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 6 (TOMORROW)
  - Sign up link via email
  - Can't find a time? Make a private post on Ed!
- IP Gearup II tonight (10/5) 5-7pm, CIT368
  - Implementation/debugging stuff; bring questions!
- HW1 due tonight; HW2 out after this class or next class

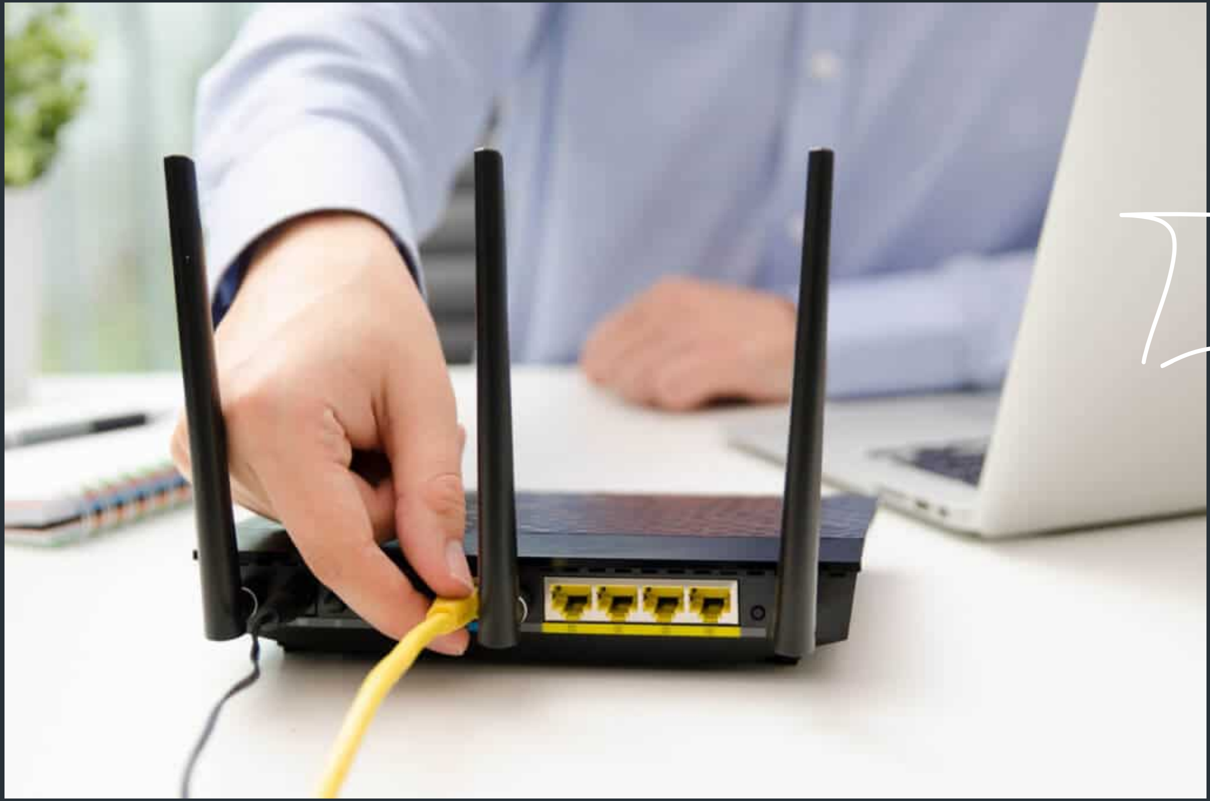
Today

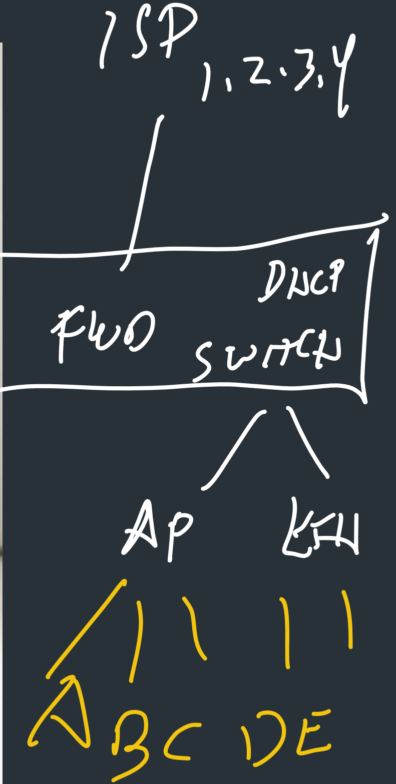
Two things

- More on NAT
- Intro to routing, RIP

Network Address Translation (NAT)

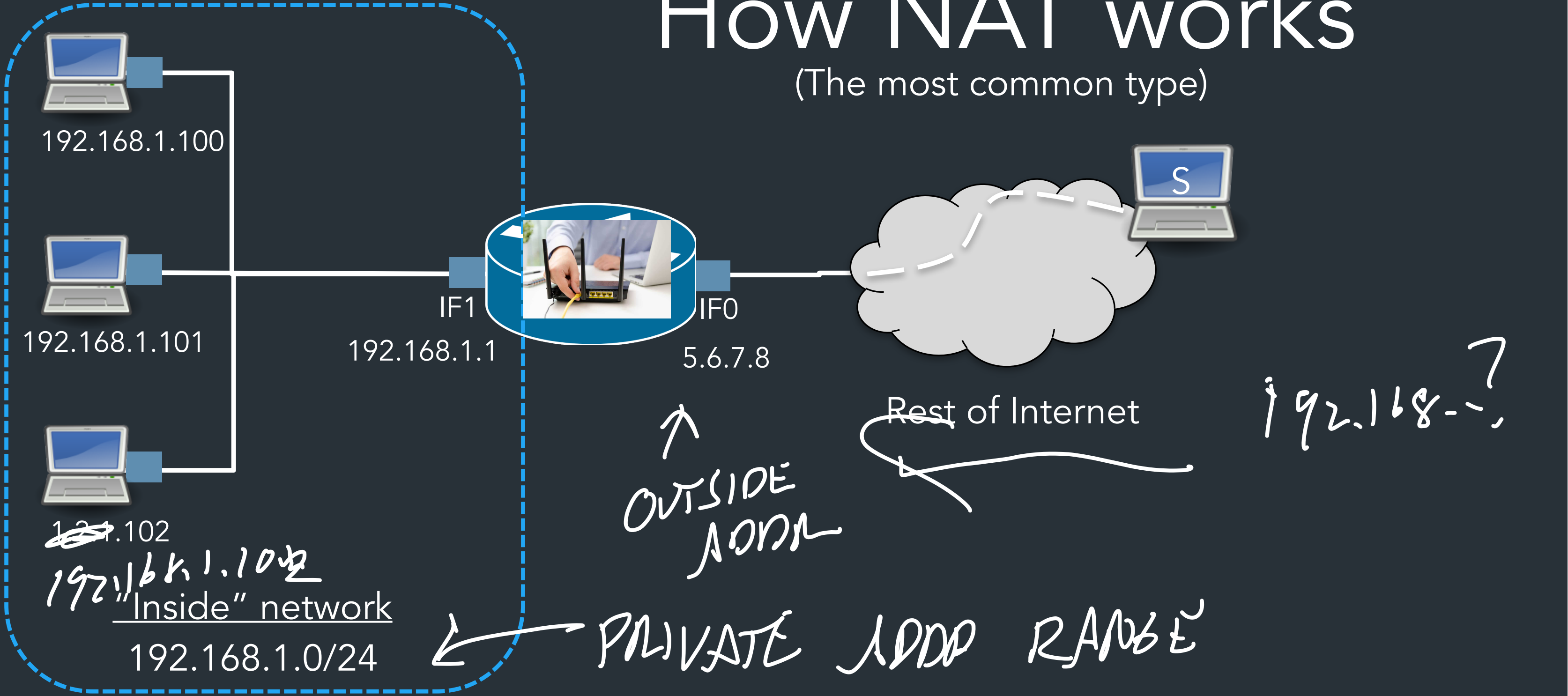
Story time





How NAT works

(The most common type)

