(Revision): Network Layers

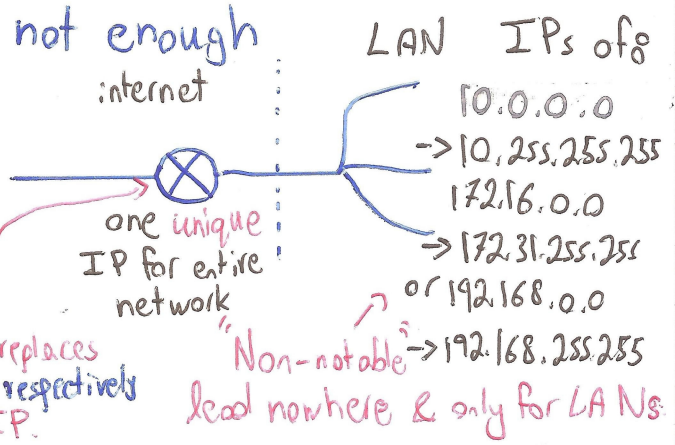
MAC		IP	Port Num. (unique to each application)	
Physical & Data	,	Network	Transport	, Application
bits / frame		packet	segment	message
• node-to-node only worry about next hop.		• give each machine unique IP. • can be: - preset (static) - fill routing table (dynamic) with topology & least cost	• logical comm. between processes • congestion control (lower/raise rate) • flow control • multiplexing & demultiplexing	• logical communication between hosts.

ex: default port for web server: 80.

- "NATing": (Network Address Translation)

IPv4 ~ 2^{32} addresses => not enough

∴ within a private network (where there is only one way in) we give out IPs but IP dest doesn't specify who in the LAN => go up one layer to the Transport layer. Give each machine a unique port a table at the LAN's router



Advantages: - Less IP use - switch ISP without changing local device address
- changable IP - Local IP not visible (Security).

ex (translation table):

IP (loc)	Port (loc)	IP (out)	Port (out)
10.0.0.1	1234	router IP	1
10.0.0.2	5161	router IP	2

- Transport Layer:

- multi/demultiplexing: The process of handling data from multiple sockets using headers

- UDP (User Datagram Protocol):

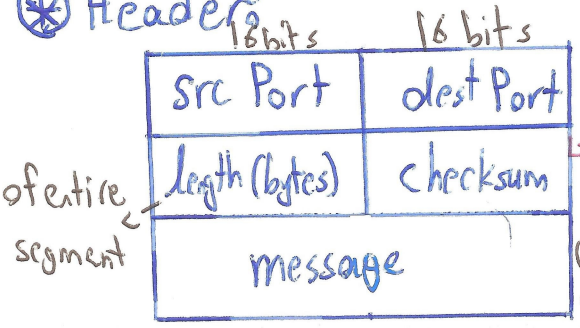
⊕ Connection-less [no handshaking] ⊕ small header

⊖ Best effort (gets lost/full of errors)

⊛ Use for less-tolerant connections: - streams - DNS (Domain name service) - SNMP...

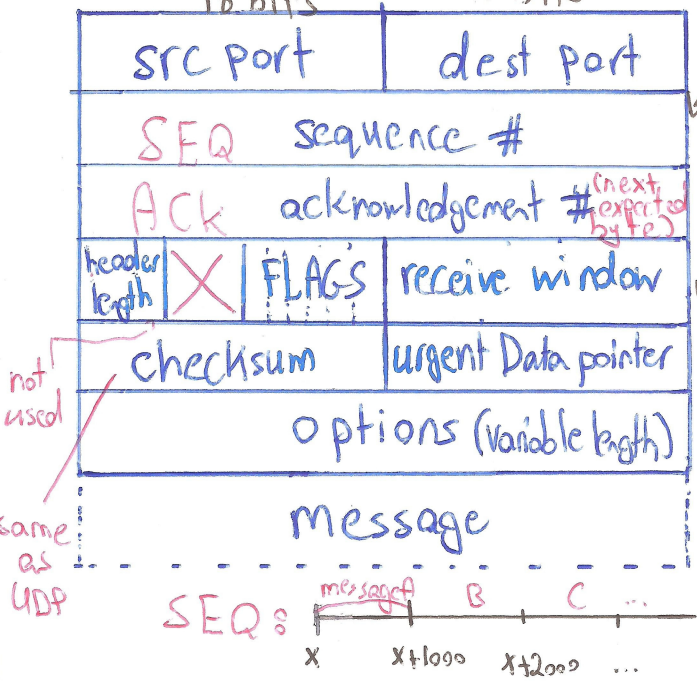
⊛ Made reliable with checksum on the Application Layer => we never use pure UDP.

