

CS 118 Midterm Cheat Sheet

Physical Layer

Layer Violation - layers may only access/view their own headers/layer content
easy way to identify: if a header of a lower layer is changed, it should not impact current layer service - if it does there is a violation

ECN - Explicit Congestion Notification - added to IP & TCP to inform source abt congestion and to decrease sent packet rate

Bandwidth/Frequency (B) = $F = 1/T$ hz

where T is period and F is frequency

Intersymbol Interference (ISI) - interference between the lag of the previous symbol and the next symbol

Nyquist Limit = $2B$ bits/s

you can bypass Nyquist limit by sending on different phases or frequency

Baud Rate = $\log_2 L \times 2B$ bits/s

where L is the number of signal amplitudes

Shannon Bound = $B \log_2(1 + S/2N)$

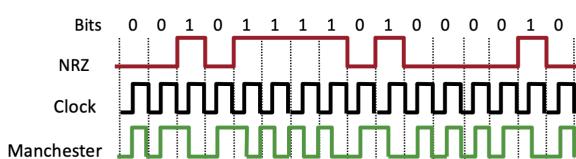


Nyquist-Shannon Sampling Thm - Anti-aliasing iff $f_s > 2f_{\max}$

where f_s is sampling freq and f_{\max} is og max freq

Synch. Clock Recovery - signals require preamble w/ transitions to reduce receiver clock overhead when sampling synchronously

Manchester Encoding - encodes bits to transitions at mid bit width: 1:hi \rightarrow lo, 0:lo \rightarrow hi
con: 50% efficient - encodes only half bit per transition



Alternate Mark Inversion (AMI) Encoding - encodes bits to alternating voltage levels: 0:0V, 1: $\pm V$. Each bit alternates positive and negative voltage.

e.g. 11100111 \rightarrow +-+00+-+

con: issues with long seq of 0s

4-5 Encoding - encodes 4 bit seq to 5 bit seq w/ transisition

e.g., 1111 \rightarrow 00001, mitigates long preamble

con: introduces new overhead for every 4 bit pattern

Broadband Encoding - Frequency Shift Keying (FSK), Amplitude Shift Keying (ASK), Phase Shift Keying (PSK)

not limited to energy levels like baseband encoding above

Signal Demux - Time/Freq/Phase Division Mux (T/F/PDM)

e.g., tv channels - signals muxed by frequency of signal

Twisted Pair Cable - low bandwidth, cheap \rightarrow Cat 5 twisted pair higher quality

Coax Cable - high bandwidth, og ether, too clunky replaced by Cat5

Fiber Optic Cable - huge bandwidth, unidirectional, but chromatic and modal (bounce) dispersion, expensive, multichannel via multicolor but expensive with prism to demux color

channels

Wireless 802.11b - broadband, requires spectrum allocation, possibly satellite, radio large passes through objects

Medium	Speed	Distance Span	Pros	Cons
Twisted Pair	1 Mbps - 1 G (Cat 1 – Cat 5)	1 – 2 Km	Cheap, easy to install	Low distance
Digital Coax	10-100 Mbps	1-2 km	broadcast	Hard to install in building
Analog Coax	100-500 Mbps	100 Km	Cable companies Use it now	Expensive amplifiers
Fiber	Terabits	100 km	Security, low noise, BW	No broadcast, Needs digging
Microwave	10-100 Mbps	100 km	Bypass, no right Of way need	Fog outages
Satellite	100-500 Mbps	worldwide	Cost independent of distance	250 msec delay Antenna size
RF/Infrared	1 – 100 Mbps, < 4 Mbps	1 km 3 m	wireless	Obstacles for infrared

Data Link Layer

Flags - wrap datagrams to fragment into frames, signify start and end

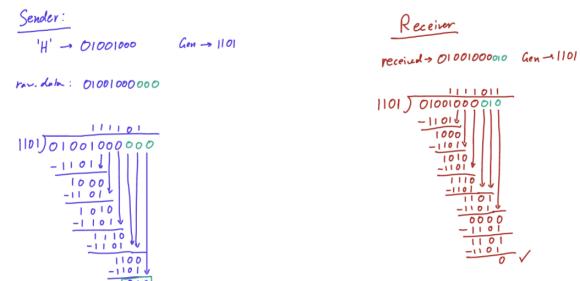
HDLC - bit stuffing for false flags, no escapes

PPP (Ethernet) - byte stuffing, with escapes

Stuffing Overhead - #stuffed bits / #og bits

Stuffing Efficiency - Probability of stuff = #flags / #bit combs/patterns

CRC32 Mod2 Div - shift left by `len(gen)-1` then long divide generator, xor only for leading



1s, if leading 0 → move right until leading 1

$$\text{CRC-16: } X^{16} + X^{15} + X^2 + 1 = 110000000000000101$$

We skip proofs of these properties this quarter but they are in your notes, Not required for HWs and tests.

Odd bit errors: can handle but not a big deal as parity can handle with using just 1 bit. 1

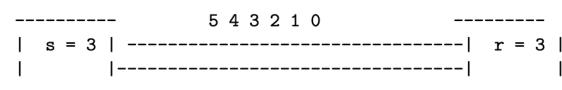
Two bit errors specially designed CRCs can do this.

Beats parity!

Burst errors: CRC-32 can catch any 32 bit burst error for sure. Further it can catch larger burst errors with very high probability: $(1 - 1/2^{32})$

Summary: So the big deal is that it can for sure catch up to 3 bit errors, and can detect **any** error with very high probability. Like a hash function with some deterministic guarantees

Band Invariance - sender and receiver will always be within $x+1$ packet ids of each other. Receiver state is id of packet waiting to receive, ack is id of receiver state



Sender state updates with ack

Throughput (bits/s) - jobs/s (usually round trip)

Latency (s) - worst case time to complete 1 job

1-way propagation delay (s) - time for transmitted bit to cross link

Transmission Rate (bps) - bit rate over link bits/s

Pipe Size aka Bandwidth Delay Product (bits) = Transmission Rate \times Round-trip propagation delay

Sliding Window Ack Protocol

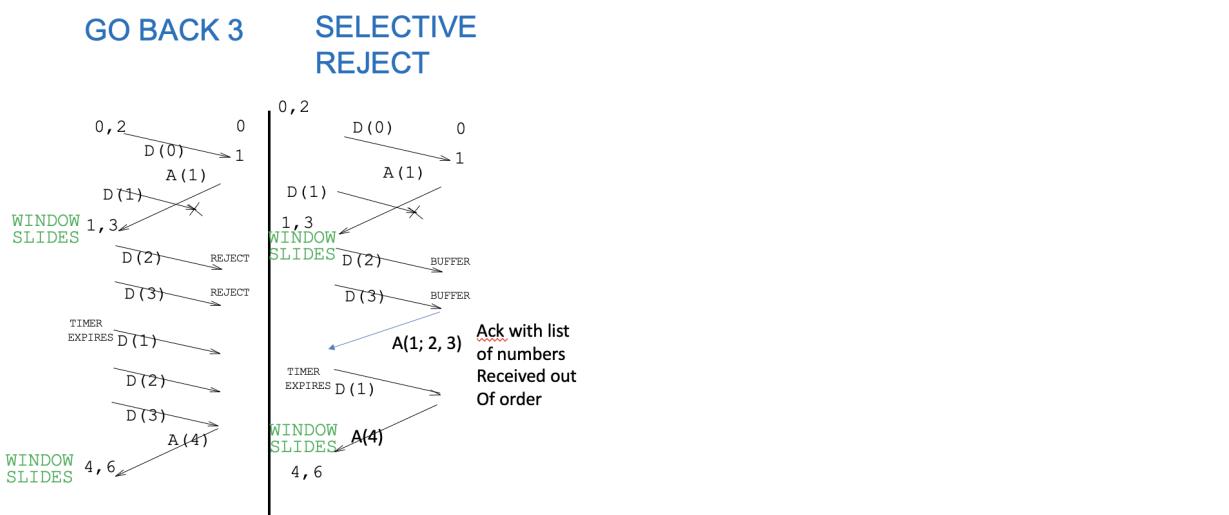
only fifo packets with ordering, fails on UDP

mod for packet ids depends on window size W (max number of packets sent in sequence)

alternating bit - mod 2

go back W - mod W+1

selective reject - mod 2W



Restart Signal - requires ids to mitigate multi restart

usually uses max frame time timer then restart - issue is must wait for timer so longer time to reboot

LAN

Ethernet - multi-access many-to-many, demux via MAC addr

01010111 preamble	Dest (6)	Source (6)	Length (2)	Data	Pad	CRC
----------------------	-------------	---------------	---------------	------	-----	-----

Total Frame length

64<=L<=1500

Strict Multiplexing (B/N) - allocate static bandwidth via TDM/FDM

Stat. Mux.(B/x s.t. x < N) - allocate bandwidth based on traffic

allows clients to use others' bandwidth when low traffic

CSMA/CD - Collision detection via carrier sense - stations must listen and detect collisions occurring at their station and propagate info to all stations

Ethernet uses min frame size of 64 bytes = Pipe size = Trans. Rate (10 Mbps) \times RTPD ($51.2\mu s$) = 512 bits

Limits cable length if link has higher transmission rate

Collision detection via voltage, if high is 1, avg volt for 0 or 1 is 0.5V, collision would cause avg volt of 1V

Jam bits during collision to extend collision to be detected by other stations

Binary Exponential Backoff - wait longer time for more collisions

Choose wait time after k collisions from $2^k - 1$ time slots

e.g., after 1st, choose 0 or 1 wait, then 2nd choose 0 to 3 wait

Hubs - Single point of connection for all nodes on ether, requires CD

Wireless 802.11b

Multichannel - 12 allocated channels, 3 orthogonal channels at a time

Stations can be on orthogonal channels so CD wont be detected if on diff channel

RTS/CTS (MACA) - before node A transmits, send couple bytes called Request-to-Send on all channels, node B hears and calls Clear-to-Send broadcast → node C hears and defers

Bridges/Switches

MAC Addr - 6 bytes (48 bits)

Unique to device, unique to terminal (rec/send)

first 3 bytes for vendor, last 3 host

MSB leading 1 → multicast

IPv4 - 4 bytes (32 bits)

allocated per network via DHCP

Accessible IPs mapped via DNS

leading 1 (or `1110`) signifies multicast

Switches - 1-to-many, point to point, buffer frames if link is busy

Entries in DB by looking at Src addr (up)

Flood down if dest not known

No loops → tree topology

Timer for buffering, timer expires → flood all buffered frames

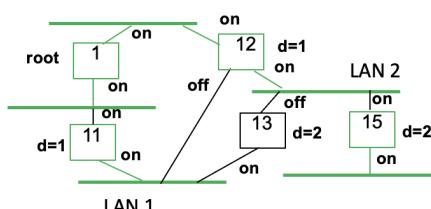
Switch Transparency - bridges must be transparent to nodes, must appear as simple ether/cable

Promiscuous Receive - Switches buffer from all stations regardless of src

Flood - Forward to all stations on line, picked up by correct MAC

Filtering - filter packets by ether header for forward or buffer

Bridge Spanning Tree Algo - bridge ids, drop longer links to same LANs



- Root is Min ID node (in this case Bridge 1)
- Other bridges finds Min port, port through which it has shortest path to root (parent), For 11 it is upper port.
- Each bridge also finds the ports for which this bridge is on the shortest path between root and corresponding LAN: Designated Ports. For example, 11 and 12 have d=2 for LAN 1, so we pick shorter ID as tiebreaker, Bridge 11 is designated bridge for LAN 1, 12 for LAN 2
- Each bridge turns ON Min port and all Designated Ports. ON,OFF are software states: always receive hello and management messages on all ports. Drop data packets to/from OFF port.

07 – Routing

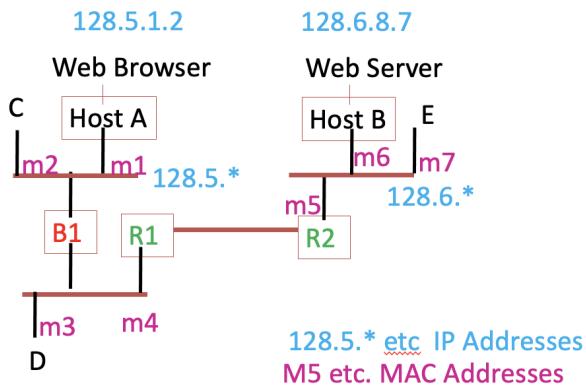
Bridge to Routers

- Data link headers came before data link relays (bridges/switches) so they had to adapt
- but routers were first-class creations. The debate is whether to use bridges universally or routers
- the big problem is no loops in switch topology → need possibly many intermediaries to communicate despite having fiber or really good hardware between stations
- switches also cannot handle **address incompatibility** (HDLC vs Ethernet (PPP)), **packet size incompatibility** and **bandwidth incompatibility** (FDDI vs Ether)

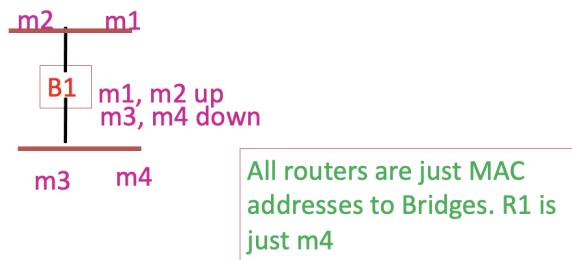
Abstraction

- Bridges extend LANs to extended LANs
- Routers connect extended LANs to WANs
- Consider the following example

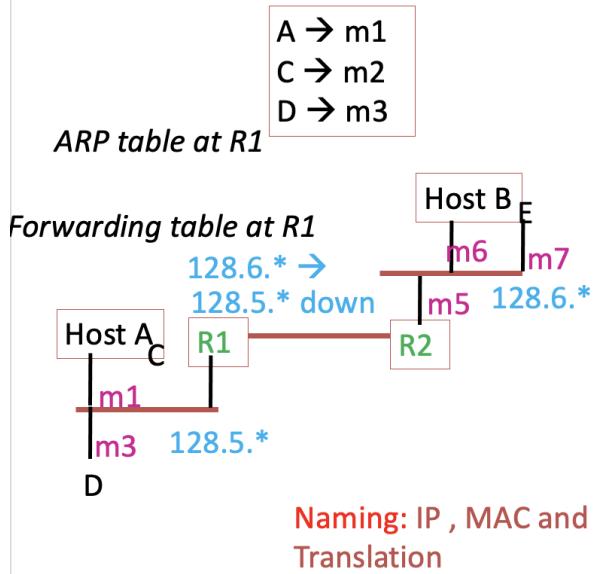
Full Topology



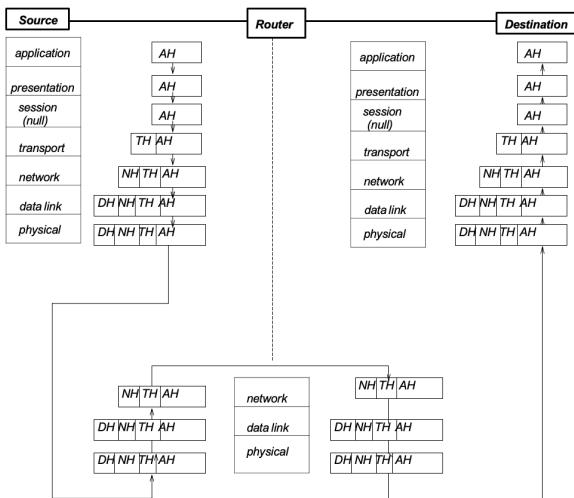
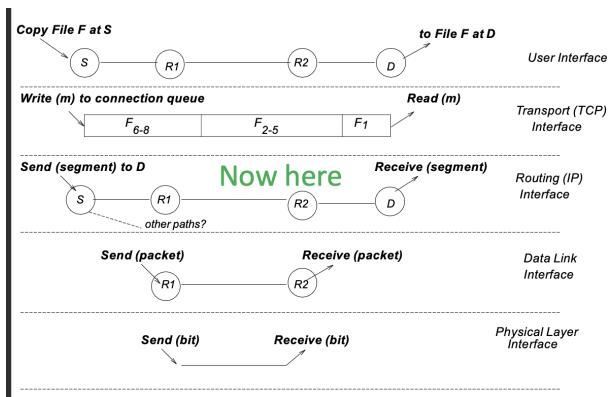
Bridge View



Router View



OSI Overview



IP Forwarding

Terms

- ISP - Internet Service Provider, usually local with regional POPs
 - usually small ISPs connect with mega ISPs then to NAPs

