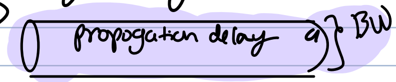
CS168 - MT1 | endhost ○——link——⊠< switch | ICANN > RIR > ISPs... > small orgs | 1 Gps = 10^9 bits/s | 1 Mps = 10^6 | TB = 10^{12}

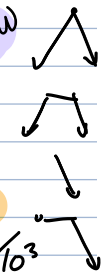
Internet: infastructure that ties together computing devices | no notification,
↳ Decentralized ↳ **Best effort service model**: no guarantee on if data will be delivered
↳ Dumb infrast. w/ smart endhosts, e2e, layering, "narrow waist"
Interoperability: entities can connect & comm w/ other entities easily
Protocol: specification of mssgs that comm. entities exchange (syntax & semantics)

Bandwidth: num bits sent/recieved per time (bps)
↳ **transmission delay**: size of packet / bandwidth



Propogation delay: len of link (m) / speed of light (m/s) (sec); non-zero ⓔⓧ Use smallest bandwidth given
Bandwidth-delay product (BDP): bandwidth × prop delay (bits)
Latency: packet delay = trans delay + prop delay + queuing delay
Utilization: usage / maximum | $1/10^3$ prop delay

ⓔⓧ 100B over 1 Mbps ⟶ 10^6 bps ⟶ 8×100 = 800 / 10^6 → 1st bit: $(1/10^6) + 1/10^3$; last bit: $(800/10^6) + 1/10^3$



Forwarding table: next hop; network addr (where) vs. net name (which host)

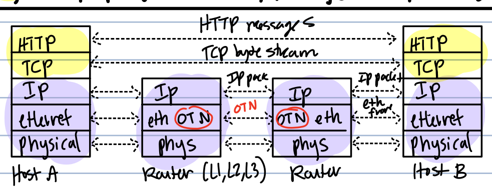
Control plane	Data plane	Management plane	Packet switching (default of internet)	Circuit switching
Compute fwd tables (routing algo/protocols), global, per network event (DV, LS)	actually fwd packets (forward), local (arriving packets & rt table) per packet arrival	interact w/ systems/humans to config & monitor device	**Best effort model** packet by packet basis. Better efficiency, fast start, easy fail recovery, simple implementation, efficient, best for bursty	**Reservation model** Predictable/understandable, good for smooth apps (bad for small data). Cons: setup/teardown time, bad failure handling, complex, endhosts need to detect failure

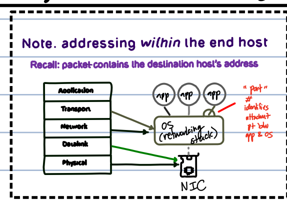
Autonomous System (AS): net/set of nets all managed/supv by single entity/org
Internet service provider (ISP): net of packet switches & comm links, 1+ ASes
statistically multiplexed: do not provision for worst case (hope all peaks don't happen at the same time; peak of aggr demand << aggr of peak demands) [P(f1+f2) vs. P(f1) + P(f2)]
Smooth: low ratio btw app. peak to avg transmission rate ⓔⓧ voice. **Bursty**: high ⓔⓧ data apps.

Transient overload: queue absorbs bursts, drain queue
Persistent overload: queue force to drop packets

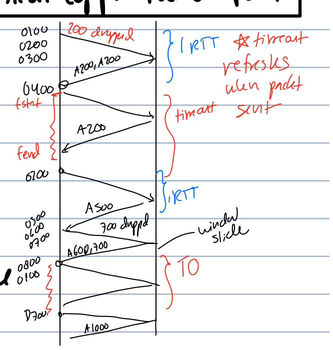
Layering: well-def interfaces; layer only interacts w one above & one below
☆ depends on layer below ☆ supports above ☆ independent ⟶ parallel innovation

layer	Desc	protocols
L7 application	Browser, mail/client, only at host, exactly-once delivery	SMTP HTTP DNS NTP
L4 transport	Part of OS (only at host), at-least-once, reliable data delivery, demux	TCP UDP
L3 Network	Best eff global packet delivery, best effort delivery, usually part of OS, IP	IP
L2 Data link	Best eff local packet deliv, hardware/drivers ex. ethernet frames	Ethernet FDDI PPP
L1 physical	Physical transfer of bits; hardware/drivers/wire	optical copper radio PSTN





