**OPERATING SYSTEM**

**CPU** – fetch, decode, execute instructions. Has general registers to hold variables and temp results. Special registers: Program counter, Stack pointer, Program Status Word (Mode bit tells if kernel or user mode, 0 for kernel, 1 for user).

**OS** – hardware abstraction, resource management. **Kernel:** core of OS with complete control over the system, ensures several processes concurrently active do not get in each other’s way. Kernel mode accessed by interrupts (timer/IO), exceptions (unexpected program behaviour, malicious behaviour like buffer overflow), and system calls (protected procedure call).

**I/O** – access via system call. Can use busy waiting, interrupts or direct memory access (device controller has own processor)

**System calls** – part of kernel code. Allows programs to switch to kernel mode and access privileged instructions and privileged memory. Steps involved = Put system call number in register, execute a TRAP instruction (exception that can’t be restarted) and start executing at fixed address within kernel. Kernel code examines system call number and dispatches to the correct system-call handle, system call handler runs and control may be returned to user program. System call may block caller -> OS will see if some other process can run next.

E.g. system calls -> fork(), fd = open(), exec, mkdir, chdir

**PROCESSES AND THREADS**

**Process** - abstraction of running program, encapsulates own resources and information (address space, registers, open files, related process, program text, program data, stack, heap etc.)

**Multiprogramming** – processes are run in pseudo parallel (actually sequential), execution interleaved, cannot make timing assumptions. Time multiplex CPU, space multiplex main memory. Increases efficiency of CPU and memory usage and gives responsiveness. Time multiplex = take turns using something, space multiplex = receive portions of a resource.

**Process creation** – startup, another process, user, batch job startup

**Fork** – create exact clone of current process in different memory

**Execve** – replace current process with new program

**Process termination** – normal exit, error exit, fatal error, killed by process

**States** – running, ready, blocked (waiting for external event like input)

**Process table** – one entry per process, contains state information required for resuming process. Linked list of PCBs (process control blocks) stored in the Operating System.

**Per-thread** – program counter, registers, stack, state

Threads can communicate with each other without invoking kernel, **share code, data, files, global variables** and dynamic heap memory. Offer reduced overhead in terms of creation, termination, switching and communication between each other. Kernel level threads are managed individually by the kernel and can be individually blocked. User-level threads don’t require system calls to be created and terminated and thus have less overhead than kernel level threads.

**Race conditions** – hard to debug as non-deterministic output, final result of threads / processes depends on order of execution.

**Mutual exclusion** – ensure one process exists in the critical region where shared resource/data is accessed.

**Good mutual exclusion** – 1 process in critical region at a time, no speed or amount of CPU assumptions, no process outside critical region may block processes, no process should wait forever to enter

**Busy waiting** – **lock variable** (interrupt before lock is set still causes race condition), **strict alternation** (for 2 processes, they each set a turn variable that lets the other process enter critical region, could still lead to blocking outside critical region depending on order of processes), **TSL** (test and set lock, uses atomic operations that must be completed to avoid interrupt problem). When process wants to enter critical region, it checks if entry is allowed, if not then continuously checks via loop. Waste of CPU and low priority process may have a lock on something that a high priority process is waiting for, or even worse a medium priority process may block the low priority process holding the lock and the high priority process could wait indefinitely (i.e. starve).

**Blocking** – **mutex locks and unlock**, schedules another thread if critical region is locked

**Deadlock** – Each process in the set is waiting for an event that only another process in the set can cause

**Four conditions for deadlock** – Each resource is currently assigned to exactly one process or is available (mutual exclusion), Processes currently holding resources that were granted earlier can request new resources (hold-and-wait), Resources previously granted cannot be forcibly taken away from a process (No-preemption), there must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain (Circular wait)

**Dealing with deadlocks** – ignore problem, detection (e.g. graph algorithms) and recovery, avoid by careful resource allocation, prevention of one of the four conditions of deadlock. Arrow pointing towards resource = process A wants this resource.

**SCHEDULING**

**When to schedule** – process creation, process exit, process block, I/O interrupt, clock interrupt (preemptive).

**Preemptive scheduling** – pick a process and let it run for a maximum of fixed time (quantum), clock interrupts are needed at the end of the time interval to give control of the CPU back to scheduler.

**Non-preemptive scheduling** – pick a process and let it run until it blocks, terminates or voluntarily releases the CPU.

**Context switch** – happens every time the OS switches between processes, expensive and involves saving and loading registers and updating memory tables. Happens more in preemptive scheduling.

**Scheduling environments** – batch (no user interaction), interactive (users expect fast response times / responsiveness from system), real time (need to meet deadlines)

**General scheduling goals** – fairness, use resources efficiently.

**Non-preemptive scheduling** – **FIFO/FCFS** (processes run in order of arrival), **Shortest Job First** (run process with shortest service time in current list of waiting processes).

**Preemptive scheduling** – **Round Robin** (let processes run until interrupt after quantum), **Shortest Remaining Time Next** (preempts when a process unblocks or a new process is created, then lets process with least remaining service time run), **Priority Scheduling** (always run the highest priority process that is ready to run, can decrease priority after quantum or use max quantum to prevent indefinite runtime of high priority processes, possible that low priority jobs will starve if high priority jobs continually arrive, should let I/O bound processes quickly request I/O so that they don’t have to wait for CPU-bound processes which use much more CPU time). **Priority Classes** (use priority scheduling among the classes, round robin within each class and lower process’ priority after class defined quantum to prevent starvation of other processes)

**MEMORY MANAGEMENT**

**Why memory management** – support multiprogramming, isolation between processesand OS memory, enable a computer to run more process with larger memory requirements, e.g.: A single process can use more memory than what is physically available, The sum of the memory requirements of all processes can be larger than what is physically available

**Memory manager** - part of the OS that keeps track of which parts of memory are in use, allocates memory to processes when they need it, deallocates memory when processes are done. Need memory abstraction as constantly swapping from HDD to main memory is expensive and multiplexing memory without abstraction would cause programs to access the wrong parts of memory. Use logical addresses, program instructions refer to logical addresses which can be converted into physical addresses by MMU in CPU package

**Base / relocation register** - start of physical address space of program

**Limit register** – specifies size of range of addresses (logical address must be less than limit register and base register is added to it then, otherwise a trap: addressing error is thrown)

Allocate free memory to processes and keep track of allocations via abstractions such as bitmaps and linked lists.

**Bitmaps** – 1s in bitmap signify used up allocation unit, 0 indicate free memory (bigger allocation unit = smaller bitmap = bitmap takes up less memory but processes may not need majority of last allocation unit and so that memory is simply wasted)

**Linked lists** – Process or hole, start of memory allocation / hole, length of memory allocation / hole

**How to choose a free block** – First fit (from start of list), Next fit (from previous allocation), Best fit (smallest hole that fits, checks whole list and may leave many small holes (internal fragmentation), Worst fit (largest hole that fits), Quick fit (separate lists for processes and holes depending on their size, cumbersome to maintain this data structure every time a hole is made from process termination)

**Virtual memory** – supports programs too large to fit in memory or collective program requirements exceeding memory. Each program has its own address space broken up into pages (contiguous range of addresses, but the pages themselves don’t need to be stored contiguously). Pages are mapped onto physical memory, and not all pages have to be in physical memory at the same time to run the program. If virtual address is in memory then MMU performs the necessary mapping to retrieve memory, otherwise OS gets missing piece from disk and re-executes the instruction.

