Routing #1

Spring 2024
cs168.io

Rob Shakir

One of the fundamental problems:
Routing

Plan for today

- Setting the scene.
    - Disclaimer
    - How will we think about addressing (today)?
    - What is a router?
    - Why do we have routers?
    - The challenge of routing
    - The challenge of forwarding
    - Forwarding vs. Routing.
- A theoretical perspective and routing validity.
    - Graph representation of routing state.
    - What does valid routing mean?
    - How do we validate routing state?
- An in-class activity.

Disclaimer

- There are an endless number of possible routing solutions.

- We’re going to talk about the “archetypal Internet”.
    - We make a number of assumptions – we’ll discuss them as we go along.
    - We’ll discuss some alternative approaches next week.

Disclaimer #2

🇺🇸 “Row-ting” vs. “Root-ing” 🇬🇧

“Row-ter” vs. “Root-er”

Recall from earlier, 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 earlier, Packets

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

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

Recall from earlier, Packets

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

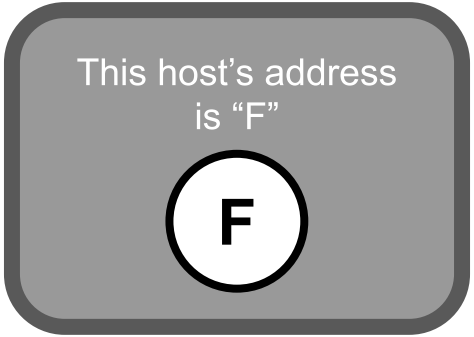
Metadata (headers)					Data/Payload
Src Addr	Dst Addr	Type	Version	...	<html><head><title>My Website</title><head> ...

Recall from earlier, Packets

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

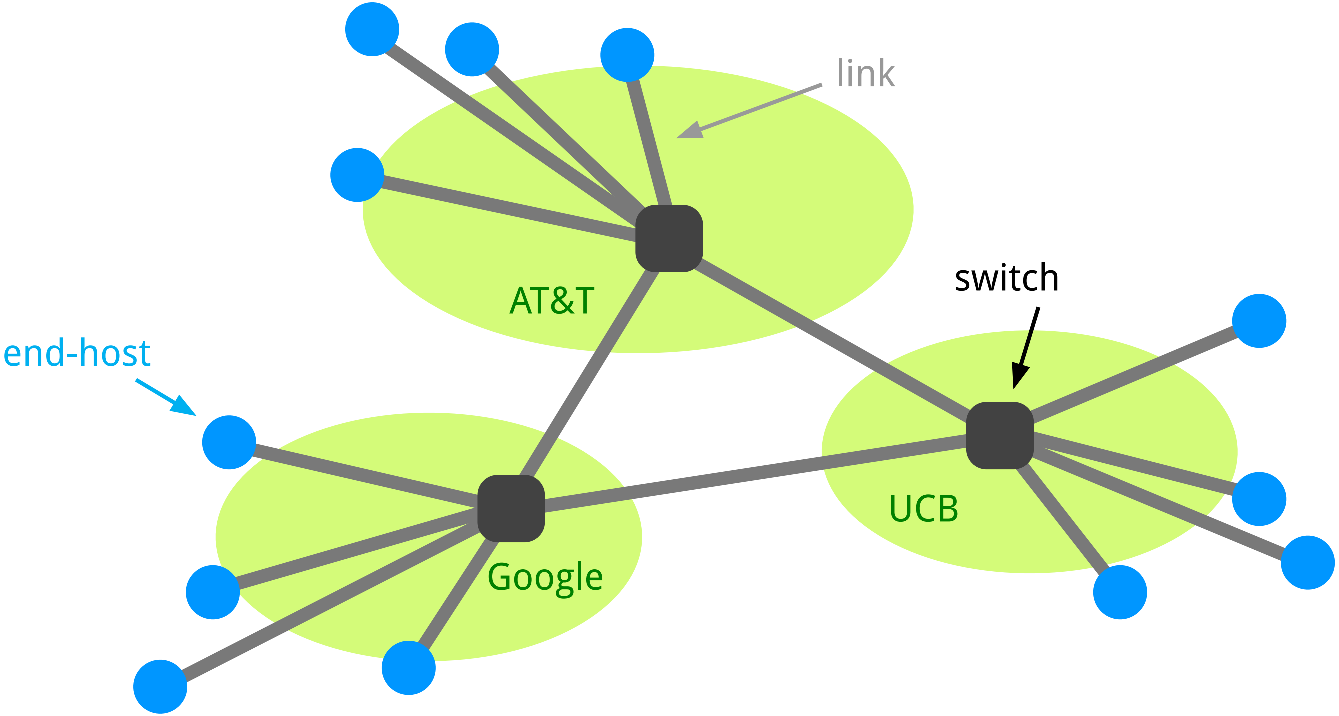
Metadata (headers)					Data/Payload
Src Addr	Dst Addr	Type	Version	...	<html><head><title>My Website</title><head> ...

Drawing Hosts…

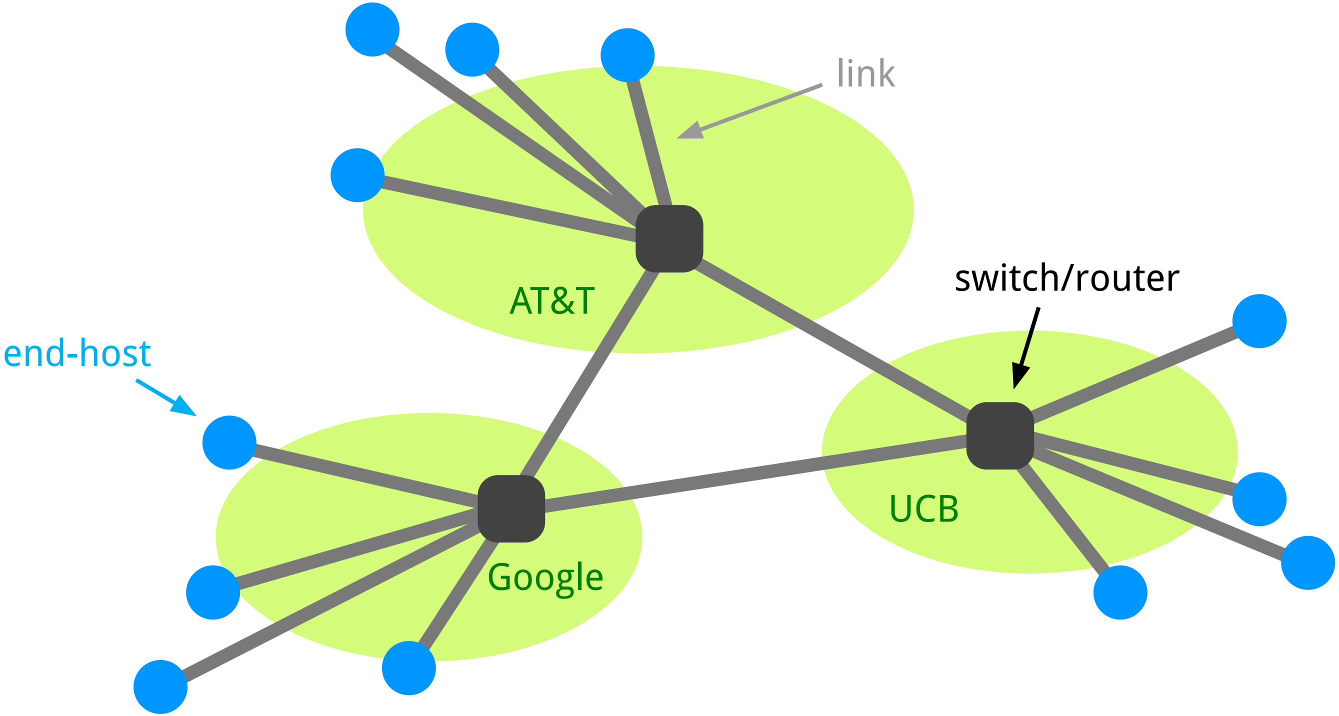


What is a router?

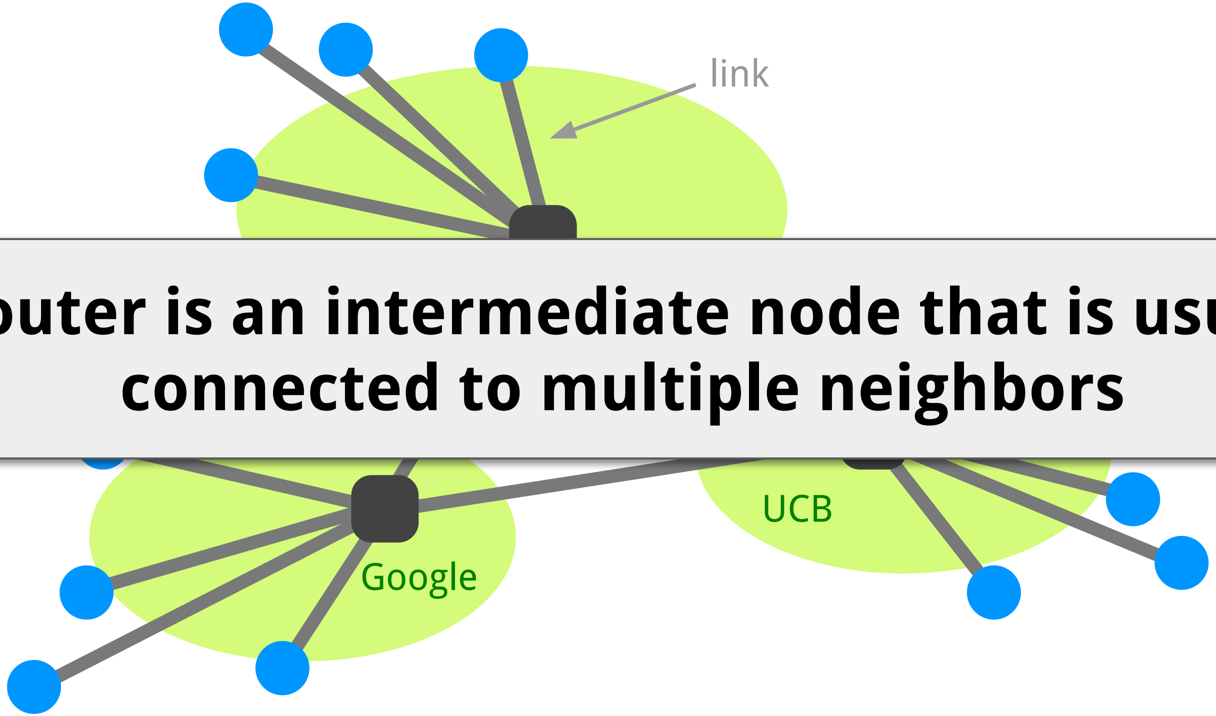
Recall from earlier…



Recall from earlier…

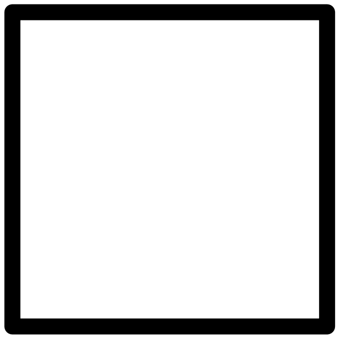


Recall from earlier…

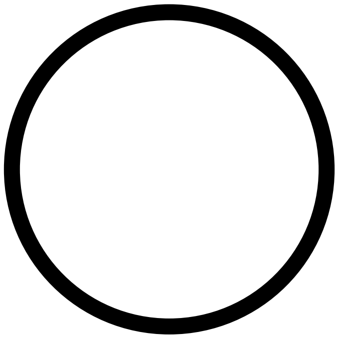


A router is an intermediate node that is usually connected to multiple neighbors

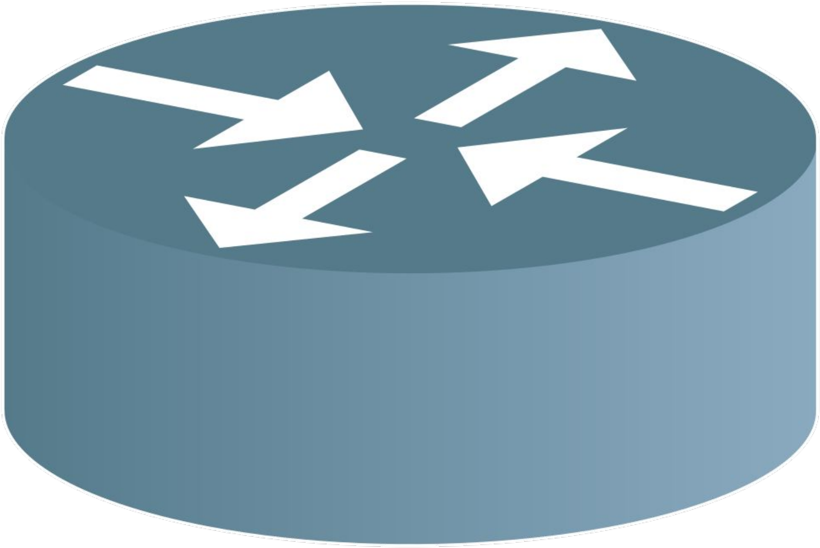
Drawing routers…



or



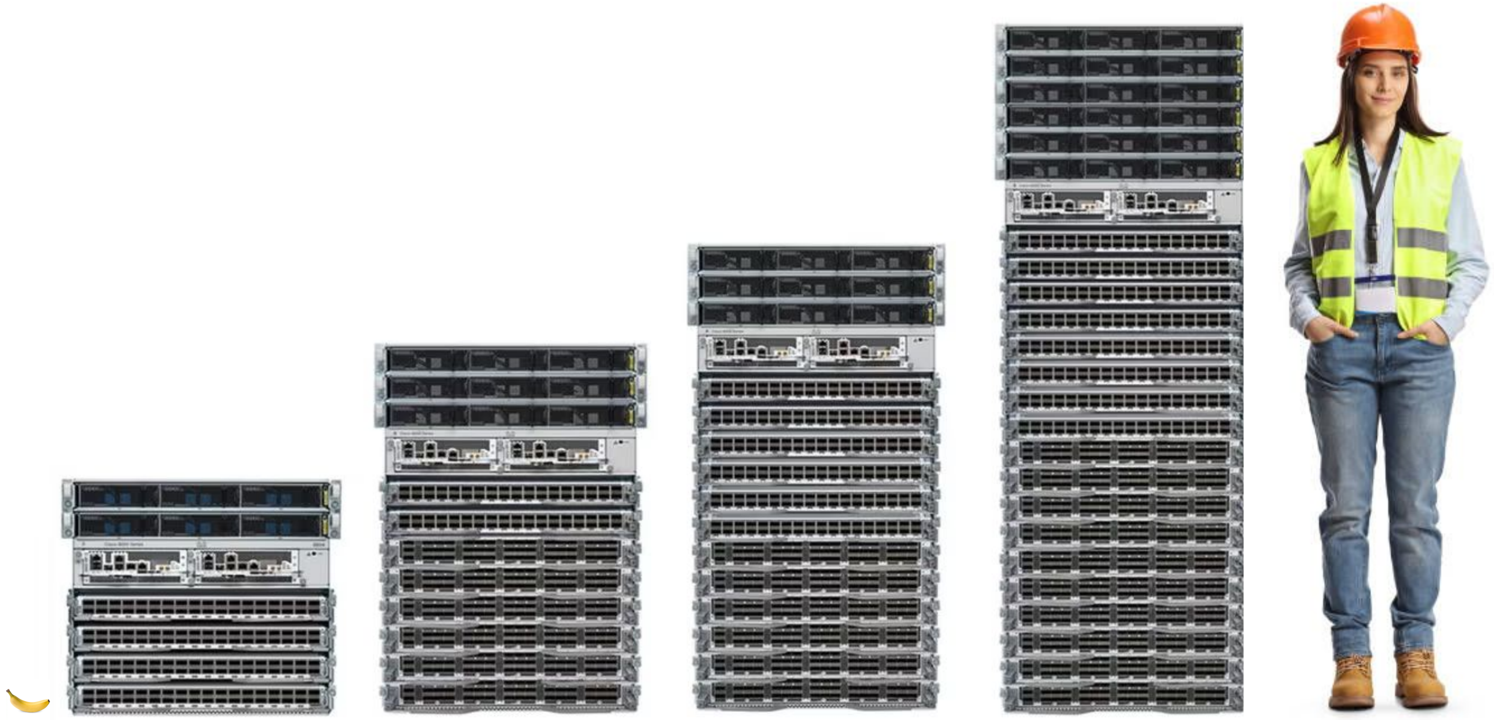
Drawing routers…



What is a router?

