CSCI-1680
Network Layer: Intra-domain Routing

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, David Mazières, Phil Levis, John Jannotti

Administrivia

- IP milestone meetings: Should meet with staff on/before October 4 (tomorrow)
  - Sign up link via email
  - Can't find a time? Make a private post on Ed!
- IP Gearup II tonight (10/3) 6-8pm, CIT368
  - Implementation/debugging stuff; bring questions!
- HW1 due tonight; HW2 out after next class

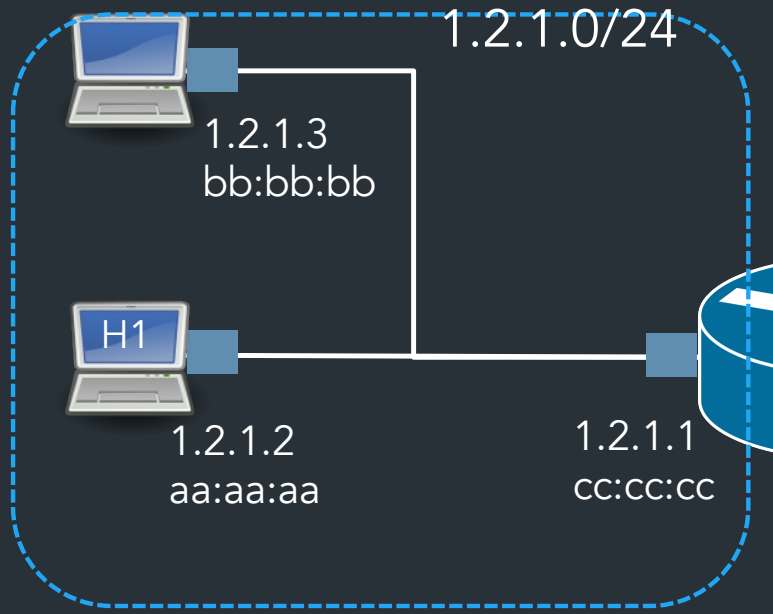
Today

Two things

- NAT
- Intro to routing, RIP

Warmup

What is the destination MAC address when H1 is sending the following packets?



1)

	Src	Dest
Link	aa:aa:aa	??? ← CC:CC:CC
IP	1.2.1.2	1.2.1.1

2)

	Src	Dest
Link	aa:aa:aa	??? BB:BB:BB
IP	1.2.1.2	1.2.1.3

H1's forwarding table:

Prefix	IF/Next hop
1.2.1.0/24	IF1
0.0.0.0	1.2.1.1

3)

	Src	Dest
Link	aa:aa:aa	??? CC:CC:CC ← NEXT HOP
IP	1.2.1.2	8.8.8.8 (Google) ← FINAL DEST OF PACKET

Recap: IP vs. Link-layer address

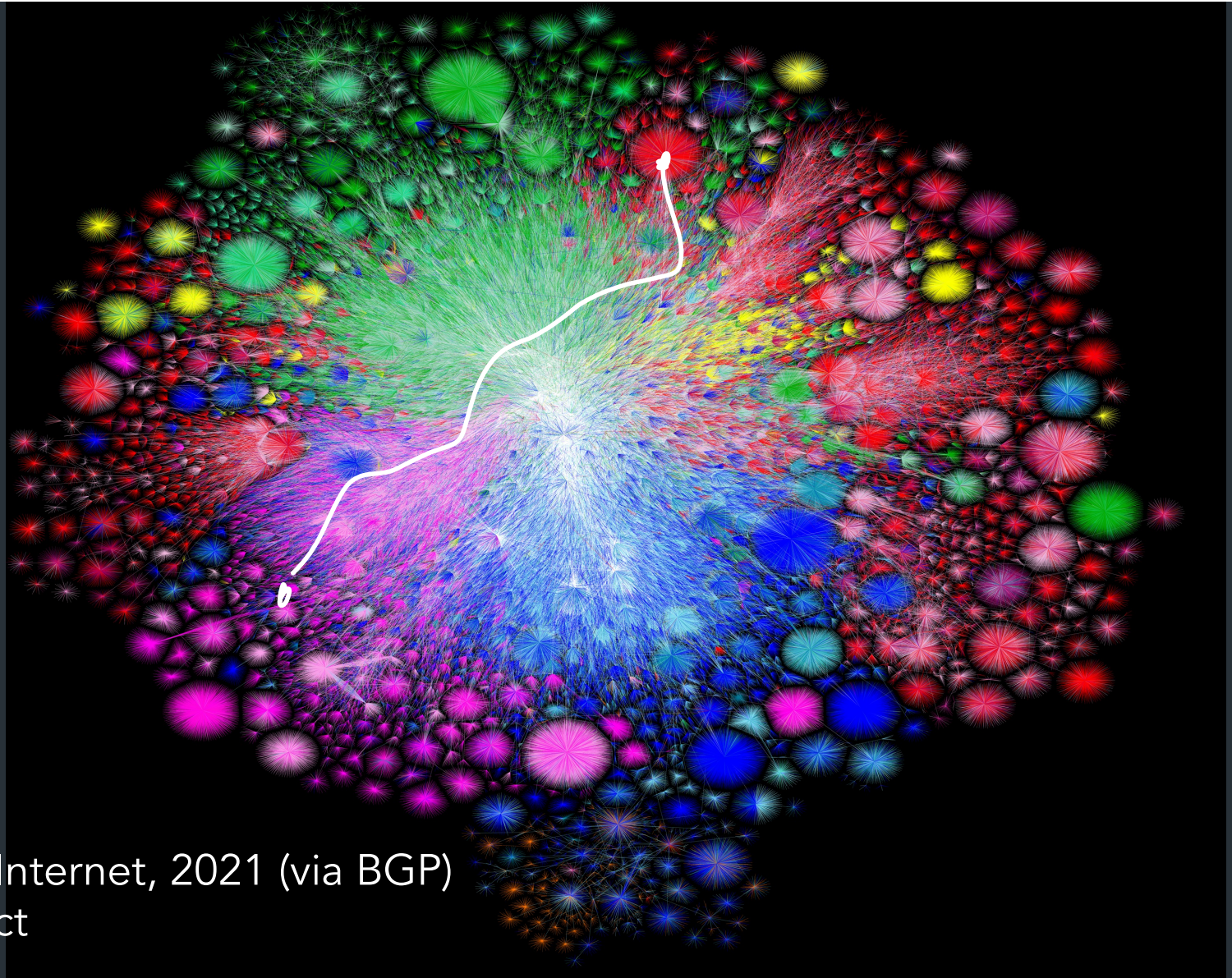
	Src	Dest
Link	aa:aa:aa	cc:cc:cc
IP	1.2.1.2	8.8.8.8 (Google)

Link-layer header info (Ethernet/Wifi/etc)

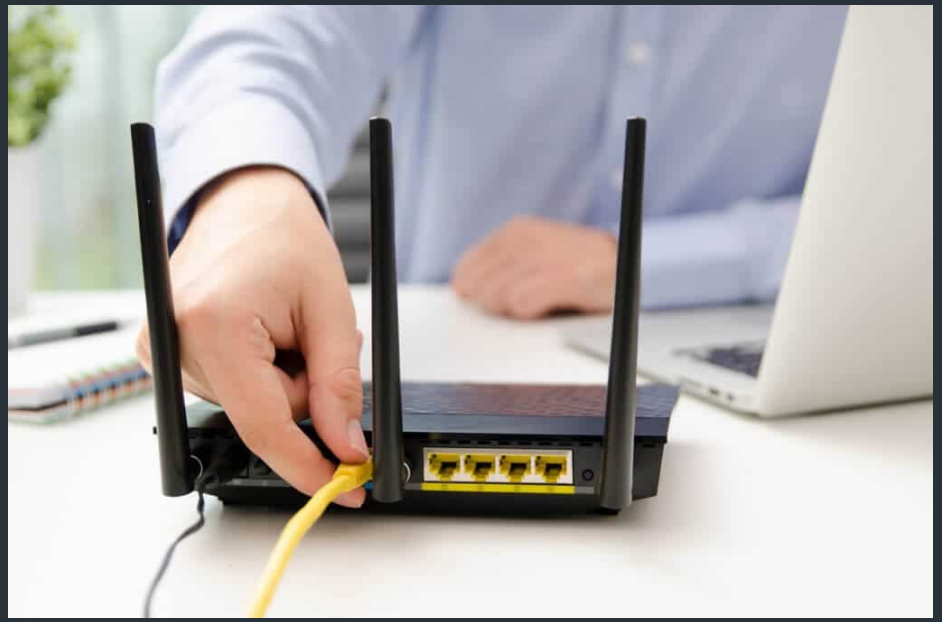
- Destination MAC address is link-layer addr for packet's next hop
- Changes every hop
- Each hop could use a different link-layer protocol!

IP header info

- Destination IP is IP address of packet's final destination
- Routers look at destination IP to figure out where packet goes next (and which MAC address goes on packet next)

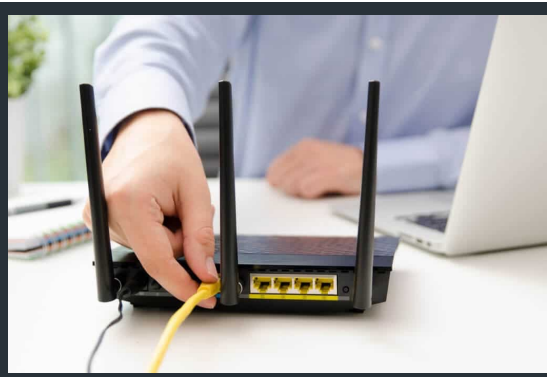


Map of the Internet, 2021 (via BGP)
OPTE project



… or does it?