**Paging** - Divide virtual memory into pages, divide physical memory into page frames, map pages into page frames, set up a page table to translate virtual to physical addresses.

**Page size** = 2^n, **Virtual Address Space** = 2^m, **page number** = m – n, **page offset** = n, **virtual address =** virtual page number x page size + offset, **virtual page number** = floor(virtual address / page size), **Number of pages** - memory / page size, **Physical memory to store PT –** number of pages x size of PTE

**Example**, 16 bit address with 4kB page size, upper 4 bits could specify one of 16 virtual pages and lower 12 bits could specify byte offset (0 to 4095). The virtual page number is used as an index into the page table to find the entry for that virtual page. The page frame number replaces the virtual page number to form a physical address that can be sent to memory.

The MMU must check that the selected page table entry has the present bit set to 1, and that the permissions allow the requested memory access. If both conditions are met, it will construct the physical address (if not present -> page fault, if violates permissions -> protection fault (e.g. segfault)). It will then set the referenced bit, and if the access was a write, it will also set the modified bit.

Mapping from virtual to physical address must be fast (but we need to access the page table and then access the data in two separate memory accesses), and if the virtual address space is large, the page table will be large. **Transition Lookaside Buffers (TLB)** used to hold copies of recently accessed page table entries (PTEs) that can be checked simultaneously due to being in a faster cache in CPU with associative circuitry. TLB is flashed when context switching as different process’ virtual pages may not map to same page frames. Note that binary arithmetic works most efficiently with powers of 2, so page sizes being powers of 2 improves performance and reduces memory usage.

**Page Fault Handling** – Page fault raises an interrupt, which suspends current process and jumps to the page-fault handler in the kernel which checks that the page reference was legal, frees up a page frame if necessary, loads the required virtual page from swap space (disk) into a free page frame (OS could give CPU time to another process while waiting for IO operation), updates the page table and restarts the process at the same instruction.

Page Replacement Algorithms – Discard page if not modified, write to disk if modified. **Optimal algorithm** (replace page we know will be used further in future (or never used), not practical to track this), **Not Recently Used** (get rid of R:0 M:0, R:0 M:1, R:1 M:0, R:1 M:1 class pages in that order, R and M bits updated on each memory reference and can set R bits to 0 at fixed time interval with clock interrupt), **Second Chance Algorithm** (FIFO but if page at head of queue has R bit set then move it to tail and set reference bit to 0, if R bit is 0 then remove page from queue), **Least Recently Used** (Good approximation of optimal algorithm, keep a bit counter for each page, on each clock tick / interrupt shift the counter by 1 bit to the right, if the page is referenced, add 1 to the leftmost bit, evict page with lowest value bit counter. Only tracks x clock interrupts / ticks so length of time between ticks and amount of bits in bit counters matters.

**Temporal locality** – a page recently accessed is likely to be accessed again in the near future

**Spatial locality** – Accessing a page is likely to lead to accesses to nearby pages

**Working set** – pages that a process is actively using within last T seconds or K memory references, maintaining working set reduces page faults, (pre-fetching/pre-paging = set up working set before running process)

**CRYPTOGRAPHY**

**Why? –** Supports secure communication by enabling **Confidentiality** (Only the sender and intended receiver should be able to understand the contents of the transmitted message), **Authentication** (Establish the identities of one or both of the end-points), **Integrity** (Ensure that messages are not altered).

Encrypt into ciphertext, decrypt back into plaintext. Cryptography is based on problems that are considered to be so computationally hard that a brute force attack would take way too long to perform.

**Symmetric Cryptography** – Same key used for encryption and decryption.

**AES** – Modern example of symmetric cryptography, breaks data into blocks and encrypts each block, has different modes of operation (ECB, CBC, CTR, etc.) which determine how each block is treated and / or linked. Encrypt(SecretKey, Message) Decrypt(SecretKey, CipherText). Have to find way of securely exchanging secret key.

**ECB –** Simple, can perform encryptions and decryptions in parallel without race conditions but repeated patterns will be evident in output as same plaintext encrypts to the same ciphertext each time.

Passwords should be hashed, not encrypted as if someone has the key then they can decrypt all passwords.

**CBC –** Non-secret Initialisation Vector (IV) that must be random and not reused. Encryption must be done sequentially (as each encryption relies on ciphertext of previous encryption), decryption can be done in parallel (as each decryption relies on ciphertext of previous decryption which is available from beginning). Loss or corruption of a ciphertext block or IV affects correct decryption of at most 2 blocks.

**Asymmetric Cryptography –** Two keys, public key and private key. Public key is shared and can be used by anyone to encrypt a message which the owner of the private key can decrypt using the private key.

Asymmetric Crypto is slower than Symmetric -> isn’t suitable for encrypting large amounts of data or even multiple blocks. Can use to exchange a joint secret key for symmetric cryptography usage.

**Digital Signatures –** Provide a link between a key holder and a message / document / file, importantly is considered to provide non-repudiation (cannot deny having signed the document), provides integrity against tampering. Bob signs (encrypts) a message using his Private key. To verify the signature, Alice decrypts using Bob’s public key and compares the decrypted message to the original message. If they match the message must have been encrypted using Bob’s private key, only Bob could have signed the message (unless his private key is compromised) and the message hasn’t been tampered, this verification process obviously ignores confidentiality.

With large documents, use a Cryptographic Hash Function which takes a near arbitrary length input and outputs a fixed length digest, sign the Hash Digest (Encrypt(SignKey, h) where h = Hash(m)). Can verify using Hash Digests by hashing the original message and comparing to the encrypted hash digest received after decrypting it.

Hash function H is a lossy compression function -> same hash output for different input documents (collision). H(x) should look “random”. A cryptographic hash function should be collision resistant, hashing doesn’t require a key and cannot be reversed. (good for password storage, software integrity).

Degree of security is linked to key size, key lengths reflect the different mathematically hard problems they are based on. Asymmetric is generally longer than symmetric.

Cryptography is dependent on randomness (key generation, IV generation, padding). But computers are pseudorandom, randomness must never be used or discoverable and generating good randomness is hard and dependent on good OS implementations.

**Message Authentication Code –** detect if message has been tampered with, s = MAC’s secret authentication key, m = message, t = MAC(s,m) [tag]; b = Verify(s, m, t) b is 0/1 indicating verification success. Verifies integrity of message using a secret key (adversary cannot create (m’,t’) such that Verify(s,m’,t’) returns b=1 for m’ it has not seen). Doesn’t maintain confidentiality by itself (as message is sent in plaintext).

