

CS/ECE 4457

Computer Networks: Architecture and Protocols

Lecture 11

Intra-domain Routing: Deep Dive

Qizhe Cai



Goals for Today's Lecture

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

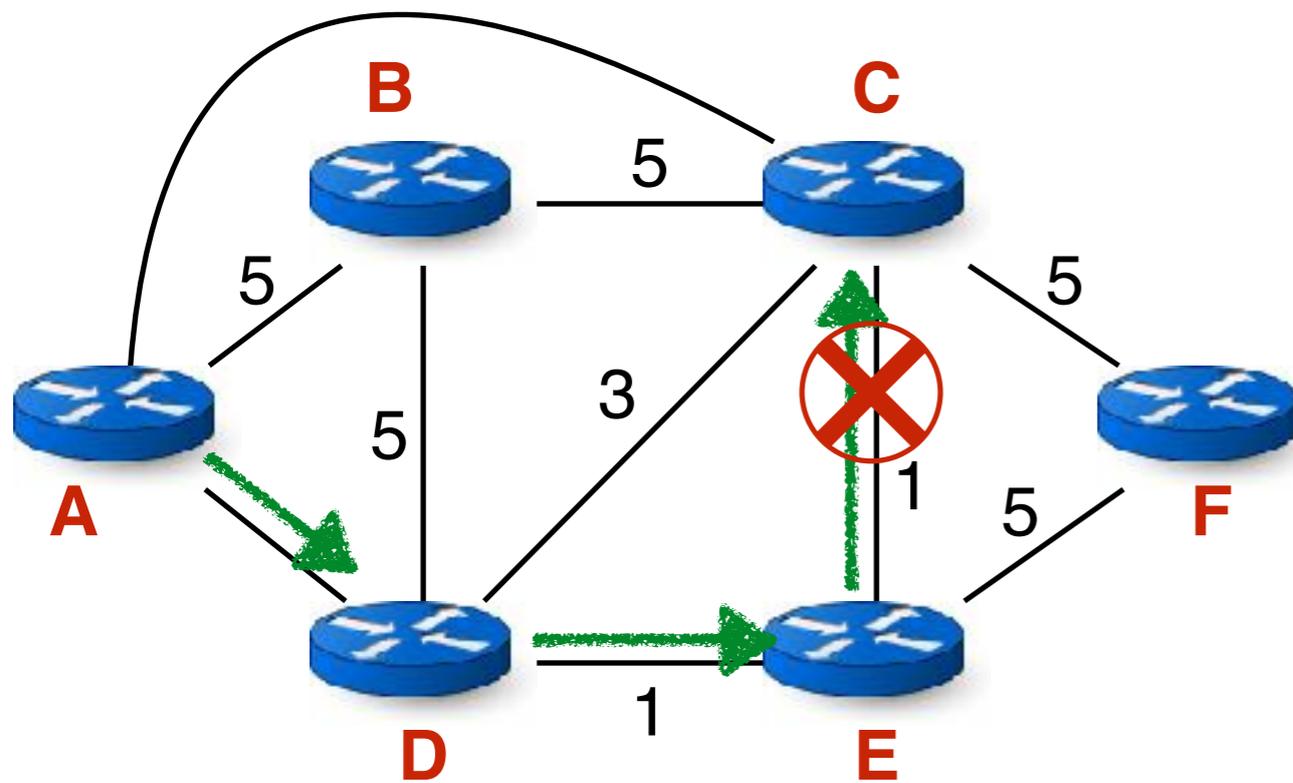
Recap: Three flavors of protocols for producing valid routing state

- **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?

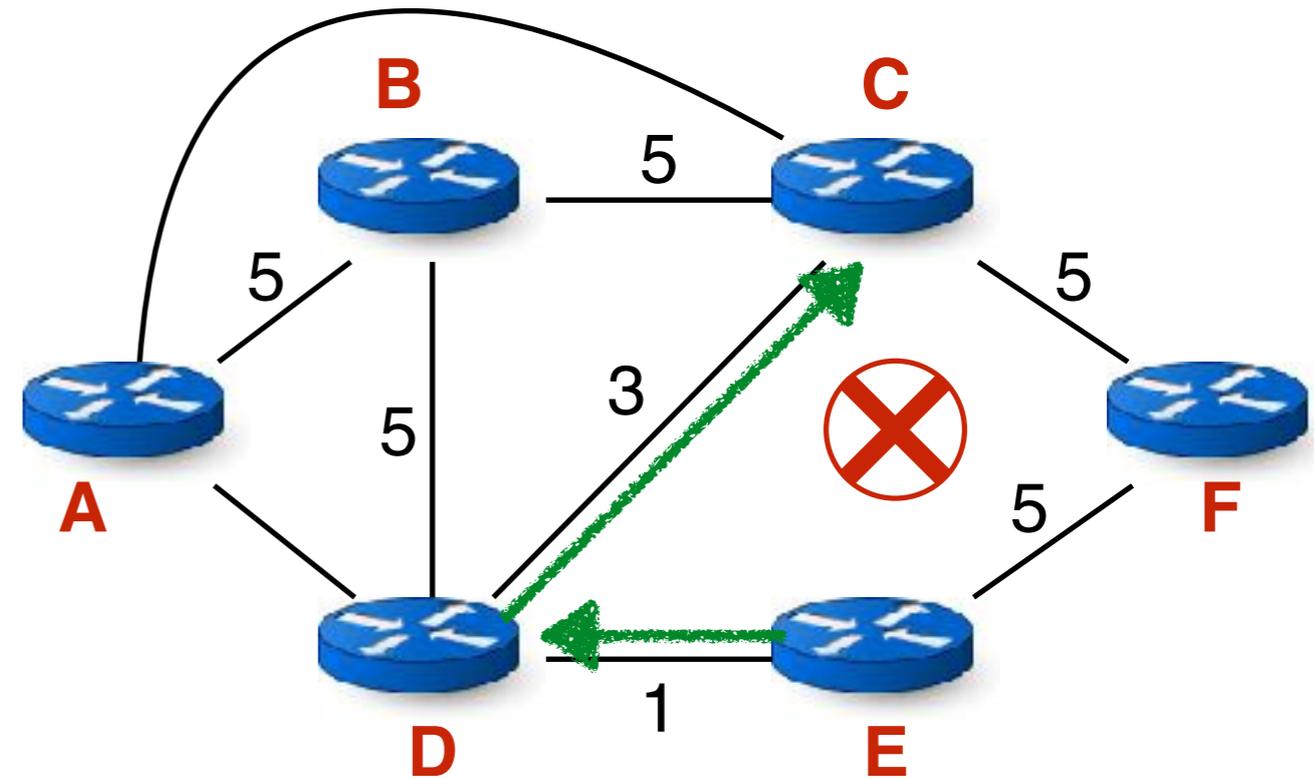
- One option: Central server
 - Collects a global view
 - Computes the routing table for each node
 - “Installs” routing tables at each node
 - **Software-defined Networks: later in course**
- Second option: At each router
 - Each router collects a global view
 - Computes its own routing table using 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

