Chapter-1

-> The Internet

-> The internet: Nuts and bolts :-

:) Diff hosts on edge.
:) Routers and switches
:) Wire connecting them
:) Groups of local networks interconnected - Network of networks

* Protocols -> Rules of sending and receiving messages on internet.
* Internet standards -> RFC, IETF - Internet Engineering Task Force

-> The internet: a services view
:) Infra that provides services to apps

---

* The network edge :- (1.2)

:) How to connect end systems to edge routers?

-> Residential access nets
-> Institutional access networks (school, company)

-> Mobile access network (wifi, 4G/5G)

:) Cable Access Network - works on FDM.
Downstream - 40 Mbps to 1.2 Gbps
Upstream - 30-100 Mbps

:) DSL -> Use existing telephone line
Downstream - 24 - 52 Mbps
Upstream - 3.5 - 16 Mbps

* Wireless nets :-

:) Connects end system to router via access point.

* WLANs :-
:) ~100 ft
:) 802.11b/g/n (wifi): 11, 54, 450 Mbps tx rate

* Wide-area Cellular access nets :-
:) 10's Km
:) 10's Mbps
:) 4G cellular networks

* Enterprise networks :-
:) Ethernet: 100 Mbps, 1 Gbps, 10 Gbps
:) Wifi: Wireless access points at 11, 54, 450 Mbps

* Tx Delay = $\frac{L}{R}$, L - bits, R - bits/sec

* Links :-
:) Guided media - cables, fiber coax
:) Unguided media - Signals propagate freely

-> Twisted pair :-
Category 5: 100 Mbps, 1 Gbps Ethernet
Category 6: 10 Gbps Ethernet

-> Coaxial cable :-
:) bidirectional
:) broadband
:) Multiple frequency channels on cable
:) 100's Mbps per channel

Fiber Optic Cable:-

- 10's to 100's Gbps
- Low error rate

→ Network Core:- (1.3)

- Two key network-core functions:-

→ Forwarding (switching) - Local action - Move arriving packets from routers input link to appropriate routers output link. (forwarding tables)

→ Routing - Global action - Determine source destination paths taken by packets (Routing algos)

* Packet switching:-
  - Store and Forward
  - Queuing

* Circuit switching → Path is finalized and reserved before starting communication.
  - → 2 ways to do this.
    - FDM
    - TDM

Q
- 1 Gbps link
- each user:-
  - 100 Mbps when "active"
  - Active 10% of time

How many users can use network under CS and PS?

Ans:-

In CS, total users = $\frac{1\text{ Gbps}}{100\text{ Mbps}}$

$= \frac{1000}{100}$

$= 10$

- In PS:-

$${}^{n}C_{10}\left(\frac{1}{10}\right)^{10}\left(\frac{9}{10}\right)^{(n-10)}$$

---

* Performance - Delay / Loss / Throughput (1.4)

* Packet delay: 4 sources
  1) Processing delay
  2) Queuing delay
  3) Transmission delay - L/R
  4) Propagation delay - d/s

→ Example for Tx / Tp

→ 10 bits passing through a router.
Tp = 12 sec

∴ Total processing delay = 12×10
= 120 sec

$T_{prop} = \frac{100\text{ km}}{100\text{ km/h}} = 1\text{ hr} = 60\text{ sec.}$

Since the bits are pipelined we calculate the T_{prop} for last bit

∴ T_{prop} = 60 min

∴ Total delay = $60 + \frac{120}{60}$ = 62 min

* Queuing delay:-
  - a: average packet arrival rate
  - L: packet length (bits)
  - R: Link bandwidth (bit Tx rate)

L·a → arrival rate of bits

R -> Service rate of bits

∴ Traffic Intensity = $\frac{La}{R}$

$\frac{La}{R} \sim 0$: avg. queuing delay is small

$\frac{La}{R} \to 1$: avg. queuing delay is large

$\frac{La}{R} > 1$: more work arriving than what can be serviced.
- avg delay is infinite

Throughput -> Bottleneck Tx rate from src to dest.

