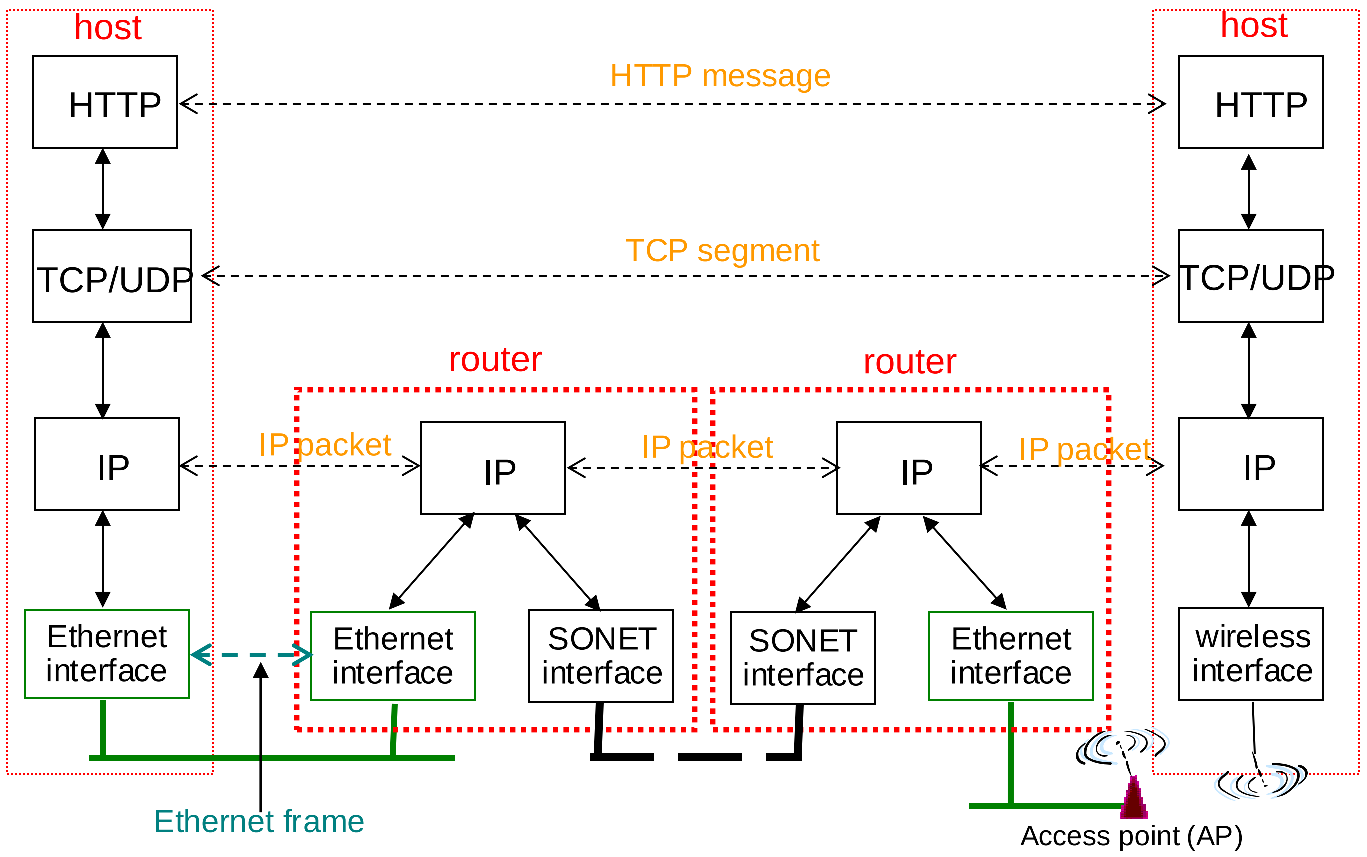
CS118 Lecture 17: Course Review

The big picture



Where is DNS?

Where is BGP?

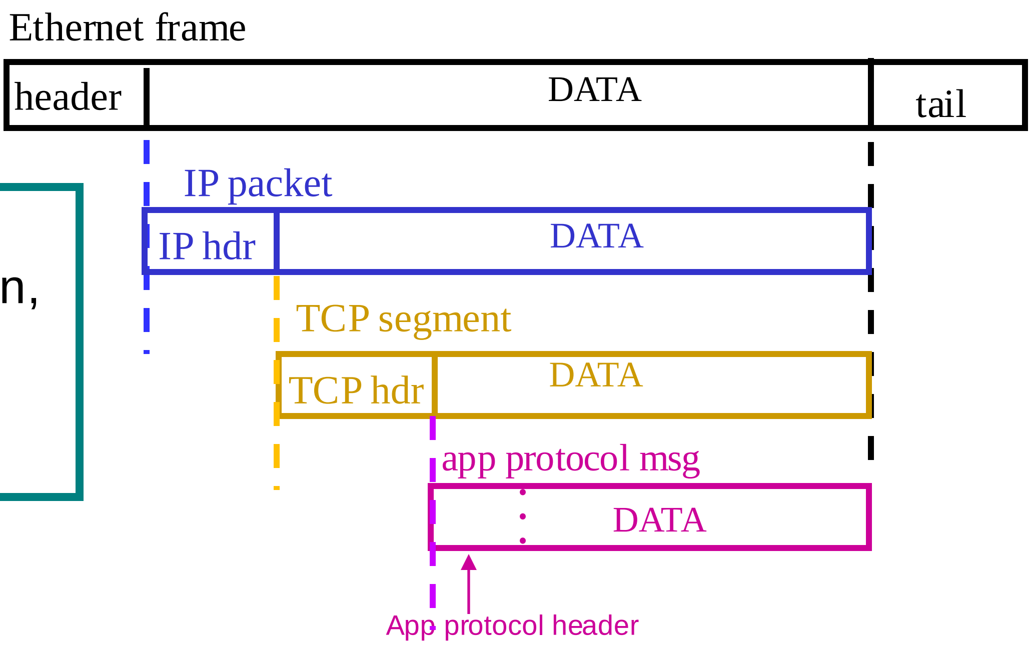
Where is ARP?

Do switches run a protocol to perform self-learning?

Layered protocol implementation

A protocol defines:

- the format of message exchanged between peer entities
- the actions taken upon receiving the message

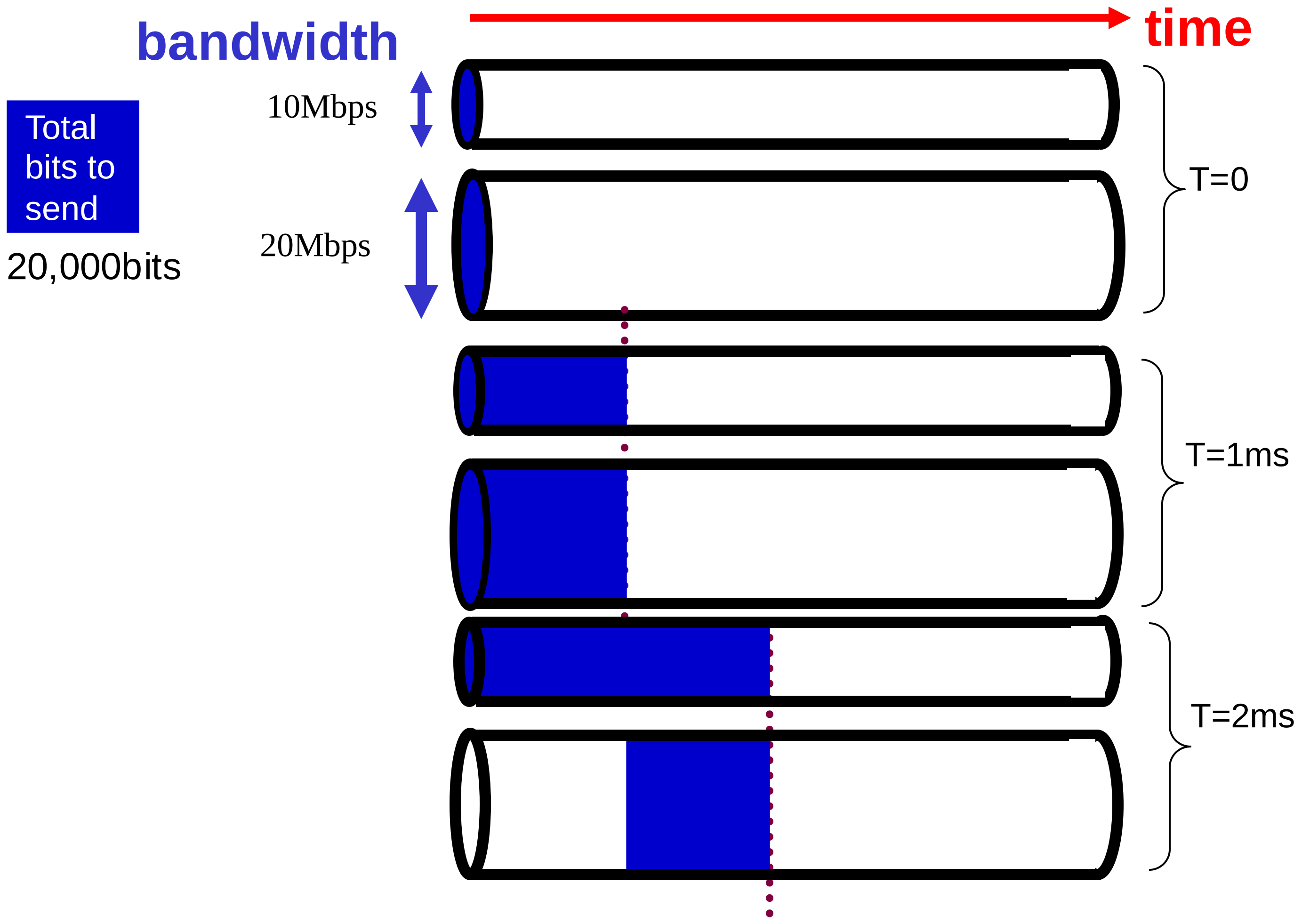


What's in the header of a protocol: all the information, and *only* the information, that's needed for the protocol's functionality

