1) Computer Networking: Interconnecting computer systems via telecommunications methods to share data and resources.
1) **Pervasive:** Networks are everywhere.
2) **Distributed:** Most mainstream software systems are spread out (cloud computing).
3) **Performance** can depend on **network usage.**
4) **Arpanet:** 1969 first internet connection. Connected US Universities. First message was "login", but it crashed after "lo". It sent the result after rebooting an hour later.
5) **Jobs: Network Engineer/Architect**: Design, build, maintain NWs. **Server App Dev:** Works on server backend for cloud apps. **Network Software Eng:** SWE oriented on NWs. **Data Center / Cloud Platform Admin:** NWs/Cloud Comp. **Network Security Eng:** NWs + Computer Security.

1.1) Network Stack: 1) Application Layer – send & receive data in a format they specify. Implementation (OS, packet type, network setup) abstracted away. Apps use **Protocols,** which define structure of data (requests & responses), port numbers and other conventions. Example: WWW, part of the internet made by TBL, uses HTTP. Web browsers send GET requests to web servers, which receives, processes, and sends a html page response. The browser receives, processes, displays. **HTTP** exists in the app layer of **TCP/IP**. 2) **Transport Layer** – Establishes basic data channels, taking data to be sent/received and converting to/from data packets. **Two types of Network Connection**: 1) **Connection Oriented TCP:** UnACKed packets resent. 2) **Connectionless UDP:** No checking, packets sent at once, + performance. **3) Network Layer** – Adds IP Addresses and other info to packets, routes them through a mesh network of hosts to reach destination. Path taken frequently changes and is per packet. **4) Data Link Layer – NIC** controls communication standards to allow phys. comm. of data to transfer packets between devices. NIC acts as an intermediate, to and from our underlying 4G/5G/WiFi connections, or Ethernet/Coaxial/Fiber Optic. **5) Physical Layer** – the hardware that transfers data.

1.2) The Internet: Packet: Formatted unit of data.
1) **Packet Switching: Switches/Routers** operate on individual packets, receiving and forwarding them (+has a processing cost) to other S/Rs depending on info in packet (-extra space in packet). Different Packets take different paths +avoids slow/disconnected paths. +No setup cost. -Cannot guarantee quality of service. +High network resource utilisation. **Example: The Internet.**
2) **Circuit Switching:** -Expensive; path is specified and connected. Connection maintained for communication duration. +No processing or space cost – data is sent straight down the link. -If link becomes slow or breaks, must obtain new one. +Quality of service guaranteed as we reserve resources (-but nobody/nothing else can use this line meanwhile). **Example: Landline telephones.** (modern phones are digital, use VoIP (voice over IP)).

1.3) Internet Protocol Stack: Protocol: Rules determining data transmission between devices. Executable, unambiguous, complete.
1) **Handshake**: Establish identities, and context to begin comm.
2) **Conversation**: The communication, done as specified by protocol
3) **Closing**: terminates convo, cleans up/notifies other.
We use the **5 layer model**: Application, Transport, Network, Data Link, Physical. There's also 7 layer OSI – with Presentation and Session between Application and Transport and 4 layer OSI (AKA **TCP/IP** stack), which flattens the bottom two layers into "Network Access" layer, and uses "Internet" rather than "Network" layer.
Service – If we are offering the HTTP Protocol then we offer a web service. We are a web server.

1.4) Internet Protocol Design & Implementation
Must consider **Addressing** (denote recip)**, Error Control** (detect/correct)**, Flow Control** (fast sender shouldn't swamp slow receiver)**, De/multiplexing** (support parallel comm)**, Good Routing.** Network Layer usually supports **connection-oriented** (setup connection with client, transmit data over channel, e.g. TCP or IP) or **connectionless** protocols (send data without formal connection – UDP on IP), or circuit vs packet switching.
When we send data in the protocol stack, the top layers send data down their stack, until the bottom layer sends it to a receiver who then processes it up their stack.
1) Application Layer in IP: Defines app functionality and message formats. e.g. **Old:** DNS, SMTP, FTP, Telnet, HTTP/S, SSH. **New:** Middleware supporting distrib. Systems (Java RMI, Apache Thrift), **High-Lvl:** e-commerce, banking (visa). **P2P:** BitTorrent, old Skype.
2) Transport Layer: offers connection-oriented/less protocols. Provides network interface via sockets. Supports secure conn. – send data **reliably, in order**. Supports **datagrams (UDP)**. **Flow Control.**
3) Network Layer: Deals with Routing & Congestion. Best Route, deal with connections going down, multi-casting/broadcasting, packet dropping when overloaded.
4) Data Link Layer: Reduce, detect, and rectify bit transmission error. **Parity Bits**. Also specify how compu-ters share common channels (**MAC** Addresses). Specify how networks connect (e.g Ethernet, FDDI, token rings) **5) Physical Layer:** Describe raw bit transmission in terms of mechanical/electrical issues.

1.5) Network Performance
KiloByte = 1000 Bytes. Kilobit = 1000 **Bits**. *Kibibyte = 1024 Bytes.*
Formulae: L: bits, t_0: time when data sent, t_1: time when first part received t_2: time when data wholly received.
1) Throughput = transferred bits/time (link bandwidth):
$R = L / t_2 - t_1$. **Actual value, Bandwidth** is just maximum possible throughput. Input throughput=output.
2) Latency = time taken for a single bit to be transmitted (propagation delay): $d = t_1 - t_0$.
3) Packetization: transmission/store&forward delay: L/R
4) Transfer Time: send time per bit: d + L/R (latency+packzation)

1.5.1) Processing Delay
1) When we send data to a router we have **Processing Delay** (it does some work/computes route), small & **Queueing Delay** – has to do other work first, packet might wait for a long time / get dropped.
R = link bandwidth (b/s), L = packet length (b),
a = average packet arrival rate.
2) Traffic Intensity (TI) = La / R.
Small Delay ⇔ TI ≈ 0. Large ⇔ TI ≈ 1. TI > 1 ⇔∞ Delay, packets will be dropped. **2) The Web:** Internet Apps are end system apps/processes. Processes run on diff hardware/OS

2) The Web: Internet Apps are end system apps/processes. Processes may exchange messages which act as inputs to others. Processes run on diff hardware/OS, but they must be able to address one another to communicate. Protocols give layer a of abstraction. End systems might have multiple processes that are networking – we distinguish them by their **port number** (used by TLayer).
Between two communication processes there are two roles:
1) Client: Initiates communication. If on a connection oriented service, the client establishes this connection.
Using Sockets: Creates Socket C by connecting to server, Use C by writing/reading to/from, Disconnect and destroy C.
2) Server: Waits to be connected to/for communication. If on a connection oriented service, the server passively accepts conn. reqs. If apps have processes acting both as Client and Server, it's P2P arch.
Using Sockets: Create Socket S by accepting connection on port P. R/W data from socket to use it. Disconnect and Destroy S

2.1) World Wide Web: Based on **hypertext/links.** Glorified FTP, transfers **plaintext web files**. Success bc of: simplicity of HTML and HTTP (Stateless Protocol). Easy to learn/use, GUI Browsers.

2.2) Web Terminology:
Document: A webpage, a website containing several.
Objects: A file, documents may contain several (HTML, JS, video).
URL: Uniform Resource Locator (specifies the address of an object).
Browser: Program to request, receive docs and process the document to display graphically.
Web Server: An application containing document and objects, serving them to clients over HTTP.

2.3) HTTP: Use connection-oriented transport(TCP)tho works w/UDP.
Stateless, each request and response is a single unit, if a request is dropped no others are affected.
1) HTTP /1.0: Uses one TCP connection per object. Inefficient, requires many objects to be spawned and destroyed.
2) HTTP /1.1: Most popular. Same TCP connection issues multiple requests and responses. Default is persistent communication, but request with "Connection: close" closes it after reqs/responses sent.
3) HTTP /2: Faster. Expected to replace 1.1. Exchanges content in binary, more compact & faster. Fully multiplexed, **allows pipelining.** Can use a single TCP connection with requests in parallel.
4) HTTP /3: Uses UDP for exchanges, faster. Still in development.

2.4) Anatomy of a HTTP Request / Response
1) **Request Contains**: Protocol version, URL specification, Connection Attrs, Content/Feature Negotiation.
2) **Response Contains**: Protocol version, reply status/value, Conn. Attrs, Object Attrs, Content Specification (type, length), Content (objs)
HTTP Methods: GET: retrieve object using URL. **POST:** Submit data to server (e.g a form, message). **HEAD:** Like get, but only recieve the header, used for testing link validity.

