The Fundamentals of Routing

EE122 Fall 2012

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson and other colleagues at Princeton and UC Berkeley

Announcements

- Participation numbers: we have a problem….
  - 30 have sent mail about participation
  - 330 are enrolled in the course
- Homework #1 due in a week
  - Get it done soon, so you can focus on project
  - **Reminder: work on homework by yourself…..**
- Project 1 is out today!
  - Due in two weeks, get started soon!
  - Colin will give a quick introduction

Project 1

- Goal: implement reliable transport protocol
- Structure of project:
  - We give you the receiver
  - **You implement the sender**
- Receiver sends back cumulative ACKs
  - You must figure out how to use these effectively


LeccePen
5
THIS ISN'T MY
BLANKET

Grading Policy

The grades will be based on correctness and performance, not adherence to a specified algorithm.

- Do you reliably deliver the file?
- Is it accomplished in a timely manner?
- And with a reasonable number of packets?

Grading Policy

- We provide you with a testing framework, including one test case
- **You need to implement further tests!**
- We will run our own tests on your code to generate a grade

Extra Credit

- You can implement optional “bells and whistles” for extra credit
- Extra credit can boost your grade by up to 10%: 5% for the first bell/whistle, 5% for the second

Collaboration Policy

Projects are designed to be solved independently, but you may work with a partner if you wish (but at most two people can work together). Grading will remain the same whether you choose to work alone or with a partner; both partners will receive the same grade regardless of the distribution of work between the two partners (so choose a partner wisely!).

Collaboration Policy (continued)

You may not share code with any classmates other than your partner. You may discuss the assignment requirements or general programming decisions (*e.g.*, what data structures were used to store routing tables) - **away from a computer and without sharing code** - but you should not discuss the detailed nature of your solution (*e.g.*, what algorithm was used to compute the routing table).

Colin is in charge of project 1

- General questions
  - Ask your TA
- Detailed questions about the project code
  - Ask Colin (cs@cs.berkeley.edu)

Questions on Project 1?

Outline

- Review of reliable transport
- Basics of routing and forwarding
- Correctness condition for routing
- Routing on spanning trees
- Preview of next lecture

Review of Reliable Transport

Review of Reliable Transport

- Restatement of correctness condition:

A transport mechanism is "reliable" if and only if it resends all dropped or corrupted packets.

- Sufficient ("if"): algorithm will always keep trying to deliver undelivered packets
- Necessary ("only if"): if it ever lets a packet go undelivered without trying again, it isn't reliable

- **Note: a transport mechanism can "give up", but must announce this to application**

Many Implementation Choices

- Feedback from receiver: ACKs vs NACKs
  - Can NACKs alone achieve “correctness”?
  - Can ACKs alone achieve “correctness”?
- Variations on ACKs
  - Full information
  - Individual packets
  - Cumulative (project; TCP)
- When to resend
  - Timeout
  - Duplicate ACKs
  - NACKs

Implementation Choices

- These implementation choices affect:
  - Timeliness
  - Efficiency
  - Fairness
  - ….
- These are important concerns
  - **but correctness is more fundamental**
- Design must *start* with correctness
  - Can then “engineer” its performance with various hacks
  - These hacks can be “fun” but don’t let them distract you

Routing

The Traditional Routing Curriculum

- Learning switches
- Link-state routing
  - Dijkstra's Algorithm
- Distance-vector routing
  - Bellman-Ford

I have some bad news…..

- Don’t have anything interesting to say about routing
- Will follow standard curriculum
  - Much of it covered in the text
- *But will focus more on principles than details*
- Will continue routing on Thursday…

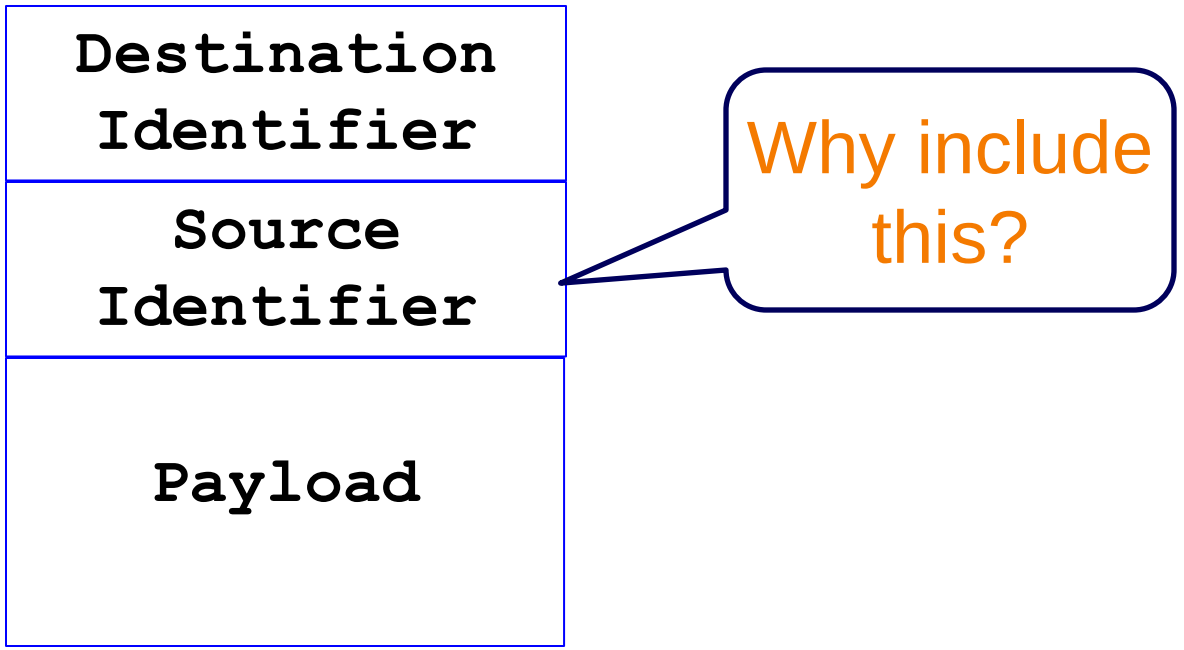
Basics of Routing and Forwarding

Addressing (at a conceptual level)

- Assume all hosts have unique IDs (addresses)
- No particular structure to those IDs
- Later in course will talk about real IP addressing

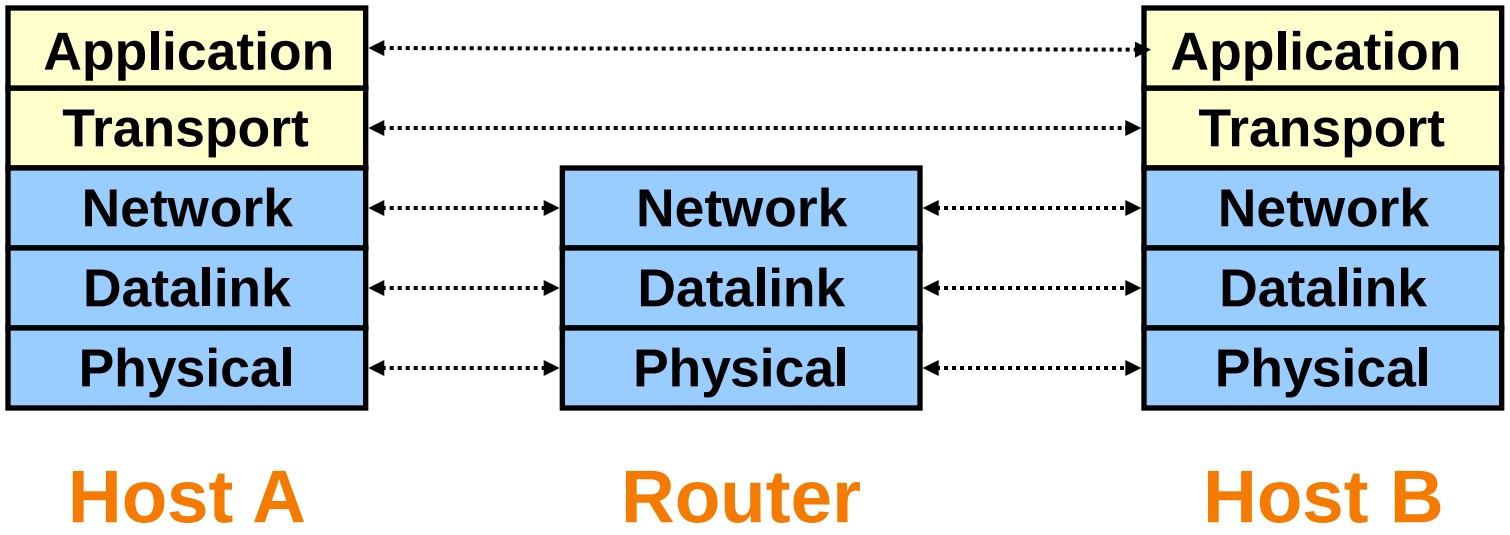
Packets (at a conceptual level)

- Routing occurs at network and datalink layers
- Assume network/datalink packet headers contain:
  - Source ID, Destination ID, and perhaps other information

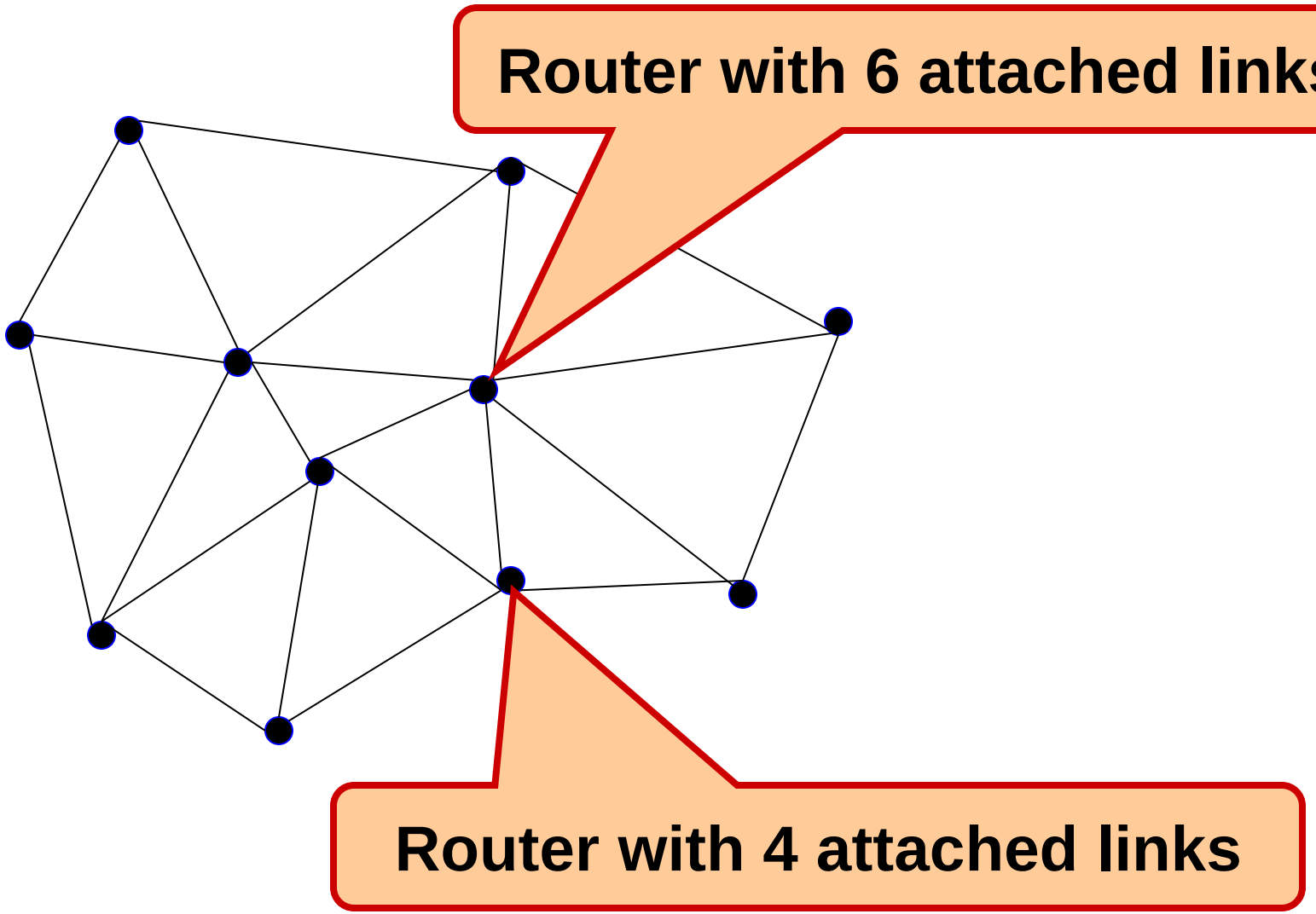


Layering Diagram

- Why would you have return address in network layer (or datalink layer)?
- Historical and network-level reasons….



