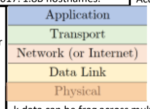
NC

Internet Evolution: Sept 1969 = 1 node. Oct: first message sent on ARPANET sys (22:30 29th). UCLA host → SRI (Stanford). Word "login" – 'l' and 'o' sent before crash, but all worked after reboot an hr later. Dec: 4 nodes. 1971: more unis in US connected. 1976: Queen sends email. 1980: more unis in US (+ London) connected. 1986: Wide expansion. 1999-2016: Num Hosts+Websites exponentially increased. 2017: 1.8B hostnames.

Hybrid 5-Layer Model: **4-Layer**: App, Trans, Internet, Network Access. **7-Layer**: App, Presentation, Session, Trans, Network, Data Link, Phys. Layer k puts all data in k-1 packet (may add header/trailer). Layer k data can be frag across multiple k-1 packets. Up /down layers add/rem H/Ts.



Networking: Pcs of interconnecting comp systems via telecommunication methods to share data+resources. Comp Networks becoming pervasive (almost everywhere). Most mainstream software sys are distributed sys (e.g. cloud computing – Amazon AWS EC2 – and smartphone apps). Performance often depends on network usage.

Vocations: *Network Engineer/Architect*: Design, build and maintain networks, *Server Application Developer*: Server Backend and communication for cloud applications, *Network Software Engineer*: Networks + Software Engineering, *Data Center / Cloud Platform Admin*: Networks + Cloud Comp, *Network Security Engineer*: Networks + Computer Security, *Consultant*: Networks + Anything.

Networking Stack: **Application Layer**: apps send/recv data in format they specify. Impl details of OS, packet types, network setups and hardware models abstracted away. Apps use proto-cols - def struct of data (reqs/resps), as well as port nums and other conventions. **World Wide Web**: part of internet dev by Tim Berners-Lee (working@CERN). Uses HTTP (HyperText Transfer Protocol). Early vers use plaintext (newer+more adv no longer always use). Level of abstraction on which we consider protocols /agreements/ transfer of app data. Browser: GET/HTTP/1.1 Host: www. … .com, Server: HTTP/1.1 200 (OK) <!Doctype html> … . **Transport Layer**: Establishes basic data channels, taking data to be sent/recv + conv to/from data packets. Can be: **Connection-Oriented TCP**: Transmission Control Protocol. Pckts not ACK'd resent. **Connectionless UDP**: User Datagram Protocol. No checking, pckts sent once, more performant.

Network Layer: internet protocol used to add IP addrs + other info to pckts, then route through mesh network of hosts to reach dst. Path taken freq changes and is per-packet. **Data Link Layer**: NIC (Network Interface Controller) hardware controlling comms over standards to allow phys comm of data to transfer data (packets) between devices. *Wired*: ethernet, fiber optic network card. *Wireless*: WiFi access points, 4G USB dongle. Reducing, detecting + rectifying bit trans layers. Adding parity bits, checksum (e.g. Cyclic Redundancy Check). Specifying how computers share common channel (MAC – Media Access Control – addrs). Specifying how network connects (e.g. Ethernet, FDDI – Fibre Distributed Data Interface), and token rings (holds 1 token+listens at a time). **Physical Layer** (comm media): actual hardware transferring data – fibre optic cable, twisted-pair copper cable, coaxial cable, wireless links (wifi: 802.11, Bluetooth). Short wire connecting comps, transmission of raw bits: set 0 = +4V, 1 = -3V, change freq of 20kHz (20,000 times per sec).

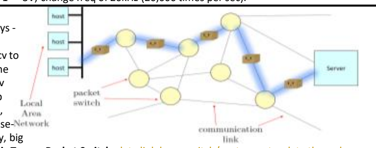
Application Layer Protocols: *Trad*: name services (DNS), Email (SMTP), FTP, Telnet, SSH, HTTP(S). *Modern*: middleware to support distributed systems (Java RMI, Apache Thrift, Google Protocol Buffers – sending serialized data). *High-Level*: e-commerce, banking (visa) etc. *P2P*: BitTorrent, Skype (old protocol).

Network Layer: Describes how routing /congestion is done. Determines best route. Deals with router unreliability (e.g. connection goes down). Supports multi/ broadcasting. Dealing with packet dropping (e.g. when router overloaded). Detection/avoidance of hotspots. *Multicasting*: sending info to many recipients from one source (red network traffic e.g. CCTV sending single vid stream to backups).

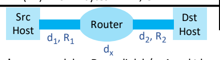
Transport Layer: Offers connection-oriented and connectionless protocols. Often provides network interface through sockets (e.g. UNIX sockets). Provides support for secure connections. Support for datagrams (unreliable but fast per-message basis sending – connectionless e.g. UDP). Provides flow control mechanisms to prevent fast senders overwhelming slow receivers.

Protocol: End syss + pckt switches run protocols. Established set of rules that determine how data is transmitted between diff devices (e.g. describe layout and meaning of packets and order they should be sent). Phases: **Handshake**: establish IDs, and context to begin comms. **Convo**: comms, exchanging data in format+way specified by protocol. **Closing**: terminates convo, performing necessary cleanup/notif to other. Executable specification, unambiguous, complete (cover all poss sitchs+msgs), define all necessary msg formats. Same service can be realised by diff protocols. **Designing Protocols**: must consider *addressing* (how to denote intended recipient), *error control* (detection + poss correction of inevitable transmission errs), *flow control* (prevent fast sender swamping slow receiver), *(de-)multi-plexing* (supporting parallel comms), *routing* (route packets to dst via best route w/ low processing/space overhead. Most network layers have both types of protocols: **Connection-oriented** (set up conn with client, transmit data over channel e.g. circuit switch, TCP on IP), **Conn-less** (send data to dst addr, no formal connection created – postal mode – e.g. packet switching, UDP on IP).

Internet Structure: **Host/End Sys**: Comp sys - src/dst of comms. E.g. smartphone (send/recv to browse internet), home security sys (send/recv security footage), web server, laptop, car, TV, toilet seat. Balance ease-of-access with security, big data with privacy. **Basic Terms**: **Packet Switch**: data link layer switch/router routes data through network. **Comm Link**: connection between packet switches &/or end syss (hosts) e.g. fiber-optic cable, twisted-pair copper wire, coaxial cable, wireless local area links, satellite channel. **Route**: Seq of switches a packet traverses to go src→dst. **Protocol**: standard concerning control and format of sending/recv data to/from end syss.



Units: Kb = 1000^1 bits, KB = 1000^1 bytes, KiB (KB) = 1024^1 bytes. M = ^{2}, G = ^{3}. T = ^{4}.



$d_{1,2}$ = prop delay, $R_{1,2}$ = link b/w, **L** = pkt len.
($R_1 < R_2$) → $d_{end-end} = d_1 + L/R_1 + d_x + d_2$ sec
($R_1 \geq R_2$) → $d_{end-end} = d_1 + d_x + L/R_2 + d_2$ sec
So, $d_{end-end} = d_1 + d_x + d_2 + L / \min(R_1, R_2)$ **sec**

Processing Delay d_{proc}: checking for bit errs, determine output link, negligible (<msec). **Queueing Delay d_{queue}**: time waiting at output link for transmission. If link congested, pckt might be queued for long time before being sent. If queue too long, may be dropped. **R** = link bandwidth (bps), **L** = packet len (bits), **a** = avg packet arrival rate, **La/R** = traffic intensity:

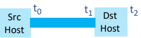
Delay = { small La/R ≈ 0; large La/R → 1; ∞ La/R > 1 } If more work is arriving than can be processed, the delay becomes infinite (packets likely dropped).

Packet Switching: Pkt = formatted unit of data. Data split into pckts which are indep routed through network. Switches+routers use packet info+network status to determine which next router/end sys to fwd packet on to. If any links in network become slow/disconnected, packets rerouted. No setup cost. Pcs'ing cost assoc with forwarding each packet. Space cost assoc with containing indep info in every pckt. Quality of Service difficult to guarantee (no connection, processing+ switching overhead, others can start using links without reservation). High network resource util (can send packets on diff routes in parallel, two connections can work on shared comms links). **Internet** packet-switched network. **Circuit Switching**: Comms require connection setup phase – network reserves all necessary resources for that connection (links, buffers, switches etc). High setup cost, Quality of Service guaranteed (as reserved). After successful setup, comm syss connected by set of links dedicated to their connection for entire duration of convo. No pcs'ing/space cost as data sent straight down link. If link becomes slow (over-saturated) or breaks, must obtain new link (slow). When ends, network destroys connection, freeing corresponding res for other conns. Only allows limited sharing of comms ress (once connection established, res blocked even if silence period). Inefficient network use. **Old Telephone Network** (modern is digital Voice over IP). Circuit (path through network) connected +maintained for call's duration.

Terms for Data: **App**: Data, **Trans**: TCP segments/UDP datagrams, **Network**: IP datagrams (packets), **Data Link**: Frames, **Phys**: Bits

End Systems Applications: Internet apps. Pcs run on diff hardware, OS etc. Protocols offer layer of abstraction (only need to comply). Pcs may exchange msgs (act as input), need to addr each other to comm. Diff pcs use diff ports, only 1 pcs can use a port on 1 IP at a time. OS provides networking primitives (sockets). End-sys/host may run multiple progs w/ multiple pcs using internet/networking. Pcs addr'd within host using port num which it is using.

Speed and Capacity: **L** = num bit msg, **R** = data amt/sec.



Throughput: (*link bandwidth*) amt of info that actually gets into/out of the connection in a time unit. At "steady state" ass 0 accum in traffic → input = output throughput. Total data recv per time (link bandwidth).
$R = L / t_2 - t_1$ (general formula = transferred bits/duration).
Latency: (*propagation delay*) time taken for single bit to go through connection (be transmitted from one end to the other).
$d = t_1 - t_0$ (general formula = dist/wave propagation speed).
Packetisation: (*transmission delay*) time per bit to be received. **L / R**. **Bandwidth**: maximum possible throughput. **Transfer Time**: send time per bit. $\Delta = d + L/R$.