⊛ Header:



- TCP (Transmission Control Protocol):
  - ⊕ one sender - one receiver (port to port)
  - ⊕ "Byte Stream": segmentation is done for us by the protocol
  - ⊕ "pipeline": no wait, just keep sending & TCP controls the flow.
  - ⊕ "full duplex": bi-directional.
  - ⊖ connection oriented.
  - ⊕ sends in order
  - ⊕ reliable
  - ⊕ no overwhelming & congestion.

⊙ Header: (all values in bytes)



by location of first byte in original data

← for flow control

Note: both sides can be sending & receiving at the same time.

FLAGS: 1 bit each

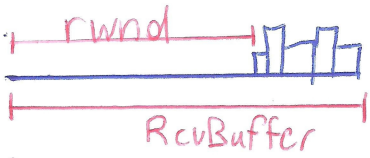
U A P R S F

- U: urgent data (not used)
- A: if the ACK is valid (read it or not)
- P: data must be pushed before others (not used)
- R S F: ↓ for 3-way handshaking

SEQ:

(x randomly generated for security)

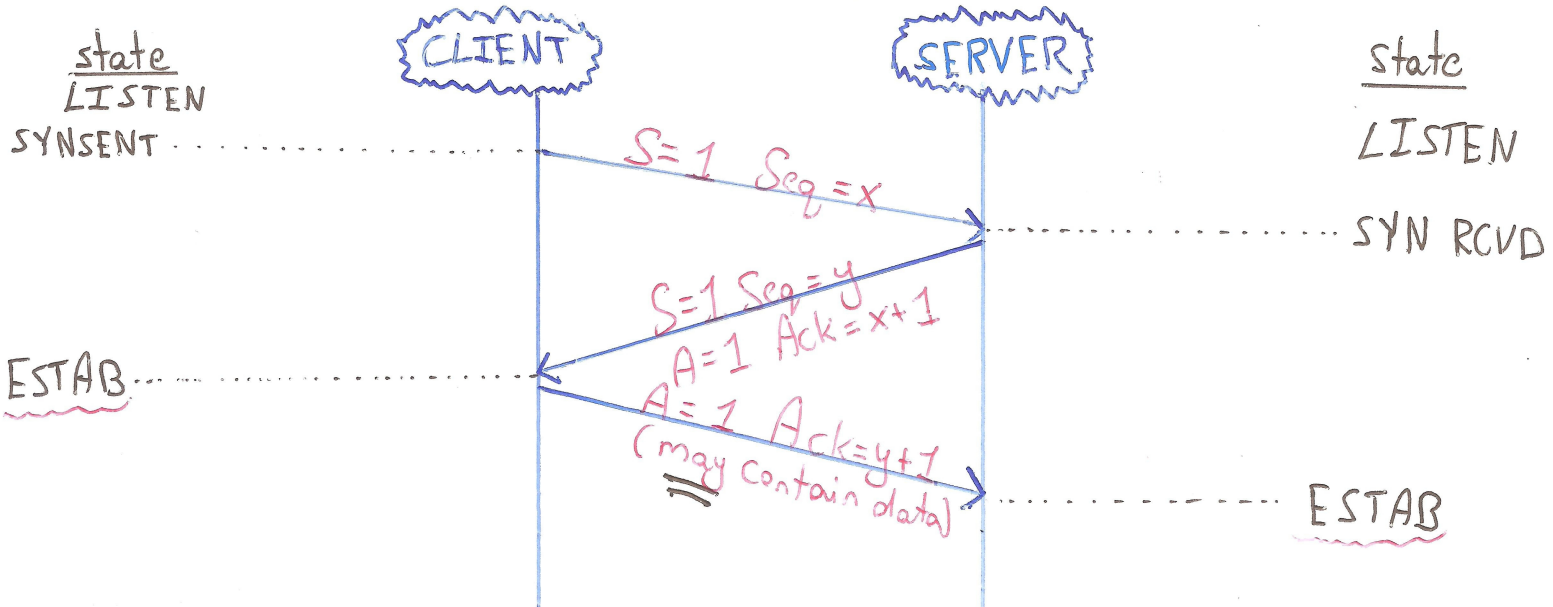
⊙ Flow Control: using receive window to tell other side how much more my buffer can fit to guarantee no overflow.