Where it gets weird…



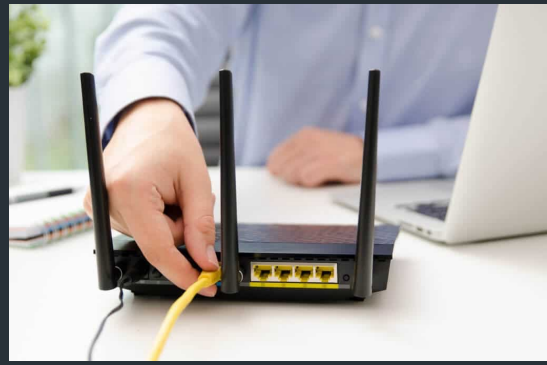
For many end hosts:

(IP assigned to your host) != (Your "public" IP)

(IP seen by other systems on the Internet)

NAT => NETWORK ADDRESS TRANSLATION

Where it gets weird…



You get just one IP from your ISP…

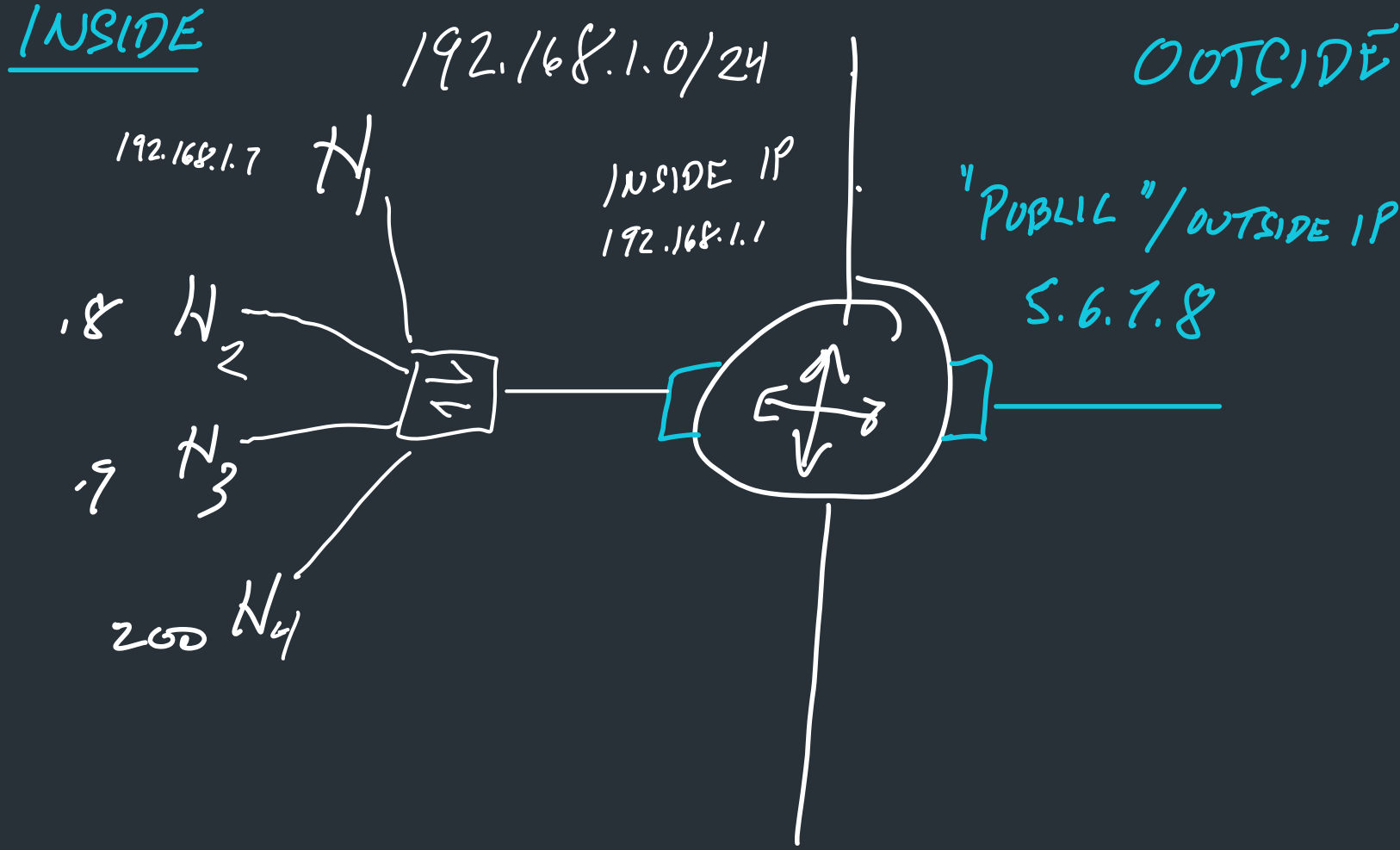
=> Need to share IP among many devices on the same network!

Solution: Create a "private" IP range used within local network

=> Routers need to do extra work to share public IP among many private IPs

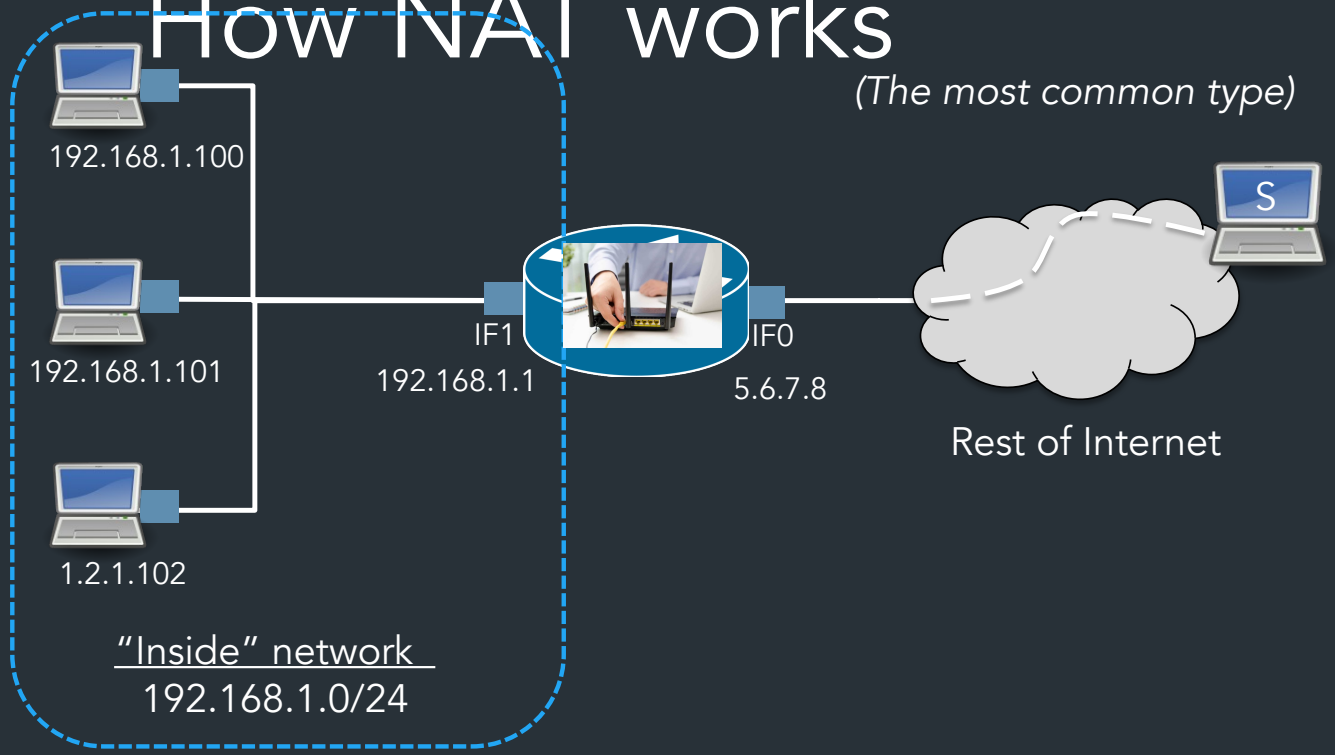
=> Network Address Translation (NAT)

(A form of connection multiplexing)


INSIDE
192.168.1.0/24
OUTSIDE
192.168.1.7
N1
INSIDE IP
192.168.1.1
"PUBLIC"/OUTSIDE IP
5.6.7.8
.8
N2
.9
N3
200
N4

How NAT works

(The most common type)



Goal: Share one IP among many hosts on a private network

Router *translates (modifies) packets from "inside" to use "outside" address*

Private IPs (RFC1918)

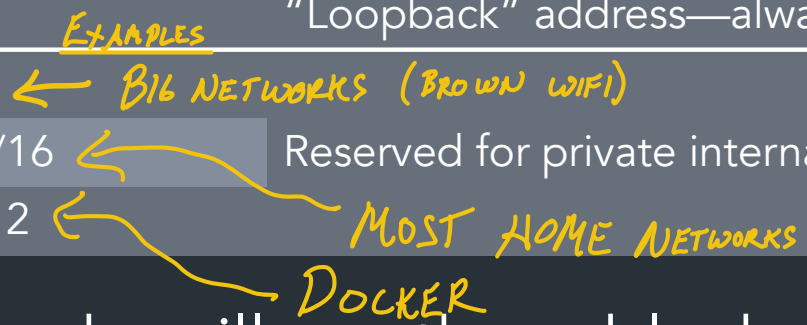
IP ranges reserved for "private" networks:

Prefix	Use
127.0.0.0/8	"Loopback" address—always for current host
10.0.0.0/8	Reserved for private internal networks (RFC1918)
192.168.0.0/16	Reserved for private internal networks (RFC1918)
172.16.0.0/12	Reserved for private internal networks (RFC1918)

Private IPs (RFC1918)

IP ranges reserved for "private" networks:

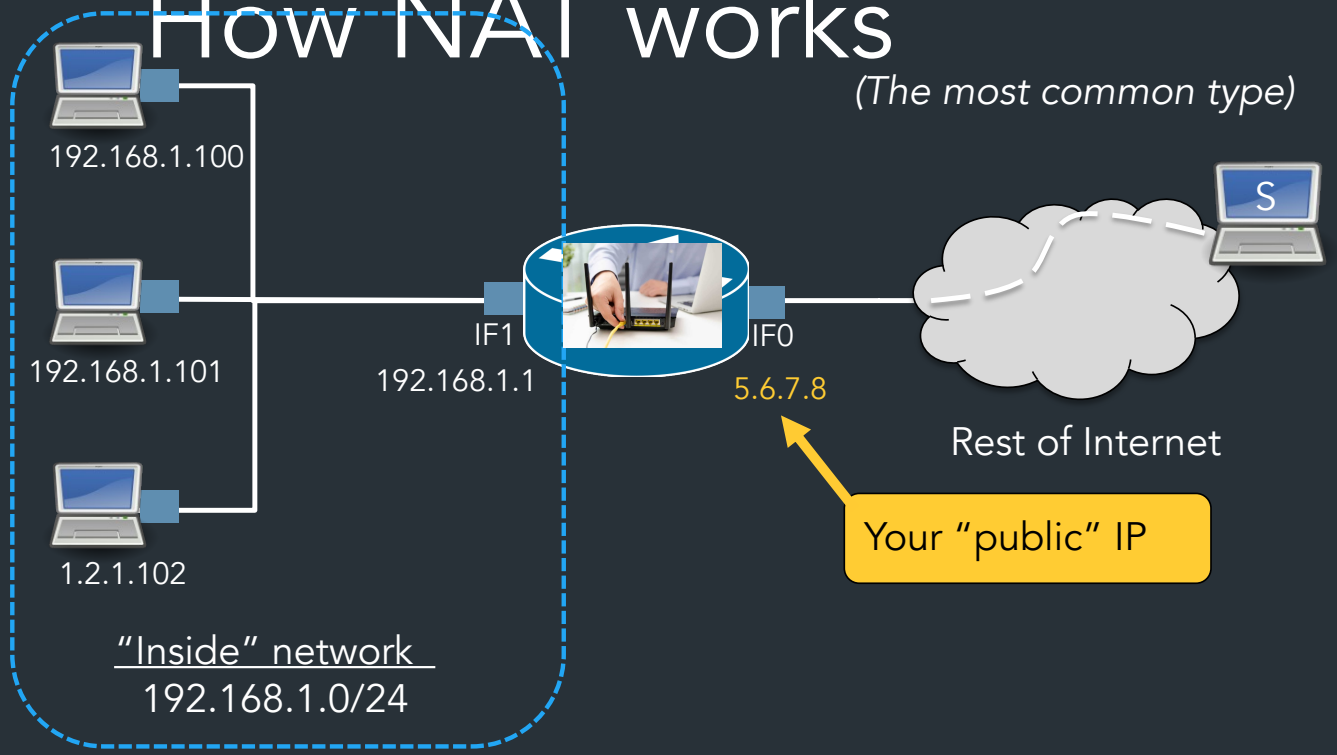
Prefix	Use
127.0.0.0/8	"Loopback" address—always for current host
10.0.0.0/8	Reserved for private internal networks (RFC1918)
192.168.0.0/16	
172.16.0.0/12	



- Many networks will use these blocks internally
- These IPs should never be routed over the Internet!
  - What would happen if they were?