Roles: **Client**: Initiates comms, if conn-oriented service establish comms. *Sockets*: 1. create socket by connecting to server, 2. use socket to R/W from it, 3. disconnect+destroy socket. **Server**: Waits for connections, if conn-oriented passively accept conn reqs. *Sockets*: 1. create server socket by accepting conn on port, 2. R/W data from socket to use, 3. disconnect+destroy socket. R/W for both part of C/S app protocol. **P2P**: both act as C+S e.g. BitTorrent, Gnutella.

HTTP: HyperText Transfer Protocol, used for transferring web objs/providing access to them. Conn-oriented (TCP), can do conn-less (UDP). Each req+resp 1 unit, no req dep on prev (stateless), all self-contained, if req dropped others not affected. 1.1 most pop, 2.0 faster+influenced by Google projs, 3 in final draft. 80 = HTTP req port. **Persistent**: connection stays open for multiple reqs, instead of new one for each. **HTTP/1.0**: 1 TCP connection/obj. Ineff use of network+OS (lots of spawn+dest). **1.1**: same TCP connection used for mult reqs+resps (can recv multile objs). Default to persistent connections (keep open for further reqs). Req containing *Connection: close* done after all resp/req sent (intention of c/s not to use persistent). **2**: exchange in binary (compact+higher speed), connection fully multiplexed (not ordered /blocking), single TCP with par reqs. **3**: UDP.

HTTP Methods: **GET**: retrieve obj using URL. **POST**: submit data to server (e.g. form/message). **HEAD**: only GET header (test link validity). **PUT**: reqs enclosed obj stored under given URL. **DELETE**: delete given obj. **OPTIONS**: req available comms options for obj. **Status Codes**: 1xx = informational, 2xx = OK, 3xx = redirection (obj moved temp/perm), 4xx = client err (00 = malformed req, 01 = unauth, 04 = obj not found, 05 = method not allowed), 5xx = server err (00= internal, 03 = overloaded). **telnet**: send plain-text comms directly to server listening on port.

```
GET /~kgk/212/index.html HTTP/1.1
Host: www.doc.ic.ac.uk
User-agent: Mozilla/5.0
Accept-Language: en-GB
```

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 16:00:00 GMT
Server: Apache
Last-Modified: Sat, 21 Jan 2017 20:05:55 GMT
Accept-Ranges: bytes
Content-Length: 5470
Content-Type: text/html
```

Followed by empty line then obj body (poss empty)

Protocol: should incl ver num (usually very first bits). Mechanism to negotiate protocol ver allows des to change. **Req**: protocol ver, URL spec, connection attrs, content/feature negotiation. **Resp**: protocol ver, reply status/val, connection attrs, obj attrs, content spec (type, len), content (objs).

HTTP Sessions: HTTP = stateless but need stateful apps, done through **Cookies**. **Set-Cookie Header**: sent within HTTP resp, from S→C. Tells C given cookie is session ID for site. **Cookie Header**: sent within HTTP req C→S, tells server that the req belongs to sesh. Some sites keep cookies between visits to track users.

IP Addrs: uniquely ID end sys by addr. Easy format for routers to pcs, not practical for ppl. Pre-1983: use file containing map: host mnemonic → IP addrs. **DNS** (Domain Name System): map: name → IP addr. **Root Server**: each top-level domain (e.g. .com, .org) is assoc with 1 of 13 root DNS servers (op by 12 indep orgs). **Top-Level Domain Servers**: DNS server assoc with top-level domain. **Authoritative Servers**: for each domain, hold master copy mapping pub hosts within that domain. Most root+lower level servers impl as distributed set of machines. **A** = load balanced traffic, lower latency. **D** = redundancy, major point of failure + bottleneck.

WWW: hypertext + hyperlinks. Glorified FTP, plaintext transfers. HTTP old concept (60's), HTML simple, HTTP protocol stateless+simple, low barrier of entry, GUI browsers = more accessible. **Document**: webpage (website has several). **Objs**: file (doc may have several e.g. HTML, JS, images). **URL**: Uniform Resource Locator (specifies obj addr). **Browser** (user agent): prog to req/recv docs+pcs docs to display graphically. **Web Server**: app containing docs+objs serving to Cs over HTTP.

Dynamic Webpages: Instead of storing/serving static pgs, gen pgs for given reqs on the fly. **CGI** (Common Gateway Interface): allows to ID a prog+params from URL. Server will start pcs to exec the prog which returns res (if any) as regular webpage. **Servlets**: Java sol to state (webserver creates new instances of JVM to run+pcs reqs for each client. **Alt**: exec code on client side instead of server. Server-side: PHP, C#, Java, Perl, Client-side: JS, CSS, HTML.

Web Caching: S→C = high latency, low bandwidth, cost = C_H. proxy→C = low latency, high bandwidth, cost = C_L. To serve 3 Cs, either $3C_L$ or $1C_H + 3C_L$. **A** = reduced latency for reqs, red network traffic, better security (server only sees proxy), when using firewall on proxy LAN protected. **D** = latency with finding entires+caching them, complexity (proxy setup), keep data fresh. Servers specify expiration time using Expires (: date) /Cache-Control (: max-age = …) header. Client/Proxy can use cond GET by including "If-Modified-Since" header. Make HEAD req to see if obj has been updated. OPTIONS reqs not cacheable. Central to several HTTP features.

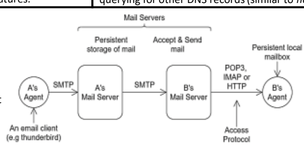
DNS (Domain Name Sys) Protocol: Connectionless (UDP port 53). Getting hostname transl only req 2 packets (overhead of TCP significant compared to msg time). Query/reply msgs, both have ID so msgs can be assoc. Both same basic format for simplicity. **Round-Robin DNS**: Load-balancing technique for geographically dist web servers. 1. DNS server req trans of hostname from auth DNS server, 2. DNS req (to get mapping) resp with list of IP addrs, 3. DNS server RRs through each addr (using each a specific num of times before moving to next) to make clients send reqs to many IPs, 4. reqs to hostname balanced across many servers. TTL low (< 18s). **Manual DNS Lookup**: *nslookup*: tool to find DNS info for hostname, first line = DNS server used, non-auth specifies the addr was extracted from cache. *dig* (Domain Info Groper): more info on name servers, enables querying for other DNS records (similar to *host –v*).

DNS Cachini: reduce load on DNS infrastructure while imp performance. Stale cache updated from auth server. **Cache Poisoning/Spoofing**: entering incorrect mappings to cache. **Features**: directory service database. Each entry is resource record describing transl of name [Name | Val | Type | TTL]. **TTL** (Time To Live): how long mapping cached before invalidated. **Type**: A = host name (IP addr), NS = domain name (auth name server), CNAME = host name alias (primary/canonical host name), MX = host name (server to recv incoming mail – MX → Mail eXchange).

CDN (Content Delivery Networks): 2 sols for storing large files: **Store and serve many copies from many geographically dist servers** (CDN): **A** = clients closer to servers (lower latency). **D** = Lots of red. **Single Powerful Server**: **D** = server down → file inaccessible. Can get overwhelmed (run out of sockets/sys res) so get slow. Local network can become congested (switches connected to server become slow+overwhelmed so drop packets). Single location – clients may be far (high latency). **Main CDN Approaches**: **Enter Deep**: Place CDN inside many access networks (e.g. inside ISP's own networks). **A** = Close to users (low latency). **D** = Large num servers to maintain on many sites. Need to access other org's networks. Akamai. **Bring Home**: Place smaller num CDN servers inside large cluters at Pop (point of presence) locations close to (not in) access networks. Limelight.

CDN Performance: To lower latency, CDN Node (server) used must be geographically closest to client (shortest delay). CDN only see local DNS server's addrs (difficult to use). For some faster DNS services e.g. Google/Cloudflare's CDNs often pick sub-optimal nodes. Or client can be given list of CDN servers then pick best (ping to get latency). Netflix: hybrid (BH /ED).

Email: Text based (with attachments) comms. Asynch, one-to-many, multimedia, no auth (msgs can be forged/modified), no confidentiality (plaintext can be read by snoopers), no delivery guarantee (accidentally dropped /intentionally blocked). **User Agent**: allows users to read, compose, reply to, forward, and save messages. Often offer searching+sorting features (+ mult mailbox management). **Mail Servers**: accepts msgs for remote (sending) and local (receiving) delivery. Persistent storage of remote delivery msgs in queue, msgs for local delivery persistently stored upon receipt, user agent access local mailbox through access protocol. Addr found with DNS, MX type (mail exchange addrs).



SMTP (Simple Mail Transfer Protocol): Simple + old using TCP on port 25. Simple: 1. set up TCP/IP connection C→S, 2. C reqs server to accept msgs, 3. S resps, if accepting, C sends msg. Restrictive: lines must be <= 1000 chars, only supp ASCII (fixed with extensions). Insecure: very simple so easily spoofable (can be used by malicious parties). **Headers**, **Content**, . Ends content.