Doing a bottom-up review

- Physical layer: know how to calculate the delay of sending packets from one node to another
  - Transmission rate (bandwidth)
  - Transmission delay
  - Propagation delay
- Link layer: move data between two *directly* connected nodes
  - Multi-access protocols: Aloha, Ethernet (CSMA/CD)
  - MAC address, Address Resolution Protocol
  - Self-learning switch
- Network layer: IP, routing, NAT, tunneling
- Transport layer: UDP, TCP, QUIC, congestion control
- Application layer: HTTP, DNS (BGP, OSPF-although it runs directly over IP)

Bandwidth, transmission delay, propagation delay



Data link layer

- The concepts of (1) framing; (2) byte stuffing
- Link layer address: MAC addresses (Medium Access Control)
  - 48-bit, flat address space, no relation to IP address
- Multiaccess channels (e.g. WiFi, Ethernet) need a channel access protocol
  - Ethernet: CSMA/CD, collision resolution by exponential backup
  - WiFi: CSMA/CA, collision avoidance
- Self-learning switch: maintains a frame forwarding table
  - Each forwarding table entry contains:

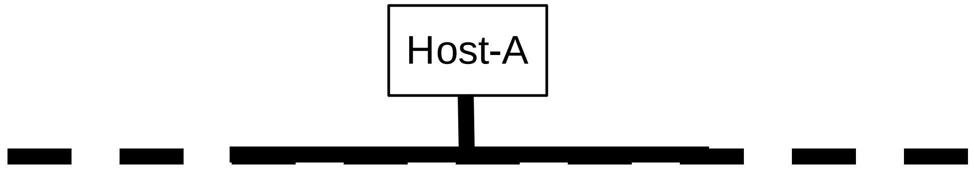
```
[MAC address, port, TTL]
```

When a switch receive a data frame:

```
▪ If the source MAC address not already in the table: add it
▪ If the destination MAC is found in the table
  then {
     if destination is in the same direction as received frame
       then drop the frame
     else forward frame to the port indicated by the entry
   }

   else flood   /* forward to all the interfaces except the arriving interface */
```

Lets try an Ethernet example



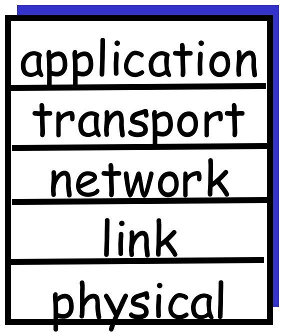
- Send a frame, detect collision: insert jam single, then stop
- What to do next?
- When try again, collide again...
- How to decide the waiting period after colliding the 3rd time?
  - What is the probability to wait for exactly 7 time slots?

Interconnecting LANs

Q: Why not just one big LAN?

- Signal attenuation; limited physical distance
- all stations on one LAN share bandwidth → limited total throughput

- A number of different types of "connectors"
  - router (already mentioned, L3)
  - bridge, Ethernet switch (L2)
    - Frame forwarding
    - Perform CSMA/CD
  - repeater, hub (L1)
    - bit level forwarding
    - Signal amplification



Self-learning switches: an example

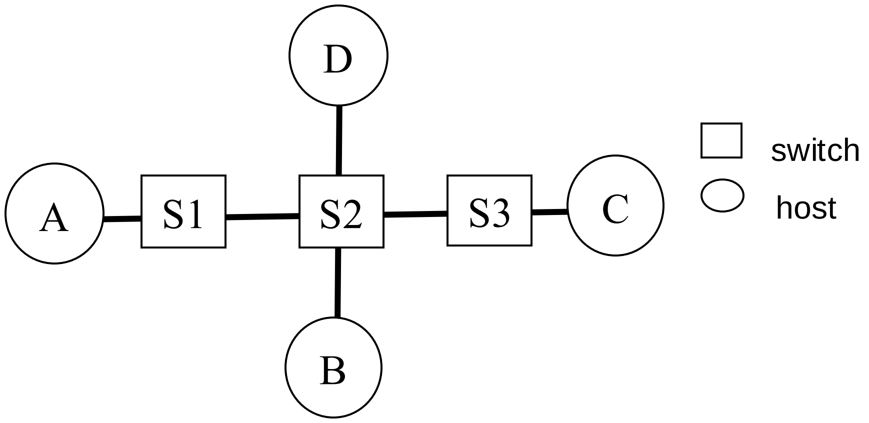
- All switches start with empty forwarding table
- Consider the following frame transmission:

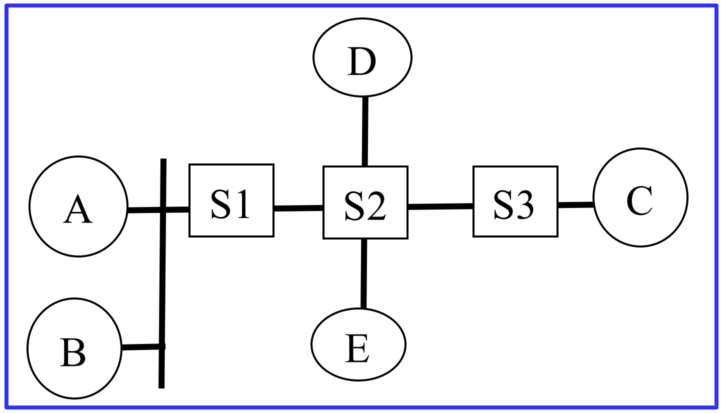
  $A \rightarrow B$, $B \rightarrow A$
  $B \rightarrow C$, $C \rightarrow B$
  $D \rightarrow C$, $C \rightarrow D$

Q: How many times does **S2** need to flood the data frame?

A→B: S2 flood

B→C: S2 flood

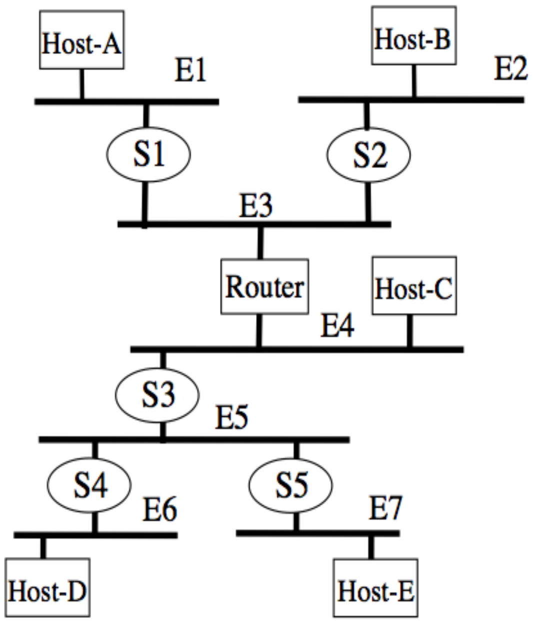




If we change the picture a bit: does the answer change?

Practice Questions

- Will Host-B ever send an ARP request asking for the MAC address for Host-C?
- Assume that Host-B and Host-C are accidentally assigned the same MAC address, will this error affec data delivery from any othe host to either Host-B or Hos C?
- Will Host-D and Host-C have the same subnet mask?

