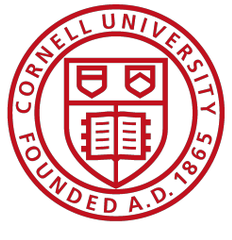
CS4450

Computer Networks:
Architecture and Protocols

Lecture 11

Intra-domain Routing: Deep Dive

Rachit Agarwal



Goals for Today's Lecture

- **Continue learning about Routing Protocols**
    - Link State (Global view, Local computation)—done
    - **Distance Vector (Local view, Local computation)—today**

- Maintain sanity: its one of the "harder" lectures
    - I'll try to make it -less- hard, but …
        - Pay attention
        - Review again tomorrow
        - Work out a few examples

Recap from last few lectures

Recap: 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

Recap: Three flavors of protocols for producing valid routing state

- **Create Tree, route on tree**
    - E.g., Spanning tree protocol (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 (last lecture)
- **Distributed route computation:**
    - E.g., Distance vector
    - E.g., Border Gateway Protocol

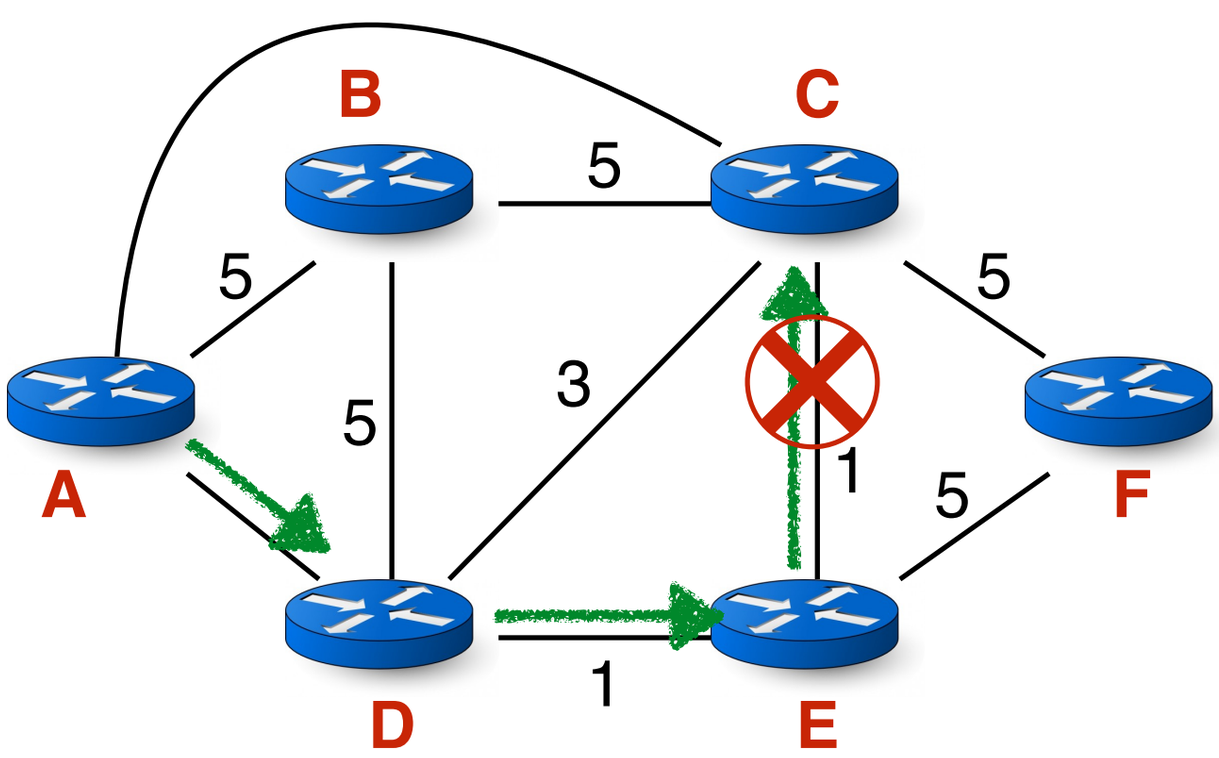
Recap: Where to create global view?

- 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

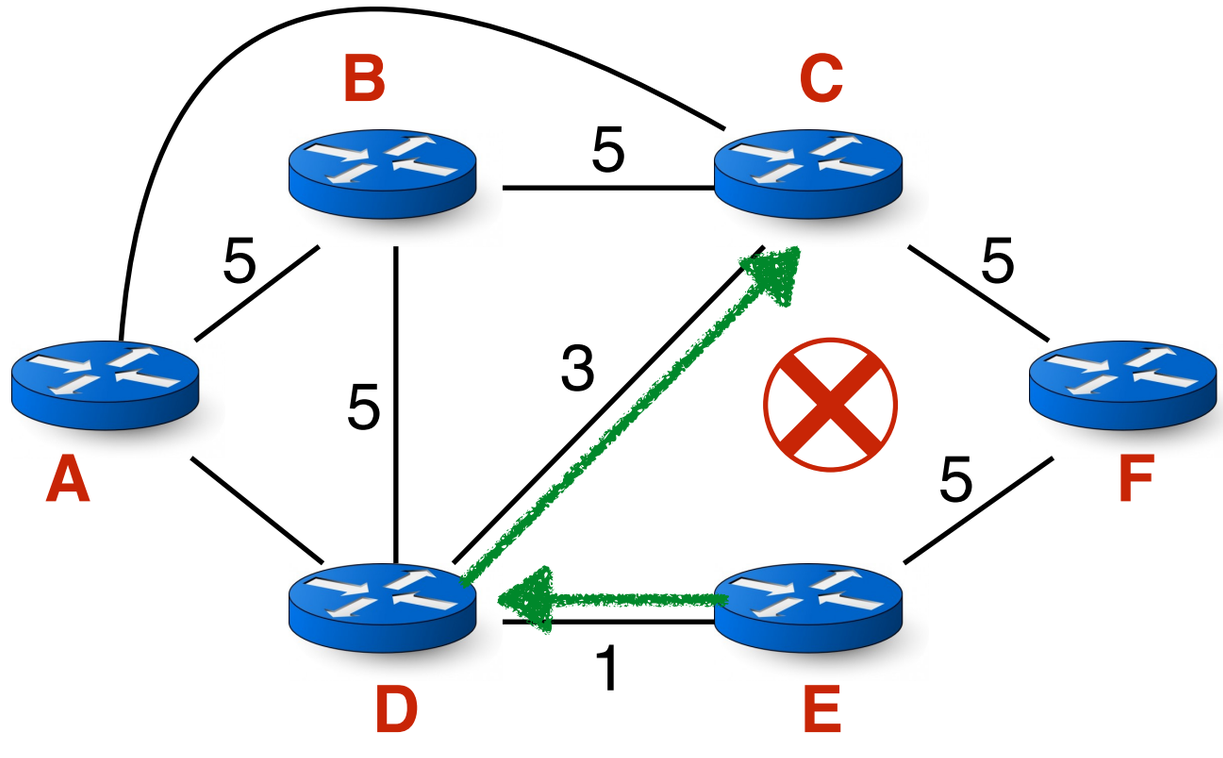
- **Link-state routing protocol**
    - OSPF is a specific implementation of link-state protocol
        - IETF RFC 2328 (IPv4) or 5340 (IPv6)

Recap: Are Loops Still Possible?



A and D think this is the path to C

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

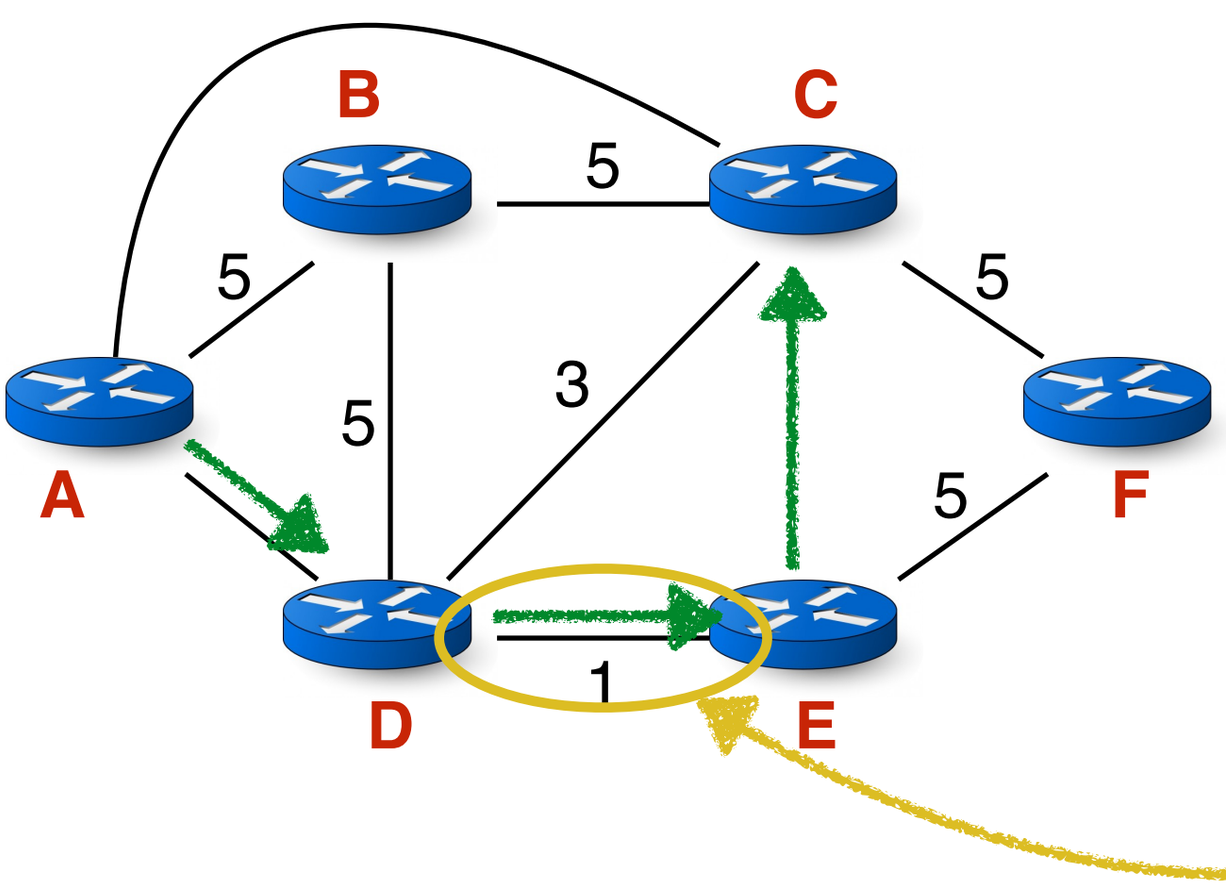


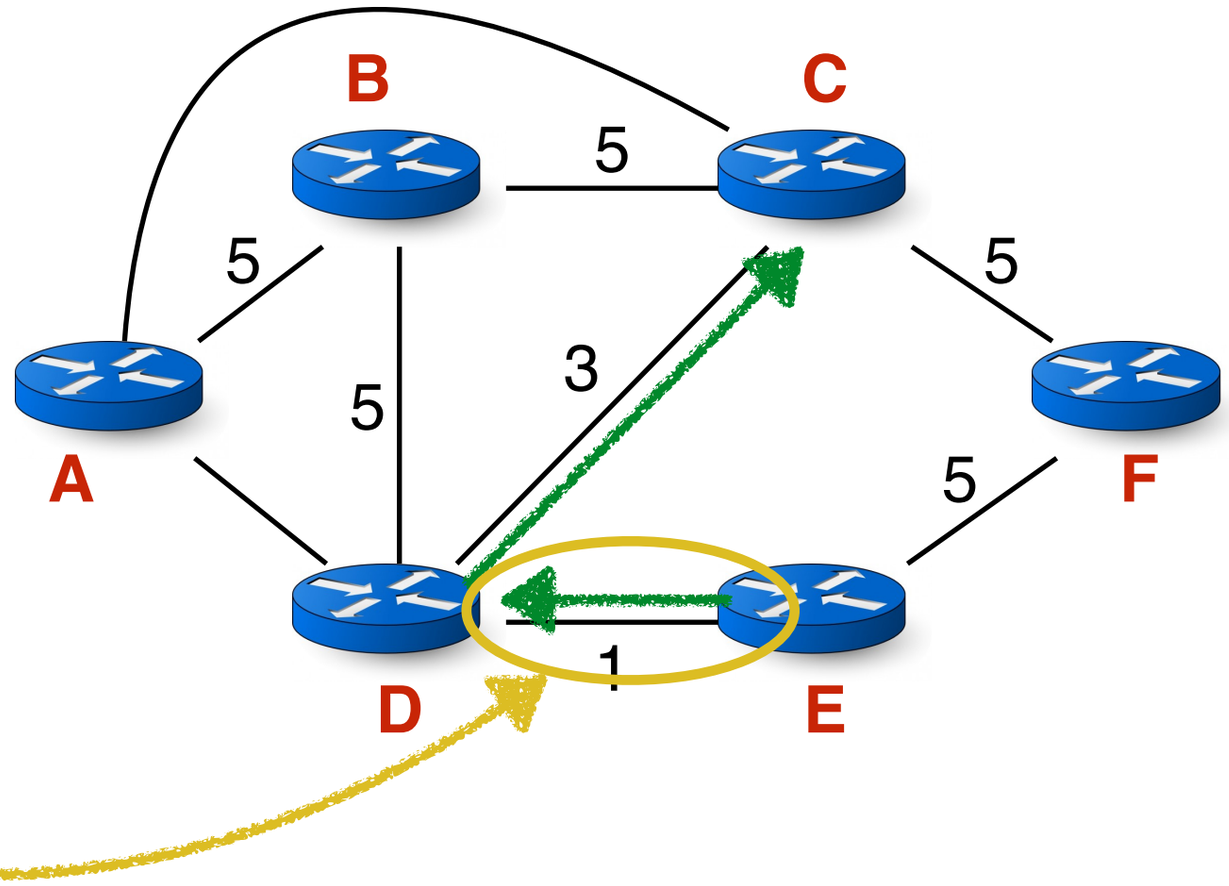
E thinks that this the path to C

E reaches C via D, D reaches C via E

Loop!

Recap: 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!**

Questions?

