CS4450

Computer Networks:
Architecture and Protocols

Lecture 10
Fundamentals of Routing
Routing Protocols

Rachit Agarwal

Announcements

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

- Problem Set 3 is released

- Reminder: this class has 3 programming assignments
    - Mostly in late October and November

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: Routing Tables

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

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

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

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

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 valid routing tables
    - What we will see:
        - "n" spanning tree protocols running in parallel

Questions?

Creating Valid Routing State

- Easy to avoid dead ends

- Avoiding loops is hard

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

Four 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

- **Link-state routing protocol**
    - OSPF is a specific implementation of link-state protocol
        - IETF RFC 2328 (IPv4) or 5340 (IPv6)

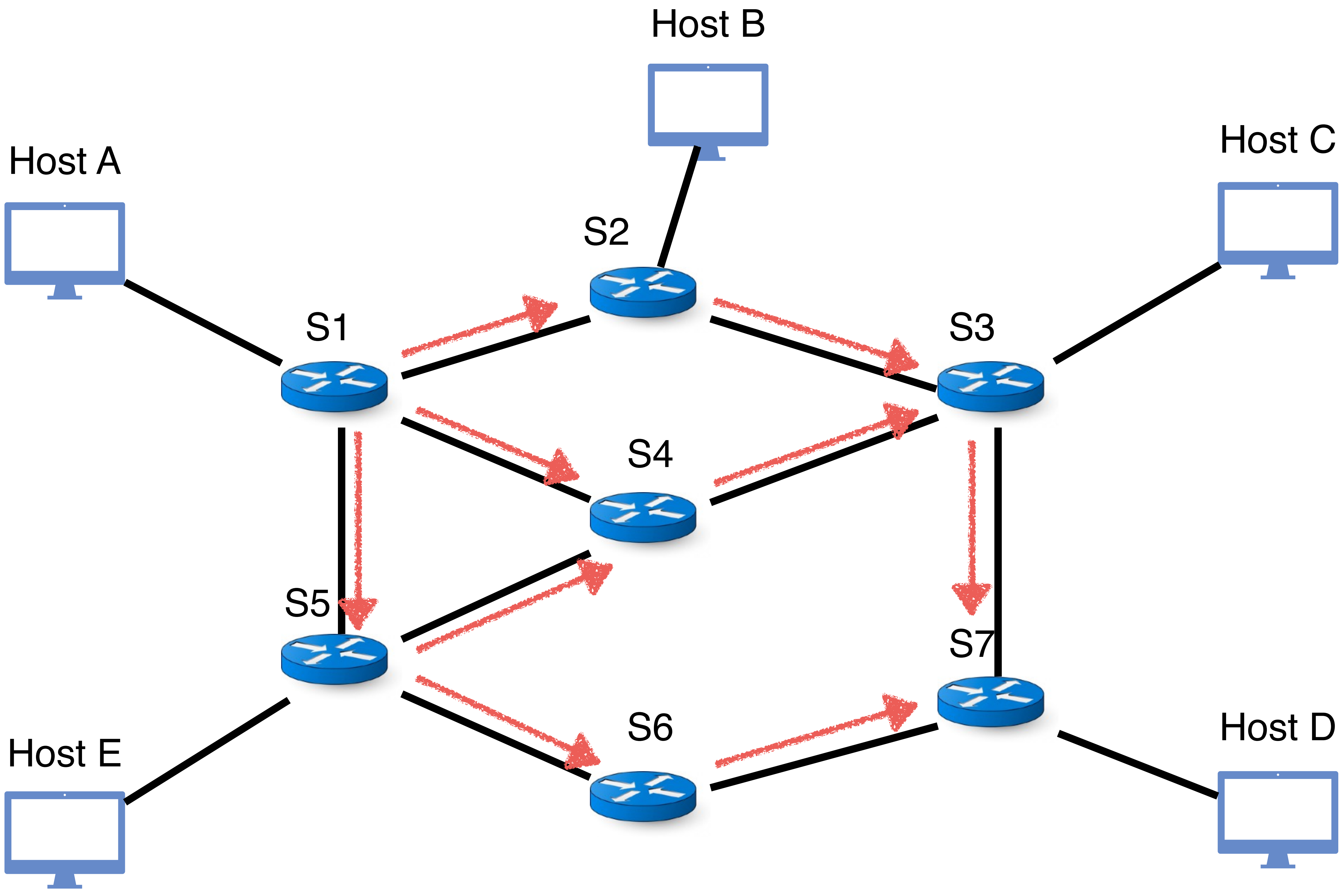
Overview of Link-State Routing

- **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

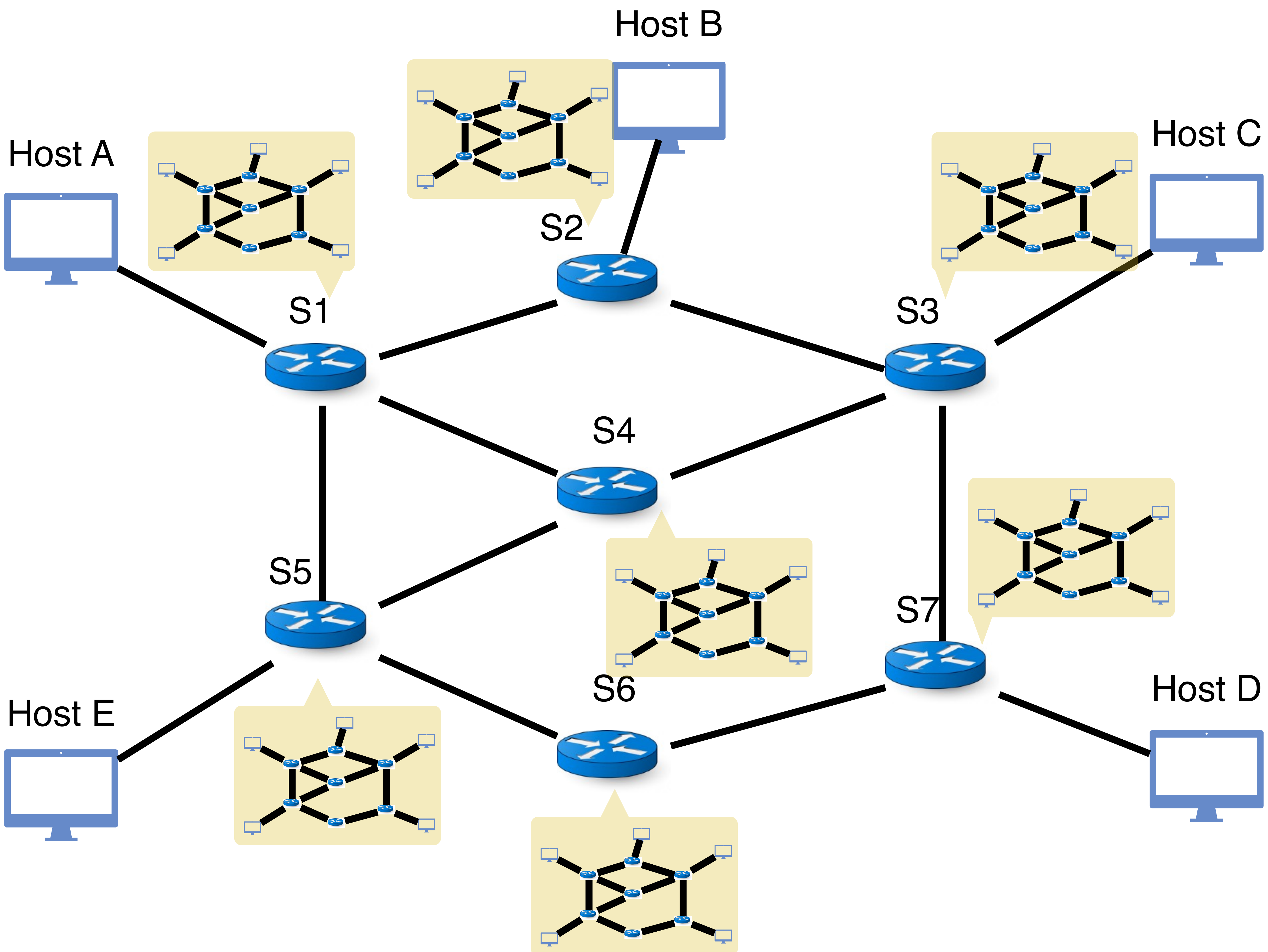
How does Flooding Work?