```
HELO example.com
MAIL FROM: sender@example.com
RCPT TO: receiver@otherdomain.com
DATA
FROM: Example sender
To: Example Receiver
Subject: fuck you
[empty line]
Hello fucker
.
QUIT
```

SMTP Headers: *To*: email addr of main dst(s), *Cc*, *Bcc*, *From*: sender name, *Sender*: sender addr, *Received:* added by transfer agent when recv by mail server, *Return-path*: return addr, *Date*: date and time email was sent, *Subject*, *Reply-To*: addr to send replies to (typically sender). Often Sender = From = Reply-To (2 can be left out). SMTP does not pcs msg content, only add "Received" header. SMTP only specifies header, nothing about msg content.

Extensions/Limitations: **SMTPS** (+Secure) adds encryption (uses STARTTLS instead of HELO, TSL/SSL). Same port (25), some servers use diff ports e.g. 465 or 587. **ESMPT** (extended): adds more methods for XML, HTML, and images. Uses EHLO to start. If receiver resp in correct way can use ESMPT otherwise send a HELO and use SMTP (or get disconnected). **MIME** (Multipurpose Internet Mail Extensions): use methods to encode non-ASCII as chars to send over SMTP. Types: text/plain (plaintext), text/html (HTML-formatted msg), image/jpeg (msg contains only image), multipart/mixed (msg consists of multiple parts).

POP3 (Post Office Protocol): to retrieve emails from mail server. Basic mail retrieval. Implicitly assumes retrieved mail deleted from mail server. Uses port 110 (unencrypted), or 995 (**POP3S** – encrypted.) **IMAP** (Internet Message Access Protocol): replaces POP3. Mail kept on server and read online, allows multiple mailboxes (backed up by ISP), gives user control over downloading mail. Can be encrypted (**IMAPS** port 993) or unencrypted (port 143, rarely used).

Feature	*POP3*	*IMAP*
Where is e-mail stored	User's PC	Server
Where is e-mail read	Off-line	On-line
Connect time required	Little	Much
Use of server resources	Minimal	Extensive
Multiple mailboxes	No	Yes
Who backs up mailboxes	User	ISP
Good for mobile users	No	Yes
User control over downloading	Little	Great
Partial message downloads	No	Yes
Are disk quotas a problem	No	Could be in time
Simple to implement	Yes	No
Widespread support	Yes	Growing

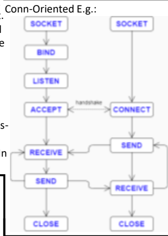
Other Protocols: **FTP**: File Transfer Protocol (exchanging files across Network. Can be combined with SSL (Secure Sockets Layer) (**FTPS**). **SSH**: Secure Shell Access (direct encrypted comms, also used to transfer files – **SFTP**). **Telnet**: plaintext direct comms for non-sensitive data exchange. **Crypto**: protocols such as Bitcoin Protocol (**BP**). **SNMP**: Simple Network Management Protocol (admin management of network+it's devices). **NFS**: Network File Sys (developed by Sun – bought by Oracle – enables file access over network). **DHCP**: Dynamic Host Configuration Protocol (allows networked devices to get an IP addr). **IRC**: Internet Relay Chat (live chat sys for chatrooms designed in 1988, rarely used).

Transport Layer: provides conn (TCP) and con-less (UDP) services to allow comms between end-systems /hosts. Connection decision made at this level, only runs on end hosts (not routers/switches). Requires lower layers in order to op (Network, Data-link, Phys). Protocols in this later work on ass that lower levels are working, but must consider that IP is best-effort (no data integrity/order of delivery guarantees). **Other Protocols**: **QUIC**: UDP based transport layer designed by google to replace TCP using multiple multiplexed UDP connections to improve HTTP performance. **UDP-Lite**: UDP-like conn-less protocol allowing potentially damaged data payloads to be propagated to app layer (allowing it to determine data integrity). **DCCP**: Datagram Congestion Control Protocol is msg-oriented protocol using reliable connection setup, close and has explicit congestion notification. **SCTP**: Stream Control Transmission Protocol msg-oriented based on UDP. **RSVP**: Resource Reservation Protocol - reserve network res to ensure quality of service.

Port Numbers: **Ports**: used to connect apps together/separate diff app's connections. Trans layer uses port nums to differentiate between diff network comms. Each app on host has unique port num. Port nums cross-platform (on diff devices/archs/OSs they are same for same apps). Ports → Use: 0:1023 → well known, reserved for certain protocols (e.g. SSH = 22), 1024:49151 any user app to use/register, 49152:65535 dynamic/ephemeral/private used by clients temporarily.

Berkeley Socket Interface: Interface adopted by UNIX sys + Windows. **SOCKET**: create new comms endpoint. **BIND**: attach local addr to socket (C+S bind trans level addr+name to locally created socket). **LISTEN**: prepare for/announce ability to accept n connections. S alerts kernel it is now waiting for connection from Cs. **ACCEPT**: block until some C wants to establish connection (S can now wait, recv req, choose to accept/deny conn). **CONNECT**: attempt to establish connection (when C connects, has to provide full trans-level addr to locate socket). **SEND**. **RECEIVE**. **CLOSE**: release connection, comms end when socket closed. In conn-less, LISTEN /ACCEPT/CONNECT not req.

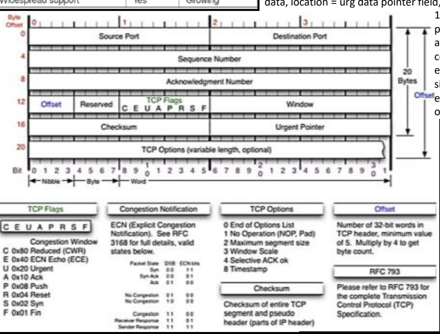
Conn-Oriented E.g.:



TCP (Transmission Control Protocol): conn-oriented. Data in segments. Reliable transfer (integrity and poss ordered delivery). Not secure. Can offer stream conns (ordered deliv, only accept segs in order). Congestion control (avoid destructive cong over network). Reqs handshake to start connection. Full-duplex (both sides can send/recv at same time). **Identifying Socket**: Use IP addr, port num + protocol (TCP/UDP): 61.195.17.146:80 TCP. **3-Way Handshake**: 1. Client sends TCP seg with SYN = true + init seq num, 2. server resp with SYN TCP seg + ACK = true + first unseen seq num, 3. client resp with ACK + first unseen server seq num + new seq num. **FIN**: FIN → ACK → FIN → ACK.

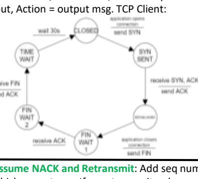
Segments: Wrapper for TCP data, transmitted within Network Layer protocol (e.g. IPv4/v6). **MSS** (Max Seg Size): max amt of app data trans in single seg (header size not incl). Usually related to MTU of connection to avoid network level frag (splitting segments into multiple packets). **MTU** (Max Trans Unit): largest link layer frame available to sender (largest unit of data that can be trans through all links to receiver without being split). **Path MTU Discovery**: determines largest frame that can be sent on all links from sender → receiver.

TCP Header: Src+Dst ports are 16-bit idents. Seq+ACK num (32 bits) used for reliable data transfer (ID segments). Recv window (16 bits) – amt data sent before ACK is recv (if rcvr cant pcs quick enough will ask to reduce TCP window). Header len determines size of TCP header in 32 bit words. Optimal/variable len field used to negotiate protocol params e.g. window scale/MSS. **Header Fields**: **URG**: 1b, signals urgent data, location = urg data pointer field, some software ignores. **ACK**: 1b, signal valid ACK num. **PSH**: 1b push flag (ask recv to push data to app immediately). **RST**: 1b, resets connection (often shutdown when error). **SYN**: 1b, synch flag. **FIN**: 1b signal shutdown. **Checksum**: 16b, err detection. **netstat –a** tcpView or currPorts.

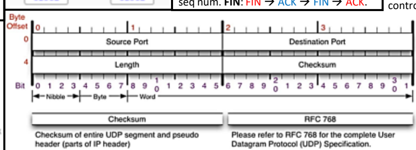


TCP vs UDP: UDP when speed vital and data can be resent. TCP when speed doesn't matter but want reliability. E.g. UDP PvP game, TCP movie player (buffer video to allow TCP to retrieve any lost frames or switch to UDP if slow).

Data Transfer: FSMs represent state of protocol → Transitions = event/action label, Event = input msg /timeout, Action = output msg. TCP Client:

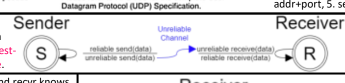


UDP: Conn-less trans layer protocol. Datagrams (≤ 65,507B (+ 20B IP Header + 8B UDP Header = 65,535B = max IP packet size)). In practice 500B-1KB to inc prop of intact packets. App ID provided ((de-)multiplexing). Integrity checked by CRC-type checksum. Simple: no flow/error control or retransmissions. Allows finer level of app layer control over when/what data sent. No connection est (faster) or related state stored. Small packet overhead (only small part of packet not payload). Useful in C-S (short msgs +quick resp, if fail resend, simpler code, fewer msgs needed). **UDP code**: **Client**: 1. make buffer, 2. make packet, 3. make socket, 4. send, 5. realloc buffer (new arr assigned to prev var), 6. get resp packet. **Server**: 1. make socket, loop: 2. new buffer, 3. get packet, 4. get client addr+port, 5. send packet.

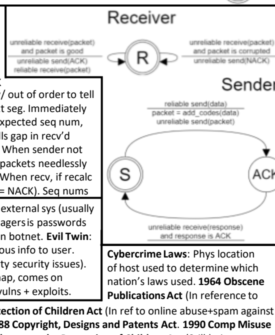


Seq Num: Each byte in data stream has seq num (indicates pos of 1st byte carried by that seg). When TCP connection set up, random Init Seq Num is decided upon to avoid leftover segs being received by mistake. When making new connections, even with same data, have diff seq nums. **ACK Num**: reps end of data recv'd, or first seq of data waiting to be recv. TCP ACKs can be cumulative (recv seg 1, 2, 3, ACK, wait for 4). Typically ACK every other packet. TCP is full duplex (multiple streams/seqs can be recv and ACK at same time). [ACK# = 101, Seq# = 200 | Data = "F"].

Data Trans FSM: TCP provides mechanisms to ensure reliability in data trans. IP in network layer is best-effort and unreliable – TCP reliable.



Error Detection: Recvr must be able to check if packet corrupted and tell the sender this, then retransmit corrupted packets. **Parity Bit**: simple err detection code where single bit is XOR of all bits in packet. **Stop-and-Wait with Err Detection**: Main issue: ACKs and NACKs can also get corrupted. If we use same scheme to reliably transfer ACK+NACK we have potentially no termination (loop of NACKs).



