

# EN.601.414/614

# Computer Networks

## Routing Algorithms

Xin Jin

Fall 2020 (TuTh 1:30-2:45pm on Zoom)



<https://github.com/xinjin/course-net>

# Agenda

- **Recap: least-cost path routing**
- **Link-state routing**
- **Distance-vector routing**

# Recap: Least-cost path routing

- **Given:** router graph & link costs
- **Goal:** find least-cost path
  - From each source router to each destination router
- **Easy way to avoid loops**
  - No reasonable cost metric is minimized by traversing a loop

# Recap:

## Dijkstra's algorithm

- **Network topology, link costs known to all nodes**
  - All nodes have same info
- **Each node (“src”) computes least-cost paths to all other nodes**
  - After k iterations, know least-cost path to k destinations

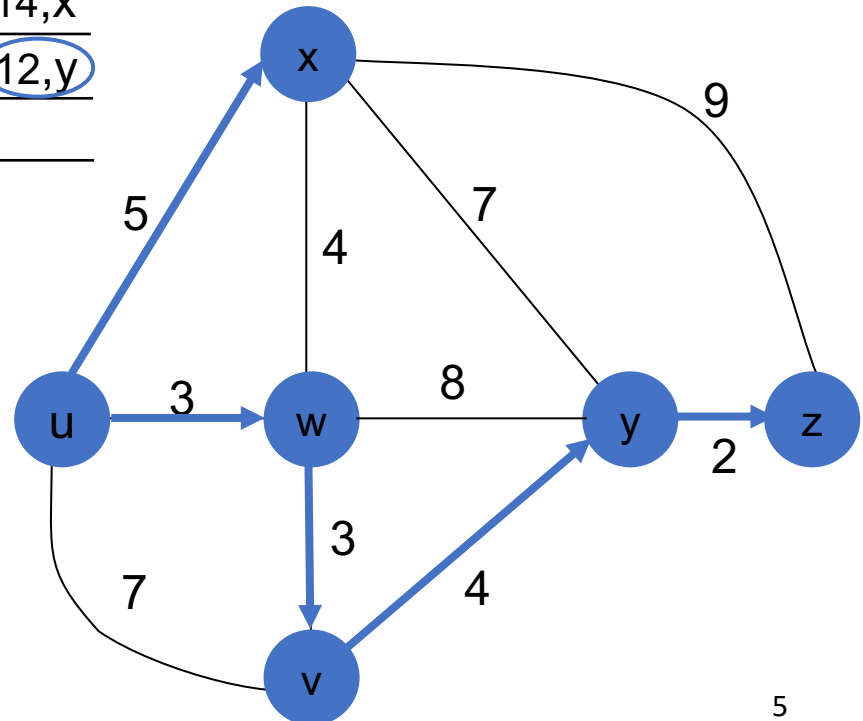
# Recap:

## Dijkstra's algorithm: Example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	$\infty$	$\infty$
1	uw	6,w		5,u	11,w	$\infty$
2	uwx	6,w			11,w	14,x
3	uwxv			10,v		14,x
4	uwxvy				12,y	
5	uwxvyz					

### Notes:

- Construct shortest path tree by tracing predecessor nodes
- Ties can exist (can be broken arbitrarily)



# From routing algorithm to protocol

- **Dijkstra's is a local computation!**
  - Computed by a node given complete network graph
- **Possibilities:**
  - **Option#1:** a separate machine runs the algorithm
  - **Option#2:** every router runs the algorithm
- **The Internet currently uses Option#2**

# Link-state routing

- **Every router knows its local “link state”**
  - Router u: “(u,v) with cost=2; (u,x) with cost=1”
- **Each router floods its local link state to all other routers in the network**
  - Does so periodically or when its link state changes
- **Every router learns the entire network graph**
  - Each runs Dijkstra’s Shortest-Path First (SPF) algorithm locally to compute forwarding table

# Flooding link state

- **Flooding**

- A node sends its link-state info out all of its links
- The next node forwards the info on all of its links except the one the information arrived at

- **When to initiate flooding?**

- Topology change (e.g., link/node failure/recovery)
- Configuration change (e.g., link cost change)
- Periodically
  - To refresh link-state information (soft states)
  - Typically (say) every 30 minutes



# Convergence

- **Why flood?**

- To get all the nodes in the network to **converge** to the new topology

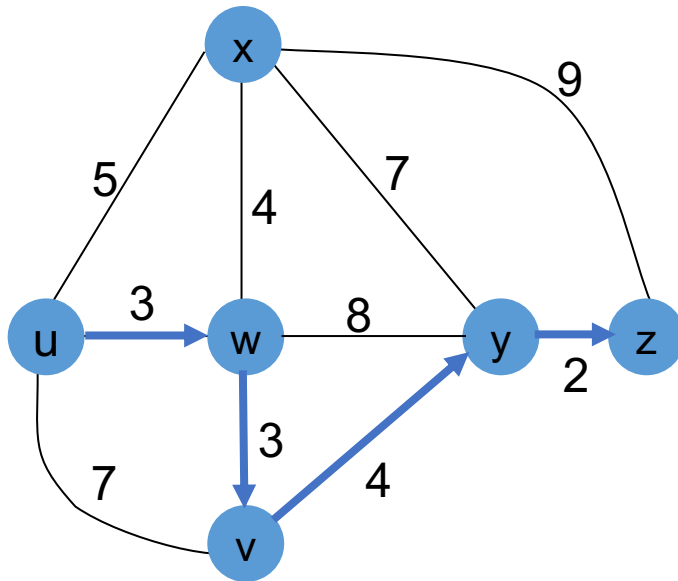
- **Upon convergence, all nodes will have **consistent routing information** and can **compute consistent forwarding**:**