* Bottleneck links tend to be at the edge of the nodes.

---

* Layering, encapsulation and service model (1.5)

* Layered Internet Protocol Stack:-

Application -> Transport -> Network -> Physical <- Link

-> Encapsulation -> The process of taking a packet from the upper layer and adding a header to it with some info.

Application Layer -> Message

Transport Layer -> Segment

Network Layer -> Datagram

Link Layer -> Frame

-> Routers and Switches do not use Application and Transport layer.

---

* Principles of the App layer (2.1)

:) Client - Server arch.
:) Peer-2-Peer arch.

Process -> Program running within a host

:) when 2 processes communicate within same host - Inter Process Communication (defined by OS)
:) Processes in different hosts communicate by exchanging messages.

Sockets -> Process sends/receives messages to/from its sockets.

-> Host device has a unique 32-bit IP address

-> Process is identified by IP address and port number.

* Open Protocols -> Defined in RFCs, definition is publically available.

* Proprietary Protocols :- eg Zoom, Teams etc.

* Internet Transport Protocol Service:-

TCP	UDP
:) Reliable transport	:) Unreliable data transfer
:) Flow control	:) Does not provide - flow/ congestion ctrl, timing, etc
:) Congestion control	
:) Connection Oriented	
:) Does not provide - Timing, min Throughput guarantee, security	

Vanilla TCP/UDP doesn't have any encryption.

- TLS V2.1 provides encrypted TCP connection, data integrity and end-point auth.

---

* Web and HTTP (2.2)

1) HTTP :-
   - Adopts client server model.
   - Uses TCP
   - Stateless

Non-persistent HTTP	Persistent HTTP
TCP open	
one obj sent over TCP	Can send multiple objs
TCP closed	

- RTT of Non persistant HTTP :-

  1 RTT -> To initiate TCP connection
  1 " -> To request file

  ∴ Total = 2 RTT + T_x,
  T_x -> Txn time of obj

* Cookies :-

- 4 components :-
  1) Cookie header line of HTTP response msg
  2) Cookie header line in next HTTP req. msg.
  3) Cookie file kept on users host, managed by users browser.
  4) Back-end database at web site

- **Web caches**: Instead of directly sending request to main server, the request is sent to cache. If obj cached, send obj. Else, request obj from original server, cache received response and return obj to client.

Caching example :-

Access link -> 1.54×10^6 bps
RTT -> 2 sec
Web obj size -> 100K bits

Avg. rate request from browser to server -> 15/sec

∴ Avg. data rate -> $15 \times 100K$
= 1.50 Mbps

Access link utilization :-

$$\frac{1.50 \text{ Mbps}}{1.54 \text{ Mbps}} \times 100 = 97\%$$

If institutional ethernet - 1 Gbps

then inst. LAN utilization $= \frac{1.50 \text{ Mbps}}{1 \text{ Gbps}}$
$= 0.0015$

End to End delay = Internet delay + Access link delay + LAN delay

Since access link utilization is 97%, the queuing delay at access link is the bottleneck.

Now, if we have a cache in inst. network which handles 40% of requests :-

Rate to browsers over access link $= 0.6 \times 1.50$ Mbps
$= 0.9$ Mbps.

Link utilization $= \frac{0.9}{1.5} \times 100$
$= 58\%$

Total end to end delay = 0.6 (1.5 RTTs) + 0.4 (cache Rate)
= 0.6 (2.01) + 0.4 (2 ms)
≃ 1-2 sec.

* Conditional Get :-

Client specifies date of cached copy in HTTP request.

if-modified-since : <date>

Server response contains no obj if cached copy is up-to-date.

HTTP/1.0 304 Not modified

* HTTP/2

To decrease delay in multi-object HTTP requests

Divide objects into frames, schedule frames to mitigate HOL blocking.

HOL blocking :-

Response to send a huge obj takes time and delays the responses of subsequent small objects.

