infinite
- Bottleneck Link: link on end-end path that constraints end-end throughput (smaller rate one)
- Packet sniffing: other host reads/records all packets b/w some host/dstn
- IP Spoofing: injection of packet with false src address
- DoS: make resources unavl to legit traffic by overwhelming resource with bogus traffic

→ Layered IP stack

message	application	HTTP, IMAP, SMTP, DNS
segment	transport	TCP, UDP
datagram	network	IP, routing protocols
frame	link	Ethernet, 802.11 (Wifi), PPP
	physical	(bits)

switch: link
physical

router: network
link
physical

Ch 2

- **Client – server**: always on host, waits for client to contact it. client initiates

server: permanent IP
client: dynamic IP, intermittent connect"
do not comm. with each other

- **P2P**: no always on host, hosts directly communicate. self scalability (new peers cater to + request for service)
have client + server process

IP addr: unique 32 bit, host id.
Port No: process id. HTTP (80) mail (25)

→ TCP: reliable, flow control, congestion control, connection oriented
NOT give: timing, security, min throughput guarantee

TLS over TCP for security.
↳ implemented in appl layer

→ **HTTP**

— uses TCP
client initiates TCP to server, port 80
server accepts. request, response object in body.

— stateless
maintains no info abt past client requests

— Non Persistent HTTP
TCP conn opened
atmost one obj sent over TCP conn
TCP conn closed.
(multiple files download → multiple conn)

Non-Pers. resp time = 2 RTT + file transm. time
per obj
2 RTT per obj ⟶ do in parallel
OS overhead for each TCP conn

— Persistent HTTP
multiple obj can be sent over single TCP conn b/w client & server

Pers. resp time = RTT + file transm time
per obj
+RTT for estab conn

→ Cookies

managed by browser
authorisation, recommendations
user session state

kept on endpoint hosts
carried by HTTP messages

first party cookie

→ Web Caches / Proxy Servers

user configures browser to point to a local Web cache, browser sends all HTTP req to cache.

- if obj in cache: cache ret obj to client
- else cache req obj from origin server, then ret obj to client

act as both client and server

server tells cache about obj's allowable caching in response header.

Cache-Control: max-age = <sec>
: no-cache

→ Browser Caching : Cond^n Get

if present & not modified since ⇒ give
else fetch from server, cache it, give

HTTP req msg : if-modified-since: <date>
HTTP resp : HTTP/1.0 304 Not Modified

HTTP resp : HTTP/1.0 200 OK <data>

→ HTTP 1.0 V/s 1.1

1.0	1.1
Non Persistent Browser Caching slow	Persistent No browser caching faster.

HTTP 1.1 :

introduced multiple pipelined GETs over

introduced multiple, pipelined GETs over single TCP conn. FCFS: HOL Blocking
loss recovery (retransmission): stalls objs.

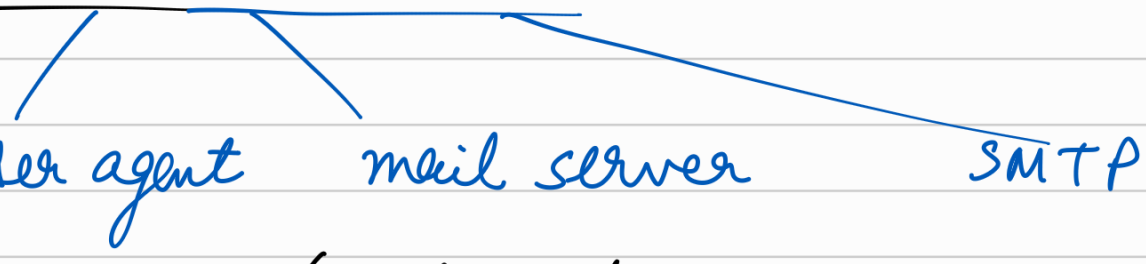
HTTP 2.0:

- transmission order - client specified priority. (Not FCFS)
- push unrequested obj to client
- divide objs into frames, schedule frames to mitigate HOL blocking
- recovery from packet loss still stalls all obj transmissions

HTTP 3.0

adds security, per obj error and congestion control (more pipelining) over UDP.

→ ## Email



- user agent/mail reader
composing, editing, reading outgoing/incoming msgs stored on server
- SMTP
uses TCP to reliably transfer email to server, port 25
① TCP handshake
② SMTP handshaking 220, 250

