Routing #1

One of the Fundamental Problems: **Routing**

Plan for today

- Setting the scene
    - A disclaimer
    - Quick statement on addresses
    - What is a router?
    - Why do we have routers? (AKA “Why is a router?”)
    - The Challenge of Routing
    - The Challenge of Forwarding (AKA “Why tables?”)
    - Forwarding vs. Routing
- Theoretical perspective & routing validity
    - Graph representation of routing state
    - Defining routing validity
    - Validating routing state
- An in-class activity

A Disclaimer

- There are an endless number of possible solutions to routing
- I’m going to constrain our initial discussion to how “archetypal Internet” works
  - Lots of assumptions based on this!
  - Planning to discuss some alternatives next week

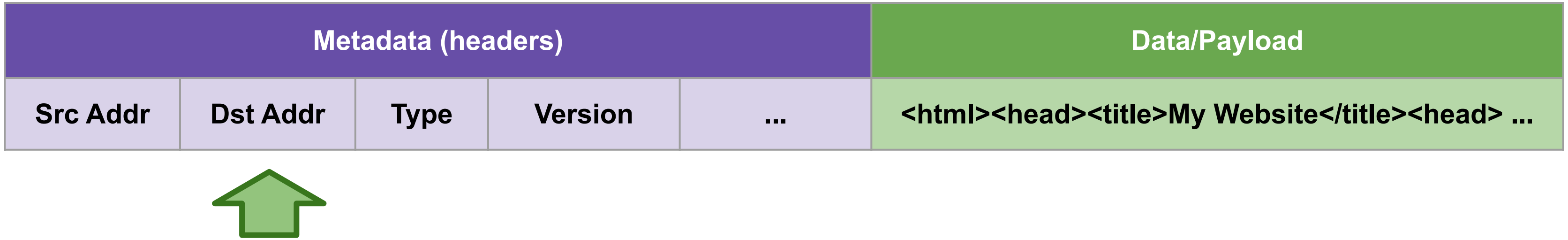
Recall from Lecture 2: Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain...

Metadata (headers)					Data/Payload
Src Addr	Dst Addr	Type	Version	...	<html><head><title>My Website</title><head> ...

Recall from Lecture 2: Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain ***destination address***



Metadata (headers)					Data/Payload
Src Addr	Dst Addr	Type	Version	...	<html><head><title>My Website</title><head> ...

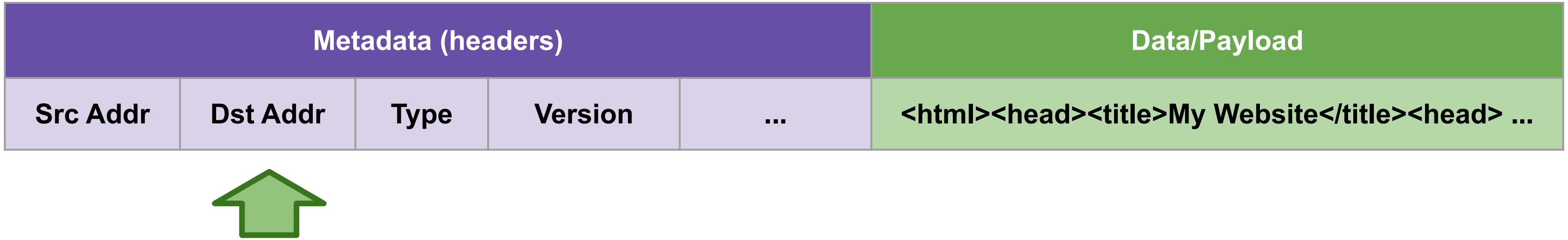
Recall from Lecture 2: Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain ***destination address***
      - .. implies that ***a host has an address***!
        - Or more than one! (Why?!)

Metadata (headers)					Data/Payload
Src Addr	**Dst Addr**	**Type**	**Version**	**...**	**<html><head><title>My Website</title><head> ...**

Recall from Lecture 2: Packets

- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain ***destination address***
      - .. implies that ***a host has an address***!
        - Or more than one! (Why?!)
        - .. but for now, we'll think of it as one address per host



Metadata (headers)					Data/Payload
Src Addr	Dst Addr	Type	Version	...	<html><head><title>My Website</title><head> ...

Recall from Lecture 2: Packets

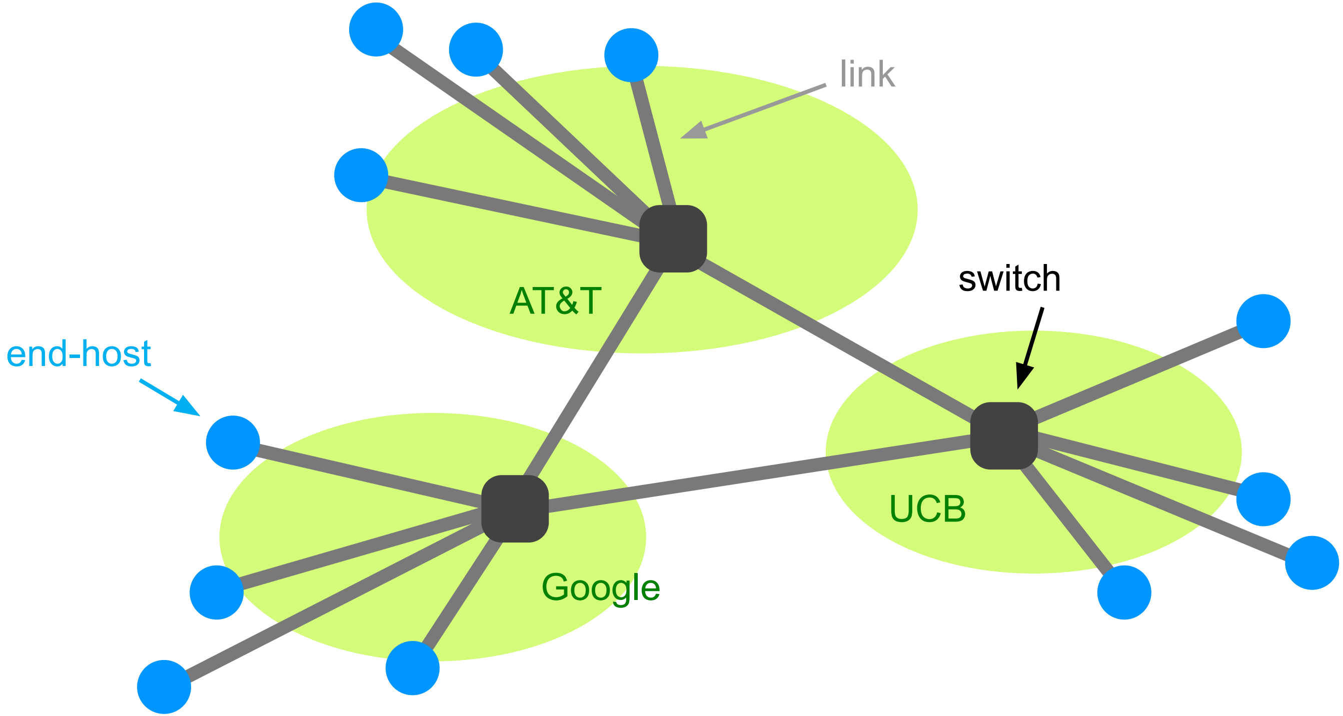
- Packet has…
  - Payload (the actual data)
  - Headers (metadata)
    - Must* contain ***destination address***
      - .. implies that ***a host has an address***!
        - Or more than one! (Why?!)
        - .. but for now, we'll think of it as one address per host

This host's address is "F"

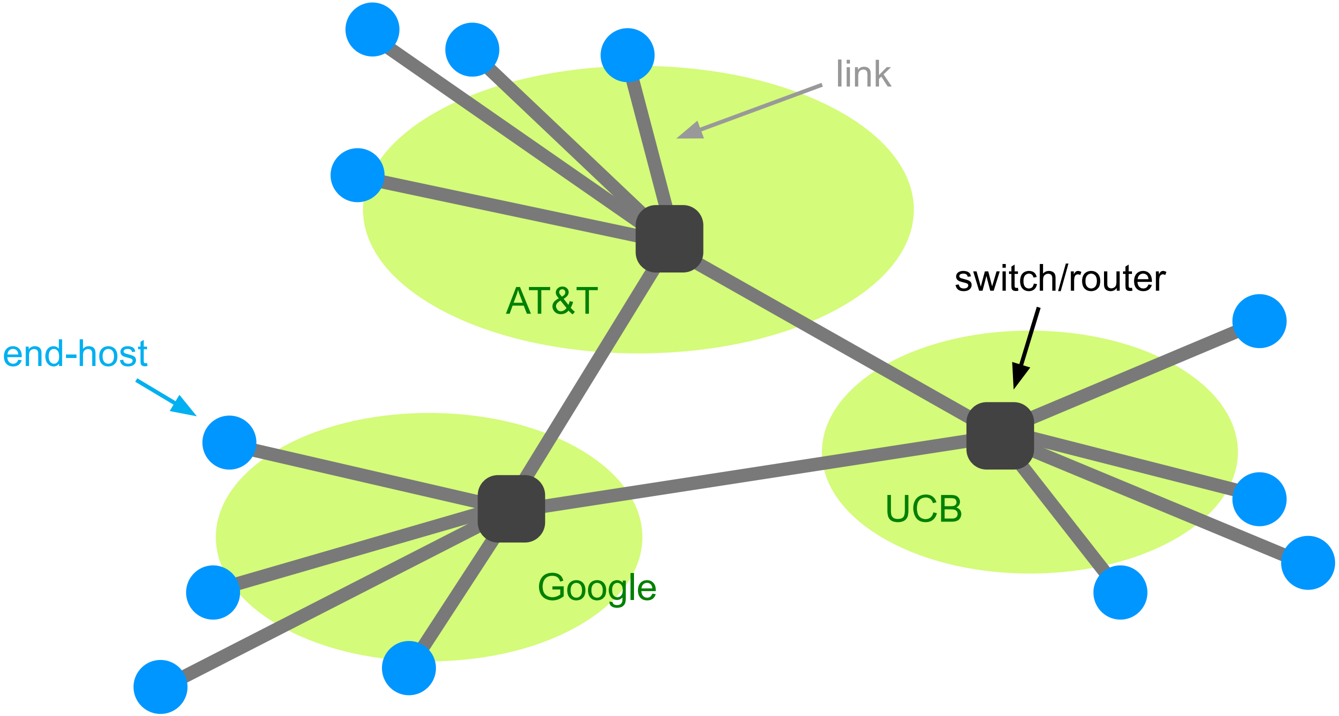
Metadata (headers)					Data/Payload
Src Addr	**Dst Addr**	**Type**	**Version**	**...**	**<html><head><title>My Website</title><head> ...**

What is a router?

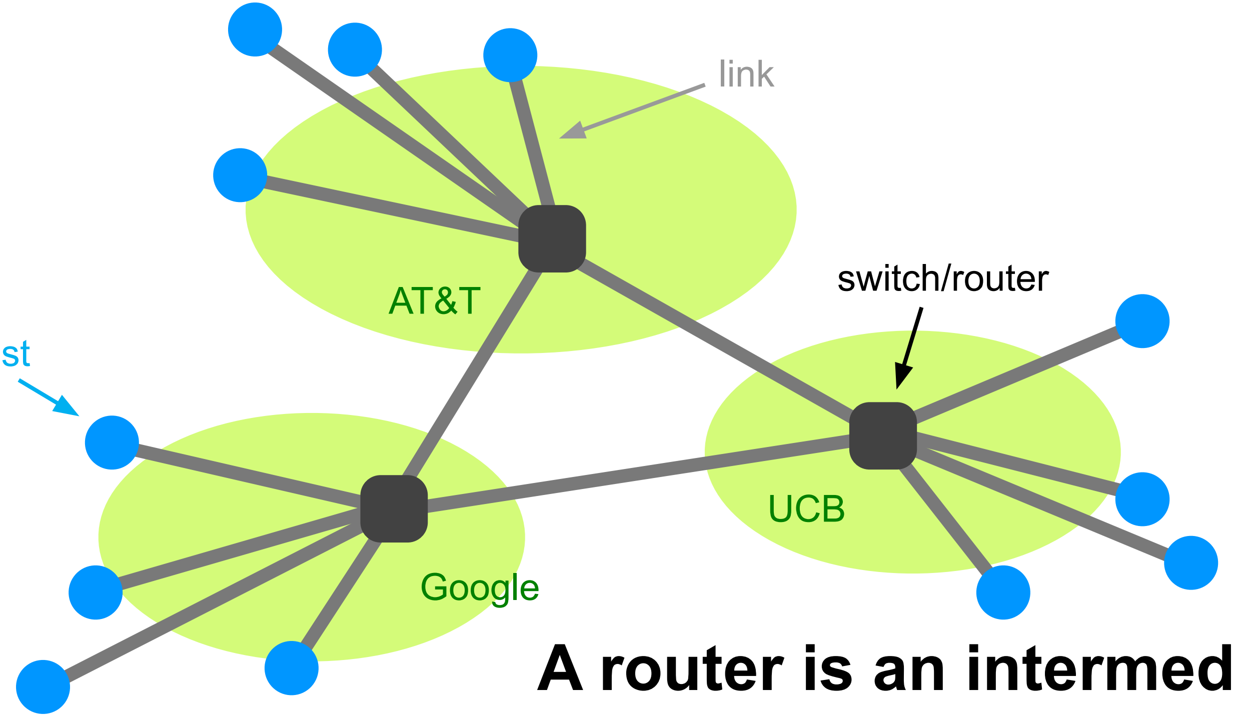
Recall from Lecture 1...



Recall from Lecture 1...



Recall from Lecture 1...



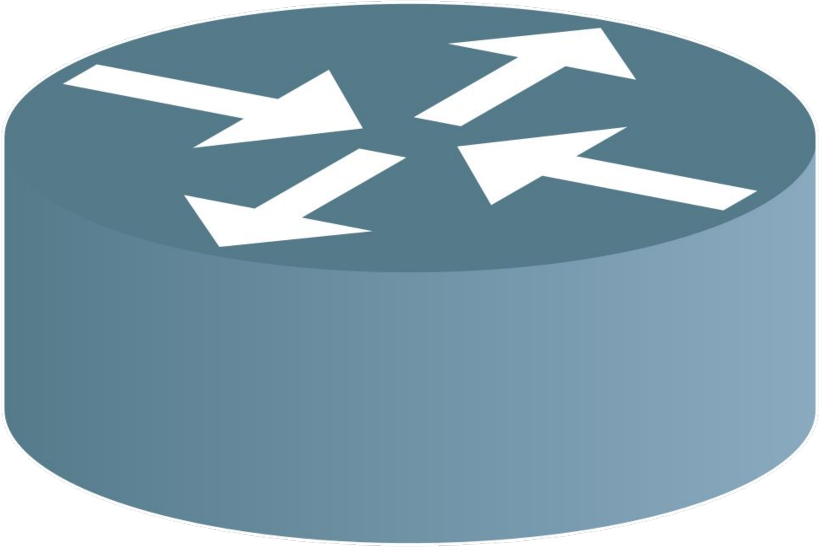
A router is an intermediate node that is usually connected to multiple neighbors

What is a router?

In this class, often:

□ or ○

What is a router?



What is a router?

What is a router?

What is a router?

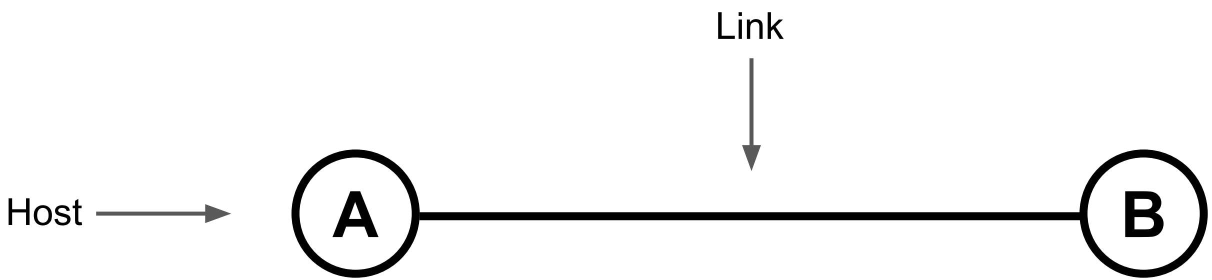
What is a router?

Taller than me

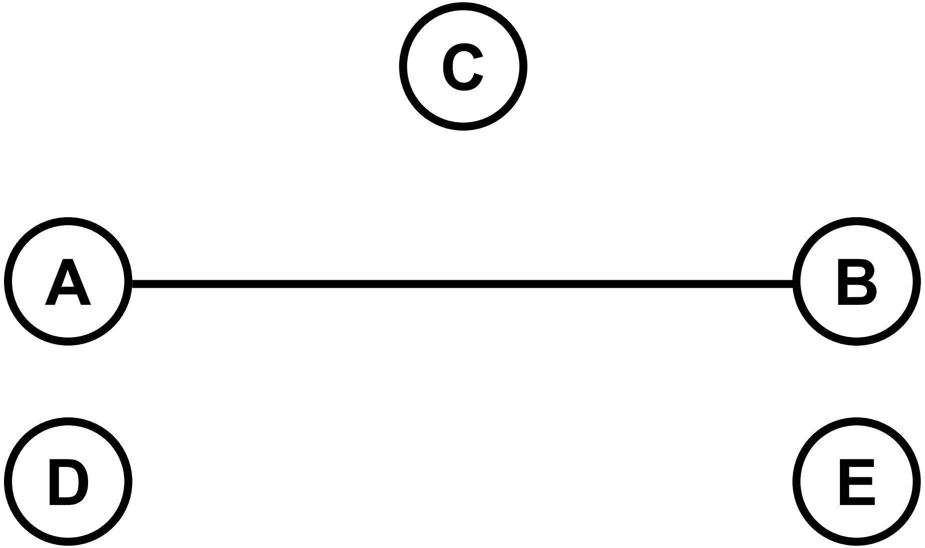
Why do we have routers?

Why is a router?

Why is a router?

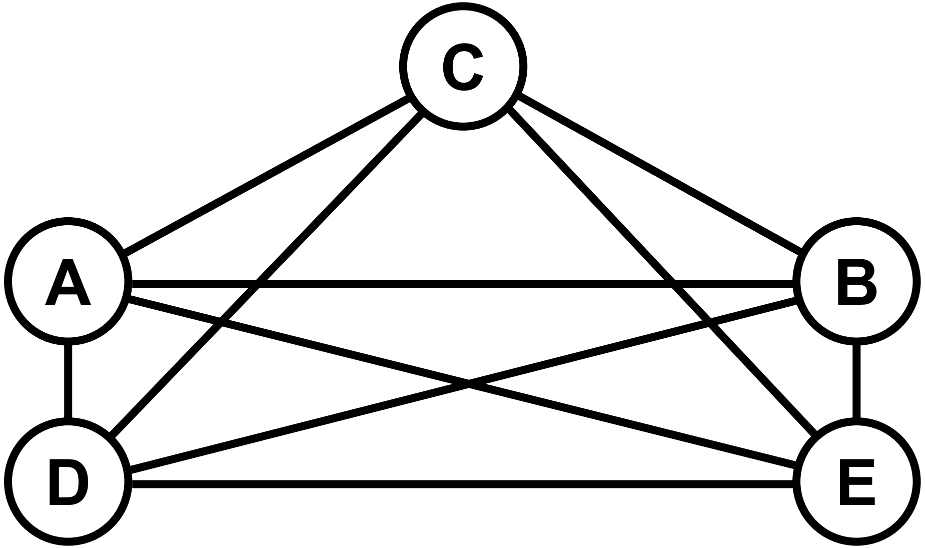


Why is a router?



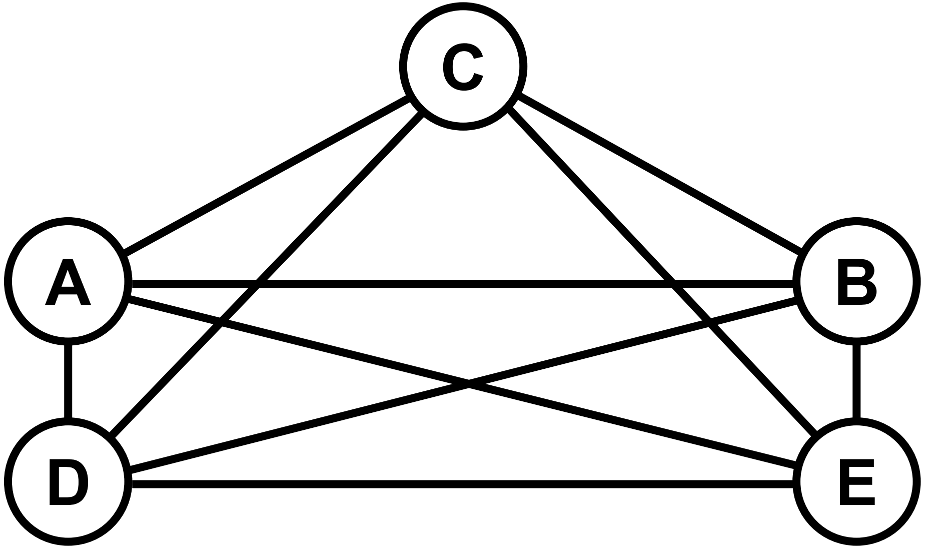
Now what?

Why is a router?