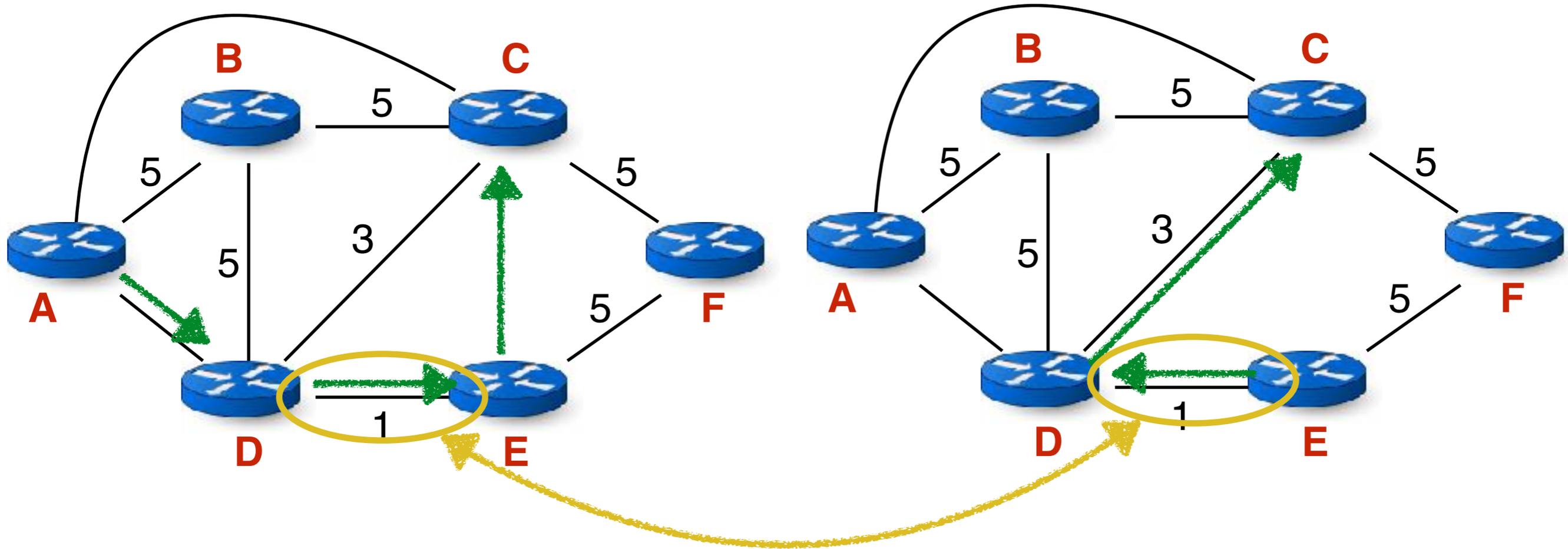
E-C link fails, but D doesn't know yet



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!

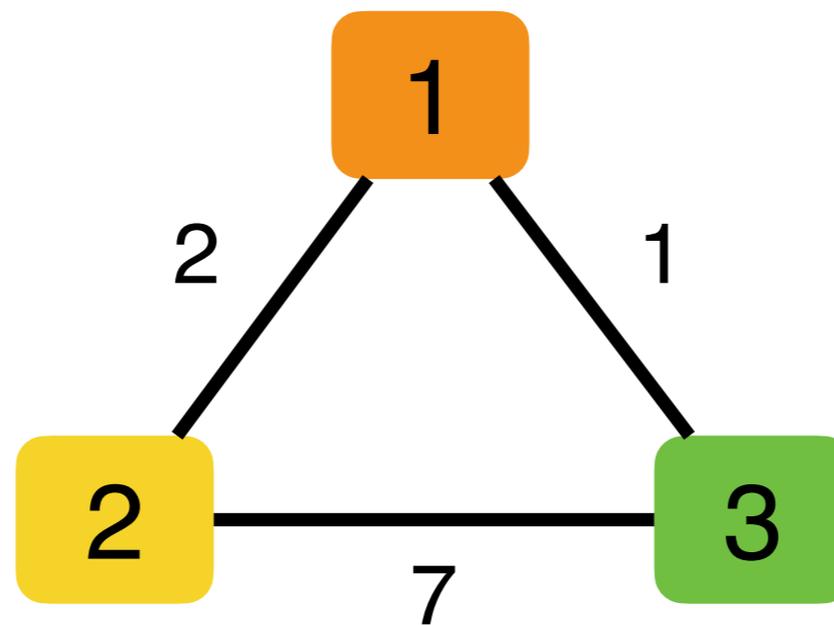
Questions?

Distributed Route Computation

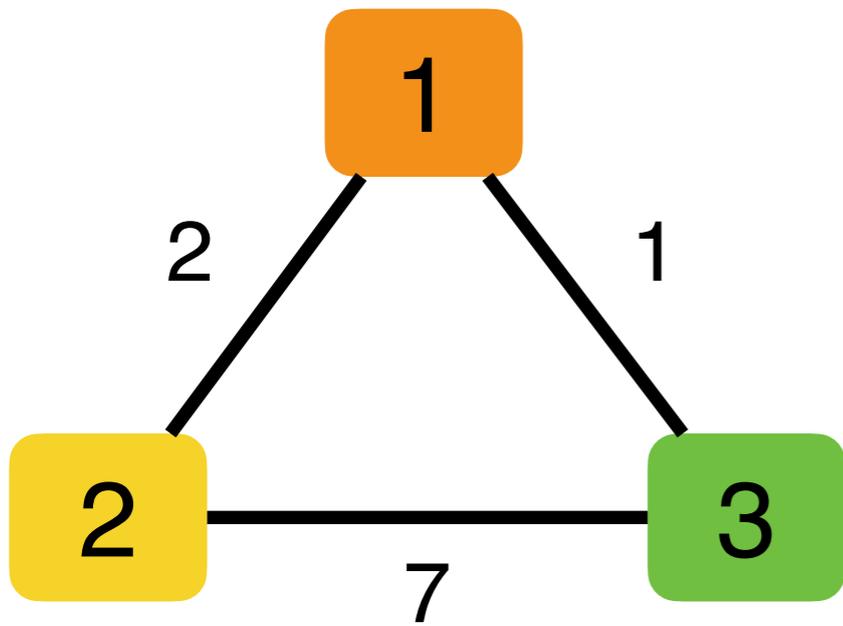
Recap: Distance-vector protocol with next-hops (no failures)