- "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

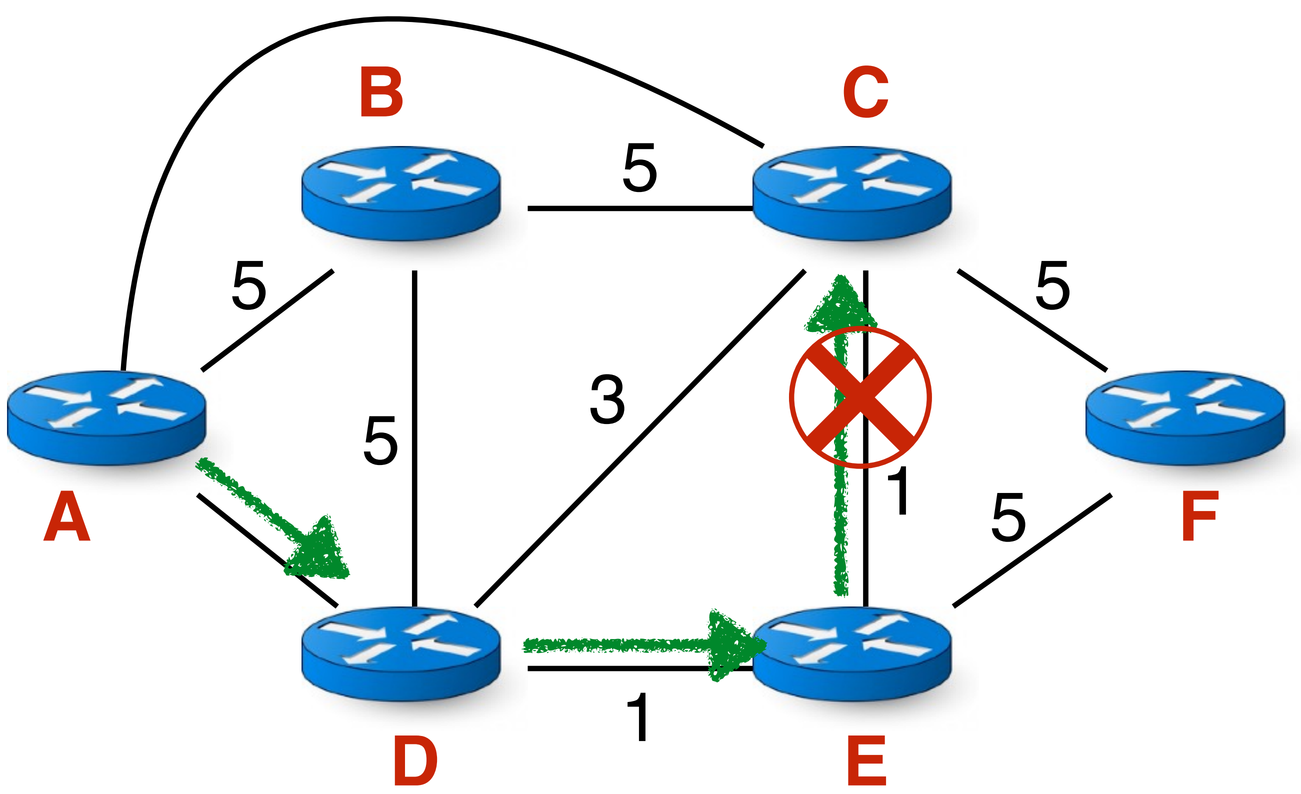
- Reliable Flooding
    - Ensure all nodes receive same link state announcements
        - No announcements dropped
    - Ensure all nodes use the latest version

- Suppose we can implement reliable flooding. How can it still fail?

- Can you ever have loops with link-state routing?

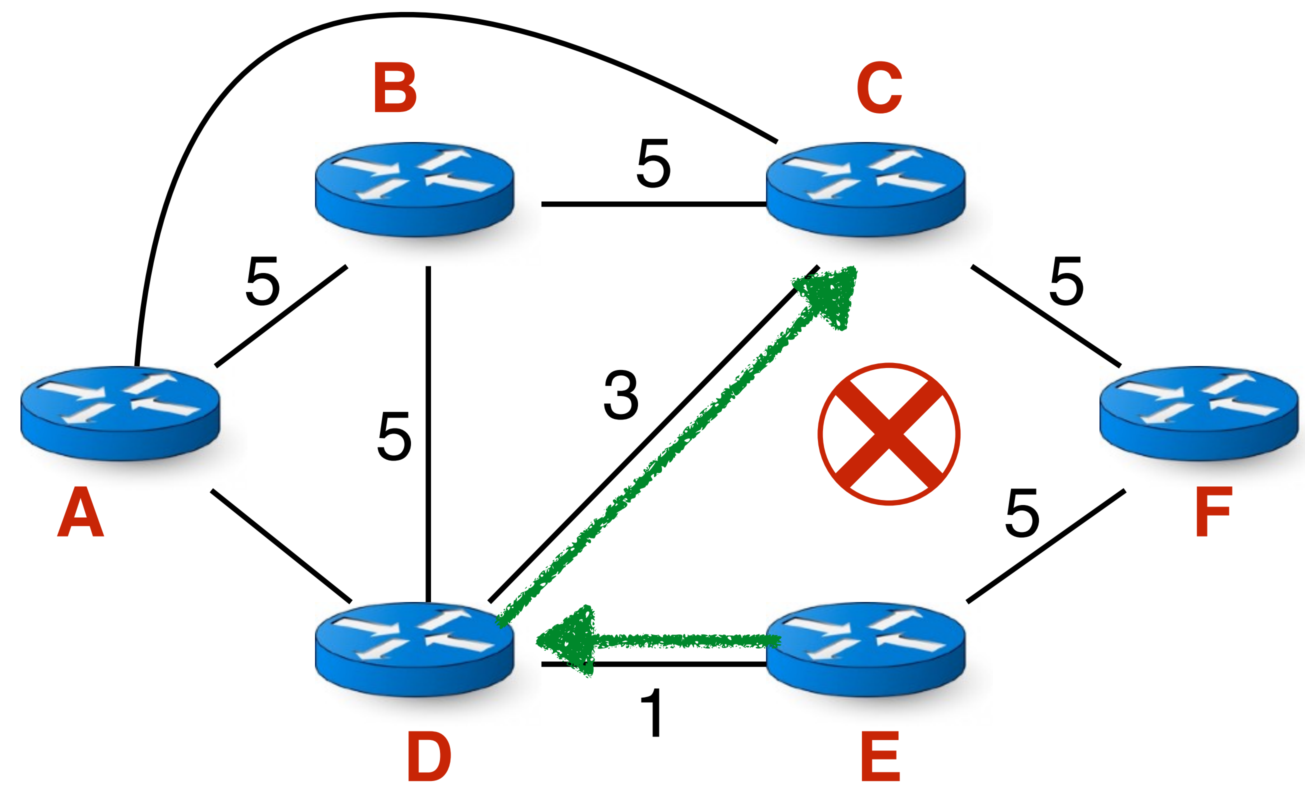
- **Again: Can you ever have loops with link-state routing?**

Are Loops Still Possible?