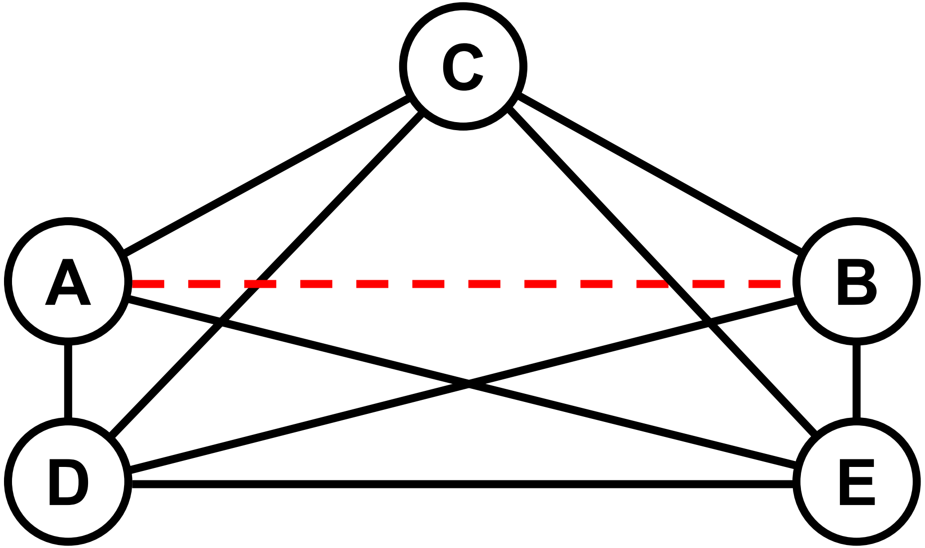
Is there a problem with this?

Why is a router?



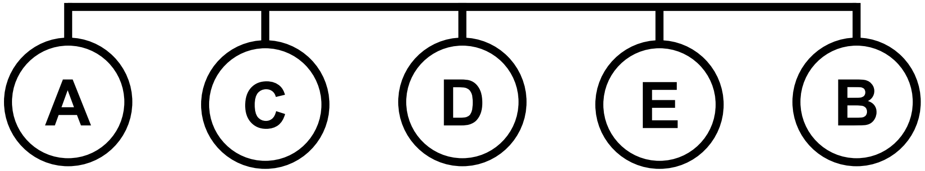
Are there good things about this?

Why is a router?

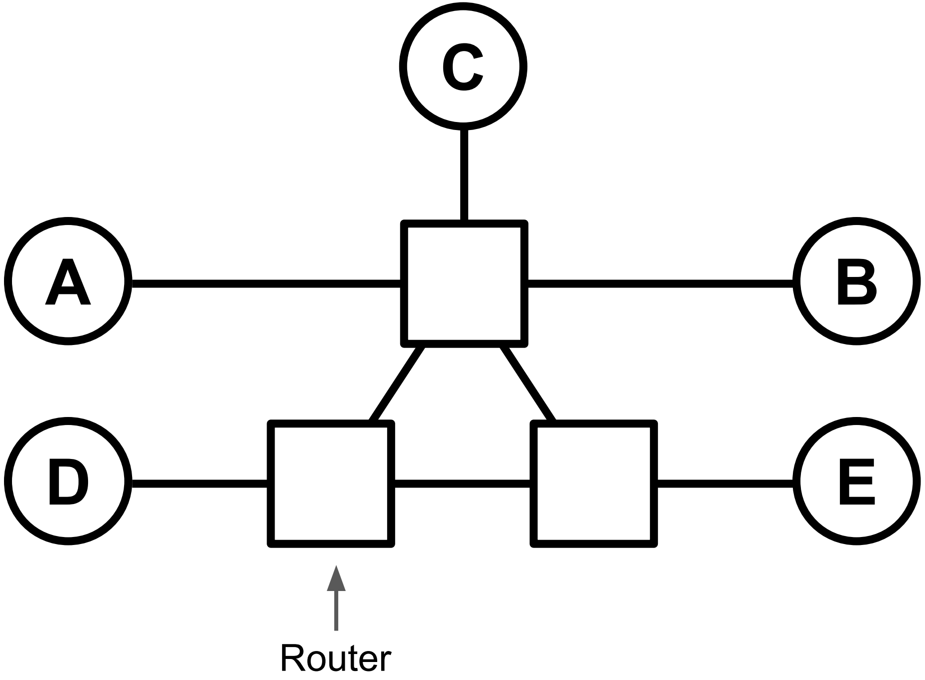


Are there good things about this?

Why is a router?

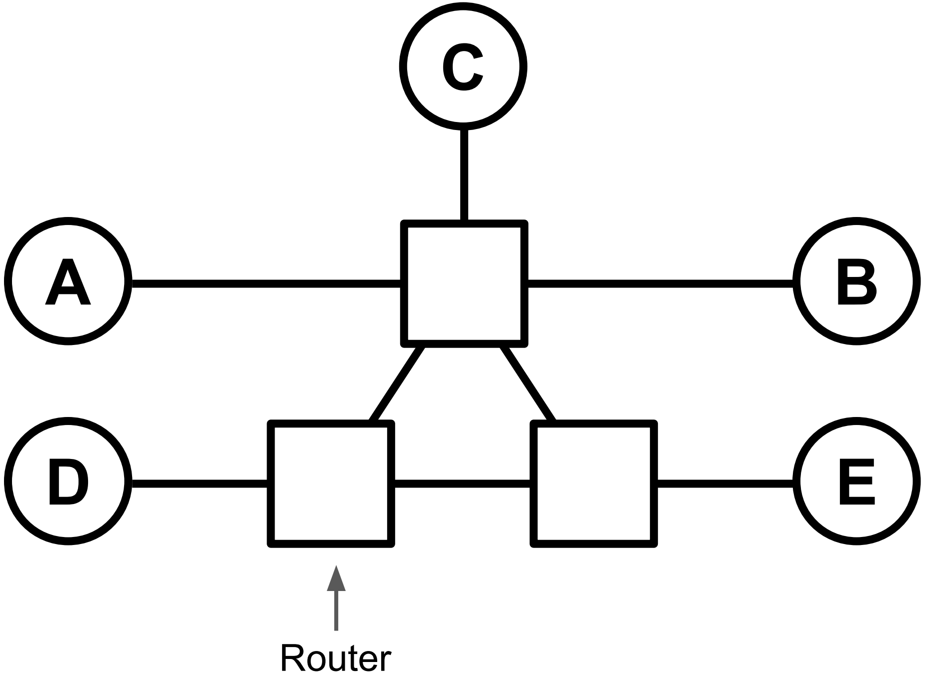


Why is a router?



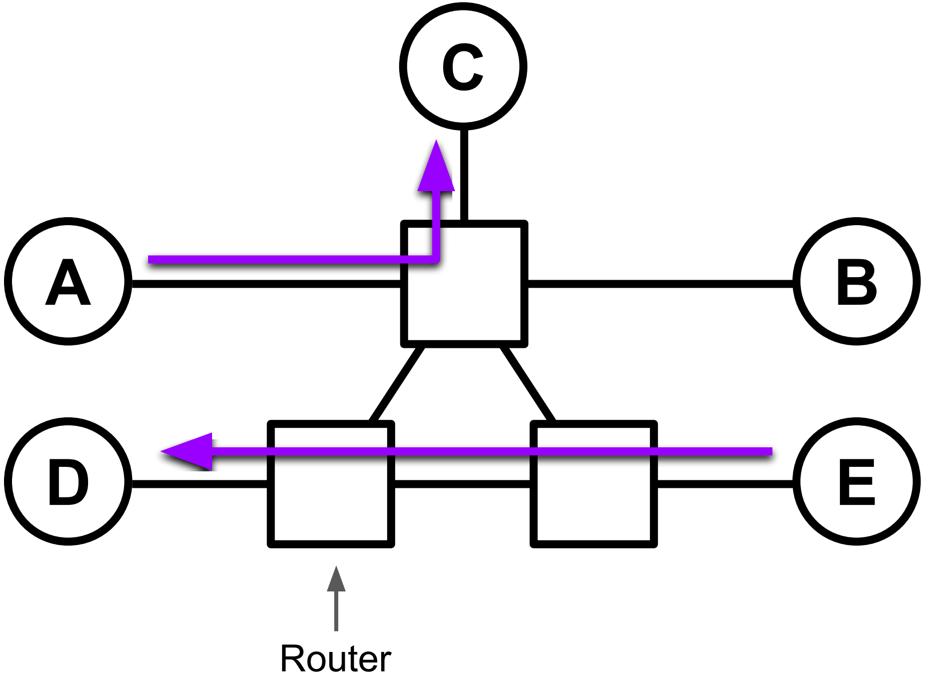
Why is a router?

- Way fewer links than a full mesh!



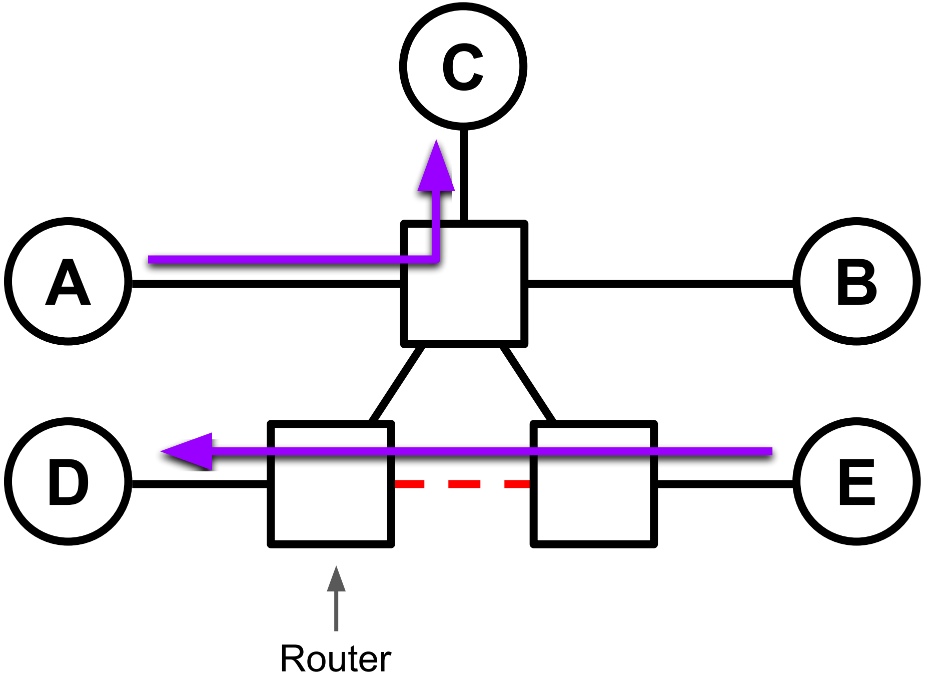
Why is a router?

- Way fewer links than a full mesh!
- But more than just a single link!



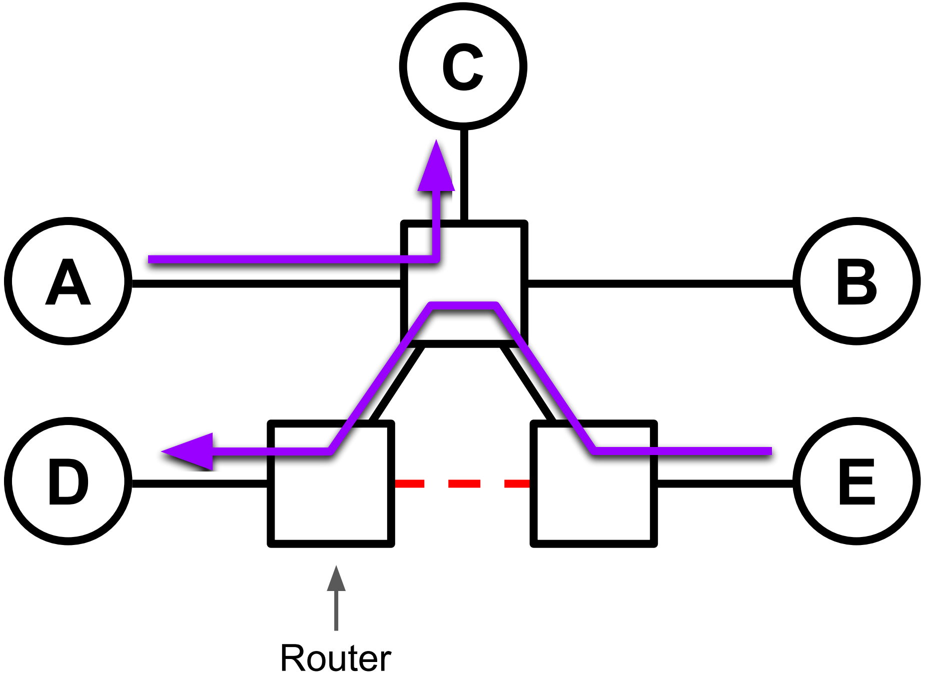
Why is a router?

- Way fewer links than a full mesh!
- But more than just a single link!



Why is a router?

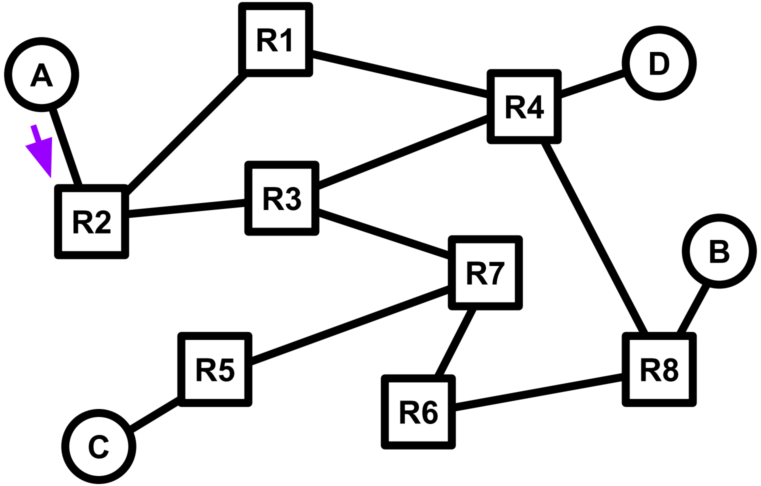
- Way fewer links than a full mesh!
- But more than just a single link!
- Alternate paths!



The Challenge of Routing

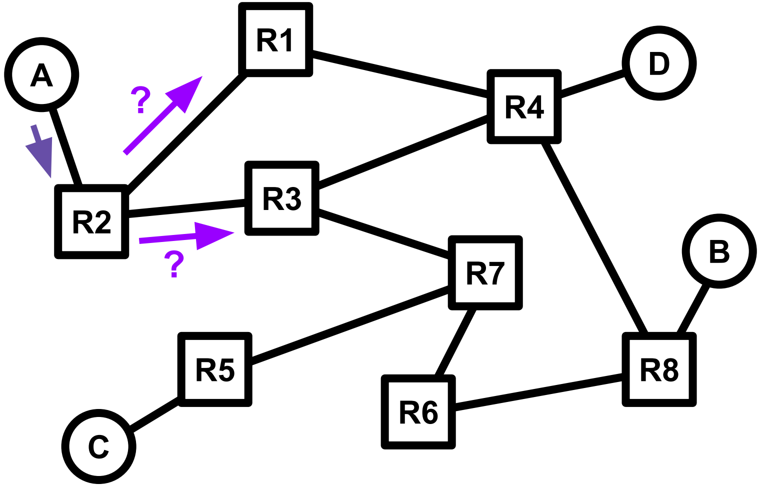
The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?



The Challenge of Routing

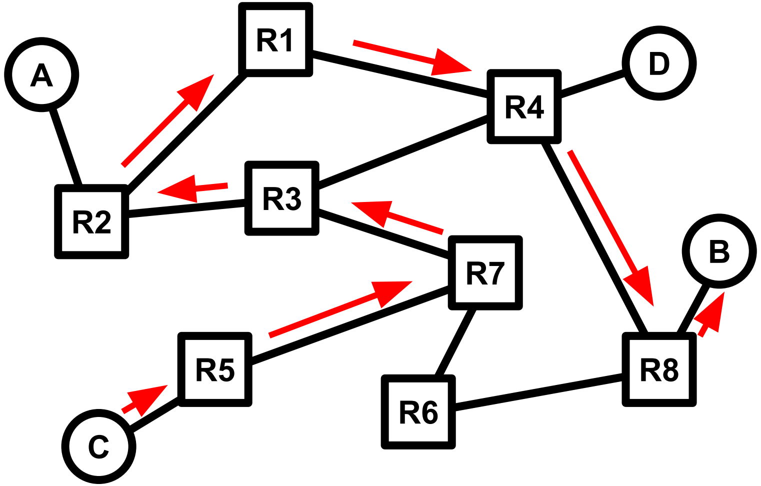
- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?



The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?

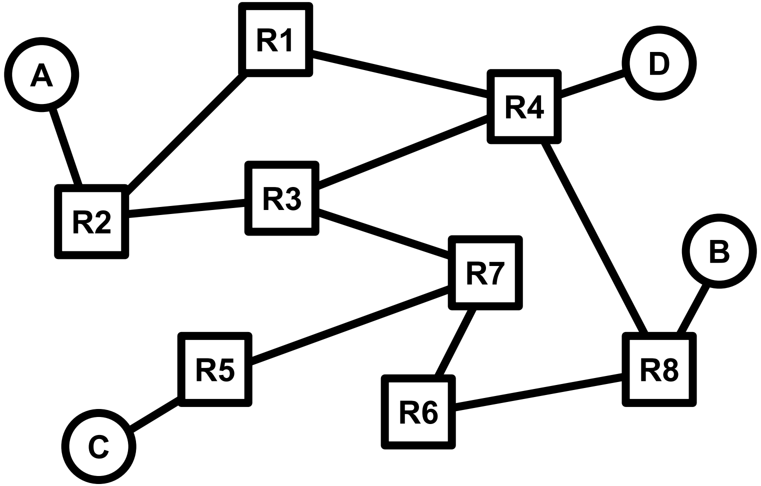
- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)



The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?

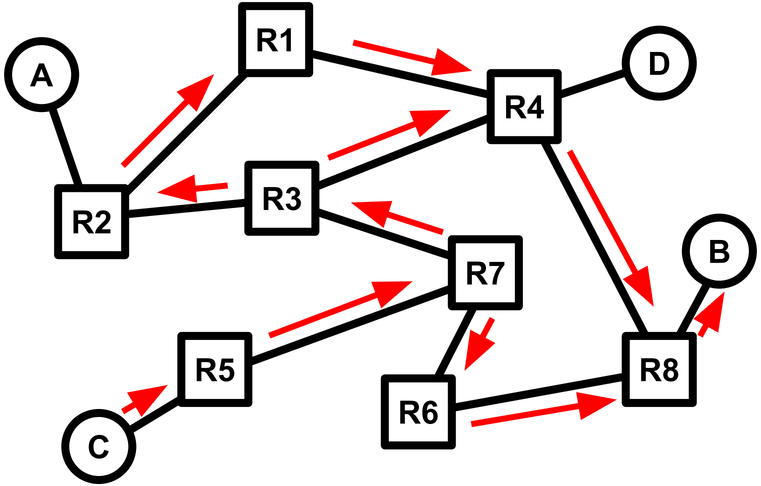
- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)
  - No random routing



The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?

- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)
  - No random routing
  - No just sending it to everyone (though we’ll come back to this in a future lecture!)

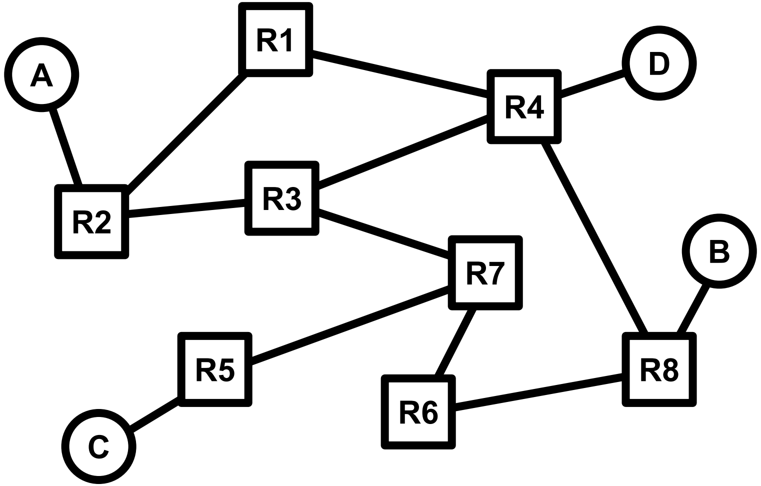


The Challenge of Routing

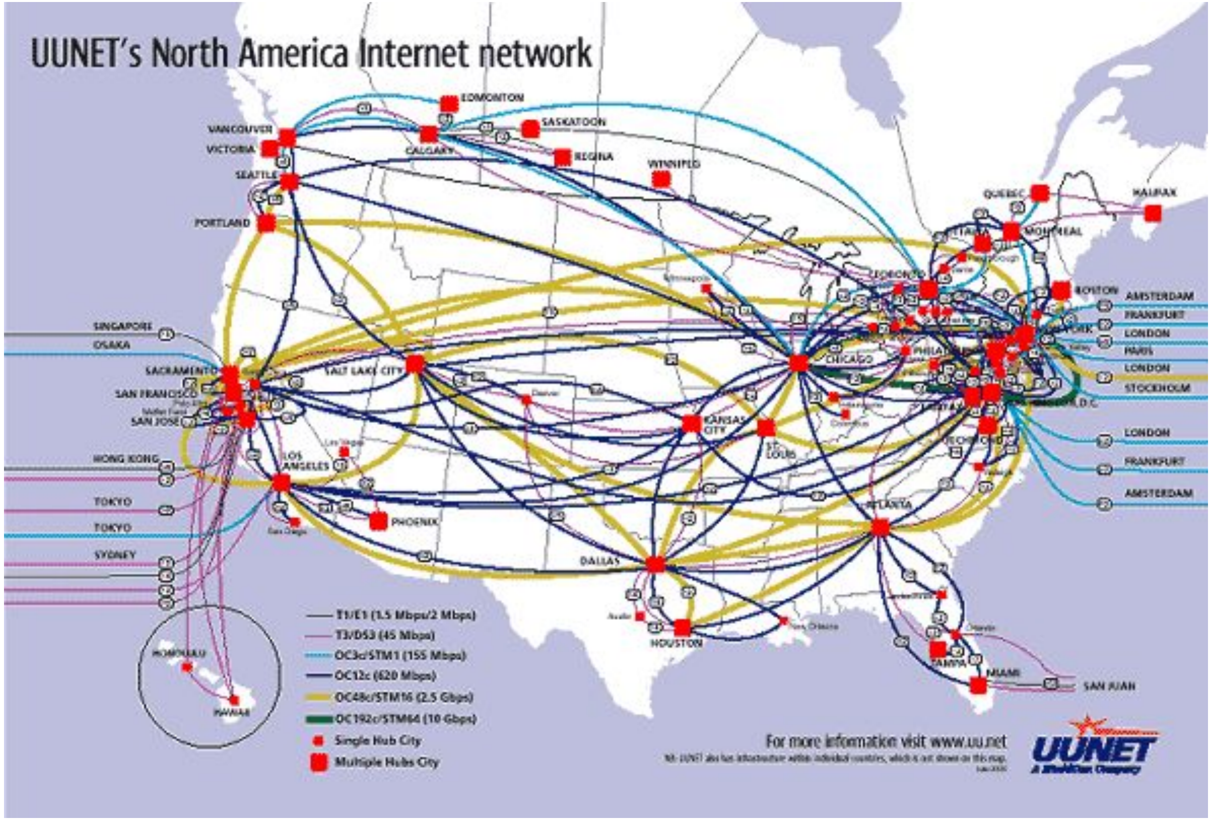
- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?

- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)
  - No random routing
  - No just sending it to everyone (though we’ll come back to this in a future lecture!)

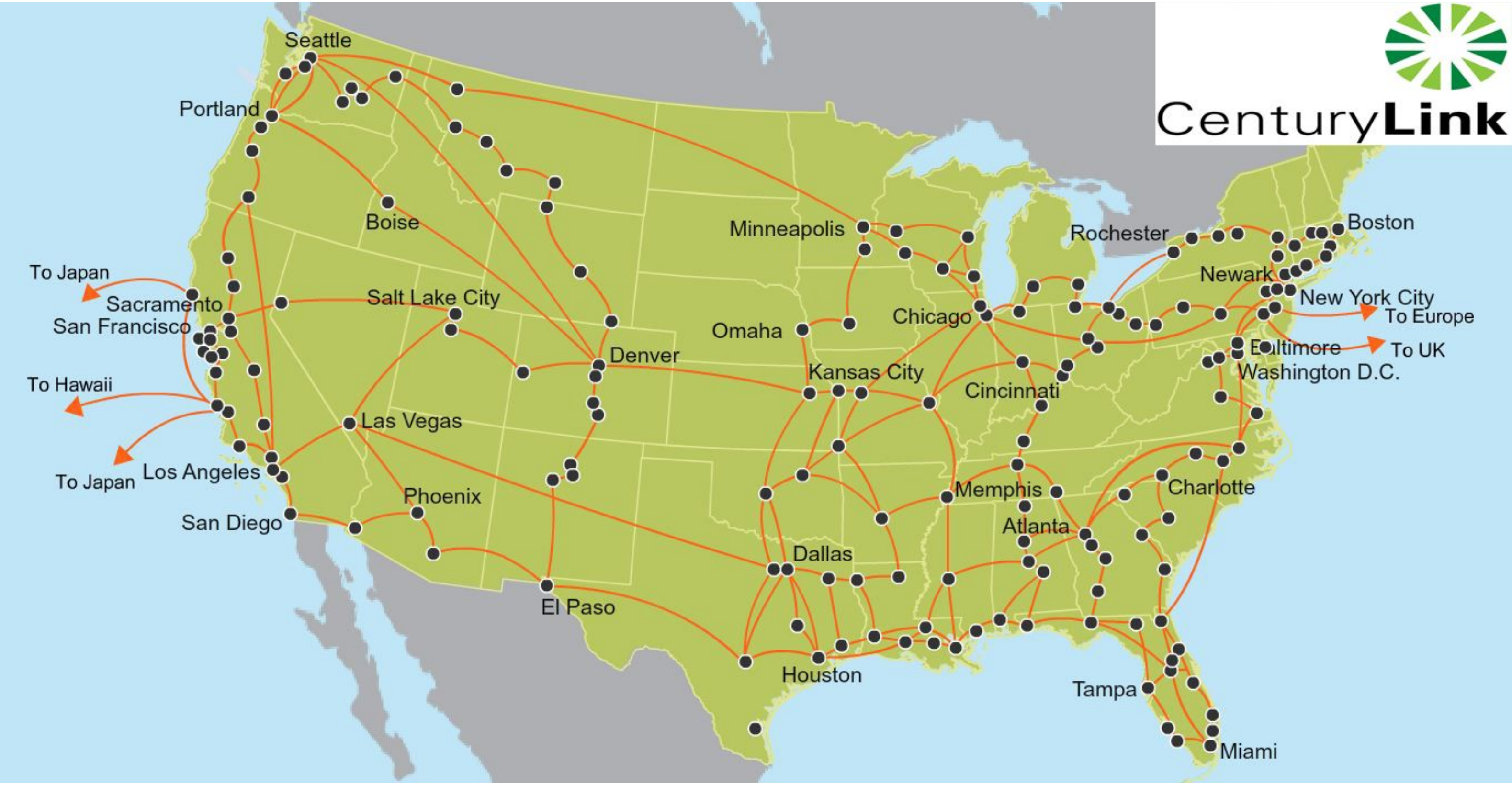
- We want it to adapt to arbitrary topologies
  - The “graph” describing a network can vary a lot!



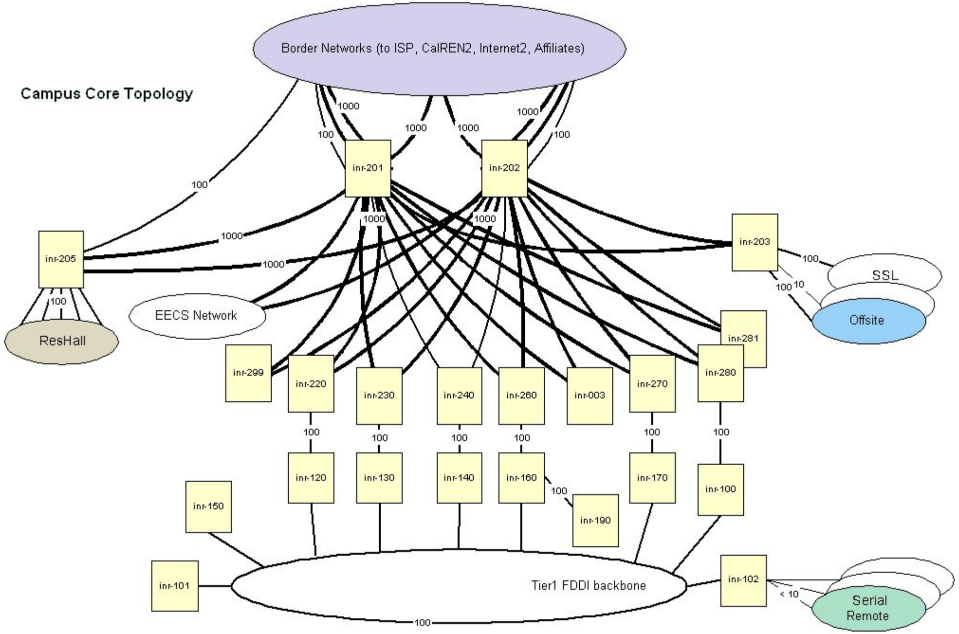
UUNET North American Network



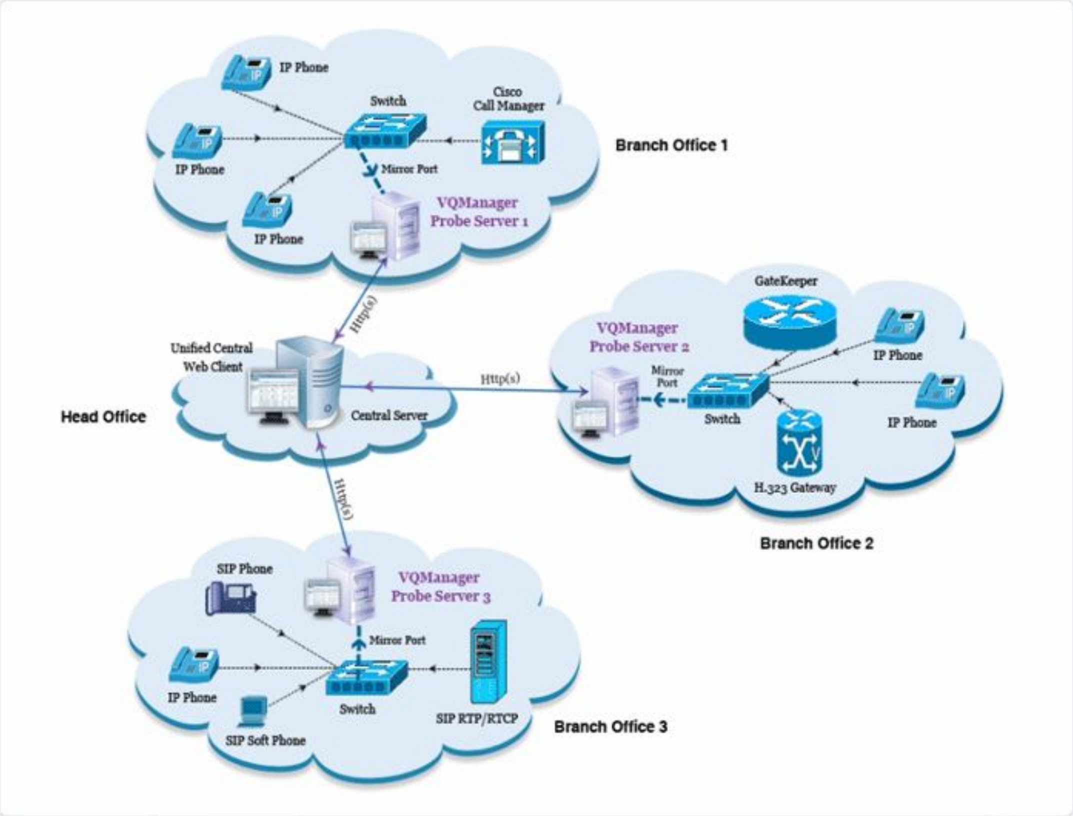
CenturyLink Domestic Network



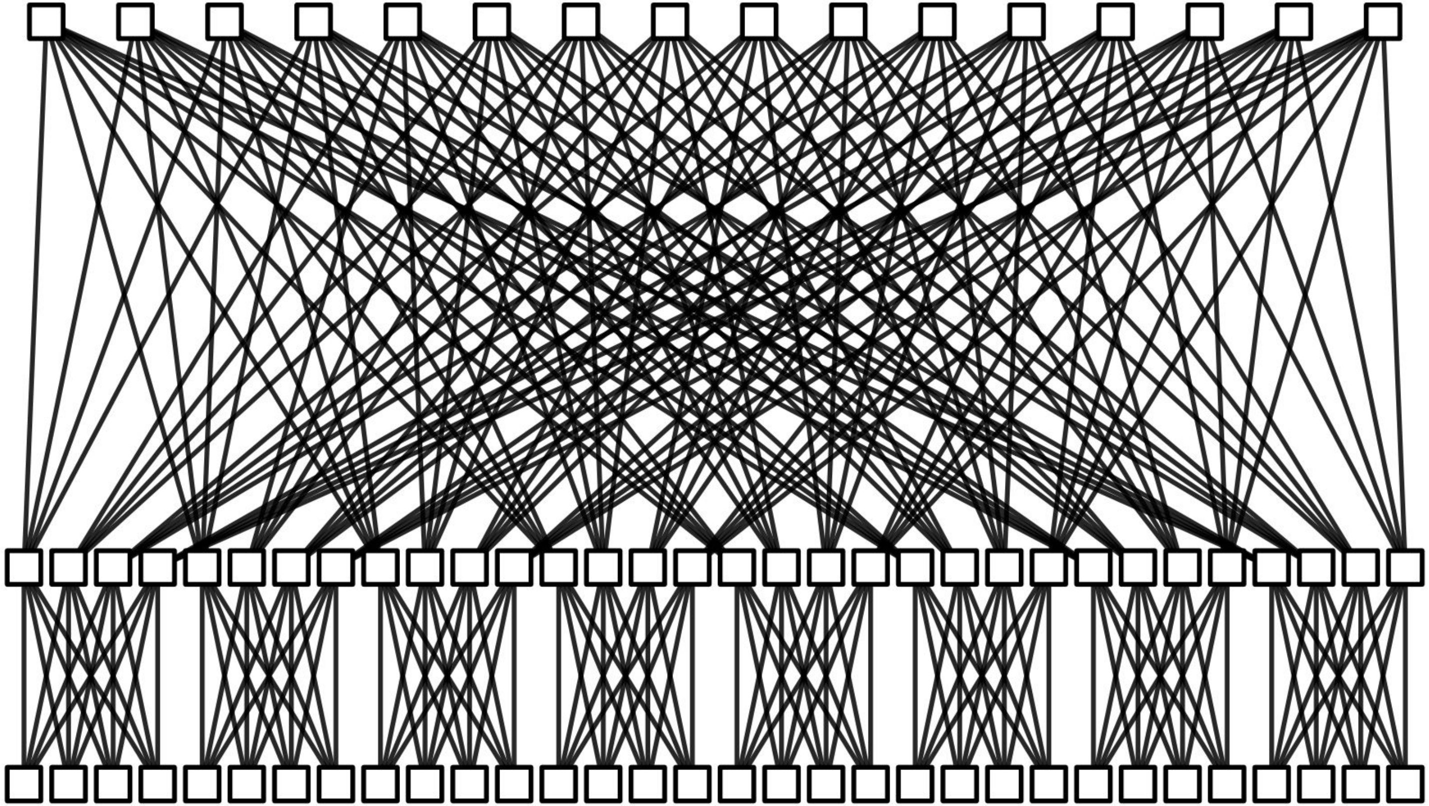
Berkeley Campus Network



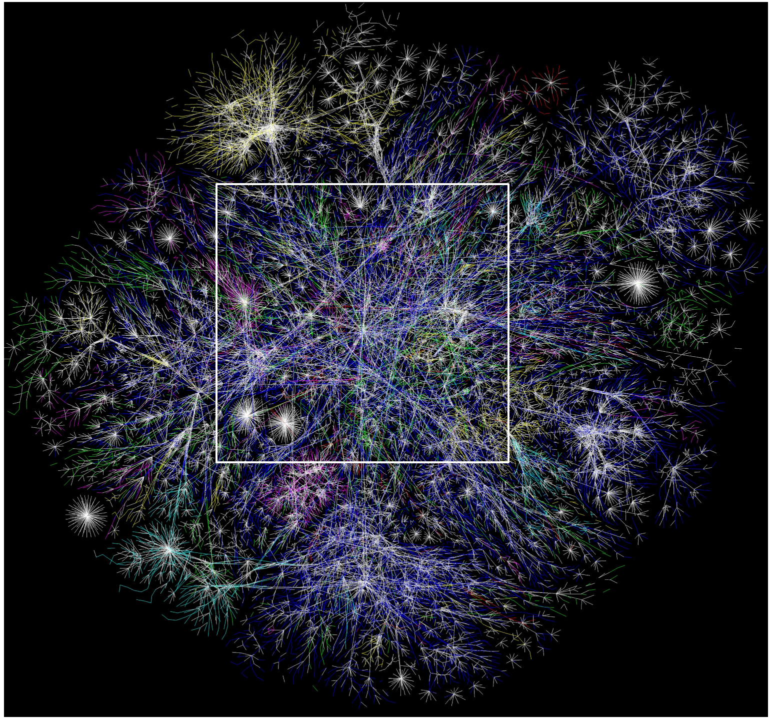
Enterprise Network



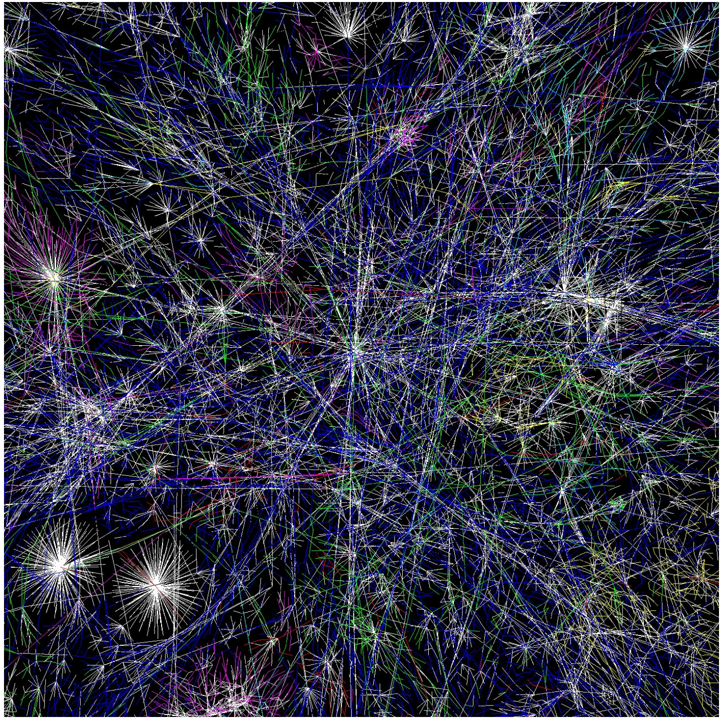
Partial Data Center Topology



The Whole* Internet



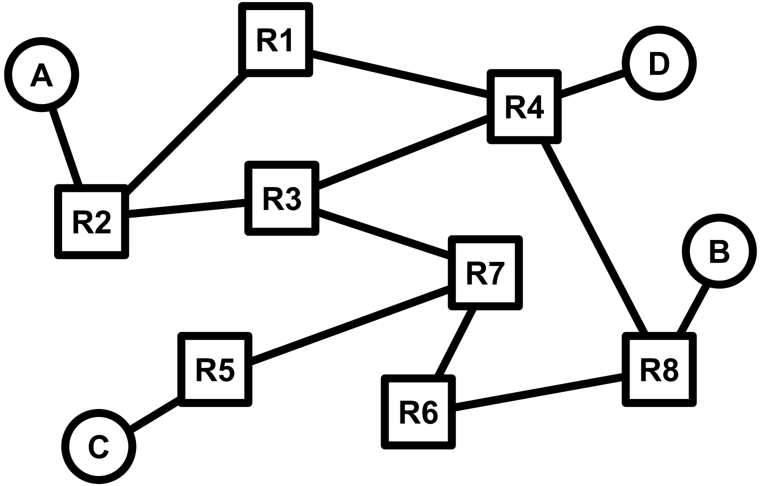
Internet (2005)



Zoomed in for detail

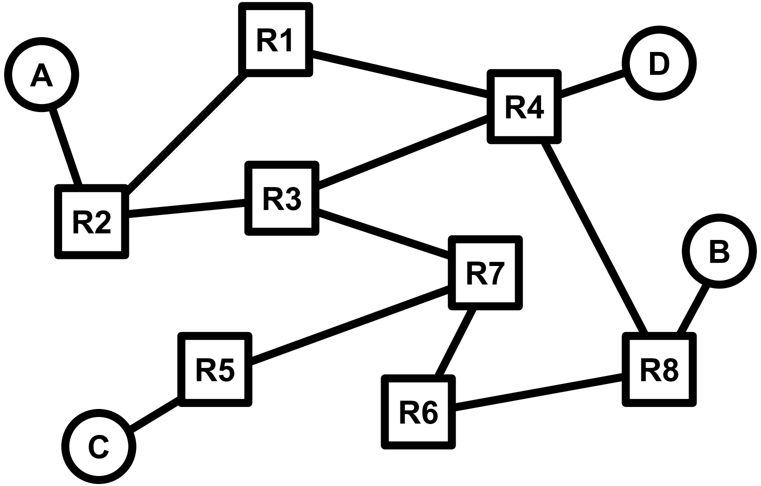
The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?
- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)
  - No random routing
  - No just sending it to everyone (though we’ll come back to this in a future lecture!)
- We want it to adapt to arbitrary topologies
  - The “graph” describing a network can vary a lot!
  - Different networks (parts of the Internet) *may* use different routing, but generality is good
  - Especially since *every* topology is *dynamic* (Why?)