Example Request:
```
GET /~kgk/212/index.html HTTP/1.1 <- Request line
Host: www.doc.ic.ac.uk          }
User-agent: Mozilla/5.0         |<- Header lines
Accept-Language: en-GB          }
                                <- Empty line, followed by possibly empty object body
```
Example Response:
```
HTTP/1.1 200 OK <- Status line
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32) <- Header Lines
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
                                <- Empty Line
<html> <body>
<h1>Hello, World!</h1><- Object Body
</body> </html>
```

2.5) Status Codes:
1xx Informational.
2xx Successful Operation (e.g 200 → OK).
3xx Redirection (object has moved temporarily or permanently).
4xx Client Error, e.g 401 (Unauthorized), 404 (Object not found)
5xx Server error, e.g 500 (internal server error), 503 (overloaded). We can use telnet to send plaintext commands to a server listening on a specific port (80 for HTTP):
```
> telnet www.imperial.ac.uk
> GET /computing/HTTP/1.1
> Host: www.imperial.ac.uk
```

2.6) Web Caching – Proxies can be used to cache requests.
The proxy simply gets the request from the client, sees if cached, if not request from server, store in cache for some time. Sends response.Reduces Latency, network traffic. Improves security, as server only interacts with proxy. Latency associated with finding entries/caching, extra complexity, need optimal cache refresh time.
HTTP can determine how each method should be handled (only cacheable if indicated by **Cache-Control / Expires**. OPTION requests aren't cacheable). Cached pages can become **stale,** a HEAD request sees if an object has been updated (cache needs refreshing). We specify **expiration time** with the Expires header or max-age directive of **Cache-Control**. Clients can use conditional GET with an If-Modified-Since header. Example use of header: 1) Cache Control: no-cache, or 2) Cache Control: max-age=100, must-revalidate

2.7) HTTP Sessions – HTTP is stateless, but sometimes we need stateful applications (e.g save shopping cart). We can do this with the **Set-Cookie <NUM_ID>** and **Cookie <NUM_ID>** headers. We do this by sending a request, then set-cookieing it with an ID we choose (and thus we know it). Later if we enter that cookie ID, we have a match, it returns the old value which was stored in DB. When we logout, the cookies are deleted usually, but some websites keep them.

2.8) Dynamic Web Pages: Instead of storing and serving static webpages, we can generate webpages for a specific request on the fly.
Common Gateway Interface: Allows a program to identify params from URL: https://www.site.com/page.html?name=rob&age=9&day=mon
Servlets: A Java solution to state, webserver creates new instances of the JVM to run & process requests for each client connecting. An alternative approach is to **execute code on the client side**.
PHP is server side, generating pages to be sent to the client.
Javascript is client side, the code is sent to the client (embedded in the web page) and run on the client side to create the page.
```
<?php echo "Hello World!" ; ?>).
document.write("Hi World!");
```

2.8) Domain Name System (DNS)
IP Addresses: Uniquely identify end systems by addresses. IPv4, 32 bit. IPv6 128 bit. It's an easy format for routers, but not for users! Before 1983 we had a file mapping mnemonics to IPs locally.
DNS: distributed lookup facility mapping hostnames to IP addresses. www.imperial.ac.uk, the top level domain is "uk". We traverse backwards into its subsets to find our domain.
Root Servers: Each top-level domain is (e.g .com, .edu, .org) associated with one of 13 root DNS servers, from 12 indep. groups.
Top-Level Domain Servers: A DNS server associated with a top-level domain.
Authoritative Servers: For each domain, a server holds the master copy mapping all public hosts within that domain.
Most root servers, and lower level ones are implemented as a **distributed** set of machines. **Distributed** copies of DNS maps (often updated) help load balance, reduce latency, add redundancy.

DNS Caching – reduces load on DNS infrastructure, improving performance. Cache needs to be updated often. Be wary of **Cache Poisoning** – (DNS spoofing) entering incorrect mappings into a DNS cache to direct users to the wrong site.

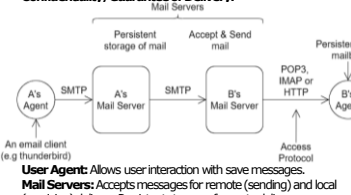
DNS Features – Each entry in the DNS is a **Resource Record** describing a translation of a name. these are cached entries.
```
Name                    Value           Type    TTL - Time to Live
www.imperial.ac.uk      146.179.40.148  A       ...
```
TTL, obvious. **Type A**: host name → IP Address,
NS: domain name → authoritative name server,
CNAME: host name alias → primary/canonical host name
MX: host name → server to get incoming mail (MX: Mail exchange)

DNS Protocol – **Connectionless**: Runs on UDP on port 53. Getting hostname translation takes two packets (request with name & reply with the value), so setting up a TCP connection is not worth it.
Messages: Has query/reply msgs with identifiers to associate msgs.
Same Format: Queries/Replies have same basic format for simplicity.

Round Robin DNS: Load balancing technique for geographically distributed web servers. Requests balanced across multiple servers.
1.DNS server requests hostname translation from **authoritative** DNS server.
2. DNS request receives list of IP addresses in response.
3. DNS server round-robins through addresses to distribute requests.
TTL (<18s) should be low so the list is DNS server updates the list often.
nslookup, tells us about this: we receive a line telling us the DNS server used, and then one with the IP. **Non-authoritative**: reply has been extracted from a **previous cache** nslookup -type=NS imperial.ac.uk tells us the address of the authoritative DNS server• nslookup www.imperial.ac.uk ns0.ic.ac.uk asks the server ns0 (the authoritative DNS) for a reply. **dig** queries name servers.

2.9) Content Delivery Networks: When storing large files:
1) Store on 1 powerful server: No redundancy. System can get overwhelmed. Latency can be high (far clients, as 1 geog. location), local net can become congested.
2) Store and server many copies from geog. distrib. servers: Redundancy, less latency as clients will tend to be closer to servers
Approaches to CDN: 1) **Enter Deep:** Place CDN servers inside many access networks (e.g inside ISPs own networks). Low latency, close to users. Large number of servers to maintain on many sites. Need access to other organisation's networks. Example: Akami
2) **Bring Home:** Place a smaller number of CDN servers in large clusters at **Pop** (point of Presence) locations very close to, but not inside, access networks. Example: Limelight (has 123 Pops).
We use the closest CDN to the client for min latency to serve resource.
Problem: The CDN doesn't know about the location of the client – only the DNS server taking their request, which could be very far (as we pick the fastest – which could be google's superfast/far DNS servers). Alternatively the client can be given a list of CDN servers, it can then pick the best (by pinging to get latency) & then choose the best (used by Netflix, who uses to be on AWS but now have their own CDN which uses both bring home and enter deep).

2.10) Email: Asynchronous (send msgs to offline users), **One-to-many** (same, many recipients), **Media, No Authentication/ Confidentiality/Guarantee of Delivery.**



User Agent: Allows user interaction with save messages.
Mail Servers: Accepts messages for remote (sending) and local (receiving) delivery. Persistent storage of remote delivery messages in a queue. Messages for local delivery persistently stored upon receipt. User agents can access local mailbox through access protocol.
Address is found using DNS, the MX type is for mail exchange addrs.

2.11) SMTP: Simple
1. Set up TCP/IP connection from client to server.
2. Client requests server to accept messages.
3. Server responds, if accepting, client sends message.
Protocol is only cares about sending the message to the mail server.
Restrictive - Lines must be ≤ 1000 characters and only supports ASCII (7 bit characters) (this has been fixed with extensions).
Insecure - As it is very simple, so easily spoofable and can be used by malicious parties

Anatomy of an SMTP Request: (each | is a newline). It is a TCP connection which sends this data:
Commands: HELO example.com | MAIL FROM: sender@example.com | RCPT TO: receiver@otherdomain.com | DATA | FROM: Example Sender | To: Example Receiver | Subject: Hi! | <content> Hello World! | <dot-ends-content> . | exit email QUIT
Since a single dot ends an email, to send an email with a single dot you actually have to send "..."
SMTP Headers: To, Cc, Bcc, From, Sender, Received, Return-path, Date, Subject, Reply-To.
SMTP Extensions: SMTPS (secure – adds TSL/SSL. We use STARTTLS > HELO, port 25).
ESMTP adds methods for XML, html, images. Use EHLO > HELO, if they respond EHLO we use ESTMP, else fall back to SMTP. **MIME** uses provided methods to encode non-ascii as ascii characters to send over SMTP. **MIME** types include: **1)** text/plain, **2)** text/html, **3)** image/jpeg (only image), **4)** multipart/mixed - message consists of multiple parts.

2.11) POP3: Retrieve emails from the mail server. Implicitly assumes retrieved mail is deleted from the mail server, uses unencrypted port 110 or 995 POP3S; encrypted.

2.12) IMAP: Replaces POP3. Mail is kept on the server, and read online. Allows for multiple mailboxes, backed up by the ISP. Gives user control over downloading mail. Can be encrypted (IMAPS port 993) or unencrypted (port 143, rarely used).

2.13) Other Protocols: FTP, SSH, Telnet, Crypto, SNMP – Simple Network Management Protocol used for admin management of network and its devices, **NFS** – Network File System, enables file access over a network, **DHCP, IRC.**

3) The Transport Layer: Provides Connection (TCP) and Connectionless (UDP) services between endsystems and hosts.
Connections decision made here, we have **params** for specifying the **Quality of Service**. Requires the lower layers in order to operate, we work off this assumption that they do - we assume each host has one IP, etc. **Internet Protocols** are only a best effort delivery service.
Terminology: What we call data at each level of TCP/IP stack:
5) Application: Data **4)** Transport: TCP Segments/UDP Datagrams **3)** Network/Internet Packets **2)** Data Link: Frames **1)** Physical: Bits

3.1) Ports Connect apps together, and separate different apps' connections. The transport layer uses port nums to differentiate between different network communications. Each app has own p num. They're cross platform. **Port Usage:** 0 → 1023 (reserved), 1024 to 49151 (any user app can use/register), 49152 → 65535 (dynamic/ephemeral/private) and are used by clients temporarily.