Router: L1, L2, L3
Switches: L1, L2



Network stack: app (data) ⟷ stack ⟷ NIC (packets) func as a feat. of arrange, not possible
physical port: link ⟷ switches **logical**: app ⟷ OS stack
Socket: app → gen sock ⟷ logical port (OS uses port # ⟷ socket)

Routers: look up where to forward / send packet; topology mangement, not host 2 host
↳ **Spanning tree**: undirected graph, symmetric, connect all nodes, dist. based routing
User traffic: packet fwd according to installed routes, go across fabric to other linecard/fwd chips
☆ forwarding path == fast path, Control == slow (parsing info)
Control plane traffic: packets to this router ⓔⓧ BGP/IGP packets, send to controller card
Punt traffic: special handling ⓔⓧ TTL, send to controller card
Router capacity = N (# of external ports) × R (speed "line rate" of port)

project 1
- don't recharge poisoned routes
- don't add entry for poison route if not in
- periodically resend poisoned routes
- poison spread doesn't depend on resend time

$2^3 = 8$ $2^7 = 128$
$2^4 = 16$ $2^8 = 256$
$2^5 = 32$ $2^9 = 512$
$2^6 = 64$ $2^{10} = 1024$

Distance-vector: tell neighbors my least cost entries to a dest ③ If ad is next hop, replace
① If dest not in table → add ② If cur_route > ad rout + dist, replace entry (Ismthing happened)
Interval: resend every X sec → reliable vs. triggered: send on change, optimization
Split horizon: don't tell ur next hop if its the dest (loops can still happen)
vs. poision reverse: tell ur next hop infinity (loop state exist until next ad vs. expire
route poisoning: ∞ if down (faster propagation than timeout) ☆PR faster SH
☆ count to inf: bouncing & incr → solve w/ a max cap ☆ SH or PR doesn't solve!

Routing state validity: (1) no loops (2) no deadends → state is valid
☆ Local cannot be evaluated, only global
Connected / directed routes: manual | exp. duplication
static: want them, no routing proto to discover (☆ neither expire) | if 2+ loops

---

Link-state: global flooding (reliable: periodic resend); no count to ∞, can have loops; delay: time to detect failure, time to flood info, time to recompute

---

CIDR: classless inter domain routing (IP space) vs. classful: A: 8, B: 16, C: 24 (net bits)
aggregate! (ex) 12.1.0.0/17 == 12.1.0.0 to 12.1.127.255 ☆ 0 out if held for prefix (not aggr.)

LPM: longest prefix match; take longest route, overlapping possible

---

Inter-domain: between ASes Transit: carry on behalf of other AS Stub (on behalf of dir. hosts)
☆ Scalability via aggregation vs. Multihoming: more than 1 provider, must announce

---

BGP: extend DV (autonomy, policy, privacy); export (which can enter) import (leaves)
① May aggr dest ② prioritize policy ③ DV → PV: send entire rout (avoid loops + flexible policy)
④ selective route ads → reachability not guaranteed even if connected graph

G-R	dest prefix ad by	export to	cust.	everyone	peer	cust	provider	cust	If AS learns a route from a customer, AS will export route to everyone

☆ Assumptions for steady state guarantees ① cust-provider acyclic (peer is ok) ② chain leads to T1
↳ Then guarantee: ① Reachability (any 2 ASes can talk) ② Convergence (routers agree on paths)

---

eBGP: learn routes to different ASes iBGP: distr. knowledge internally IGP: internal routing (OSPF / RIP)

#	Rule (N=eBGP announces)	Desc ☆ May conflict
1 L	LOCAL PREF	Pick highest #
2 N	ASPATH	Pick shortest len
3 L	IGP path	Pick lowest cost
4 N	MED	Pick lowest
5	Router ID	Smallest next hop addr tie break

IPv4 (32 bits)