What is a router?

What is a router?

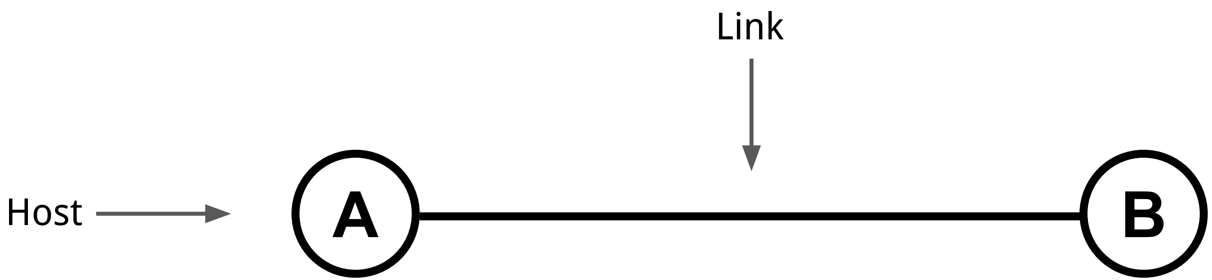


We'll come back to more routers in a later lecture.

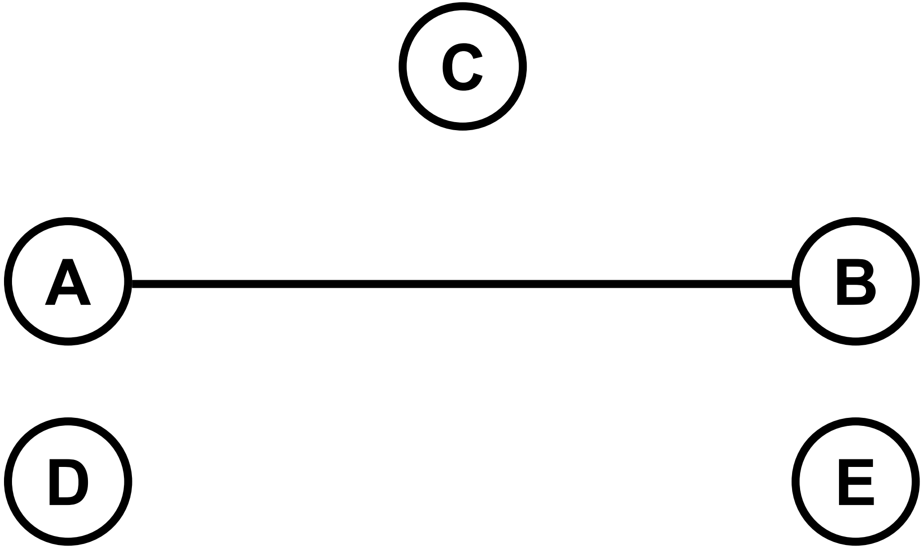
Why do we have routers?

Why do we have routers?

Why do we have routers?

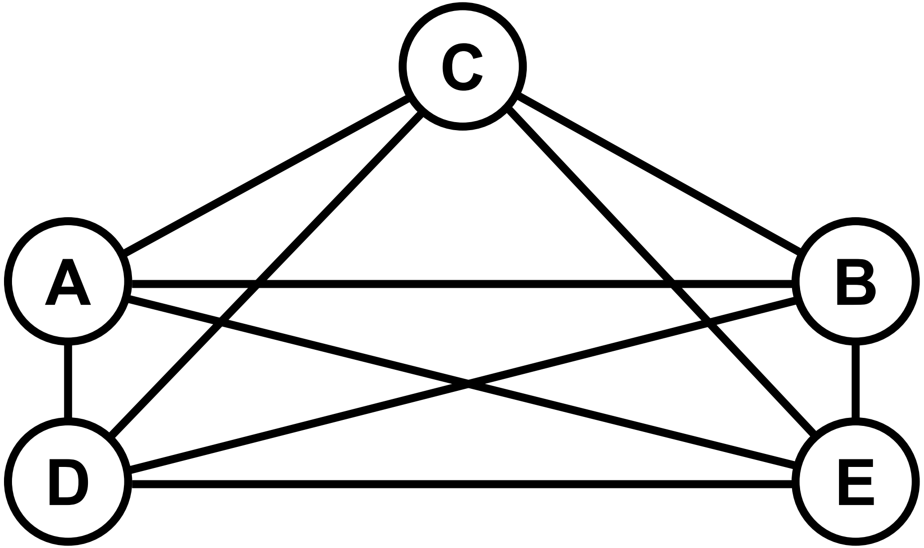


Why do we have routers?



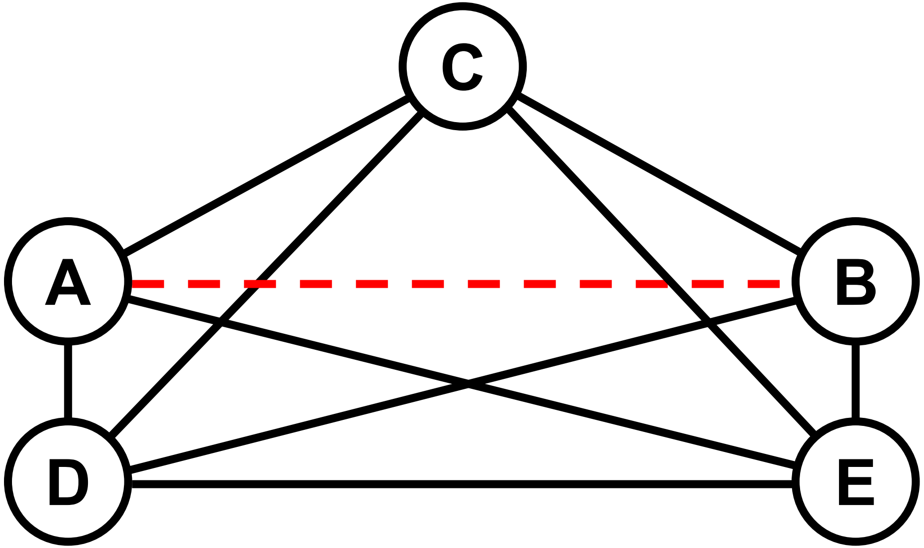
Now what?

Why do we have routers?

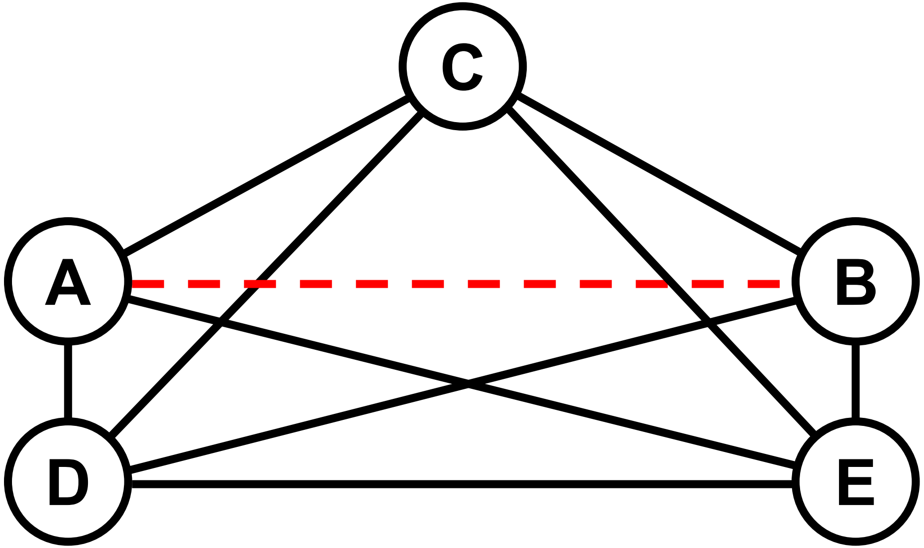


Is there a problem with this approach?

Why do we have routers?

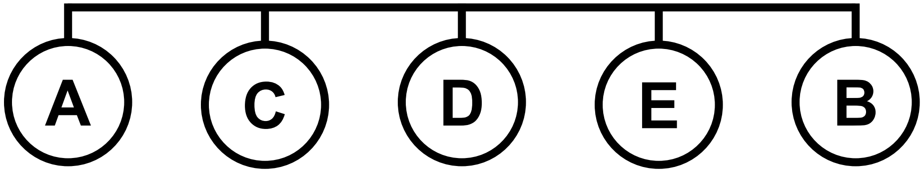


Why do we have routers?

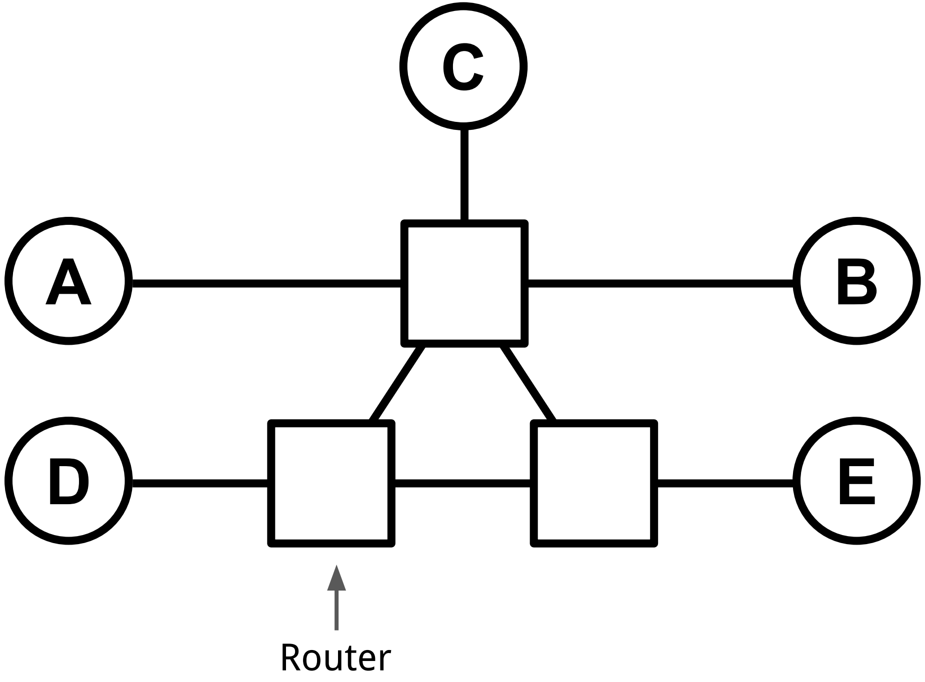


Is there anything *good* about this approach?

Why do we have routers?

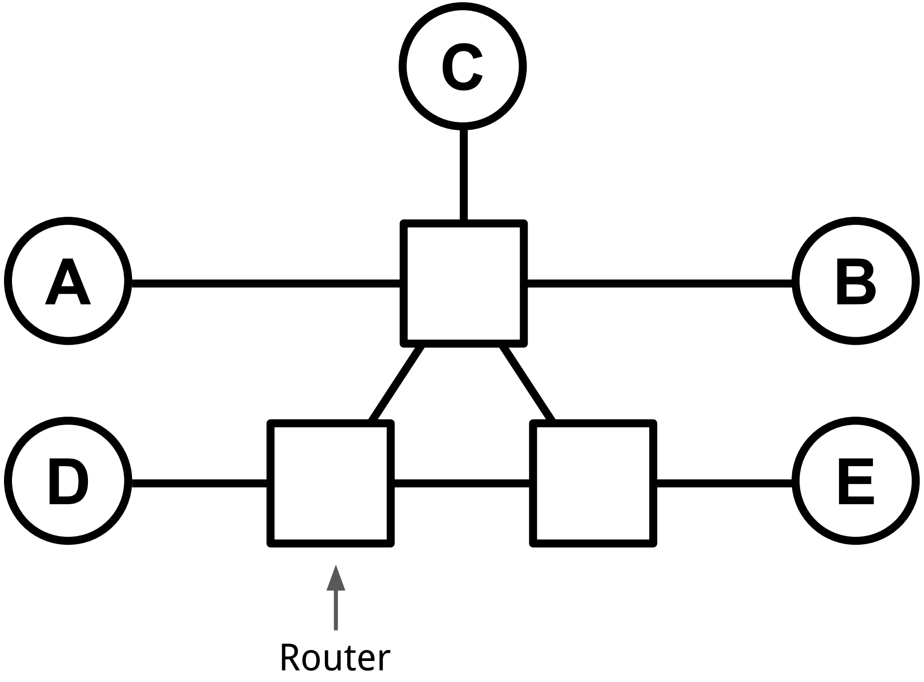


Why do we have routers?



