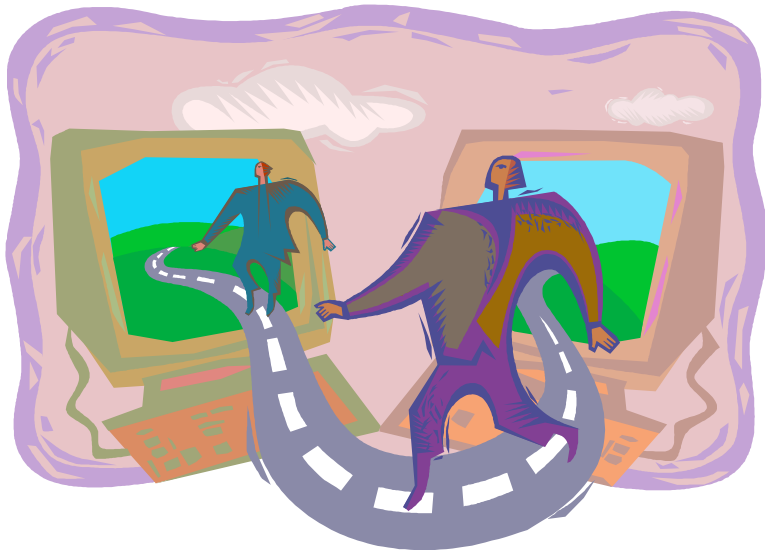
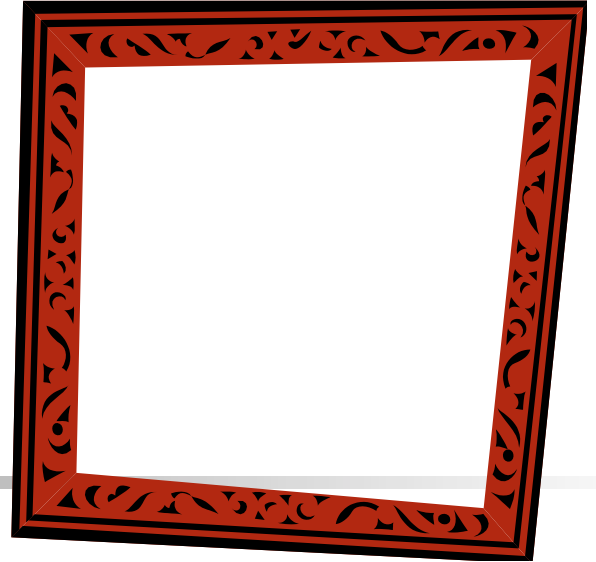


Discussion #10

EE450



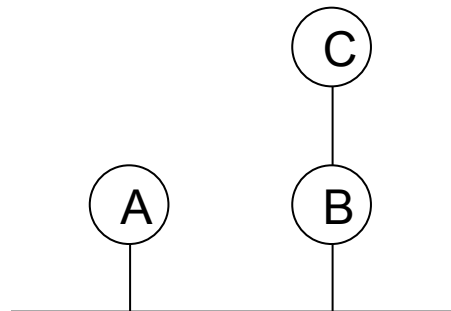
-Sample Problems on
Routing Protocols



Problem#8: Description



- Suppose hosts A and B are on an Ethernet LAN with class C IP network address 200.0.0.
- It is desired to attach a host C to the network via a direct connection to B (see the Figure below).
- Explain how to do this with subnets, give sample subnet assignments.
- Assume that an additional network address is not available. What does this do to the size of the Ethernet LAN?





Problem#8: Solution



To solve this with routing, C has to be given its own subnet. Even if this is small, this reduces the available size of the original Ethernet to at most seven bits of subnet address. **Here is a possible routing table for B.** Subnet numbers and masks are in binary. Note that many addresses match neither subnet.

net	Subnet	Mask	Interface
200.0.0	0/000 0000	1000 0000	Ethernet
200.0.0	1000 00/00	1111 1100	direct link

Here C's subnet has been made as small as possible, only two host bits are available (a single host bit can't be used because all-zero-bits and all-ones-bits are reserved in the host portion of an address.) C's address might now be 200.0.0.10000001, with the last octet again in binary.