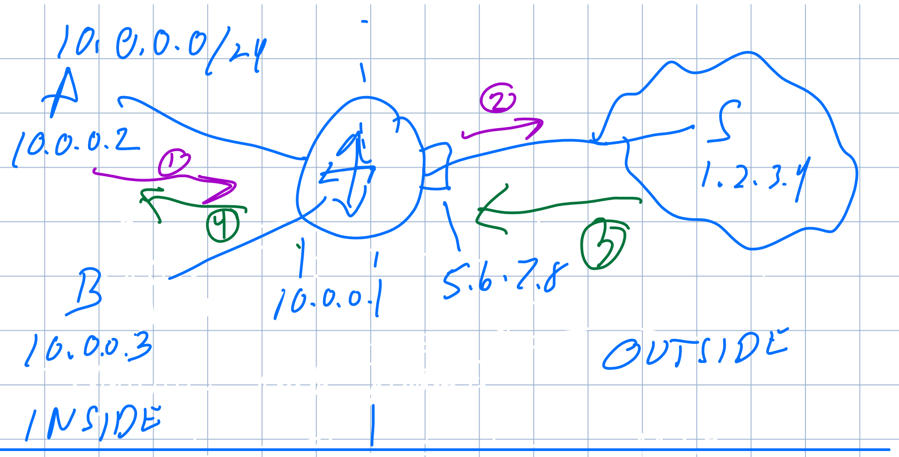


Goal: Share one IP among many hosts on a private network

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

=> Router needs to **remember connection state**

=> Router makes some (sketchy) assumptions about traffic



A WANTS TO CONNECT TO S ON PORT 80.

INSIDE

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

ROUTER REMEMBERS

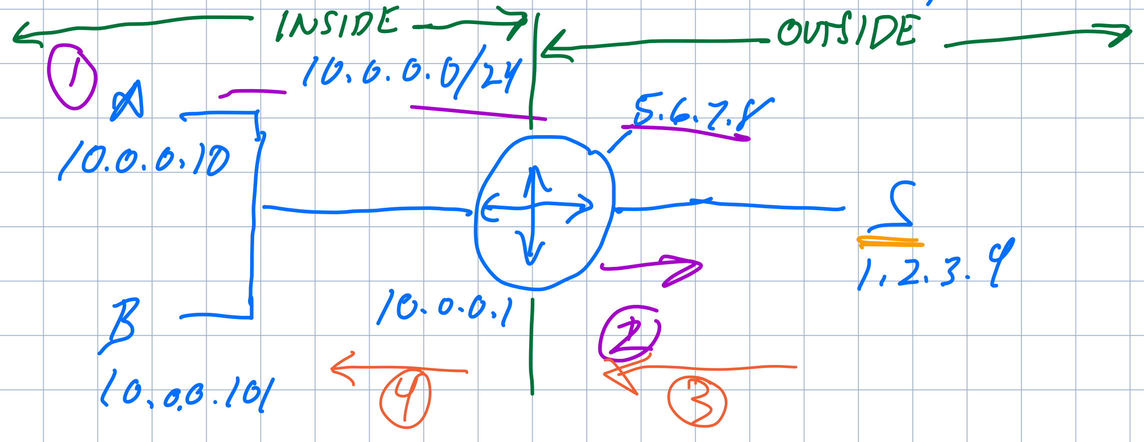
5.6.7.8:7777 → 10.0.0.2:5555

OUTSIDE

⇒ COULD YOU RUN OUT OF PORTS/ TABLE SPACE? YES.

⇒ NEED TIMEOUTS/RULES ON WHEN TO EXPIRE..

How NAT Works (In General)



INSIDE — OUTSIDE

① SRC DST — OUTSIDE SRC DST

10.0.0.1:5000 1.2.3.4:80 TCP → 5.6.7.8:8888 1.2.3.4:80

① Packet from A

② Router translates

Router stores:

10.0.0.1:5000 ⇒ 5.6.7.8:8888

↑ Inside IP — ↑ Outside — ↑ Port the router picks

③ Response from S

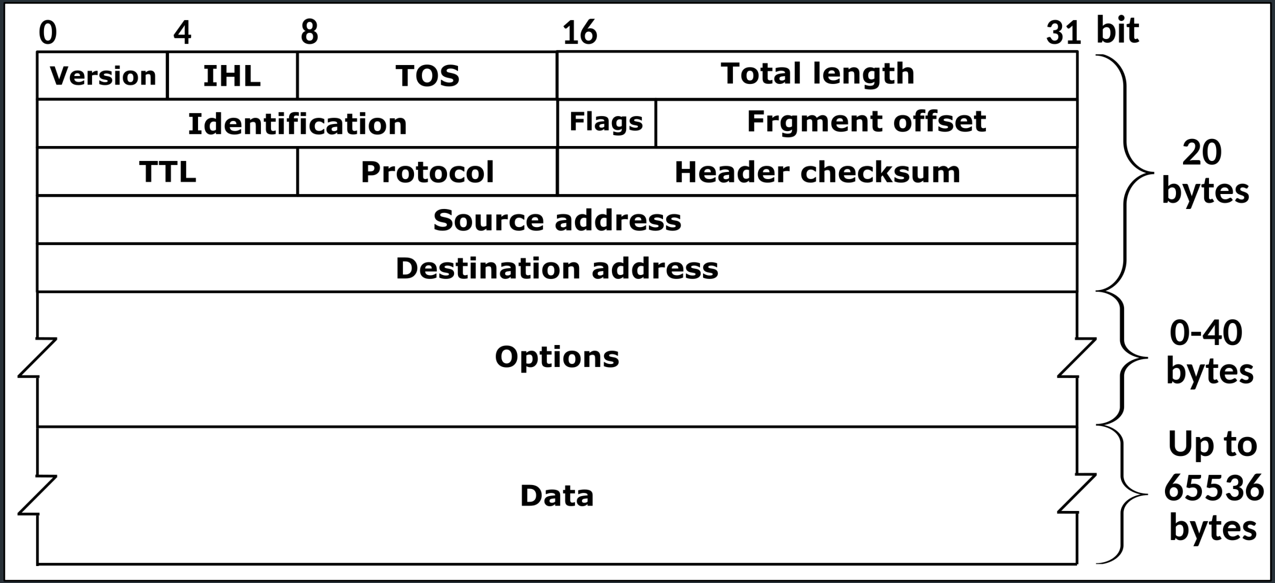
SRC DST

③ 1.2.3.4:80 5.6.7.8:8888

↓ NAT

1.2.3.4:80 10.0.0.10:5000

IP Header

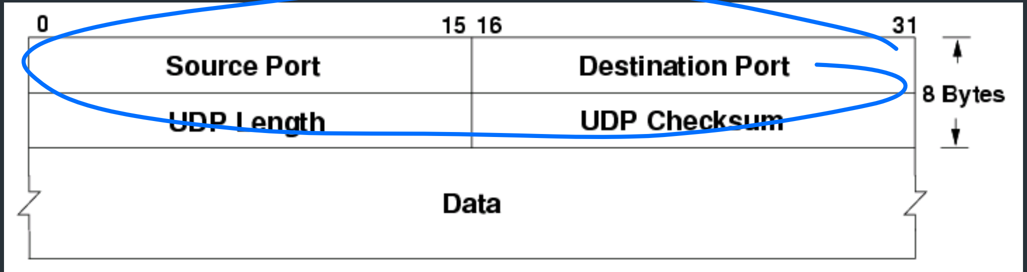


0	4	8	16	31 bit
Version	IHL	TOS	Total length	
Identification			Flags	Frgment offset
TTL		Protocol	Header checksum	
Source address				
Destination address				
Options				
Data				

Where are the port numbers?????

… ports are actually part of the transport layer header!