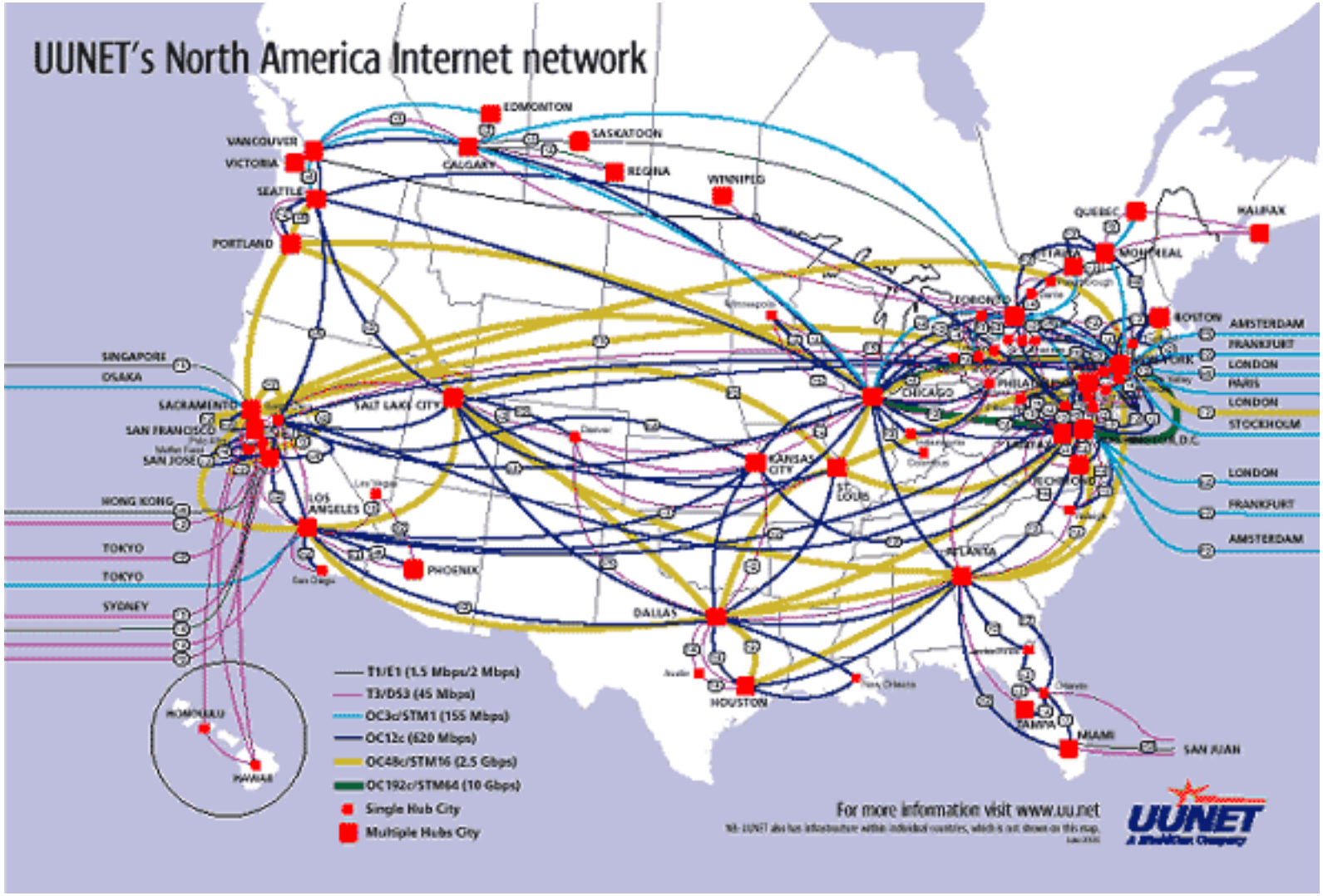
Example of Network Graph



A Variety of Networks

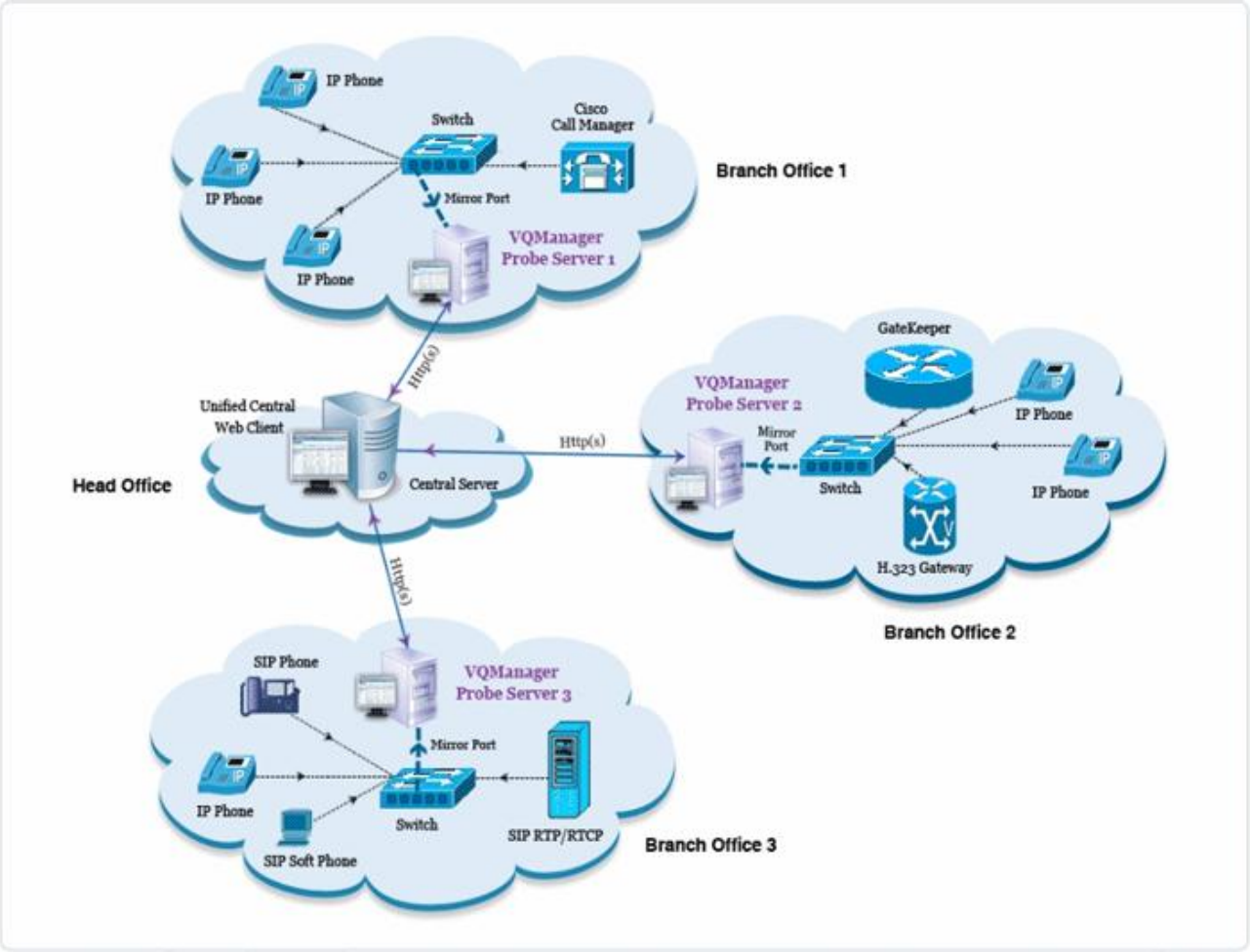
- **ISPs**: carriers
  - Backbone
  - Edge (connecting to customers)
  - Border (to other ISPs)
- **Enterprises**: companies, universities
  - Core
  - Edge (connecting to hosts)
  - Border (to outside)
- **Datacenters**: massive collections of machines
  - Aggregation and Core
  - Top-of-Rack
  - Border (to outside)