Problem#5: Obtaining a Block of Addresses

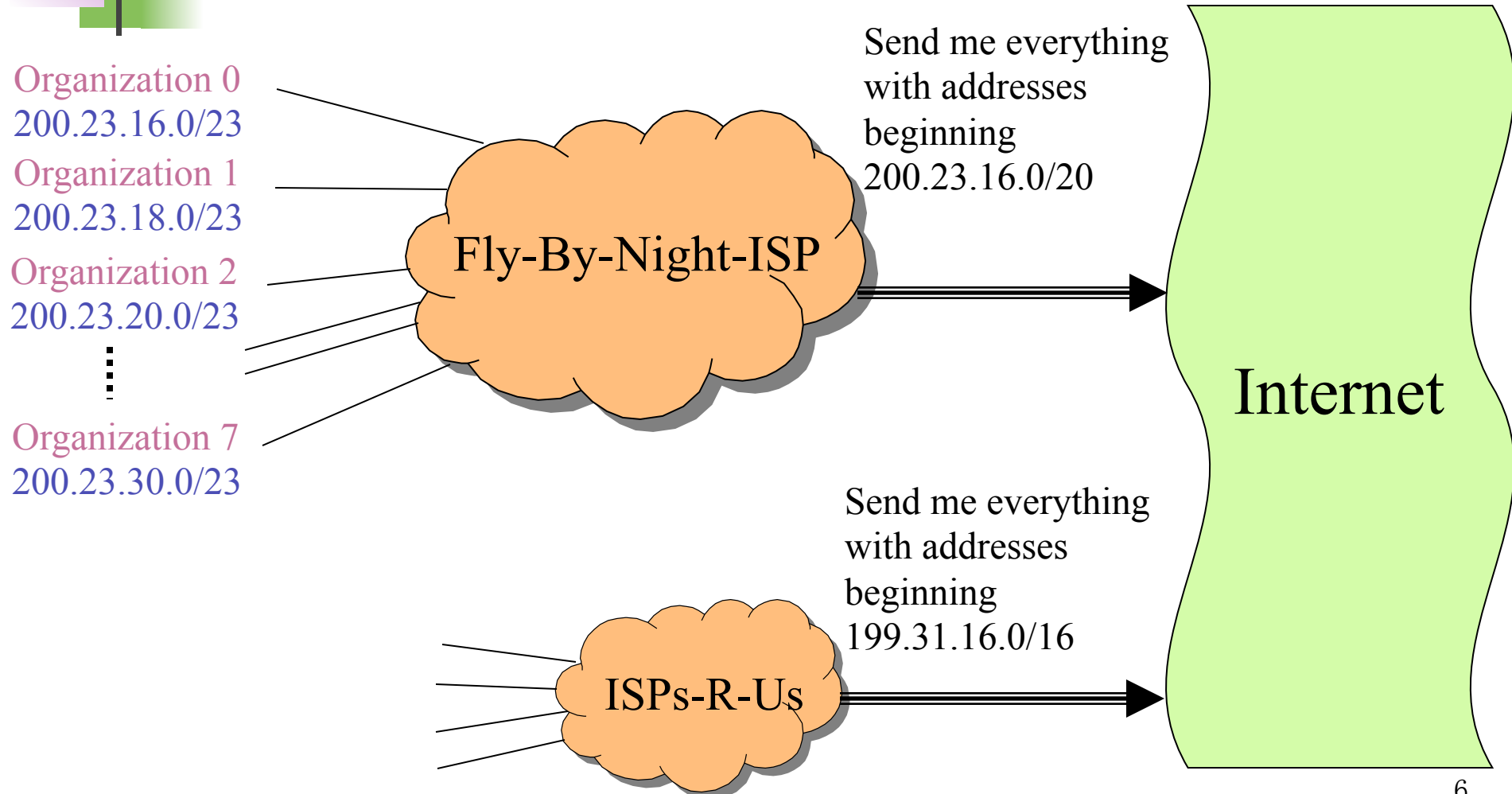
- In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP.
- The ISP would provide addresses from a larger block of addresses that had already been allocated to it.
- For example, the ISP may itself have been allocated the address block 200.23.16.0/20.
- It in turn could divide its address block into 8 equal sized contiguous address blocks and give out one of these address blocks to each of the eight organizations that are supported by this ISP.



The eight address blocks

ISP's block	200.23.16.0/20	<div>203</div> <div>11001000 00010111 00010000 00000000</div>
Organization 0	200.23.16.0/23	<div>23</div> <div><u>11001000 00010111 00010000</u>0 00000000</div>
Organization 1	200.23.18.0/23	<div><u>11001000 00010111 00010010</u>0 00000000</div>
Organization 2	200.23.20.0/23	<div><u>11001000 00010111 00010100</u>0 00000000</div>
Organization 3	200.23.22.0/23	<div><u>11001000 00010111 00010110</u>0 00000000</div>
Organization 4	200.23.24.0/23	<div><u>11001000 00010111 00011000</u>0 00000000</div>
Organization 5	200.23.26.0/23	<div><u>11001000 00010111 00011010</u>0 00000000</div>
Organization 6	200.23.28.0/23	<div><u>11001000 00010111 00011100</u>0 00000000</div>
Organization 7	200.23.30.0/23	<div><u>11001000 00010111 00011110</u>0 00000000</div>

Hierarchical Addressing and Route Aggregation





Address Aggregation And Route Aggregation

- As was shown in the previous Figure, the ISP Fly-By-Night advertises to the outside world that it should be sent any datagrams whose first 20 address bits match 200.23.16.0/20.
- The rest of the world need not know that within the address block 200.23.16.0/20 there are in fact eight other organizations, each with their own subnets.
- This ability to use a single prefix to advertise multiple networks is often referred to as address aggregation (also route aggregation or route summarization).
- This works extremely well when addresses are allocated in blocks to ISPs and then from ISPs to client organizations.



What if the addresses are not allocated in such a hierarchical manner?

