## DNS

Hardwired **Root**, then professionally managed **Top-Level Domain** servers, then locally maintaned **authoritative**. Root servers replicated through **anycast** among locations A,. . .,M with same address s.t. packets delivered to closest location with that addr. Hierarchy helps with name resolution (walking up/down) and name allocation (control over namespace partitioned).
**Recursive** query asks server to get answer (eg: you to/from local DNS server), **Iterative** query asks server who to ask next (local DNS server to {root DNS, TLD DNS, authoritative DNS}).
DNS servers delete cached entries when TTL expires. (Optional) negative caching remembers what doesn't work.
DNS resource record format: (name, value, type, ttl).
Type **A**=hostname → IP addr. Type **NS**=domain → hostname of authoritative name server for this domain. Type **PTR**=reversed IP quads → hostname. Type **CNAME**=alias name for some "canonical" name → canonical name. Type **MX**=value is name of mailserver, includes weight/preference.
Flags for {query, reply}, recursion {desired, available}, reply is authoritative.
Reliability through replication, using UDP, trying alternate servers on timeout (exponential backoff when retrying same server), same identifier for all queries (don't care which server responds).
Vulnerabilities: recursive resolution, cache poisoning ("additional" records can be anything! ie: for google.com).

## Web

HTTP request: request line (method, resource, protocol ver), request headers, newline, optional data in body).
HTTP response: status line (protocol ver, status code, status phrase), response headers, newline, optinal data in body.
Response codes: 1xx informational, 2xx success, 3xx redirection, 4xx client error, 5xx server error.
**Performance** for fetching objects: one at a time (naive, incurs $\geq$ 2 RTTs per object), concurrent (in parallel, unordered responses, makes more network traffic), pipelined (batching requese and respones reduces connection overhead, maintains order of responses, in single TCP conn), persistent (default in HTTP/1.1, maintain TCP conn across multiple requests to avoid connection overhead, allow TCP to learn more accurate RTT estimate, allow TCP congection window to increase).
**HTTP caching**: use "If-modified-since" in GET requests (server returns "304 Not Modified" in response or "200 OK" with latest object), response headers have "Expires" (how long safe to cache) or "No-cache" (always get from server)
**Reverse proxies** cache close to server, decrease server load, only for static content.
**Forward proxies** cache close to clients, reduce network traffic and latency, done by ISPs or LANs.
**CDN**s integrate forward and reverse caching, processes dynamic web pages and transcodes (ie: modify embedded URLs to reference CDN domains).
Multiple sites per machine: include "Host" header to disambiguate. Multiple machines per site (helps handle load or when content isn't cacheable): load balancer to ensure packets from single TCP conn goes to same replica. Multiple addresses, multiple machines: DNS server returns different addrs.

## Interdomain Routing

Internet is made of set of "Autonomous Systems" (ASes sometimes called "domains").
Border Gateway Protocol (**BGP**) makes destination prefix to egress point. Interior Gateway Protocol (**IGP**) computes paths within AS by mapping agress point to outgoing link.
**Autonomous ASes** choose internal routing protocol, rotue externally based on policy, want to keep connections and policies private.
**Business relationships** {customer, provider, peer}: "when sending traffic, route through customers over peers, and peers over providers" and don't carry traffic from one provider to another provider.
**Path-vector routing** sends distance matric per destination along with the entire path. **Selection** policy says which paths I want my traffic to take. **Export** policy says whose trafic I am willing to carry (ie: let customers use any of my routes, let anyone route through me to my customer, don't export route to someone on that route, block all else) **P-V issues**: reachability (connected graph doesn't assure reachability), security ("blackhole": AS can claim to serve prefix that don't have route to), performance (nonissue: policy-chosen paths aren't shortest; real issue: convergence times (never converging due to policy oscillations), path changes must be re-advertized to every upstream node, each BGP router must know path to every other IP prefix and # of prefix growth more than linear)

