CS 1680: Lecture 8

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

Administrivia

- Look for announcement to sign up for IP milestone meetings, preferably with your mentor TA; meetings on Fri/Mon/Tues
  - You will be expected to talk about your design
  - Don't need to have working code, but you should get started

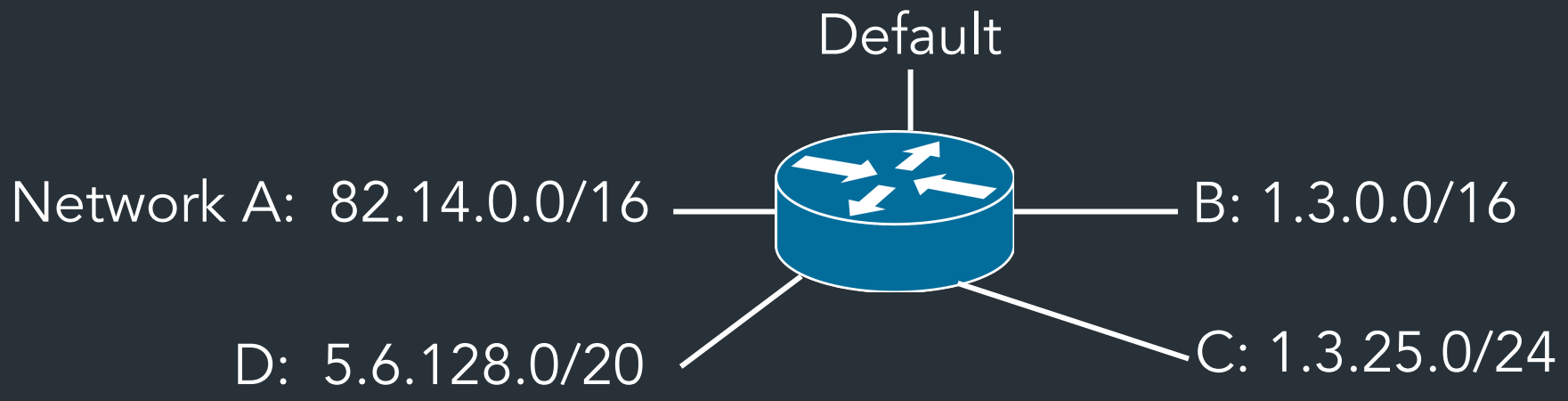
- IP gearup II: Look for an announcement
  - Implementation and debugging tips

Today

Fill in missing pieces about how IP works in practice, start routing

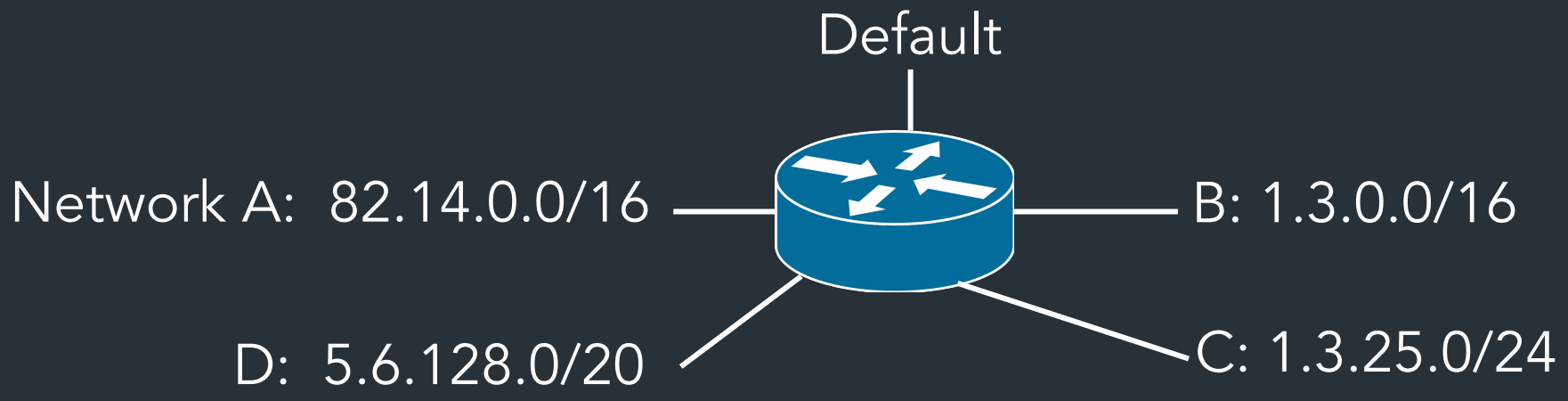
- Longest Prefix Match
- IP<->Link layer (ARP)
- Start of routing

After this: Routing



Prefix	IF/Next hop
82.14.0.0/16	(A)
1.3.0.0/16	(B)
1.3.4.0/24	(C)
5.6.128.0/20	(D)
0.0.0.0/0	(Default)

(X) is placeholder—could be an IP or an interface name



Prefix	IF/Next hop
82.14.0.0/16	(A)
1.3.0.0/16	(B)
1.3.4.0/24	(C)
5.6.128.0/20	(D)
0.0.0.0/0	(Default)

(X) is placeholder—could be an IP or an interface name

Warmup: based on the table, where would the router send packets destined for the following addresses:

1. 5.6.128.100
2. 1.3.1.1
3. 8.8.8.8
4. 1.3.4.8

What happens when prefixes overlap?

An IP can match on more than one row
=> need to pick the most specific (longest) prefix

Prefix	IF/Next hop
1.3.0.0/16	(B)
1.3.4.0/24	(C)
1.3.4.5/32	
0.0.0.0/0	(Default)

1.3.0.0/16 `00000001 00000011 xxxxxxxx xxxxxxxx`

1.3.4.0/24 `00000001 00000011 00000100 xxxxxxxx`

What happens when prefixes overlap?

An IP can match on more than one row
=> need to pick the most specific (longest) prefix

Prefix	IF/Next hop
1.3.0.0/16	(B)
1.3.4.0/24	(C)
1.3.4.5/32	
0.0.0.0/0	(Default)

1.3.0.0/16 `00000001 00000011 xxxxxxxx xxxxxxxx`

1.3.4.0/24 `00000001 00000011 00000100 xxxxxxxx`

More specific => best match!

What happens when prefixes overlap?

An IP can match on more than one row
=> need to pick the most specific (longest) prefix

Prefix	IF/Next hop
1.3.0.0/16	(B)
1.3.4.0/24	(C)
1.3.4.5/32	
0.0.0.0/0	(Default)

1.3.0.0/16 `00000001 00000011 xxxxxxxx xxxxxxxx`

1.3.4.0/24 `00000001 00000011 00000100 xxxxxxxx`

More specific => best match!

Other examples you'll see…

0.0.0.0/0 `xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx`

1.2.3.5/32 `00000001 00000011 00000100 00000101`

What happens when prefixes overlap?

An IP can match on more than one row
=> need to pick the most specific (longest) prefix

Prefix	IF/Next hop
1.3.0.0/16	(B)
1.3.4.0/24	(C)
1.3.4.5/32	
0.0.0.0/0	(Default)

1.3.0.0/16 `00000001 00000011 xxxxxxxx xxxxxxxx`

1.3.4.0/24 `00000001 00000011 00000100 xxxxxxxx`

More specific => best match!

Other examples you'll see…

0.0.0.0/0 `xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx` => Least specific!
(Used for default "catchall" routes)

1.2.3.5/32 `00000001 00000011 00000100 00000101` => Most specific!
(Refers to a single host,
often a local IP)

What happens when prefixes overlap?

An IP can match on more than one row
=> need to pick the most specific (longest) prefix

Prefix	IF/Next hop
1.3.0.0/16	(B)
1.3.4.0/24	(C)
1.3.4.5/32	
0.0.0.0/0	(Default)

1.3.0.0/16 `00000001 00000011 xxxxxxxx xxxxxxxx`

1.3.4.0/24 `00000001 00000011 00000100 xxxxxxxx`

More specific => best match!

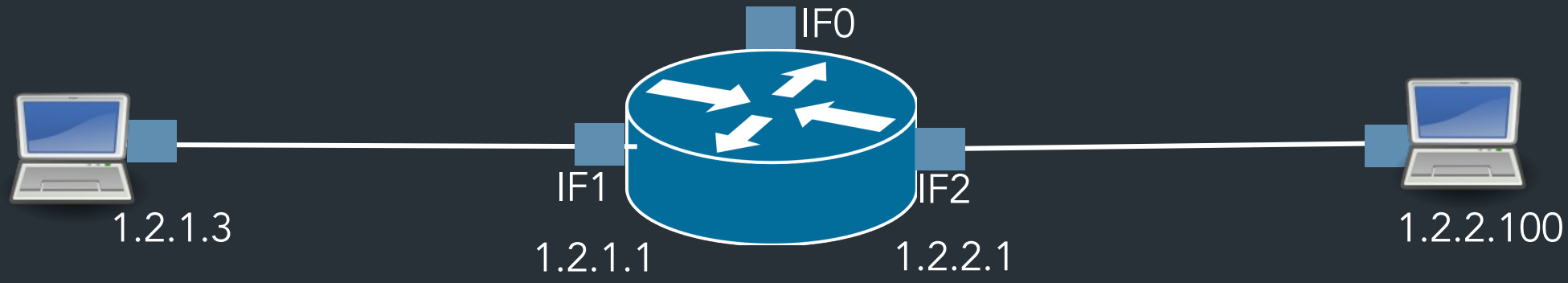
Other examples you'll see…

0.0.0.0/0 `xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx` => Least specific!
(Used for default "catchall" routes)

1.2.3.5/32 `00000001 00000011 00000100 00000101` => Most specific!
(Refers to a single host, often a local IP)

=>Longest prefix matching: can keep forwarding tables small by summarizing routes where possible, otherwise using specific prefixes

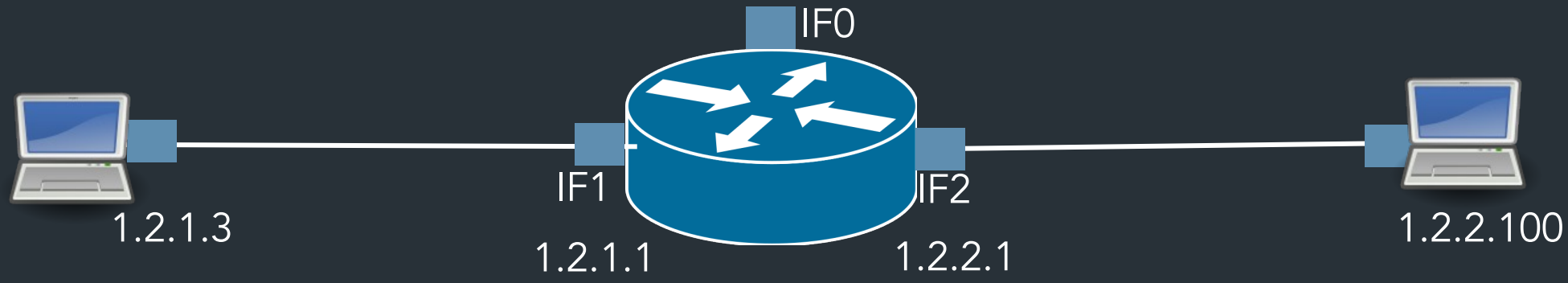
What happens at the link layer?