How NAT works

(The most common type)

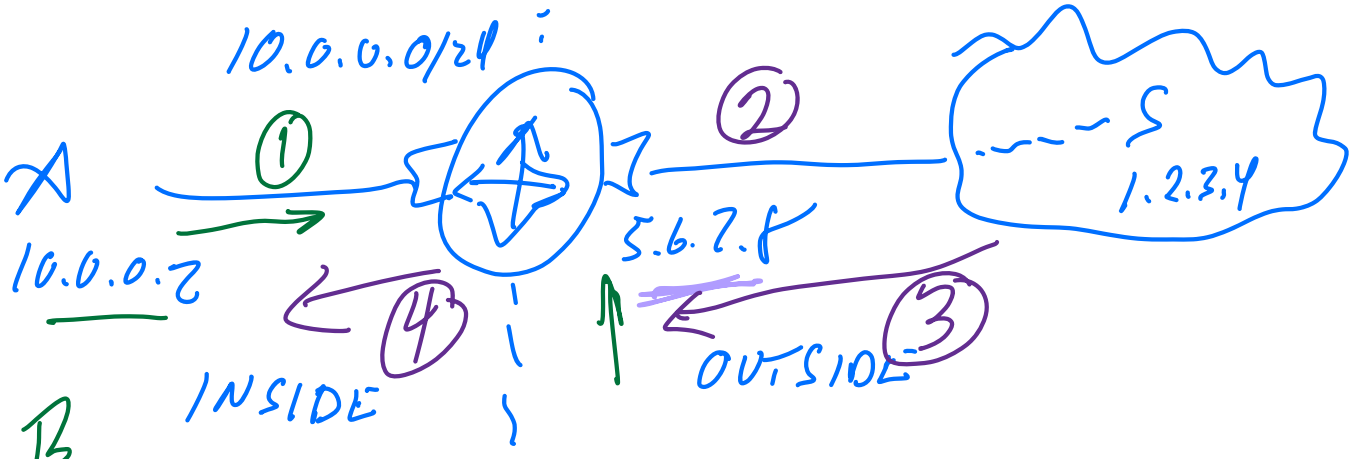


Goal: Share one IP among many hosts on a private network

Router *translates (modifies) packets from "inside" to use "outside" address*

=> Router needs to remember connection state

=> Router makes some (sketchy) assumptions about traffic

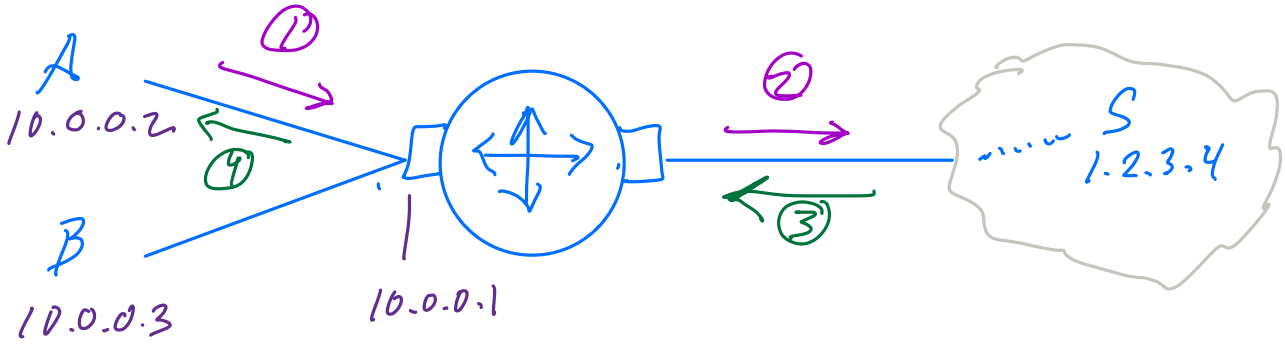


GOAL: A WANTS TO CONNECT TO S. ON PORT 80

	INSIDE		OUTSIDE	
	SRC	DST	SRC	DST
1	10.0.0.2:5555	1.2.3.4:80	5.6.7.8:7777	1.2.3.4:80
4	10.0.0.2:5555	1.2.3.4:80	5.6.7.8:7777	1.2.3.4:80

Router remembers: 5.6.7.8:7777 → 10.0.0.2:5555

NAT translation: an example



Suppose A wants to connect to S on port 80:

	INSIDE		OUTSIDE	
	SRC	DST	SRC	DST
①	10.0.0.2:5555	1.2.3.4:80		
	1.2.3.4:80	10.0.0.2:5555		
NAT → ②			5.6.7.8:7777	1.2.3.4:80
NAT ← ③			1.2.3.4:80	5.6.7.8:7777

ROUTER MUST REMEMBER:

5.6.7.8:7777 → 10.0.0.2:5555

Key points

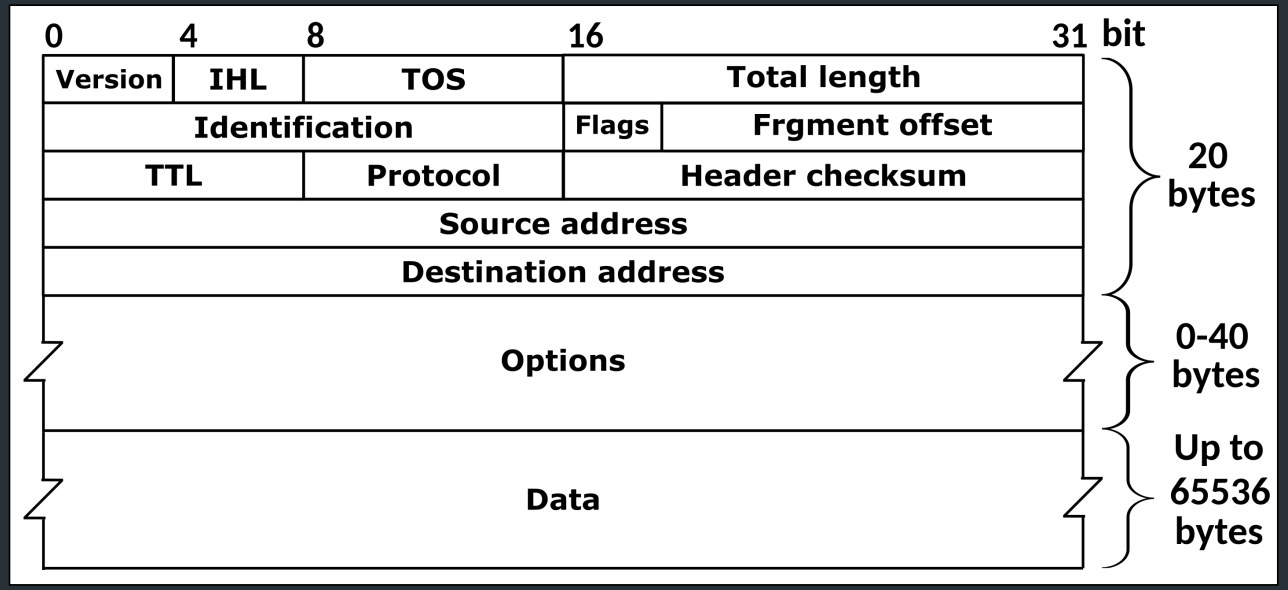
-- Router needs to keep track of state to remember how to "translate" packets

-- Can run out of possible translations (number of ports, space in table)
=> Big NATs (eg. Brown) have multiple outside IPs to increase number of possible translations

-- Router needs to figure out when connections start and end, so it can clean up table
=> Kinda works for TCP (protocol has well-defined start and end), sketchy for UDP (need to make assumptions based on timing)

Aside: About Port Numbers + NAT

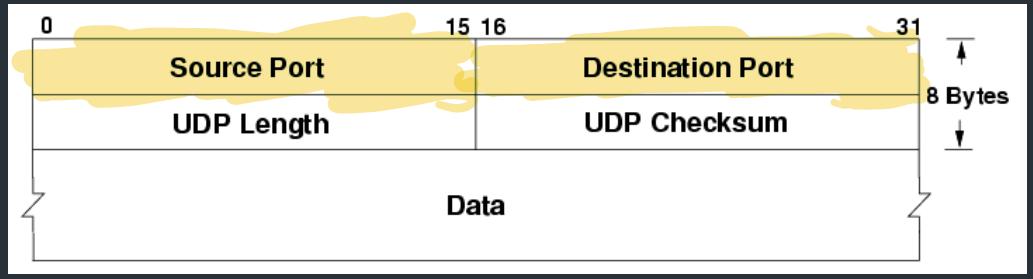
IP Header



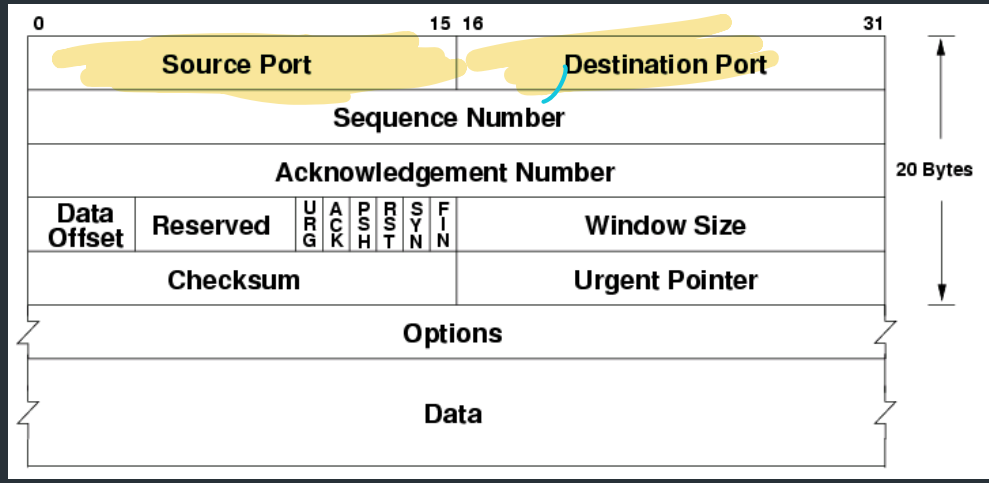
Where are the port numbers?????

