**Operating Systems:** uses abstraction to hide intricate details of underlying hardware and software components - easier to manage tasks. Provides and manages orderly&controlled allocation of resources, tracks resource usage, resolves/prevents conflicts and interferances.

**Time Multiplexing:** Sharing a resource among programs or users in time intervals, taking turns. OS determines the order/duration of resource usage. Examples: CPU time allocation, printer job scheduling.

**Space Multiplexing:** divides a resource among programs or users, allows multiple programs to be in memory or use disk space concurrently. Handled by OS. Examples: divided main memory - efficient utilization of of memory resources. Disk space-multiple users’ file stored on single disk. Efficient storage and file retrieval.

**OS History:** 1stGen-Vacuum tubes (1945-55): plugboards. 2ndGen-Transitor&batch systems: punch cards writting in assembly/FORTRAN. Batch system introduced to reduce wasted time-precursor to OS. 3rdGen-Integrated circuits: multiprogramming(IBM360). Timesharing(MULTICS)-influenced UNIX+LINUX. 4thGen-Large scale integrated circuits: PC’s. 5thGen-Mobile computers

**CPU:** Fetches instructions from memory, decodes operation type/operands, executes, repeat. Has specific set of instructions, cannot execute programs from another processor. Accessing memory takes too long-use registers.

**Instruction Set:** defines the set of operations that the CPU can perform. Instructions to load a word from memory into a register and store a word from a register into memory or both. Example operations: arithmetic/logical/memory

**Registers:** small amount of storage avail. on CPU. General register: hold variables and temporary results. Special registers: program counter-holds mem.addr of next instruction to be fetched. Stack pointer: points to top of current stack. PSW(program status word): contains condition bits indicates the current status of CPU+mode bit for kernel or user mode bit.

**User mode:** restricts access to a subset of instructions and features. Cannot issue privileged instructions.

**Privileged instruction:** instructions and memory locations use that can interfere w/other processes (i.e: accessing I/O devices)

**Kernel mode:** provides full access to hardware/memory and instruction set.

**I/O devices:** interact w/OS; consists of a controller and the device(disk)(standardized for compatibility). Controller providing simplified interface to the OS and phys.ctrl of the device. Device drivers: software that allows OS to control the devices, & they are installed in kernel. Device registers use:communication between the driver & controller. 3 I/O methods: Busy waiting-driver continuosly polls device until I/O operation complete-wastes CPU resource. Interrupt: driver starts device, waits for interrupt signal (handled by interrupt handlers) when operation is finished. DMA:(direct memory access) uses dedicated chip to transfer data btwn mem and device w/o CPU intervention. CPU can temp disable interrupts to prevent interruptions during critical operations.

**I/O interrupt:** driver communicates instructions to controller for I/O operation->when controller finishes cur operation, it triggers an interrupt (tells CPU to switch to kernel) that invokes appropriate interrupt handler in OS (also provides device num causing interrupt). Runs interrupt handler specific to that device->handles interrupt->finishes I/O->blocked process is unblocked and resumes.

**System calls:** called by user program to obtain services from OS by trapping into the kernel. Puts syscall num in place where its expected(register). Execute trap instructions-switches from user to kernel to start OS. Starts execution at fixed address in kernel; kernel code examines the system call num and dispatches correct syscall handle. After completing work, control may rturn to user program. CPU generate traps for exceptional sit. I.e: division by 0/floating.pt. Underflow.

**OS abstractions**:Processes: program in execution; instance; Encapsulates all the resources/information needed to run a program: address space, registers (program counter, stack pointer), open files, related processes, etc. Address space: set of memory address a process may reference. text(low): code/instructions->program. data: stores global variables (static keyword). heap: dynamically allocated mem. Stack: local vars, tracks called funcs that hvnt returned;essential to program exe.

**Memory:** slowest med/larger capactiy to fastest/smallest: Magnetic disk-SSD-Main memory-cache(managed by hardware)-registers

**Process:** Support the ability to run multiple programs *simultaneously* on a single *CPU*

**Multiprogramming:** allows multiple pgms to share a CPU in pseudo-parallel provided by OS quick switch btwn processes. Processes created during system initialization/through system calls/user requests/batch jobs-involves exe.sys like fork(). Parent&child processes have separate address spaces, with limited or no memory sharing. Termination: voluntary-normal exit/error exit. Involuntary:fatal err(overflow/segfault)/killed by another proc. 3 Process states: running(using CPU), ready, blocked

**Threads:** entities scheduled for execution on the CPU and must execute within a process. can communicate w/each other w/o invoking kernel-shared global var&dynamic mem. Offers reduced overhead: faster thread creation, termination, switching,communication, easier resource sharing cmp to processes. Implementation: user lvl: kernel is unaware of threads, everything is implemented by run-time library at user space. V. fast, same adv as prev. If one thread blocked-entire process blocks. Kernel lvl: one process-multiple threads. No issue if one process gets blocked-OS realizes it has to give time to another. Bit more exp: trap to kernel. Lots of syscalls:expensive/less flex. Subject to scheduling policy impl.by OS

**Race conditions:** occur when concurrent processes or threads access shared data;outomce determined by order of exe. I.e: multiple processes trying to write to same file same time. Can lead to unpredictable results/diff to debug(sporadic). Avoiding rc: prohibit concurrent access to shared data. Mutual exclusion: only 1 proc. in critical region at any given time. Critical region: Part of program where shared resource/data is accessed.

**Critical region solution:** no two processes may be simul. in critical cr. no assumptions may be made abt speed or num of cpu. No process running outside cr may block other processes. No processes should wait forever to enter their cr.

**Approaches to mutual exclusion:** Disabling interrupts: may hold CPU for extended time if process is intensive. No I/O interrupt; wouldnt work for multiprocessors. Blocking, Message passing.

**Busy waiting:** when process wants to enter cr, check if entry allowed, if not: loop and continuously check. Lock variables, strict alternation (one variable can block another from entering cr even if it is outside of cr.) or TSL(test and set lock). Cons: wastes CPU, priority inversion problem, low prio may starve or high prio may be blocked by low prio process.

**Blocking:** Mutex (mutual excl): binary var representing lock/unlock state. atomic(invisible). Invoke before cr. call unlock after cr. Does not req. kernel (can be impl. in user space). Busy waiting not suitable for user-level threads: thr yield: voluntarily relinquish cpu to avoid infinite loops.

**Deadlocks:** A set of processes is deadlocked if each process in set is waiting for an event that only another process in set can cause. Four conditions: mutual exclusion: each resource is either currently assigned to exactly one process or is available. Hold-and-wait: Processes currently holding resources that were granted earlier can request new resources-to prevent: request for all resources first before execution. No-preemption: Resources previously granted cannot be forcibly taken away from a process. Forcefully take away to prevent. Circular wait: circular list of two or more processes. Each is waiting for a resource held by the next mem of chain. to prevent: only allow to request one at a time, must release before req-ing new one

**Solutions:** ignore problem. Detection and recovery. Avoid by careful resource allocation. prevention(atk one of the 4 deadlock conditions)

**When to Schedule:** process creation: depends on scheduling policy. Process exits: have to decide who uses CPU next. Process block. I/O; interrupt. Clock interrupt: preemptive vs non-premptive.

**Preemptive scheduling:** pick process, let it run for max of some fixed time(quantum). Clock interrupts needed at end of quantum to give control of CPU back to scheduler.

**Non-preemptive scheduling:** pick process to let run until it blocks or voluntarily releases CPU.

**Context switch:** expensive, involves saving and reloading process states: registers (program counter/stack ptr etc) and updating memory tables. Longer the quantum, less frequent a process runs.