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

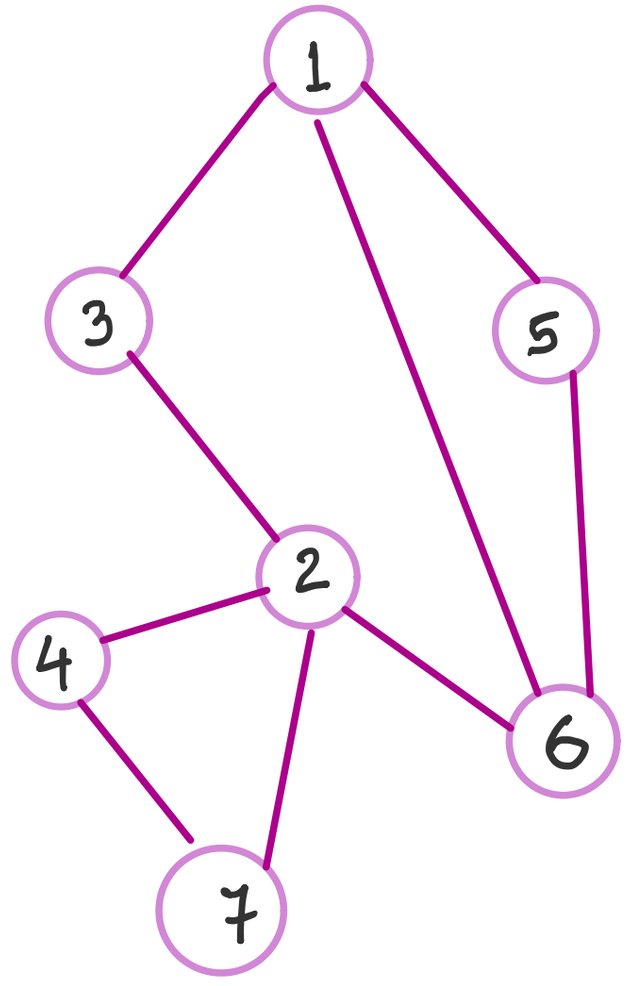
Towards Distance Vector Protocol (with no failures)

- **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
- Each switch X updates its view upon receiving each message
  - 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)

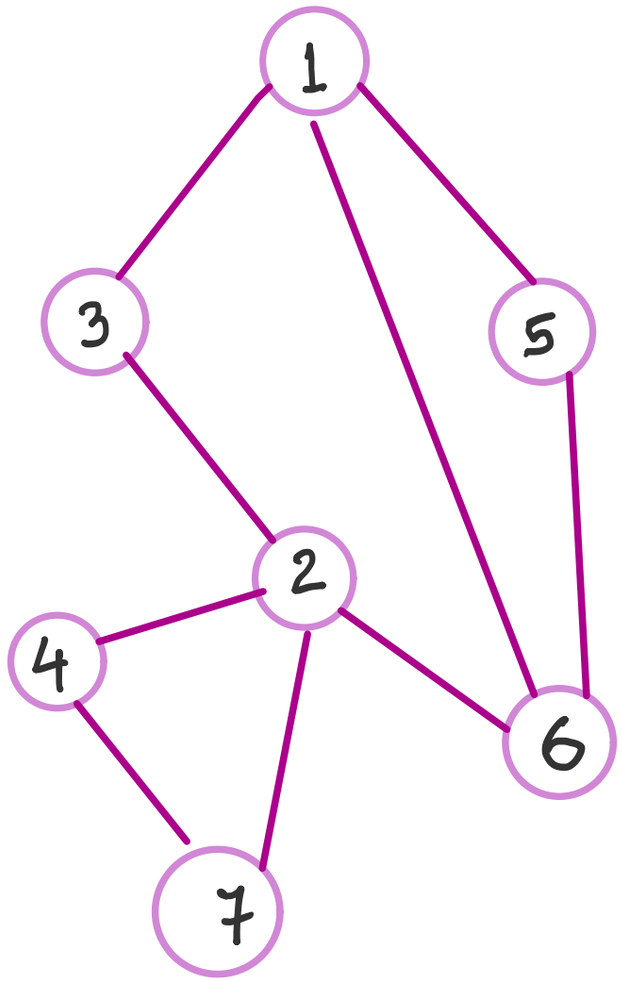
Distance vector: a collection of “n” STP in parallel

Lets run the Protocol on this example

(destination = 1)

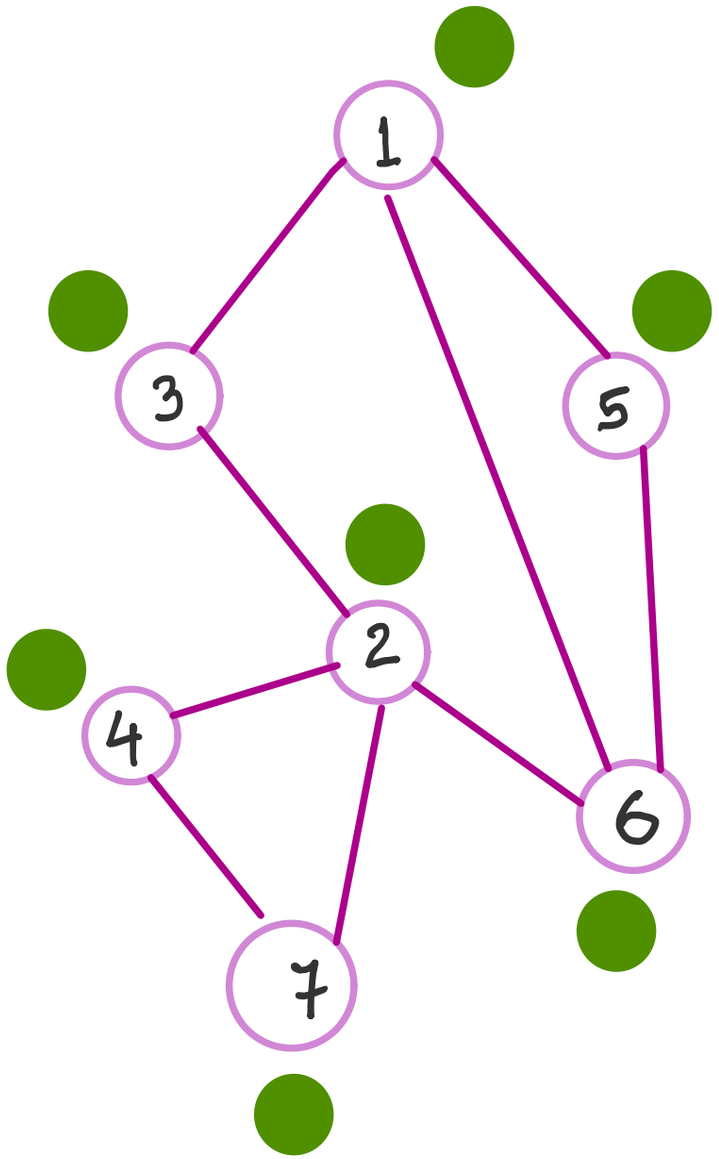


Round 1



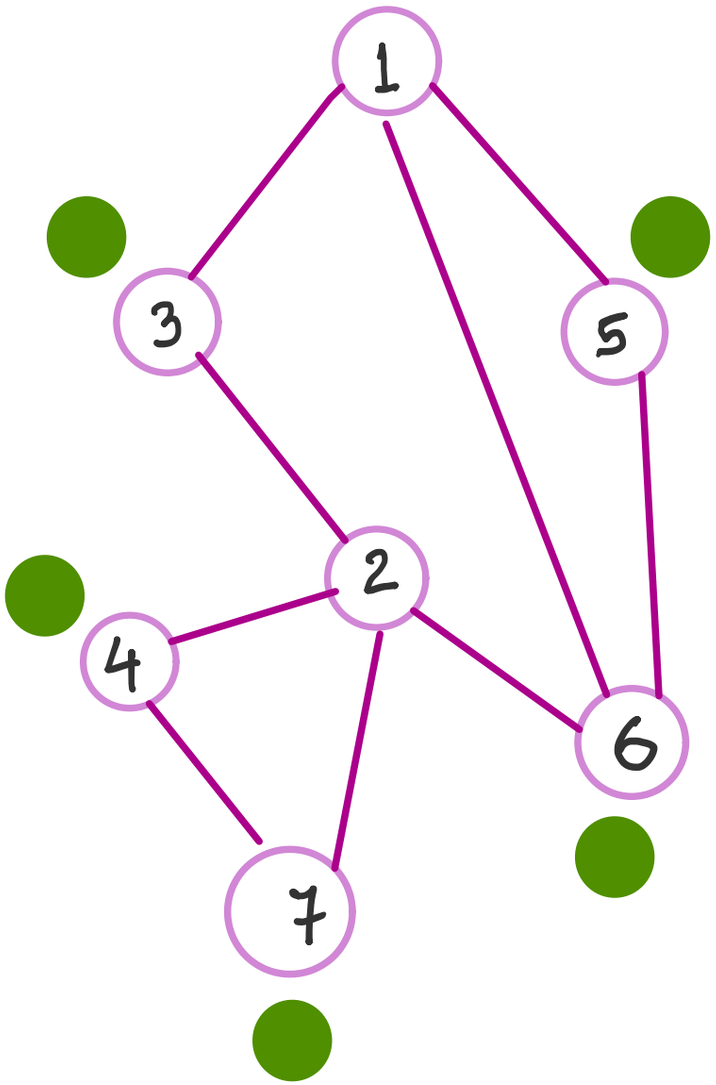
	Receive	Send
1		(1, 0, 1)
2		
3		
4		
5		
6		
7		

Round 2



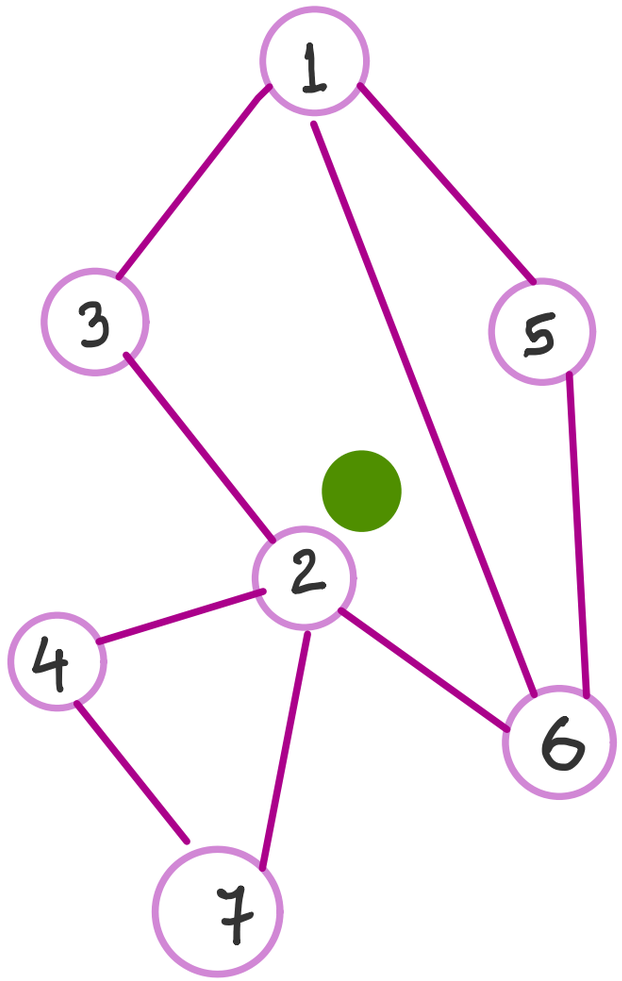
	Receive	Send
1 (1, 0, 1)		
2		
3	(1, 0, 1)	**(1, 1, 3)**
4		
5	(1, 0, 1)	**(1, 1, 5)**
6	(1, 0, 1)	**(1, 1, 6)**
7		

Round 3



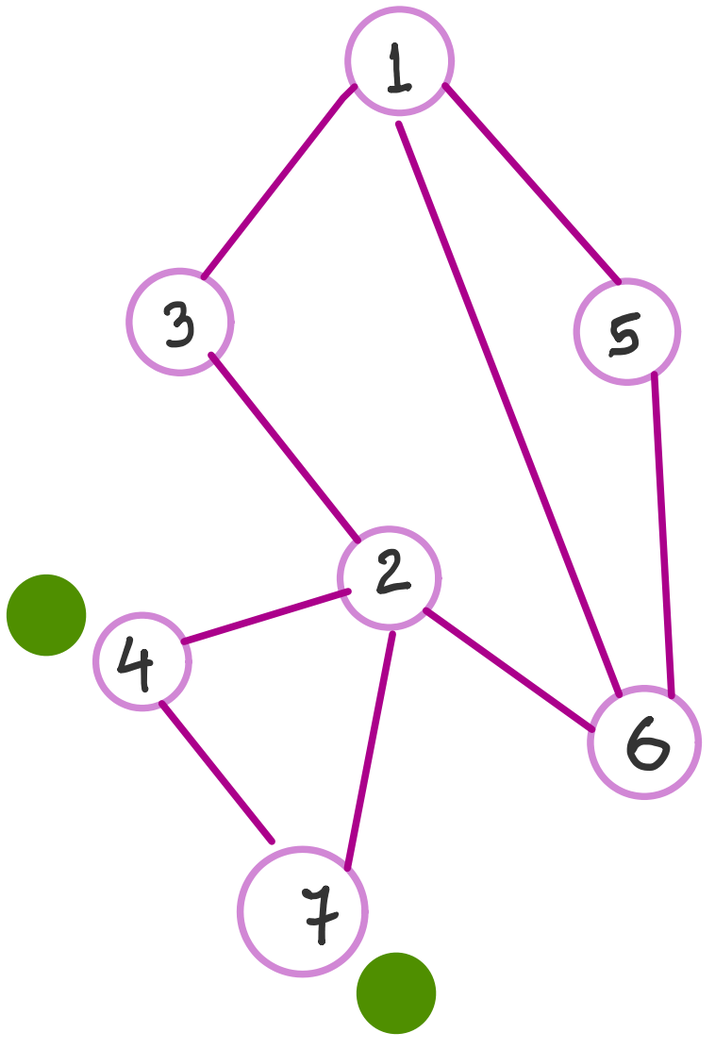
	Receive	Send
1 (1, 0, 1)	(1, 1, 3), (1, 1, 5), (1, 1, 6)	
2	(1, 1, 3), (1, 1, 6)	**(1, 2, 2)**
3 (1, 1, 3)		
4		
5 (1, 1, 5)	(1, 1, 6)	
6 (1, 1, 6)	(1, 1, 5)	
7		

Round 4



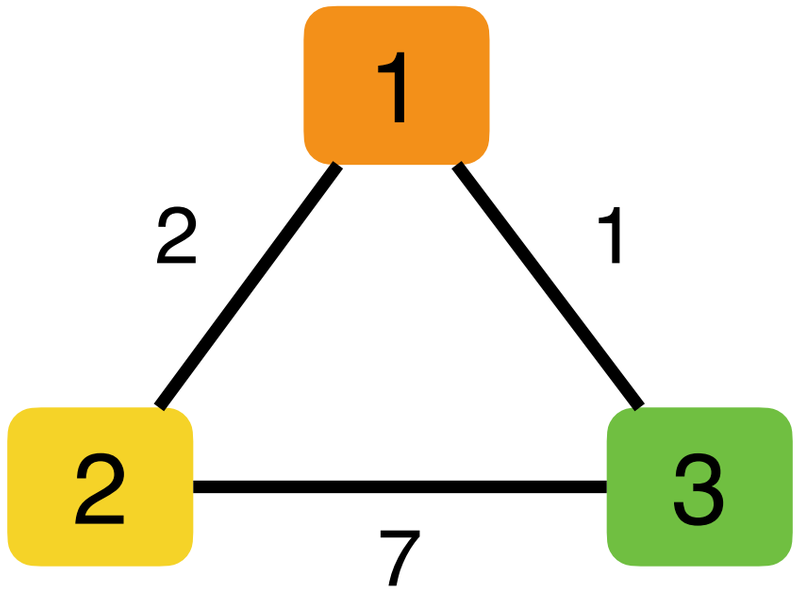
	Receive	Send
1 (1, 0, 1)		
2 (1, 2, 2)		
3 (1, 1, 3)	(1, 2, 2)	
4	(1, 2, 2)	**(1, 3, 4)**
5 (1, 1, 5)		
6 (1, 1, 6)	(1, 2, 2)	
7	(1, 2, 2)	**(1, 3, 7)**