… ports are actually part of the transport layer header!

UDP



TCP



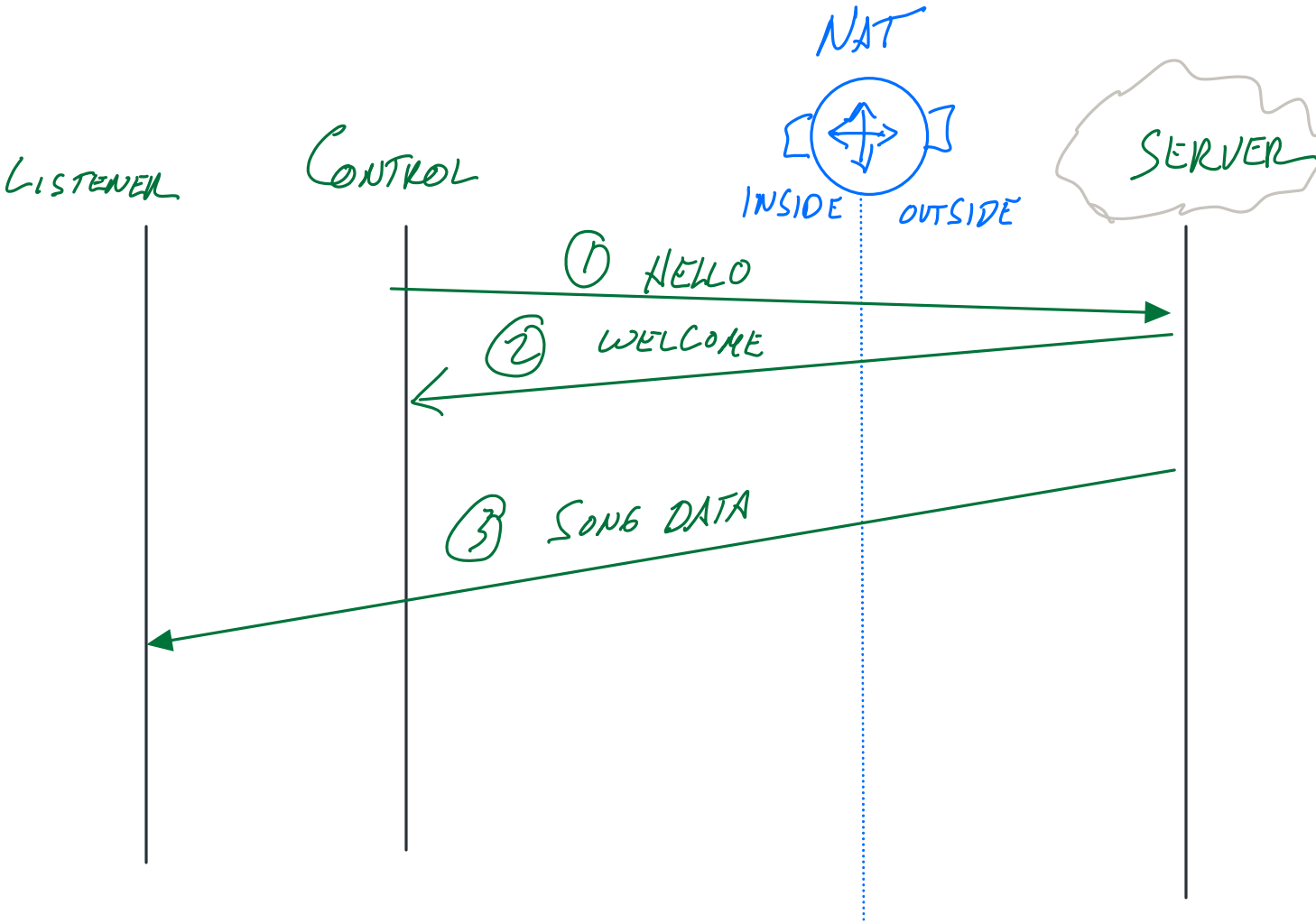
Problem?

⇒ Technically a violation of layering! Network layer shouldn't care about port numbers, but here it matters

⇒ NAT needs to know semantics of TCP/UDP (how connections start/end…

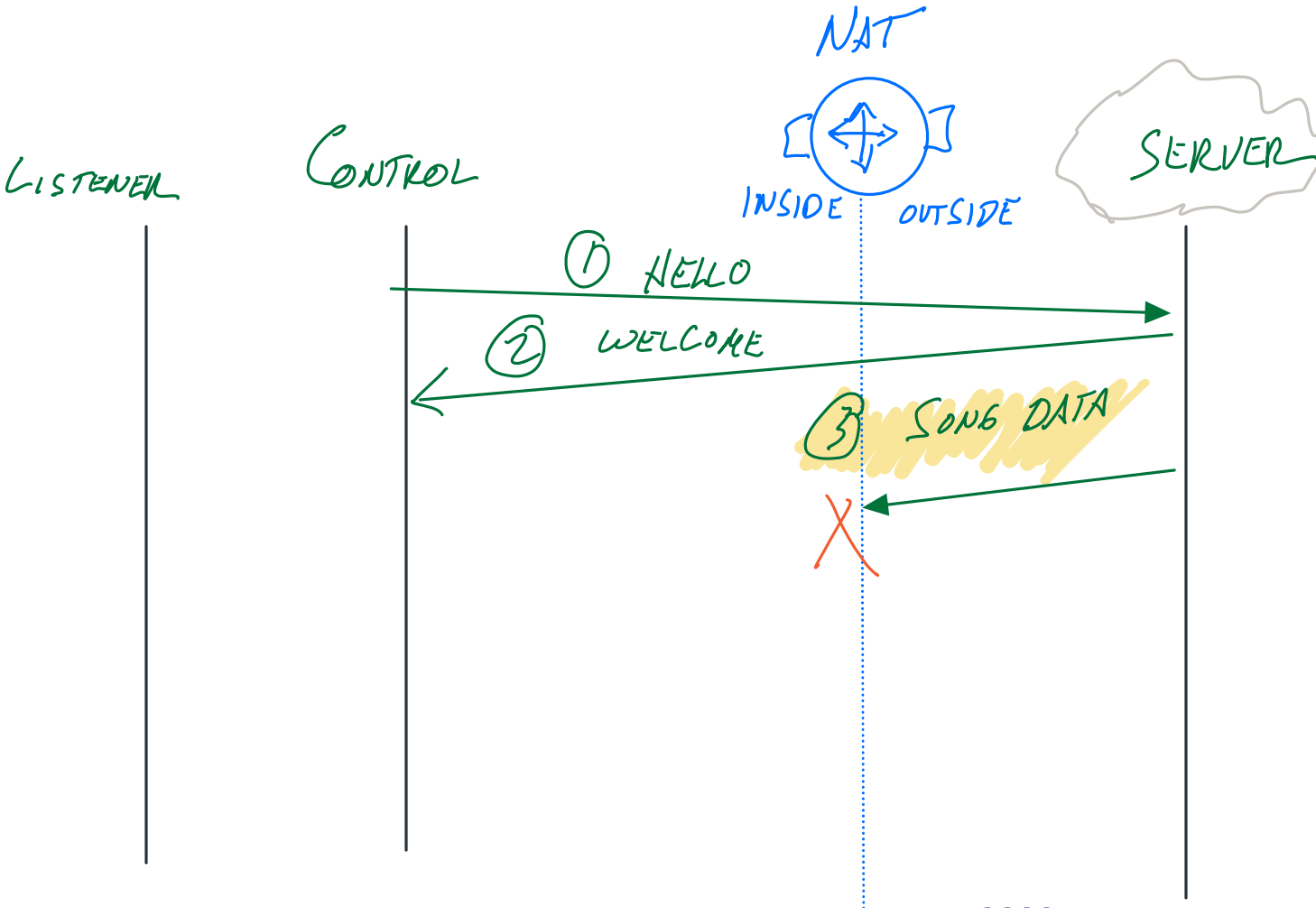
…but wait there's more…

NAT R. SNOWCAST



Which of these steps will not work if the client is behind a NAT????

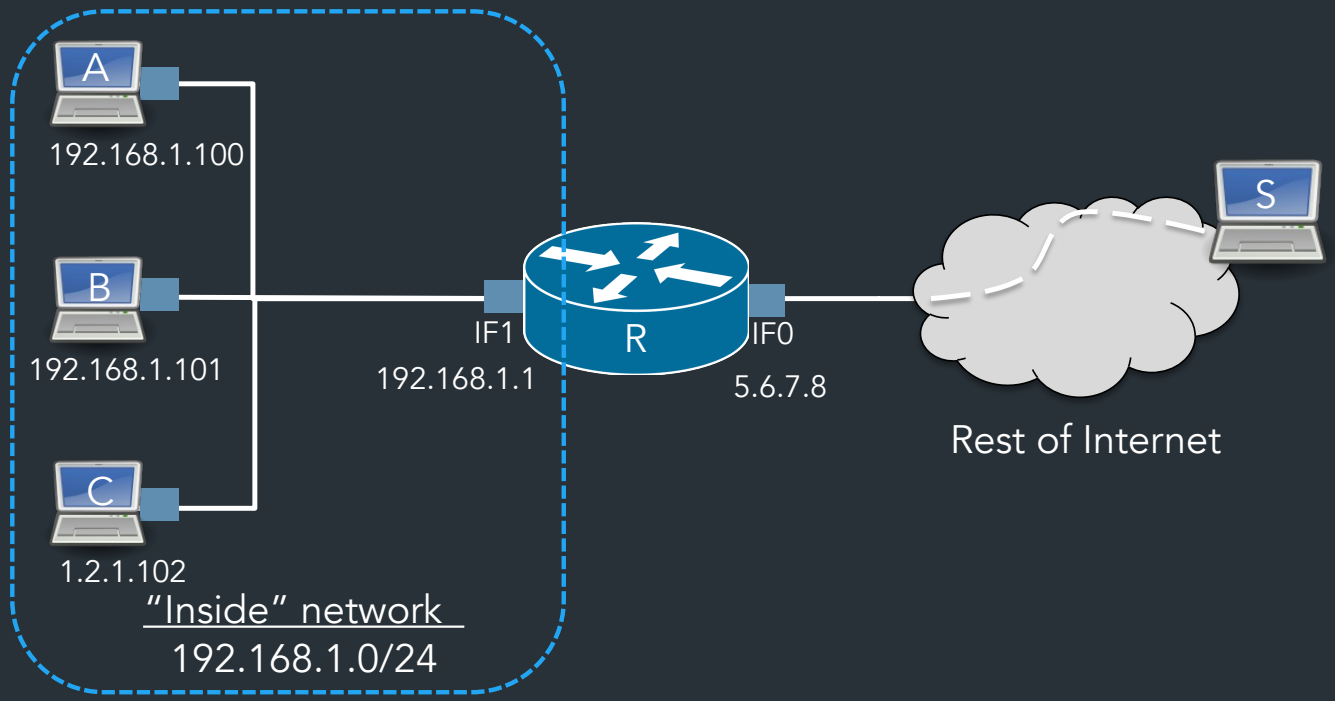
NAT R. SNOWCAST



Which of these steps will not work if the client is behind a NAT????