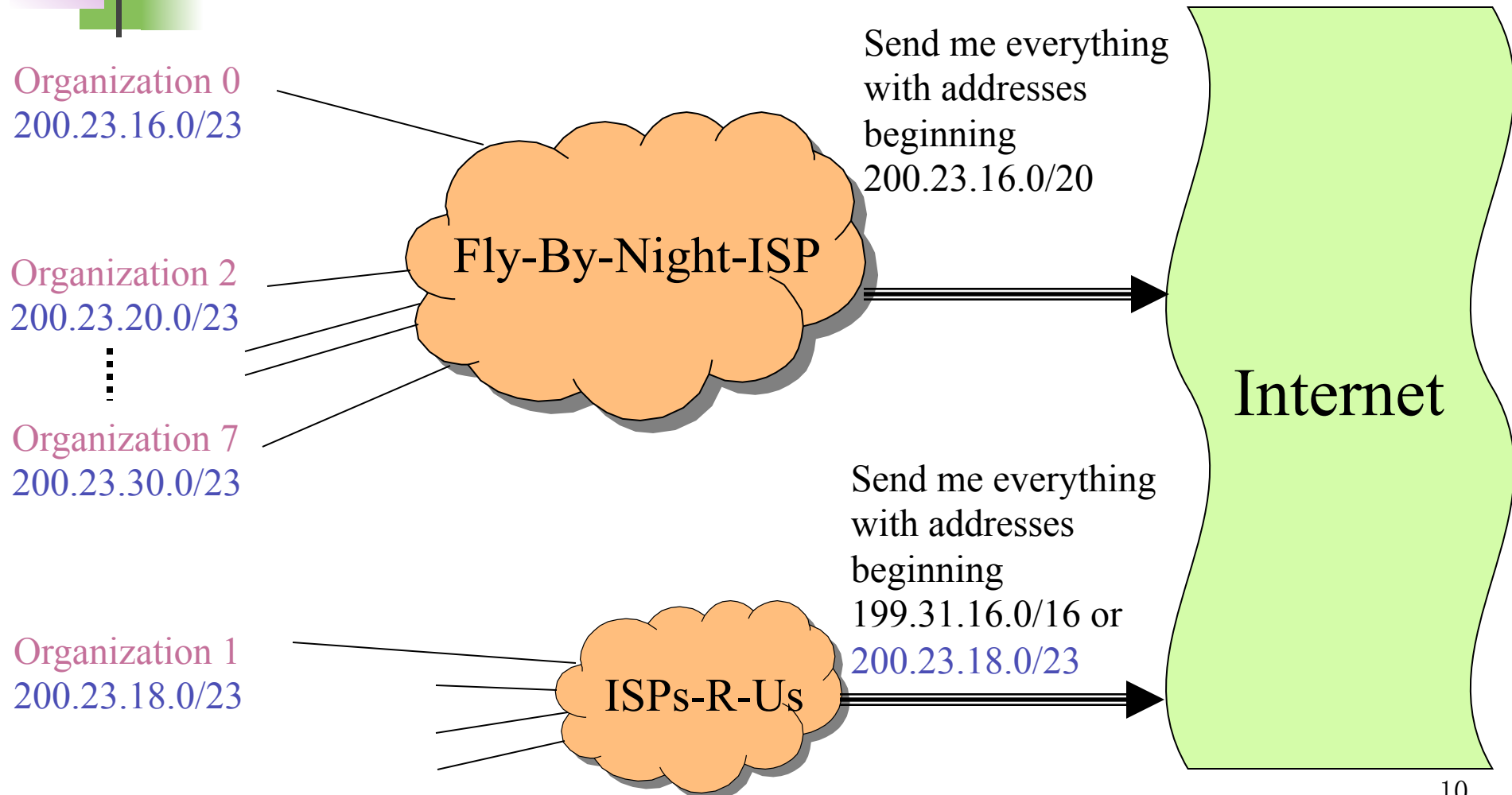
- For example , what would happen if ISP Fly-By-Night acquires ISPs-R-Us and then has Organization 1 connect to the Internet through its subsidiary ISPs-R-Us?
- As was shown in the Figure, ISPs-R-Us owns the address block 199.31.0.0/16 but Organization 1's IP addresses are unfortunately outside of this address block.
- What should be done here?



Proposed Solutions

- Organization 1 could renumber all of its routers and hosts to have addresses within the ISPs-R-U's address block.
 - It's a costly solution.
 - Organization 1 might well be reassigned to another subsidiary in the future.
- Organization 1 keeps its IP addresses in 200.23.18.0/23 and ISPs-R-U's advertises the block of addresses for Organization 1 (in addition to its own block of addresses.)
 - When routers in the Internet see the address block 200.23.16.0/20 (from Fly-By-Night) and 200.23.18.0/23 (from ISPs-R-U's), and want to route to an address in the block 200.23.18.0/23, they will use longest prefix matching and route towards ISPs-R-U's as it advertises the longest (most specific) address prefix that matches the destination address.

ISPs-R-Us has a more specific route to Organization 1





Link-state Routing Protocol

- One of the two main classes of routing protocols used in the computer networks (e.g. OSPF).
- Is performed by every node (i.e. *router*) in the network
 - Every node receives a *map* of the connectivity of the network, in the form of a graph showing which nodes are connected to which other nodes.
 - Each node then independently calculates the best *next hop* from it for every possible destination in the network.
 - The collection of best next hops forms the routing table for the node.
- This contrasts with distance-vector routing protocols, which work by having each node share its **routing table** with its neighbors.
- In a link-state protocol, the only information passed between the nodes is information used to construct the connectivity maps.



Creating Network Connectivity Map

- Determining the neighbors of each node using a simple *reachability protocol*
- Distributing the information for the map by periodically sending *link-state advertisement* which:
 - Identifies the node which is producing it.
 - Identifies all the other nodes to which it is directly connected.
 - Includes a *sequence number*
 - Is *flooded* throughout the network.
- Creating the map by iterating over the collection of link-state advertisements
 - For each one, it makes links on the map of the network, from the node which sent that message, to all the nodes which that message indicates are neighbors of the sending node.
 - No link is considered to have been correctly reported unless the two ends agree.

