CS4450

Computer Networks:
Architecture and Protocols

Lecture 12
Distance Vector Routing

Spring 2018
Rachit Agarwal

Announcements

- Problem Set 3 will be up soon (definitely by Saturday)

- During my first lecture, I promised you:
  - **I care about you(r learning)!**
  - **If you stick to the contract, I'll bring my A game in every lecture!**

- **You have been great so far!**

- I will stick to my promise
  - We are almost half-way through
  - If you think I am not bringing my A-game in the course
    - I want to know and improve!!!
  - **Please fill out the mid-term evaluation (this weekend)**
    - **Completely anonymized; only for my eyes; max 5 min**

Goals for Today's Lecture

- **Distance Vector (Local view)**

- 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: Four flavors of protocols

- **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

- **Today: Distributed route computation**

Recap: Spanning Tree Protocol

- 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
- Switches update their view
  - Upon receiving message (Y,d,Y) from Z, check Y’s id
  - If Y’s id < current root: set root = Y
- Switches compute their distance from the root
  - Add 1 to the shortest distance received from a neighbor
- If root changed OR shortest distance to the root changed, send all neighbors updated message (Y,d+1,X)

Distributed Route Computation

Distributed Computation of Routes

- Each node computing the outgoing links 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

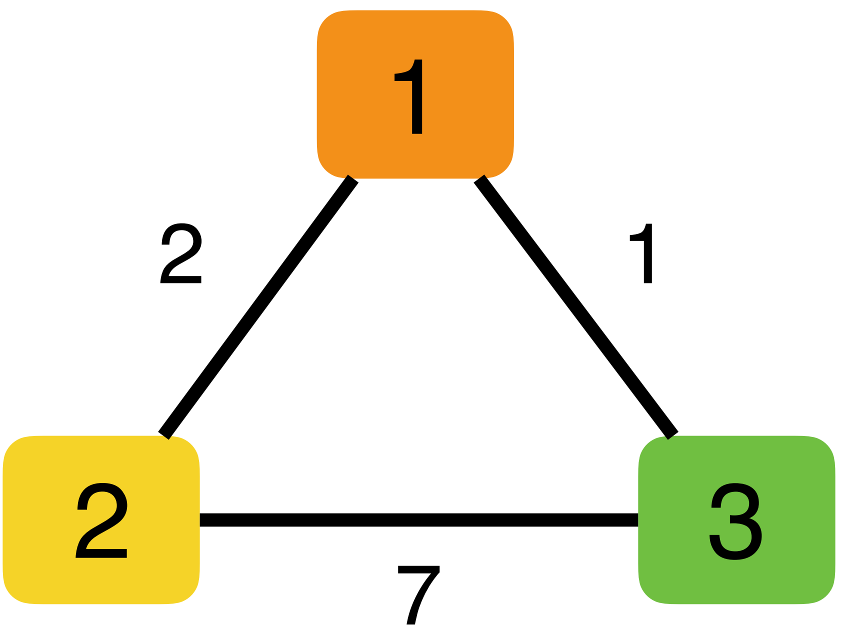
- Switches update their view
  - Upon receiving message (Y,d,Z) from Z~~, check Y's id~~
  - ~~If Y's id < current root: set root destination = Y~~

- Switches compute their shortest distance from the ~~root~~ destination
  - If current_distance_to_Y > d + cost of link to X:
    - update current_distance_to_Y = d

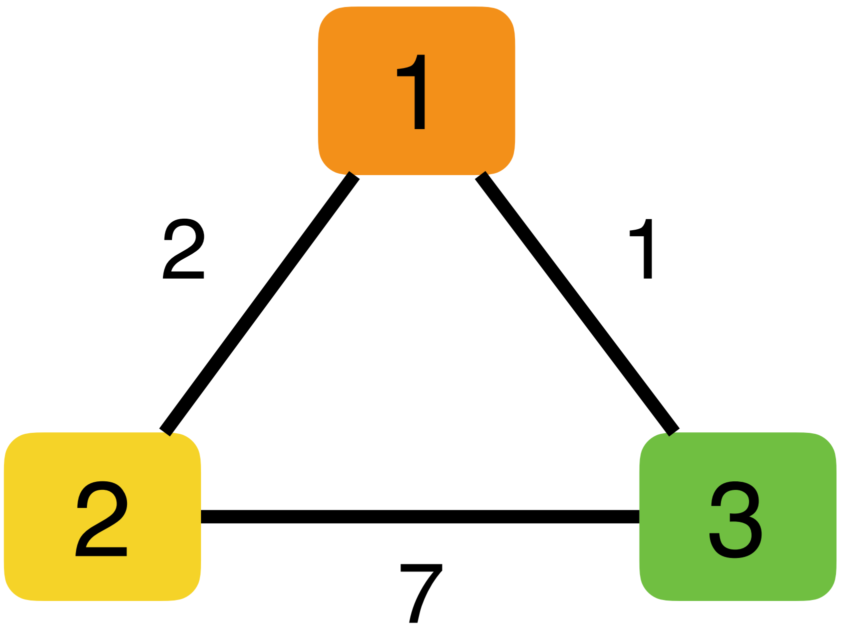
- If ~~root changed OR~~ shortest distance to the ~~root~~ destination changed, send all neighbors updated message (Y,d+c,X)

Group Exercise:

Lets run the Protocol on this example

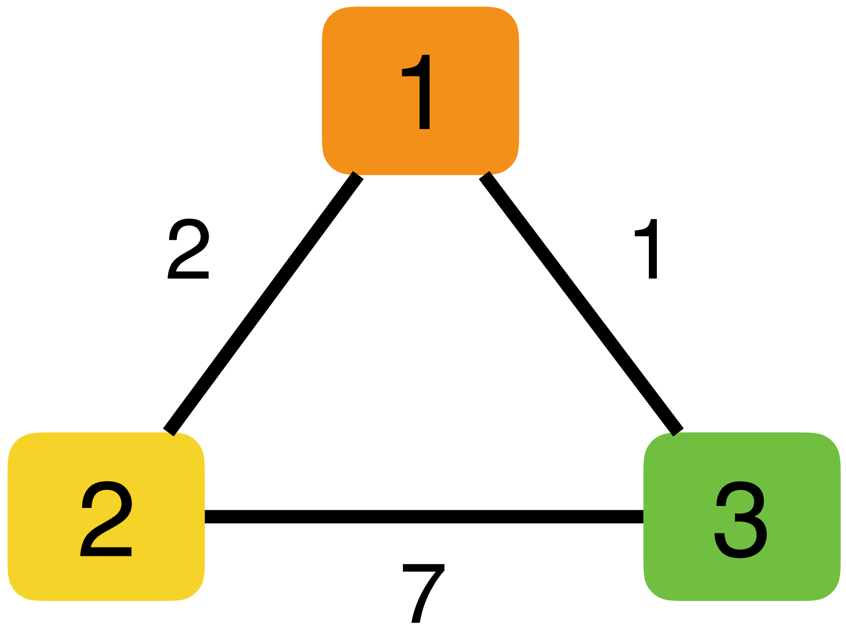


Round 1



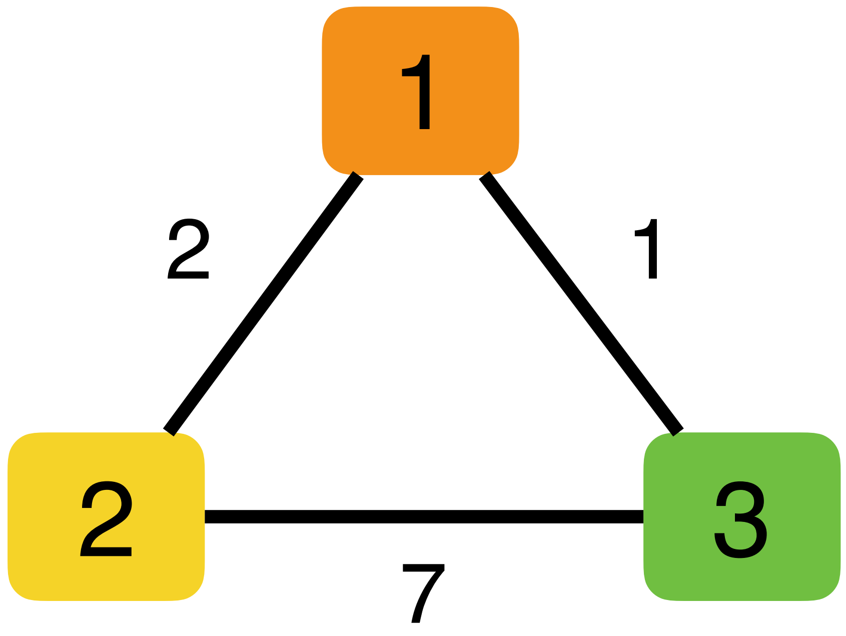
	Receive	Send
1		(1, 0, 1)
2		(2, 0, 2)
3		(3, 0, 3)