=> Server won't be able to send data to the listener! When the server sends a packet to the listener client, there's no translation rule for the listener's port, so the packet will get dropped!

=> In general, an outside host can't send packets to an inside host unless the inside host has made a connection first



What happens when outside host S wants to connect to inside host A?

Can't do it (at least without special setup)!

⇒ By default, R only knows how to translate packets for connections originating from INSIDE the network

⇒ Breaks end to end connectivity!!!

Why is this bad?

NAT is used in just about every consumer network

- Generally: can't connect directly to an ~~end~~ host unless it connects to you first

- Need extra work for any protocols that need a direct connection between hosts

⇒ Protocols that aren't strictly client-server
⇒ Latency critical applications: voice/video calls, games

NAT Traversal

Various methods, depending on the type of NAT

Examples:

- Manual method: port forwarding
- ICE: Interactive Connectivity Establishment (RFC8445)
- STUN: Session Traversal Utilities for NAT (RFC5389)

One idea: connect to external server via UDP, it tells you the address/port

Routing

Challenges in moving packets

- Forwarding:

  given a packet, decide which interface to send the packet (based on destination IP)

  => Occurs on every packet

- Routing:

  network-wide process of determining the packet's *path* => how routers figure out what to put in their forwarding tables

  => figuring out how to keep table updated as network changes

  => Slower process, not per packet (many seconds, minutes)

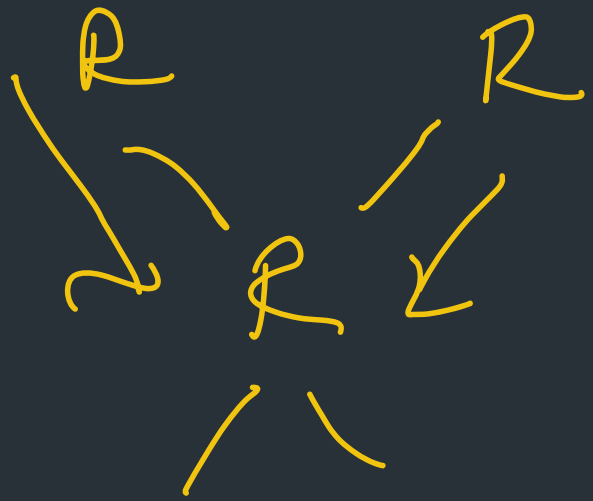
Routing is the process of updating forwarding tables

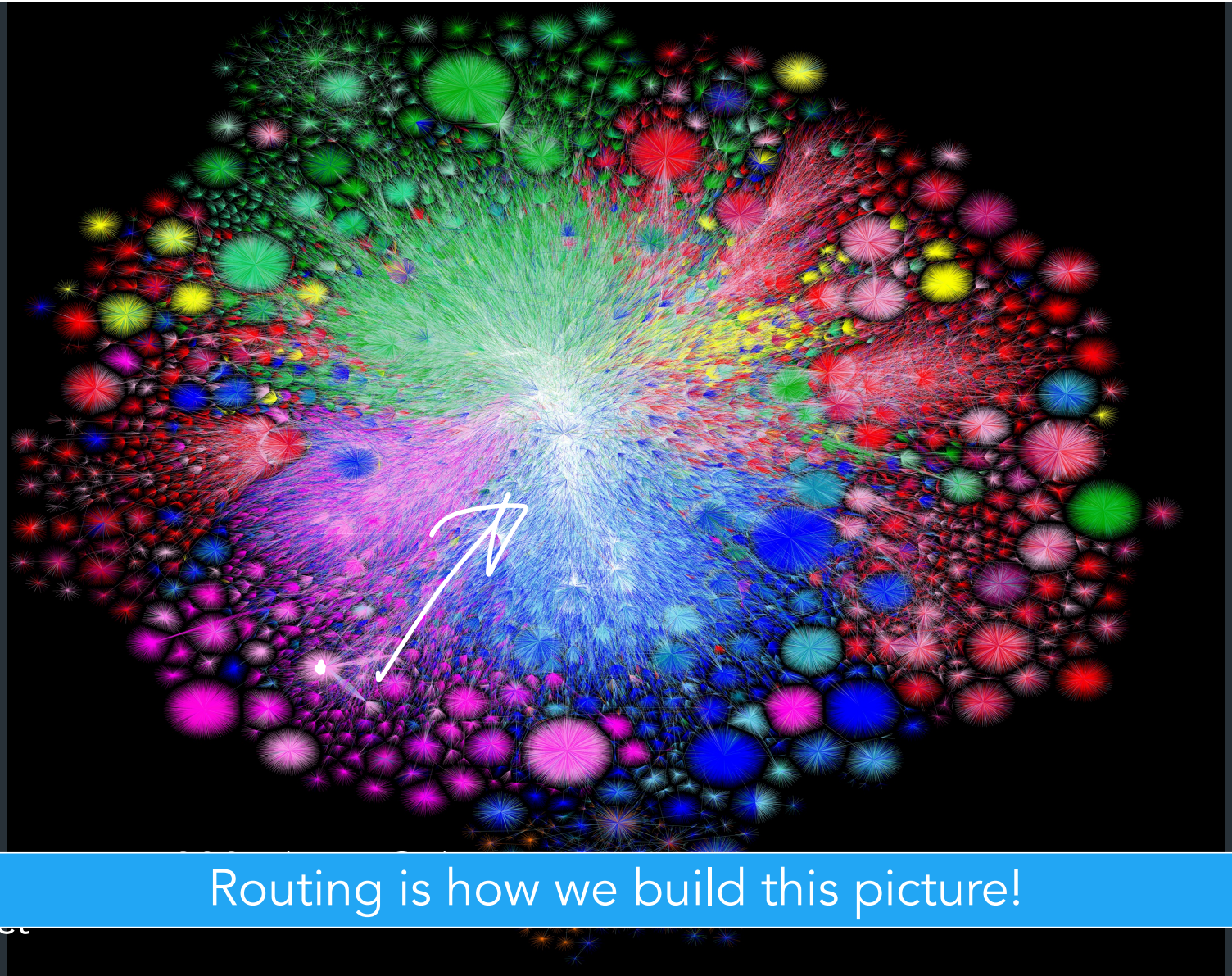
- Routers exchange messages about networks they can reach

Goal: find optimal route (or *any* route…) for every other destination

This is a hard problem

- Decentralized
- Topology always changing
- Scale!





Map of th
OPTE proj

Routing is how we build this picture!

How do we connect *everything*?

Relies on hierarchical nature of IP addressing

- Smaller routers don't need to know everything, just another router that knows more

  ⇒ Has default route

- Core routers know everything => no default!

A forwarding table (my laptop)

0.0.0.0/0

```
deemer@ceres ~ % ip route
default via 10.3.128.1 dev wlp2s0
10.3.128.0/18 dev wlp2s0 proto dhcp scope link src 10.3.135.44 metric 3003
172.18.0.0/16 dev docker0 proto kernel scope link src 172.18.0.1
192.168.1.0/24 dev enp0s31f6 proto kernel scope link src 192.168.1.1
```

A large table

```
rviews@route-server.ip.att.net>show route table inet.0 active-path

inet.0: 866991 destinations, 13870153 routes (866991 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both


0.0.0.0/0          *[Static/5] 5w0d 19:43:09
                    > to 12.0.1.1 via em0.0
1.0.0.0/24         *[BGP/170] 1d 10:24:47, localpref 100, from 12.122.83.238
                      AS path: 7018 3356 13335 I, validation-state: valid
                    > to 12.0.1.1 via em0.0
1.0.4.0/22         *[BGP/170] 1d 10:24:47, localpref 100, from 12.122.83.238
                      AS path: 7018 3356 4826 38803 I, validation-state: valid
                    > to 12.0.1.1 via em0.0
1.0.4.0/24         *[BGP/170] 1d 10:24:47, localpref 100, from 12.122.83.238
                      AS path: 7018 3356 4826 38803 I, validation-state: valid
                    > to 12.0.1.1 via em0.0
1.0.5.0/24         *[BGP/170] 1d 10:24:47, localpref 100, from 12.122.83.238
                      AS path: 7018 3356 4826 38803 I, validation-state: valid
                    > to 12.0.1.1 via em0.0
1.0.6.0/24         *[BGP/170] 1d 10:24:47, localpref 100, from 12.122.83.238
                      AS path: 7018 3356 4826 38803 I, validation-state: valid
                    > to 12.0.1.1 via em0.0
```

Thinking about the scale

At this stage, we think about routing to *whole networks*, ie, some entity with some set of IP prefixes:

eg. Brown University @ 128.148.0.0/16, 138.16.0.0/16

We call each entity an Autonomous System (AS):
a single administrative domain that lives on the Internet

Routing is organized in two levels:

- Intra-domain (interior) routing: routing within an AS
  - ~ 100 PREFIXES / ROUTERS
  - ADMINISTRATOR CONTROLS ALL ROUTERS
  - KNOW ABOUT ALL ROUTERS ⇒ CAN TRY TO FIND SHORTEST PATH

  (RIP, OSPF)

  WE ARE HERE

- Inter-domain (exterior) routing: routing between ASes
  - NO SINGLE ADMIN
  - DON'T HAVE ALL INFO.
  - DECISIONS MADE BY POLICY

  (BGP)

  ⇒ INTERNET-SCALE.