3.2) Berkeley Socket Interface: 1) **fd=SOCKET(ip, port):** Create a new communication endpoint. 2) **BIND(sock_fd, sock_addr):** Attach a local address to the socket. 3) **LISTEN (sock_fd, n)** Announce ability to accept, n connections. 4. **ACCEPT (sock_fd, cli_sock)** Block until some remove client wants to establish a connection 5.**CONNECT(sock_fd, &servaddr)** Attempt to establish a connection. 6. **SEND** Send data over the connection. 7. **RECEIVE** Receive data over a connection. 8. **.CLOSE** Release the connection. Note server data exchange port = 20. In a connectionless scenario we don't need LISTEN, ACCEPT or CONNECT. For a client we make a socket, send (and Receive), and close. For a server, we make a socket, receive, send and close.

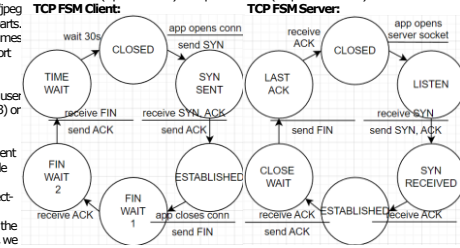
3.3) TCP: Connection-oriented. **Data is split into segments**. Reliable (integrity, usually maintain order), Not secure, Can offer stream connections (only accept segments in order, e.g received 4, waiting for 5, but received 6, ignore/drop 6 until 5 is received.),
Congestion Control, Handshake starts the connection. Full-Duplex.
To identify a socket connection: IP Address: Port Protocol
(e.g 61.195.17.146 : 80 TCP)
Segment: A wrapper for TCP data, transmitted within the Network Layer protocol (e.g IPv4 or IPv6)
Maximum Segment Size (MSS): The maximum amount of application data transmitted in a single segment (header size is not included). Usually related to the MTU of the connection to avoid network level fragmentation (splitting segments in the network layer into multiple packets).
Maximum Transmission Unit (MTU): The largest unit of data that can be transmitted through all links to the receiver without requiring it to be split. **Path MTU Discovery** determines the largest frame that can be sent on all links from the sender to receiver.

3.3.1) TCP Header: 20 Bytes, followed by a set of options (variable length). 1) 4 Bytes contains Source and destination ports (16 bit ids). 2) 4 Bytes has Sequence number, the 4 Bytes after has Acknowledgement Number (32 bits), used for reliable data transfer. 3) The next nibble has offset, then the next nibble is reserved, then the next byte has TCP flags. 4) The last 2 bytes of this group is Receive window (16 bits), the amount of data that can be sent before an acknowledgement is. 5) The next 2 bytes is the Checksum, the next 2 are "Urgent Pointer", and then 6) we have the variable length TCP **options**. Options negotiate protocol parameters such as Window scale/ MSS. Header length determines the size of the TCP header in 32 bit words. Several Header Fields: 16 bit checksum. The following are 1 bit: URG: Signals data as urgent, urgent data pointer field points to urgent data. ACK: Signals that the ack num is a valid ack. PSH: Asks the receiver to push data to the app immediately. RST: Resets connection / often due to errors. SYN: Synchronisation flag, used as part of the handshake. FIN: Signals connection to finish/shutdown.
Sequence Number: Bytes have sequence numbers (aka, what number byte they are). The ***sequence number of segment*** is the sequence number of the first byte it contains. We intialise TCP connec-tions with a random **Initial Sequence Number** to avoid receiving leftover segments in error.
Acknowledgement Number: the first sequence number **not yet seen by the receiver** (aka what we're next waiting for). Usually ack every other packet. TCP is full duplex, multiple streams/sequences can be received, and acknowledged at the same time.

3.3.2) The Three Way Handshake: 1) The client **sends a TCP Segment** with the **SYN** flag to true, **and sets an initial sequence number**. **2)** The server responds with another **SYN TCP** segment, which also has the **ACK flag set to true**, and the first unseen client SEQ#, as well as an initial sequence number for the server
3) Lastly, the client responds with **an ACK**, including the first unseen server SEQ#, and the client's new SEQ#.
Connection termination is similar, but uses FIN > SYN.

3.4) UDP: Connectionless transport layer protocol. **1)** Datagrams cannot be larger than 65,507 bytes (20B IP Header +8B UDP Header + 65, 507B = 65, 535B which is the maximum IP packet size).
Usually use smaller 500B to 1KB datagrams to increase the proportion of packets that are intact and less data is corrupted by an error.
2) App identif. provided for (de)multiplexing). **3)** CRC-type checksum. UDP is **simple** - no flow control, no error control, no retransmissions. UDP similar to IP, but adds 8B UDP header with **port** num. Since its connectionless, each packet contains address:port of recipient.
Reasons to use UDP > TCP: 1. Small Packet overhead
2. Finer App Level control over what data is sent and when
3. Connectionless; faster than TCP. 4. No connection state. Header **only contains** source/dest port, length, checksum **Useful in Client-Server interactions:** short msg, quick response, if fail, resend. Simple code, fewer messages. **DNS** uses this as mentioned. TCP is slower, but we can prebuffer data for reliable good performance. Livestream = UDP or buffer.

3.5) Finite State Machines: very useful formalism to specify and implement network protocol. States represent state of a protocol, transitions are characterised with an event(input we send) on top of an action(output from other) .



We can view TCP states using netstat –a tcpview on windows. Linux: htop, iptraf
TCP provides mechanisms to ensure reliability in data transfer. IP is best effort but we can still make it a reliable connection by going through TCP (unreliably send reliable data on an unreliable channel)

Bits may be flipped in transmission due to interference / imperfect physical hardware.
Error Detection: Receiver must be able to check if packet is corrupted.
Receiver Feedback Receiver must be able to tell sender the if packet was corrupted.

3.6) Ways of dealing with errors: Parity Bit: XOR of all bits.
1) **Stop and Wait with Error detection:** We can add error detection to the data transfer FSM. The protocol is synchronous, meaning for each segment the sender must receive back an acknowledgement before the next segment is sent. The issue is, ACKs and NACKs can also get corrupted. We could have no termination if we keep sending NACKs that get corrupted on a noisy channel. To fix, we can **assume a NACK and retransmit:** but leads to dupe packets. To deal with this we add a sequence number to each packet, so that receiver knows which packets are retransmissions. For stop and wait we only need 1 bit for the sequence number, 0 original, 1 retransmission.
2) **Use Sequence Numbers instead of NACKs:** If the packet is not acknowledged then an ACK is not sent. ACKs are sent for the last good packet, hence the receiver **can use a lack of ACKs** to determine that it must retransmit some data. In TCP the ACK number would be the next packet to be sent.
Possible ACK interactions and responses:
1) *Arrival of in-order segment with expected seq num.* **Delayed ACK:** Wait X ms for another in-order segment, if it doesn't arrive send ACK **2)** *Arrival of an inorder segment with expected seq num, and another waiting for ack* **Cumulative ACK:** Send ack for both. **3)** *Arrival of out of order segment with higher seq num (gap detected):* **Immediately send Duplicate ACK**. **4)** *Arrival of segment that fills a gap in received data*: **Immediately send ACK if segment starts at lower end of gap.** Other ways of sending data reliably over unreliable networks: timeouts, checksums.

3.7) Congestion Detection: If too many packets are sent to a router, the queue overflows, and some are dropped. The server assumes congestion if: **timeout /** multiple ACKs (we consider this a NACK). There are many Congestion Control algorithms (giving rise to different TCP version), with different characteristics:
1) **Tahoe**: Slow Start, AIMD, Fast Retransmit, 2) **Reno** (TCP version we study): Fast Recovery 3) **Vegas**: Congestion Avoidance (Detect congestion in advance: predicts packet loss using RTT, Larger RTT → more congestion), 4) **TCP Cubic**: Linux standard. To avoid advantaging smaller RTTs (with TCP Reno), grows window as func of time rather than RTT.
Congestion Window: (Number of bytes that can be sent before blocking to wait for acknowledgements. The sender and receiver can define the window size; the size used is the minimum of both. With a given RTT and window size W:
maximum rate output $\lambda \approx W / RT$, as we can't send more than this, our window limits us. We have a few ways of modulating our output rate:
1) Slow Start: 1) Set Initial Window size to MSS (max segment size)
2) For every ack, add size of data acked to our window size (roughly doubling it).
3) Continue this exponential increase until **ssthresh**.
4) We then use **Congestion Avoidance**.
2) Congestion Avoidance: 1) Increase window size linearly, (1 MSS per RTT).
For each ack, $W = W + MSS^2/W$. 2) When congestion is detected switch to diff strat.
3) Additive Incr/Multiplicative Dec: For every ack: $W = W + MSS^2/W$. For every packet loss / congestion event: $W = W / 2$.
Timeout: We need to detect packet loss (no ack sent back), using an interval T. T must be larger than RTT, but not too large otherwise we retransmit too slowly. **TCP handles** this by continually estimating RTT, and then setting T = SRTT + 4RTTVAR. SRTT and RTTVAR are determined via complex, black box (to us) means. **Fast Retransmission:** Three duplicate ACKs = NACK. Tradeoff between fast/premature transm.
4) Fast Recovery: Starting window size is W:
1) ***If timeout occurs*** 1.1) Set W = MSS 1.2) Run Slow Start until W = W / 2
1.3) Use **Congestion Avoidance**.
2) ***If NACK*** **(3 Dup) ACKs** 2.1) W = W / 2 2.2) Use **Congestion Avoidance**.
Fast as window size isn't reset back to MSS, so we ramp up the window size quicker.
1) Window Based Techniques: 1.1) Sliding Window – Go Back N: 1) Sender transmits multiple segments without waiting for ack, having up to W bytes of unack segments in its pipeline. 2) Sender's state is a queue of acks. 3) When we receive some acks, we can move the slide the window along. 4) If we had an issue (e.g. repeated acks, we slide the window back and resend).
1.2) Sliding Window – Selective Repeat: 1) Sender only re-transmits those segments it suspects were dropped or corrupted. 2) Sender keeps list/vector of acks. Receiver keeps L/V of ack segments. 3) When segments are received out of order, **they're held onto, and added into the data once the missing segments arrive**. 4) Sender **keeps a timer** for each segment it is waiting for acknowledgement of, **resending when the timer expires**. 5) Sender **slides the window when the lowest pending segment is acknowledged.**