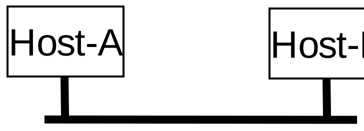


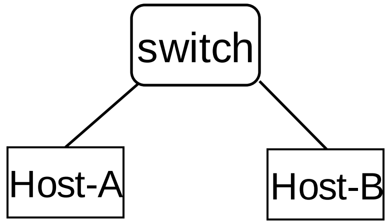
Connecting L2-L3: Address Resolution Protocol (ARP)

Given an IP address, find the corresponding MAC address

- Host A sends an IP packet to Host B on the same Ethernet
  - A knows B's IP address
- To find B's MAC address: A broadcasts *ARP query*
  - which contains B's IP address (also contains A's IP & MAC address)
  - broadcast MAC address = FF-FF-FF-FF-FF-FF
- B receives ARP query, send ARP reply
  - adds A's IP-to-MAC address mapping in its ARP table
    - *All IP nodes that heard A’s broadcast can add this entry*
  - The reply sent to A’s MAC address (unicast)
  - The reply contains B’s MAC address
- A saves B’s IP-to-MAC address mapping until it times out

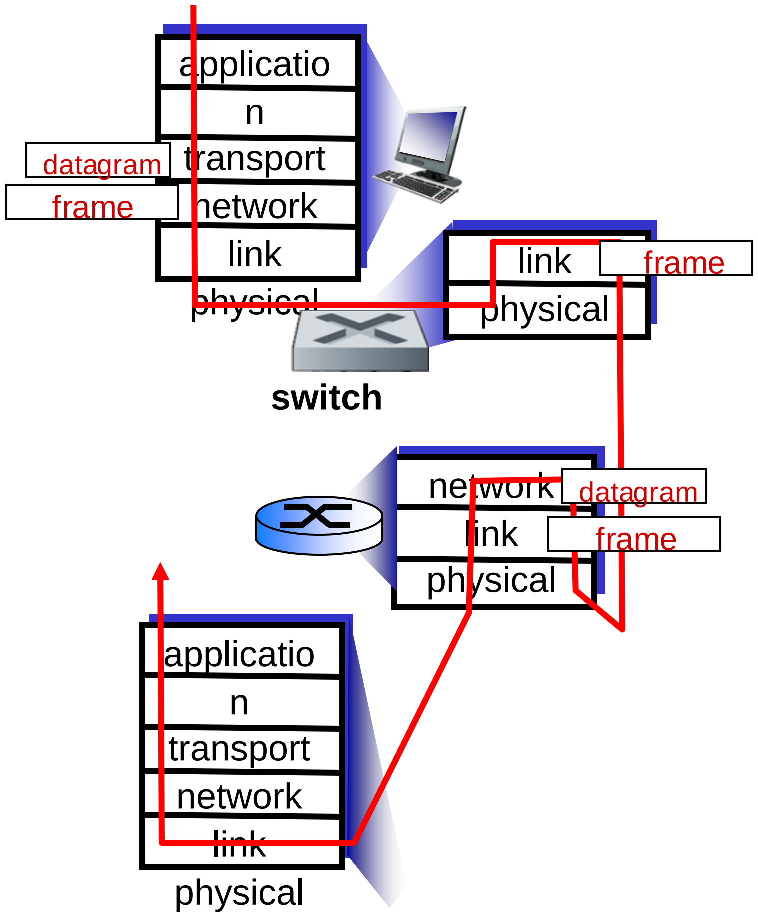
ARP Table entry: <IP address; MAC address; TTL>





Switches vs. Routers

- both are store-and-forward devices
  - routers: network layer devices (examine IP headers)
  - Switches: link layer devices (examine Ethernet headers)
- How each builds forwarding table
  - routers run routing protocols
    - A router's forwarding table (also called FIB) is filled-in *proactively*
  - switches implement self-learning algorithms
    - A switch's forwarding table is filled in *reactively*

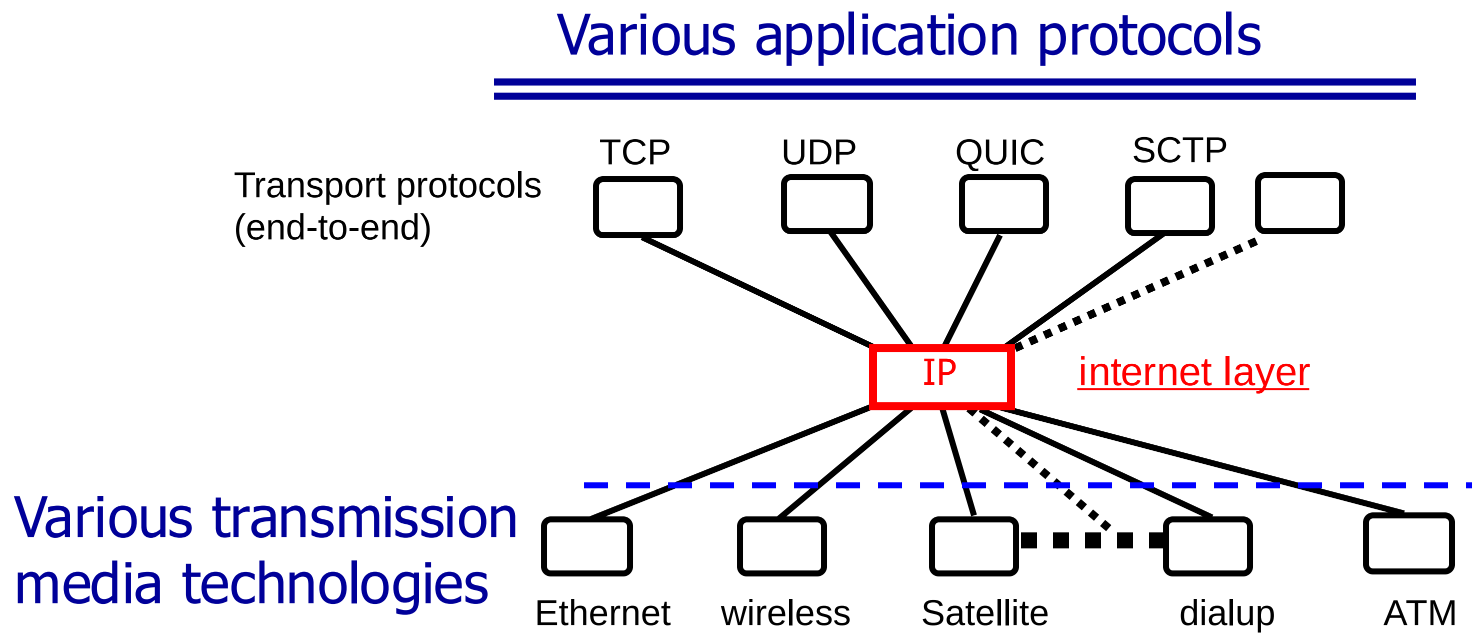


Make a summary: How many different tables?

that are used in packet forwarding

- Router: forwarding table (IP addr → interface)
- Both router and host: ARP table (IP addr → MAC)
- Switch: self-learned forwarding table (MAC addr → interface)

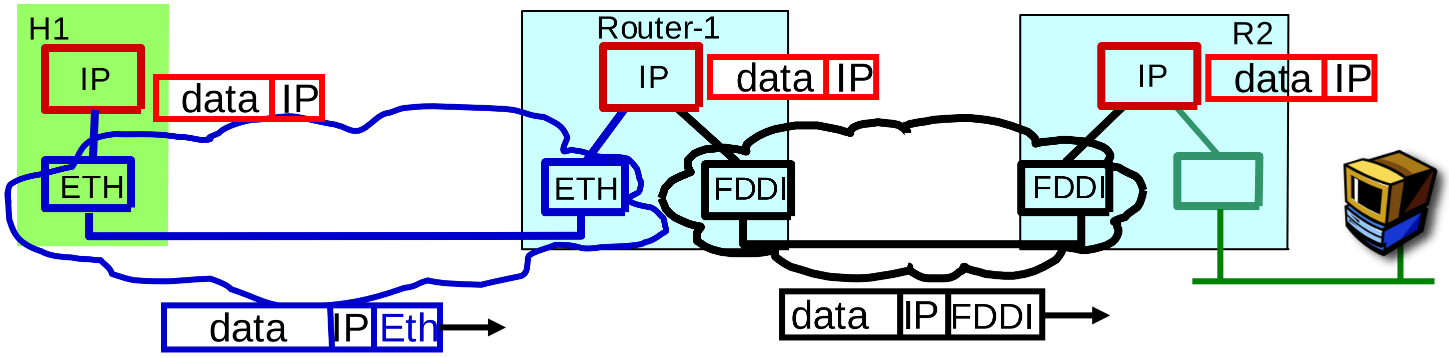
The picture of the world according to IP



Interconnection by encapsulation

- An IP packet is wrapped in a local network protocol to travel through that hop
- A router un-wraps the packet, then looks up its IP destination address
  - on an attached network: send to the destination directly
  - on a remote network: look up Forwarding table, send to the next hop router

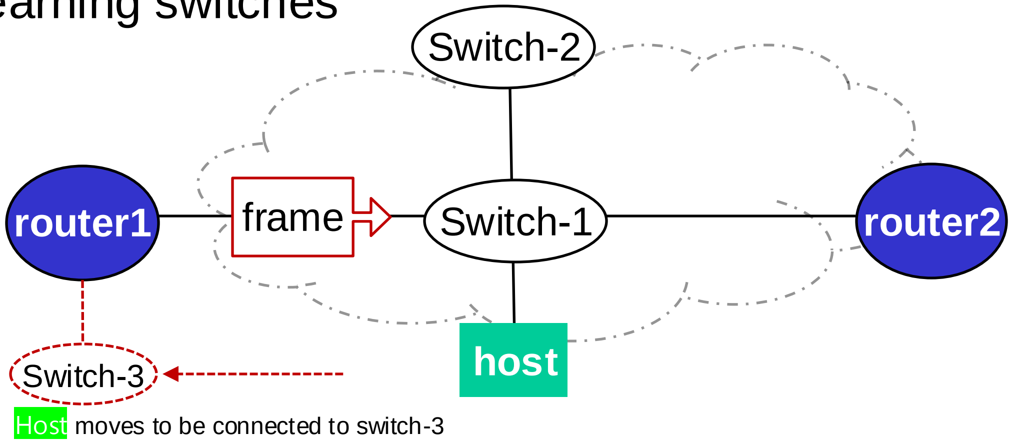
How does the router decide?