* Email :- (2.3)

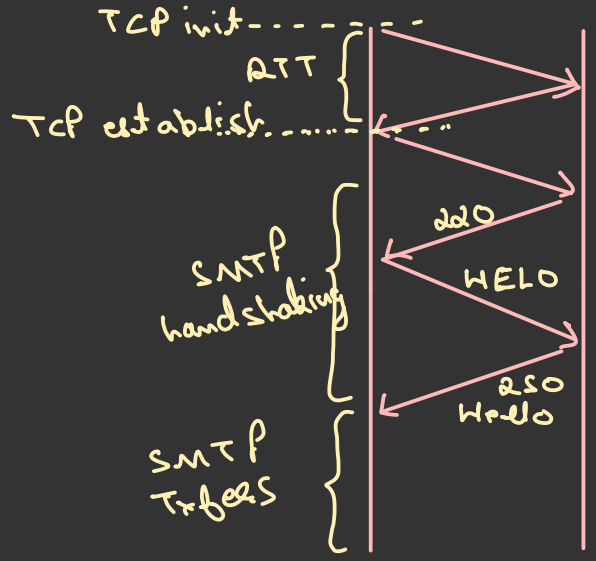
3 components → User agent
Mail server
SMTP

Outgoing / incoming messages from / to user agents are stored on mail server

Mail server → Mailbox - contains incoming messages for user.

Message queue of outgoing (to be sent) mail msgs.

SMTP connection life :-



:) HTTP is a pull protocol
:) SMTP is a push protocol

:) HTTP: Each obj encapsulated in its own response msg.
:) SMTP: Multiple objs sent in multipart msgs.

* DNS :- (2.4)

:) Distributed database implemented in hierarchy of many name servers.
:) App-layer protocol : Host DNS servers communicate to resolve names.

DNS services :-

:) Hostname to IP addr. translation
:) Host aliasing
  • Canonical alias name
:) Mail server aliasing
:) Load distribution
  :) Replicated web servers: many IP addresses correspond to one name.

→ There are 13 logical root servers

* We enter URL, req. goes to local DNS. It then uses

iterative query or recursive query to get the result.

- TLD servers are cached in local name servers.
- Every cache entry has a TTL.

DNS record :-

DNS: Dist. DB storing resource records (RR)

RR format: (name, value, type, ttl)

Type = A
- name is hostname
- value is IP address

Type = NS
- Name is domain (eg. foo.com)
- Value is hostname of authoritative name server for this domain

Type = CNAME
- name is alias name for some "canonical (the real) name.
- www.ibm.com is really servereast.backup2.ibm.com
- value is canonical name.

Type = MX
- value is name of SMTP mail server associated with name.

Getting your info into DNS :-

- Let's register name - xyz.com at DNS registrar
  - Provide names, IPs of A name server (primary and secondary)
  - Registrar inserts NS, A RRs into .com TLD server
- (xyz.com, dns1.xyz.com, NS)
- (dns1.xyz.com, 212.212.212.1, A)
- Creat 'A' server locally with IP - 212.212.212.1
  - Type A record for www.xyz.com
  - Type MX record for xyz.com

P-2-P architecture :-

- Peers request service from other peers, provide services in return to other peers.
- Peers are intermittently connected and change IPs
  - Complex management

Q How much time to distribute file (size - F) from one server to N peers?

u_s - Server upload capacity
d_i - Peer i download capacity
u_i - Peer i upload capacity

- Server Txn: Must sequentially send (upload) N file copies:
  - Time to send 1 copy: F/u_s
  - " " " N " : NF/u_s
- Client: Each client must download file copy
  - d_{min} = min client download rate
  - Min client download time = F/d_{min}

Time to distribute F to N client using client-server approach $D_{c-s} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$

) Now lets check for P2P :-

) Server tx: Must upload atleast one copy.

) Time to send one copy: F/u_s

) Client: Each client must download file copy.