**First come, first serve:** nonpreemptive, FIFO, assigns CPU to processes in order of req. Single queue, first job executes till block or completes, new/recently unblocked process joins end of queue. D.adv: processes w/long wait times.

**Shortest job first:** nonpreemptive. Assumes runtimes are all known in advance thus minimizes turnaround time. These two usually designed for batch systems, cuz no interaction with users in real-time.

**Shortest time remaining:** preemptive version of SJF. selects process with shortest time remaining run time. Compares remaining time of cur process with recently unblocked or new process’ total time. If new job req less time, cur process is suspended, new job exe’s. Allows shorter jobs that arrive later to receive preferential treatment. May starve long processes if many short ones.

**Round-robin:** preemptive. scheduler sets a quantum to generate an interrupt for each process. Process is placed at end of list after its quantum. Too short quantum: excessive switching, too long: poor response time for short interactive requests.

**Priority Scheduling:** preemptive. Runs highest prio process that is ready to run first. Prio can be assigned statically (prio assigned on external factors ie. job importance) or dynamically (adjusted by scheduler over time: decreased prio after quantum to prevent indefinite run). D.adv: starvation; lower prio jobs may never exe if more important jobs arrive

**Priority class:** processes can be grouped into prio classes, prio scheduling used among classes while using round robin scheduling within each class. D.adv: lower prio jobs may starve or receive inadequate CPU time.

**Batch environments:** nonpreemptive algos or preemptive alogs with log quantums, no interactions with user. Goal to maximize number of jobs completed per hour, minimizing time between job submission and termination, reducing num of context switching.

**Interactive environments:** users expect fast responsiveness from sys, preemption is essential to prevent one process from hogging CPU and denying service to others. Ensure processes will respond faster, meet users’ expectations of resp time. Non-premp makes process depend on q.

**Real-time environments:** Preemption is not always needed, processes are designed to run for specific periods and block quickly. Goals to meet deadlines-avoid data loss or miss time sensitive tasks. Real-time systems have strict timing requirements, hard real-time systems provide absolute guarantees(industrial/military); soft real-time systems allowing occasional deadline misses(mobiles/multimedia sys)

**Memory Management:** supports multiprogramming: ensures processes dont mess up mem of another proess or OS. Security-isolation between processes and OS. Enables compsys to run more processes with larger mem reqs i.e: processes using more mem than physically avail;Sum of mem can be larger than avail asw. Swapping processes between mem and disk may be too inefficient: too much swapping. Solution: multiplex mem(allocate a process a section in mem)

**Memory manager:** part of OS that manages part o mem hierarchy(main mem). Keeps track of which parts of mem are in use. Allocates mem to processes when they need. Deallocate mem when processes are done.

**Logical Address Space:** set of addresses that a process can use to access mem. All programs refer to logical addr, logical program addresses must be converted to phys machine addr. One approach to implementing address spaces is:

**Base and limit registers:** hardware registers that store base address and length of a process’ mem segment. Used to translate logical addresses to physical addresses w/o need for explicit address relocation during loading (programs in some systems may need relocation during loading to ensure internal mem refs match actual phys addr during load). Program is executed-Base register is loaded w/starting physical addr of the process in mem-limit register is loaded w/length of program. Allows CPU to automatically add base value to each memory ref before accessing memory (effective translating the program’s log. addr to phys. addr w/o relocation). CPU also checks if address within limit specified by limit register, fault generated+access aborted if address exceeds limit. ADV.: loading process becomes simpler and faster. Programs can be loaded into consecutive memory locations w/o address modification-improves efficiency and reduces complexity.

**Memory management:** adv: efficient use of available memory, effective allocation and deallocation of memory resources. D.adv: overhead in managing memory, potential fragmentation issues.

**Swapping:** a memory management technique where processes are brought into memory, executed, then swapped out to disk when not actively running. Adv: efficient use of limited phys memory. D.adv: slower access time, increased I/O operations, potential performance degradation due to excessive swapping. Swapping creates holes in memory-fragmentation: enough individual memory but not enough in a contiguous block. Memory compaction: combining multiple holes in memory into a single contiguous block-avoided due to high CPU time reqs.

**Bitmaps:** keeps track of memory usage. Data structure representing memory allocation units with bits to indicate free (0) or occupied (1). Adv: simple and efficient representation, small amount of memory overhead. D.Adv: Slow when searching for consecutive free units, larger bitmaps requires smaller allocation units.

**Linked list:** nodes connected by pointers, maintains a list of free and allocated memory segments. A node is either a hole or a process, contianing information such as start address, length and pointer to next. Adv: simple management, straightforward updates when process terminates or swapped. D.Adv: slow search time as list grows, potential fragmentation.

Memory allocation: First fit (fast but may generate largers holes); Next fit; Best fit (minimizes wasted memory but slower due to entire list search,creates fragments);worst fit(largest hole to prevent fragments but inefficient use of mem);quick fit (one list for holes, one for processes; fast allocation but potentially slower deallocation, merging of neighbr holes, increased complexity)

**Virtual Memory:** solution to inefficiency of swapping. System that allows programs larger than available memory to run by dividing their address space into pages and mapping them to physical memory. Uses paging:

Pages: contiguous ranges of addresses mapped onto physical memory. Not all pages need to be in phys memory at same time to run program. If yes: MMU performs necessary mapping from virtual to phys addr. If not, OS gets missing page and re-executes instructions (expensive due to disk access).

**Paging Operation:** Page table: index table for v.mem to phys mem. Also stores information about page attributes: present/absent bit to indicate whether a page is in mem or not; Protection bits control permission (read/write/execute); modified+refernce: track page usage and help with page replacement. Requires extra memory (increased mem overhead) but efficient (caching for specific pages).

Conversion: index x 4096 + given virtual addr.

**Speeding up paging:** major issues: 1. Mapping to virtual addr to phys addr must be fast (solution: TLB) 2. Large virtual address space = large page table (affects overall system performance)

**Translation Lookaside buffer (TLB):** page table cache (recently accessd page table entries), in cpu or even mmu→really fast. holds information about a page, including virtual page number, modified bit, protection code, and physical page frame. Allows search via searching all keys to find match->faster lookups. Small in size, dont want to be large to prevent long lookup times.

**Page fault handling:** page fault when table maps to nothing in phys mem = raises interrupt. Interrupt suspends cur process-jumps to pagefault handler in kernel: checks page ref legality (permission), frees up page frame (if none avail), loads required virtual page from swap space into free page frame (if page has been modified, have to rewrite back to disk, if not, new page can overwrite)(OS can block cur processs to give time to another process via context switch, one I/O is done (page is in memory), page that triggered fault can then go into ready state), updates page table, restarts process at same instruction.

**Optimal Algorithm:** remove pages that have instructions that will be used the furthest away to make space for one that is closer in steps. Not optimal: need to know how & when page will be accessed.

**Not recently used algorithm:** status bits: R(referenced), M(modified). R bit is periodically cleared, pages are categorized into four classes based on R/M bits. Class: 0-0R/0M, 1-0R/1M, 2-1R/0M, 3-1R/1M. NRU selects random page from lowest numbered non-empty class, prioritizes removing pages that havent recently been referenced. Not optimal performance but straightward and reasonably efficient.

**Second-Chance algorithm:** based off FIFO but addresses issue of evicting heavily used pages. Operates by inspecting R bit of oldest page in mem, if R bit=0→immediately replaced. if R=1, its given a “second-chance”: R bit cleared, page moved to end of pages list, updates load time. Repeats until suitable evictable page found or acts as FIFO till found.

**Least recently used algorithm:** relies on observation, evicts page unused for longest time. Hardware implementation: global instruction counter increment w/each instruction cpu uses, evict page w/lowest value-expensive and time-consuming. Software simulation (aging): keep bit counter for each page, on each clock tick, shift counter 1 bit to the right. If page is referenced, add 1 to leftmost bit. Cannot distinguish between references early&late in clock interval and has finite past horizon due to the limited number of counter bits.