- POP - Point of Presence, a physical location with a link that ISPs usually have per region
- Autonomous Systems - network managed by 1 manager, entirely contained in a LAN or alike
- NAP - Network Access Point, single connect for all ISPs
 - legacy, very cluttered and congested, instead peering
- Peering - Interconnect b/w ISPs w/o NAPs
 - megacorps like google, meta have their own WANs and peer with everyone with POPs everywhere

Internet

- IPs goal was to interconnect (internet) different network like DECNET, SNA, XEROX Net, Apple Talk
 - eventually all disappeared and IP works directly
- error message backward using ICMP (protocol)

History

- 1970s, ARPANET - linked govt and university ite (UCLA) in 1970s
 - was shutdown by gov too risky
- 1983, NSFNET - 1983 ARPANET splits up into MILNET and ARPANET. In 1984 NSF establishes NSFNET to be backbone. Campuses attached to backbone via regional networks (NYSERNET etc.) Strict hierarchy breaks down because of direct connections between providers
- late 1980s - multiple providers

Domain Name Server (DNS)

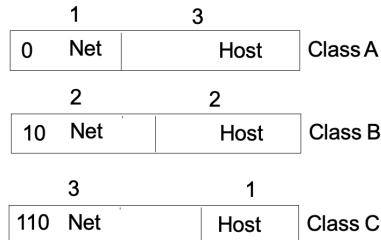
- servers that map domain names (urls) to 32-bit (5-byte) IP addrs
- hierarchical, local DNS knows translations for all in network devices, then subnet IPs and move up the hierarchy for wider prefixes
- then there are root DNSs which store common IP translations, e.g. google DNS IP: 1.1.1.1

DHCP Server

- dynamic host control protocol
- when a station or node connects, it multicasts to DHCP server which allocates a local IP for the node, the prefix tells us the router IP
- this is slightly different when considering local vs public IPs

Original/Old Model

- small number of large networks (class A), moderate number of campus networks (class B),



- Find Dest - parse Network number of dest addr and check for class of addr
- Final hop reached? - if network number of dest = network number of this router' local interface(s) then deliver packet. Map to local address using ARP

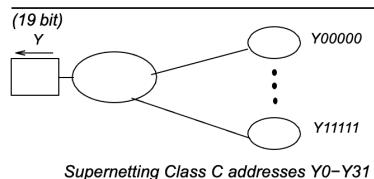
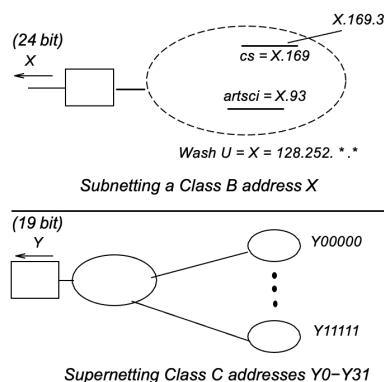
- lookup router table - lookup network number in routing table, if exists → forward. else → send to default router (e.g., many addrs in campus network → have default router and then route internally in campus network)

Challenges

- inefficient address usage - any org that need >255 addrs needed class B → quickly ran out of class B addrs
- routing table growth - response to above → allocate more class C addrs → each core router needed much larger routing tables
- sol - change IP forwarding to longest matching prefix

Subnetting/Supernetting

- slash at the end of an IP tells us how many of the prefix bits to consider to route
- this simplifies comms between networks/LANs by only considering the prefix that matters because all network devices within the network have the same IP prefix
- e.g., 128.32.0.0/16 ⇒ all network devices share the same first 16 bits 128.32
- can encode with slash bits or subnet mask using bitwise & between ip and mask - use 1s for all leading bits that matter i.e. in ints: 255.255.0.0 ⇒ [0-255].[0-255].x.x

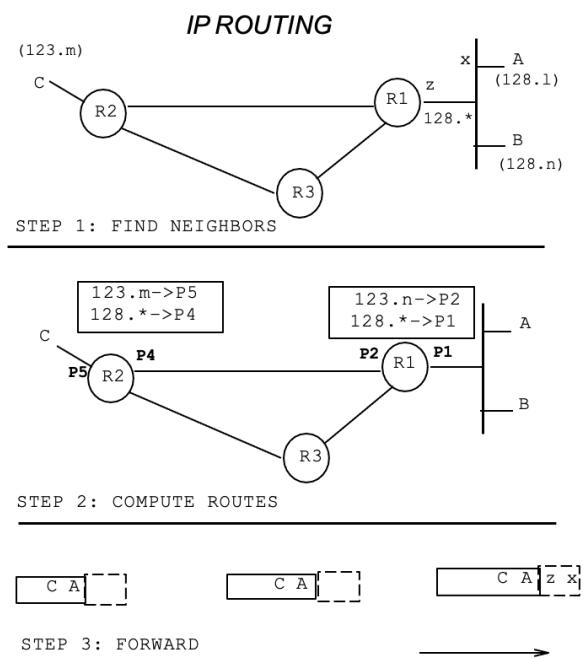


- **Supernetting:** Done recursively, leads to backbone routers only having hundreds of thousands of prefixes of lengths 8-32
- **Temporary Measures:** Often today new organizations are give 1 IP address and use NAT. Need the move to IPV6 (128 bits)

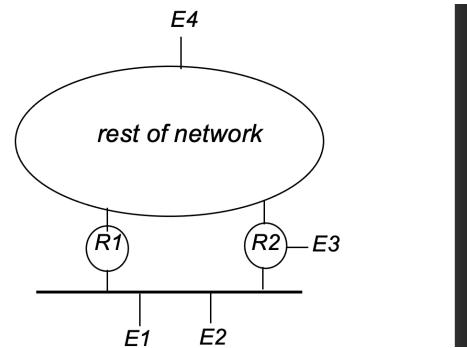
New IP Forwarding

- CIDR - classless inter-domain routing - no more fixed length prefixes (IP classes from original model)
1. lookup - find longest matching prefix P of dest IP addr from forwarding table
 2. default or local
 1. if P is nil → forward on default
 2. elif P is associated with local interface → deliver and map to local addr using ARP,
 3. else → forward to next hop associated with P
- router internally has a crossbar switch which is used to bridge packets to the correct data link terminal. i.e. line

- the forwarding table maps ip prefixes to output links, this is constructed by routers comms with each other telling them that ips are on a specific direction → maps to specific data links → maps to specific mac addrs/ethernet terminals
- NOTE: IP does not yet know the mac addrs it needs to send → see prob and sol below

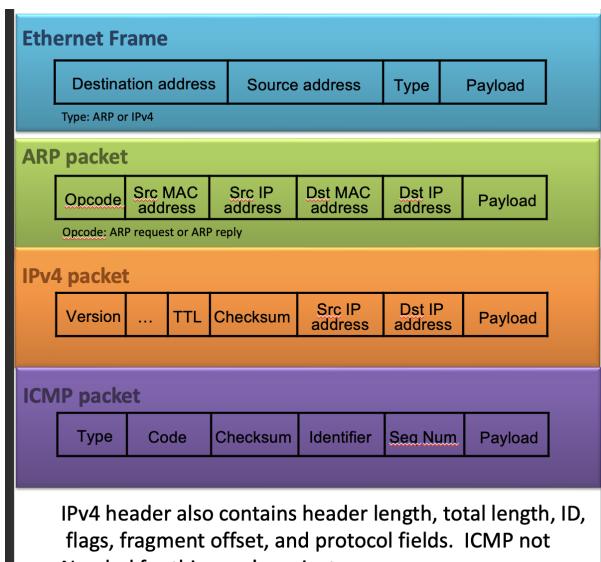


IP Solution to End-node Problem



- Given top.
- Prob1 - Routers need Data Link Addresses of endnodes
 - Sol1 - ARP for MAC address of destination
- Prob 2 - Endnodes need DL address of 1 router
 - Sol2 - a service called DHCP gives you the IP address of one router (auto-configuration)
- Prob3 - E1 and E2 should be able to communicate without a router
 - Sol3 - two endnodes know they are on same subnet by comparing masks. Then ARP
- Prob4 - E1 to E3 traffic should go through R2
 - Sol4 - send to router and router sends redirect if packet returns on interface it entered router. (Ignore this code in project),

Routing Packet Structure



ARP

- sends signal on connected mac terminal to lan, router picks up and propagates back its mac addr, now client knows which router to send to the next hop
- ARP, or Address Resolution Protocol, is a network protocol used to map an Internet Protocol (IP) address to a physical machine address that is recognized in the local network. This is particularly important in IPv4 networks, where devices communicate using IP addresses, but the actual data transmission occurs over the physical network using MAC (Media Access Control) addresses.

How ARP Works:

- ARP Request:** When a device wants to communicate with another device on the same local network, it needs to know the MAC address corresponding to the target device's IP address. If the sender does not have this information in its ARP cache (a table that stores IP-to-MAC address mappings), it broadcasts an ARP request packet to all devices on the local network. This packet contains the sender's IP and MAC addresses, as well as the target IP address for which it is seeking the MAC address.
- ARP Reply:** All devices on the local network receive the ARP request, but only the device with the matching IP address will respond. This device sends back an ARP reply, which includes its MAC address. The reply is sent directly to the sender's MAC address.
- Updating ARP Cache:** Upon receiving the ARP reply, the sender updates its ARP cache with the new IP-to-MAC address mapping, allowing for faster communication in future interactions without needing to broadcast another ARP request.

ARP Cache:

The ARP cache is a temporary storage area where the mappings of IP addresses to MAC addresses are kept. Entries in the ARP cache can expire after a certain period, requiring the device to send a new ARP request if it needs to communicate with that IP address again.

Types of ARP:

- Proxy ARP:** This allows a router to respond to ARP requests on behalf of another device that is on a different network. This can help devices communicate across different subnets.
- Gratuitous ARP:** This is a type of ARP request sent by a device to announce its IP address to the network. It can be used to update other devices' ARP caches or to detect

IP address conflicts.

Security Considerations:

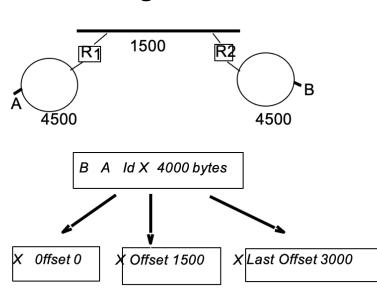
ARP is inherently insecure because it does not include any authentication mechanisms. This makes it susceptible to attacks such as ARP spoofing, where a malicious actor sends false ARP messages to associate their MAC address with the IP address of another device, potentially allowing them to intercept or manipulate network traffic.

Conclusion:

ARP is a fundamental protocol in networking that enables devices to discover each other's MAC addresses based on their IP addresses, facilitating communication within local networks. Understanding ARP is crucial for network configuration, troubleshooting, and security.

Bandwidth Incompatibility (Path MTU)

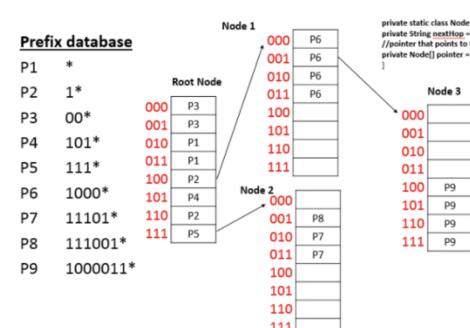
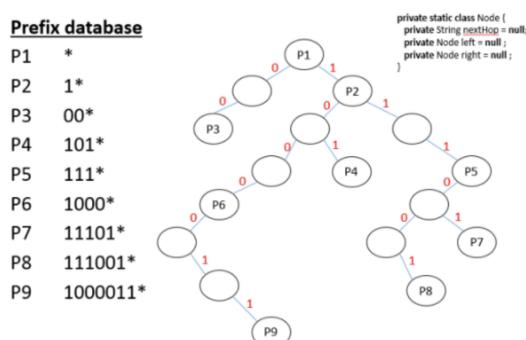
- sps packet is larger than the data link bandwidth



- old IP said to fragment and reassemble but was too expensive
 - modern end node find right size known as **Path MTU** and sends request for this size instead of asking routers to fragment

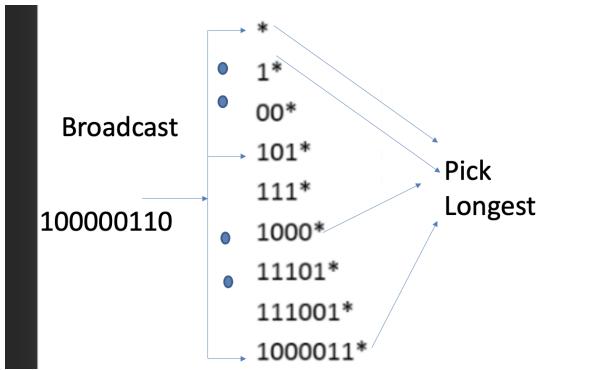
Fast IP Lookups

- Sol1 - Use unibit prefix trie as lookup table - too slow, 32 steps in worstcase



- Sol2 - Multibit trie - too slow and tm memory

- Sol3 - ternary CAM (content addressable memory) - memory where each bit can be 0, 1, or * that can be searched in parallel because we search by content instead of id or position- but requires tm power at high speed access, low longevity of memory



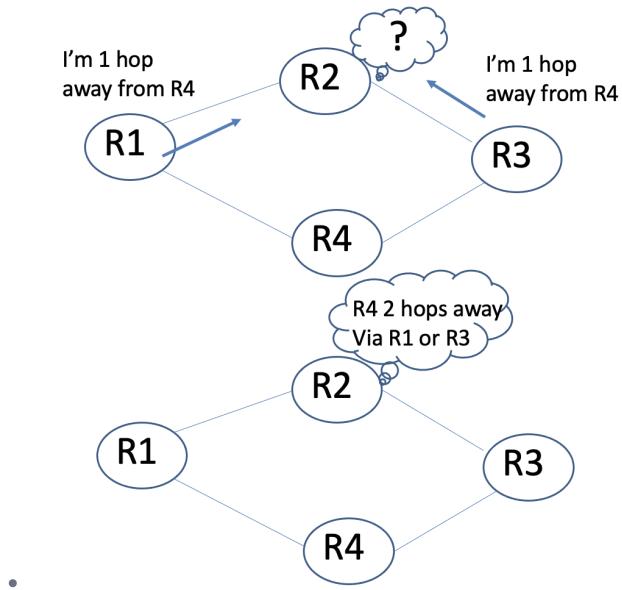
- diff router size ue diff types: compressed multi-bit tries, ternary CAMs

Route Computation

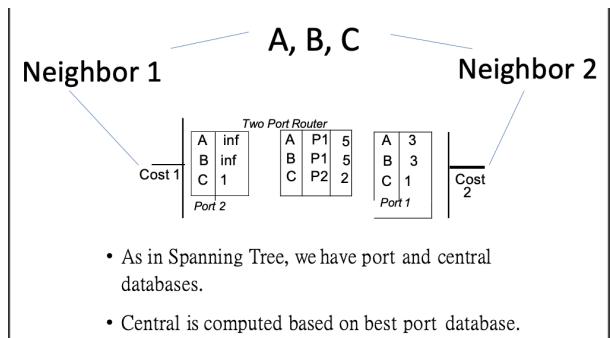
- Flavors:
 - Intradomain routing - within an autonomous entity
 - Distance vector - problems with count to infinity
 - Link State - often used
 - Interdomain routing - between ISPs

Distance Vector, Gossip

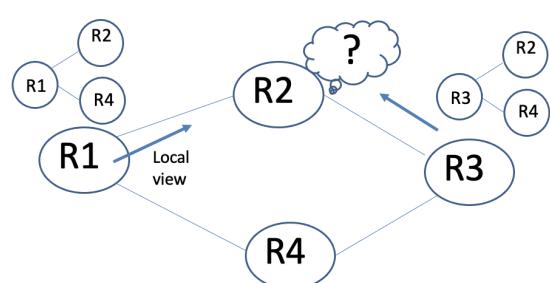
- Routers begin with comms between themselves and figure out neighbors and hops away from routers >1 hop → propagate
- bad if a router fails then no way to know neighbors until they tell u they are there so assume the are and lead to infinite hop count
 - consider that each router stores a routing table with distance to each router (via rumor, we don't actually know the state), sps the following link where $A \leftrightarrow B \leftrightarrow C$ where $B \leftrightarrow C$ fails. Then, C's table is cleared, B realizes it can't reach C directly, but looks at other neighbors and sees A can reach C (A sets the distance to C as dist to B + dist to C), so B updates with A's DV when it's actually not true. Because B's table changed, it propagates changes to neighbors. Now A realizes B's dist to C changed so it updates its count → infinite count on both A & B.
- Now all vectors know their distances to all other routers