## Routing Challenges

**Policy dispute resolution**: if policies follow normal business practices ("Gao-Rexford conditions" are essentially provider/peer/customer policy categories), stability is guaranteed.
**Precedence** to solve policy oscillation: Computed local "precedence" is higher for less preferred routes. Incoming precedence carried by packet, and local determined by past history. Most preferred route picked from the lowest incoming precedence values. Outgoing precedence is sum of incoming and local. *Not deployed!*.
**Routing resilience**: routing needs to be both consistent/convergent and timely. **Multipath routing** provides more than one path for S-D pair, and allows endpoints to choose, but has large delay while endpoints detect failure (RTT).
**Failure-Carrying Packets** (FCP): ensure routers have consistent view of network, even if it's out of date. Use reliable flooding. Each map has sequence number. Routers write this number into packet headers, and only decrement the counter. When packet arrives and next-hop link for the path computed with consistent state is down, insert failure information into packet header. if failure persists, it will be included in next consistent picture of network. Guarantees packet delivery if valid path exists during failure process. *Problems*: requires changes to packet header, header could get long, requires fast recomputation of routes, doesn't address traffic engineering.
**Traffic engineering** distributes load on the network. Connectivity is necessary, but also needs to provide decent service, requiring links to not be overloaded.
**Routing Along DAGs** (RAD): routing tables are per-destination. Directed-acyclic graph gaurantee loop-free, local decision for failure recovery, and adaptive load balancing. Local decision chooses which outgoing links to use, decides how to spread load, and pushes back when all outgoing links are congested (send congestion signal on incoming links to upstream nodes). *Theorem*: when all traffic goes to a single destination, local load balancing leads to optimal throughput. **Computation** for destination $v$: shortest-path computation with consistent method of breaking ties. *Algorithm*: when packet arrives, send out any outgoing link. When outgoing link fails (or is reversed): if another outgoing links exist, do nothing, else reverse all incoming links to outgoing. *Link reversal guarantees connectivity. Proof*: DAG guarantees no loop at beginning, and link reversal never creates loop. *Problems*: computation takes time, packets can be lost in meantime, link reversals are on "control plane"
**Data-Driven Connectivity** (DDC): define link reversal as event of packet arriving in "reverse" direction, and action of removing that link from outgoing set.

## Ethernet

Originally broadcast, but now "switched", so no sharing channel. Switches (switch on frames in Link Layer) enable concurrent comm while completely avoiding collissions (if hosts directly attached). Switches map destination MAC to outgoing interface, construct switch table automatically, and floods when no entry in table. Use spanning tree to avoid flooding through links that form loops. **Spanning tree algo**: pick a root (destination for all shortest paths) with the smallest identifier (MAC addr). Compute shortest paths to the root, only keep links that are on shortest path. To break ties: choose path that uses neighbor swith with lower ID. Messages in format (Y, d, X) propose Y as root from node X, with distance d. *Steps*: each switch proposes itself as root[$\forall x \in Switches$, x announces (x,0,x)]. Switches update their view of root. Upon receiving message (Y, d, Z) from Z, if Y's id is smaller, view Y as new root. Switches compute distance from root by adding 1 to distance received from neighbor. Identify interfaces not on shortest path to root, and exclude from spanning tree. If root or shortest distance to it changed, "flood" updated message (Y, d+1, X). **Robustness**: root switch periodically reannounces itself as root, detecting failures through timeout (soft state; if no word from root, time out and claim to be root).
**Negatives**: network unuseable until tree rebuilt, and forwarding is only over spanning tree (unused bandwidth)

## Broadcast vs. Point-to-Point

**Point-to-point** is dedicated pairwise communication, like Ethernet switch to host.
**Broadcast** is shared wire or medium, like traditional Ethernet.

### Multiple Access Algorithms

**Time Division Multiple Access**: time-slot length is packet transmission time, unused slots idle, each station gets fixed length slot in each round.
**Frequency Division Multiple Access**: channel divided into frequency bands, each station assigned fixed band, unused bands go idle.
**Polling**: master node invites slave nodes to transmit, but sucks because of polling overhead, latency, and single point of failure.
**Token passing**: control token passed from one node to next sequentually, node must have token to send, but sucks because of token overhead, latency, being at mercy of any node.

### Random Access MAC Protocols

Node transmits packet at full channel data rate without previous coordination. Collisions result in data loss. *Carrier sense* is checking if someone else is already sending data and wait until other node is done. *Collision detection* is if someone else starts talking at the same time, stop, and make sure everyone knows there was a collision. *Randomness* is waiting for a random time before trying again.