(2) SMTP transfer of messages 354
(3) SMTP closure 221

HTTP : client pull
SMTP : client push

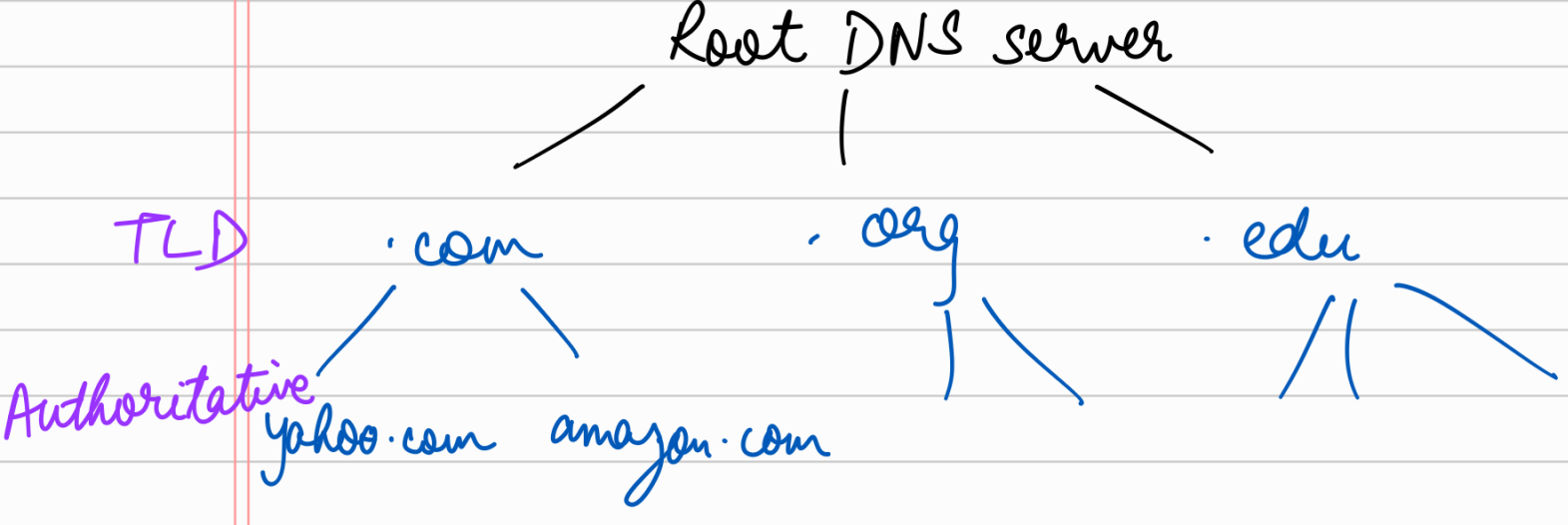
- IMAP : messages stored on server. IMAP — retrieval, deletion, folders etc.

HTTP :
SMTP (send) IMAP (or POP) to retrieve or HTTP from server

→ Domain Name System

- appl. layer protocol
- distributed database
- name ⟶ IP addr mapping
- host aliasing (canonical, alias names)
- mail server aliasing
- load distribution

replicated Web servers — many IP addrs correspond to one name



Root Servers : 13 logical, replicated

TLD :
all country domains : .us, .in, .jp
all other domains : .edu, .org, - - -

Authoritative :
- Organisation's own DNS server : provides hostname - IP mapping for org's named hosts.
- maintained by org or service provider

local DNS servers
- each query goes to local DNS server.
- returns from recent name-address transl^n cache (maybe out of date)
- fwd req to DNS hierarchy for resolution

Iterative :
local DNS server performs each query.

Recursive :
puts heavy load at upper levels of hierarchy.

→ Peer-to-Peer (P2P)

N peers, file size F :

client server: $t \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$

P2P: $t > \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \Sigma u_i}\right\}$

→ Bittorrent

- file divided into 256 Kb chunks
- peers in torrent send/recv file chunks
- tracker: tracks peers participating in torrent
- torrent: grp of peers exchanging chunks of a file.
- request rarest chunk first.

- Send: tit for tat.
  - send to 4 peers currently sending u at highest rate.
  - re-evaluate top 4 every 10 sec.
  - every 30 sec: randomly select new peer (optimistically unchoke)

→ Video Streaming and CDNs

- use redundancy within and between img to dec # bits to encode img. spatial, temporal.
- CBR: (const bit rate): video encoding rate fixed
- VBR: variable.

client side buffering and playout delay. compensate for network added delay, jitter

- DASH (Dynamic Adaptive Streaming over HTTP)

server:
video file ← chunk 0, chunk 1, : } each chunk encoded at multiple rates and diff encodings stored in files at diff CDN nodes

manifest file: URL for diff chunks

client:
estimates bandwidth & req. one chunk at a time of max possible coding rate

i) when to request (no buffer starvat" / overflow)
ii) what encoding rate (acc to avl bw)
iii) where to req (URL server closer / has high bw)

stream video: encoding + DASH + playout buffer"

→ Content Distribution Network

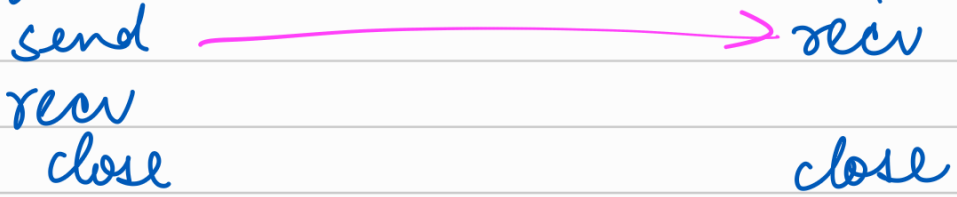
multiple copies at multiple geo sites

enter deep: push CDN servers deep into many access networks
bring home: small no. of larger clusters in POPs near access nets
(points of presence)

OTT: host-host comm as a service

→ Socket prog

Server:	client
get adder Info	get adder Info
socket (fd)	socket
bind (port)	
listen	
accept ←	connect
recv ←	send


send
recv
close
recv
close