Round 5

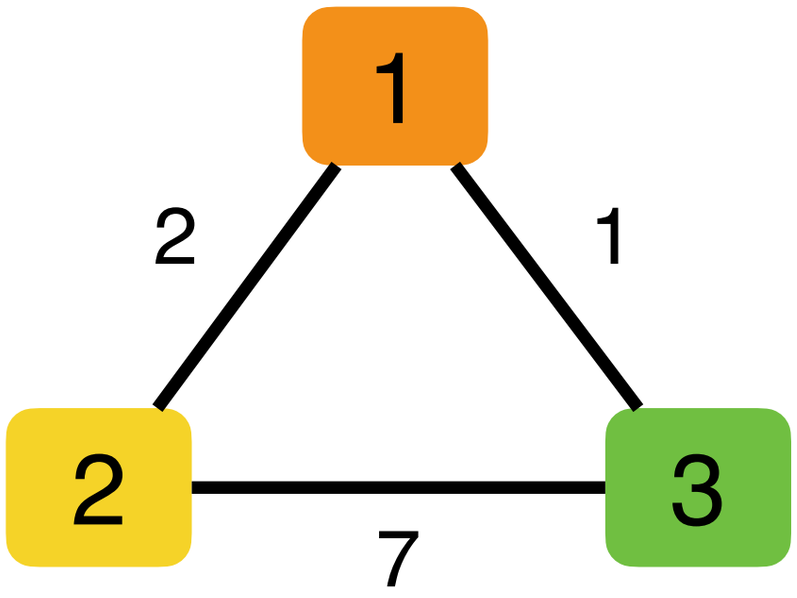


	Receive	Send
1 (1, 0, 1)		
2 (1, 2, 2)	(1, 3, 4), (1, 3, 7)	
3 (1, 1, 3)		
4 (1, 3, 4)	(1, 3, 7)	
5 (1, 1, 5)		
6 (1, 1, 6)		
7 (1, 3, 7)	(1, 3, 4)	

Lets run the Protocol on this example

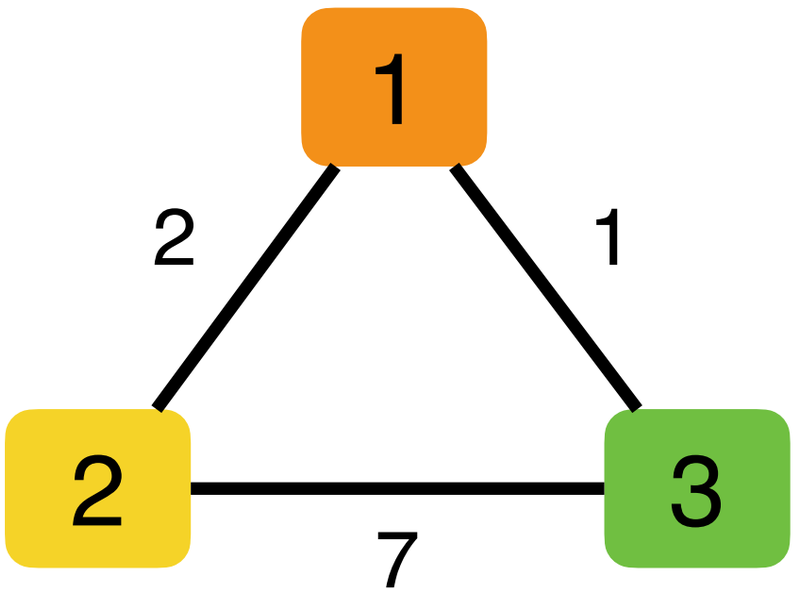


Round 1



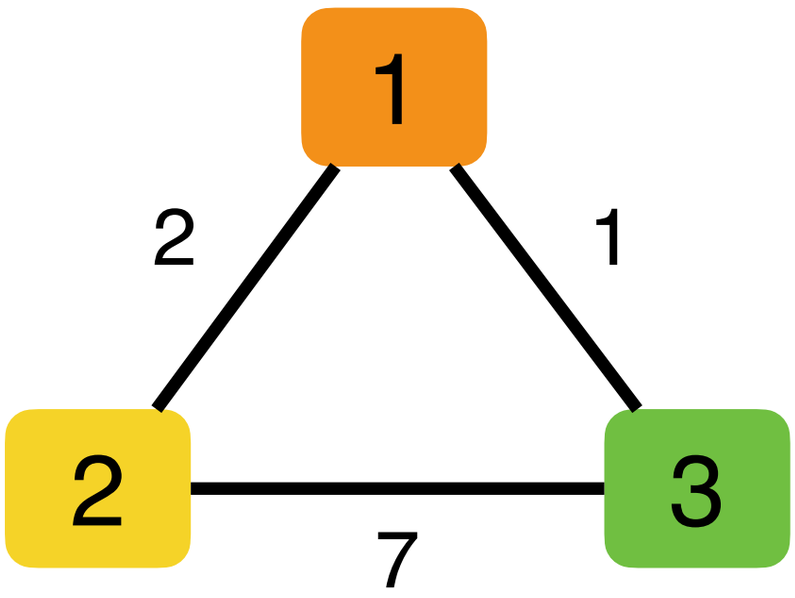
	Receive	Send
1		(1, 0, 1)
2		(2, 0, 2)
3		(3, 0, 3)

Round 2



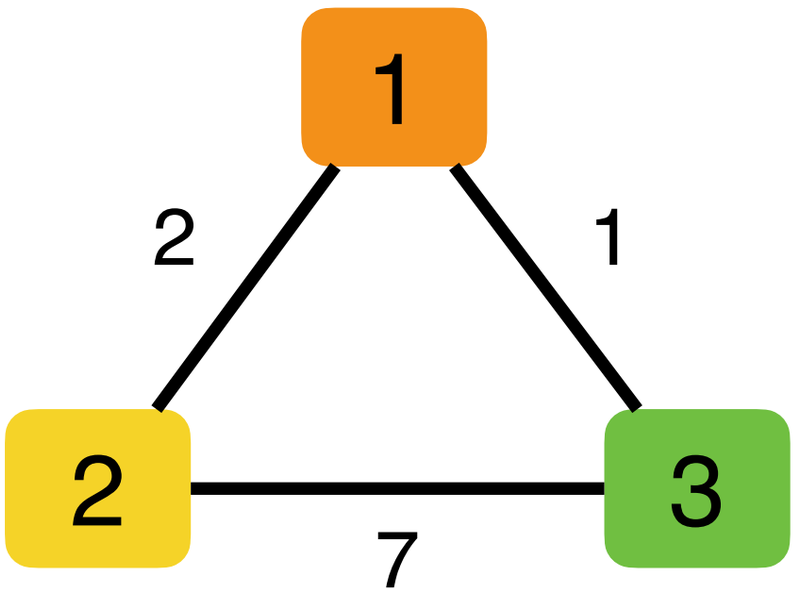
	Receive	Send
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)

Round 2



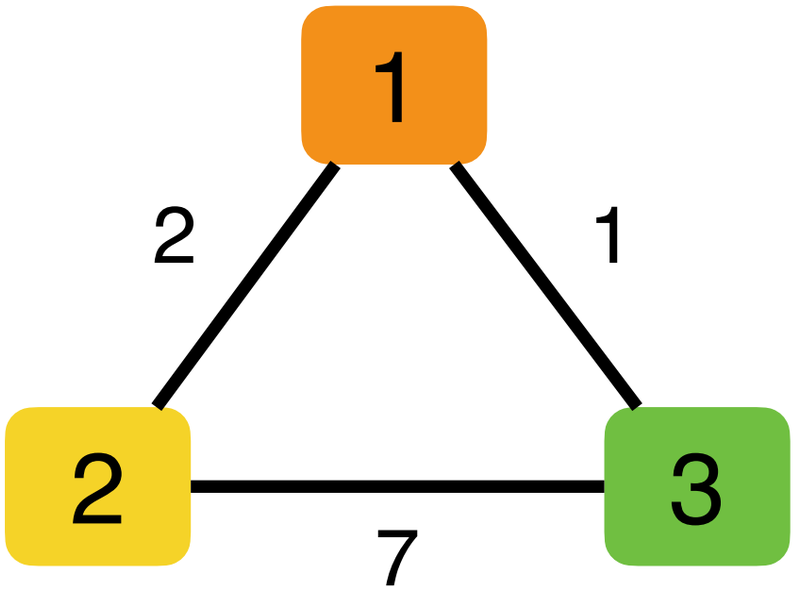
	Receive	Send
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)
2 **(2, 0, 2)**	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)

Round 2



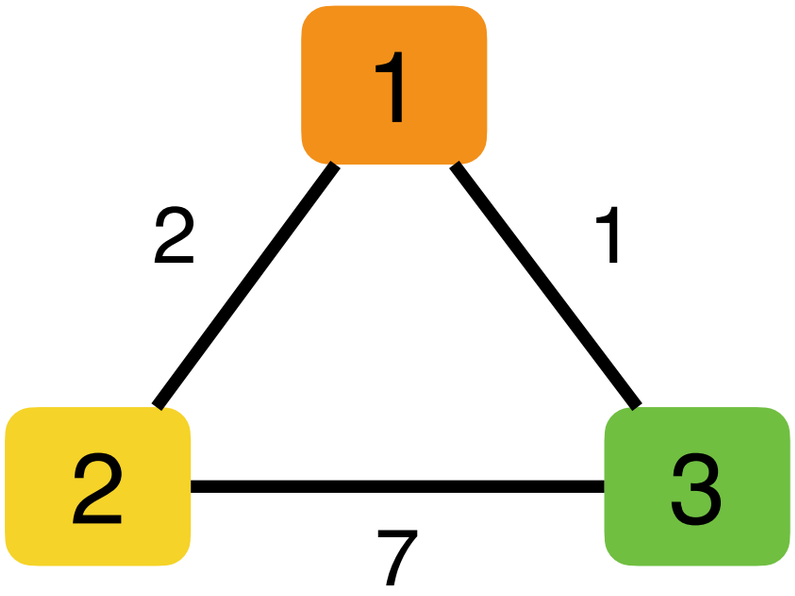
	Receive	Send
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)
2 **(2, 0, 2)**	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)
3 **(3, 0, 3)**	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)

Round 3



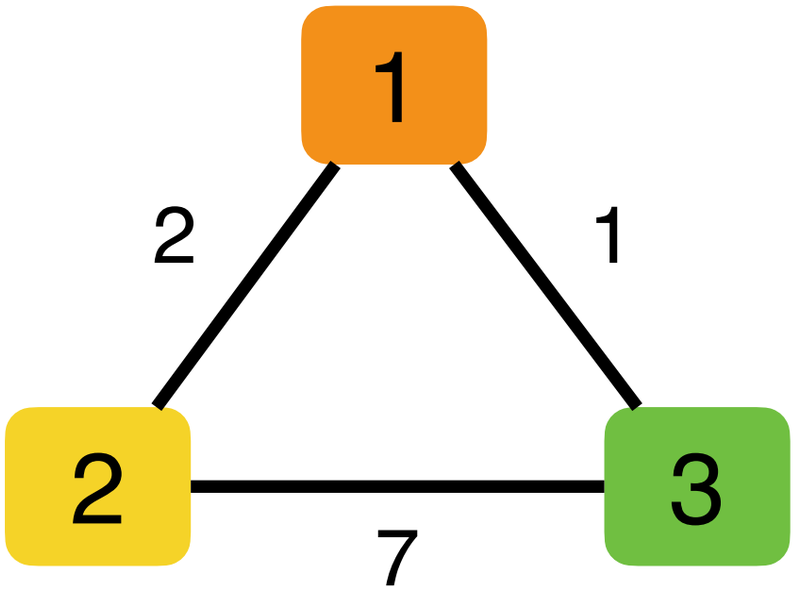
	Receive	Send
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)	

Round 3



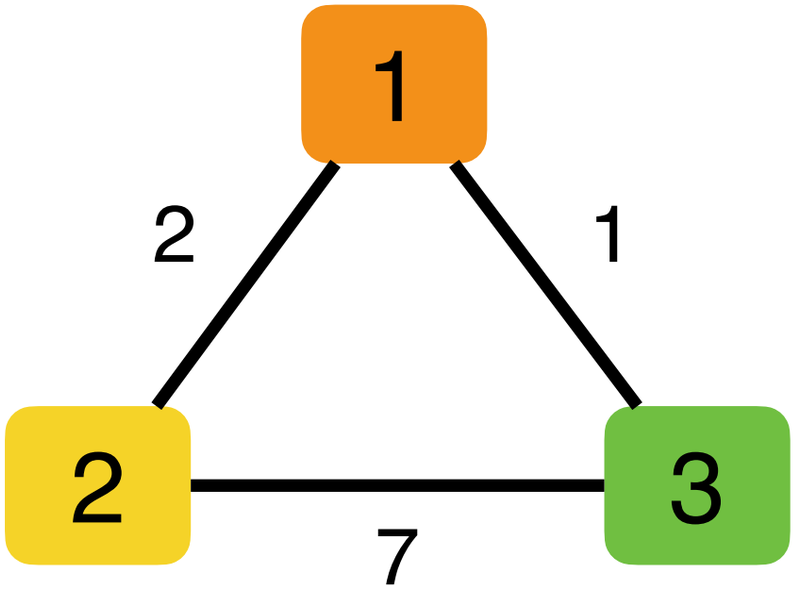
	Receive	Send
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)	
2 (1, 2, 2), (2, 0, 2), (3, 7, 2)	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)