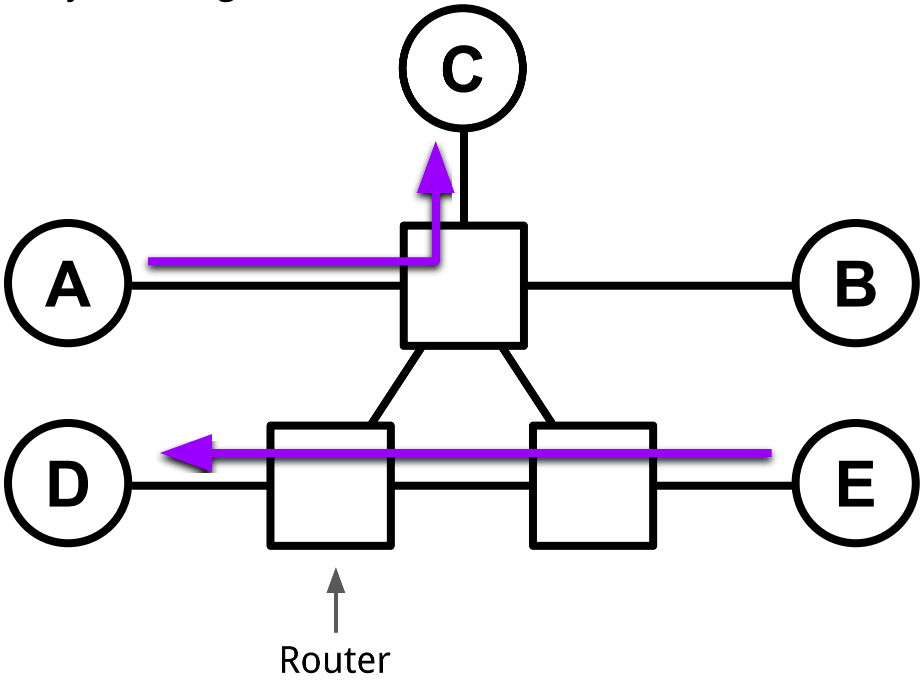
Why do we have routers?

- Way fewer links than a full mesh!



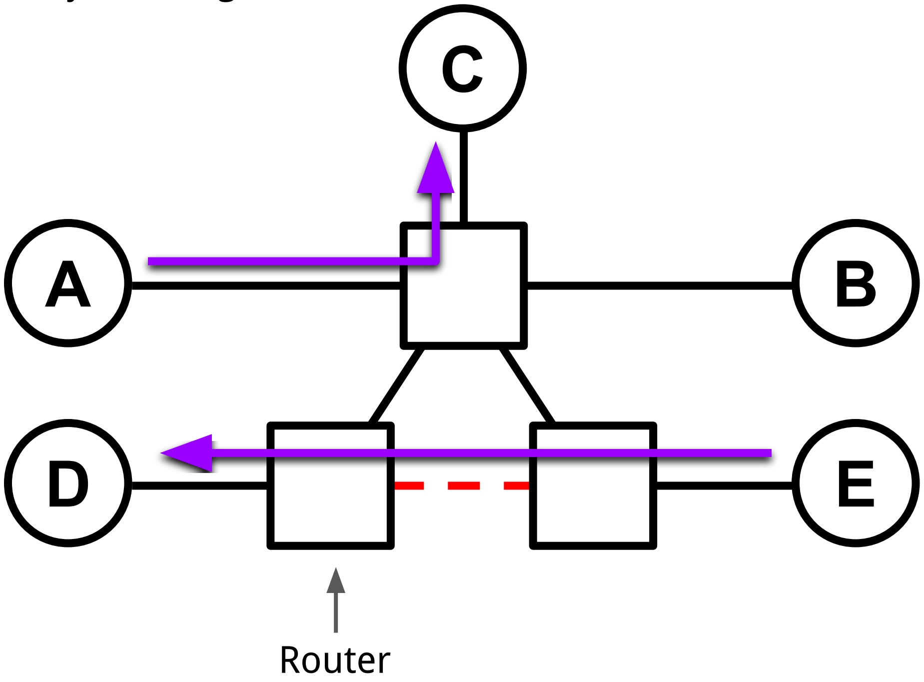
Why do we have routers?

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



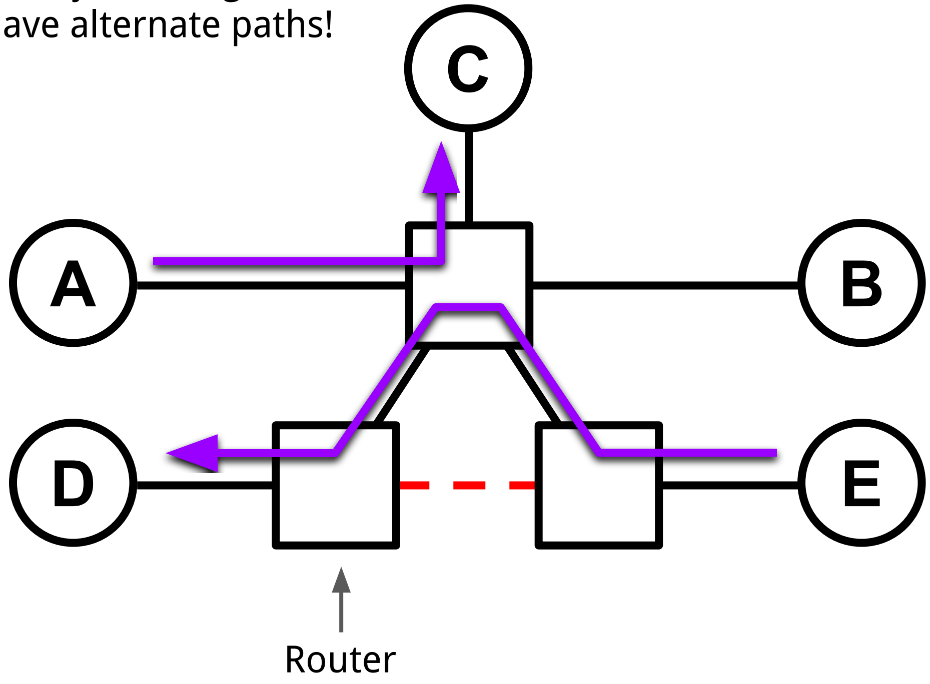
Why do we have routers?

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



Why do we have routers?

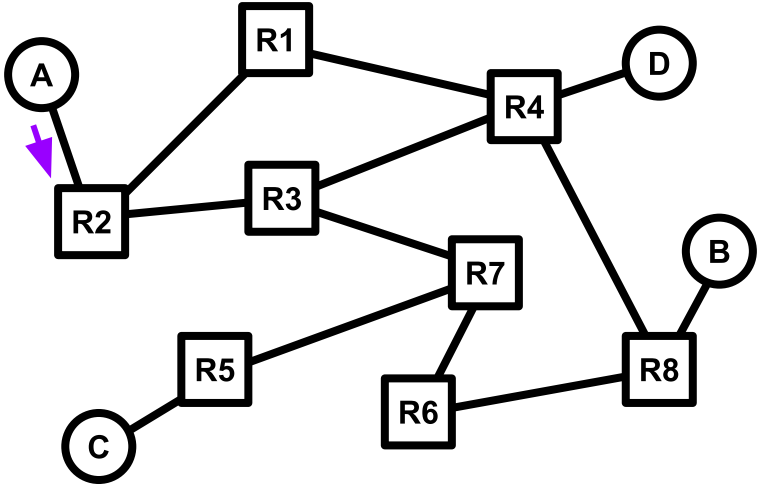
- Way fewer links than a full mesh!
- But more capacity than just a single link!
- With the ability to have alternate paths!



The challenge of routing

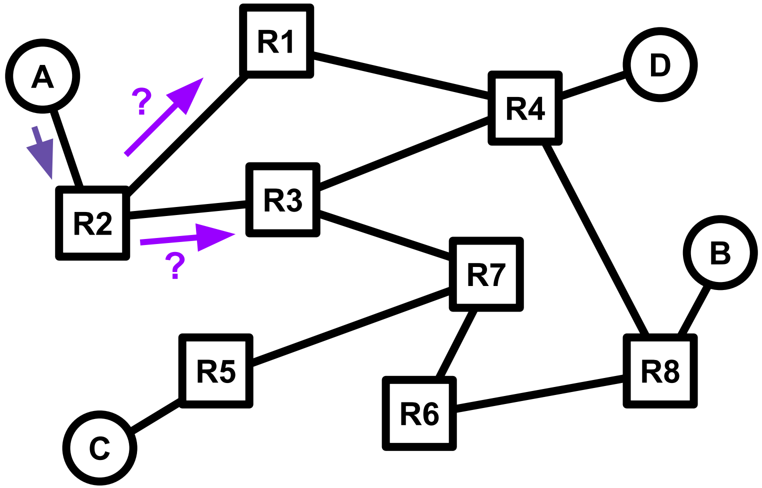
The Challenge of Routing

- 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

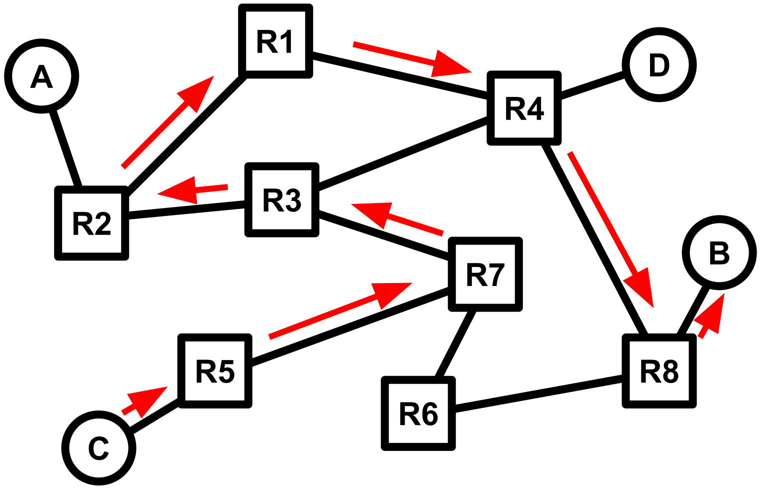
- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?



The Challenge of Routing

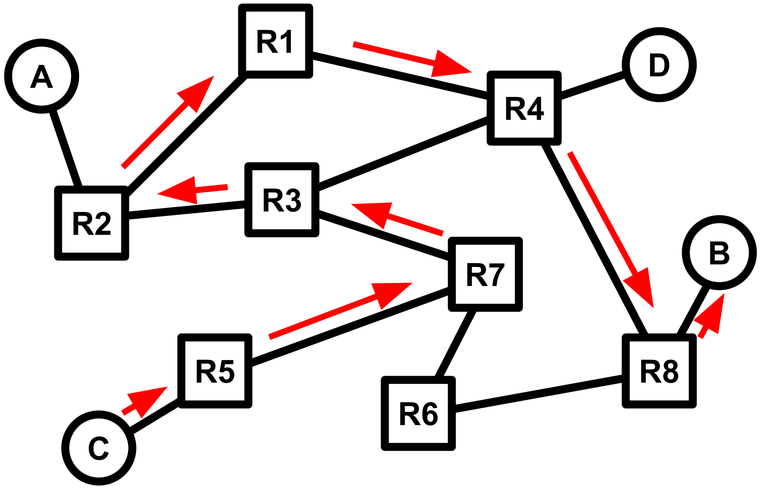
- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?

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



The Challenge of Routing

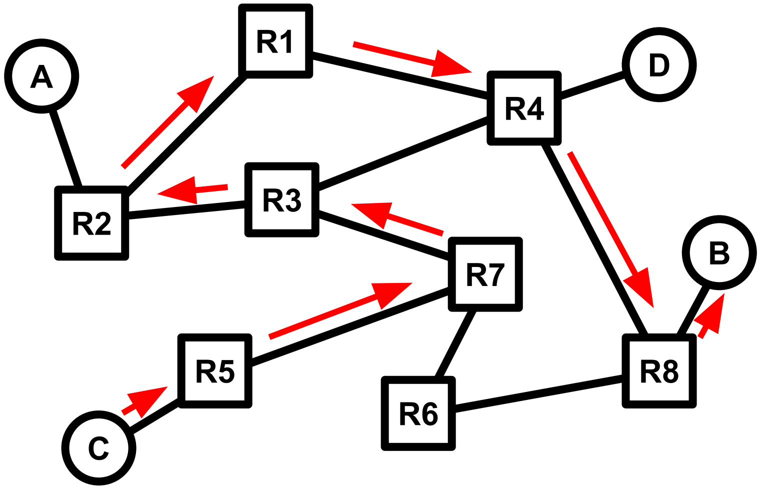
- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?
- We want to find ***paths*** which are *"**good**"*
  - "Good" may have many meanings (e.g., short)
  - Not random routing



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)
  - Not random routing
  - Not just sending a packet to everyone

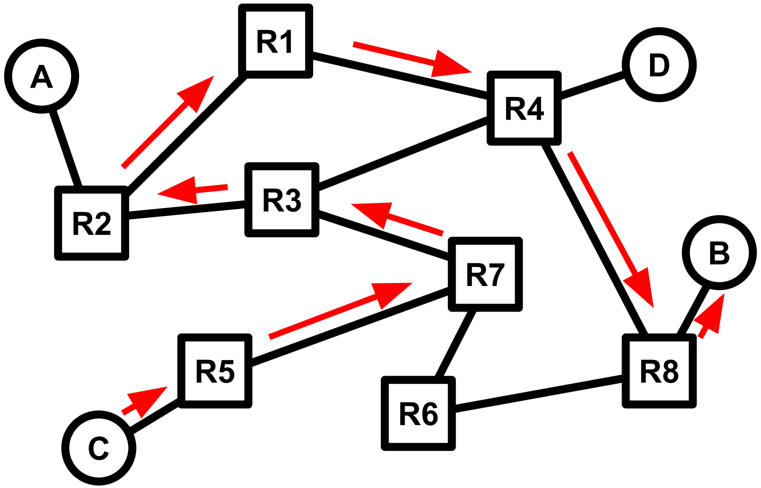


The Challenge of Routing

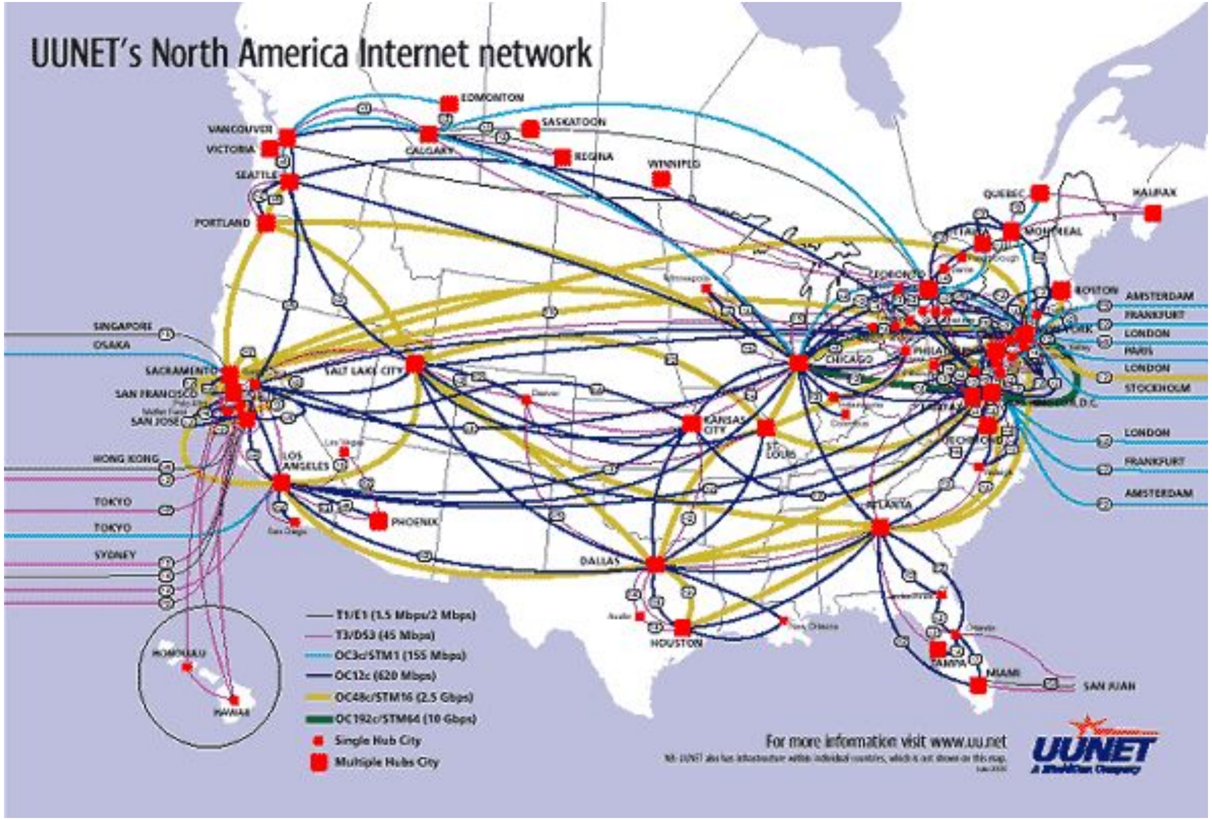
- The basic challenge:
  - When a packet arrives at a router, how does the router know where to send it next such that it will eventually arrive at the desired destination?