Q: a packet traverses 4 routers from source to destination, how many times does it get encapsulated and de-capsulated along the way?

Clarifying some basic concepts

- At layer 2: nodes are identified by static MAC addresses
  - a frame is forwarded from the Ethernet interface of one node to the Ethernet interface of another
  - The nodes can be hosts, switches, routers: all treated the same way as far as layer 2 functions are concerned
    - e.g. Ethernet protocol, or self-learning switches



- At layer 3: node(interface)s are identified by IP addresses
  - Each subnet (a layer-2 network) is assigned an IP address block
  - A host moving from one subnet to another requires *reconfiguration*
    Among 4 configured parameters (IP addr, mask, default router, DNS caching resolver IP addr)
    - *what must change?*
    - *What may change?*

Internet Protocol

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |u|D|M|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                              data                             /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                        IP packet format
```

Basic header (20bytes)

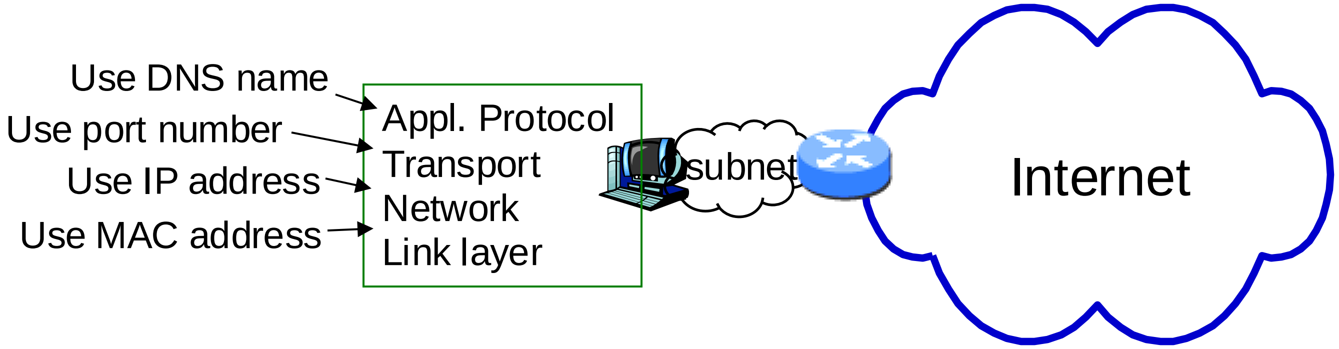
- What is the purpose for each header field?
  - e.g. why do we need IHL, TTL?
- the unit of counting for header length, total length, fragment offset fields, and why
- IP header format stays the same since 1981; the interpretation of IP addresses has changed multiple times
  - subnet, Classless Inter-Domain Routing, private addresses

IPv4 address structure

- 4 bytes
- Hierarchical (i.e. not flat, as MAC addresses are)
  - network ID, host ID
- What is the boundary between these 2 parts? A few answers
  - history: Classful address
    - classes A, B, C: network ID is determined by the value of the first few bits (lecture-10, slide-15)
  - Subnet
  - Classless Inter-Domain Routing (CIDR)
- subnet mask: indicates the portion of the address considered as “network ID” *by the local site*
  - subnets are invisible outside the local site

Necessary Information for a host to get Internet Connectivity

- An IP address (for the host itself)
- IP address(es) for local DNS resolver
- IP address for Default router
- Subnet mask



Following an IP packet from source to dest.

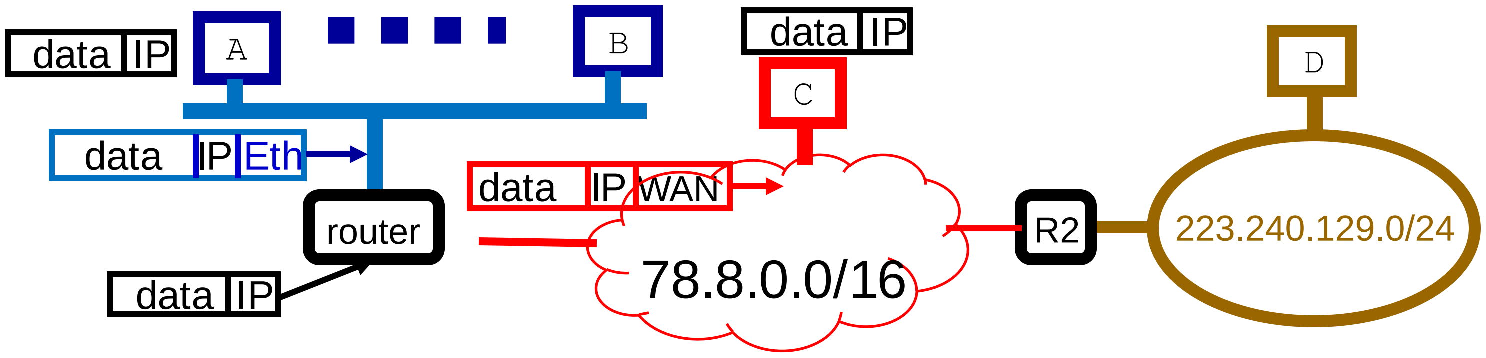
Source host A first uses subnet mask M to figure out whether dest. host is on the same network

3 examples:

1. Dest. = B: find B's MAC address, send data

2. Dest. = **C**, A sends packet to its default router

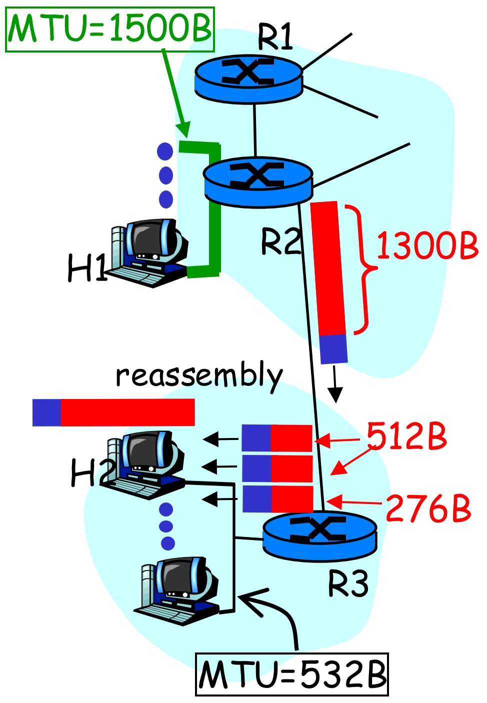
3. Dest. = **D**:



IP Fragmentation & Reassembly

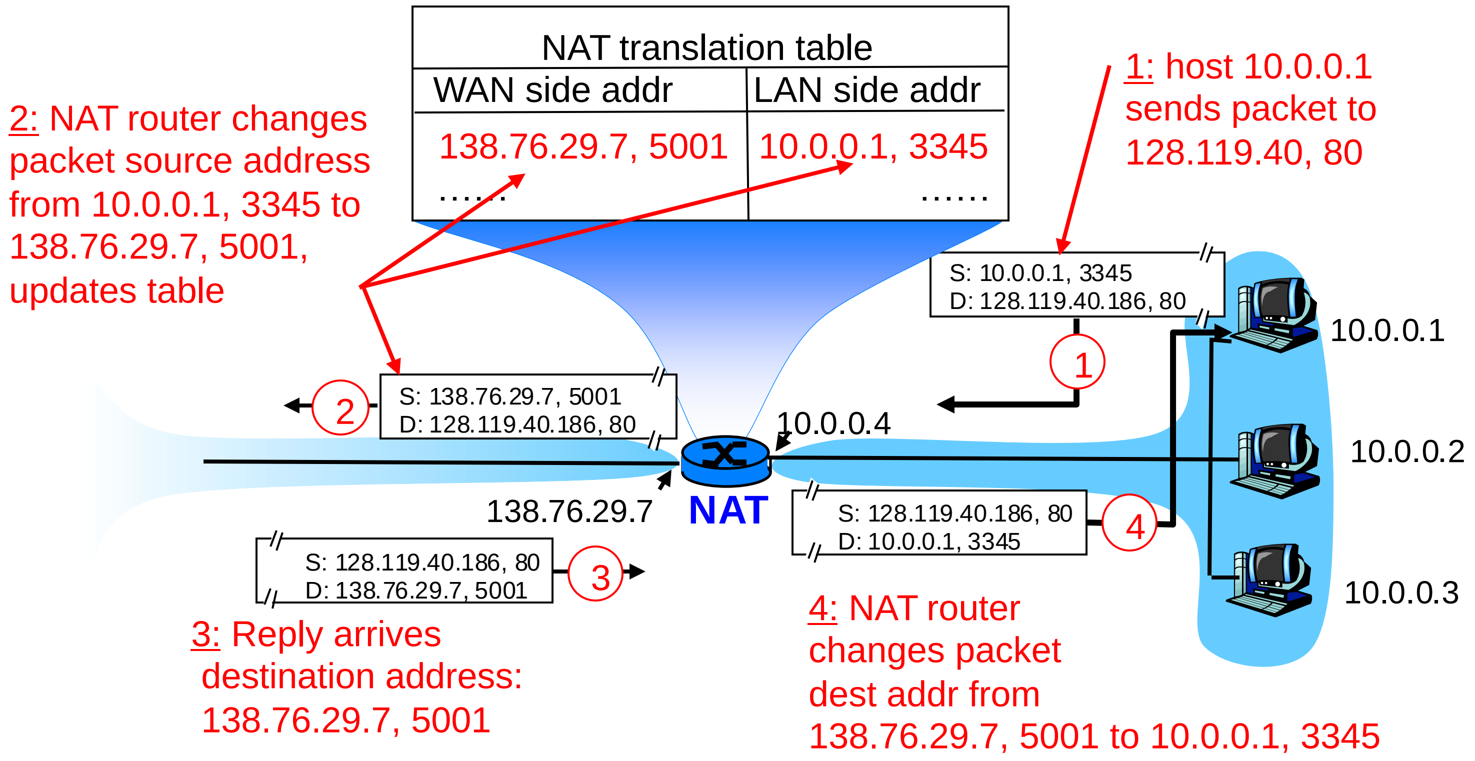
- Different **links** have different Maximum Transmission Unit (MTU)
- Sending host uses its local MTU size
- if the next link has a smaller MTU, routers fragment IP packets
  - chop packets to the MTU size of next link