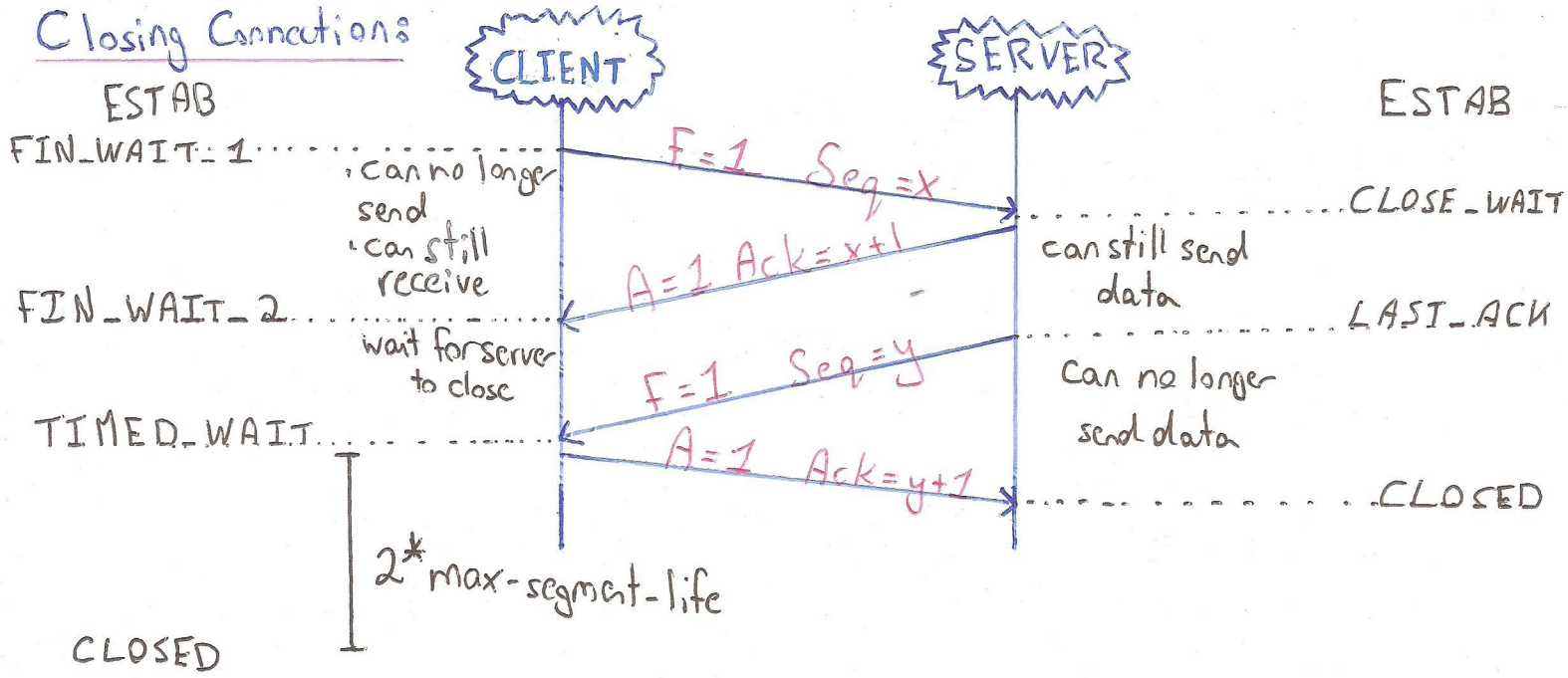
rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]