3.8) Flow Control: avoid overflowing receiver. 1) Receiver sends **RecieverWindow** size along with acks – usually the size of buffer left to fill. 2) When a buffer is full and the receiver can take no more, it sends an acknowledgement with RecieverWindow set to 0. 3) Repeats a 1-byte ping to the sender to indicate that it is actively processing.

3.9) Wireless TCP: TCP assumes IP is working across wires.
1) Wired Networks: Packets lost → Congestion. Reduce packets sent, CA & recovery.
2) Wireless: Packets lost → Poor Channel Reliability. Resend Packets as much as possible.
These are conflicting requirements. We can fix by: **1) Splitting TCP Connections:** Use separate connections for wired/less so can distinguish between the two and thus use different algorithms for Congestion Avoidance. **2) Use Base Station:** Have the wired base station do some retransmissions without informing the wireless source. The base station tries to improve wireless IP reliability using TCP.

3.10 Network Usage
1) Utilisation Factor = Network Use / Maximum Theoretical Usage
If we have RTT, packet size L and transmission rate R, we can use the time on the connection used out of the possible time length:
2) $d_{trans} = L / R$ 3) Utilisation Factor = $d_{trans} / (RTT + d_{trans})$
4) Network Security: 1) Attackers: 1) **Hacker** (Grey, White, Black Hat)
2) **Phreaker**: Phone hackers – more common as phone networks digitise.
3) **Virii**: Computer Virus Creators. Virus Types: Ransomware, Trojan, Spyware.
4) **Anarchist/hacktivist**: Political hackers. Example: Anonymous (hacker group).
5) **Crackers:** Use tools made by others (virus, infiltration tools). Most frequent.
6) **DDoSers** 7) **Spammers/Botters**: Send unsolicited messages enmasse.
8) **Warez**: Info pirary, distributing software, images, videos illegally. Piratebay.
9) **Whistleblowers:** Former members of a group that leak information on malicious/illegal activities, even when prohibited from doing so by NDAs.
10) **Social Engineers: Manipulation**. Phishing (pretending to be a company in email), Vishing (via voice), Smishing (SMS), Catfishing (Social Media).

2) Black Hat Methods: 1) **Credential Reuse/Stuffing** (haveibeenpwned)
2) **Packet Sniffing** 3) **Code/SQL Injection** 4) **Wardriving**: Searching for and abusing unsecure WiFis. 5) **Dumpster Diving for Info/Disks**
6) **Clickjacking:** Using hidden html divs/popups to redirect clicks to malware.
7) **Bait & Switch**: Luring a user with a seemingly legit advert, redirect.
8) **Spoofing**: Falsify ID to receive another's packets: IP, MAC, DNS (cache poison)
9) **Session/Cookie Hijacking:** Stealing a session cookie to be authenticated as the genuine user. 10) **Rootkits:** Allow secret access to systems, part of virus
11) **Keyloggers:** Thwarted by password managers. 12) **Trojans**.
13) **Evil Twin:** Lure victims onto their network to steal info/send malware.

3) White Hat Tools: 1) Tails: Portable OS used from USBs, leaves no footprint. Doesn't store data. **2) Kali Linux:** OS uses for pentesting. Has Metasploit and Nmap (discover hosts and services on a network) – supported by ARM and WSL.
3) Metasploit: Scan for vulnerabilities based on large database of known vulnerabilities and exploits.

4) Cybercrime Laws: UK Laws listed below, but physical location of hosts determines what nation's laws are used (making US law very important too):
1964 Obscene Publications Act (Spam & Porn)
1978 Protection of Children Act (Online abuse & spam against children online)
1988 Copyright, Designs and Patents Act
1990 Computer Misuse Act
1999 Amendment to the Protection of Children Act (Still being changed)
2000 Freedom of Information Act
2000 Regulation of Investigatory Powers Act (Surveillance)
2002 e-Commerce Regulations Directive
2003 Criminal Justice Act
2005 Disability Discrimination Act (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)**

5) Standards Organizations:
IANA (Internet Assigned Numbers Authority, deal with DNS, IP addr),
IEFT (a collection of working groups (e.g routing, transport, security) concerned with developing the internet.). **W3C** – help build web tools.
Protection: Accounts, Proxies, Firewalls, Antivirus, Cryptography, Backups, Physical protection (locks, guards, alarms), People (DBS check, training, policy)

6) Past Attacks/Exploits: 1)Heartbleed: Bug in OpenSSL (software implementing TSL/SSL protocol for secure web access) allowing server memory to be leaked. **2)KRACK:** WPA2 had issues – android devices could be forced to use a zerobased key, making encryption useless. WPA3 also has issues.
3)WEP: Security algorithm for wireless network. Has had many vulnerabilities

4.1) Network Security Issues. 1) Access Control
The principal (user) requests access to a resource – which is protected by a **Guard**. They do this over a secure channel. The Guard must determine:
1) Which principals can access the resource. **2)** Where principals can be locat-ed. **3)** What requests principals can make for this resource (access rights). **Security** is hard as organizations use diff systems, users can be careless (reuse PWs)

1.1) Firewalls: Barrier between internal and external networks. 1) **Applic-ation Level Gateway:** Application that runs, checks requests in the application layer, can also be a proxy. **SOCKS,** netfilter iptables.
2) Proxy Server. 3) Circuit Level Gateway Circuit of Proxies (TOR)
4) Packet Filtering: Filtering with rules on contents/source/destination IP address/port. Can also be stateful, considering not just a single packet traffic to a host over some time period. 5) **Hybrid** Use a combination of all the above. Software or hardware. Hardware firewalls are faster, but difficult to change.

1.2) Proxy: Requests/responses on behalf of client. Caching, Filtering.
1) Normal: Client is aware of proxy, connects to use it
2) Transparent: Client is unaware. Requires no intervention from client.
3) Reverse: Runs on receiving side, impersonates/insulates server (Like CDN).

1.3) Bastion Host: Server expecting attacks. Runs minimal secure OS, with only essential apps, with strictest restrictions (read only, no mount, file perms), managed over a dedicated terminal. It passes requests on from the external network, and acts as a proxy firewall. It drops any connections it determines suspicious, using packet filtering (usually stateful) and other techniques.
iptables: Linux Root tool for Packet Filtering. Tables with chains of filter rules.
tcpd: Daemon controlling access to unix services. Consults ../hosts.allow, ../hosts.deny

1.4) Access Control Lists: Has list of rules, specifying what Inside/Outside IPs/Ports are allowed, and a description. * = all. We should add an end rule blocking anything else

1.5) Firewall Avoidance: 1) **Use SSH/VPN:** Send requests using SSH (or any allowed protocol) to internal network and use services the firewall normally blocks.
2) **Spoof MAC Address** 3) **Spoof IP Address** (detected by stateful firewalls).

4.2) Further Security Measures: 1) **IDS (Intrusion Detection System):** Detect intrusions and informs the system. **2) IPS (prevention)**: Prevents intrusions (block SYN floodders for example – prevent DDOS). Can work with IDS.
3) Next Gen Firewall (NGFW): Stateful firewall that comes with IDS and IPS.
4) United Threat Management: Like NGFW, but has spam filter, antivirus etc.
5) DMZ: Between our private internal network, and outside networks (internet), we have a **DMZ**. External hosts can only speak to internal hosts in the DMZ, everything else is hidden and protected by a gateway, router or firewall. We use **NAT** to get external messages to the correct internal host. If we want to expose an internal host without it being in the DMZ, we use **Port Forwarding.**
NAT: Instead of exposing the LAN IP address of an internal host, when sending routers map LAN IPs -> own public IP, and when receiving public IP -> LAN IPs.
Port Forwarding: To expose an internal host to the external network without placing it in the DMZ, we can set the router to forward all packets arriving at a given port straight to the internal host. Any packet received on the router's IP at port 3472 could be immediately forwarded to the NAT based LAN IP of "host A" on port 80. Useful for hosting servers, even for games (e.g minecraft servers

4.3) Logging and Auditing: most systems keep logs. Useful to **check for breaches, forensics, determining how systems were exploited in attacks, ensuring good practice**, detecting other network issues **(congestion?)**. Logs can be found on linux at /var/log, and the event viewer in Windows.

