CS4450

Computer Networks:
Architecture and Protocols

Lecture 10
Fundamentals of Routing
Routing Protocols

Prof. Rachit Agarwal

Announcements

- **Please submit regrade requests for Exam 1 <u>before 11:59PM on Wednesday</u>**

- Problem Set 3 was released last week

- As much as I would like to, I will not be able to predict your grade
    - We grade on a curve
    - Depends on performance across students, and across exams

Goals for Today's Lecture

- Learning about Routing Protocols
  - Link State (Global view, Local computation)
  - Distance Vector (Local view, Local computation)

Recap from last lecture

Recap: Routing using Spanning Trees

- Easy to design routing algorithms for (spanning) trees
  - **Step 1**: Source node “floods” its packet on its spanning tree links
  - **Step 2**: Whenever a node receives a packet:
    - Forwards incoming packet out to all links **other than the one that sent the packet**

- **Amazing properties:**
  - No routing tables needed!
  - No packets will ever loop.
  - At least (and exactly) one packet must reach the destination
    - Assuming no failures

Recap: Why do we need the network layer?

- Spanning Tree Protocol used in switched Ethernet to avoid broadcast storm
- Can be used for routing on the Internet (via “flooding” on spanning tree)
- **Three fundamental issues**:
  - Unnecessary processing at end hosts (that are not the destination)
  - Higher latency
  - Lower available bandwidth

Recap: The right way to think about Routing Tables

- Routing tables are nothing but ….
  - A collection of (directed) spanning tree
  - One for each destination

- **Routing Protocols**
  - Mechanisms to producing routing tables
  - What we will see:
    - “n” spanning tree protocols running in parallel

Questions?

Routing Tables and Routing State

- **Routing table:**
  - Each switch: the next hop for each destination in the network

- **Routing state**: collection of routing tables across all nodes

“Valid Routing Tables” (routing state)

- Global routing state is valid if:
  - it **always** results in deliver packets to their destinations

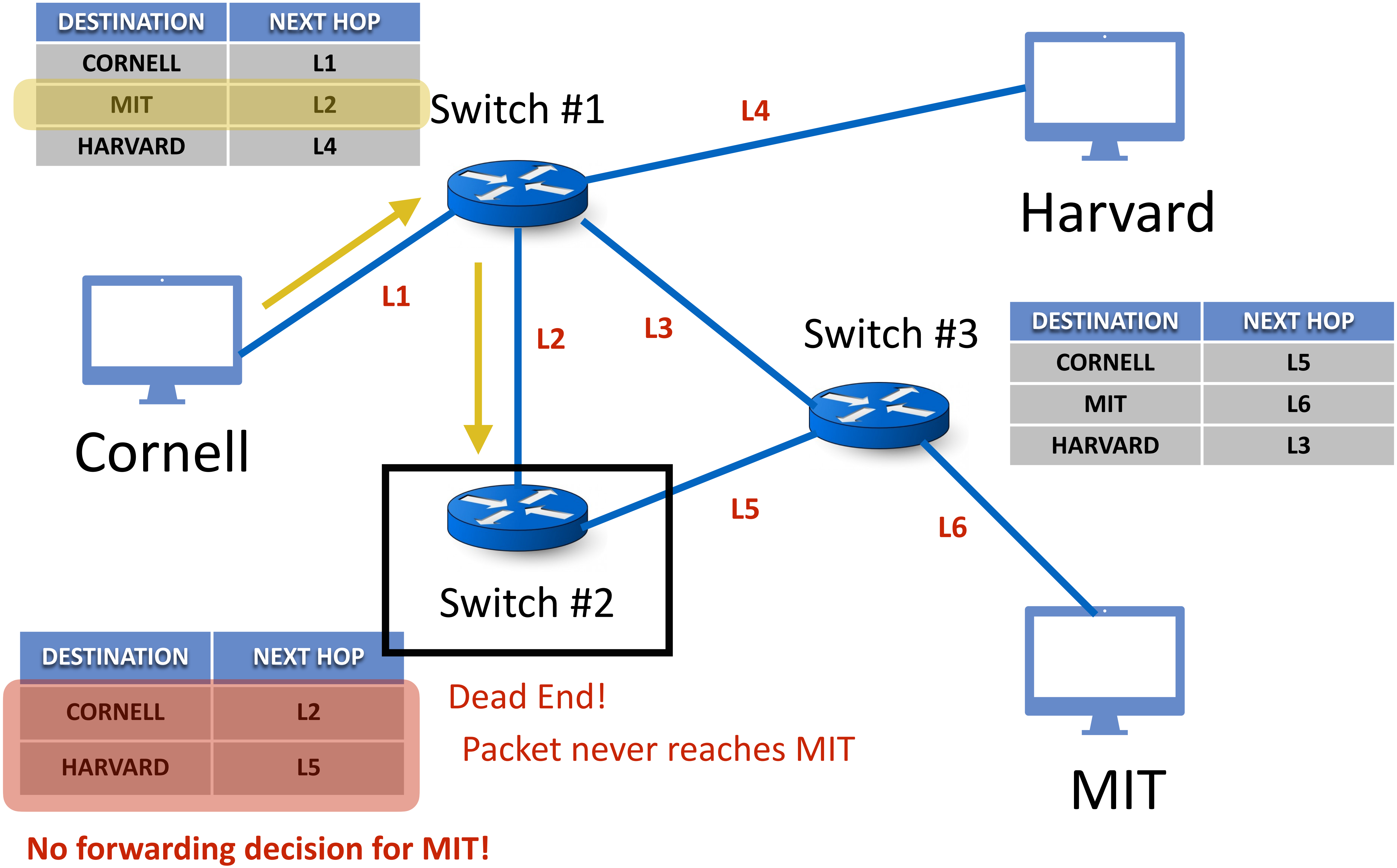
- **Goal of Routing Protocols**
  - Compute a valid state
  - But how to tell if a routing state is valid?…
  - Think about it, what could make routing incorrect?

Validity of a Routing State

- Global routing state 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 link**
  - A packet arrives, but ..
    - the routing table does not have an outgoing link
    - And that node is not the destination
- A **loop** is when a **packet cycles around** the same set of nodes forever
  - There are no **"persistent" loops**
  - **"Transient" loops?**

Example: Routing with Dead Ends

- **Suppose packet wants to go from Cornell to MIT using given state:**



Switch #1:

DESTINATION	NEXT HOP
CORNELL	L1
MIT	L2
HARVARD	L4

Switch #3:

DESTINATION	NEXT HOP
CORNELL	L5
MIT	L6
HARVARD	L3

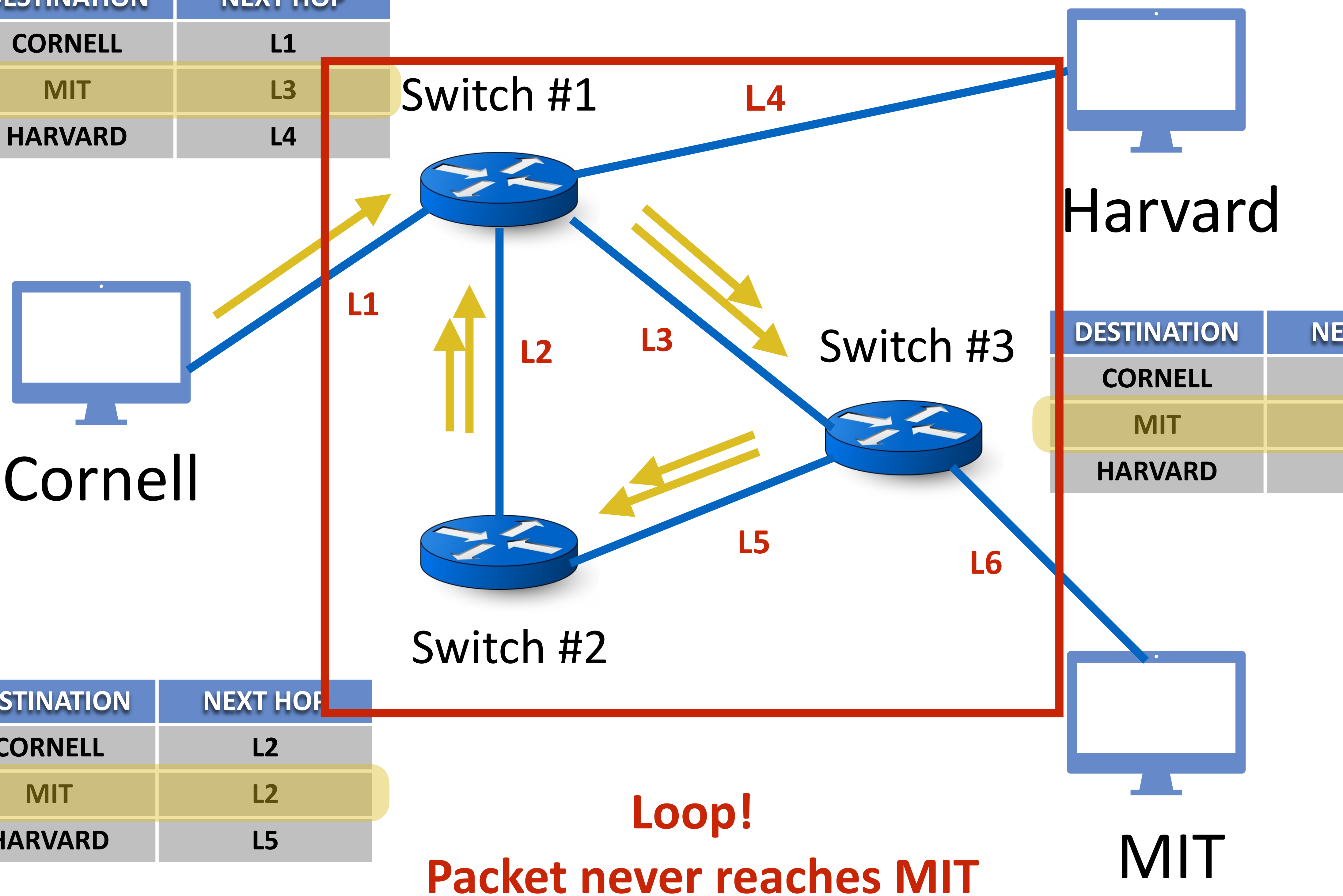
Switch #2:

DESTINATION	NEXT HOP
CORNELL	L2
HARVARD	L5

Example: Routing with Loops

- **Suppose packet wants to go from Cornell to MIT using given state:**

DESTINATION	NEXT HOP
CORNELL	L1
MIT	L3
HARVARD	L4



DESTINATION	NEXT HOP
CORNELL	L5
MIT	L5
HARVARD	L3

DESTINATION	NEXT HOP
CORNELL	L2
MIT	L2
HARVARD	L5

Loop!

Packet never reaches MIT

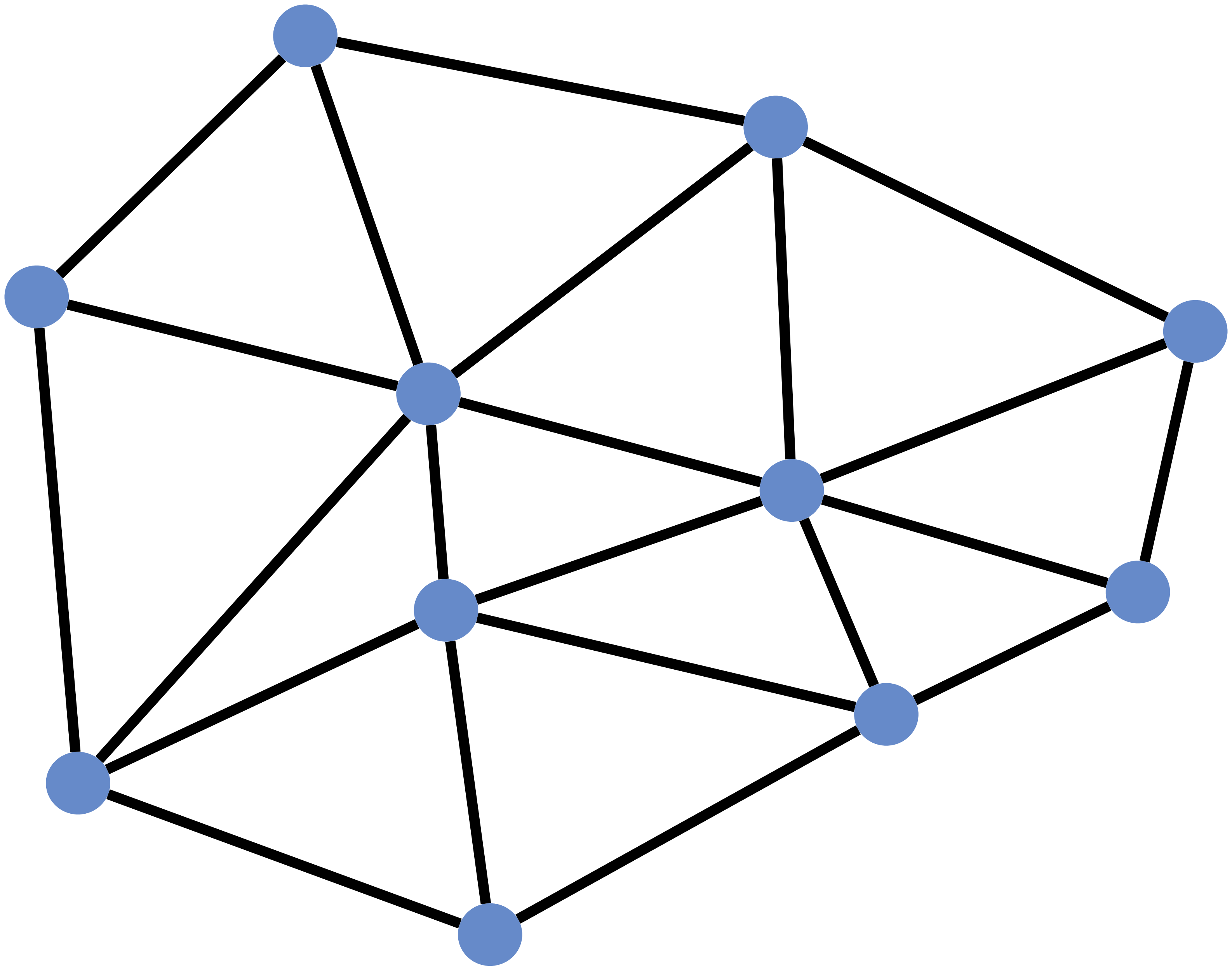
Two Questions

- How can we **verify** given routing state is valid?
- How can we **produce** valid routing state?