⊙ Connection management (3-way-handshaking):

opening connection:



Closing Connections:



◎ **Congestion Control:** Manage rate of sent data segments/bytes

Congestion Window (cwnd): initially 1 MSS (max segment size)

Round-trip-time (RTT)

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

Slow-start-threshold (ssthresh): initially value is the advertized window size.

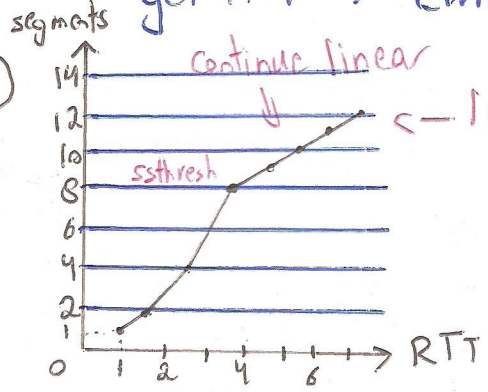
Two Modes: slow start: (cwnd < ssthresh)

get Ack => cwnd = cwnd + 1 MSS

Congestion avoidance: (cwnd >= ssthresh)

get Ack => $\text{cwnd} = \text{cwnd} + 1\text{MSS} \times \left(\frac{1\text{MSS}}{\text{cwnd}}\right)$

ex)



<- loss happens somewhere

=> TCP changes size of sending window.

ssthresh = last cwnd / 2

	TCP RENO	TCP TAHOE
timeout	cwnd = 1MSS	cwnd = 1MSS
3 duplicate ACKs	cwnd = cwnd/2	cwnd = 1MSS

Window then continues normally (s.s. -> c.a.) till another loss occurs.

RTT (Round Trip Time): we need RTT to set our TCP Timeout (time to wait for ACK before retransmission) => $TT > RTT$

too short => unnecessary re-transmissions
too long => time wasted if loss occurs.

But RTT varies => estimate it.

$$EstRTT = (1-\alpha)\,EstRTT + (\alpha)\,sampleRTT$$

(α: pre-set value (usually 0.25))

- $EstRTT$: last estimate
- $sampleRTT$: last received ack − last sent.

Now that we estimated RTT, we must calculate a Timeout.
RTT deviates => calculate possible deviation:

$$DevRTT = (1-\beta)\,DevRTT + (\beta)\,[sampleRTT - EstRTT]$$

- $DevRTT$: last deviation
- β: pre-set value (usually 0.25)

$$Timeout = EstRTT + (4)\,DevRTT$$

- $EstRTT$: how long we expect the transmission to-and-from to take
- $4\,DevRTT$: safety margin

IPV6:
- improves quality of service
- header format => faster processing & formatting
- came out of fear of IPv4 being completely allocated.

- Fixed 40-byte header
- No Fragmentation

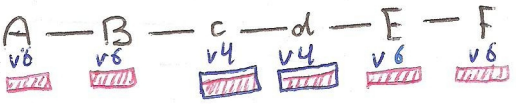
ver	pri	flow label	
payload len		next hdr	hop limit
src address			
dest address			
data			

32 bits

pri: priority in own "flow"
ver: IPv6
flow label: ID of "flow"
next header: location of "options" in data

Note: ICMPv6 has new messages & functions
ex: Packet too big.