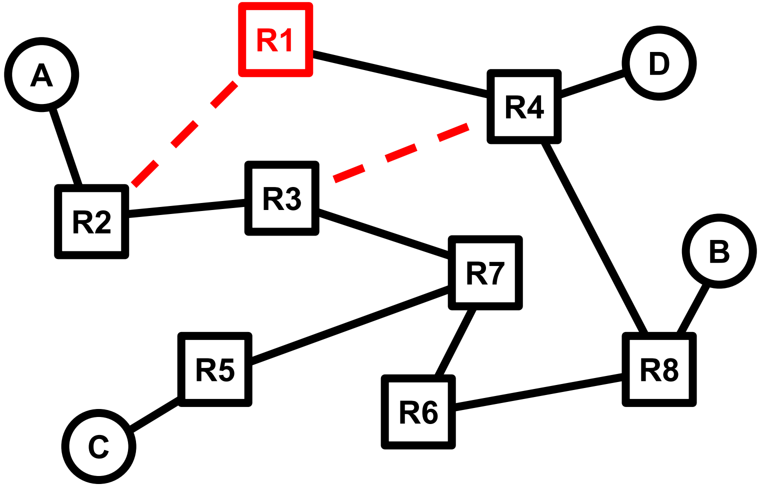
The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?
- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)
  - No random routing
  - No just sending it to everyone (though we’ll come back to this in a future lecture!)
- We want it to adapt to arbitrary topologies
  - The “graph” describing a network can vary a lot!
  - Different networks (parts of the Internet) *may* use different routing, but generality is good
  - Especially since *every* topology is *dynamic* (Why?)
    - May add more links, customers, equipment…



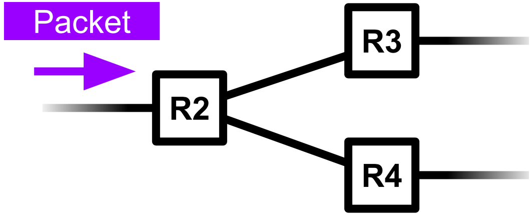
The Challenge of Routing

- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?
- We want to find ***paths*** which are *“**good**”*
  - “Good” may have many meanings (e.g., short)
  - No random routing
  - No just sending it to everyone (though we’ll come back to this in a future lecture!)
- We want it to adapt to arbitrary topologies
  - The “graph” describing a network can vary a lot!
  - Different networks (parts of the Internet) *may* use different routing, but generality is good
  - Especially since *every* topology is *dynamic* (Why?)
    - May add more links, customers, equipment…
    - *Definitely* need to deal with *failures*

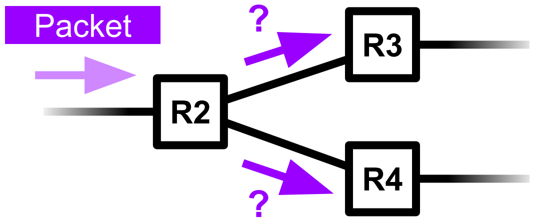


The Challenge of Forwarding

- When packet arrives...

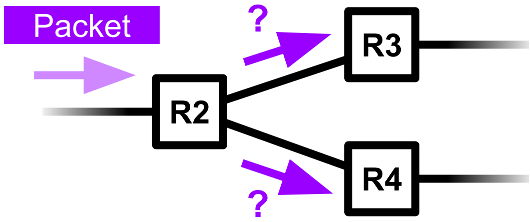


The Challenge of Forwarding



- When packet arrives, router ***forwards*** it to one of its neighbors

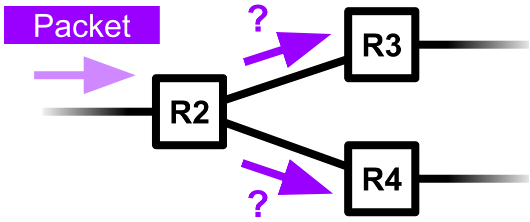
The Challenge of Forwarding



- When packet arrives, router ***forwards*** it to one of its neighbors

- You want to make the decision about which neighbor *fast*
- Implies the decision process is *simple*

The Challenge of Forwarding

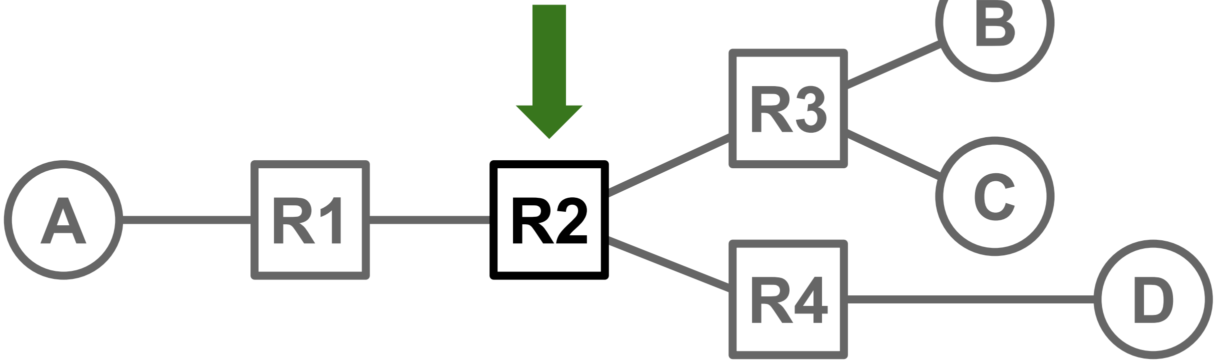


- When packet arrives, router ***forwards*** it to one of its neighbors

- You want to make the decision about which neighbor *fast*
- Implies the decision process is *simple*

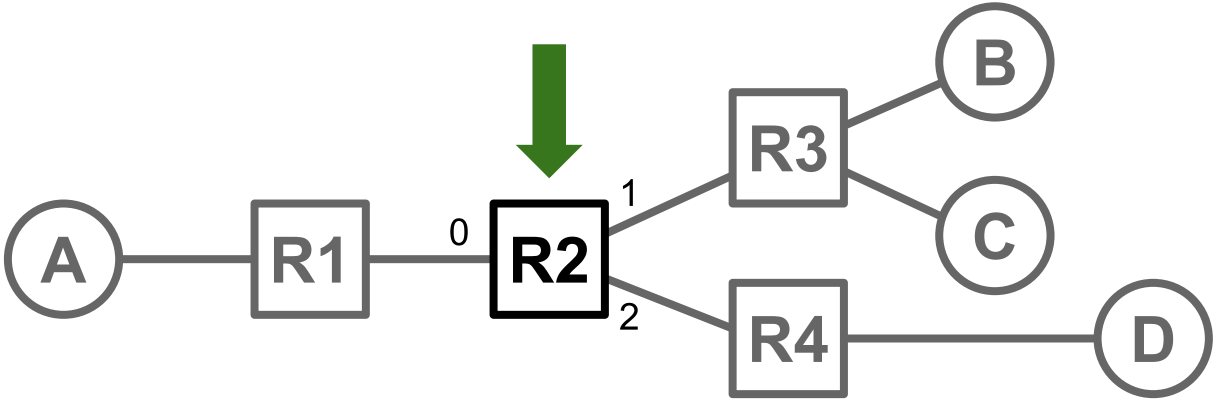
- Solution: Use a table

Forwarding with a table



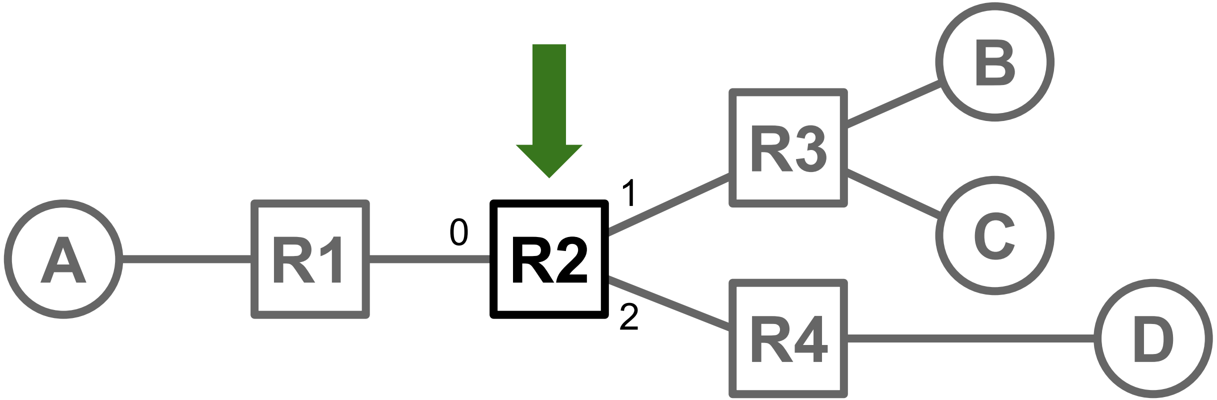
R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

Forwarding with a table



R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

Forwarding with a table



R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

.. or ..

R2's Table	
Dst	***Port***
A	0
B	1
C	1
D	2

Forwarding with a table

- Given the tables, decision *depends only on destination field of packet*
- .. we are doing what’s called ***destination-based forwarding/routing***
  - Very common
  - One of those “archetypal Internet” things I mentioned earlier
  - We’ll think about some alternatives later

R2’s Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

.. or ..

R2’s Table	
Dst	***Port***
A	0
B	1
C	1
D	2

Two Things Routers Do

Forwarding

Routing

Two Things Routers Do

Forwarding

- Looks up packet’s destination in table and sends packet to given neighbor

Routing

Two Things Routers Do

Forwarding

- Looks up packet’s destination in table and sends packet to given neighbor

Routing

- Communicates with other routers to determine how to populate tables for forwarding

Two Things Routers Do

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor

- *Inherently local:* depends only on arriving packet and local table

Routing

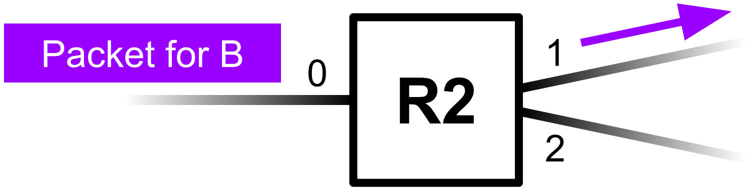
- Communicates with other routers to determine how to populate tables for forwarding

Two Things Routers Do

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor

- *Inherently local:* depends only on arriving packet and local table



Routing

- Communicates with other routers to determine how to populate tables for forwarding

R2's Table

Dst	***Port***
A	0
B	1
C	1
D	2

Two Things Routers Do

Forwarding

- Looks up packet’s destination in table and sends packet to given neighbor

- *Inherently local:* depends only on arriving packet and local table

Routing

- Communicates with other routers to determine how to populate tables for forwarding

- *Inherently global:* must know about *all* destinations, not just local ones

Two Things Routers Do

Forwarding

- Looks up packet’s destination in table and sends packet to given neighbor

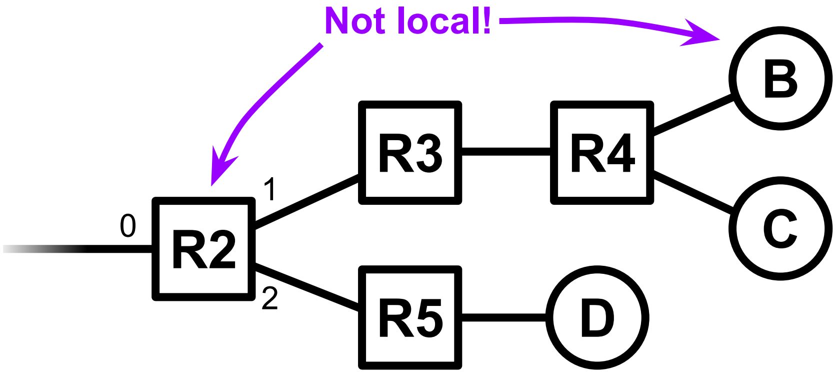
- *Inherently local:* depends only on arriving packet and local table

R2’s Table	
Dst	***Port***
A	0
B	1
C	1
D	2

Routing

- Communicates with other routers to determine how to populate tables for forwarding

- *Inherently global:* must know about *all* destinations, not just local ones


Not local!

Two Things Routers Do

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor
- *Inherently local:* depends only on arriving packet and local table
- Primary responsibility of router's *data plane*

Routing

- Communicates with other routers to determine how to populate tables for forwarding
- *Inherently global:* must know about *all* destinations, not just local ones

Two Things Routers Do

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor
- *Inherently local:* depends only on arriving packet and local table
- Primary responsibility of router's *data plane*

Routing

- Communicates with other routers to determine how to populate tables for forwarding
- *Inherently global:* must know about *all* destinations, not just local ones
- Primary responsibility of router's *control plane*

Two Things Routers Do

Forwarding

- Looks up packet’s destination in table and sends packet to given neighbor
- *Inherently local:* depends only on arriving packet and local table
- Primary responsibility of router’s *data plane*
- Time scale: per packet arrival (nanoseconds?)

Routing

- Communicates with other routers to determine how to populate tables for forwarding
- *Inherently global:* must know about *all* destinations, not just local ones
- Primary responsibility of router’s *control plane*

Two Things Routers Do

Forwarding

- Looks up packet’s destination in table and sends packet to given neighbor
- *Inherently local:* depends only on arriving packet and local table
- Primary responsibility of router’s *data plane*
- Time scale: per packet arrival (nanoseconds?)

Routing

- Communicates with other routers to determine how to populate tables for forwarding
- *Inherently global:* must know about *all* destinations, not just local ones
- Primary responsibility of router’s *control plane*
- Time scale: per network event (e.g. per failure)

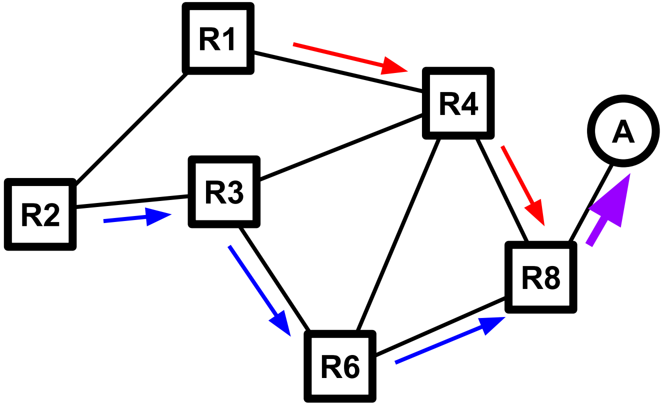
Getting a little theoretical

Graph representation and validity of routing state

“Delivery trees” in destination-based routing

- We can graph paths packets ***to a destination*** will take if they follow tables

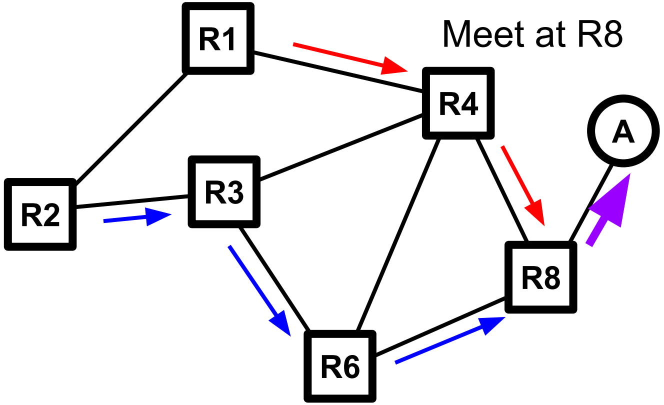
- NextHop becomes an arrow

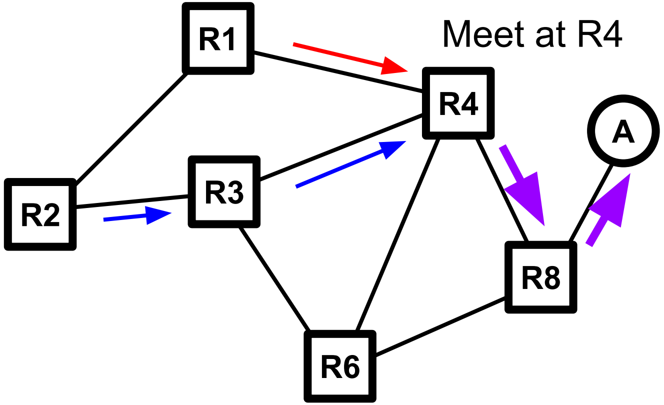


R6’s Table	
Dst	***NextHop***
A	R8
...	

“Delivery trees” in destination-based routing

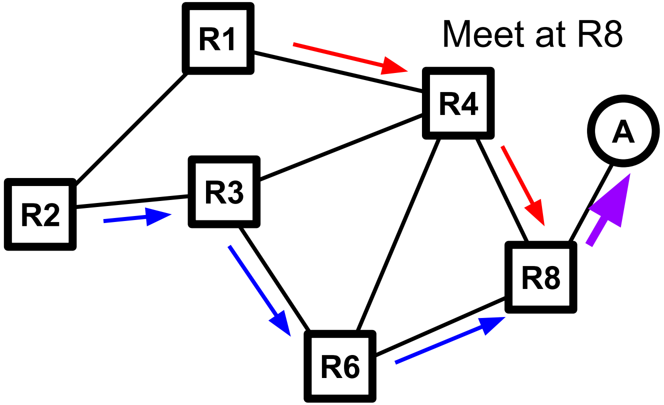
- We can graph paths packets ***to a destination*** will take if they follow tables
- NextHop becomes an arrow
  - Only one NextHop per destination…
  - .. means *only one outgoing arrow per node*!
  - .. once paths “meet”, they never split

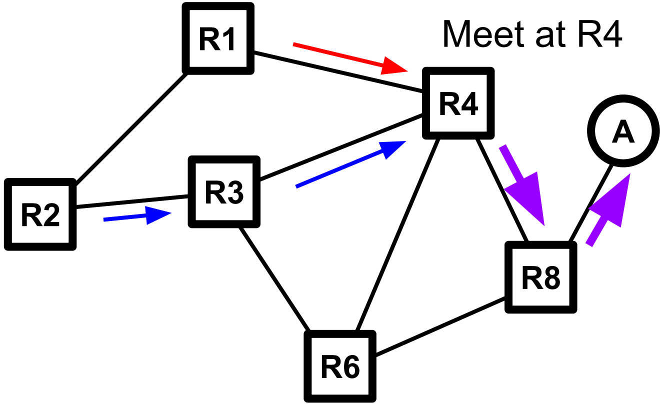




“Delivery trees” in destination-based routing

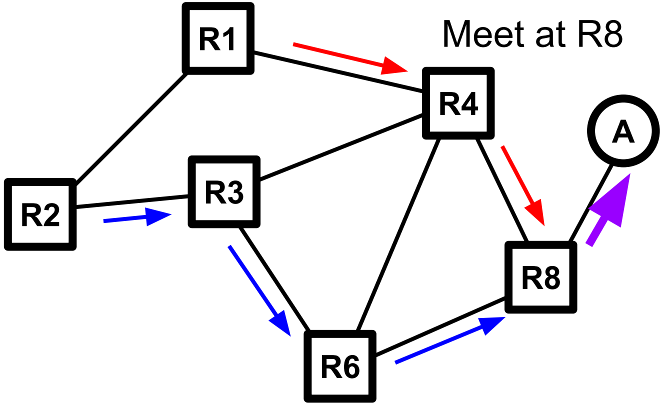
- We can graph paths packets ***to a destination*** will take if they follow tables
- NextHop becomes an arrow
  - Only one NextHop per destination…
  - .. means *only one outgoing arrow per node*!
  - .. once paths “meet”, they never split
- Set of all paths create ***“directed delivery tree”***
  - *Must cover every node* (We want to be able to reach it from anywhere!)

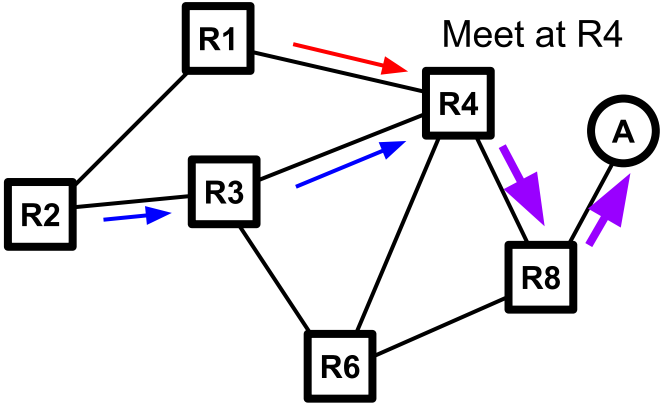




“Delivery trees” in destination-based routing

- We can graph paths packets ***to a destination*** will take if they follow tables
- NextHop becomes an arrow
  - Only one NextHop per destination…
  - .. means *only one outgoing arrow per node*!
  - .. once paths “meet”, they never split
- Set of all paths create ***“directed delivery tree”***
  - *Must cover every node* (We want to be able to reach it from anywhere!)
- It’s an ***oriented spanning tree*** rooted at the destination
  - Spanning tree: a tree that touches every node





Routing State Validity

- Earlier, said we wanted “good” paths between hosts
- Notion of goodness is flexible, but…
- Minimum requirement *must* be that *packets actually reach their destinations*

- It’d be useful to be able to reason about this!
- This is articulated by Scott Shenker as ***routing state validity***
    - (We use this term here at Berkeley, but it’s not standard routing terminology)

Routing State Validity

- **Local** routing state is table in a single router
  - By itself, the state in a single router can’t be evaluated for validity
  - It must be evaluated in terms of the global context

Routing State Validity

- **Local** routing state is table in a single router
  - By itself, the state in a single router can’t be evaluated for validity
  - It must be evaluated in terms of the global context

? ? ?

R2’s Table	
Dst	***Port***
A	3
B	1
C	3
D	0

Is this local state valid?

Will it get my packets to their destinations?

No way to tell from just this info!