What does it mean to send to IF1?

Prefix	IF/Next hop
1.2.1.0/24	IF1
1.2.2.0/24	IF2
8.0.0.0/30	IF0
Default	8.0.0.2

What happens at the link layer?

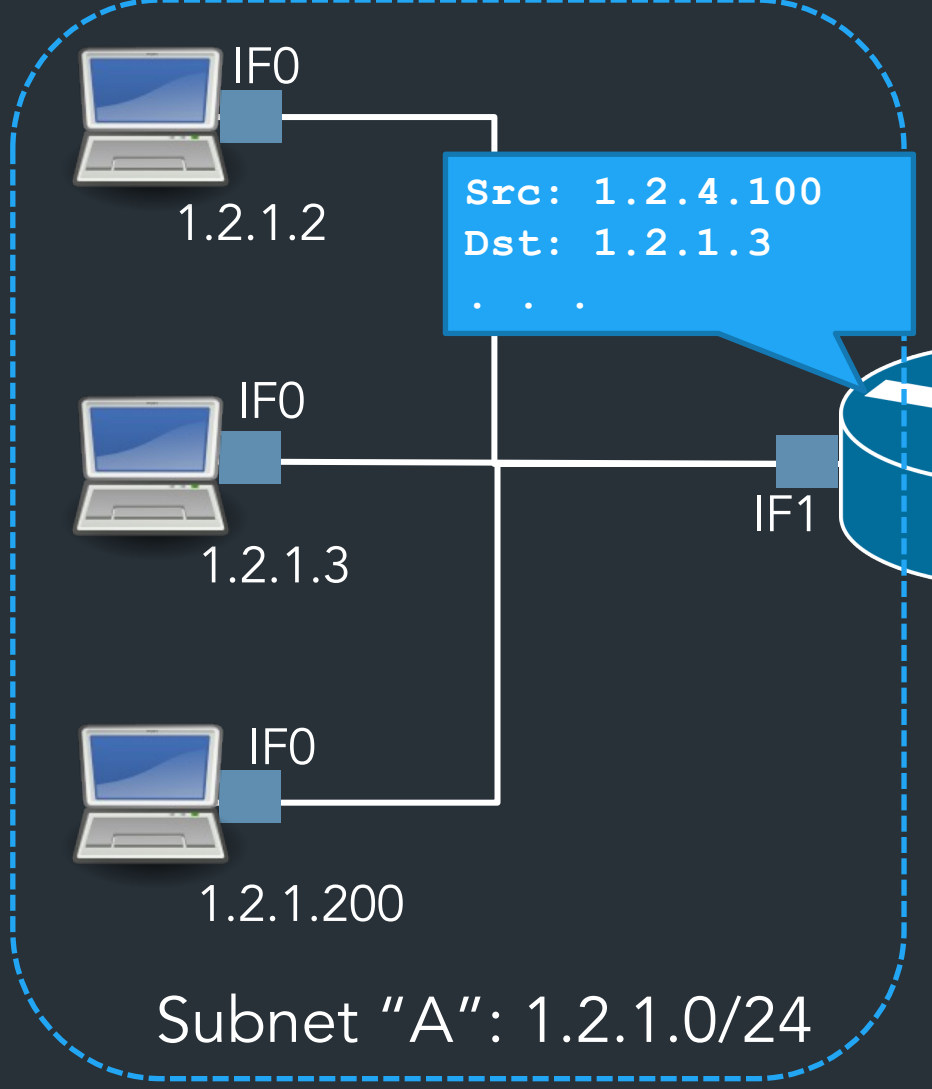


What does it mean to send to IF1?

The story so far:
=>Can “easily” communicate with nodes on the same network,
but what about other networks?
=> Routers know about multiple networks, forward packets between them

Prefix	IF/Next hop
1.2.1.0/24	IF1
1.2.2.0/24	IF2
8.0.0.0/30	IF0
Default	8.0.0.2

“Local delivery”: what does it mean to send to IF1?



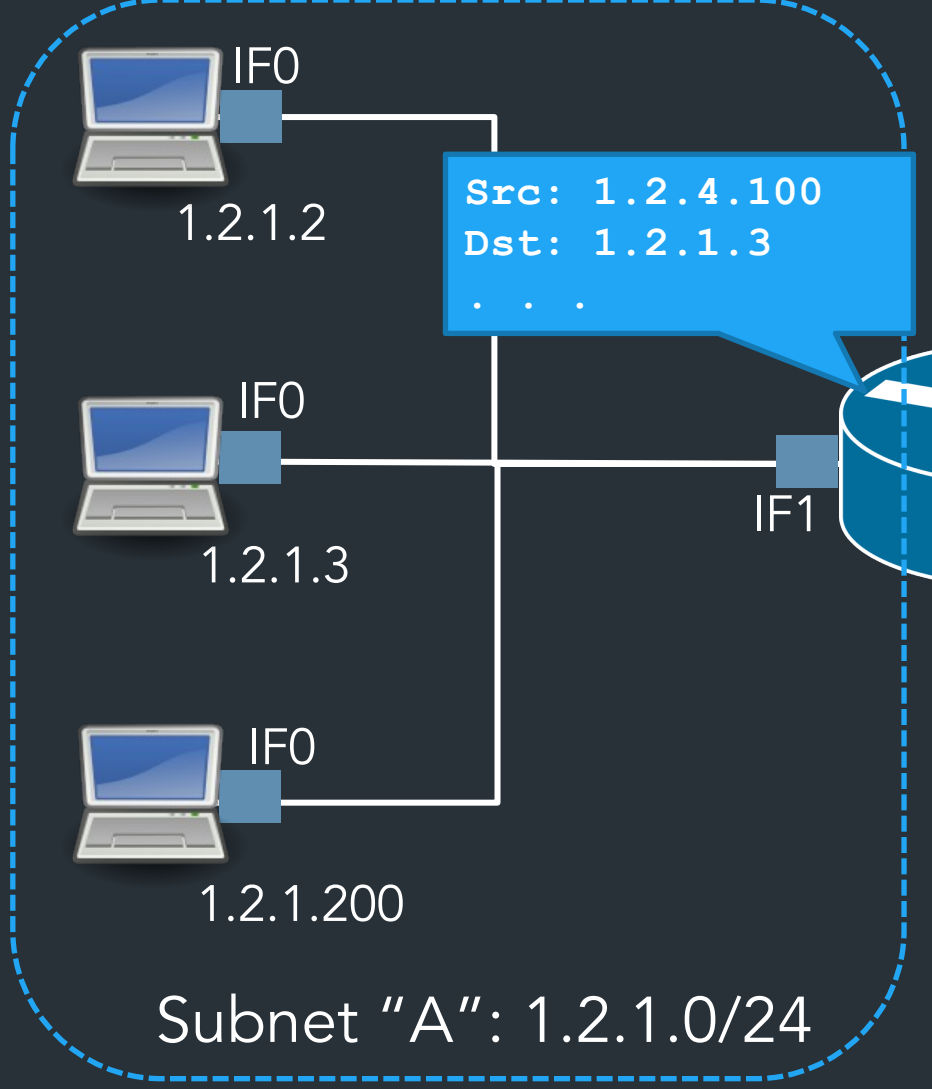
Prefix	IF/Next hop
1.2.1.0/24	IF1
…	…

“Local delivery”: what does it mean to send to IF1?

IF0, IF1, etc are interface names

=> Identify a network physically connected to a certain device

=> Names are specific to that device (and get reused)



Prefix	IF/Next hop
1.2.1.0/24	IF1
…	…

"Local delivery": what does it mean to send to IF1?

So far: "easy" to communicate with nodes on the same network. But how?

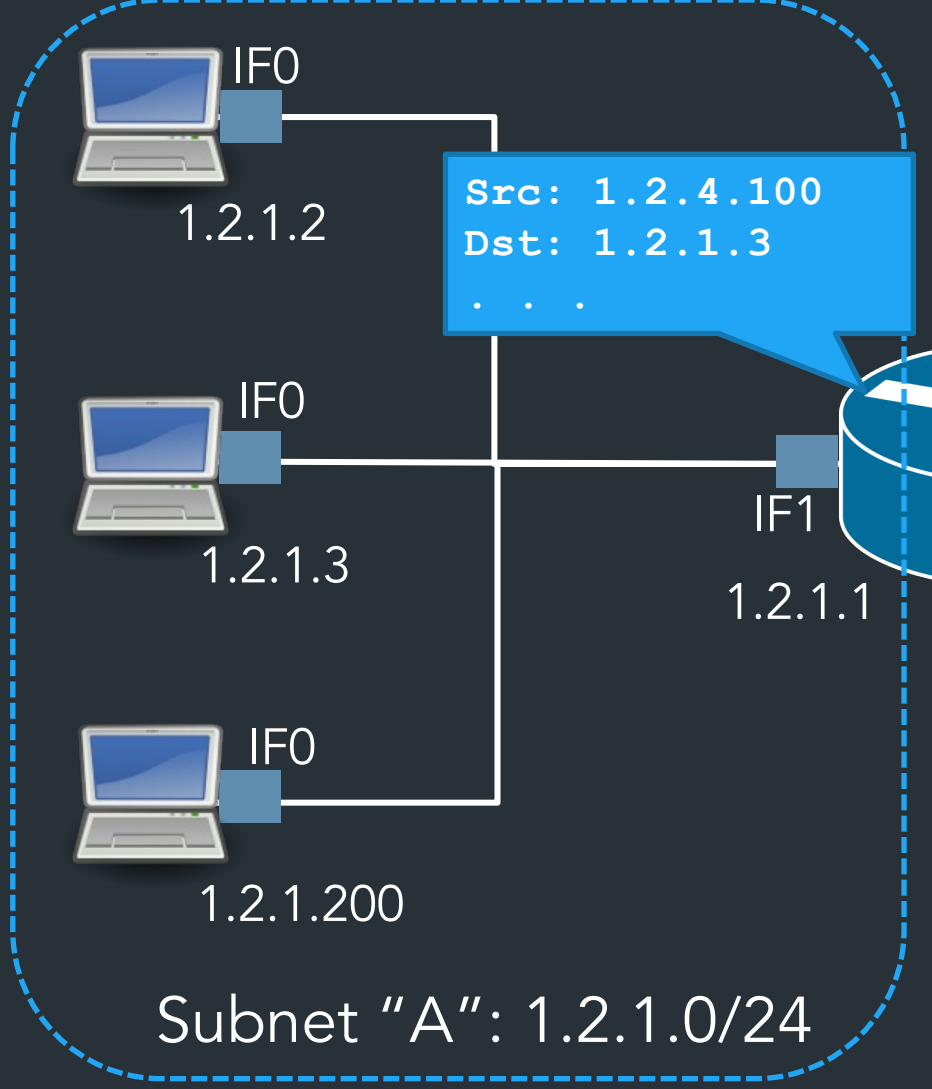
To send a packet on a local network, we need:

- Dest. IP (Network layer)
- Dest. MAC address (Link layer)

	Src	Dest
Link		???
IP	10.2.4.100	1.2.1.3

Assume: link layer can figure out the rest once we fill in this info

=> How do we get the MAC address?