UUNET's North American Network

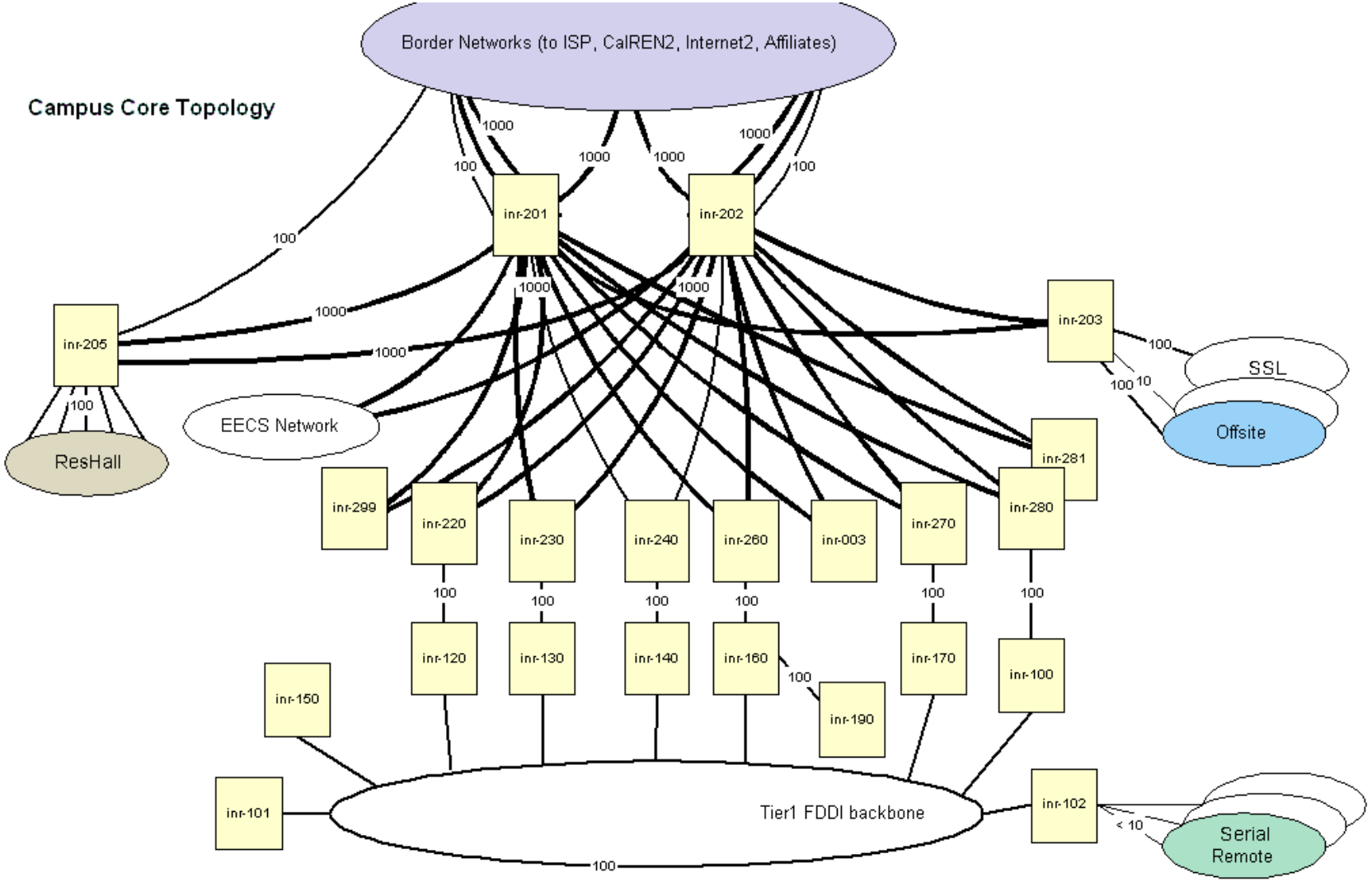


Level3's American Network

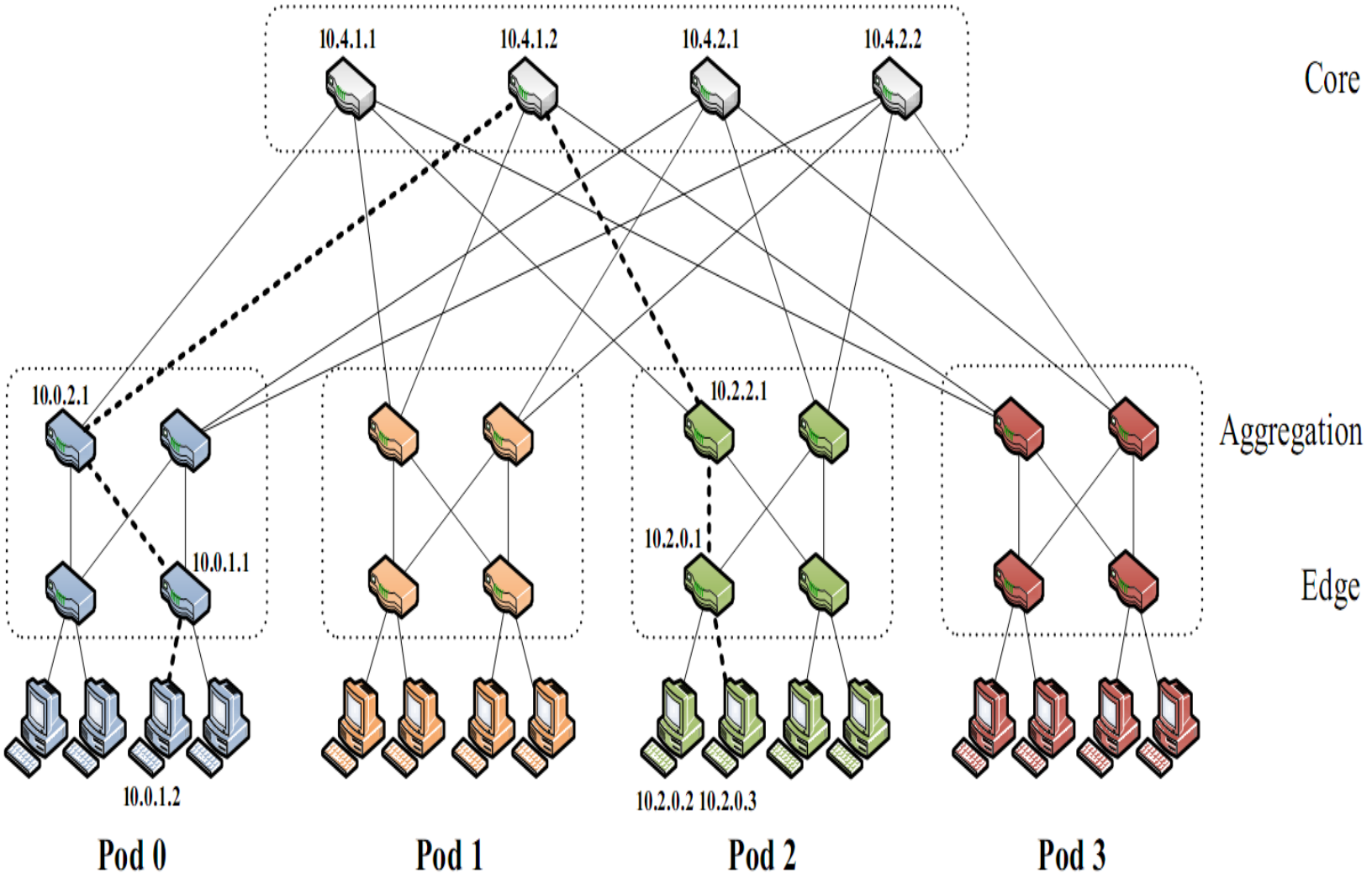
Enterprise Network



Berkeley's Campus Network



Partial Datacenter Network

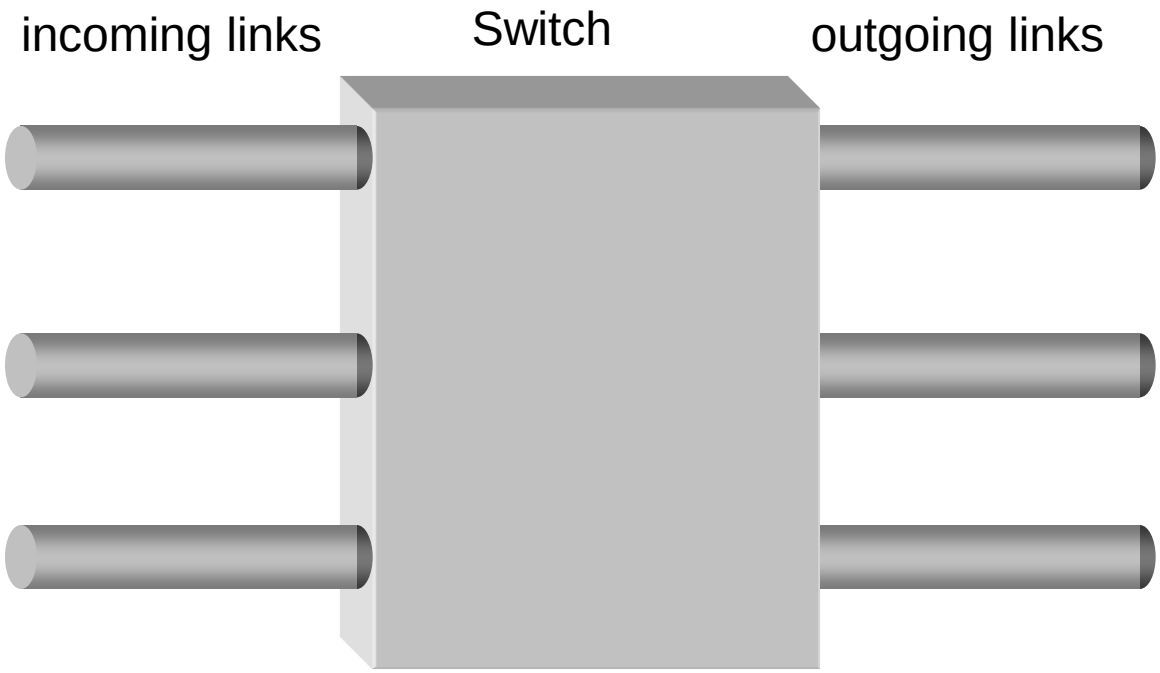


Switches

- Enterprise/Edge: typically 24 to 48 attached links
- Aggregation switches: 192 or more
- Backbone: typically fewer attached links
- Border: typically very few attached links

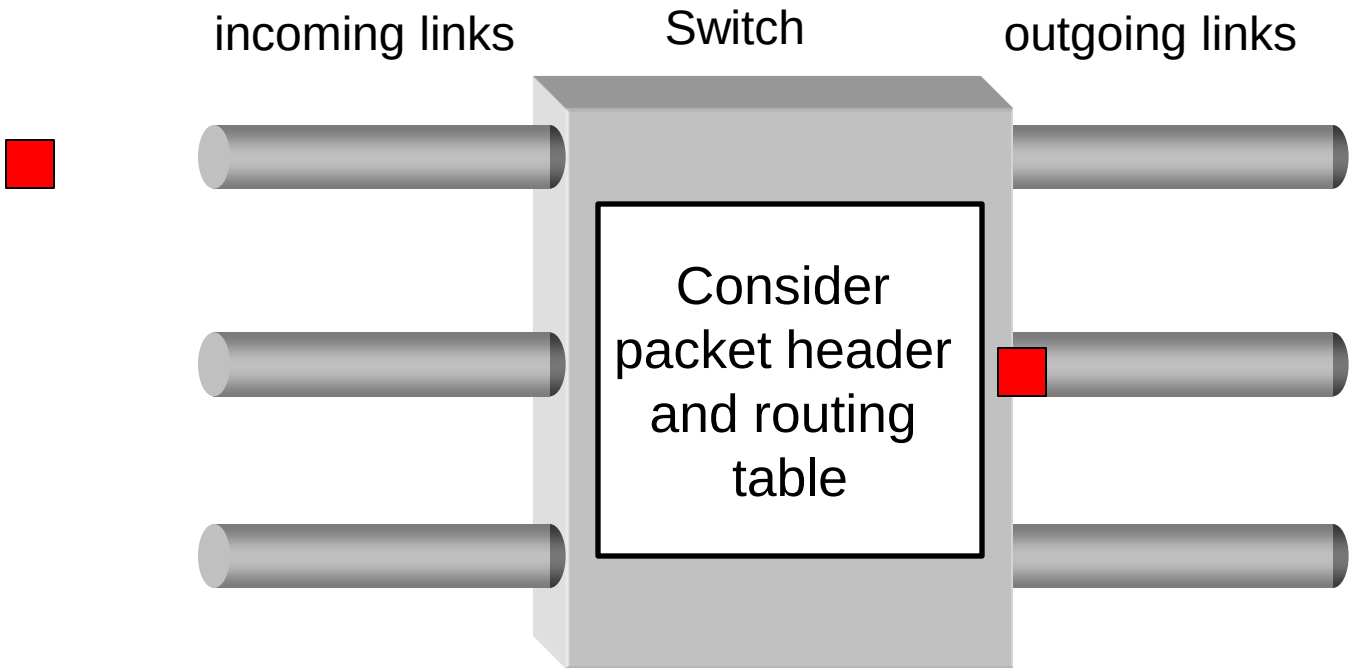
Switches/Routers

- Multiple attached links, often called “ports”
  - Ports are typically duplex (incoming and outgoing)
    - **But in this picture will show them separately**
  - (Don’t confuse this notion of “port” with transport “ports”)