A and D think this is the path to C

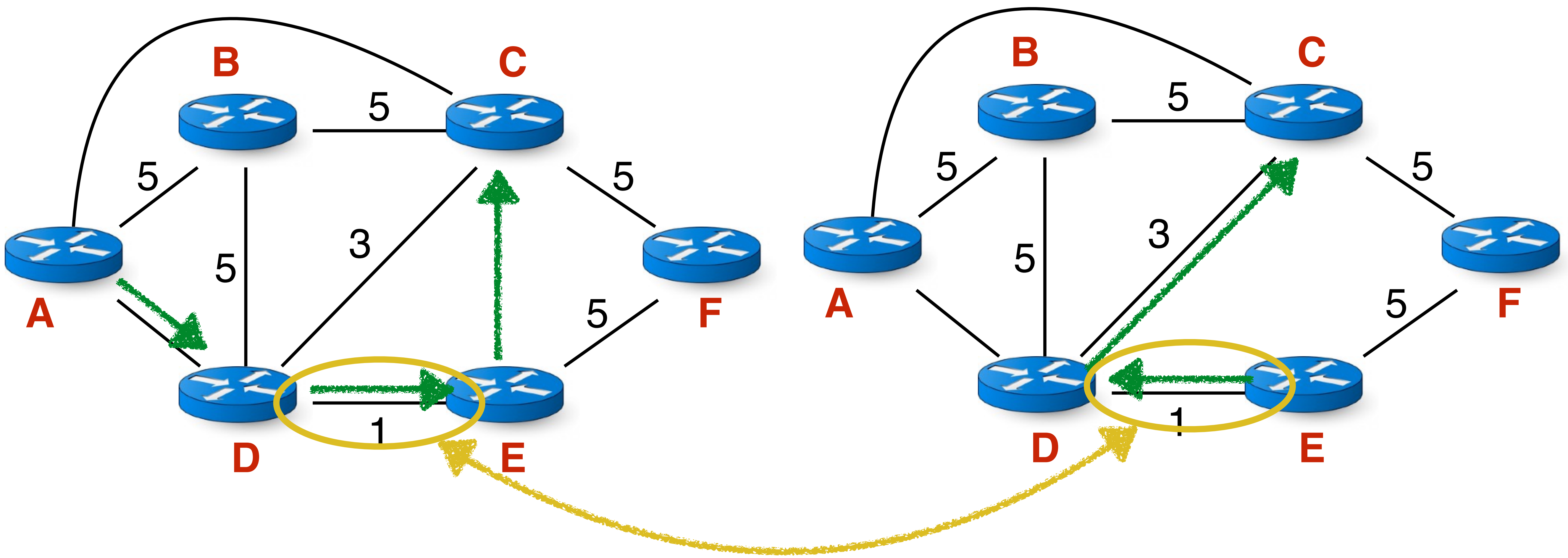
E-C link fails, but D doesn't know yet



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**

Local view, distributed route computation

#3: Distributed Route Computation

- Often getting a global view of the network is infeasible
    - Distributed algorithms to compute feasible route

- **Approach A**: Finding optimal route for maximizing/minimizing a metric

- **Approach B**: Finding feasible route via exchanging paths among switches

Distributed Computation of Routes

- Each node computes the outgoing links (for each destination) based on:
  - Local link costs
  - Information advertised by neighbors
- Algorithms differ in what these exchanges contain
  - **Distance-vector**: just the distance (and next hop) to each destination
  - **Path vector**: the entire path to each destination
- We will focus on distance-vector for now

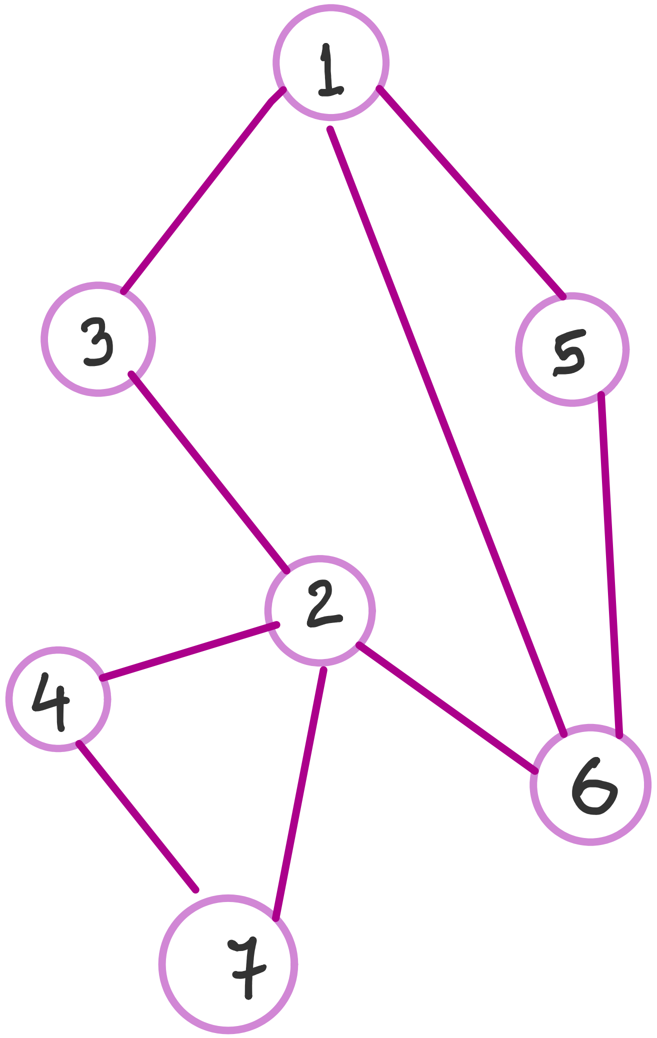
Recall: Routing Tables = Collection of Spanning Trees

- **Can we use the spanning tree protocol (with modifications)?**
- **Messages (Y,d,X): For root Y; From node X; advertising a distance d to Y**
- Initially each switch X announces (X,0,X) to its neighbors

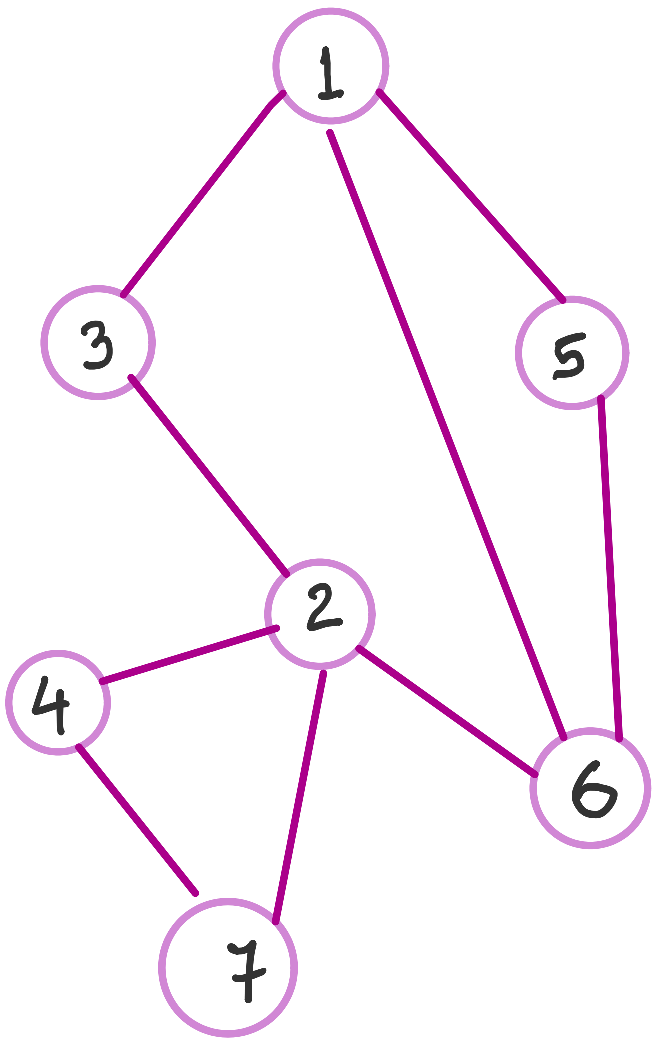
Distance vector: a collection of “n” STP in parallel