4.4) Cryptography: K/K¹ = En/Decryption Key, M = plaintext, M_c = ciphertext Good Encryption Algorithms ensure: 1) Given M_c it's only possible to get M by going through all possible K¹. Given M and M_c we can't easily get K and K¹.

1) Symmetric and Secret Key Encryption Algorithm: The same key is used for encryption and for decryption. Key must be secretly shared bw sender and receiver on a private channel. Faster than Asymmetric. Example: Data Encryp Std.

2) Asymmetric Public Key Encryption: Each user has a public and private key. **For confidentiality**: Sender encrypts with receiver's public, receiver decrypts with their private. **For signing:** Sender encrypts with their private key, and receiver decrypts with their sender's public key – if successfully decrypted then we have verified the message was from the sender with the public key we used.
We combine this to encrypt a message, but also include a signed segment to verify the sender. We can also combine with **symmetric encryption** to sign symmetrically encrypted files (e.g check if a password protected file is from the correct sender/is not tampered with). For example Gnu Privacy Guard.
No need to disdose private info, so we can communicate on unsecure channels. Slower to encrypt and decrypt. RSA uses this (prime factor decomp too), as its **very hard** to use the public key to get the private key as a result.

3) Diffie-Helman Key Exchange: When beginning comm. we publically agree on a number **g**, and a large prime **p**. User a owns a secret key a and user b owns a secret key b. UserA sends x = g^amod p to UserB, and B sends y = g^bmod p to A. From this, A can do modular arithmetic to get g^{ab}mod p, as can b. This gives us the **private key g^{ab}mod p,** which only a and b know, as only they have a and b.

4) Kerberos: Key distribution system using a trusted server.
1) Kerberos authenticates you with a password, and the user you intend to communicate with. **2) Generates** a **ticket** for communication
3) Ticket can be used for some time, until a new one is needed.
Used to be vulnerable to Man in the Middle attacks. Has been addressed.
5) **Hashing:** I already know these. Many hash funcs are obsoleted though now, either algorithm issues or due to rainbow tables.

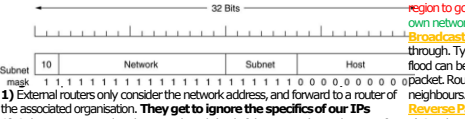
5) The Internet/Network Layer: Responsible for routing packets though the internet, and across networks with differing hardware, protocol stacks. We resolve the differences with gateways: interconnect networks by knowing many protocols.

Terminology:
1) **PAN: Personal** Your phone connected to your PC via Bluetooth (only you)
2) **LAN: Local** People in the vicinity could use the network
3) **MAN: Metropolitan –** a large network that doesn't use the internet
4) WAN: Goes over the internet.

5.1) Devices, Layer, Purpose: 1) **Repeaters: Physical Layer** – Boost Signals.
2) Switches & Bridges: Data Link: make interconnections based on MAC Addr
3) Routers (Gateways): Network/T/A: Forward & Split packets based on IP. To connect IP Based Networks we need a Router acting as Gateway bw them.
Internet Protocol: Main Protocol. Defines datagram format, how we do fragmentation, IP addressing and Packet Handling. The type of service our packet does is called **DiffServ**, we don't use the IP options usually due to security issues.
DHCP, **ICMP** (Internet Control Message) for error reporting/handling.

5.2) Fragmentation:
When our data sent (via IPv4) is larger than the **MTU** (found by Path MTU Discovery) of the path it's crossing, the IP datagram needs to be **Fragmented**.
1) Fragmentation occurs at the start, or any intermediate routers.
2) Fragments only reassembled at the destination.
3) Each fragment is identified by its 16-bit fragment identifier. Fragment Offset are the offset in units of 8-bytes (fragments are multiples of 8 bytes, plus a last byte).
4) The more fragments bit (M) is used for the receiver to ensure it waits for them all, and is set when an intermediate router fragments a packet.
Our fragmentation scheme must let any intermediate router fragment, and recognise fragments of the same datagram.

5.3) IP Addressing: IPv4: 32 bit addresses. XXX.XXX.XXX.XXX, XXX ∈ [0, 255] Each IP address is associated with an interface, so hosts may have more than 1.
1) Classful Addressing: IP addresses split into different classes:
A [0, 7 bit NA (2^7-2 networks, subtract 2 always), 24 bit HA]1.0.0.0-127.255.255.255
B [1 0, 14 bit Network Address, 16 bit Host Address] 128.0.0.0–191.255.255.255
C [1 1 0, 21 bit Network Address, 8 bit Host] 192.0.0.0 – 223.255.255.255
D [1 1 1 0, 28 bit Multicast Address] 224.0.0.0 – 239.255.255.255
E [1 1 1 1, 28 bit Reserved for Future Use] 224.0.0.0 – 239.255.255.255
Calculate start/end address by just computing the binary num. **Issue:** All hosts on network share the network address. If an organization has hosts with many different IPs it must announce and publicly daim many net IDs. No longer used. ICANN manages Net IPs to avoid conflicts.
Subnet Masking: Allows us to discover which parts of an IP are used for the Subnetwork, and which parts are used for the hosts of the Subnetwork.
2) Classless Addressing: Use a single network address for the entire organisation, and divide internally into subnet addresses and host ids



1) External routers only consider the network address, and forward to a router of the associated organisation. **They get to ignore the specifics of our IPs**
2) Subnet routers apply subnet mask and check if the IP is in their subnet, or if they need to forward to another subnet in the organisa-tion. **3)** Once a host is found, routers know which interface to forward packets to.

M1) Subnet Addressing: To find range of addresses of 128.138.207.160/27: 27 indicates Host = 5 bits, Network, Subnet = 27 Bits. We take the base address 128.138.207.160 and since Host is 5 bits, 2^5 = 32, so range is 128.138.207.160 to 128.138.207.191. **The subnet mask is** M = 11...1 **N** times 00..0 **32-N** times
1) Whenever we try to request something to a specific IP, then the router finds and forwards to the **Network Address** by **ANDing** the IP with the **Subnet mask**.
2) We find the **First Host Address** by incrementing the network address by one, as the first address is not usable.
3) **The Broadcast Address** is gotten by setting final byte of the network address 255.
4) **The Last Host Address** is gotten by making the final byte of the network address 254, as the 255th one is not usable.
While we're doing this the Routers try to match on the longest possible prefix.

5.4) Dynamic Host Control Protocol - Allows hosts' interfaces to safely be assigned an **IP Address**. When booting a host broadcasts a DHCP Discover packet, and a **DHCP server will respond with an assigned IP**. DHCP servers maintain mappings of host to IPs, and assign different IPs each time a host connects. **Hosts lease an IP**, and refresh periodically (to prevent hosts hogging an IP). My Router would request an IP from my ISP's DHCP servers periodically.

5.5) Network Address Translation: An attempt to solve IPv4 shortage. **NAT:** Instead of exposing the LAN IP address of an internal host, when sending routers map private IPs -> own public IP, and when receiving public IP -> private IPs.
table of mappings. Following address ranges are **private**, can only be used locally.

10.0.0.0 → 10.255.255.255/8	16,777,216 addresses
172.16.0.0 → 172.31.255.255/12	1,048,576 addresses
192.168.0.0 → 192.168.255.255/16	65,536 addresses

NAT violates IP Model (an IP doesn't uniquely identify a host). Changes internet from connectionless to connection oriented (routers keep track of connections). Violates Protocol Stack rule: relies on TL layer info, so must use specific TL protocol (can't easily support new ones). Many P2P protocols require full connectivity between hosts which NAT cannot provide. Hence prot forwarding, TURN relays, NAT punching holes, 3rd party servers and other solutions are required.
Special IP Address: 0.0.0.0/0 Default route, used when no other IP matches. **0.0.0.0/8** This host on this interface. Must not be sent, only used to acquire an IP. **127.0.0.0/8** "Loopback"(reference to host), can be sent (127.0.0.1=localhost). **169.254.0.0/16** "Link Local" (Error with acquiring an IP Address).

5.6) IPv6: Intended to fix IPv4 address shortage. 1) **Has 128 bit addrs** (8 4 digit hex nums). 2) Reduce routing table size, simplifies protocols for higher performance. Improves security. Support scopes with multicasting (sending a packet to many hosts in a certain scope, e.g network). Better support coexistence of old and new protocols, while making it easier to develop new ones.
2) Key functional changes with IPv4: Fragmentation is done by end systems, No header checksum as Transport Layer and Data Link have error recovery, Fixed Header length (IPv4's options scarcely used), better modularity for extensions.
3) Extensions are done by adding an extending header after IPv6 header: e.g: Routing Path, Fragmentation Info, Authentication Info, Destination Options.