Forwarding Decisions

- When packet arrives, must choose outgoing port



- Decision is based on routing state (table) in switch

Forwarding Decisions

- Switches and routers make the following mapping:

  PacketState + RoutingState ➔ OutgoingPort

- Most do so in single transmission time
  - Forwarding decisions must be ***simple***

- Assume forwarding decisions are **deterministic**
  - Packets with same state always routed to same port

Packet State

- Destination ID
- Source ID
- Incoming Port *(from switch, not packet header)*
- Other packet header information?
  - Ignore for now…

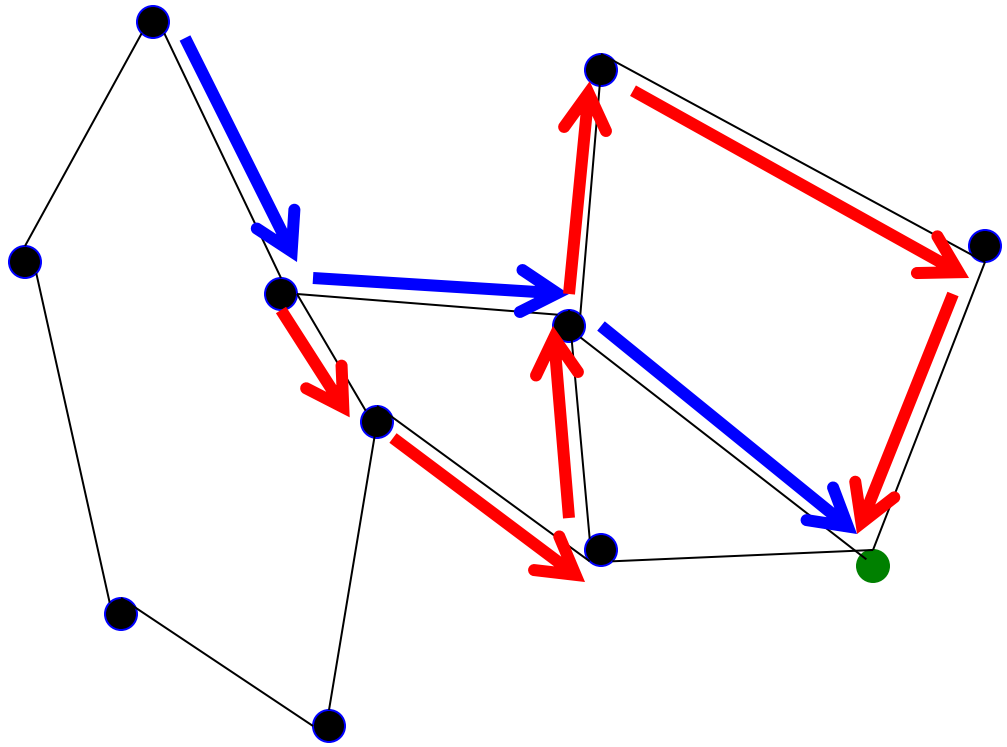
Forwarding Decision Dependencies

- ***Must*** depend on destination

- Could also depend on :
  - **Source**: requires n^2 state
  - **Input port**: not clear what this buys you
  - **Other header information**: ignore for now

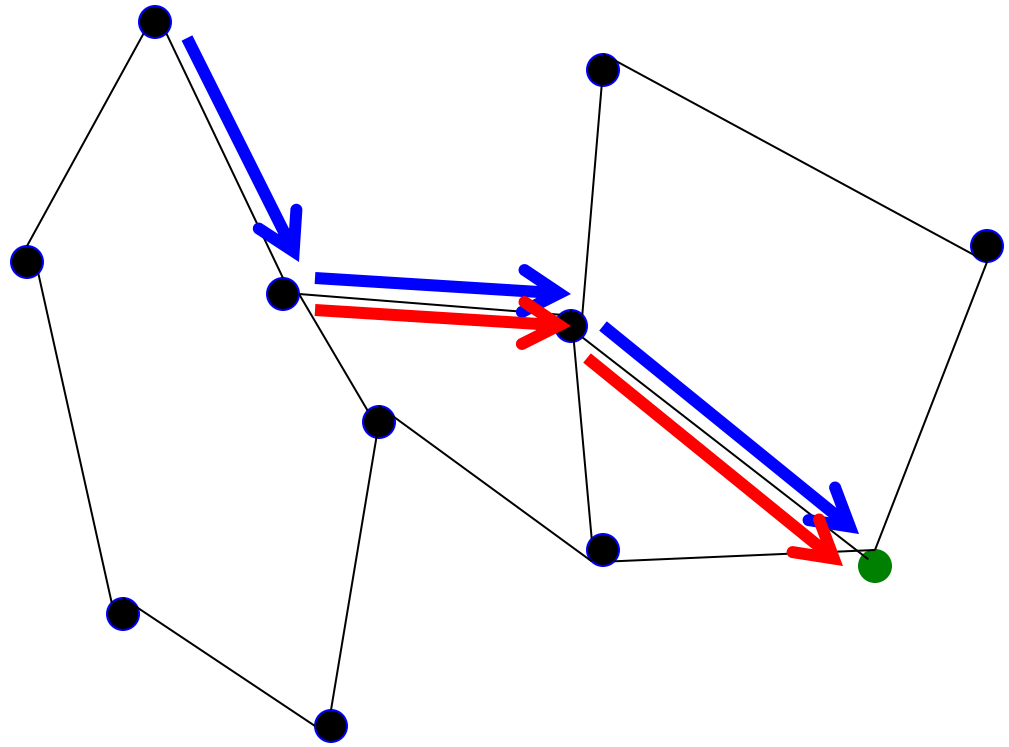
- We will focus only on destination-based routing
  - But first consider the alternative

Source/Destination-Based Routing



Paths from two different sources (to same destination) can be very different

Destination-Based Routing

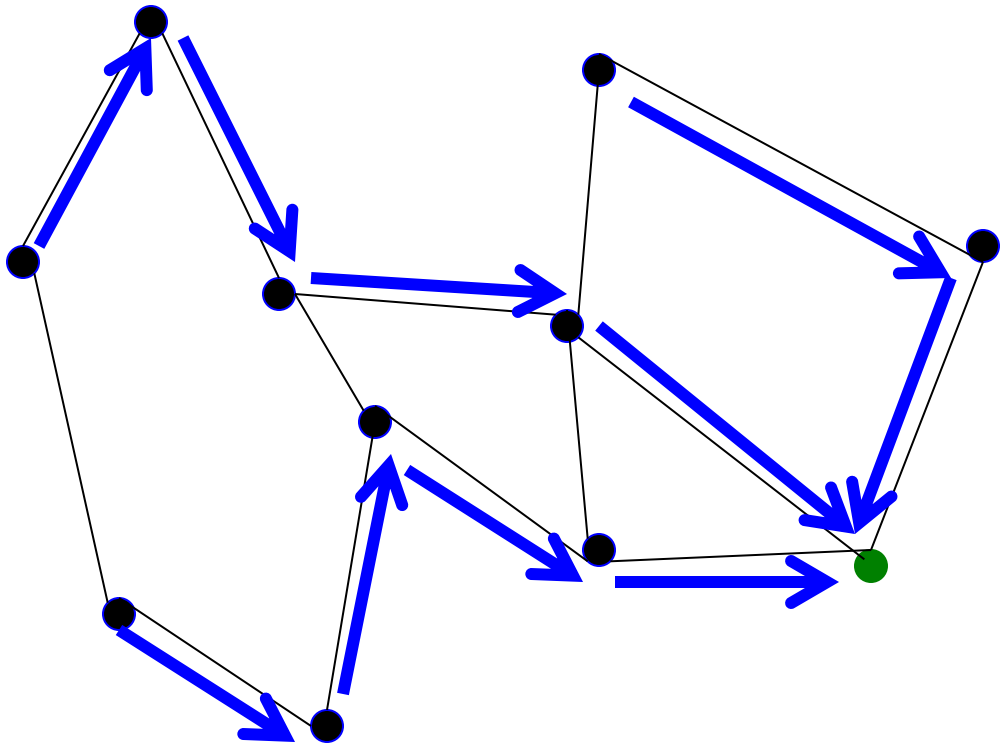


Paths from two different sources (to same destination) must coincide once they overlap

Destination-Based Routing

- Paths to same destination never cross
- Once paths to destination meet, they never split
- Set of paths to destination create a “delivery tree”
  - Must cover every node exactly once
  - **Spanning Tree rooted at destination**

A “Delivery Tree” for a Destination



Assume Destination-Based Routing

- For rest of lecture (and course)…..

5 Minute Break

Validity of Routing State

Local and Global Routing State

- *Local routing state* is the state in a single router
  - By itself, the state in a single router can't be evaluated
  - It must be evaluated in terms of the global context
- *Global routing state* means collection of routing state in each of the routers
  - Global state determines which paths packets take
  - Will discuss later where this routing state comes from

“Valid” Routing State

- Global routing state is “valid” if it produces forwarding decisions that always deliver packets to their destinations
  - ***Valid is my terminology, not standard***
- Goal of routing protocols: compute valid state
  - But how can you tell if routing state if valid?
- Need a succinct correctness condition for routing
  - Suggestions?

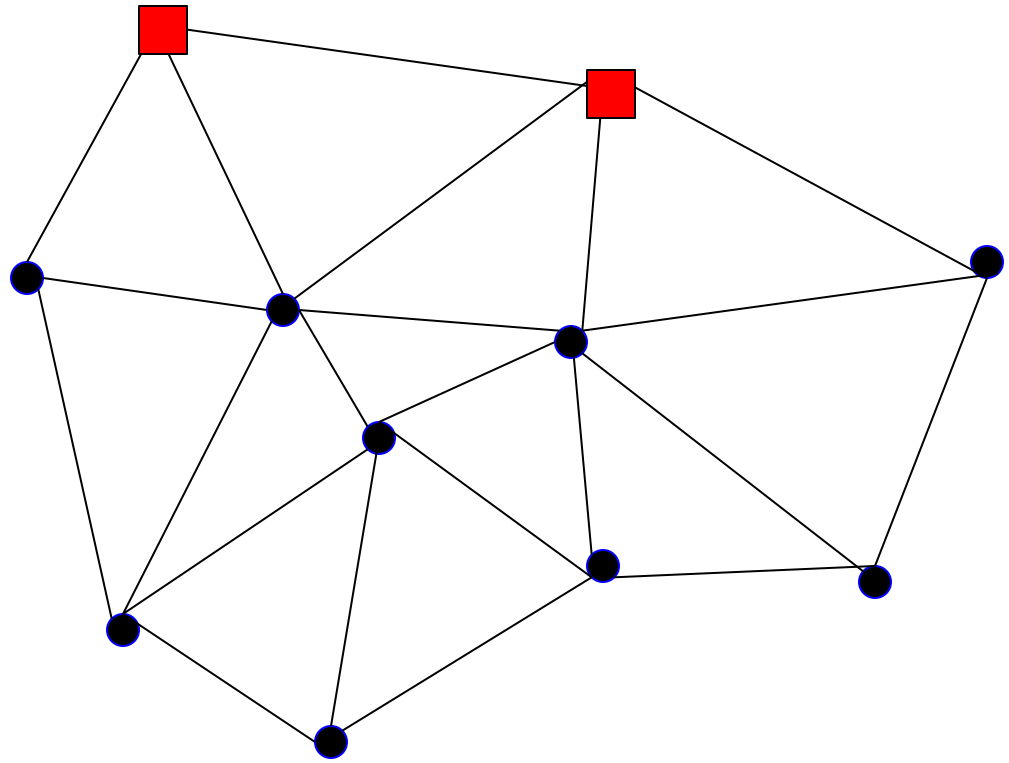
Necessary and Sufficient Condition

- Global routing state is valid ***if and only if***:
  - There are no dead ends (other than destination)
  - There are no loops