Distance Vector (DV) Database

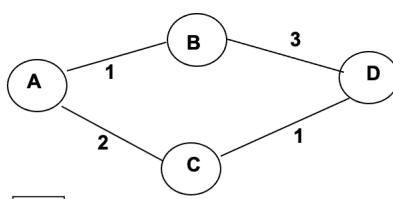


Link State



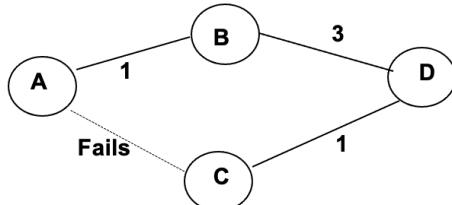
- each router store its nuclear neighbors as map
- Then each node floods its LSP to all other nodes
- now use any shortest path - Djikstra
- broadcast local state via Link State Packets (LSPs)
- but this causes underutilization - Djikstra enables only shortest path → many links of high bandwidth unused, so we should compute routes using distributed algos

LSP Generation



LSP
A
B
1
C
2

which means:



LSP
A
B
1

which means:

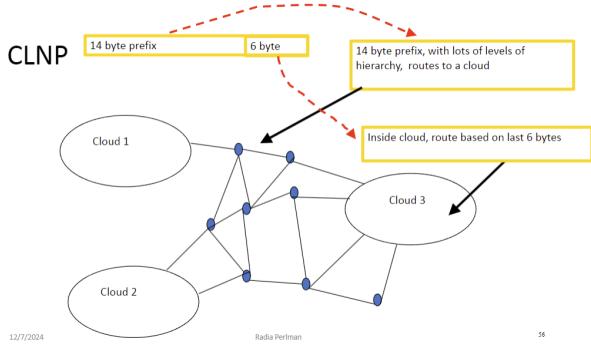
- If link AC fails, neighbor discovery in A and C will eventually detect failure.
- Only A and C recompute their LSP values and broadcast their LSPs again to all other nodes. Other nodes do not recompute or rebroadcast their LSPs.
- on failure

Wide Area SDN

- Google B4, Microsoft SWAN, Amazon SDWAN
- within a WAN, just send all packets to the Central SDN controller instead of finding the shortest path
- just forward all packets to SDN and run an algo to find best distributed route, forget protocols

CLNP: IP+Ether Alternative

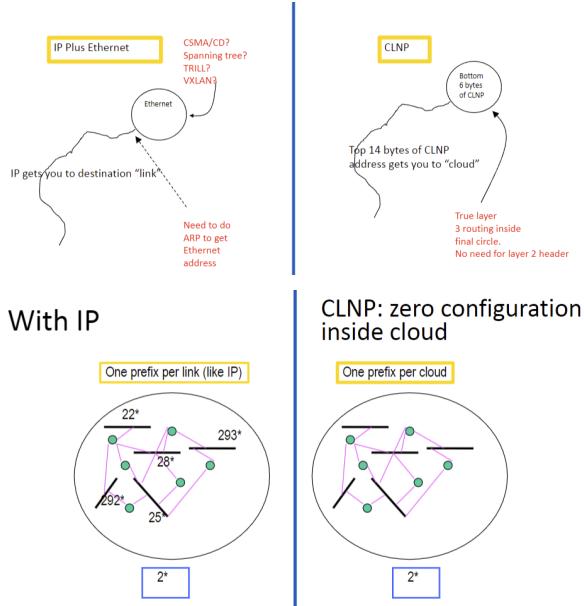
- redundancy and issues with Ethernet (Layer 2) + IP (Layer 3)
 - Ethernet was meant to be configuration free → 6 byte addresses even just to communicate to fewer nodes on the same link
 - Ethernet never meant to be forwarded, but now it is
 - Ethernet requires MST, no loops
 - IP is HEAVY on configuration and nodes moving readdresses and susceptible to changes
- CLNP – 20 byte addresses per host, shared 14 byte prefix per "Cloud" (the organizational structure, analogous to AS (Autonomous Systems)), node-specific 6 byte suffix



12/7/2024

36

- "ES-IS" protocol where nodes announce themselves to the routers
- CLNP vs current:

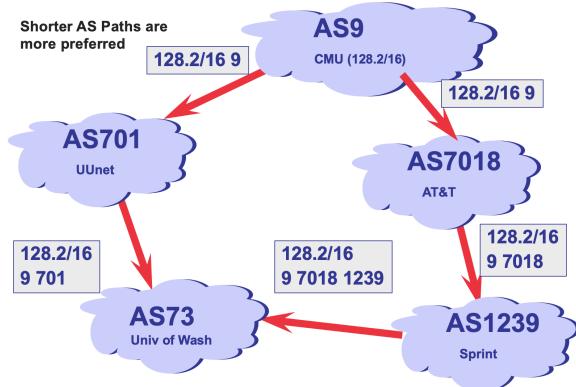


- advantages
 - no need for dhcp or extra configs per link
 - allows for loops between links and routes
 - no need for NAT (address space is large enough for each node to have its own address, no way there are millions of nodes in a single cloud)
- why it wasn't adopted
 - people didn't like that it didn't follow OSI
 - we still had some IPv4 addresses left, so switch to IPv6 instead of CLNP
 - There was a TCP already built on top of CLNP

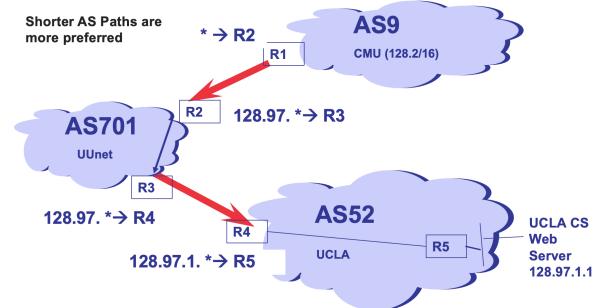
08 – Border Gateway Protocol

Border Gateway Protocol (BGP)

- border router \Leftrightarrow edge router
- protocol for inter-AS comms (AS = Autonomous System)
- ASes have AS ids bc ASes may have many prefixes
- routing is done hierarchically with shortest AS paths



- hierarchical routing because not all routers store routes to all ASes or even networks, so make routers hierarchical and AS edge routers just route all to root/major routers



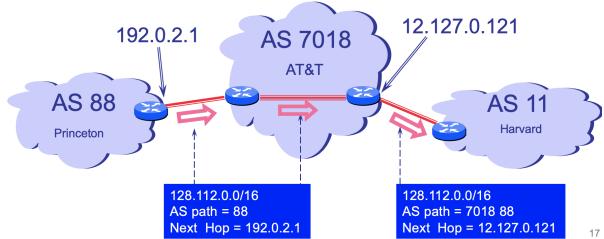
which know dest routing and proceed with hops

- however, because weaker edge routers don't know abt all domains/ASes, they use the BGP to optimize routes without knowing abt all paths
 - BGP uses path vector protocol instead of distance vector to know shortest path instead of storing all possible distances
- e.g., only allow govt packets through ARPANET, if don't know the dest domain \rightarrow just forward to 701 (image above)
- multihoming - multiple ISPs service the domain
- peer-to-peer - usually bw ISPs to share paths

BGP Session

- basic operation steps:
 1. Establish session
 1. requires TCP connection between edge routers of 2 domains
 2. exchange all active routes
 3. while connection is true, exchange updated routes
- nodes learn multiple routes between domains and store in a routing table - w/ incremental updates

- routing packets (not packet forwarding) sent to fill routing table
 - Destination prefix (e.g., 128.112.0.0/16)
 - Route attributes, including
 - AS path (e.g., "7018 88")
 - Next-hop IP address (e.g., 12.127.0.121)

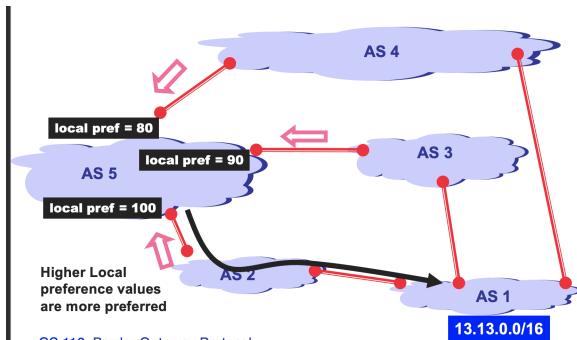


17

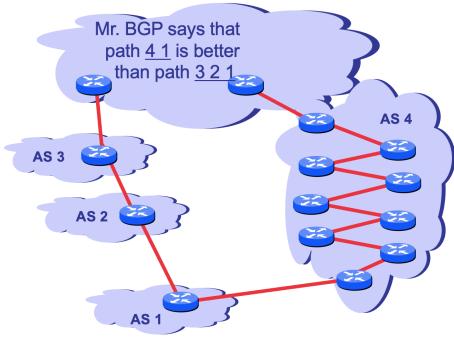
- generally speaking, most of the time just dump to IP which knows AS paths to get to dest
- edge routers only need to know next hop addr
- once packet is on the line, it ARPs to get MAC of router and propagate
- origin - route from inside (IGP) or outside (EFP)
- local pref - stat ranked paths within AS (preferred entry)
- multi-exit discriminator - decide which router to exit from
- community - opaque data used for tag routes that are treated equivalently?

BGP Decision Tree

- Default decision for route selection
 - Highest local pref, shortest AS path, lowest MED, prefer eBGP over iBGP, lowest IGP cost, router id
 - prefer eBGP over iBGP bc eBGP is more direct edge router from AS to AS
- Many policies built on default decision process, but...
 - Possible to create arbitrary policies in principal
 - Any criteria: BGP attributes, source address, prime number of bytes in message, ...
 - Can have separate policy for inbound routes, installed routes and outbound routes
 - Limited only by power of vendor-specific routing language

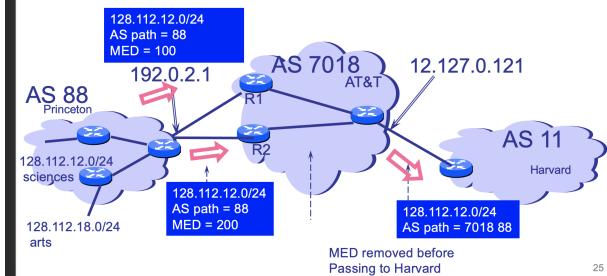


- CS 118: Border Gateway Protocol
- shortest path preference is AS context not router/hop context \Rightarrow greedy relative to AS paths but may not be optimal in number of router hops WITHIN an AS



Optimizations

- MEDs - router-level load balancing via MEDs to prefer router delivery



- example Cisco config to set MEDs

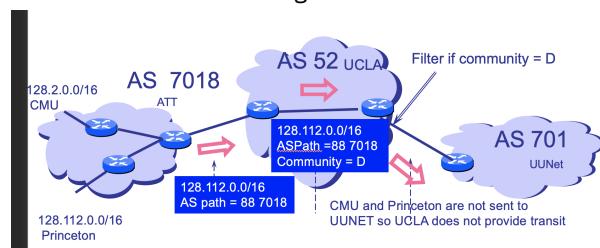
```
neighbor R1 route-map setMED-R1 out
neighbor R2 route-map setMED-R2 out

access-list 1 permit 128.112.12.0 255.255.255.0 //sciences
access-list 2 permit 128.112.18.0 255.255.255.0 // arts

route-map setMED-R1 ... match ip address 1 set metric 100
// for R1 send science prefix with lower MED priority
route-map setMED-R1 ... match ip address 2 set metric 200
// for R1 send arts prefix with higher MED priority

route-map setMED-R2 ... match ip address 1 set metric 200
// for R2 send science prefix with higher MED priority
route-map setMED-R2 ... match ip address 2 set metric 100
// for R2 send arts prefix with lower MED priority
```

- community - way to tag multiple equivalent routes with same tag value
 - remote routers can filter via tag
 - e.g., NOTTRANSIT if not in ISP network
 - add community tag to routing packets which is the same tag for all ISPs the AS pays



- now create routes as sets of route via union

BGP Optimization Preference

- First Local Preference
 - ◆ Operator knows best
- AS Path Length
 - ◆ After that shortest path (roughly speaking) makes sense
- MED
 - ◆ Other things being equal, honor MED priorities
- eBGP over iBGP
 - ◆ Other things being equal, a route from an external border router makes more sense than one from an internal router
- Shortest IGP weight (from Link State, or Distance Vector)
 - ◆ Other things being equal, pick shortest cost to border router

BGP Drawbacks

- Instability
 - Route flapping (network x.y/z goes down... tell everyone)
 - Long AS-path decision criteria defaults to DV-like behavior (bouncing)
 - Not guaranteed to converge, NP-hard to tell if it does
- Scalability still a problem
 - 500,000 network prefixes in default-free table today
 - Tension: Want to manage traffic to very specific networks (eg. multihommed content providers) but also want to aggregate information.
- Performance
 - Non-optimal, doesn't balance load across paths
- multi-homing gaming
 - extra reliability but vulnerable to gaming by switching ISP networks depending on cost
 - ISP usually charge at 95th percentile traffic/usage

BGP is Suboptimal

- Local knowledge only:
 - ◆ your neighbors best routes may not be your best
- AS Path Length
 - ◆ Does not measure real distance or latency
- Other Metrics
 - ◆ May care about cost etc. and have to hack BGP attributes
- New: Software Defined Networks within organizations
 - ◆ Google Espresso has BGP speakers but they send all BGP messages to a central cluster that also does measurements and picks more globally optimal route to customer ISPs
 - ◆ Read Google blog: Search for "Google Blog Espresso"
- Google Espresso, use central SDN to determine which BGP router to forward to external outside of WAN
- "hack" others' BGP by calculating latency across external WANs and store in central SDN to forward externally

[Optional] Scaling iBGP

- The default way of a full mesh between all border routers has $O(N^2)$ overhead, where N is # border routers

- Two common ways to scale IBGP in large ISPs: confederations and route reflectors
 - In confederations, we divide a large AS into stub AS's hierarchically, so stub AS's don't know internals of each
 - In route reflectors, leaf border routers send BGP messages to a central reflector that sends to all clients. Can generalize to a tree of reflectors.

When to Use Confederations

1. **Very Large, Hierarchical Networks:** Confederations are useful in large networks where the ISP has a very complex, hierarchical design,
2. **Administrative Control and Scalability:** Each sub-AS can have its own policies, making it easier to delegate control over different parts of the network.

When to Use Route Reflectors:

1. **Simplified Design for Medium to Large Networks:** Route reflectors simplify BGP by reducing the need for a full iBGP mesh without introducing the complexity of confederations. They're a good choice for ISPs looking to scale a network that isn't complex enough to justify a confederation.