Routing is organized in two levels:

(RIP, OSPF)

- Intra-domain (interior) routing: routing within an AS
  - => Full knowledge of the network inside the AS
  - => One administrator, routing policy
  - => Strive for optimal paths

^ We are here today

(BGP)

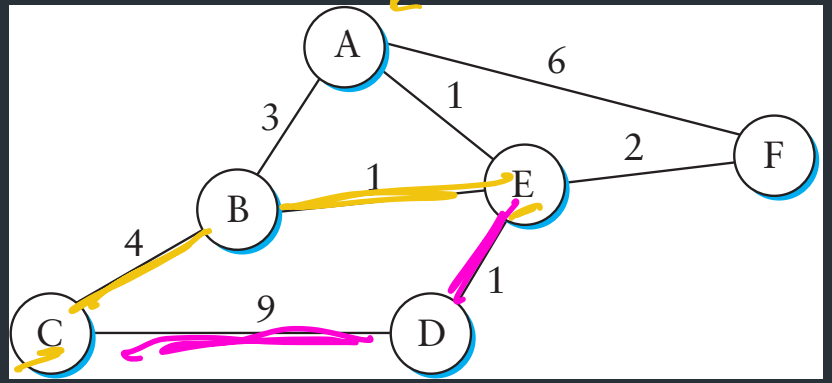
- Inter-domain (exterior) routing: routing between ASes
  - => None of the above, decisions instead made by *policy* (later)

=> INTERNET-SCALE

Intra-Domain (Interior) Routing

Typically, view network as a graph

- Nodes are routers
- Assign some *cost* to each edge
  - latency, b/w, queue length, …



ONE ROUTER

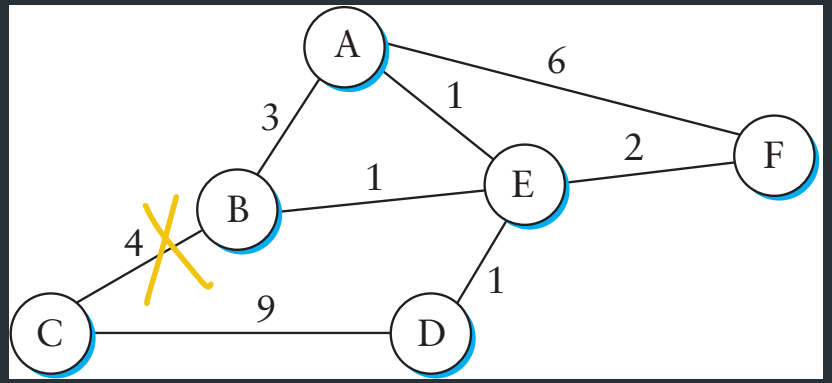
"COST" OR "METRIC" SET BY ADMIN

Goal: find lowest-cost path between nodes

- Each node individually computes routes

Typically, view network as a graph

- Nodes are routers
- Assign some *cost* to each edge
  - latency, b/w, queue length, …



DECENTRALIZED.

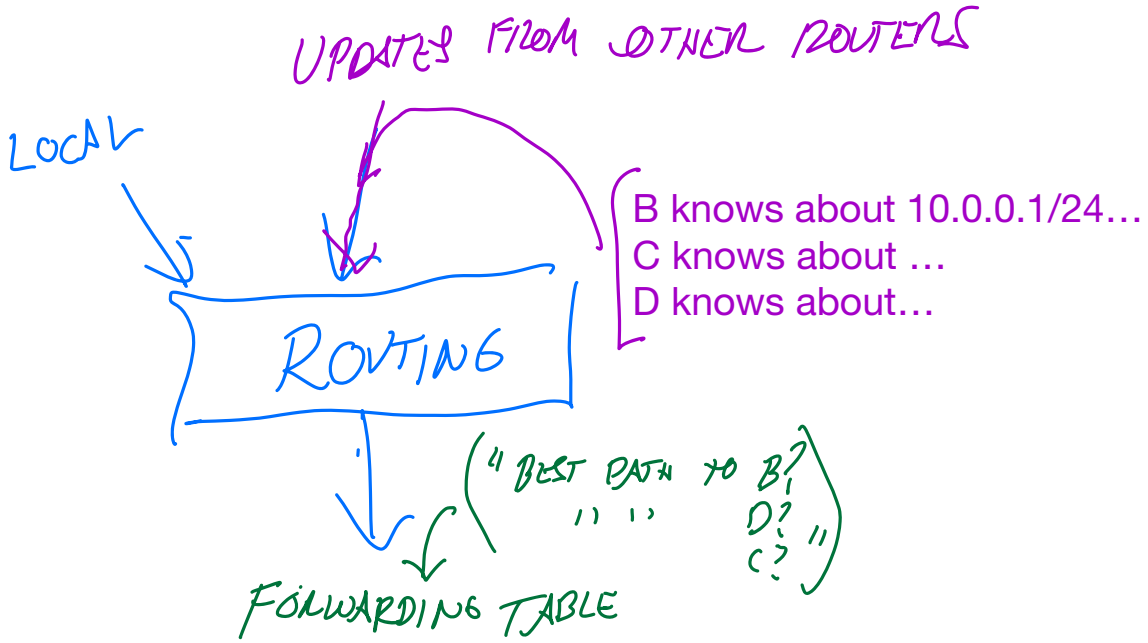
Goal: find lowest-cost path between nodes

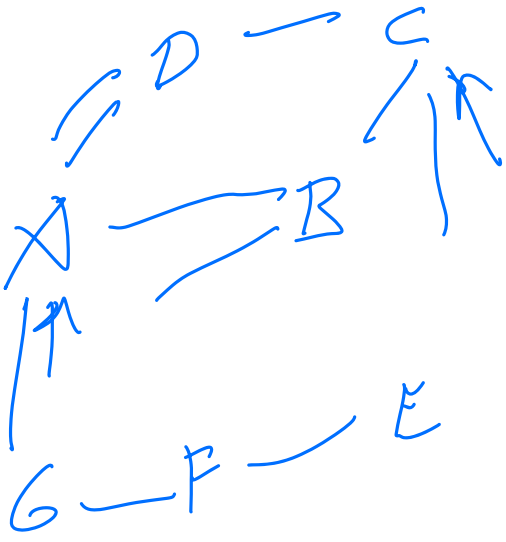
- Each node individually computes routes

Collect routes into a *routing table*, used to generate the forwarding table based on lowest-cost path

Routing process is decentralized:

- No one entity telling routers what routes to have
- When we do interior routing, there is algorithm that routers are using, independently, to figure out what to do





Generally: routing algorithms are *decentralized*

=>In general, no one entity telling routers what routes to use

=> Even for “interior” routing, where there is one admin, routers independently compute how to update their tables based on latest info from other routers

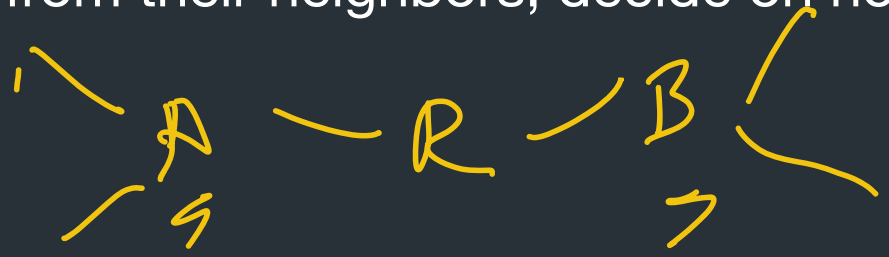
Two classes of intra-domain routing algorithms

TODAY

Distance Vector (Bellman-Ford shortest path algorithm)

RIP

Routers learn info from their neighbors, decide on how to update tables



Link State (Djikstra/Prim shortest path algorithm)

OSPF

Distance Vector Routing

- Each node maintains a *routing table*

ROUTES CURRENTLY KNOWN

- Exchange *updates* with neighbors about node's links:
  => List of <Destination, Cost> pairs

"I KNOW ABOUT A AT COST 2"
(A, 2)

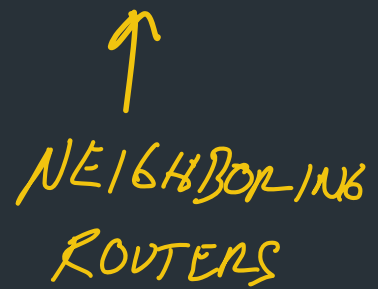
Distance Vector Routing

- Each node maintains a routing table
- Exchange *updates* with neighbors about node's links:
  => List of <Destination, Cost> pairs
- When to send updates?
  - Periodically (seconds to minutes)
  - Whenever table changes (*triggered* update)
  - Time out an entry if no updates within some time interval

Distance Vector Routing

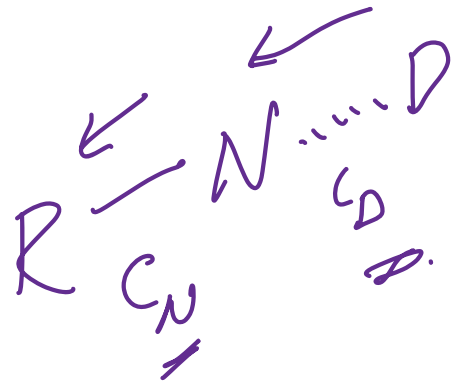
- Each node maintains a routing table
- Exchange *updates* with neighbors about node's links:
  => List of <Destination, Cost> pairs
- When to send updates?
  - Periodically (seconds to minutes)
  - Whenever table changes (*triggered* update)
  - Time out an entry if no updates within some time interval

Dest.	Cost	Next Hop
A	3	S
B	4	T
C	5	S
D	6	U



DV routing: update rules

Suppose: router receives an update about some destination D from neighbor router N



(D, C_D)

$\Rightarrow$ R CAN REACH D WITH COST $C = C_D + C_N$

(D, C_D)

Table has format <Destination, Cost, Next hop>

If D isn't already in the table, add it <D, C, N>

If you have an existing entry <D, c_old, M>

- c < c_old => Update table with new route <D, c, N> (BETTER ROUTE!)

- if c > c_old {
  if (N == M) // Topology has changed, route has higher cost now
  Update your table: <D, c, N> (HIGHER COST!)

  else // N != M
  // Can ignore (current route is better!)
  }
- c == c_old and N == M
  // Repeat of same route => No change (but refresh timeout)

Separately: need to keep track of last update time for each route, delete old entries that expire

Distance Vector: Update rules

Say router R receives an update <D, c_D> from neighbor N at cost C_N

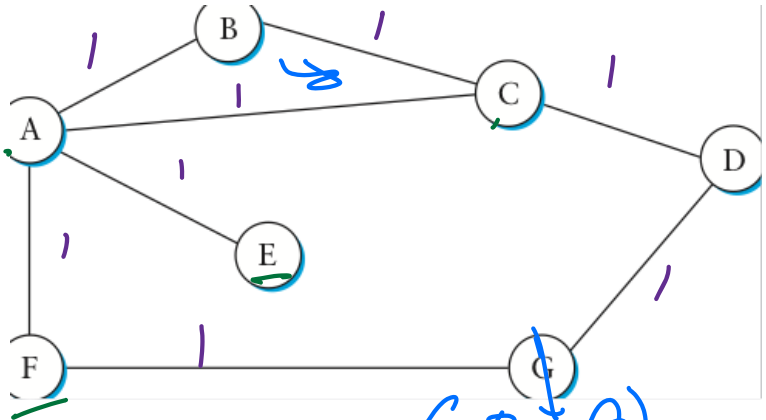
=> Know: R can reach D via N with cost c = c_D + c_N

How to update table?

1. If D not in table, add <D, c, N> (New route!)
2. If table has entry <D, M, c_{old}>:
   - if c < c_{old}: update table to <D, c, M>. (Lower cost!)
   - if c > c_{old} and M == N: update table to <D, c, N> (Cost increased!)
   - if c > c_{old} and M != N: ignore (N is better)
   - if c == c_{old} and M ==N: no change (No new info)
     (Just refresh timeout)

DEST	COST	NEXT
B	0	B *
A	1	A
C	1	C
E	2	A
F	2	A

*: Router always knows about its local prefixes (don't usually write this)



TO: B SENDS UPDATE (B, 0) TO A, C

A SENDS UPDATE (A, 0)

C " " (C, 0)

T_1: — A SENDS UPDATE (A, 0)

(E, 1) NEW!