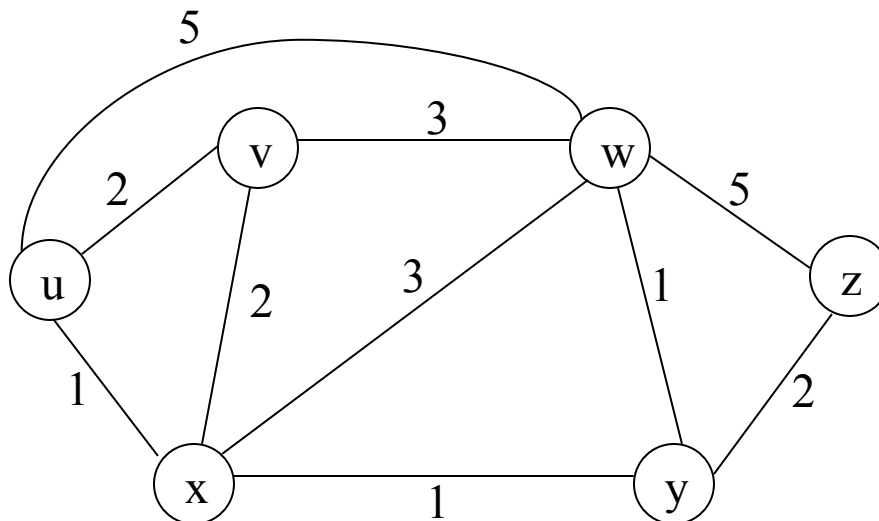


Dijkstra's Algorithm

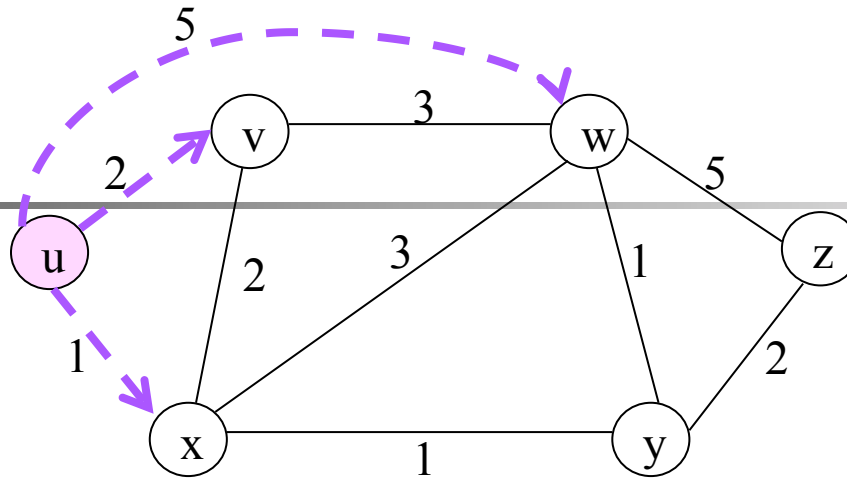
- The input of the algorithm consists of a weighted directed graph G and a source vertex s in G .
 - V is the set of all vertices and E is the set of all edges in G .
 - Weights of edges denoted by $w(u,v)$ is the cost of moving from u to v .
- The algorithm keeps for each vertex v the cost $d[v]$ of the shortest path found so far between s and v .
- Initially, for the source vertex s , $d[s]=0$, and for every v in V except s , $d[v]=\infty$.
- Eventually, $d[v]$ will be the cost of the shortest path from s to v — or infinity, if no such path exists.
- The algorithm maintains two sets of vertices S and Q .
 - Set S contains all vertices for which we know that the value $d[v]$ is already the cost of the shortest path and set Q contains all other vertices.
 - Set S starts empty, and in each step one vertex is moved from Q to S . This vertex is chosen as the vertex with lowest value of $d[u]$.
 - When a vertex u is moved to S , the algorithm relaxes every outgoing edge (u,v) .

Problem #1: How the *Link-State* Algorithm Builds the Routing Table for Node u

- $D(v)$: Cost of the least-cost path from the source node to destination v as of this iteration of the algorithm
- $p(v)$: Previous node (neighbor of v) along the current least-cost path from the source to v .
- N' : Subset of nodes; v is in N' if the least-cost path from the source to v is definitely known.