09 – Naming

Overview

Protocols

- DHCP (Dynamic Host Configuration Protocol) - to get an IP addr
 - MAC → IP (given MAC, provision IP)
 - any node can request from multicast DHCP server → this means DHCP mac addr and IP begins with leading 1 (multicast)
- DNS (Domain Name Server) - map domain names to IPs
 - domain name ↔ IP
 - given from DNS, hierarchical until root DNS
 - root DNS (.com, .uk, etc.)
 - Top Level Domain (TLD) e.g. google, apple, etc.
 - sub domains - r.g., docs.google.com
- NAT (Network Address Translation) - proxy a network (multiple users) through a single IP
 - Private IP ↔ Public IP
 - simplest is 1:1 mapping
 - stored on local/private router
- ARP (Address Resolution Protocol) - map IP to MAC addr
 - IP → MAC (given IP, reply with MAC)
 - every router has a ARP table for translations

Names

- Domain/Host name - user readable, variable length, hierarchical
- IP addr - 32 bit, topologically unique, hierarchical (subnetting)
- MAC addr - 48 bit, fixed name, globally unique

Translation Patterns

- DNS -
- ARP -
- DHCP -

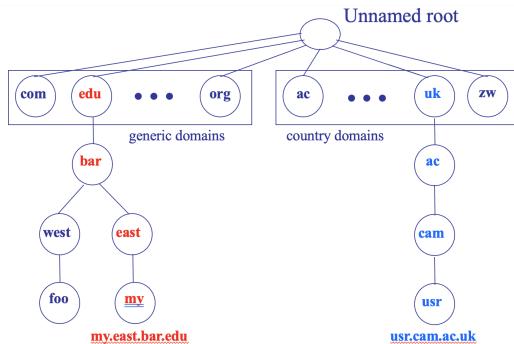
DHCP

- DHCP (Dynamic Host Configuration Protocol) - to get an IP addr
 - Broadcast-based LAN protocol algorithm
 - ◆ Host broadcasts "DHCP discover" on LAN (e.g. Ethernet broadcast)
 - ◆ DHCP server responds with "DHCP offer" message
 - ◆ Host requests IP address: "DHCP request" message
 - ◆ DHCP server sends address: "DHCP ack" message w/ IP address
 - DHCP request

DNS

DNS (Domain Name Server) - map domain names to IPs

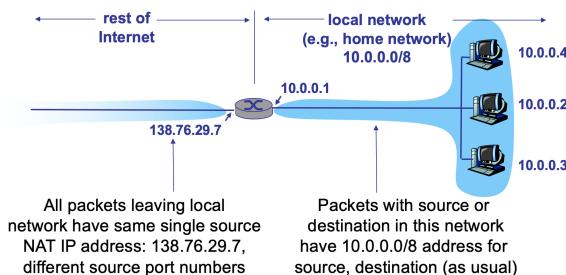
- Distributed administrative control
 - Hierarchical name space divided into zones
 - Distributed over a collection of DNS servers
- Hierarchy of DNS servers
 - Root servers
 - Top-level domain (TLD) servers
 - Authoritative DNS servers
- Performing the translations
 - Local DNS servers
 - Resolver software
- overview
- radix tree structure for DNS hierarchy (almost always cached after 1st request to root)



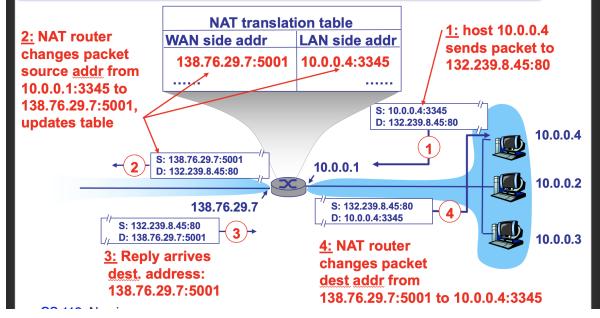
- DNS replicated to many servers, use UDP queries for robust requests, use exponential backoff for requests to the same server
- SD-DNS e.g., Akami server store shortest paths as SDN/CDN and propagate (similar structure to other SD-WAN)

NAT

- NAT (Network Address Translation) - proxy a network (multiple users) through a single IP
 - NAPT - Network Address and Port translation
 - PAT - port address translation
 - NAT encompasses all
- due to IP limits and flexibility to change ISP, local networks manage a private IP space using bogons
- bogons - IPs with prefixes that indicate private IPs e.g., 192.168 ... , 176.1 ...
- local routers (closest to private network) store private to public IP mappings in a table
 - route packets based on mapping
- simplest case is 1-to-1 mapping, in some clever cases map set of private IPs to smaller set of public IPs
- in some cases map all private IPs to 1 public IP



- example NATed network



- NApTed example
- NATs may be layer-violating as they may change existing protocols like FTP to support private address mappings (e.g., when IP in application data)

The trick to Many:1

- Mapping many private addresses to 1 public addr makes it impossible to anti-alias which actual device should receive the packet
- Hack is to use the **TCP Port** \Rightarrow 32 bit IP + 32 TCP ports = 64 bit address space
- enables **port forwarding** for private IP to be directly accessible over the internet
 - other forms of NAT traversal also possible

11 – TCP

TCP Overview

Tasks

- Muxing – UDP and TCP
- Reliability – TCP only
- Flow Control – TCP only
- Congestion Control – TCP only

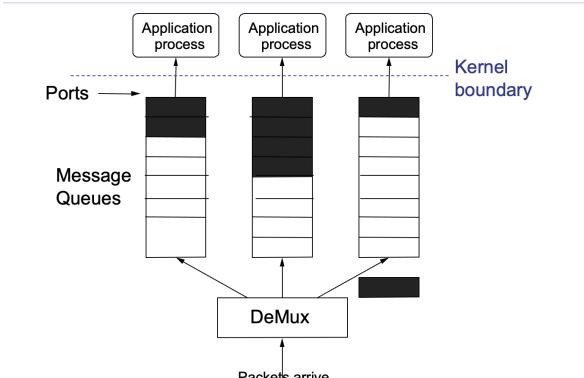
Process Abstraction

- transport protocol abstract processes to ports
- processes are identified by IP addr (32-bits), protocol, and port (16 bits)
- services abstracted to their required protocol: http (web), smtp (mail), dns, etc.

Port Selection

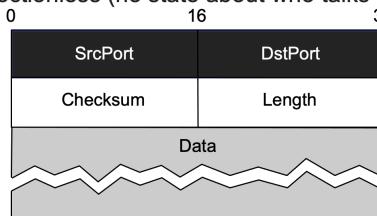
- OS assigns permanent ports to values ≤ 1024 e.g. http 80, smtp 25, dns 53
- OS allocates ephemeral (temp) ports to values above 1024

UDP – User Datagram Protocol



- only handles multiplexing
- no reliability in ordering, etc.
- no need for retransmit or acks
- used for low latency reqs e.g. streaming, dns, NTP (Network Time protocol - clock syncing), video games

- Connectionless (no state about who talks to whom)

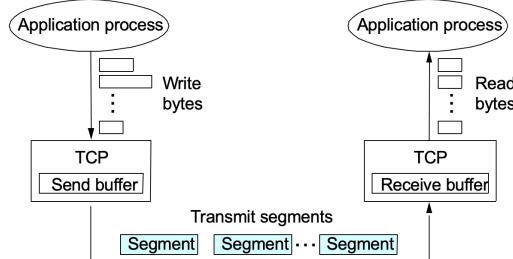


- "connectionless" packet delivery

TCP – Transmission Control Protocol

- Reliable bi-directional **bytestream** between processes
 - Uses a sliding window protocol for efficient transfer

- Huge sequence numbers (4-bytes) because of possibly very old out of order packets
- Connection-oriented
 - Conversation between two endpoints with beginning and end
- Flow control (generalization of sliding window)
 - Prevents sender from over-running receiver buffers (tell sender how much buffer is left at receiver)
- Congestion control (next lecture)
 - Prevents sender from over-running network capacity



- TCP delivery

Reliability - Data Link vs TCP

- Network instead of single FIFO link
 - packets can be delayed for large amounts of time
 - duplicates can be created by packet looping: delayed duplicates imply need for large sequence numbers.
 - packets can be reordered by route changes.
- Connection management
 - Only done for Data Link when a link crashes or comes up
 - Lots of clients dynamically requesting connections
 - HDLC didn't work: here more at stake, have to do it right.
- Data link only needs speed matching between receiver and sender (flow control). Here we also need speed matching between sender and network (congestion control)
- Transport needs to dynamically round-trip delay to set retransmit timers.

Similarity to Project

Similarities

- 3-way handshake with syn acks
- retransmission after 3 dup acks
- large seq num to handle retransmitted old packets

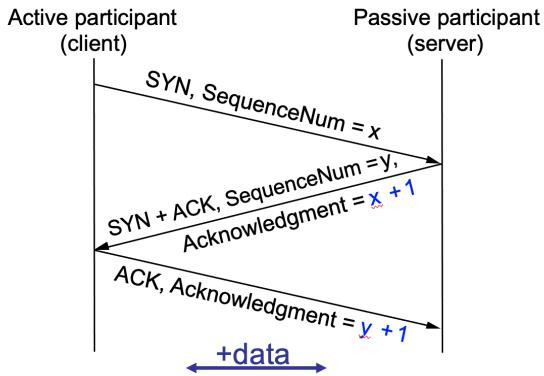
Differences

- retransmission timer set dynamically by calc round trip delay/time
- window is fixed but TCP calcs dynamically based on flow and congestion control
- selective reject support (TCP SACK) in addition to go back N

TCP Connection Handling

- nodes identified by 4/5-tuple
 $(\text{src IP}, \text{src PORT}, \text{dst IP}, \text{dst PORT}, \text{protocol})$
- seq number assigned per byte (allows for packet sizes to change between transmissions)

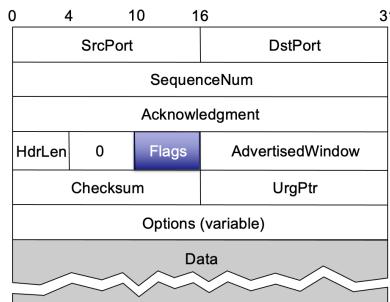
3-Way Handshake



- 3-Way Handshake with nonce seq
- We could abbreviate this setup, but it was chosen to be robust, especially against delayed duplicates
 - Three-way handshake first described in Tomlinson 1975
- Choice of changing **initial sequence numbers (ISNs)** minimizes the chance of hosts that crash getting confused by a previous incarnation of a connection
- How to choose ISNs?
 - Maximize period between reuse
 - Minimize ability to guess (why?)
 - Random works OK, as in Project 2
- Operation
 1. Server: If in LISTEN and SYN arrives, then transition to SYN_RCVD state, replying with ACK+SYN.
 2. Client: active open, send SYN segment and transition to SYN_SENT.
 3. Arrival of SYN+ACK causes the client to move to ESTABLISHED and send an ack
 4. When this ACK arrives the server finally moves to the ESTABLISHED state.

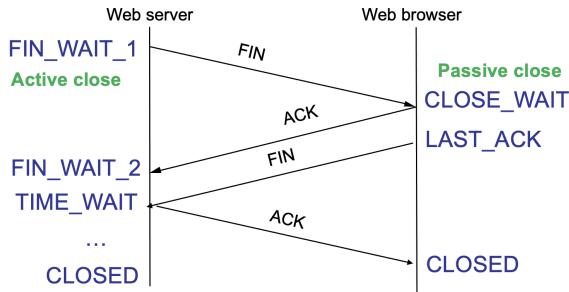
TCP Header

- src dest identified by IP + PORT
 - Flags may be ACK, SYN, FIN, URG, PSH, RST



TCP Disconnect

1. Need timers anyway to get rid of connection state to dead nodes.
2. However, timer should be large so that "keepalive" hello overhead is low.
3. If communication is working, would prefer graceful closing (so receiver process knows quickly) to long timers.
4. Hence 3 phase disconnect handshake After sending disconnect and receiving disconnect ack, both sender and receiver set short timers.



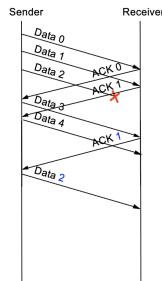
- enter **TIME_WAIT** to handle very old packets that may not have been retransmit or have not been acked yet
- We wait $2 \times MSL$ (maximum segment lifetime of 60 seconds) before completing the close
- Why?
 - ACK might have been lost and so FIN will be resent
 - Could interfere with a subsequent connection
- Real life: Abortive close
 - Don't wait for $2 \times MSL$, simply send Reset packet (RST)

TCP Reliability

- Usual sequence numbers except:
 - Very large to deal with out of order (modulus > 2 W etc. only works on FIFO links). As in Project 2
 - TCP **numbers bytes** not segments: allows it to change packet size in the middle of a connection
 - The sequence numbers don't start with 0 but with an **ISN**.
- Reliable Mechanisms similar except:
 - TCP has a quicker way to react to lost messages
 - TCP does a **crude form of selective reject** not go-back N
 - TCP does flow control by allowing a dynamic window which receiver can set to reduce traffic rate (next lecture)
- recall go-back-N: retransmit all packets from last ack
- real timer for retransmit is based on calculated **Round Trip Time (RTT)**

TCP Fast Retransmit

- like selective reject but immediate retransmit with sliding window buffer

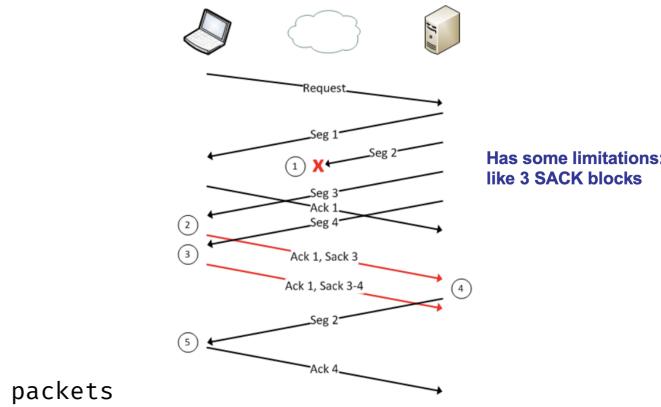


- timer as backup, use 3 dup acks to identify

- Don't bother waiting
 - Receipt of duplicate acknowledgement (**dupACK**) indicates loss
 - Retransmit immediately
- Used in TCP
 - Need to be careful if frames can be reordered
 - Today's TCP identifies a loss if there are **three** duplicate ACKs in a row. Project 2 did this!

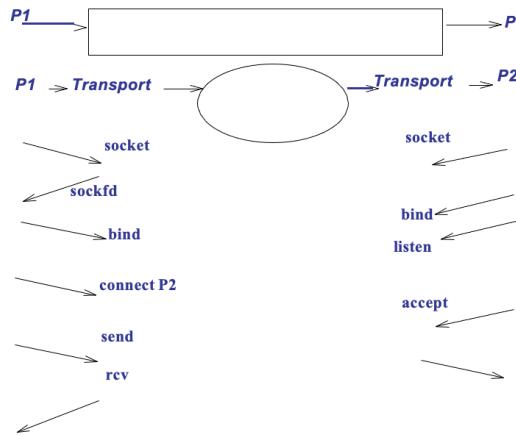
TCP SACK - Selective ACK

- ack with last received in-order packet, BUT also send **sack** with received out of order



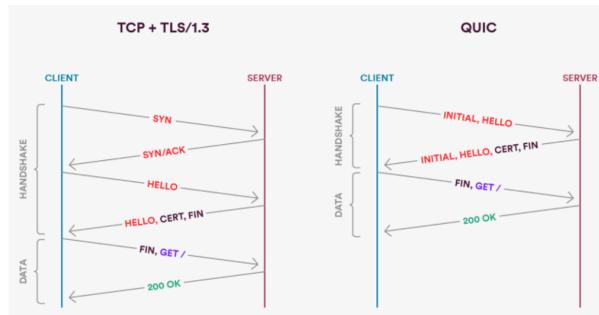
- limitations: ack/sack info could be very long if many drops and many out of order packets
- solution: limit to 3 sack blocks (ranges) - act like any out of order packets after reporting at most 3 sack ranges are considered to be lost
- e.g., ACK 1, SACK 3-8, SACK 10-15, SACK 17-25 \Rightarrow packets 2,9,16 are lost and any after 25 are considered lost or not sent yet

Socket API



QUIC - Quick UDP Internet Connections

- developed by google to make initial handshake faster with TLS



- Runs on UDP
- also adds other stuff like congestion and flow control for UDP

The trick to lower latency

- Idea 1: Combine security handshake and sequence number handshake on first connection to server

- Idea 2: If server remembers information about client, need 0 handshakes on later connections
- Three way handshakes are required because server forgets info on client. Important in old days but no longer as servers have massive memory
- A round trip is a big deal (several hundred msec across US) at today's high speeds

Additional Stuff

- Stream multiplexing: multiple streams in a single QUIC connection between client and server for HTTP/2
- No head of line blocking: can do HTTP/2 over a single TCP connection but a single loss stalls all streams. Not so in QUIC
- Shared congestion information: as we will see it takes TCP a long time to ramp up. In QUIC all congestion information is shared.
- Wave of future: 9% of all websites use QUIC (4/2023). 40% of Chrome traffic uses QUIC 2

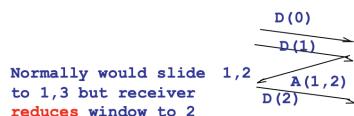
Congestion/Flow Control

- **Flow Control** - Changing sender speed to match receiver speed
- **Congestion Control** - Changing sender speed to match network speed

Flow Control

- flow control - receiver sets the recv window size = 0, but then deadlock if window size = 0, so periodically receiver needs to send a packet or probe to maintain the connection
- FREEZE - receiver could also send back a FREEZE message (which contains a timeout) that keeps the connection open but stops the sender from sending until it receives another packet from the node that sent the freeze - needs seq numbers for freeze because delayed duplicates may restart freeze timeout

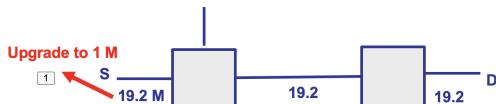
Windows provide static flow control. Can provide dynamic flow control if receiver acks indicate Lower and Upper Window Edge.