- We want to find ***paths*** which are *"**good**"*
  - "Good" may have many meanings (e.g., short)
  - Not random routing
  - Not just sending a packet to everyone

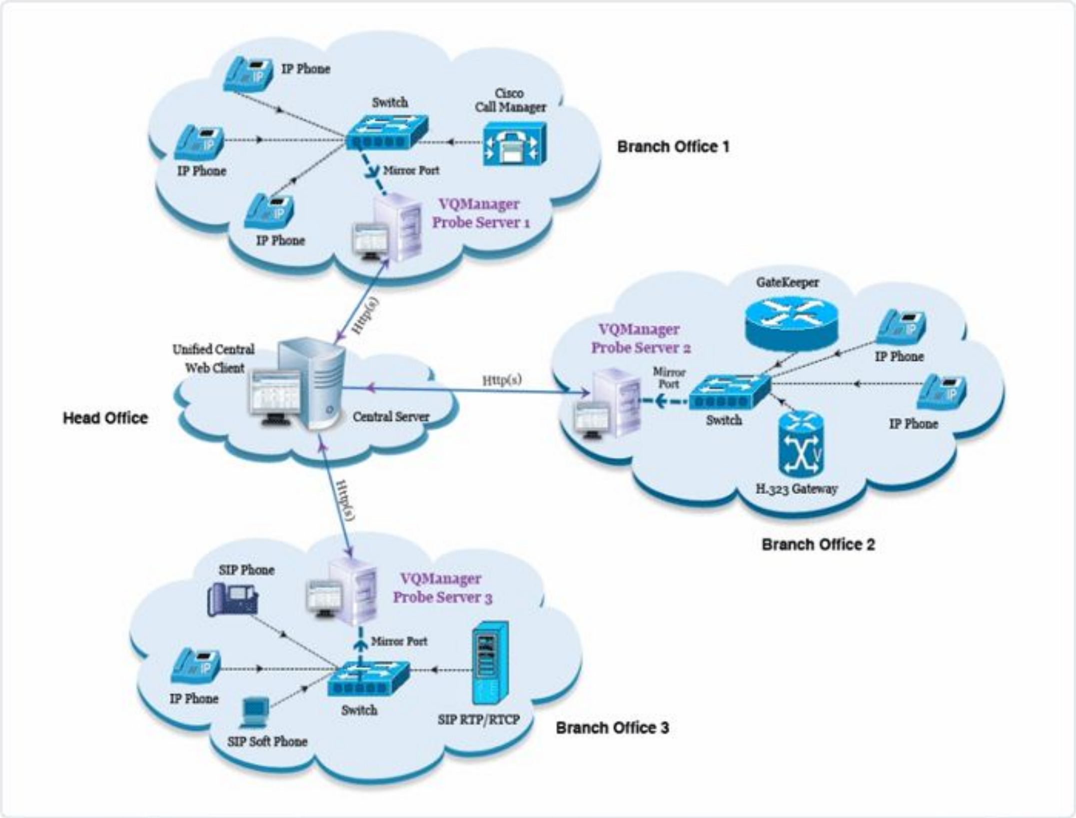
- We want it to adapt to arbitrary topologies.
  - The "graph" (topology) describing a network can vary a lot.



UUNET’s North American Network

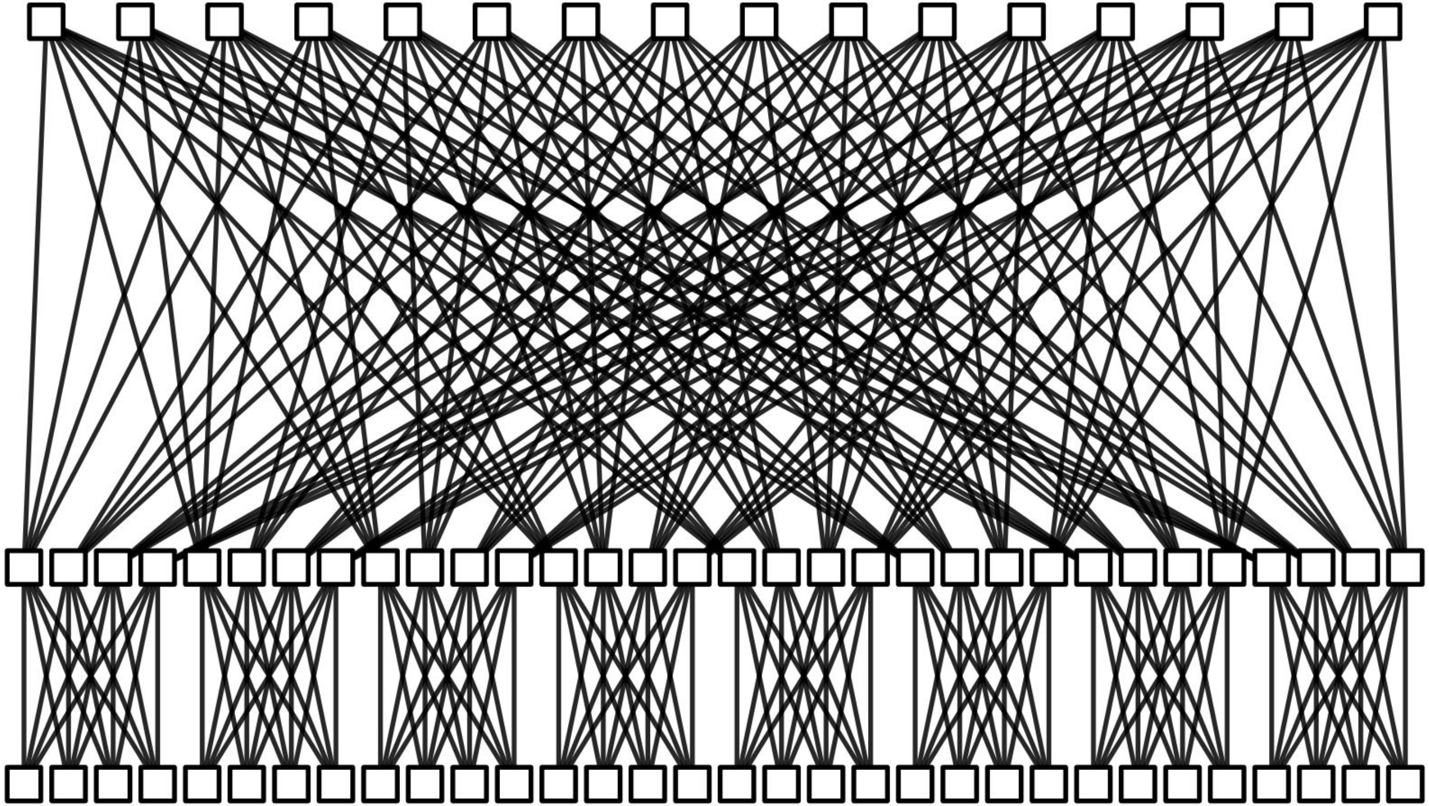


Enterprise Networks

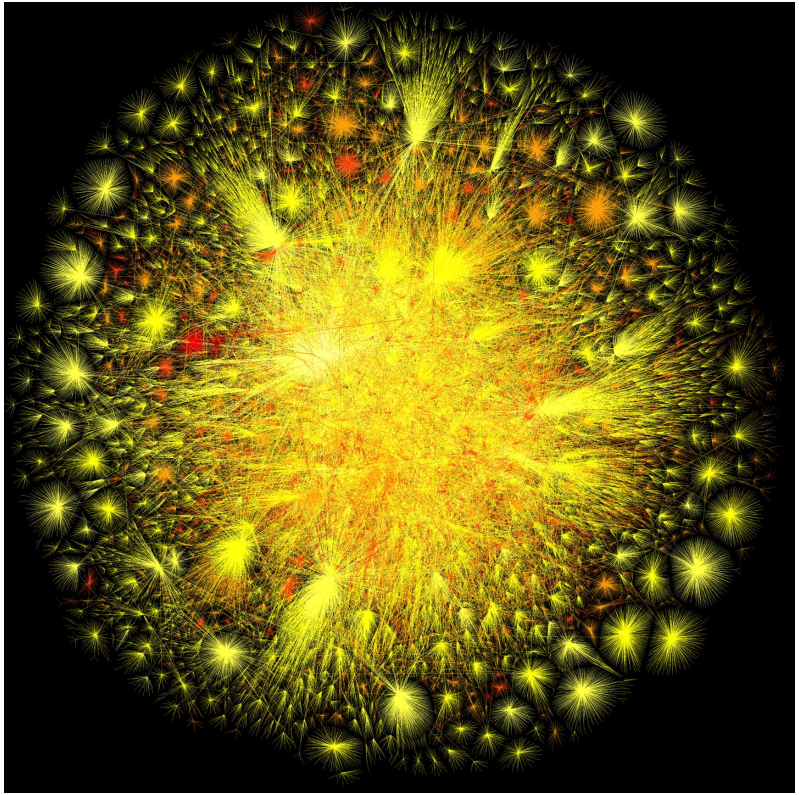


CenturyLink Domestic US Network

A small data center topology



The Internet

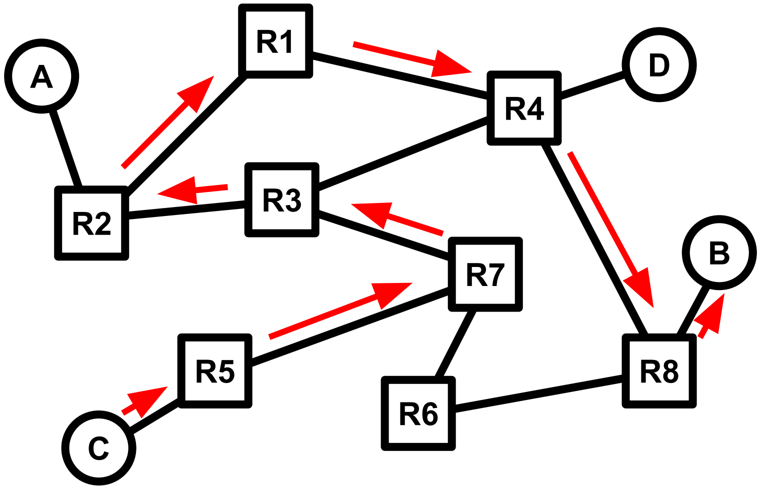


The 2010 Internet

Zoomed in for detail

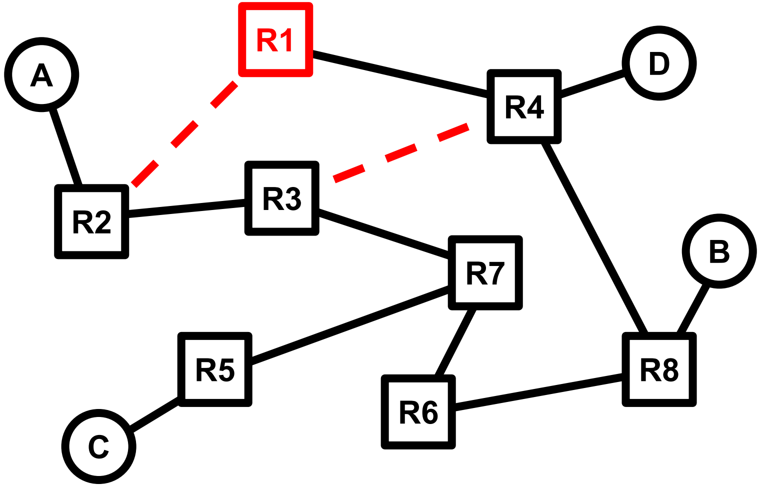
The Challenge of Routing

- 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)
  - Not random routing
  - Not just sending a packet to everyone
- We want it to adapt to arbitrary topologies.
  - The “graph” (topology) describing a network can vary a lot.
  - Different networks use different routing.
  - Every topology is dynamic (why?)



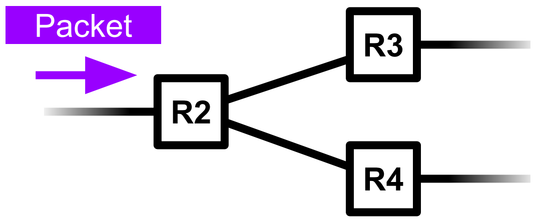
The Challenge of Routing

- 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)
  - Not random routing
  - Not just sending a packet to everyone
- We want it to adapt to arbitrary topologies.
  - The "graph" (topology) describing a network can vary a lot.
  - Different networks use different routing.
  - Every topology is dynamic.
    - More customers
    - Failures!

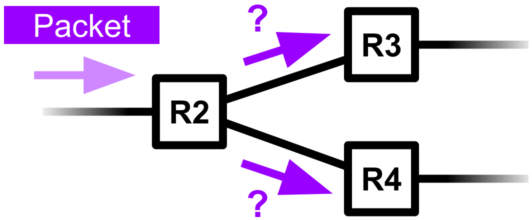


The Challenge of Forwarding

- When packet arrives...