5.7) Routing: Providing facility for moving data from source to destination.
Goals: **1) Allow multiple hops on network nodes 2) Consider topology of network for routing 3) Load Balance 4) Deal w network heterogeneity**
Remember, **The Internet** is packet switched, connectionless, best effort.
Forwarding Tables determine which router to send packets to, based on final dest.
Sink Tree: A tree from a source node to every destination node, with no cycles, for which each path to a node is optimal. Produced by **Dijkstra's Algorithm**.
DA: From start node (has 0 cost), compute the cost of every fringe node. Select lowest cost fringe node, compute new costs to each fringe node (by doing cost to this node + cost of going
to that node). and update if any new costs are smaller. Continue for every node.
Shortest Path Routing uses Dijkstra's Algorithm to build a Sink Tree. Packets are forwarded using this accordingly.
Flood Routing (FR) Forward incoming packets to **every outgoing link,** except the one that sent us the packet. To avoid drowning the network we can use:
1) Hop Counting: Disregard packet after it reaches a chosen number of max hops.
2) Forward Once: If receiving same packet, don't forward again. Ensures they don't get sent in cycles. Must store. We decide how long to store sequence nums.
3) Selective Flooding: Flood only in specific directions, based on a heuristic.
We use FR when the packet must be sent successfully, but the route is unknown.
Distance Vector Routing (DVR) Dynamic algorithm – takes into account changing network conditions. Each router advertises its cost to each destination, Router's use their cost to neighbours, and their neighbour's costs to determine how to route packets for minimum cost. Uses Bellman-Ford Equation:
$D^x[v] = \min_{\text{x's neighbours}\{u\}}(\text{cost}(u, x) + D^x[v])$
(Cost from u to v = min(cost from u to a neighbour + cost of that neighbour to v)
Problem: When a node goes down, routers continually update their costs based off others, resulting in the cost incrementing constantly. Resolved by defining ∞ cost as: cost ∞ = longest acceptable path + 1. RIP algorithm uses this.
Link State Routing: Replacement for DVR. **1)** Broadcasts all info on network topology to all routers, which each produce a sink tree. **2**) Identifies neighbours using special "hello" packet, neighbours respond with their network address. Link costs is determined using special "echo" packet, and measuring the **RT** delay. Algorithm: **1)** Get direct neighbours & their network addresses ("hello").
2) Calculate the cost of sending packet to each neighbour ("echo").
3) Build a **Link State Advertisement/LSA** describing the router & its connections to neighbours. **4)**Send LSA packet to all routers on the network (**flooding**)
5) Receive LSA packets from every other router on the network.
6) Routers have status of links between all routers, use DA to make **sink trees**.
This algorithm allows better routes to be chosen using current network conditions. However routers may redirect traffic towards the best routes so much, that these routes become overloaded, and are no longer the best routes. An example of Link State Routing is OSPF. **Comparison of DVR and Link State Routing:**
1) Network Info: DVR – Local, LSR – Global.
2) Computation: DVR – Global, LSR – Local.
3) Synchronisation: DVR – Gradual (routers update & advertise), LSR – Instant.
Hierarchical Routing All prev routing techniques can't be scaled – each router has to know about all others. Too much memory to be practical on the Internet.
To solve this the network is split into regions. Different algorithms used for intra-region and inter-region routing. Can scale the network massively. Suboptimal routes are chosen between node in different regions, (we only know which region to go to). Can use separate algos bw regions, we consider each region its own network with unique design and structure. 2/3 levels of regions are enough.
Broadcast Routing To solve scaling, we send to every host on a network even through. Types of routing we could use: **1) Flood Routing** - acceptable if the flood can be limited, **2) Multi Destination** - A list of destinations is sent with the packet. Routers check this list, splitting the list and forwarding the packet to its neighbours. The packet must contain all destinations. **3) Multicast Routing**
Reverse Path Forwarding Constructs spanning trees from a router, at low cost.
1) Send packets to all neighbours (like flooding). 2) Routers only accept packets if they're on a direct path from the source. Hence packets travel a MST from source router. Also can be used to detect and prevent IP spoofing (packet will come from an odd path, given spoofed IP)
Multicast Routing We want to send a message to a subset of the nodes (with group id). We need to know when a host leaves or enters a multicast group.
1) We could create a spanning tree at each router, and send only to paths that have members of the group we want to send to, using the group id to check.
Scales Poorly, need a spanning tree per source.
2) Else we can use **Core based trees.** We have a single spanning tree per group. We select a node near the middle as our core (cheaper to then broadcast from it). Now, to send a multicast message to the group, just send it to the core and retransmits to all nodes in the group. Not always optimal, but scalable and has lower overhead. Internet uses this (multicast IP/Broadcast Address is the core).
Thus far we've been looking at routing in a flat network model – small in scale, without having different administrative groups that want to run their own choice of routing protocols. The Internet is organized as **Autonomous Systems** – with independent administrative domains connected by **Gateways**.
Intra AS Routing determines routing **within** autonomous systems, attempting to provide **optimal routes on small networks**, whereas **Inter AS** is between them, attempting to provide as **good routes as practically possible**.

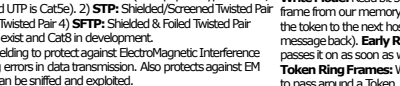
Open Shortest Path First (OSPF)
LSR algo to replace RIP (DVR).
1) Algo publically available for anyone to implement.
2) Supports different distance metrics (hops, delays, etc).
3) Adapts dynamically to changing network topology (nodes add /rem)
4) Supports routing based on ToS (Type of service).
5) Supports load balancing (not overwhelming routers e.g by flooding
6) Offers some security features (though some have been compromised) Supports hierarchical routing. It is possible to split an AS into several areas", and we route traffic between "areas" using "border routers".
Border Gateway Protocol Inter-AS protocol used by the Internet.
1) Neighbouring routers **maintain connections for reliability**. Provides reachability info from neighbour ASs and transmits reachability info to all internal routes in AS.
2) **Determines good routes** by reachability info and routing policies.
3) **Routers only check for & discover new paths if allowed.** 4) Path vector protocol, **paths are advertised.** No infinity problem. BGP **advertises** paths to networks: 1) Destinations are denoted using the address prefixes (subnetting). 2) ASes don't propogate advertisements by gateways.3) **Routers can aggregate prefixes**: We can merge several ips (with prefixes) into one, with a shorter subnet mask: 127.134.126.0/24 & → 127.134.126.0/23
In BGP each AS has a unique ASN. Route adverts have attributes:
1)**AS-PATH** Seq of ASNs identifiers through which the advert was sent
2)**NEXT-HOP**: specifies interface (IP address) for forwarding packets towards advertised destination, resolving the ambiguous case where AS reachable through multiple interfaces
The BGP import policy determines to accept or reject route advertisements. **Router preference is ranked according to**: 1. Policy used. 2. Shortest AS-PATH 3. closest NEXT-HOP router.
The count-to-infinity problem is solved by **path exploration / hunting** (actively seeks paths), furthermore routers can send **withdrawal messages** (e.g before taking a node down, can tell others to remove the path). **This allows routers to identify invalid paths**, at the expense of some delays.

5.8) Wireshark – I already know about Credential Reuse/Packet Sniffing/Code Injection/Session Hijacking/Wardriving (in security)
Wireshark is a network protocol analyser. Allows users to capture, analyse, deconstruct network traffic. Wireshark has 2 listening modes:
1) Promiscuous Mode: Works for Wired and wireless. NIC does not drop ANY packets. When wireless, only listens on the connected network. Some NICs ignore this (considered impolite and easily abused).
2) Monitor Mode: Only works on **wireless networks**. **NIC listens on all networks in range/that it can receive from**. Wifi networks secured with authentication (e.g password) *will appear scrambled* (encryption). Most NICs do not support this, may require new drivers or a special NIC. WinPcap (windows) does not support though AirPcap and Npcap on linux do.
To avoid capturing other people's packets, turn off Promiscuous Mode. If the wifi net has a PW, you need to tell Wireshark about it.
Wireshark Packet Capture:
1) **Hub:** L1, Broadcast/Multicast,(Promiscuous)Entire Network.
2) **Switch:** Local Traffic, Broadcast/Multicast, (Promiscuous Mode) Network connected to the same switch port. 3) **WLAN:** Local Traffic, Broadcast/Multicast, (Promiscuous Mode) Entire WLAN, (Monitor Mode) All wireless packets physically receivable/in range.
Example Wireshark Filter

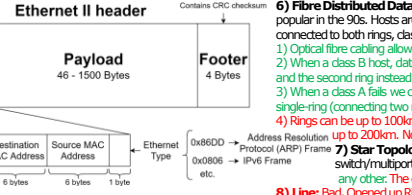
```
http.request.method == GET &&  http contains "password"
&& (ip.src != 10.43.54.65 || ip.dst != 10.43.54.65)
```

(show us all packets except those to src IP address 10.43.54.65 and destination 10.43.54.65. We can also use Wireshark for MAC Addresses.)
Ports of Protocols
UDP broadcast src: 0.0.0.0, port 68, dst:255.255.255.255, port 67)
DNS goes over UDP Port 53.
FTP (TCP ports 20 (data), 21(control)); **SSH** (secure shell, TCP port 22), **DHCP** (UDP ports 67/68) **IRC** (TCP ports 194/6667), **TOR** (TCP ports 9001, 9030, 9040, 9050, 9051, and 9150.
SMTP (TCP port 25). **SMTPS**(TCP 25/465 or 587).
POP3 (TCP port 110, 995 for encrypted POP3S)
TCP port 143 for unencrypted, 993 for encrypted

6) Data Link Layer – transmission of data (frames), multiplexing.
9.1) Ethernet: Data-Link Layer protocol used for communications. Made in 1980, IEEE Standard 802.3 in 1983. Started with Coaxial ≈ 2.94 Mbps. Fibre optic, twinaxial (two coaxial) cable ≈ 100Gbps.
Cables: 1) **UTP:** Unshielded Twisted Pair (Most popular type of cable, and most used UTP is Cat5e). 2) **STP:** Shielded/Screened Twisted Pair 3) **FTP** Foiled Twisted Pair 4) **SFTP:** Shielded & Foiled Twisted Pair
Ca76a, Cat7a exist and Cat8 in development.
Cables use shielding to protect against ElectroMagnetic Interference (EMI) causing errors in data transmission. Also protects against EM leakage that can be sniffed and exploited.
9.1) Ethernet Pinouts:
1) **Straight Through:** Communicate bw diff Layers. Switch-Router.