Need to avoid **deadlock** if window is reduced to 0 and then increase to $c > 0$. In OSI, receiver keeps sending c. In TCP, sender periodically probes an empty window.

Congestion Control

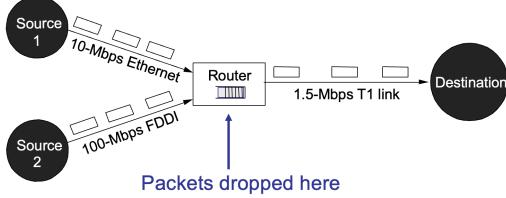
- different connection bandwidths leads to congestion on high output senders



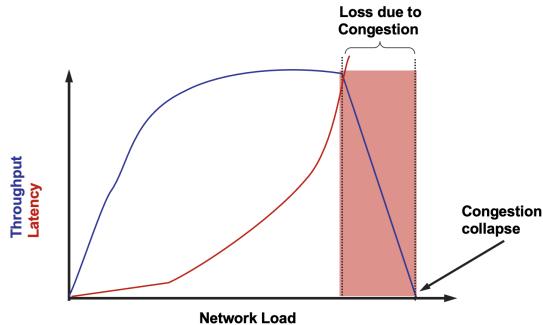
- [2] 1) WHEN LINK FROM S TO FIRST ROUTER WAS UPGRADED FROM 19.2 Kbps TO 1 MBPS, THE TIME FOR A FILE TRANSFER WENT UP FROM A FEW SECONDS TO A FEW HOURS
- [2] 2) THIS HAPPENED IN AN EXPERIMENT IN DEC IN THE 1980s. SHOWED THE NEED FOR CONGESTION CONTROL (DECBIT)
- [2] 3) VERY SIMILAR EXPERIENCES IN INTERNET LED VAN JACOBSEN TO PROPOSE TCP CONGESTION CONTROL. VAN IS AN AFFILIATE PROFESSOR AT UCLA!

- queue builds if send rate > drain rate → dropped packets

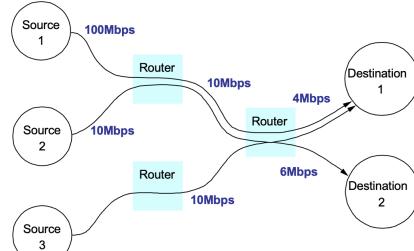
- "goodput" amount of packets that come "out" of the network



- Congestion causes both collapsed throughput ("goodput") and massive latency

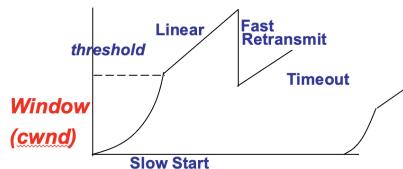
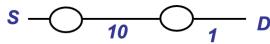


Fair Bandwidth Allocation



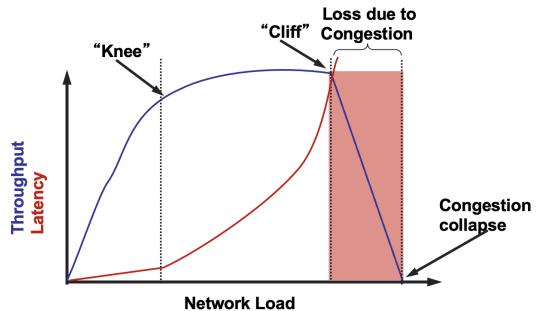
- Can use Ford-Fulkerson Maximum Flow Algo

Mitigation

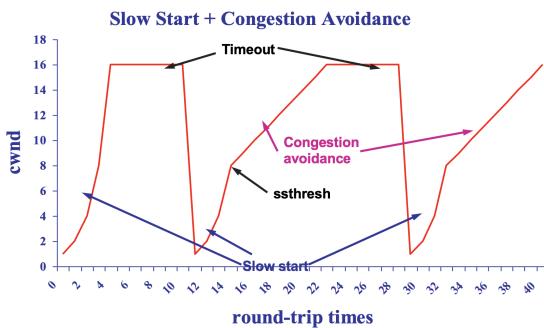
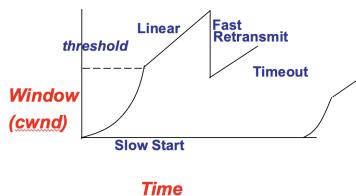
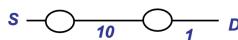


- Time**
- "Slow start" - start with sender window size of 1; double every time we get a successful ack
- tighten window if timer elapses or 3 dup acks
- AIMD** - Additive Increase, Multiplicative Decrease - tighten multiplicatively (div by 2) and open additively (by MSS) the window size (aperture)
 - used for congestion avoidance (proactive)
- ECN** - Explicit Congestion Notification - ECN bit, requires another packet from dest to sender

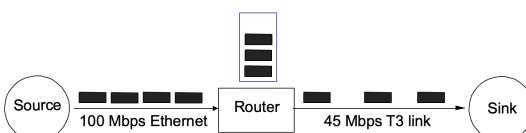
- Proactive Congestion Control - stay left of knee; Reactive - stay left of cliff



- Compromise: adaptive approximation
 - If congestion signaled, reduce sending rate by x
 - If data delivered successfully, increase sending rate by y
 - Window-based congestion control
 - ◆ Unified congestion control and flow control mechanism
 - ◆ $rwin$: advertised flow control window from receiver
 - ◆ $cwnd$: congestion control window
 - » Estimate of how much outstanding data network can deliver in a round-trip time
 - ◆ Sender can only send $\min(rwin, cwnd)$ at any time
 - Idea: decrease $cwnd$ when congestion is encountered; increase $cwnd$ otherwise
- Basic Algo
 - ssthresh cliff to mitigate exponential opening on slow start



Probing the Network



- Each source independently probes the network to determine how much bandwidth is available
 - ◆ Changes over time, since everyone does this
- Assume that packet loss implies congestion
 - ◆ Since errors are rare; also, requires no support from routers

- TCP uses AIMD to adjust congestion window
 - ◆ Converges to fair share of bottleneck link
 - ◆ Increases modestly in good times
 - ◆ Cuts drastically in bad times
- But what rate should a TCP flow use initially?
 - ◆ Need some initial congestion window
 - ◆ We'd like to TCP to work on all manner of links
 - ◆ Need to span 6+ orders of magnitude, e.g., 10 K to 10 Gbps.
 - ◆ Starting too fast is catastrophic! So start with $cwnd = 1$
-

ECN

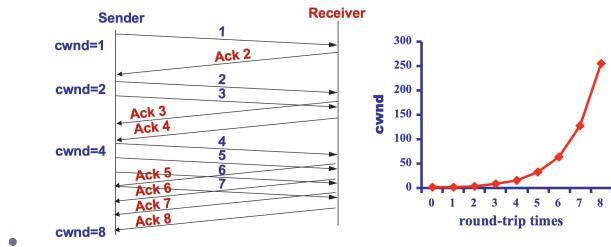
- Explicit congestion signaling
 - Source Quench: ICMP message from router to sender
 - DECBit / Explicit Congestion Notification (ECN):
 - Router marks packet based on queue occupancy (i.e. indication that packet encountered congestion along the way)
 - Receiver tells sender if queues are getting too full
- Implicit congestion signaling
 - Packet loss
 - Assume congestion is primary source of packet loss
 - Lost packets indicate congestion
 - Packet delay
 - Round-trip time increases as packets queue
 - Packet inter-arrival time is a function of bottleneck link

Throttling

- Window-based (TCP)
 - Constrain number of outstanding packets allowed in network
 - Increase window to send faster; decrease to send slower
 - Pro: Cheap to implement, good failure properties
 - Con: Creates traffic bursts (requires bigger buffers)
- Rate-based (many streaming media protocols)
 - Two parameters (period, packets)
 - Allow sending of x packets in period y
 - Pro: smooth traffic
 - Con: fine-grained per-connection timers, what if receiver fails?

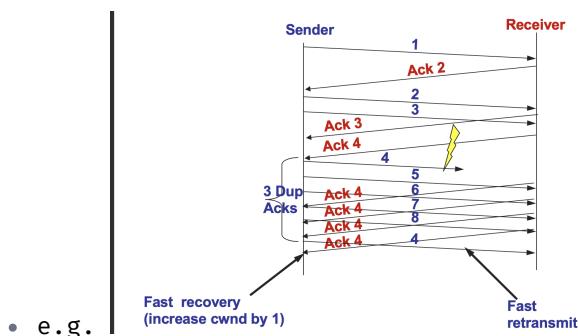
Slow Start

- Goal: quickly find the equilibrium sending rate
- Quickly increase sending rate until congestion detected
 - ◆ Remember last rate that worked and don't overshoot it
- TCP Reno Algorithm:
 - ◆ On new connection, or after timeout, set $cwnd=1$ MSS
 - ◆ For each segment acknowledged, increment $cwnd$ by 1 MSS
 - ◆ If timeout then divide $cwnd$ by 2, and set $ssthresh = cwnd$
 - ◆ If $cwnd \geq ssthresh$ then exit slow start
- Why called slow? Its exponential after all...

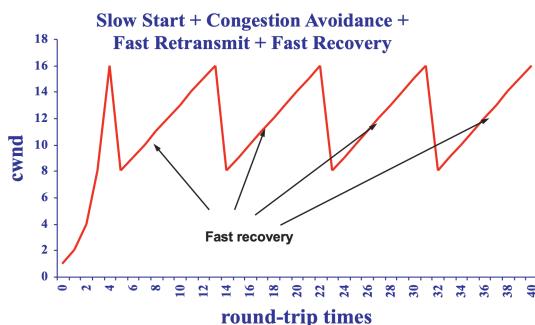


Fast Recovery

- Fast retransmit
 - Timeouts are slow (default often 200 ms or 1 second)
 - When packet is lost, receiver still ACKs last in-order packet
 - Use 3 duplicate ACKs to indicate a loss; detect losses quickly
- Fast recovery
 - Goal: avoid stalling after loss
 - If there are still ACKs coming in, then no need for slow start
 - If a packet has made it through → we can send another one
 - **Divide cwnd by 2** after fast retransmit
 - Increment cwnd by 1 MSS for each additional duplicate ACK



• e.g.



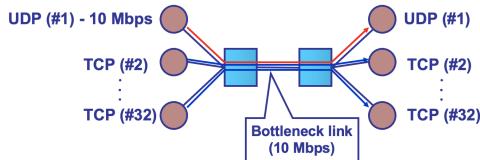
• all together

Open Problems

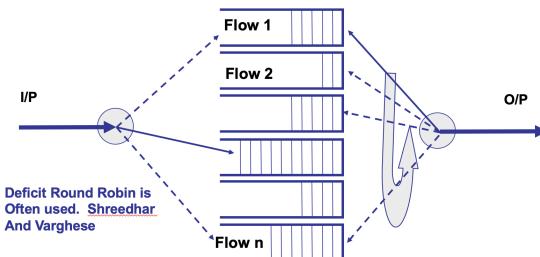
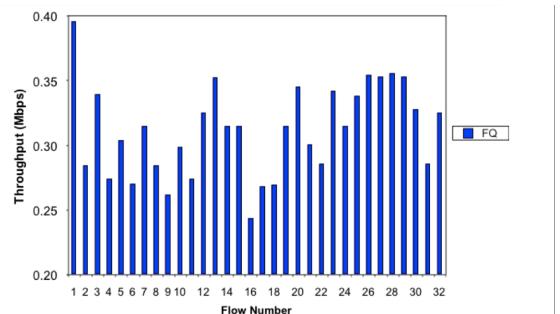
- most connections short, possibly no gain from low start
- magic number to run low start or fast recovery, etc.
- shared bandwidth for UDP and TCP
- decide which packets can be dropped: syn, ack, none

Router Scheduling

- router also need to schedule packets to support congestion control
1 UDP (10 Mbps) and 31 TCPs sharing a 10 Mbps line



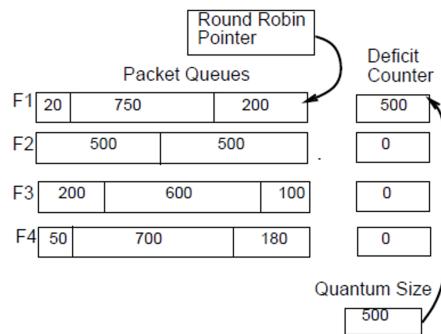
- use fair queueing → also allow TCP-UDP bandwidth share



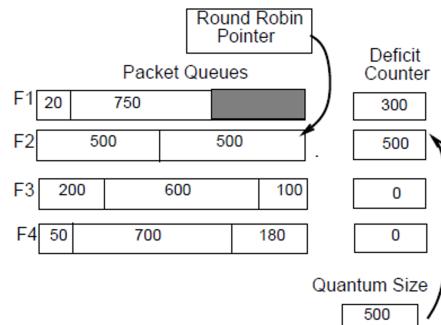
- Round-Robin
- but we need to weight round robin due to differing packet sizes on top of queue frequent use

DRR - Deficit Round Robin

- use a deficit counter that starts with some quantum size, then decrease the deficit by the packet size sent

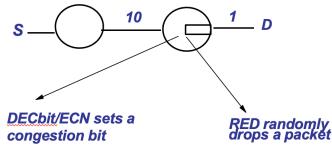


the packet size sent



- after packet sent at robin pointer

- also enable **Random Early Detect** - randomly drop a packet early before congestion if ECN



not set so that an ECN may be sent early

Q: Why would a router drop a perfectly good packet even if has buffer space
A: As an early form of congestion warning if one does not have a congestion bit.
 Many IP routers have such a bit today, called the ECN bit

Novel Approaches

- HTTP Pipelining: To reduce latency browser opens up multiple connections. Still slow over TCP.
- QUIC: Layered below HTTP and above TCP, places multiple streams in a single connection. Finesses slow start. Loss on one stream does not interrupt other stream. Avoids extra handshakes for TLS
- TCP fluctuates too much. New protocols like DCTCP adjust more smoothly in Microsoft Data Centers. Also used by Apple

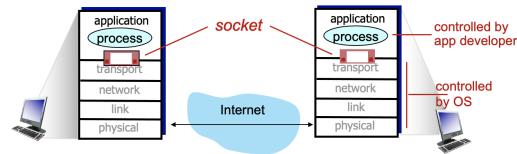
12 – Application Protocols

Client-Server vs. P2P

- Client-server - server always on with permanent IP, client connects with dynamic IP
 - e.g., HTTP, IMAP, FTP
- P2P - no such server, all inter client connects, scalable
 - e.g., torrent file sharing - bittorrent

Process Communication

- processes on the same host comms via Inter-Process Comms (IPC) via OS
- client-server - processes are either clients or servers and are always listening or trying to connect
- p2p - procs have both client and server ports



- comms via sockets (abstraction)
- procs identified by both IP and Port
 - port e.g., HTTP:80, SMTP:25 (mail)

Application Level Protocols

- protocols contain/define
 - types of messages exchanged,
 - e.g., request, response
 - message syntax:
 - what fields in messages & how fields are delineated
 - message semantics
 - meaning of information in fields
 - rules for when and how processes send & respond to messages
- apps require transport service
 - data integrity
 - some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
 - other apps (e.g., audio) can tolerate some loss
 - timing
 - some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"
 - throughput
 - some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
 - other apps ("elastic apps") make use of whatever throughput they get
 - security
 - encryption, data integrity, ...

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

- e.g. proc examples

TCP vs. UDP

- **reliable transport** between sending and receiving process
 - **flow control**: sender won't overwhelm receiver
 - **congestion control**: throttle sender when network overloaded
 - **connection-oriented**: setup required between client and server processes
 - **does not provide**: timing, minimum throughput guarantee, security

UDP service:

 - **unreliable data transfer** between sending and receiving process
 - **does not provide**: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? Why is there a UDP?

Q: why bother? Why
is there a UDP?