UDP



0 15 16 31

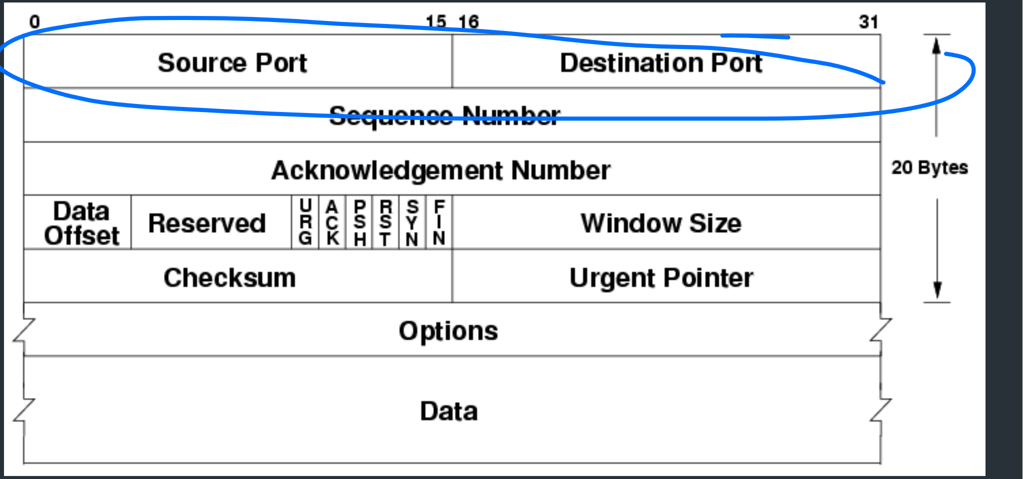
Source Port | Destination Port

UDP Length | UDP Checksum

Data

8 Bytes

TCP



0 15 16 31

Source Port | Destination Port

Sequence Number

Acknowledgement Number

Data Offset | Reserved | URG | ACK | PSH | RST | SYN | FIN | Window Size

Checksum | Urgent Pointer

Options

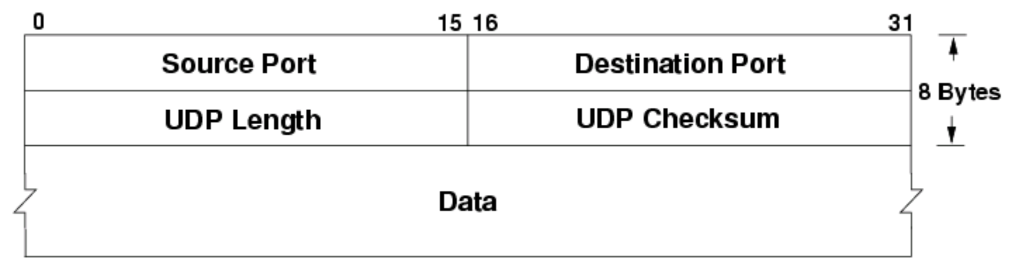
Data

20 Bytes

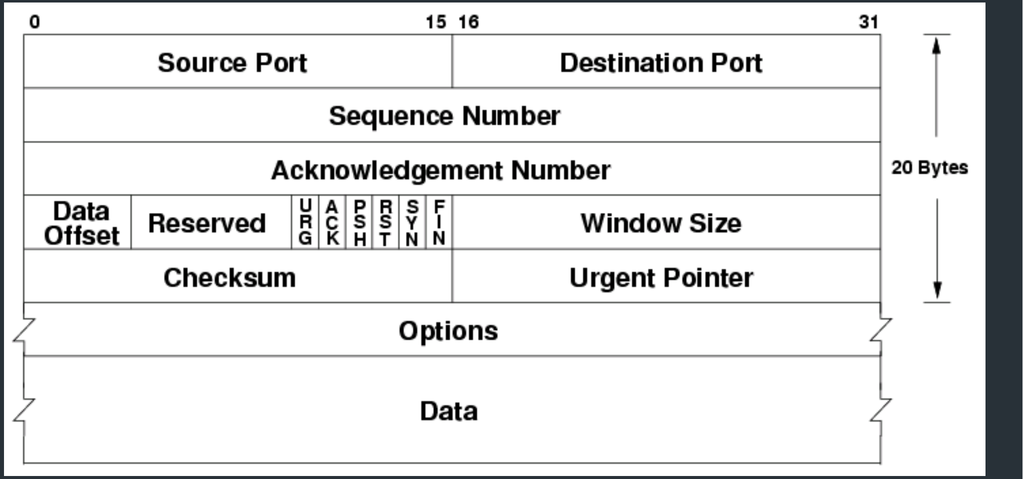
Problem?

… ports are actually part of the transport layer header!

UDP



TCP



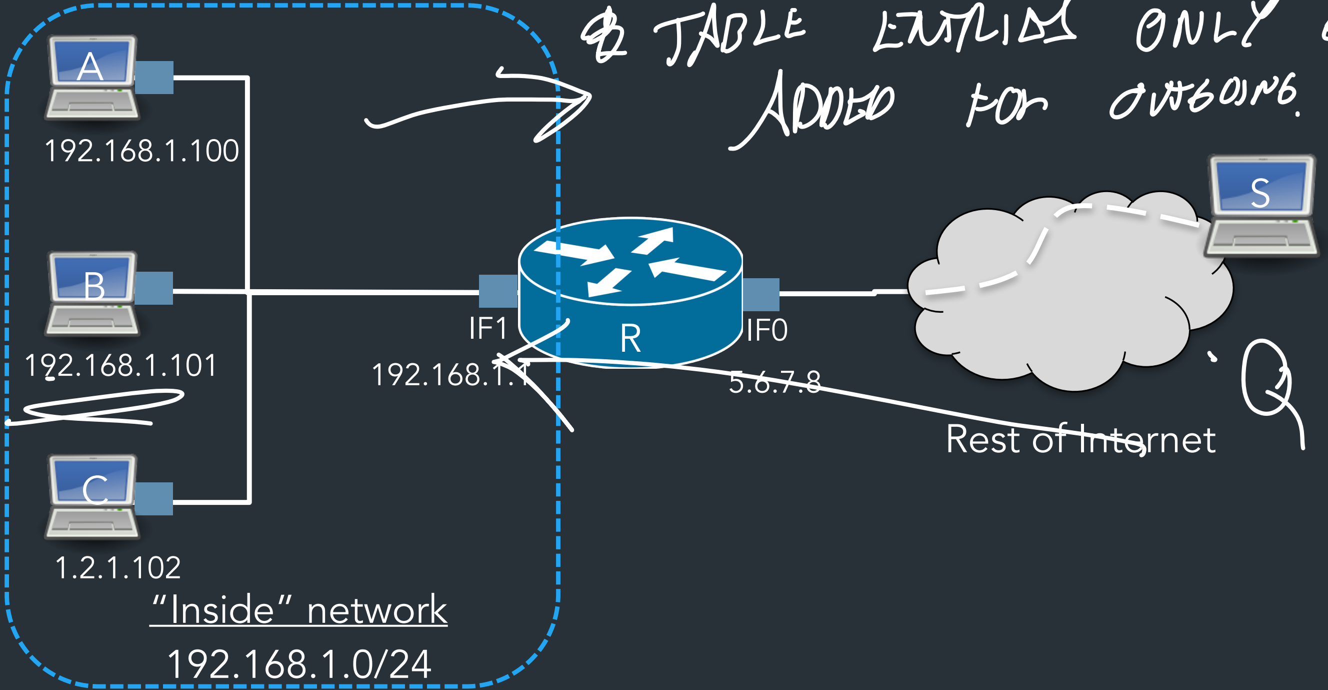
Problem?

⇒ 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…

& TABLE ENTRIES ONLY GET ADDED FOR OUTGOING CONNECTIONS!



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!!!**

End to end connectivity, you say?

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

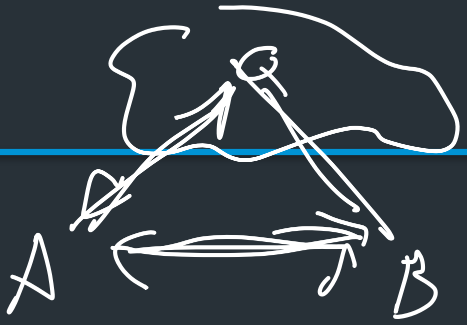
- Need extra work for any protocols that need a direct connection between hosts

⇒ **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 IP destination)

=> ON EVERY PACKET

Challenges in moving packets

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

- Routing: network-wide process of determining a packet's path through the network

  => How each router builds its forwarding table + KEEP IT UPDATED OVER TIME.

⇒ SLOWER PROCESS, NOT PER-PACKET.