Routing State Validity

- **Local** routing state is table in a single router
  - By itself, the state in a single router can’t be evaluated for validity
  - It must be evaluated in terms of the global context

- **Global** state is collection of tables in all routers
  - Global state determines which paths packets take
  - It’s ***valid*** if it produces forwarding decisions that always deliver packets to their destinations

Routing State Validity

- **Local** routing state is table in a single router
    - By itself, the state in a single router can't be evaluated for validity
    - It must be evaluated in terms of the global context
- **Global** state is collection of tables in all routers
    - Global state determines which paths packets take
    - It's ***valid*** if it produces forwarding decisions that always deliver packets to their destinations
- Goal of routing protocols: compute valid state
    - We *will* eventually talk about how you build routing state!
    - But given some state… how can you tell if it's valid?
        - Need a *succinct correctness condition for routing…*
            - What makes routing correct / incorrect? Take a few seconds...

Routing State Validity

- A *necessary and sufficient condition* for validity

- **Global routing state is valid *if and only if:***
    - **For each destination…**
        - **There are no dead ends**
        - **There are no loops**

- A ***dead end*** is when there is no outgoing link (next-hop)
    - A packet arrives, but is not forwarded (e.g., because there's no table entry for destination)
    - The destination doesn't forward, but *doesn't count as a dead end*!
    - But other hosts generally are dead ends, since hosts don't generally forward packets

- A ***loop*** is when a packet cycles around the same set of nodes
    - If forwarding is deterministic and only depends on destination field, this will go on indefinitely

Necessary (“only if”)

For state to be valid, it is necessary that there be no loops or dead ends

.. because if there were loops or dead ends, packet wouldn’t reach destination!

(This is pretty straightforward)

- If you hit a dead end before the destination...
  you’ll never reach the destination
  - Obviously

- If you run into a loop…
  you’ll never reach the destination
  - Because you’ll just keep looping (forwarding is deterministic and destination addr stays same)
  - And we know destination isn’t part of a loop (it wouldn’t have forwarded the packet!)

- ***Thus:*** *it’s necessary there be no loops or dead ends!*

Sufficient (“if”)

If there are no loops or dead ends, that is sufficient to know the state is valid
(This is more subtle…)

- Assume the routing state has no loops or dead ends

- Packet can’t hit the same node twice (just said no loops)
- Packet can’t stop before hitting destination (just said no dead ends)

- So packet *must* keep wandering the network, hitting *different* nodes
  - Only a finite number of unique nodes to visit
  - *Must* eventually hit the destination

- ***Thus:*** *if no loops and no dead ends, then routing state is valid*

Break

(When we return: Doing validation)

Putting it to use: verifying routing state validity

- We now have this simple condition to check validity

- Let’s see how to put it to use

A Couple Notes

- Hosts generally do not participate in routing
  - In common case, hosts:
    - Have a single link to a single router
    - Have a *default route* that sends everything to that router
      - (unless they're the destination!)
  - They're not interesting, so we often ignore them except as destinations

A Couple Notes

- Hosts generally do not participate in routing
  - In common case, hosts:
    - Have a single link to a single router
    - Have a *default route* that sends everything to that router
      - (unless they're the destination!)
  - They're not interesting, so we often ignore them except as destinations

- Routers might be legal destinations (in addition to hosts)
  - Depends on the network design
  - Internet Protocol routers can be!
  - But how often have you wanted to talk to a specific router?
  - Host-to-host communication much more common; we'll often ignore routers as destinations
  - But *do* think of all routers as *potential sources* (packets may arrive in unexpected ways!)

Putting it to use: verifying routing state validity

- Focus only on a single destination
  - Ignore all other hosts
  - Ignore all other routing state (why can we do this?)

- For each router, mark outgoing edge with arrow (point at next hop)
  - There can only be one at each node (destination-based)

- Eliminate all links with no arrows

- Look at what's left….
  - State is *valid if and only if* remaining graph is a ***directed delivery tree***

Putting it to use: verifying routing state validity

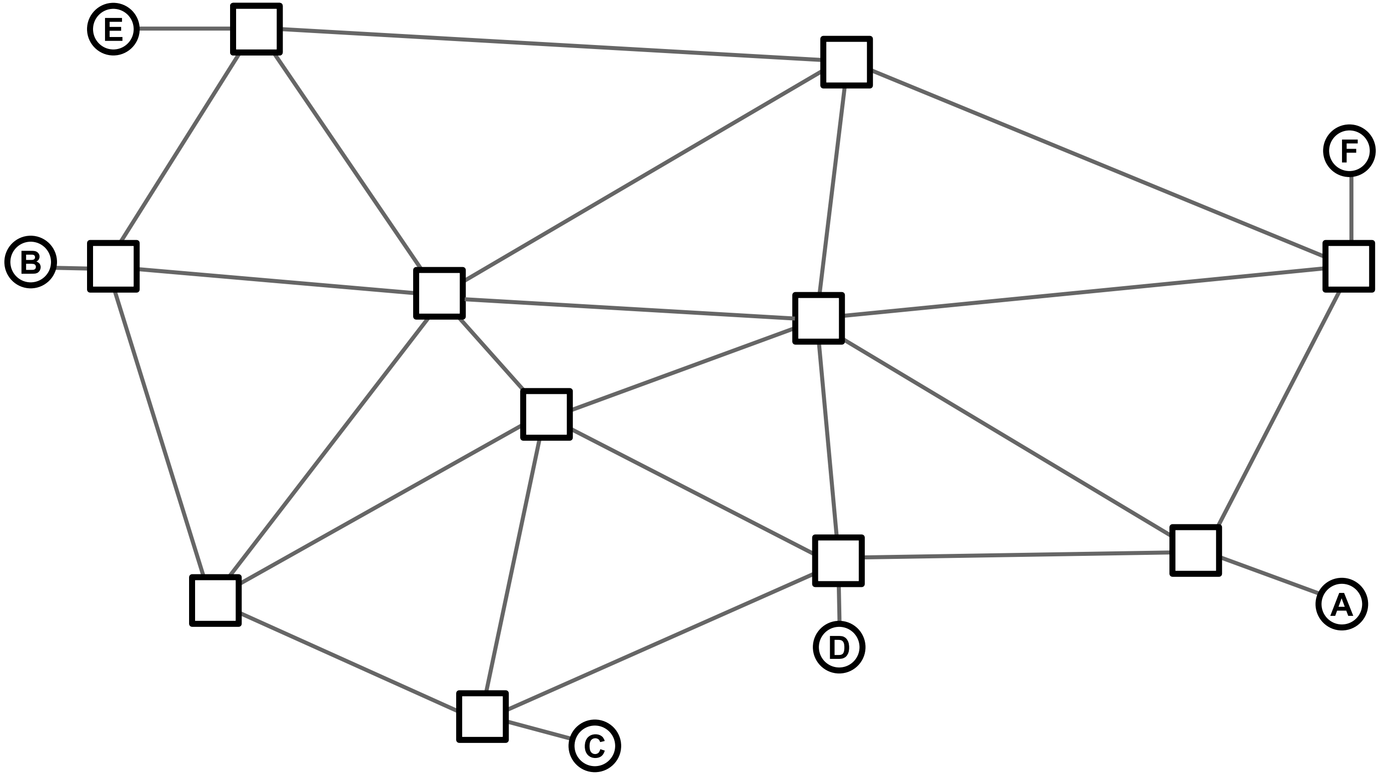
- Focus only on a single destination
  - Ignore all other hosts
  - Ignore all other routing state (why can we do this?)

- For each router, mark outgoing edge with arrow (point at next hop)
  - There can only be one at each node (destination-based)

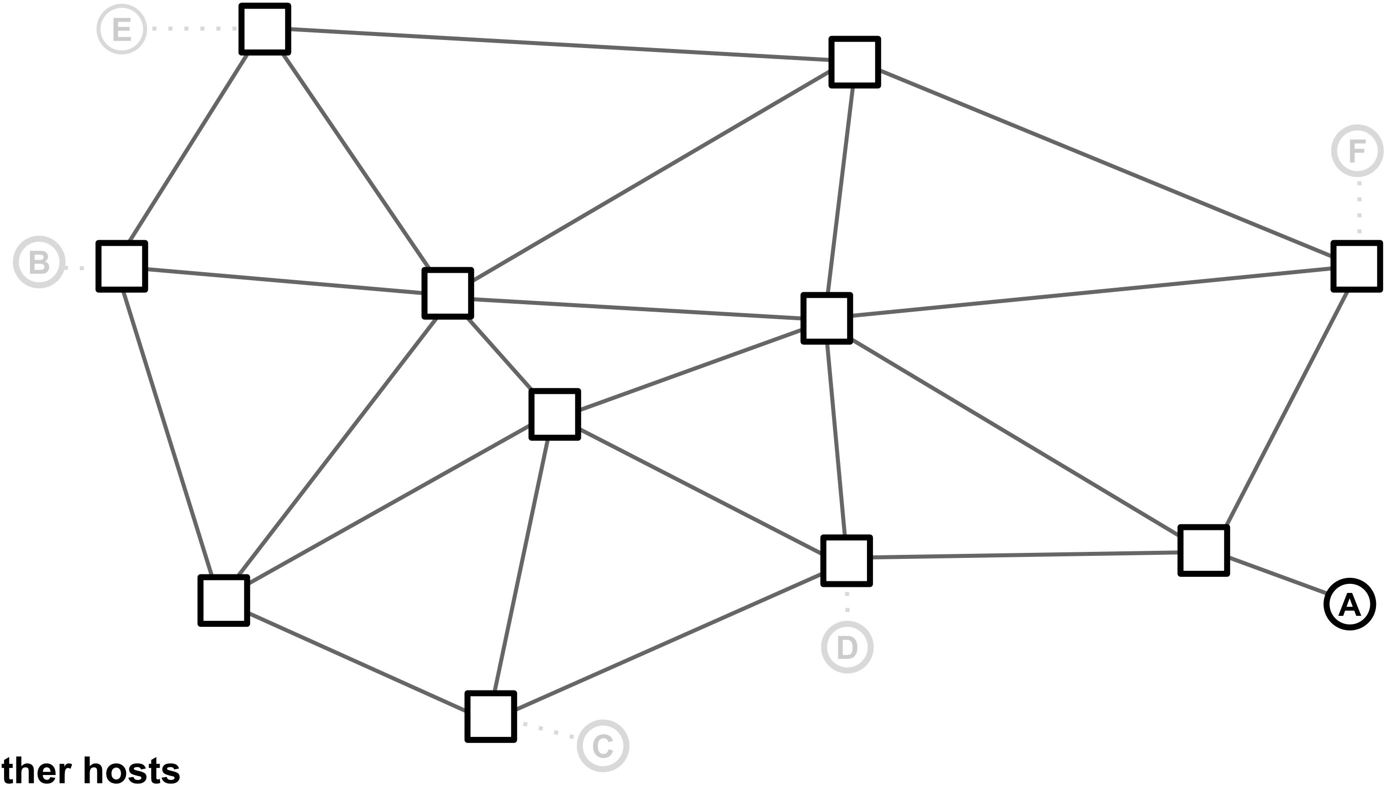
- Eliminate all links with no arrows

- Look at what’s left….
  - State is *valid if and only if* remaining graph is a ***directed delivery tree***
    - Remember: a directed *spanning tree* where all paths point toward destination

Checking validity of state to “A”

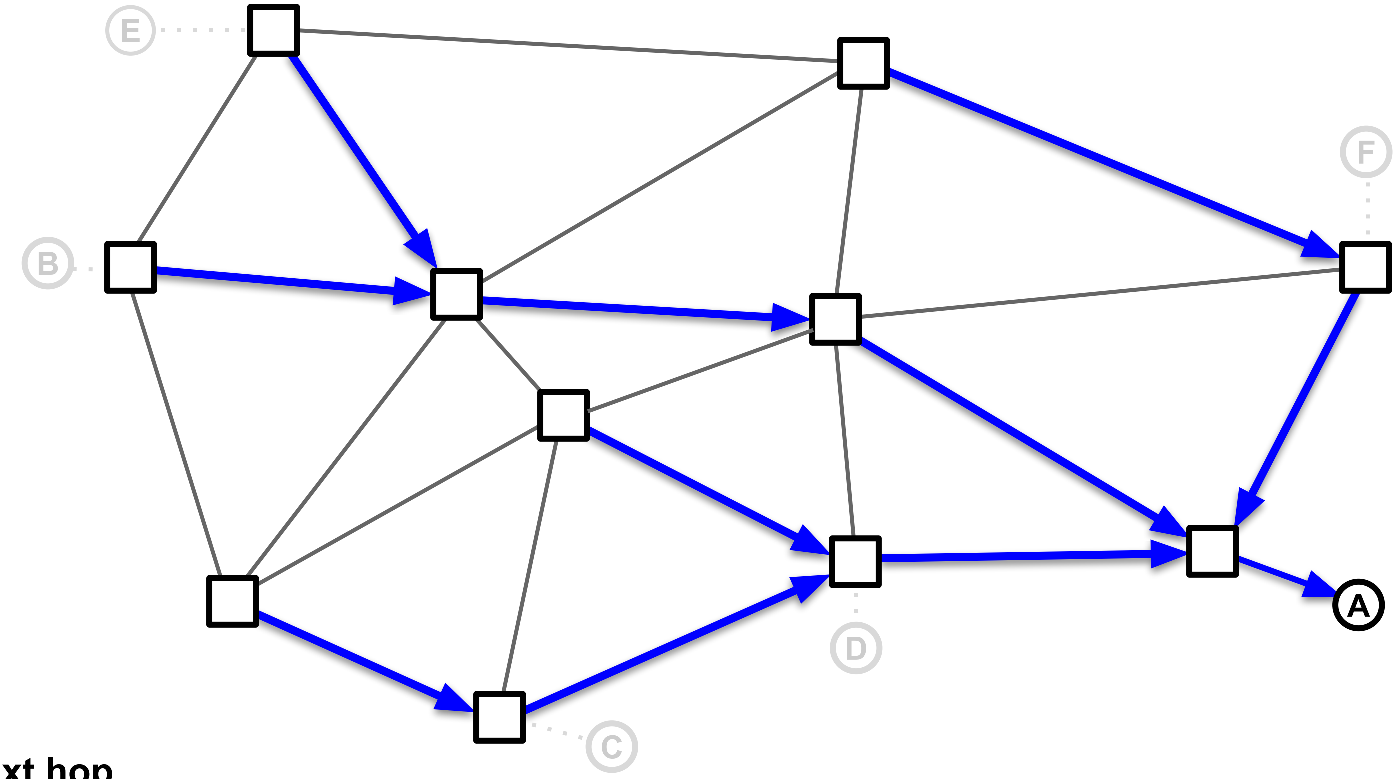


Checking validity of state to "A"



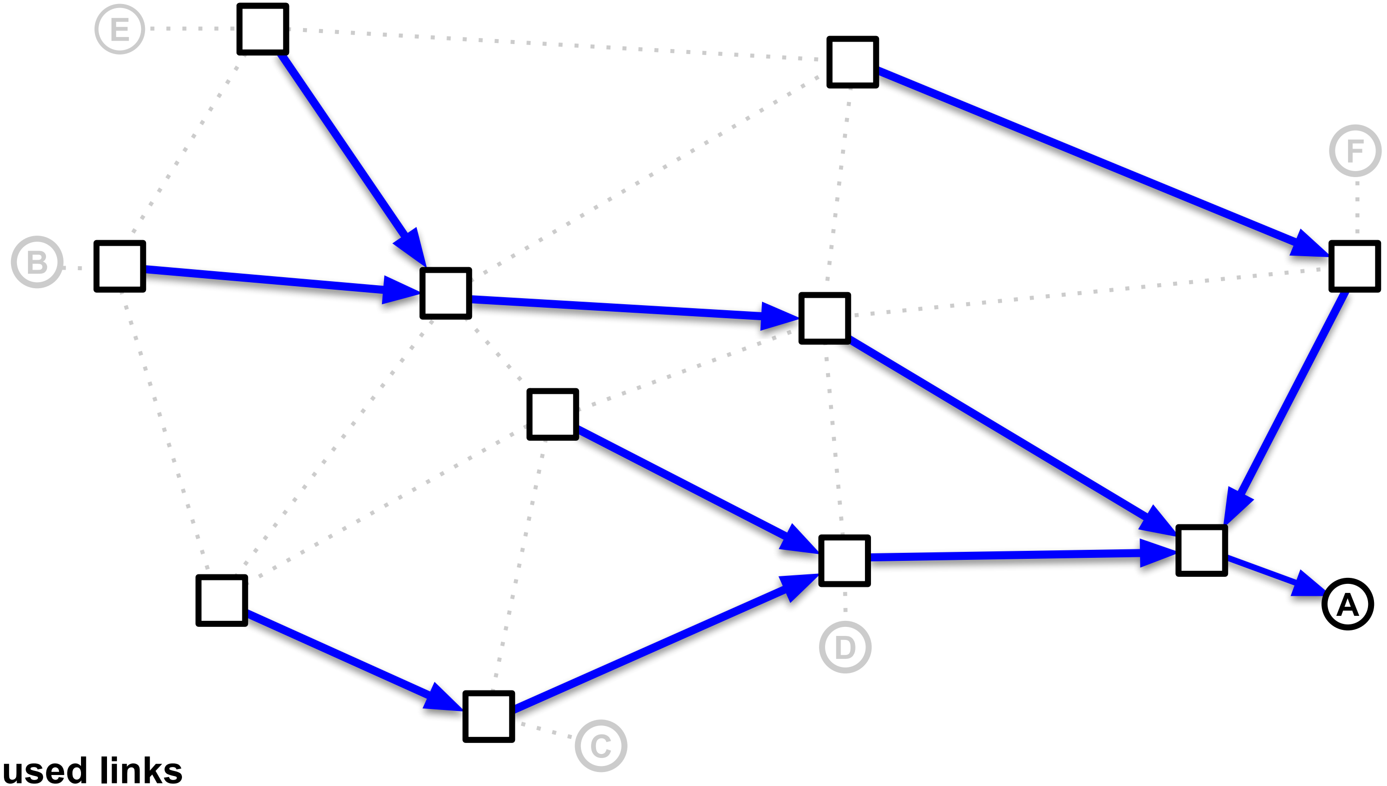
Ignore all other hosts

Checking validity of state to "A"



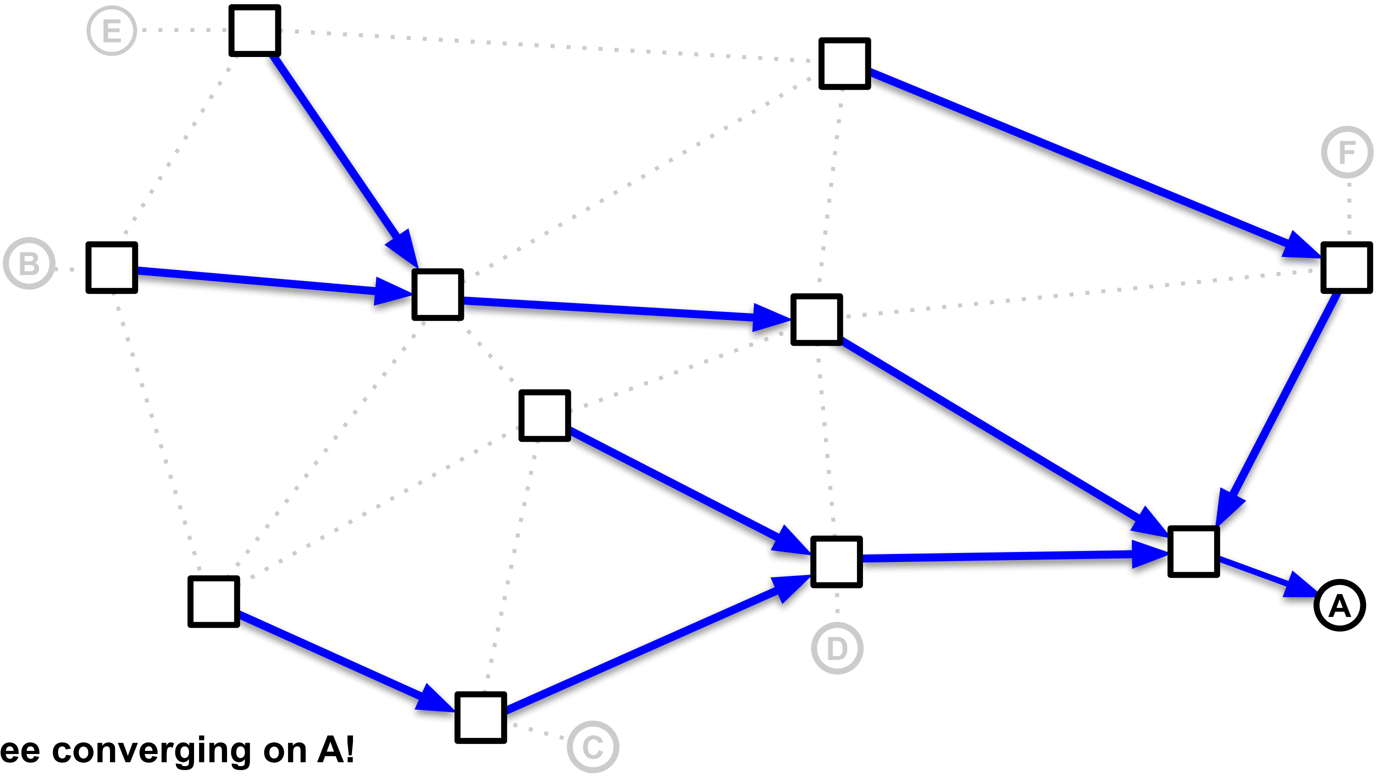
Point at next hop

Checking validity of state to "A"