Round 2



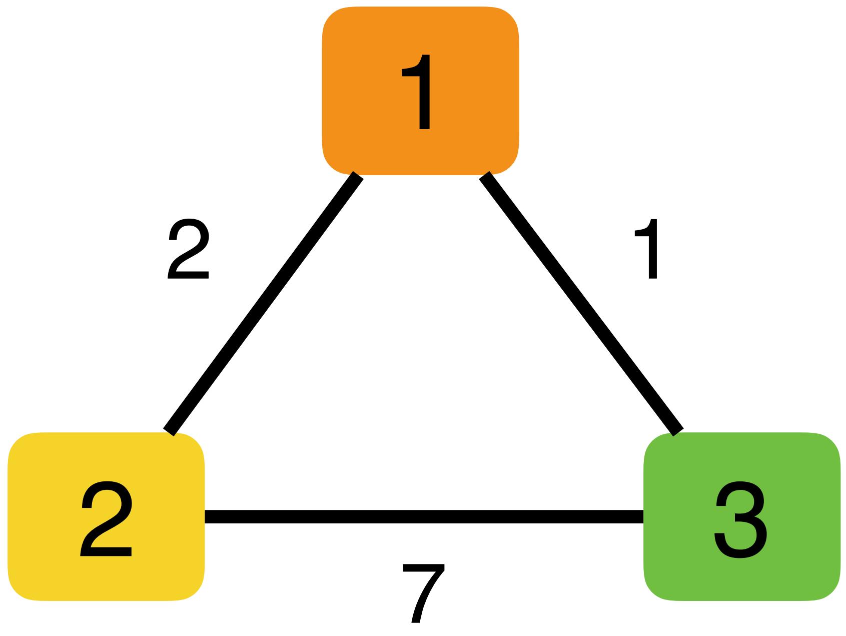
	Receive	Send
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)
2 **(2, 0, 2)**	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)
3 **(3, 0, 3)**	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)

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

- Switches update their view
  - Upon receiving message (Y,d,Z) from Z~~, check Y's id~~
  - ~~If Y's id < current root: set root destination = Y~~

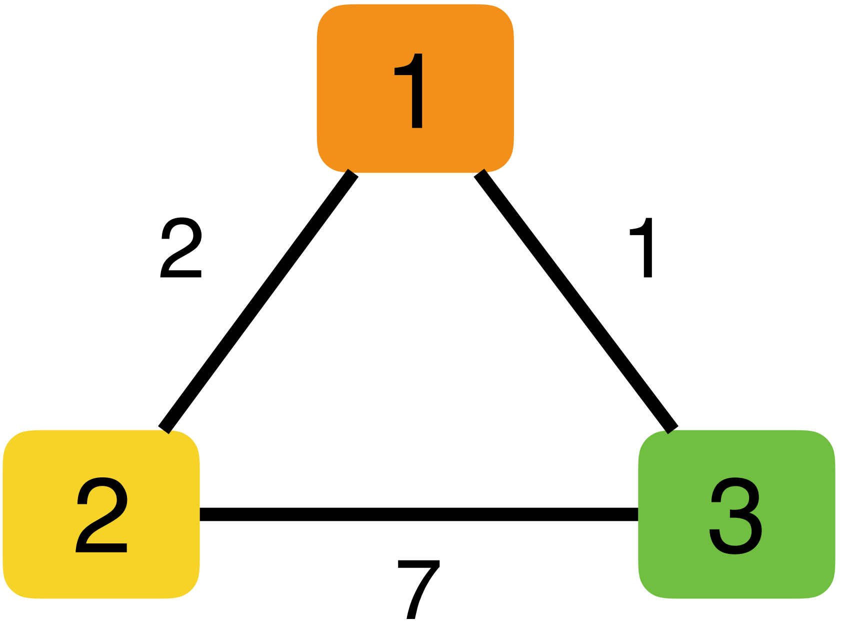
- Switches compute their shortest distance from the ~~root~~ destination
  - If current_distance_to_Y > d + cost of link to X:
    - update current_distance_to_Y = d
    - **update next_hop_to_destination = X**

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

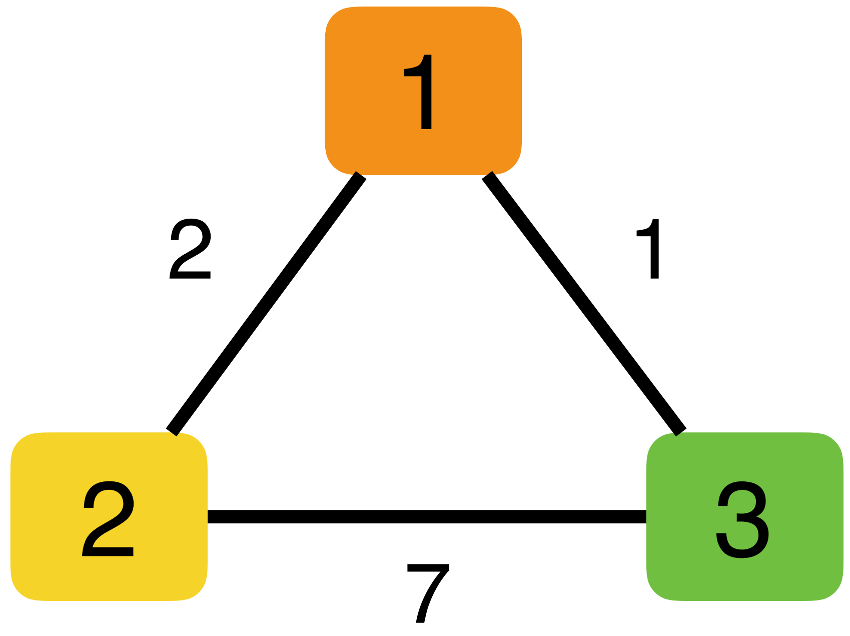
Group Exercise:

Lets run the Protocol on this example

(this time with next-hops)