Routing

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

Routing

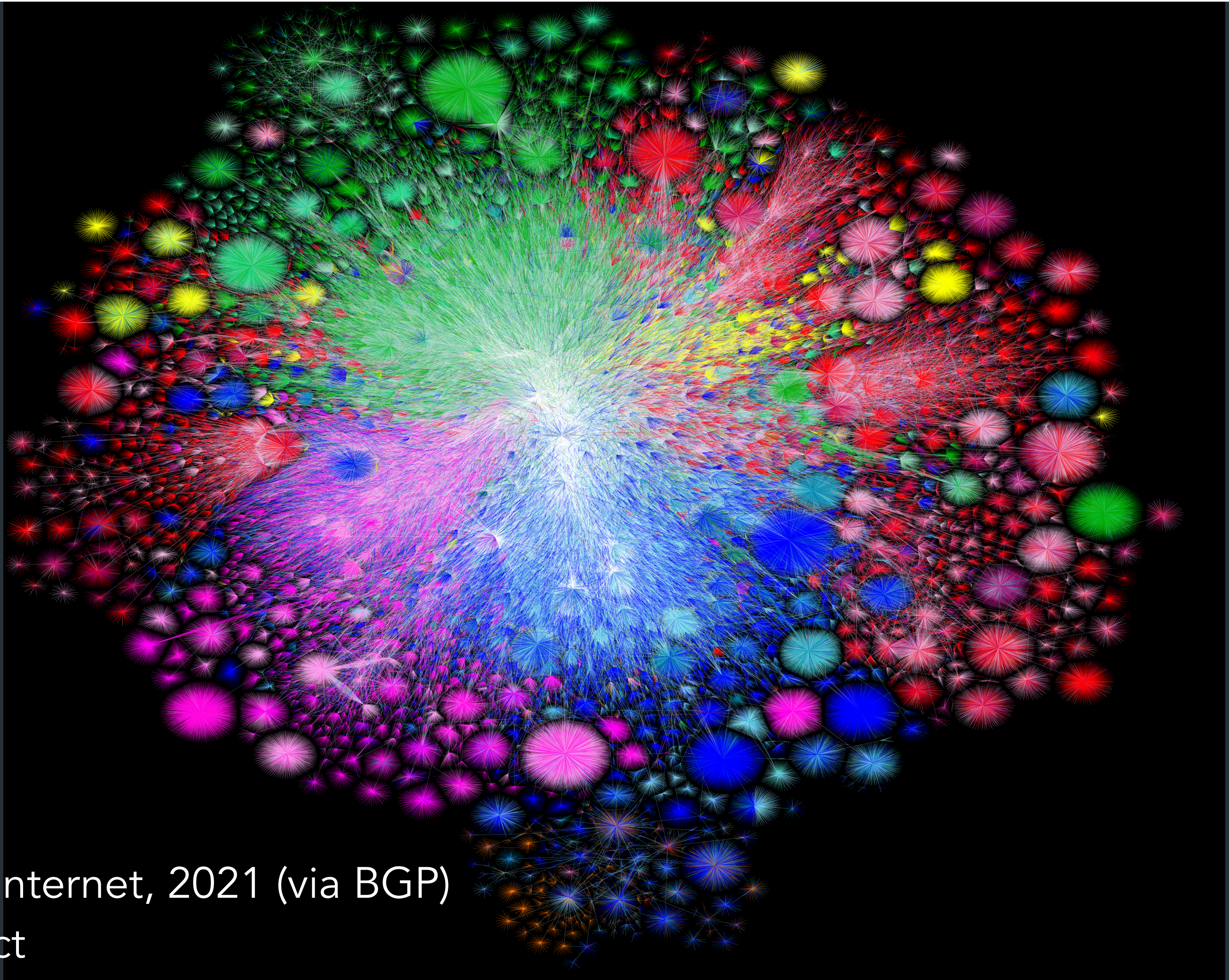
Routing is the process of updating forwarding tables

- Routers exchange messages about networks they can reach

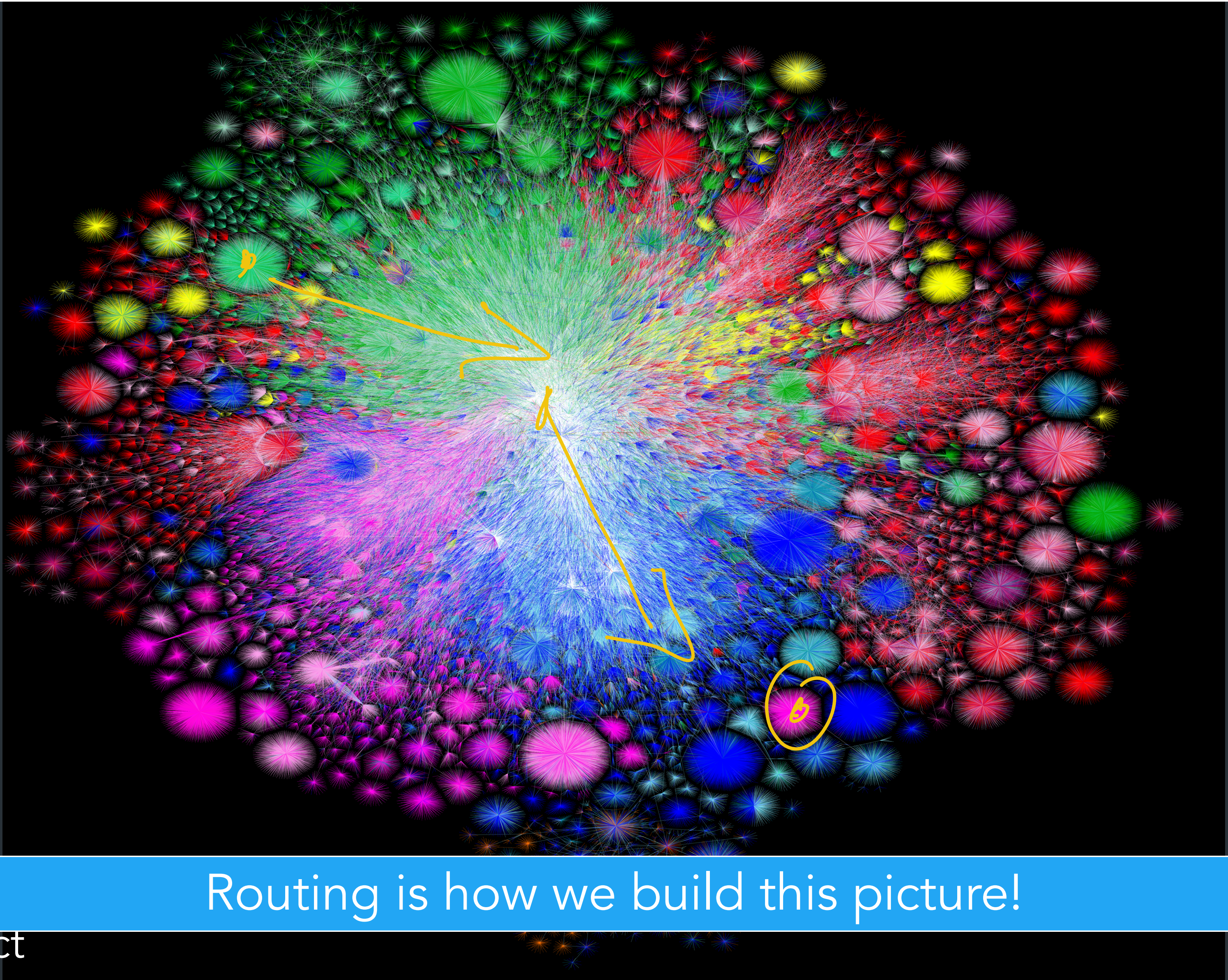
Goal: find optimal route (or any route…) for every other destination

This is a hard problem

- Decentralized
- Topology always changing
- Scale!