Router's forwarding table

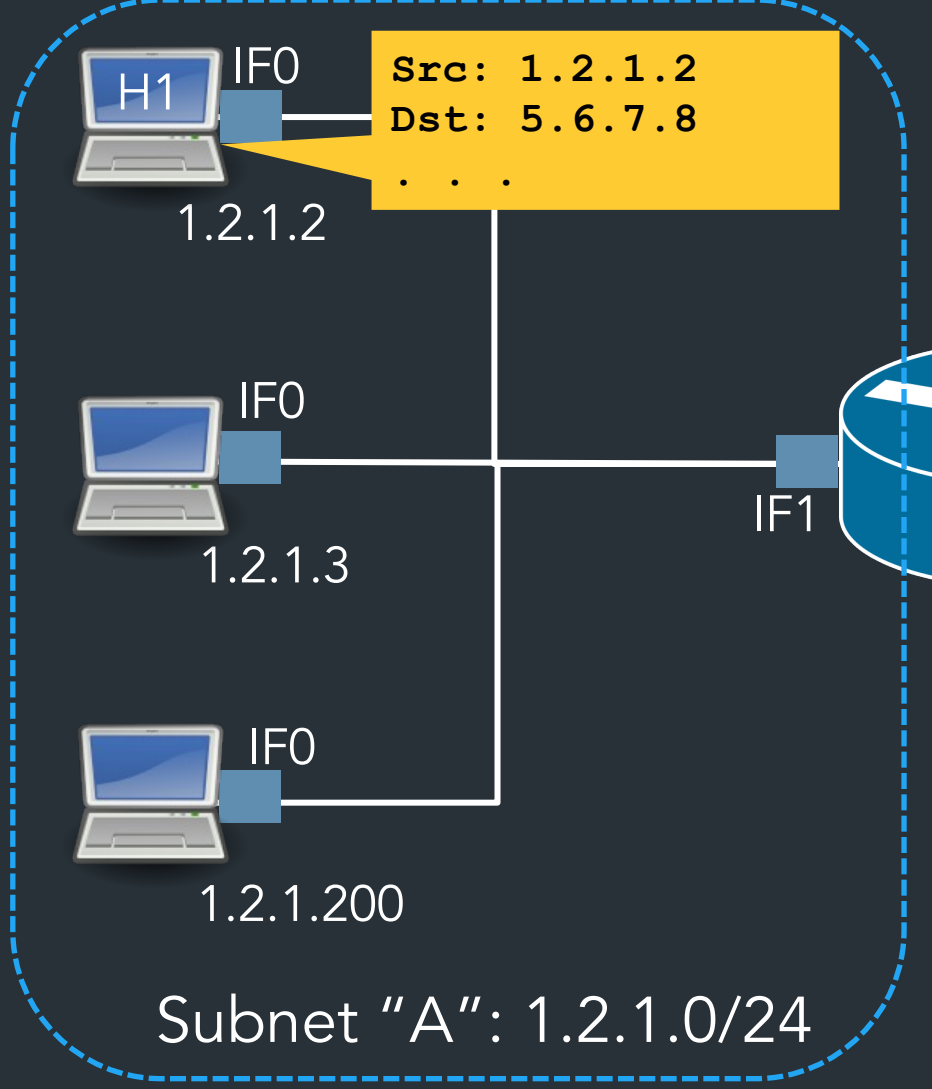
Prefix	IF/Next hop
1.2.1.0/24	IF1
…	…

What about non-local delivery?

H1's forwarding table

Prefix	IF/Next hop
1.2.1.0/24	IF0
0.0.0.0/0	1.2.1.1

	Src	Dest
Link		
IP	1.2.1.2	5.6.7.8



IP address: where the packet is going globally (i.e., on the internet)
=> Final destination

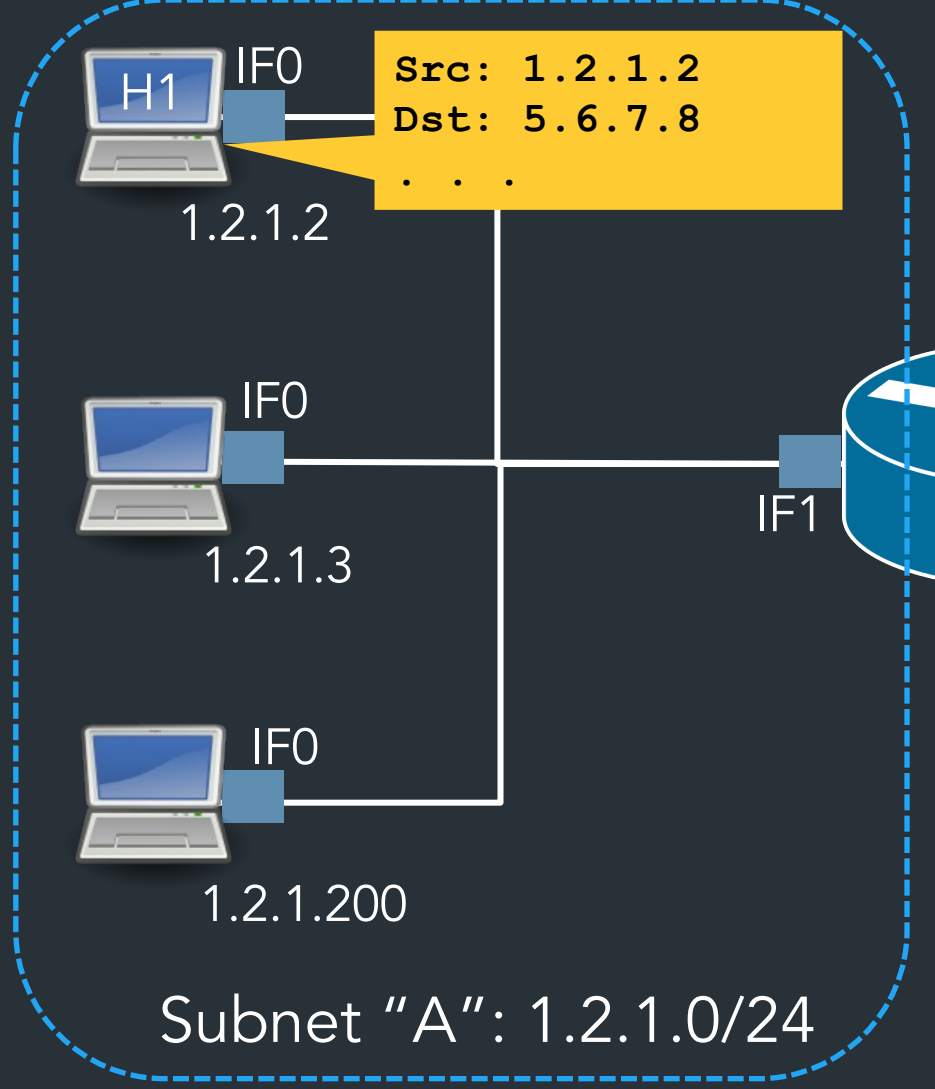
MAC address: where the packet is going locally: on this network
=> "Next hop"

What about non-local delivery?

H1's forwarding table

Prefix	IF/Next hop
1.2.1.0/24	IF0
0.0.0.0/0	1.2.1.1

	Src	Dest
Link		
IP	1.2.1.2	5.6.7.8



“Glue” between L2 and L3

Need a way to connect get link layer info (mac address) from network-layer info (IP address)

“What MAC address has IP 1.2.3.4?”

“Glue” between L2 and L3

Need a way to connect get link layer info (mac address) from network-layer info (IP address)

“What MAC address has IP 1.2.3.4?”

Solution: ask the network!
=> Address Resolution Protocol (ARP)