**Locality of Reference;** Temporal locality: pages heavily used recently will probably be heavily used soon. Spatial Locality: accessing a page is likely to lead to accesses to nearby pages. Working set: pages that a process is actively using. Maintaining working set of a process in memory reduces the number of page faults → more efficient

**Calculating page:** page index\*4096 + offset (offset is how many off from multiplication of 4096)

**Cryptography:** takes a msg (plaintext), encrypts into ciphertext w/key in a way that only authorized people know how to decrypt (w/decryption key) into plaintext. Supports secure communication by enabling: Confidentiality:only sender and intended receiver should see msg, authentication and verification, Integrity. Is susceptible to brute force attacks, but challenge is to make brute force take an infeasible amount of time w/in useful lifetime of data. Dependent on randomness.

**Symmetric Cryptography**: same key is used for encryption and decryption, needs secure way to exchange secret key. I.e: **AES** (advanced encryption standard): breaks data into blocks and encrypts each block (block cipher), diff modes of operation.

**ECB (Electronic Codebook)**: simple, parallelisable (can execute in parallel). Breaks into fixed size blocks, encryption algo cant take arbritray params - useful for large amount of data. Encrypt each block with algo and key, same w/decrypt algo and key. Problem: Repeated patterns will be evident in output-> lower fidelity but easy to detect

**CBC (Cipher Block Chaining)**: Initialization Vector (IV)-not secret, but must be random and not reused. If its not random or reused, its deterministic. Encryption must be done sequentially-Plaintext XORed with IV, then produced ciphertext is XORed with next block, repeat. Decryption can be done in parallel, ciphertext are alr available. Loss or corruption of a ciphertext block or IV affects correct decryption of at most one block. Adds additional layer of security, prevents patters. (XOR compares two input bits and generates one output bit)

**Asymmetric Cryptography (Public Key Crypto):** two different keys, one encrypt, one decrypt. Used in TLS, digital signatures, secure messaging, end-to-end encryption. Cannot do both w/one key, has to be one encrypt, and one decrypt.

Process example: A generates key pair (public key, private key), public key is online, B can send A secret message by encrypting with public key and then sends ciphertext to A. A can decrypt with prv key and receives secret msg.

**Asymmetric vs Symmetric:** A is much slower than S - thus not suitable for encrypting large amnts of data. Often used with symmetric as way of exchanging a joint secret key (SK).

Process example: A generates key pair, pb key is online, B generates SK with symmetric encryption. B encrypts SK with A’s pb key, A receives SK by decrypting with A’s prv key. A and B now share SK and can exchange symmetrically encrypted msgs.

**Digital Signatures:** provides link between key holder and a msg/doc/file. Cannot deny having signed the doc (non-repudiation). Provides integrity and against tampering. No confidentiality but is for verification of integrity.

Examples: B encrypts msg using prv key. To verify signature, A decrypts with B’s pb key. Compares decrypted msg to orig msg. If they match (not confidentiality, just integrity so have access to orig msg): msg must have been encrypted w/B’s prv key. Only B has access to his prv key thus non-repudiation.

Large documents: use cryptographic hash function. Takes in any length outputs fixed length. Sign the hash digest instead, use public key to verify. Hash function is a lossy compression function: must have certain properties. Hashing: additional security, turn to fixed length-cheaper.

Key size: determines degree of security, expressed in bits.

**Secure communication:** The objective is to provide secure private communication between two end-points, with integrity checks to ensure data does not change in transit, and authentication to establish the identities of one or both of the end-points

Advesary controls: Wi-Fi, DNS, routers, can create its own websites, can listen to any packet, modify packets in transit, inject its own packets into the network

**Tampering:** attacker can reorder ciphertext, flip bits. Every possible cipher text corresponds to some valid plaintext

**Message Authentication Code (MAC):** provides integrity w/o encryption. Detects if message has been tampered with. A and B use a shared secret authentication key (s). A creates message (m), concatenates m with s = m+s. Calculate the hash H(m+s) using hash function - hash is called MAC. A appends MAC to m, to create extended msg(tag) (m, H(m+s)) to send to B. B received extended msg, calculates MAC using shared secret key, compares it with the received MAC. If match, msg has not been tampered. Suitable when confidentiality is not needed.

**Authenticated Encryption:** confidentiality and integrity of msgs exchanged btwn A and B. Encrypt then MAC. Encrypt the msg to produce ciphertext, use the ciphertext along with an authentication key to generate MAC, attach MAC with ciphertext to send. Only decrypt when MAC is a match. (A knows cipher text has not been modified)

**Diffie-Hellman Key Exchange:** cryptographic algorithm that enables 2 parties to establish a shared secret key over an insecure channel without prior communication or need for a shared secret key. Both A and B independently generate their own pb and prv key. Both share their pb key with each other. Using this, both A and B perform mathematical computations to derive a shared secret key (s). s can be used for symmetric encryption to secure future communication. Computationally infeasible for eavesdropper to derive secret key w/o prv keys.

**Man-in-the-Middle (MITM) attack:** cyber attack where an attacker secretly intercepts and potentially alters communication between two parties who believe they are directly communicating with each other. DH is secure against MITM because its computationally hard.

**Digital Certificates:** securely associate identities with cryptographic public keys. Certificate often signed (certificate signature) by third party.

Chain of trust/Certificate Hierarchies: chain of trust where trust in one CA extends to all certificates it signs. Example: CertA (self signed), Cert.B (signed by A), Cert.C (signed by B). Alice receives Cert.C but only trusts A. Alice verifies (B’s pb key, Signature in certC, Cert.C) = B did sign C if verification is match. Then Verify(A’s pb key, Signature in cert.B, Cert.B).

**Certification Authority (CA):** binds a pb key to a specific entity. Root certificates. Includes verifying the identity of the entity, can be person, router, domain etc. Relies on CA’s credibility for accurate verification. Once CA verifies identity, it creats certificate that associates the entities pb key with its identity. Certificate contains pb key, unique identifying info of owner i.e IP addr, and digitally signed by CA, can have more info. Root certificate is self-signed.

**Transport Layer Security (TLS):** Protocol for secure communication over Internet Transport Layer Security protocol. HTTPS: implementation of HTTP over TLS. Basics: 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

**Handshake Protocol:** Uses public-key cryptography to establish several shared secret keys between the client (i.e. web browser) and the server (i.e: website). An initial negotiation between client and server that establishes the parameters of their subsequent interactions within TLS 1. Negotiate version and set of cryptographic algos to be used. 2. Authenticate: server and client (optional) using digital certificates 3. Establish/exchange shared secret key.

\*Example: Client (C) sends ClientHello to server asking for secure connection, listing its supported cipher suites. C assumes server will support a given key exchange protocol (i.e: DH), sends public component of key. Server (S) responds with ServerHello, selects one of the cipher suites that it supports, sends its corresponding pb key. S sends cert msg containing its pb key cert and other supporting cert in the cert chain. S sends Certificate CertificateVerify message containing a signature over the entire handshake using the prv key corresponding to pb key in the Certificate message. S sends Finished message consisting of a MAC over the entire handshake. C validates: certificate, signature, and MAC. C sends a Finished message consisting of the MAC of the entire handshake. Server validates MAC. Handshake concludes, both parties can use symmetric cryptography to communicate securely

**Internet:** composed of the aggregation of many smaller networks. 3 phases: ARPANET (1960’s-early 1970’s), NSFNET (1970’s - early 1980’s), Internet (1980’s - present). Brief history: Opposing views- one side ARPANet: designed so universities could share expensive supercomputers, no thoughts on scale. Other side-OSI: wanted a data version of the telephone system, a global standard for data networks, wanted interoperability between equipment vendors via rigid, well tested standards. Slow process.