Lets run the Protocol on this example

(destination = 1)

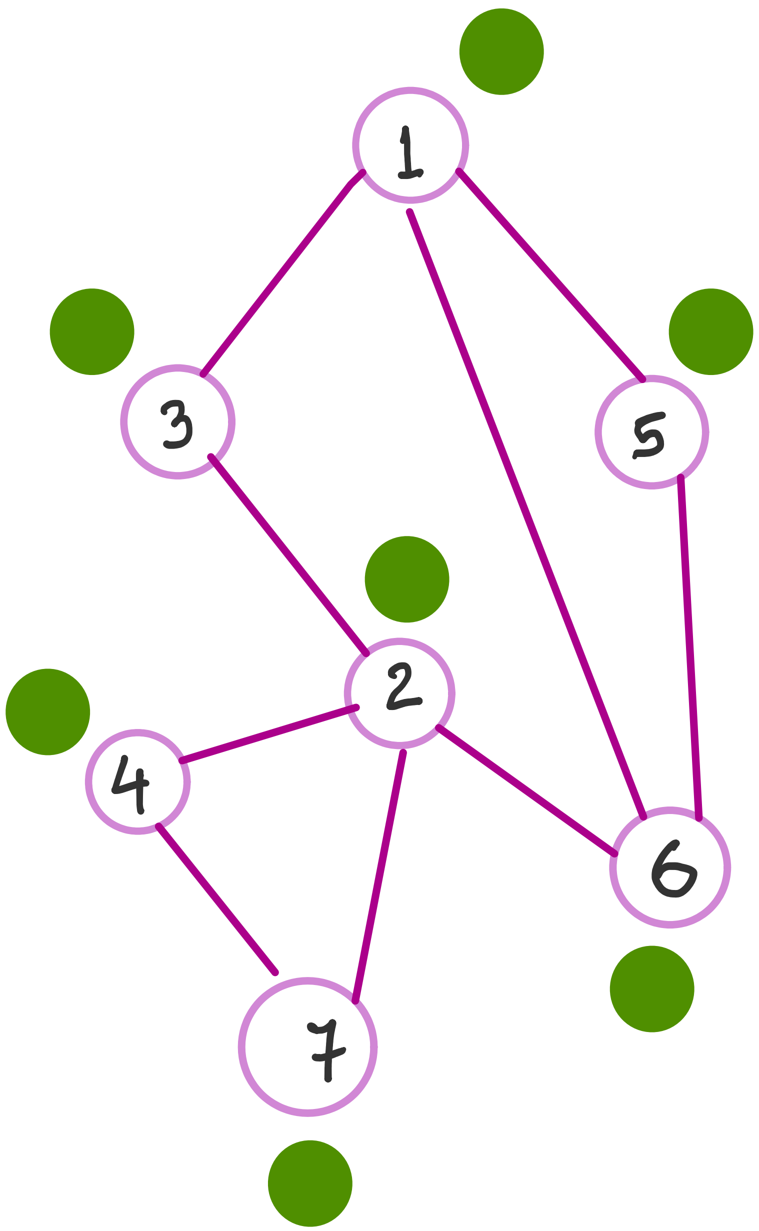


Round 1



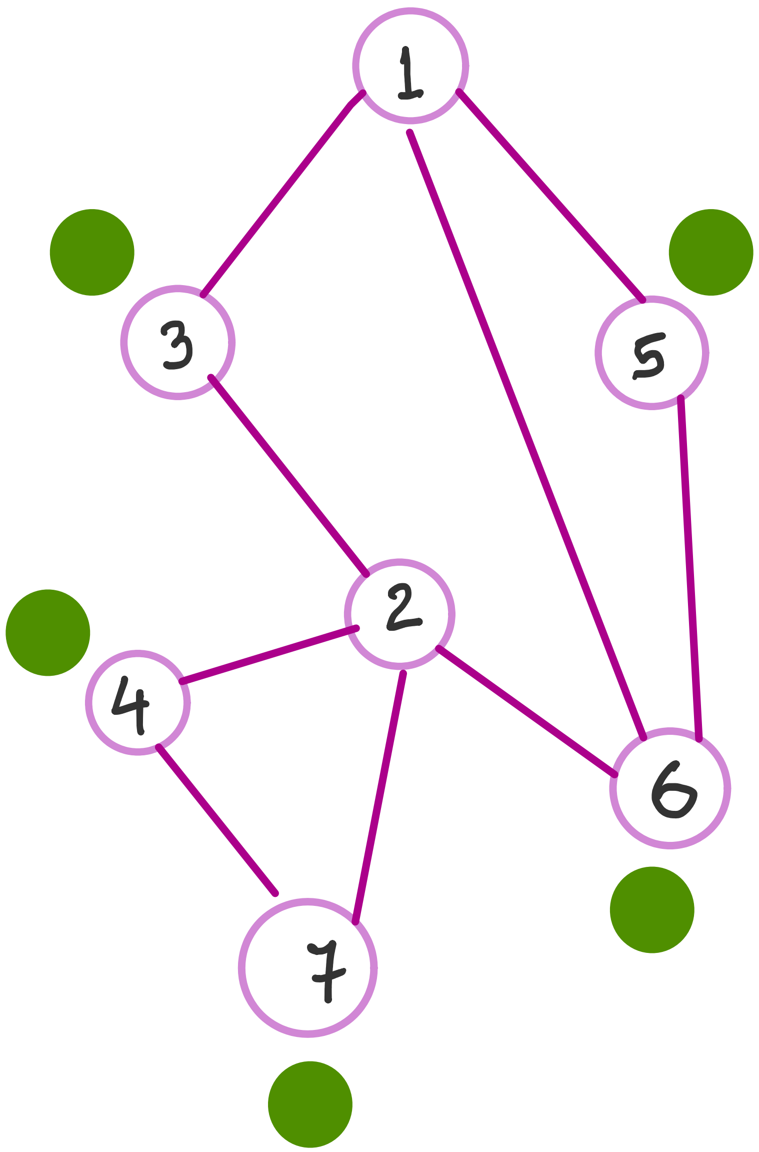
	Receive	Send
1		(1, 0, 1)
2		
3		
4		
5		
6		
7		