- **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 $\text{current_distance_to_}Y > d + \text{cost of link to } Z$:
 - update $\text{current_distance_to_}Y = d + \text{cost of link to } Z$
 - **update next_hop_to_destination = Z**
- If ~~root~~ **changed** OR shortest distance to the ~~root~~ destination **changed**, send all neighbors updated message $(Y, \text{current_distance_to_}Y, X)$

**Lets run the Protocol on this example
(with next-hops)**

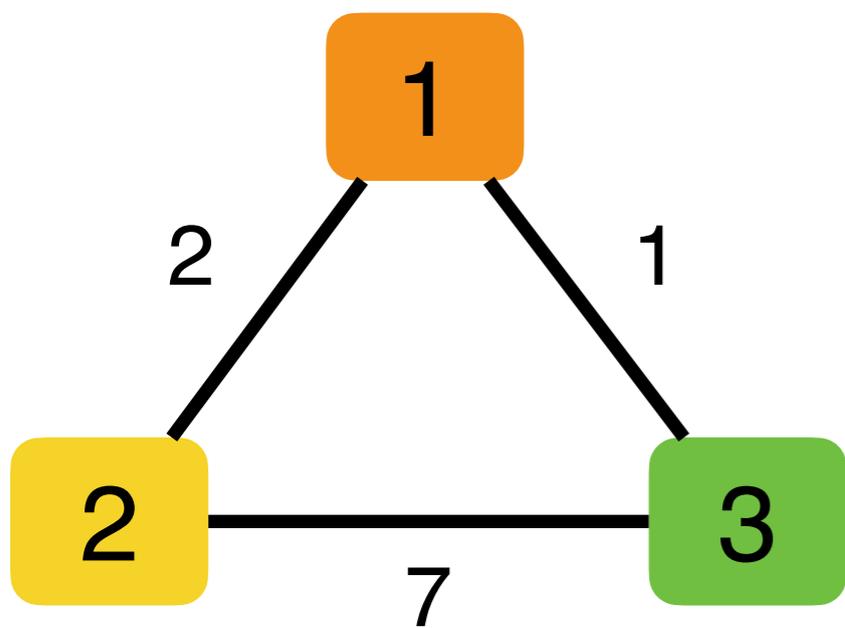


Round 1



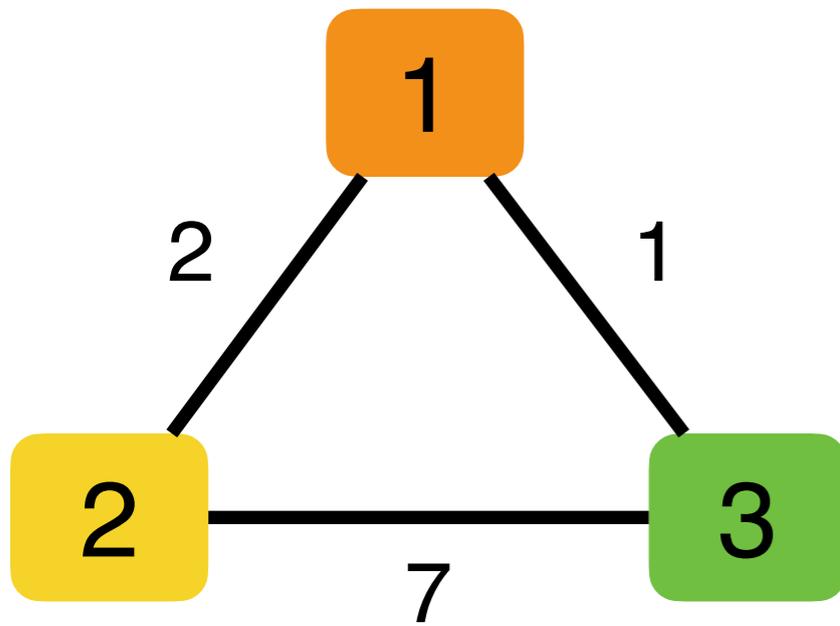
	Receive	Send	Next-hops
1		(1, 0, 1)	[-]
2		(2, 0, 2)	[-]
3		(3, 0, 3)	[-]

Round 2



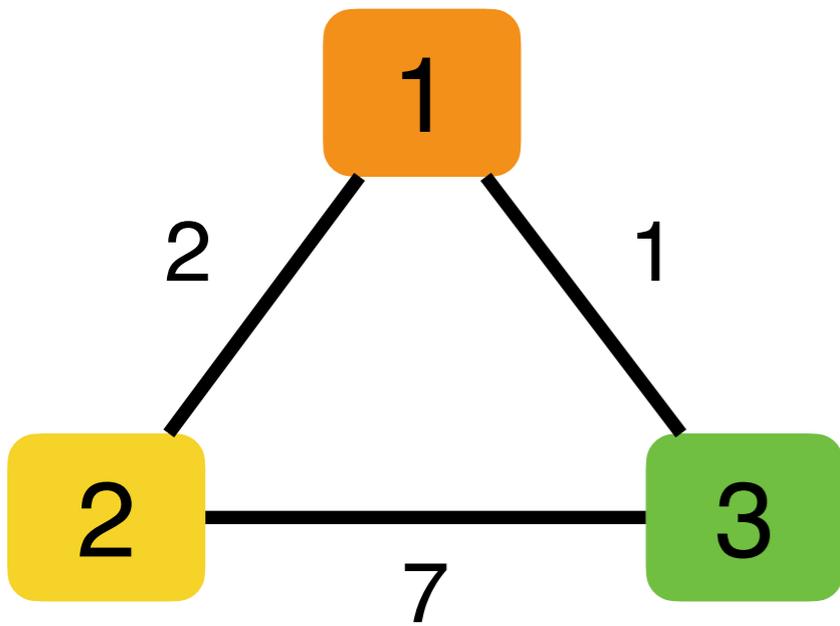
	Receive	Send	Next-hops
1 (1, 0, 1)	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 (2, 0, 2)	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)	[1, -, 3]
3 (3, 0, 3)	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)	[1, 2, -]