Assume NACK and Retransmit: Add seq num to each packet so can be retrains and recvr knows which are retrans. If use stop+wait only need 1 bit for seq num (0 = og, 1 = RT). **Seq Nums Instead of NACK**: If packet not ACK then recv use lack of ACK to determine to send more data. Note with TCP ACK contains start of the packet to be sent/resent next/end byte seq num of the data recv. **Out of Order Seq Nums**: sender may send many ACKS instead of stop-and-wait. *Delayed ACK:* recvr ignores packets out of order. When get packet in order wait 500ms then ACK (allows more in-order packets to be recv – resetting the wait). Send ACK when server being slow/ out of order to tell it which u have and need. *Cumulative ACK*: recv in-order seg with expected num, waiting on next seg. Immediately send cum ACK. *Duplicate ACK*: send ACK for next seg. Out of order seg arrives with higher than expected seq num, there is gap. Immediately send duplicate ACK. *Immediate ACK*: seg recv'd partially/completely fills gap in recv'd data. Immediately send ACK for lower end of gap (to fill). **Timeouts**: send timeout for recv ACKS. When sender not recv ACK in time, retransmits. If timeout too long, have to wait long time fo retrans, if too short packets needlessly retrans. Can do the bangbangbang technique. **Checksums**: used by TCP, calc from payload data. When recv, if recalc checksum not matchthen retransmit. ACK+NACK also protected by err recovery (corrupted ACK = NACK). Seq nums allow recvr to ignore duplicate segs.

Wireless TCP: TCP designed before popularisation of wireless. **Wired**: When packets lost, indicates congestion. Red packets sent. Use CA and recovery strats. **Wireless**: packets lost most likely channel reliability issue. Resend packets as much as poss, gives best chance of one being recv correctly. Fix conflicting reqs in 2 ways: **Split TCP connections**: use sep connections for wired /wireless so use diff algos for each. **Use base station**: wired base station do some retrans without informing wireless src. Base station tries to improve wireless IP reliability using TCP.

Detecting Congestion: **Congestion**: So far roughly described **TCP Reno**. Routers have limit to num pkts they can route, pkts held in queue. If too many pkts sent to router between sender&rcvr, queue overflow and some segs dropped, hence server assumes congestion when detect seg loss from: timeout (no ACKS), multiple identical ACKS can be considered NACK. **Congestion Control Algos**: CLAMP, ECN, FAST, BBR, Compound TCP, Vegas. **TCP Vegas**: pop TCP impl, attempt to detect cong before losses, predict pkt loss using RTT, larger RTT = greater cong. **TCP CUBIC**: Linux standard. To avoid adv smaller RTTS (can happen with Reno) grows window as func of time, not RTT. **Cong Window**: num bytes that can be sent before waiting for ACK. Sender+rcvr can define window size: **LastByteSent – LastByteAck'd ≤ W**, **W = min(CongestionWind, RecvWind)**, **max output rate = λ ≈ W/RTT**.

Congestion Methods: **Slow Start**: 1. Init window size = MSS (max seg size), 2. Every good ACK→inc window size (W) by size of data ACK'd (size roughly x2 every RTT), 3. Continue expone-ntial inc until reach ssthresh (seg size threshold), 4. Then use congestion avoidance. **Congestion Avoidance**: Window size inc roughly linearly (≈ 1 MSS per RTT). Each good ACK: $W = W + MSS^2 / W$. When congestion detected switch to diff strat. **AIMD** (Additive Inc/Multiplicative Dec): Every good ACK: $W = W + MSS^2 / W$, every packet loss: $W = W / 2$. **Timeout**: need to detect pkt loss when no ACK sent back. Timeout interval T > RTT (otherwise retrans unnec). T too large→too slow to re-trans. TCP continuously estimates RTT, sets T using smoothed RTT (SRTT) and RTT variation (RTTVAR). **T = SRTT + 4 * RTTVAR**. **Fast Retrans**: 3 dup ACKS interpreted as NACK. Num 3 tradeoff between fast and unnecessary. Timeout suggests congestion. 3 dup ACKS (in addition to og) suggests network can still deliver some segs on path. **Fast Recovery**: Given curr window size W^C, if timeout: 1. **W = MSS**, 2. Run slow start until $W = W^C / 2$ = **ssthresh**, 3. switch to congestion avoidance. If 3 dup ACKs then fast recovery: 1. $W = W^C / 2$, 2. congestion avoidance. FR fast as W not reset back to MSS, so can ramp up more quickly.

Window Strats: **Go Back N**: Sender trans mult segs without waiting for ACK. Sender can have up to W bytes of un-ACK segs in pipeline. Sender's state = queue of ACKs. When recv some ACKs, slide window along. **Sliding Window – Selective Repeat**: sender only retrans segs it suspects were dropped/corrupted. Sender keeps list of ACKs, recv keeps list of ACK's segs, when segs out of order, kept to be added to later data when gaps filled. Sender keeps timer for each seg waiting for ACK of (resend when timer run out). Sender slides window when lowest pending seg ACK'd. **Flow Control**: attempts to prevent recv from being overwhelmed/overflowing. Recv sends RecvWindow size along with ACKs. Typically size of buf left to fill. When buf full and recv can take no more, sends ACK with RecvWind = 0, and repeats 1B ping to sender to indicate it is not down/deadlocked, just pcsing.

Black Hat Tools: **Rootkits**: allow attacker to secretly enter external sys (usually installed with virus). **Keylogger**: record keyboard input and send to hacker, adv of passwork managers is passwords aren't typed so can't be logged. **Trojans**: allow hacker to remotely control entire sys as zombie on botnet. **Evil Twin**: hacker attempts to lure victims into using their network gaining info + potentially sending malicious info to user. **White Hat Tools**: **Tails**: portable OS usable from USB and never stores data (removes data integrity security issues). **Kali Linux**: OS designed for pen testing/other sec work. Comes with tools like Metasploit and Nmap, comes on ARM+WSL too. **Metasploit**: tool to automatically scan sys for vulns based on large DB of known vulns + exploits.

Black Hat Methods: **Credential Reuse**: using prev leaked credentials on many sites. **Packet Sniffing**: monitor networks traffic not intended for your NIC. **Code/SQL Injection**: using data input to get a sys to run your code (most commonly SQL injection to get data from DB). **Session/Cookie Hijacking**: stealing session cookie to be auth as someone else in ongoing browser sesh. **Wardriving**: searching for/abusing open/insecure WiFis. **Trashing/Dumpster Diving**: checking physical waste for useful info. **Clickjacking**: using hidden HTML divs/popups to force user to redirect to malicious dest. **Bait n Switch**: lure user to click seemingly legit ad. **Spoofing**: Falsifying ID to get packets for someone else (fake IP addr, change MAC addr, poison DNS cache).

Cybercrime Laws: Phys location of host used to determine which nation's laws used. **1964 Obscene Publications Act** (In reference to spam). **1978 Protection of Children Act** (In ref to online abuse+spam against c-ren online). **1988 Copyright, Designs and Patents Act**. **1990 Comp Misuse Act**. **1999 Amendment to the Protection of Children Act** (Still being changed). **2000 Freedom of Info Act**. **2000 Regulation of Investigatory Powers Act** (In ref to comp/phone surveillance). **2002 e-Commerce Regulations Directive**. **2003 Criminal Justice Act**. **2005 Disability Discrimination Act** (In ref to online abuse & spam). **2010 Amendment to the Copyright, Designs and Patents Act**. **2013 Defamation Act** (In reference to online abuse & spam). **2017 Digital Economy Act**. **2018 Data Protection Act**. In the US there is the **DMCA** (Digital Millennium Copyright Act).

Attack Egs: **Heartbleed**: bug in OpenSSL (impl of TLS/SSL) allowing users to revel server mem. **KRACK**: WPA2 used in WiFi – android devices could use 0-baed key so encryption useless. **WEP**: Wired Equiv Privacy is security algo for wireless networks (vulnerable).

Basic Security Concepts: **Access Control**: certain users can access certain res. **Authentication**: user and res know each other are who they say they are. **Confidentiality**: users can limit access to their res/info and limit access to see their traffic over network. **Data Integrity**: users cannot damage integrity of res (e.g. crash webserver by visiting). **Non-Repudiation**: User cannot deny comms occurred (secure logs held+can be audited). **Security Policy**: All companies/orgs need to define Network/IT Security Policy (useful standard: ISO/IEC).

H/P/V/A/C: **Hackers**: highly competent computer engineer. White hat = informs orgs of vulnerabilities before going public, Grey = only informs if paid, Black = malicious. **Phreaker**: phone hacker, network more digital now so most are hackers. **Virii (creators)**: Ransomware (encrypt, decrypt for ransom), Spyware (keyloggers, browser tracking addons, adware), trojans (software for botnet zombies, appear legit). **Anarchist**: politically active hackers (peaceful = hacktivists). **Crackers**: use hacker's tools to hack. **DDoSers**: participates in Dist Denial of Service attacks. **Spammers/Botters**: send unsolicited msgs (often ads) en-masse using botnets. **Warez**: ino piracy (software, images, videos). **Whistleblowers**: former employees expose malicious shit even when illegal (signed NDA). **Social Engineers**: compromise human security of org. Phishing = usually over email pretending to be someone they aren't. Vishing = voice msgs, Smishing = SMS, Catfishing = impersonation.

Standards Orgs: **IANA** Internet Assigned Numbers Authority, deals with DNS, IP Addressing and more standards. **ICANN** nonprofit organisations responsible for coordinating standards for the maintenance and running of namespace and numerical space databases for the internet. **IEFT** Internet Engineering Task Force, a collection of working groups (e.g routing, transport, security) concerned with developing the internet. **ISOC** Internet Society, dedicated to furthering beneficial use of the internet. **EFF** Electronic Frontier Foundation, a politically active nonprofit dedicated to defending privacy, free speech and freedom to innovate online. **W3C** The World Wide Web Consortium develops standards to help developers build tools on the web smoothly. **ISO** International Organisation for Standardization.

iptables: linux to set packet filters. Req root as interfaces with kernel firewall. **tcpd**: daemon controls access to unix services + monitor service reqs.