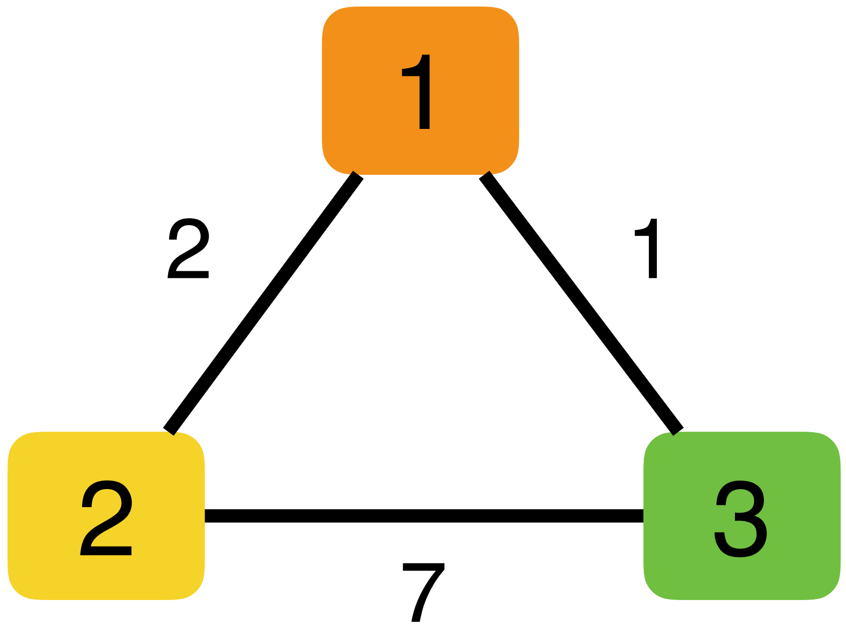


Round 1



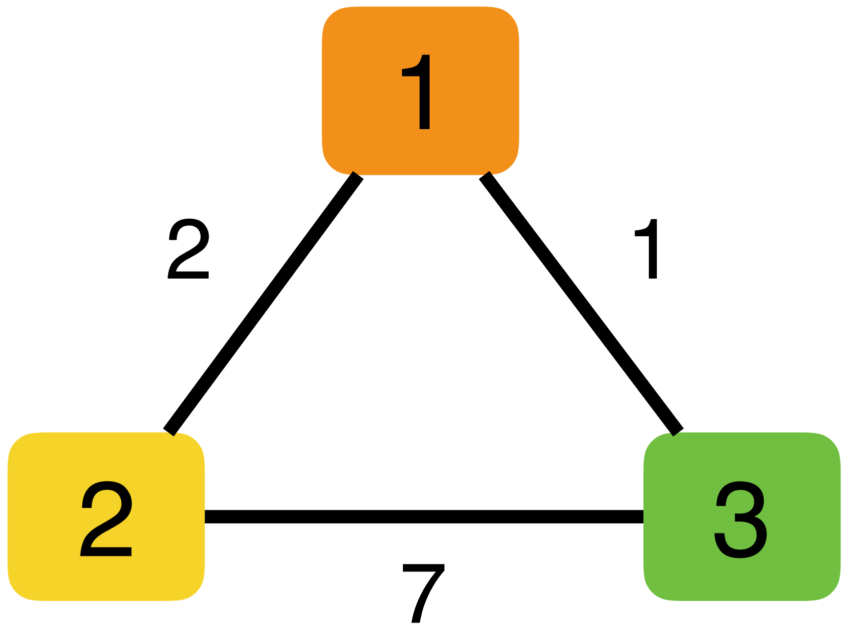
	Receive	Send	Next-hops
1		(1, 0, 1)	[-]
2		(2, 0, 2)	[-]
3		(3, 0, 3)	[-]

Round 2



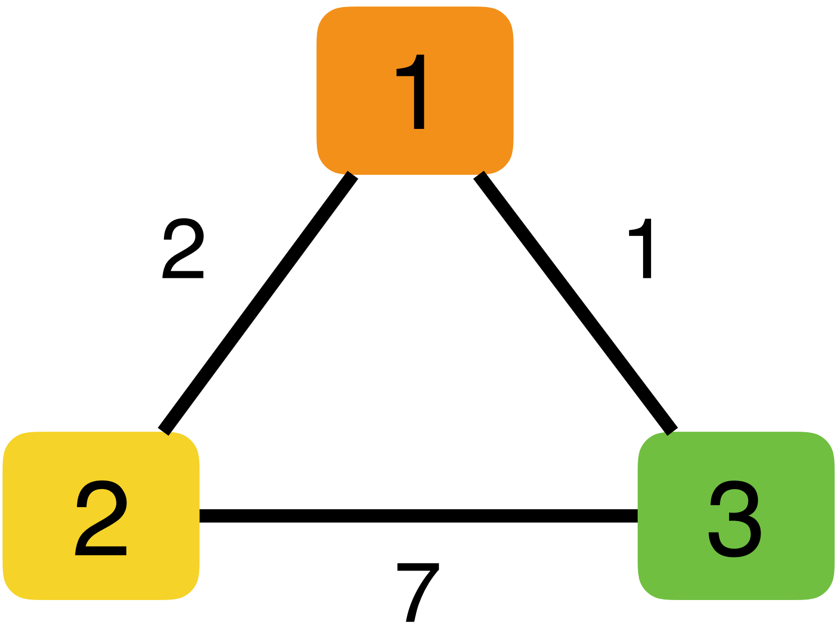
	Receive	Send	Next-hops
1 **(1, 0, 1)**	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 **(2, 0, 2)**	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)	[1, -, 3]
3 **(3, 0, 3)**	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)	[1, 2, -]

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, -]

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

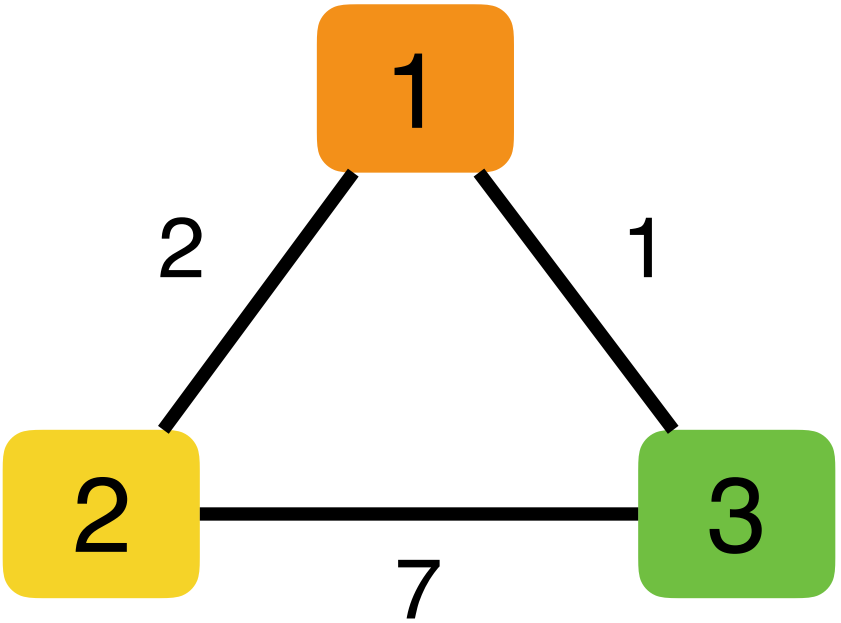
Two Aspects to This Approach

- **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?

Group Exercise:

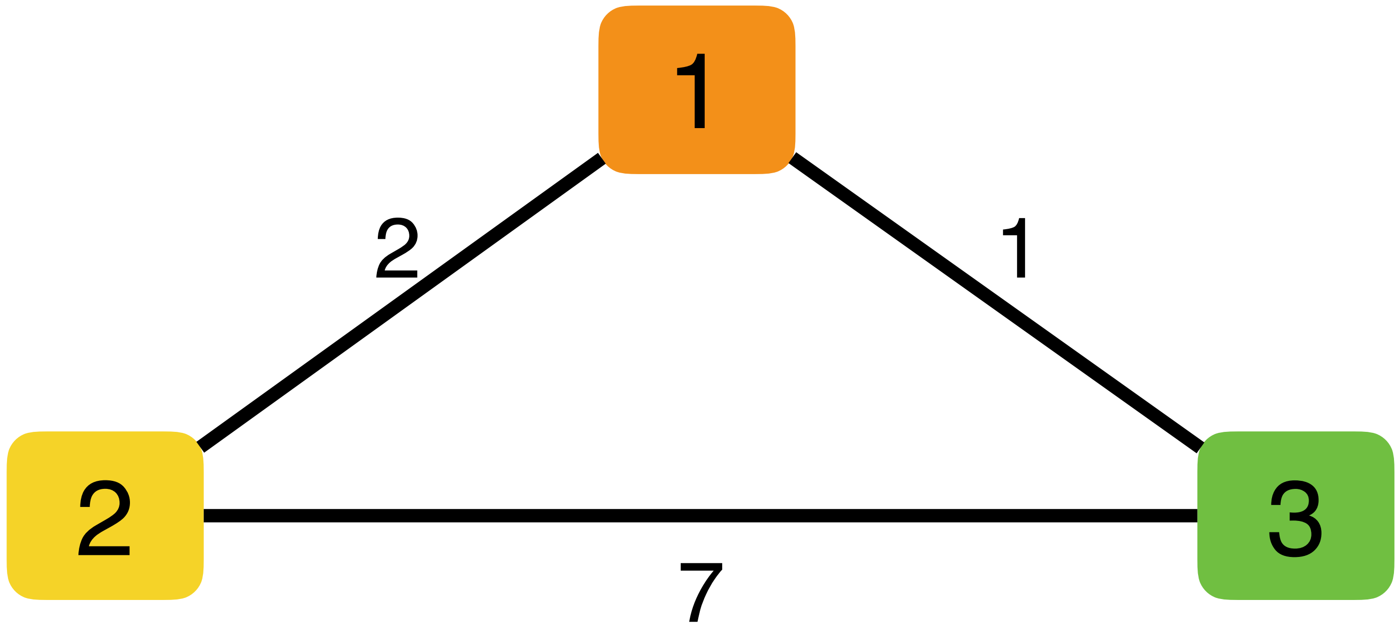
Lets run the Protocol again on this example