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 , -]

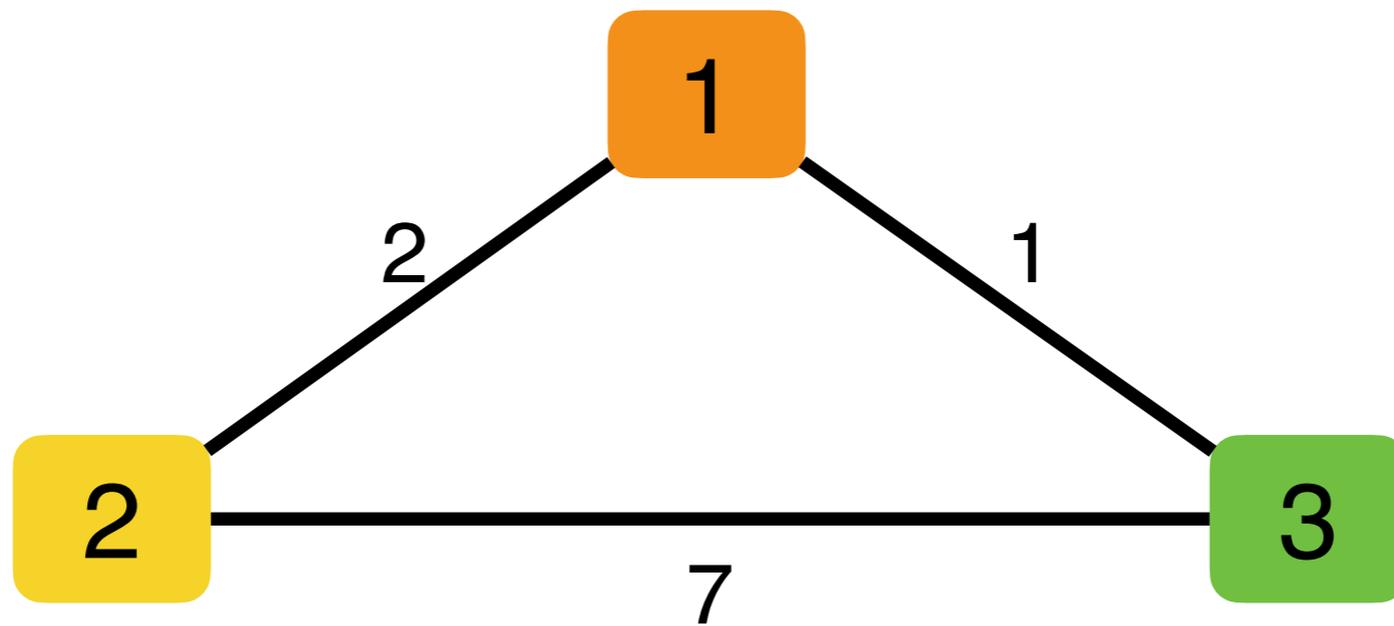
Round 4



	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, -]

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	-

Distance Vector Protocol

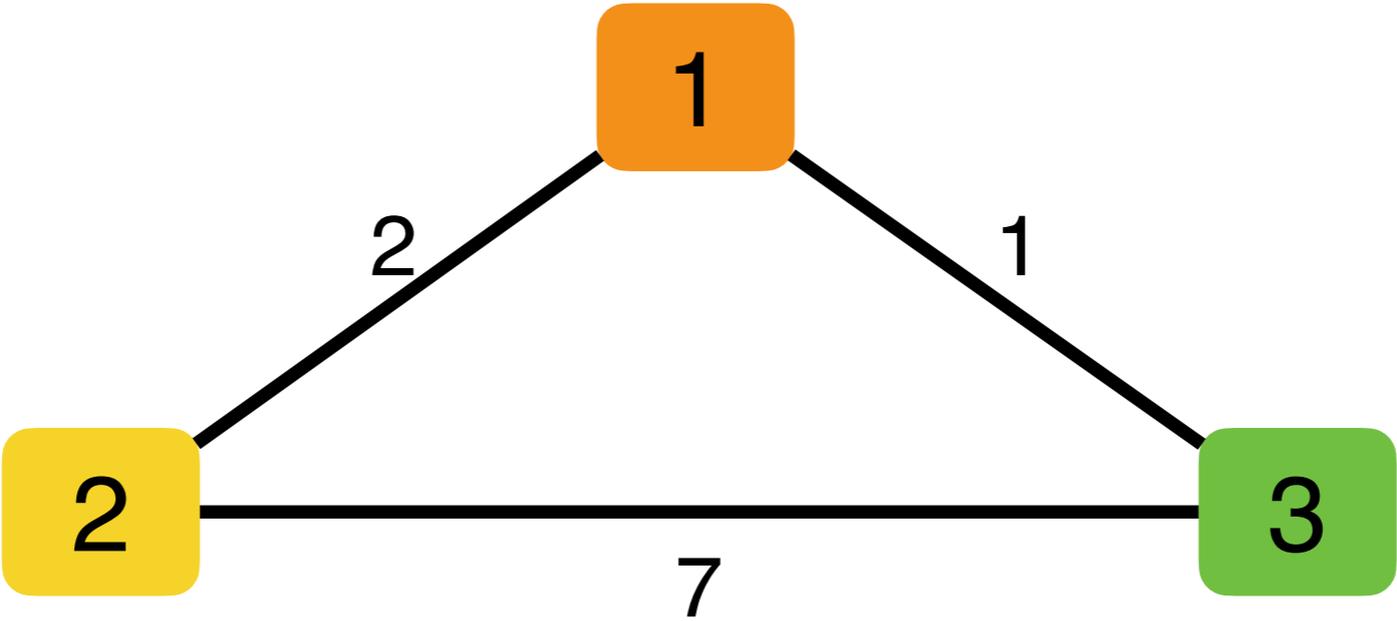
- **Messages (Y,d,X):** For root Y; From node X; advertising a distance d to Y
- Initially each switch X initializes its routing table to (X,0,-) and distance infinity to all other destinations
- Switches announce their entire distance vectors (routing table w/0 next hops)
- Upon receiving a routing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