Version	4	Src IP	TTL
Hdr len	size	Dst IP	options
Type of service		priority/delay	ECN + DSCP
Total len	size	cksum	
ID	uniquely ID group of fragments		
Flags	No frag (1) More frag (2)		1, 2
Frag offset			
Protocol	which L4 / demux		6 = TCP 17 = UDP

(ctrl plane) (TTL)
cksum: updated at every router; entire header + TTL → new calc
Frag. off: 8 byte boundary: add bytes before frag, div by 8
☆ 1380 + 1220 + 1200 − 2(20) = OG size ☆ don't forget hdr!
VS. IPv6 (128): no cksum, frag, hdr len (const len), + flow label (related streams) (Hop limit == TTL)

Reliable transport: W based: allow W packets in flight, allow sender to transmit for entire RTT size
W × Packet size = RTT × B (bottleneck) ↳ As long as src retransmits unack'd packets → reliable ≠ in order

Timeout: resend of W size
Indiv: resend as soon as W ↑ leaves
Full: same ↗
Cum: send dup ACKs

Indiv ACK: 1 → ack(1) (ex) 1,2,3,4 sent, acks sent, ack 2 lost → ack 2 retransmit (loss = retransmission, waste)
Full info ACK: highest + additional (cum ack) (ex) 1,2,3,4 → acks (<=1)... (<=4) (ex) 1,2 lost, 3,4 lost → ack (<=1), ack (<=1 +3) (sizeable overhead, resillient)
Cum. ACK: highest seq. for which all recieved (ex) 1,2 lost, 3 → ack(<=1), ack(<=1).. (compact, more resillient, but incomplete info) W = 2BL

TCP: cum ACKs + byte streams + seq num (byte offsets) + timers kth byte, ISN+k+
CWND ≤ RTT × B MSS (seg size) = MTU − IP hdr − TCP hdr | seq # == ISN + k ACK = len(data)

src/dst: demux
seq #: start byte offset
Ack: next expected byte
cksum: hdr + payload
hdr len: 4 bits (4 byte, min is 5 words 20 B)
0 (reserved)
Flags: SYN ACK FIN
RST PSH URG
Urgent ptr: (URG flag to show priority)
options (ignore)

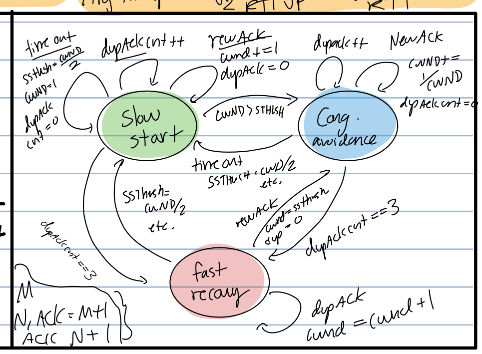
(ex) X, len B → ACK = X+B back → X+B len B sent → ACK = X+2B (no longer send but can recieve)
Establish: SYN SYN ACK ACK (ex) ISN = 19 ↳ send 19+1 → W = 10, ACK = ISN + MSS × (packets sent) (1-10 ↑ upt W) VS. FIN ACK FIN vs. RST

AIMD (most fair) AIAD (unfair) MIMD (unfair) MIAD (max. unfair) X1 + X2 > C then congested
Flow cntrl: prevent sender RWND (advertised) Congestion: avoid overload CWND (how much to send to links) RWND >> CWND
Sender-side = min (CWND, RWND) Avg num packs in flight = (CWND max + CWND min) / 2 Avg thruput = $\sqrt{\frac{3}{2}} \frac{MSS}{RTT\sqrt{p}}$ = avg # bytes in flight / RTT

Slow start ☆ MSS = packet size
CWND init = small const × MSS
Each ACK → CWND = CWND + 1 (☆ same as 2x CWND per RTT)
End: → CWND > SSTRESH → Cong. avoidance
↳ 3 dup Acks: fast recovery
↳ Timeout: rstrt SSTHRESH = CWND/2, CWND = 1, dup Ack cnt = 0