Not all routers can be switched over in 1 day => Tunneling:



v6 carried as a payload in v4 datagrams.

Routing Algorithms (Network Layer)

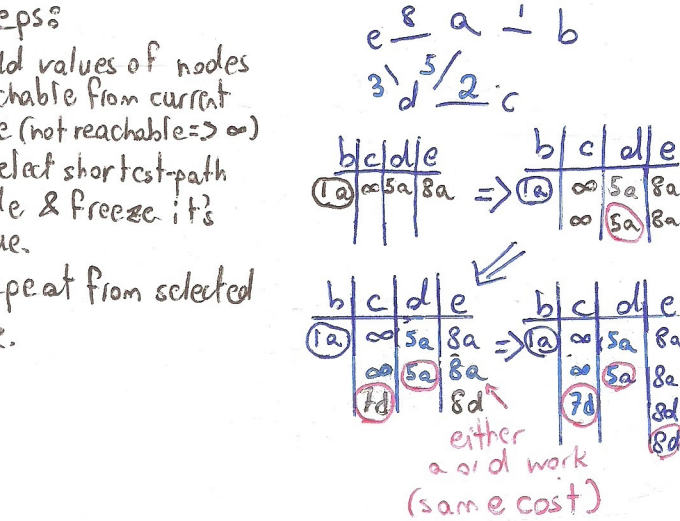
least-cost path finding

- Classification:
  - * Global: (link state algorithms) all routers have complete topology. (map + link costs)
    Decentralized: (Distance vector algorithms) router knows its neighbors only.
    - iterative process of communication/exchange with neighbors.
  - * Static: routes slowly change over time.
    Dynamic: routes change - periodically.
    - in response to link cost changes.

Link State Algorithms:

Dijekstra's: - topology known to all nodes (including link costs) by link state broadcasting => all have same info
- computes least cost from source node to all other nodes.

steps:
1- add values of nodes reachable from current node (not reachable => ∞)
2- select shortest-path node & freeze it's value.
3- repeat from selected node.



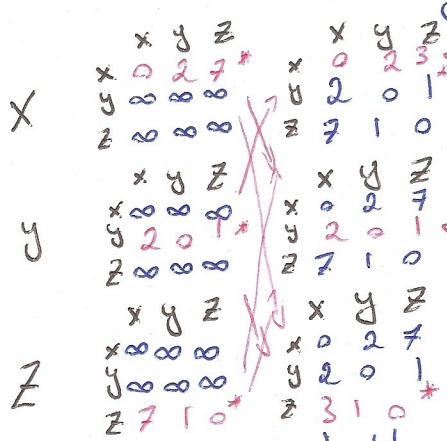
Distance Vector Algorithms:

Bellman-Ford: $d_x(y) = \min\{ c(x,v) + d_v(y) \text{ for all neighbors } (v) \}$

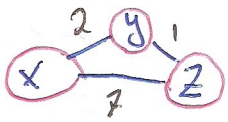
each node has: - cost to each node (neighbor) v
- $d_v(n)$ where n is every other node in the network.

periodically, each node sends "$D_x(n)$'s" to it's neighbors prompting them to update their own distance vectors.

Nodes only notify all neighbors if:
- local link cost changes
- notified of DV change by neighbor that causes a change in the saved values



(the y row in x and z's graphs won't change)



possibility of ∞ loop => we use poisoned reverse:
if z routes to x through y, z will tell y that it's distance to x is ∞ (so y won't route to x through z / itself)

Application Layer

Web objects: HTML file, JPEG image, audio file, Java applet, etc...