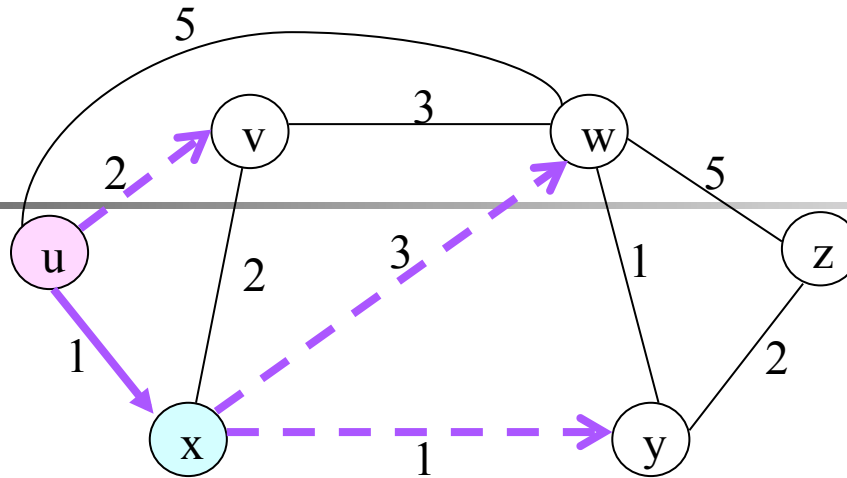


Running the link-state algorithm on the this network



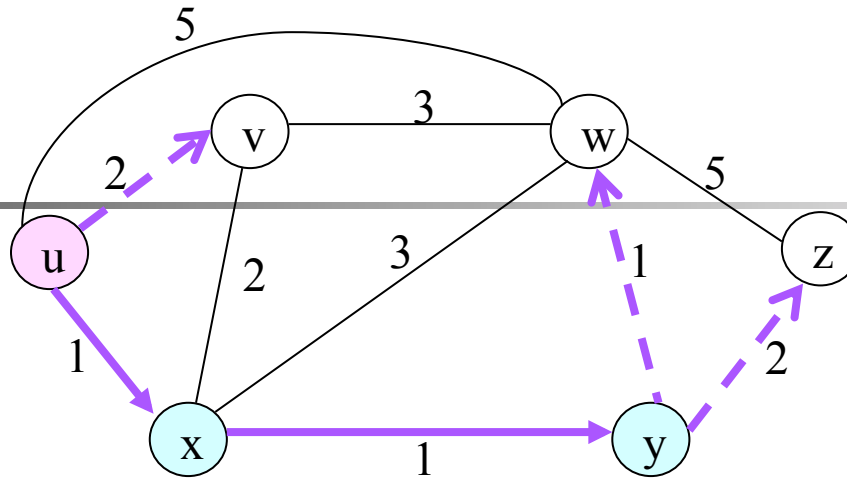
Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞

Running the link-state algorithm on the this network



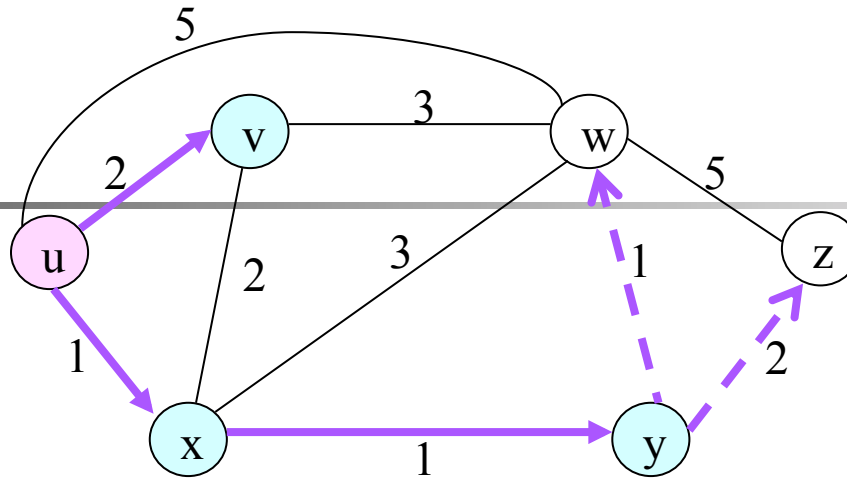
Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞

Running the link-state algorithm on the this network



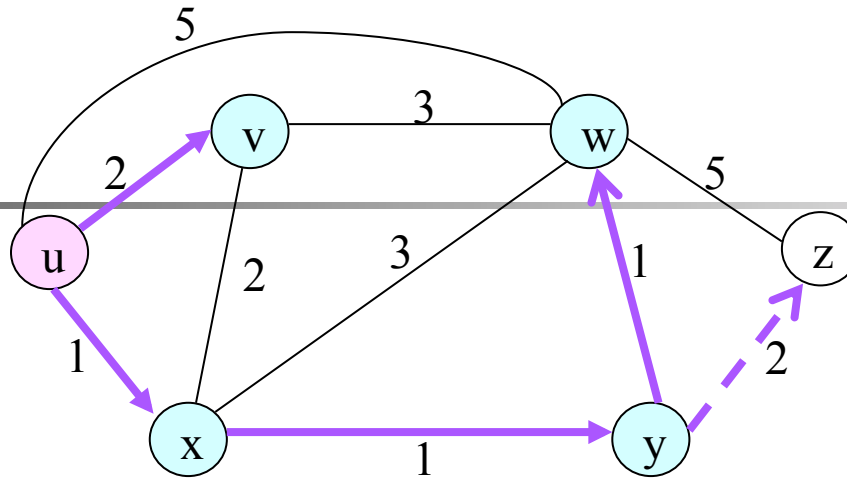
Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y

Running the link-state algorithm on the this network



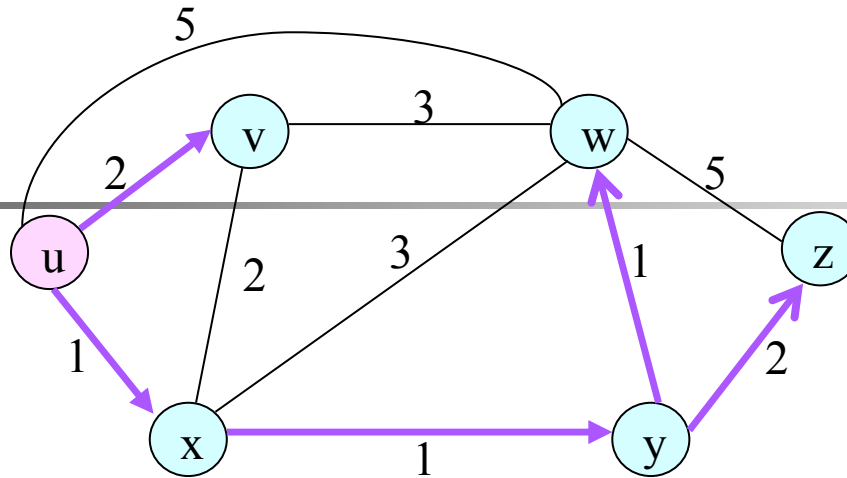
Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y