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



Map of th
OPTE project

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

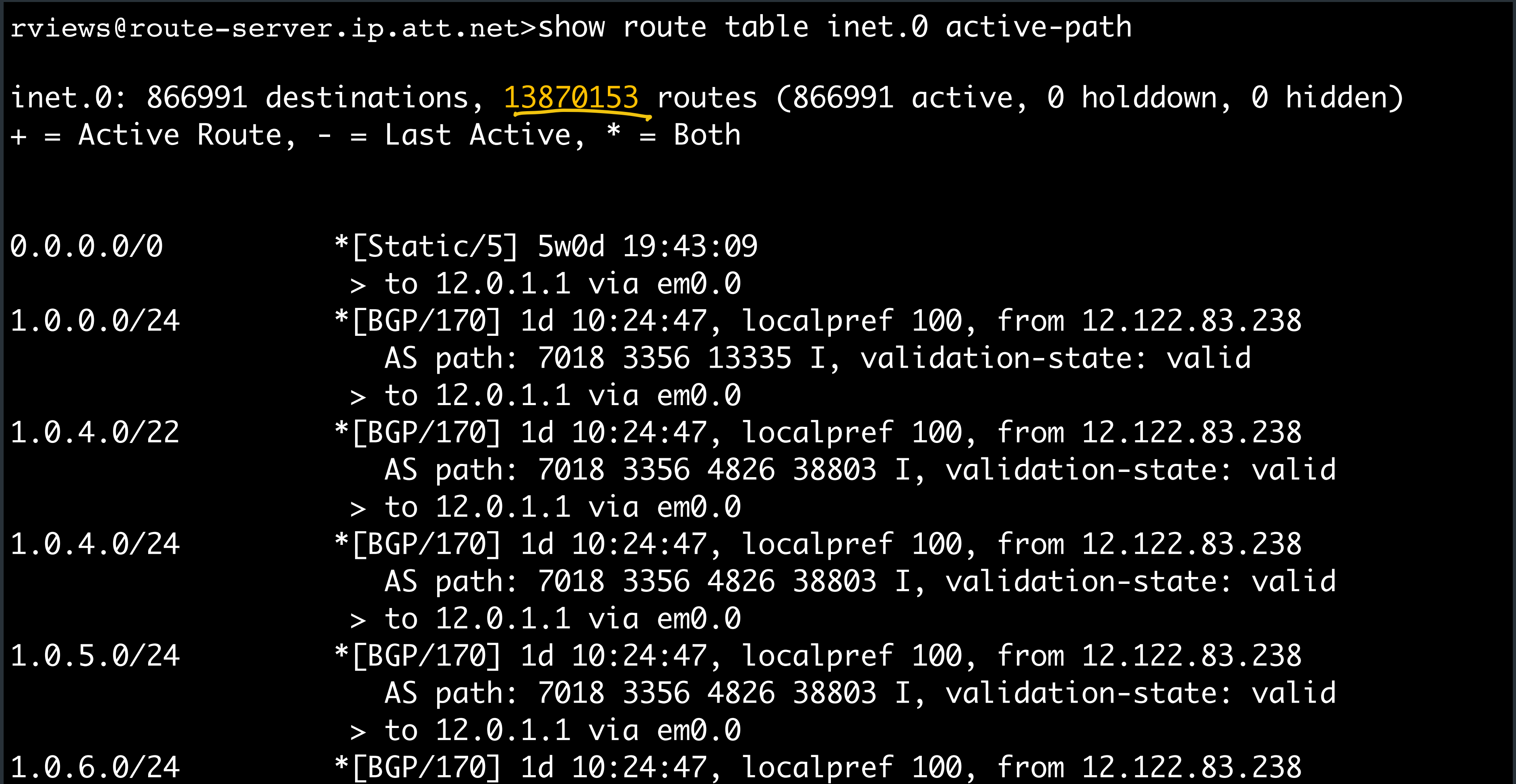
- Core routers know everything => no default!