- All nodes have the same link-state database
- All nodes forward packets on shortest paths
- The next router on the path forwards to the expected next hop

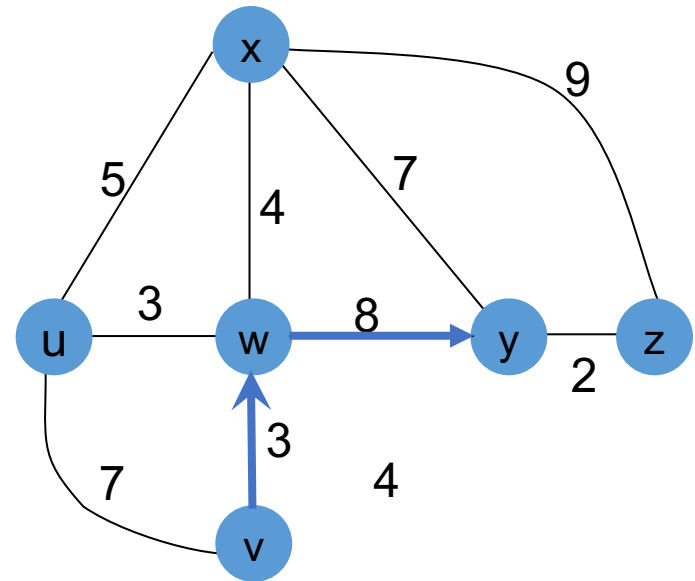
# Convergence delay

- **Time to achieve convergence**
- **Sources of convergence delay**
  - Time to detect failure
  - Time to flood link-state information
  - Time to re-compute forwarding tables
- **What happens if it takes too long to converge?**

# Loop from convergence delay



u and w think that the path to y goes through v



v thinks that the path to y goes through w

# Performance during convergence period

- Looping packets
- Lost packets **due to black holes**
- Out-of-order packets **reaching the destination**

# Link-state routing

- **Scalability?**

- $O(NE)$  messages
- $O(N^2)$  computation time
- $O(\text{Network diameter})$  convergence delay
- $O(N)$  entries in forwarding table

# Link-state routing protocols

- **OSPF: Open Shortest Path First**
- **IS-IS: Intermediate System to Intermediate System**
  - Similar to OSPF

# OSPF:

## Open Shortest-Path First

- **Open:** publicly available
- **Uses link-state algorithm**
  - Link-state packet dissemination
  - Topology map at each node
  - Route computation using Dijkstra's algorithm
- **Router floods OSPF link-state advertisements to all other routers in entire AS**
  - Carried in OSPF messages directly over IP (rather than TCP or UDP)
    - Requires reliable transmission

# Distance-vector protocol

- **Link-state routing protocol**

- Each node **broadcasts** its **local** information

- **Distance-vector routing protocol**

- The opposite (sort of)

- Each node **tells its neighbors** about its **global** view



# Bellman-Ford equation

- **Let**

➤  $d_x(y) :=$  cost of least-cost path from  $x$  to  $y$

- **Then**

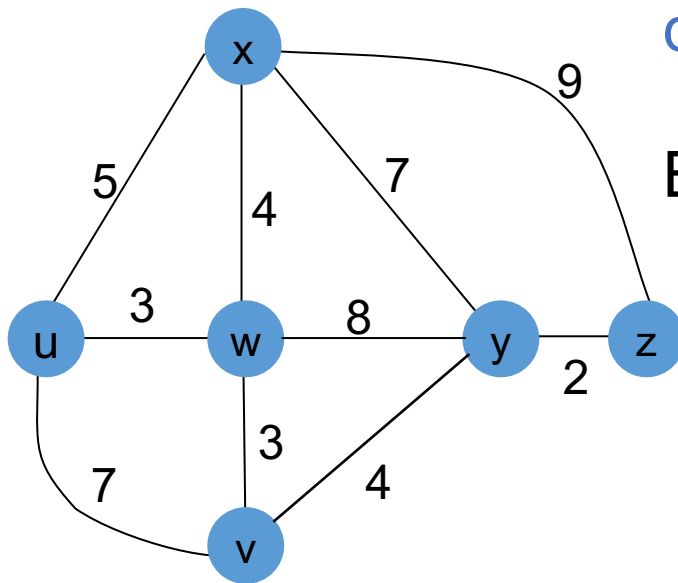
➤  $d_x(y) = \min_v \{ c(x, v) + d_v(y) \}$

$\min$  taken over all neighbors  $v$  of  $x$

cost to neighbor  $v$

cost from neighbor  $v$  to destination  $y$

# Bellman-Ford example



$$d_x(z) = 9, d_w(z) = 9, d_v(z) = 6$$

B-F equation says:

$$\begin{aligned} d_u(z) &= \min \{c(u,x) + d_x(z), \\ &\quad c(u,w) + d_w(z), \\ &\quad c(u,v) + d_v(z) \} \\ &= \min \{5 + 9, \\ &\quad 3 + 9, \\ &\quad 7 + 6\} = 12 \end{aligned}$$