  - further fragmentation down the path possible
- packet fragments are reassembled at destination host
  - Receiving host sets a timer when receiving the first segment of an IP packet
  - When assembles a full packet: pass to transport
  - When timeout: delete received segment(s)

H1 sending an IP packet of 1300 byte data to H2:



NAT: Network Address (and port) Translation

Would the NAT table overflow after running for a long time?

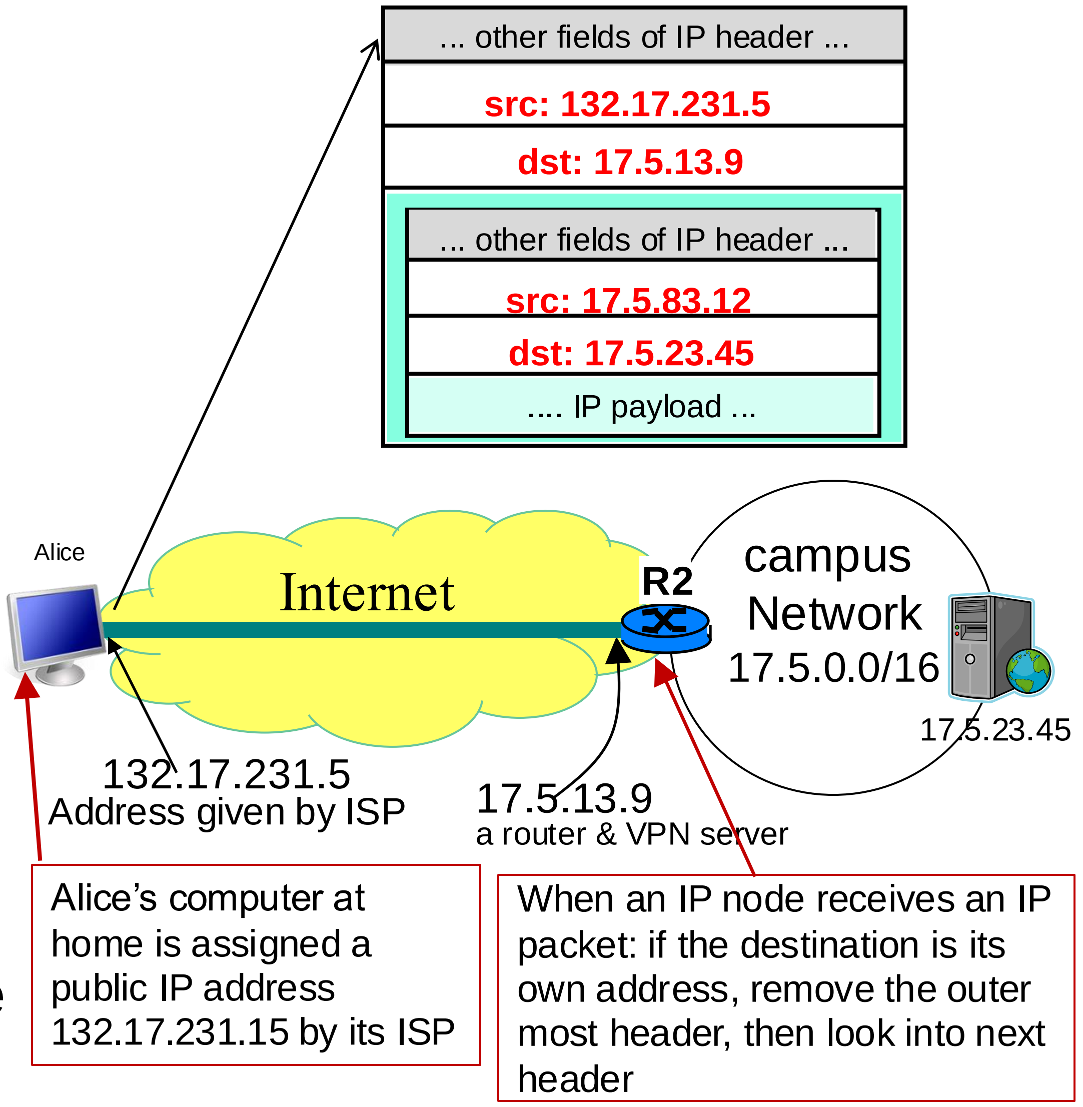


FYI:
- RFC 6056 says that the range for ephemeral ports should be 1024–65535.
- Internet Assigned Numbers Authority (IANA) and RFC6335 suggests the range 49152–65535

IP-in-IP: IP tunneling

Example:

- the campus server only allows access by computers on-campus
  - i.e. host addresses within the range of 17.5.0.0-17.5.255.255
- Alice can gain access at home using IP-in-IP tunneling
  - Obtain a campus address 17.5.83.12
  - Encapsulate packets with the ISP address as source



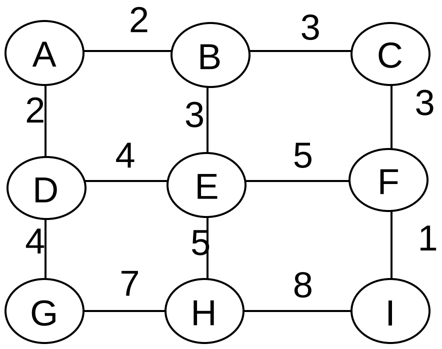
Alice's computer at home is assigned a public IP address 132.17.231.15 by its ISP

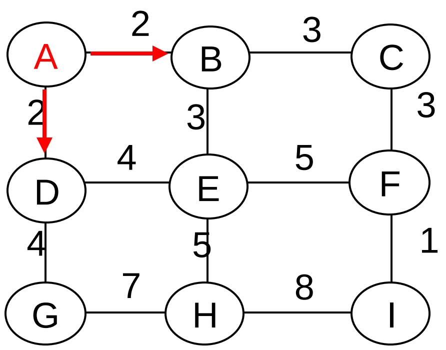
When an IP node receives an IP packet: if the destination is its own address, remove the outer most header, then look into next header

Routing

- Distance vector (DV)
- Link-state algorithm (LS), OSPF
- Comparison of LS and DV routing
  - distance vector: updates go to direct neighbors only; an update contains one’s distances to all the destinations
    - each node only knows distances to destinations, not the network topology
  - link state: updates broadcast to entire network; an update con one’s link distance to all direct neighbors
    - each node knows entire topology map
- BGP: updates go to BGP-connected neighbor nodes; update contains prefix reachability announcements
  - Which prefix sent to which neighbor: determined by policies
  - BGP has no routing metric

An example: OSPF update propagation



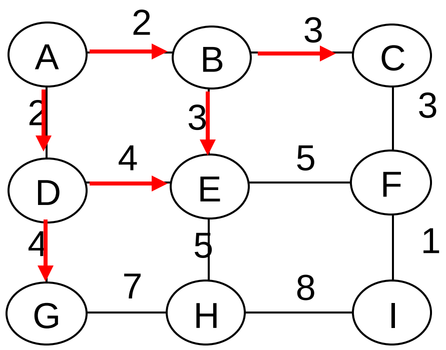


A sends its link-state update to B & D

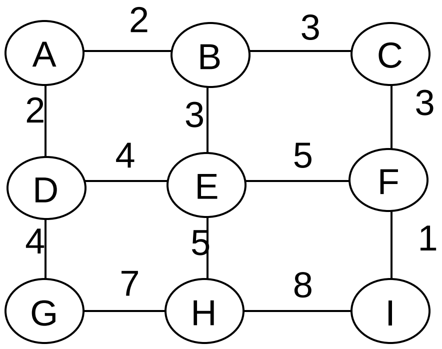
B & D each "*forward"* A's msg to their own neighbors

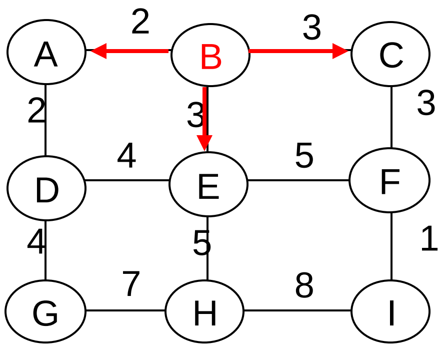
OSPF updates are sent reliably over each hop

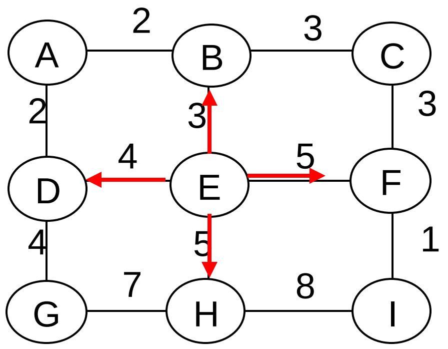
- Receiver sends ACK
- Without ACK, sender retransmits



An example: distance vector update propagation







B sends its distance vector to A, C, E

E (B's other neighbors too): does B's update change my distance vector?

- No: do nothing
- Yes: send my changed update to neighbors

Routing protocols

All routing protocols:

- Monitor connectivity between adjacent nodes
  - 2 BGP neighbors use TCP to send routing updates
  - in the absence of routing updates, they also send periodic Hellos to each other over this TCP connection
- Send updates upon failure detection