Outcome: Two protocol stacks: TCP/IP-effectively standardised post implementation. OSI-standardised pre-implementation, but not widely implemented.

**Network protocols:** modelled as stack of layers to provide structure and facilitate design. Each protocol belongs to a specific layer, conducts inter-layer exchange. Layers offer services to the layer above by performing actions within the layer and utilizing the services of the layer directly below. Peer entity: resides on the same layer. Layering enables large and complex systems to be updated efficiently by facilitating independent changes in specific layers w/o disrupting entire system.

**Services:** set of primitives that a layer provides to a layer above it. If service within a layer changes, it only impacts the layer above it.

**Protocol:** set of rules and conventions that govern the communication/msg exchange and the actions taken based on these msgs. It specifies the format and order of the msgs exchanged. All communication btwn remote entities is governed by protocols.

**TCP (transmission control protocol)/IP(internet protocol)**: was designed to be independent of data link and physical layers. Only physical, link, network, transport, application.

**OSI (Open Systems Interconnection) model:** conceptual framework for network design focus on protocols, layer created when different abstraction needed, performs well-defined function, layer boundaries should be chosen to minimize info flow between interfaces. Layers bottom up: Physical: transmits individual bits over physical mediums i.e copper wire, fiber optics. Diff phys layer protocols are used depending on specific transmission medium used by link. Link: Takes string of bits, works out next step, tidys up p2p, network (get data end2end), transport (tidy up e2d), session (responsible for managing communication btwn applications), presentation (handles interpretation, compression, & encryption of data exchanged btwn applications) and application (encompases protocols & services that directly interact with end-user applications, use data) layer.

**Encapsulation**: involves adding protocol-specific header information at each layer of the protocol stack. At each layer, a packet consists of header fields (containing protocol-specific information) and a payload field (carrying the packet from the layer above).

**IP “narrow waist”:** hourglass; many application protocols, one network protocol (IP), many link layer protocols. Part of network architecture. Don’t need to write different application layer versions for different physical layers.

**International Standards for Network Protocols:** Pros: standardized communication, economies of scale. Cons: Difficult to change, potential to be obsolete after accepted

**HTTP:** hypertext transfer protocol. Application layer protocol at the core of the Web. Enables standardized communication between web browsers (client) and web servers. Defines the structure of messages exchanged btwn them and how the msgs are transmitted. Ensures reliable data transfer over TCP. Stateless protocol: server doesnt store client-specific info.

**URL/URI:** Uniform resource locator, an address for a resource. URL: protocol + DNS name + file name

**Non-persistent connections:** involves establishing a separate TCP for each request/response pair. Adv. Simplicity-each req is independent thus its easier to implement on both client and server sides Flexibility-cliets can make requests intermittently w/o need to maintain cont. connection. D.Adv: Overhead-Creating and destroying TCP connections for each req adds overhead in time and resources. Latency-connection establishment and termination take time impact overall response time. Limited pipelining-Pipelining (which allows sending multiple requests without waiting for the corresponding responses), is not efficient with non-persistent connections due to connection overhead. Connections: HTTP 1.0

Protocol Overview: 1. Client initiates TCP connection (creates socket) to server (port 80-default port for HTTP). 2. C sends HTTP req msg to S via TCP connection, specifying URL of page HTML file wanted. 3. S receives req through its socket, fetches requested object, encapsulates object in HTTP response msg and sends to C. 4. S instructs TCP to close connection after sending resp.msg. TCP waits until it receives acknowledgement from C to ensure response was received. 5. C receives resp.msg and TCP connection terminates. C extracts file from encapsulated msg.

**Persistent connections:** all requests and their corresponding responses sent over same TCP connection (it remains open). Adv. reduces overhead & latency by reusing same connection for multiple requests. Enhanced pipelining. Default use in HTTP for improved efficiency and performance but can be configured to use non-persistent depending on server load, network conditions and performance reqs. Connection: HTTP 1.1

**HTTP/2**: evolution of HTTP 1.1, compatible with HTTP 1.1 at high level. Aims to decrease latency-compress HTTP headers, server push, multiplexing reqs over TCP connections.

**HTTP/3:** decrease latency aim. Runs over QUIC not TCP: more parallelism esp w/packet loss.

**HTTP request:** consists of request line (3 fields: method field-GET,POST,HEAD, URL field-identifies requested obj, HTTP version field) and header lines (provides additional info i.e: host, user agent & content prefs).

**HTTP response:** consists of initial status line (protocol ver. Status code, status msg), header lines, entity bdy. Codes: 1xx=information, 2xx=success, 3xx=redirection, 4xx=client err, 5xx, server err.

**\*Wireshark:** network protocol analyzer. Used to capture and analyze HTTP requests and response msgs exhcnaged btwn browser and web server. By capturing and inspecting packets, can identify issues and errors in communication + insights into network performance.

**DNS**: Domain Name System, provides a directory service for mapping domain names (host.domain.com) to network addresses (IP address). Browser asks DNS for IP address of the Server - resolving URL. Domain names: Not case sensitive, up to 63 char btwn dots, 255 char total. Bad security, spoofing (imitate server), flooding (DDoS=overwhelming server w/hella queries => unresponsive). Solutions: DNSSEC (Domain Name System Security Extensions).

DNS 4 elements: Domain name space: tree-structured name space to identify resources. DNS Database: node/leaf in name space tree names a set of info contained in resource record (RR). Name servers: server programs that hold info about portion of domain name tree structure and assoc. RR. Resolves: queryers, extract info from name servers in response to client reqs.

**Name server zones:** DNS namespace divided into overlapping zones. Zones allow decentralized administration of DNS records, diff orgs and entities to manage their own zone. Arranged in hierarchical structure extending from set of root servers. Helps distribute load of DNS queries. Root name servers: form authoritative cluster for DNS enquiries, acts as starting point for resolving domain names (maintain top-level domain (TLD) info i.e .com, .org and .countrycode, 13 in total).

Authoritative DNS servers: organizations DNS servers, providing authoritative hostname to IP mappings for organizations servers. Maintained by organization itself or service provider.

Local DNS server: specifies local host i.e computer itself, and also blocks ads by mapping ad-serving domain names to a non-existent or local IP address

**Query:** finding the IP address associated with a given hostname. Resolver client asks the local DNS for the domain to IP mapping: if ans is known->sends answer, not known->DNS queries up hierarchy to top lvl (root) DNS for the domain, then relays the answer to the client. Subject to timers to avoid long wait time.

**HOSTS file:** before DNS. Text files used for local DNS lookups on a computer, mapping hostnames to IP address.

**RPC:** Remote Procedure Call. Allow calling procedures on a remote server as if its local to client. Hides networking aspects from programmer. How: Client on machine A calles procedure on machine B. process on A is suspended, execution of procedure takes place on B. B responds with result to A, A continues processing. Client and server bound to stub, operate in their own addr space, cannot use ptrs. Mashalling: data strict to form that can be stored/transmitted. Unmarshalling: opposite. Challenges: cant pass ptrs, global variables arent shared, problems from weakly typed langs like C, unable to deduce param types

**User agents:** UA/MUA. Run on end user’s device, provides interface for users to send, receive, manage emails. Functions: compose, report, display, dispose. Encapsulation of transport related info. Header (actually sent to), body (who we say we’re sending to).  Message transfer agents: MTA’s. Transports msgs from source to destination.

**Mail Message Format:** Header (To, from, subject), blank line, Body (message, ASCII)

**SMTP:** (simple message transfer protocol). Standard protocol, uses TCP, defaut port 25. Direct transfer: sending server to receiving server. Three phases: handshake-greeting, transfer of msg, closure. Must be 7-bit ASCII.