Neighbor achieving the minimum (w) is next hop in shortest path, used in forwarding table

# Distance vector algorithm

- $D_x(y)$  is the estimate of least cost from  $x$  to  $y$ 
  - $x$  maintains its own distance vector  $D_x = [D_x(y): y \in N]$
- **Node  $x$ :**
  - Knows cost to each neighbor  $v$ :  $c(x,v)$
  - Maintains its neighbors' distance vectors
    - For each neighbor  $v$ ,  $x$  has  $D_v = [D_v(y): y \in N]$

# Distance vector algorithm

- From time-to-time, each node sends its own distance vector estimate to neighbors
- When  $x$  receives new DV estimate from neighbor, it updates its own DV using B-F equation
  - $D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\}$  for each node  $y \in N$
- Eventually, the estimate  $D_x(y)$  may converge to the actual least cost  $d_x(y)$

# Distance vector algorithm

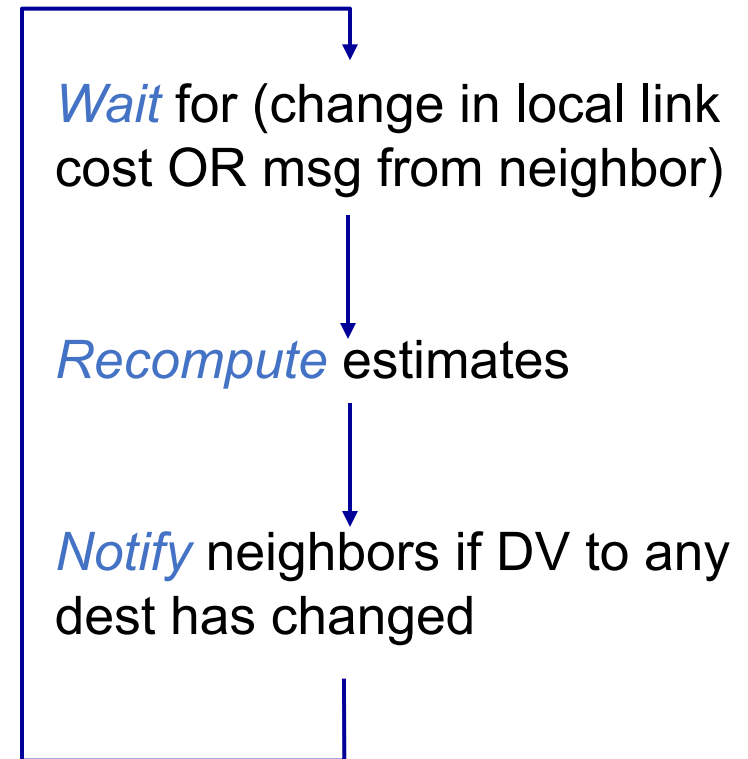
- **Iterative, asynchronous**

- Local iterations caused by
  - Local link cost change
  - DV update message from neighbor

- **Distributed**

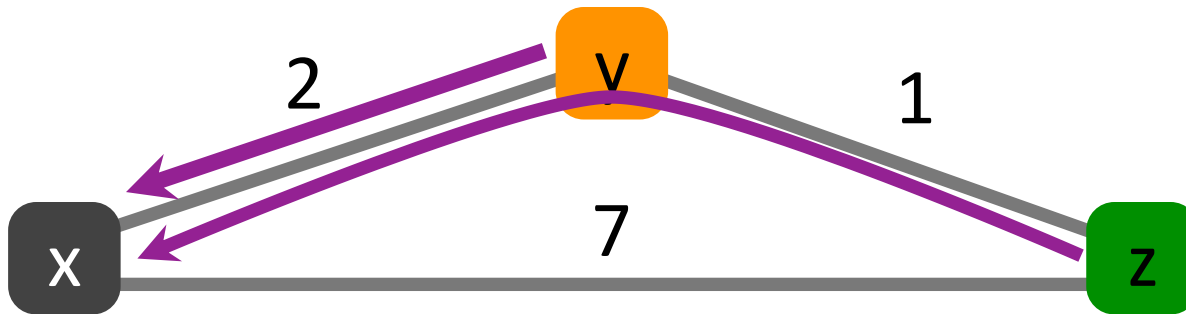
- Each node notifies neighbors only when its DV changes
  - Neighbors then notify their neighbors if necessary

@each node:



# Example

	x	y	z
y	2	0	1
z	3	1	0

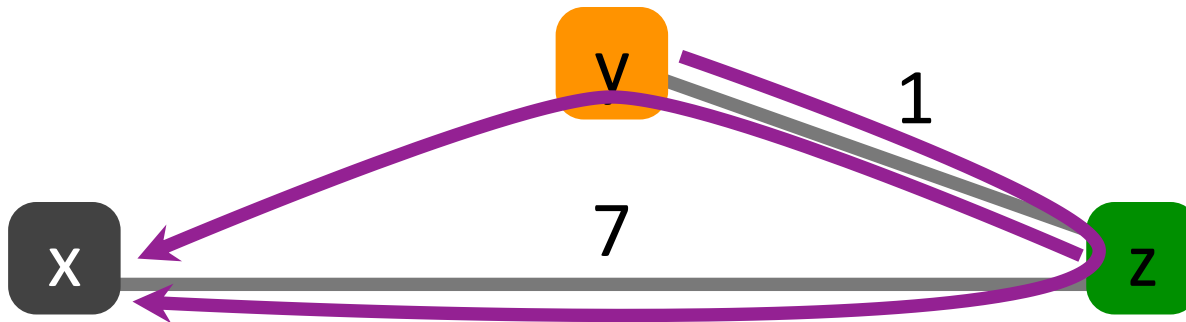


	x	y	z
y	2	0	1
z	3	1	0

# Example

	x	y	z
y	4	0	1
z	3	1	0

routing loop!