in addition to updates triggered by failures ,OSPF and distance vector protocols also sends updates periodically in the absence of failures

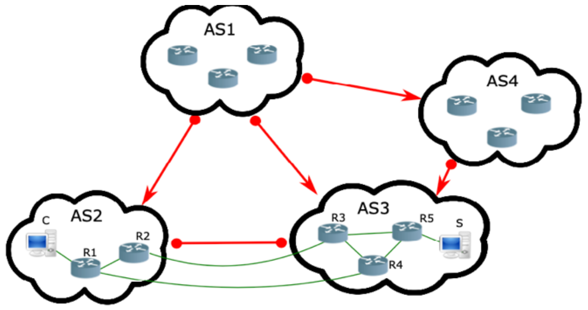
Monitor connectivity between adjacent nodes

- OSPF: send periodic Hellos
- Distrance Vector: send updates periodically
- BGP: send Keep-Alive messages periodically

Handling routing packet losses

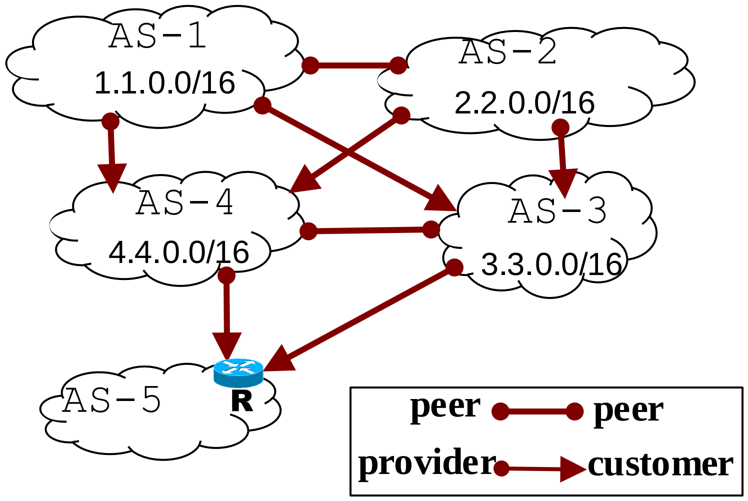
- OSPF: hop-by-hop ACK
- Distance Vector: send updates periodically
- BGP: set up TCP connection between neighbor BGP routers

Router R2 (AS2) sends an UPDATE message to R3 (AS3), but the packet carrying the message gets lost. What happens next? What does the BGP daemon at R2 do?



An exercise: BGP's Routing Policy

- Valid path: *can* be used if needed
- Preferred path: the path to use
- to reach destination prefix 3.3.0.0/16 in AS3
  - List all valid path(s) AS-1 can take
  - List all valid paths that AS-5 can take.
- Considering the reachability to destination prefix 4.4.0.0/16,
  - List *all* the valid path(s) that AS-1 can take.
  - Among these path(s), which valid path does AS-1 prefer the most?



AS-1 to 3.3.0.0/16: all valid paths

- AS1 → AS2→AS3
- AS1 → AS3

AS-5 to 3.3.0.0/16: all valid paths

- **5**-3, 5-4-3, 5-4-1-2-3, 5-4-1-3, 5-4-2-1-3, 5-4-2-3

AS-1 to 4.4.0.0/16: all valid paths

- 1-4
- 1-2-4

Reliable Transport: basic concepts

- Needs 3 things: data identifier, ACK, retransmission timer (or trigger)
- Window flow control is used for reliable delivery
  - cumulative ACK(n): everything up to n has been received
- GBN (go-back-N): the earliest approach to reliable data delivery: when retransmission timer expires: send the whole window of data again
- Selective repeat: selectively repeat the lost segment(s)
  - Used together with cumulative ACK

Congestion Control (CC) Window Adjustment

- Two phases:
  - slow start: set CC window (cwnd) size to 1 *segment*
    - Start slow but *rapidly* increase CC window size
  - congestion avoidance
    - Slowly but continuously increase CC window size
- Use Slow-Start Threshold (ssthresh) to define the boundary between these two phases
  - When cwnd < ssthresh: in slow-start phase, increase cwnd quickly
  - When cwnd ≧ ssthresh: in congestion avoidance phase, increase cwnd by one segment per RTT

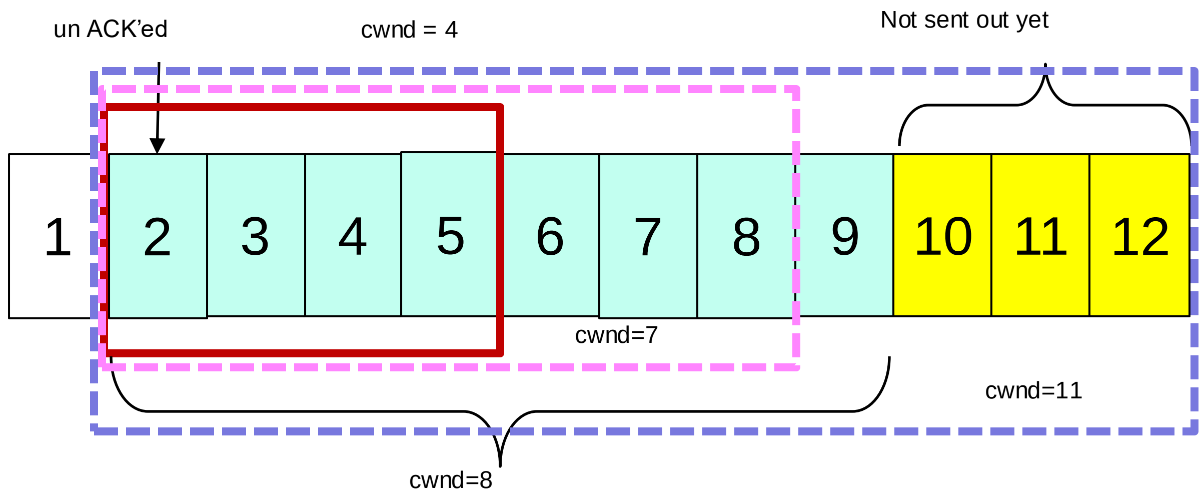
Congestion Avoidance

Objective: in steady state, the sender gently probe for unused resources

- Send cwnd packets
- If receives an ack
  - cwnd += MSS^2/cwnd
- If detect loss by 3 duplicate ACKs: *packets continue to arrived at receiver* → network not badly jammed
  - cwnd = ssthresh = cwnd / 2

Additive Increase, Multiplicative Decrease (AIMD)

Counting inflight packets



cwnd = 4, should allow 4 packets in the network

But we cannot slide the window to the right to allow more transmission

Why?

How to fix it 3 dup-ACKs inform us that 3 packets have been out of network