Round 3



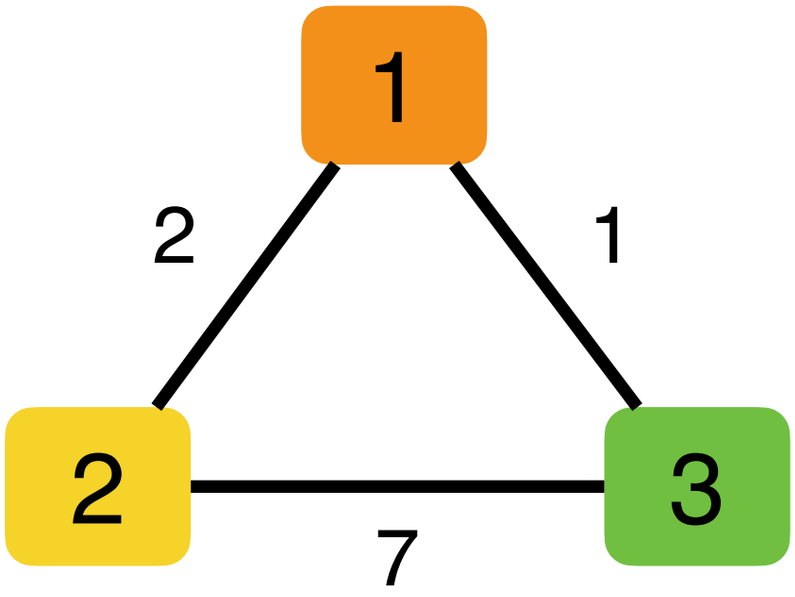
	Receive	Send
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)	
2 (1, 2, 2), (2, 0, 2), (3, 7, 2)	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)
3 (1, 1, 3), (2, 7, 3), (3, 0, 3)	(2, 2, 1), (3, 1, 1), (1, 2, 2), (3, 7, 2)	(2, 3, 3)

Round 4



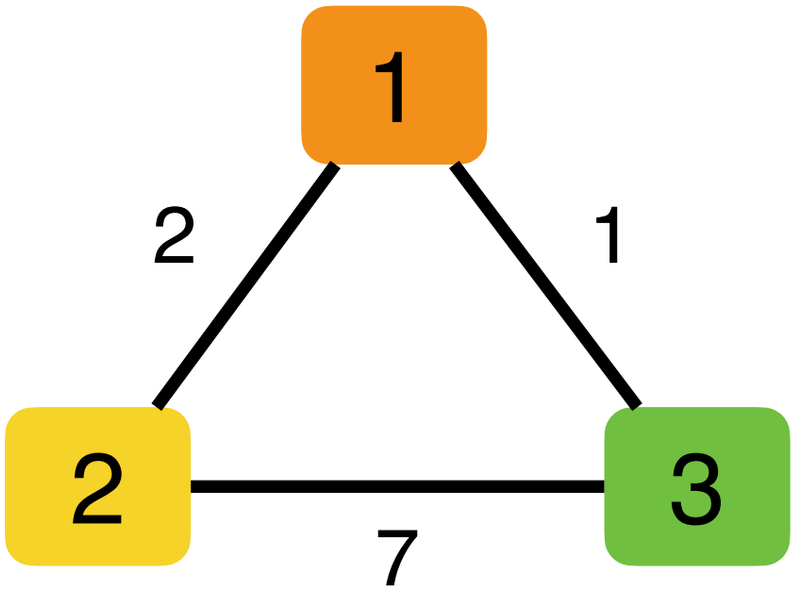
	Receive	Send
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(3, 3, 2), (2, 3, 3)	

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)	

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)	

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

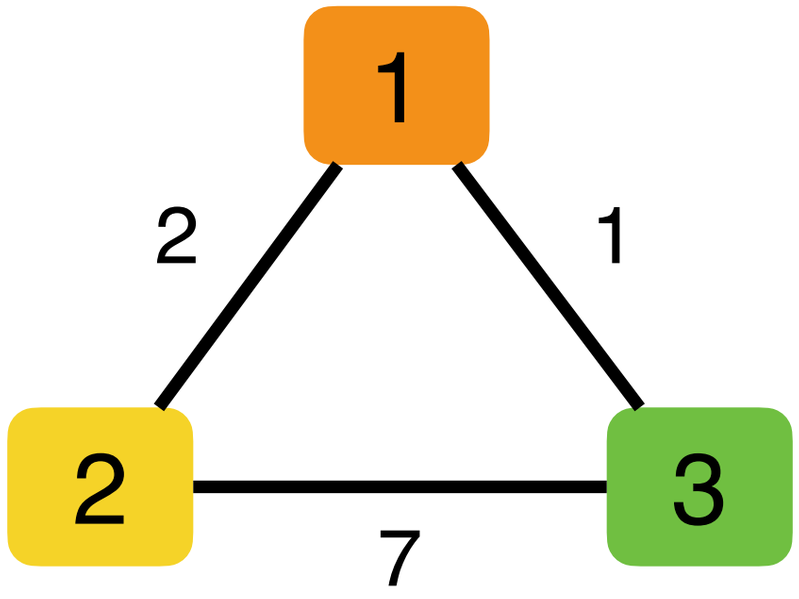
- **Messages (Y,d,X): For root Y; From node X; advertising a distance d to Y**
- Initially each switch X announces (X,0,X) to its neighbors
- Each switch X updates its view upon receiving each message
  - Upon receiving message (Y,d,Z) from Z~~, check Y's id~~
  - ~~If Y's id < current root: set root 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
      - **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)

Why not Spanning Tree Protocol? Why Distance “Vector”?

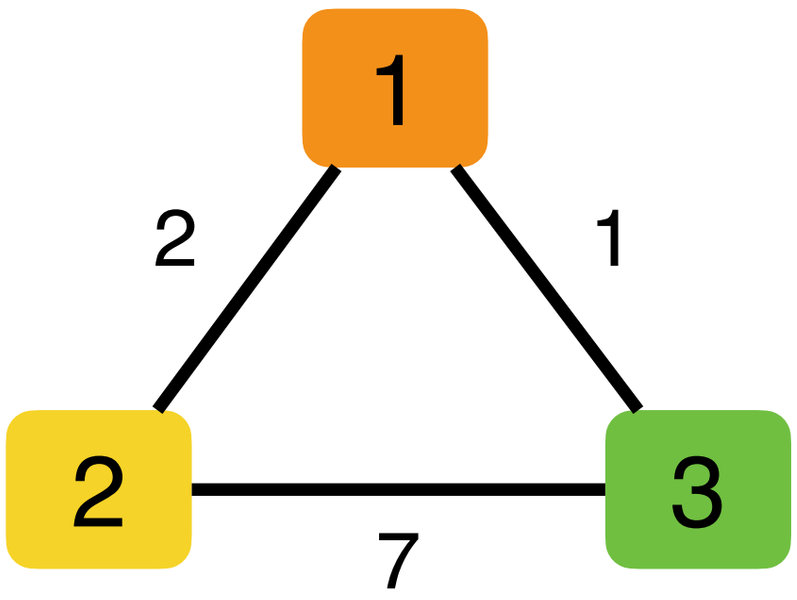
- The same algorithm applies to all destinations
- Each node announces distance to **each** dest
    - I am distance d_A away from node A
    - I am distance d_B away from node B
    - I am distance d_C away from node C
    - …
- Nodes are exchanging a **vector** of distances

Lets run the Protocol on this example

(with next-hops)

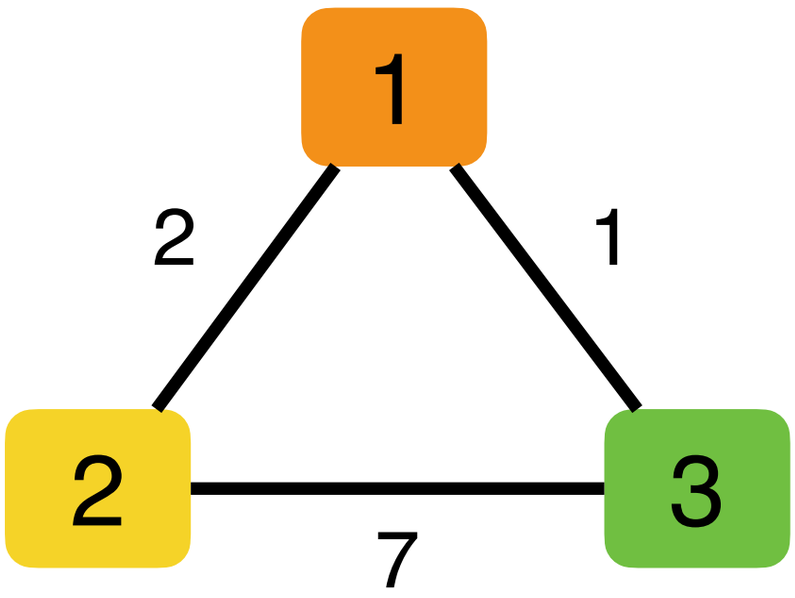


Round 1



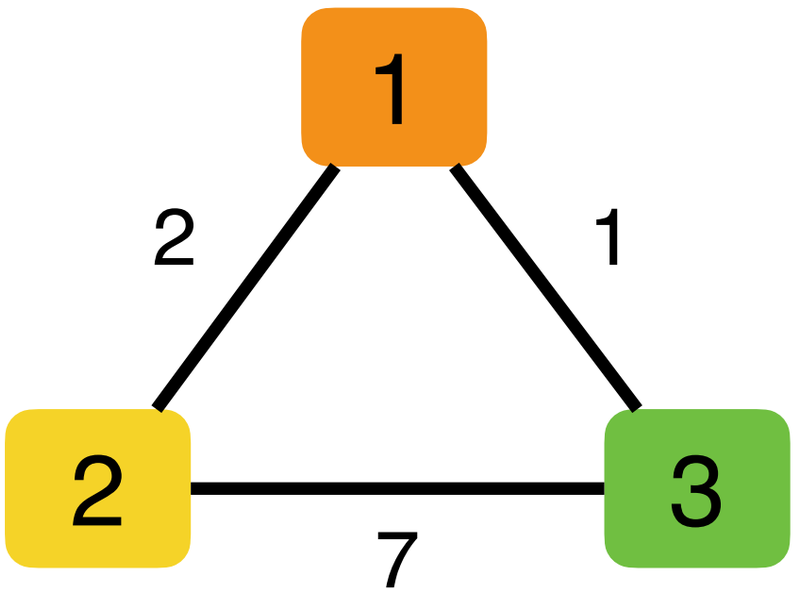
	Receive	Send	Next-hops
1		(1, 0, 1)	[-]
2		(2, 0, 2)	[-]
3		(3, 0, 3)	[-]

Round 2



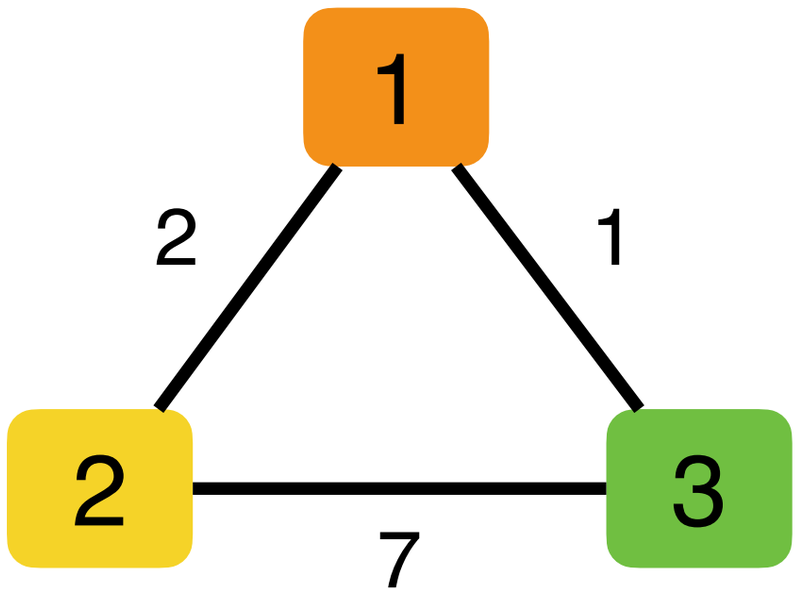
	Receive	Send	Next-hops
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]

Round 2



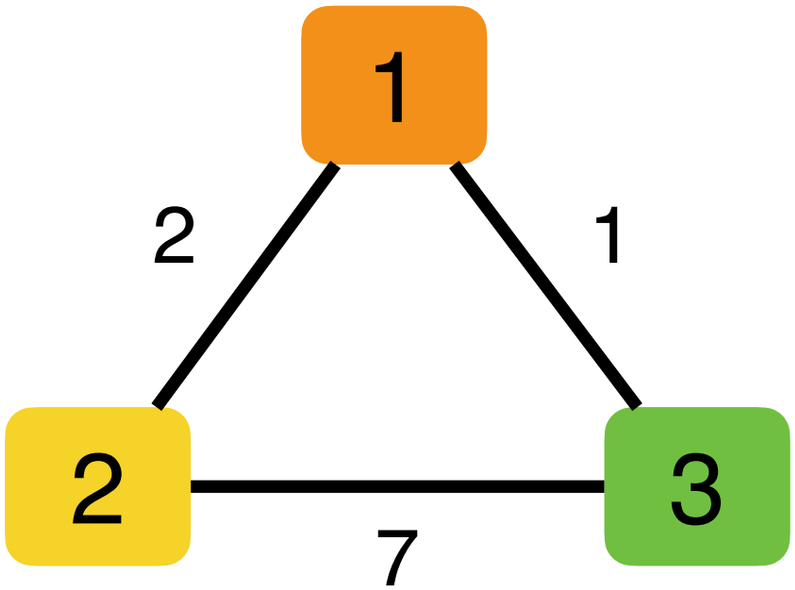
	Receive	Send	Next-hops
1 (1, 0, 1)	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 (2, 0, 2)	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)	[1, -, 3]

Round 2



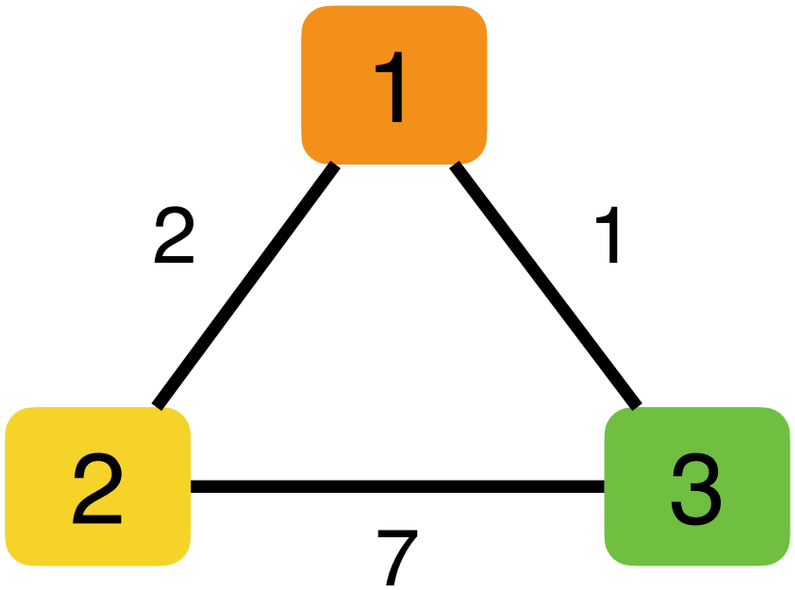
	Receive	Send	Next-hops
1 (1, 0, 1)	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 (2, 0, 2)	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)	[1, -, 3]
3 (3, 0, 3)	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)	[1, 2, -]