**Authenticated Encryption - Confidentiality** and **integrity** of messages exchanged between Alice and Bob, **Encrypt-then-Mac:** c = Encrypt(SK, m), t = Mac(s, c). If Verify(s,t,c) returns 0 -> don’t decrypt. (know ciphertext has not been modified if returns 1)

**Diffie-Hellman Key Exchange –** Agree on a shared key, provides perfect forward secrecy: exposure of long-term keys does not compromise security of past sessions (cannot decrypt past sessions with key of current session). Sends information in a way that allows both parties to calculate a shared key without having to ever explicitly communicate the shared key. Generate public large prime p, primitive root modulo p (generator g). Alice picks a random value x and compute X=g^x mod p, send X to Bob, Bob picks random value y and computes Y=g^y mod p. Alice calculates the secret s=Y^x mod p = g^(yx) mod p and Bob calculates the secret s=X^y mod p = g^(xy) mod p = g^(yx) mod p. No authentication, vulnerable to MITM.

**Man-in-the-Middle attack –** Trick one party into thinking they are communicating with the other by modifying messages (impersonation), or silently snoop on the messages sent between parties (eavesdropping) (DH isn’t secure against impersonation but is secure against eavesdropping as attacker needs x or y to decrypt)

**Digital Certificates and Public Key Infrastructure –** Digital certificates securely associate identities (name, domain, organization, etc.) with cryptographic public keys (proof of identity). The certificate itself is signed often by a third party. Identities are verified by certificate authority and signed (encrypted with private key). CA creates a certificate for Bob including a digital signature (a hash of Bob’s information, including his public key) which is signed by CA’s private key. To verify, Bob sends Alice his public key in plaintext along with the certificate signed by a root CA. Verify certificate by using CA’s public key. Check if Bob has private key by picking a random number and sending to Bob decrypted by his public key, if the random number (nonce) can be sent back in plaintext, it must be Bob.

**Certificate Hierarchies –** The signer is vouching for the identity behind the public/private key pair. Certificates can be chained: A signs B’s certificate, B signs C’s certificates and C runs a website or issues digital signatures. If you trust A, then you trust what A has signed and therefore what those entities have signed and so on. Verify (PKofB, SigOfCertCbyB, CertC), Verify(PKofA, SigOfCertBbyA, CertB).

**Trust anchors –** entities that are explicitly trusted, most commonly found as root certificates, points from which all trust is derived. Certificate Authorities are most common trust anchors.

**TLS –** Transport Layer Security Protocol, supported by popular web browsers, web servers, internet commerce sites. Has a handshake protocol where public-key cryptography is used to establish several shared secret keys between client and server. An initial negotiation between client and server establishes the version of the protocol and set of cryptographic algorithms to be used. Record protocol uses the secret keys established in the handshake protocol to protect confidentiality, integrity, and authenticity of data exchange between the client and the server. Can authenticate server and client by using digital certificates to learn each other’s public keys and verify each other’s identity. **Process:** TCP connection established, client sends ClientHello to server asking for secure connection with cipher suites. Server responds with ServerHello and selects one cipher suite, also includes its certificate and can request client to send its certificate (mutual authentication). Client confirms validity of certificate. Client generates session key by random number or Diffie-Hellman. Handshake concludes and both parties share a secret key.

**NETWORKING**

**Internet –** Aggregation of many smaller networks

**ARPANet vs OSI –** Fast implementation vs careful planning, standaradisation pre-implementation, and concern for scalability.

**Network Models –** Model network as stack of layers, each layer offers services to layers above it. Inter-layer exchanges are conducted according to a protocol (rules which govern the format and meaning of packets exchanged between peers within a layer).

**Connection Oriented –** connect, use, disconnect like a phone call

**Connectionless –** Message routed through intermediate nodes like a text message

Choice of service type affects the reliability, quality and cost of service

The TCP/IP model reflects what happens on the internet whereas the OSI model helps reflect the thought process that should be followed when designing a network or diagnosing a fault (ideal model).

**OSI –** A layer should be created where a different abstraction is needed, each layer should perform a well defined function.

(Physical get p2p, Data Link tidy p2p, Network get e2e, Transport tidy e2e, Session, Presentation, Application)

**P2P:** direct hop between two network nodes (routers), **E2E:** concerned only with beginning and final destination.

**TCP/IP –** Presentation and Session included in Transport Layer, no physical layer, just link.

Data travels from application layer down, and back up at receiving end, may also pass up to link and network layer when passing by routers.

IP over everything, everything over IP narrow waist architecture.

**HTTP AND HTML**

**HTTP –** HyperText Transfer Protocol, defines everything needed for the web by allowing file transfer. Application Layer.

**URL –** Uniform Resource Locater, address for a resource which can be relative or absolute

Client initiates TCP or QUIC connection (creates socket) to server, port 80, server accepts connection, HTTP messages exchanged between browser and web server, TCP connection closed.

**Non-persistent –** requires 2 response times per object + file transmission time, OS overhead for each TCP connection. **Persistent –** Server leaves connection open after sending response, 1 RTT for connection establishment, 1 RTT per object + file transmission time.

**Link is selected –** Browser determines URL, browser makes TCP connection, sends HTTP request for page, server sends page as HTTP response, browser fetches other URLs as needed, browser displays the page as content arrives, TCP connection released

**HTTP Requests – GET (request data from specified resource URL)** Safe (just retrieves info) Idempotent (identical requests have same effect) Cacheable (result can be stored), **HEAD (requests header that would be returned if GET was used with supplied URL, can check info without downloading file)** SIC, **POST (send data to server to create / update resource)** C(can also not be), **PUT** I, **DELETE** I, **CONNECT**, **OPTIONS** SI, **TRACE** SI, **PATCH**. Request line, header lines, **blank line indicates end of message**

**HTTP Response Codes –** 1xx Information, 2xx Success, 3xx Redirection, 4xx Client error, 5xx Server error. **200 = request succeeded, 404 = page not found.** Status line (protocol status code and phrase), header lines, data (e.g. requested HTML file)

**HTTP Headers – Host** (get Server’s DNS name), **Authorisation** (list of client credentials), **Server** (Information about the server), **Accept** (the type of pages the client can handle), etc.

**HTTPS –** HTTP over TLS, everything encrypted, not just payloads. **HTTP/2 and /3 –** decrease latency from previous iterations

**Multi-threaded Web Server –** multiple threads (processing modules) handle client requests **Web Cache –** can store data instead of having to always retrieve it. **HTML –** Hypertext Markup Language, simple language that encodes content and presentational information <head> </head> <img src=”pic1.gif” alt=”pic1.gif”>

**DNS, RPC, EMAIL**

**DNS** – domain name space, technology behind mapping host.domain.com to IP address. Distributed (tree shaped) hierarchical database of mappings of host names to IP addresses. Runs over UDP as DNS is idempotent, fast (no setup) and has low overhead. Organization provides publicly accessible DNS records that map host names to IP addresses for their web servers, mail servers, etc. No security in original DNS design, vulnerable to spoofing (altered DNS records redirect to fraudulent website) and flooding particular domain’s DNS servers.