(this time with distance vectors)



Round 1

	distance	next-hop
1	0	-
2	infinity	
3	infinity	

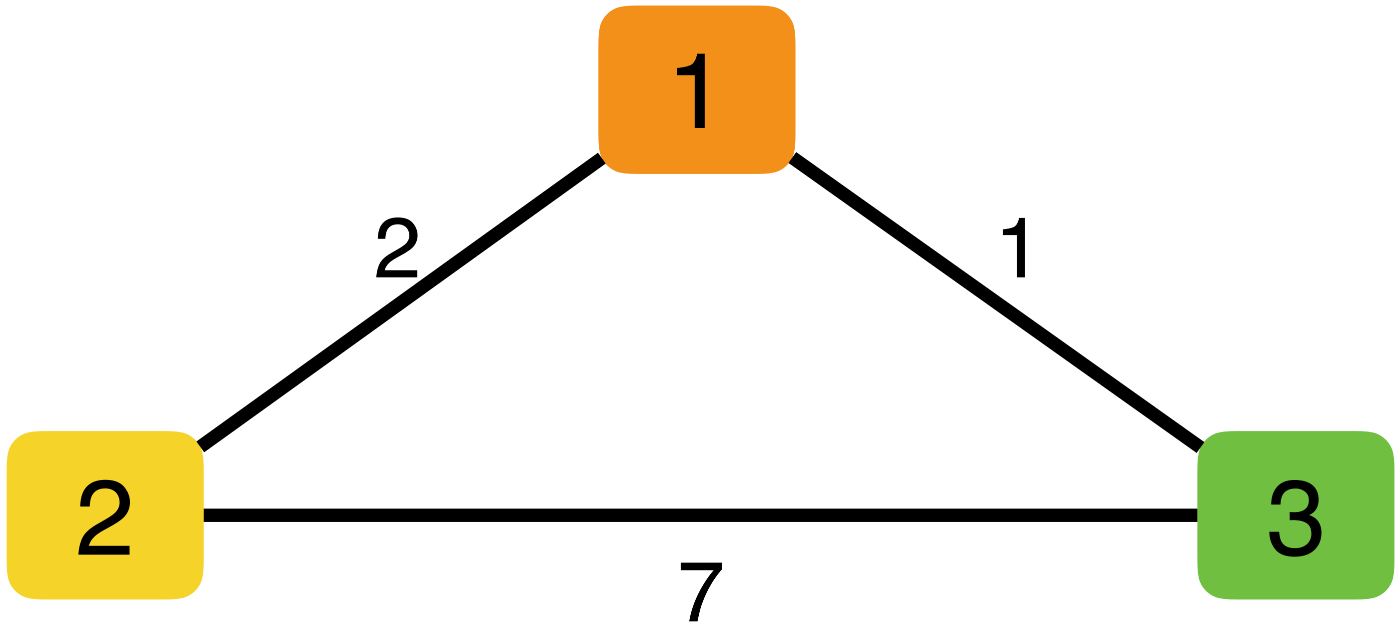


	distance	next-hop
1	infinity	
2	0	-
3	infinity	

	distance	next-hop
1	infinity	
2	infinity	
3	0	-

Round 2

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

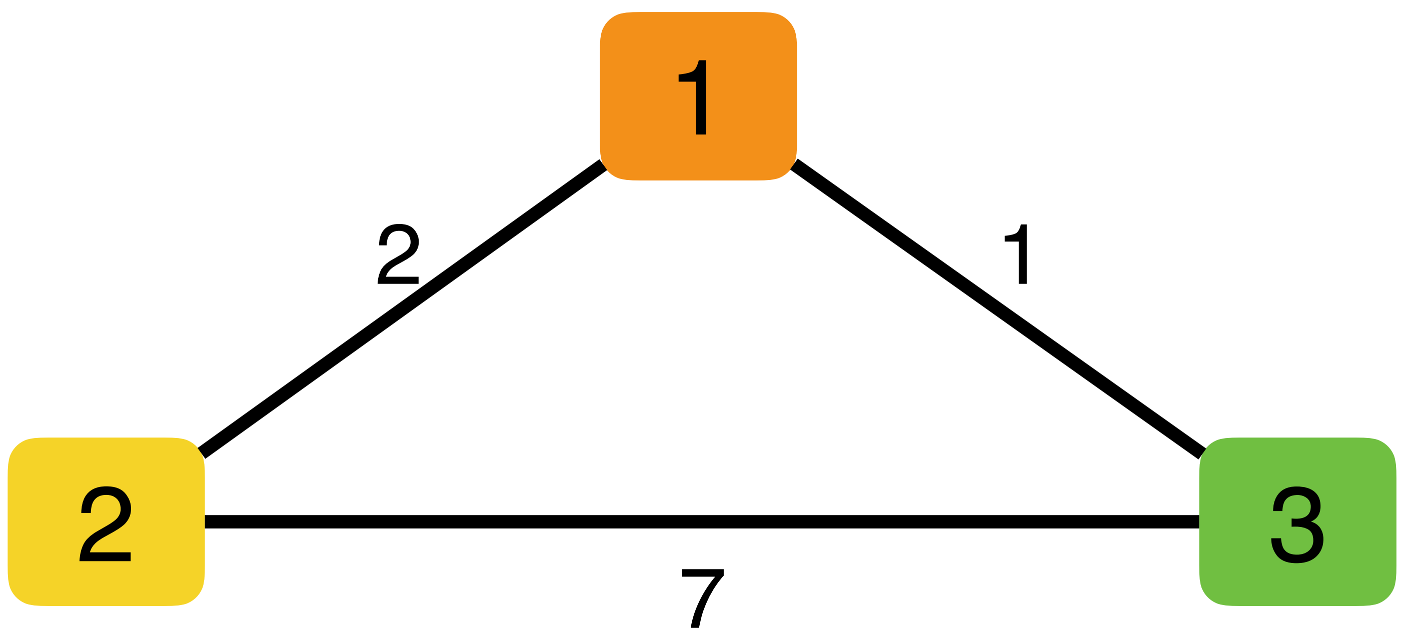


	distance	next-hop
1	2	1
2	0	-
3	7	3

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

Round 3

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

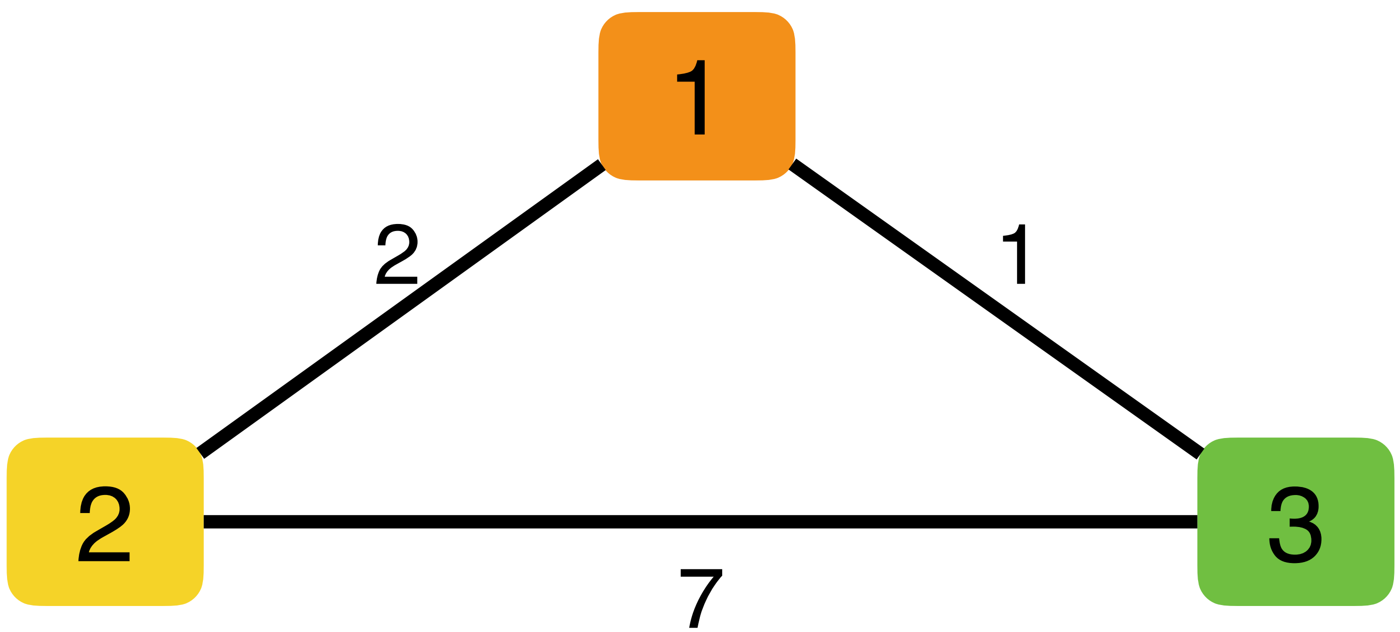


	distance	next-hop
1	2	1
2	0	-
3	3	1