• HTTP: (HyperText Transfer Protocol)

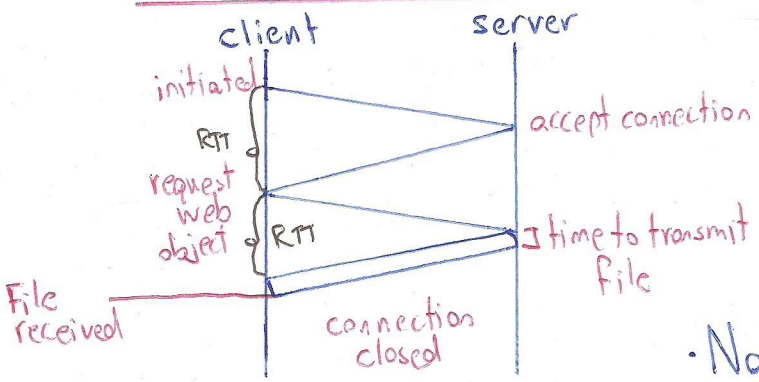
client (browser) requests, receives, & displays web objects

server sends objects in response to client

- overview:
  - client initiates TCP connection to server
  - server accepts connection
  - HTTP messages exchanged
  - TCP connection closed

Note: HTTP is stateless (no saved information => treat any new connection as first-time)

• Non-Persistent HTTP: (HTTP/1.0)



- Full proces is repeated per HTML page and for every file, image, script, etc..

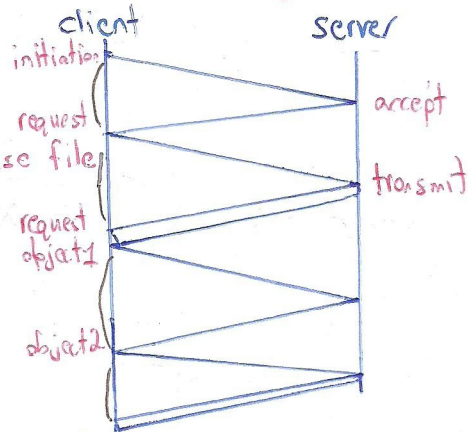
$$time = (2RTT + t_{trans(file)}) \text{ per object}$$

- Note: Parallel connections => open 2 or more connections to get objects faster.

• Persistent HTTP: (HTTP/1.1)

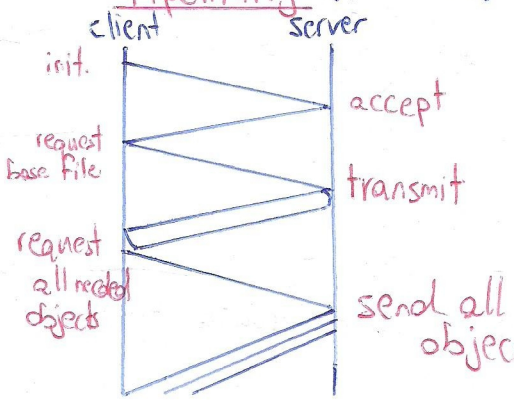
Connection is left opened for subsequent HTTP messages

No Pipelining

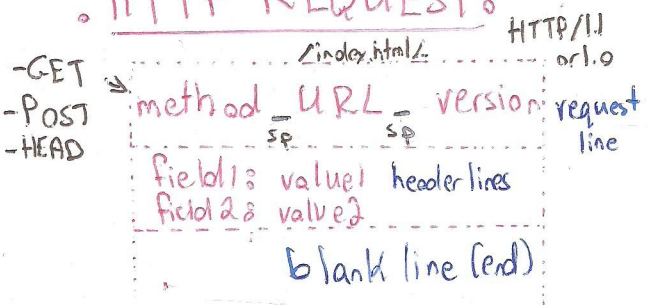


$$time = (2RTT)_{\text{establishment}} + (1RTT + t_{trans})_{\text{per object}}$$

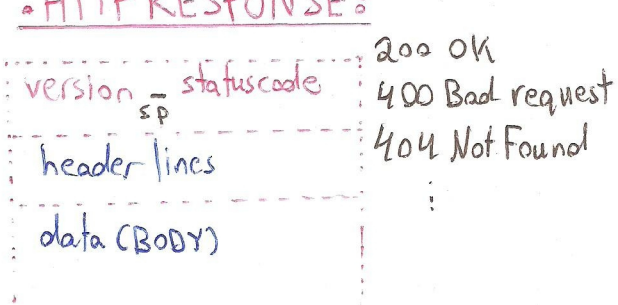
Pipelining (HTTP/1.1)