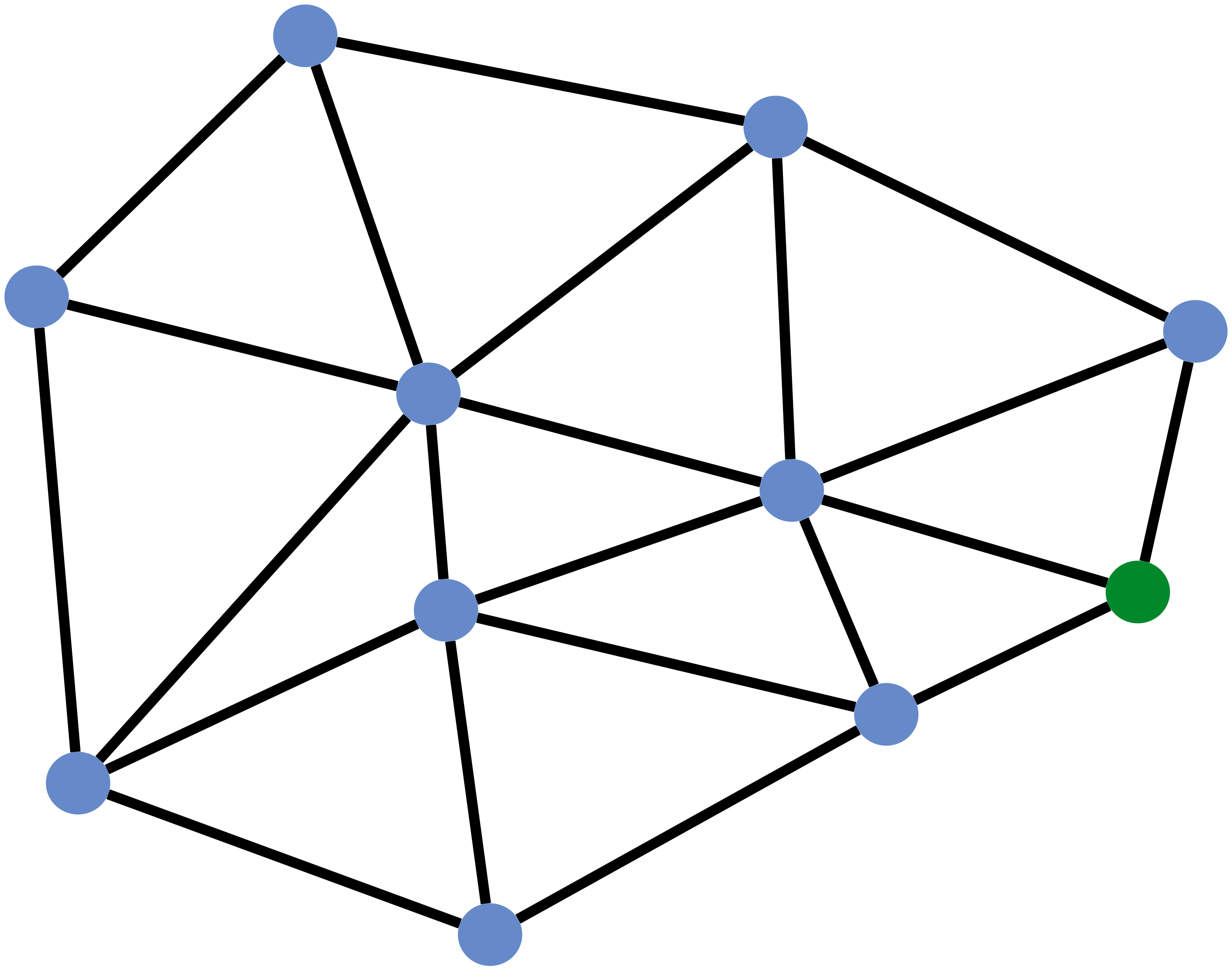
Checking Validity of a Routing State

- Check validity of routing state for one destination at a time…
- For each node:
  - Mark the outgoing link with arrow for the required destination
  - There can only be one at each node
- Eliminate all links with no arrows
- Look what's left. **State is valid if and only if**
  - Remaining graph is a spanning tree with destination as sink
  - Why is this true?
    - Tree -> No loops
    - Spanning (tree) -> No dead ends