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
```

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

WE ARE HERE,

Routing is organized in two levels:

- Intra-domain (interior) routing: routing within an AS
  - ~100 PREFIXES/ROUTERS (RIP, OSPF)
  - ADMINISTRATOR CONTROLS ALL ROUTERS
  - KNOW ABOUT ALL ROUTERS ⇒ CAN TRY TO FIND SHORTEST PATH
- Inter-domain (exterior) routing: routing between ASes
  - NO SINGLE ADMIN (BGP)
  - DON'T HAVE ALL INFO. ⇒ INTERNET-SCALE
  - DECISIONS MADE BY POLICY

Routing is organized in two levels:

- Intra-domain (interior) routing: routing within an AS

  => Full knowledge of the network inside the AS
  => One administrator, routing policy
  => Strive for optimal paths

- Inter-domain (exterior) routing: routing between ASes

  => None of the above, decisions instead made by policy (later)

Routing is organized in two levels:

- Intra-domain (interior) routing: routing within an AS

  => Full knowledge of the network inside the AS
  => One administrator, routing policy
  => Strive for optimal paths

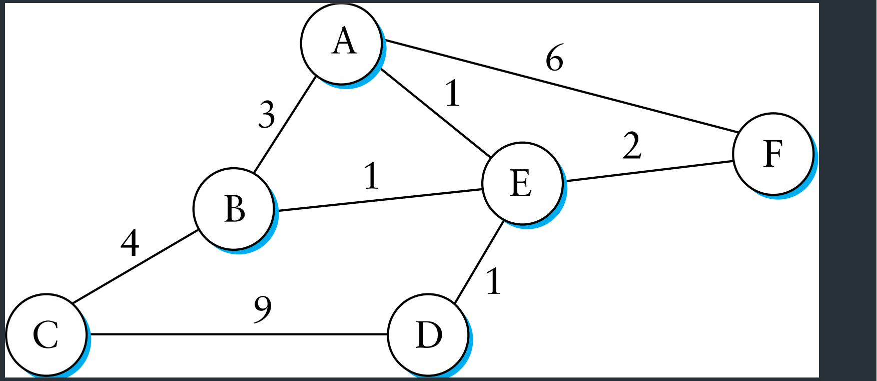
  ^ We are here today

- Inter-domain (exterior) routing: routing between ASes

  => None of the above, decisions instead made by policy (later)

Intra-Domain (Interior) Routing

ONE ROUTER.



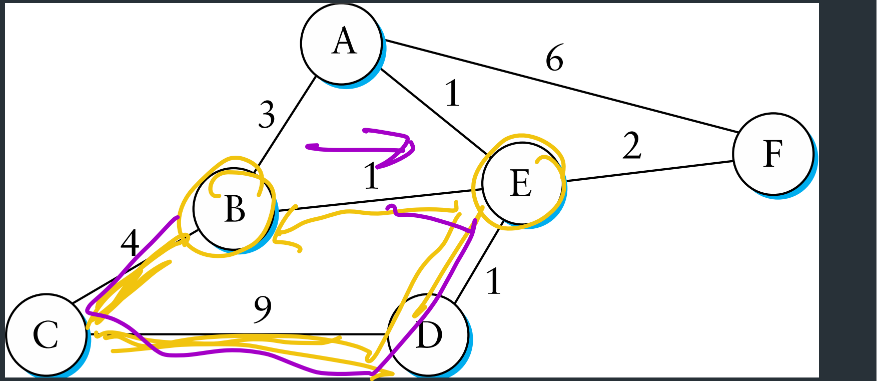
Typically, view network as a graph

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

→ ADMINISTRATOR PICKS

Typically, view network as a graph

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

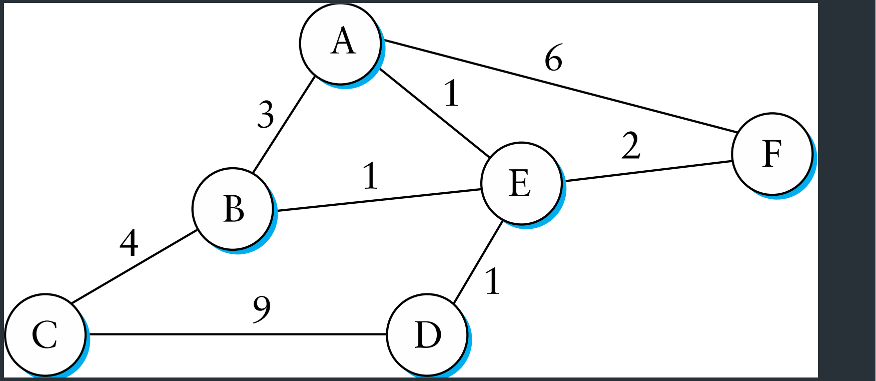


Goal: find lowest-cost path between nodes

- 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, …



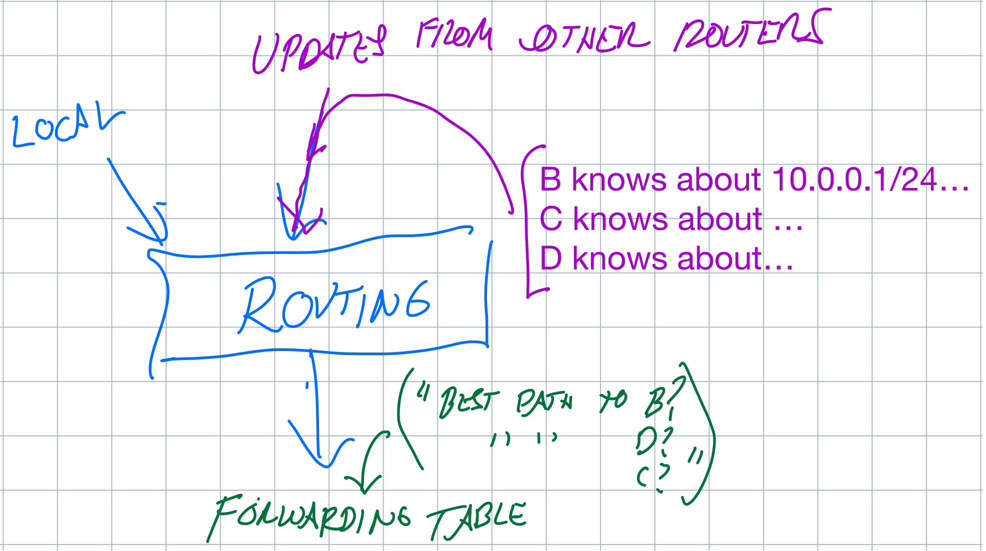
Goal: find lowest-cost path between nodes

- Each node individually computes routes

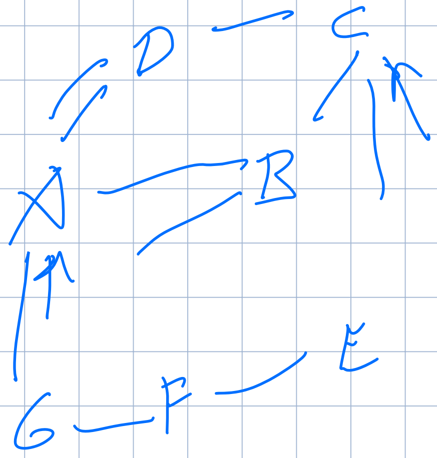
Collect routes into a routing table, used to generate the forwarding table based on lowest-cost path

Is the routing process centralized?

- 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



B knows about 10.0.0.1/24…
C knows about …
D knows about…



Distance Vector Routing