Running the link-state algorithm on the this network



Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y

Running the link-state algorithm on the this network



Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Distance-Vector Routing Protocol

- One of the two main classes of routing protocols used in the computer networks (e.g. RIP).
- Uses the *Bellman-Ford algorithm* to calculate paths
- A distance-vector routing protocol requires that
 - A router informs its neighbors of topology changes
 - Periodically
 - Whenever a change is detected in the topology of a network
 - Unlike link-state protocols doesn't require the router to inform all the nodes in a network of topology changes
 - Hence less computational complexity and message overhead



Bellman-Ford Algorithm

- A distributed version of Bellman–Ford algorithm is used in distance-vector routing protocols, (e.g. RIP).
 - Each node (i.e. router) calculates the distances between itself and all other nodes and stores this information as a table.
 - Each node sends its table to all neighboring nodes.
 - When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.
- Disadvantages of the Bellman–Ford algorithm in this context
 - Scalability
 - Slow convergence
 - Count-to-infinity problem
 - If failure of a link or a node renders a node unreachable from some other nodes, those nodes may indefinitely and gradually increase their cost estimates of the distance to the unreachable node, and routing loops may also be formed.



Count-to-Infinity Problem

- Consider an example topology A—B—C—D—E (hop-count is the metric)
 - A goes down.
 - B does not receive the vector update from A so it concludes that its route of cost 1 to A is no longer available.
 - C doesn't know yet that A is down and tells B that A is 2 hops away from it
 - The wrong info propagates until it reaches infinity.
 - The algorithm then corrects itself using the "Relax property" of Bellman Ford.
- **Partial Solutions**
 - ***Split Horizon:*** prohibits a router from advertising a route back out the interface from which it was learned
 - ***Split Horizon with poison reverse:*** Allows a router to advertise the route back to the router that is used to reach the destination, but marks the advertisement as unreachable.
 - More theoretical than practical, allows more scalable and complex DV protocols

Problem #2: How the *Distance-Vector* Algorithm Builds the Routing Table for Node z

- Consider the network shown below and assume that each node initially knows the costs to each of its neighbors. Consider the distance vector algorithm and show the distance tables entries at node z.

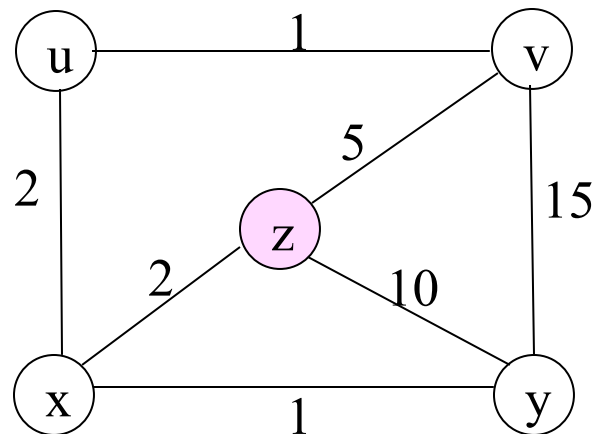
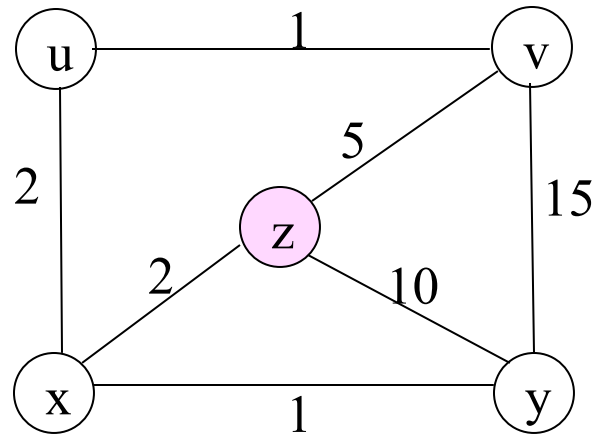
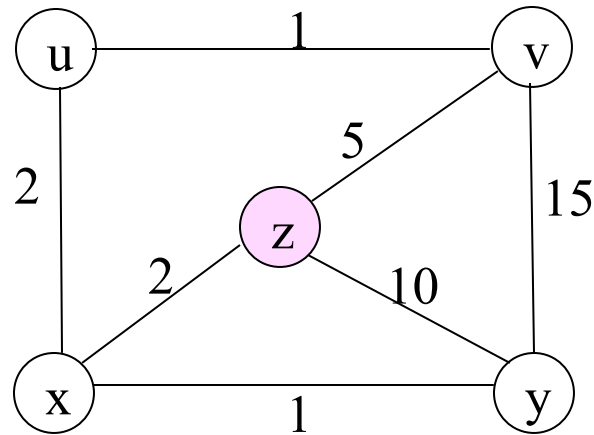