Round 2



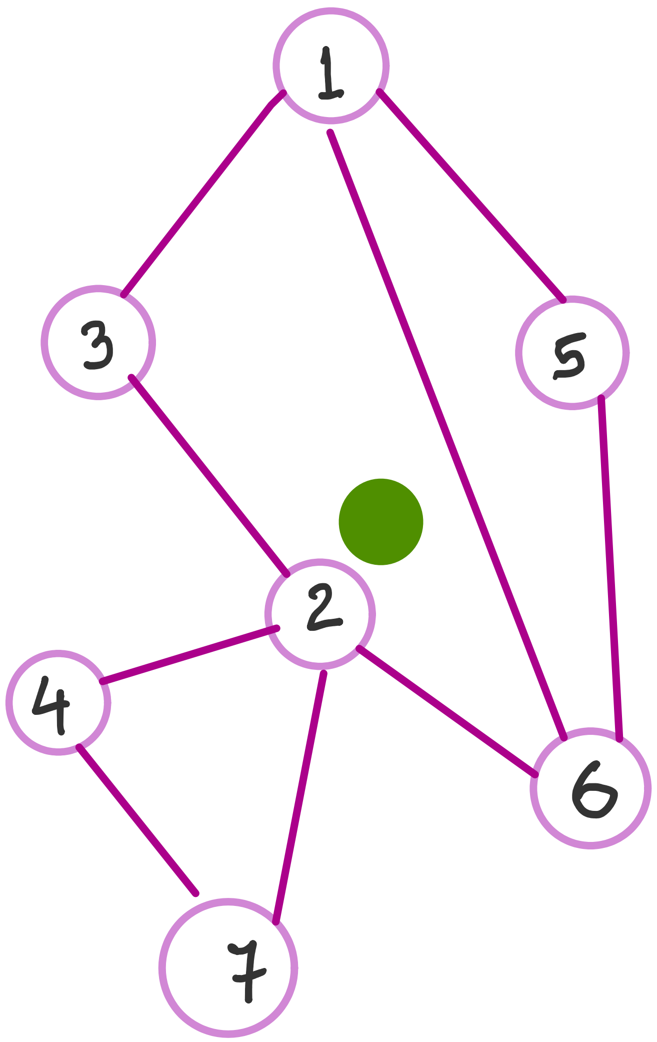
	Receive	Send
1 (1, 0, 1)		
2		
3	(1, 0, 1)	**(1, 1, 3)**
4		
5	(1, 0, 1)	**(1, 1, 5)**
6	(1, 0, 1)	**(1, 1, 6)**
7		

Round 3



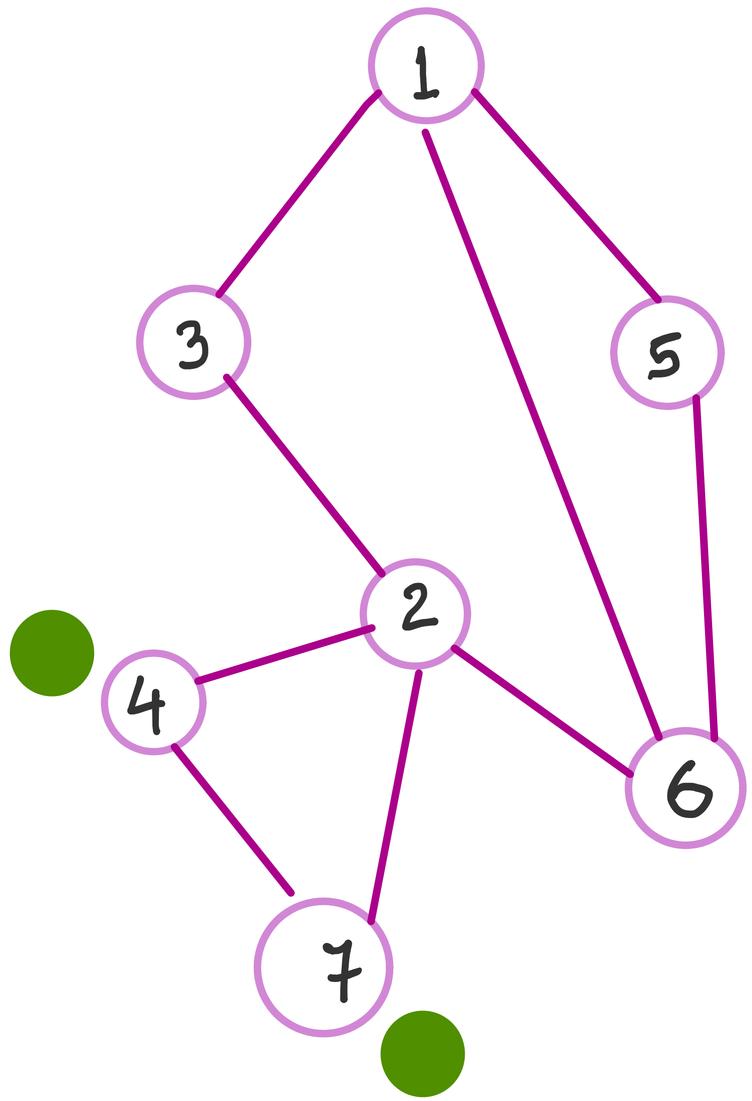
	Receive	Send
1 (1, 0, 1)	(1, 1, 3), (1, 1, 5), (1, 1, 6)	
2	(1, 1, 3), (1, 1, 6)	**(1, 2, 2)**
3 (1, 1, 3)		
4		
5 (1, 1, 5)	(1, 1, 6)	
6 (1, 1, 6)	(1, 1, 5)	
7		

Round 4



	Receive	Send
1 (1, 0, 1)		
2 (1, 2, 2)		
3 (1, 1, 3)	(1, 2, 2)	
4	(1, 2, 2)	**(1, 3, 4)**
5 (1, 1, 5)		
6 (1, 1, 6)	(1, 2, 2)	
7	(1, 2, 2)	**(1, 3, 7)**

Round 5



	Receive	Send
1 (1, 0, 1)		
2 (1, 2, 2)	(1, 3, 4), (1, 3, 7)	
3 (1, 1, 3)		
4 (1, 3, 4)	(1, 3, 7)	
5 (1, 1, 5)		
6 (1, 1, 6)		
7 (1, 3, 7)	(1, 3, 4)	

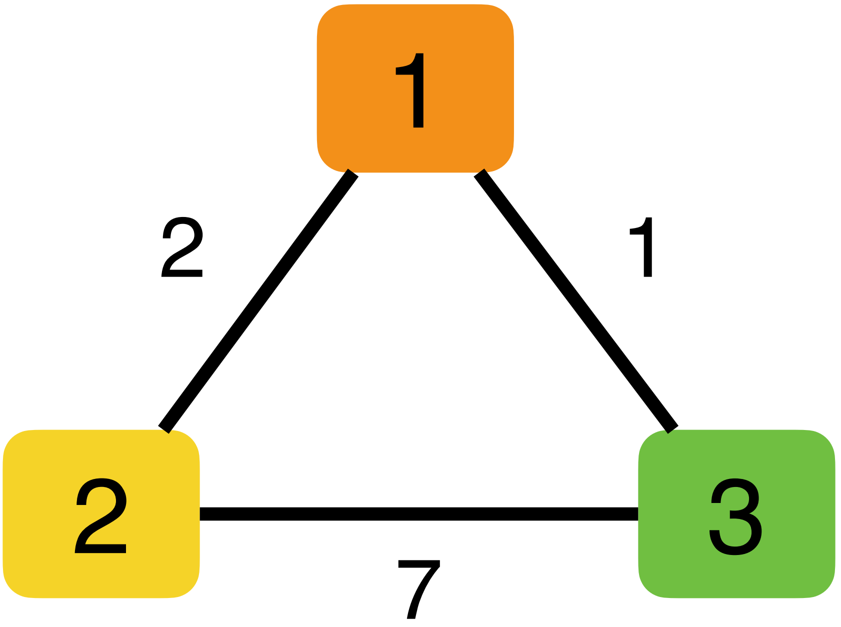
Why not Spanning Tree Protocol? Why Distance "Vector"?