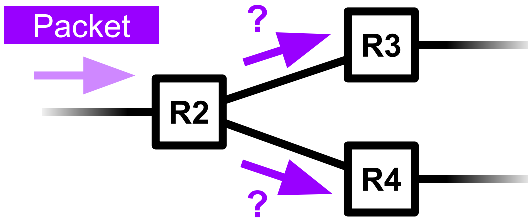
The Challenge of Forwarding



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

- You need to make the decision about which neighbor *fast* (~nanoseconds)
- Implies the decision process is *simple*

The Challenge of Forwarding

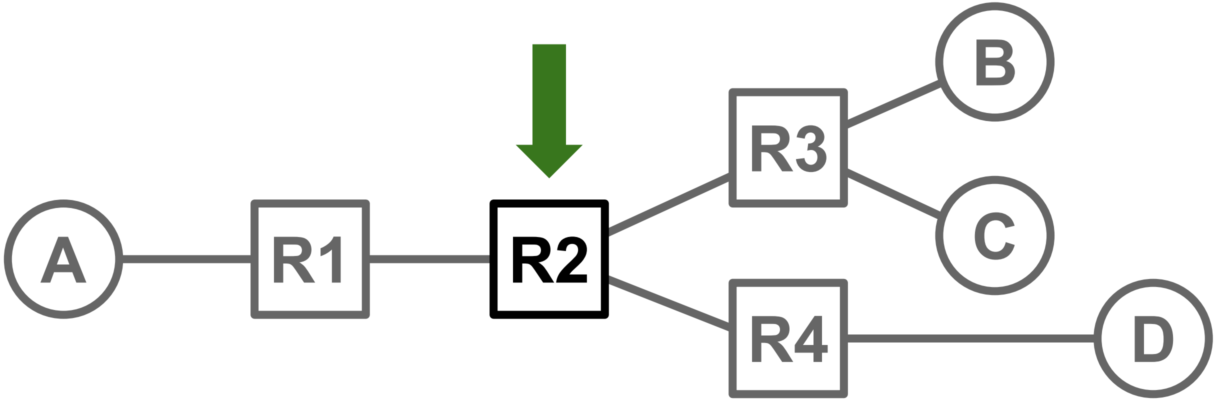


- When packet arrives, router ***forwards*** it to one of its neighbors

- You need to make the decision about which neighbor *fast* (~nanoseconds)
- Implies the decision process is *simple*

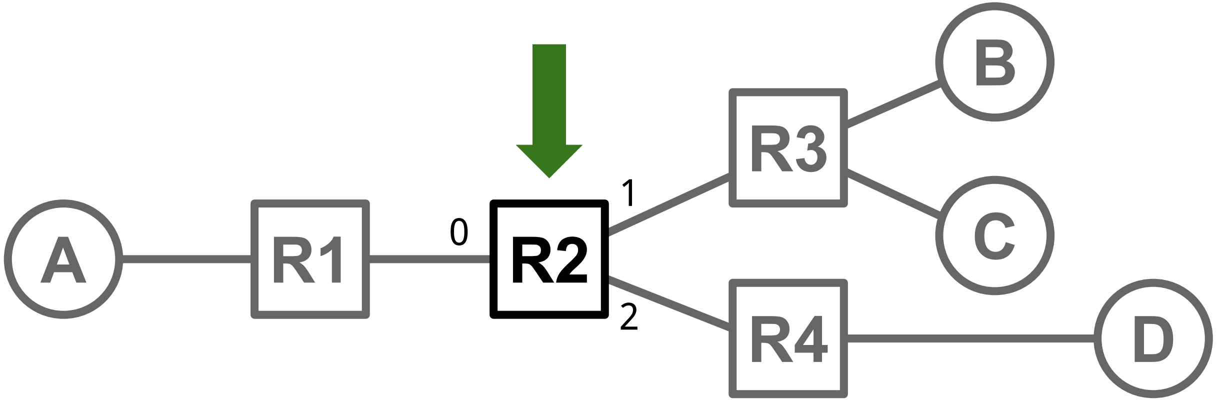
- Solution: Use a table

Forwarding with a Table



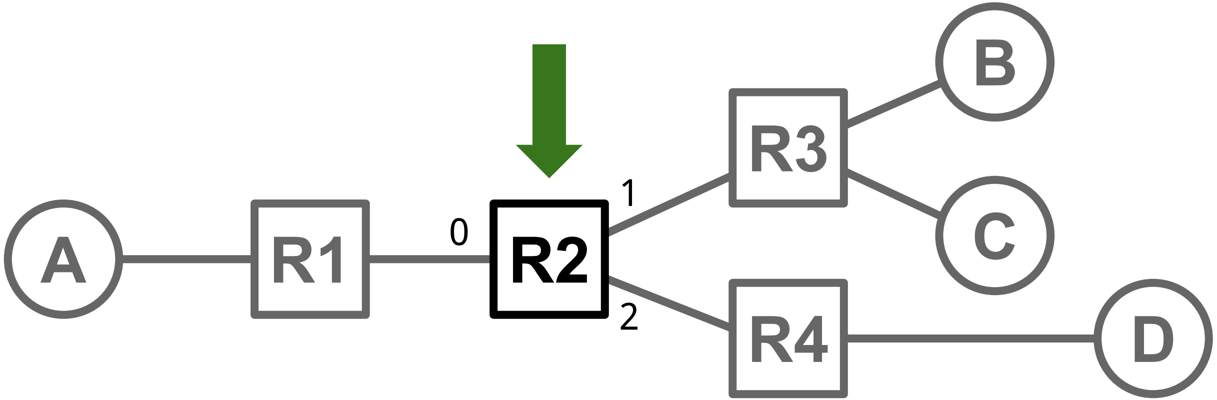
R2's Table	
Dst	***NextHop***
A	R1
B	R3
C	R3
D	R4

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
  - An "archetypal" Internet assumption (and basically the default)

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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor

Routing

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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor

Routing

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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor

- Inherently local - depends on arriving packet and local table.

Routing

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

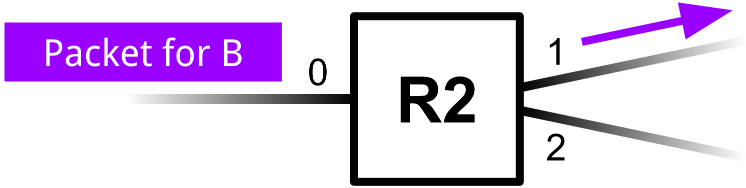
Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor
- Inherently local - depends on arriving packet and local table.

Routing

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



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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor

- Inherently local - depends on arriving packet and local table.

Routing

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

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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor

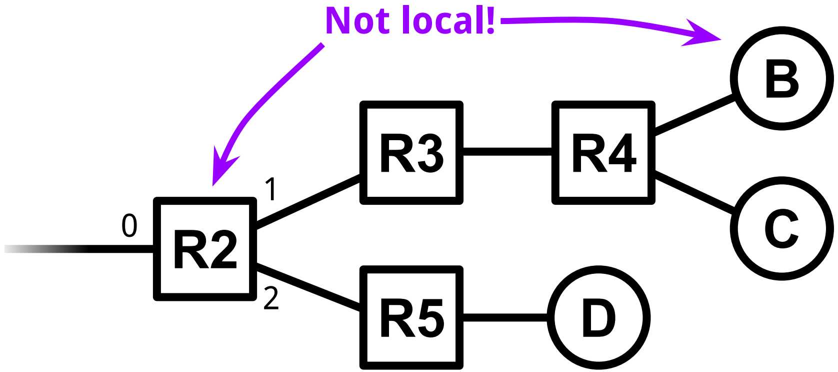
- Inherently local - depends on arriving packet and local table.

Routing

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

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

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



Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor
- Inherently local - depends on arriving packet and local table.
- Primary responsibility of the router's **data plane**

Routing

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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor
- Inherently local - depends on arriving packet and local table.
- Primary responsibility of the router's **data plane**

Routing

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

Routing + Forwarding: Definitions & Differences

Forwarding

- Look up packet's destination in table, and send packet to given neighbor
- Inherently local - depends on arriving packet and local table.
- Primary responsibility of the router's **data plane**
- Timescale: per packet arrival (ns?)

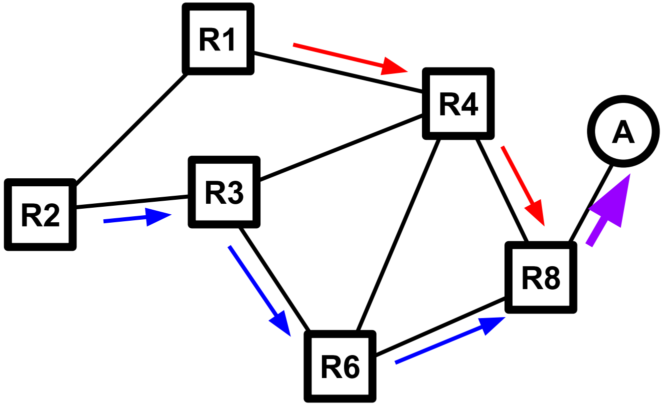
Routing

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

Graph representation and validity of routing state

"Delivery trees" for destination-based routing

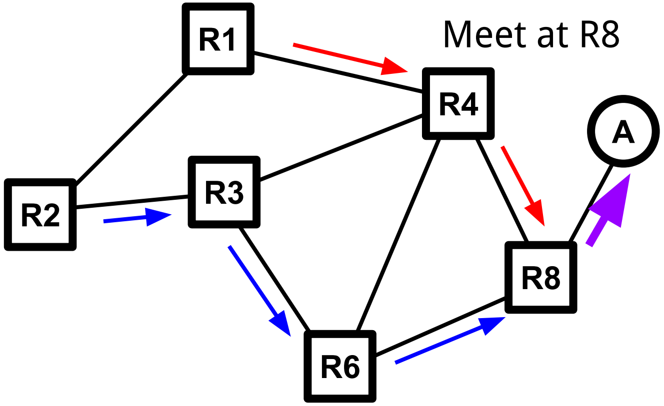
- We can graph paths packets take ***to a destination*** will take if they follow tables
- "Next hop" becomes an arrow

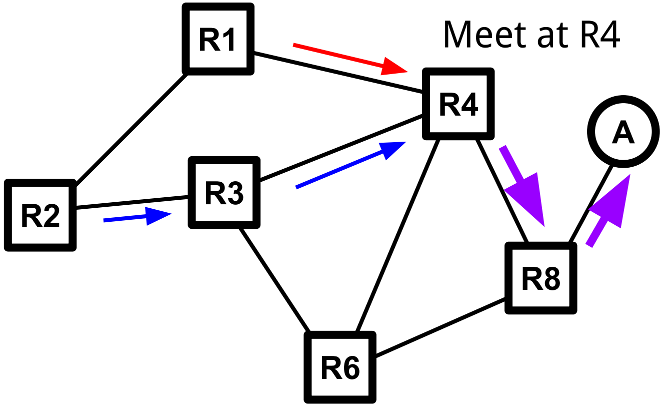


R6's Table	
Dst	***NextHop***
A	R8
…	

“Delivery trees” for destination-based routing

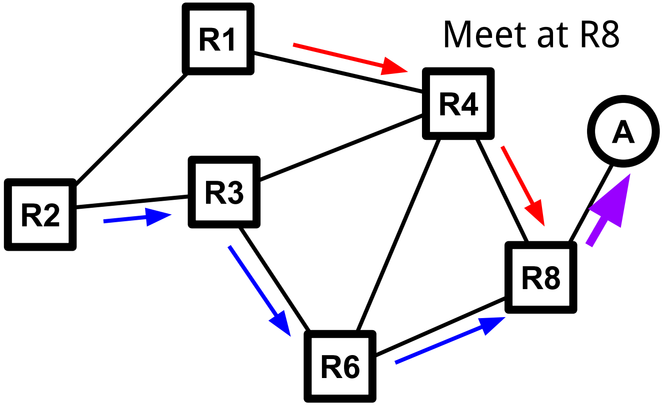
- We can graph paths packets take ***to a destination*** will take if they follow tables
- “Next hop” becomes an arrow
  - (Simplification) only one next hop per destination…
  - …means *only one outgoing arrow per node!*
  - …once paths meet, they never split!

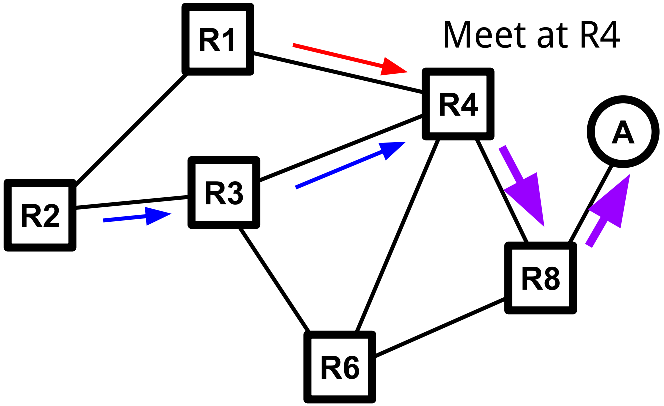




“Delivery trees” for destination-based routing

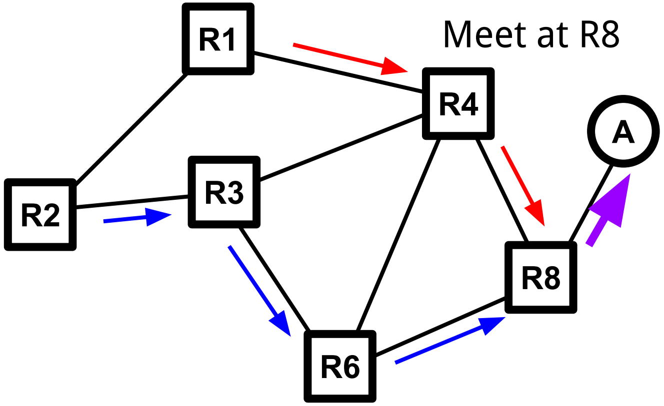
- Set of all paths create ***“directed delivery tree”***
  - *Must cover every node* (We want to be able to reach it from anywhere!)

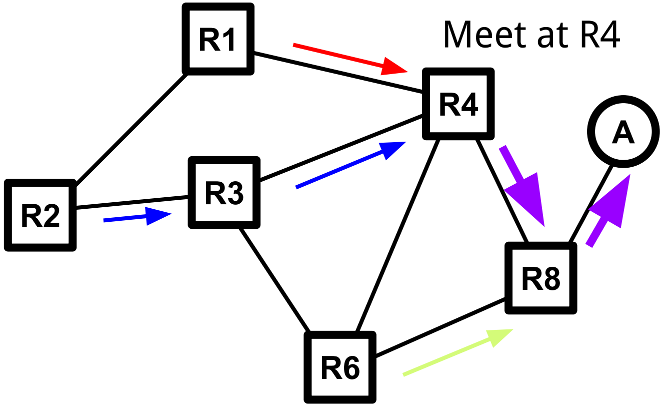




“Delivery trees” for destination-based routing

- 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





What is valid routing?

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

What is valid routing?

- 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***
    - (This term is used 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

- 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?

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**

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

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!

- 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

- 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*

Putting it to use: verifying routing state validity

- How do we apply the condition we discussed to check validity?

A couple of 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 of 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.

Putting it to use: verifying routing state validity

- Focus only on a single destination
    - Ignore all other hosts
    - Ignore all other routing state.

- For each router, mark outgoing port with arrow

  There can only be one at each node (destination-based)

Putting it to use: verifying routing state validity

- Focus only on a single destination
  - Ignore all other hosts
  - Ignore all other routing state.