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

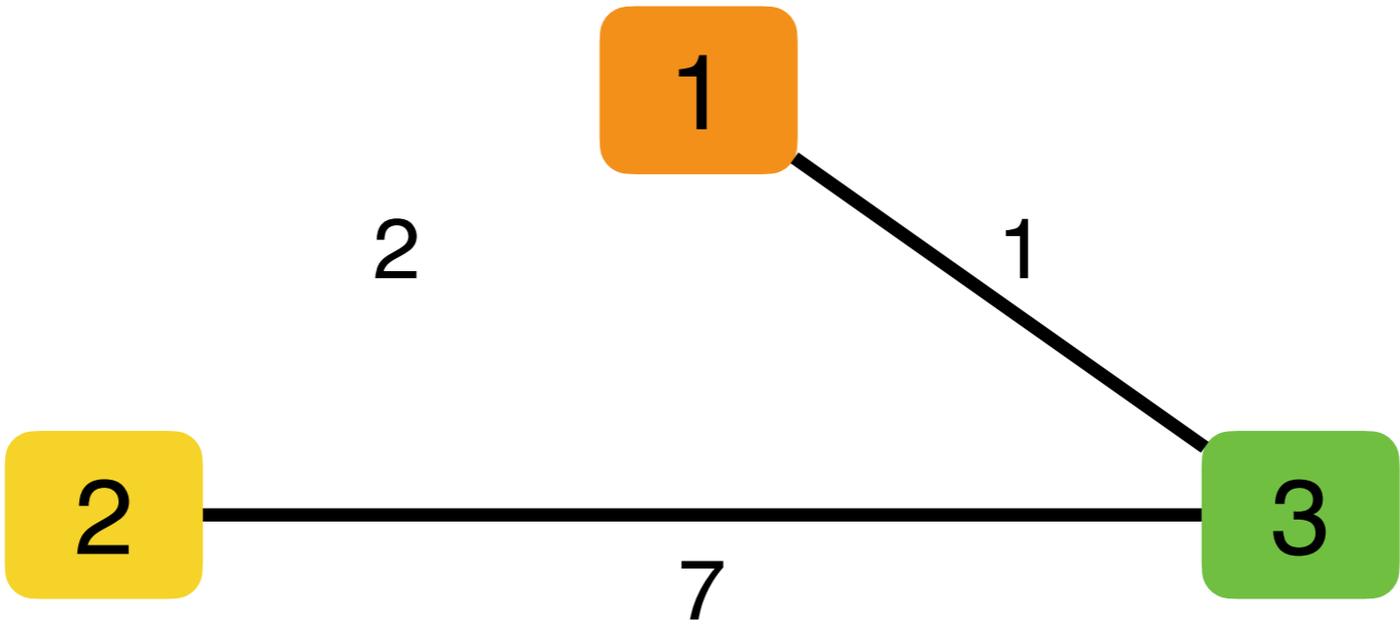
	distance	next-hop
1	0	-
2	2	2
3	1	3



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

Three node network

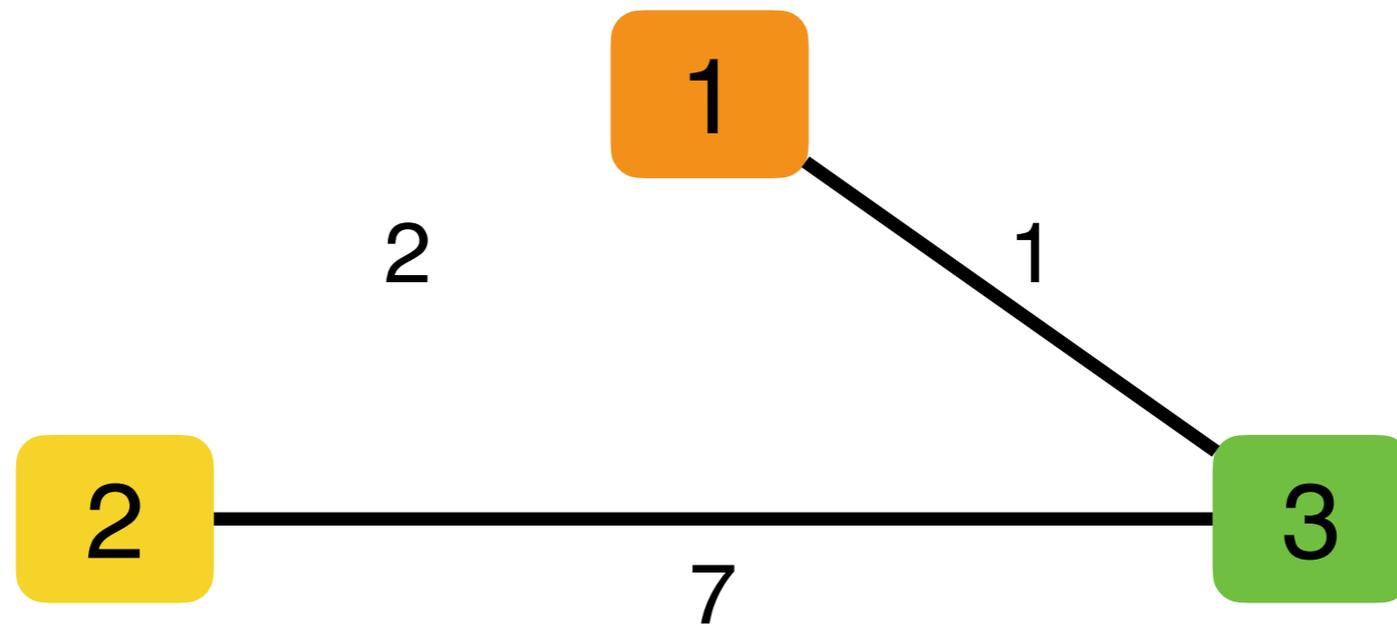
	distance	next-hop
1	0	-
2	infinity	
3	1	3