**DNS database –** Each node / leaf in the name space tree names a set of information that is contained in a resource record (RR).

**Name servers –** Server programs that hold information about a portion of the domain name tree structure and associated RRs

**Resolvers –** Programs that extract information from name servers in response to client requests.

DNS namespace divided into overlapping zones, name servers are authoritative for that zone (usually two name servers for that zone). Name servers are arranged in a hierarchical manner extending from a set of root servers which are contacted when local name servers cannot resolve names.

**Authoritative DNS server:** DNS server will contain Type A record for the host name. i.e. a direct IP address mapping. Final destination for the address. **Non-authoritative DNS server:** Server contains a Type NS record for the domain that includes the hostname, and it will contain a Type A record that provides the IP address of the DNS server in the value field of the NS record.

**Top-level domain** DNS servers responsible for com, org, net, edu, uk, au, etc. **Authoritative** DNS servers responsible for mappings for organization servers. Typically, each ISP has a local default name DNS server **(root)** which handles DNS queries and returns cached value if one exists, otherwise acts as a proxy and forwards request up the query hierarchy. (Note that timeouts exist for queries). Root -> TLD -> authoritative.

**Record Types: A:** Name is hostname, value is ip (relay1.bar.foo.com -> IP). **NS:** Name server, name is a domain, value is host-name of authoritative DNS server that knows how to obtain the IP for hosts in the domain (foo.com -> dns.foo.com). **CNAME:** canonical name for the alias hostname (foo.com -> relay1.bar.foo.com). **XS:** value is canonical name of mail server that has an alias name (foo.com -> mail.bar.foo.com).

**Remote Procedure Calls –** Allow calling procedures on a remote server as if they are local to the client, dozens of variants exist for RPC. Client process is suspended whilst execution takes place on server machine. To hide networking, client and server bound to their own stubs and from the perspective of both, all calls are local. Parameters passed and returned in ‘marshalled’ format (so that there aren’t issues with data types, array lengths, addresses, etc)

**Email –** User agent program allows user to perform actions related to sending and receiving mail. Mail message has To, From, Subject, and other hidden header lines, blank line, and body containing the message.

**SMTP –** Simple Message Transfer Protocol uses direct persistent TCP connection to reliably transfer email message from client to server (default port 25). There’s the handshaking, transfer of messages, and closure. **Messages must be 7 bit ASCII**. Commands in ASCII text and response consists of status code and phrase. Can be many back-and-forth exchanges, slow on modern networks where latency is larger than serialization delay (putting data into transmission line), modern approach of one header has lower delay. Pushes information whereas HTTP pulls things from servers. HTTP puts each object in separate response message, SMTP combines messages into one sent message.

**MIME –** Multipurpose Internet Mail Extensions, has additional message headers and allows sending of different content types besides ASCII text.

**SMTP handles delivery and storage to receiver’s server, accessing mail requires different protocols and a message transfer agent using these protocols.**

**POP3 –** Post Office Protocol. Authorization phase through user and pass client commands with OK or ERR server response. Transaction phase where can list message numbers and retrieve messages by number, delete them and then quit.

**IMAP –** Internet Message Access Protocol, POP3s download and delete operation does not allow messages to be re-read, IMAP keeps user state across sessions. Retains mailbox contents online and allows manipulation of messages.

**TRANSMISSION CONTROL PROTOCOL**

Network layer below gets packets e2e, but there are many issues that need to be tidied up by the transport layer. TCP lets applications transmit and receive a byte stream without worrying about segmenting into IP datagrams (stream oriented), bytes being dropped or duplicated (reliable), and bytes arriving out of order (in order). TCP entity segments user data streams into pieces < 64kB (often 1460 bytes in order to fit IP and TCP headers into single ethernet frame), and sends each piece as a separate IP datagram. Recipient TCP entities reconstruct original byte streams from the encapsulation.

**TCP Primitives – Listen** (block until something tries to connect), **connect** (actively attempt to establish connection), **send** (send information), **receive** (block until data packet arrives), **disconnect** (this side wants to release the connection). Select is a non-TCP primitive allowing non-blocking receive.

Sender and receiver both create sockets (5-tuple of IP address and port number of sender and receiver, and the protocol). For TCP, connections must be explicitly established between a socket at a sending host and a socket at a receiving host.

**TCP Features – Full duplex** (data in both directions simultaneously), e2e (exact pair of senders and receivers), **byte streams** (not message streams and so message boundaries are not preserved), **buffer capable** (can choose to buffer prior to sending -> less headers but increased delay obviously).

**TCP Properties** - Data is exchanged between TCP entities in segments (each has a 20-60 byte header, plus zero or more data bytes), TCP entities decide how large segments should be, given two constraints: IP payload < 65,515 bytes, Maximum Transfer Unit (generally 1500 bytes). Sliding window protocol.

**TCP Important Headers** – **Source Port**, **Dest port**, **Sequence Number** (for sliding window, if SYN = 1 then is initial sequence number, else is accumulated sequence number of the first data byte of this segment), **Acknowledgment Number** (if ACK=1 next sequence number that the sender of the ACK is expecting), **Data offset** (size of TCP header 20-60 bytes = 5-15 x 32-bit), **Flags** (single bit flags), **Window Size** (Size of receive window – how much data sender of this segment is willing to receive)

**TCP Flags – URG** (notify the receiver to process urgent packets before all other packets), **ACK** (acknowledge successful receipt of a packet, receiver sends an ACK as well as a SYN in second step of the three way handshake to tell sender it has received the initial packet), **PSH** (similar to URG, process these packets instantly instead of buffering), **RST** (reset the connection, indicates to the receiver to terminate the connection when unrecoverable errors occur), **SYN** (used in connection establishment to tell other end which sequence number they should accept, set to 1 when connection established), **FIN** (used to request connection termination).

TCP is connection-oriented running over connectionless network layer (IP). When networks can lose, store and duplicate packets, connection establishment can be complicated (congested networks may delay acknowledgements, incurring repeated multiple transmissions, any of which may not arrive at all or out of sequence – delayed duplicates).

**Reliable connection establishment –** ensure one and only one connection is established even if some set-up packets get lost and/or retransmitted. Establish initial sequence numbers for sliding window. Three way handshake avoids problem of both sides allocating same sequence number by accident. Sender and receivers exchange information about which sequencing strategy each will use, and agree on it before transmitting segments.

SYN is used for synchronization during connection establishment, sending SYN or FIN causes sequence number to be incremented by 1. **Sequence number** = first byte of this segments payload (1 + data sent prior to this segment. Offset by a random number, initial value is arbitrary, offset will be reflected in both sequence and acknowledgement numbers. **Acknowledgement number** = next byte the sender expects to receive (data successfully received + 1, doesn’t increase if it receives later segments before receiving what’s expected). Connection request has SYN=1, ACK=0. Connection reply has SYN=1, ACK=1.