Round 3



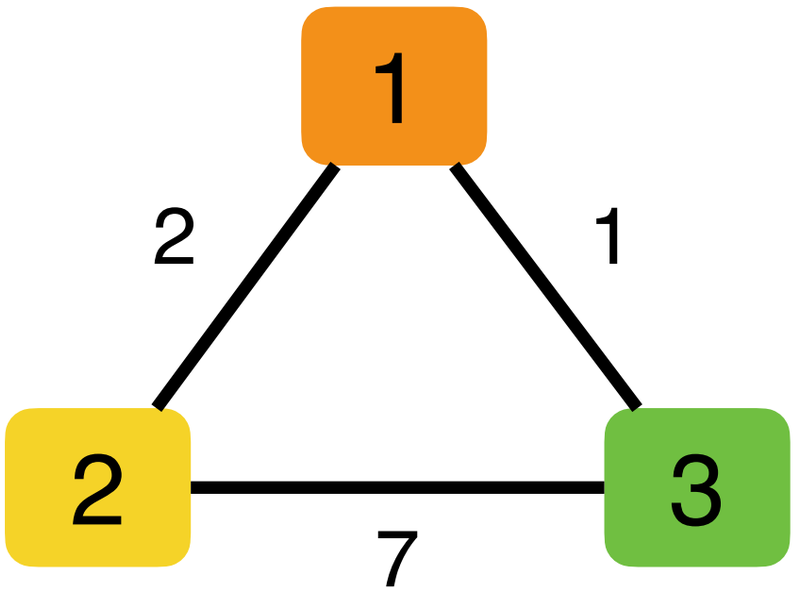
	Receive	Send	Next-hops
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)		[-, 2, 3]

Round 3



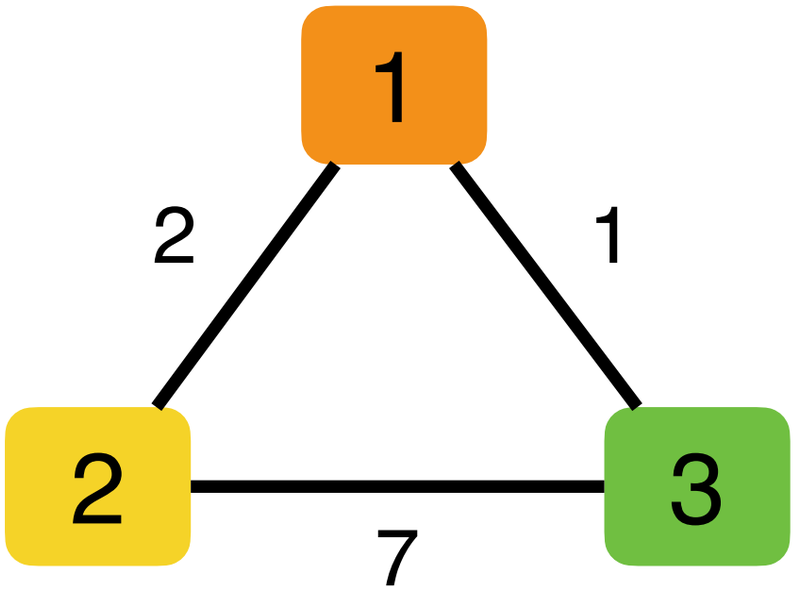
	Receive	Send	Next-hops
1 **(1, 0, 1)** **(2, 2, 1),** **(3, 1, 1)**	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)		[-, 2, 3]
2 **(1, 2, 2),** **(2, 0, 2),** **(3, 7, 2)**	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)	[1, -, **1**]

Round 3



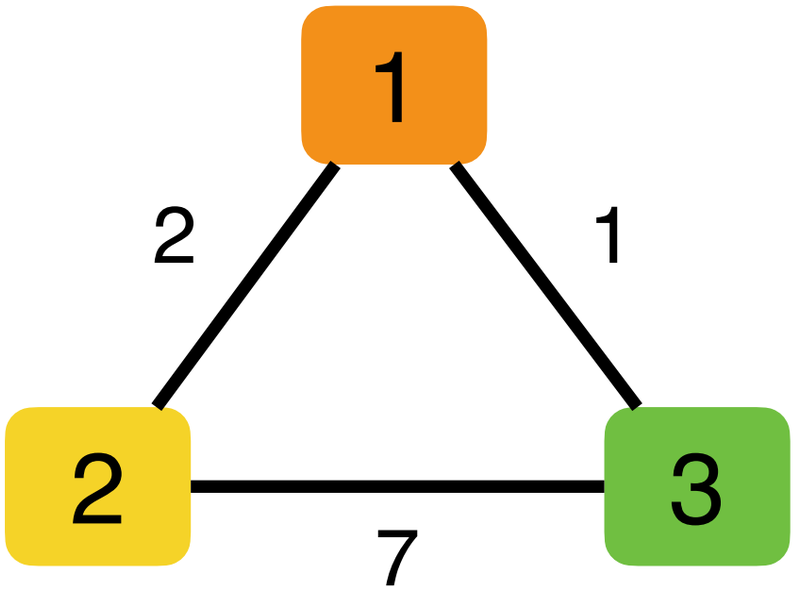
	Receive	Send	Next-hops
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)		[-, 2, 3]
2 (1, 2, 2), (2, 0, 2), (3, 7, 2)	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)	[1, -, **1**]
3 (1, 1, 3), (2, 7, 3), (3, 0, 3)	(2, 2, 1), (3, 1, 1), (1, 2, 2), (3, 7, 2)	(2, 3, 3)	[1,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

Round 4



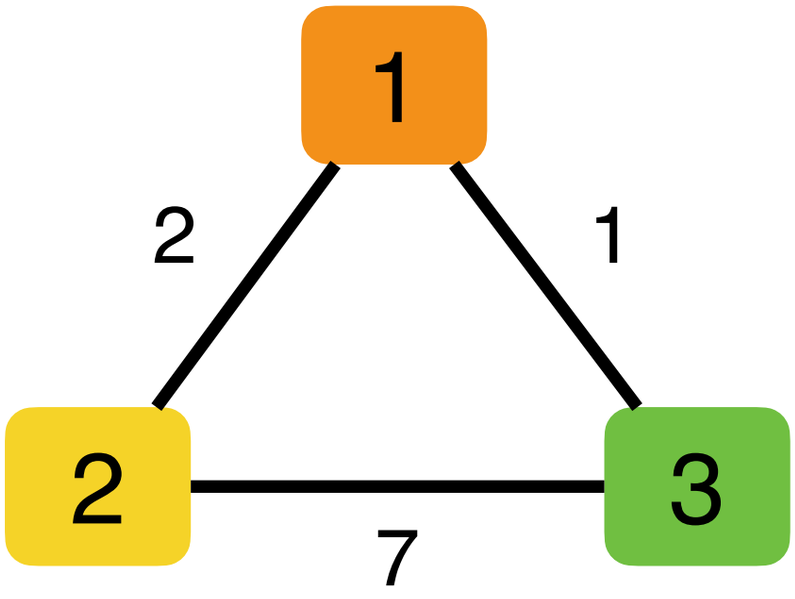
	Receive	Send	Next-hops
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(3, 3, 2), (2, 3, 3)		[-, 2, 3]

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]

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, -]

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 Z), each node X does:
    - For each destination Y in the announcement (distance(Y, Z) = d):
        - If current_distance_to_Y > d + cost of link to Z:
            - update current_distance_to_Y = d + cost of link to Z
            - update next_hop_to_destination = Z
- If shortest distance to any destination changed, send all neighbors your distance vectors

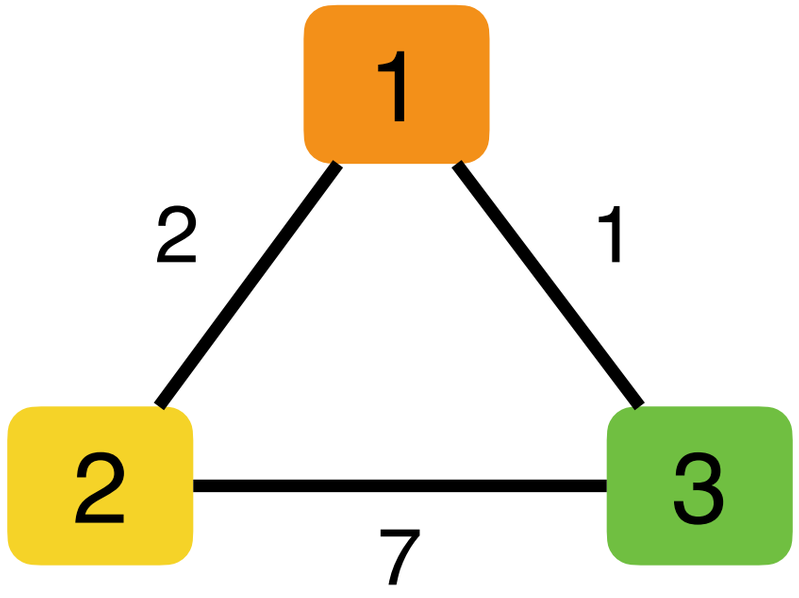
Two Aspects to This Approach

- **Protocol**:
    - Exchanging that routing information with neighbors
    - What and when for exchanges
    - RIP is a protocol that implements DV (IETF RFC 2080)

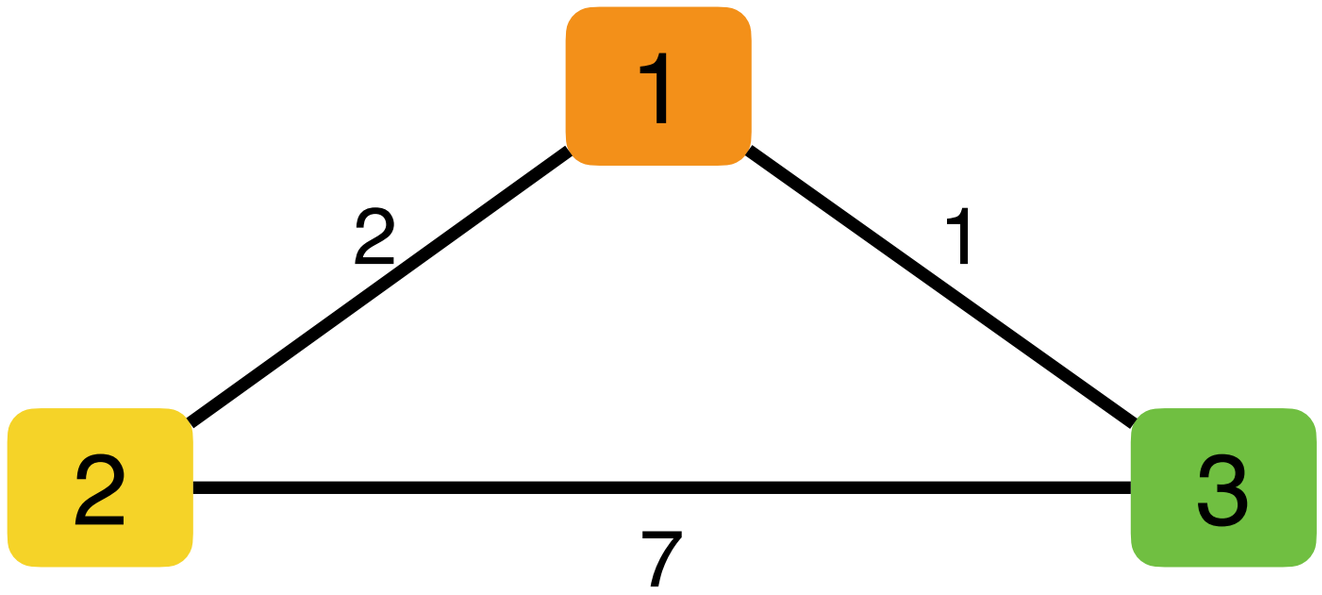
- **Algorithm**:
    - How to use the information from your neighbors to update your own routing tables?

Lets run the Protocol again on this example

(this time with distance vectors)