- The same protocol/algorithm applies to all destinations
- Each node announces distance to **each** dest
    - I am 4 hops away from node A
    - I am 6 hops away from node B
    - I am 3 hops away from node C
    - …
- Nodes are exchanging a **vector** of distances

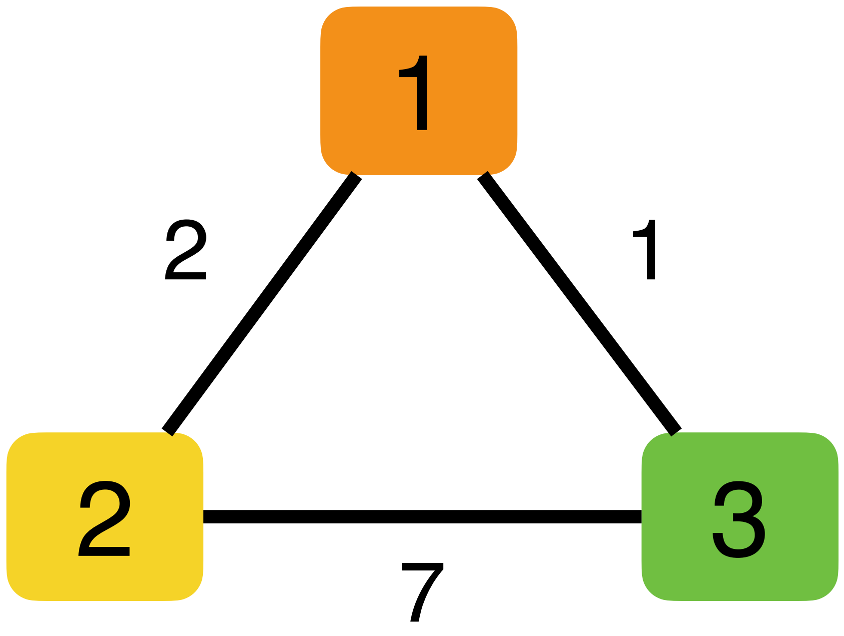
Towards Distance Vector Protocol (with no failures)

- **Messages (Y,d,X): For root Y; From node X; advertising a distance d to Y**
- Initially each switch X announces (X,0,X) to its neighbors
- Switch X updates its view
  - Upon receiving message (Y,d,Z) from Z~~, check Y's id~~
  - ~~If Y's id < current root: set root destination = Y~~
- Switch X computes its shortest distance from the ~~root~~ destination
  - If current_distance_to_Y > d + cost of link to Z:
    - update current_distance_to_Y = d + cost of link to Z
- If ~~root changed OR~~ shortest distance to the ~~root~~ destination changed, send all neighbors updated message (Y, current_distance_to_Y, X)

Lets run the Protocol on this example

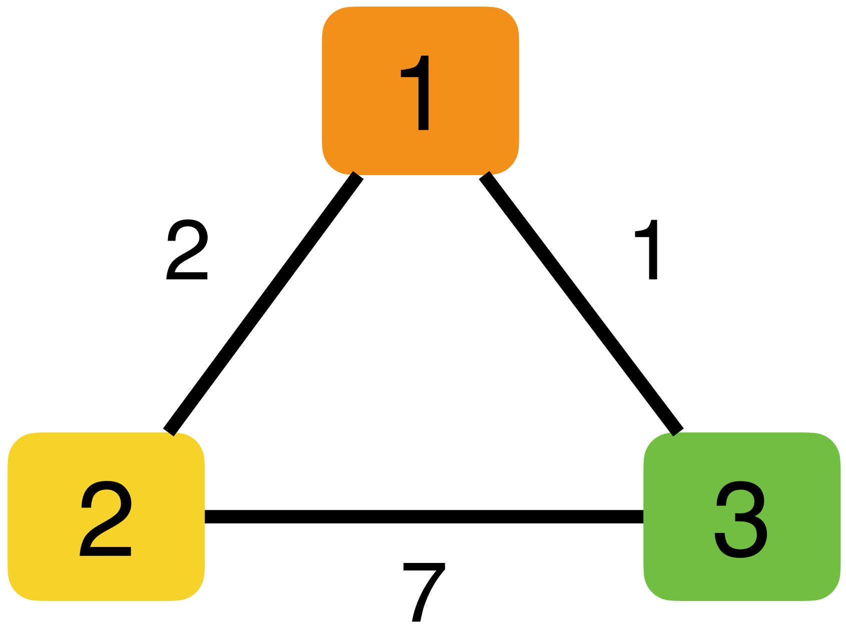


Round 1



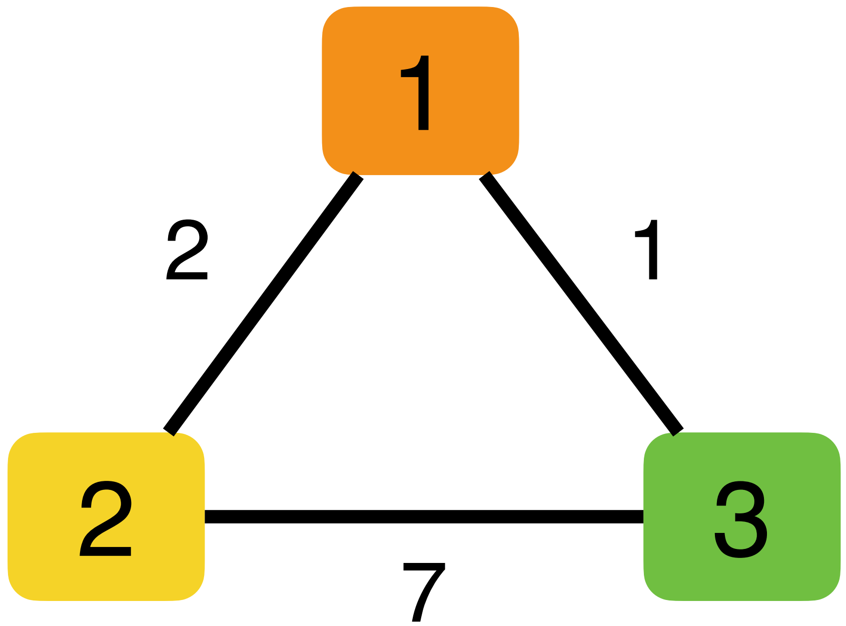
	Receive	Send
1		(1, 0, 1)
2		(2, 0, 2)
3		(3, 0, 3)

Round 2



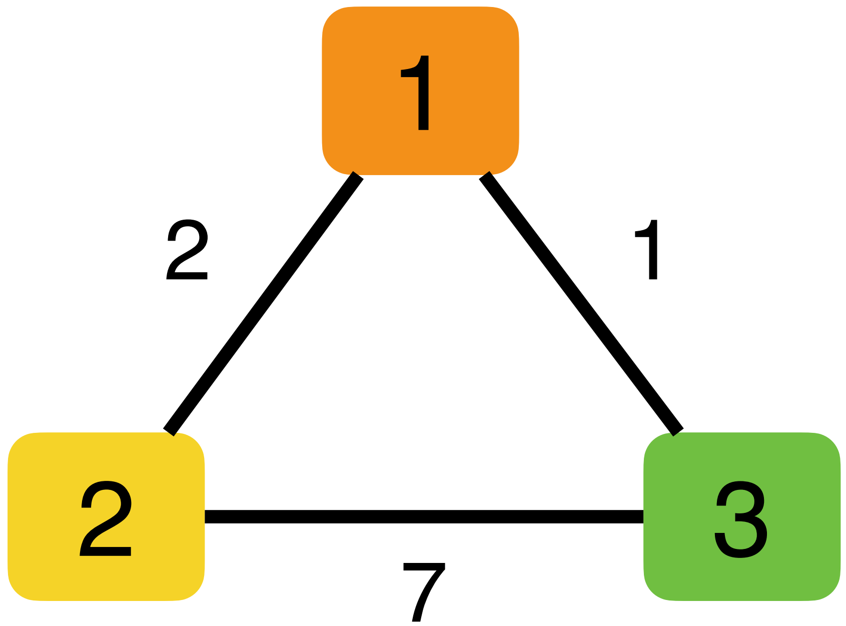
	Receive	Send
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)
2 **(2, 0, 2)**	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)
3 **(3, 0, 3)**	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)