- Each node maintains a routing table

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

(A, 2)

"I KNOW ABOUT A AT COST 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

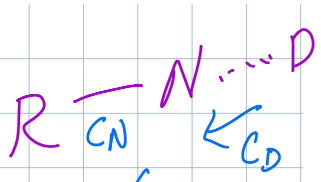
Distance Vector Routing

- 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

↑ NEIGHBORING ROUTERS,

DISTANCE VECTOR ROUTING:



(D, C_D)

$\langle D, C_D \rangle$

UPDATE RULES:

SUPPOSE: ROUTER RECEIVES UPDATE ABOUT DESTINATION D FROM NEIGHBOR N

⇒ R CAN REACH D VIA N WITH COST $C = C_D + C_N$

HOW TO UPDATE TABLE?

Table has format <Destination, cost, next hop>

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

If you have existing entry <D, C_old, M>

- c < c_old => Update table <D, c, N> (BETTER ROUTE!)
- If c > c_old N == M
    // Topology has changed, route has a higher cost now (HIGHER COST!)
    Update table. <D, c, N>
- If c > c_old and N != M => Ignore (N is better)
- If c == c_old and N == M: refresh timeout

SEPARATELY: KEEP TRACK OF LAST UPDATE TIME FOR EACH ROUTE, DELETE OLD ROUTES THAT EXPIRE.

Distance Vector: Update rules

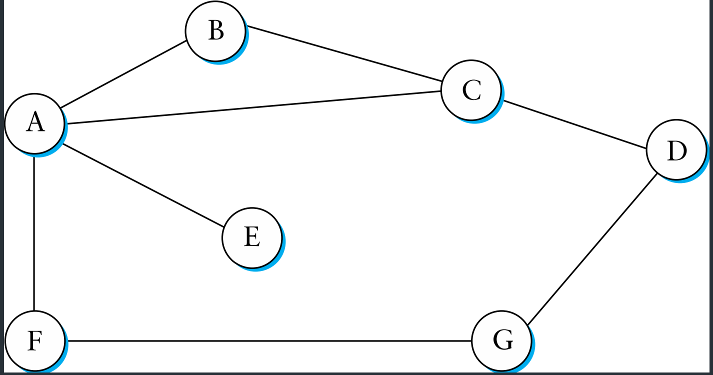
Say router R receives an update <D, c_D> from neighbor N at cost C_N

=> 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 $c > c_{old}$ <u>and M == N</u>: update table to <D, c, N> (Cost increased!)
   - if $c > c_{old}$ <u>and M != N</u>: ignore (N is better)
   - if $c == c_{old}$ and M ==N: no change (Just refresh timeout) (No new info)

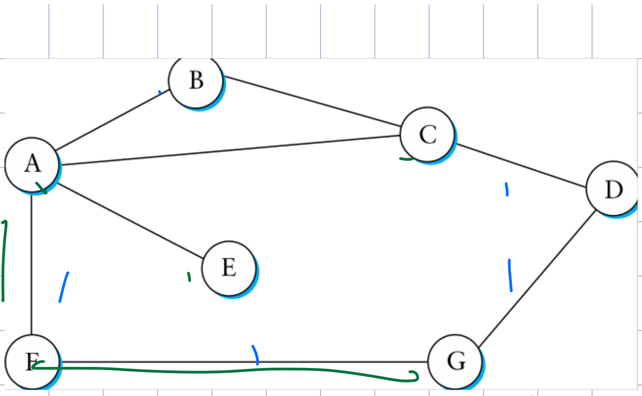
DV Example



B’s routing table

Dest.	Cost	Next Hop
(B)	(0)	(B)
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

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)

T_0 : B SENDS ITS TABLE (B, 0) to A, C

- A SENDS UPDATE (A, 0) ←
- C " " " (C, 0)

T_1 : - A SENDS UPDATE (A, 0)
(E, 1)
(F, 1)
(C, 1)

(E, 1) NEW! ADD

(F, 1) NEW! ADD

(C, 1) ALREADY HAVE THIS, NOT BETTER COST => NO CHANGE.

- . . .

Warmup

Suppose router R has the following table:

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

What happens when it gets this update from router S?

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