**TCP Retransmission –** Retransmission Time Out (RTO), initial timeout is default value, updated based on network performance. If the timer expires before an ACK is received, the segment is resent. Resets whenever a “new” ACK arrives (“the window slides”). If the receiver receives segment with a sequence number higher than expected (segment has been lost, or out of order), then receiver sends ACK with sequence number it is expecting (i.e. next byte it expects – also implies what data it has received). This is a DupACK if it is ONLY an ACK, not piggybacked on data as well. After receiving 3 DupACKs (4 ACKs) the sender resends the lost segment. This is **“fast retransmission”** (no RTO)

**TCP Closing –** FIN flag signifies request to close connection (once acknowledged, no new data can be SENT from FIN sender to receiver but can still be received, sender of FIN will also still retransmit unacknowledged segments. Typically requires 1 FIN and 1 ACK segment for each direction, can be optimized with FIN, FIN/ACK, ACK if there’s no unfinished business. RST flag signifies a “hard” close of a connection where sender will not listen for any further messages, usually sent in reply to a packet sent to a socket with a non-open connection. Can be used but FIN is preferred as a more orderly shutdown.

**SOCKET PROGRAMMING**

**Socket –** doorway leading in/out of the application, 5 tuple (src ip, src port, dest ip, dest prt, protocol). UDP 3 tuple (src ip, src port, protocol) UDP packet header has dest ip and dest port.

**Socket Primitives – SOCKET** (create communication endpoint), **BIND** (associate a local address with a socket / bind to a specific port number), **LISTEN** (announce willingness to accept connections; give queue size), **ACCEPT** (passively establish an incoming connection, block until then), **CONNECT** (actively attempt to establish a connection), **SEND** (send some data over a connection, write()), **RECEIVE** (receive some data from connection, read()), **CLOSE** (release the connection).

**Socket Finite State Machine** – shows different states of sockets and transitions, simplified diagram has idle to established to idle with passive and active connects and disconnects between established and the idles.

**Multi-threaded Server –** A server needs to be able to handle concurrent connections from multiple clients and can do this through a multi-threaded server with a single dispatcher and multiple worker threads.

**TCP SLIDING WINDOW**

**Sliding Window –** controlled by receiver, determines amount of data the receiver is able to accept, sender and receiver maintain buffers independently of application, no guarantee that data is immediately sent or read from buffers. When the window is 0, the sender should not send any data, can send urgent data outside of main stream though. Can also send “zero window probe” which is 0 byte segment that causes the receiver to re-announce ACK and window size. Senders may delay sending data to fill the receive window in one send. **Send window =** what data the sender is able to send, unacknowledged segments and unsent data that will fit into the receive window. = **min(RWND,CWND), if RWND > CWND then send size of CWND which JUST avoids network congestion. If CWND > RWND, send size of RWND which leaves room in CWND.** **Receive window =** Amount of data the receiver is willing to receive – window size in ACK. Other windows are maintained for congestion control. The receive window slides as the application reads data in it, and can send a **WindowUpdate** to sender to let them know this has happened and prevent a deadlock where sender sees window size is 0 and receiver won’t receive anything so won’t send any ACKs to increase window size (note this doesn’t count as a DupACK). Sender can also send a **ZeroWindowProbe** based on a persist timer when they know the Window Size is 0 to keep checking when they can start sending again (ZWP has 0 data bytes). Sender has to make sure LastByteSent – LastByteAcked <= ReceiveWindowAdvertised (i.e. cannot send data past current send window) SEQ(packet about to be sent) <= last Acknowledged packet + min(CWND, RWND)

**QUIC –** another reliable transport protocol, runs over UDP as firewalls block protocols other than TCP/UDP. Multiple byte streams instead of just one, TLS handshake combined with TCP handshake (fewer idle round trip times (RTTs)), can connect to a different IP address mid flow unlike TCP). Has listener sockets like TCP/UDP, has connections sockets which unlike TCP/UDP are only used to manage the streams belonging to the connection. Stream sockets used for data transfer.

**TCP CONGESTION CONTROL**

**Flow control** deals with managing the sender and receiver buffers and ensuring the receiver’s buffer doesn’t overflow, **Congestion control** relates to managing the buffers of routers in between (think of flow control as managing the table which letters are placed onto and congestion control as managing the post office / mailman’s intake.

**Segment Loss –** Receiver will send 3 DupACKS and cause fast retransmission if it receives 3 segments in a row without receiving the bytes it currently is ACKing. Window size does not change as window size is number of slots in buffer beyond last ACK (i.e. doesn’t count slots ahead of ACK as being full).

**Flow+loss control** had existed on single point-to-point links for a long time, TCP used experience from the link layer which led to bad design. Two types of flow control – **“Go-back-N”** where you go back to a lost packet and retransmit everything from that point onward which means receiver doesn’t need to store/reorder packet. **“Selective Repeat”** where you only retransmit the lost packet, which is much, much faster but packets arrive out of order and receiver must store out-of-order packets and send them in-order to the application. Selective Repeat is more complex and only helps if loss is common.

When networks are overloaded, congestion occurs, potentially affecting all layers. Lower layers (link and network) attempt to reduce congestion but TCP ultimately affects congestion the most as it offers methods to reduce data rate and therefore congestion. In original TCP: Initially, the receiver chooses a window based on its buffer size, if the sender is constrained to this size then congestion problems should not occur due to buffer overflow at the receiver but may still occur due to network congestion. Jacobson introduced the **Congestion Control Window (CWND)** at the same time as selective repeat. CWND is additional window that is dynamically adjusted based on network performance to aid efficient transfer. **The congestion window size is maintained by the sender, unlike the sliding window controlled by the receiver.** It is also only used at the sender and doesn’t require any changes to packet formats for additional headers.

**What happens when an ACK is received: Slow start:** increase CWND exponentially (**double until loss or you get to SSThresh which is set by TCP state).** If you get to SSThresh, move into congestion avoidance (increases CWND linearly (1 per RTT)). **ACK Loss?:** Timeout occurs (set CWND to 1MSS, SSThresh = CWND / 2), back to slow start. 3 dupACK (SSThresh and CWND set to CWND/2), triggers fast recovery then back to congestion avoidance if receive new ACK, otherwise if a timeout occurs back to slow start.