Cong. avoidance (AIMD)
Each ACK: CWND = CWND + 1/CWND
End: timeout: back to slow start
↳ 3 dup Acks: fast recovery

Fast recovery: retrans single packet (dupack := CWND += 1)
end: timeout → back to slow start
new ACK → congest avoidance

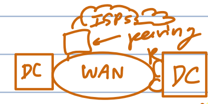


Final Cheatsheet

~1-2 servers per "U" → ~40 "U" per rack + ToR

Datacenters

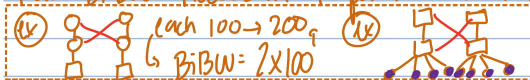
Anatomy: DC, Wide Area Network (WAN); mul DCs



Features: single organization, homogeneous, emphasize performance. E-W: machine-machine, N-S: outside

Bisection BW: Σ BW cut to partition network in half → high to support E-W, low = cong.

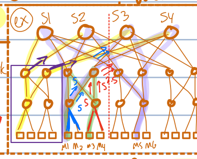
Full Bi BW: nodes in 1 partition can comm. simultaneously w/ nodes in other part at full rate



Full BiBW: cost of each (8/2) x 100G = 400G

Oversubscription: BiBW / Full ☆ if = 1 then worst case achievable vs. full. Full = Bi

Design 0: Fat tree	Design 1: Fat tree	Design 2: Clos tree
# ports = O(# racks), speed of ports; Switching speed = O(# servs x server) ☆ Bad fault tolerance, not scalable; Large cross bar switch	High BW btw layers; port can is down ☆ link speed / switching capacity is still restricted ↳ too expensive, not scalable	Interconnect smaller/cheaper switches into high BiBW network; Switch → multiple commodity switches; Oversubscription ratio = 1, Only w/ optimal load balancing



paths source→dest? only min len. (1) M1 → MS, M2 → M6
paths btw M1 M5 (2) # M1 M5 (4)
paths M1 M5 go thru S2? (1)
Bi BW = (1/2) × 10 (full) = 80 Gbps
TCP-optimal, ECMP = worst, ☆ both take same path
S1, S3 fail → 20 Gbps (each (10Gbps per link))
avg data rate: M1 can send to M5!
M2 → S, M5 → S, M2 S, M6 S → 1.5/4 = 2.5 since splitting link 2.5 Gbps

Congestion Control

Congestion Control: Queuing delay → TCP deliberately fills queues (elephant flows delay mice (short / latency sensitive)) Throughput-sensitive

Explicit Congestion Notif (ECN): router marks packet when queue len exceeds threshold, srcs cut rate ☆ responds before full

#1 DCTCP: routers mark packets earlier, sender reduce CWND on seeing ECN bit set (vs. waiting for packet loss)

Flow completion time (FCT): flow starting to last byte received, best to min!) Normalized = FCT / ideal FCT

pFabric: packets carry single priority # = # unack bytes, send highest / drop low, does well (mice higher priority) Harder to implement

#2 Swift: SIMD, delay based CC. RTT < target → + cwnd else: decrease cwnd (mul.) #1 & #2 easy to add

Path Selection

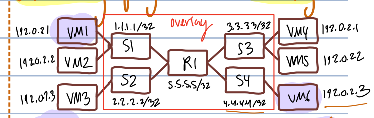
Path Selection: fully utilize high BW → need multi-path Rout

Equal Cost Multi Path (ECMP): next hop info for all equal cost paths; pick nxt hop using hash → "per flow" ☆ prevent re-ordering; same flow, same path

H(F) = src_ip, dst_ip, protocol, tcp_src_port, dst port

Routing: extend DV (store mul. next hops per dest), extend LS (calc all shortest paths btw src & dest)

Underlay: phy. hosts. Overlay: VM



VM1 → S1 → R1 → S2 → VM6

Server/R	Adds/del addr?	overlay addr	Underlay addr
S1	Adds	4.4.4.4/32	192.0.2.3
S5	N/A	4.4.4.4/32	192.0.2.3
S4	Delete	4.4.4.4/32	192.0.2.3

How many dest does S5 track w/o encaps? w/? (can't underlay)
w/o encapsulation = 11, w/ = just PAs = 5 underlay