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
web documents	HTTP [RFC 7230, 9110]	TCP
voIP telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

- e.g., app transport protocols

Vanilla TCP & UDP sockets:

- no encryption
 - cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS)

- provides encrypted TCP connections
 - data integrity
 - end-point authentication

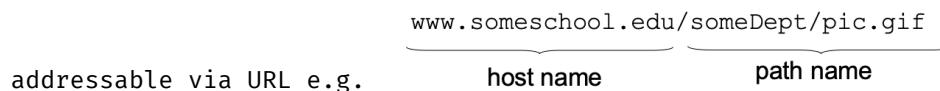
TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
 - cleartext sent into “socket” traverse Internet *encrypted*
 - See our earlier lecture or Chapter 8 in Kurose-Ross

- TCP security (TLS)

Web and HTTP

- web page consists of objects stored in a DOM structure
 - object can be HTML file/element, images, applets, audio, etc.
 - web-page consists of base HTML file (e.g. index.html) with several referenced objects



HTTP

- **HTTP** - hypertext transfer protocol

HTTP: hypertext transfer protocol

- Web's application-layer protocol
 - client/server model:
 - *client*: browser that requests, receives, (using HTTP protocol) and "displays" Web objects
 - *server*: Web server sends (using HTTP protocol) objects in response to requests



- HTTP is persistent or non-persistent because storing state is complex

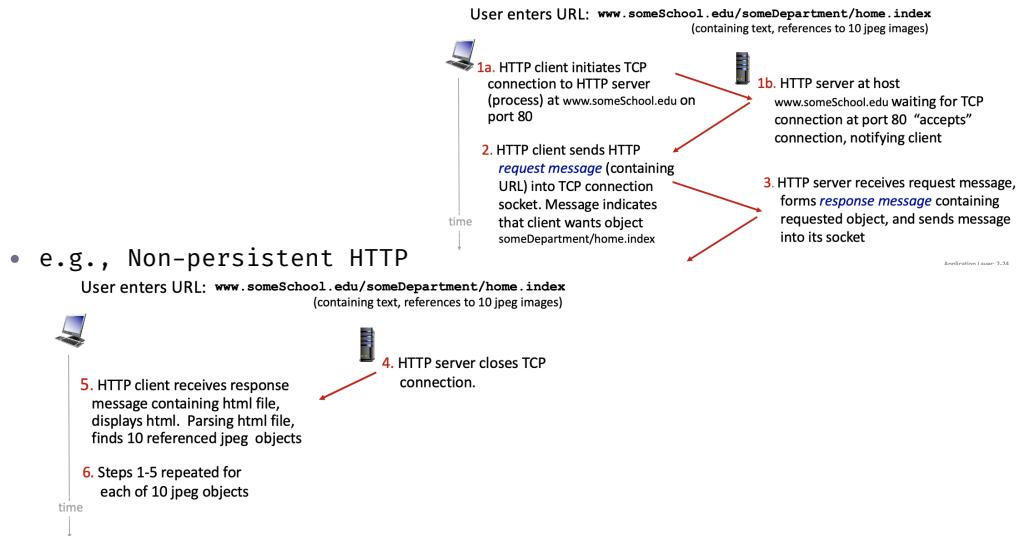
Non-persistent HTTP

1. TCP connection opened
 2. at most one object sent over TCP connection

downloading multiple objects required multiple connections

Persistent HTTP

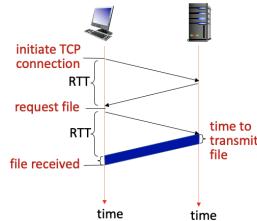
- TCP connection opened to a server
 - multiple objects can be sent over *single* TCP connection between client, and that server
 - TCP connection closed



RTT (definition): time for a small packet to travel from client to server and back

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



$$\text{Non-persistent HTTP response time} = 2\text{RTT} + \text{file transmission time}$$

- non-persistent performance
- persistent (HTTP 1.1) v. non-persistent pro cons

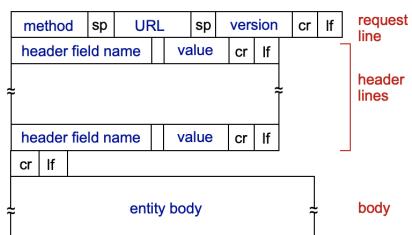
Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

Persistent HTTP (HTTP1.1):

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects (cutting response time in half)

- messages are request (GET/POST/PUT ...) or response (status code, e.g. 404)



POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

HEAD method:

- requests headers (only) that would be returned if specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?')

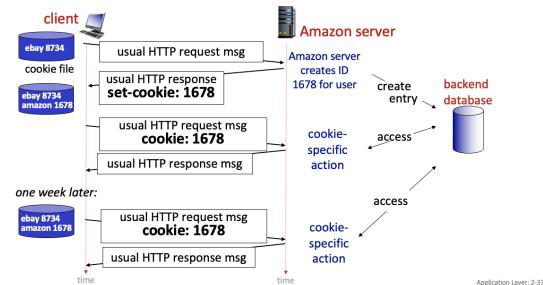
- request messages

`www.somesite.com/animalsearch?monkeys&banana`

- status code appears in 1st line in server-to-client response message.
- some sample codes:
 - 200 OK**
 - request succeeded, requested object later in this message
 - 301 Moved Permanently**
 - requested object moved, new location specified later in this message (in Location: field)
 - 400 Bad Request**
 - request msg not understood by server
 - 404 Not Found**
 - requested document not found on this server
 - 505 HTTP Version Not Supported**
- request messages

Cookies - State Management

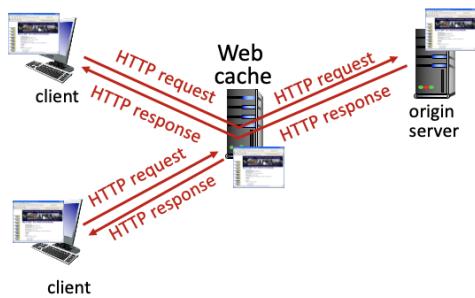
- HTTP requests are stateless so for multi-step exchange or repeated connections, store cookies to maintain state (stored in the browser)



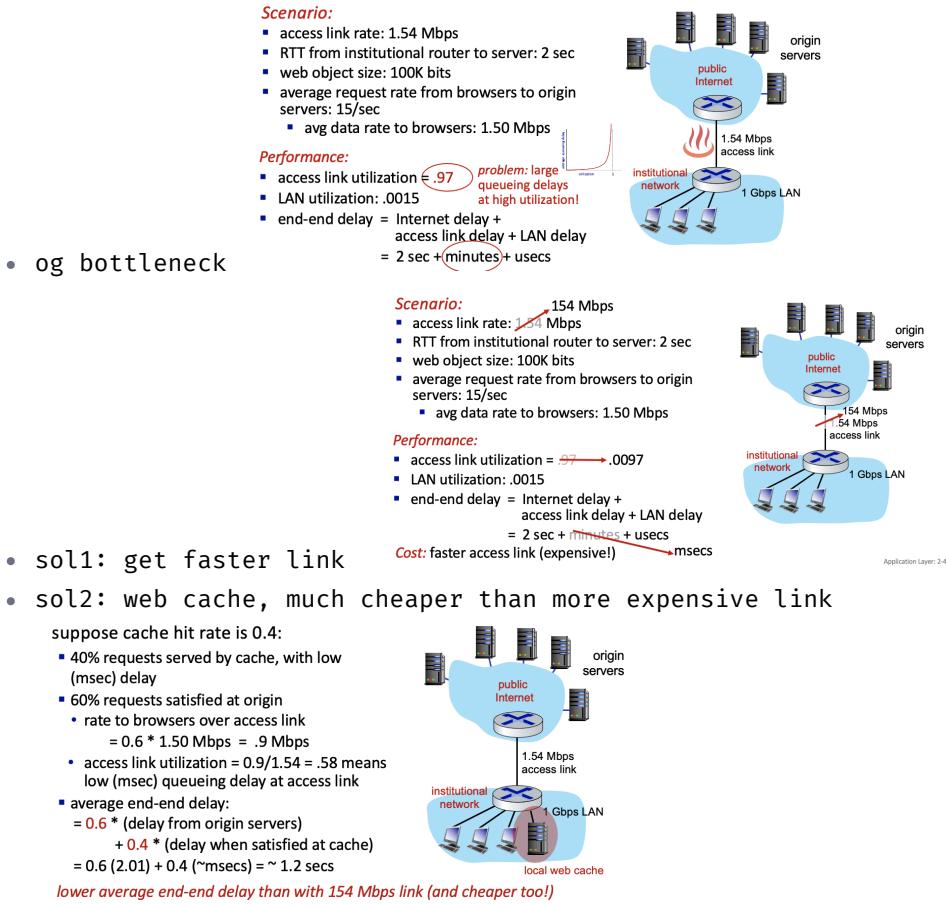
- components
 - cookie header line of HTTP **response** message
 - cookie header line in next HTTP **request** message
 - cookie kept on user's host and managed by user's browser
 - back-end DB at server website containing cookie id val to map state
- **first party cookies** - track user behavior for the given website
- **third party cookies** - tracks user behavior across multiple websites without visiting third party site
- GDPR (EU general data protection regulation) requires sites to inform users about 3rd party cookies

Web Cache

- to inc performance and decrease load on server, initial http get may include data and web cached data
- may be done via proxy server which acts both as client and server intermediary



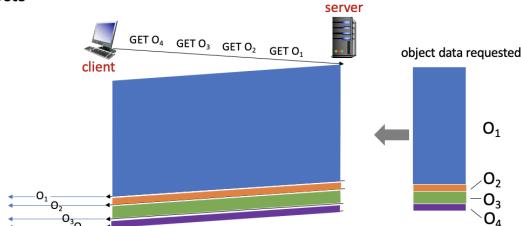
- example



HTTP/2

- dec delay in multi obj HTTP requests
 - HTTP1.1 introduced multiple pipelined gets over single TCP connection
 - server responds in order (FCFS) - this may cause head of line (HOL) blocks for small objs behind large objects → loss recovery stalling transmission
- HTTP/2:** [RFC 7540, 2015] increased flexibility at server in sending objects to client:
- methods, status codes, most header fields unchanged from HTTP 1.1
 - transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
 - push unrequested objects to client
 - divide objects into frames, schedule frames to mitigate HOL blocking

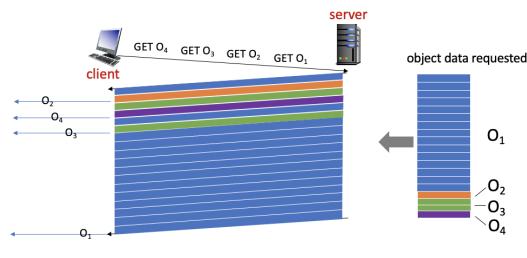
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



- e.g., HTTP/1.1 HOL issue

objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: objects divided into frames, frame transmission interleaved



- e.g., HTTP/2 frame sol

HTTP/3

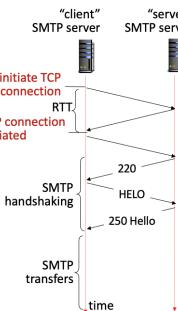
- HTTP/2 disadvantages
 - single connection means recovery from packet loss still stalls object transmissions
 - no security over vanilla TCP
- HTTP/3 adds security and per object error and congestion control via pipelining over UDP

Email, SMTP, IMAP

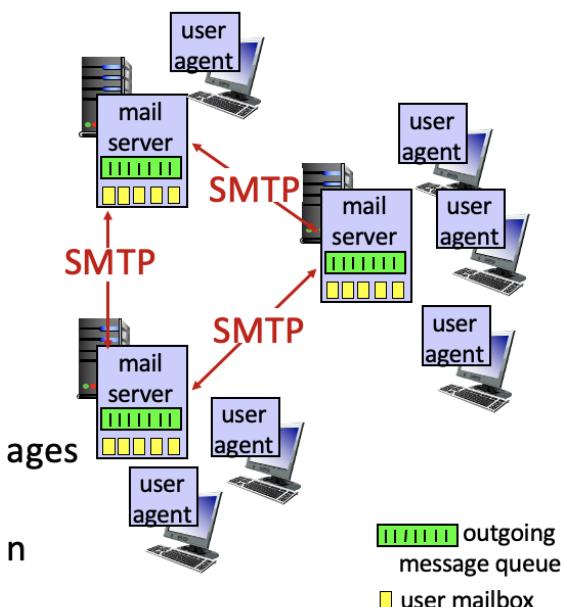
- composed of: user agents, mail servers, and SMTP
- **SMTP** - simple mail transfer protocol
 - between mail servers on client (sender) - server (receiver) protocol

SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
 - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
 - SMTP handshaking (greeting)
 - SMTP transfer of messages
 - SMTP closure
- command/response interaction (like HTTP)
 - commands: ASCII text
 - response: status code and phrase

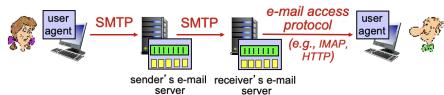


- **user agent** - mail reader, creating editing reading sending mail on various clients - outlook, gmail, iphone, etc,
- **mail server** - composed of mailbox (contains incoming messages) and message queue (outgoing mail queue)



-

- request and response messages
- **IMAP** - internet mail access protocol - provides retrieval and deletion



- comparison with HTTP:*
- HTTP: client pull
 - SMTP: client push
 - both have ASCII command/response interaction, status codes
 - HTTP: each object encapsulated in its own response message
 - SMTP: multiple objects sent in multipart message

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF,CRLF to determine end of message

- **SMTP**: delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
 - **IMAP**: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- **HTTP**: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

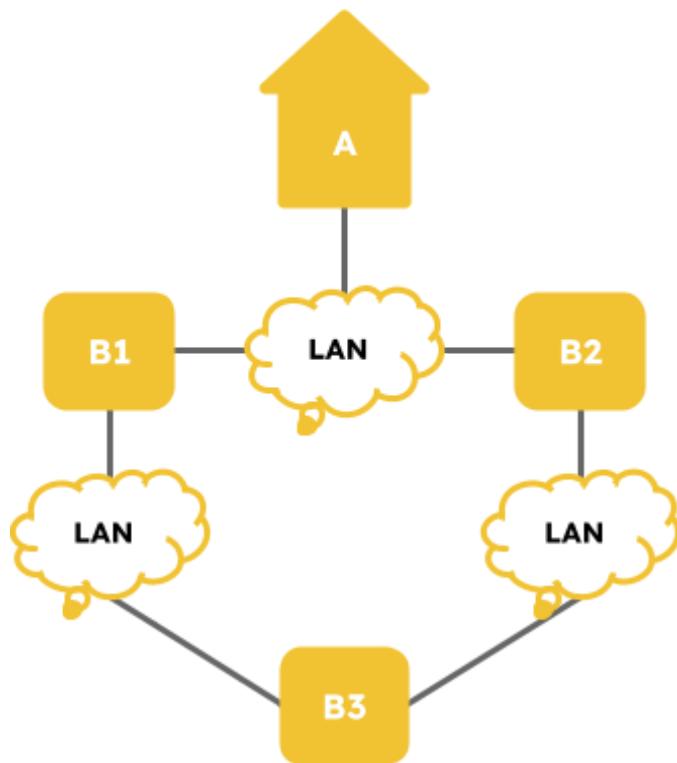
Homework 3: Bridging, Routing, and IP Forwarding

Due on Tuesday, November 12th, 2024 at 2:00 pm (Week 7)

Note: Question 5 is merely for exam practice; do not turn it in. We will also just pick **two** of the four mandatory questions to grade.

Question 1: Bridging and Loops

The Spanning Tree algorithm cannot prevent temporary loops (for example, if two separate LANs are connected by a bridge, and then someone plugs them together using a repeater). Eventually, the loop will be broken, and the right bridges will turn off. However, packets can circulate at very high speeds until the loop is broken.



Alyssa P. Hacker has thought of a way to improve the situation during temporary loops. Consider a temporary loop of bridges shown above and assume that bridges **B1**, **B2**, and **B3** all think they are ON. Suppose **A** sends a multicast packet. Both **B1** and **B2** pick up the packet, and the packet will circulate in two directions, both clockwise and anti-clockwise. This will also happen if the destination is unknown.

Questions

1. **Modifying Bridge Learning** (15 points)

Based on Alyssa's observation, how could you modify the bridge learning and forwarding to prevent multicast and unknown destination packets from circulating continuously in a temporary loop?

Bridge learning already defines a mapping between interfaces and source MAC addresses. To temporarily break loops, we can simply drop packets that arrive from an interface that's different than expected.