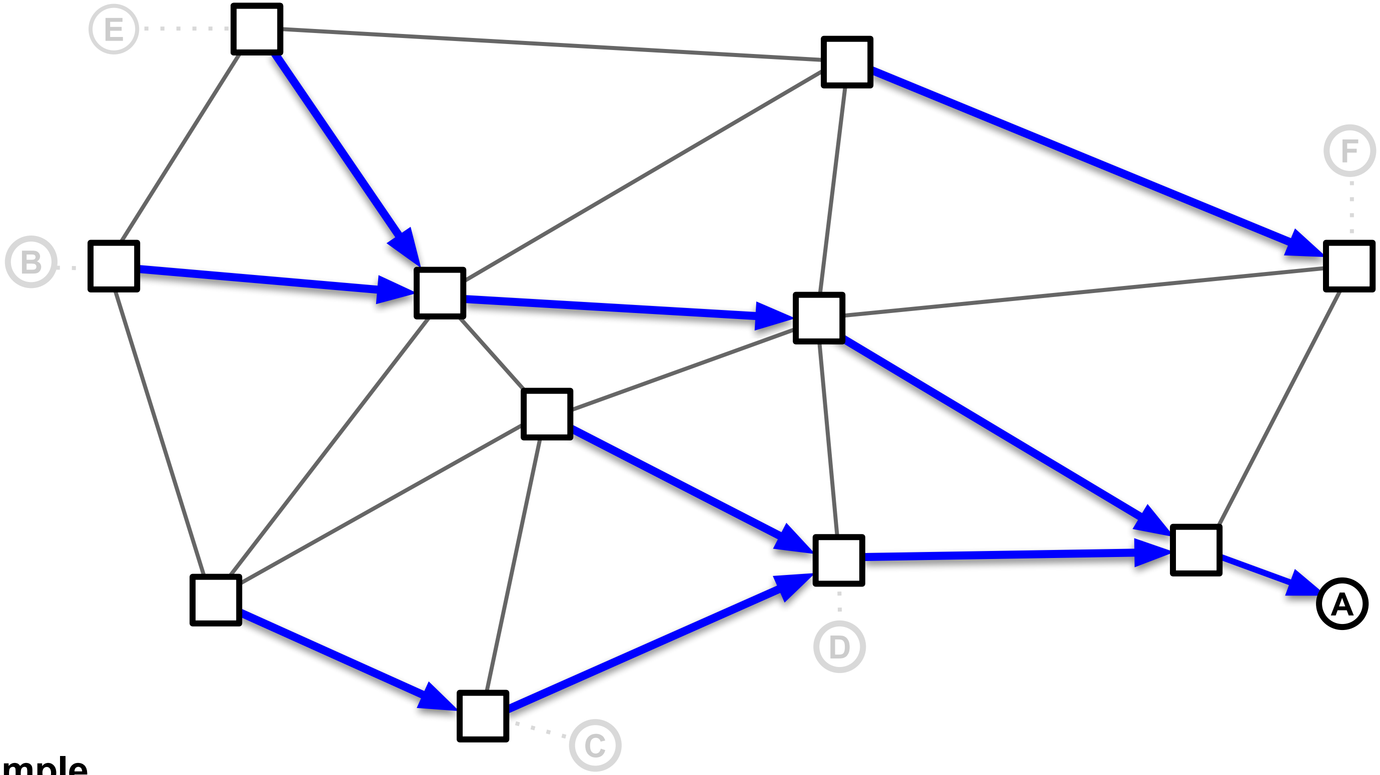
Remove unused links

Checking validity of state to “A”



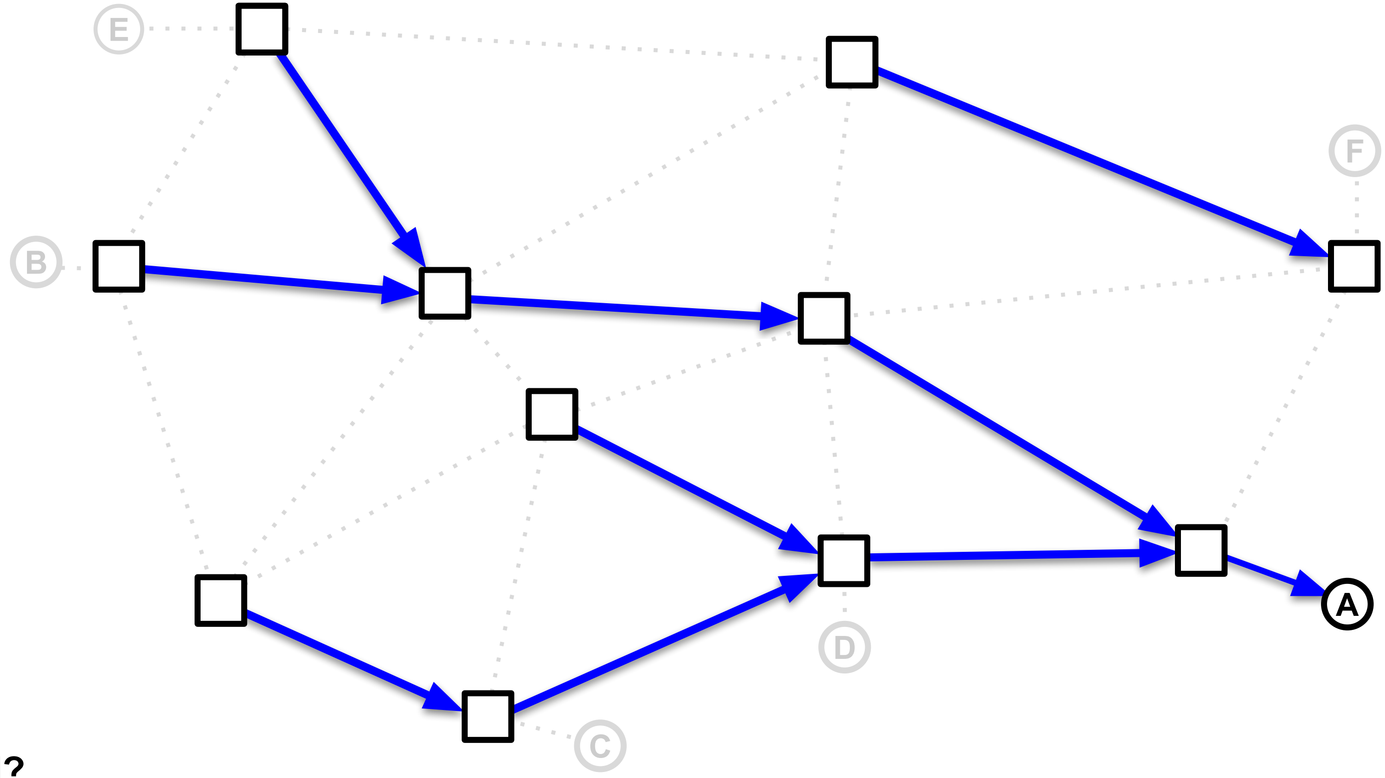
Spanning tree converging on A!
→ It’s valid!

Checking validity of state to “A”



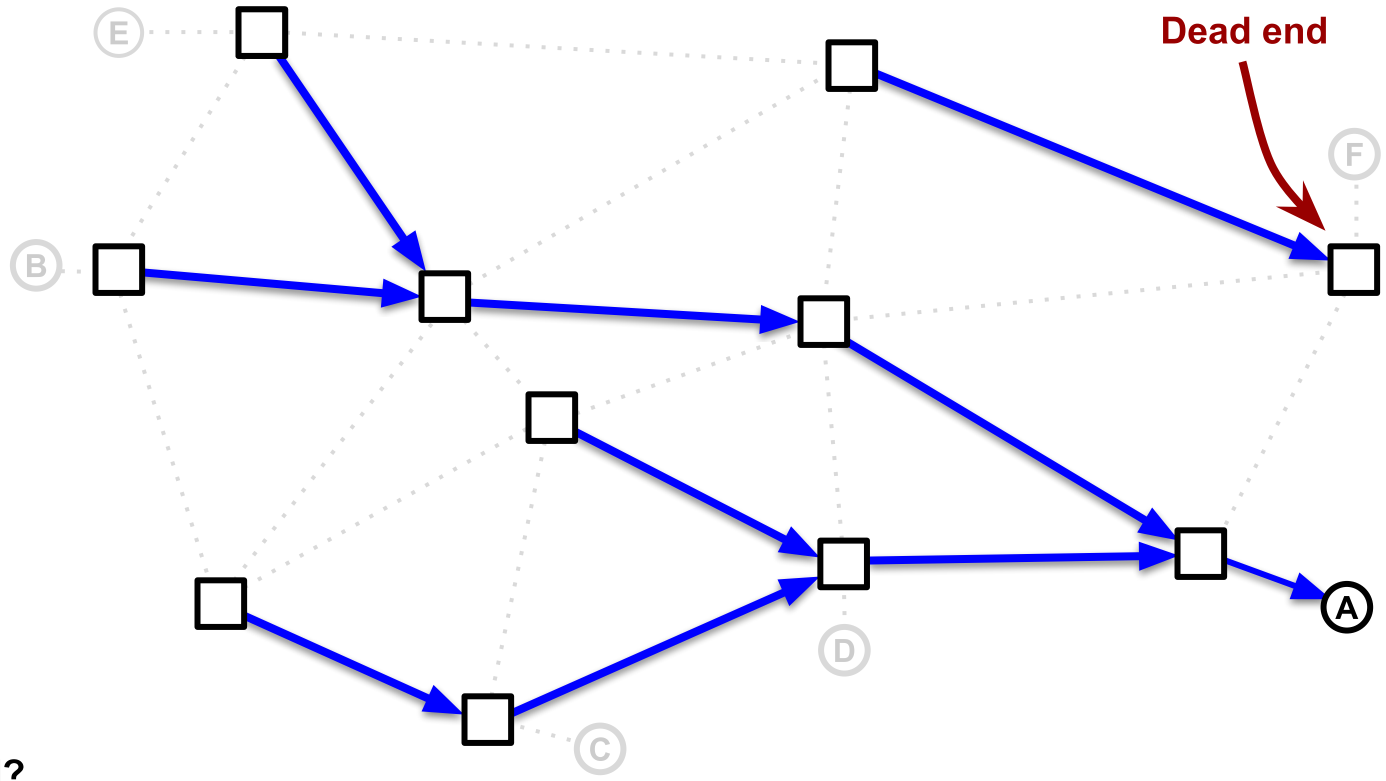
Second example...

Checking validity of state to “A”



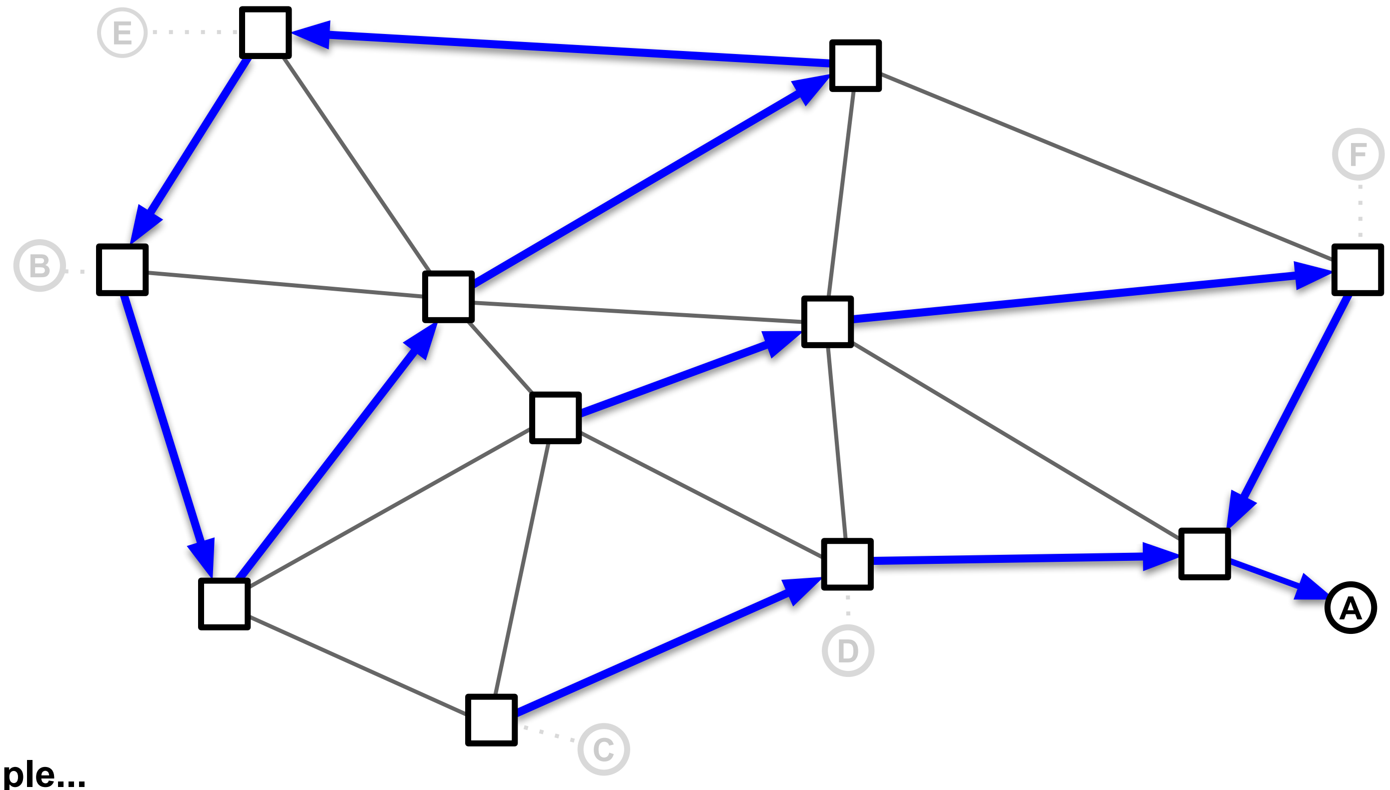
Is this valid?

Checking validity of state to "A"



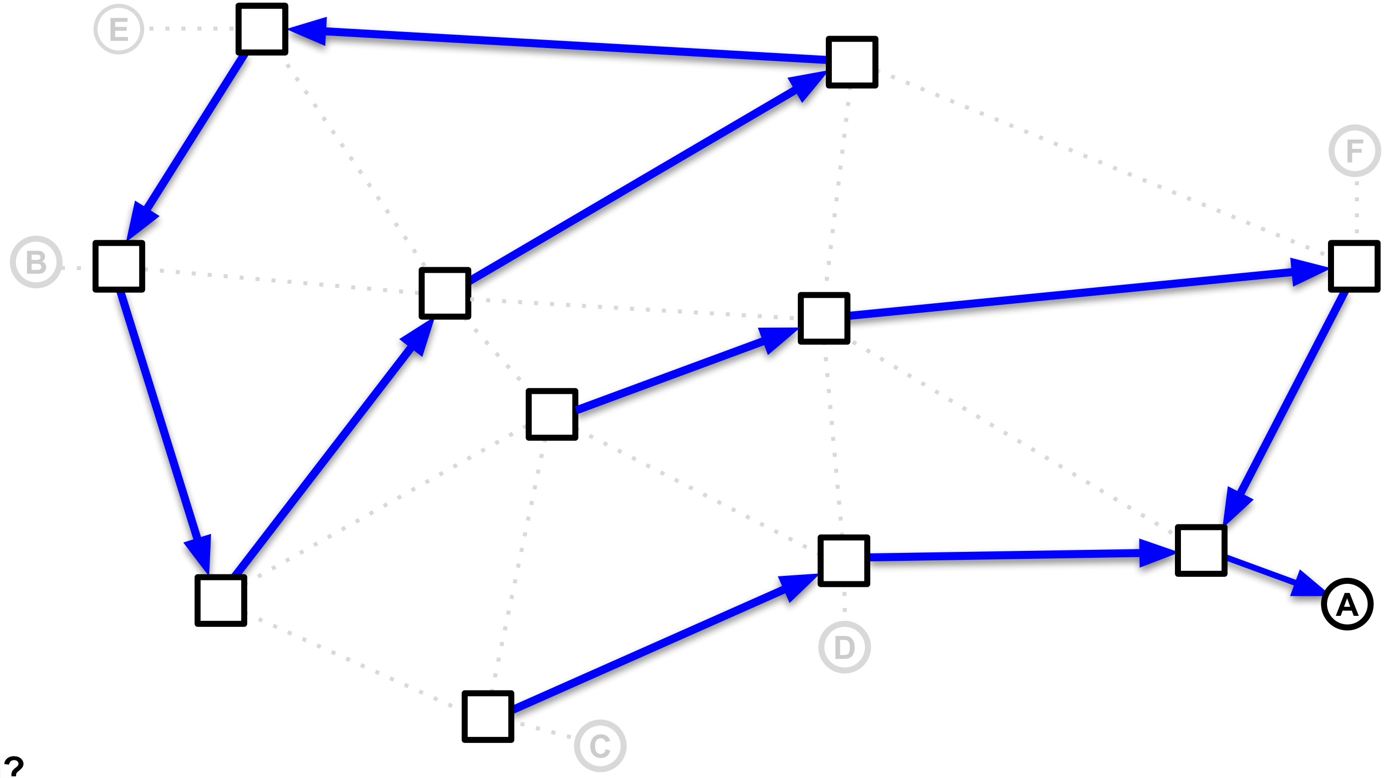
Is this valid?

Checking validity of state to "A"



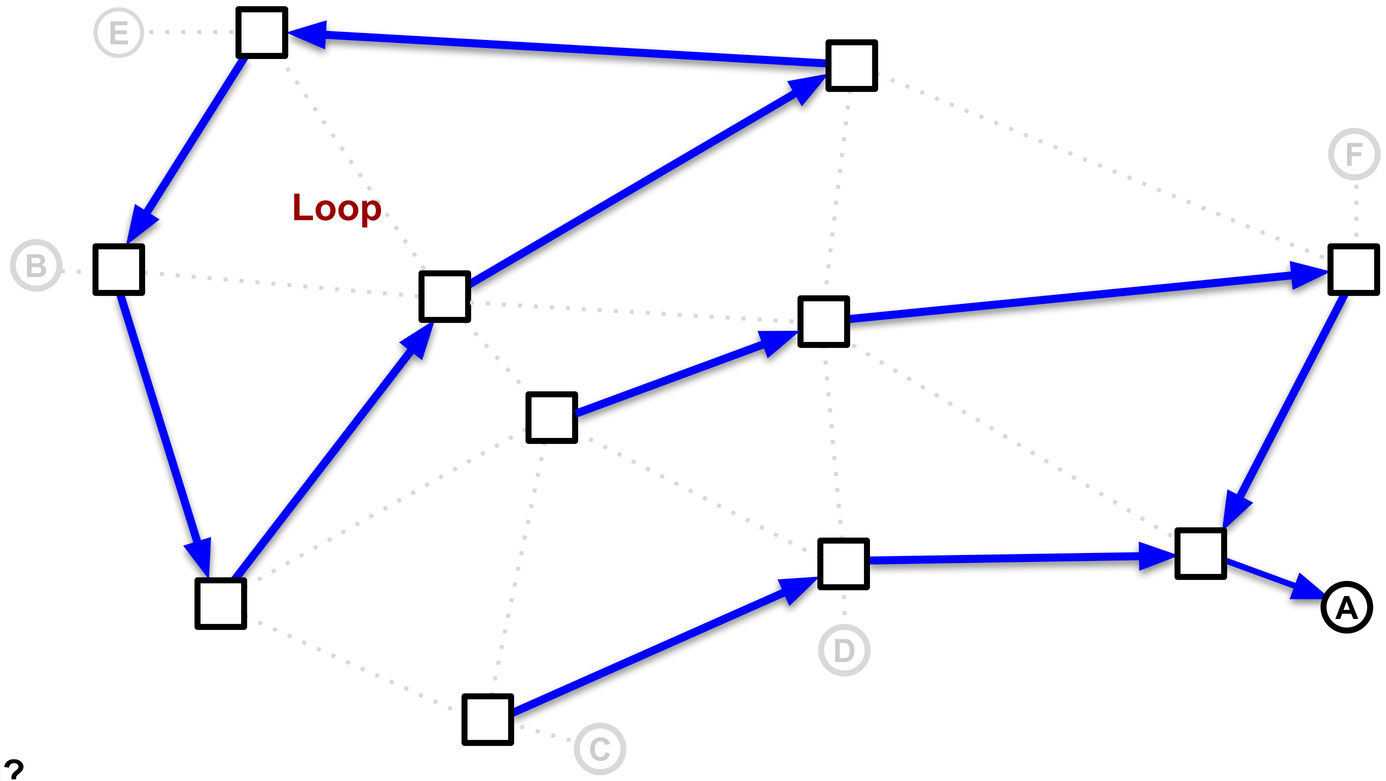
Third example...