- A dead end is when there is no outgoing port
  - A packet arrives, but the forwarding decision does not yield any outgoing port

- A loop is when a packet cycles around the same set of nodes forever

Wandering Packets



Packet finds its destination

Packet reaches end host

Packet reaches destination

Necessary and Sufficient Condition

- Global routing state is valid ***if and only if***:
  - There are no dead ends (other than destination)
  - There are no loops

Necessary ("only if"): Obvious

- If you run into a deadend before hitting destination, you'll never reach the destination

- If you run into a loop, you'll never reach destination
  - With deterministic forwarding, once you loop, you'll loop forever (assuming routing state is static)

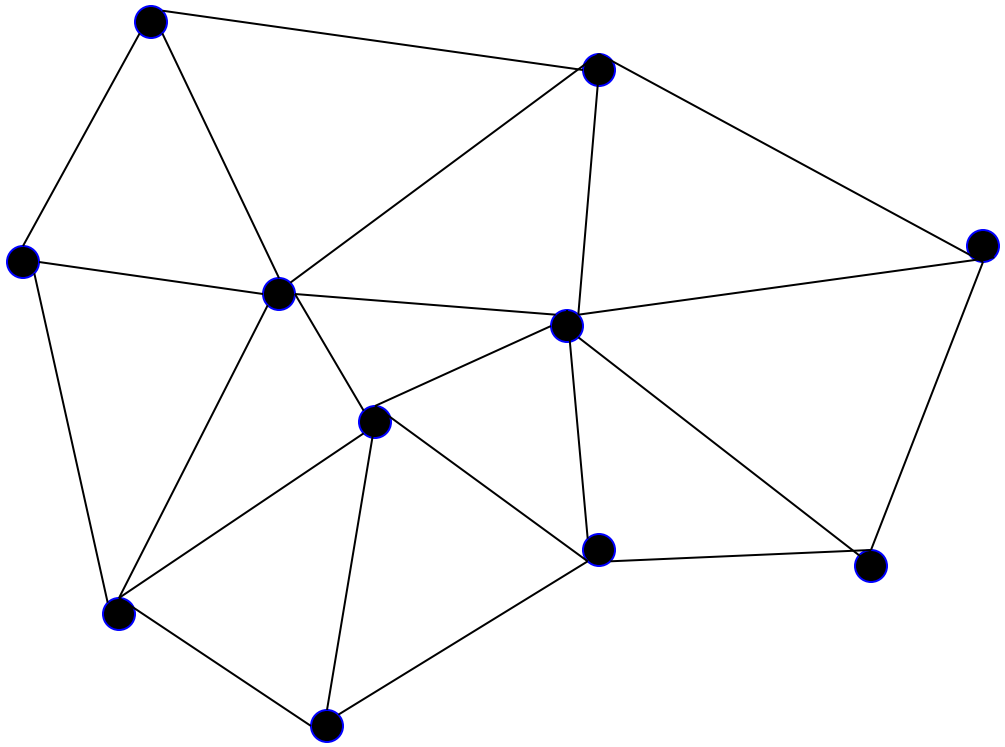
Sufficient ("if"): Easy

- Assume no deadends, no loops
- Packet must keep wandering, without repeating
  - If ever enter same switch from same port, will loop
  - Because forwarding decisions are deterministic
- Only a finite number of possible ports for it to visit
  - It cannot keep wandering forever without looping
  - Must eventually hit destination

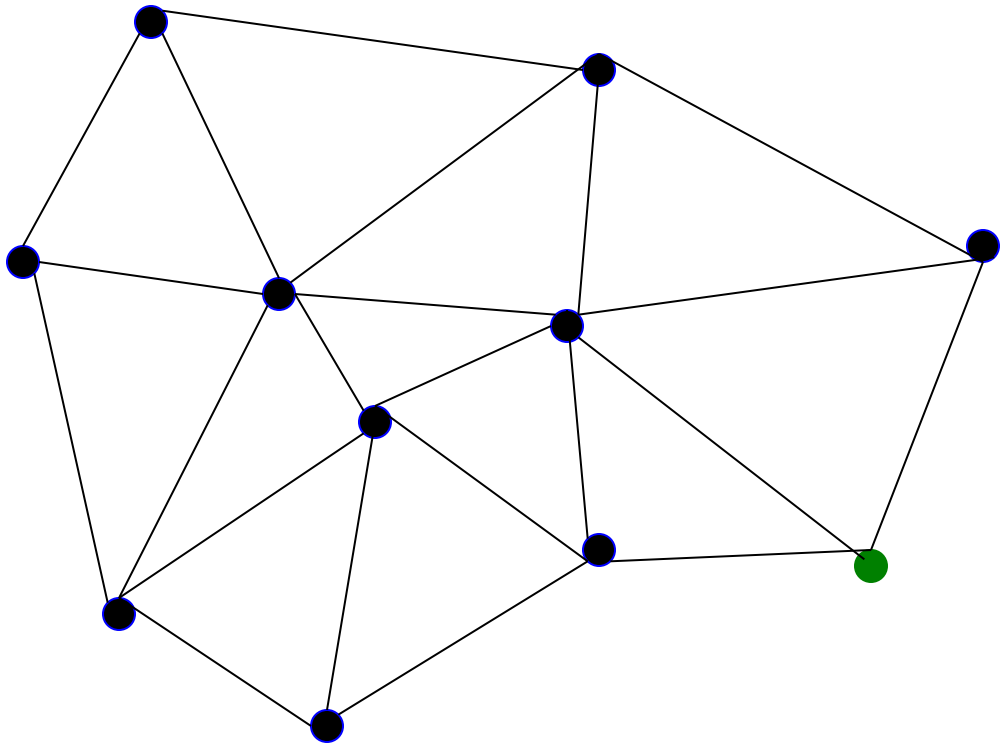
Checking Validity of Routing State

- Focus only on a single destination
  - Ignore all other routing state
- Mark outgoing port with arrow
  - There can only be one at each node
- Eliminate all links with no arrows
- Look at what's left….