Interaction example: Sender mail serv (S) introduces with HELO cmd, recipient mail serv responds(R). S specifies sender’s email addr w/MAIL FROM, R acknowledges it. S specifies recipients email addr w/ RCPT TO, R acks. S initiates data transfer with DATA, R indicates ready to recv. S sends mail content to R. S indicates end with “.”, R confirms msg accepted for delivery. S terminates connection w/QUIT. R acks. D.adv. Slow on modern networks where latency is larger than serialization delay.

**MIME:** multipurpose internet mail extension. Encodes to 7-bit ASCII. Sends in header to let recver knows how to decode.

**POP3:** post office protocol. Simple mail access protocol where user’s email client connects to mail server, downloads msgs to client device for offline access, deletes from server. Good storage control, but no re-read, inconvenient on multiple devices

**IMAP:** internet message access protocol. More advanced mail access protocol, allows user’s mail client to access and manage msgs directly on mail server, retains mailbox contents online. More complex, reliant on network.

**TCP:** transmission control protocol provides reliable e2e stream of bytes w/o worry of problems from network layer: loss packets, out of order, duplication, corruption. Is continuous/stream oriented, reliable, in order). Operates on top of IP. TCP entity accepts data as a continuous stream, divides it into chunks/segments < 64Kb (often 1460 bytes to fit IP/TCP header), then sends each piece as a separate packet over the network. On the receiving end, recipient TCP entities receive packets and reconstruct original byte streams in the correct order. Segments have 20-60 byte header and size is determined by TCP entities w.r.t two constraints: IP Payload < 65,515 byte (mentioned above). and Maximum Transfer Unit (MTU): maximum packet size transmittable without fragmentation (1500 bytes).

**IP datagram:** data stream segmented pieces <64kb (often 1460 bytes to fit IP/TCP headers)

**TCP primitives:** core functions: LISTEN (block until smth tries to connect), CONNECT (packet: connection req, actively atmpt to establish connection), SEND (packet: data, send information), RECEIVE (block until DATA pckt arrives), DISCONNECT (side wants to release connection).

**TCP service activation:** connections must be explicitly established at sending host (src-host, src-port), and at receiving host (dest-host, dest-port). Sockets: a kernel data structure, named by 5 tuple of IP addr, port number of sender and receiver, and protocol

**TCP connection features:** Full duplex: allows data to be transmitted simultaneously in both directions of connections. End-to-end: reliable&ordered connection established between communicating hosts. Byte streams: not msg streams Buffer capable: TCP entity can choose to buffer pre-send or not (send evrythng at once instead of pckt by pckt, reduces overhead, increases delay)

**TCP header:** Source port, destination port, sequence number (for sliding window), acknowledge number, data offset, flags (SYN, ACK, RST, FIN. SYN initializes TCP connection. ACK acknowledges received packets), window size (how much data sender of segment is willing to recv).

**Connection establishment issues:** because TCP is connection-oriented over connectionless network layer (IP), networks can lose, store and duplicate packets. Congested networks may delay acknowledgments, incurring repeated multiple transmissions, any of which may not arrive at all or out of sequence – delayed duplicates

**Three-way handshake:** establishes one and only one reliable connection. A solution to avoid problems that occur when both sides allocate same sequence numbers by accident. Exchanges 3 packets SYN. SYN-ACK, ACK.

Process: 1) SYN: (SEQ=x) client (init host) sends TCP segment with SYN flag set to server (recv host). Used to init connectiion req, includes init seq num (ISN) chosen by client. 2) SYN-ACK: (SEQ = y, ACK = x+1) upon receiving SYN, server acknowledges request by sending SYN-ACK packet back to client. Packet contains server’s ISN and acknowledgement of clients ISN+1. Signals host2’s willingness to establish connection and acknowledges client’s request. 3) ACK (SEQ = x+1, ACK = y+1): after receiving SYN-ACK packet from server, client sends ACK packet back, acknowledging servers ISN+1. Signals both sides successfully agreed on ISN and ready to exchange data.

**SYN:** synchronisation during and used for connection establishment. Sending of SYN or FIN causes seq num to increment by 1. Sequence number:  counts bytes not packets, first byte of this segment’s payload (offset by random num, for security offset reflected in both syn and ack numbers). Acknowledgement number: next byte the sender expects to receive (bytes rcved w/o gaps - a missing segment will stop this incrementing, even if later segments have been recvd)

**TCP retransmission (Error recovery):** timeout value for retransmitting segment. initially set2 default value+updated based on network performance. If RTO expires before receiving ACK->segment’s resent. RTO timer resets when receiving new ACK. Receiver sends duplicate ACKs (DupACKs) for out-of-order/missing segments. After receiving 3 DupACKs/4 ACKs>sender lost segment, resendt: fast retransmission: don’t wait for RTO.

**TCP Closing:** **RST** (reset) flag = “hard” close connection: states sender’s closing connection and won’t listen for any further messages. Used for “u got wrong sender”, reset it. **FIN**: request to close connection, directional (after ack->new data can’t be sent one way)

**SYN Flooding:** atk sends initial SYN request but not send appropriate ACK, causes sever to fill up its queue w/seq numbs for defunct connections. Solution: SYN cookies-derive seq numb from connection info and timer that creates stateless SYN q from hashing (high performance cost)

**Window Size**: size of receive window: how much data sender of segment is willing to receive

TCP Sockets, Sliding Window, QUIC, Flow/Congestion Control, UDP

**Socket:** data structure in kernel, named by 5 tuple of IP addr of src and dest, port number of src and dest (exists only for naming), and protocol. Used as endpts for communication between sender and receiver. Use of unique port numbers allows distinguishing between multiple connections to the same server.

**Socket Primitives:** SOCKET (create new comm endpt), BIND (asso. localAddrr w/socket), LISTEN (announce willingness2accept conn + give queue size), ACCEPT (passively (no pcketss sent) establish incoming conn (block until then)), CONNECT (actively attempt2establish conn), SEND (send data over conn), RECEIVE, CLOSE

**Multi-threaded server:** allows server to handle multiple client connections simultaneously. Server creates separate thread for each client connection. Dispatcher thread on continuous loop to listen for incoming client cononectionson the bound socket. Server accepts new connection request and creates/wakes a thread to handle the clint. New thread can read from and write to socket independently w/o affecting other threads. - efficient utilization of servers resources

**Sliding window:** to manage data amnt between sender+receiver. Sliding window determines max data amnt receiver is able to accept (**WIN=** window size avail.**)**(hence controlled by receiver). Both S+R maintains buffers to store data independently of application, **BUT** no guarantee data’s immediately sent/read; S may delay sending data. When R’s window size=0 -> S shouldn’t send data. However, can still send packets w/ACKS or urgent data (data not in same stream as TCP).

**Deadlock**: sender wont send data cuz window size = 0, rcver wont recv anything so wont send any ACKS to increase window size. To prevent deadlock, S can also send zero-window probe segments after persist timer runs out: 0-byte segs requesting an updated window size and next expected byte->ensure data transmission can resume when R’s window size = 0. another way: solve via sending windowUpdate after window has moved from new data read. Not duplicate ack because window size has changed.

Send window: data amt S curr able2send (includes both unacknowledged seg + unsent data fitting within receiver’s window). Receive window: data amt R willing2receive. Specified in ACK sent by R.

**QUIC (Quick UPD Internet Connections)**: reliable transport protocol originally designed for running HTTP. But standardized 2021 and operates over UPD instead of IP - recognized by firewalls. Multiple byte streams instead of one (TCP), sends multiple streams, in order within stream but not between streams. Prevents delay from loss recovery.