(troubleshooting a router). Wires go directly opposite.
568-B is quite common (the above), a **568-A** is also widely used, in which the White-Orange/Orange cables have been swapped with the White-Green/Green ones. We **conform** to same order on both ends.
Example: To connect a computer to a router, we will be connecting to a switch **inside the router**. The computer is on the Network Layer as that's the highest we can directly connect, and the switch is on the **DLL**. Hence we use Straight-Through.
Frame: the unit of data we use. CRC checksum. Interface to NL for sending and receiving packets – we take our data, wrap it in a frame, and the other side opens it up to get the data.
Modem devices know how to "fake-swap" bw straight and crossover using internal, software-based, remapping (Auto-MDI/MDIX).



6.3) Ethernet Cables

Cable Code	Name	Cable	Max Length	Top
10Base5	Thick Ethernet	1/2 inch coaxial cable	500	Bus
10Base2	Thin Ethernet	75-Ohm coaxial cable	180	Bus
10BaseT	Twisted Pair Ethernet	Category 3 UTP	100	Star
100Base-TX	Fast Ethernet	Category 5 UTP	100	Star
100Base-FX	Fast Ethernet	Fiber Optic	185	Star
1000Base-T	Gigabit Ethernet	Category 6 UTP (4 pairs)	100	Star
10GBase-T	10 Gigabit Ethernet	Category 6a UTP (4 pairs)	100	Star

6.4) IEEE MAC Addressing
MAC Address: 48 bit addr. Each NIC has a unique MAC addr. Written as <byte>:<byte>:...:<byte> 6 times in all. The first 3 octets (bytes) are Manufacturer specific OUIDs, and the last 3 are NIC specific.
In the first byte, the 2nd to last bit refers to whether we are **Universal**(Globally Unique)/**Local**(Locally Unique) and the last refers to **Individual** (Unicast, data sent is intended for one NIC)/**Group** (Multicast, data can be sent to many). **Broadcast Address:** FF:FF:FF:FF:FF:FF. It's not too rare we'll find two MAC addresses that are the same on the internet – but they're only relevant in LANs. If we run out, we just start over.
6.5) Switch Forwards to correct port using MAC, or floods if it can't find; uses a MAC table to remember addresses associated with ports. Difficult to network-sniff as packets are only directed to intended recipients. Can connect them to other switches or hubs, allowing networks to connect together. Replaced Bridges
Forwarding methods below:
1) **Store and Forward Switching:** Once a whole frame is received, integrity check w checksum. If correct, forward to port using MAC. Slower, must wait for entire frame. Error checks. Supported by bridges & switches.
2) **Cut Through Switching:** As soon as we have enough info (dest), begin forwarding. Faster, but doesn't error check (so the final receiver must). Supported by switches only.
6.6) Wireless: Standardised for wireless communication. Uses 2.4Ghz or 5Ghz radio (open for unlicensed use). Hub: Repeats all received). Can connect together to extend range. Can also act as a bridge to connect to a wired network. Easy to network-sniff, as all devices within range of the WPA can receive frames.
6.7) Topologies
1) Switched Ethernet – Many switches, each connecting hosts. Each switch port connected to ONE host, or machine. No collisions as a result, but expensive.
2) Internetworking Ethernet: A combination of networks sharing a cable. We might have a switched ethernet component, a bus component etc. **Repeaters** boost signal to extend the range of the network. **Hubs** are used (forward recieved frames out of every port)
3) Bus – End systems wired to one central link, go up/down. Data travelled up and down link. Terminators at end of cable absorb signals. To add new hosts, a BNC coaxial T conn-ector was required, cable cut and readded with new host. Popular early on
4) Ring – All hosts connected in physical ring. Each host needs two NICs, data flows around ring. If a link cuts, network dies. Dual Ring attempted – 4 NICs, data flows both ways. Expensive, doesn't scale. Dual Ring allows for one of the rings to be cut.
5) Token Ring – Uses an **MSAU** to connect physical devices as a **logical ring**. One host has the token at a time, and thus it is in **transmit mode** and can write to the network. No collisions as only one host can have the token at a time. All hosts can listen to the network (listen mode). Frames always fit inside the ring, as the host holding the token sends as a stream. If a Host dies it's no problem.
Listen Mode: If we receive a message for us, we keep a copy. Else we pass it on. We do this by passing our input stream to output through a buffer, which we can read from (if for us).
Write Mode: Read bit stream from the previous host, and transmit a frame from our memory to the next host. When we're done we pass the token to the next host (we know we're done when we receive our message back). **Early Release Mode** doesn't wait for this – it just passes it on as soon as we've finished writing.
Token Ring Frames: When there's no frame to send we only need to pass around a Token. Tokens are a subset of a normal frame, containing Start Delimiter, Access Control and End Delimiter headers. When someone picks it up to use it we embed the rest of the data to produce a **complete frame**. Frames have **Frame Check Sequence** – checksum for integrity, **Frame Status** (Set by destination host; A = 1 Destination host is working. C = 1 Destination host has read it.)
Interframe Gap (if we send many frames we can have a gap to separate them).
Supports Priority: Only claim token if the priority of data we want is at least as high as token priority
Supports Reservation: Host in listen mode might **have high priority data to send**, they can increase reservation priority. When a token is created, it has the priority of the reservation bits in the frame.
1) Low priority data may be delayed indefinitely.
2) High priority data will be sent quickly (has real-time applications).
3) LANs use it so we can trust hosts to not abuse priority.
Complexities of Token Ring Maintenance:
1) The **Frame Control** field is used to create control frames.
2) Frames may become orphaned (never received, loop around).
3) One host is the **Active Monitor** and is responsible for generating tokens and removing orphaned frames.
4) But Active Monitor may fail, so any host must be able to become the Active Monitor.
5) We need rules to determine which host becomes Active Monitor. This together adds complexity to token passing and reduces reliability In order to improve reliability, when a host fails it can be bypassed.
6) Fibre Distributed Data Interface: Token ring based topology popular in the 90s. Hosts are divided into two classes, class A connected to both rings, class B in an inner ring.
1) Optical fibre cabling allows for networks to be geographically large.
2) When a class B host, data can be re-routed through class A hosts and the second ring instead.
3) When a class A fails we can short-circuit two class As to create a single-ring (connecting two rings at disconnected ends).
4) Rings can be up to 100km long, so FDDI must work with a length up to 200km. No longer popular
7) Star Topology: All hosts connected directly to a switch/multiport-bridge. Any host can communicate with any other. The central switch is a single point of failure.
8) Line: Bad. Opened up Ring. **9) Star-Bus/Tree**: A Hybrid **10) Mesh**: Useful, especially in Wifi Mesh impractical.