	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….

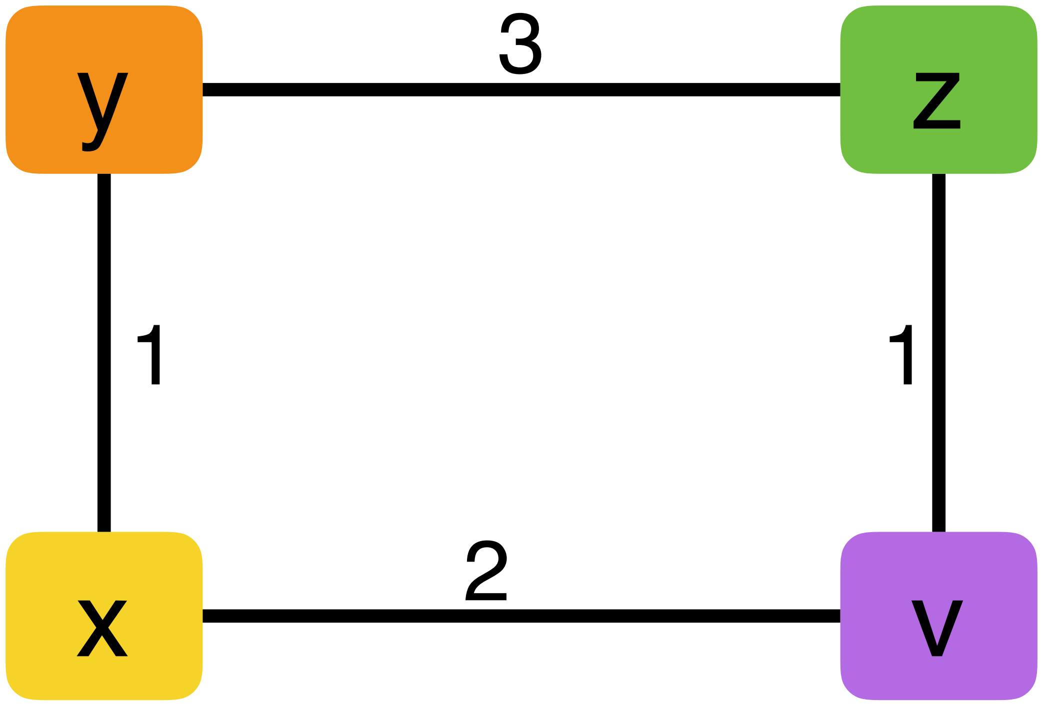
A More Complicated Case

- The three node network:
  - Everyone was neighbors with everyone else
- What happens in a larger network?
- Lets see ….

Group Exercise:

Lets run the Protocol on this example

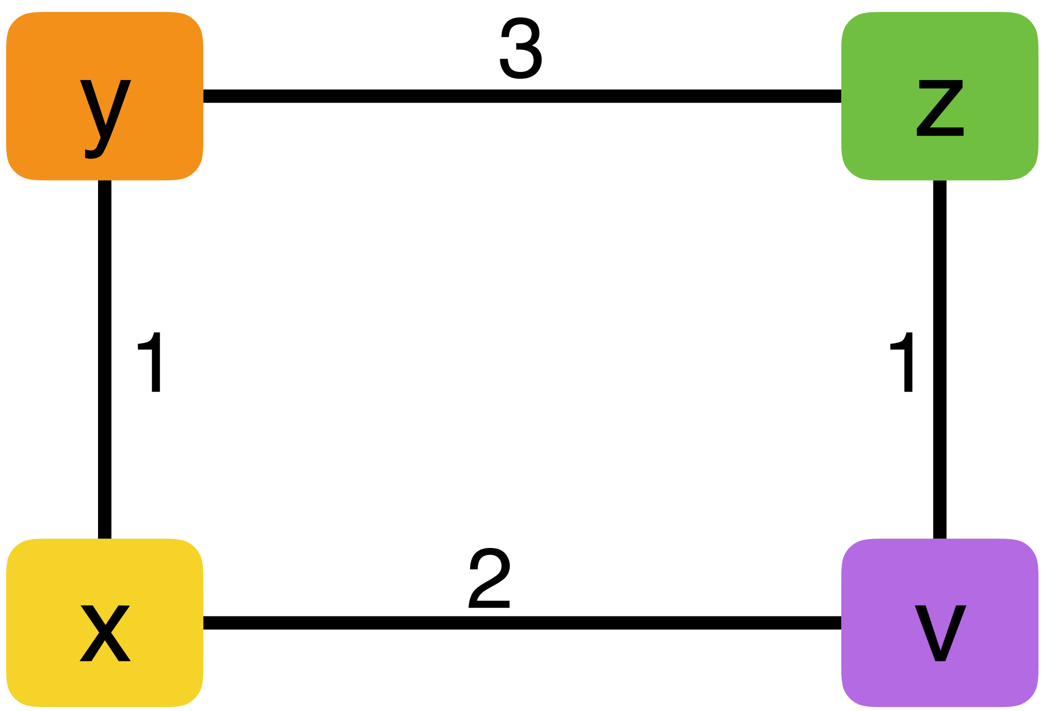
(this time with next-hops)



Round 1

	distance	NH
x	infinity	
y	0	-
z	infinity	
v	infinity	

	distance	NH
x	infinity	
y	infinity	
z	0	-
v	infinity	



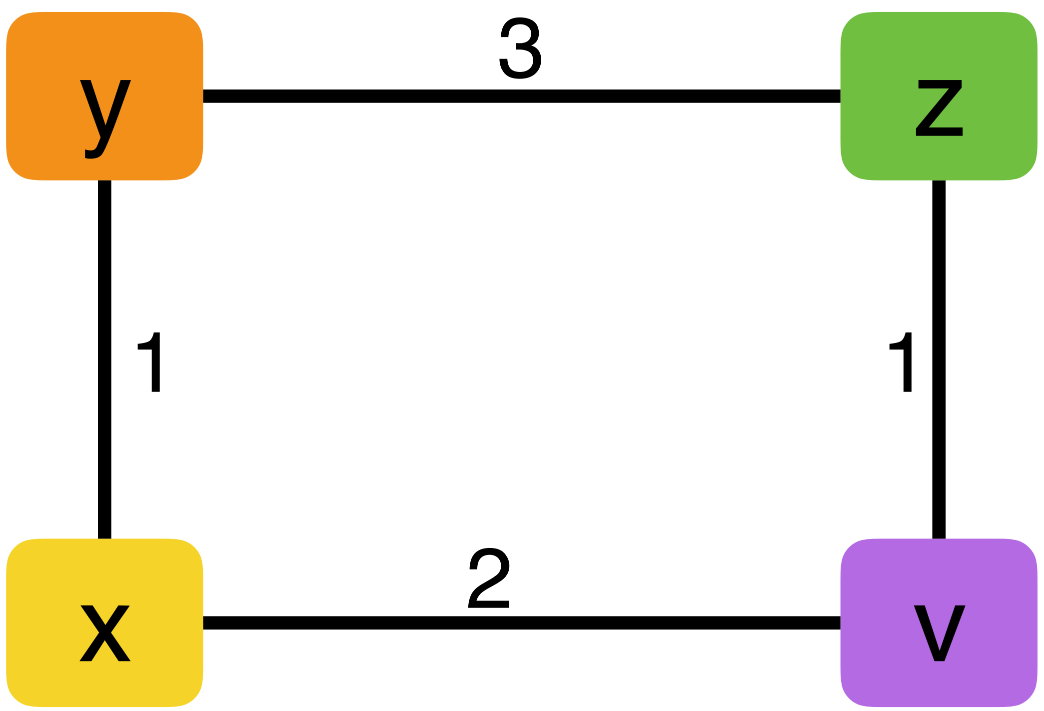
	distance	NH
x	0	-
y	infinity	
z	infinity	
v	infinity	