Multitenancy: 2+ same VM addr in 1 server → add Virtual network ID (ex) VM1 & 2 = 192.0.2.2 at S2 ⇒ tag Pepsi vs. tag Coke

SDN

SDN: Management plane: config, monitoring, how operators tell routers what to do, slow change, bottleneck for network ops ⇒ Sol: decouple planes w/ APIs

☆ Flexibility in who implements control / mngmnt planes, flexibility in when we run the planes. OpenFlow: API data ↔ control, program fwd plane, flexible, control plane

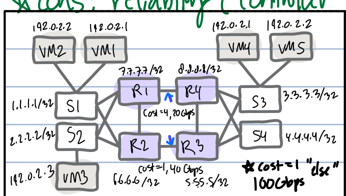
(OF cont) constrained by Fwd'ing chips (NOT SDN; OF only does fwd); is interface btw Networking OS & switches ☆ Data plane remains unchanged

SDN: fix inflexibility of control plane; API data ↔ control (control abstractions ① fwd model of switches e.g. OF ② network state - global view via NOS ③ specification of intended behavior, goals of oper ator)

Commodity network devices, logically centralized controller (topology changes, overlay: handle Encap/Decap) operator specific control (biz policies)

Traffic engineering: constrained shortest path first (CSPF): traffic should be on shortest path w/ sufficient capacity ☆ May take more expensive path

☆ cons: reliability (controller failing = single point), scalability, security ☆ Geofencing ☆ Path diversity (never intersect) ☆ Fwding = controller propagates



(ex) I S1 → S4 = 30 Gbps
II S2 → S4 = 20 Gbps
R2 - R3 - S4: no traffic eng.
☆ go by cost of paths (low)
There is congestion 50 > 40
S1 R2 R3 S4
S2 R1 R4 S4 } efficient route

(ex) Traffic engineering (to S4)

Workload	Location	TE header	Dest IP
I	leaving S1	R2 or R3	4.4.4.4
I	Arriving at S4	R1 not enough, it goes R2	4.4.4.4
II	leaving S2	R4	4.4.4.4
II	Arriving at S4		4.4.4.4

(ex) Traffic eng. for best Util: I VM1 → VM5 = 40 Gbps, II VM3 → VM5 = 10 Gbps

Wkld	Location	TE label	Underlay dst	Overlay Dst
I	leave VM			192.0.2.2
I	leave S1	R2 or R3	3.3.3.3	192.0.2.2
I	Arrive S3		3.3.3.3	192.0.2.2
I	Arrive VM2			192.0.2.2

Wkld	Location	TE label	Underlay dst	Overlay Dst
II	leave VM3			192.0.2.2
II	leave S2	R4	3.3.3.3	192.0.2.2
II	Arrive S3		3.3.3.3	192.0.2.2
II	Arrive VM5			192.0.2.2

TE hdr = only for routers

DNS

Domain Name System: host name ↔ IP ☆ use UDP ☆ DNS root (ICANN), Top level domains (TLDs): .com, .mil, controlled by parties

Zone: admin auth responsible for part of hierarchy; delegate auth to another NS in zone. DNS Name lookup → root server → redirect to NS for

DNS Query: client ⇄ resolving server (DNS servs) ⇄ recursive server (ISPs) (ex) dig +trace ① root replies with NS → A (IPv4) or AAAA (IPv6)

DNS $ing: cached in resolver for popular requests (reduce load on auth servers)

Hostname	IP	TTL

client → DNS serv $ → root, .com, ...