• HTTP REQUEST:



• HTTP RESPONSE:



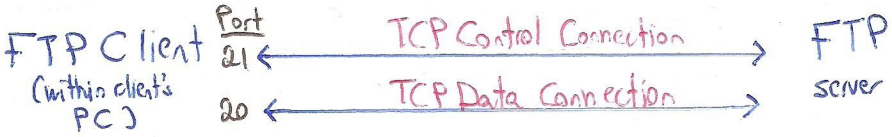
• Web Caches (Proxy Server):

Satisfy client requests without involving original server

Client requests object => check cache or proxy first → object found => ask if modified since last request => ask for new copy; else send back current copy

→ not found => request object from original server

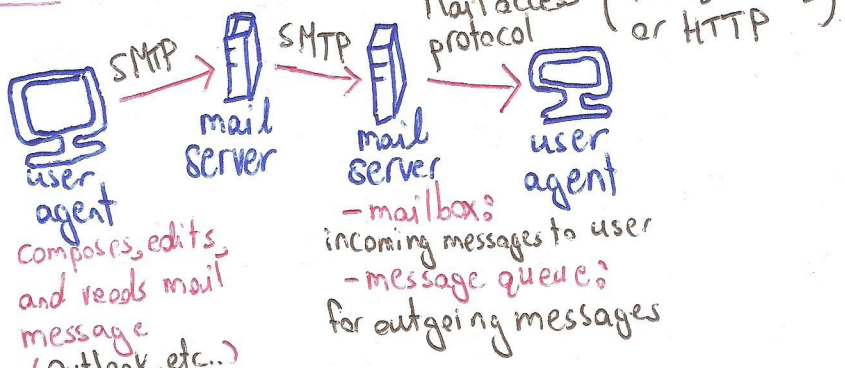
• FTP (File Transfer Protocol):



- client initiatetes control (obtains ontrol over it) & sends file transfer commands
- server creates Data connection, uses it, and destroys it. Repeat for every file

Note: TELNET: protocol that provides a bi-directional, 8-bit byte oriented connection over TCP.
telnet: program used to execute TELNET protocols.

• Electronic Mail (SMTP, POP3, IMAP):



SMTP [RFC 2821]:

- uses TCP port 25 (7-bit ASCII messages)
- 3 phases (- handshaking
  - message transfer
  - closure)

POP3 [RFC 1939]:

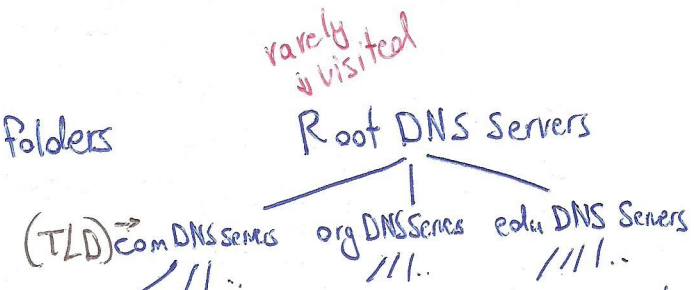
messages stay on server and only copies are temporarily downloaded

IMAP [RFC 1730]:

every thing stays on the server + ability to use folders

• DNS (Domain Name System):

DNS is distributed and hierarchial to prevent traffic, single point failure, and lower costs on maintenance.