Round 3



	Receive	Send
1 **(1, 0, 1)** **(2, 2, 1),** **(3, 1, 1)**	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)	
2 **(1, 2, 2),** **(2, 0, 2),** **(3, 7, 2)**	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)
3 **(1, 1, 3),** **(2, 7, 3),** **(3, 0, 3)**	(2, 2, 1), (3, 1, 1), (1, 2, 2), (3, 7, 2)	(2, 3, 3)

Round 4



	Receive	Send
1 **(1, 0, 1)** **(2, 2, 1),** **(3, 1, 1)**	(3, 3, 2), (2, 3, 3)	
2 **(1, 2, 2),** **(2, 0, 2),** **(3, 3, 2)**	(2, 3, 3)	
3 **(1, 1, 3),** **(2, 3, 3),** **(3, 0, 3)**	(3, 3, 2)	

Towards Distance-vector protocol with next-hops (no failures)

- **Messages (Y,d,X): For root Y; From node X; advertising a distance d to Y**

- Initially each switch X announces (X,0,X) to its neighbors

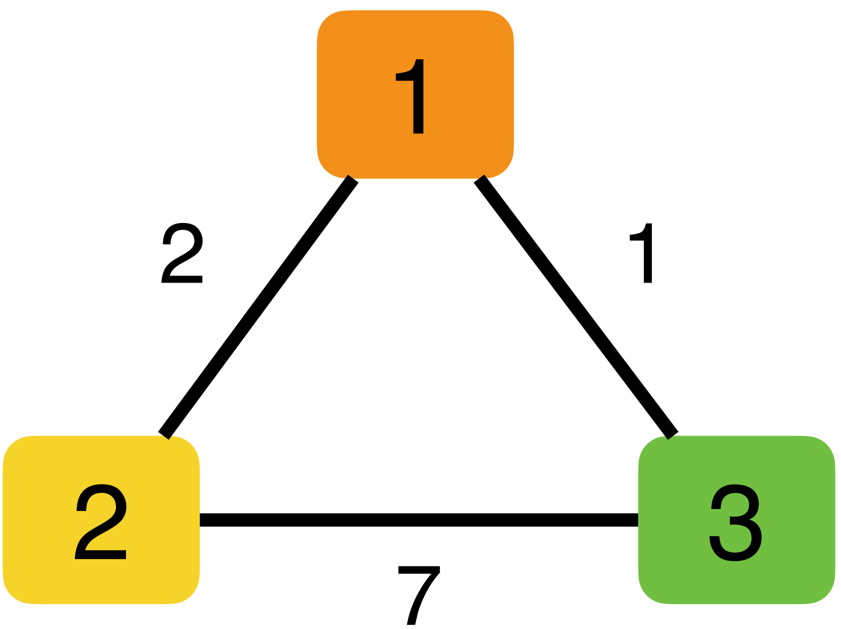
- Switch X updates its view
    - Upon receiving message (Y,d,Z) from Z~~, check Y’s id~~
    - ~~If Y’s id < current root: set root destination = Y~~

- Switch X computes its shortest distance from the ~~root~~ destination
    - If current_distance_to_Y > d + cost of link to Z:
        - update current_distance_to_Y = d
        - **update next_hop_to_destination = Z**

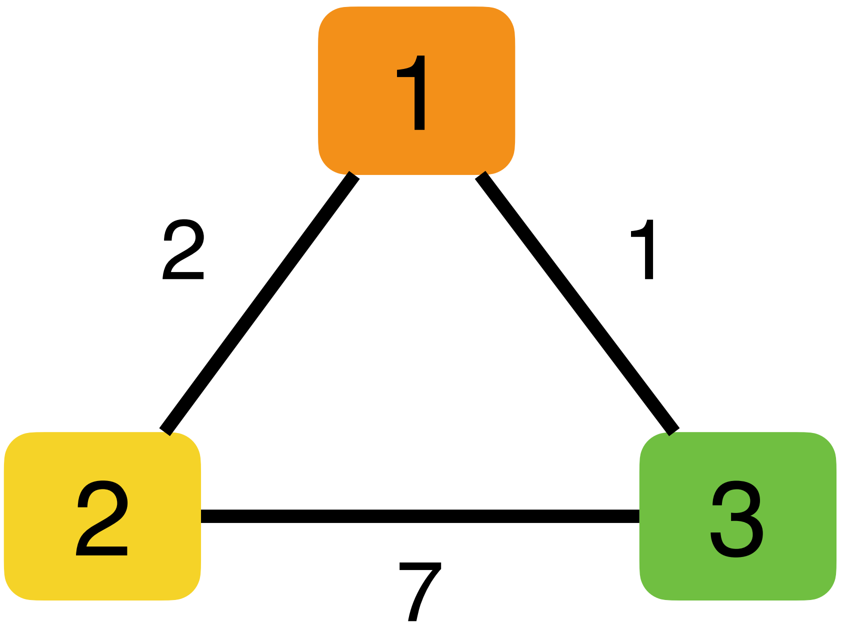
- If ~~root changed OR~~ shortest distance to the ~~root~~ destination changed, send all neighbors updated message (Y, current_distance_to_Y, X)

Lets run the Protocol on this example

(this time with next-hops)