Warmup

ALL COSTS 1

R IS NEIGHBOR OF S

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 TO TABLE)

(TIE)

(COST INCREASE!)

(BETTER ROUTE!)

What happens when it gets this update from router S?

S

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

+1

+1

+1

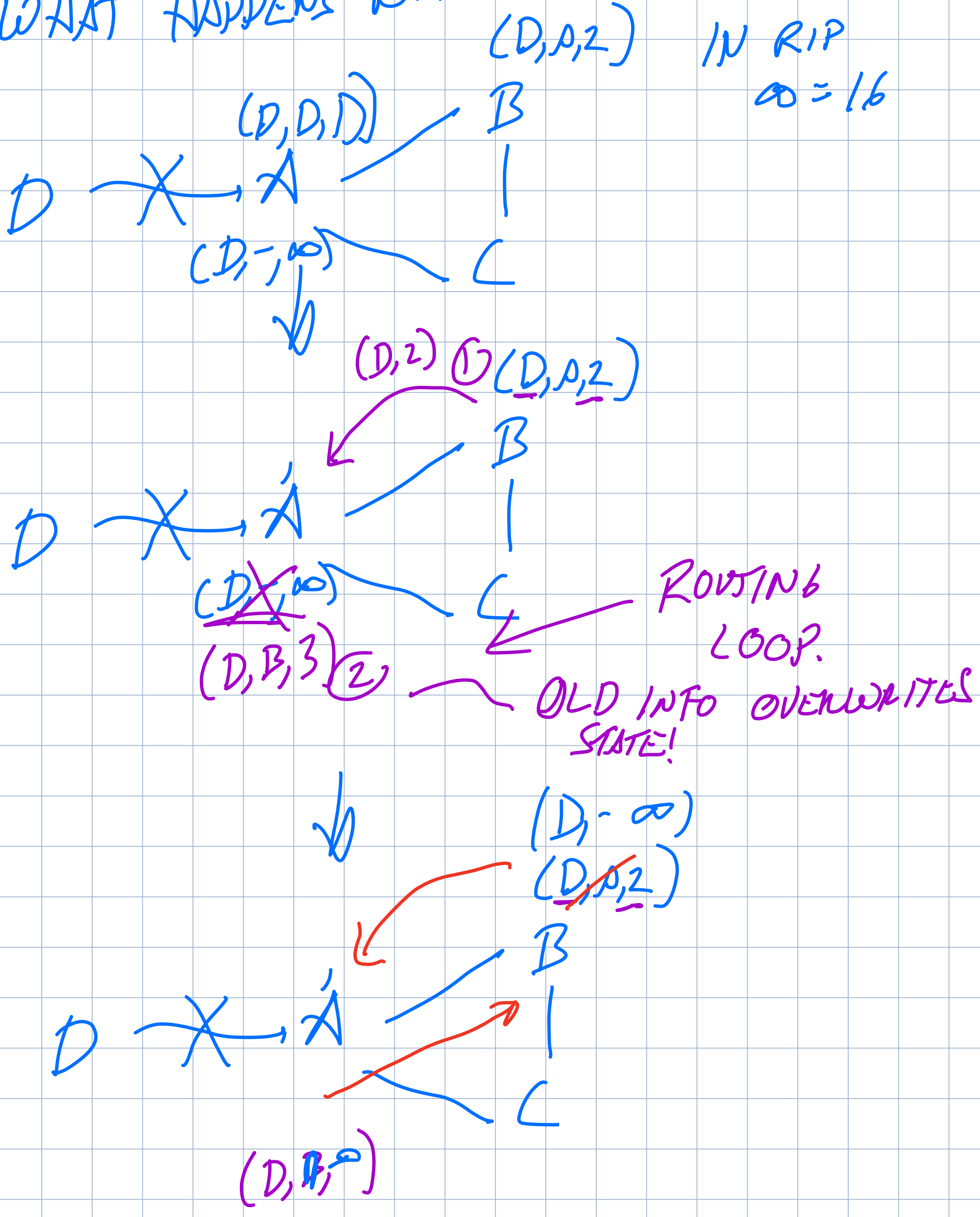
+1

+1

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?

IN RIP ∞ = 16



ROUTING LOOP.

OLD INFO OVERWRITES STATE!

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

⇒ COUNT TO INFINITY

- LONG CONVERGE TIME.

How to avoid loops

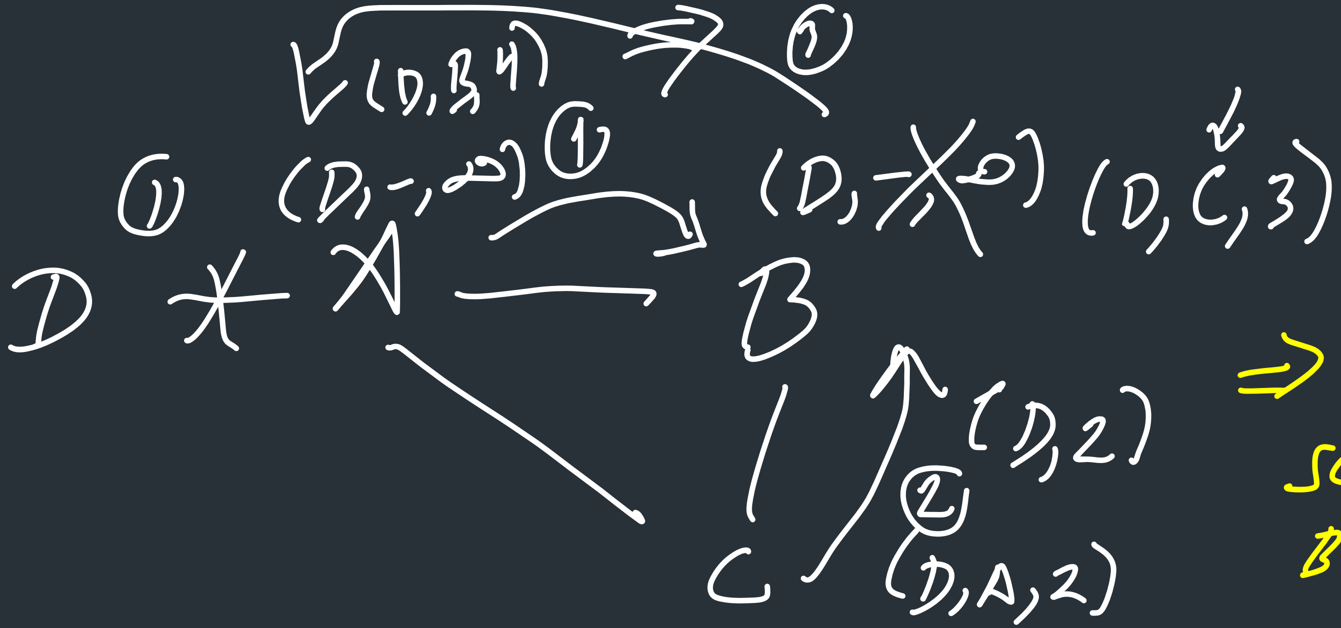
- Does IP TTL help? ⇒ DOESN'T SOLVE ROUTING PROBLEM.
- Simple approach: consider a small cost n (e.g., 16) to be infinity
  - After n rounds decide node is unavailable
  - But rounds can be long, this takes time

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



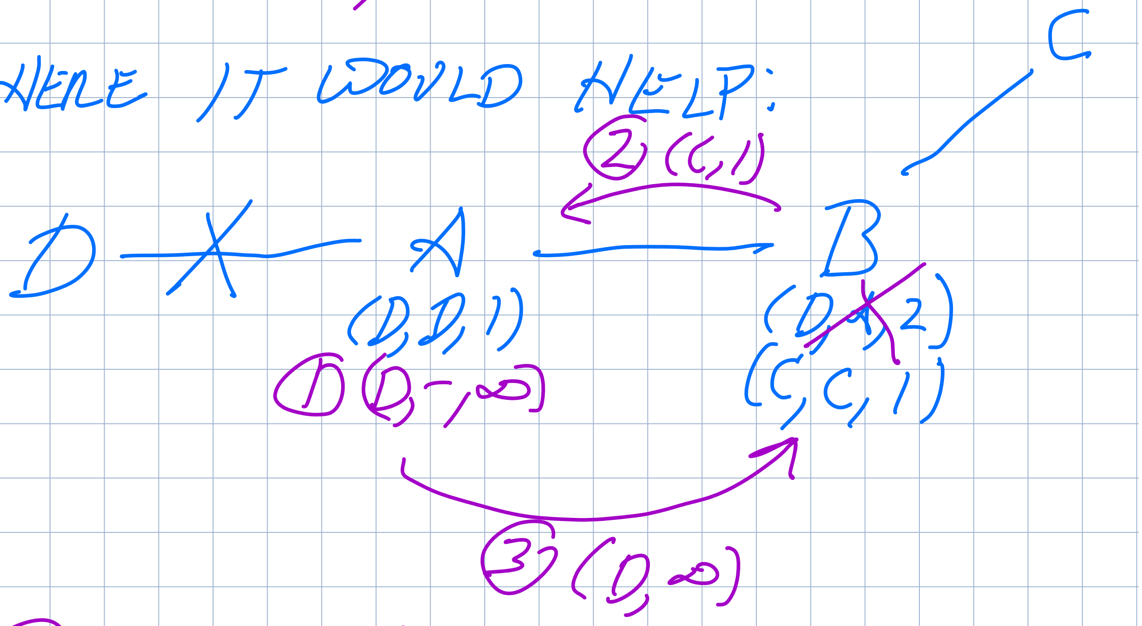
⇒ CAN PREVENT SOME LOOPS BUT NOT ALL.

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:



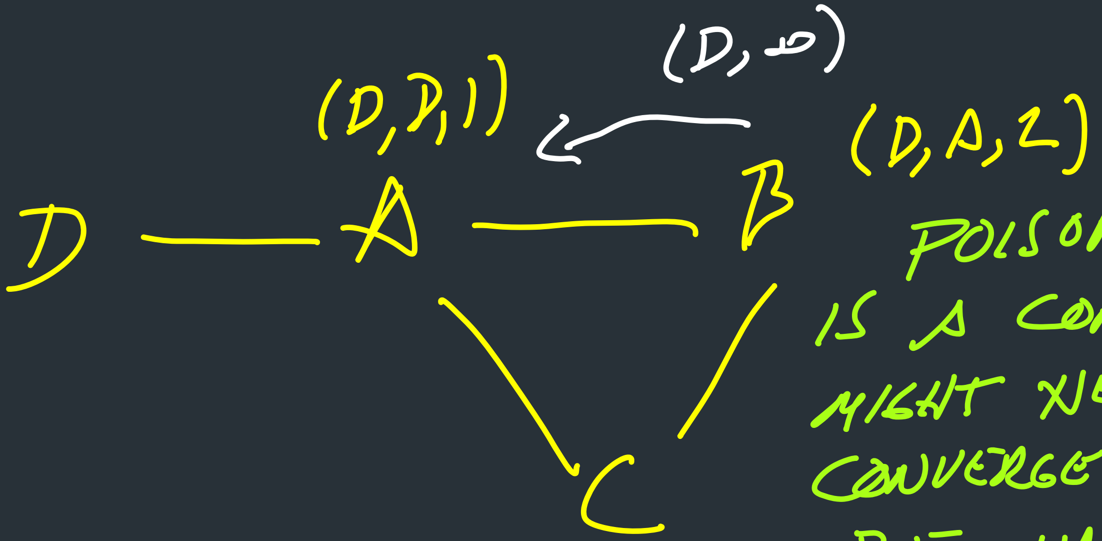
(1) D-A link goes down

(2) When B sends update to A, it would not tell include A

(3) A updates B w/ (D,∞)

Split Horizon + Poison Reverse

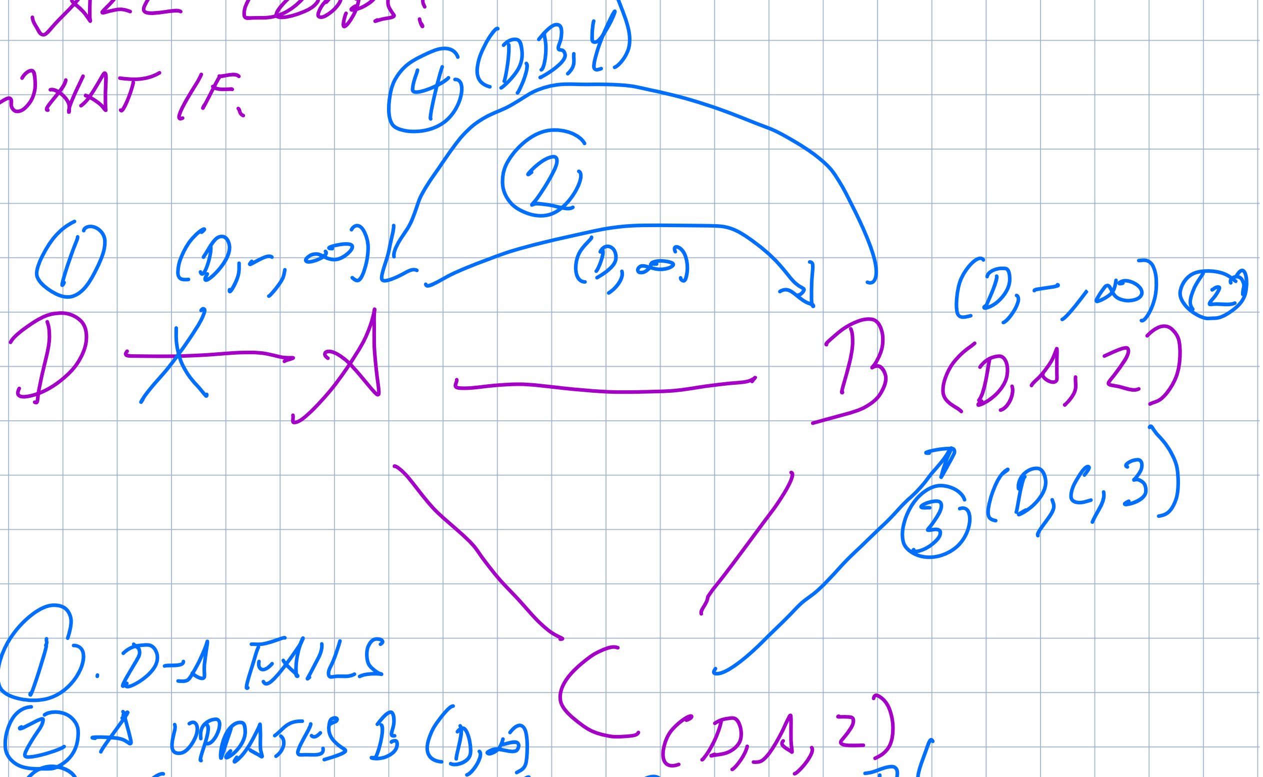
- Rather than not advertising routes learned from A, explicitly include cost of ∞.
- Faster to break out of loops, but increases advertisement sizes



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,∞)
③. C SENDS (D,2) TO B!

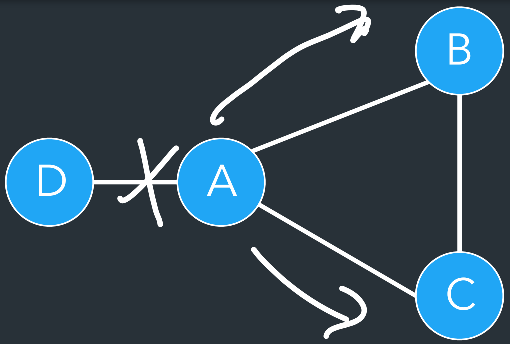
↳ RACE CONDITION! C MIGHT SEND OLD UPDATE TO B BEFORE C GETS UPDATE FROM A!

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

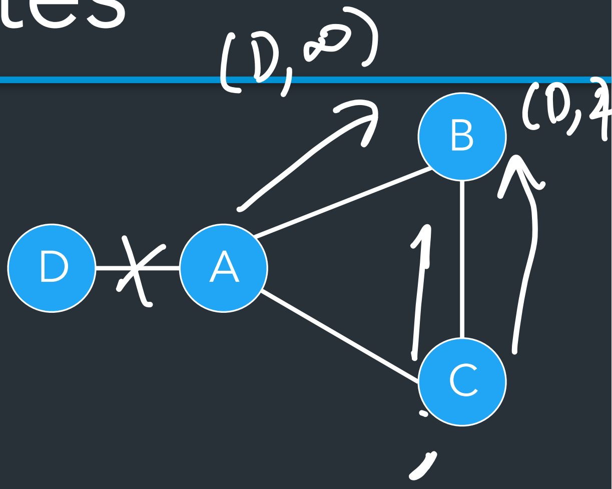


What else can we do? TRIGGERED UPDATES.

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

Distance-vector updates

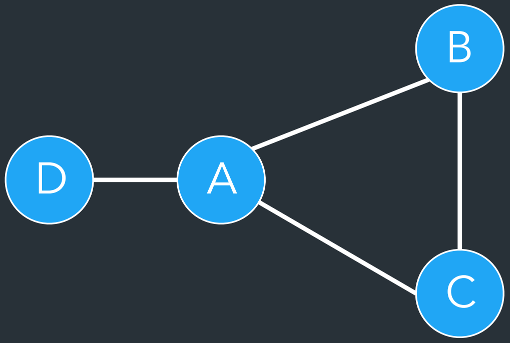
Even with split horizon + poison reverse, can still create loops with >2 nodes



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

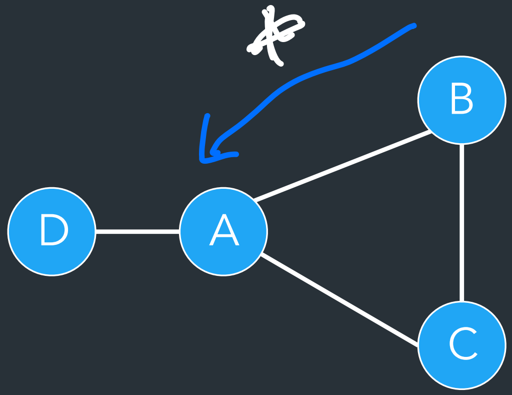


B's routing table

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.	SH + PR
(A, 1)	(A, ∞)
(C, 1)	(C, 1)
(D, 2)	(D, ∞)