**QUIC BENEFITS:** 1) support multiple byte streams instead of single. Hence if packet’s lost in one stream, doesn’t impact others, resulting in faster recovery and more efficient transmission. 2) enables faster establishment of secure connections by combining TLS handshake w TCP handshake, reducing # of idle round-trip times (RTT). 3) ability 2 reconnect to different IP in middle of flow. Useful when switching network conn (i.e transition from wifi 2 4G cellular/Ethernet).

**QUIC SOCKETS:** NOT named by 5-tuple. Categorized into 3 types:

1) Listener sockets: similar2 regular TCP/UDP listeners. Bind 2 port + accept incoming connections. Closed when app no longer wants 2 accept new conn.

2) Connection sockets: created by connect/accept calls. Unlike UPD sockets, they can’t be used 2 send/receive data. Only can manage streams belong 2 conn.

3) Stream sockets: send/receive fcns so peers can exchange data. Created by either accepting on conn socket or call connect on conn sock to create new, locally initiated stream. Can be closed at any pt during conn.

**CONGESTION CONTROL**

**Flow Control:**

1) Go-back-N: when packet’s lost, retransmit all packets from pt of loss onward. Pro: R don’t need2 store then reorder packets. Con: if only few packets are lost, leads to inefficient retransmissions.

2) Selective Repeat: only lost packet’s retrans, put packet in right spot then send off. Correctly recv packets ackl. Pro: more efficient if packet loss common. Con: more complex: requires R2 store out-of-order packets and reorder them before delivering to app.

**Congestion Collapse:** late 1980s, internet took long time for packets to be transmitted w/in short distances. Router buffers overflowing bc of go-back-N, affects all layer. Solution: selective repeat.

**CONGESTION CONTROL WINDOW (CWND):** add. window dynamically adjusted by sender, based on network performance, to aid efficient data transfer. Allows S to adapt its sending rate to network’s congestion lvl. Unlike sliding window controlled by R, CWND controlled by S. Pro: enables congestion control w/o requiring packet format changes, or adding more fields. Crucial for backwards compatibility + widespread adoption of TCP protocol. (S knows whats send & whats ack’ed)

I**NCREMENTAL CONGESTION CONTROL:** CWND starts with Max Seg Size (MSS). S transmits 1 seg. For each seg ack’d, CWND += MSS. With each full window of ackl, window size doubles until timeout or reaching slow start threshold (SStresh). Slow-start algo gradually increases windowsize exponentially. Once SStresh reached, growth slows to linear (additive increase). **On segment loss**, SSthresh set to CWND/2->triggers slow start again. Once reached SSthresh reached again, growth slows into linear

**CONGESTION CONTROL OPTIMIZATION:** TCP RENO introduces fast recovery: start from new SSthresh value to avoid slow start. Needs unrealistically small packet loss **SACK** (Selective Acknowledgements)allows receivers to selectively ack individual packets, reducing unnecessary retransmissions w/up to 3 byte range specified - improves network efficiency.

**Macrospcopic model:** packet-lvl rules affect fairness between flows, response to long round-trip times (RTTs) and response to random pckt loss.

**WINDOW SIZE**: W increases once per window. Each packet arriving increases W by 1/W. When loss occurs (w prob p), W is halved. Hence avg increase in window size=(1-p)/W-p(W/2)

**Equilibrium:** avg increase must be 0; i.e W = sqrt(2/p)

Rate which packets are sent over network = W/T = 1/T \* sqrt(2/p) , W windowsize, T round-trip time (RTT), 1/T maximum rate, packet loss rate (p). Clearly if p fixed => longer RTTs get less rate (RTT unfairness). Clearly if RTT small->TCP forces packet loss rate to be high.

**Wk 10: Presentation/Session/Network Layers, Packet Switching, IP Addressing**

**PRESENTATION LAYER (6):** provides encryption, compression, data conversion, mapping btwn charSet. Typically implemented by apps: i.e not a separate layer. IETF considers these services as part of Application Layer; encryption protocol is relatively simple and separate from algos used. Tasks below Transport Layer handled by kernel, while App Layer tasks are in user layer. RTP similar.

**SESSION LAYER (5):** provides authentication (verifies user identity), authorization (checks authority), session restoration (resumes failed downloads/logins). Ex: QUIC, RPC, PPTP (point to point tunneling protocol), PAP/EAP (password/extensible authentication protocol) - last used in network and link layers.

**TRANSPORT LAYER (4):** layer 4, facilitates logical communication between aplication processes running on diff hosts, provide services needed by applications using services avail. by network layer, regardless of phys structure and network routers. Transport provides to application: data doesn’t arrive faster than can handle, data from one application is not mixed with data from another. (UDP doesn’t provide:) data is stream of bytes, data arrives reliably (or we know when packet has been lost), data arrives in order). What network layer provides to transport:get packets from host to host (most of the time, sometimes multiple copies). Encapsulation of segments (transport layer units) in packets (network/internet layer units) in frames (data/link layer units)

**Connection-oriented transport services**: can provide a reliable service on top of an unreliable network. Provides reliable connection orientated service: hides acknowledgements, congestion control, lost packets. Doesn’t provide privacy(needs TLS), or isochrony (spacing in time between packets at sender is same as revcrs) needs RTP.

**Unreliable connectionless service:** UDP. provides only multiplexing between diff processes

**Addressing**: specification of who is that its talking to is required at both application and transport layers. Typically done using port numbers (i.e: port 80 for web servers). Sometimes spawns a server to handle incoming connection requests based on port number. No need to have server always running and accept()ing for rare services. Full address is 5-tuple. TCP listen sockets and most UDP sockets only have 3 tuple: local IP, local port, protocol

**Port Allocation**: port numbers are 16 bits (0-65535). Well Known Ports (0-1023), Registered/User Ports (1024-49151), Dynamic Ports for clients (49152-65535) – used for making connection on well-known port

**Multiplexing/MUXING:** combining multiple distinct IP packet streams into single shared data stream. **Demultiplexing/DEMUXING:** distinct streams extracted from single.

**UDP–User Datagram Protocol:** connectionless transport layer protocol for transmitting encapsulated IP datagrams, done by users on application layer. Transmits segments consisting of header+payload. UDP header contains src+dest ports -> PROS: enable muxing+demuxing data, no root/admin priviledge unlike IP, reduced overhead – no windows.

Still requires both source and dest ports because there will be replies.

CONS: lacks flow/error/congestion control, retransmission. Practical Usage: use UDP for apps favoring precise control over packet flow + error handling not needed, good for short requests/responses (DNS), real-time services like VoIP.

**UPD Header**: Source port, destination port, UDP length, UDP checkum: error detection in UDP packet. Calculated over entire pkt, including pseudoheader (contains src/dest IPaddrr)

**RTP–Real-Time Transport Protocol:** used2 stream media and VoIP apps. Multiplexes multiple streams into single UDP stream. Layer’s debatable; **IETF POV**: appLayer (runs userSpace, use UDP from transport layer). OSI POV: neither transport nor app layer (pres). **Header**: payloadType: which encoding (MP3, etc), Seq#: counter increm on each pkt, Timestamp (time of pkt relative to start of stream).

**RTCP–Real-Time Transport Control Protocol):** control protocol for RTP. handles fdbck, synch, user-interface in realTime apps. Provides feedBack to data src: for delay, jitter (variation in pkt delay), bandwidth, congestion. Achieves synch among diff streams w varying clocks/drift. Facilitates user interface by naming/identifying sources in conference call.

**NETWORK/INTERNET LAYER (3)**: responsible for getting data from src to dest with the right path through network: may involve multiple physical connections, multi-hop paths. Most code runs on routers, which performers traffic routing. Nodes are given addresses (names) to help communication. Protocol data units are called packets. Provides two services:

**1) Connectionless**: each pkt sent indep 2 dest using IP pkt switching. Routing table simply has a dest/line columns for each node. “datagram” network

**2) Connection-oriented**: establish virtual circuit before data transfer. Used in ATM/MPLS. Routing table now has In/Out forwarding table with virtual circuit number local to pair.