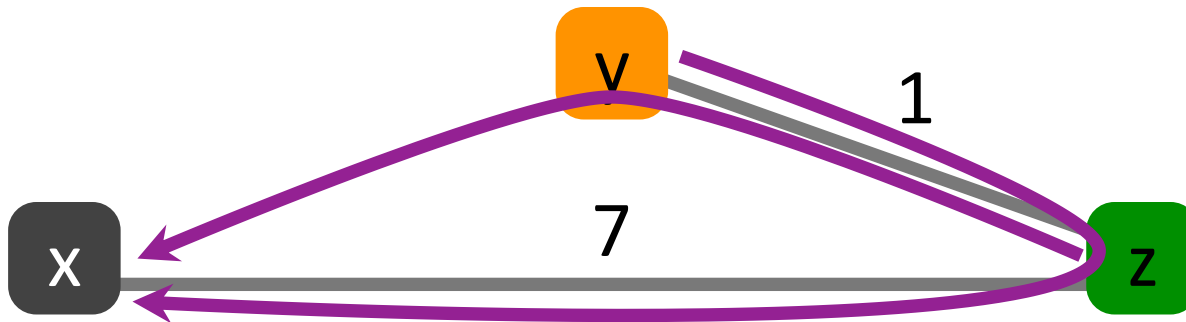


	x	y	z
y	2	0	1
z	3	1	0

# Example

	x	y	z
y	4	0	1
z	3	1	0

routing loop!



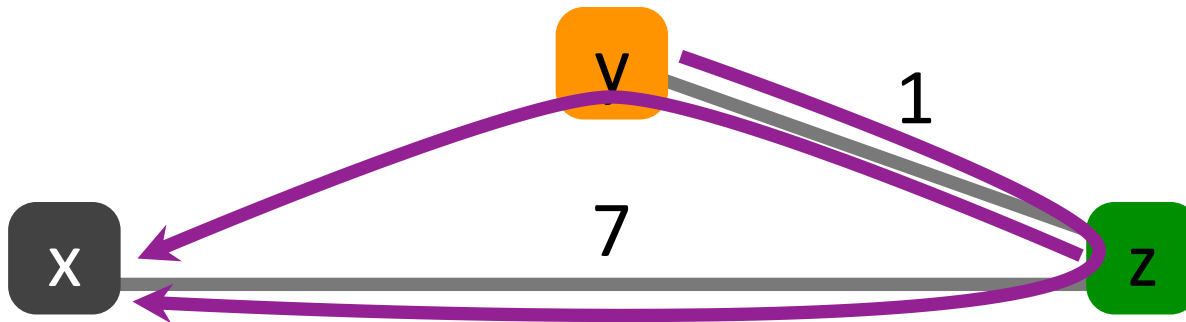
	x	y	z
y	4	0	1
z	3	1	0



# Example

	x	y	z
y	4	0	1
z	3	1	0

routing loop!

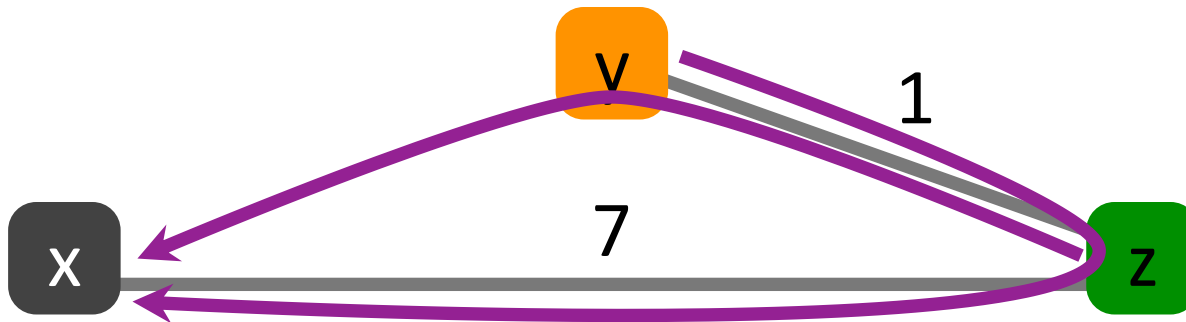


	x	y	z
y	4	0	1
z	5	1	0

# Example

	x	y	z
y	4	0	1
z	5	1	0

routing loop!