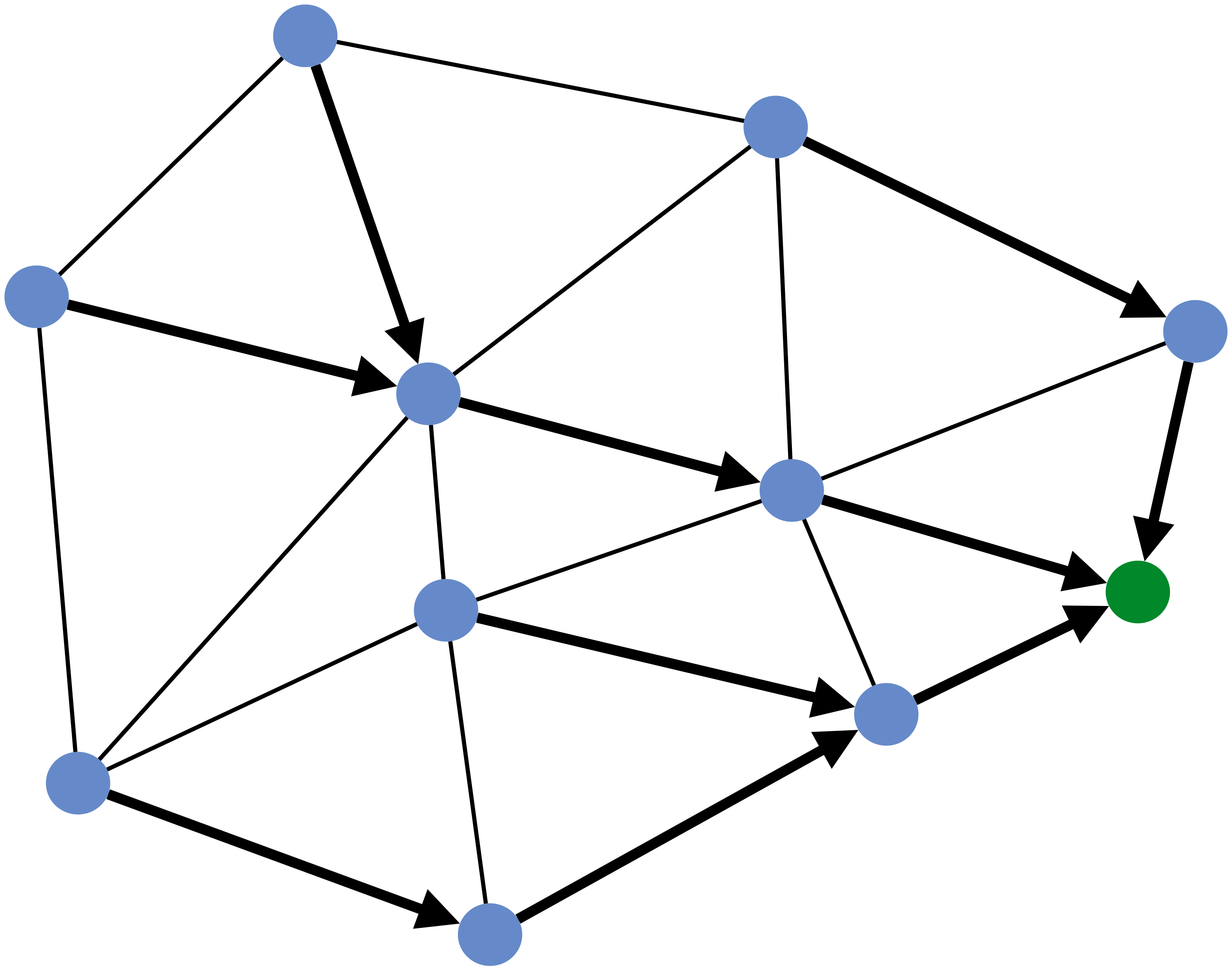
Example 1



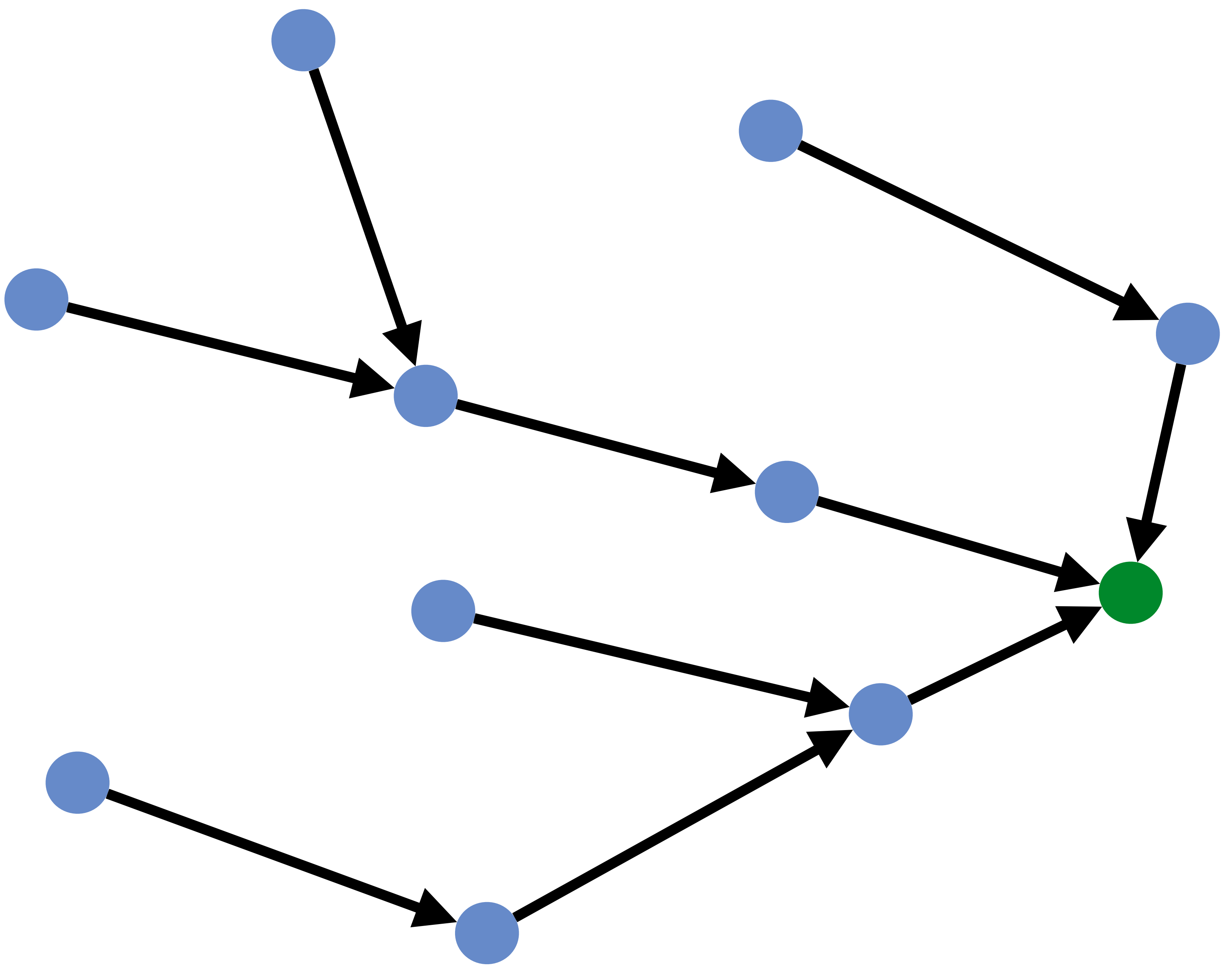
Example 1: Pick Destination



Example 1: Put Arrows on Outgoing Ports

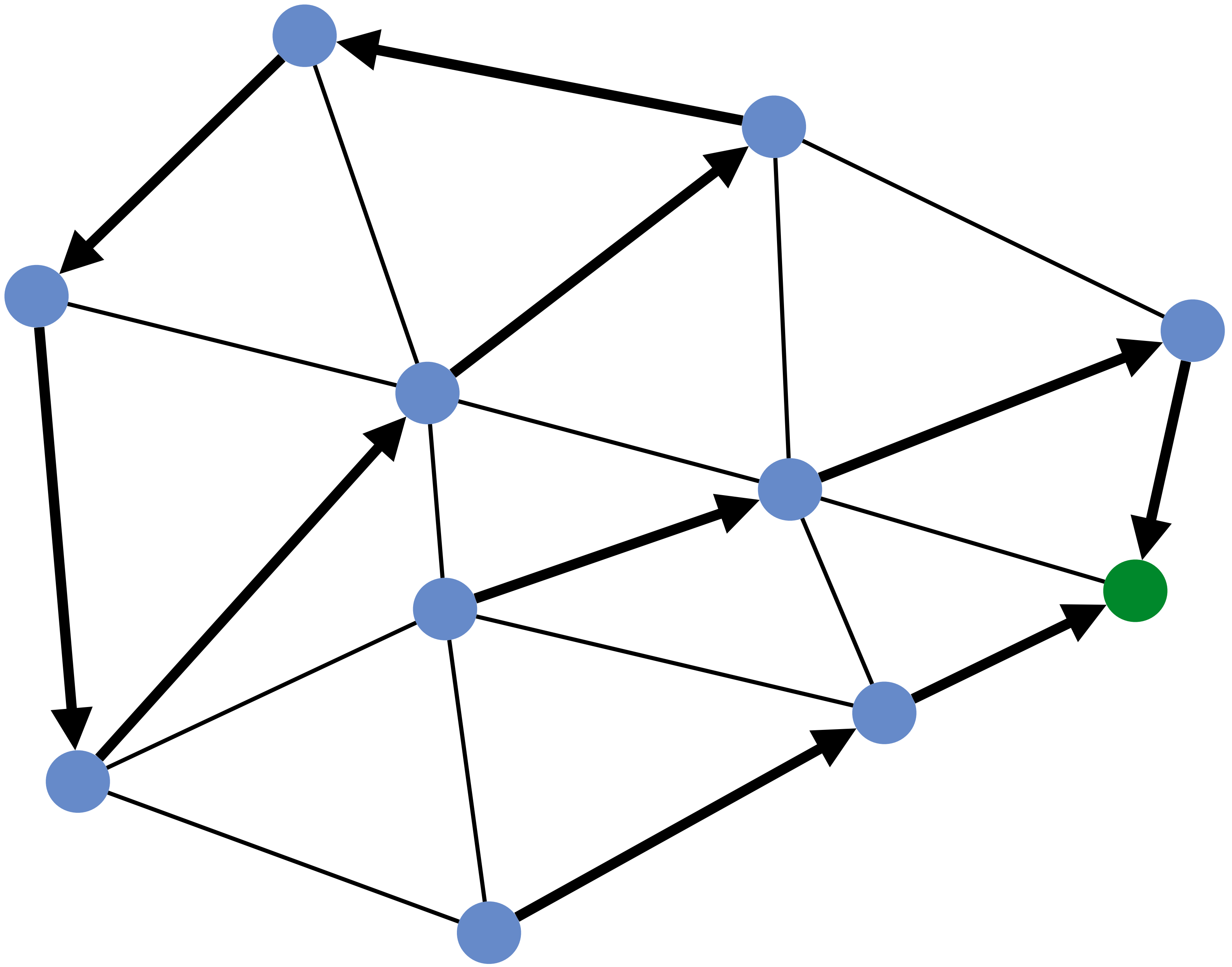