Inflate cwnd by 3 pkts→ cwnd = 4+3 = 7 (still nothing new can go yet)

Receive next dup-ACK (triggered by P6): cwnd =8: still can't send new packet
Receive next 3 dup-ACKs (triggered by P7-9): cwnd=11, sends P10-12

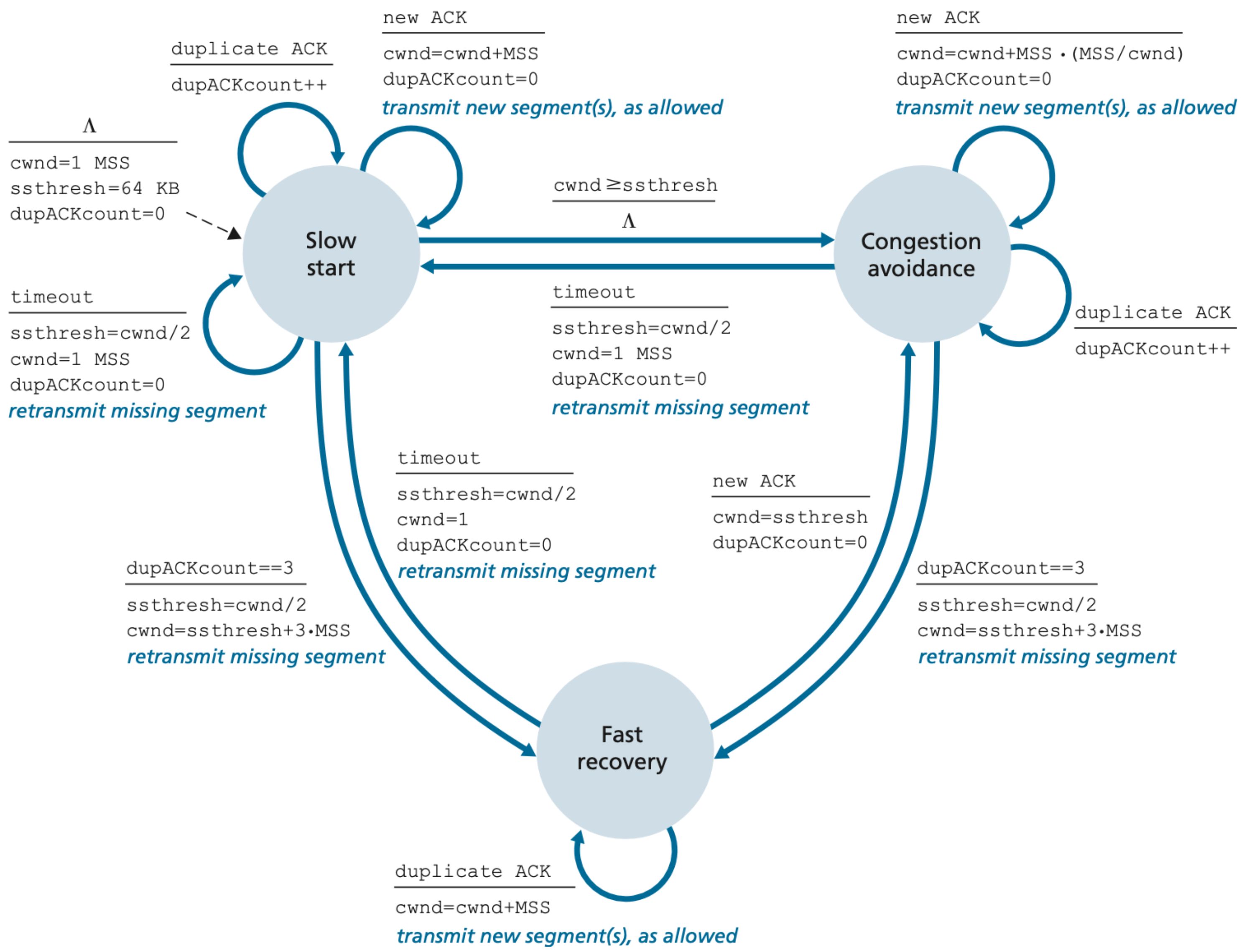


Figure 3.51 ♦ FSM description of TCP congestion control

Comparison between TCP and QUIC

- TCP: window flow control with cumulative ACK
  - (there exists TCP extension with selective ACK, not covered here)
  - Assign each data byte a sequence#
  - Sends data in segments; each segment carried in an IP packet
  - Congestion control window added later: tangled with flow control window
- QUIC: a QUIC connection
  - can contain multiple streams; each stream has its own sequence# and flow control window, sends data in frames
    - Each data frame is uniquely identified by its stream#+sequence#
    - A QUIC stream ≈ a TCP connection, a frame ≈ a TCP segment
  - assigns each QUIC packet (carrying data frames) a *packet sequence#*
  - Data sender keeps track of the mapping between packet# and data frames, sets a packet loss detection timer

How data is acknowledged

- TCP header has an ACK field, each segment carries the cumulative ACK to the other end
- QUIC uses a separate “ACK frame” (not limited in size),which carries
  - the largest packet number ever received
  - selective ACK: all the packet numbers that have been received
- How *each* QUIC stream achieves reliable delivery: in a way similar to TCP
  - puts all data frames allowed by its flow-control-window on QUIC connection’s transmission queue (to be carried in next outgoing QUIC packets)
  - If the packet carrying a frame F is lost, resends F (putting F into transmission queue again)

Application protocols: HTTP, DNS

- HTTP
  - Runs over TCP
  - Request / reply
    - Browser sends requests
    - server sends replies
  - data caching not in the original design
    - Early days: caching at user end
    - Today: caching by CDNs (Content Distribution Network)

  HTTP design has a number of other issues, the above are most relevant to network delivery

issues in matching the 2 layers (HTTP1.0, HTTP1.1, use of multiple TCP connections)

- DNS
  - Runs over UDP (by the original design; can run over TCP as well)
  - Request / reply
    - Stub resolver sends requests to local caching resolver,
    - caching resolver sends request to authoritative server
    - Server sends reply
  - Data caching is explicitly designed in

no issues in matching the 2 layers

DNS must take care of reliable resolution itself

Final

- Friday, March 21st, Classroom 3-6pm
- Focusing on the content starts from congestion control
  - Pre-midterm content mostly used as problem background
- Two pages of cheatsheets (i.e., one more than midterm)
  - Recommendation
    - Putting common protocol headers on it, say IP, UDP, TCP