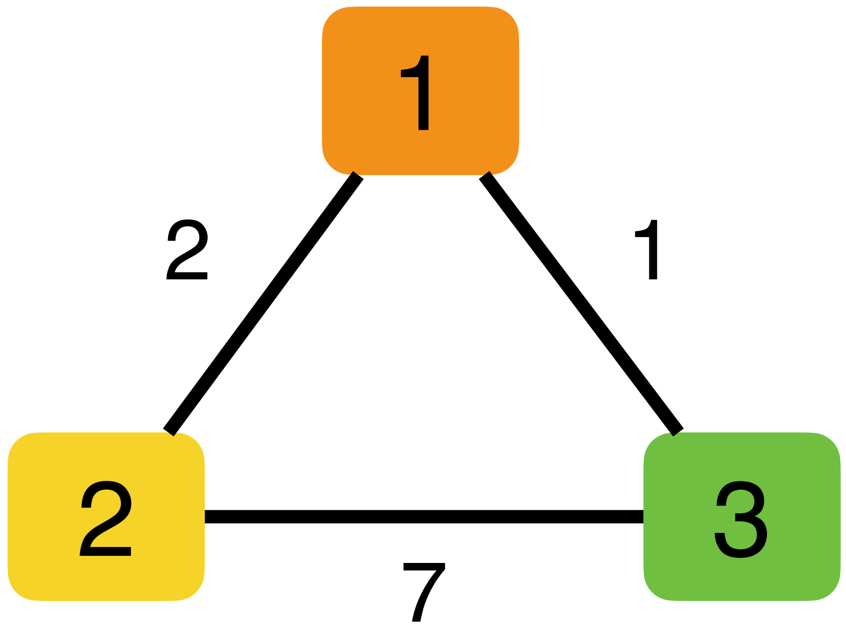


Round 1



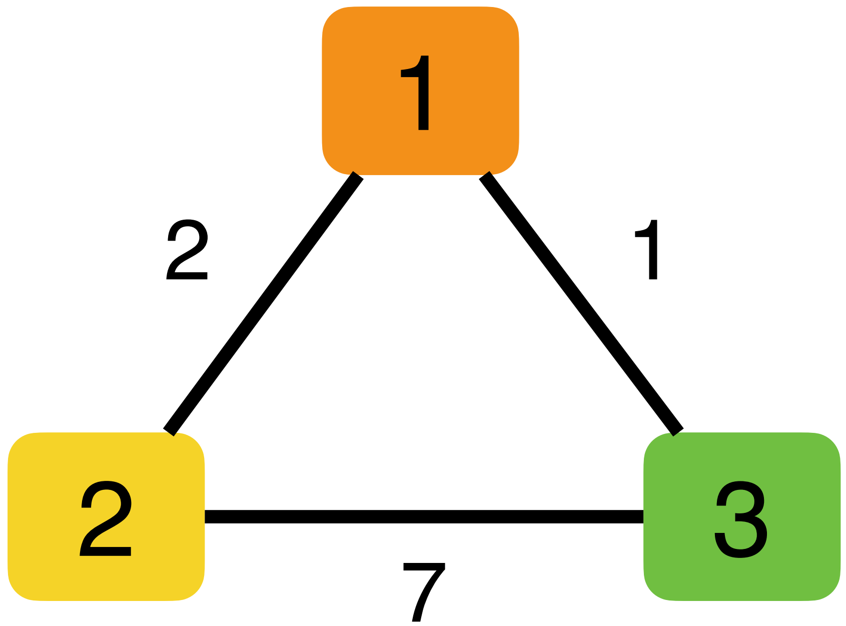
	Receive	Send	Next-hops
1		(1, 0, 1)	[-]
2		(2, 0, 2)	[-]
3		(3, 0, 3)	[-]

Round 2



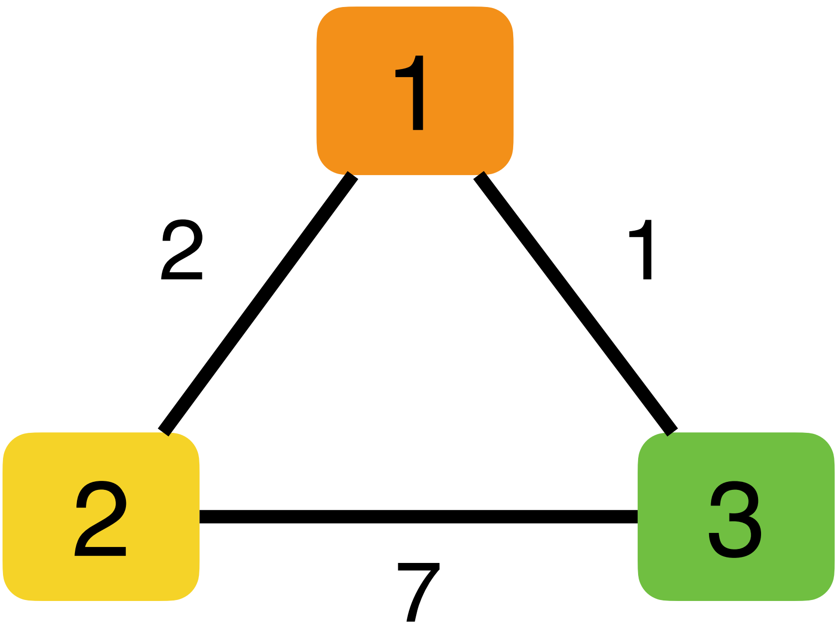
	Receive	Send	Next-hops
1 (1, 0, 1)	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 (2, 0, 2)	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)	[1, -, 3]
3 (3, 0, 3)	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)	[1, 2, -]

Round 3



	Receive	Send	Next-hops
1 **(1, 0, 1)** **(2, 2, 1),** **(3, 1, 1)**	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)		[-, 2, 3]
2 **(1, 2, 2),** **(2, 0, 2),** **(3, 7, 2)**	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)	[1, -, 1]
3 **(1, 1, 3),** **(2, 7, 3),** **(3, 0, 3)**	(2, 2, 1), (3, 1, 1), (1, 2, 2), (3, 7, 2)	(2, 3, 3)	[1, 1, -]

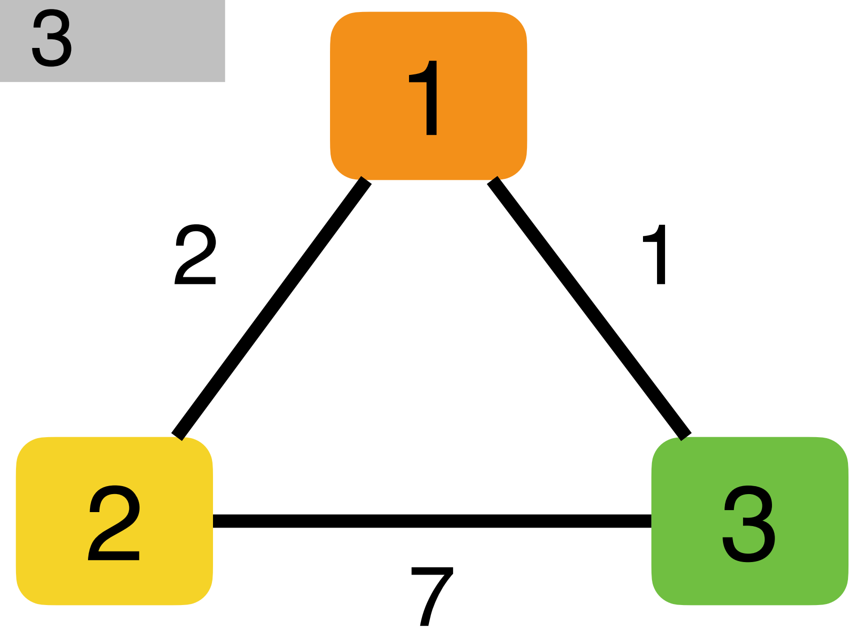
Round 4



	Receive	Send	Next-hops
1 **(1, 0, 1)** **(2, 2, 1),** **(3, 1, 1)**	(3, 3, 2), (2, 3, 3)		[-, 2, 3]
2 **(1, 2, 2),** **(2, 0, 2),** **(3, 3, 2)**	(2, 3, 3)		[1, -, 1]
3 **(1, 1, 3),** **(2, 3, 3),** **(3, 0, 3)**	(3, 3, 2)		[1, 1, -]

Routing tables

	distance	next-hop
1	0	-
2	2	2
3	3	3



	Next-hops
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 (1, 2, 2), (2, 0, 2), (3, 3, 2)	[1, -, 1]
3 (1, 1, 3), (2, 3, 3), (3, 0, 3)	[1, 1, -]

	distance	next-hop
1	2	1
2	0	-
3	3	1

	distance	next-hop
1	1	1
2	3	1
3	0	-

Why not Spanning Tree Protocol? Why Distance "Vector"?

- The same algorithm applies to all destinations
- Each node announces distance to **each** dest
    - I am distance d_A away from node A
    - I am distance d_B away from node B
    - I am distance d_C away from node C
    - …
- Nodes are exchanging a **vector** of distances

Distance Vector Protocol

- **Messages (Y,d,X): For root Y; From node X; advertising a distance d to Y**
- Initially each switch X initializes its routing table to (X,0,-) and distance infinity to all other destinations
- Switches announce their entire distance vectors (routing table w/0 next hops)
- Upon receiving a routing table from a node (say X), each node does:
  - For each destination Y in the announcement (distance(X, Y) = d):
    - If current_distance_to_Y > d + cost of link to X:
      - update current_distance_to_Y = d
      - update next_hop_to_destination = X
- If shortest distance to any destination changed, send all neighbors your distance vectors