Example 1: Remove unused Links

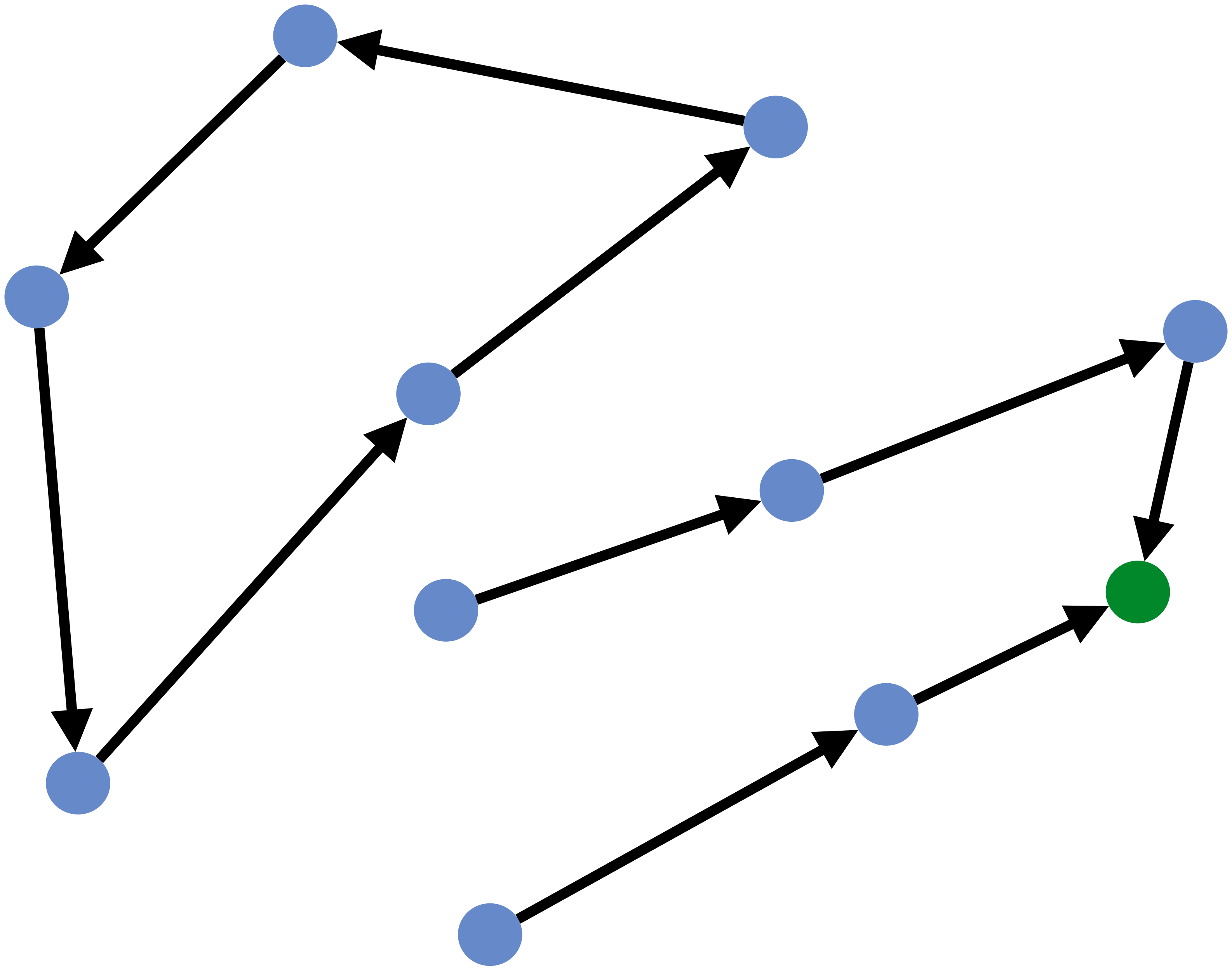


Leaves Spanning Tree: Valid

Example 2:

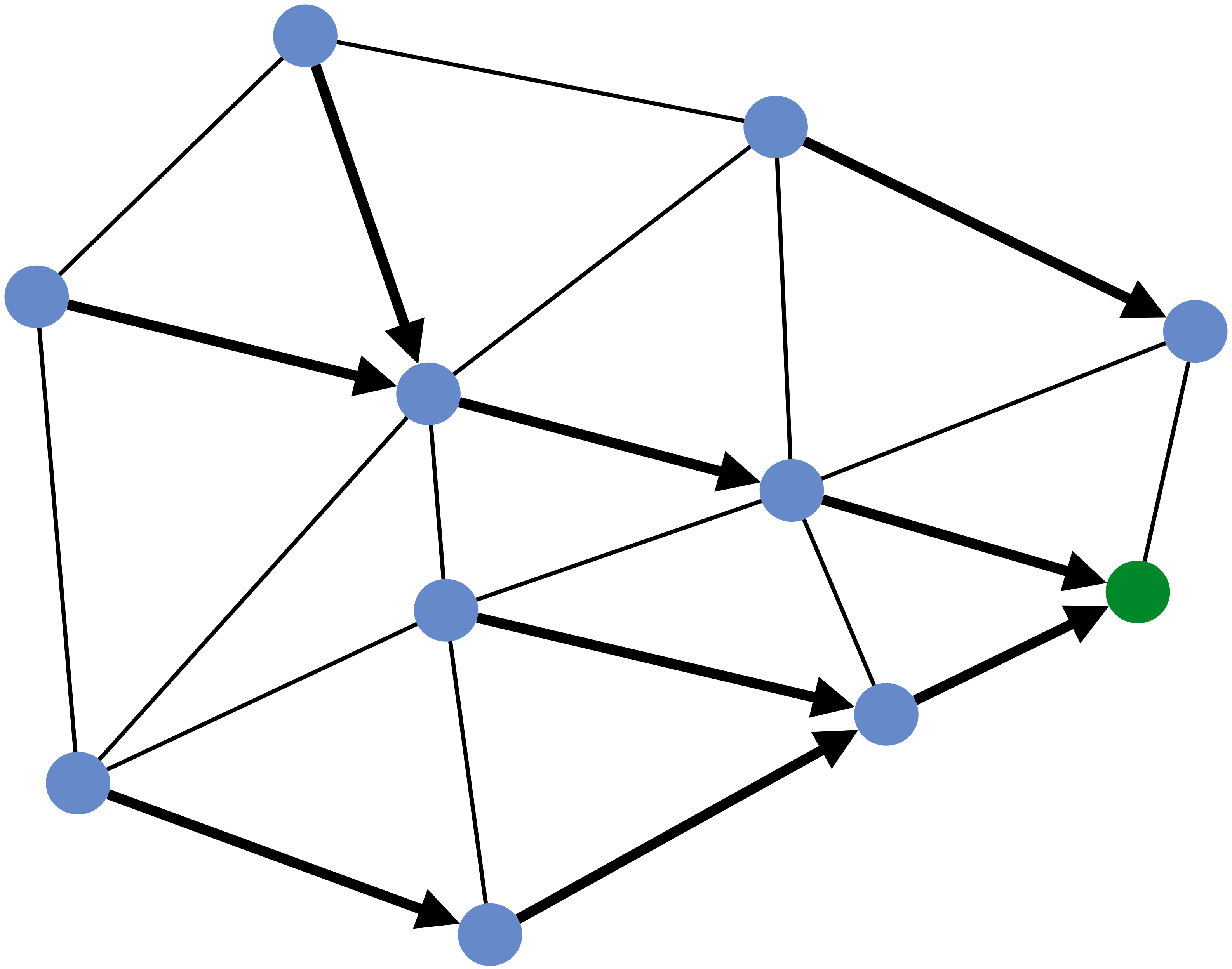


Example 2:

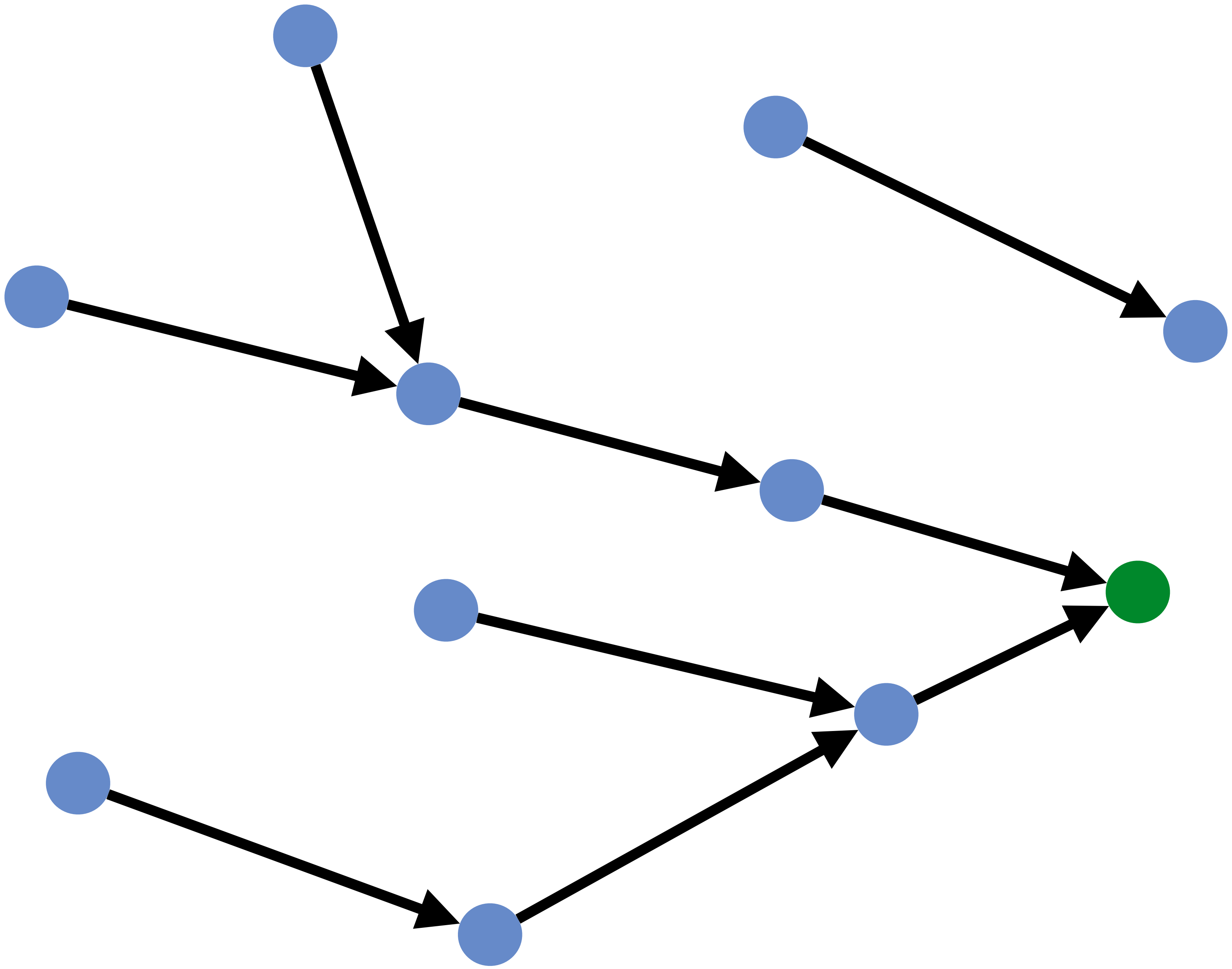


Is this valid?

Example 3:



Example 3:



Is this valid?

Checking Validity of a Routing State

- Simple to check validity of routing state for a particular destination
- Dead ends: nodes without arrows
- Loops: obvious, disconnected from destination and rest of the graph

Two Questions

- How can we **verify** given routing state is valid?

- How can we **produce** valid routing state?

Creating Valid Routing State

- Easy to avoid dead ends

- Avoiding loops is hard

- **The key difference between routing protocols is how they avoid loops!**

Three flavors of protocols

- **Create Tree, route on tree**
  - E.g., Spanning tree protocol (as in switched Ethernet)
  - **Good:** easy, no (persistent) loops, no dead ends
  - **Not-so-good:** unnecessary processing, high latency, low bandwidth
- **Obtain a global view:**
  - E.g., Link state
- **Distributed route computation:**
  - E.g., Distance vector
  - E.g., Border Gateway Protocol

Routing Metrics

- Routing goals: compute paths with minimum X
    - X = number of “hops” (nodes in the middle)
    - X = latency
    - X = weight
    - X = failure probability
    - …
- Generally assume every link has “cost” associated with it
- We want to minimize the cost of the entire path
    - **We will focus on a subset of properties X, where:**
    - **Cost of a path = sum of costs of individual links/nodes on the path**
    - E.g., number of hops and latency

#1: Create a Tree

#1: Create a Tree Out of Topology

- Remove enough links to create a tree containing all nodes
- Sounds familiar? Spanning trees!
- If the topology has no loops, then just make sure not sending packets back from where they came
  - That causes an immediate loop
- Therefore, if no loops in topology and no formation of immediate loops ensures valid routing
- However… three challenges
  - Unnecessary host resources used to process packets
  - High latency
  - Low bandwidth (utilization)

Global view

Two Aspects of Global View Method

- **Protocol**: What we focus on today
    - Where to create global view
    - How to create global view
    - Disseminating route computation (if necessary)
    - When to run route computation
- **Algorithm**: computing loop-free paths on graph
    - Straightforward to compute lowest cost paths
        - Using Dijkstra's algorithm (please study; algorithms course)
    - We won't spend time on this

Where to create global view?

- One option: Central server
  - Collects a global view
  - Computes the routing table for each node
  - “Installs” routing tables at each node
  - **Software-defined Networks: later in course**
- Second option: At each router
  - Each router collects a global view
  - Computes its own routing table using Link-state protocol
- **What does fate sharing principle tells us?**
  - Routing state should be at the routers
- **Link-state routing protocol**
  - OSPF is a specific implementation of link-state protocol
    - IETF RFC 2328 (IPv4) or 5340 (IPv6)

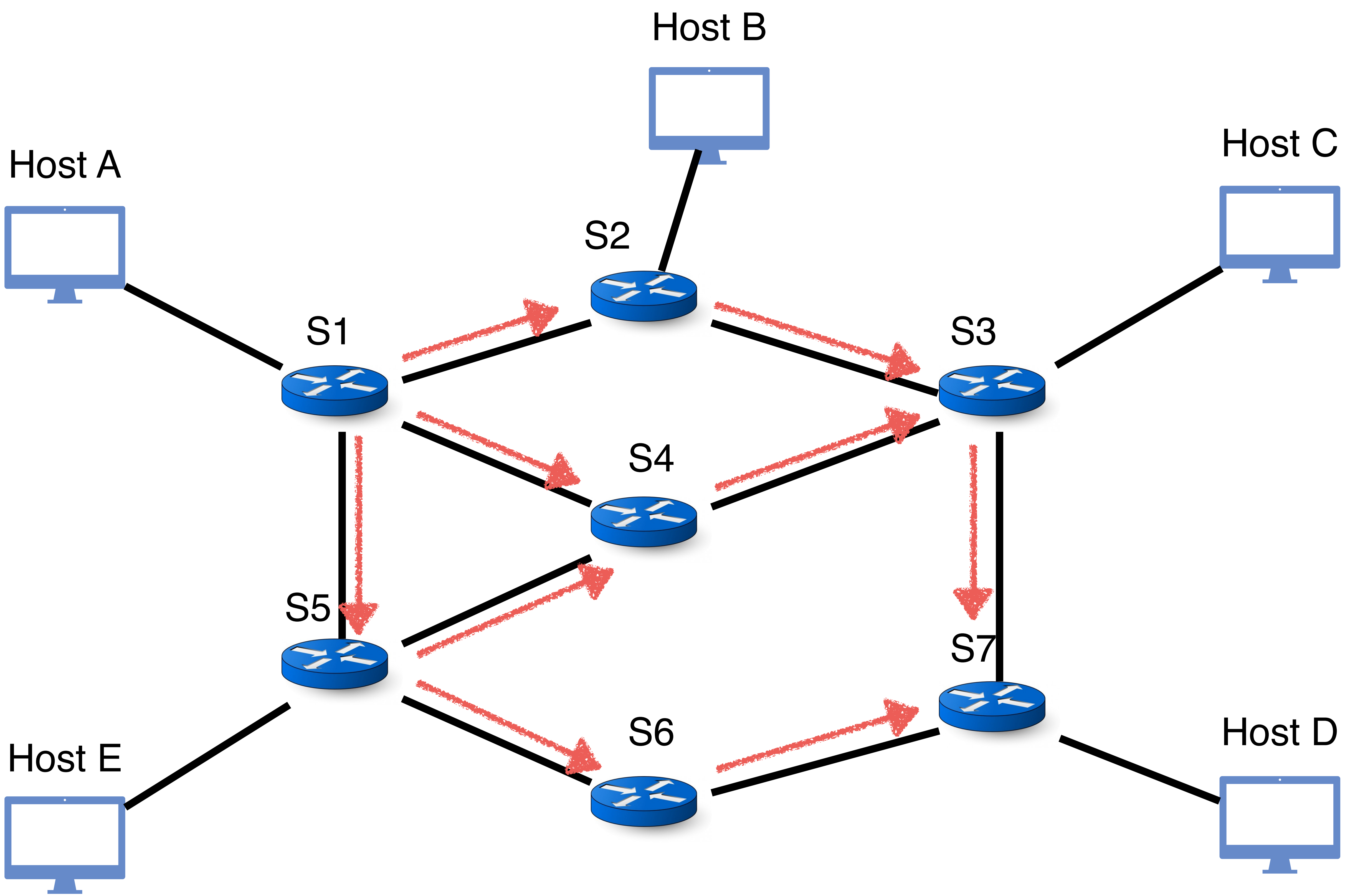
Overview of Link-State Routing

- **Every router knows its local "link state"**
  - Knows state of links to neighbors
  - Up/down, and associated cost
- **A router floods its link state to all other routers**
  - Uses a special packet — Link State Announcements (LSA)
  - Announcement is delivered to all nodes (next slide)
  - Hence, every router learns the entire network graph
- **Runs route computation locally**
  - Computing least cost paths from them to all other nodes
  - E.g., using Dijkstra's algorithm