☆ DNS zone availability (2+ NS per zone) ☆ DNS root server availability → Anycast: advertise same IP prefix from mul locations → least cost routing to best

CNAME: alias MX: mail exchange record ☆ DNS load balancing: use geographic addr to choose best reply (geog = closest)

(ex) No $: 2a+2b+2f → # requests = ⌊T min / 2a+2b+2f⌋ (ex) $, TTL L > T: # req = {1 + ⌊T - (2a+2b+2f) / 2a+2f⌋} ← time to get info out of $

(ex) T = 600, TTL = 30, a = b = c = ... = f = 1. cached in: a + 2b + 2c + 2d + 2e = 9 +30 → 39 expires, get back at 10, 12 make 1st req → 39 - 12 = 27 sec ⌈27/4⌉ = 7

Expires at 40 sec → T/40 = 600/40 = 15 times can repeat → 15(7+1) = 120 total requests. HEAD (get just headers, not content)

HTTP

Hyper Text protocol: L7 stateless protocol; TCP based on port 80. request <method> <requested URL> <version>

(ex) GET /test.html HTTP/1.1 User-Agent: robjs <Headers> (ex) POST <url> HTTP/1.1 <hdrs> <content by client> PUT ... <content>

response (same TCP connection) <version> <status code> <optional message> <content> Status code: server propagates info about result of request to client

100: informational	200 (OK) 201 (created)	300 (redirection) 301 (moved perm) 302 (found) 304 (not modified)	400 client err 401 unauth 403 forbidden 404 not found	500 (server err) - internal server err 503 (server unavailable)

HTTP $: located at browser. Private $: tied to specific client. Proxy: in network, reduce Network BW btw origin server. Best when low BW

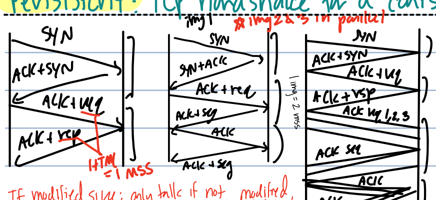
Managed: run by application provider, "close" to end users; redirect mechanism; reduce latency. fwd: $ near client. Reverse: reduce server load

Content Delivery Networks (CDNs): deployments of servers that can serve content; reduce latency of loading + spread load

☆ usually placed on edge of application / peering; in ISP (deployment depth) limited efficiency but reduces network capacity

URL: scheme :// host [:port] / path / resource. Request patterns: Concurrent: same link shared, one reply at a time. pipelined: 3 → 3 sharing BW

Persistent: TCP handshake for a consistent session vs. + SYN, SYN+ACK, ACK+rel. ☆ if concurrent, need to split



If modified since: only talk if not modified, responds w/ hdrs, 304
Cache-control: no cache

(ex) Download HTML size P; 2 images size M; SYN, ACK, SYNACK, HTTP = time Z to travel, HTTP = throughput T, compute time

① Sequential, non-persist: 2Z (SYN+SYNACK) + Z (ACK/HTTP) + (P/T + Z) (HTML webpage); 1 img = 2Z + Z + (M/T + Z) ... 2nd img

② Concurrent, non-persist: 2Z + Z + (P/T + Z) (webpage); 2Z + Z + (M/(T/2) + Z) aka (2M/T + Z)

③ Sequential req, persist: 2Z + Z + (P/T + Z); Z (HTTP req) + (M/T + Z), X2 for 2nd img

④ Pipelined, persist: 2Z + Z + (P/T + Z); Z (both HTTP req); M/T (first img); (M/T + Z) (2nd img)

☆ Small file == prop delay. ☆ Big file == transmission delay (last bit)

Ethernet L2; local area network (LAN) needs Media Access control (MAC): 48 bits, burned in, globally unique — random exp. back off

Original: polling (coordinator), token-passing; carrier sense Mul. Access w/ collision detection (CSMA/CD): listen when talk, stop if hear

Unicast: 1 recipient, use frame (ethernet packet; checksum, preamble for sep. Dest/src MAC); Broadcast: FF:FF; Multicast: to all in a group, 01:00:00:00 set bit

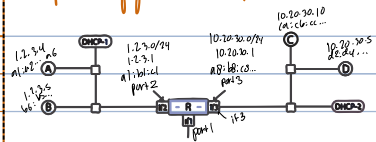
ARP (for same subnet): Broadcast request, unicast reply → ARP table IPv4, MAC, interface, ttl (IP addr might change over)

☆ L2 may change in hops, but not L3 (IP) ☆ ARP across LANs: check if addr in network (subnet mask) → first hop router ARP & send

---

Dynamic Host Config protocol (DHCP): application; learn self IP, network/subnet mask, first hop router's IP addr, DNS servers IP