ARP: Address resolution protocol

Given an IP address, ask network for the MAC address

- Request: `“Who has 1.2.3.4?”`
- Response: `“aa:bb:cc:dd:ee:ff is at 1.2.3.4”`

How ARP works

ARP: Address resolution protocol

Given an IP address, ask network for the MAC address

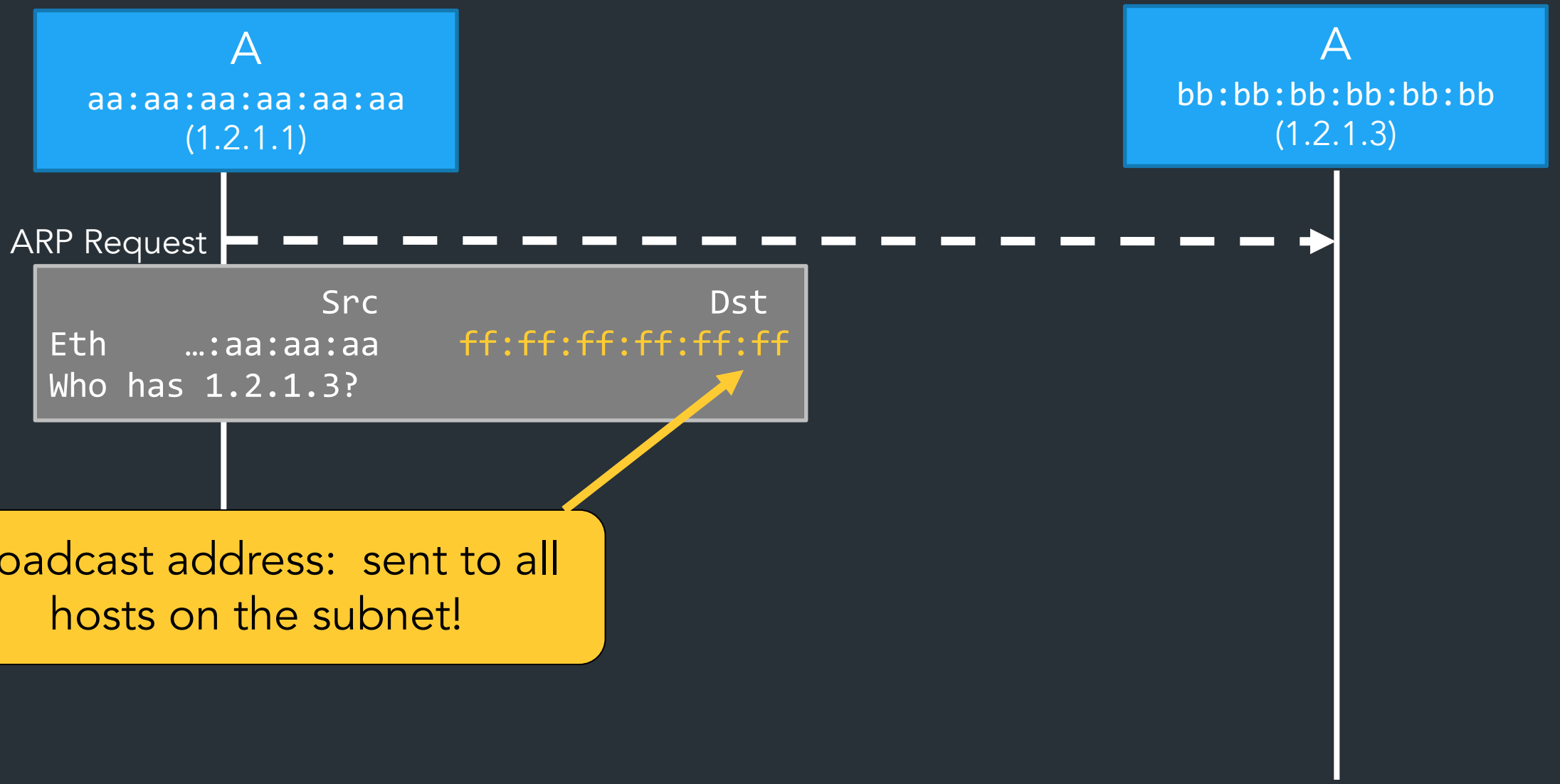
Request: “`Who has 1.2.3.4?`”

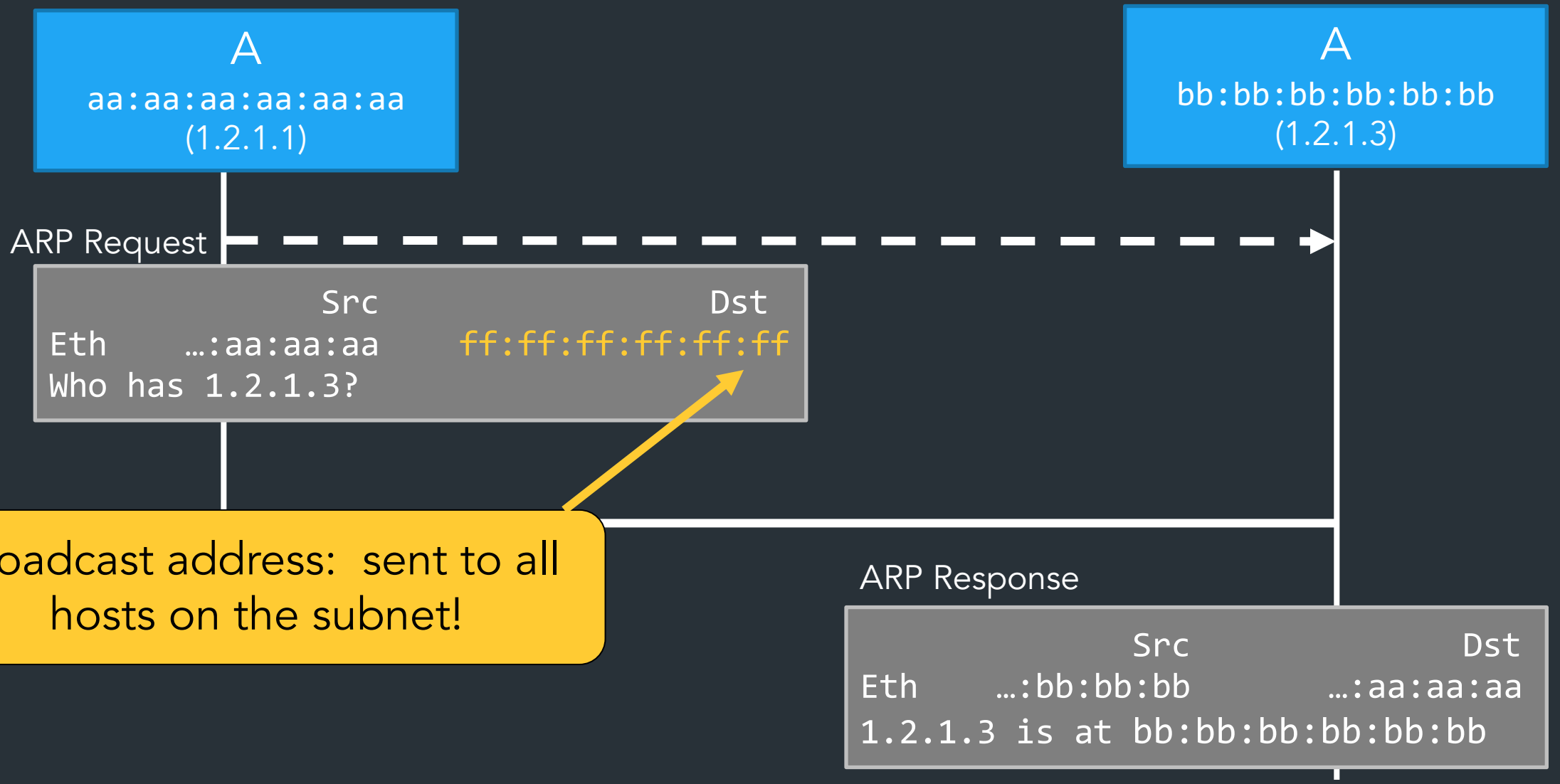
Response: “`aa:bb:cc:dd:ee:ff is at 1.2.3.4`”

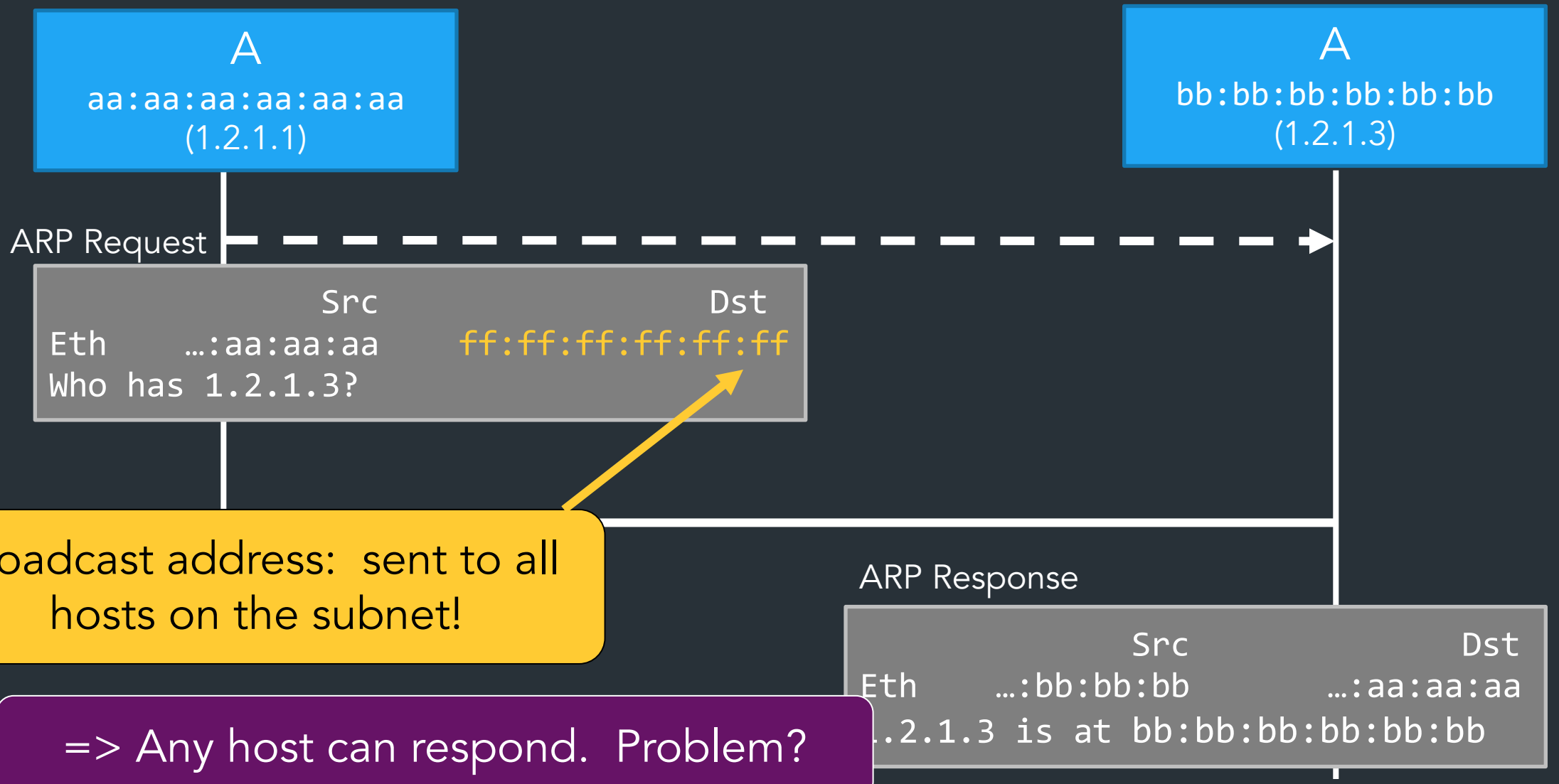
Key data structure: ARP table: map of IP -> MAC address

- All devices use ARP protocol to build their own table
- Requests send to *broadcast address*: `ff:ff:ff:ff:ff:ff`


A
aa:aa:aa:aa:aa:aa
(1.2.1.1)
A
bb:bb:bb:bb:bb:bb
(1.2.1.3)


A
aa:aa:aa:aa:aa:aa
(1.2.1.1)
A
bb:bb:bb:bb:bb:bb
(1.2.1.3)
ARP Request
Src
Dst
Eth …:aa:aa:aa
ff:ff:ff:ff:ff:ff
Who has 1.2.1.3?
Broadcast address: sent to all hosts on the subnet!