2. **Dropped Packets** (10 points)

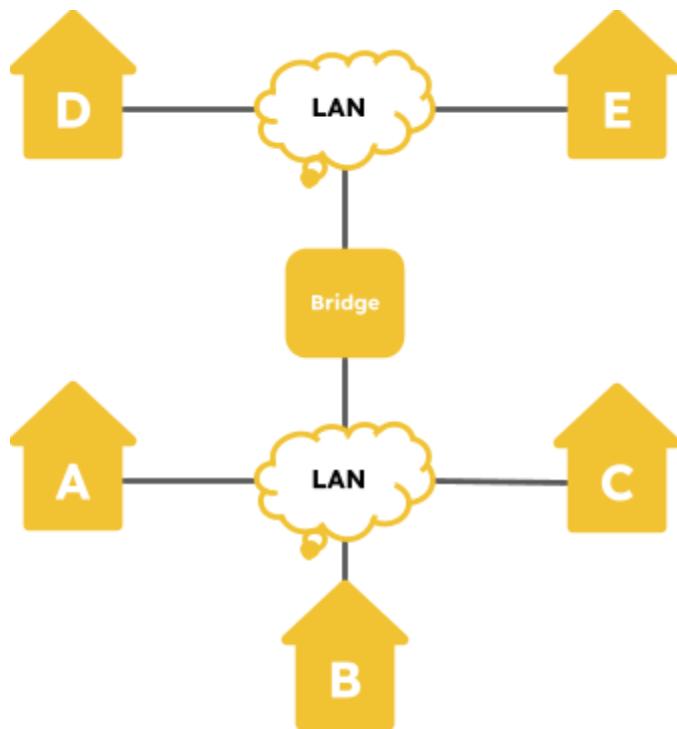
What goes wrong with your method if packets can be dropped?

Let's say that **A** sends a packet to the upper LAN, but for whatever reason, it's dropped on the way to **B2**. **B1** will only send **A**-originating packets from the **right** interface. **B2** will now receive the packet from the **lower** interface through **B3**, so it will only send **A**-originating packets from the **lower** interface. This creates a loop.

Question 2: IP Broadcast Storms, Bridges vs. Routers

(Adapted from Perlman's book.) A broadcast storm is an event that causes a flurry of messages. One implementation that caused broadcast storms was the Berkeley UNIX endnode IP implementation. In this implementation, an endnode attempts to forward a packet that it mysteriously receives with a network layer (IP) address that is different from itself. This is what you would do if you found a neighbor's letter wrongly placed in your mailbox. However, this seemingly helpful policy can cause problems.

Consider the figure below which shows 2 LANs connected by a bridge, with several IP endnodes on each LAN. There are no IP routers. All IP endnodes are configured with the same mask and so can tell that they have the same net number/prefix. Suppose IP endnode A is incorrectly configured and incorrectly thinks its data link address is all 1's. The data link address of all 1's is the broadcast address: any packet sent to such an address is received by all stations on a LAN (it is the ultimate multicast address!).



Questions

1. **Address Resolution Protocol** (10 points)

What happens when another IP endnode D decides to send a packet to IP endnode A? Assume that D initially does not have A's data link address in its cache, and so must do the ARP protocol. Give the sequence of events.

- D broadcasts an ARP request for A's IP address.

2. A responds with the faulty broadcast address.
3. D then sends the data packet (IP addressed to A) to everyone in the network (the bridge forwards this packet since it has the destination as the faulty broadcast address).
4. All the hosts running the Berkeley IP endnode implementation attempt to forward this packet to A. They will all send an ARP request for A.
5. A responds to all these requests with the faulty broadcast address.
6. All the hosts will then attempt to forward the data packet to A, but with the faulty broadcast address.
7. All these hosts will receive these packets and attempt to forward the data packet to A. (For each iteration k , there will be T^k (the total number of endnodes in the network of Ethernets) packets sent.)

2. **IP Router** (15 points)

Suppose the bridge is replaced by an IP router. (Of course, the masks at the nodes must be changed so that there are now two masks, one for each LAN. Note a mask is just a bitmap of 1's in the most significant bits that tells you how long the prefix for that subnet is) The problem does not disappear, but it does get a little better. Explain as precisely as you can the improvement using two parameters: T the total number of endnodes in the network of Ethernets, and M the maximum number of endnodes in a single Ethernet.

In 2.1, we know that the number of packets sent is T^k . If we add an IP router, it will request the MAC address of A itself. This limits the storm to the lower subnet, or M^k .

Question 3: IP Forwarding, Longest Matching Prefix using Tries

To prepare for the Arista interview, you give ChatGPT the prompt:

“Can you write C code to do longest matching prefix of a 32-bit destination IP address in an Ethernet packet with an IP routing header which contains the destination IP address at the right offset. Do longest matching prefix on a database of prefixes called ForwardingTable implemented as a binary trie. The result should return the output interface and IP address of the next hop.”

ChatGPT's reply. Here's a basic implementation of a longest prefix match using a binary trie data structure in C. This code will parse the destination IP address from the Ethernet packet, traverse the trie to find the longest matching prefix, and return the output interface and next hop IP address.

In this example, I assume:

1. Each trie node has children for **0** and **1** bits.
2. Each leaf node contains the next hop information, including the output interface and next hop IP.
3. The trie is built ahead of time and passed to the longest prefix match function.

Here's the code:

```
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

// Define a structure for the trie node
typedef struct TrieNode {
    struct TrieNode *children[2]; // 0 and 1 children
    int is_end_of_prefix; // Q2: Why is this field needed?
    uint32_t next_hop_ip; // IP address of the next hop
    int output_interface; // Output interface
} TrieNode;

// Initialize a new trie node
TrieNode* create_node() {
    TrieNode *node = (TrieNode*)malloc(sizeof(TrieNode));
    node->children[0] = node->children[1] = NULL;
    node->is_end_of_prefix = 0;
    node->next_hop_ip = 0;
    node->output_interface = -1;
}
```

```
    return node;
}

// Insert a prefix into the trie
void insert_prefix(TrieNode *root, uint32_t prefix, int prefix_length, uint32_t
next_hop_ip, int output_interface) {
    TrieNode *node = root;
    for (int i = 31; i >= 32 - prefix_length; i--) {
        int bit = (prefix >> i) & 1;
        if (!node->children[bit]) {
            node->children[bit] = create_node();
        }
        node = node->children[bit];
    }
    node->is_end_of_prefix = 1;  Q3: Why do this?
    node->next_hop_ip = next_hop_ip;
    node->output_interface = output_interface;
}

// Perform the longest prefix match
int longest_prefix_match(TrieNode *root, uint32_t dest_ip, uint32_t *next_hop_ip,
int *output_interface) {
    TrieNode *node = root;
    TrieNode *last_matching_node = NULL;
    for (int i = 31; i >= 0; i--) {
        int bit = (dest_ip >> i) & 1;
        if (!node->children[bit]) break;
        node = node->children[bit];
        if (node->is_end_of_prefix) {
            last_matching_node = node;
        }
    }
    if (last_matching_node) { // Q4: Why do this?
        *next_hop_ip = last_matching_node->next_hop_ip;
        *output_interface = last_matching_node->output_interface;
        return 1;  // Match found
    }
    return 0;  // No match found
}

// Sample Ethernet packet structure
typedef struct {
```

```

    uint8_t destination_mac[6];
    uint8_t source_mac[6];
    uint16_t ethertype;
    uint8_t payload[1500];
} EthernetPacket;

// Extract destination IP from an IP header within the Ethernet payload
uint32_t extract_destination_ip(EthernetPacket *packet) {
    return ((uint32_t)packet->payload[16] << 24) |
        ((uint32_t)packet->payload[17] << 16) |
        ((uint32_t)packet->payload[18] << 8) |
        ((uint32_t)packet->payload[19]);
} // Q5: Explain why the code uses the constants 16, 17, 18, 19. Explain shifts
and ORs.

int main() {
    TrieNode *root = create_node();

    // Example: Insert prefixes into the trie
    insert_prefix(root, 0xC0A80000, 16, 0xC0A80001, 1);
    // 192.168.0.0/16 -> next hop 192.168.0.1, interface 1
    insert_prefix(root, 0xC0A80100, 24, 0xC0A80101, 2);
    // 192.168.1.0/24 -> next hop 192.168.1.1, interface 2

    // Example Ethernet packet with destination IP 192.168.1.5
    EthernetPacket packet;
    memset(&packet, 0, sizeof(EthernetPacket));
    packet.payload[16] = 192;
    packet.payload[17] = 168;
    packet.payload[18] = 1;
    packet.payload[19] = 5;

    uint32_t dest_ip = extract_destination_ip(&packet);
    uint32_t next_hop_ip;
    int output_interface;

    if (longest_prefix_match(root, dest_ip, &next_hop_ip, &output_interface)) {
        printf("Match found:\n");
        printf("Next hop IP: %u.%u.%u.%u\n",
            (next_hop_ip >> 24) & 0xFF,
            (next_hop_ip >> 16) & 0xFF,
            (next_hop_ip >> 8) & 0xFF,

```

```

        next_hop_ip & 0xFF);
printf("Output interface: %d\n", output_interface);
} else {
    printf("No matching prefix found.\n");
}
// Free trie memory (not shown here)
return 0;
}

```

Questions

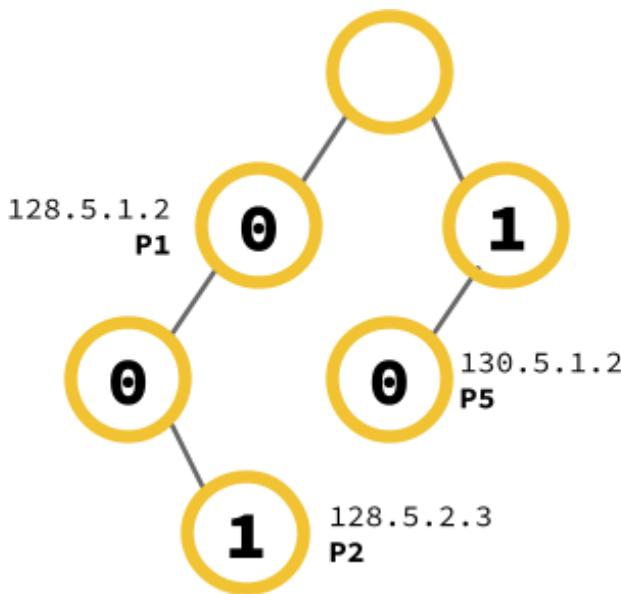
1. Trie Structure (10 points)

Draw an example of the trie data structure after you insert three prefixes:

0* whose next hop is **128.5.1.2** and interface is **P1**,

001* whose next hop is **128.5.2.3** and interface is **P2**, and

10* whose next hop is **130.5.1.2** and interface is **P5**.



2. End of Prefix (3 points)

What is the point of `is_end_of_prefix` (see comment)? This seems useless at first because the loops for insertion and search terminate when they see a null pointer. Drawing the picture for part 1 will help as it will for any code you get from ChatGPT.

The end of a prefix isn't always once you terminate your trie search. For example, what if your IP address starts with three **0s**? **000*** isn't a valid prefix, so you have to keep track of which prefixes are valid.

3. Resetting End of Prefix (3 points)

Again, when you insert a prefix the code sets `is_end_of_prefix` to 1 (see comment). Why is that needed and why does it not need to be reset?

This is needed since we need to indicate which nodes in our trie have next hops. Note that we don't ever need to reset `is_end_of_prefix` since we don't have a mechanism to remove prefixes. Also, when adding longer prefixes with a shared sub prefix, we need to be able to distinguish between them.

4. Last Matching Node (3 points)

At the end of search (see comment), the code retrieves the next hop from `last_matching_node`. Why not pick it up directly as the code walks the trie?

We don't want to greedily take the first end of prefix. Since we're looking for the longest prefix match, we'll want to find the last matching node.

5. IP Addresses (3 points)

The code that retrieves the Dest IP address (see comment) has some interesting constants including 16, 17, 18, and 19. It also does some shifting and ORing. Explain why.

16, 17, 18, and 19 represent the bytes in the Ethernet payload that correspond to the destination IP address. The shifting and ORing is to convert from network order (big endian) to host order (little endian on most machines).

6. Missing Code (3 points)

Explain what code may be missing besides deallocating memory.

Routers need a default address to forward to. This code as written does not do such a thing. Other answers may include:

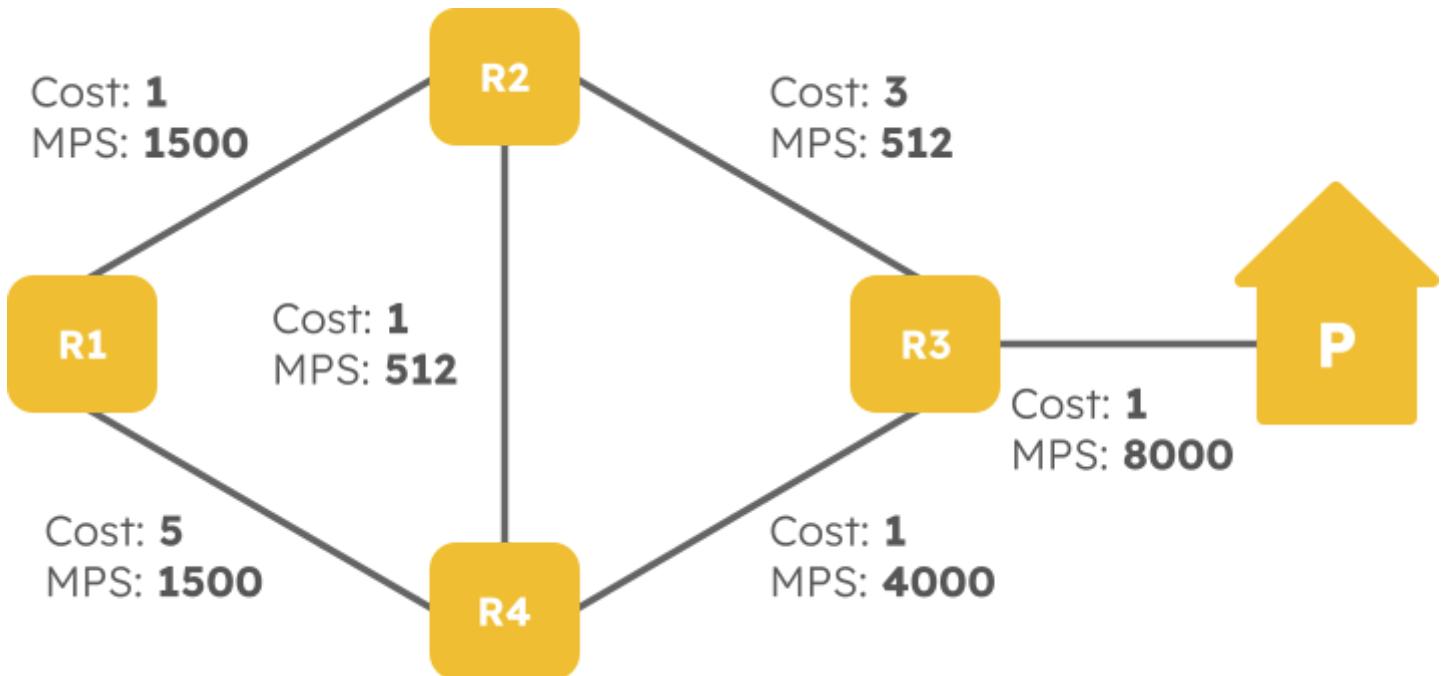
- Input sanitation/format checking
- Removing prefixes
- Modifying prefixes (changing next hops)

Really, anything reasonable receives credit.

Question 4: Modifying Routing to Avoid Fragmentation

We learned in class that in order to avoid fragmentation, IP tries various packet sizes until it finds one that works. Alternatively, we could modify the routing protocol to compute the minimum of the maximum packet sizes of all links on the best route to each destination.

In distance vector routing, a router **R** computes its own distance $\text{Distance}(P, R)$ to a destination prefix **P** using the distances sent by its neighbors as follows: $\text{Distance}(P, R) = \text{Minimum across all neighbors } N \text{ of } \text{Distance}(P, N) + \text{Distance}(R, N)$. We want to see how to modify this protocol to also compute the minimum max packet size on the shortest distance route to **P**.



Questions

1. **Shortest Path** (5 points)

What is the shortest path between **R1** and **P**? What is the largest packet size that is guaranteed to get through without fragmentation on this path?

R1 -(1)-> **R2** -(1)-> **R4** -(1)-> **R3** -(1)-> **P** (**4** units in total)

The largest packet size is **512** units.

2. **Minimum Max Packet Size** (5 points)

Assume each router receives the additional variables $\text{Distance}(P, N)$ and $\text{MinMaxPacketSize}(P, N)$ from each neighbor **N**. Write an equation to compute these two variables from the corresponding variables of all a router's neighbors.