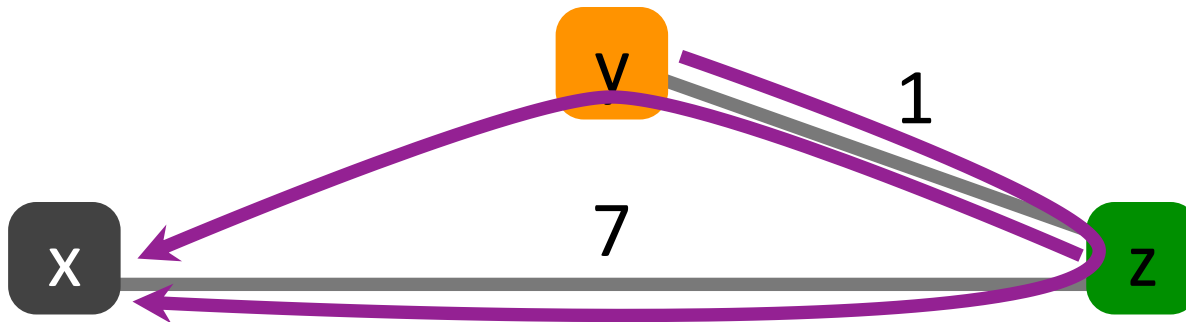


	x	y	z
y	4	0	1
z	5	1	0

# Example

	x	y	z
y	6	0	1
z	5	1	0

routing loop!

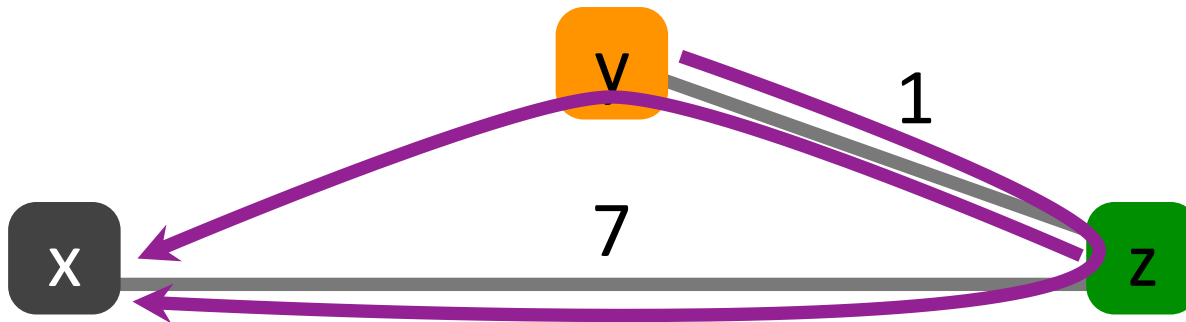


	x	y	z
y	4	0	1
z	5	1	0

# Example

	x	y	z
y	6	0	1
z	5	1	0

routing loop!

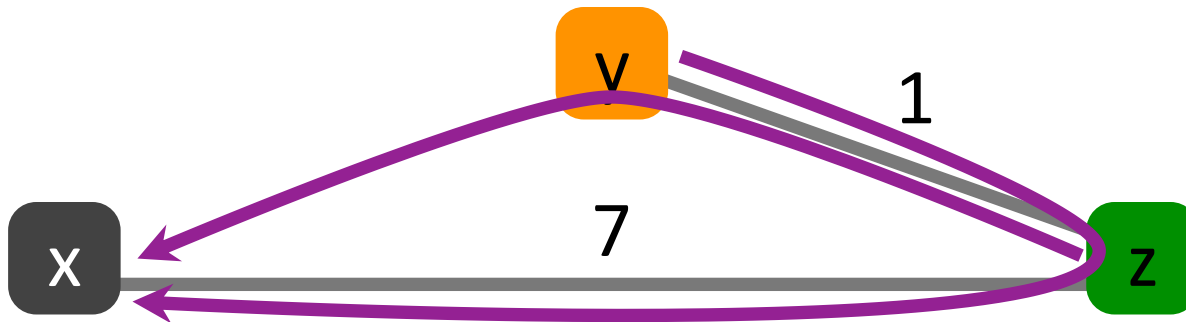


	x	y	z
y	6	0	1
z	5	1	0

# Example

	x	y	z
y	6	0	1
z	5	1	0

routing loop!

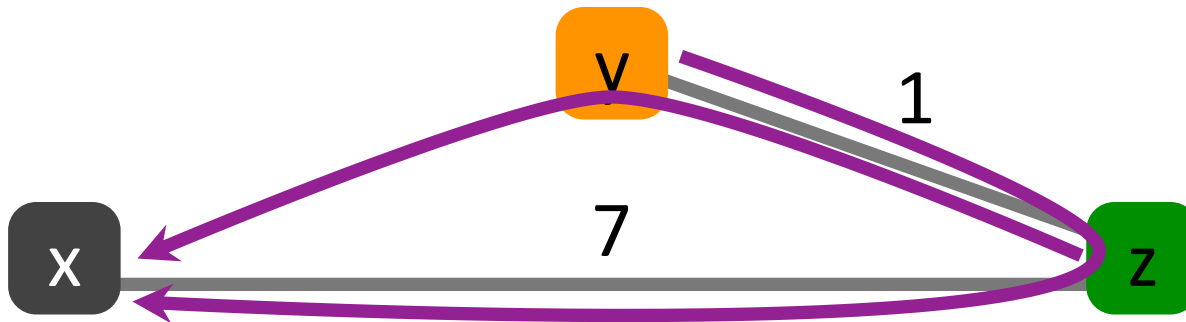


	x	y	z
y	6	0	1
z	7	1	0

# Example

	x	y	z
y	6	0	1
z	7	1	0

routing loop!