Checking validity of state to “A”



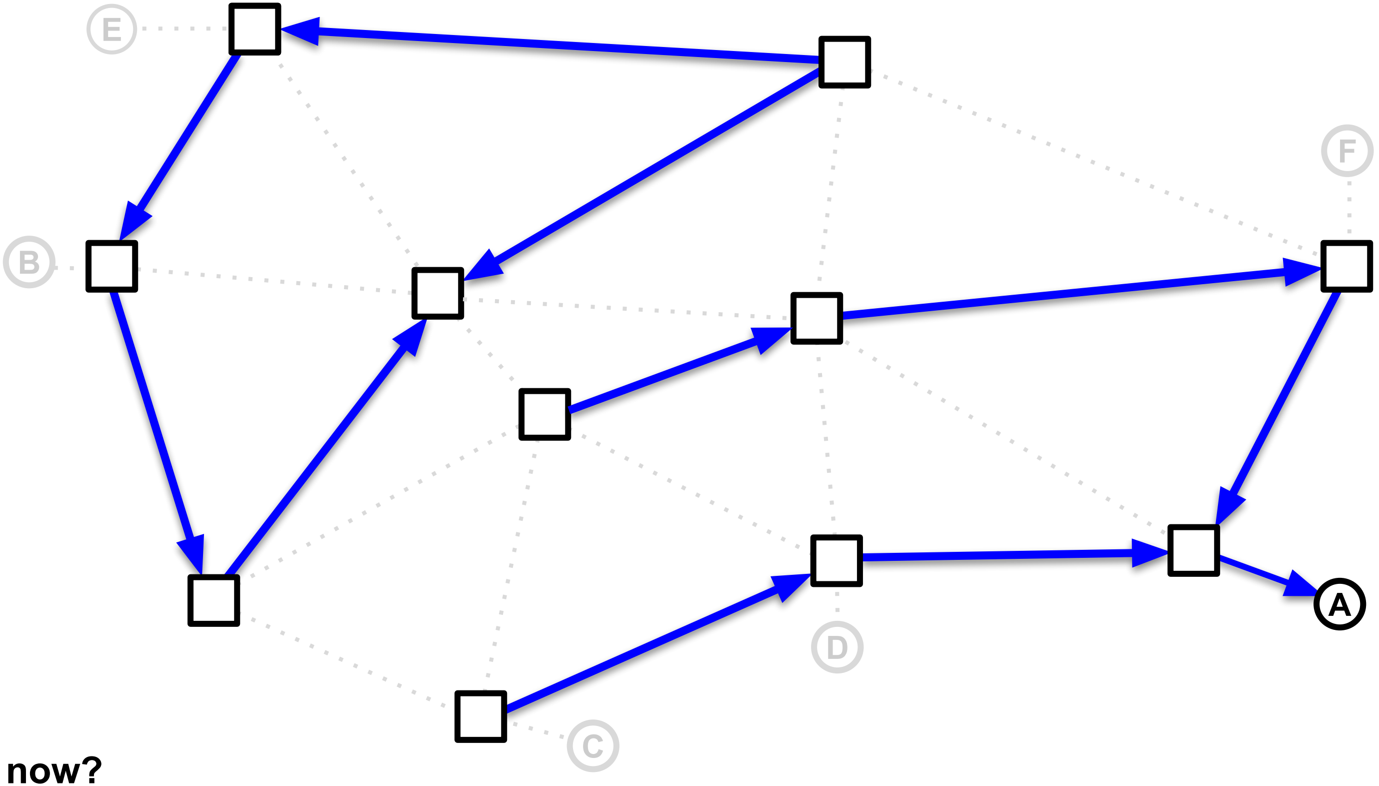
Is this valid?

Checking validity of state to “A”



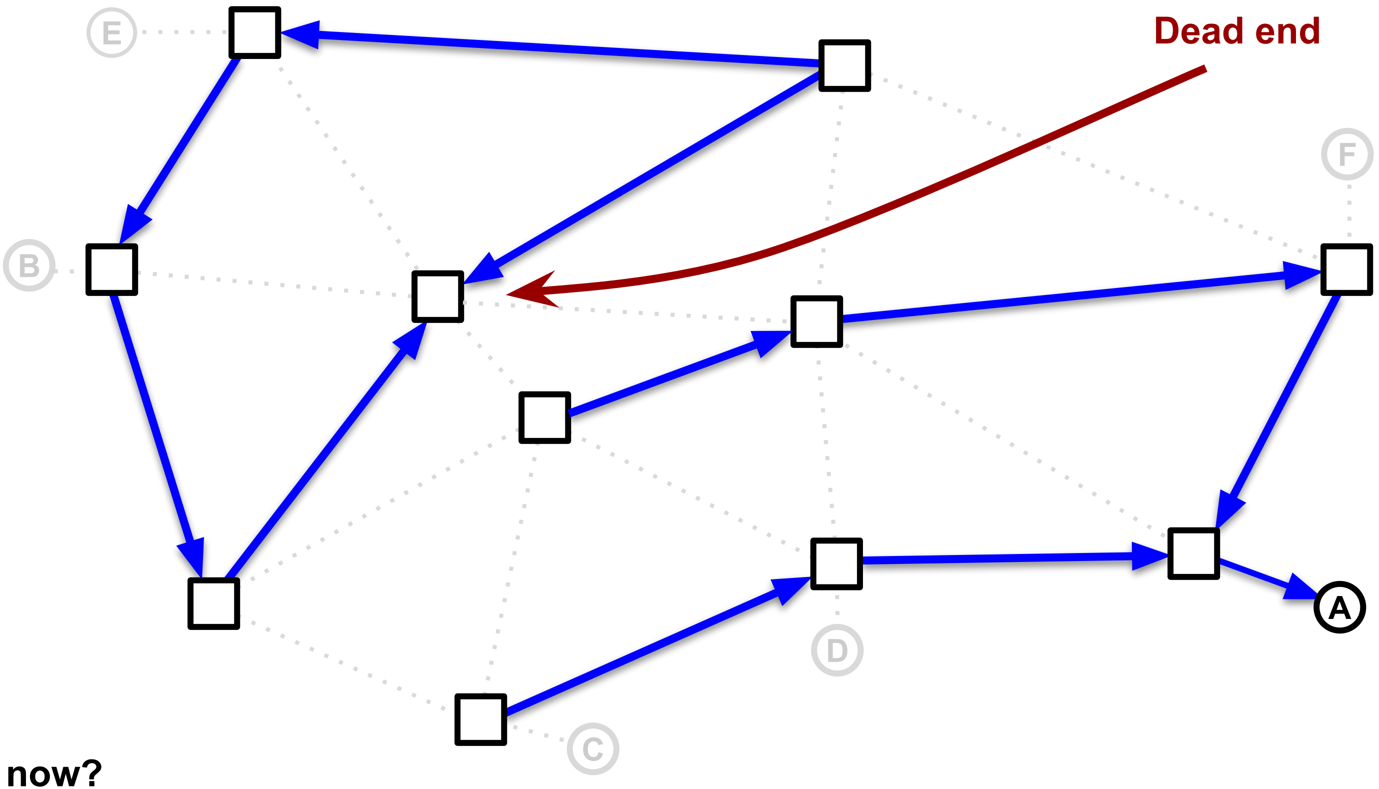
Is this valid?

Checking validity of state to “A”



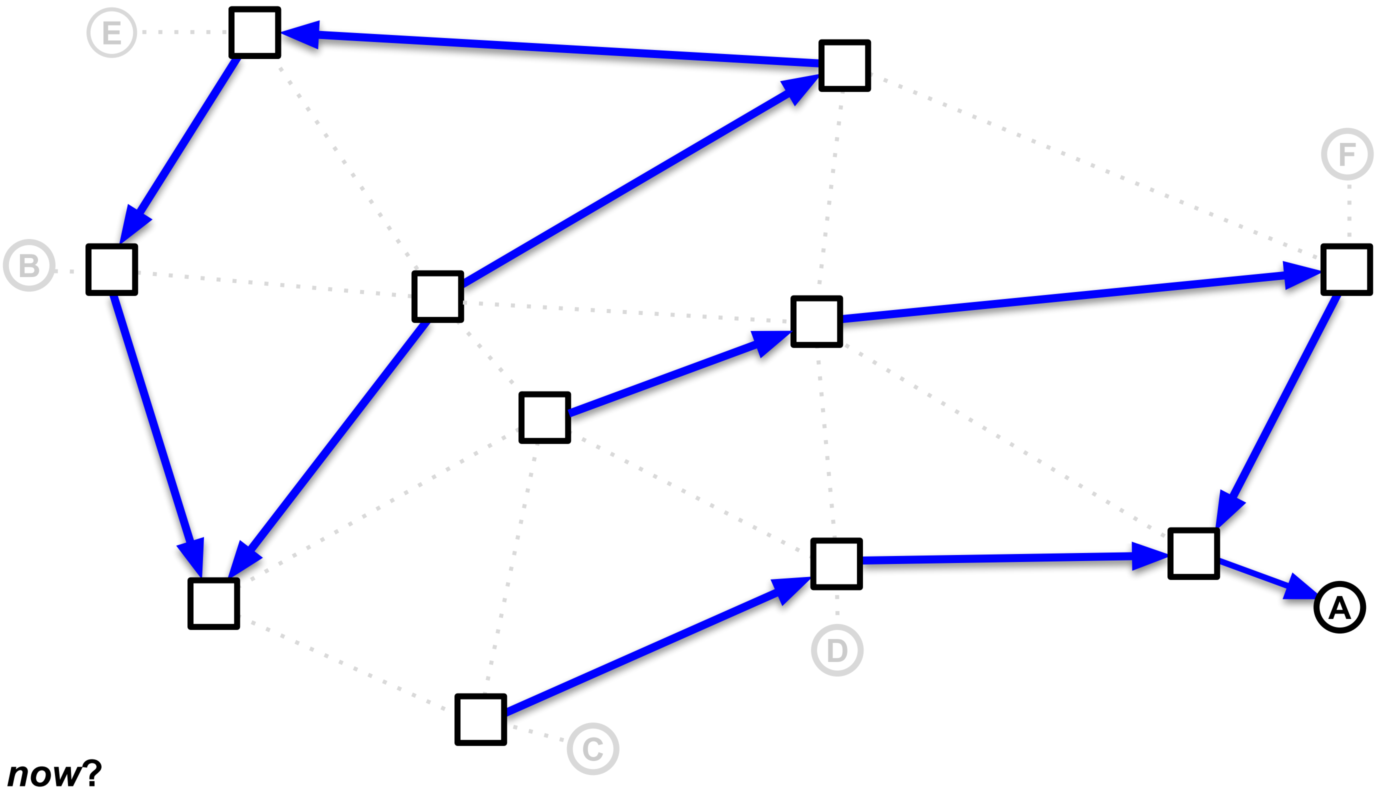
What about now?

Checking validity of state to "A"



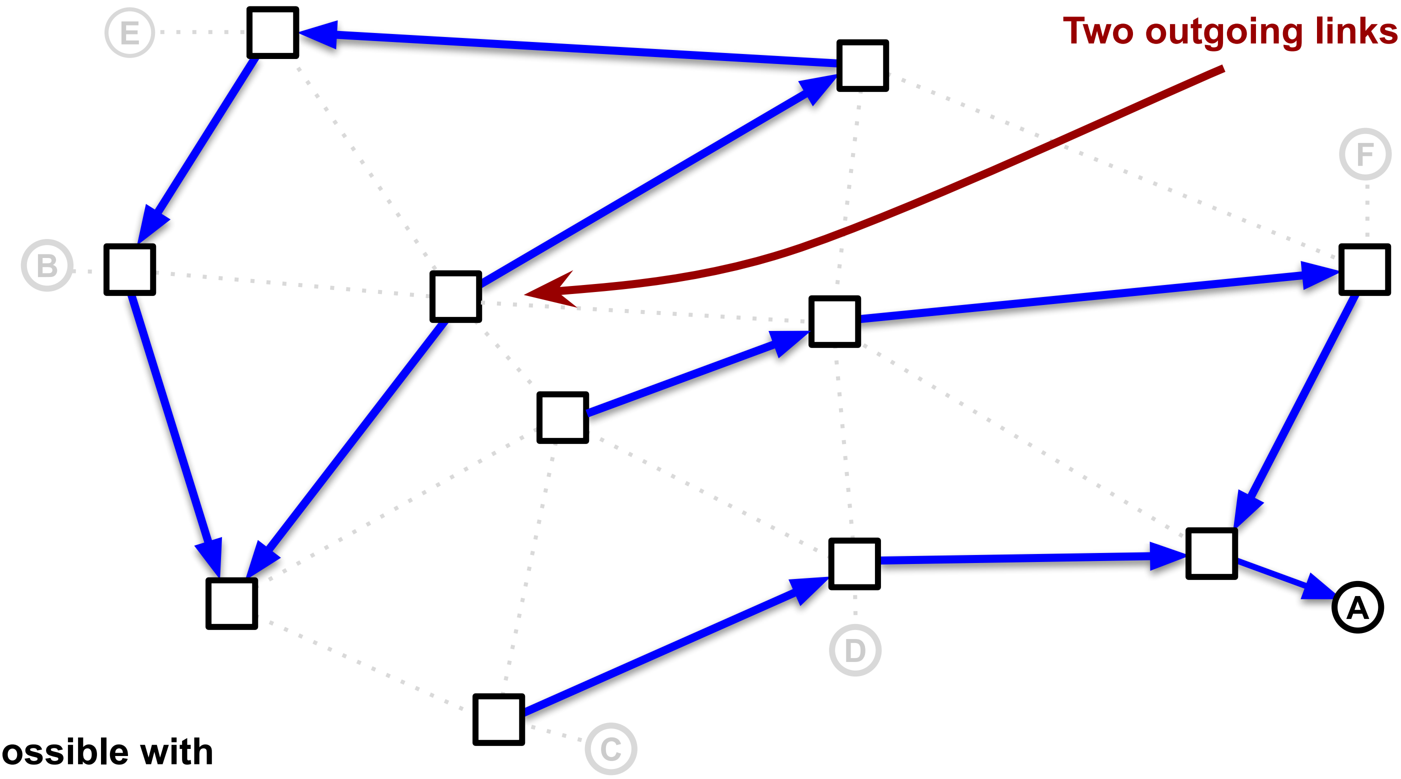
What about now?

Checking validity of state to "A"



What about *now*?

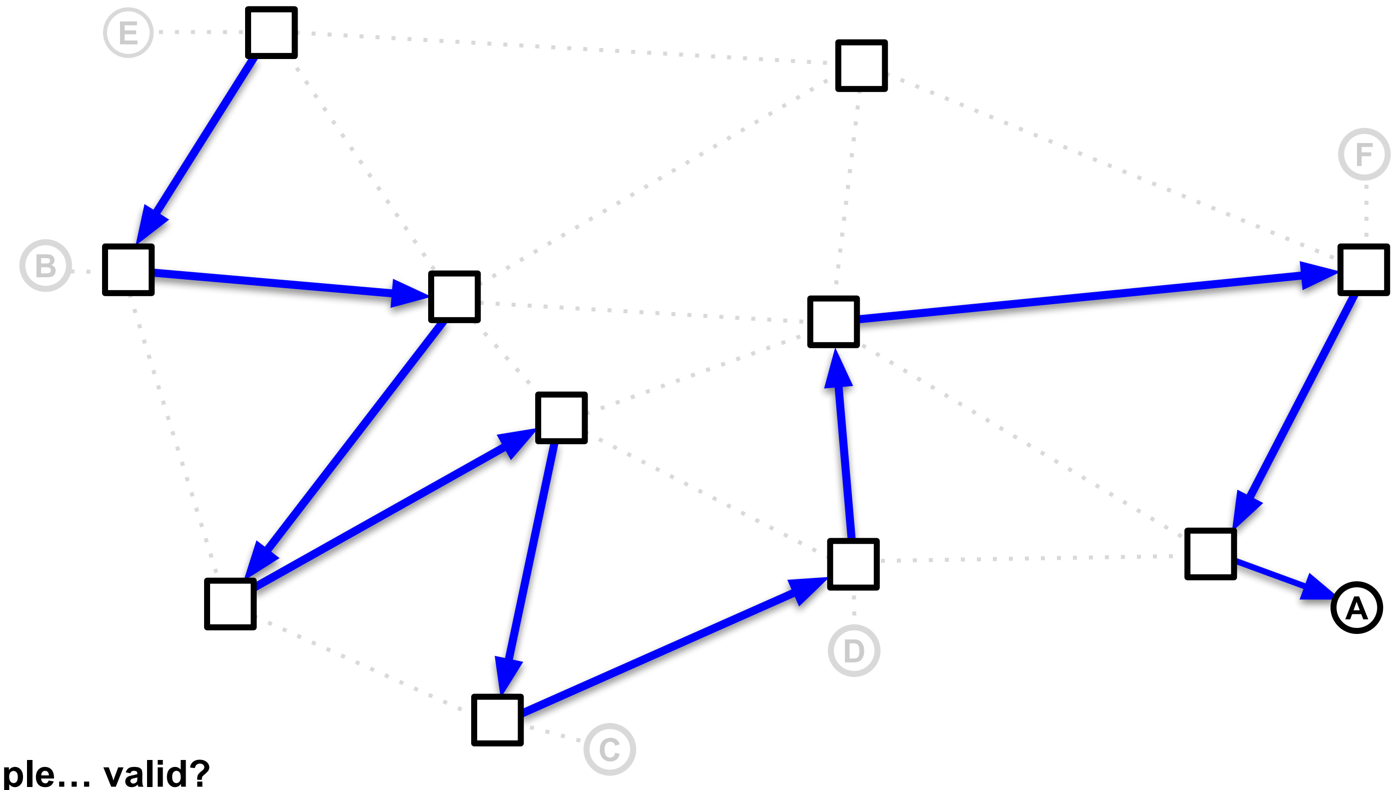
Checking validity of state to “A”



Two outgoing links?!

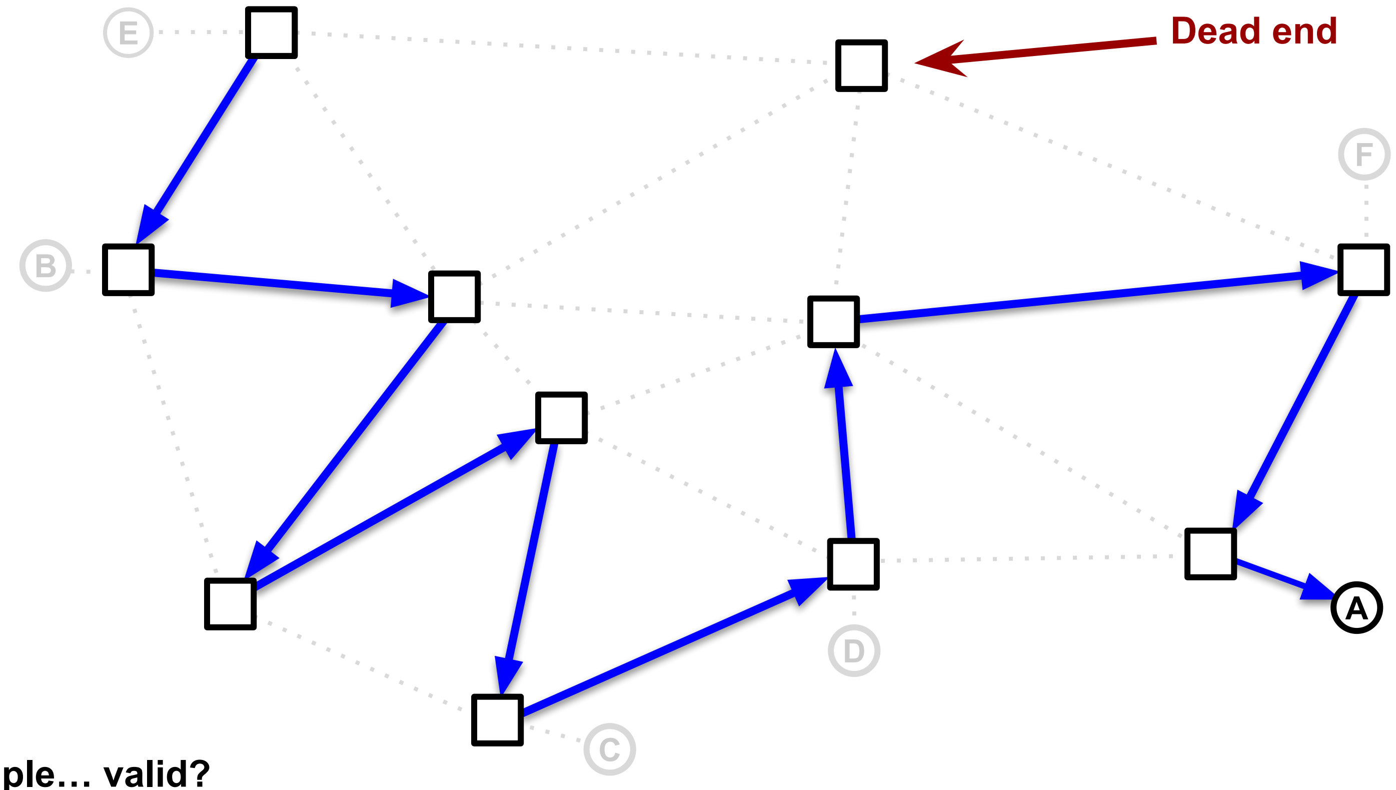
Not even possible with destination-based routing!