Table at Node Z: Step 1



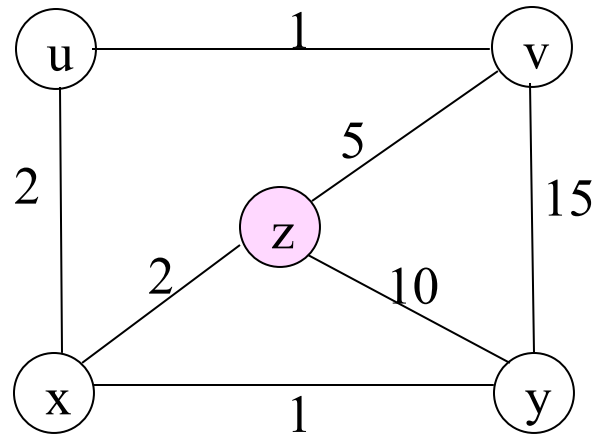
		Cost to				
		u	v	x	y	z
From	v	∞	∞	∞	∞	∞
	x	∞	∞	∞	∞	∞
	y	∞	∞	∞	∞	∞
	z	∞	5	2	10	0

Table at Node Z: Step 2



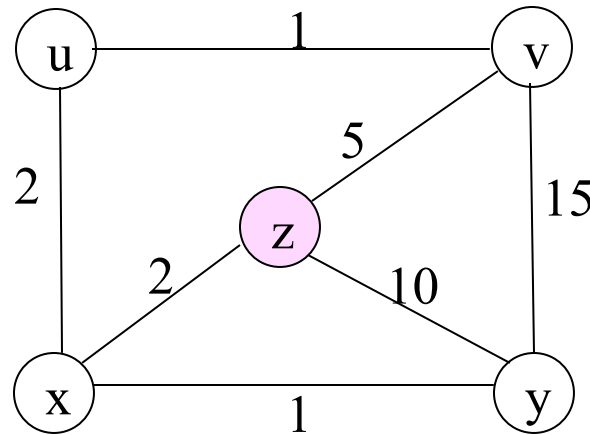
		Cost to				
		u	v	x	y	z
From	v	1	0	∞	15	5
	x	2	∞	0	1	2
	y	∞	15	1	0	10
	z	4	5	2	3	0

Table at Node Z: Step 3



		Cost to				
		u	v	x	y	z
From	v	1	0	3	15	5
	x	2	3	0	1	2
	y	3	15	1	0	3
	z	4	5	2	3	0