**Aloha Signaling**: channel for random access and another for broadcast. Sites send packets to hub on random, and resends if no ACK received. Hub broadcasts packets to all sites, and sites can receive while sending. Resent with probability p. Max efficiency is approx 1/3. **Pros of Slotted Aloha**: single active node can continuously transmit at full rate, highly decentralized since only need slot synchronization, simple. **Cons of Slotted Aloha**: wasted slots from idle and collisions, collisions consume entire slot, and clock synchronization.
**Carrier Sense Multiple Access**: listen before transmit. If channel sensed idle, transmit entire frame, else if sensed busy, defer transmission. Reduces, but doesn't eliminate all collisions because of nonzero propagation delay. **CSMA/CD**: collisions detected within short time, and colliding transmissions aborted to reduce wastage. CD easy in wired LANs, but not in WLANs. Imposes restrictions on maximum length of wire and minimum length of frame. *Performance*: time wasted in collisions proprtional to distance d. Time spend tramsitting packet is packet length p divided by bandwdith b. For some constant K, efficiency $\approx \frac{\frac{p}{b}}{\frac{p}{b}+kd}$. Since E decreases as b increases, high-speed LANs are all switched.
**Ethernet Multiple Access**: CSMA/CD protocol has carrier sense, collision detection (on collisiosn, abort and send jam signal), and binary exponential back-off random access (after m-th collision, choose K randomly from 0 to $2^m - 1$, and wait K * 512 bit times before retry. In reality, it performs well, not optimal, and is mostly irrelevant, since current ethernet is switched. Stability for finite N, so Ethernet can handle nonzero traffic load without collapse. For infinite N, all backoff algorithms unstable.
When 2 hosts, each with infinite packets to send, compete, there is chance the first to successfully transmit will never relinquish the channel, and the other host will never send. If hosts have finite # of packets to send, time waiting for loser to start is proprtional to time winner was sending. Exponential backoff has channel capture and efficiency less than 1. Superlinear polynomial backoff has channel capture and effiency is 1 (for finite # of hosts N). Sublinear polynomial backoff has no channel capture (loser not shut out) and efficiency is less han 1 (goes to 0 for large N) since time is wasted resolving collisions.

## Congestion Control

**Flow control** keeps 1 fast sender from overwhelming a slow receiver. **Congestion control** keeps a set of senders from overloading the network. Internet traffic is bursty, so if many packets arrive in short time period, there are delays or buffer may overflow. End hosts adjust sending rate based on feedback from network. Drawbacks is suboptimal efficiency, relying on end system cooperation, and messy dynamics (all end systems simultaneously adjusting). Detect congestion from network telling source (risky, since signal itself could be dropped or add to congestion), packet delays going up (tricky because of noisy signal), and packet loss (but there are non-congestive losses like checksum errors). *Adjust CWND* because consequences of over-sized window (packets dropped and retrans) much worse than under-sized window (lower throughput), so gently increase and rapidly decrease. **AIMD** increases by one MSS on success of last window, and divides window in half on packet loss, creating TCP sawtooth. **Slow-Start**: AIMD starts too slowly, so start with small congestion window (ie: CWND=1 MSS), but increase exponentially until first loss. **Congestion Avoidance** gently increases CWND by fraction of MSS per received ACK: CWMD += MSS divided by # packets per window. **Fast Retransmission**: halve CWND when sender sees 3 dup ACKs. **Timeout**: sender starts timer for RTO seconds, restarting timer whenever ACK arrives. If timer expires, sets Slow-Start Threshold SSTHRESH to CWND/2, set CWND to 1 MSS, retransmit first lost packet, and execute Slow Start until CWND > SSTHRESH, then switch to Additive Increase.
**Throughput** = (MSS / RTT) * sqrt(3/2p) with packet drop rate p. *Why AIMD?* congestion control challenges: when single flow adjusting to bottleneck bandwidth (slow-start can result in many loses when CWND is double pipe-filling value), single flow adjusting to bandwidth variations (AIAD=gentle inc+gentle dec; AIMD=gentle inc+drastic dec; MIAD=drastic inc+gentle dec(too many loses); MIMD=drastic inc+drastic dec), and when multiple flows must share bandwidth "fairly" and without overloading network (eliminates MIMD and AIAD). AIMD is only "fair" choice, but hosts that send more flows get more bandwidth.
**Random Early Drop** (RED) drops packets before queue is full with drop probability D (function of queue size) to keep queue average small, but tolerate bursts, and reducing synch between flows. **Explicit Congestion Notification** (ECN) is bit in IP packet header that is set instead of RED router dropping, so corruption isn't confused with congestion and recovery isn't confused with rate adjustment.