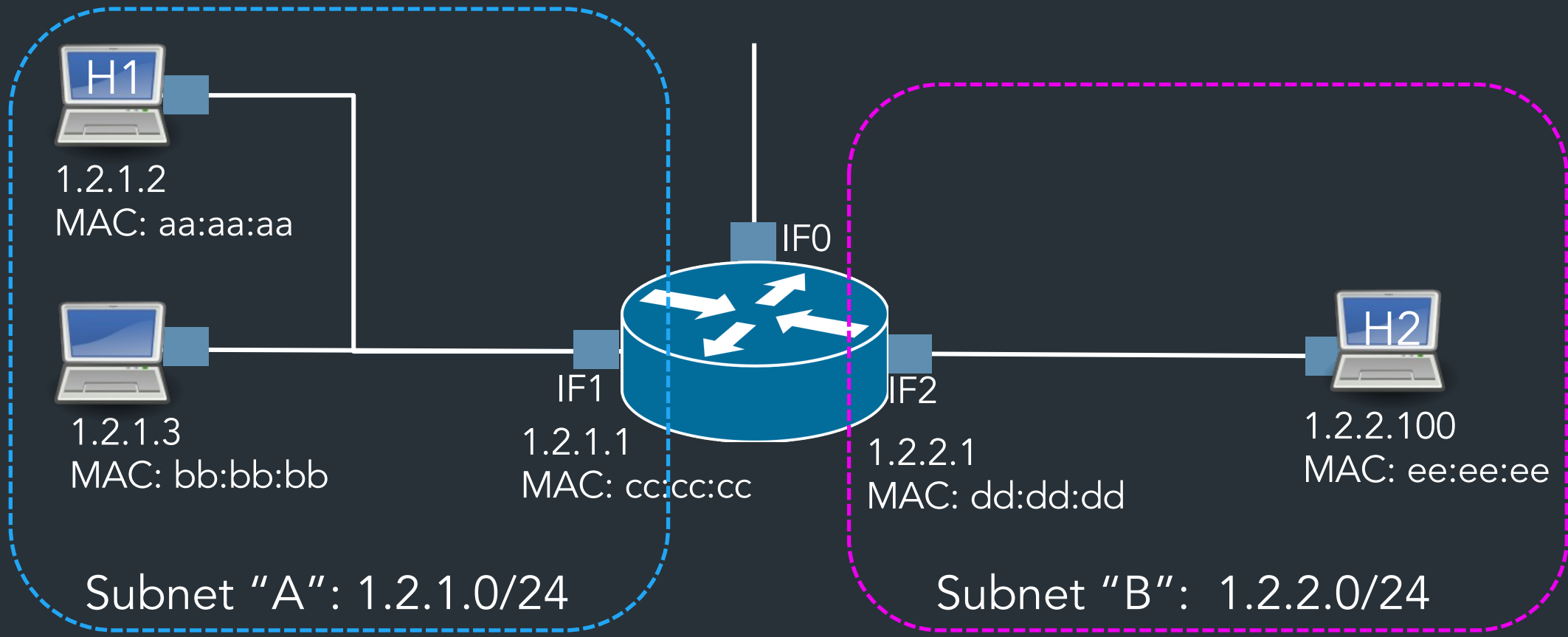

A
aa:aa:aa:aa:aa:aa
(1.2.1.1)
A
bb:bb:bb:bb:bb:bb
(1.2.1.3)
ARP Request
Src
Dst
Eth …:aa:aa:aa
ff:ff:ff:ff:ff:ff
Who has 1.2.1.3?
Broadcast address: sent to all hosts on the subnet!
ARP Response
Src
Dst
Eth …:bb:bb:bb
…:aa:aa:aa
1.2.1.3 is at bb:bb:bb:bb:bb:bb

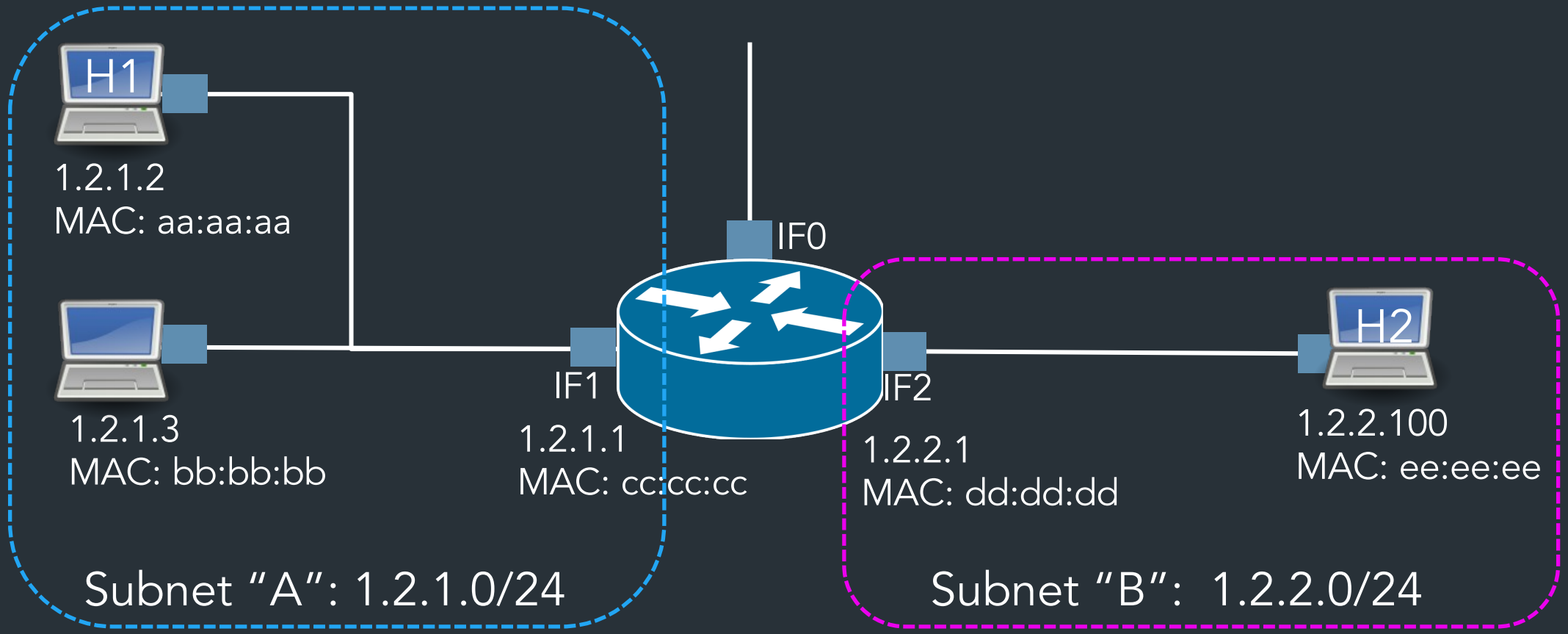

A
aa:aa:aa:aa:aa:aa
(1.2.1.1)
A
bb:bb:bb:bb:bb:bb
(1.2.1.3)
ARP Request
Src
Dst
Eth …:aa:aa:aa
ff:ff:ff:ff:ff:ff
Who has 1.2.1.3?
Broadcast address: sent to all hosts on the subnet!
ARP Response
Src
Dst
Eth …:bb:bb:bb
…:aa:aa:aa
.2.1.3 is at bb:bb:bb:bb:bb:bb
=> Any host can respond. Problem?

Example

```
# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
172.17.44.1              ether   00:12:80:01:34:55   C                     eth0
172.17.44.25             ether   10:dd:b1:89:d5:f3   C                     eth0
172.17.44.6              ether   b8:27:eb:55:c3:45   C                     eth0
172.17.44.5              ether   00:1b:21:22:e0:22   C                     eth0
```

Putting it all together….


H1
1.2.1.2
MAC: aa:aa:aa
1.2.1.3
MAC: bb:bb:bb
IF0
IF1
1.2.1.1
MAC: cc:cc:cc
IF2
1.2.2.1
MAC: dd:dd:dd
H2
1.2.2.100
MAC: ee:ee:ee
Subnet "A": 1.2.1.0/24
Subnet "B": 1.2.2.0/24



Suppose H1 wants to send a packet to H2.

Q: What would the headers look like when the packet leaves H1?

Q: Would it change after reaching R?

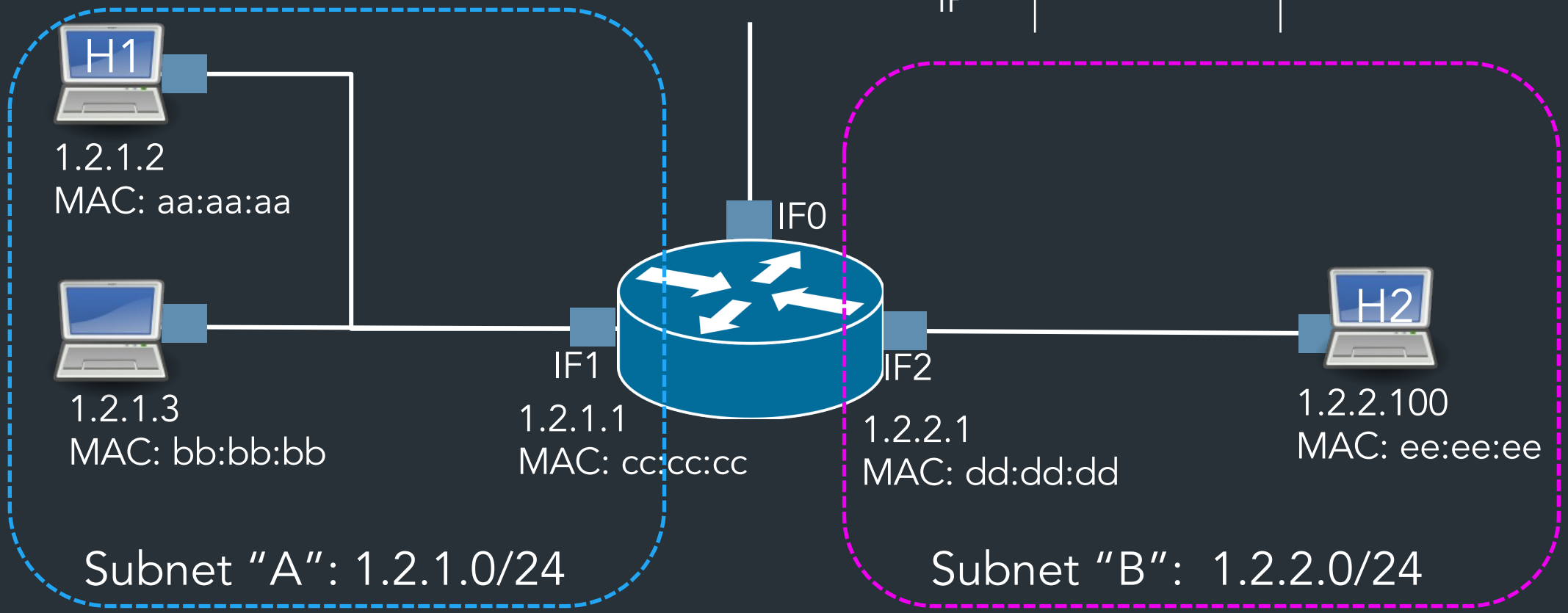
	Src	Dest
Link		
IP		

Suppose H1 wants to send a packet to H2.