	distance	NH
x	infinity	
y	infinity	
z	infinity	
v	0	-

Round 2

	distance	NH
x	1	x
y	0	-
z	3	z
v	infinity	

	distance	NH
x	infinity	
y	3	y
z	0	-
v	1	v



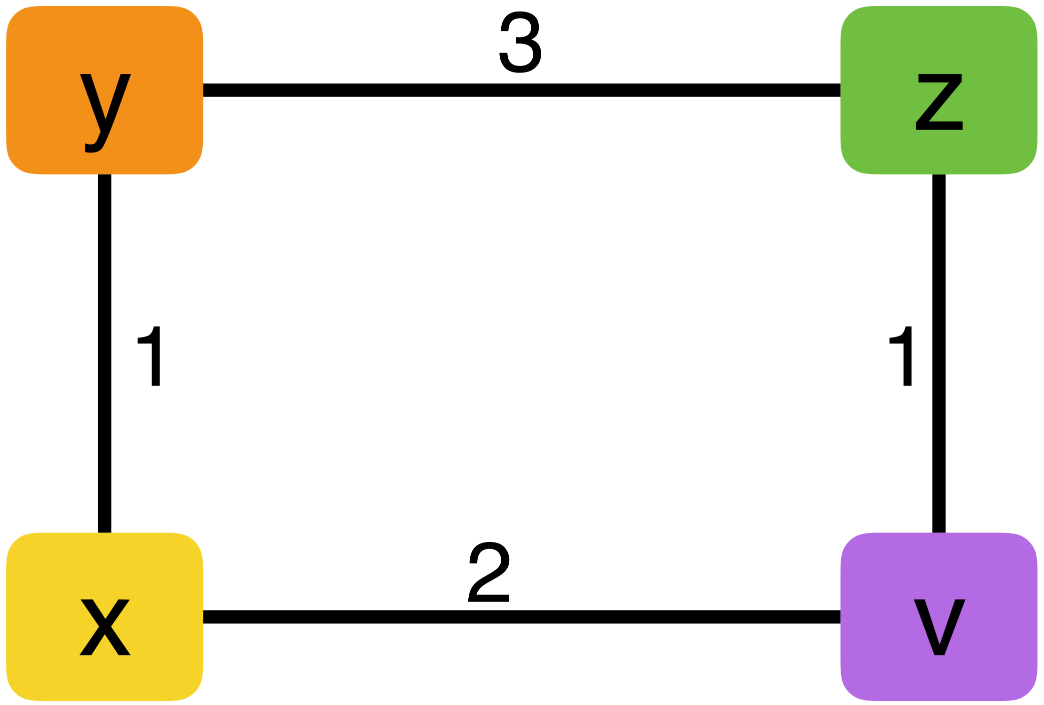
	distance	NH
x	0	-
y	1	y
z	infinity	
v	2	v

	distance	NH
x	2	x
y	infinity	
z	1	z
v	0	-

Round 3

	distance	NH
x	1	x
y	0	-
z	3	z
v	**3**	**x**

	distance	NH
x	**3**	**v**
y	3	y
z	0	-
v	1	v



	distance	NH
x	0	-
y	1	y
z	**3**	**v**
v	2	v

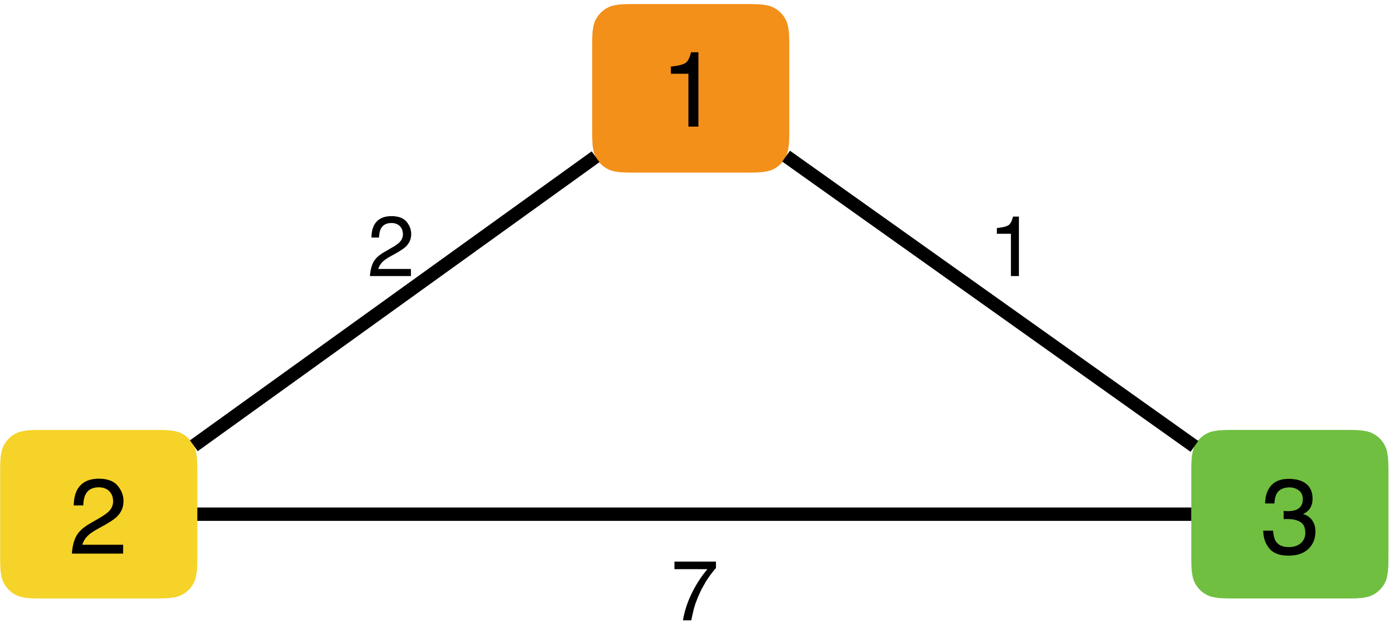
	distance	NH
x	2	x
y	**3**	**x**
z	1	z
v	0	-

Other Aspects of Protocol

- When do you send messages?
  - When any of your distances d(u,v) change
    - What about when c(u,v) 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