- For each router, mark outgoing port with arrow

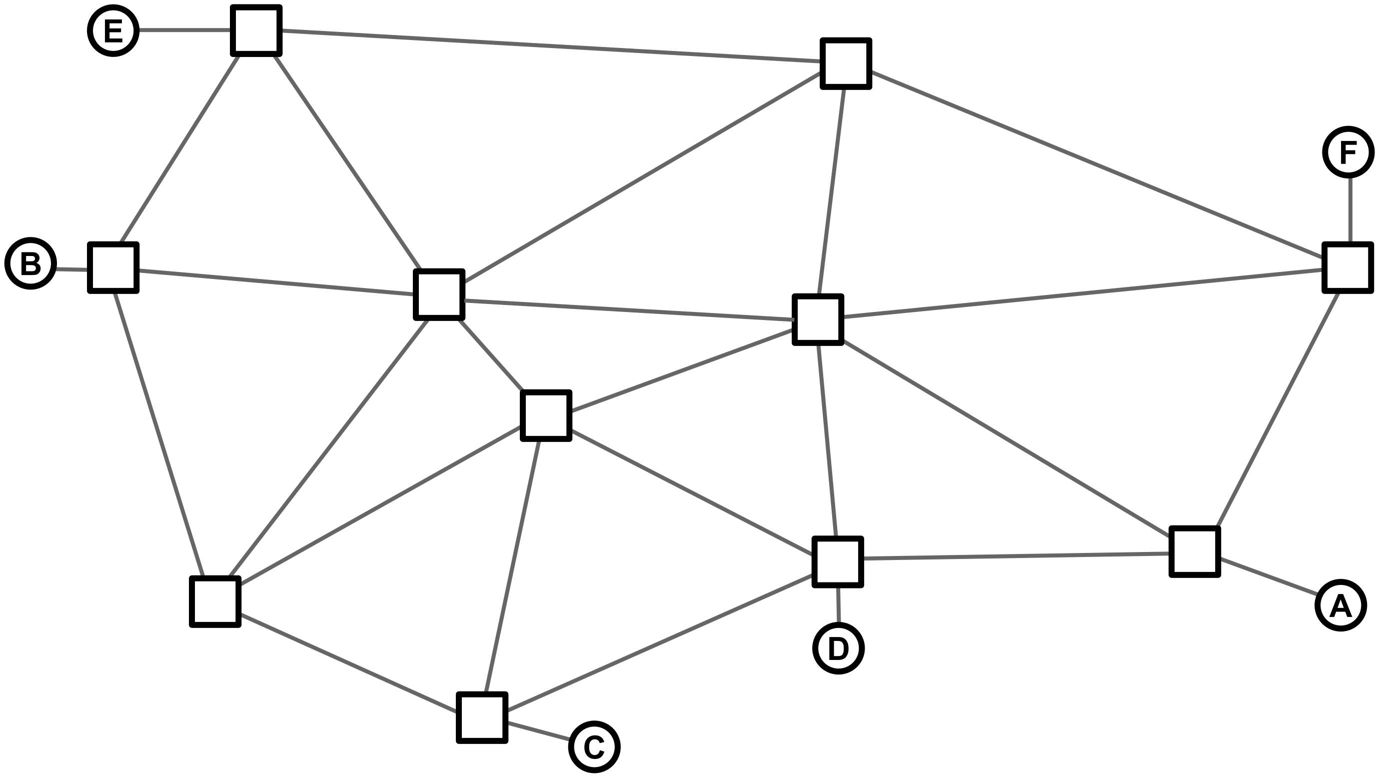
  There can only be one at each node (destination-based)

- Eliminate all links with no arrows

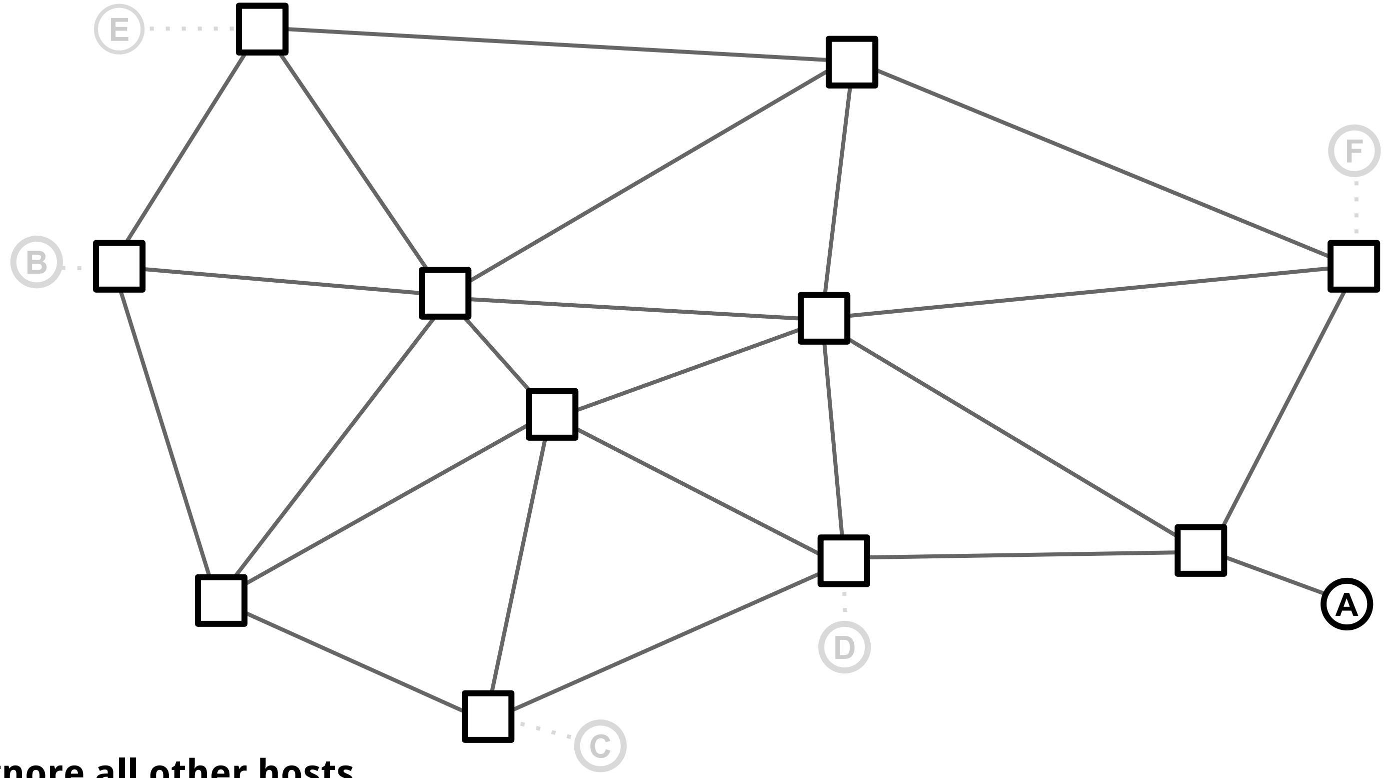
Putting it to use: verifying routing state validity

- Focus only on a single destination
  - Ignore all other hosts
  - Ignore all other routing state.
- For each router, mark outgoing port with arrow
  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**
  - Recall:
    - a directed spanning tree where all edges point toward destination

Checking validity of state to “A”

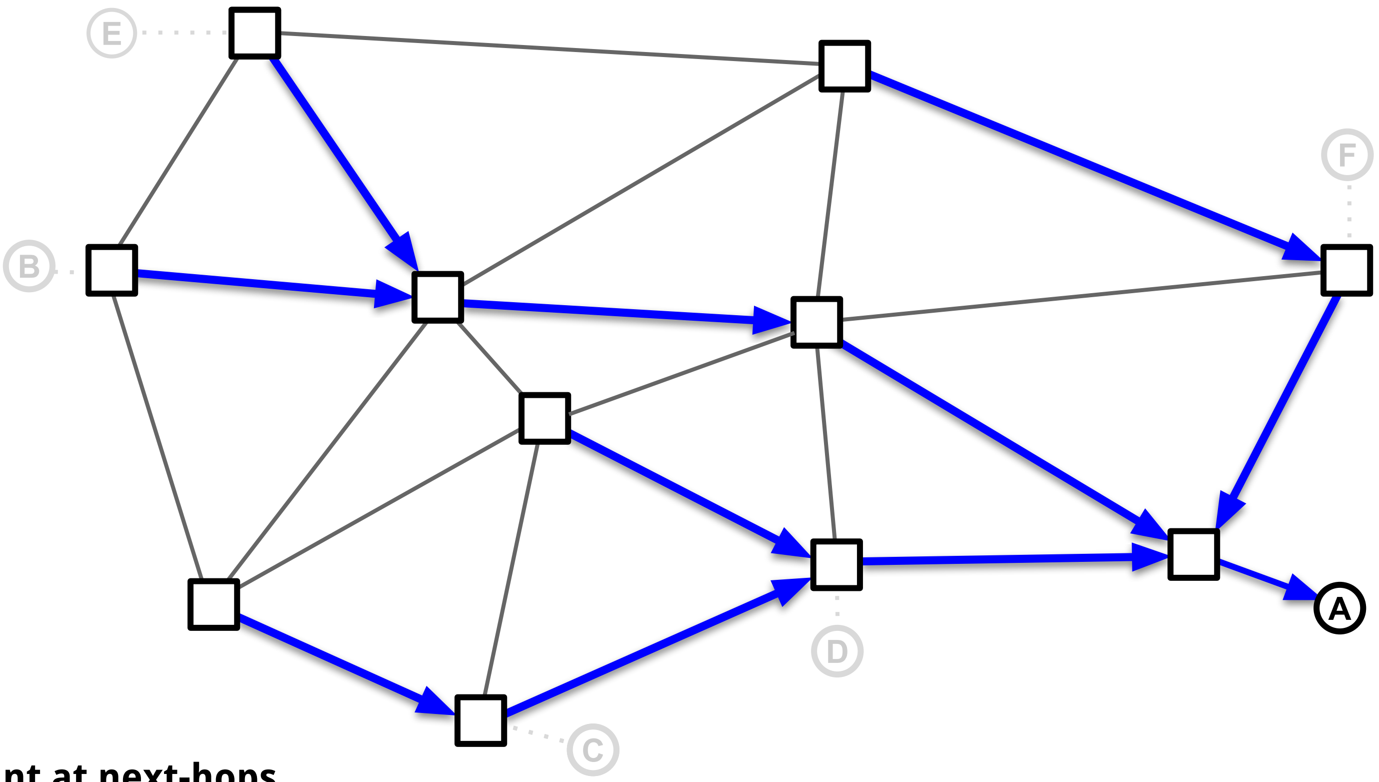


Checking validity of state to “A”



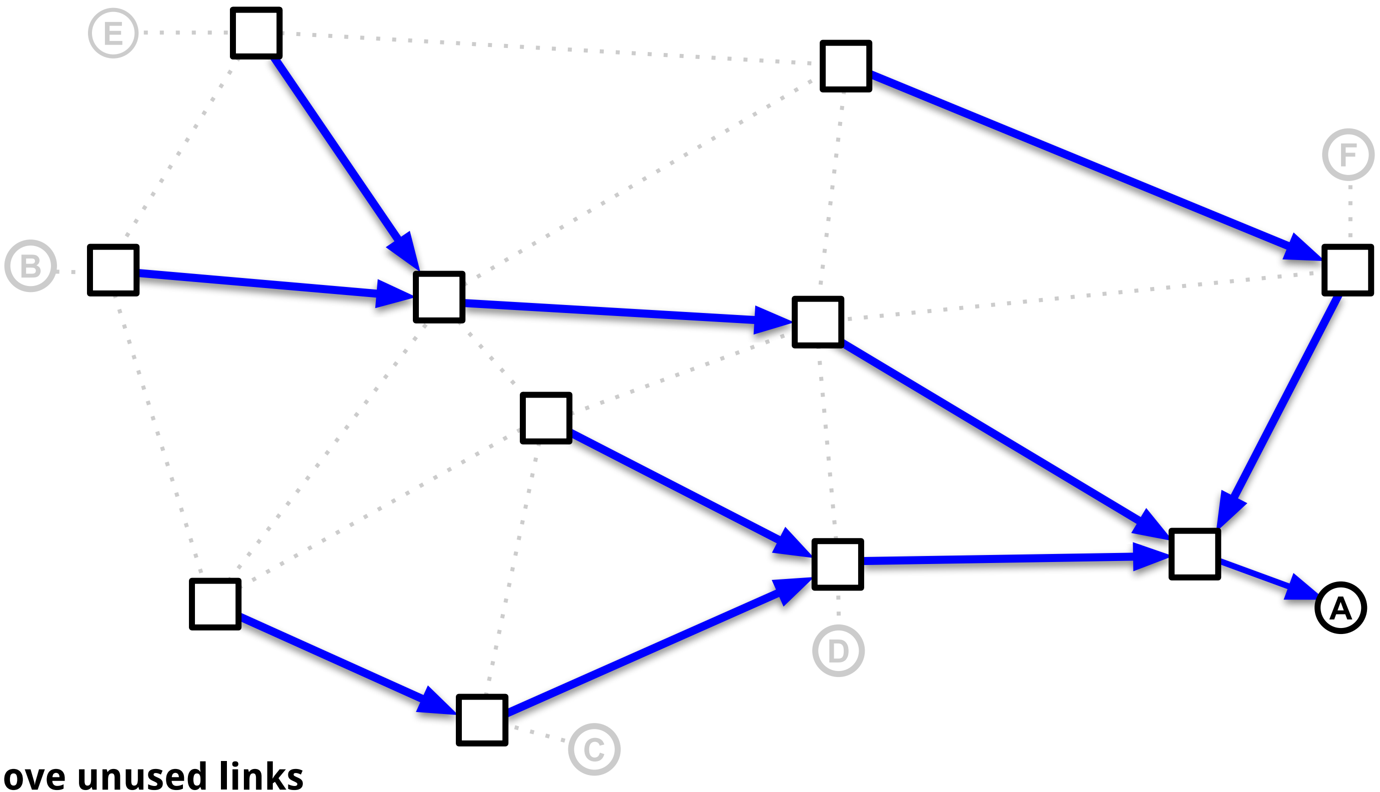
We’ll ignore all other hosts.