Q: What would the headers look like leaving H1?

Q: Would it change after leaving the router?

	Src	Dest
Link		
IP		



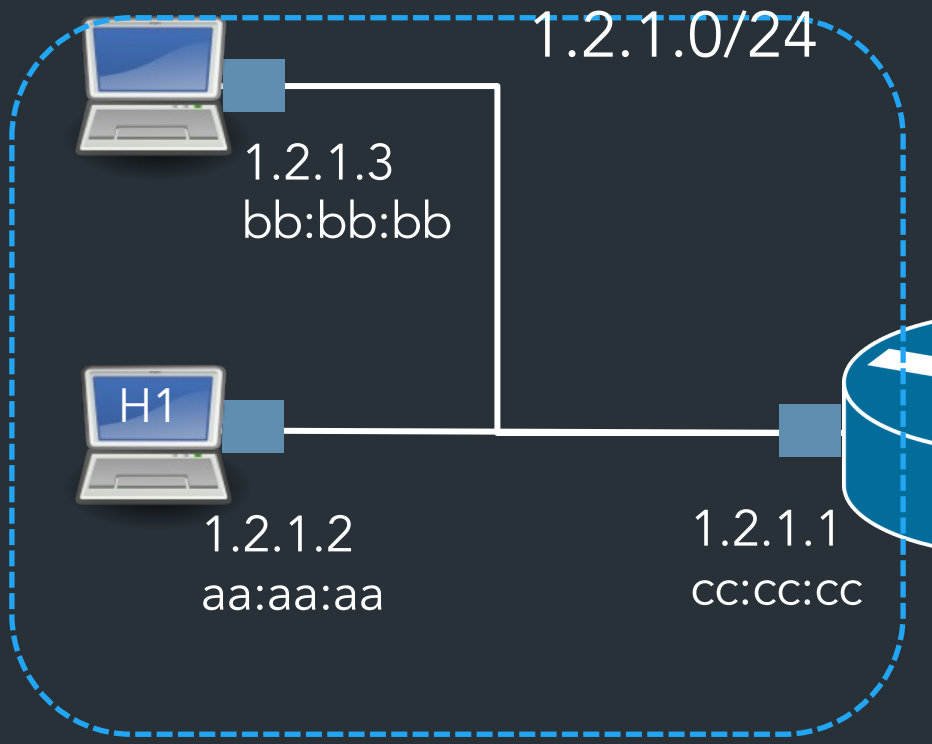
H1's forwarding table

Prefix	IF/Next hop
1.2.1.0/24	IF0
0.0.0.0/0	1.2.1.1

Router's forwarding table

Prefix	IF/Next hop
1.2.1.0/24	IF1
1.2.2.0/24	IF2

Practice: what is the destination MAC address when H1 is sending the following packets?



1)

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

2)

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

3)

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

H1's forwarding table:

Prefix	IF/Next hop
1.2.1.0/24	IF1
0.0.0.0	1.2.1.1

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)

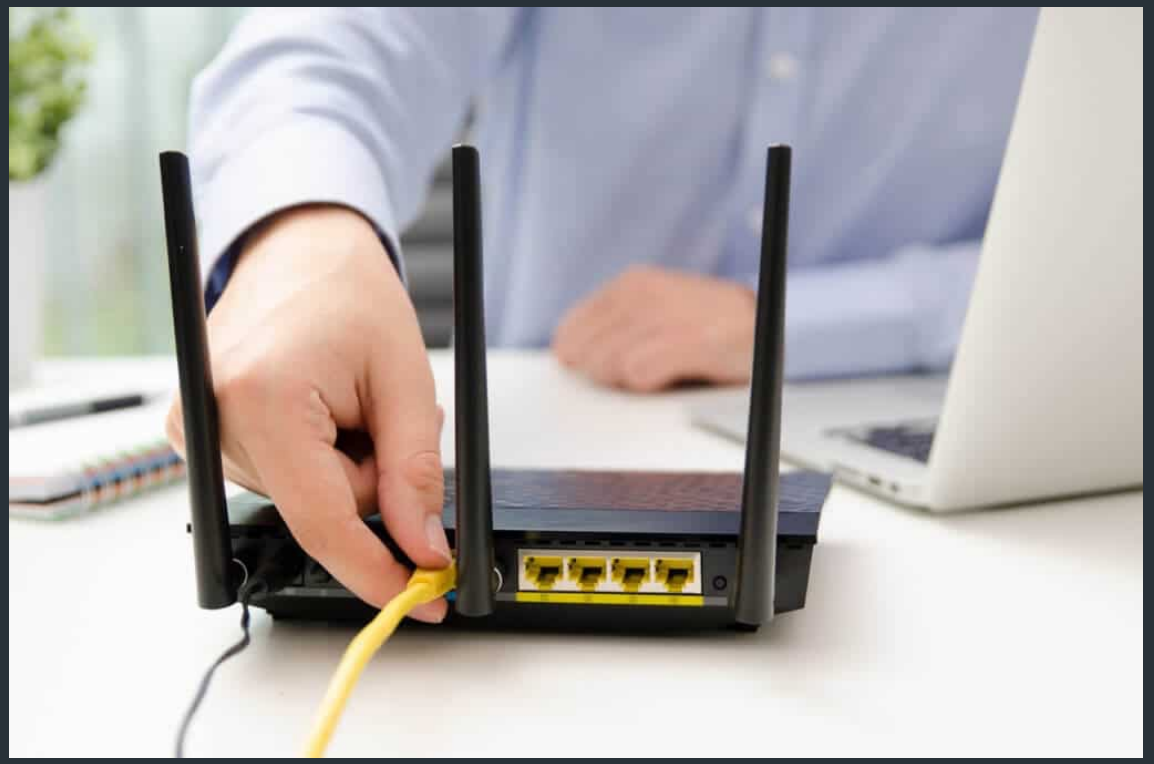
Link-layer header info (Ethernet/Wifi/etc)

- 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)

What we're still missing



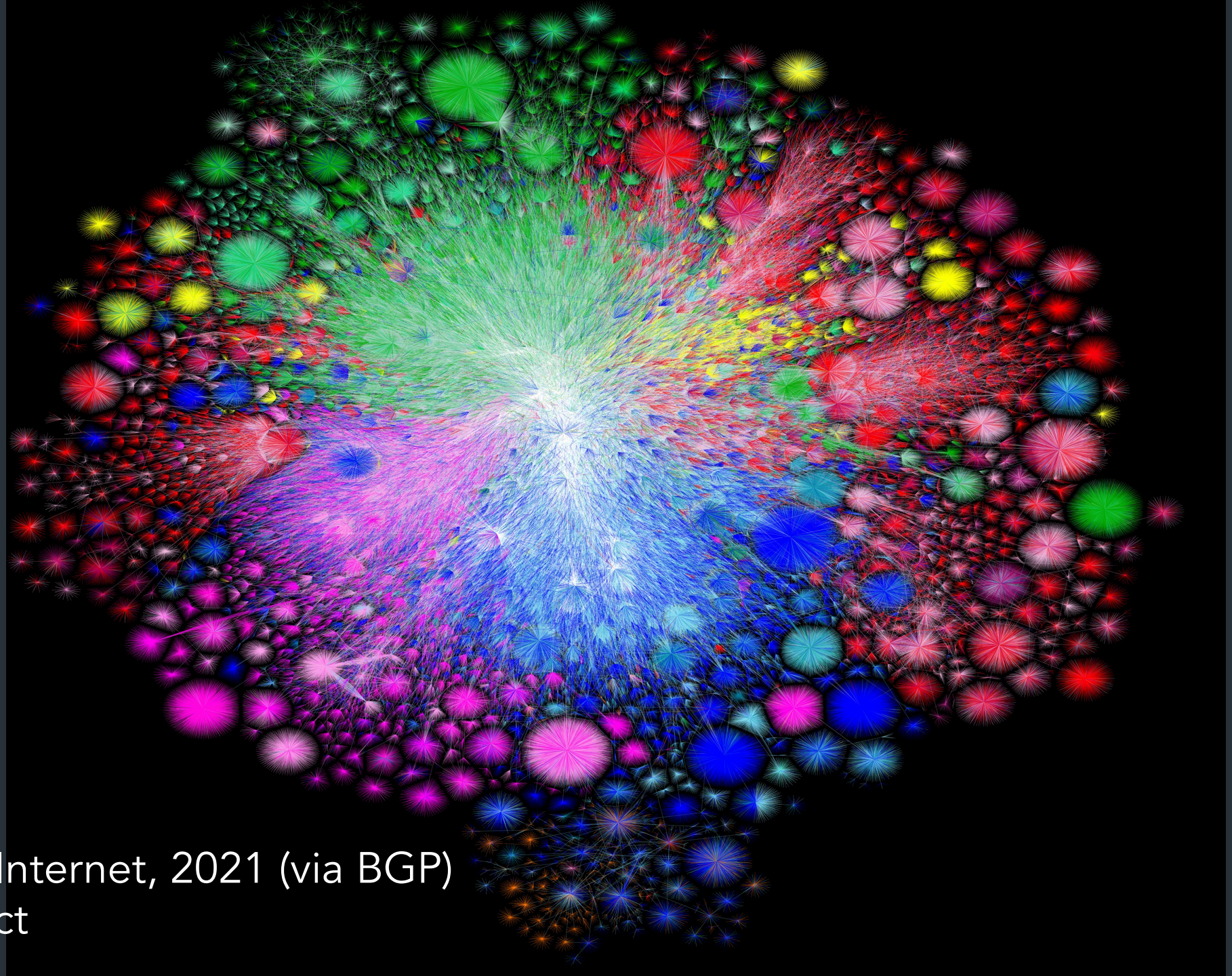
What does this thing do?