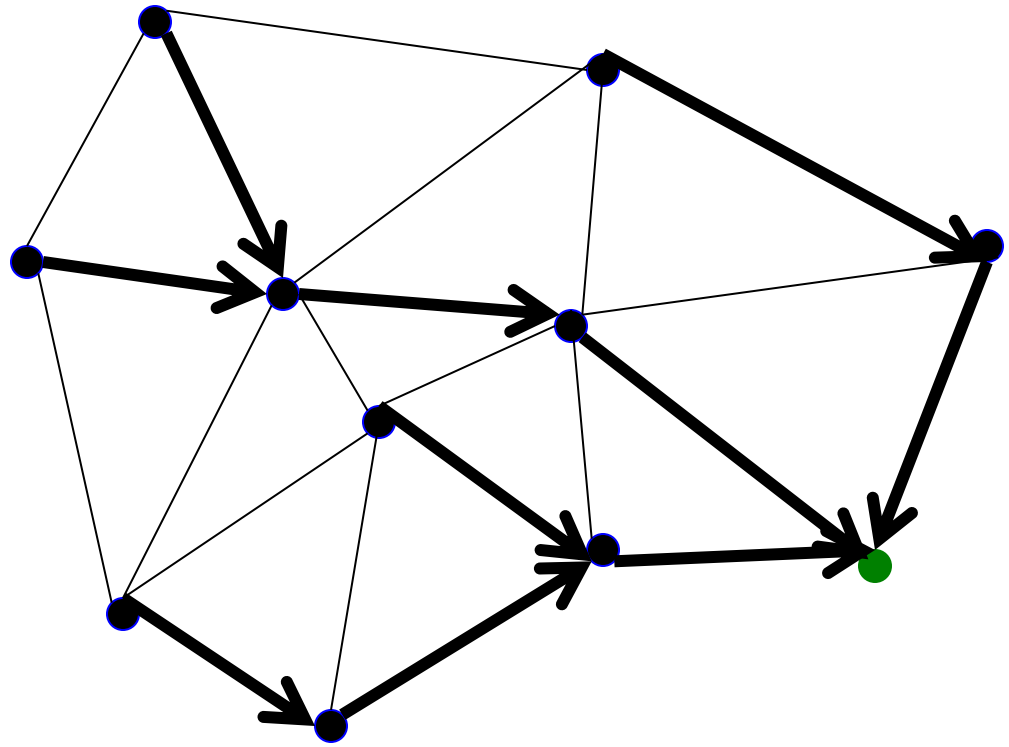
Example 1



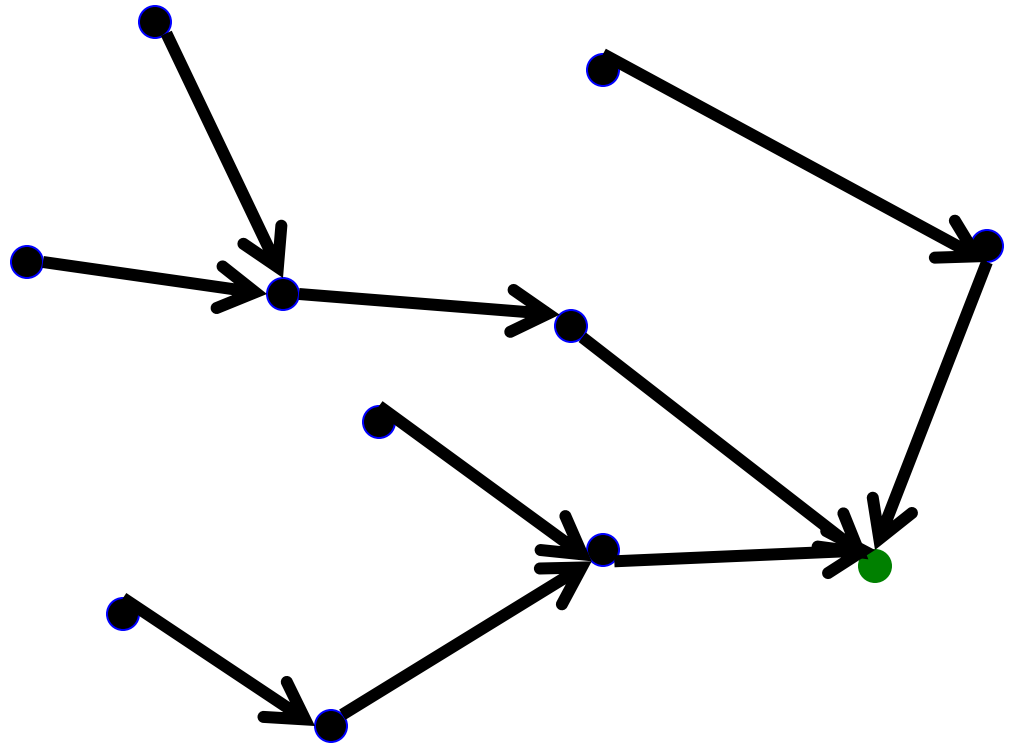
Pick Destination



Put Arrows on Outgoing Ports

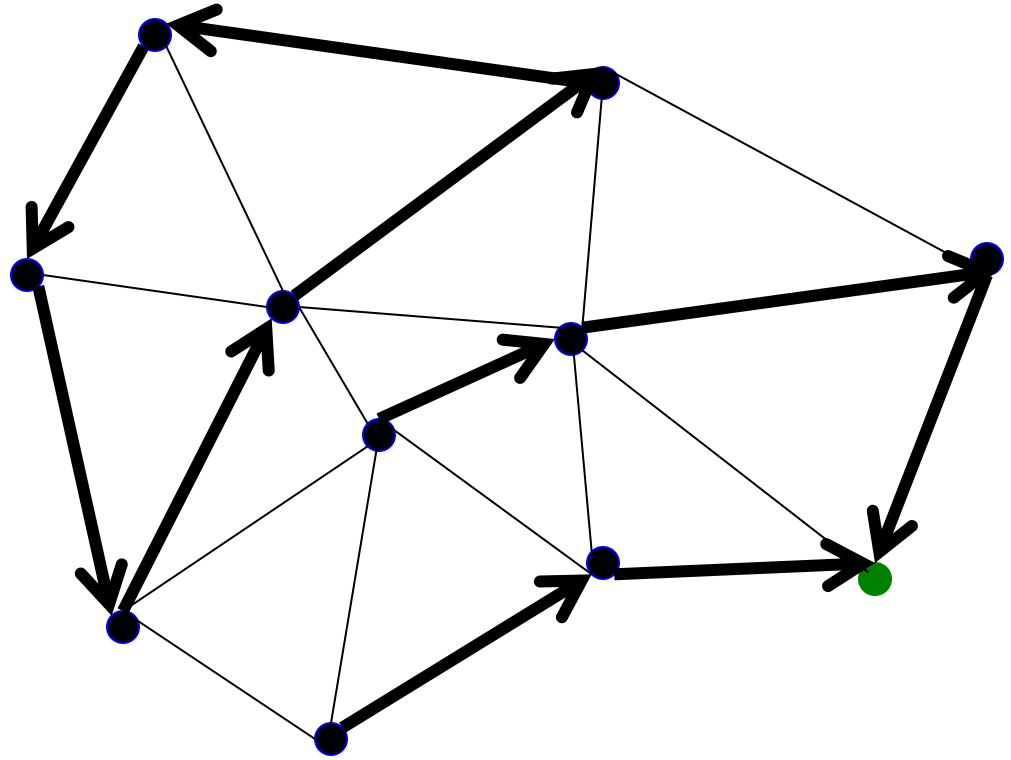


Remove Unused Links

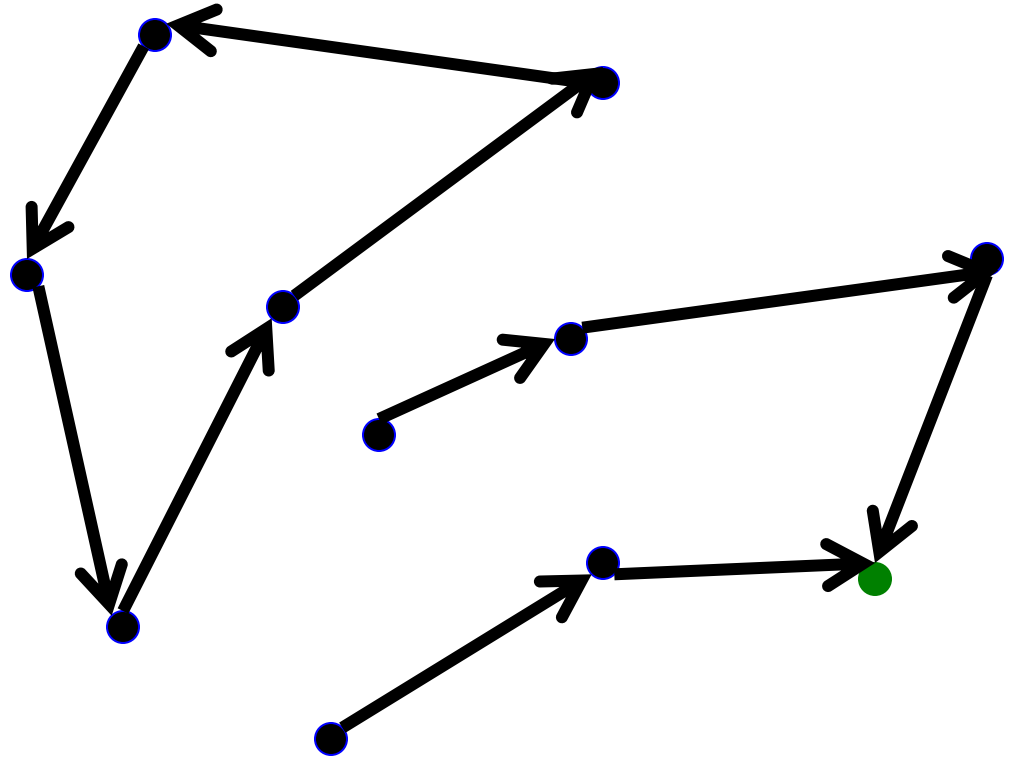


Leaves Spanning Tree: Valid

Second Example



Second Example



Is this valid?

Lesson….

- Very easy to check validity of routing state for a particular destination
- Deadends are obvious
  - Node without outgoing arrow
- Loops are obvious
  - Disconnected from rest of graph

Computing Routing State

Forwarding vs Routing

- Forwarding: "**data plane**"
  - Directing a data packet to an outgoing link
  - Individual router using routing state
- Routing: "**control plane**"
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Jointly creating the routing state
- Two very different timescales….
  - Forwarding: single packet transmission times: μs
  - Routing: can be seconds
  - **6 orders of magnitude!**

The “Secret” of Routing

- Avoiding deadends is easy
- Avoiding loops is hard
- **The key difference between routing protocols is how they avoid loops!**
  - Don’t focus on details of mechanisms
  - Just ask “how are loops avoided?”

How Can You Avoid Loops?

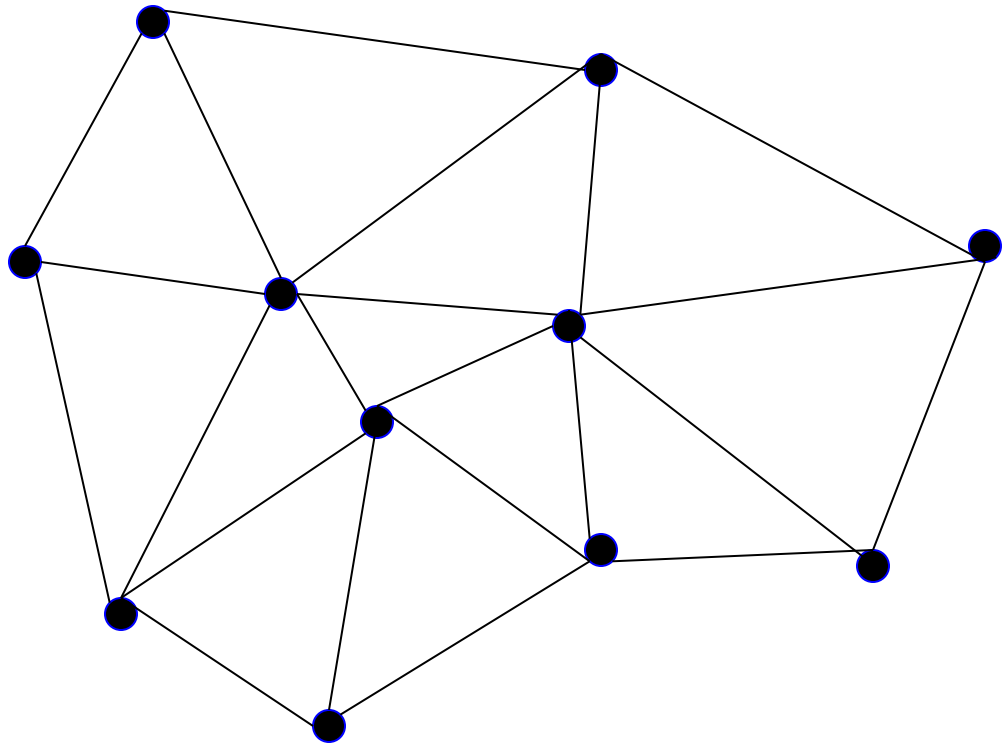
- Easiest way: Restrict topology to spanning tree
  - If the topology has no loops, packets can't loop!
  - (without making a u-turn, which can be locally prevented)

Routing on Spanning Tree

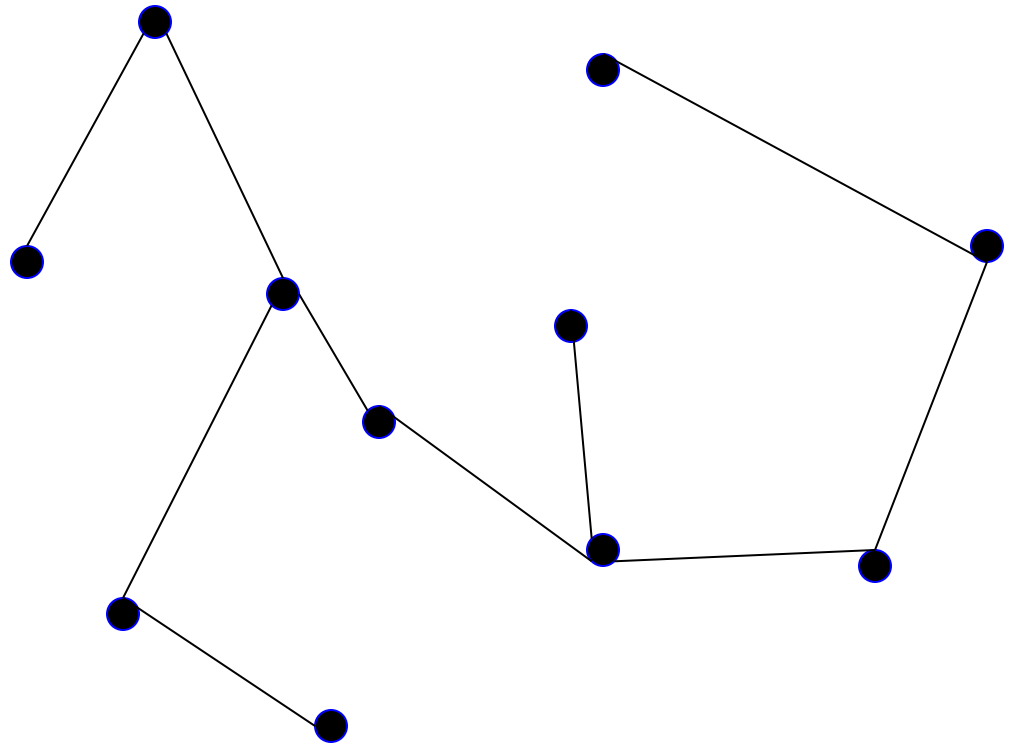
Easiest Way to Avoid Loops

- Use a topology where loops are impossible!
- Take arbitrary topology
- Build spanning tree (algorithm covered later)
  - Ignore all other links (as before)
- Only one path to destinations on spanning trees
  - So don't have to worry about loops!