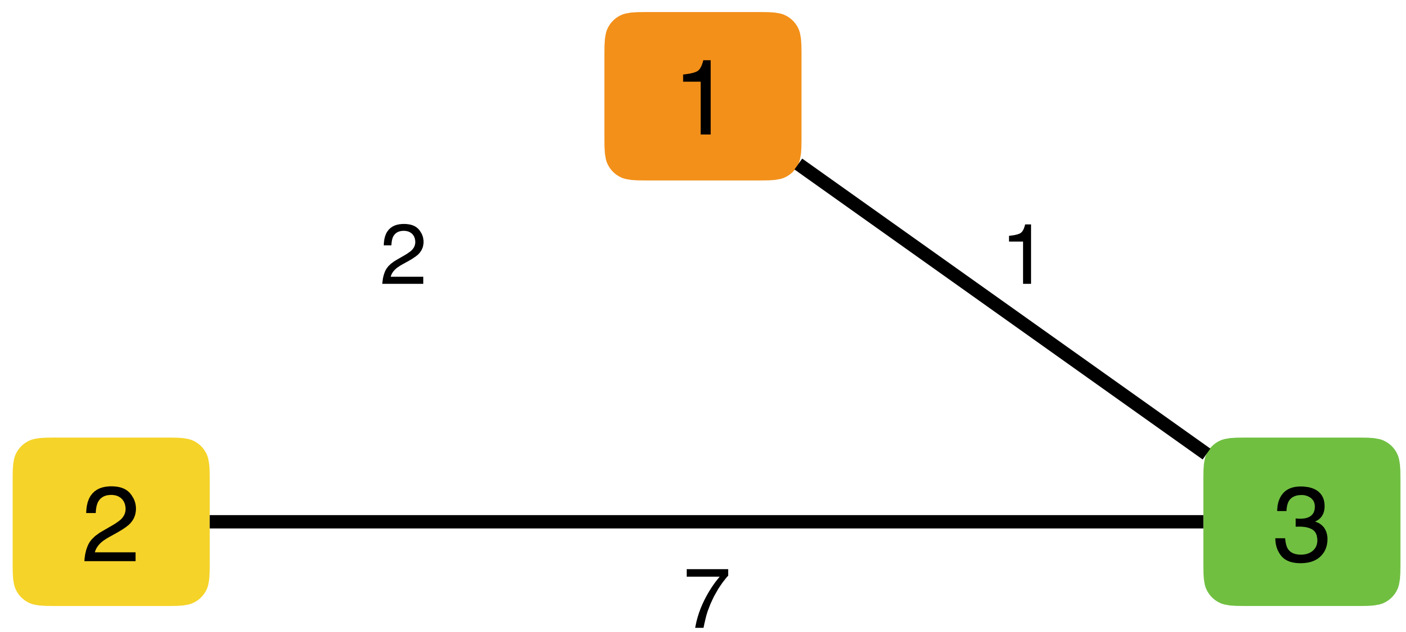
	distance	next-hop
1	0	-
2	2	2
3	1	3



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

Three node network

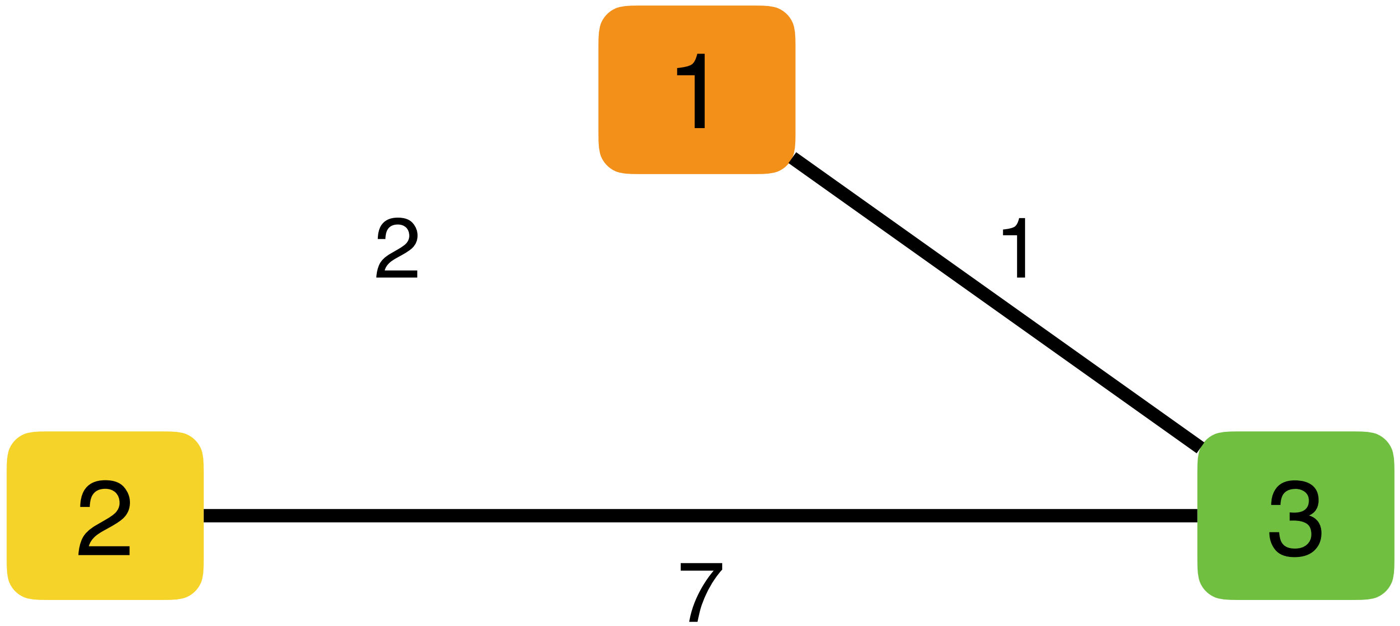
	distance	next-hop
1	0	-
2	infinity	
3	1	3