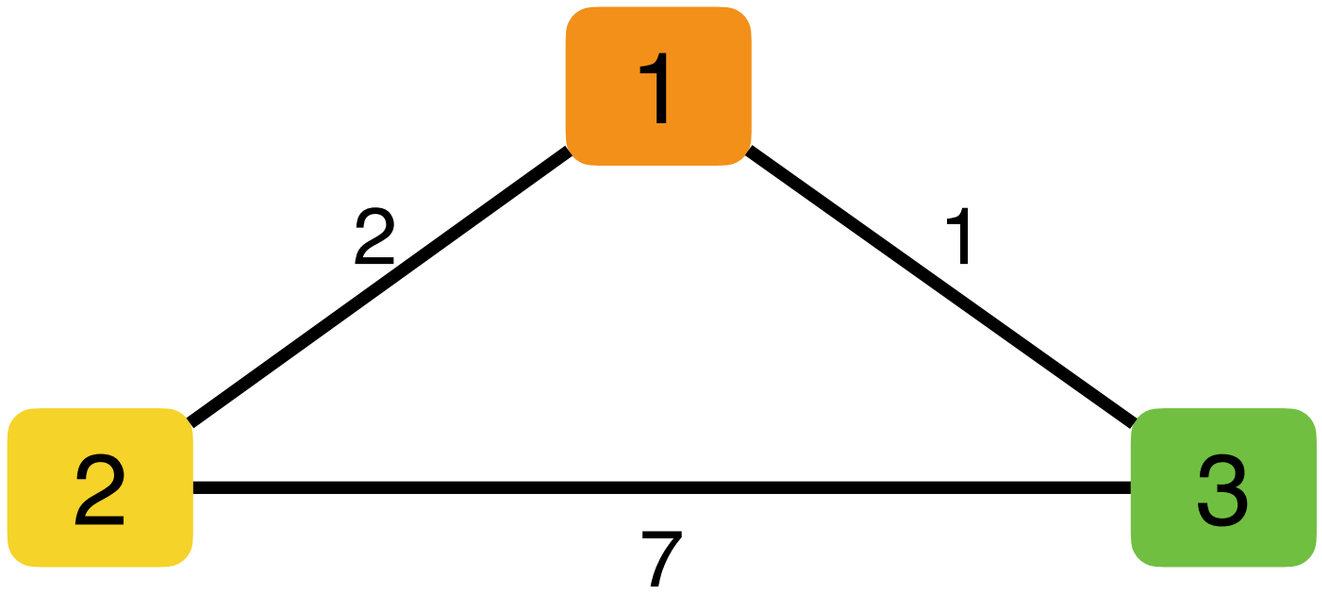


Round 1



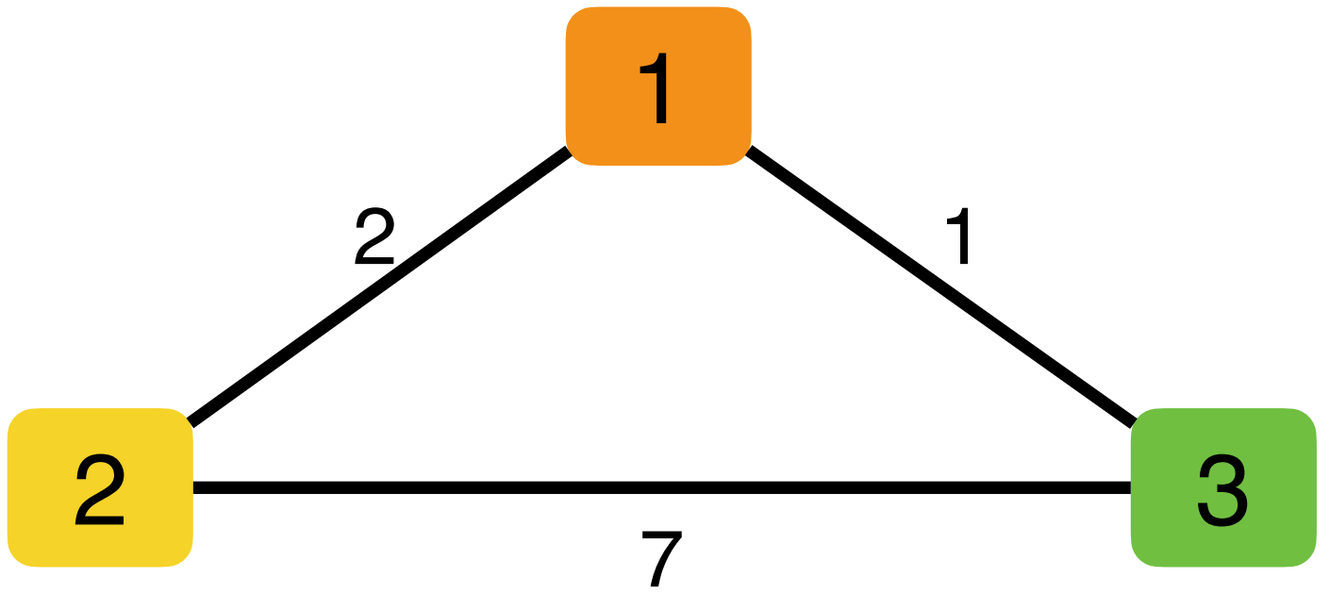
Round 1

	distance	next-hop
1	0	-
2	infinity	
3	infinity	



Round 1

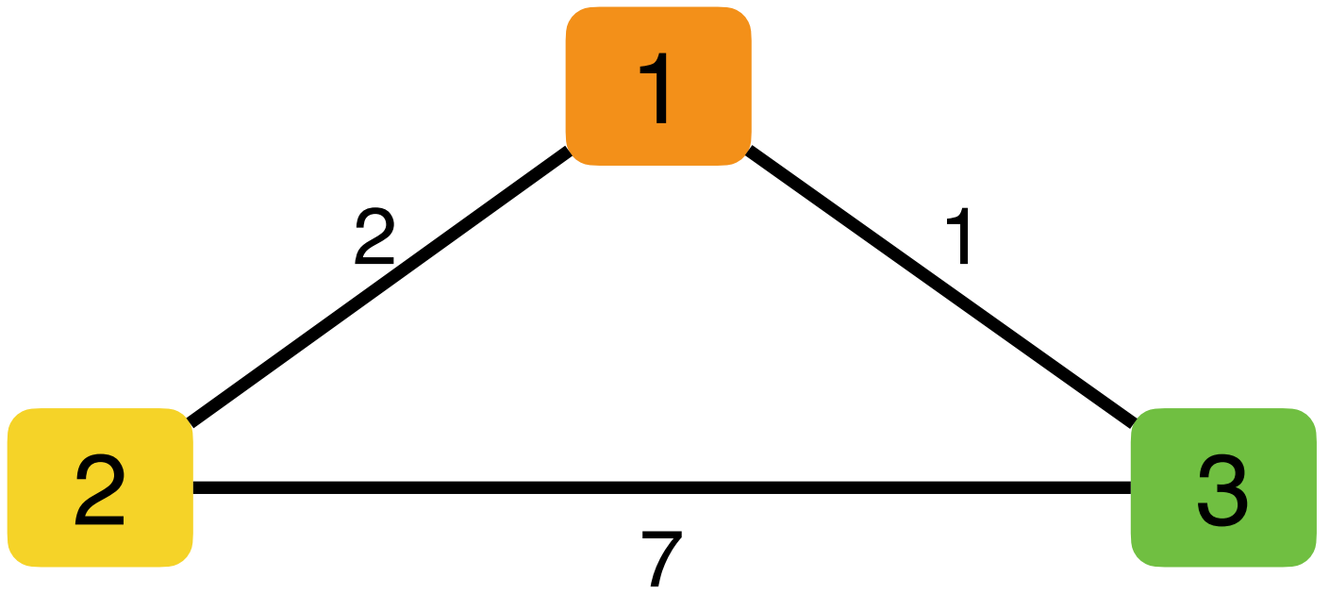
	distance	next-hop
1	0	-
2	infinity	
3	infinity	



	distance	next-hop
1	infinity	
2	0	-
3	infinity	

Round 1

	distance	next-hop
1	0	-
2	infinity	
3	infinity	

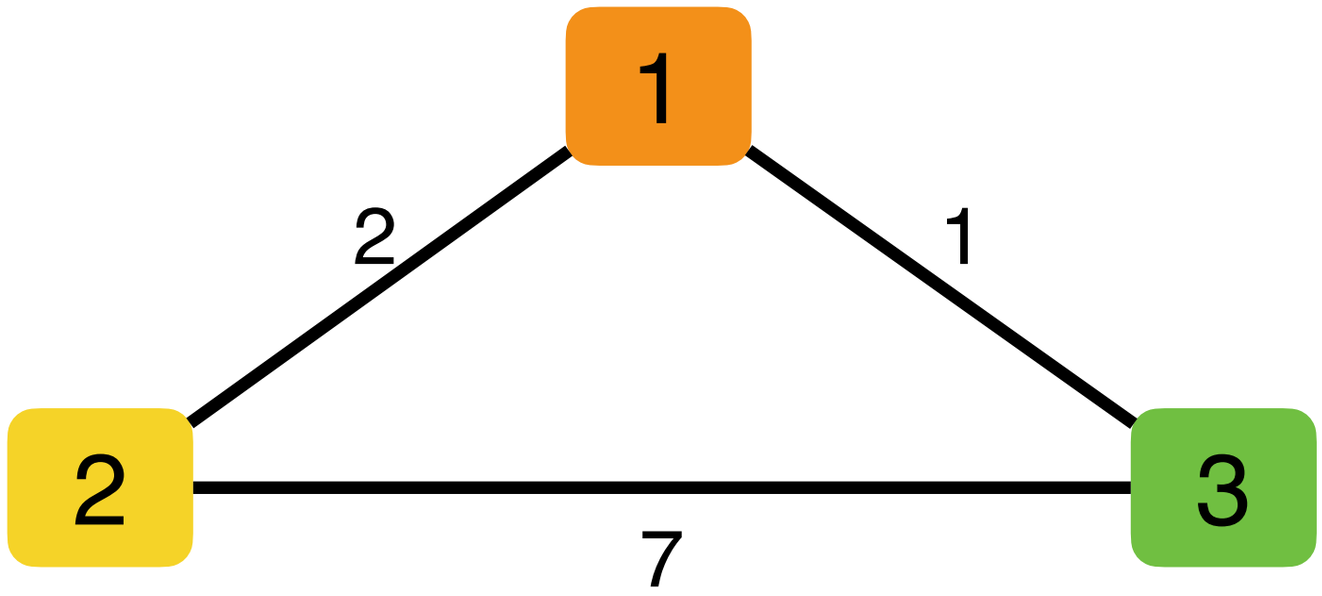


	distance	next-hop
1	infinity	
2	0	-
3	infinity	

	distance	next-hop
1	infinity	
2	infinity	
3	0	-

Round 2

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

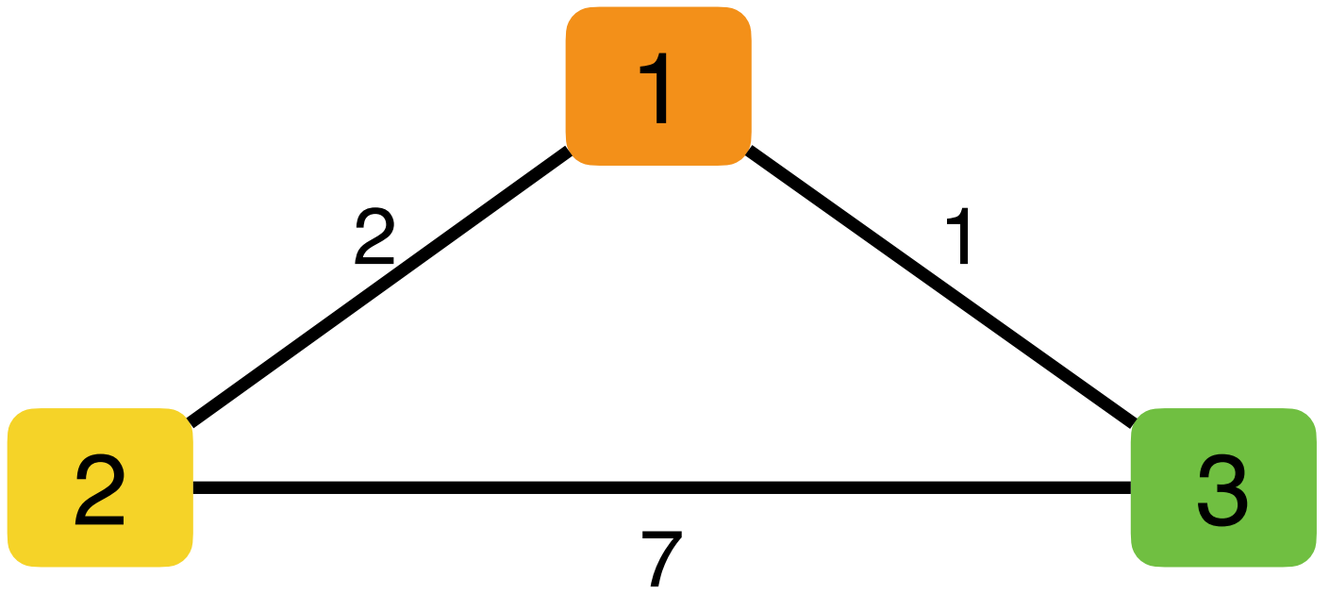


	distance	next-hop
1	**2**	**1**
2	0	-
3	**7**	**3**

	distance	next-hop
1	**1**	**1**
2	**7**	**2**
3	0	-

Round 3

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

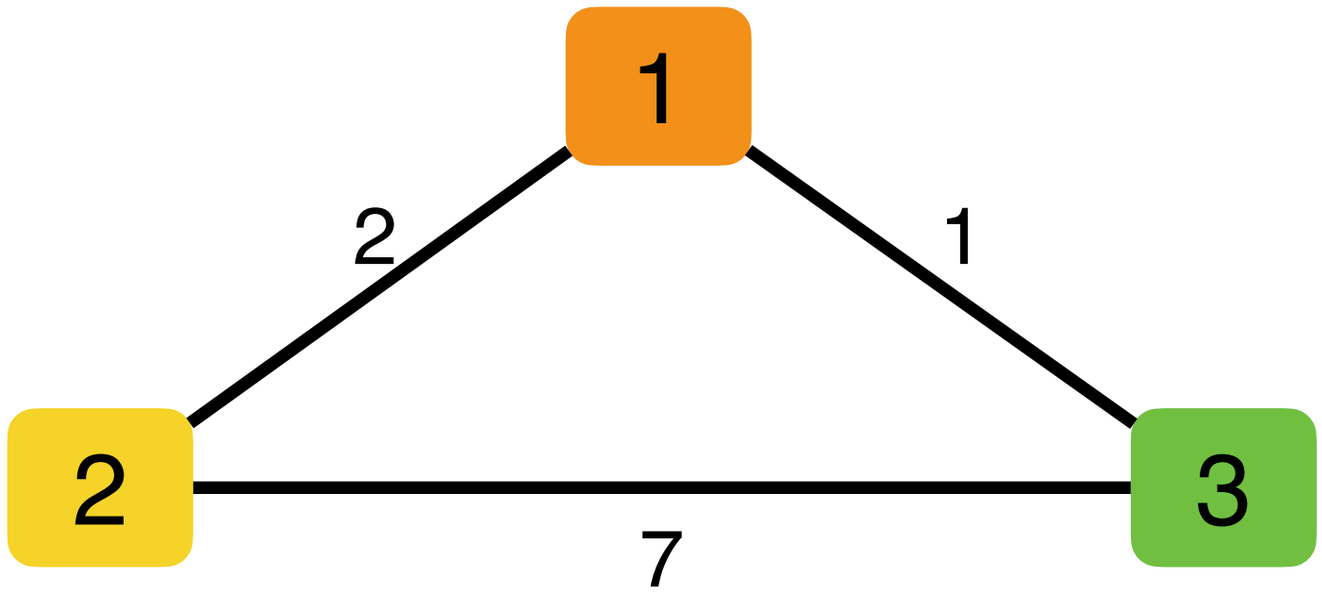


	distance	next-hop
1	2	1
2	0	-
3	**3**	**1**

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

Round 4

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



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

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

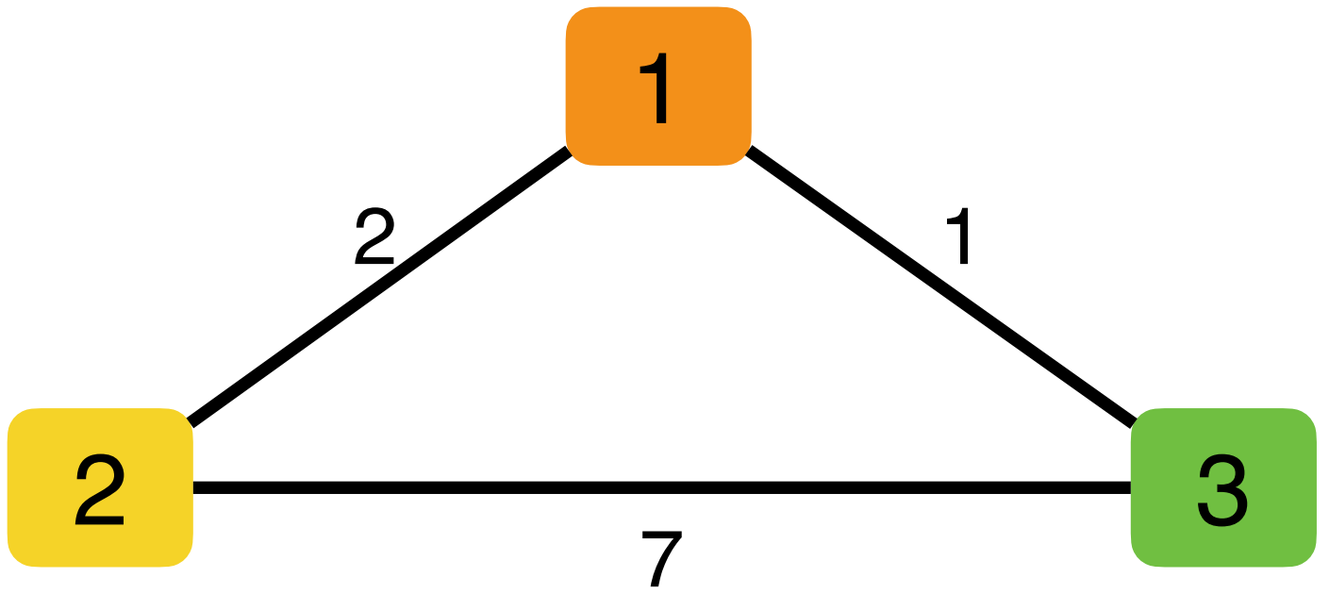
From Algorithm to Protocol

- Algorithm:
    - Nodes use Bellman-Ford to compute distances
- Protocol
    - Nodes exchange distance vectors
    - Update their own routing tables
    - And exchange again…
    - Details: when to exchange, what to exchange, etc….