Network Usage: **Utilisation Factor = network use / max theoretical usage**. When we have **RTT**, packet size **L**, and trans rate **R**, can use time on connection used out of poss time len: $d_{trans} = L / R$, **Utilisation factor** = $d_{trans} / RTT + d_{trans}$.

Network Tools: netstat (see open ports), netcat (R/W data across network connections w/ TCP), Windows sysinternals (networking tool).

Logging and Auditing: keep logs for checking missed breach /attacks, forensics (to find who did attack), determine how sys was exploited, ensure good practices followed (no unsafe features used), detecting other network issues. Linux: syslog-ng. Logs at /var/log.

Access Control: user sends req over secure channel to guard which guards res. Assuming secure channel, guard determines: which users (principals) can access res, where principals can be located (e.g. user's IP is outside org's network), what reqs users can make for this res. Security can be difficult as many sys used by org can be diff (heterogenous sys), and users (inc admins /managers) can be careless (e.g. password reuse). **Access Control List**: Packet filtering rules (checked top→bottom until match).

Rule	Dir.	Action	Inside Addr.	Inside Port	Outside Addr.	Outside Port	Description
1	In	Block	*	*	9.9.9.0	*	Don't let these people in
2	In	Allow	*	*	6.6.6.6	*	We trust this host
3	*	Allow	1.1.1.7	300	5.5.5.5	300	Very specific access
4	Out	Allow	1.1.1.1	*	*	*	Allow this inside host access
5	Out	Allow	1.1.1.0	*	4.4.4.3	80	Allow access to this service
6	Out	Block	*	*	*	*	Block anything else

Proxy: makes reqs/resps on behalf of client, can filter in /outgoing traffic. **Normal**: client aware of proxy, connects to use it. **Transparent**: client unaware e.g. local router is proxy. No intervention required from client. **Reverse**: runs on recv side impersonating server+protecting it from external network (much like CDN load-balancing).

Bastion Host: server that expects to be attacked. Runs minimal trusted/secure OS, only essential apps (e.g. no window manager needed). All poss limits enables (read only filesys, no mounts, no user accts). Typically managed over dedicated terminal. Relays connections/maintains connection state, can auth users, can drop connections based on dest/incorrect connection packets etc (packet filtering). Acts as proxy firewall (in midst of logical connection allowing to monitor traffic, block/filter/report based on app-level msg content, scan for data leaks/virii/worms).

Firewall: security barrier between internal/external networks. Application Level Gateway: app that runs, checks reqs in app layer. Can also be proxy using extra rules to decide to share reqs/resp or send on reqs. Runs on single host and only protects that one host. E.g. SOCKS. **Proxy server**: protect entire LAN by making reqs and recv resps on it's behalf (can also cache res). **Circuit Level Gateway**: circuit of proxies, sending data between each node in circuit (e.g. Tor). Non-caching proxy (fully takes over host's comms with recipient and decides what to allow/block). **Packet Filtering**: filter w/ set of rules based on contents, src and dst IP addr/port, only allowing non-suspect through. Can be stateful (consider past traffic over some time). **Hybrid**: combination of all. Can be software or hardware based (hardware faster but more difficult to change if vuln found).

Firewall Avoidance: **SSH**: tunnel through with allowed protocol to use internal (get through firewall on ssh, send reqs through ssh to get resps). **Spoof MAC addr**: can re-write MAC addr if firewall blocking reqs based on it (black /whitelisting). **Spoof IP Addr**: stateful firewalls will detect. **VPN**: like SSH can tunnel through firewall. Provided secure tunnel, firewall can't decipher traffic.

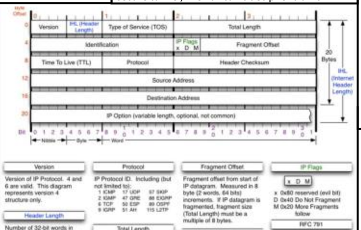
DMZ (Demilitarised Zone): area between you and the outside world (neutral zone). External hosts can only speak directly to internal hosts that lie within DMZ. All other non-DMZ hosts are hidden/protected by gateway/router/firewall. **NAT** (Network Addr Trans): rather than expose LAN IP addr of internal host, routers trans IP to their own pub IP to send (trans back when recv). **Port Forwarding**: to expose internal host to ext network without putting in DMZ, can set router to forward all packets arriving at given port straight to internal host. Useful for hosting servers/games.

Other Security: **IDS** (Intrusion Detection Sys): informs sys but does not stop detected intrusion. **IPS** (Intruction Prev Sys): actively prev ints (e.g. block SYN flooders), can work with IDS. **NGFW** (Next Gen Firewall): stateful firewall that comes w/ IPS/IDS sys. **UTM** (Unified Threat Management): similar to NGFW with added features e.g. spam filters/antivirus.

Cryptography: **M** = message, **K** = key, **E** = encrypt, **D** = decrypt. Ciphertext M_C = E(K, M). Plaintext **M** = D(K^{-1}, M_C). Given M_C should only be able to find M by brute forcing K^{-1}. Given M and M_C should be difficult to get K and K^{-1}. **Symmetric +Secret Key Encryption**: $K = K^{-1}$. **A** = Faster enc/dec than asymm. **D** = Must secretly disclose key to comm (secure channel). E.g. DES (Data Encryption Standard) – short key length, too insecure. **Asymm +Pub Key Encryption**: each user has public and private key. For *confidentiality*: sender encrypts with recvr's pub, recvr decrypts with their priv. For *signing*: sender encrypts with private key, recvr decrypts with sender's pub key. If successful then know msg was from sender. Can combine – encrypt msg including signed segment to verify sender. Combine with symm to sign symm encrypted files (e.g. check integrity - file not tampered with, GnuPG). **A** = Don't need to disclose priv info (more secure). **D** = Slower enc/dec than sym. E.g. RSA (uses difficulty in prime factor decomp).
Auth +Confidentiality: Encrypt/sign with private key: $E(K_t, M)$. Encrypt msg using dest's pub key: $E(K_r^{-1}, E(K_t, M))$. Proof only H_r may read it: $D(K_r, E(K_r^{-1}, E(K_t, M))) \rightarrow E(K_t, M)$. Proof only H_t could send: $D(K_t^{-1}, E(K_t, M)) \rightarrow M$.

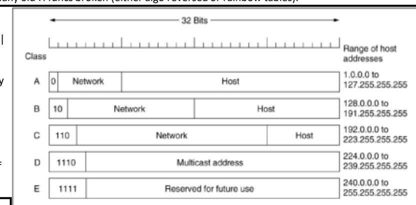
Diffie-Hellman Key Exchange: users A and B each pick a secret value (**a** and **b**), and each agree on a public value generator **g** and large prime number **p**. Each use their vals to calculate their pub vals which are exchanged: $x = g^a \bmod p$, $y = g^b \bmod p$. Then, use each other's pub val to get shared secret key: $x^a \bmod p = (g^b \bmod p)^a \bmod p = g^{ba} \bmod p$ (and sim for Bob). A and B never shared so eavesdropper can't know key. **Kerberos**: key dist sys for secret keys using trusted server. Auth you with pwd, can also auth user/res you intend to comm with. KDC gens ticket allowing for comms, can be used up to time lim, after which must get another. Originally vuln to Monster-In-Middle but has been addressed. **Hashing**: converts data to fixed-size alphanumeric string, same input → same out. Can't get og input from output. Used as checksum to verify data (e.g. fike contents unchanged, checking pwds – sites should only store hashed ver). Many old H funcs broken (either algo reversed or rainbow tables).

Network Layer: contains Internet Protocol, responsible for routing packets through internet across networks with diff hardware/protocol stacks. **IP**: main protocol in this layer. Datagram format, fragmentation, IP addr'ing, packet handling. IP Header: (note Type of Service now called DiffServ, most IP options not used - security issues).



Fragmentation: when data sent to IPv4 is larger than MTU (max trans unit) of output link it is being fwded through, datagram must be split. Frag at start or inter-med routers, only reass at dest (push complexity out of network). Each frag has 16-bit frag ID, each frag osset is offset in units of 8B (all frags mult of 8B + last byte). More frags bit (M) informs recv there are more fs on the way – set when intermed router frags a packet. **Max Frags**: not poss to fit max num of frags allowed by 13-bit frag offset (8192) inside IP Datagram/Packet. **[20B = IPv4 header (no options) | 8189 8B frags | 3 B final frag]**. Total len in IP header 16 bits, hence max = $2^{16} - 1$ = 65535B (65536 inc 0). Max amt data that can be payload is 65515B (max num 8B frags = **floor(65515/8) = 8189**).

IPv4 Addr'ing: Addr contained in 32 bits. XXX.XXX.XXX.XXX (XXX ∈ [0, 255]). Each IP addr assoc w/ interface (not host) so hosts may have +1 addr. **Classful Addr'ing**: no longer used. IP addrs split into classes with diff len prefix to denote org. Issue: all hosts on network must share same network addr sec (if org has hosts with several IPs, must pub-ly announce). [A | B | C] = max networks [16,777,214 | 65,536 | 254] = subnet mask [255.0.0.0 | 255.255.0.0 | 255.255.255.0]. **Classless Addr'ing**: Single network addr used for entire org, internally addrs divided into subnet addrs+host IDs. Ext routers only consider network addr, forward to router of assoc org. Subnet routers apply mask+check if IP in their subnet (otherwise forward to another subnet). Once host found, routers know which interface to fwd packets to. Network, subnet, and host can have diff sizes for each network – CIDR (Classless Inter-Domain Routing) = any-len prefix scheme. Routers match longest prefix. **Subnetting**: 192.168.1.10/17 → first 17 bits the same (mask = 17 1s: 15 0s), Network addr = lowest (192.168.0.0), broadcast = highest (192.168.127.225). Between = host. Subnet addrs must be contiguous. Use table to work out.