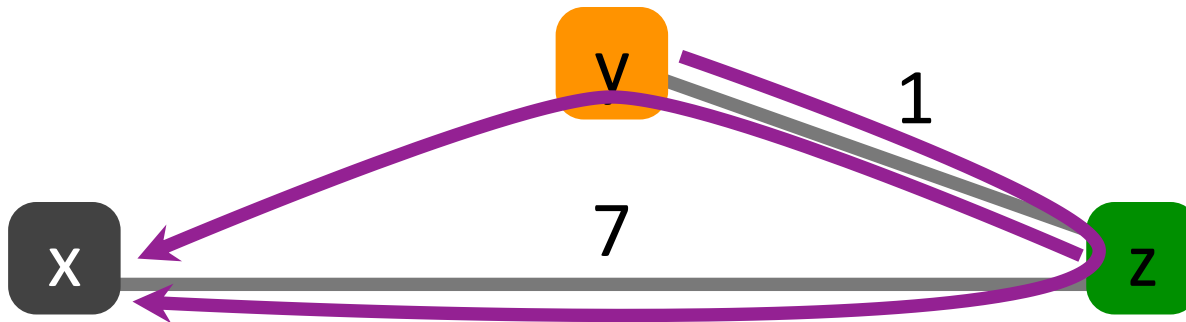


	x	y	z
y	6	0	1
z	7	1	0

# Example

	x	y	z
y	8	0	1
z	7	1	0

routing loop!

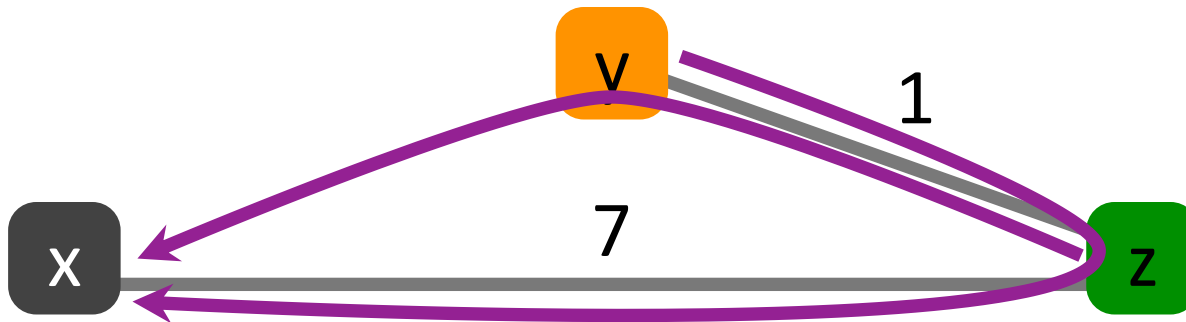


	x	y	z
y	6	0	1
z	7	1	0

# Example

	x	y	z
y	8	0	1
z	7	1	0

routing loop!



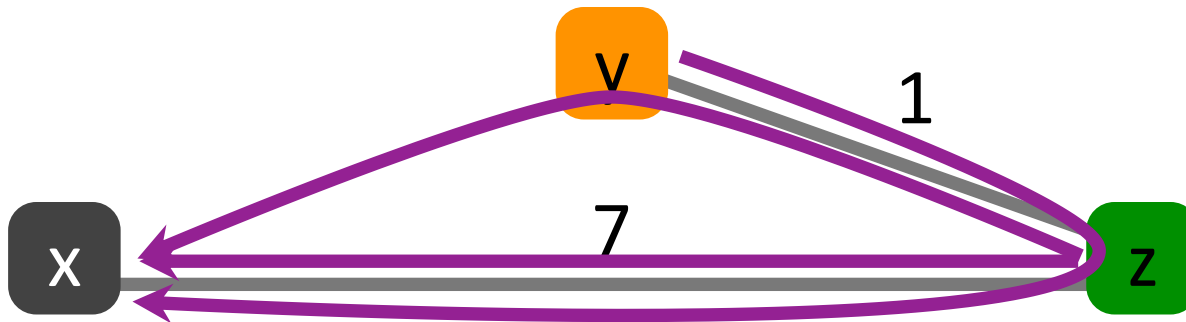
	x	y	z
y	8	0	1
z	7	1	0



# Example

	x	y	z
y	8	0	1
z	7	1	0

routing loop!



Count-to-infinity scenario

	x	y	z
y	8	0	1
z	7	1	0

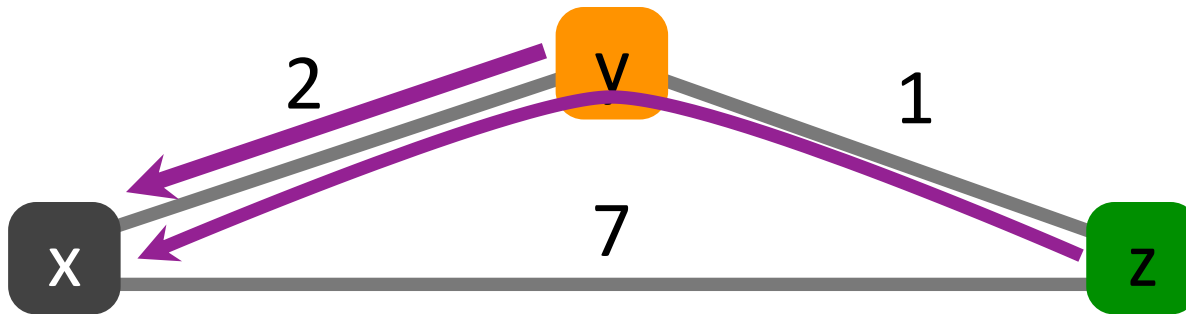
# Problems with Bellman-Ford

- **Routing loops**
  - z routes through y, y routes through x
  - y loses connectivity to x
  - y decides to route through z
- **Can take a very long time to resolve**
  - Count-to-infinity scenario