**Incremental Congestion Control** – At first, CWND = maximum segment size MSS, sender transmits 1 segment. For each segment acknowledged, CWND += MSS, each full window of acknowledgements doubles the congestion window which grows until a timeout or a threshold is reached SSthresh. **Slow-start algorithm:** Initial rate is slow, grows exponentially from there. Once a segment loss occurs, SSthresh = CWND / 2, start slow start again. Once SSthresh is reached, growth is slowed to linear (stage is called congestion avoidance), achieved by CWND += MSS for each complete window of ACKs (additive increase). Can also react to known lost segments via fast retransmission. Further optimisations were made such as starting from new ssthresh instead of original starting value, effectively avoiding the slow start phase and going straight to additive increase (multiplicative decrease, additive increase). **SACK (Selective Acknowledgements)** provide greater ability to track segments in-flight, by allowing up to 3 ranges of bytes received to be specified, e.g. [ACK 1, SACK: 6, 3-4 (missing packets 2 and 5)]

**Macroscopic Model –** These packet-level rules affect many things such as fairness between flows, response to long round-trip times (RTTs), response to random packet loss so algebraic expressions are useful. W increases once per window (approximation - each packet that arrives increases W by 1/W), when a loss occurs with probability p, W is halved -> with probability p, W = W/2. Therefore the average increase in window size is (1-p)/W – p x W/2. To be in equilibrium, the average increase must be 0 -> W ~ sqrt(2/p). The window is sent at most once per RTT, T.

Rate is W/T ~ 1/T x sqrt(2/p). -> for a given packet loss rate p, longer RTTs get less rate (RTT unfairness). If RTT is small, TCP forces the packet loss rate to be high. This formula is very approximate as window only responds to one packet loss per RTT and packet losses are clustered (not independent), however the insights still are applicable.

**TRANSPORT LAYER SERVICES AND UDP**

**Presentation Layer –** Meant to provide encryption, compression, data conversion, mapping between character sets. Done by application layer nowadays as negotiating encryption is simple, and there aren’t simple common services needed by all applications, and application is not in the kernel so is much more flexible in handling these matters. RTP (Real Time Protocol) is closest thing to presentation layer.

**Session Layer –** provides authentication (confirm identity), authorization (confirm permissions), session restoration. QUIC does all of these. Some protocols which do these are used between layer 2 and layer 3 (note that ethernet “layer 2” for example has many properties of layer 3 and even 4).

All transport layers make sure that data can’t arrive faster than we can handle and that data from one application is not mixed with that for another.

**Packets -** Packets can be used interchangeably for a unit sent by a layer, but segment specifically refers to transport layer, packet supposed to refer to network layer, and frames supposed to refer to link/data link layer.

In the case of reliable connection oriented service – provides notional “perfect” connection between two nodes but doesn’t provide privacy or isochrony (equal delay between packets), hides acknowledgements and congestion control and lost packets and provides this service to higher layers. Unreliable connectionless service provides multiplexing between different processes on the other hand. To make something reliable or unreliable, add or remove acknowledgement. Port numbers can range from 0-65535 (16 bits) allocated by Internet Assigned Numbers Authority (IANA). Classified into **Well Known Ports** (0-1023) [21 FTP, 22 SSH, 23 Telnet, 25 SMTP, 80 HTTP, 110 POP3, 179 BGP, 443 HTTPS), Registered Ports (1024-49151) and Dynamic Ports (49152-65535).

**MUXING and DEMUXING –** Multiplexing is combing multiple distinct streams into a single shared stream and demultiplexing is reversing this process to receive into a single socket.

**UDP –** User Datagram Protocol allows applications to transmit encapsulated IP datagrams without a connection, segments consist of header and payload, **headers contain source and destination ports.** Payload is handed to the process which is attached to the particular port at the destination (using BIND, etc.). UDP provides multiplexing of port numbers over IP. Both source and destination ports are required (destination allows initial routing for incoming segments, source allows reply routing for outgoing segments). UDP has no delay waiting to recover lost packets, but as a result has no flow control, no error control and no retransmission of bad segments. -> Use UDP where you require a precise level of control over packet flow/error/timing.

**UDP Header –** Src port, Dest port, UDP length, UDP checksum, IPv4 pseudoheader included in the UDP checksum.

Overall UDP is simple and efficient, suitable for some settings where only one message in each direction is required and client sends a short request to server expecting a short response (e.g. DNS). Suitable for real-time services like VOIP where losses are concealed by filling in time with “best guesses”.

**RTP –** Real-Time Transport Protocol. RTP multiplexes several streams (e.g. video and audio) into a single stream of UDP segments. Sits in the user space above socket interface and UDP but below application, can be considered transport, application or presentation layer (OSI). Header -> Payload type – encoding used (MP3, etc. can vary each time), Sequence Number – counter incremented on each packet, Timestamp – source controlled relative to start of the stream.

**RTCP –** Real-time Transport Control Protocol, control protocol for RTP, handles feedback, synchronization, and UI. Feedback to source (delay, jitter / variation in packet delay, bandwidth, congestion) used by encoder to adaptively encode to suit network conditions.

**IP ADDRESSES AND PACKET SWITCHING**

Get data from source all the way to destination, maybe in multiple hops. Traffic must be routed efficiently by network devices called routers. Nodes must be given names (addresses). Transport layer cleans up messes of IP. Both Transport and Network Layer work to enable “logical communication” between two destinations (as if they are directly connected and can understand each other perfectly).

Connectionless network layer = packet switching like IP, minimum required service: send packet, called “datagram” network. Connection-oriented network layer – Virtual Circuit Switching (inspired by physical circuits of old phone connections), called “virtual circuit” network, usually act a single “link” of an IP network.

The internet is a packet switched network, host transmits packet to the nearest router, the packet is buffered while it is arriving and the checksum is verified. If valid, the packet is stored until the outgoing interface is free, then the router forwards the packet onto the next router in the path. Repeat. Therefore forwarding is **P2P**

**Connectionless packet forwarding** – done via forwarding/routing tables which have destination and line (next hop / router). Paths can change for packets in same transport layer connection. **Connection-oriented packet forwarding –** Forwarding table with In: Connection ID / sender and their virtual circuit number, Out: next hop and new virtual circuit number. VC number is local to a hop (i.e. pair of nodes in forwarding table). VC number can be represented with less bytes than a full address.

**Connectionless pros and cons –** Routers don’t need to hold state info about connections, simple link failure recovery. However, each packet has full source and destination, providing QoS is difficult and congestion control is difficult. Each packet is routed independently (i.e. routing can change constantly).

**Connection-oriented pros and cons –** Each packet only needs a short VC number. QoS (Quality of Service) is easy if enough resources, Congestion control is easy if enough resources. However, each VC requires router table space, router reboots are a problem as everything is wiped. Link failure recovery requires extra work. Routing is defined at setup.

**MPLS –** MultiProtocol Label Switching, Virtual Circuit Network Layer Protocol below internet sublayer, MPLS network is one IP hop. Primary purpose is Quality of Service (prioritizing traffic, service level agreements for network performance, reliable connectivity with known parameters). Popular with businesses that want to connect to multiple sites and phone companies carrying voice traffic. QoS is important as not all services are equally important or robust to network delay (e.g. VoIP vs file downloads, VPN vs web browsing).

**Internet Protocol –** designed with the idea of keeping it simple and something that works “OK” is better than something ideal “in progress”. Strict when sending, tolerant when receiving (e.g. web browsers handle pages with invalid HTML). Make clear choices, negotiate options and parameters at runtime. Think about scalability. Multiple paths exist through the network as backups, routing algorithms used to determine best path.