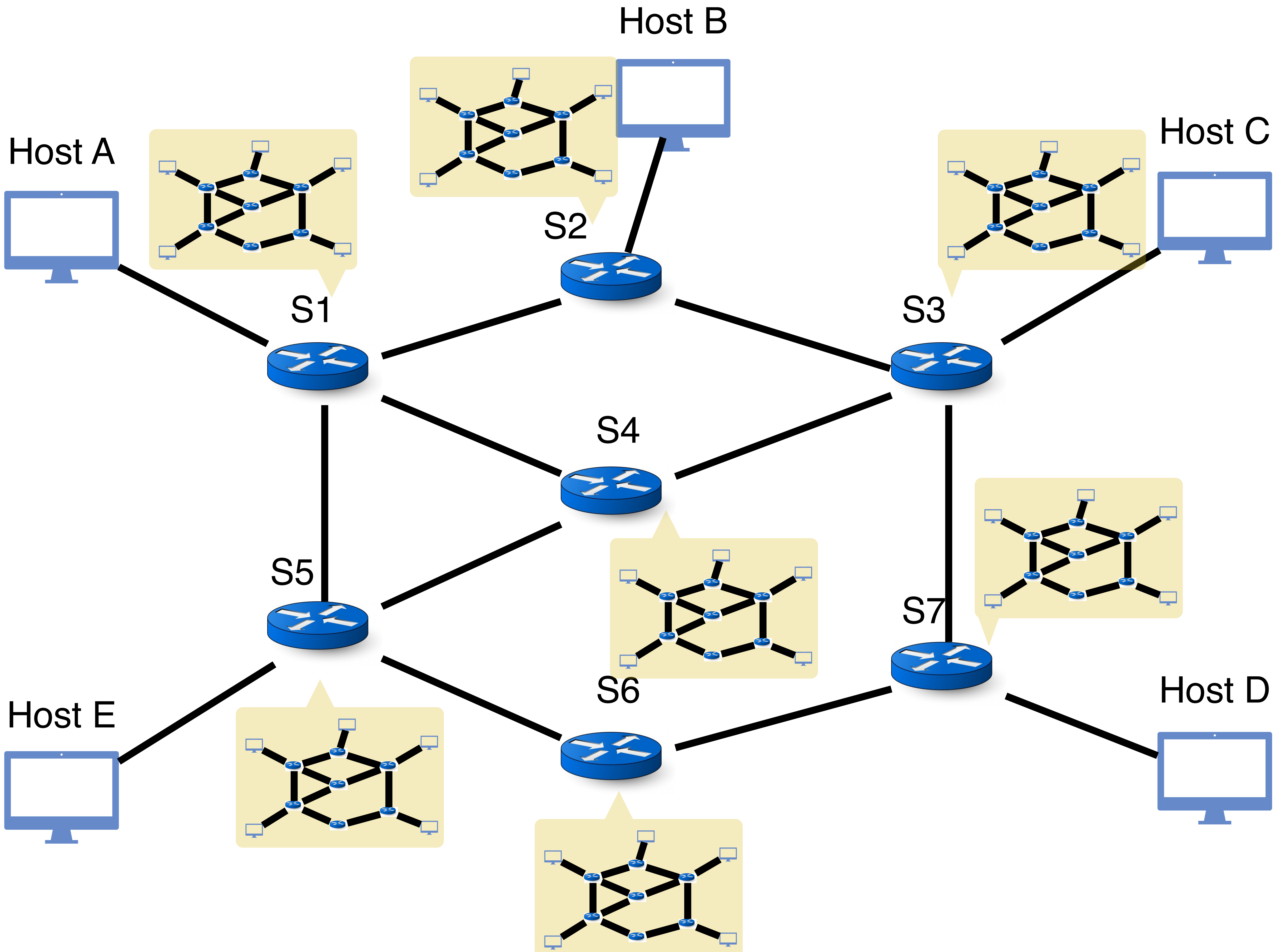
How does Flooding Work?

- "Link state announcement" (LSA) arrives on a link at a router
- That router:
    - Remembers the packet
    - Forwards the packet out all **other links**
    - Does **not** send it out the incoming link
        - Why?
- If a previously received announcement arrives again…
    - Router drops it (no need to forward again)

Link-State Routing



Each Node Then has a Global View



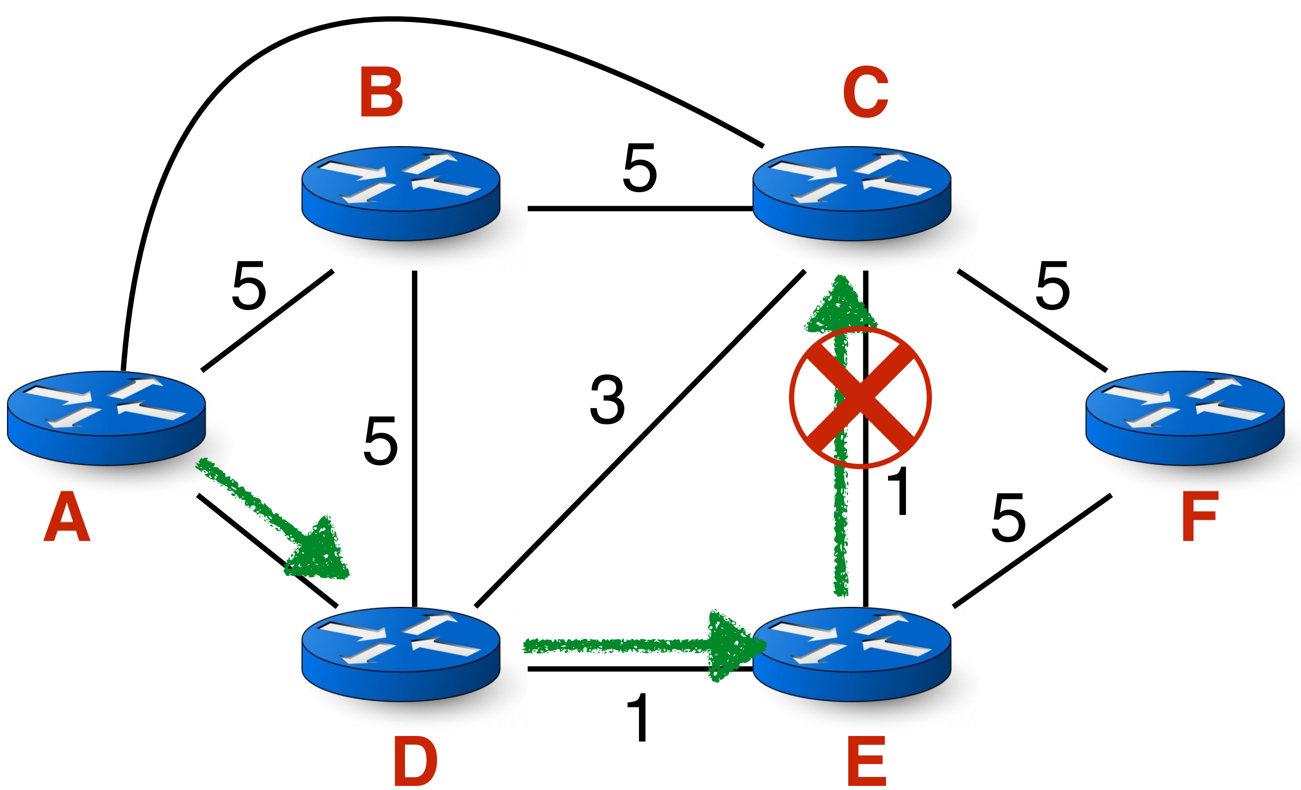
When to Initiate Flooding of announcements?