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

Round 1

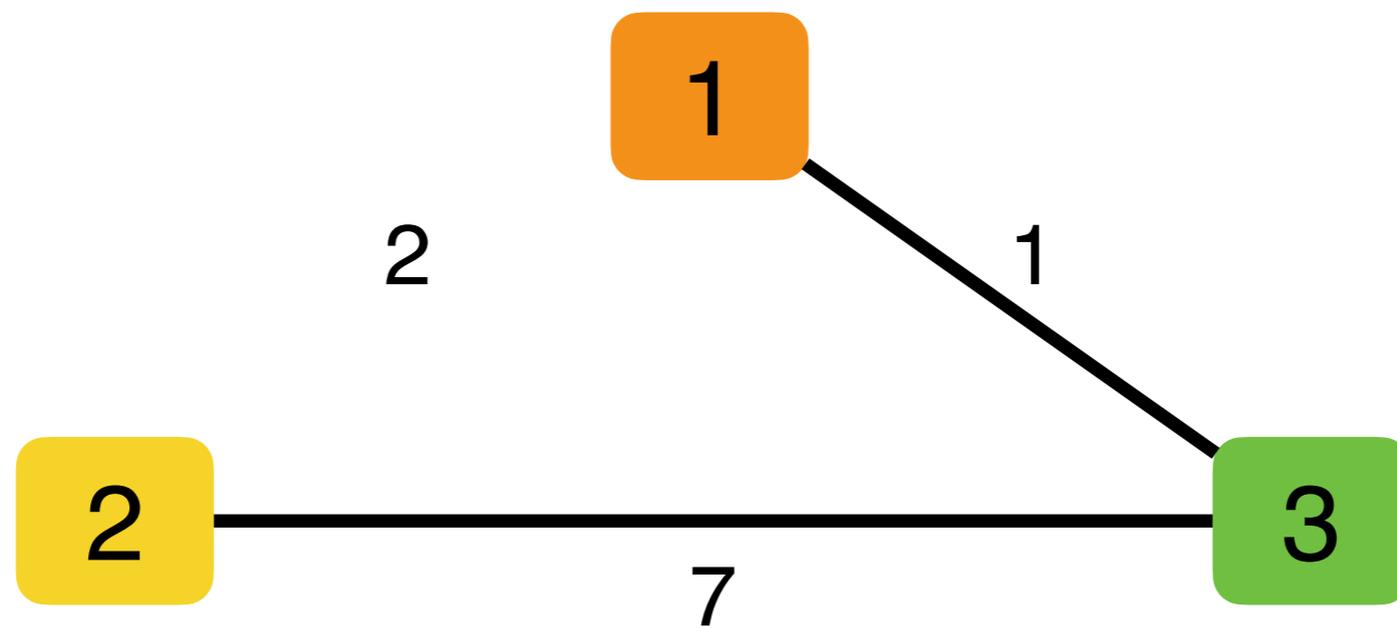
	distance	next-hop
1	0	-
2	4	3
3	1	3



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

Round 2

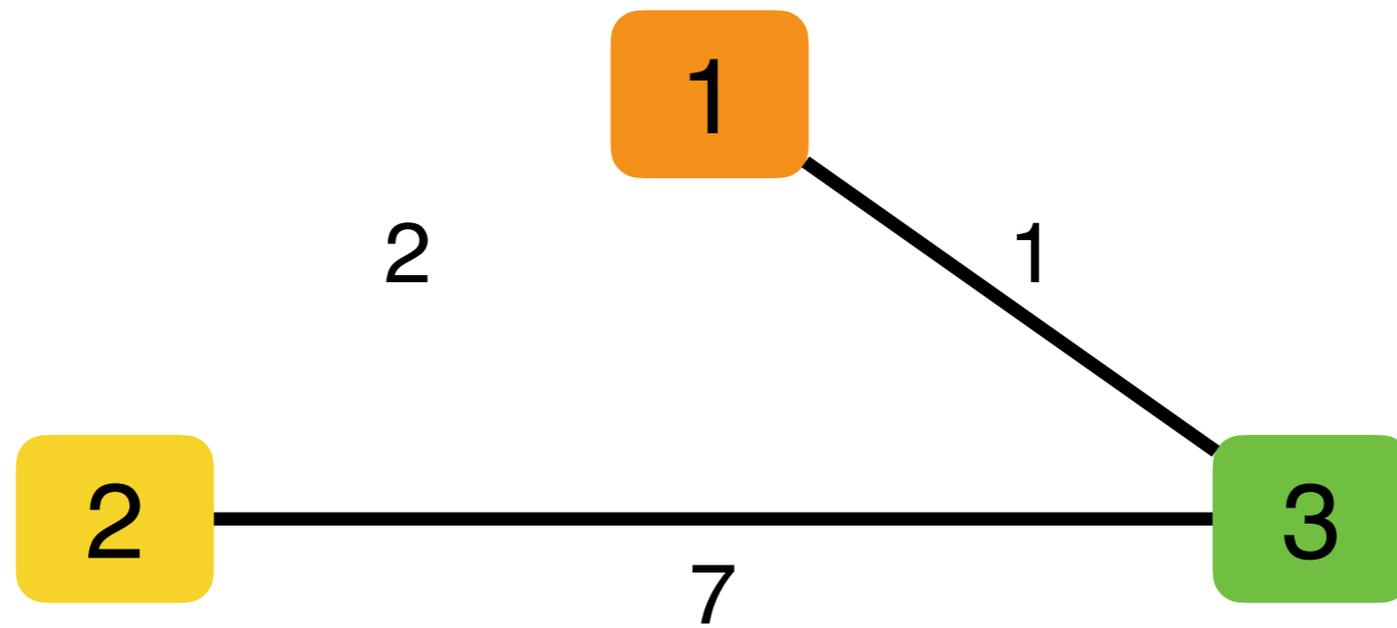
	distance	next-hop
1	0	-
2	4	3
3	1	3



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

Round 3

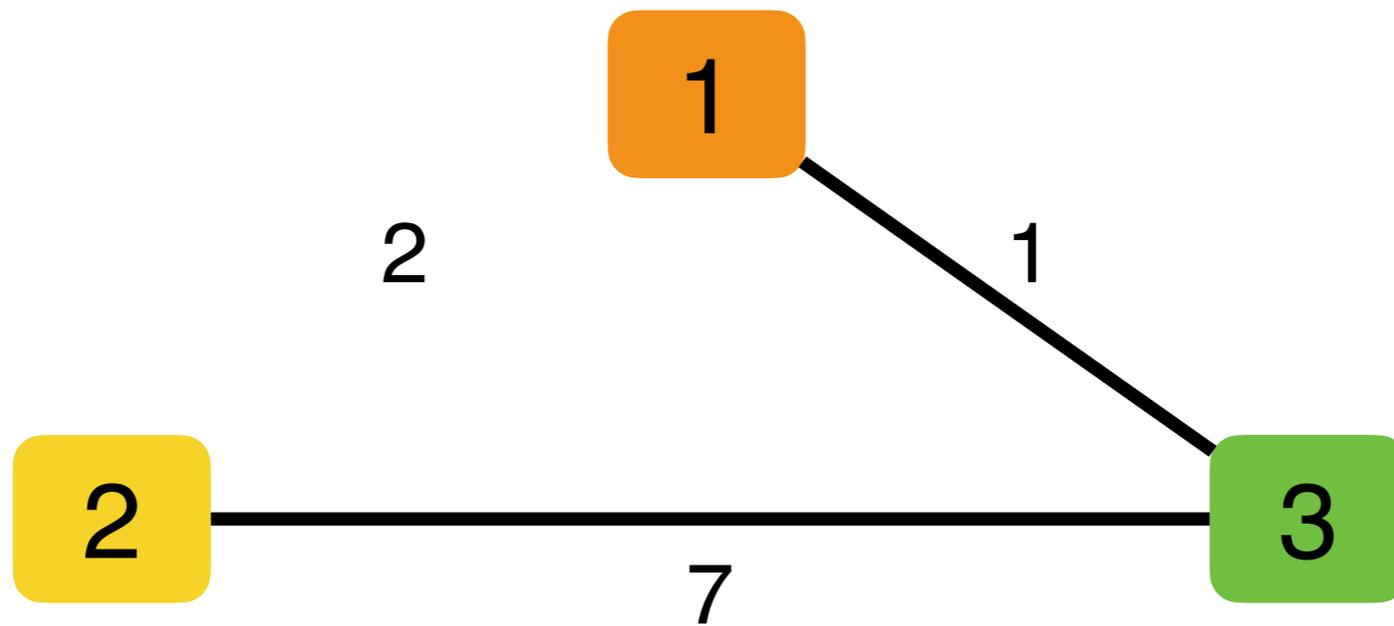
	distance	next-hop
1	0	-
2	6	3
3	1	3



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

Round 4

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

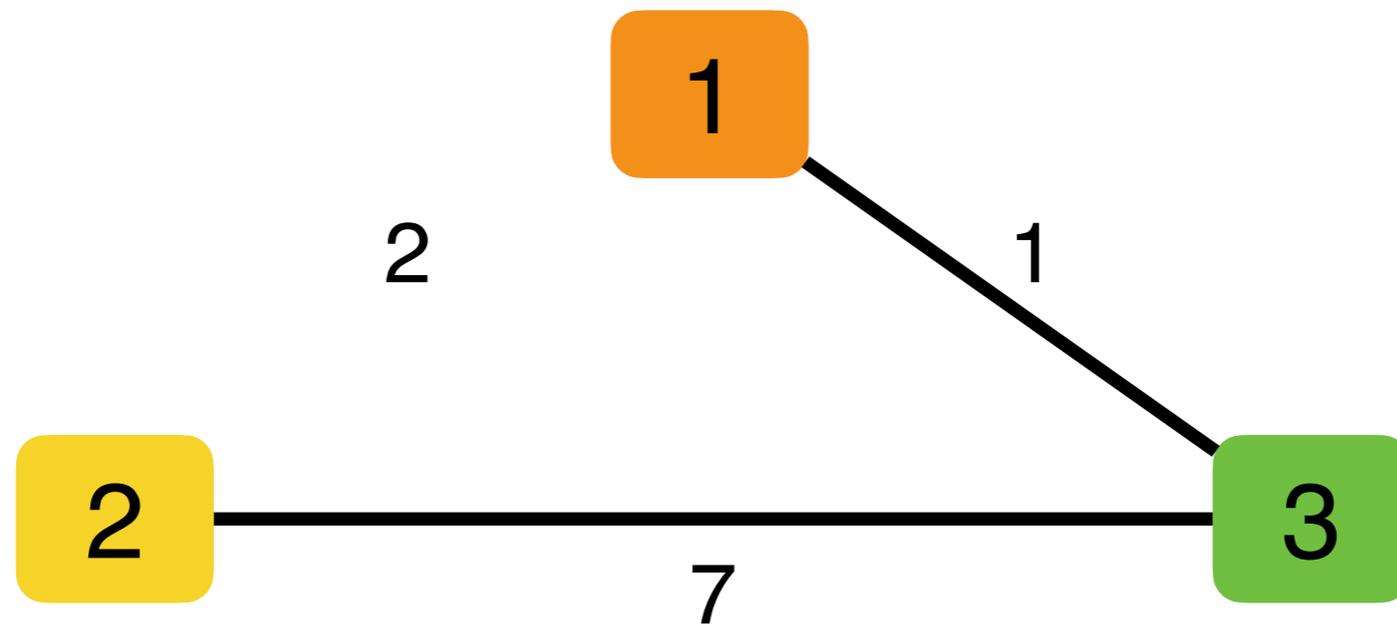


**COUNT-TO-INFINITY
problem!!!!**

	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



**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

Internet Addressing

Addressing so far

- Each node has a “name”
 - We have so far worked only with names
 - Assumed that forwarding/routing etc. done on names
- Today:
 - Why do we need addresses?
 - Why do we assign addresses the way we assign addresses?

Three requirements for addressing

- **Scalable routing**
 - How much state must be stored to forward packets?
 - How much state needs to be updated upon host arrival/departure?
- **Efficient forwarding**
 - How quickly can one locate items in routing table?
- **Host must be able to recognize packet is for them**

Layer 2 (link layer): “Flat” Addressing

- Uses MAC address
 - “Names”, remember? Used as identifier
- Unique identifiers hardcoded in the hardware
 - No location information
- Local area networks route on these “flat” addresses
 - **That is, the routing protocol runs on switches**
 - Each switch stores a separate routing entry **for each host**
 - End-hosts store nothing
- When sending a packet, an end-host:
 - Puts destination’s and its own MAC address in the header
 - Forwards it to the switch it is connected to
- **Destination is able to recognize the packet is for them using address**

How does this meet our requirements?

- **Scalable routing**

- How much state to forward packets?
 - One entry per host per switch
- How much state updated for each host arrival/departure?
 - One entry per host per switch

- **Efficient forwarding**

- Exact match lookup on MAC addresses (exact match is easy!)

- **Host must be able to recognize the packet is for them**

- MAC address does this perfectly

Conclusion: L2 addressing does not enable scalable routing

How would you scale L2?