**IPv4 –** Version field, IHL (header length in 32 bit words), Total length (including payload, max 64kB), TTL (Time to live, countdown of hops, subtracted by 1 at each router, packet discarded at 0), Transport Layer Protocol, Source and Destination address. IPv4 addresses are 32 bit numbers where each byte is shown as a decimal separated by a period. (0.0.0.0 – 255.255.255.255)**.** IP addresses are given to interfaces not hosts (i.e. a host with multiple network cards will have multiple IP addresses). Note there are unicast addresses (one destination), multicast (multiple nodes), broadcast (everyone), anycast, geocast, etc.

Originally IPv4 addresses were allocated based on **classes**, routing was performed based on the class, which could be derived from the first part of the address. Classes simplify routing as size of “network” field is implicit in address, but if a network has slightly more nodes than a certain class provides, it would have to move to the format of the next class which could lead to a massive waste of addresses. (e.g. network with 260 nodes must be class B with 65,536 host addresses).

**CIDR –** Classless InterDomain Routing. Each interface / route explicitly specifies which bits are “network” field -> networks with 260 nodes only need 9 bits for “host” field = 512 addresses = can have many more of these networks than class B networks. Network top bits, host bottom bits, network part is same for all hosts on that network. Prefixes are lowest IP address followed by a slash and the size of the network portion. E.g. 192.0.2.0/24 (24 network bits, last 8 bits are for host which means up to 256 host addresses). Can also be written as subnet mask, a binary mask of 1’s, in the case of /24: subnet mask is 255.255.255.0 (all 1s for network bits, 0 for host).

**Network/subnet number = network mask/subnet masks (bitwise-AND) incoming IP address.** Crucial for efficient routing on the internet, since networks are assigned in blocks intermediate routers need only maintain routes for the prefixes, not individual hosts. Only when packet arrives at the destination network does the host portion need to be read. **Aggregation** is performed automatically, currently it roughly halves the size of the routing table. Prefixes can overlap – longest (most specific) matching prefix is selected.

**Special IP Addresses –** 10.0.0.0 - 10.255.255.255 = prefix (10.0.0.0/8) mask (255.0.0.0), 16,777,216 available host addresses.

172.16.0.0 – 172.31.255.255 = prefix (172.16.0.0/12) mask (255.240.0.0), 1,048,576 available host addresses.

192.168.0.0 – 192.168.255.255 = prefix (192.168.0.0/16) mask (255.255.0.0), 65,536 available host addresses. These are private address ranges.

169.254.0.0 – 169.254.255.255 = prefix (169.254.0.0/16) mask (255.255.0.0), 65,536 available host addresses. Link local / zero config address range.

00000..00000000 = this host (placeholder for invalid address)

0000..000 | Host = a host on this network

111111..111111 = broadcast on the local network

Network | 111..1111 = broadcast on a distant network **this address can’t be used by a host**

127. | Anything = loopback address

**IPv6 –** Address problem of exhausting IPv4 address space. Some other changes were also made to improve upon IPv4 such as simpler header, improved security and further QoS support. 128 bits, unlikely to run out unless wasteful allocation schemes are used. Checksum already at transport layer so removed from header.

**IPv6 Addressing –** Written as 8 groups of (up to) 4 hex digits. 8000:0000:0000:0000:0123:4567:89AB:CDEF. Can be optimized by stripping one group of consecutive 0s – 8000::123:4567:89AB:CDEF, if stripped more then wouldn’t know how many 0s were stripped. Backwards compatibility with IPv4 via ::ffff:192.31.2.46 for example. Not widely deployed yet.

**NAT AND SUBNETS AND FRAGMENTATION**

**NAT –** Network Address Translation, each customer / home is assigned one public IP address, businesses might be issued a few. Internally, hosts / interfaces are issued private IP addresses, recall 10.0.0.0/8 – 10.255.255.255 as an example. Internal IP addresses are used for communicating among hosts in the Local Area Network (LAN). They must never be used on the public internet. When a packet is heading out of the network (to the ISP) the internal address is translated to the public IP address and port by a NAT box. NAT assumes TCP/UDP location of source and destination port fields. NAT box replaces IP source address (e.g. 10.x.y.z) with public IP address. TCP source port replaced with index of entry in NAT translation table (16 bit same as TCP port field), each entry contains original IP address (private IP) and original source port number. IP and TCP checksums are recalculated. When a packet arrives from the internet at the NAT box, it looks up the destination port from the TCP header in the translation table, retrieves original source port and source IP address, updates header and checksums and sends to the internal host.

**NAT is assumed to be NAPT in this subject (i.e. the tables save port numbers and maps a single public IP).**

**Criticisms of NAT –** Breaks end-to-end connectivity: an interface in the private network can only receive packets once it has sent packets out and created a mapping. Violates IP architectural model that states every interface on the internet has a unique IP address (thousands of interfaces connecting to the internet have 10.0.0.1). “Layering violation” by assuming nature of payloads contents – initially only worked for TCP and UDP. Must snoop on FTP messages. Changes internet from connectionless to pseudo-connection-oriented (NAT maintains connection state, if it crashes all connections are lost. Limits number of simultaneous connections since port numbers are 16 bits.

**Despite criticisms**, it is widely deployed, particularly in homes and small businesses. Carrier grade NAT: ISP only gives customers private addresses, NAT translation performed at ISP. Significant security advantage. Since packets can only be received once an outgoing connection has been created, the internal network is greatly shielded from attacks from incoming unsolicited packets. NAT should not replace firewalls. Likely to remain in use even after IPv6 solves scarcity problem.

**Subnets –** Prefixes in IP addresses indicating different destination networks can also be done within an organization to increase the flexibility of using their IP prefix. Subnetting is the splitting of a network into several parts for internal use while acting like a single network externally. Subnet masks are written same as network masks: “dotted decimal” (e.g. 255.255.255.128) or “slash” notation (e.g. /25). Example, a university with a /16 prefix could subnet its network as follows: Computer Science /17 (half of allocation), Electrical Engineering /18 (quarter allocation), Arts /19 (eighth allocation). Splits don’t need to be even but bits must be aligned to allow hosts portion to be used. When a packet arrives from the internet, router can use the subnet masks (bitwise AND) with the address from the internet to find a match and therefore which subnet it should send the packet to without knowing all the hosts on the subnet. Future changes can be made without any external impact, no need to request additional IP address allocation, routing on the internet isn’t affected either, only internal routing is. Internet becomes network of networks of networks of .. .. ...

**Fragmentation –** division of IP packets into fragments allows network gateways to meet size constraints.

**ROUTING**

Routing algorithms decide which output line an incoming packet should be transmitted on, combination of an algorithm local to each router and a protocol to gather the network information needed by the algorithm.

**Good routing algorithm properties** – Correctness (finds a valid route between all pairs of nodes), Simplicity, Robustness (a router crash should not require a ‘network’ reboot), Stability (stable algorithm reaches equilibrium and stays there), Fairness, Efficiency, Flexibility to implement policies

**Fairness vs Efficiency –** If there is enough traffic between A and A’, B and B’, C and C’, (vertically linked) to saturate the horizontal link, what is the most efficient course of action for handling traffic between X and X’? (i.e. what maximises network throughput?)