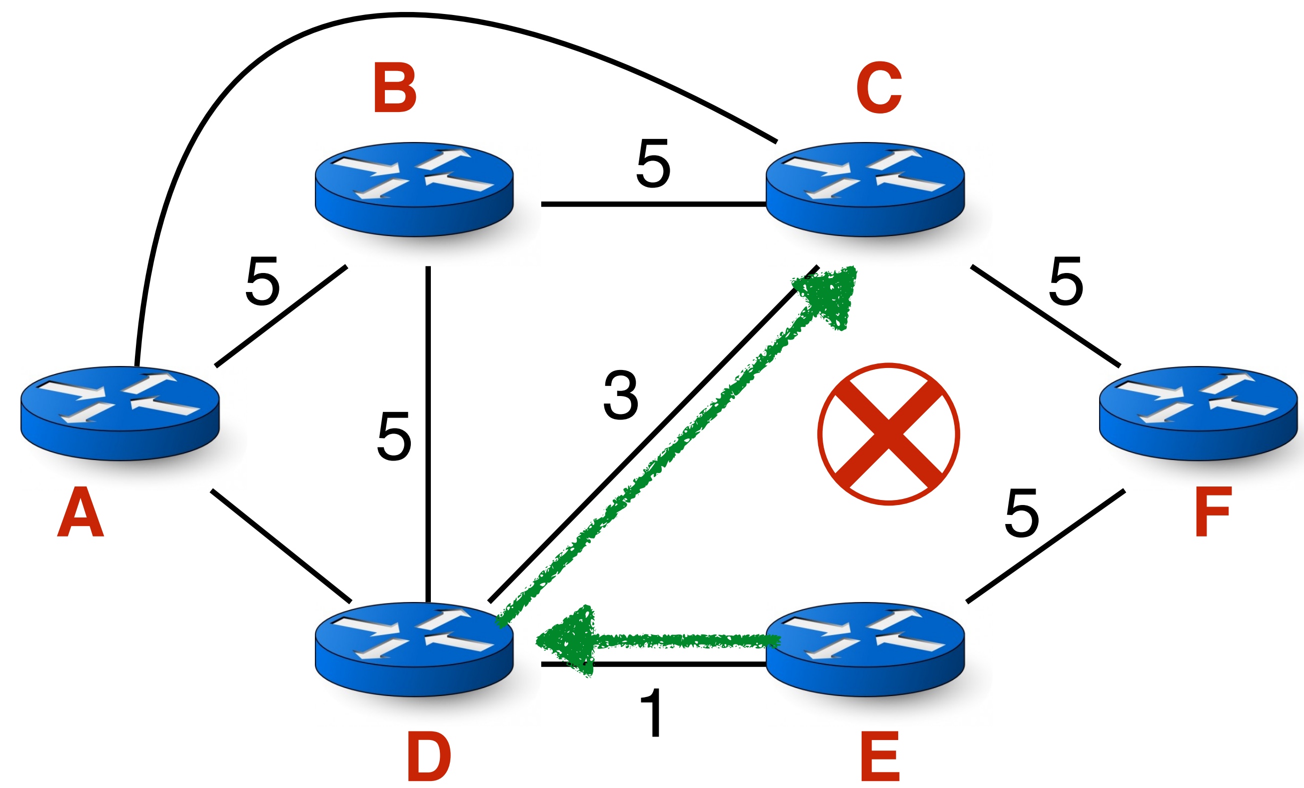
- **Topology change**
  - Link failures
  - Link recovery
- **Configuration change**
  - Link cost change (why would one change link cost?)
- **Periodically**
  - Refresh the link-state information
  - Typically (say) 30 minutes
  - Corrects for possible corruption of data

Making Floods Reliable

- Reliable Flooding
    - Ensure all nodes receive same link state announcements
        - No announcements dropped
    - Ensure all nodes use the latest version
- Suppose we can implement reliable flooding. How can it still fail?
- Can you ever have loops with link-state routing?
- **Again: Can you ever have loops with link-state routing?**

Are Loops Still Possible?





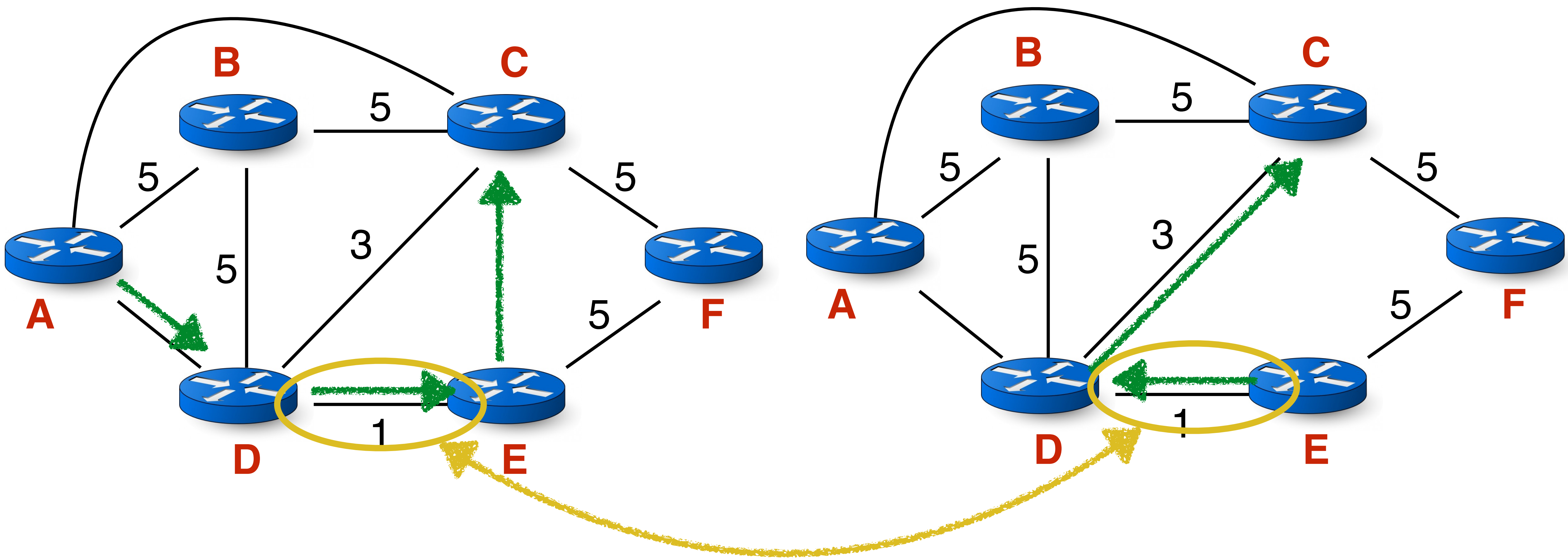
A and D think this is the path to C

E-C link fails, but D doesn't know yet

E thinks that this the path to C

E reaches C via D, D reaches C via E
Loop!

Transient Disruptions



- Inconsistent link-state views
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause **transient forwarding loops**
    - **Transient loops are still a problem!**

Convergence

- **Eventually**, all routers have consistent routing information
    - E.g., all nodes having the same link-state database
    - Here, eventually means “if nothing changes after a while”

- Forwarding is consistent after convergence
    - All nodes have the same link-state database
    - All nodes forward packets on same paths

- **But while still converging, bad things can happen**

Time to Reach Convergence

- Sources of convergence delay?
    - Time to detect failure
    - Time to flood link-state information (~longest RTT)
    - Time to recompute forwarding tables
- Performance problems during convergence period?
    - Dead ends
    - Looping packets
    - And some more we'll see later ….

Link State is Conceptually Simple

- Everyone floods links information
- Everyone then knows graph of the network
- Everyone independently computes paths on the graph
- **All the complexity is in the details**