**MPLS (MultiProtocol Label Switching):** widely deployed VC, conn-oriented network layer protocol. Provides QoS for prioritizing traffic. Enables Service Level Agreements (SLAs) for guaranteed ntwk performance. Popular for businesses connecting multiple sites + carrying voice traffic. Con: $$$ more than standard internet connection

**Store-and-Forward Packet Switching:** Recall TCP+above, didn’t care how data sent. But at network layer, **store-and-forward process:** each pkt sent a hop at a time. Transmit pkt to nearest router, pkt is buffered while arriving, checksum verified. Pkt received, validated (chk for errors), stored in buffers (queue) at each router along path before forwarded to next rtr. Introduces delay due to buffering/validation. Used in SMTP (msgs stored/forwarded on msgBymsg basis rather than pktBypkt).

**Alternative: cut-through-routing**: allows pkt forwarded ASA necess hdr info examined, w/o waiting for entire pkt to arrive.f

**Pro/Con Connection Type.**

**ConnLess**: state: (+) routers don’t hold state info about conn->reduces mem/proc req, addressing: (-) each pkt includes full src/dest->increase overhead esp for IPv6(16bytes), routing:each pkt indep defined at setup, QoS: (-) Diff, CongCtrl: (-) Diff, Link failure recover: (+) Simple since no need2 re-establish conn

**ConnOrr:** state: (-) each VC req routr table space, rtr reboots->pblm, addrsing: + each pkt has short VC#->reduce overhead, routing:each pkt indep defined at setup, QoS: (+) easy if enough rsrcs, CongsetCtrl: (+)... since state info’s available for each VC, can deny req to setup circuit, Link failure recover: (-) extra work cos established conn need to be re-established or rerouted

**Quality of Service (QoS):** not all services have same priority/tolerance for network delay. Allows prioritization within network. use Differentiated Services header field in IP. Useful for office buildings with internet/telephony traffic.

**INTERNET PROTOCOL:** Responsible for moving packets through the various networks from source to dest host. Multiple paths through the network (imprtnt for redundancy), routing algos used to determine best path. Designed for best effort, not guaranteed performance. Smth that works ok, is simple, strict when sending, tolerant when recving, avoids static optns and params – negotiate at runtime

**IPv4**

**HEADER FIELDS**: **version** (protocol v4), **IHL** (headerLen in **32 bit** words: min5, max15), **DifferentiatedServices** (QoS sService class), **ECN** (explicit congestion notification), Total Length (including payload, max 65,535), **Identification+DF+MF+FragOffset** (Used in frag handling), Time to Live (TTL) (hop countdown -1: at 0, pkt is dropped to prevent pkt circling indefinitely in network), Protocol (tells us payload, i.e is it tcp/udp pkt etc), src/dest addr (IPv4 addr), options (rarely used, poorly supported), checksum (check if pkt err)

**Addresses**: 32bit(0.0.0.0–255.255.255.255). Assigned to interfaces, NOT hosts. Address supply has been exhausted. Types: unicast (1-1), broadcast (1-all), multicast (1-many), geocast (1-all)

**Classes (A,B,C,D):** each class had fixed portion of addr dedicated to network, remaining for host addr. Pro:simplified routing. Con: inefficient address allocation: networks having more addr than needed. Sln: CIDR

**CIDR (Classless InterDomain Routing):** flexible IPAddr alloc. Allows explicit spec of network(top bits, same for all hosts on the network) + host(bottom bits) bits within IPAddr, enabling networks to be allocated in smaller blocks-> optimized addr Usage. Greatly improved address usage + routing efficiency.

**Notation**: lowest IPaddr followed by slash, then prefix length (size of network portion)

**Prefix**: indicates different destination networks. variable-sized blocks where IPAddr are allocated; contiguous range of IPaddr assig to network.

**Prefix Length:** determines network portion size, and available host addresses. Shorter prefix ->larger# of hostADDr available. Larger -> smaller#.

Ex: 192.0.2.0/24: /24 is a smaller network of host addresses than /8

/24 => network portion size is first 24 bits of IPAddr, remaining 8 bits used for host portion => 2^8 possible host addresses.

But /8 => network portion size is first 8 bits of IP -> remaining 24bits used for host ->2^24  possible host addresses. Reserved private block

Range cheat: /20 has +15. +255, /21 has +7.+255, so /22 should have +3.+255 and /23 should have +1.+225

CIDR also introduced **subnet masks**; explicitly define network’s address bits. Consists of binary digits: 1s rep network portion, 0s rep host portion. Ex:255.255.255.0 -> /24 subnet.

**Network number (addr num):** Network mask (in binary, doing a bitwise-AND) IP addr. Crucial for Efficient routing on the internet, since networks are assigned in blocks, intermediate routes need only maintain routes for the prefixes, not every individual host.

**Subnetting**: splitting network into several parts for internal use, while acting like a single network externally. **Key takeaway:** no single router needs to know info about all hosts in organization. Only the directly connected ones! Pros: scalability, more flexibility in management, future changes can be made without any external impact.

**Route Aggregation**: combines smaller network prefixes into larger, more generalized. This reduces routing table size, improves routing efficiency. When multiple overlapping prefixes are present, router selects longest (most specific) matching prefix to determine appropriate route. Ex: 192.24.12.0/22 is chosen over 192.24.0.0/19 since the former is longer.

**IPv6**: solves problem of IPv4 addr space exhausted. Addr now 128 bits: huge, unlikely to run out. Not widely deployed, because IPv4 is smaller->faster

**New changes:** simpler header->allows faster processing, improved security->now backwards compatibility to IPv4.), further QoS support.

**HEADER FIELDS:** **Version**(6), **Differentiated Services** (6 bits for service class, 2 bits for congestion control (ECN)), Flow Label (Peudo-Virtual Circuit identifier), Payload Length (bytes after 40 byte header),  Next header (used to specify additional headers or Protocol (TCP/UDP)), Hop Limit (same TTL as (Time to Live)), Src/Dest (16 bytes IPv6 addresses)).

**Key Differences with IPv4 Headers**: increased addr size, simplified pkt format. Instead of complicated option + protocol fields, simply combined into a “next header” field. Eliminated checksum since it’s used in TCP. No need to do it both in TCP/IPv6

**Addressing:**

**Format**: 8 groups of up to 4 hexadecimal digits, separated by colons. Grouping by pairs of bytes makes easier readability. Colons ->address compression i.e compressing 0’s. Backward compatibility w IPv4 by representing IPv4 addr within Ipv6 addr w specific format: series of zeroes followed by “ffff”, then 4 bytes of IPv4 addr.

**NAT, FRAGMENTATION**:

**Private addresses**: used within company’s internal network for hosts only needing internal access

**Network Address Translation (NAT):** allows multiple devices within private network to share single public IP address, when connecting 2 internet.

**PROCEDURE**: replaces source IP of outgoing pkts w/public IP(src addr) + assigns unique srcPrt num for each conn. Maintains translation table to map OG privIP + srcPorts to translated pubIP + srcPrts, IP & TCP checksums are recalculated. When pkt arrives from internet, NAT BOX uses destPrt to look up translation table, then sends pkt to correct internal host.