Other Aspects of Protocol

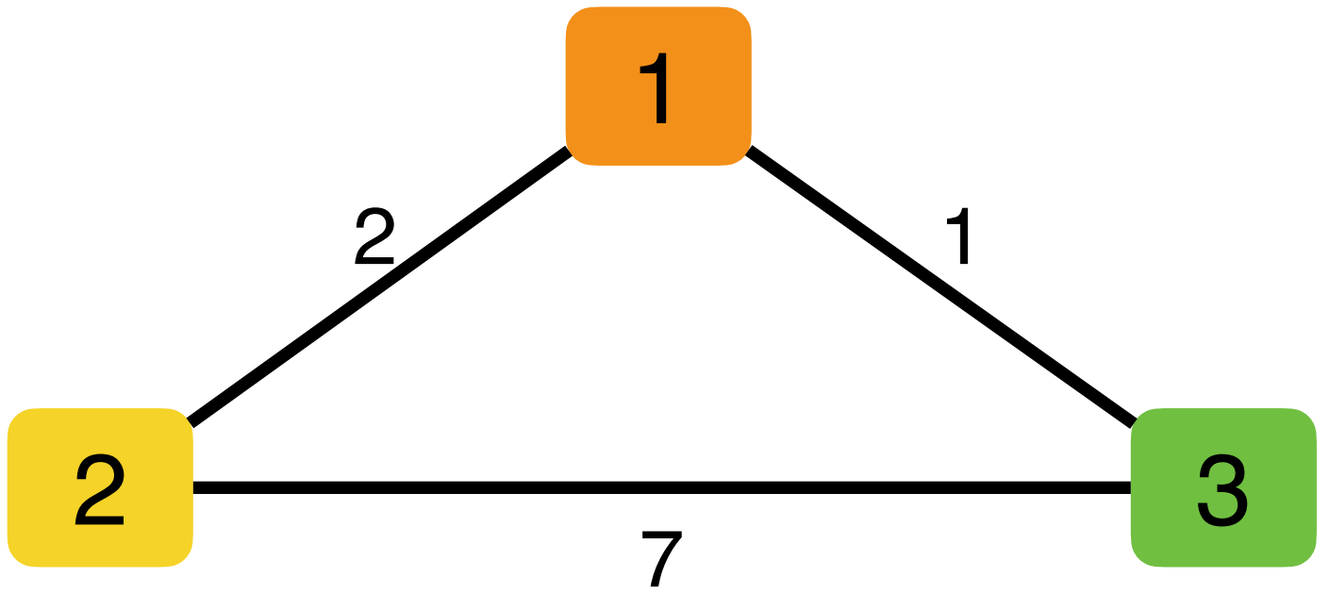
- When do you send messages?
    - When any of the distance changes
        - What about when the cost of a link changes?
    - Periodically, to ensure consistency between neighbors
- What information do you send?
    - Could send entire vector
    - Or just updated entries
- Do you send everyone the same information
    - Consider the following slides

Three node network



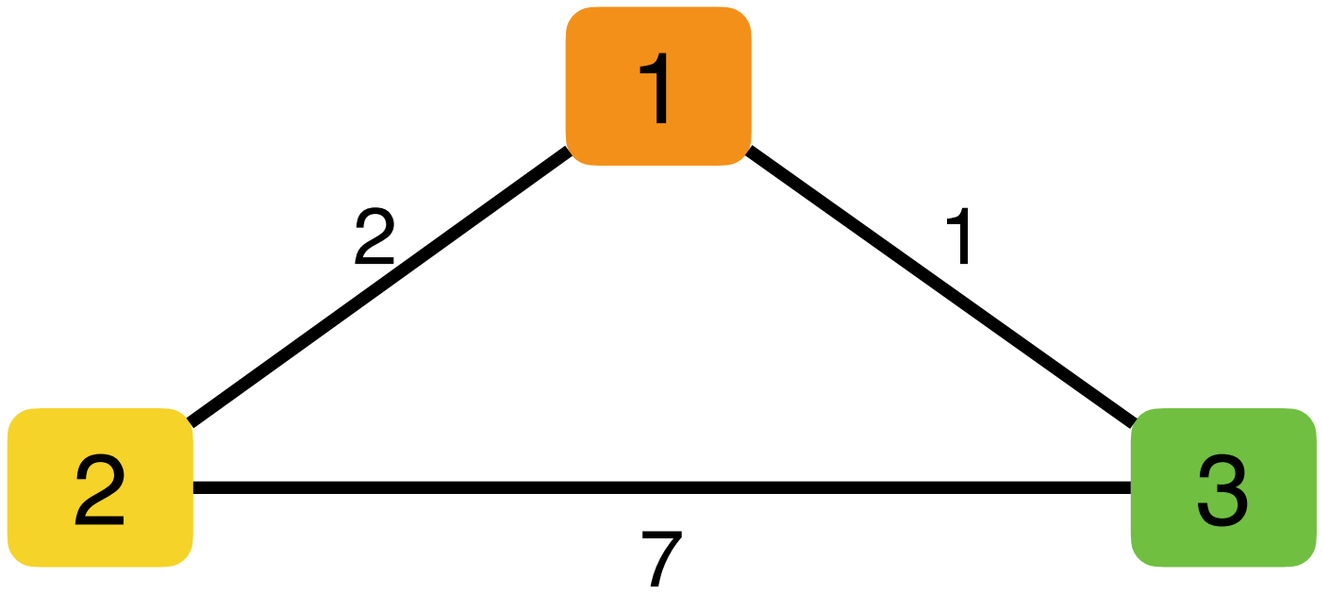
Three node network

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



Three node network

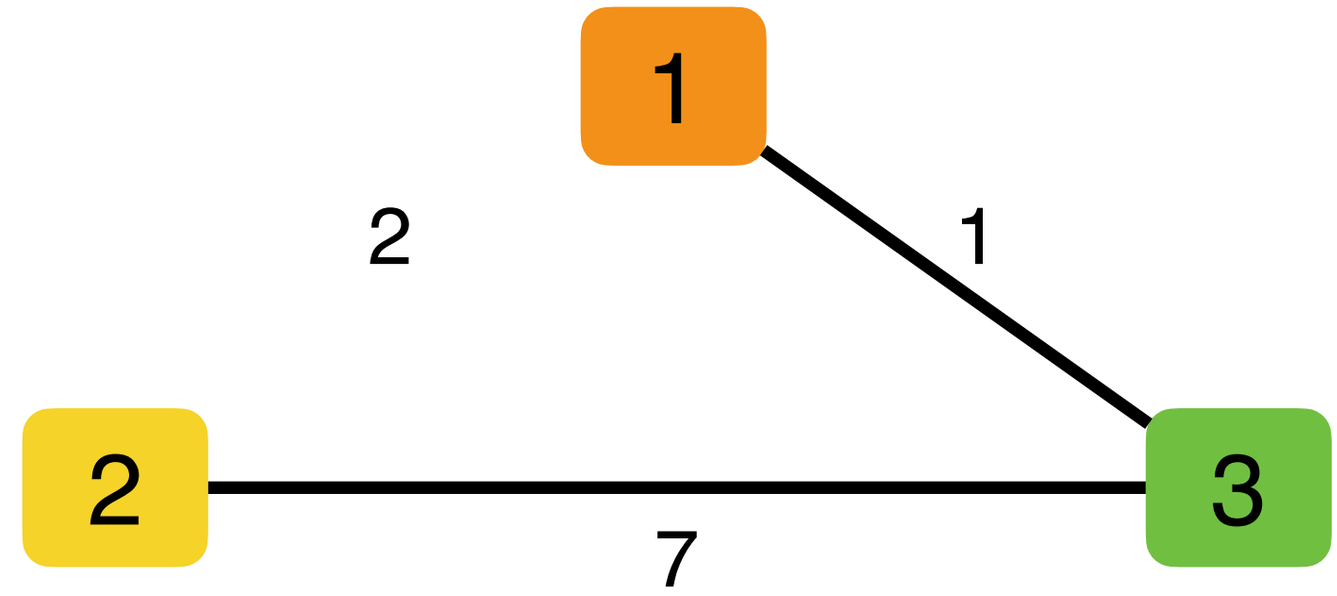
	distance	next-hop
1	0	-
2	2	2
3	1	3



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

Three node network

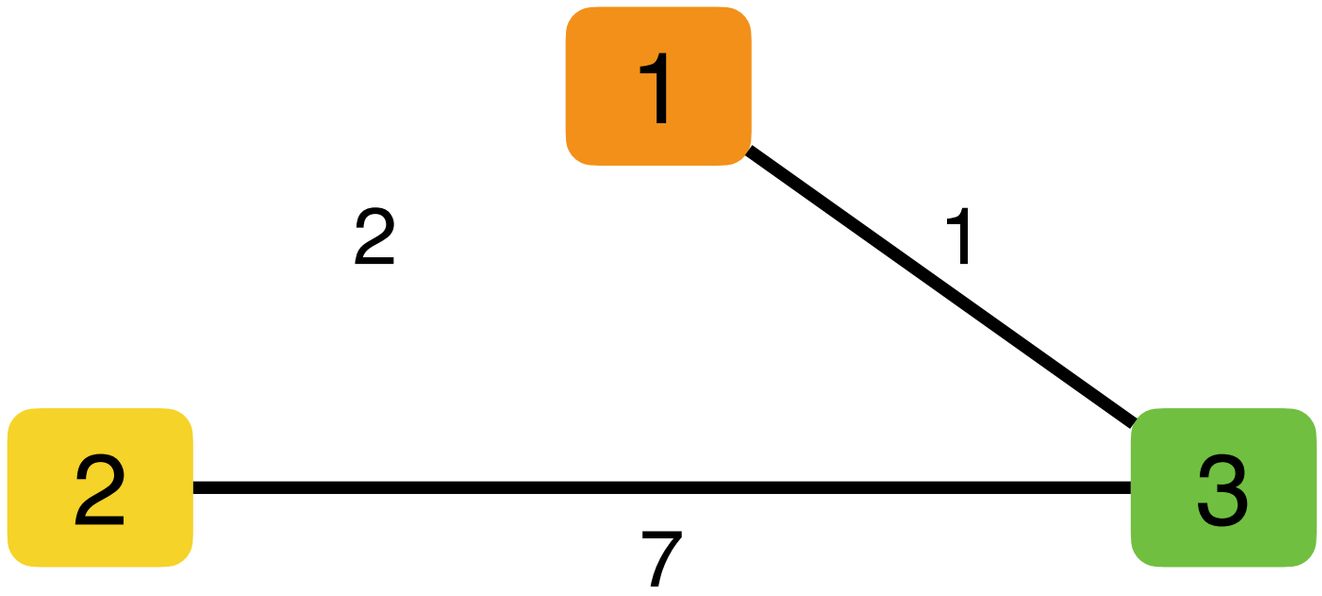
	distance	next-hop
1	0	-
2	infinity	
3	1	3



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

Round 1

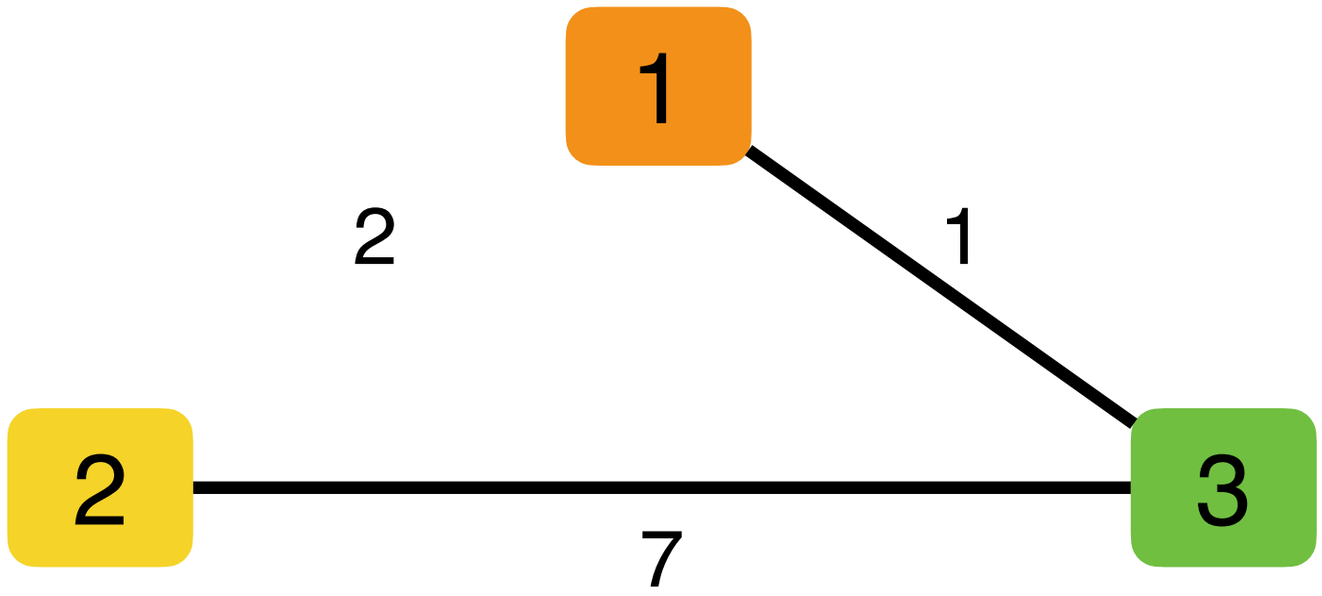
	distance	next-hop
1	0	-
2	**4**	**3**
3	1	3