# Poisoned reverse

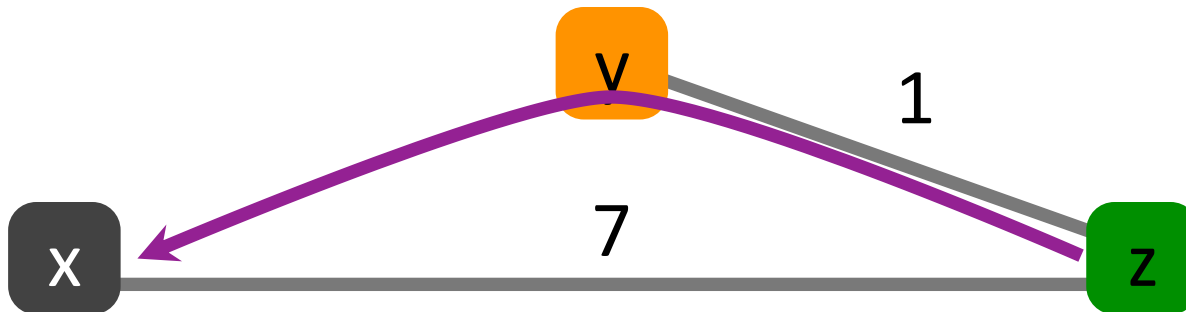
- **One heuristic to avoid count-to-infinity**
  - If z routes to x through y,
    - z advertises to y that its cost to x is infinite
  - y never decides to route to x through z

	x	y	z
y	2	0	1
z	$\infty$	1	0



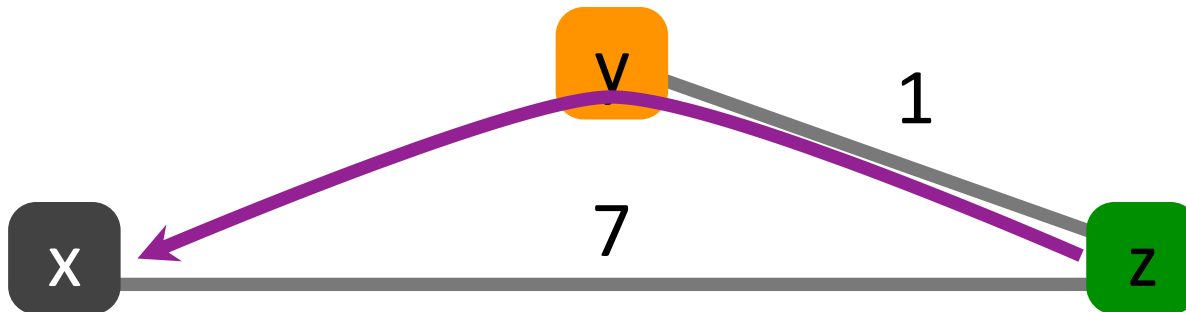
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y	$\infty$	0	1
z	$\infty$	1	0



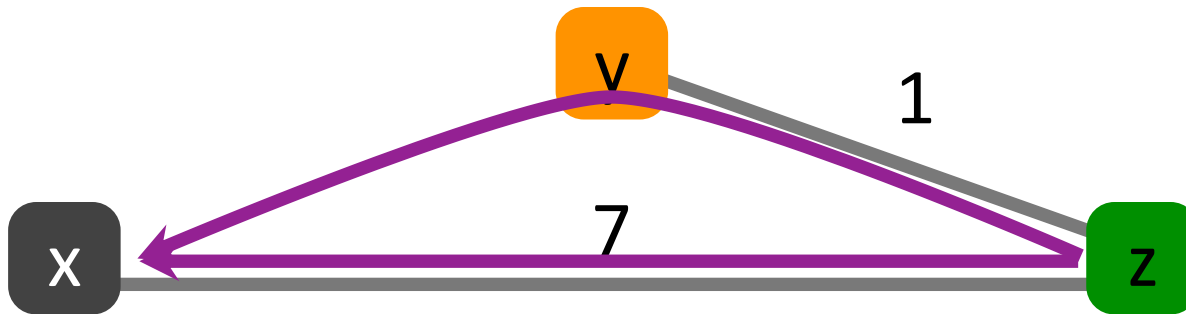
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y	$\infty$	0	1
z	$\infty$	1	0