	IPv4 src	Dst	MAC src	MAC dst
Discover	0.0.0.0	255.255.	senders MAC	FF:FF
offer (all send)	server IP	255.255	senders MAC	client MAC
Request	0.0.0.0	255.255.	senders MAC	FF:FF
ACK	server IP	255.255.	senders MAC	client ACK



(ex) A determines C on same subnet? → use A IP, subnet mask, C IP /24

☆ A IP (AND) subnet == C IP (AND) subnet — Sub: 1111111 1111111 0000; A: 00001 00010 00010; C: 001010 00100 00110 — X not the same

(ex) Router sent to C; Src IP: 1.2.3.4 (A), MAC = if3 on R, dest IP (C), Dest MAC (C's MAC) ☆ Note MAC changed across subnets

IPv6 uses Auto Config / extended unique identifier; Stateless Addr Auto Config (SLAAC): use neighbor disc (NDP) to solicit routers

---

E2E Network Addr translation (NAT) (breaks e2e): edge of network, port addr translation; L4 (TCP/UDP) flow awareness, many hosts share single pub addr

Retranslate: Src IP → public IP, modify TCP src port to distinguish 2 flows ☆ IP src addr (privacy) TCP src port (2 diff hosts contacting same remote addr)

DHCP	ARP for MAC (to use for DNS)	DNS for request B IP	ARP req for B's MAC	TCP connect	HTTP req (TCP)	TCP teardown	A wants C	DNS req to C	ARP for 1st hop	TCP handshake	TCP (HTTP) data transfer	TCP teardown
Discover (send), offer (receive), Request (sent), Ack (receive)	ARP req (send), ARP reply (receive)	DNS req (send), DNS reply (receive)	ARP req, ARP reply	TCP SYN, TCP SYN+ACK, TCP ACK (no piggy backing)	HTTP req (send), ACK to req (recv), HTTP reply (recv), ACK to reply (send)	Fin (send), ACK (receive), Fin (receive), Ack (send)	☆ already have ARP of DNS	DNS req (send), DNS reply (receive)	ARP req, ARP reply	TCP SYN, TCP SYN+ACK, TCP ACK	HTTP req (send), ACK to req (recv), HTTP reply (recv), ACK to reply (send)	Fin (send), ACK (receive), Fin (receive), Ack (send)

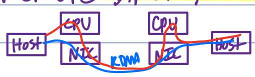
Host Networking ☆ implement functions @ host to support abstraction of network as fast, reliable, secure, ordered byte stream

☆ Traditional OS-based networking = hard (copies, development, CPU resources) Kernel/OS Bypass: data copy App → OS' NIC vs. store data into shared memory; "offload copy work; packet processing & network protocols in userspace" ☆ still need to reduce CPU

Offload to NIC (max CPU cycles, efficiency, performance) ☆ phases: offload stack/functions to NIC, free CPU → ① Simple stateless: checksum, segment (sep into of packets), tx/rx queue select ☆ complex/$
② simple stateful/accelerating data plane: BW mngmt, forwarding ③ Protocol offloads: TCP/RDMA, programmable data plane

---

Remote Direct Mem Access (RDMA): minimize CPU transfers; mem-2-mem btw servers; CPU just inits, while NIC & network responsible



Queue pair; send/recieve qus w/ WQE: ptr to mem to transfer (send a) / recieve data (recieve a); NIC notifies CPU via completion queue elements (CQEs)

Pros: high performance (low latency by removing SW var), CPU efficiency | Cons: more complex, limited protocol compat.