) Min client download time: F/d_{min}

) Clients: As aggregate must download NF bits

) Max upload rate (limiting max download rate) is $u_s + \Sigma u_i$

Time to dist. F to N clients using P2P approach $D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{(u_s+\Sigma u_i)}\right\}$

---

* Video Streaming :- (2.6)

) Constant Bit Rate - video encoding rate is fixed.

) Variable Bit rate - Video encoding rate changes as amount of spatial, temporal coding changes.

Streaming : client plays out early part of video while server is still sending later parts of video.

Challenges :-

) Continuous playout constraint :-

During client video playout, playout timing must match original timing

· but network delays are variable (jitter), so will need client side buffers to match continuous playout constraint

* Dynamic Adaptive Streaming over HTTP (DASH)

Server:

) Divides video file into multiple chunks

) Each chunk encoded at multiple different rates

) Diff rate encodings stored in diff files

) Files replicated in various CDN nodes.

) manifest file: Provides URLs for different chunks.

Client:

· Periodically estimates server to client bandwidth

) Consulting manifest, one chunk at a time

) Chooses maximum coding rate sustainable given current bandwidth

) Can choose different coding rates at different points in time (depending on available bandwidth at time), and from different servers

) Client determines when to request chunk, what encoding rate to request and where to request chunks.

* CDN → store multiple copies of videos at multiple sites.

:) Enter deep: Push CDN servers deep into many access networks
- close to users
- Akamai

:) Bring home: Smaller number of large clusters in POPs near access nets
Used by Limelight

* OTT (Over The Top):-

Network is not an ISP. It's a content provider. But it uses the networks provided by ISPs to deliver the content over the ISP's networks at the app. layer.
That's why it's called OTT since it's an app level service running on top of the IP infra

---

* Intro to transport layer

:) Provides logical communication blw app processes running on diff hosts. (3.1)

:) Transport protocols actions in end systems

---

* Multiplexing and Demultiplexing

:) Multiplexing at sender - (3.2)

Handle data from multiple sockets, add transport header (later used for demux)

At receiver - Use header info to deliver received segments to correct socket

* How demux works:

:) Host receives IP datagrams
:) Each datagram has source IP address, destination IP address
:) Each datagram carries one Transport layer segment
:) Each segment has source, destination port number

:) Host uses IP addresses & port numbers to direct segment to appropriate socket.

TCP/UDP Segment format

→ Note that if IP/UDP datagrams with same dest port #, but different src IP addressed and/or source port numbers will be directed to same socket at receiving host.

:) TCP socket identified by 4-tuple:
:) Src IP
:) Src port no.
:) Dest IP
:) Dest port no.

---

* Connectionless transport: UDP

:) No fields, bare bones protocol. (3.3)
:) Best effort service, UDP segments may be:
:) lost
:) delivered out-of-order to app
:) Connectionless

:) UDP use:
- :) Streaming (loss tolerant, rate sensitive)
- :) DNS
- :) SNMP
- :) HTTP/3

* UDP checksum

Goal: Detect errors (i.e. flipped bits) in transmitted segment.

Sender
- Treats content of UDP segment (including UDP header fields and IP addresses) as seq. of 16-bit integers.

:) Checksum: Addition (one's comp. sum) of segment content
:) Checksum value put into UDP checksum field

Receiver
- Compute checksum of received segment
- :) check if computed checksum equals checksum field value
  - Not equal - Error detected.
  - Equal - Can still have error

:) Handling duplicates:
- Sender retransmits current packet if ACK/NAK corrupted
- Sender adds seq. no. to each packet
- Receiver discards duplicate packet.

This results in rdt 2.1

:) rdt 2.2
-> rdt 2.1 without NAK
Instead of NAK, receiver sends ACK for last pkt received OK
- :) Receiver must explicitely include seq # of pkt being ACKed

:) Duplicate ACK at sender results in same action as NAK: retransmit current pkt

TCP uses same technique as rdt 2.2

:) rdt 3.0 -> Underlying channel can also lose data
∴ we will add timeout

* Principles of Reliable Data Transfer (3.4)

:) rdt 1.0 -> underlying channel is perfectly reliable

:) rdt 2.0 -> Underlying channel may flip bits
- checksum to detect bit errors

- the question: How to recover from errors?