- Suppose we want to design a much larger L2 network
- Must use MAC address as part of the address
 - Only way host knows that the packet is for them
- **But how would you enable scalable routing?**
 - Small #routing entries (less than one entry per host per switch)
 - Small #updates (less than one update per switch per host change)

One possible Solution: Towards Internet-scale addressing

- Assign each end-host an addresses of the form — Switch:MAC
- So, each switch has one entry per switch (rather than per host)
- When sending a packet, an end-host:
 - Puts destination's and its own Switch:MAC address in the header
 - Forwards it to the switch it is connected to
- **Switches forward the packet using first part of the address**
- **Destination is able to recognize the packet is for them using second part of the address**

Layer 3: Hierarchical addressing

- Routing tables cannot have entry for each switch in the Internet
- Use addresses of the form — Network:Host
- Routers know how to reach all networks in the world
 - Routing algorithms only announce “Network” part of the addresses
 - Routing tables now store a next-hop for each “network”
- Forwarding:
 - Routers ignore host part of the address
 - When the packet reaches the right network
 - Packet forwarded using Host part of the address
- **This was the original IP addressing scheme**

Aggregation

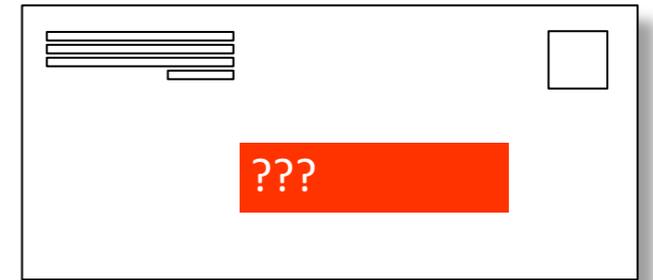
- **Aggregation:** single forwarding entry used for many individual hosts
- Example:
 - In our scalable L2 solution: aggregate was switch
 - In our scalable L3 solution: aggregate was network
- Advantages:
 - Fewer entries and more stable
 - Change of hosts do not change tables
 - Don't need to keep state on individual hosts

Hierarchical Structure

- The Internet is an “inter-network”
 - Used to connect networks together, not hosts
- Forms a natural two-way hierarchy
 - Wide Area Network (WAN) delivers to the right “network”
 - Local Area Network (LAN) delivers to the right host

Hierarchical Addressing

- Can you think of an example?
- Addressing in the US mail
 - Country
 - City, Zip code
 - Street
 - House Number
 - Occupant “Name”



IP addresses

- Unique 32 bit numbers associated with a host
- Use dotted-quad notation, e.g., 128.84.139.5

Country	City, State	Street, Number	Occupant
(8 bits)	(8 bits)	(8 bits)	(8 bits)
10000000	0-1010100	10001011	00000-101
128	84	139	5



Network

Host

Original Addressing mechanism

- First eight bits: network address (/8)
 - Slash notation indicates network address
- Last 24 bits: host address
- Assumed 256 networks were more than enough!!!
 - Now we have millions!

Suppose we want to accommodate more networks

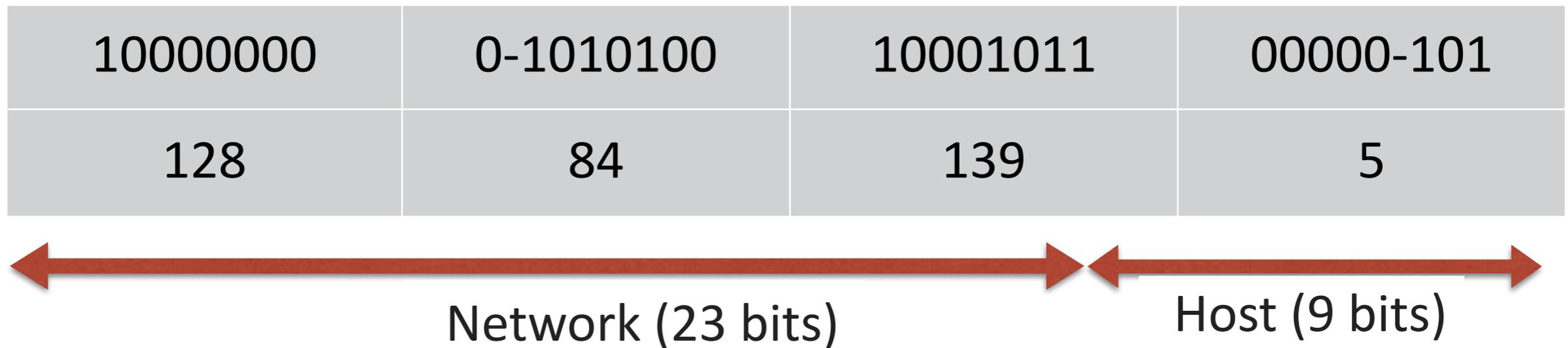
- We can allocate more bits to network address
- Problem?
 - Fewer bits for host names
 - What if some networks need more hosts?

Today's Addressing: CIDR

- Classless Inter-domain Routing
- Idea: Flexible division between network and host addresses
- Prefix is **network address**
- Suffix is **host address**
- **Example:**
 - **128.84.139.5/23 is a 23 bit prefix with:**
 - First 23 bits for network address
 - Next 9 bits for host addresses: maximum 2^9 hosts
- **Terminology: "Slash 23"**

Example for CIDR Addressing

- **128.84.139.5/23** is a 23 bit prefix with 2^9 host addresses



Allocating addresses

- Internet Corporation for Assigned Names and Numbers (ICANN) ...
- Allocates large blocks of addresses to Regional Internet Registries
 - E.g., American Registry for Internet Names (ARIN) ...
- That allocates blocks of addresses to Large Internet Service Providers (ISP)
- That allocate addresses to individuals and smaller institutions
- Fake example:
 - ICANN -> ARIN -> AT&T -> Virginia -> CS -> Me

Allocating addresses: Fake example

- ICANN gives ARIN several /8s
- ARIN given AT&T one /8, **128.0/8**
 - **Network prefix: 10000000**
- AT&T gives Virginia one /16, **128.84/16**
 - **Network prefix: 10000000 01010100**
- Virginia gives CS one /24, **128.84.139/24**
 - **Network prefix: 10000000 01010100 10001011**
- CS given me a specific address **128.84.139.5**
 - **Network prefix: 10000000 01010100 10001011 00000101**

How does this meet our requirements?

- To understand this, we need to understand the routing on the Internet
- And to understand that, we need to understand the Internet

More next lecture!