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

Round 2

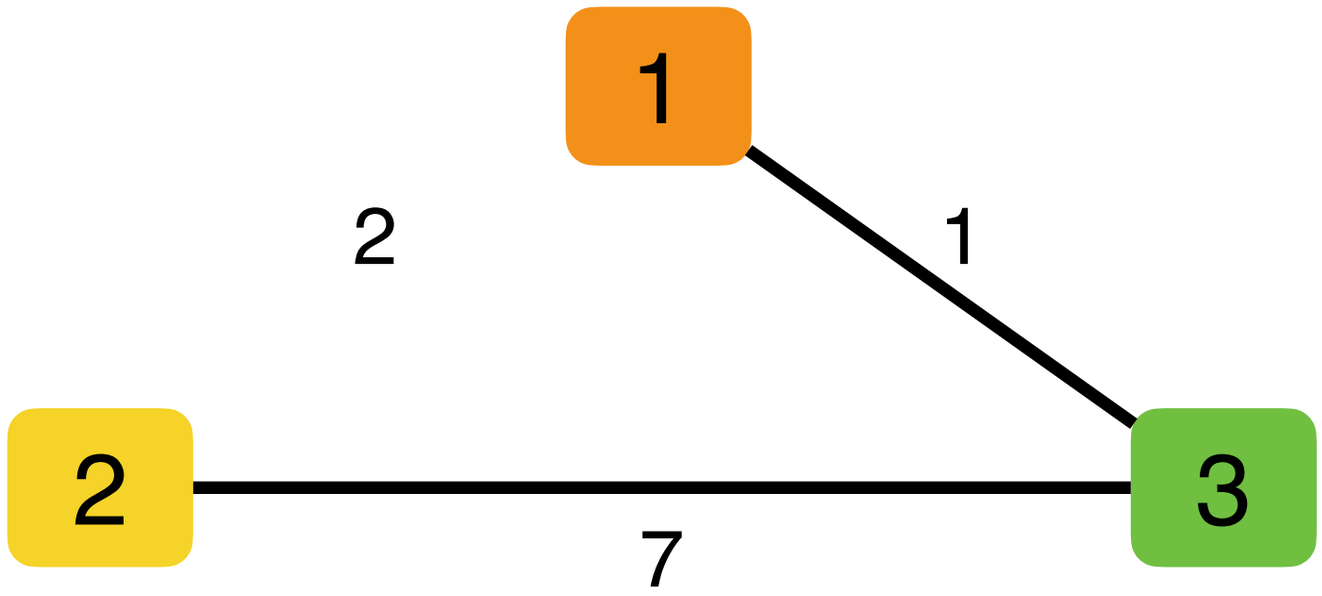
	distance	next-hop
1	0	-
2	4	3
3	1	3



	distance	next-hop
1	1	1
2	**5**	**1**
3	0	-

Round 3

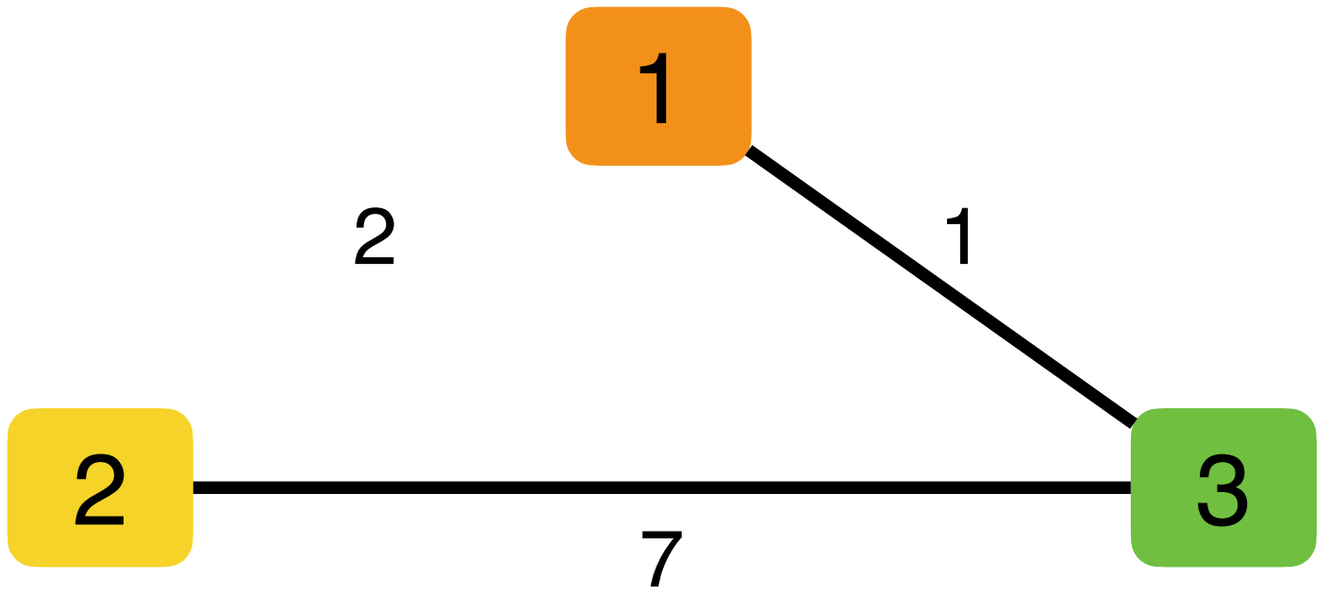
	distance	next-hop
1	0	-
2	**6**	**3**
3	1	3



	distance	next-hop
1	1	1
2	5	1
3	0	-

Round 4

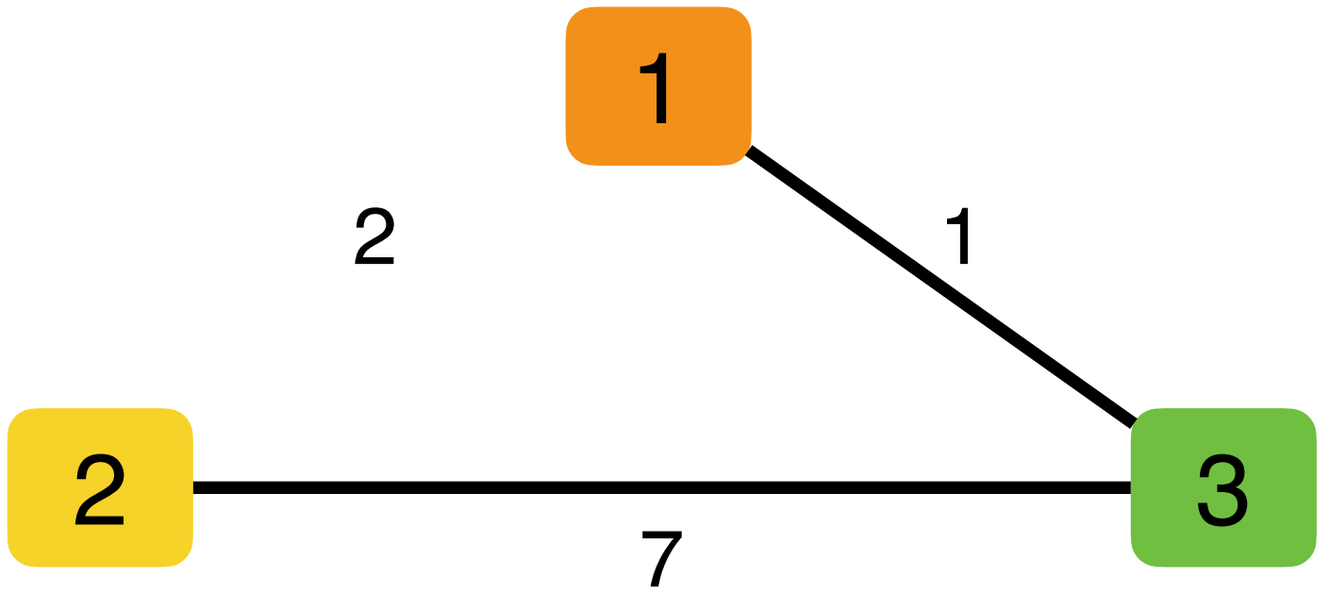
	distance	next-hop
1	0	-
2	6	3
3	1	3



	distance	next-hop
1	1	1
2	7	1
3	0	-

Round 4

	distance	next-hop
1	0	-
2	6	3
3	1	3

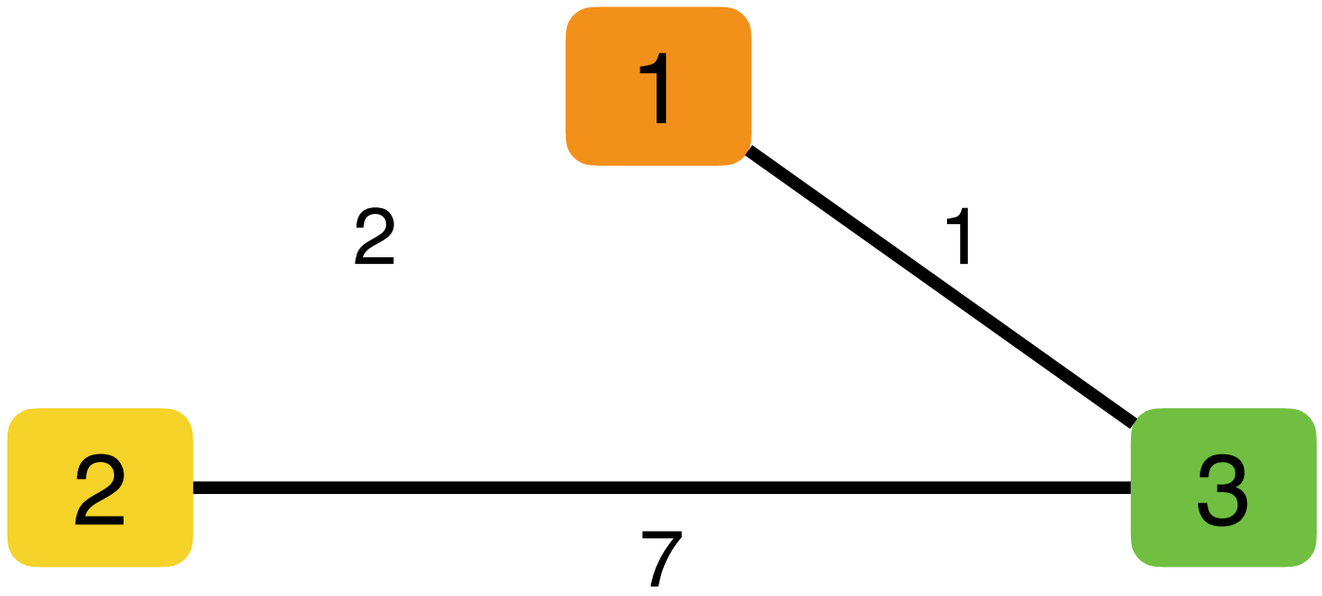


COUNT-TO-INFINITY problem!!!!

	distance	next-hop
1	1	1
2	7	1
3	0	-

Count-to-infinity problem

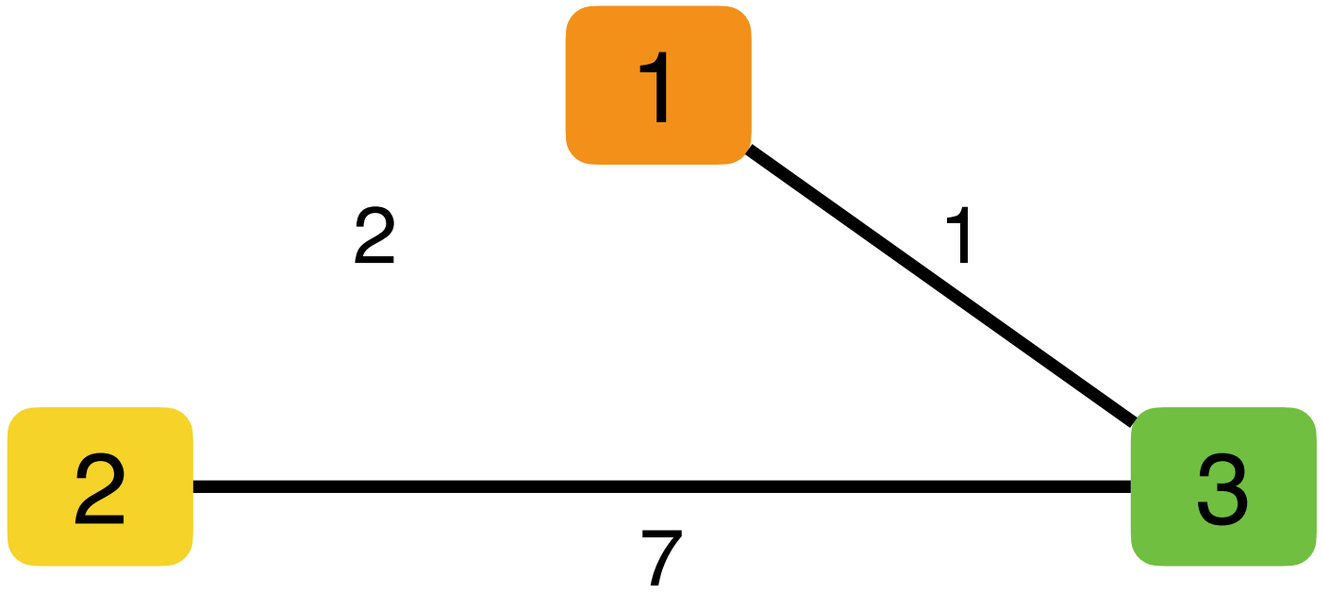
	distance	next-hop
1	0	-
2	6	3
3	1	3



	distance	next-hop
1	1	1
2	7	1
3	0	-

Count-to-infinity problem

	distance	next-hop
1	0	-
2	6	3
3	1	3



Not just due to failures:
Can happen with changes in cost!

	distance	next-hop
1	1	1
2	7	1
3	0	-

How Can You Fix This?

- **Do not advertise a path back to the node that is the next hop on the path**
  - Called **"split horizon"**
  - Telling them about your entry going through them
    - Doesn't tell them anything new
    - Perhaps misleads them that you have an independent path
- **Another solution: if you are using a next-hop's path, then**:
  - Tell them not to use your path (by telling them cost of infinity)
  - Called "**poisoned reverse**"

Convergence

- Distance vector protocols can converge slowly
    - While these corner cases are rare
    - The resulting convergence delays can be significant

Comparison of Scalability

- **Link-State:**
    - Global flood: each router's link-state (#ports)
    - Send it once per link event, or periodically
- **Distance Vector:**
    - Send longer vector (#dest) just to neighbors
        - But might end up triggering their updates
    - Send it every time DV changes (which can be often)
- **Tradeoff:**
    - LS: Send it everywhere and be done in predictable time
    - DV: Send locally, and perhaps iterate until convergence

End of Distance-vector Routing