### Router-Assisted Congestion Control

Isolation by giving each flow its own FIFO queue, and router services flow round-robin. *Fair Queuing* (FQ) is round-robin generalized to case where not all packets are MTUs, where weighted fair queueing (WFQ) lets router assign different flows different shares. Flows can pick whatever CC scheme, bandwidth share doesn't depend on RTT. Adjustment helps flows get speed. Isolation protects flows from cheaters and allows innovation in CC algos. **Rate-Control Protocol** (RCP): router inserts Fair Share f into packet header s.t. total bandwidth C = sum( min(f, bandwidth demanded $r_i$))
**Frank Kelly** suggested using ECN as congestion markers, charging money when ECN bit set, since giving equal bandwidth shares to "flows" doesn't make sense (if i have more flows than you?, if you use more congested hops?, is a flow a TCP conn or source-destination pair or just source?)

### Odds & Ends

**Delayed Acknowledgements** set timer upon receiving packet so that if application generates data, it can piggyback the ACK. If timer expires or out-of-order segment arrives, send non-piggybacked ACK. **ACK-splitting**: growing window by 1 packet for each ACK received, and sending M distinct ACKs for 1 packet caues growth factor proprtional to M. **High Speed TCP**: increase CWDN by more on each success, and decrease CWDN by less.

## Link-Layer

**Services**: encoding (into binary), framing (packet into frame, adding header, trailer), error detection (from signal attenuation, noise, and receiver may ask for repeat), resolving contention (for shared media), flow control (pacing between sender and receiver). Link-level broadcast uses addr ff:ff:ff:ff:ff:ff. IP-level broadcast can only do "local" broadcast using 255.255.255.255. Link-layer adapters only understand MAC. **Sending packet** to same "local" (local if netmask with IP addr the same as our own masked addr) subnet uses MAC addr of dest, to other subnet uses MAC addr of first-hop router (do local case for router's ip, instead of ultimate dest ip). Determine whether host is on same subnet by using netmask (with DHCP).
Determine MAC addr using **Address Resolution Protocol**. Each node has ARP table of IP addr and MAC addr pair. If IP addr not in table, sender broadcasts requested IP (ie, for gateway) , receiver responds with MAC, and sender caches result in its ARP table. Impersonation is easy because any node can say whatever they want, and legit receiver never realizes.

## Internet Control Messages Protocol

Triggered when IP packet encounters a problem. ICMP packet sent back to source IP with error info and *excerpt* from original packet. Source inspects except to identify socket. ICMP packet not sent if problem packet is ICMP, and only sent for fragment 0 of fragment groups. **Types**: Need Fragmentation (too large, and DF set), TTL Expired (if TTL decrements to 0), Unreachable (subtypes for network, host, and port), Source Quench (old signal asking sender to slow down), Redirect (tells source to use different local router). Path MTU is minimum end-to-end Maximum Transmission Unit, found by trying desired value with DF set, then trying smaller after receiving Need Frag. Traceroute sends packet with TTL ranging from 1 to n, and each router along path sends Time Exceeded, and is identified by the IP source addr (the router).

## Multicast

At routers, copy data s.t. a most one copy of a data packet per link. LANs do link layer multicast by broadcasting. Receivers join multicast group with address G, sender(s) send data to address G, and network routes data to each receiver. Senders don't know list of receivers. *Barriers*: hard to change IP, not consistent with ISP economic model **Link Layer**: NIC normally listens for unicast addr A and broadcast addr B, but after joining group G, listens to packets to multicast addr G. Packets to group G flooded on all LAN segments like broadcast. **IP Layer**:
**Reverse Path Flooding**: if incoming link is shortest path to source, send on all links except incoming, else drop. Issues: links can receive multiple copes, and every link receives each multicast packet. **Reverse Path Broadcast**: choose single parent (based on distance, lower addr for tie-breaking) for each link along reverse shortest path to source, and only the parent forwards to child link.
**Pruning**: prune (source, group) at leaf if no members by sending Non-Membership Report (NMR) up tree, and if all children of router R send NMR, propagate prune for (S,R) to parent R.
**Core-Based Tree**: build tree from all members to core/root (spanning tree of members), packets are broadcast on tree, requires knowing root per group.

## Wireless

**MACA** needed because nodes can't hear *hidden terminals*, so Carrier Sense by itself is ineffective. *Exposed node*: Carrier sense prevents successful transmission because node hears other transmission and doesn't send even though it won't cause interference. **MACA**: no concept of global collision, collisions are at receiver (not sender) (doesn't matter if sender hears someone else), detect if receiver can hear sender, tells senders who might interfere with receiver to stop. Since we need ACKs, use carrier-sense for sender congestion, and MACA for receiver congestion. Send RTS-CTS-DATA-ACK. **RTS/CTS** (request to send, clear to send): shut up when hear either CTS or current transmission is over (hear ACK) (carrier sense). If only hear RTS and not CTS, you can send.