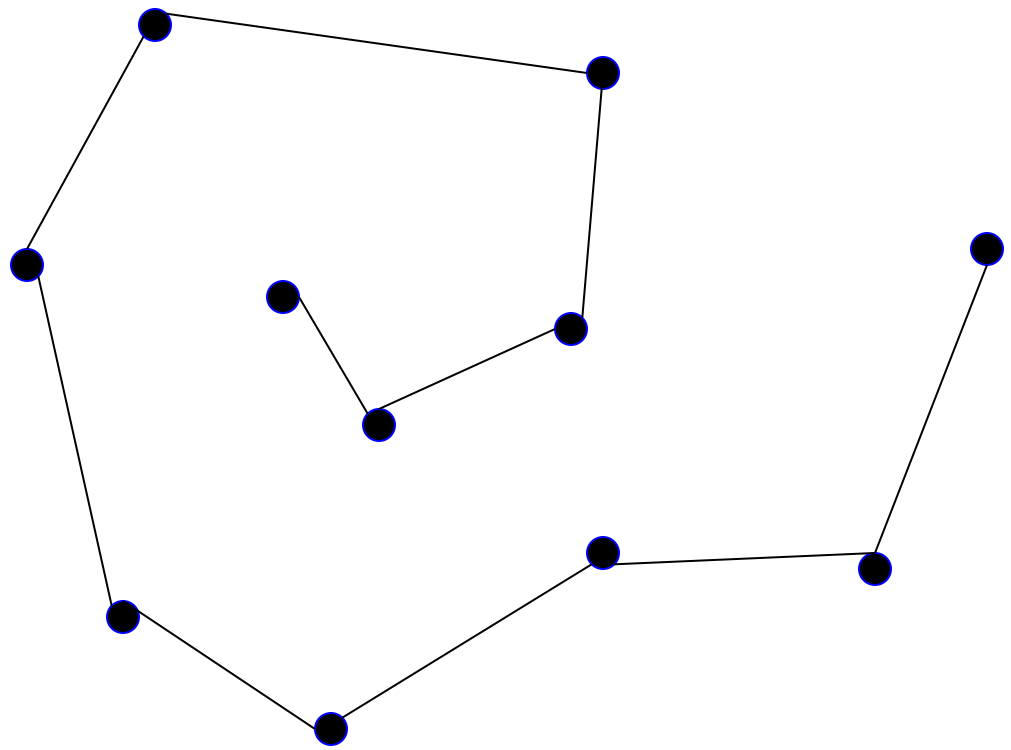
Consider previous graph



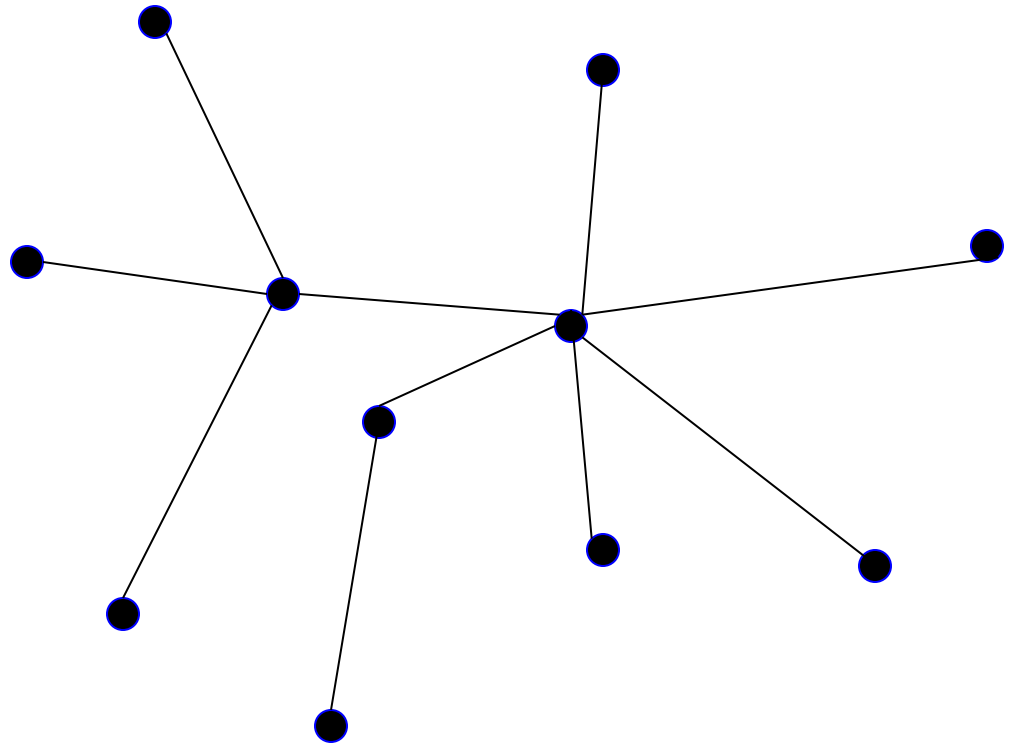
A Spanning Tree



Another Spanning Tree



Yet Another Spanning Tree



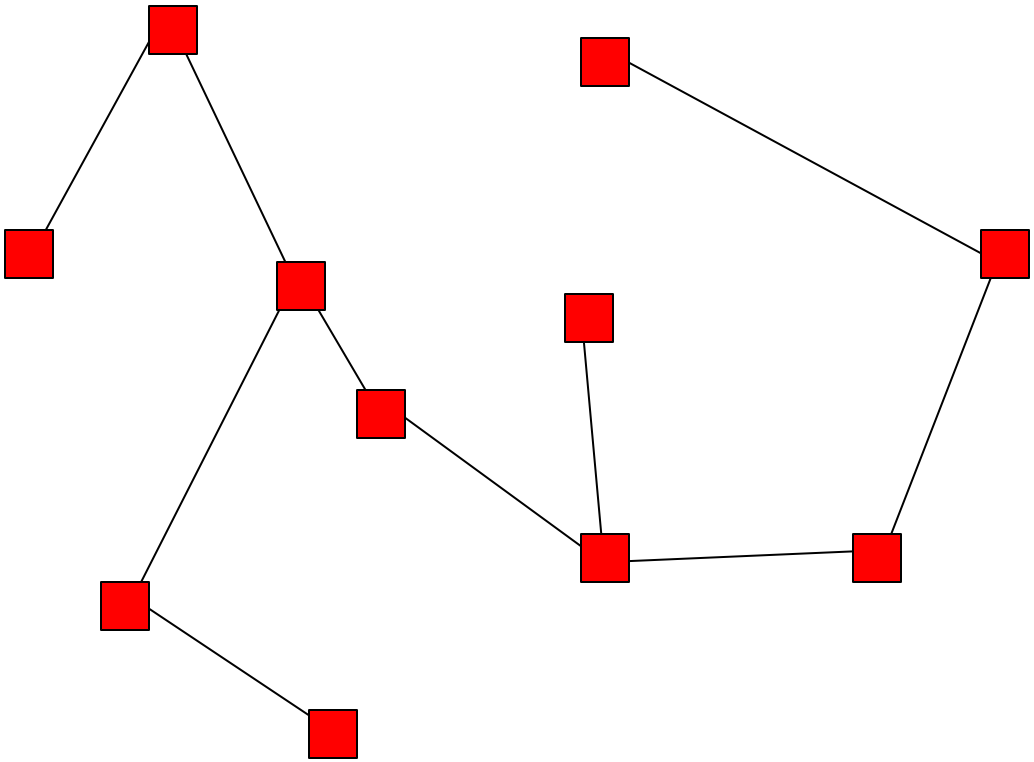
Routing on a Spanning Tree

- There is only one path from source to destination
- How do you find that path?
- Why bother? Just send packets along all paths
  - No packets will loop, but some will hit deadends
  - But one (and exactly one) will reach destination

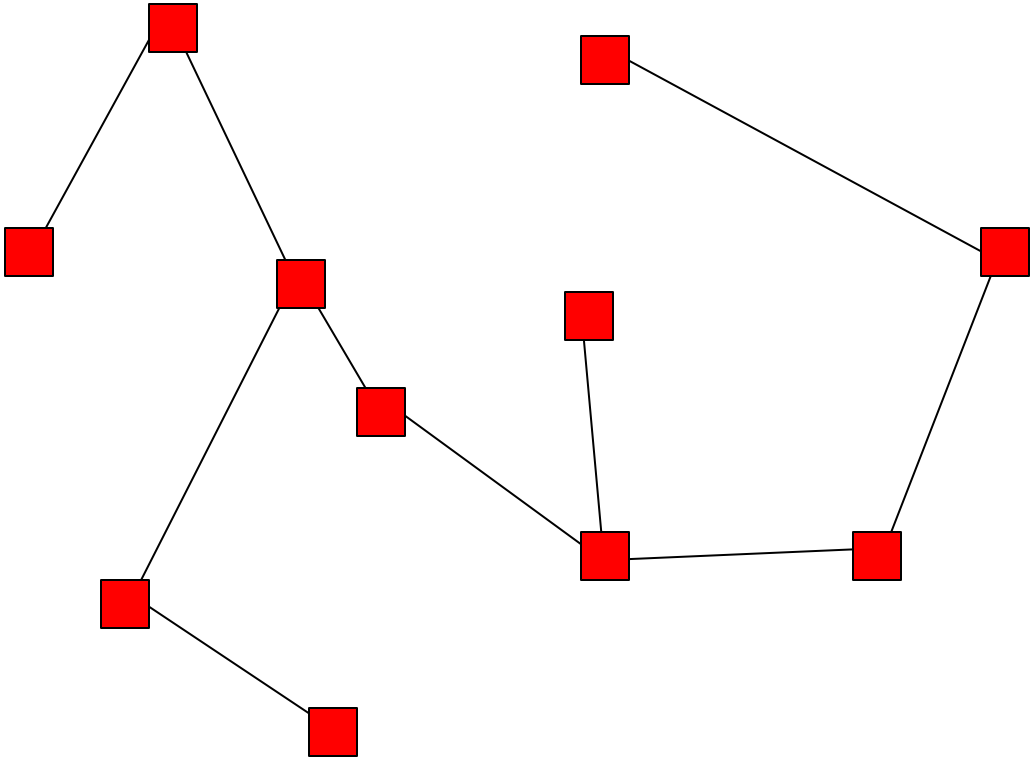
Flooding on a Spanning Tree

- If you want to send a packet that will reach all nodes, then switches can use the following rule:
  - **Ignoring all ports not on spanning tree!**
- Originating switch sends “flood” packet out all ports
- When a “flood” packet arrives on one incoming port, send it out all other ports

Flooding on Spanning Tree



Flooding on Spanning Tree (Again)



Flooding on a Spanning Tree

- This works because the lack of loops prevents the flooding from cycling back on itself
- Eventually all nodes will be covered, exactly once