- TLD (Top Level Domain) servers each handle an extension (com, org, net, edu, uk, fr, lb, etc...).
- Authorative DNS Servers: private to an organization, provide hostnames to own IPs (web and mail) can be maintained by organization or a service provider.
- Local DNS Servers: speed up DNS lookups locally (usually in companies)
  - each ISP has one

- check local first. Not found => local will send the request up the hierarchy.
- Server finds request => cache it (they time out after some time)

⚠ Port Numbers to Remember:

FTP (Data): 20	DNS: 53	IMAP4: 143
FTP (Control): 21	HTTP: 80	
SMTP: 25	POP3: 110	

Security

The more advanced we get, the less skills an intruder needs to perform a successful attack

Computer security: collection of tools to protect data & thwart hackers

network security: measures to protect data during transmission

internet security: protect data during multi interconnected network transmission.

✱ Security Attacks: actions that compromise the security of an organization's data.

passive attacks:

- **Interception:** - observing and analyzing traffic to/from a target
  - read actual content in messages.
  - hard to detect.

active attacks:

- **Masquerade:** pretending to be someone else to gain privileges.
- **Modification:** changing content of a message
- **Fabrication:** creating a completely new message
- **Replay:** Capturing the message and waiting before sending it forward.
- **deny:** stop sending/receiving messages
- **Denial-of-Service:** prevent normal use of servers, end users, or entire network.

require the attacker to be the only node between the target & their messaging destination.

✱ Security Services: (Important)

Authentication: assuring the right entity gets the right data.
- **peer-entity auth:** confidence in the id of parties involved in connection
- **data-origin auth:** assurance about source of data being received.

Access Control: prevent unauthorized access to a resource.

Data Confidentiality: protect data from eavesdropping

Data Integrity: prevent modification, insertion, deletion, and replay.

Non Repudiation: track user actions (log) => protect against users denying any of their actions

✱ Encryption:

- **Symmetric key Cryptography:**
  A & B share a key (K)
  A encrypts with K(m), B decrypts with K(K(m)) to get message

  how can A & B securely agree on a shared K? not possible =>

- **Public key Cryptography:**
  - A uses B's public key to encrypt messages $K_B^+(m)$
  - B decrypts with his own private key $K_B^-(K_B^+(m))$

- **Some cryptos:**

DES (Data Encryption Standard) → **3DES** → **AES** (Advanced Encryption Standard)

- DES: 56 bit symmetric key, 1 day to brute force
- 3DES: DES 3 times with 3 different keys
- AES: process data in 128b blocks, 256 bit keys, 149 trillion years to brute force

- Digital Signatures: anything encrypted with the private key can be decrypted with the public key.

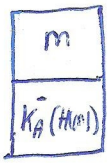
=> Anyone can decrypt it => Anyone can know for certain who the sender is => Authentication achieved.

A -> B

How: send a message + m, $K_A^-(m)$ => B uses A's public key & compares m with $K_A^+(K_A^-(m))$

+non repudiation (Sender can't deny that he sent)
+integrity (changes to data => keys stop working)

Hash method:
instead of sending an encrypted m, we send a hashed $H(m)$.



=> $H(m)$ of fixed size is easier to compute & encrypt

issues: we can find many messages of the same hash checksums.

Note: destination will compare the decrypted $H(m)$ and the received m hashed

=> if someone accesses a DB to get $H(m)$ he can't use it to gain access.

the most used hash function algorithm is MD5

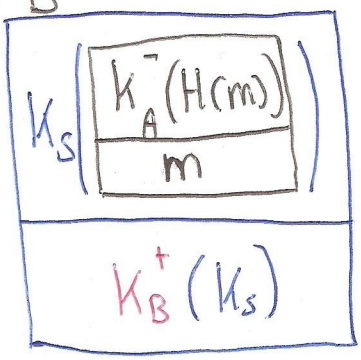
SSL (Secure Socket Layer): widely used security protocol (HTTPs) that relies on an API on the application layer to share a digital signature alongside the message while encrypting it.

=> confidentiality, integrity, & authentication.

PGP:

A ⟶ B

A sends:



now B can use lower part to get K_S, a symmetric key that it can use to decrypt the upper part.

=> we used public key cryptography to share a symmetric key and now we can use K_S to send & receive data.