Checking validity of state to "A"



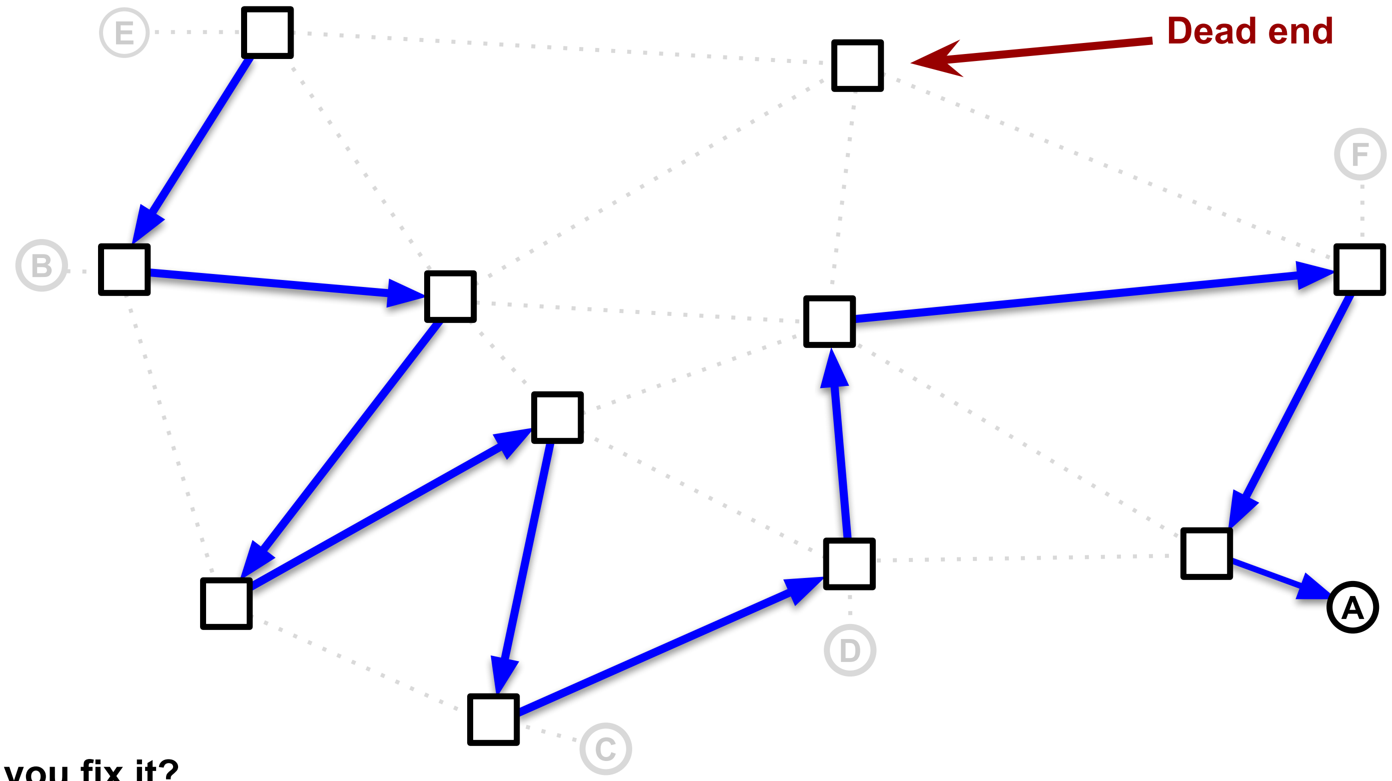
Final example… valid?

Checking validity of state to “A”



Final example… valid?

Checking validity of state to “A”



How could you fix it?

Verifying routing state validity

- Very easy to check validity of routing state for a particular destination...
- Dead ends are obvious
  - A node with no outgoing arrow can’t reach destination
- Loops are obvious
  - Disconnected from destination (and entire rest of graph!)
- .. now just repeat for each destination!

Finally: A note on generality

- We're looking at this from perspective of destination-based routing

- Same basic *no loops or dead ends* condition generalizes to *at least** any other system that does deterministic forwarding based on fixed packet headers (that is, it's not *limited* to destination-based routing)

- We just need to:
    - Make one minor addition
    - Carefully consider what constitutes a loop

- We'll probably revisit this next week

Let’s try something...

Here are the rules...

- We’re all going to work together
- You’re going to talk to your neighbors (people sitting to your left and right and in front and behind you)
  - Obviously, you may not have neighbors on all sides! (But hopefully at least one!)

- Everyone has a *magic number*
- Your magic number is *initially infinity*
- You *want* to have *as low of a magic number as possible*
- If your neighbor offers you a lower number…
  - *Take it!* It’s now your magic number
  - *Immediately* offer your magic number *plus one* to all your neighbors
  - *Try* to remember who gave you your magic number
- If someone offers same number or greater, ignore it

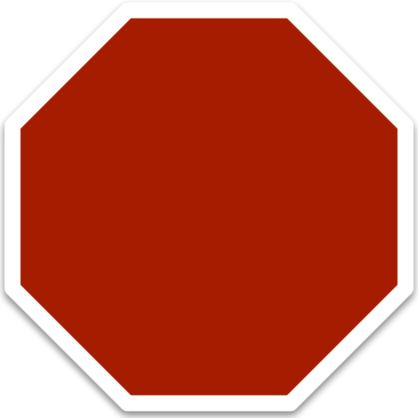
- Initialize:
    - Your number is infinity!
    - Tell your neighbors your name and offer them your number + 1 (i.e. **offer them infinity**)

- While True:
    - If a neighbor offers you a **lower** number:
        - That's now your number! **Remember it**! You want a low number!
        - **Immediately** offer **number + 1** to all your neighbors

… or ...

```
my_number = infinity
offer_to_neighbors(my_number + 1)

while True:
    offer = wait_for_offer_from_a_neighbor()
    if offer < my_number:
        my_number = offer  # I want lower
        offer_to_neighbors(my_number + 1)
```



Stop!

(But remember your number!)

Your Best Friend

- The person who gave you your current number is your *best friend*
    - Note that you are never your best friend’s best friend. Sorry. 😢

    - Did you forget who gave you your number?
        - Easy enough to figure out
        - You must have at least one neighbor whose number is yours - 1
            - Any of those could have given you your number
            - Pick any such person to be your best friend

- If someone gives you an envelope, give it to your best friend
    - If your best friend gives you an envelope, they must be confused!

With any luck...

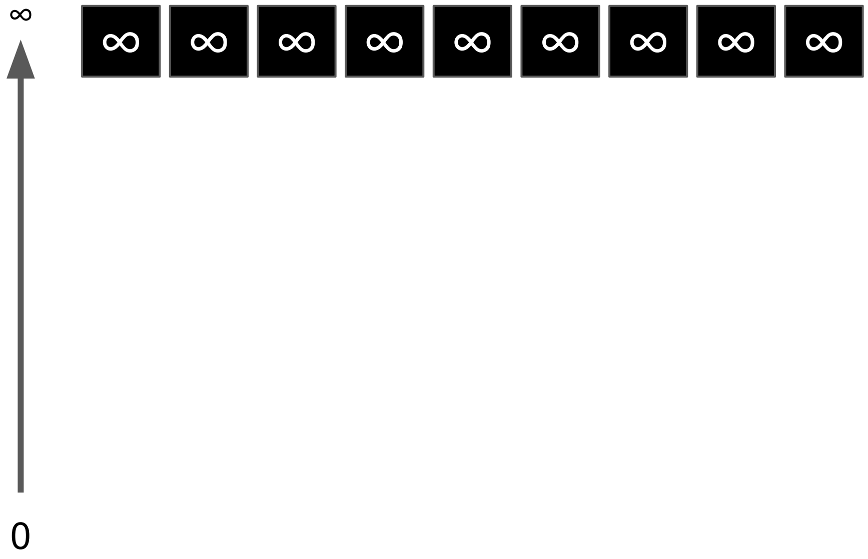
- With any luck, envelopes got to their intended destination!

With any luck...

- With any luck, envelopes got to their intended destination!
- How?

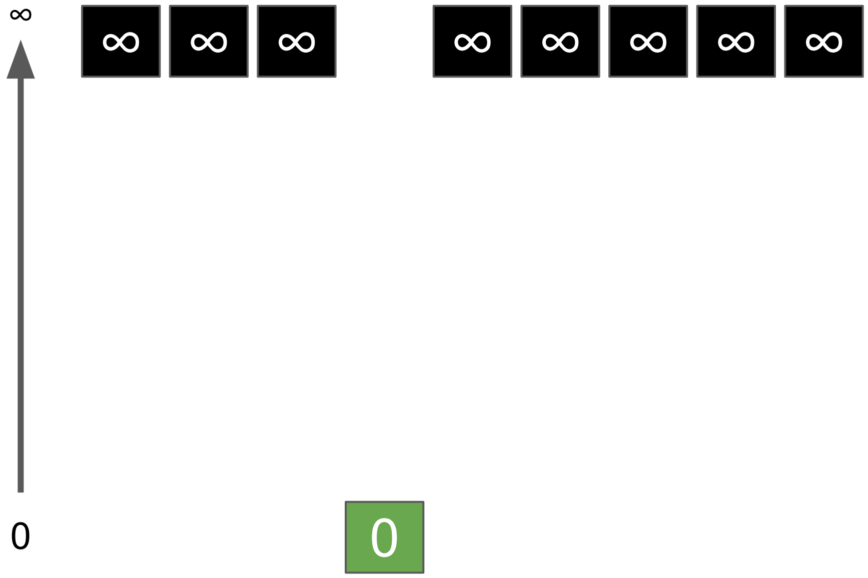
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity



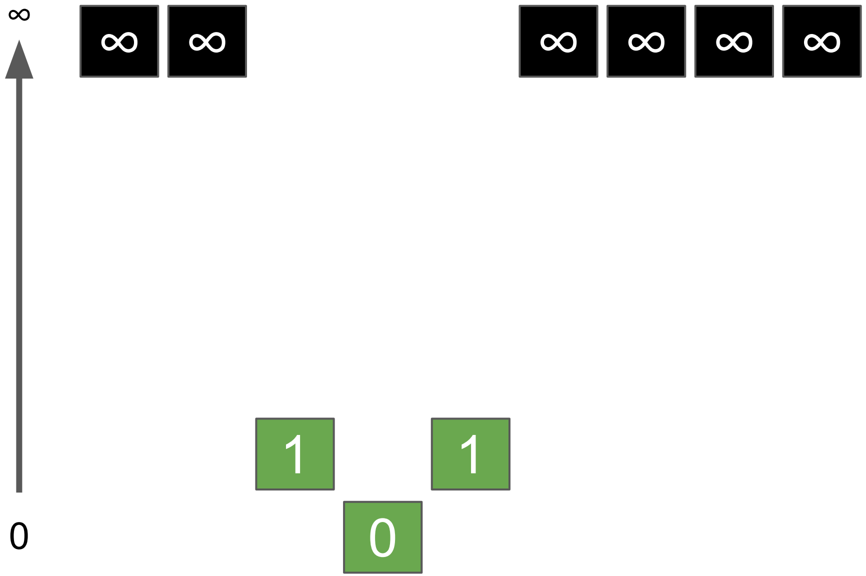
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity
  - We gave one person (destination) zero



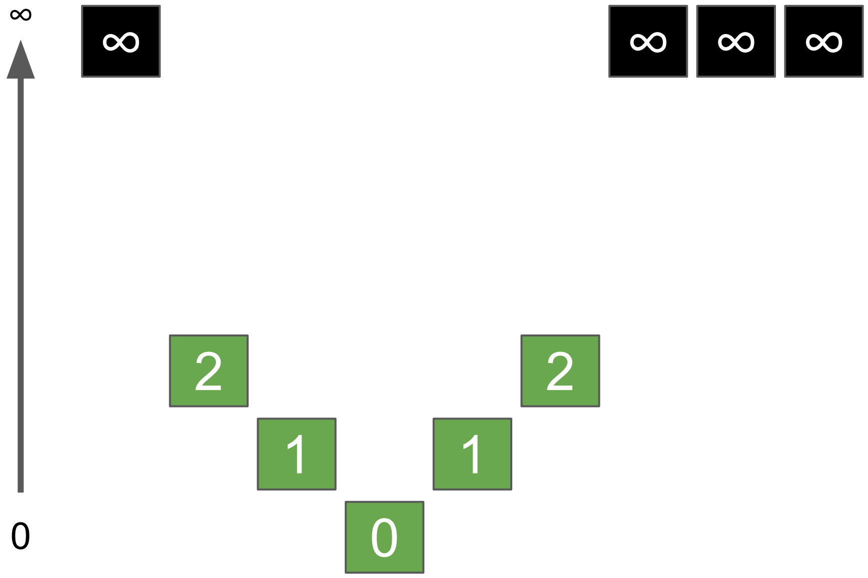
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
    - Everyone started with infinity
    - We gave one person (destination) zero
    - Who gave their neighbors one



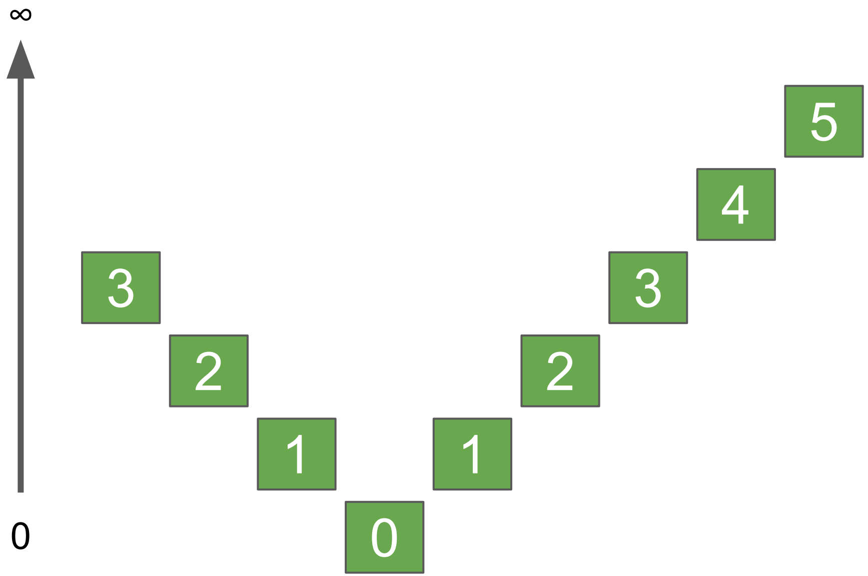
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity
  - We gave one person (destination) zero
  - Who gave their neighbors one
  - Who gave their neighbors two



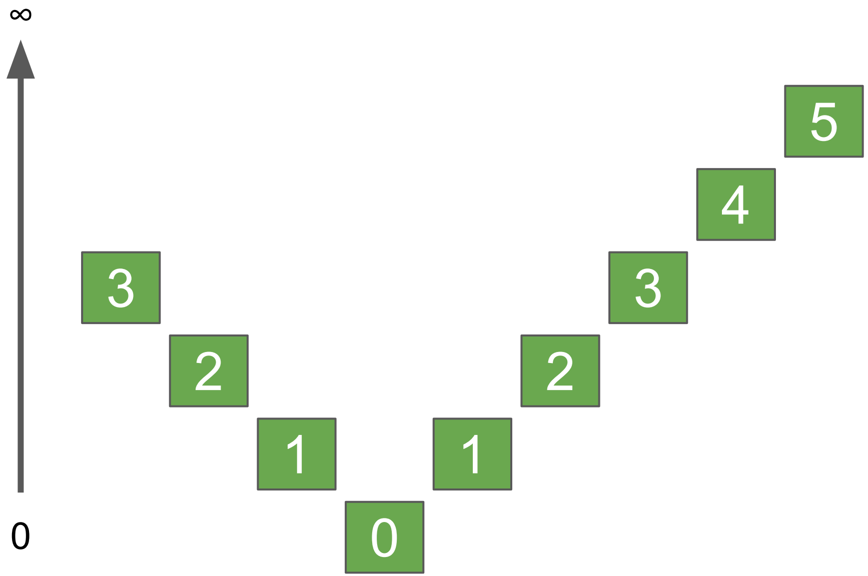
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity
  - We gave one person (destination) zero
  - Who gave their neighbors one
  - Who gave their neighbors two
  - etc.



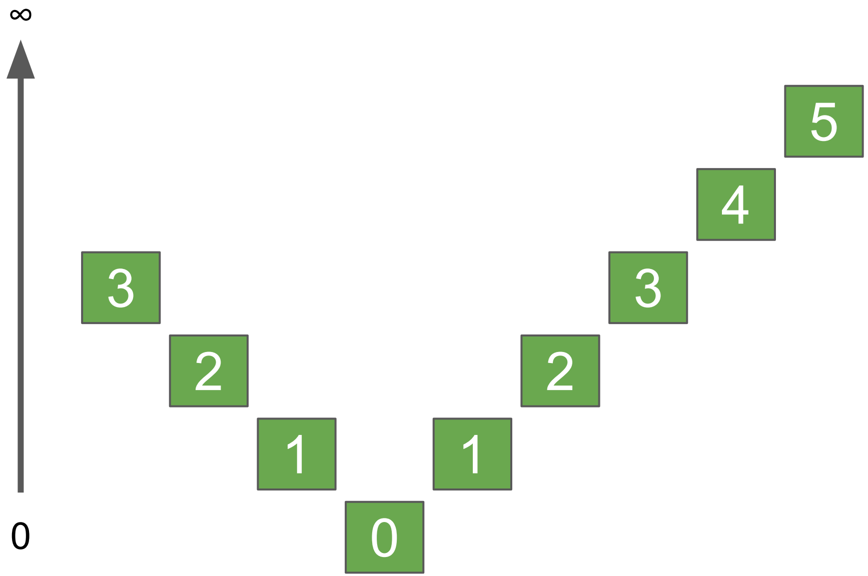
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity
  - We gave one person (destination) zero
  - Who gave their neighbors one
  - Who gave their neighbors two
  - etc.
  - Created this slope down to destination!



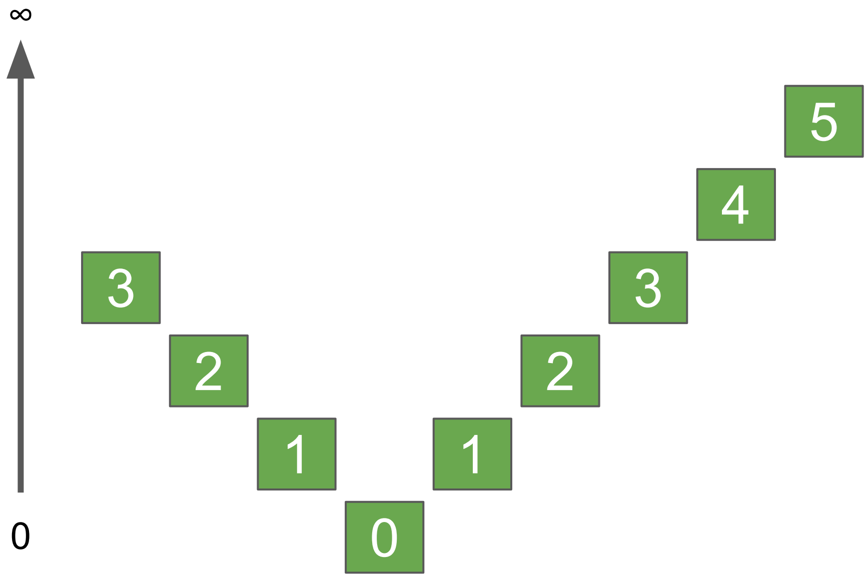
With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity
  - We gave one person (destination) zero
  - Who gave their neighbors one
  - Who gave their neighbors two
  - etc.
  - Created this slope down to destination!
- Stage 2:
  - From wherever, hand envelope down slope
  - It arrives at destination!



With any luck...

- With any luck, envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity
  - We gave one person (destination) zero
  - Who gave their neighbors one
  - Who gave their neighbors two
  - etc.
  - Created this slope down to destination!
- Stage 2:
  - From wherever, hand envelope down slope
  - It arrives at destination!
- Generalizes to many destinations
  - Just keep a separate number per destination!



What could go wrong?

(Or maybe what *did* go wrong?)

Thoughts?

What could go wrong?

- Sitting too far apart (network is partitioned)
- Forgot magic number (router failed/rebooted)
- Mis-remembered or mis-updated magic number (implementation bug)
- Neighbor didn’t hear an update (packet drop)
- Someone left *after* a neighbor accepted their offer (link failure)
- Someone lied about their number (malicious actor)
- Others?

The Bellman-Ford Algorithm

- This was a *distributed* & *asynchronous* version of the Bellman-Ford algorithm
    - (or maybe that should be the Shimbel-Ford-Moore-Bellman algorithm?)

- Two things we know about networks…
    - They’re distributed (many independent components)
    - They’re asynchronous (components don’t operate in sync)

- .. this seems like a promising algorithm to turn into a routing protocol!
    - We will do this on Thursday!

Distance-Vector Protocols

- Routing protocols that work like this are called *Distance-Vector* protocols
  - Adjacent routers conceptually exchange a *vector* (i.e., array) of distances
    - More like a vector of (destination,distance) tuples?
- Used in ARPANET (Internet precursor) as far back as 1969
- Later used by XEROX
- Then by *`routed`* in Berkeley Software Distribution Unix 1983
  - `routed`'s protocol standardized as RFC 1058 (***Routing Information Protocol*** / ***RIP***) in 1988
  - Updated for classless addressing (we'll find out about that) in RFC 1723 in 1994
  - Updated for IPv6 in RFC 2080 in 1997
  - Kind of the "prototypical distance-vector protocol" (Sorry)
  - **Our investigation of D-V (and the project) largely inspired by it!**
- D-V pretty widely deployed historically; less popular today, but not dead!
- Cisco ***EIGRP*** (1993) is more advanced, published in RFC 7868 in 2016
- ***Babel*** published as experimental RFC 6126 in 2011; actively worked on

Fujitsu and Cisco CR-1 [16 slot chassis type]
https://www.fujitsu.com/global/about/resources/news/press-releases/2005/0524-01.html
https://www.fujitsu.com/global/Images/20050524_16a_l_tcm100-929965.jpg

Cisco 3750G, Catalyst 5505, Artist: Stefan Thierolf
License: CC BY-NC-SA 4.0
https://www.thierolf.org/albums/the-cisco-hardware-i-have-at-home.html

Router schematic image
License: CC-BY-SA-3.0-migrated
https://commons.wikimedia.org/wiki/File:Router.svg

TP-Link WR841ND WiFi router
License: CC-BY-4.0
https://commons.wikimedia.org/wiki/File:TP-Link_WR841ND_WiFi_router_transparent.png

OPTE Internet maps
License: Creative Commons
http://www.opte.org/maps/