Terminology: **Network Types**: PAN = personal (phone connected to PC /Bluetooth speakers), LAN = local (home PC connected to home wireless network), MAN = metropolitan (city-wide e.g. subway digital signalling), WAN = wide (internet). **Devices**: Repeaters/Hubs L1 (repeat wireless network traffic to boost signal, no processing). Switches/Bridges L2 (make inter-connections based on MAC addrs which ID given NIC). Gateways/Multi Protocol Routers L3 (make decisions on forwarding packets/fragging based on IP addrs. To connect 2 IP based networks a gateway is required between em). **Internet Protocols**: **Internet**: collection of autonomous sys (separate networks run indep) connected by backbones (larger long-dist network infra to link networks). Designed in accordance with RFC 1958 (simplicity, modularity, scalability). Apps send data through connection(-less) trans layer protocol, trans layer creates TCP segs/UDP datags, network layer conv TCP/UDP → IP datagrams, data link pass datagrams between routers across networks, phys layer trans data. **ICMP** (Internet Control Msg Protocol): used for sending standardised control msgs (error signalling) in IP dgrams (e.g. ping → ICMp type 8, code = 0). Each msg has type (e.g. dest unreachable, time exceeded), and code (dst unreachable = 3, unsupp protocol = 2). **Dynamic Routing Protocols**: RIP (Routing Info Protocol), BGP (Border Gateway Patrol). Determine how packet routed through networks, create/manage routing+fwding tables.

NAT (Network Addr Trans): Attempt to solve IPv4 addr shortage – trans many priv IPs to single pub IP addr. Trans when packets leave/enter local network. On local, every comp gets unique IP addr. Managed with table of mappings between hosts+their pcs (L4 header contains this info). Criticism: violates IP model (each IP addr uniquely ID host), changed internet from C-less to C-oriented (router keeps track of connections+mapping), violates fundamental role of protocol stack (layers don't make ass about protocol above), can't easily support new transport protocols, many P2P require full connectivity between hosts (NAT can't provide) so 3rd party servers/TURN relays req'd. **Private IP ranges** (use in local networks): 10.0.0.0 → 10.255.255.255/8 (16,777,216 addrs), 172.16.0.0 → 172.31.255.255/12 (1,048,576), 192.168.0.0 → 192.168.255.255/16 65 (536). **Reserved IPs**: 0.0.0.0/0 = default route (when no other IP matches), 0.0.0.0/8 (host on this interface, used to acq IP addr), 127.0.0.0/8 (Loopback – localhost), 169.254.0.0/16 (Link Local – smth went wrong with acq IP addr).

IPv6: Intended to fix IPv4 addr shortage (≈ 3.8 x 10^{38} addrs) also: flow label used to set up pseudo-connection between src+dst, simpler header structure (red pcs'ing cost+ band-width usage), 128b addrs disp in hex (2001:630:12:600:1:2:0:10b), red router table size+ simpler protocol = higher performance, better security, better type of service supp (DiffServ), support scope when multicasting, support roaming hosts w/ no addr changes, better supp for coexinstence of new+old protocols (ez to dev new ones also). **Diff with IPv4**: frag done by end-sys, no header checksum (redundant as trans+data link have err detection), fixed len header easier to pcs (IPv4 options almost always unused), better modularity for extensions. **Extensions**: done by placing extending header after IPv6 one. Hop-by-hop opts, routing, frag, auth, encrypted payload, dest options.

DHCP (Dynamic Host Config Protocol): Allow host's inter-faces to safely be assigned IP addr. On boot host broadcasts DHCP discover pckt, listening DHCP server will resp with assigned IP addr. DCHP server can maintain static mappings (host → addr) + assign diff addrs each time host connects. Hosts lease IP, refresh periodically (prevent hogging IPs).

Routers: **Reqs**: provide facilities for moving data src→dst, multiple hops on nodes in network, consider topology of network to choose appropriate routes, load balancing, deal with network heterogeneity (diff networks connected together). Internet is packet-switched (C-less service), best effort (no delivery guarantees, max latency, bandwidth, congestion indication, or in-order delivery).

Datagram Networks: Potentially many diff paths for same src→dst, can be asymm (A→B not necessarily same as B→A). Routers use fwding table+final dst to determine which router to fwd pckts to.

Routing: **Sink Tree**: tree from src node to every dst, each path is optimal route (shortest path) to ech dst. Tree → no cycles. **Dijkstra's**: Each arc labelled with cost (e.g. delay, hops, some func of params potentially including congestion). 1. Visited = {}, 2. Add start node (dist = 0), 3. Loop while unvisited nodes: a) label each fringe node with min of (weight from visited node) + (weight of connecting arc), b. add closest node, 4. shortest path = dist each node labelled with. Routers co-op to find best routes between all pairs of nodes in network. **SPR** (Shortest Path Routing): use Dijkstra's to forward packets along shortest route. **Flood Routing**: Forward incoming packets to every outgoing link (except one recv from). Strats to avoid drowning network in packets: *Hop Counter*: discard pckt after reached max num of hops (need right num to avoid loss or drowning). *Forward Once*: if recv same packet again, don't forward again. **A** = solves packets being sent through cycles (A→B→C→A). **D** = reqs storing seq nums (how long for?). *Selective Flooding*: flood only in selective dirs (instead of every outgoing, send to some based on heuristic). Flooding always chooses shortest path (all explored in parallel) but significant overhead. Use case: Packet must be recv, but route to dst is unknown (req robustness). **DVR** (Dist Vec Routing/Bellman-Ford): Unlike prev 2 (static) DVR dynamic + takes into account network conditions (e.g. load). Every router advertises cost to each dst, routers use **Bellman-Ford Equation** = $D_u'[v] = \min_{x \in neighbours(u)}(cost(u, x) + D_x[v])$ to calc cost (cost from u to v is min of cost from u to neighbour + cost of that neighbour to v). **D** = count-to-infinity problem: when node goes down, routers continually update their cost based off each other's → cost keeps incrementing to infinity if one missing (A→C = B→C + 1, B→C = A→C + 1). Resolution: cost ∞ = longest acceptable path + 1. E.g. RIP. **Link State Routing**: Replacement for **DVR**. Broadcast all info on network topology to all routers, each can use to calc sink tree. *Basic Algo for Router*: 1. ID neighbours using "hello" packet to which neighbour resp with their network addr, 2. Link costs determined using "echo" packet and measuring round-trip delay, 3. build LSA (Link State Ad) describing router+connections to neighbours, 4. send LSA packet to every other router on network (flooding), 5. receive LSA from every other, 6. Router has status of all links in network, make sink tree. **A** = Algo allows better routes to be chosen using curr network conditions. **D** = routers redir traffic to best routes so often they become overloaded. E.g. OSPF. **DVR vs Link State**: [Thing | D | L] → [Network Info | Local | Global], [Computation | Global | Local], [Synch | Gradual (update+advertise) | Instance (once SPR comp done)]. **Hierarchical Routing**: prev methods difficult to scale as each router needs to know about each other. Network split into regions to solve this – diff algos used for intra- and inter- region routing. **A** = scale massively, can use diff algos in each reg, each reg can operate on their own. **D** = suptopimal routes chosen between nodes in diff regions. 2/3 levels of regs generally enough, consider each reg as diff networks. **Broadcast Routing**: another way to solve scaling – send to every host on network (only feasible for LAN/small WAN). *Send msg to each host individually*: not efficient, *flood routing*: acceptable providing we can limit flood, *multi-dest*: list of dst sent with pckt. Routers check list, splitting+fwding packet to it's neighbours. Packet must contain all dsts (size limitations). *Multicast routing*: sink tree at source+use? Sink tree must be spanning+routers need to agree on tree somehow. **RPF** (Reverse Path Forwarding): to construct spanning trees from router at low cost. Every router fwds/broadcasts a pckt on every adj routier, except one recv from. Routers only accept pckts if on direct path from source, hence paths of pckts fwd'ed+accepted represents spanning tree from src router. Can also be used to detect+prevent IP spoofing (packet will come from odd path given spoofed IP). **Multicast Routing**: send msg to subset of nodes (groups, each with GID). First sol to construct spanning tree at each router, prune all paths that don't contain members of group we want to send to. Alt: use **core based trees**: single spanning tree per group with a root (central to red cost between it and group members). OT send multicast msg send to core then that will retrains. **A** = scalable+lower overhead. **D** = not optimal for all srcs. Used by internet (multicast IP Addr/Broadcast addr effectively core for entire network).

Inter- vs Intra-AS Routing:

Inter-AS Routing	Intra-AS Routing
- Routing between autonomous sys (e.g. between 2 diff networks). - Autonomous syss can be hetero-geneous (diff protocols, routing algos, topologies, hardware), so use Gateways to link between them. - Can't support optimal routes at scale, but makes best attempt practical.	- Routing within autonomous sys (e.g. within LAN). - Within autonomous sys (dep on size) typically uses 1 design controlled by 1 organization. - Attempts to provide optimal routes on smaller network.

Ext→Ext: Gateway (inter-AS router) recv pckt, if can fwd to next G then does otherwise intra forwards to diff G to send on. **Ext→Int**: G recv packet, sends on intra routers to dst. **Int→Ext**: intra sends packet on to G that as it can reach dst, G then fwd to relevant G (routing across networks). **Int→Int**: intra routers route packet.

RTT = 2 * Propagation Delay
(**+ Transmission Delay** if object is large).

BGP (Border Gateway Protocol): Inter-AS protocol used on Internet. Adj routers maintain connections for reliability. Gateways transmit reachability info to routers inside an AS. Good routes determined based on reachability info+routing policies. Routers only check for +discover new paths if allowed. Use path-vector protocol (based on DVR but announce dists not paths). *Advertising Routes/Paths*: Dsts denoted using addr prefixes (subnetting). ASes may not propagate an ad by gateway, as doing so would imply network willing to carry traffic through AS. Routers can aggregate prefixes (merge prefixes together = **Supernetting**: 127.134.126.0/24 & 127.134.127.0/24 → 127.134.126.0/23). In BGP each AS has unique ID (ASN = Autonomous Sys Num) and several attrs: AS-PATH (seq of AS IDs through which ad was sent), NEXT-HOP (next IP addr to fwd packets towards advertised dst – resolves ambiguity when multiple AS reachable through multiple interfaces). BGP inport policy determines acc/rej route ads. Router preference ranked according to: policy used, shortest AS-PATH, closest NEXT-HOP router. Count-to-inf problem solved by path exploration/hunting (actively seeks paths), and routers can send withdrawal msgs (e.g. before taking node down tell others to remove path). Allows to ID invalid paths (at expense of some delays).