6.8) MAC The data link layer is actually split into the higher LLC, and lower MAC (media access control) layers. **MAC** controls channel access, as a broadcast channel can have multiple receiving hosts and transmissions at the same time (frame collisions).
Station – a transmitting on a shared medium.
MAC Strategies:
1) **No Control**: When a frame isn't received, station retransmits as it pleases. Fine if low channel utilisation. Inefficient when contention is high (many transmitting stations means constant collisions & attempted re-transmissions). 2) **Round Robin:** Stations take turns to transmit. Used in token-based MAC systems.
3) Reservations: Stations obtain channel reservations prior to transmitting. Stations can only transmit for the time interval they have reserved. Requires a system to manage reservations. Used in slotted systems.
Static Channel Allocations - each station is allocated a fixed schedule of times it is allowed to transmit. For a shared channel of n stations:
1) Time Division Multiplexing (TDM): Stations wait for their time slot to transmit. Station's transmission rate limited to R/n; R = max channel rate.
2) Frequency Division Multiplexing: Stations given a limited frequency band. Each station can use B/n where B = total channel bandwidth. Bad for large n or traffic that is in bursts.
3) Dynamic Channel Allocation – ALOHA Protocol:
Stations transmit whenever they want to. If a collision occurs, stations wait a **random period of time** before attempting to re-transmit.
1)Low channel efficiency as there's a large **vulnerable period**.
2) If a frame transmission is interrupted by another at any point, the both frames must be re-transmitted (new frames can destroy old frames).
3) Maximum efficiency of 18% at 50% load.
Slotted ALOHA Protocol: Like ALOHA but can only transmit on specific discrete time intervals (slots) (managed by a synchronous global clock).W Reduces opportunities for a new frame to collide with an old one.
Can only collide with exact overlap (contention for a slot)
Maximum efficiency of 36% at 100% load.
For example, if frames are 20s long, **then our discrete time slots are 20, 40, 60, etc.** So we can only start transmitting at these time (round near).
Carrier Sense Multiple Access (CSMA)
Listen before transmitting, transmission only occurs when the channel is idle/free. 1) Reduces collisions over ALOHA as new frames are not sent during another's transmission 2) **Collisions can still occur** due to transmission delay (e.g two stations see idle channel, start transmitting, or one starts transmitting after another, but signal has yet to reach it).
CSMA/CD Algorithm We can combine CSMA with **Collision Detection; Used for** ethernet where all collisions result in frames being destroyed.
1) Station listens to channel during transmission to check for collisions.
2) Transmission stop when collision detected, then sends a **jamming signal** (so other transmitter knows).
3) Host must transmit long enough to be able to tell the frame has not been collided. Hence minimum frame length is 2η where η = end-to-end transmission delay.
Collisions are **inevitable** as there is no central authority controlling transmission. Best effort, frames might be infinitely delayed. **Suitable** for most LANs. Unacceptable for real-time systems (these require maximum wait time and minimum bandwidth assurances)
Channel Back-off (we stay back when the channel is busy)
1) 1-persistent (Aggressive algorithm) Continually check channel. Transmit as soon as the channel is free. Used by Ethernet.
2) **Non-persistent** (Non-Aggressive) Check channel, if idle translate immediately, else wait a random period of time before checking again.
3) **P-persistent** Keep checking channel, if free transmit with probability p
4) **Binary Exponential Back Off:** We use this when network load is high (contention for channels). Slot length is the minimum frame length. If a collision occurs in transmission, wait 0 or 1 slots before attempting again. After c collisions, wait 0 to 2^{c-1} slots (up to limit of 1023 slots / 10 collisions). High contention → lots of collisions → **binary exponential back-off** → reduced transmission attempts spread out → fewer collisions.
Carrier Extension: There is a minimum frame size requirement (to hold channel until bits reach destination), so for some frames an padding is required. This is inefficient as we waste time transmitting the extension.
Frame Bursting: Rather than padding with an extension, multiple frames are buffered, and packeted together and sent at once. **Superior.**
The following methods actually ensure stations get a turn to transmit:
Medium Access through Token Passing (The Token Ring Idea!!)
There is a single token, stations can only transmit when they have the token.
1) Token transferred with special token frame.
2) If a station has the token w/o a frame to send, pass token on immediately.
3) If a station has the token and a frame to send, it sets timer and transmits until the timer expires or there is no more data to send. Then passes token.
Ethernet became much more popular, and hence this is standard.
Avoiding Wired Collisions using a Fully Switched Topology
A switch can remove the possibility of collisions by buffering frames and retransmitting when a channel becomes available.
1) Each channel (ethernet cable) has only two stations (host and switch)
2) Hosts can transmit simultaneously, switch receives and forwards frames.
3) Maximum cable length determined by signal strength.
4) **Switches** act as **repeaters**, refreshing the signal to pass it further.
Address Resolution Protocol (ARP) We need IPs to communicate **outside** of our network. Translate IPs → MAC Addrs and vice-versa, we use ARP. In summary it is used to **discover Network Addresses:**
1. **Router:** ARP lets them ask all hosts if they have a **given IP Address** Places ARP Message query in a Data-link frame and broadcasts.
2 **Host:** Checks if it has the requested address, if so sends a reply with its MAC Address
3. **Router:** Receives ARP Message with MAC Address and uses it.
Will forward IP Datagrams (encapsulated in a Data-Link Frame).
Usually also caches the IP → MAC translation
Optimisations: Caching recent ARP Message replies, having hosts broadcast their IP and MAC address on boot (as a network policy).
As always, Caches are liable to **Poisoning**. Malicious users can send spoof ARP Messages to attempt to associate their MAC Address with a victim's IP Address and receive their IP Datagrams.

7) Physical Layer: Network Architects design networks, Engineers set them up. **Patch Panel:** Cables from the Computer Lab arrive into there. These link into a Network Switch, or Private Branch Exchange (used for phones unless its IP based, in which case we don't need PBX)

7.1) Wire Transmission:
UTP: Two wires twisted together. Cheap, twisted pair reduces interference. Used by telephone. CAT:1Mbps, CAT5:100Mbps (10 Base T), CAT6:1000Mbps (100 Base T). **Freq Range**: 0-1MHz, **Attenuation**: 0.7dB/km at 1KHz, **Delay**: 5μs/km, **Rep Spac**: 2km
Coaxial Cable: Conductors placed concentrically, surrounded by an insulator. (inner conduct circle, outer insulator, outer outer conduct). Good shielding – EM field is between inner conducter and outer outer. High bandwidth from range of freqs. But more expensive than UTP.
Freq Range: 0-500MHz, **Attenuation**: 7dB/km at 10MHz, **Delay**: 4μs/km, **Repeater Spacing**: 1-9km
Fibre Optic: Uses light and reflection. An optical fibre is 2-125 μm in diameter. **Attenuation** (signal loss) very low. Very high bandwidth. Expensive. In 2021 Japanese Researchers reached 1000 Tbps.
Freq Range: 186-370THz, **Attenuation**: 0.2-0.5dB/km,
Delay: 5μs/km, **Repeater Spacing**: 30km
7.2) Wireless Transmission: Done using EM Waves (Radio usually).
No wires (expensive, takes time), Bidirectional Comm., Broadcast (so all receivers see transmission), Inverse² law for signal strength, Environment affects signal. **Ways of Representing Information:**
1) Digital: Discrete info, represented by finite states (e.g. 0, 1 Binary).
2) Analogue: Continuous info – represented by physical changes in state. Voltage and Light Intensity.
Baud Rate: Symbol (not always 1 bit) rate/second for digital channels
Modem: Implements a digital channel using an analogue channel.
Codec: Implements an analogue channel using a digital channel.
Modems and **Codecs** have a DAC (Digital to Analogue Converter) and (ADC) Analogue to Digital Converter.
7.3) Waves: Wavelength: λ Length of a single cycle.
Amplitude: Maximum displacement/strength of the signal.
Period: p The time taken to complete a cycle = 1 / f.
Frequency: f The number of cycles per second.
Wavespeed = fλ. **Phase**: Two waves with the same f and λ might be offset by the same distance – known as Phase Difference. Max phase difference is π – where the waves are at exactly opp displacements everywhere in their cycle. c = fλ for radio transmission. (c = 3*10⁸m)
In copper or fiber transmission speed usually slows to 2/3 c.
7.4) Modulation: Modulation schemes are used to change an information signal into one more suitable for transmission.
Baseband Modulation Transmit unmodified (for dedicated lines)
Broadband Modulation Uses a basic carrier signal to encode information. The carrier signal has modifications added to encode information (e.g changing amplitude, frequency/phase - shared lines)
Better Modulation: To improve the data rate we transmit multiple bits per symbol (in modulation scheme). Use more phase differences, amplitudes. QAM does this by altering phase and amp at the same time, which lets it communicate multiple bits at once.
7.5) Digital Subscriber Line (DSL): We used to have 3kHz bandwidth limit on phone lines (intended for human voice). This was removed for a higher bandwidth, but now **noise** is a limiting factor.
7.6) Asymmetric DSL: We divide our of bandwidth into 4000 Hz channels (1.1 MHz = 256 channels). Channel 0: Voice, 1-5 Unused to prevent interference. The rest split into upload, download (more download as used more). **ADSL Splitter** splits voice/data. **ADSL** modem performs modulation. **DSL Access Multiplexer** (owned by ISP) connects local telephone cables to the ISP. 1)**ADSL2**: 12Mbps, 2.2 MHz 2)**ADSL2+:** 12 Mbps, 2.2 MHz, More bits per symbol
3) **VDSL:** 52 Mbps, 12 MHz, Very-high-bit-rate DSL 4) **VDSL2** 200Mbps, 30 MHz, Currently popular.

8) The Future of Networking:
1) Faster Networks: Use of ASICs (Application Specific Integrated Circuits) to make faster network switches. Barefoot Networks (purchased by Intel in 2019). They create high speed Ethernet ASICs with a programmable pipeline (using a language called P4). Their Tofino 2 switch can handle 12.8 Tbps. Many other companies such as Cisco also vend ASIC based network gear. Another consideration is using light as a medium for secure communications, better fibre optics.
2) Faster Wireless: Kumu Networks have developed programmable filters to allow wireless devices to cancel out their own transmissions. This allows full-duplex wireless as wireless devices can receive and transmit simultaneously on a single channel. Scientists in Japan and Germany have developed on terahertz transmitters (1.1 THz).
3) Legislation Net-Neutrality laws in the USA (though can affect the entire world as they affect internet infrastructure) allow ISPs to be selective about services provided for content on the internet (e.g slowing down a competitor's website, offering special packages allowing access to a limited number of sites).
4)Wireless Mesh Allowing many wireless devices to form a mesh network. For example Cisco Meraki allows for networks to self-heal when parts of the network (e.g switches) fail, by rerouting data wirelessly.
5) Software Defined Networking (SDN) and Network Functions Virtualization (NFV)
A network architecture where applications and services are abstracted from the network infrastructure & control. Useful for containerisation, and being developed by Nicira (now owned by VMWare), Cisco and others.
6) Web Decentralization
The internet has become more centralised around large CDNs such as Amazon's, Google's, and around few large services (e.g facebook, youtube). This is bad for reliability (if a few backbones go down, large services disappear). New protocols such as IPFS intend to resolve this. Many users can aide decentralisation by using their own domains, storage (e.g instead of google drive, dropbox) and their own hosting