Ans -> ACKs and NACKs

-> rdt 2.0 has a flaw!
what happens if ACK/NAK is corrupted?

-> Stop and Wait:-

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

-> GBN:-

Sender: Window of up to N, consecutive transmitted but unAcked pkts
- k-bit seq # in pkt header

-> SR:-
:) If next available seq # in window, send packets.

Timeout(n):
- Resend packet n, restart timer

Ack(n) in [sendbase, sendbase+N]

:) Marks packet n as received
:) If n smallest unACKed packet, advance window base to next unACKed seq #

---

* TCP (3.5)

-> Point 2 Point
-> Reliable, in-order byte stream
-> Full-duplex
-> Cummulative ACKs
-> Pipelining
-> Connection Oriented
-> Flow Control

:) EstimatedRTT = (1-α) EstimatedRTT + α SampleRTT
· Exponential weighted Moving Average

:) Influence of past sample decreased exponentially fast

:) Typical value: α = 0.125

Timeout interval: EstimatedRTT plus safety margin

· Large variation in EstimatedRTT : want a larger safety margin

TimeoutInterval = EstimatedRTT + γ * DevRTT

DevRTT = (1-β)*DevRTT + β* |SampleRTT - EstimatedRTT|

13/10/23

Ans2 MSS = 1460 bytes
Total seq no = 2^{32}

$\frac{L}{1460} \leq 2^{32}$

Total segments = $2^{32} - 1$

Total file = $(2^{32} - 1) \times 1460$

b) Total data to send = $(2^{32}-1)\times 1460 + (2^{32}-1)\times 6$
$= (2^{32}-1)\times 1536$

$\therefore$ Time = $\frac{(2^{32}-1)\times 1536}{100\times 10^6}$

Ans5 F = 15×10^9 b
$U_s = 30 \times 10^6$
$d_p = 2\times 10^6$
$U_p = 300 \times 10^3$

$T_s = \frac{15\times 10^9}{30\times 10^6} = 5\times 10^2$

$T_p = \frac{15\times 10^9}{2\times 10^6} = 7.5 \times 10^3$

$T_{p_2} = \frac{15\times 10^9 \times 10}{30\times 10^6 + 3000\times 10^6}$

$= \frac{15\times 10^3}{3030} = \frac{15}{3.030} \times 10^3$

$\frac{F}{U_s} > \frac{F}{d_p} > \frac{NF}{U_s + \sum U_i}$

Ans6
Conditional GET
- Modified-since field

Ans7
F = 500×10^3 bits
obj = 5 — 10^5 bits each

RTT = 250 ms
R = 100×10^6 bps
Prop delay = 125 ms

Tx for file = $\frac{500\times 10^3}{10^8} = 5$ ms

Tx for image = $\frac{10^5}{10^8} = 1$ ms

a) 2RTT + file tx
500 + 5ms
= 505 ms

1 image = 500 + 1
= 501

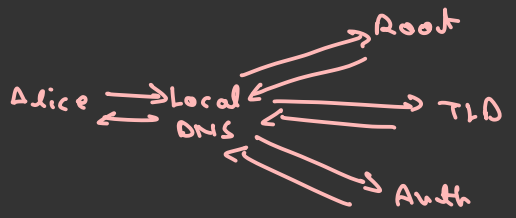
Total = 505 + (5 × 501)

b) Request — 2RTT
= 500 ms
File → 5ms

Image = RTT + 1

2RTT + 5 + RTT + 1
3RTT + 6

Ans 4



a) Total DNS messages = 8

b) 4 (RTT - DNS) + (2RTT - TCP handshake)
+ Tx (HTML file)

c)