- (tentative) AB version, 5-6 questions
  - 3-6 parts each
  - Basically, the same format with midterm, but
    - Calculation should be lighter
    - Points are more distributed

Distance Vector Protocol

Iterative, asynchronous:

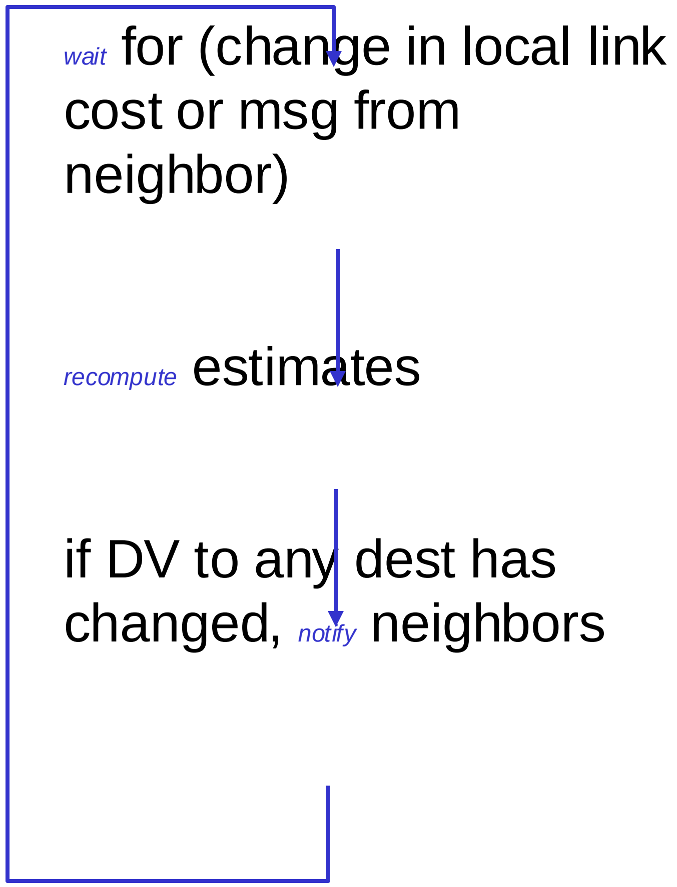
- Each local iteration caused by:
  - local link cost change
  - DV update message from neighbor
- Continues until no nodes exchange info.

Distributed:

- Each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

Asynchronous: nodes need *not* Exchange info/iterate in lock step

Each node:



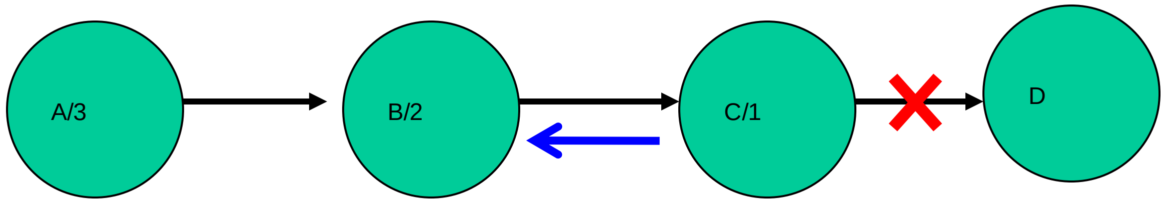
wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, *notify* neighbors

Count-To-Infinity Problem

- Assume we use hop count as metric
  - A uses B to reach D with cost 3
  - B uses C to reach D with cost 2
  - C reaches D with cost 1



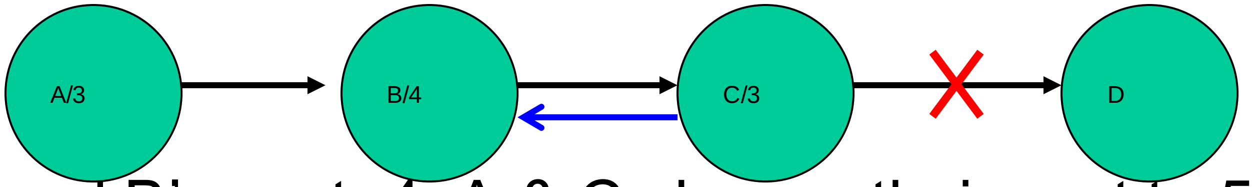
C sends to B the packets destined to D

- Suppose link between C and D breaks
  - Since B informs its neighbors its distance to D is 2, C switches to B, sets its cost to (2+1= 3)

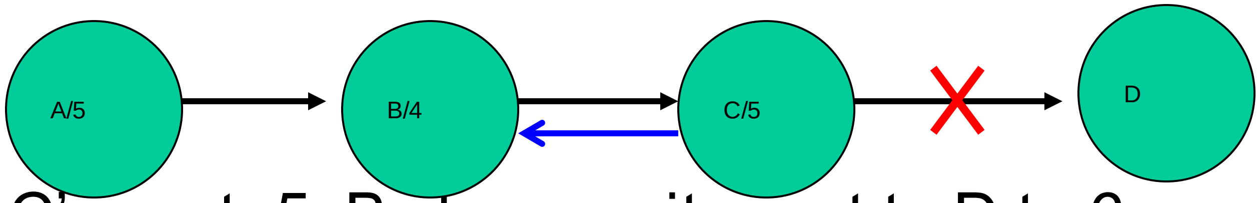
B's cost to reach D

Count-To-Infinity Problem (cont.)

- B’s path cost is now 4
  - A has not realized what has happened yet



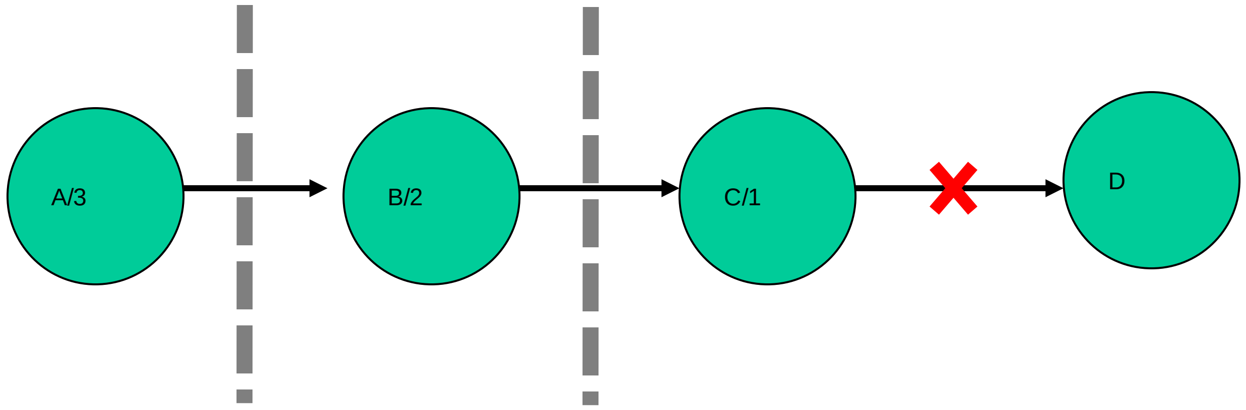
- Once heard B’s cost=4, A & C change their cost to 5



- B hears C’s cost=5, B changes its cost to D to 6
  - Cycle repeats, the distance “counting to infinity”
  - Meanwhile data packets from A to D loop between B and C

Split Horizon

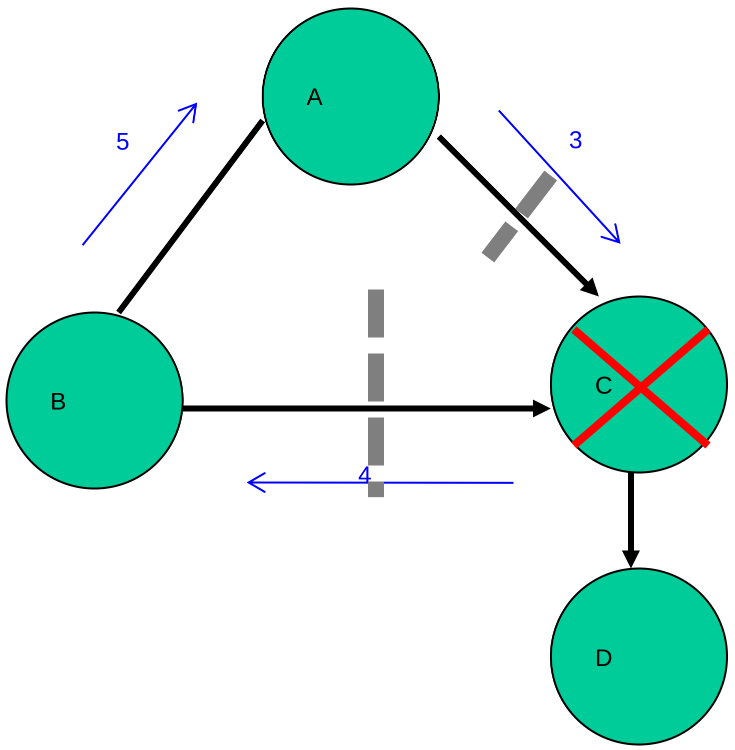
- Because B reaches D via C, B tells C nothing about node D
  - A tells B nothing about nodes C and D



A router should **not** advertise a route back to the same interface from which it learned it

Split Horizon: not effective in many cases

- Suppose the link between C and D breaks



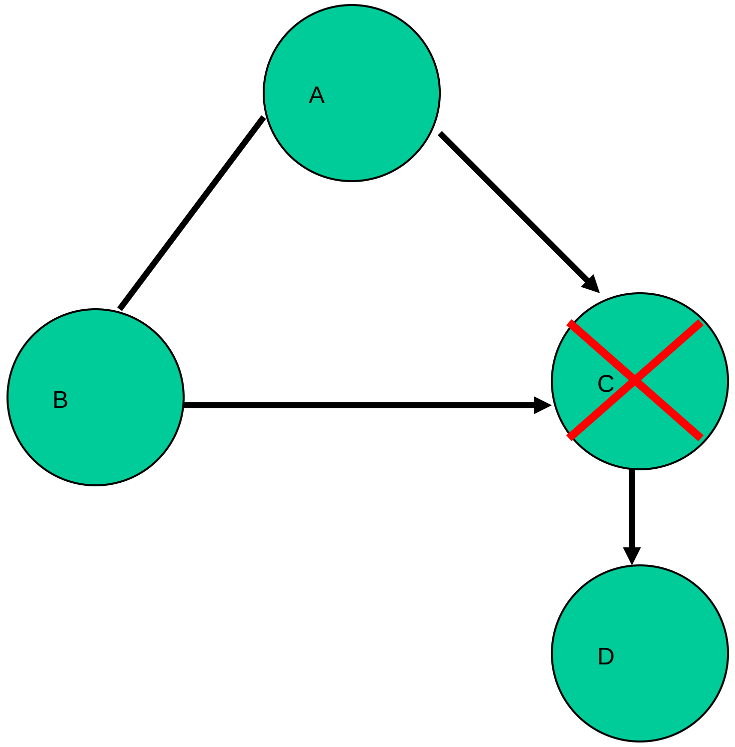
1. A and B do not tell C they can reach D
   - They do tell each other
     - A→B: my distance to D is 2
     - B→A: my distance to D is 2
2. When C fails
   - A sends to B the packets with destination=D
   - B sends to A the packets with destination=D

Packets can still loop

Split Horizon with poison reverse

important

- A & B go through C to reach D: both tell C that their distance to D is *infinite* (*poison reverse*)
  - C never attempts to reach D via A or B



When node C fails,

1. A tries to reach D via B: A tells B $D_D = \infty$
2. B does the same thing
3. Both A and B realize that D is no longer reachable, prevent packet looping.

Routing Information Protocol (RIP): a distance vector protocol. From specification https://datatracker.ietf.org/doc/html/rfc2453
"... Split horizon with poisoned reverse will prevent any routing loops that involve only two routers. However, it is still possible to end up with patterns in which three routers are engaged in mutual deception..."