Checking validity of state to “A”



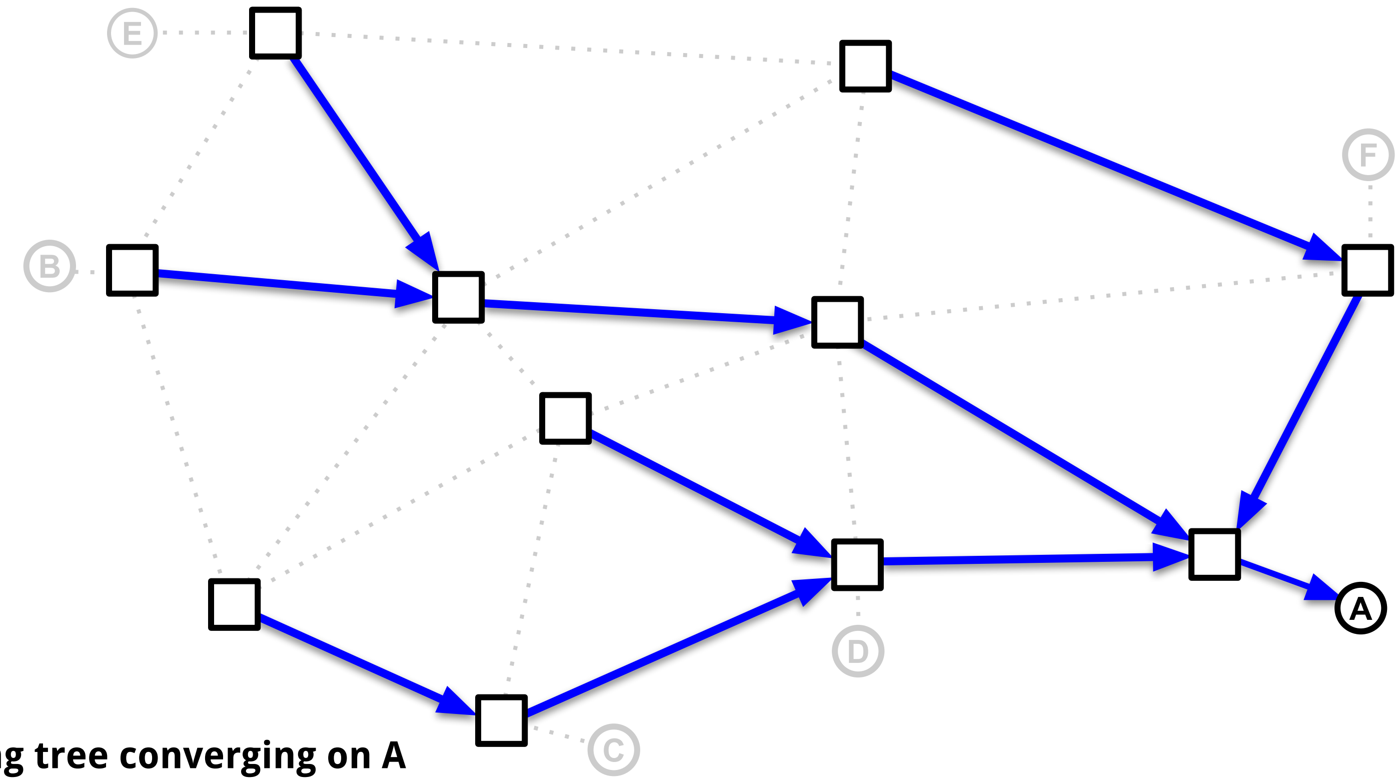
Point at next-hops

Checking validity of state to “A”



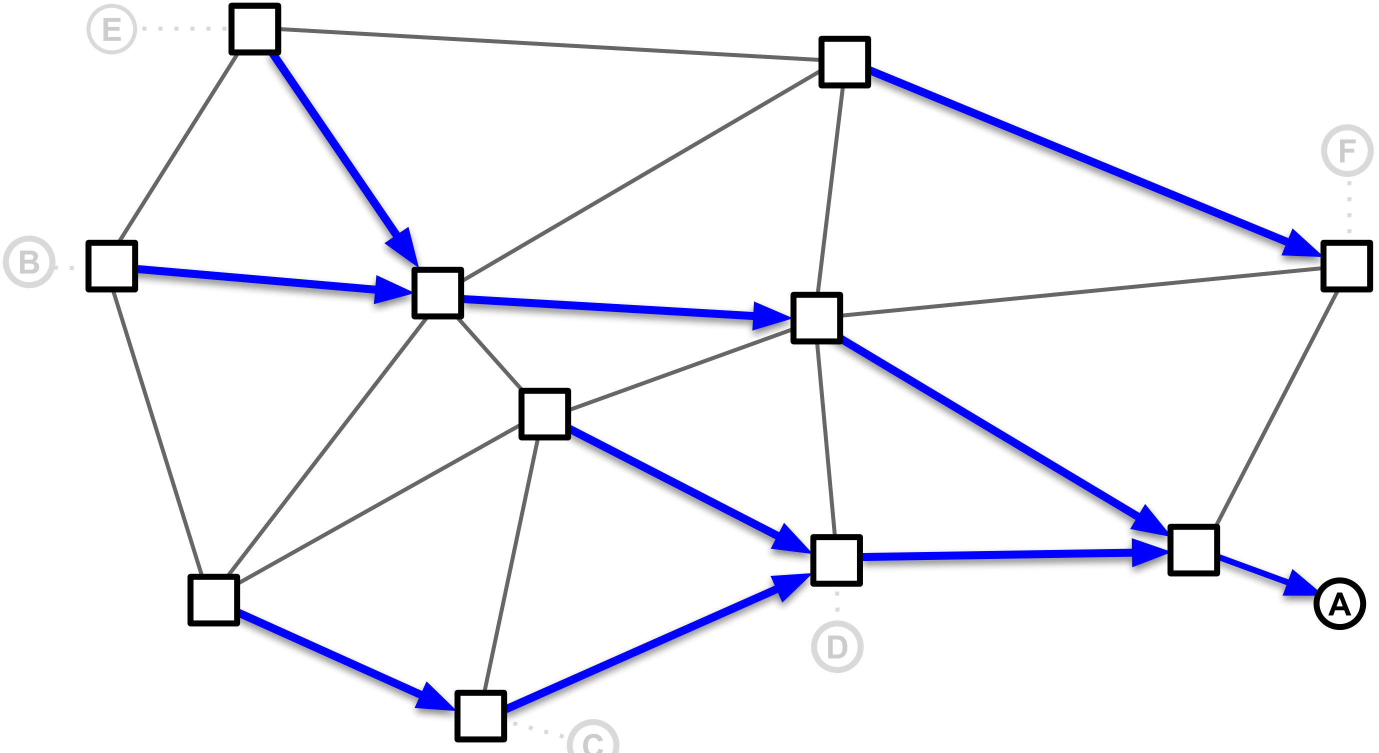
Remove unused links

Checking validity of state to “A”



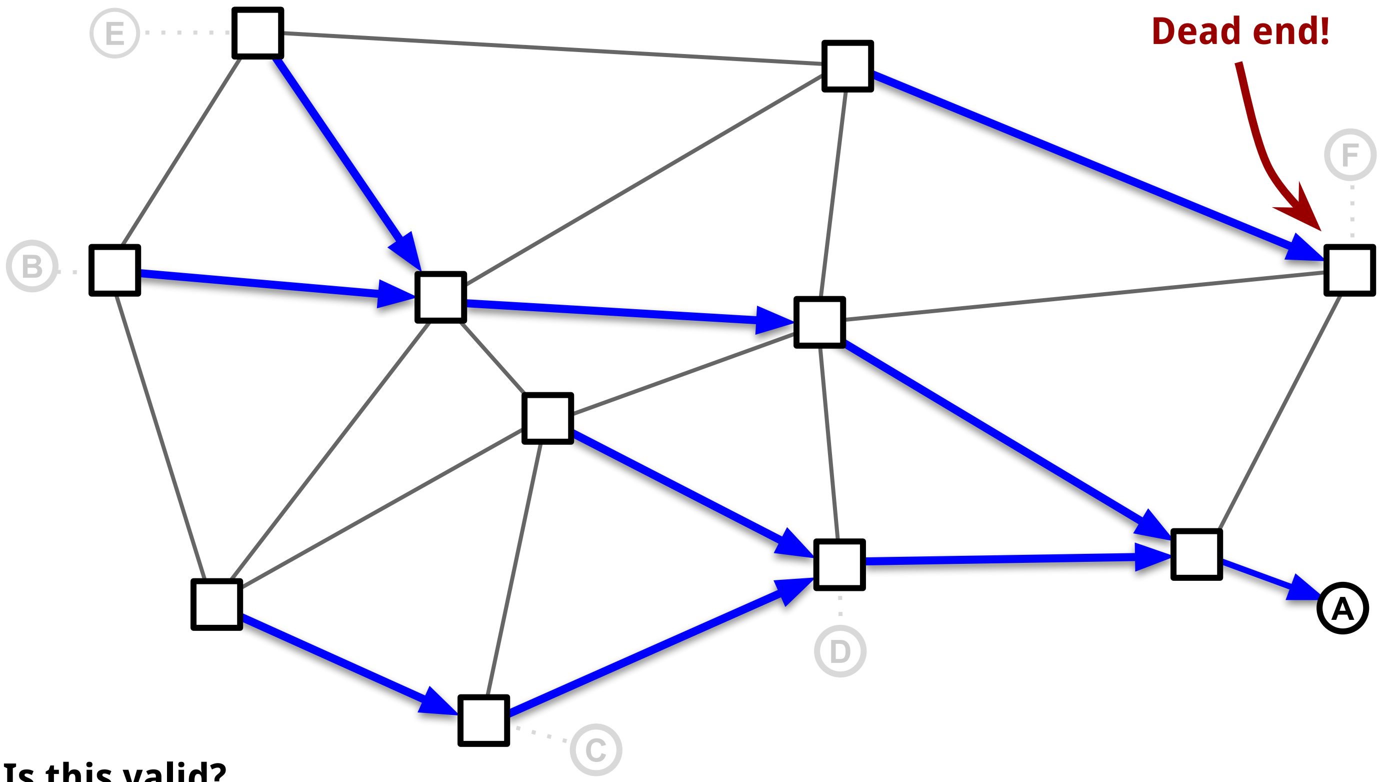
Spanning tree converging on A
=> it’s valid!

Alternate state for A



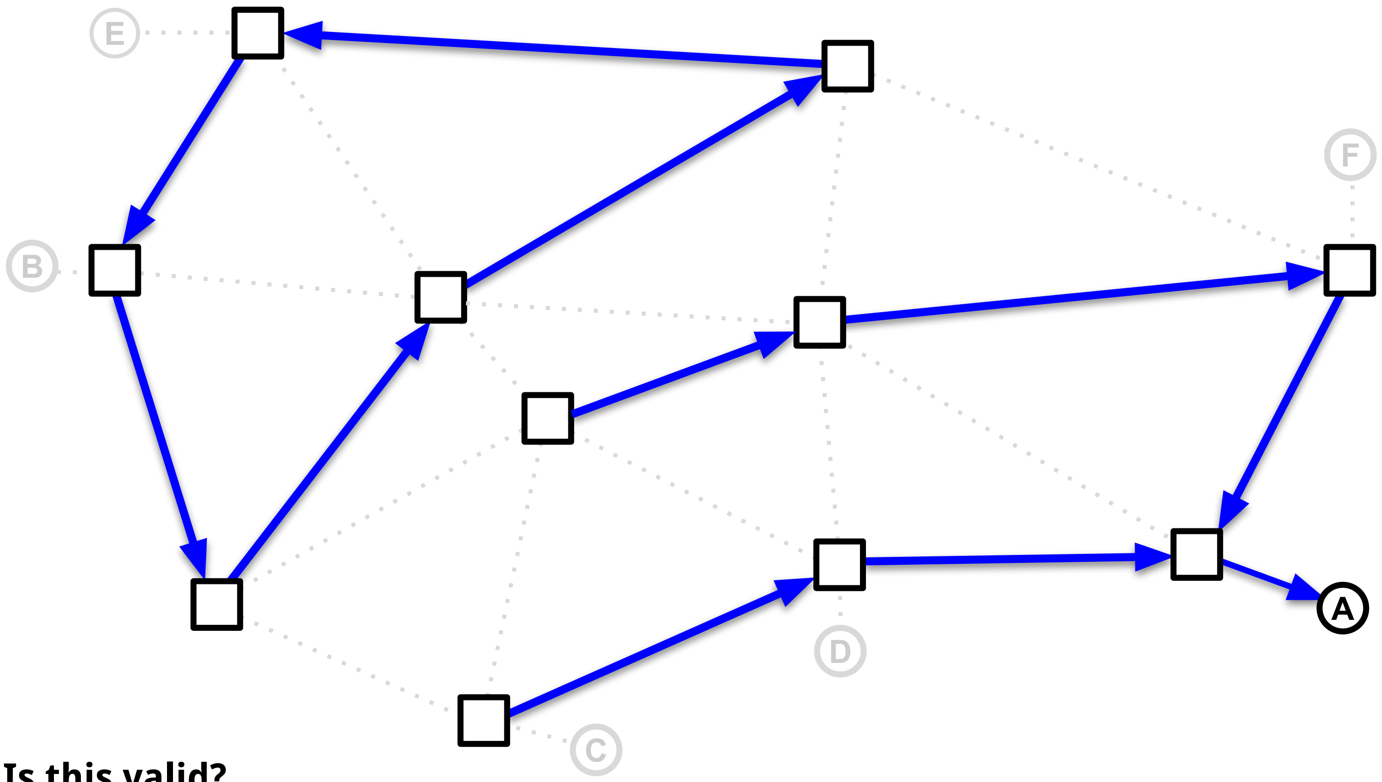
Is this valid?

Alternate state for A



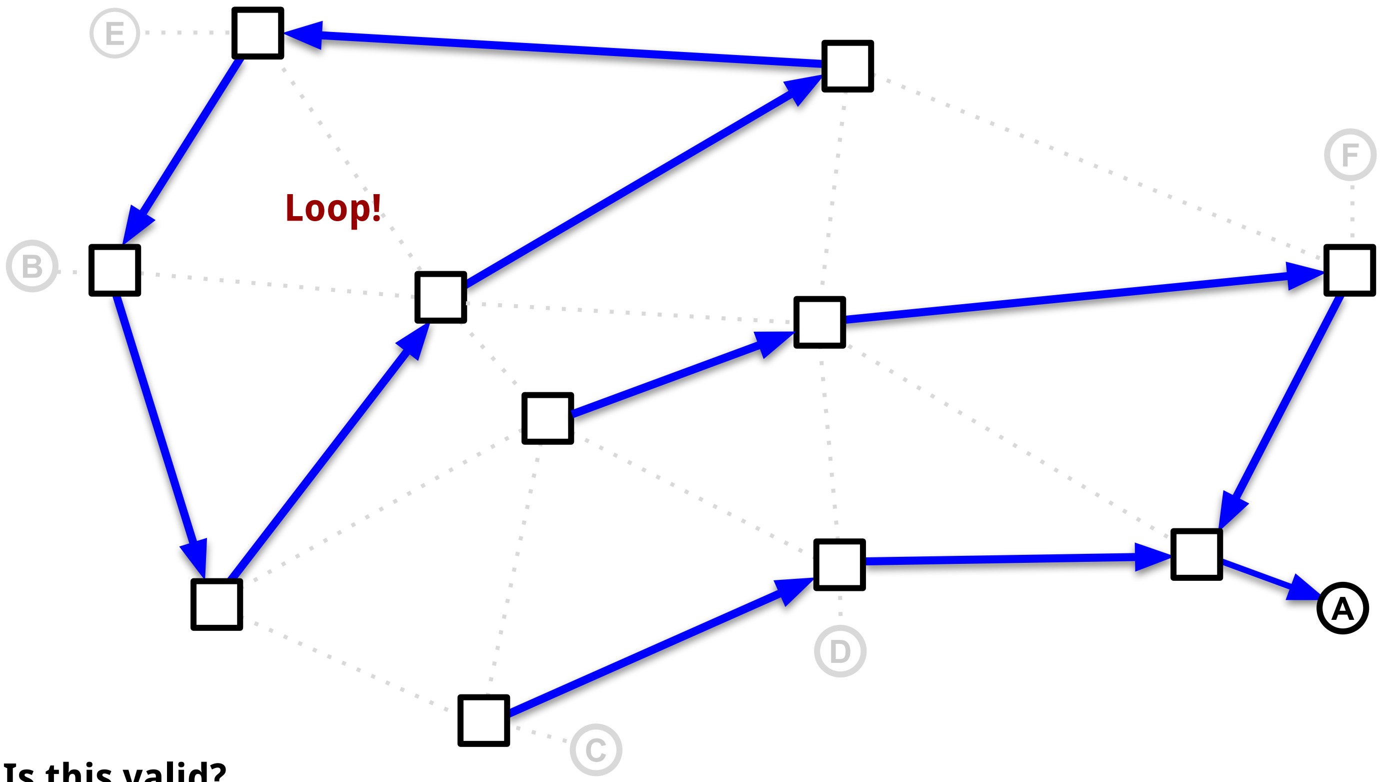
Is this valid?