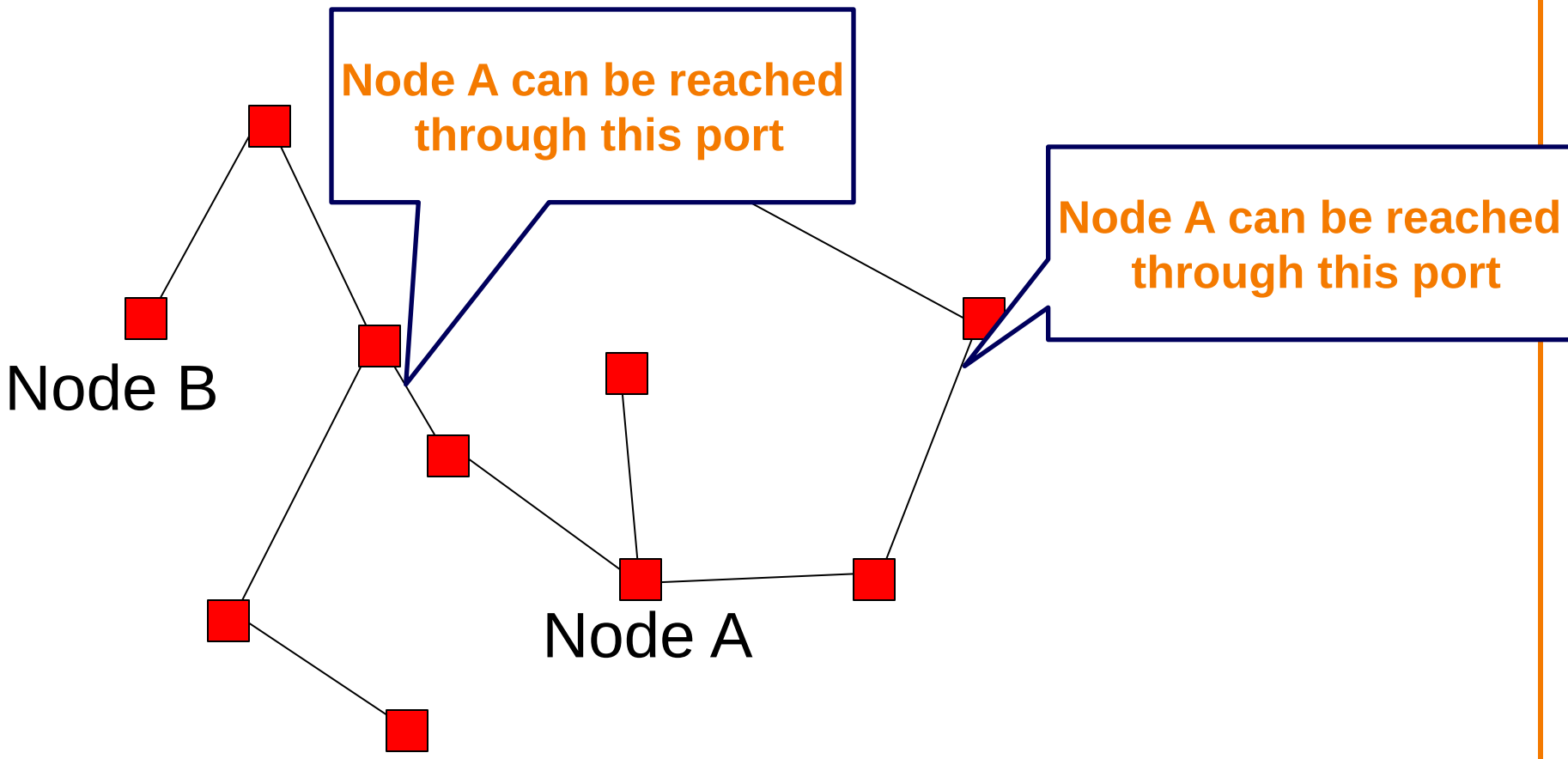
But isn’t flooding wasteful?

- Yes, but you can watch the packets going by, and learn from that
- There is a single path between any two nodes
- If node A sees a packet from node B come in on a particular port, what can it conclude?
- **It knows what port to use to reach B!**

Nodes can “learn” routing tables

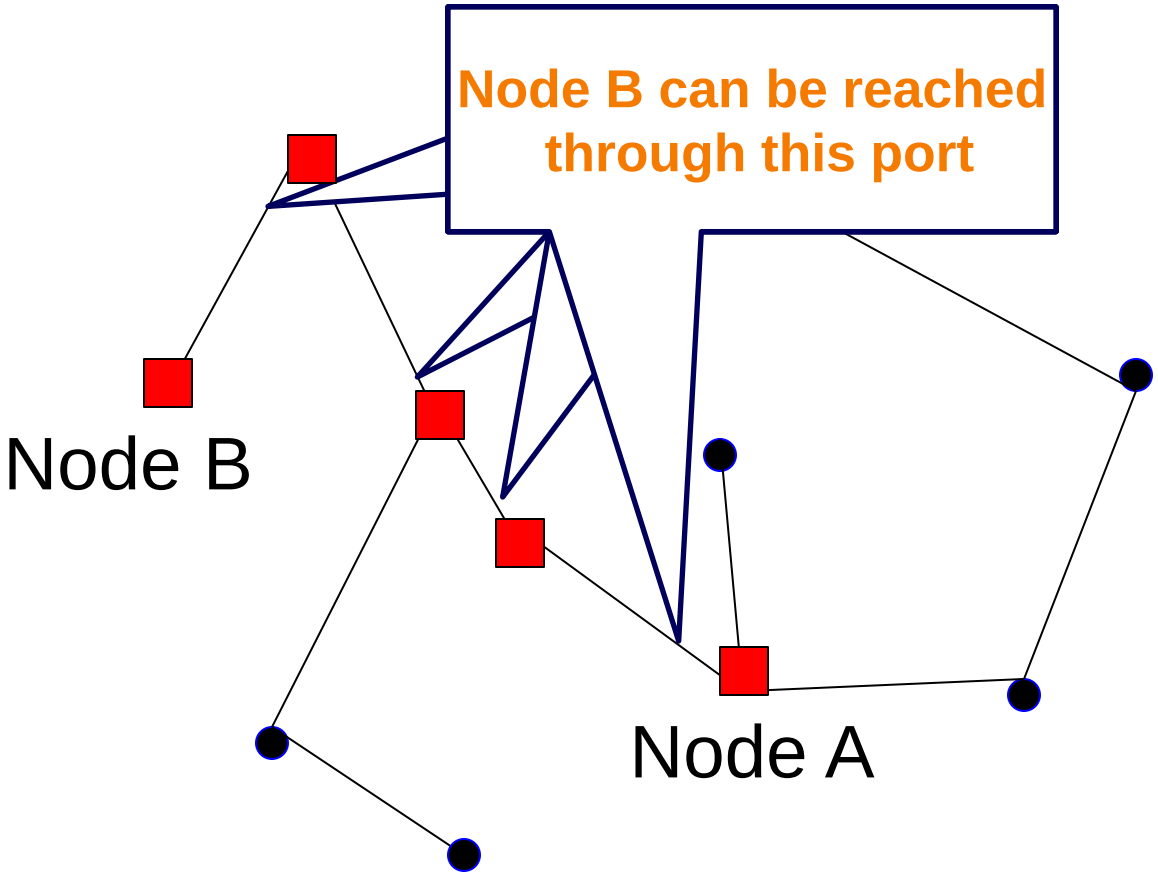
- Switch can learn how to reach nodes by remembering where flooding packets came from!
- If flood packet from Node A entered switch from port 4, then switch uses port 4 to reach Node A

Learning from Flood Packets



Once a node has sent a flood message, all other switches know how to reach it….

Node B Responds



When a node responds, some of the switches learn where it is

General Approach

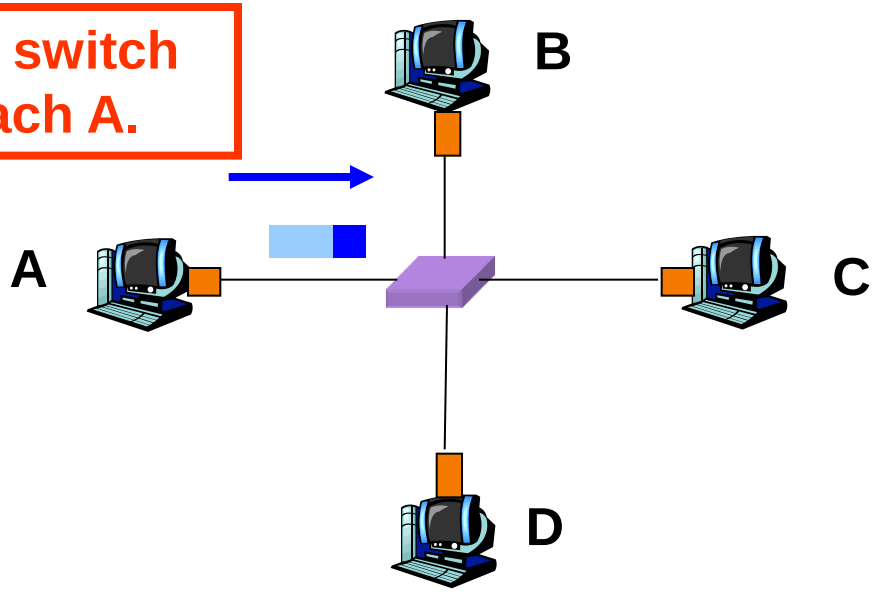
- Flood first packet to node you are trying to reach
- All switches learn where *you* are
- When destination responds, some switches learn where *it* is…
  - Only some switches, because packet to you follows direct path, and is not flooded
- The decision to flood or not is done on a switch-by-switch basis….

Self-Learning Switch

When a packet arrives

- Inspect *source* ID, associate with *incoming* port
- Store mapping in the switch table
- Use time-to-live field to eventually forget mapping

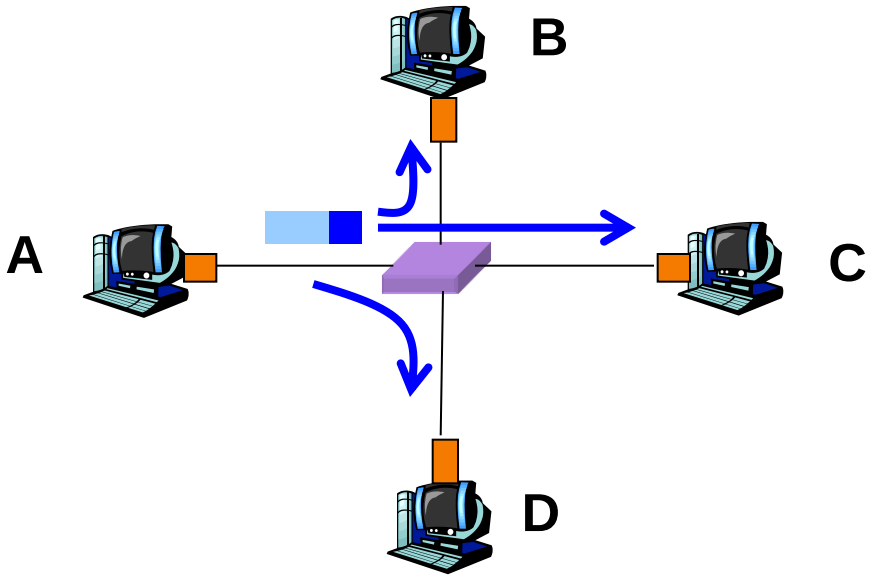
Packet tells switch how to reach A.



Self Learning: Handling Misses

When packet arrives with unfamiliar destination

- Forward packet out **all** other ports
- Response will teach switch about that destination



General Rule

When switch receives a packet:

index the switch table using destination ID

if entry found for destination **{**

if dest on port from which packet arrived
then drop packet

else forward packet on port indicated

}

else flood

Why do this?

forward on all but the interface on which the frame arrived

Summary of Learning Approach

- Avoids loop by restricting to spanning tree
- This makes flooding possible
- Flooding allows packet to reach destination
- And in the process switches learn how to reach source of flood
- No route “computation”

Weaknesses of This Approach?

- Requires loop-free topology (Spanning Tree)
  - Must eliminate many links from physical topology
  - Reducing bisection bandwidth (jargon)
  - Very little control over paths (traffic engineering)
- Slow to react to failures
  - Tree must be recomputed
- Slow to react to host movement
  - Entries must time out
- Spanning Trees suck (just ask an operator)

On to other routing techniques…

- How do we compute loop-free routes in arbitrary topologies?
- Suggestions?

Avoiding Loops

- Central computation
  - Can make sure no loops
- Minimizing metric in distributed computation
  - Loops are never the solution to a minimization problem
  - (for well-behaved metrics)