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

Round 1

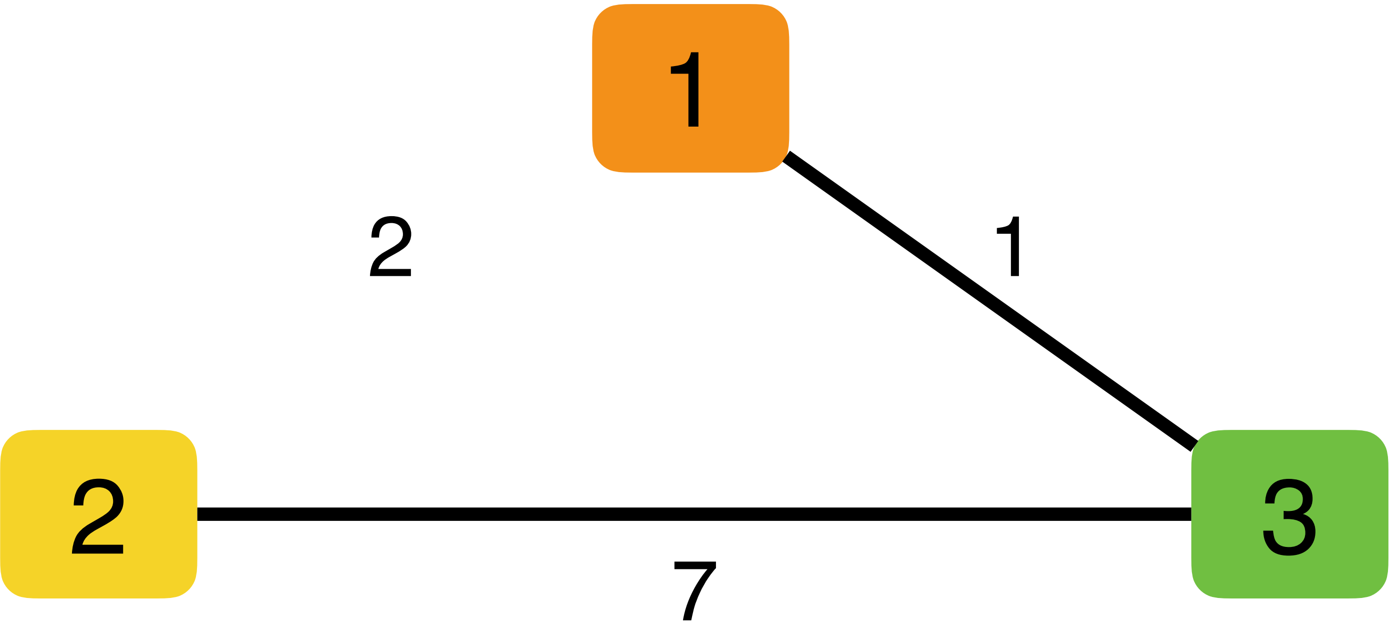
	distance	next-hop
1	0	-
2	4	3
3	1	3



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

Round 2

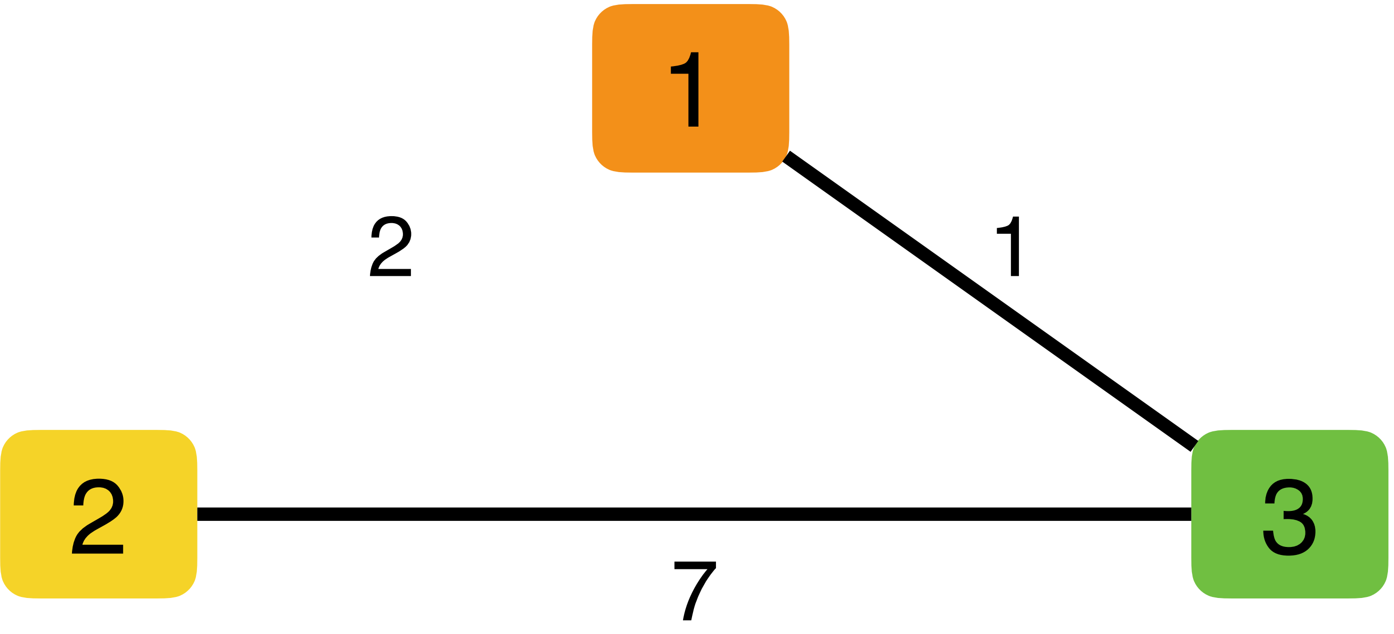
	distance	next-hop
1	0	-
2	4	3
3	1	3



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

Round 3

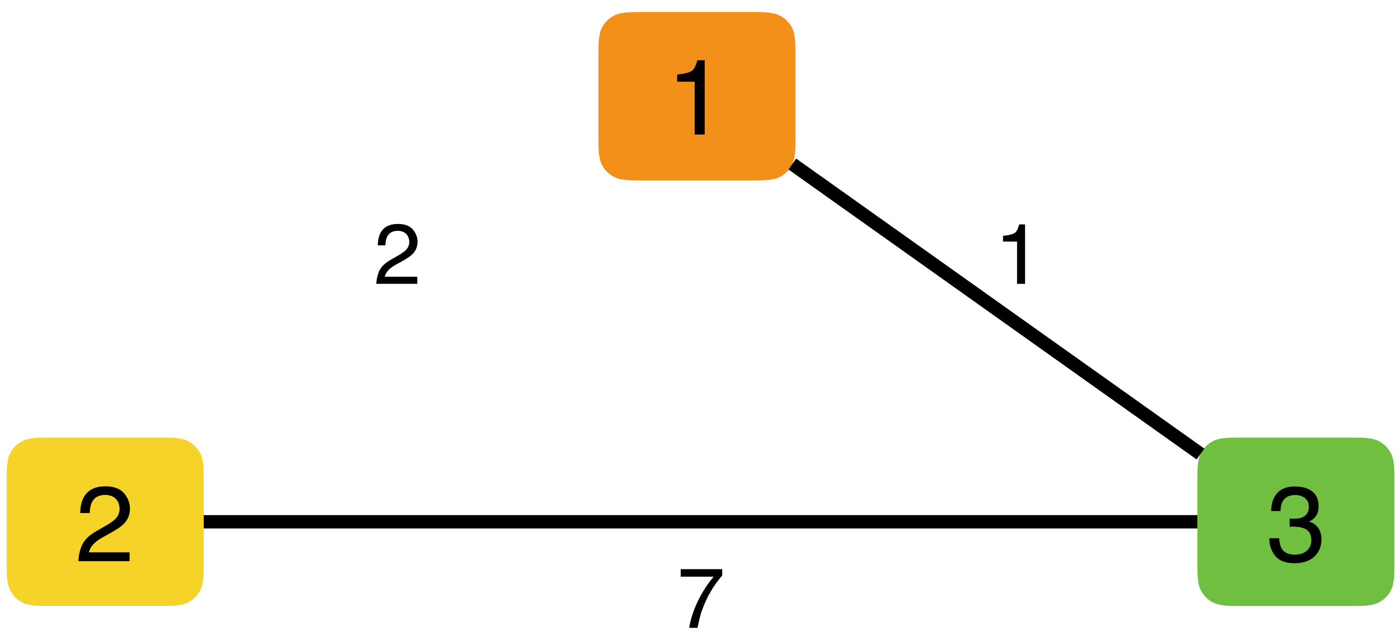
	distance	next-hop
1	0	-
2	6	3
3	1	3



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

Round 4

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

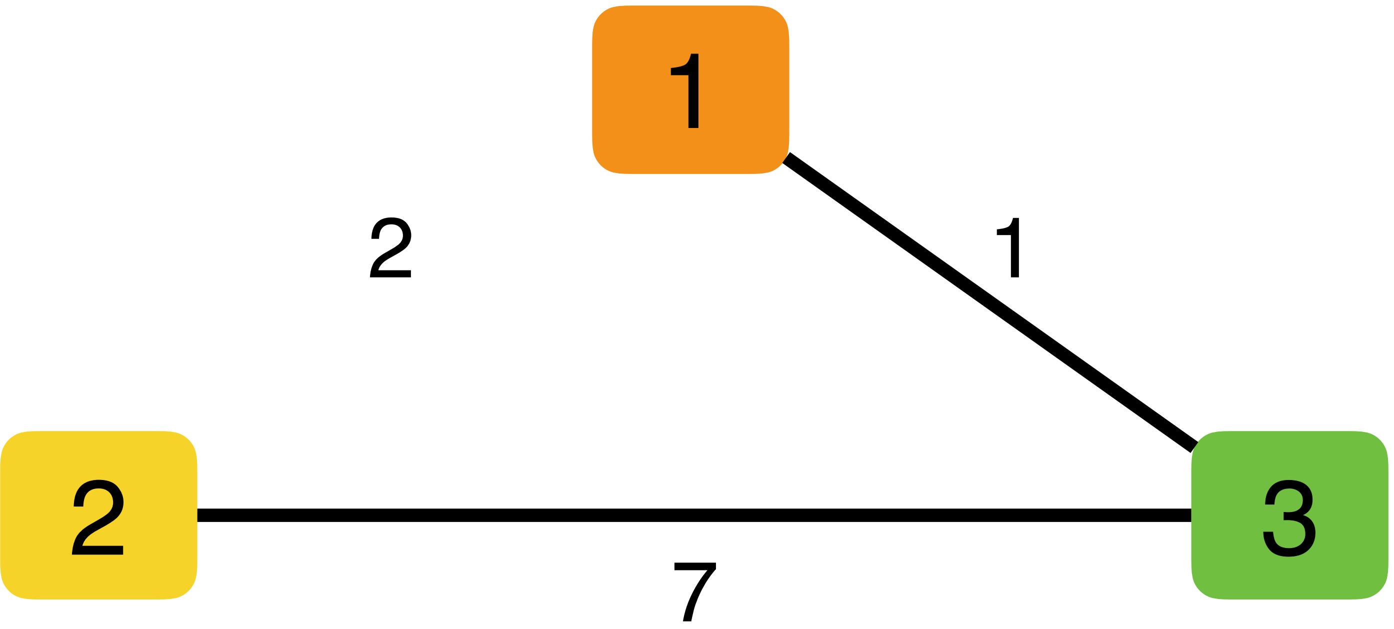


COUNT-TO-INFINITY problem!!!!

	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



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

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

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

Now you know just as much as my PhD students :-)