Another Alternate State



Is this valid?

Another Alternate State



Is this valid?

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!

An experiment…

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

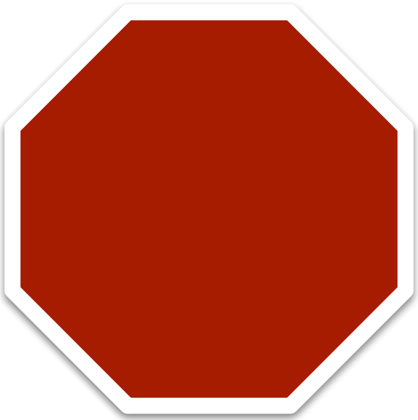
- Initialize:
    - Your number is infinity!
    - Tell your neighbors your name and offer them your number + 1 (i.e. **offer them infinity**)

- 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. 😢

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

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!

Where did the envelopes end up?

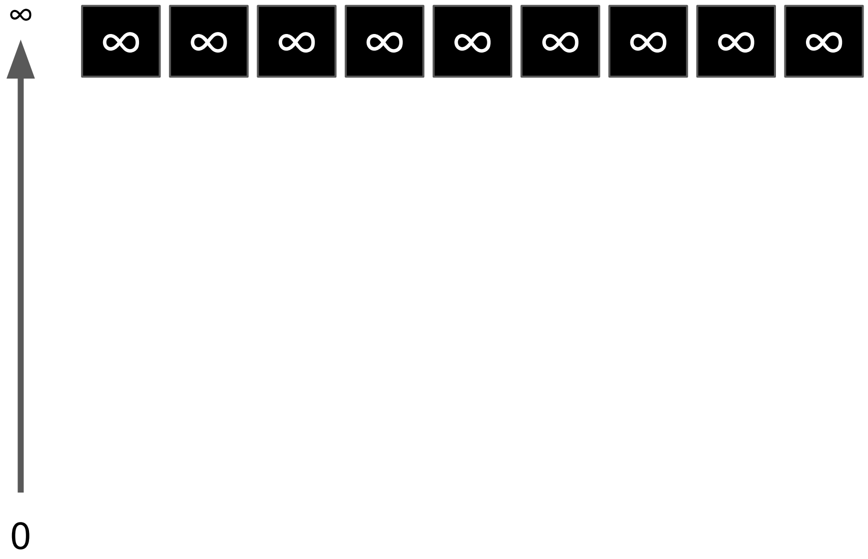
- With any luck, the envelopes got to their intended destination!

Where did the envelopes end up?

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

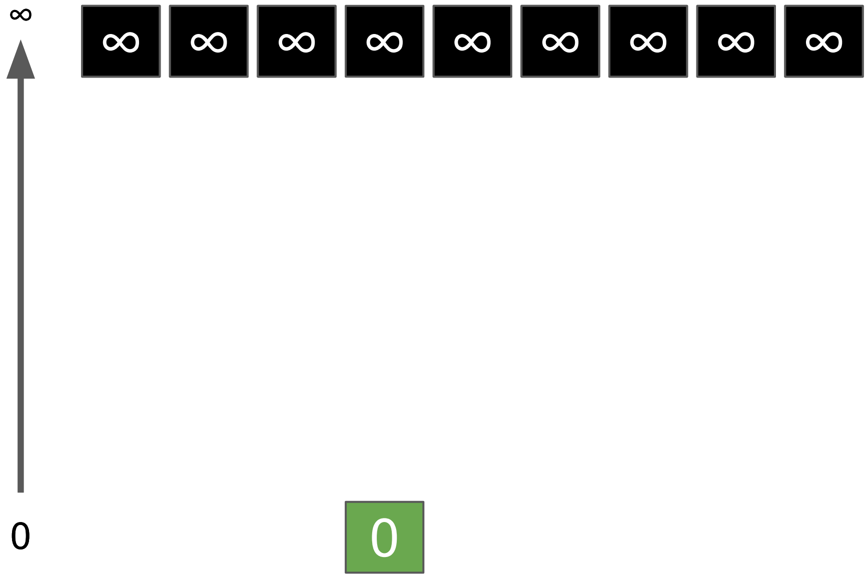
Where did the envelopes end up?

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



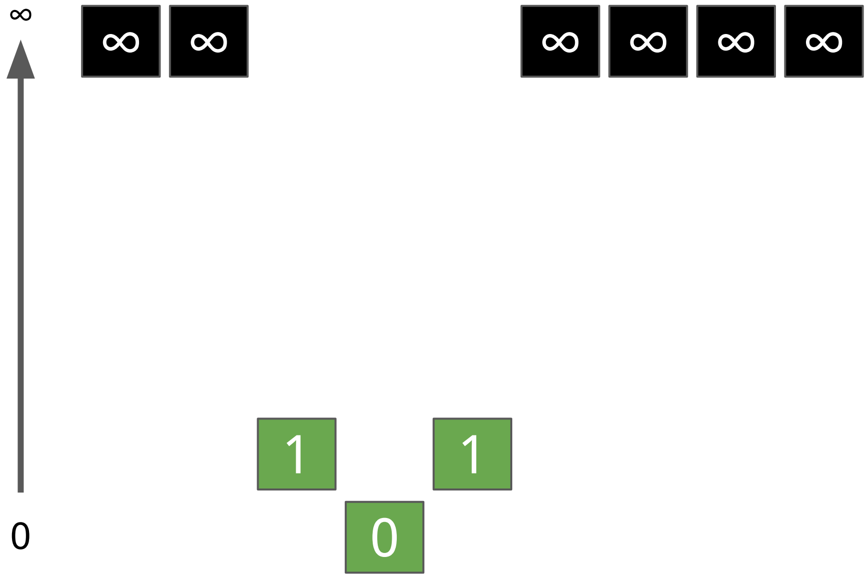
Where did the envelopes end up?

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



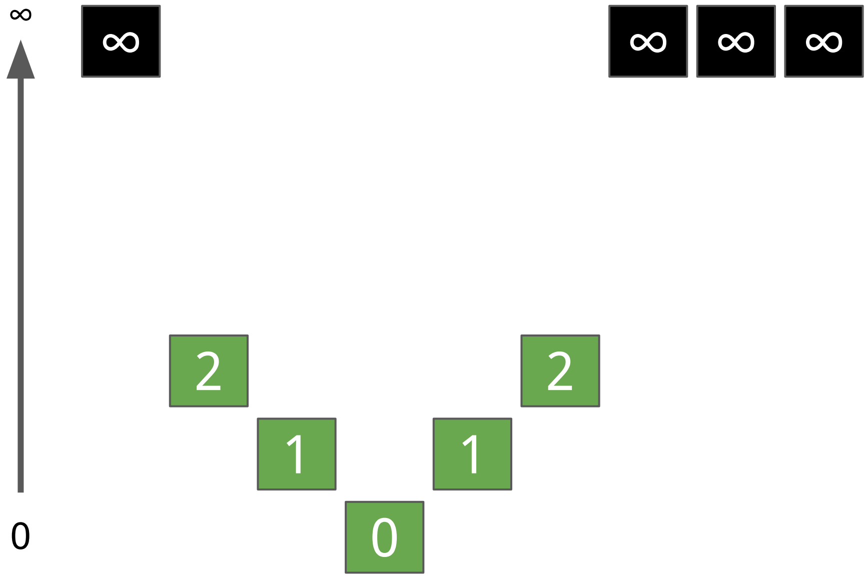
Where did the envelopes end up?

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



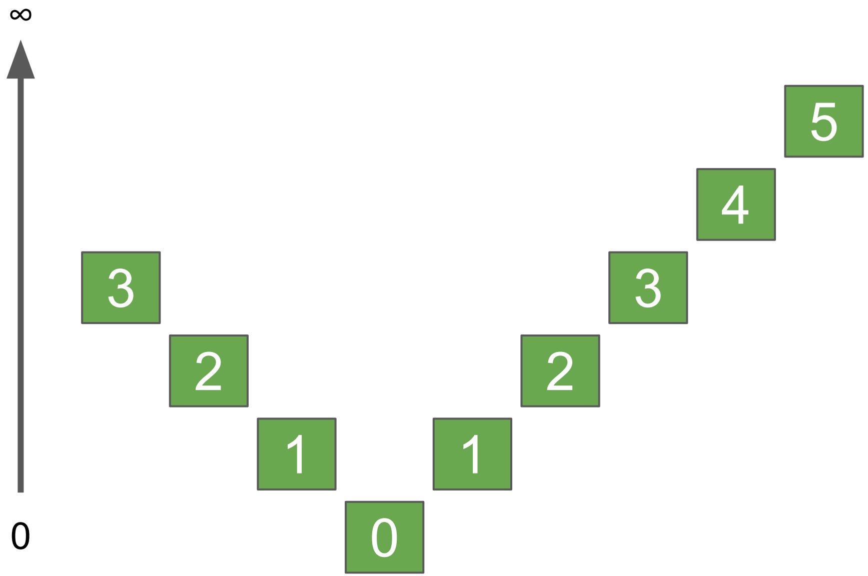
Where did the envelopes end up?

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



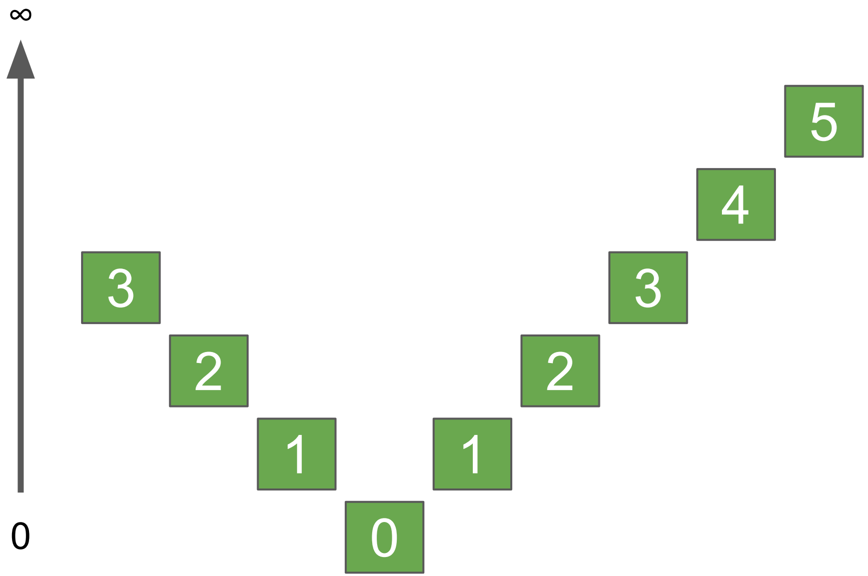
Where did the envelopes end up?

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



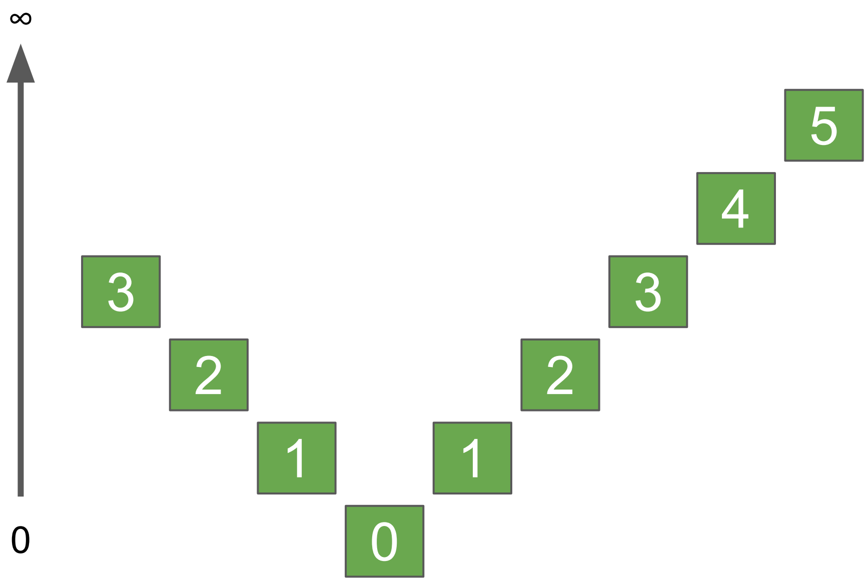
Where did the envelopes end up?

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



Where did the envelopes end up?

- With any luck, the envelopes got to their intended destination!
- How?
- Stage 1:
  - Everyone started with infinity.
  - We gave one person (the destination) zero
  - Who gave their neighbours one
  - Who gave their neighbours two
  - Etc.
  - Created this slope down to the destination
- Stage 2:
  - From wherever, hand envelope down slope
  - It arrives at destination!
- Generalises to many destinations
  - As long as we have a separate number per destination node



What could, or did go wrong?

What could, or did 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

The Bellman-Ford Algorithm

- This was a *distributed* & *asynchronous* version of the Bellman-Ford 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'll talk about one next time.

Distance-Vector Protocols

- Routing protocols that work like this are called *Distance-Vector* protocols
  - Adjacent routers conceptually exchange a *vector* of distances (or <dst,distance> tuples).

- Used in ARPANET as long ago as 1969, and then XEROX.
- Then by *routed* in Berkeley Software Distribution Unix in 1983.
  - *routed's implementation standardised as* **Routing Information Protocol** in 1988
  - Updated for classless addressing in 1994, and then IPv6 in 1997.
  - **A prototypical distance-vector protocol** - albeit not widely deployed any more!

- Some alternative DV protocols (EIGRP, Babel) exist too.

Next Time

- More on *Distance-Vector* protocols.
- Different types of routing protocols - *Link State*.