CC: delay based CC; Signals ⇓ Swift: delay based, measure RTT as true, AIMD on packet delay, add'l: high packet delay in loss CC

Loss	BW	ECN	Delay
Drops; mssg is late, aka rate increased when info received	measure rate at which RX ACKs; could mix up data / not enough data	Mark if queue > byte or time threshold; ☆ respond to cong. before queues full	array /2u; coarse signal; fails to handle implicit challenges

Load balancing: ECMP to balance; dynamic bursts still imbalance → protection "switches"

Protective Load balancing (PLB): on congestion, change "flow label" in IP hdr → change H()

Wait until path goes idle, then reorder to minimize reording; move mice away from elephant flows

Traffic shaping: spreading traffic over time to constrain to constrain BW use by users; based on policies/business priorities (w/o shaping BW alloc via CC, no policy)

Traffic Pacing: spreading traffic over time; reduce bursts, queues/delay, packet loss; insert delays + window clocking/sliding to prevent bursts

Time wheel / Single time indexed queue: timestamp buffs; t determines rate buffs empty into Q on wheel; wheel to spin send — Timestamper: Earliest Departure time via policy

---

Quality of Service (QoS): priority of this traffic for allocating buffers & BW at each hops at < RTT time scales?

Issue: cong at link → how alloc rsrcs? (weighted fair Q) Rank: Sender app ≥ priority class (latency, thruput...) ≥ QoS (encoded in hdr) ≥ 9 (mod, 10) ≥ 1: Buff, 2: BW

☆ QoS only matters when switch port is 100% utilized when packets arrive ☆ QoS priority policy under cong.; MUXing priorities

---

(ex) Timing wheel: class A rate 1 Gbps, B: 0.4 Gbps. Last packet A @ t = 100 μs = B. Packet size 1500B; traffic class allowed 1 packet in q

Horizon (time to cycle thru Qs) = 4, "now" = 3rd Q, time count = 110 μs, "now" last moved at 107 μs

① Max rate 1.5 Gbps for any traffic Q → time granularity (amount of time covered by each q)? Max packets/sec = m = $1.5 \times 10^9 \times (8 \times 1500)$ min in q = $\frac{1}{m}$ = 8 μs

② Min rate supported? Q 1 packet per horizon, min rate = (1500 × 8) / 4s = 3 Kbps ☆ slowest we can send out, num Qs unknown

③ How many queues are there? # queues = horizon / granularity = 4s / 8 μs = 0.5×10^6 — 107 + time granularity = time when next Q selected

④ Packet ready in A; timestamp will it get? which queue? Timestamp = 100 μs + $\frac{\text{pkt\_size}}{\text{rate}}$ = 100 μs + $\frac{1500 \times 8}{1\text{ Gbps}}$ = 112 μs vs. 115 μs, 112 falls in 3rd

---

Wireless shared, attenuating signals w/ distance, env. change rapidly, packet collision hard to detect; multipath fading, path loss, shadowing (reflections, obstructions)

Medium access controls: mul. devices sharing transmission CSMA (listen for others, don't transmit if they are)

Hidden terminal problem: A → B ← C, A & C out of range to each other; A cannot sense C collision → CSMA fails (A(B)C) → RTS/CTS

RTS/CTS (req/clear to send): broadcast medium for all A ⇄ B (RTS, CTS, Data) $\xrightarrow{CTS}$ C; carrier sense (wait for transmissions to finish)

Exposed terminal problem: B → A, C → D, but B & C cannot transmit at same time (A (B C) D) → MACA (back off unfair) → MACAW

Cellular Introduce ① Discovery: which tower to connect to ② Authentication: provide service? ③ seamless comm: no disrupt ④ Accountability: rsrc limits using plans

Radio Access Network (RAN): collection of towers (given set of freq); Cellular core: RAN ⟷ Radio gateway ⟷ mesh ⟷ Mobility mngr (control functions) ⟷ DB (subscriber info) ⟷ Packet gateway (cellular network & internet boundary)

☆ IMSI: unique ID for subscrib ☆ IMEI: physical device ☆ MSISDN: phone #

① Registration (DB stores IMSI, secret key) ② Discovery (tower const. broadcast, pick tower via band scan + best)

③ Attachment (IMSI + attach req → (radio GW) mobility manager → config data plane → record in DB) ④ Roaming (home routing vs. local breakout over)

⑤ Handover: closer to new tower → disconnect; co-op btw tower, + user device, mobility mngr, radio gateway ☆ want highest RSSI (closest distance)

TCP throughput = $\sqrt{3/2}\,(\text{MSS}/(\text{RTT}\sqrt{p}))$