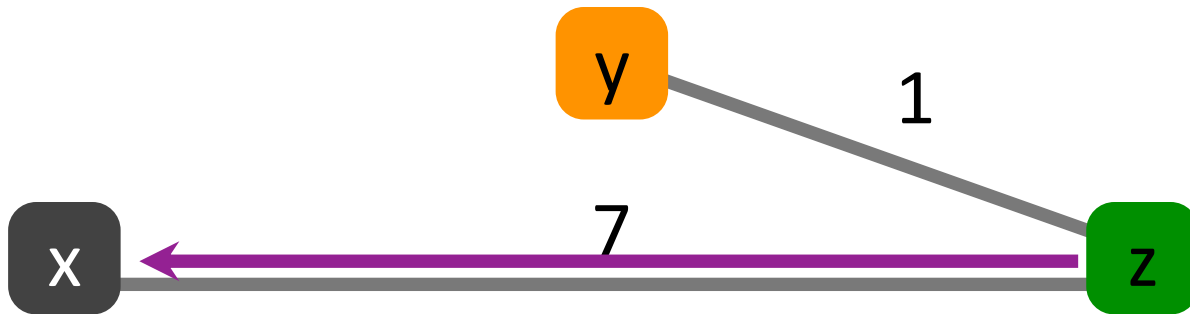
	x	y	z
y	$\infty$	0	1
z	3	1	0

	x	y	z
y	$\infty$	0	1
z	$\infty$	1	0



	x	y	z
y	$\infty$	0	1
z	7	1	0

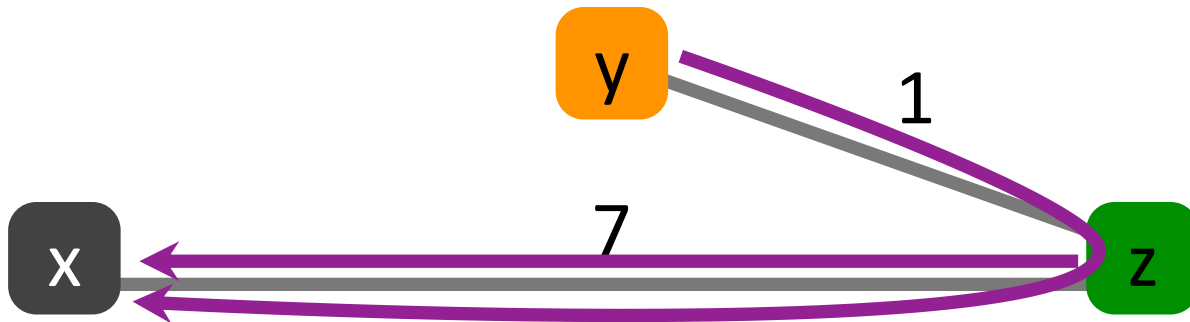
	x	y	z
y	$\infty$	0	1
z	7	1	0



	x	y	z
y	$\infty$	0	1
z	7	1	0



	x	y	z
y	8	0	1
z	7	1	0



	x	y	z
y	$\infty$	0	1
z	7	1	0

# Poisoned reverse

- **One heuristic to avoid count-to-infinity**
  - If z routes to x through y,
    - z advertises to y that its cost to x is infinite
  - y never decides to route to x through z
- **Not guaranteed (why?)**
- **Loop-free routing examples include**
  - Path vector
  - Source tracing

# Distance-vector routing

- **Scalability?**

- Requires fewer messages than Link-State
- $O(N)$  update time on arrival of a new DV from neighbor
- $O(\text{network diameter})$  convergence time
- $O(N)$  entries in forwarding table

- **RIP is a protocol that implements DV (IETF RFC 2080)**

# Comparison of LS and DV routing

## Messaging complexity

- **LS:** with  $N$  nodes,  $E$  links,  $O(NE)$  messages sent
- **DV:** exchange between neighbors only

## Speed of convergence

- **LS:** relatively fast
- **DV:** convergence time varies
  - Count-to-infinity problem

## Robustness: what happens if router malfunctions?

- **LS:**
  - Node can advertise incorrect **link** cost
  - Each node computes its *own* table
- **DV:**
  - Node can advertise incorrect **path** cost
  - Each node's table used by others (error propagates)

# Similarities between LS and DV routing

- **Both are shortest-path based routing**
  - Minimizing cost metric (link weights) a common optimization goal
    - Routers share a common view as to what makes a path “good” and how to measure the “goodness” of a path
- **Due to shared goal, commonly used inside an organization**
  - RIP and OSPF are mostly used for **intra**-domain routing

# Group Discussion

- **Topic: poisoned reverse**

- Poisoned reverse is a heuristic to avoid count-to-infinity. How does it work? However, poisoned reverse cannot guarantee to avoid count-to-infinity. Can you give an example?

- **Discuss in groups, and each group chooses a leader to summarize the discussion**

- **Everyone should speak.**

- **Turn on your audio and video. Do not mute.**

# Assignment 3

- **Intra-domain routing algorithms**

- Design and implement simple versions of link state and distance vector protocols by yourself
- Hands-on experiences on routing protocols: real-world protocols in Cisco routers are just more complicated than the ones you designed and implemented!
- Use NetworkX to compute shortest path if you do not want to implement it yourself

- **Implement either link-state or distance-vector**

- Implementing both gives you a bonus of 2 points

# Summary

- **Intra-AS routing**
  - Link-state routing
  - Distance-vector routing
- **Next class: Midterm re-cap**
- **Next Tuesday: Inter-AS routing**



Thanks!  
Q&A