OSPF (Open Shortest Path First): link state routing algo to replace RIP (dist vec routing algo). Algo pub avail to be impl. Supports diff dist metrics (hops/delays etc). Can adapt dynamically to changing network topology (nodes + or -). Suports routing based on Type of Service. Supports load balancing (avoiding flooding). Offers some security features (some have been compromised tho). Supp hierarchical routing (can split AS into several areas then each has 1 or more area border routers which are backbone area – contain all border routers – to route traffic between areas). Abstracts collection of networks/routers/links into directed graph. Use same algos for areas+backbone.

Wireshark: Network protocol anal'er. Allows users to capture, anal + deconstruct packets to anal traffic on network. **Promiscuous Mode**: Works for wired+less, NIC does not drop packets (retains all recv'd). When wireless only listen to connected network. Some NIC ignore this (considered impolite+easily abused). **Monitor Mode**: Only on wireless networks. NIC listened on all networks in range/can recv from. WiFi networks secured with auth (e.g. pwd) will appear scrambled (encryption) unless supply network pwd to Wireshark (same for others e.g. RSA key for SSL or pwd for WPA/WEP). Most NIC do not support (many need new drivers/special NIC). WinPcap (Windows) does not support but AirPcap+Npcap (Linux) do. **Sniffin Efix**: when monitoring network make sure you have permission to. **Packet Capture**: *Hub*: local traffic, broad/multicast, (in prom mode) entire network. *Switch*: local traffic, broad/mult, (prom) network connected to same switch port. *WLAN*: local traffic, broad/mult (prom) entire WLAN, (monitor) all wireless pkts phys receivable/in range. **Display Filters**: http.request.method == GET && http contains "pwd" || ip.src != 10.43.54.65. eth = all ethernet based traffic, .addr = MAC, (eth.dst[0]&1 = multicast only, !(eth.dst[0]&2) = globally unique addrs only. **NMAP**: network scanning tool which sends raw IP pkts+monitors resps+determine services provided by network+it's hosts. Used to detect vulnerable hosts on network. *Quick scan, don't check ports*: nmap –sn <IP addr>, *scan range on host*: –p <start p>-<end p> <IP addr> (scan all just <IP addr>), *scan w/o discovery* (even if host don't resp to ping, can check it's ports): -Pn <IP addr>.

Device Terminology: *Repeaters/Hubs (Phys)* – boost signals by repeating all recv. *Switches /Bridges (DL)* – interonn decisions based on MAC addrs. *Multi-Protocol Routers/Gateways (DL)* – Fwds (poss frag) pkts using IP addr. Trans + App layer gateways also exist. Routers act as Gs to connect IP-based networks.

DL Layer Protocols: 802.3 Ethernet LAN 1-persistent CSMA/CD Star/Bus, 802.5 Token Ring LAN Token Passing Bus/Tree, 802.11 WiFi LAN CSMA/CA Cellular.

Ethernet: DL protocol used for L/M/WAN comms. Spec in 1980, IEEE standard 802.3 in '83. Og coaxial cable ≈ 2.94Mpbs, curr fibre optic twinaxial ≈ 100Gbps. **Cables**: UTP = Unshielded Twisted Pair = Most pop (Cat5e). STP = Shielded/Screened TP. FTP = Foiled TP. SFTP = Shielded +Foiled TP (Cate6, 7a, +8 in dev). Shielding protects against EM Interference (crosstalk), and protects against EM leakage that can be sniffed+exploited (Lantenna Attack).

Ethernet Pinouts: 2 main pinout wirings (Top to bottom: TIA/EIA 568B = W+O, O, W+G, B, W+B, G, W+Br, Br 568A swap O+Gs). **Straight-Through**: Comms between diff OSI layers (e.g. switch→router). Also called Medium Dependent Interface (MDI). **Crossover**: comms between devs on same level (e.g. switch→switch). Medium Dependent Interface with Crossover (MDIX). Order for B: W+G, G, W+O, W+Br, Br, O, B, W+B. **Rollover**: directly tap into network device (e.g. console to debug router setup issue). Order: flipped. **Fake-Swap**: swap between straight +crossover via software remapping (Auto-MDI(X)).

Topologies: **Switched Ethernet**: switches connected to each other+hosts, H only connect to Ss. Colls avoided by small coll domains. Ideal but expensive. **Internetworking Eth**: combi of networks sharing medium (e.g. cable), repeaters boost sig to extend range of network (longer cables). Hubs used (fwd recv frames out of every port, generally discouraged). **Bus**: data travels on bus coaxial cable, add new H: cable cut, BNC T connector added to connect host in, terminators at end absorb sigs (not refl back). **(Dual) Ring**: 2 NIC per H for Ring, 4 for Dual. Link removed→ network fails (every host single pt of failure) unless des to adjust (change flow, become bus). Single ring→1 way data flow, dual→2 way. Dual allows for one ring cut. Dual expensive→don't scale well. **Star**: all H conn to 1 S/multiport bridge. Any host comm with any other (provided mech to prev talking over each other). Central S single pt of failure. **Line**: split ring (bad), **Star-Bus/Tree**: stars with Ss in bridge w/ each other (hybrid). **Mesh**: useful sometimes (expensive tho). **Fully Connected**: ultra-fast (every H conn to every other), but expensive+nearly impossible to effectively manage. **FDDI** (Fibre Dist Data Interface): Ring-based token passing (pop in '90s). H div into 2 classes – A connected to both rings, B only 1. Optical fibre cabling allows geographically large networks (rings up to 100km long so FDDI must work w/ length 200km). Class B fail→data re-reouted through A hosts in 2nd ring. A fails→short circuit 2 As to make single ring (connect 2 rings at 2 disconnected ends). **Token Ring**: Physical: like star (central MultiStation Access Unit – MSAU). Logical: Ring (data flows by passing token around Hs in ring fashion). *Listen Mode*: Transmit input stream→output through single-bit buf. Keep copy on any frame addr to this host. *Transmit Mode (Has Token)*: Read input to mem, transmit frame from mem to next H, whole frame doesn't need to fit on ring (any len). Wait until fram recv by all in ring before passing token. **Early Release**: release tok before all recv frame (inc performance). **Token**: [Start Delim | Access Control | End Delim]. When no frame to be sent just pass token around circle. *FCS (Frame Check Seq)*: CRC checksum (ensure data integrity). *FS (Frame Status)*: def if addr was found+frame recv/copied from ring (set by list mode dst host) – when set can be removed from ring. *IFG (InterFrame Gap)*: gaps between frames (smaller = less ineff, need enough to regoc start/end of frames. Specified by protocol). **Priority+Reservation**: priority scheme can be impl so Hs only claim tok if pri lev of data they want to send is as high as tok's pri. H in list mode may have high pri data to send – tok created w/ pri of the reservation bits in the frame. Low pri data may be delayed indef, high pri always sent quick (good for real-time apps), used for LANs (ass can trust hosts to not abuse pri). **Tok Ring Ack**: Recvr can alter frame status: A = 1 (dst host working), C = (dst host correctly read frame). **Ring Maintenance Complexity**: frame control file used to create control frames, frames may become orphaned (e.g. never recv, infinite loop). 1 H = active monitor (responsible for gen toks+remove orphans). May fail so any host able to become AM (contention rules/protocol determine which). Prev points add sig complexity+red reliability. **Wiring Concentrators**: when host fails can be bypassed→improve reliability (have 2 links – 1 through host, 1 to next link w/o host).

Switch: Allow many devs to be conn to same subnet. Fwd msgs to ports based on MAC addrs (if can't determine, send to all). Use Forwarding Info Base (FIB) MAC table to remember addr assoc w/ port. Difficult to sniff. Connect networks by connecting to other swtiches/hubs. Replaced network bridges. To allow msgs to leave subnet, router+IP addr provided by DHCP or set statically. **Store+Forward Switching**: once whole frame recv, check w/ checksum, discard at switch if inv, fwding slower, supp by bridges+Ss. **Cut-Through Switching**: as soon as enough info recv (dst addr) start fwding. Only supp by Ss. Rcvr does err check.

Ethernet Frame: **Octet**: a byte/8 bits. Was useful when bytes used to be hardware-specific size. **Frame**: core concept of DL. Provides well-def interface to Network L for sending/recv pkts. ID trans errs with CRC (Cyclic Red Check) checksum. **F = [Header | Payload (Packet) | Trailer]**. Header = Pramble, Dest MAC Addr, Src MAC addr, Ethernet Type (Addr res, protocol (ARP) frame, IPv6 frame). Footer = has CRC checksum in.

WAP: IEEE 802.11 for wireless comms. 2.4/5GHz radio, acts as hub, can connect WPAs together to extend range, act as bridge to connect to wired network. Easy to sniff.

IEEE MAC Addr'ing: 48 bit (6B) addr etched into IEEE 802 conforming NICs as unique ID. <byte> : ... : <byte>. First 3B: OUID (Organisationally Unique Ident). B1b7 = Indiv/Grp (1 = unicast, 1 = multicast), B1b8 = Univ/Local (0 = Globally Unique, 1 = Locally U). *Broadcast Addr*: All F, group multicast, locally administered.