What we're still missing

... in order to make IP actually work in practice

=> How do you get an IP address? (DHCP)
=> What your home router does (NAT)
=> IPv6

We'll see the good, the bad, and the ugly...



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

Routing

Challenges in moving packets

- Forwarding:

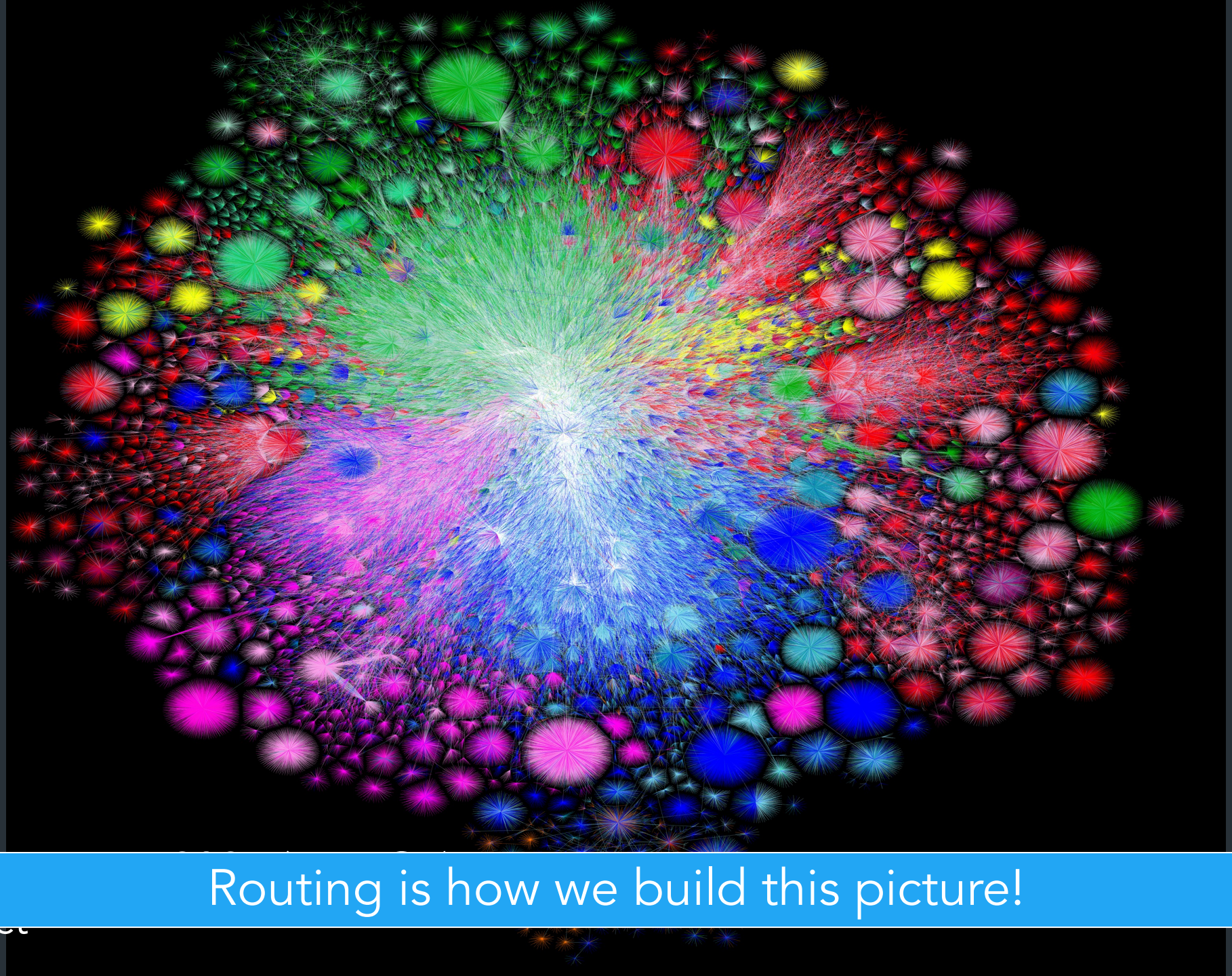
- Routing:

Challenges in moving packets

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

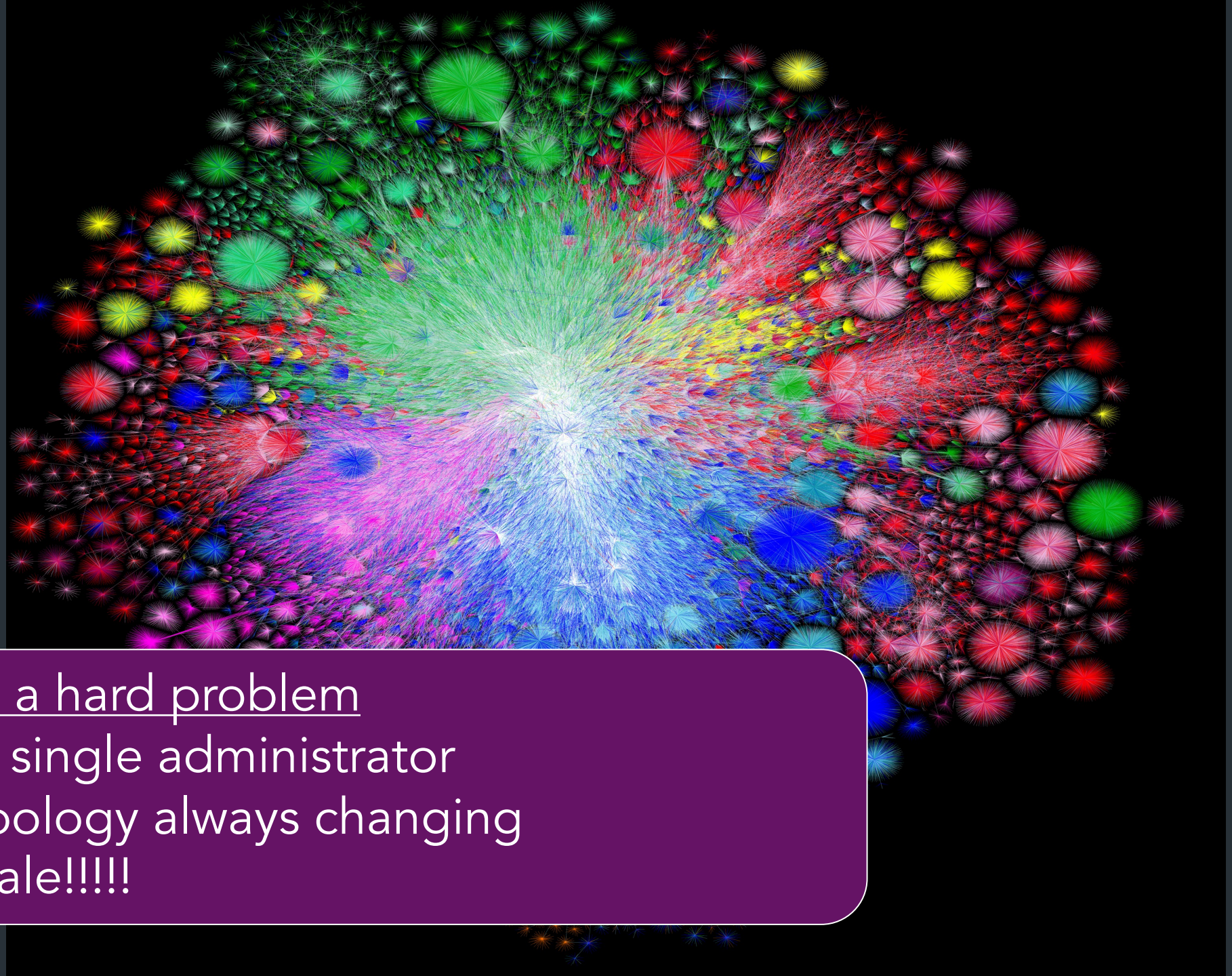
- Routing: network-wide process of determining a packet's path through the network

  => How each router builds its forwarding table



Map of th
OPTE proj

Routing is how we build this picture!



This is a hard problem

⇒ No single administrator

⇒ Topology always changing

=> Scale!!!!!

A forwarding table (my laptop)

```
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
```

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

- Core routers know everything

How do we connect *everything*?

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

⇒ Has default route

Core routers have no default, but may not have details…

=> Route packets to a certain network, which can figure it out from there

At scale, 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

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

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

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

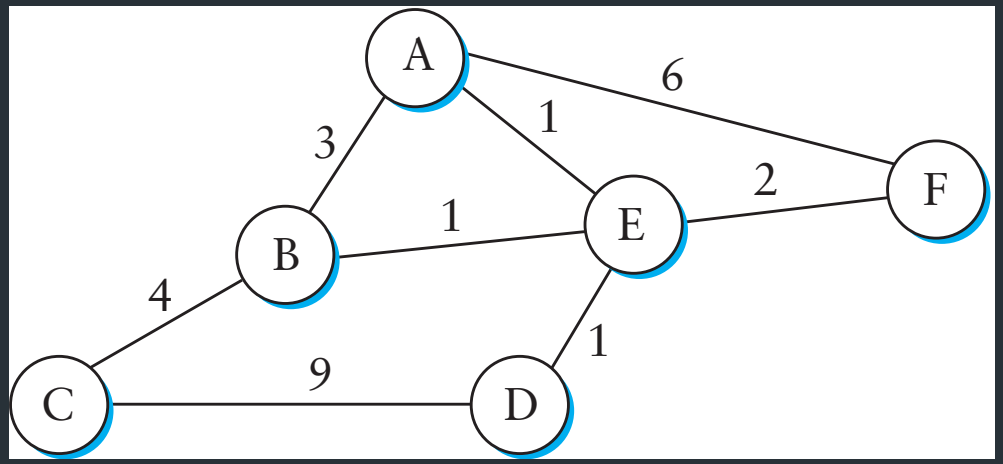
- Inter-domain (exterior) routing: routing between ASes