Distance(R, P) = min { Distance(R, N) + Distance(N, P) | N in all neighbors of R }

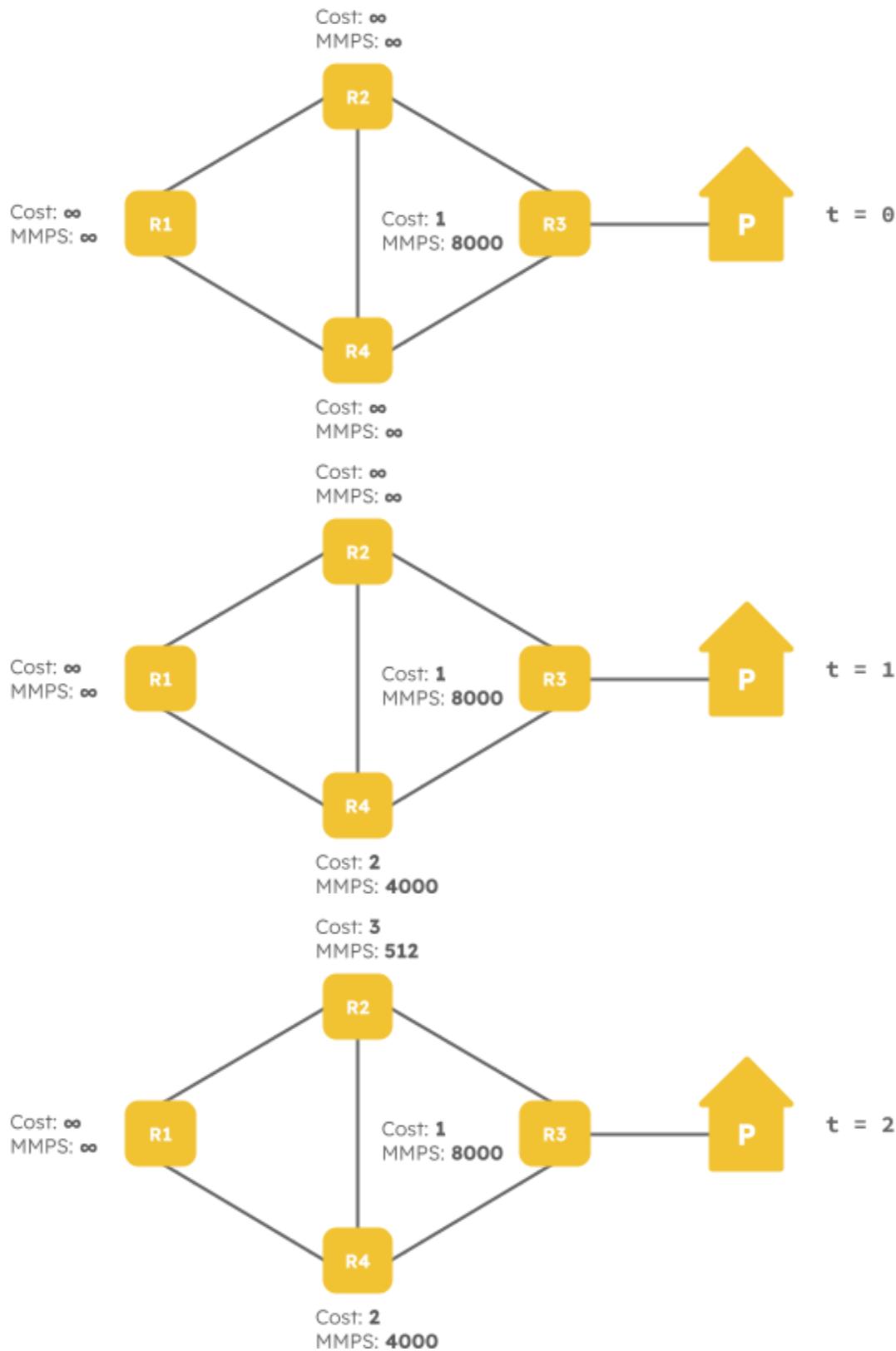
MinMaxPacketSize(R, P) = max { min(MaxPacketSize(R, N), MinMaxPacketSize(N, P)) | N in all neighbors of R such that N is the minimum distance to P }

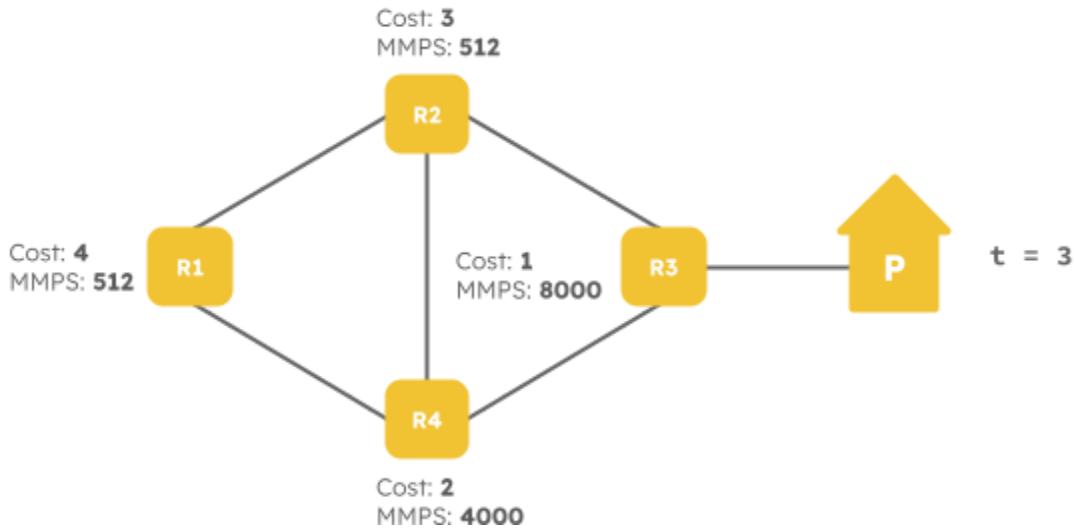
(If your second equation is a bit simpler since it doesn't consider neighbors with the same cost, that's okay).

3. **Packet Size Propagation** (10 points)

Assume we are calculating these estimates only for distances to **P**. Assume that at **t = 0**, router **R3** has **Distance (P, R3) = 1** and **MinMaxPacketSize (P, R3) = 8000** and all other routers have the distance and min packet size to **P** set to a default of **infinity**. Assume that at **t = 0**, each router sends an update to each neighbor. Draw several pictures of the same topology with the changing estimates of each router for its two variables based on the equation you wrote down until all estimates stop changing. After how much time do all the estimates converge (i.e. do not change any more)?

The answers below assume that each propagation happens in link cost units. Your solution may slightly differ (and that's okay) if you let each propagation happen in one unit of time.

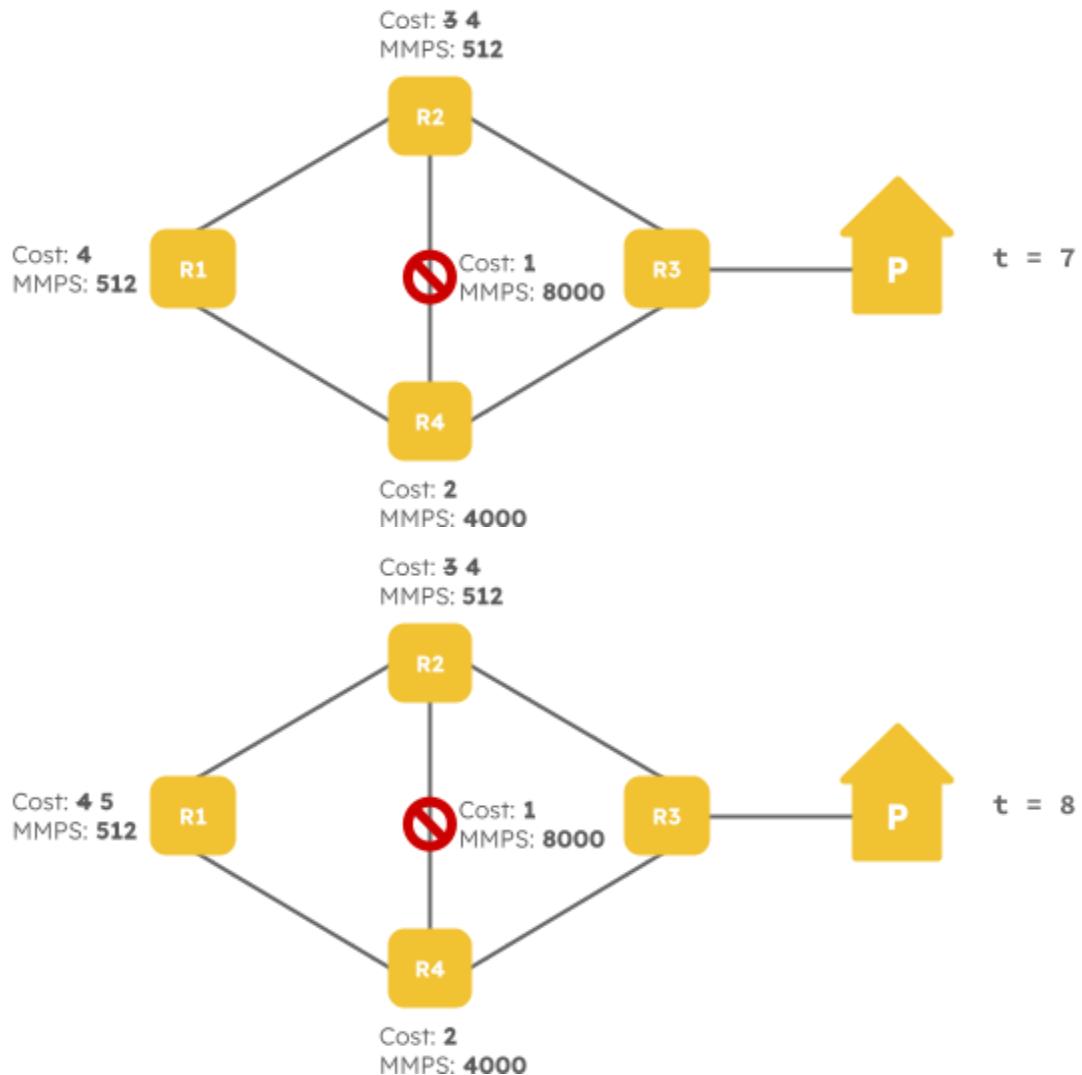




Another update occurs at $t = 6$, but it doesn't change **R1**'s calculations.

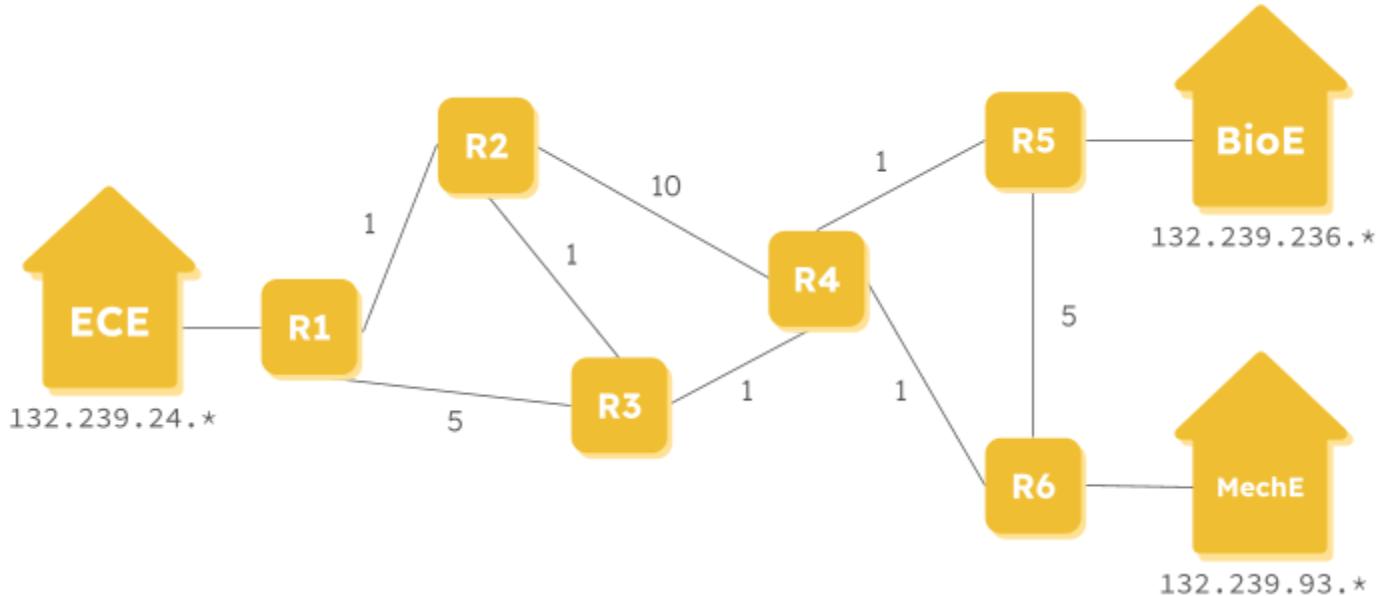
4. **Crash** (5 points)

Now assume the link to **R2** to **R4** crashes at time $t = 7$. Draw similar pictures for the time it takes to converge after the crash.



Question 5: Link State Routing and Crashes (*Optional, Exam Practice Only*)

Consider the topology shown below. This is the same topology we used in the slides to show the count up phenomenon for distance vector when both the links from **R4** to **R2**, and the link from **R4** to **R3** went down. We want to see why the count up phenomenon *does not happen with link state*.



Questions

1. **Normal Operation** (5 points)

Draw the link state packets sent by **R4**, **R5** and **R6** before any link crashes.

R4's LSP		R5's LSP		R6's LSP	
R2	10	R4	1	R4	1
R3	1	R6	5	R5	5
R5	1	BioE		MechE	
R6	1				

2. **Double Crash** (5 points)

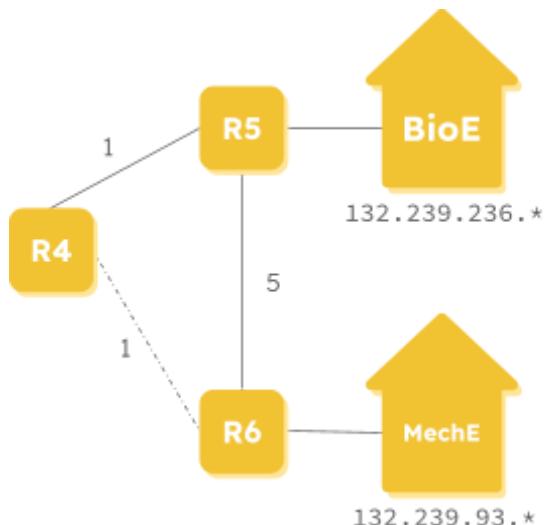
Draw the link state packet sent by **R4** after the links from **R4** to **R2**, and from **R4** to **R3** crash (the links from **R5** and **R6** stay the same).

R4's LSP	
R5	1
R6	1

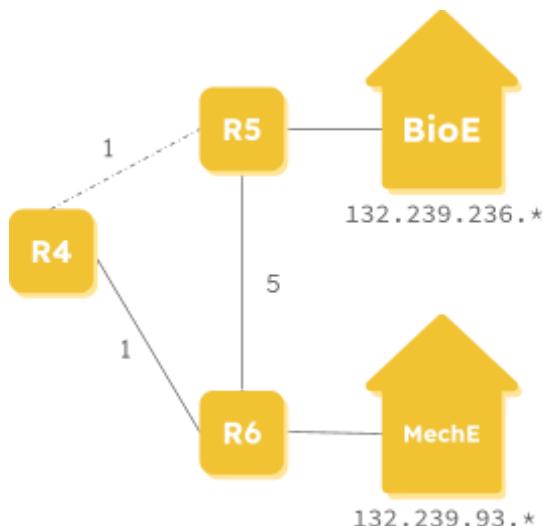
3. Dijkstra Tree (15 points)

Assume this new LSP from **R4** gets to **R5** and **R6** in a few msec. Show the Dijkstra tree at **R5** and **R6**, and explain why after **R5** finishes its Dijkstra calculations (after a few msec), it will conclude that **ECE** is unreachable even though it still has link state packets from **R1**, **R2**, and **R3**.

Tree at R5:



Tree at R6:



These routers won't see any new link state updates from the other routers, concluding that they're unreachable; the only link states that are updated are from **R4** and **R6** (or **R4** and **R5** for **R6**).