**Delay vs bandwidth –** What is being optimized? Mean packet delay or max network throughput? Simplest approach is to minimize the number of hops a packet has to make, tends to reduce per packet bandwidth and improve delay. Hopefully also reduces the distance travelled – but not guaranteed. (e.g. crossing Pacific is technically one hop). Actual algorithms give a cost to each link, more flexible but still cannot express all routing preferences.

**Routing Algorithm –** Non-adaptive **(static routing)** does not adapt to the network topology and calculated offline, reasonable when there is a clear or implicit choice of route. Adaptive – dynamic routing, adapts to changes in topology and potentially traffic levels. Optimise some property (e.g. distance, hops, estimated transit time, etc.), may get information from adjacent routers, or all routers in the network.

**Flooding –** Simplest adaptive routing, guarantees shortest distance and minimal delay, useful benchmark in terms of speed, extremely robust – if there is a path it will find it. Highly inefficient – generates many duplicate packets. Have to have a way of discarding packets (TTL) (if unknown can be network diameter). Routers that receive duplicate packets must keep track of packets it has forwarded and only forward 1 copy.

**Bellmann’s Optimality Principle –** If router J lies on the optimal path from router I to K, then the optimal path from router J to K lies on the same path. If a better route existed then it would be combined with the path from I to J to form the optimal path. (With BGP will see that doesn’t always apply).

**Sink Tree –** The optimality principle means that a set of optimal routes from all sources form a tree rooted at the destination. Shows optimal routes from all nodes to a destination root.

**Djikstra’s Algorithm –** Divide nodes into three groups: “unseen”, “open”, “closed” (**unseen:** Not a neighbor of any node we have processed, **open:** we have “visited” a neighbour, but not it. We know a path, **closed**: We have visited it. We know the best path to it.) unseen -> open -> closed. All nodes have initial value of infinity (“unseen”). Labels updated as algorithm proceeds.

**Link State Routing –** 5 step process at each router: Discover its neighbours and learn their network address, set the distance, or cost metric to each of its neighbours, construct a packet containing all it has just learned, send the packet to, and receive packets from, all other routers, finally compute the shortest path to every other router. Router send HELLO on bootup to discover neighbours. Link State packet consists of ID, sequence number, age, and a list of neighbours and their respective costs. Building packet is easy, deciding when to build is difficult (at intervals, when a change occurs e.g. link disconnect?). Reliable flooding used to send packets to other routers, (uses acknowledgements to guarantee every other router receives the packet).

**Border Gateway Protocol –** External routing protocol for routing between autonomous systems (big companies / entities / ISP collection of networks). BGP needs to consider politics (e.g. don’t send traffic through hostile countries, don’t send ISP traffic over other ISPs networks, etc.). Can’t always say which route is optimal, better in some respects and worse in others (Bellman’s optimality principle doesn’t always apply). Provider advertises routes for the entire internet, customer only advertises routes for their network to avoid transiting other traffic. “Valley free routes”.

**INTERNET CONTROL PROTOCOLS**

**Data plane vs control plane –** Protocols don’t really form a single stack, BGP (network layer) uses TCP (transport layer) for updates. A better model is to think of different “planes” of protocols. Data plane (network layer: forwarding), Control plane (network layer: choosing routes), Management plane (network layer: setting BGP policies). IP packets go through data plane, routing happens at control plane. Protocols used at the internet layer to manage functionality.

**Internet Control Message Protocol – Traceroute** – exploits the time exceeded message in ICMP, sender sends out packets to same destination with incremented TTL, counters will hit zero at successive routers, causing the router to return a Time Exceeded message, revealing the IP address of the router. Sender can use this info to determine path and timings of the route a packet will take.

**Dynamic Host Configuration Protocol –** DHCP, the internet layer requires each host / interface to have a unique IP address. Could manually configure each host (certain networks do this), difficult to administer, error prone. Slow to respond to new devices. DHCP is the automated way of handling IP address allocation. Security concerns – connecting any device will issue an IP address (but can just apply restrictions). Host sends DHCP DISCOVER packet to DHCP server for the network, routers can relay these to a DHCP server if there isn’t one directly connected to the network. DHCP server receives request and responds with a DHCP OFFER packet containing an available IP address, IP addresses are typically issued on a lease – after which time the IP address will be reclaimed by the server and re-issued (Hosts can request a renewal). DHCP used to set a number of parameters (Default gateway, DNS servers address, time servers). Since we don’t have an IP address yet, MAC Address (Media Access Control) is used and is a globally unique identifier for the interface, 48-64 bits long and hardcoded by the manufacturer. MAC Address sometimes used at host-to-network/data link layer and is sometimes called a physical address but technically isn’t used at the physical layer.

**Address Resolution Protocol –** ARP is the link between the internet layer and the underlying network layer, it translates an IP address into a MAC address. Broadcasts an Ethernet (or WiFi, or...) packet asking who owns the target IP address. Broadcasts arrives at every host on the network, the owner will respond with its MAC address. The low level sending is done via MAC addresses. This protocol is run a lot, even to find out how to communicate with the nearest router. Incredibly simple. Not particularly efficient, security nightmare (no authentication, caching of responses, even when not directly requested, ARP spoofing is the gateway attack for most man-in-the-middle attacks, provides a way of intercepting and spoofing ARP messages to associate the attackers MAC address with another hosts IP address (i.e. default gateway, DNS server, website).

Global communication depends on a network of undersea cables.

No.

TCP doesn't have the idea of "client" or "server". It treats both sides the same.

Each side chooses an initial sequence number, and labels the bytes it sends as starting from that sequence number.

In the case of an HTTP client contacting a server, the client will send its request using sequence numbers starting from what was in its SYN packet, and the server will send the response using sequence numbers starting from the (different) sequence number in its SYN packet.

A screenshot of a computer

Description automatically generated with low confidence

1. Not entirely sure about this but I don't treat congestion window as like this variable that determines how much you're sending. Instead it's something used when you need to fast retransmit or a timeout occurs (which leads to congestion avoidance).
2. For every segment acknowledge by the sender, the sender will send two more packets in return.

Say the current CWND is 5, and we just received an ACK, firstly we increment CWND by 1 to 6, so we can send one more packet from this extra 1. However, we can actually send another packet because one of the previous 5 that we sent have been acknowledged (I think about it as one spot frees up, also given that it is not a duplicate ACK), so every ACK corresponds to sending 2 new packets.

However, because we might lose packets, we set a rule that packet send over must have the following constraint:

SEQ(packet about to be send) <= last Acknowledged packet + min(CWND, RWND)

Note that last acknowledged packet should be last acknowledged number -

If we are at a situation where we have slot to send more packets but cannot due to the constraint, we wait for timeout and reset CWND to 1 (unless in fast retransmission).