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

Intra-Domain (Interior) Routing

Typically, view network as a graph

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

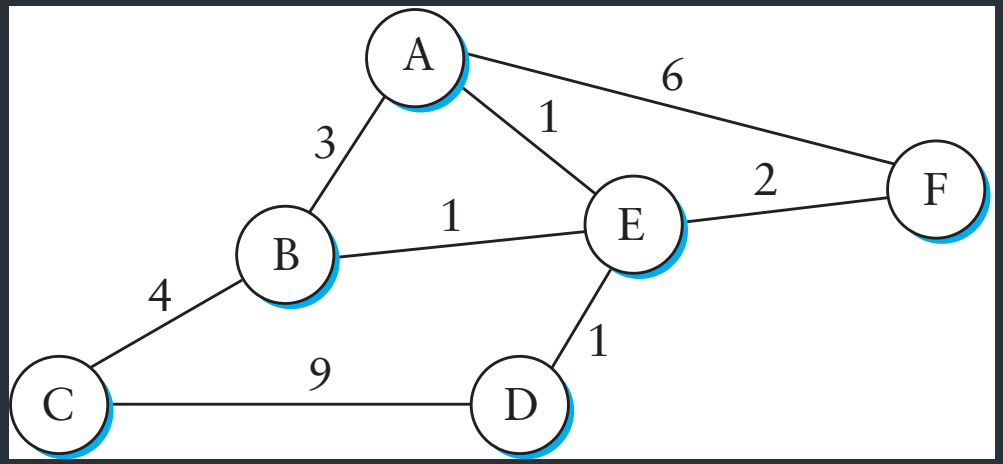


Goal: find lowest-cost path between nodes

- 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

Generally: routing algorithms are *decentralized*

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

Distance Vector (Bellman-Ford shortest path algorithm)

Link State (Djikstra/Prim shortest path algorithm)

Two classes of intra-domain routing algorithms

Distance Vector (Bellman-Ford shortest path algorithm)

- Idea: routers get updates from their neighbors

Link State (Djikstra/Prim shortest path algorithm)

Distance Vector Routing

- Each node maintains a *routing table*

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

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

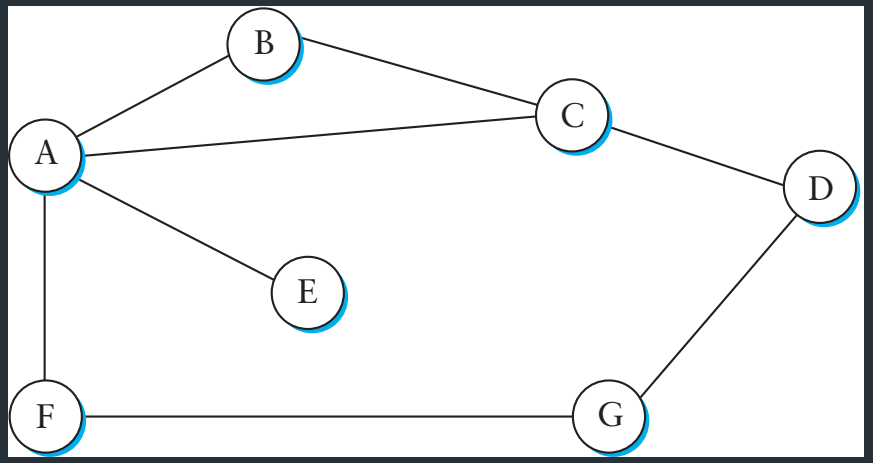
Dest.	Cost	Next Hop
A	3	S
B	4	T
C	5	S
D	6	U

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

How it works



Distance Vector: Update rules

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

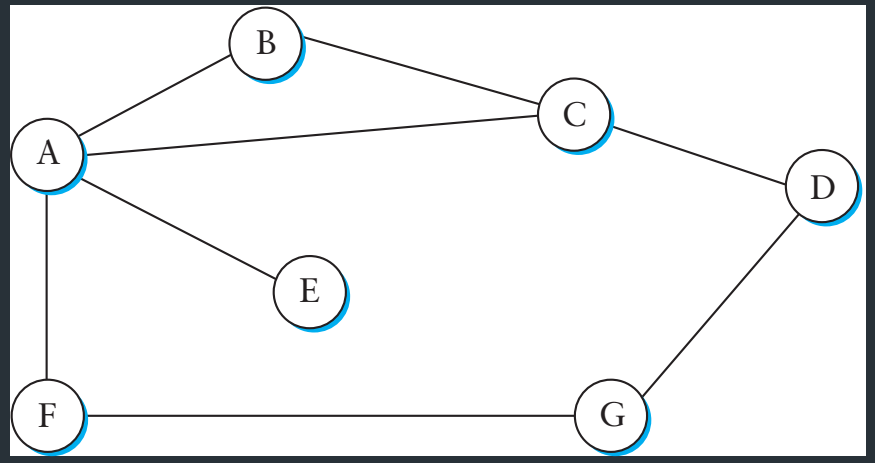
=> Know: R can reach D via N with cost c = c_D + c_N

How to update table?

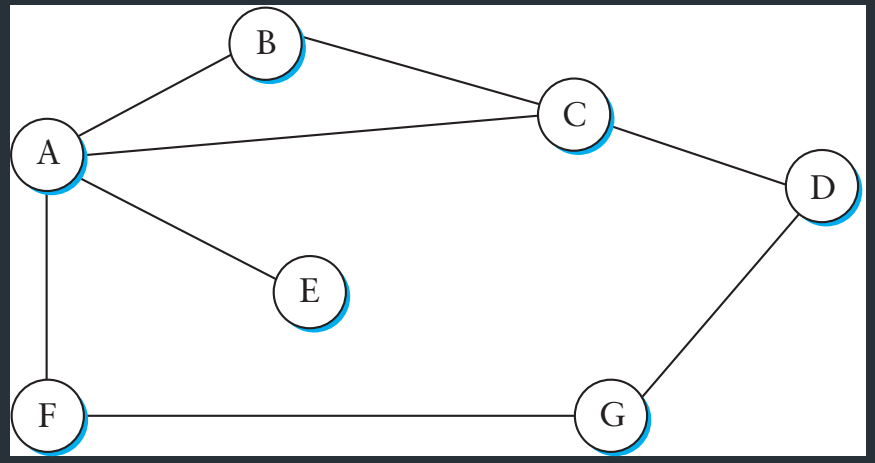
1. If D not in table, add <D, c, N> (New route!)
2. If table has entry <D, M, c_{old}>:
   - if c < c_{old}: update table to <D, c, M>. (Lower cost!)
   - if c > c_{old} and M == N: update table to <D, c, N> (Cost increased!)
   - if c > c_{old} and M != N: ignore (N is better)
   - if c == c_{old} and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

(Just refresh timeout)

DV Example



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

Warmup

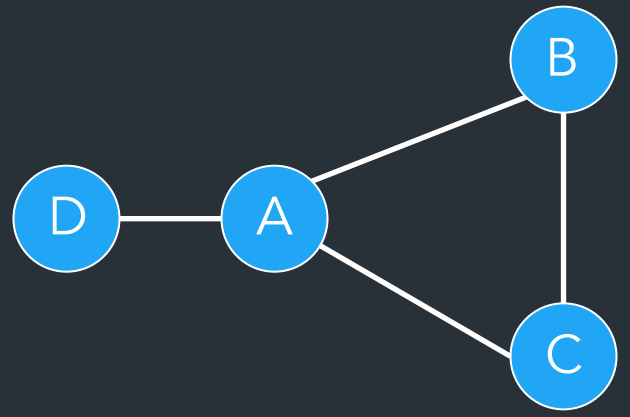
Suppose router R has the following table:

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

Dealing with Failures



- What happens when the D-A link fails?

=> “Count to Infinity” problem

How to avoid loops

- Does IP TTL help?
- 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

One way: Split Horizon

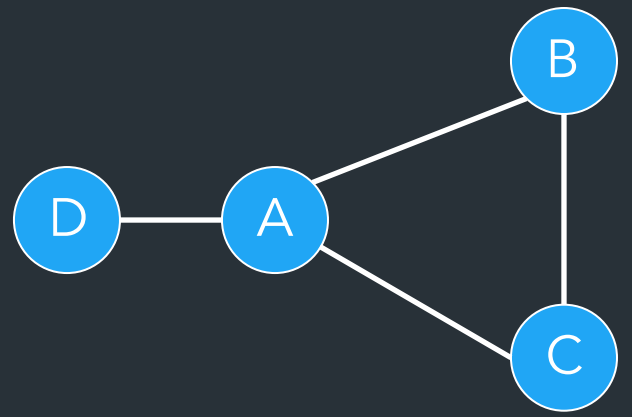
- When sending updates to node A, don’t include routes you learned from A
- Prevents B and C from sending cost 2 to A

Split Horizon + Poison Reverse

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

Distance-vector updates

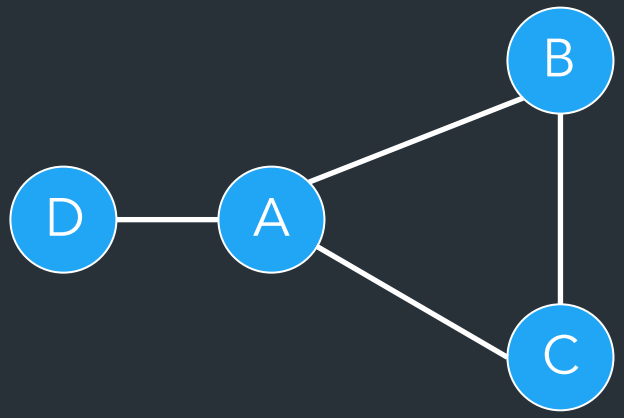
Even with split horizon + poison reverse,
can still create loops with >2 nodes



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



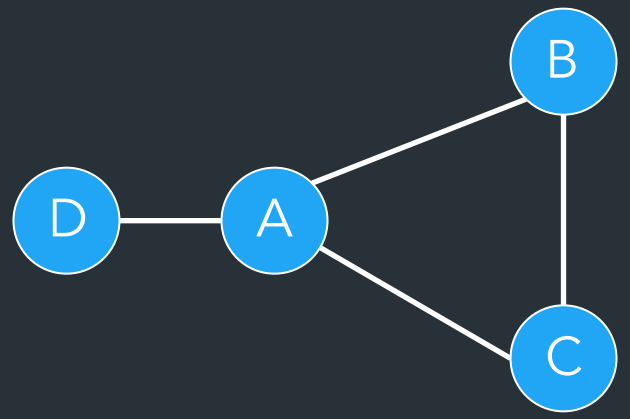
B's routing table

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?

Dealing with failures



- What happens when the D-A link fails?