**NAT PROS:** widely used in homes/small business. Security adv by shielding internal network from incoming unsolicited packets,until outoging conn is established. Simplifies mngmnt

**NAT CONS:** 1) breaks e2e connectivity by intro firewalls + privAddr 2) violates IP archModel by assigning same IP to multiple interfaces 3) layering violation by modifying payload+transport layer protocols (prt numbers) 4) changes internet from connLess to connOriented, if NAT crashes and gets rebooted, it clears all the TCP connections that were using it 5) limits # of conn due to finite port num range

**FRAGMENTATION**: process of dividing (large) IP pkts into smaller frag 2 meet network size constraints. Each network has Maximum Transmission Unit (MTU) that determines maximum packet size. IP headers contains frag info: id, flags (DF/MF), fragOffset.

**fragOffset**: must be 8-byte boundary (to save 3 bits) and last frag has MF flag set to 0. Ex: 13 bits -> max offset (2^13 -1) \* 8 = 65,528. Allows receiving host to reconstruct out-of-rder frag in buffer, similar to TCP segments.

**IPv4:** allows nontransparent fragmentation or pathMTUDiscv, max accept size 576 bytes

**IPv6:** expects hosts to discover optimal path MTU, routers will not perform fragmentation, done at sender. Min accept 1280 bytes.

**Cons**: reassembling pkts correctly is challenging (**Transparent Frag**: reassemble at nxt router, later routers unaware that frag occurred, **Nontransparent** **Frag**: used by IP, reassemble at destHost). Incurs overhead. Can result in pkt loss if frag’s loss

**Path MTU Discovery:** alternative approach to frag. TCP attempts to dynamically find optimal ptk size that can be transmitted without frag, by probing the network path.

IPv4: supports both NontransparentFrag + path MTU. IPv6 requires pathMTUDiscv

**ROUTING ALGOS, ICMP, DHCP, ARP:**

**Routing**: creates forwarding tables via routing algo. Properties of good routing algorithm: correctness, simplicity, robustness (router crash shouldn’t need network reboot), stability (reaches equib and stays there, fairness (no starving), efficiency, flexibility.

Fairness vs Eff: choosing most efficient course for handling traffic between specific points. Delay vs Bandwidth: Optimization considerations for mean pkt delay or max network throughput. Simplest Approach: minimizes num of hops a packet has to make (reduce per pkt bandwidth and improve delay. Reduces distance travelled but not guaranteed). Actual algo: cost given to each link, flexible but cant express all routing prefs.

**Non-Adaptive Routing Algo:** calculated offline, updated at boot and doesn’t respond to changes/failures. Reasonable where there is clear or implicit choice i.e: home router

**Adaptive routing:** adjusts to changes in network topology (i.e: updated costs) and traffic level. More eff than flooding but adjusting to traffic level can redirect congestion resulting in root flapping/instability. Aims to optimize some property.

**Simplest example is flooding:** send packet on every possible links. guaranteeing shortest distance but generates duplicate packets. Extremely robust due to brute force yet highly inefficient since it can’t discard packets. Used as base comparison.

**OPTIMALITY PRINCIPLE (BELLMAN):** optimal path from one router to another also applies to subsequent paths on the way. i.e: If router J is on optimal path from router I to K, then optimal path from J to K also is on same route. However is dependent on what is best/optimal.

**Sink Tree**: set of optimal routes from all sources forming a tree rooted at the destination

**Shortest Path Algo:** uses labelled graphs and algos like Dijkstra’s to find shortest path between a sink and all sources. Distances must be non-negative.

**DIJKSTRA’S:** nodes divided into 3 groups:

Unseen: not neighbors of any processed nodes

Open: visited neighbors for which path is known

Closed: nodes have been visited, best path to them is known. Moves nodes from unseen->open->closed

All nodes initially have value of INFINITY (“unseen”) since no path is known

Steps: visit source node, open its neighbors, set their labels as distance to them. Repeat until all nodes are visited or the destination is found:

1) examine adjacent nodes to current “working node”, calculate distances, and update labels if improved.

2) examine all tentative (open) nodes in the graph and select the one with the LOWEST distance as new working node

3) make new working node permanent (CLOSED) and go back to initial step for its neighbors

**LINK STATE (LS) ROUTING**: combination of information sharing and shortest path algorithm=distributed algo used to determine shortest paths in network. Replaced Distance Vector Routing (Bellman-Ford) due to its faster convergence.

**5 steps**: discover neighboring routers & learn their network addr, set distance/cost metric to (can be set auto or manually, based on bandwidth) to each neigh, construct pkt containing learned info, send+receive pctks 2/from all other routers, compute shortest path to every other router in the network.

**Neighbors**: discovered by sending HELLO packets on each interfaces. In return, other router on end must reply with unique ID.

**LS Packets contain:** ID, seq# (32 bits to prevent wrap-around issues, starts from 0 at boot, increments by 1, handles reordering), age (helps discard outdated routing+LS pkt info, reduced by 1 every second, helps with accuracy of routing info -helps with router crashes), neighborList w respective costs

Reliable Flooding is used to send packets to all other routers reliably, with acks to ensure successful delivery. To avoid unnecessary flooding, routers compare seq#s and discard packets with lower or equal numbers.

**Autonomous Systems (ASes):** collections of routers under same administrative protocol

**BGP (Border Gateway Protocol):** runs on top of TCP. for external routing between ASes in internet. Allows network admins 2 control how network’s being used and ISPs to restrict traffic from other ISPs. Goal: establish “Valley Free Routes” that avoid ASes charging for transit, Tier 1 ISP: v big reach, hierarchy. Routes are based on prefs + policies + political factors: not always clear “best” route. Supports customer/provider relationships. Providers advertise routes for entire internet, while customers advertise routes for own network.

**Different Planes of Protocols in Networking**

**Planes**: protocols don’t really form a single stack. Instead multiple stacks side by side called planes.

**Data Plane:** handles pkt forwarding at network layer

**Control Plane**: determines route selection and management

**Management Plane:** deals with setting policies and configurations

**Internet Control Protocols**: used at internet layer to manage functionality = ICMP, DHCP, ARP

**Traceroute:** find all the routers on the path on path between yourself and dest. determines path&timings of packets between src + dest, uses time-exceeded msgs and TTL (hop count) to identify routers on the path. Each pkt has an incremented TTL by to reveal each router IP addr on the way to destination. Useful but imperfect: packets with diff TTL but don’t necessarily take the same path, might look adjacent but is adjacent distance on a diff path.

**ICMP: Internet Control Message Protocol (ICMP):** used for control messages in internet layer. Includes and basis of functions like echo requests (**PING**) and time exceeded messages (Traceroute). Msgs that can be sent using ICMP:

**DHCP: Dynamic Host Configuration Protocol:** solution for automating IP address handling for hosts/interfaces. DHCP server assigns IP addresses + other params (gateway, DNS server addr, time servers) to hosts. Host has a DHCP server for issuing IP addresses, Host sends out DHCP DISCOVER packet (routers can be configured to relay these to DHCP server if not directly connected to same network). DHCP server receives req and responds with DHCP OFFER pckt containing an avail IP address. Leases are used to manage IP address allocation, time after IP address allocation will be reclaimed by server and reissued (host can req a renewal before lease end).

**MAC Address**: (media access control) solution if we don’t have IP address but DHCP svr needs to know where to send response to. globally unique interface identifier, used at host-to-network/data-link layer, typically hard-coded by the manufacturer, btwn 48-64 bits, translates IP addresses to network addresses for communication. Every ethernet/WiFi device has unique MAC address. Used at Host-to-network/data link layer.

**ARP: Address Resolution Protocol**: translate IP addresses to MAC addr. Broadcasts requests (an ethernet or wifi pcket etc) to every host on network asking who owns target IP addr, IP owner will respond with MAC addr. Used to communicate between internet layer and underlying network layer. **Pros**: Simple, used frequently to find communication routes, esp w routers. **Con**: inefficient, lack of auth=> security vulnerabilities, caching of ARP responses => spoofing (false identity) attacks