(F, 1) NEW!

(C, 1)] NOT BETTER!

(B, 1)]

Warmup

Suppose router R has the following table:

Dest.	Cost	Next Hop
A	3	S
B	4	T
C	5 6	S
D	6 5	U S

NO CHANGE

TIE

COST INCREASED!

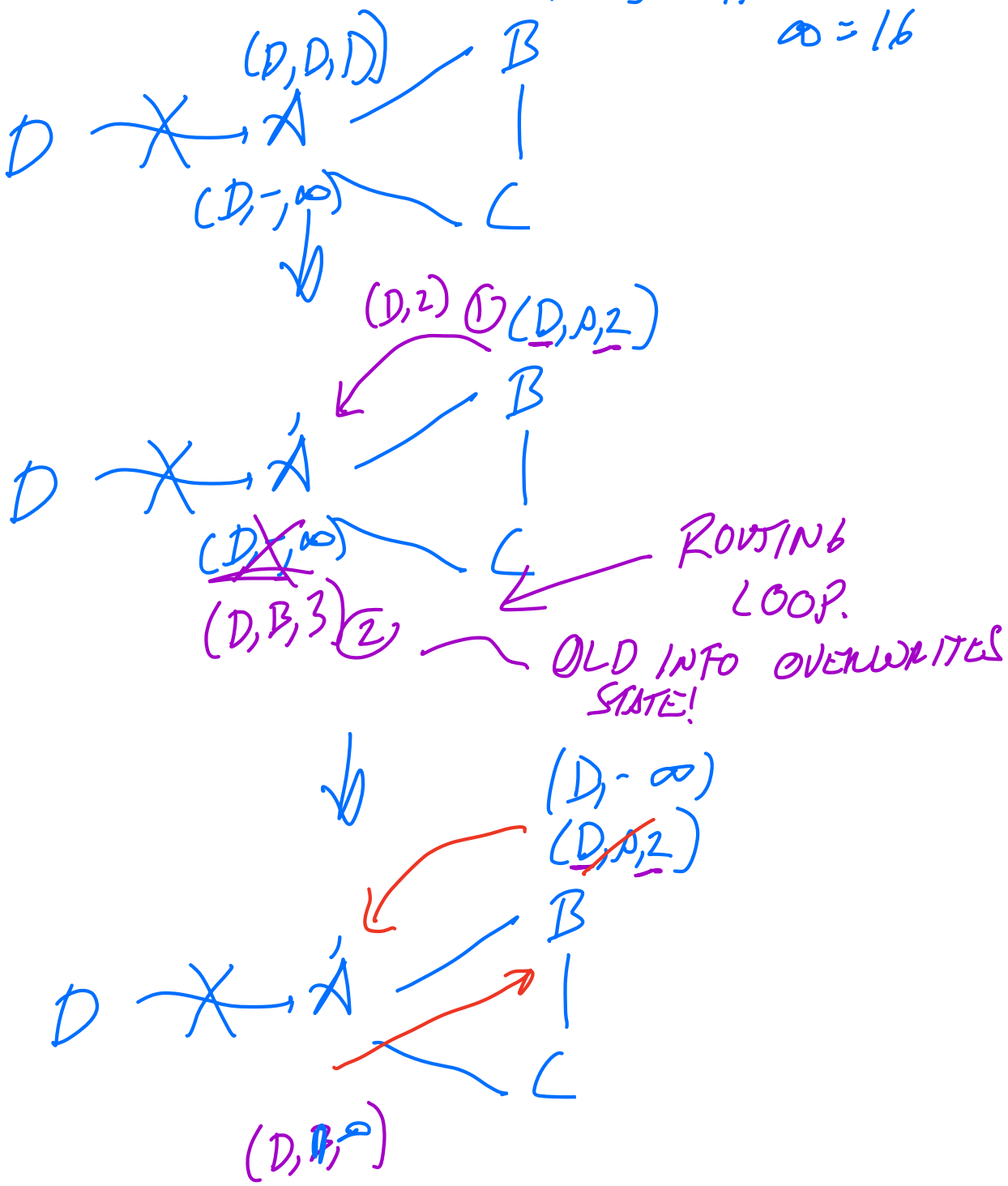
What happens when it gets this update from router S?

Dest.	Cost
A	2
B	3
C	5
D	4
E	2

← NEW!

AFTER THIS PAGE ARE MORE NOTES FROM THE NEXT LECTURE FROM LAST YEAR — FEEL FREE TO READ AHEAD!

RIP: WHAT HAPPENS WHEN D-A LINK FAILS?



⇐ - UPDATES OCCUR IN A LOOP W/ INCREASING COST UNTIL COST REACHES ∞

⇒ COUNT TO INFINITY - LONG CONVERGE TIME.

How to avoid loops

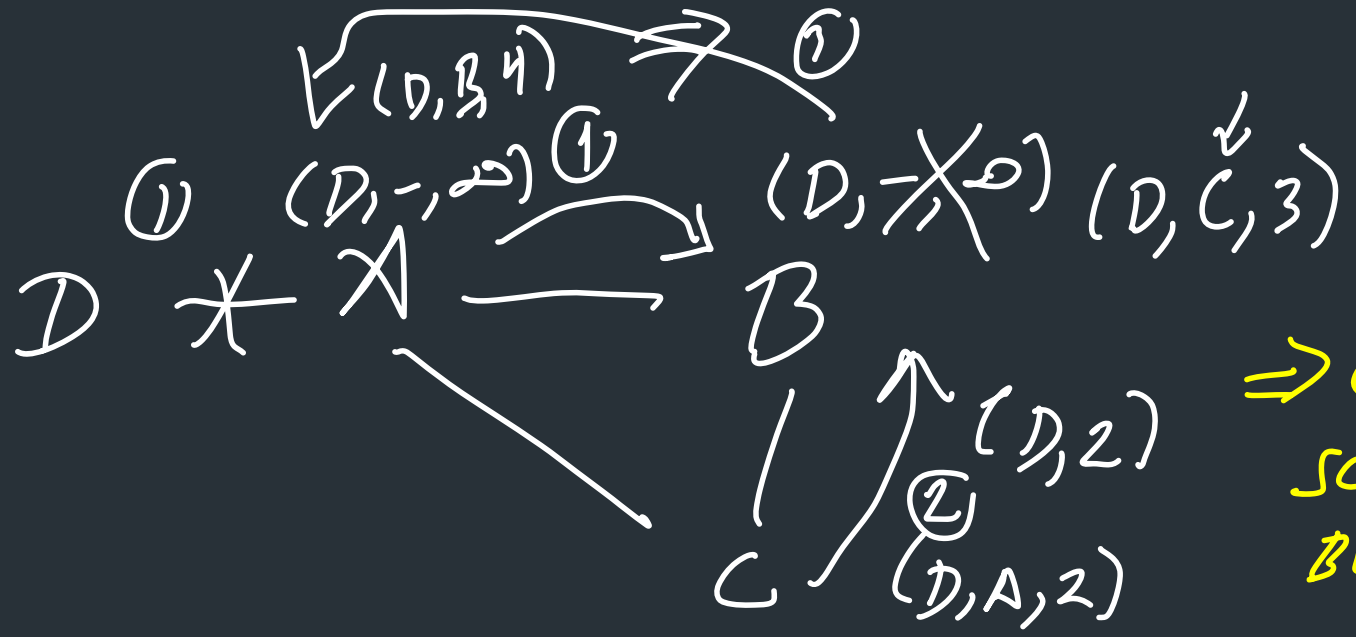
- Does IP TTL help? => doesn't solve routing problem.
- Simple approach: consider a small cost n (e.g., 16) to be infinity
  - After n rounds decide node is unavailable
  - But rounds can be long, this takes time

Problem: distance vector based only on local information

↳ not enough to resolve loops, count-to infinity (but there are tricks we can do)

One way: Split Horizon

- When sending updates to node A, don't include routes you learned from A
- Prevents B and C from sending cost 2 to A

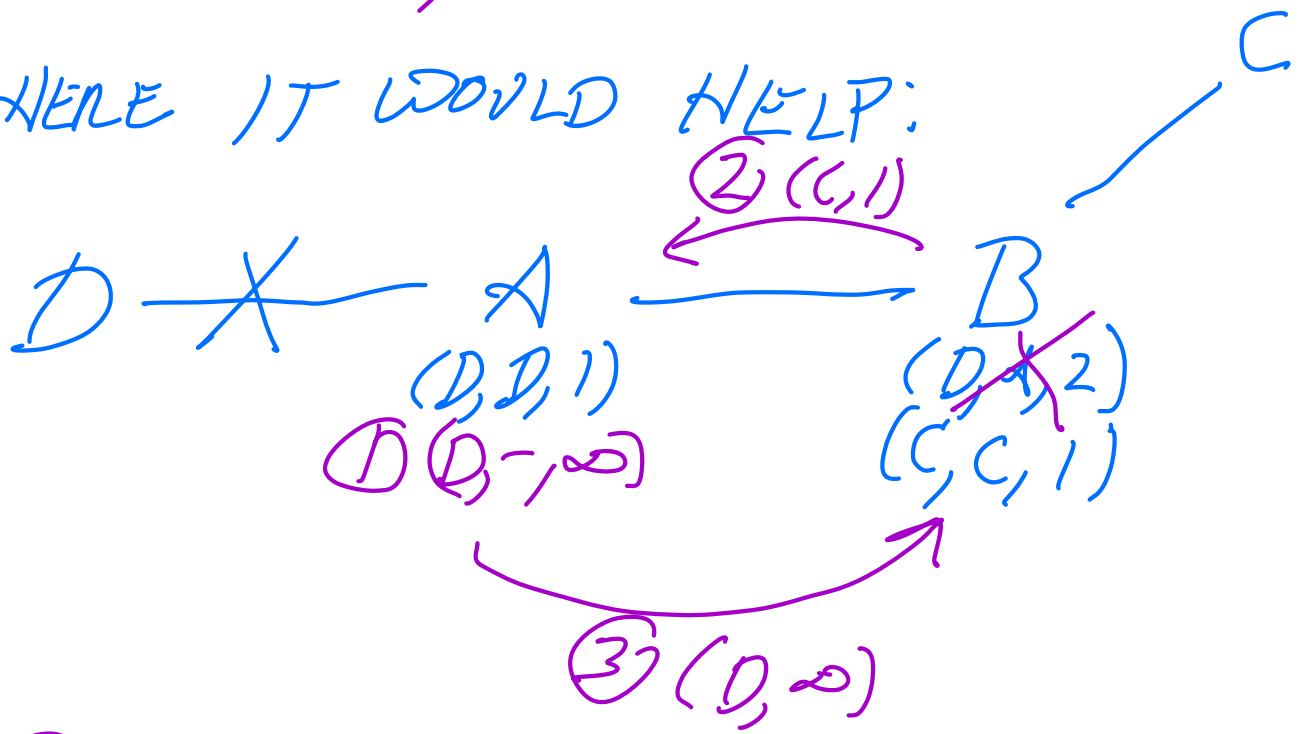


SPLIT HORIZON

- IF A USES N AS NEXT HOP FOR D, DO NOT REPORT TO N ABOUT D

⇒ PREVENTS "LINEAR" ROUTING LOOPS, BUT NOT OTHERS

WHERE IT WOULD HELP:



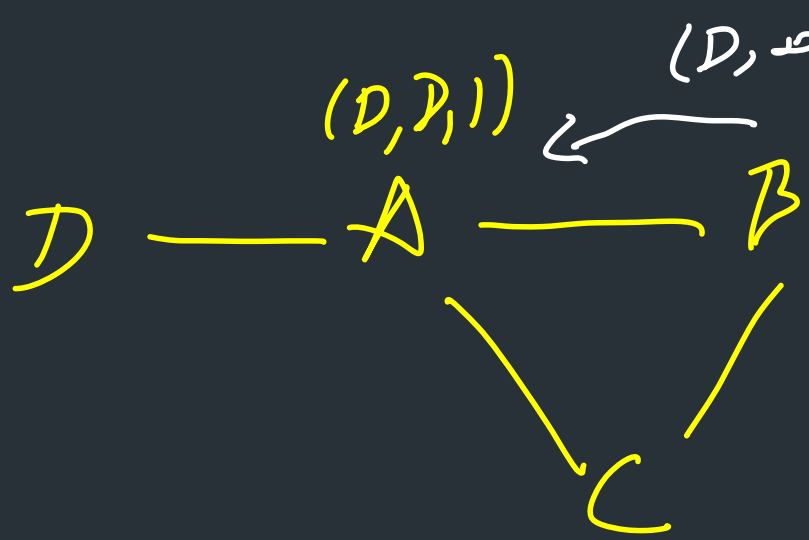
① D-A LINK GOES DOWN

② WHEN B SENDS UPDATE TO A, IT WOULD NOT TELL INCLUDE A

③ A UPDATES B W/ (D, ∞)

Split Horizon + Poison Reverse

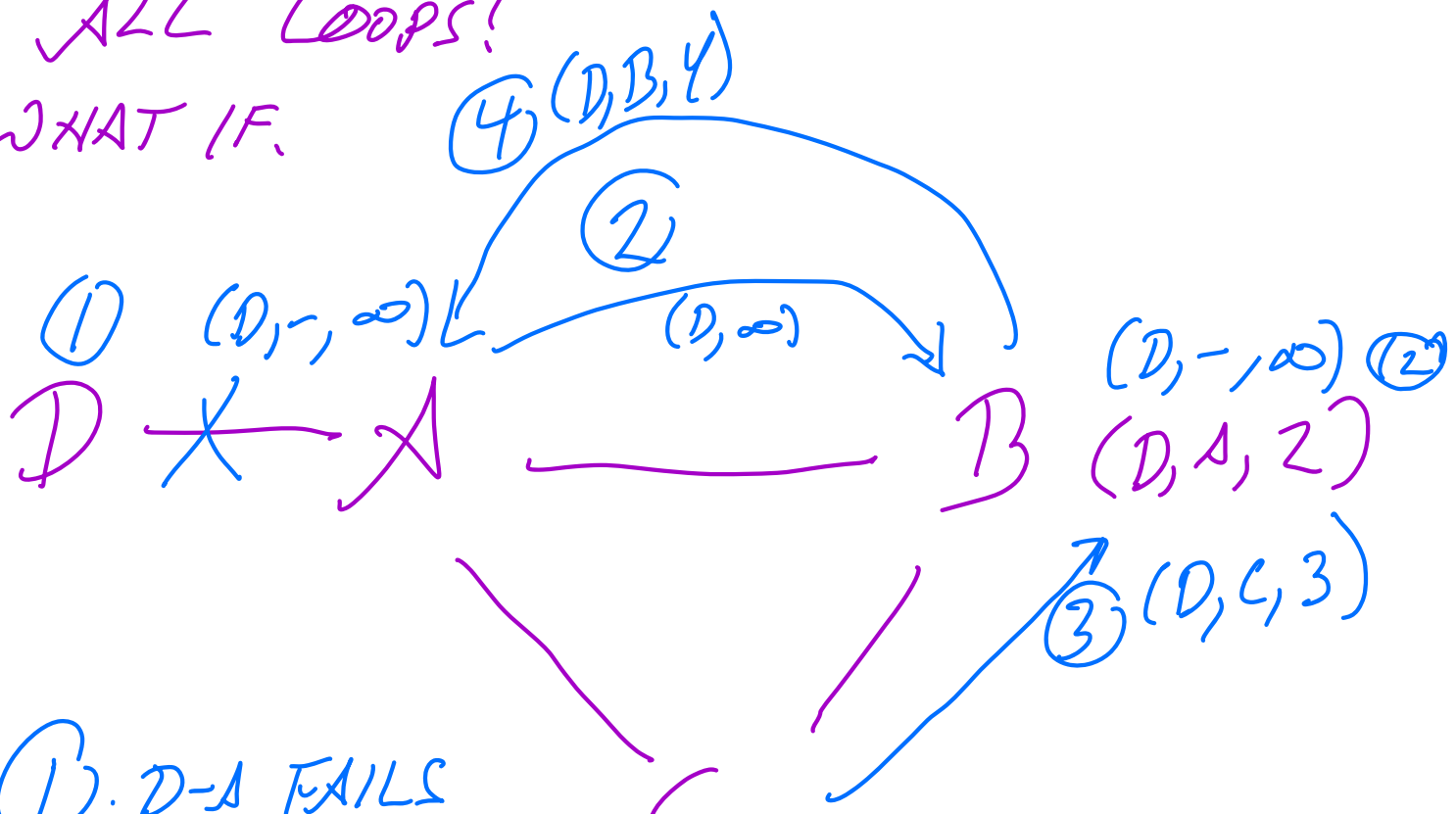
- Rather than not advertising routes learned from A, explicitly include cost of ∞.
- Faster to break out of loops, but increases advertisement sizes



POISON REVERSE IS A COMMON CONVENTION, MIGHT HELP TIME TO CONVERGE IN SOME CASES, BUT HARD TO SEE EFFECT VS. SPLIT HORIZON.

BUT EVEN W/ SPLIT HORIZON + POISON REVERSE, CAN'T PREVENT ALL LOOPS!

WHAT IF.

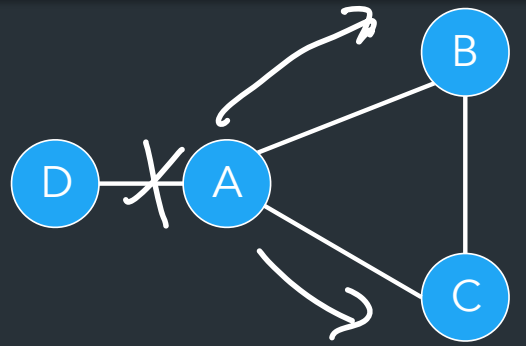


① D-A FAILS
② A UPDATES B (D, ∞) (D, A, 2)
③ C SENDS (D, 2) TO B!
↳ RACE CONDITION! C MIGHT SEND OLD UPDATE TO B BEFORE C GETS UPDATE FROM A!
④ B UPDATES A, OVERWRITES A'S ENTRY
⑤ ... COUNT TO INFINITY...

WHAT CAN WE DO?

Distance-vector updates

Even with split horizon + poison reverse,
can still create loops with >2 nodes

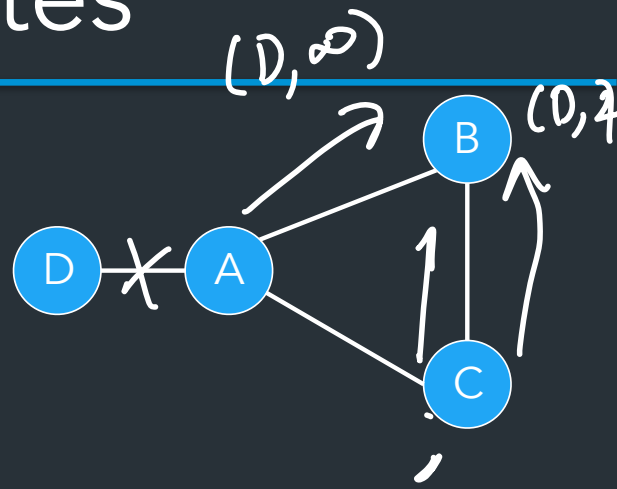


What else can we do? TRIGGERED UPDATES.

- IF A TELLS B AND C IMMEDIATELY THAT ITS ROUTE CHANGES, IT CAN PREVENT THIS LOOP.

Distance-vector updates

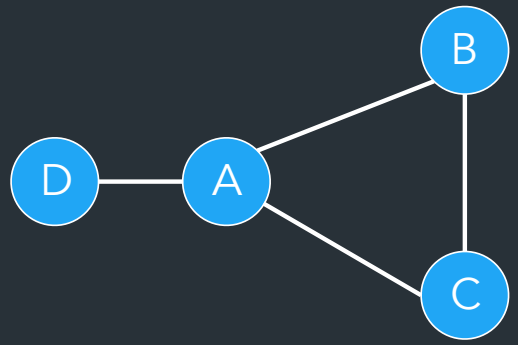
Even with split horizon + poison reverse,
can still create loops with >2 nodes



What else can we do?

- Triggered updates: send update as soon as link state changes
- Hold down: delay using new routes for certain time, affects convergence time

Practice

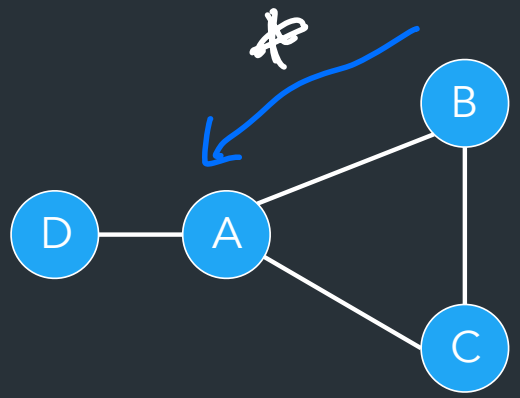


B's routing table

Routers A,B,C,D use RIP. When B sends a periodic update to A, what does it send…

- When using standard RIP?
- When using split horizon + poison reverse?

Practice



B's routing table

Dest.	Cost	Next Hop
A	1 ∞	A
C	1	C
D	2 ∞	A

Routers A,B,C,D use RIP. When B sends a periodic update to A, what does it send…

- When using standard RIP?
- When using split horizon + poison reverse?

STANDARD:
(A, 1)
(C, 1)
(D, 2)

SH + PR
(A, ∞)
(C, 1)
(D, ∞)