MAC: Can div DL layer into LLC (Log Link Control)+MAC (Media Acc C). MAC coordinates channel access. *Station*: host trans on shared medium. Wired →frame collisions = both need retrans. W-less→one trans may be stronger so one may be recv. **Strats**: *No Control*: frame not recv→station retrans whenever. Fine if channel util low, ineff when contention high. *RR*: stations take turns, used in token-based MAC sys. *Reservations*: stat res chann before trans, only trans for time interval res. Sys needed to manage ress. Used in slotted sys. **Static Channel Alloc**: fixed sched of times to trans. For n stations: *Time Div Multiplexing (TDM)*: S wait for time slot, rate limited to R/n, R = max chann rate. *Freq DM (FDM)*: S get lim freq band, each use B/n, B = total chann bandwidth (bad for large n/bursty traffic). **Dyn CA**: *ALOHA Protocol*: S trans whenever, if coll, S wait rand period before re-trans. **A** = Fair chann access. **D** = Low channel efficiency (large vuln period), if frame trans interr at any point both resent, max efficiency 18% at 50% load. *Slotted ALOHA*: Only trans on disc time intervals, man by synchronous global clock. **A** = red opps for new frame to coll with old, only coll with exact overlap (slot contention). **D** = max eff 36% at 100% load. When diag pure→slotted, start at soonest *after* slot.

CSMA (Carrier Sense Mult Access): **Carrier Sensing**: listen before trans, only trans when channel idle (red colls over ALOHA – frames not sent while another happening). Colls happen bc of trans delay – 2 stations see idle channel, both trans, sig not got to each other yet. **CSMA/CA** (Avoidance, WiFi). **CSMA/CD** (Coll Detection, Ethernet): stat listens to channel during trans to check for colls, trans stop+send jamming sig when coll (tells others abt coll). Host must trans long enough to tell the frame not collided – min frame len = 2n (n = end-to-end trans delay). Best-effort (no central auth controlling trans), suitable for most LANs, unacceptable for real-time sys (req max wait time, min bandwidth assurances). **Carrier Extension**: min frame size req (to hold channel until bits reach dst) so: [Header | Payload | Footer | Garbage]. Wasted time for garbage→ineff. **Frame Bursting**: Mult frames buffered then sent together (need CE between frames to sep). **Channel Back-Off**: *1-persistent*: continually check chann, trans as soon as free (Ethernet, aggressive). *Non-p*: if chann idle trans imm else wait rand period of time before check again (non-agg). *P-p*: continually check chann, if free trans with prob p (strat between 1 and non). **Binary Exponential Back-Off**: used when network load high. Slot len = min frame len. If coll in trans, wait 0/1 slots before trying again. After C colls, wait $2^C - 1$ slots (up to lim 1023 – 10 colls). High contention→many colls→bin exp→re-trans attempts spread out→fewer colls. **Medium Acc through Token Passing**: single token, stations only trans when have. Token trans with token frame, pass immediately if no frame to send, otherwise set timer and trans until time/frames expire then pass. Ethernet more pop so is standard. **Avoiding Wired Colls w/ Switches**: buff frames+retrans when chann (Ethernet cable) ava-il. Each only 2 stations (host+switch), hosts trans simul, switch recv+fwds frames. Max cable len determined by sig strength. Switches act as repeaters. **ARP** (Addr Resol-tion Protocol): to comm outside of network need IP addr (static manual/dyn by DHCP). Switches don't use IP (that is network). IP spec hosts on internet – no translation when passing through router (can with NAT tho). MAC addrs for hosts on same network/subnet, typically don't change when passing thru routers as pkts. Use ARP for IP→MAC. 1. Router: ask all hosts if have given IP addr (DL frame+broadcast), 2. Host: check if has addr, if so sends it's MAC addr, 3. send IP datagram (in DL Frame) with info, usually cache mapping. Can optim by collecting all at boot. **ARP Cache Poisoning**: send spoof ARP msgs to assoc their MAC with victim's IP (thus rcving their IP datagrams).

Wired Transmission: **UTP** (Unshielded Twisted Pair): 2 wires twisted together. Cheap+ez to mass-prod, twisting red interference+crosstalk, used in telephone syss. CAT1 1Mbps (voice grade for POTS), CAT6 1,000 Mbps (1000Base-T Gigabit Ethernet). **Coaxial Cable**: Conductors placed concentrically (one inside other) separated by insulator [Conductor | Insulator | Conductor]. Good shielding (EM field mainly between inner/outer conductor), large bandwidth from high freq range, higher cost per m (hence UTP more pop for common consump). **Optical Fibre**: transmit data using light+refraction. Single optical fibre = 2-125 micro m in diameter. Attenuation (sig loss) low so can use for long dists. High bandwidth. 2011 26Tbps, 2014 255 Tbps, 2021 1000 Tbps.

	Freq Range	Attenuation	Delay	Repeater Spacing
UTP	$0 - 1\ MHz$	$0.7\ dB/km\ @\ 1\ KHz$	$5\ \mu s/km$	$2\ km$
Coaxial Cable	$0 - 500\ MHz$	$7\ dB/km\ @\ 10\ MHz$	$4\ \mu s/km$	$1 - 9\ km$
Optical Fibre	$186 - 370\ THz$	$0.2 - 0.5\ dB/km$	$5\ \mu s/km$	$40\ km$

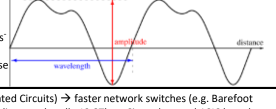
Physical Layer: **Network Arch**: Architect→design network (topology, standards, connections, where to put cables). Engineer→installs equip to setup network. **Patch/Socket Panel**: telephones → socket panel (cables arrive here) → network switch (creates LAN) → Private Branch Exchange (interlinks phone sys). PBX (Private Branch Exchange) used for phones, but if IP based sys separate PBX not needed.

Wireless Transmission: using EM radiation (usually radio). No wires (expensive+take time to install), bidirectional comms by default, typically broadcast (all/most recvrs can see trans). Inverse sq law: sig strength red with range. Environment degrades signal (interference obstruction, reflection of sig).

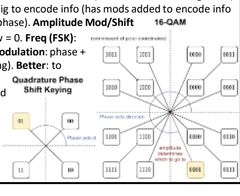
Information Representation: **Digital**: discrete info, rep by finite num of states. **Analogue**: continuous info rep by changes in some phys state (light intensity, voltage). **Baud Rate (Bd)**: Symbol rate per s for digital channel, where symbol may rep >1b. How many times per time unit the symbol/waveform changes – interlinked w/ bit rate. Depends on medium. **Binary Digital Channel**: dig chann which uses 2 symbols/states (1 and 0).

Modem: Modulator-Demodulator impl digital chann using analogue chann. $H_1 \leftrightarrow_{digital}$ Modem $\leftrightarrow_{analogue} \dots \leftrightarrow_{analogue}$ Modem $\leftrightarrow_{digital} H_2$. D → DAC → A, A → ADC → D. **Codec**: Code-Decoder impl analogue channel using digital.

Waves: **Amplitude**: max displacement/sig strength. **Wavelen (λ)**: Len of single cycle. **Period (p)**: time taken to complete 1 cycle. **Freq (f)**: num cycles per sec. $p = 1/f$, **wavespeed** $= f\lambda$ (for radio WS = c = 3 x $10^8 ms^{-1}$). **Phase**: 2 waves with same λ&f, offset = phase diff (angle units – 360° / 2π). Max phase diff = π (opp displacements during cycle).



Modulation: mod scheme changes some info sig into another more suitable for transmission. **Baseband Mod**: transmit unmodified (dedicated line sending in full). **Broadband Mod**: basic carrier sig to encode info (has mods added to encode info e.g. changing amplitude, freq, phase). **Amplitude Mod/Shift Keying (ASK)**: high amp = 1, low = 0. **Freq (FSK)**: high freq = 1, low = 0. **Phase Modulation**: phase + π = 1, normal phase = 0 (see diag). **Better**: to improve data rate, transmit multiple bits per symbol (in mod scheme). Use more phase diffs/amplitudes or use combi of both e.g. QPSK/QAM.



Digital Subscriber Line (DSL): with V.90 Modem Standard, use conventional phone lines to transfer data. Max 56,000 bps downstream (download), 33,000 bps upstream. Limited→phone lines lim to 3,000 Hz bandwidth (human voice lim, phones dev for human voice init). Anything outside filtered as noise. By removing lim (by removing bwidth filter) DSL allows more bwidth and higher data rate. Noise becomes limiting factor. **Asymm DSL (ADSL)**: 1.1 MHz of bwidth div into 256 4,000 Hz channels. Chann 1→5 (4 →25 kHz) unused to avoid interference between voice+data channels. Voice = 0→4 kHz. V.24 mod uses 224 downstream channs (13.44 Mbps). ADSL splitter separates voice from data bands, ADSL modem does mod. [Voice 0→4 kHz | Unused 4→25 kHz | Upstream Channs then Downstream Channs 25→110 kHz]. DSL Access Multiplexer (DSLAM) typically owned by ISP connects local telephone cables to ISP. ADSL2 12 Mbps 2.2 MHz, VDSL 52 Mbps 12 MHz, VDSL2 200 Mbps 30 MHz (curr pop).

Network Simulation: used to des networks cheaply. *Cisco Pkt Tracer*: strong academic backing, *gns3*: strong, open comm backing, *OPNET*: professional use, quite technical. Cisco: IOS commands, terminal commands inside apps on Desktops/Laptops, web docs, python+JS. **Network Programming**: **Simple Echo**: 1. Run server, waiting for connections on a user-defined port, 2. Client connects to the port, 3. Server listens for input from the client, 4. User types into client, client sends message to server, 5. Server echoes recieved data back to client, 6. Client disconnects, 7. Server closes.

Future: **Faster Hardware**: use of ASICs (App Specific Integrated Circuits) → faster network switches (e.g. Barefoot Networks → high speed Ethernet ASICs with prog-able pipeline, can handle 12.8Tbps, Cisco also vend ASIC based gear). Using light as medium for secure comms, better fibre optics. **Faster Wireless**: Kumu networks dev'd prog-able fileters to allow wireless devs to cancel out their own transmissions (allow full-duplex wireless – can recv+transmit simult on single channel). **Legislation**: Net-Neutrality laws in US allow ISPs to be selective about services provided for content on Internet (e.g. slowing down competitor's website). **Wireless Mesh**: many wireless devs form mesh networks, e.g. Cisco meraki allows networks to self-heal when parts (e.g. switches) fail by wirelessly rerouting data. **SDN** (Software Defined Networking) and **NFV** (Network Functions Virtualisation): network arch where apps+services abstracted from network infra+control. Useful for containerisation (being dev by Nicira, Cisco + others). **Web-Decentralisation**: internet is centralised around large CDNs (Amazon, Google, FB etc) – bad for reliability if a few backbones go down. New protocols e.g. **IPFS** - users can aide decent by using own domains, storage (instead of Google drive etc) and own hosting services. **Jobs**: Engineer specialising in managing comp networks, typically w/ expertise in infrastruct, virtualisation (e.g. VLANs), servers, switches, firewalls, meraki, WatchDog. Certifications include: CCNP (cisco professional level cert), CISSP (cybersec competency cert), RHCE (Red Hat Certified Engineer).