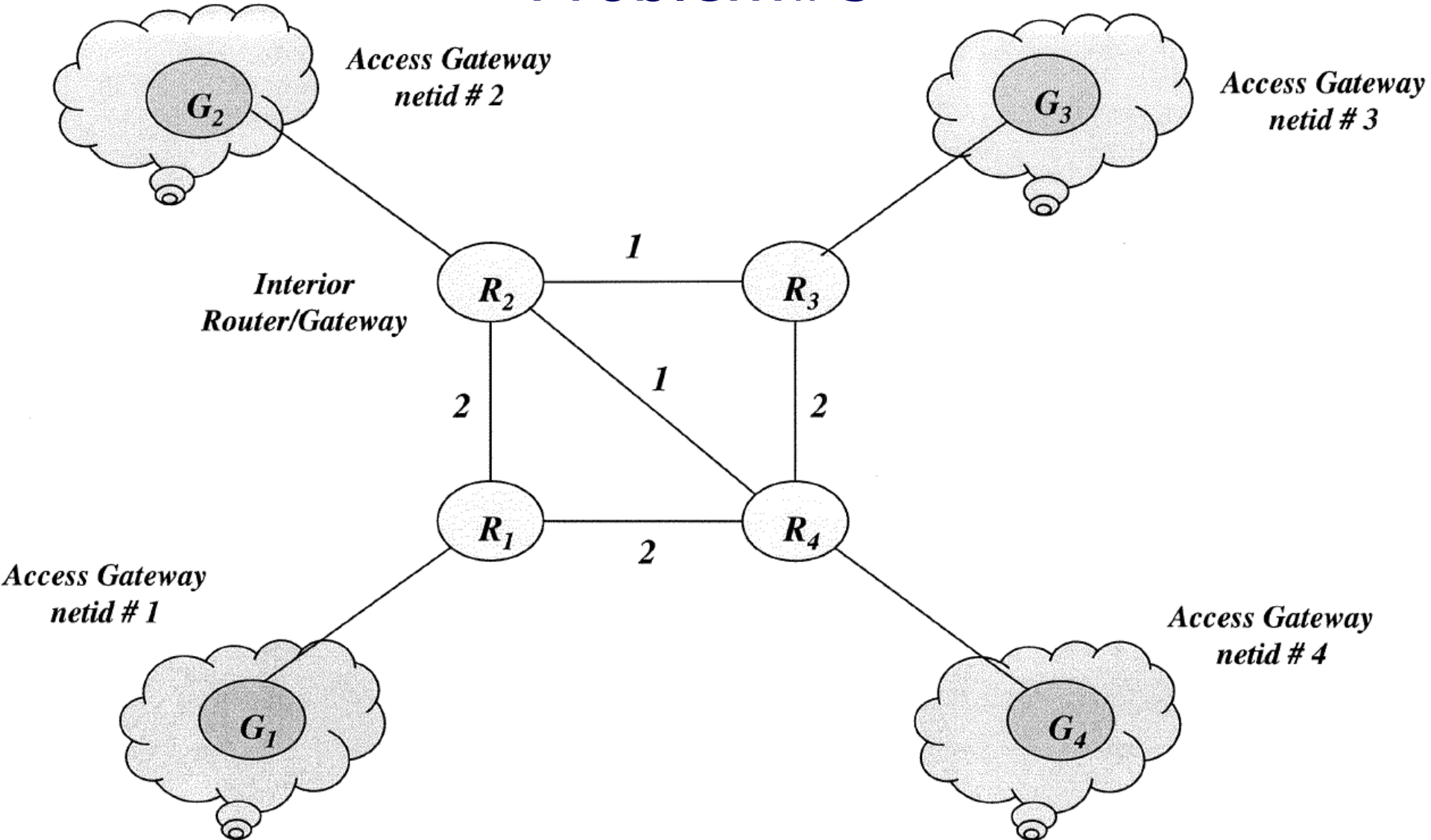
Table at Node Z: Step 4

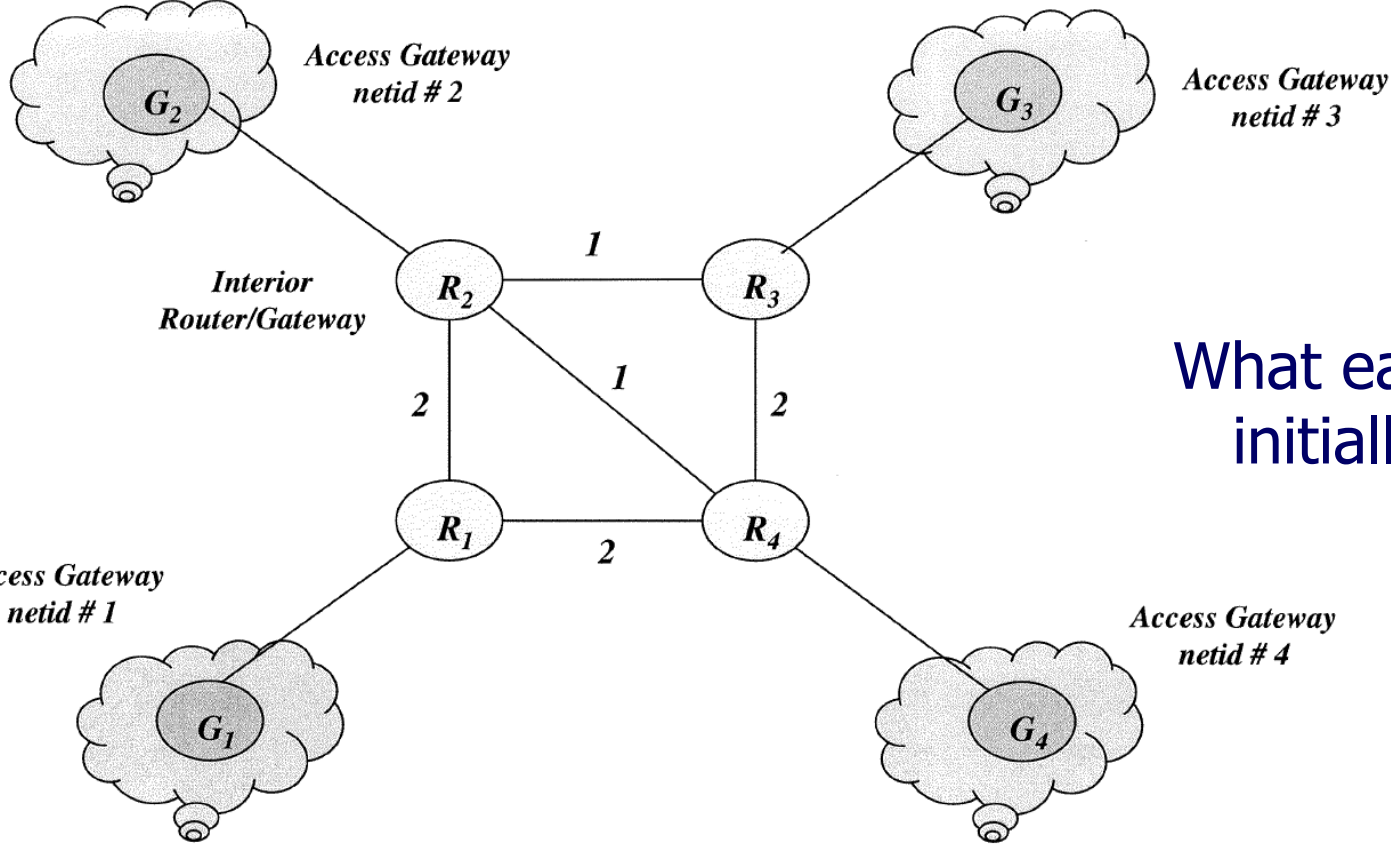


		Cost to				
		u	v	x	y	z
From	v	1	0	3	4	5
	x	2	3	0	1	2
	y	3	4	1	0	3
	z	4	5	2	3	0

Example of a DV Routing

Problem#3






Info at	(Next Hop, Distance to)			
	netid1	netid2	netid3	netid4
R1	R1,0	∞	∞	∞
R2	∞	R2,0	∞	∞
R3	∞	∞	R3,0	∞
R4	∞	∞	∞	R4,0


Initial/Intermediate/Final Routing Tables

@ R_1

<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 0$



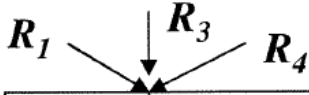
<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 0$
2	$R_2, 2$
4	$R_4, 2$



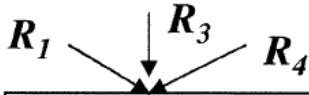
<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 0$
2	$R_2, 2$
3	$R_2, 3$
4	$R_4, 2$

@ R_2

<i>Netid</i>	<i>Next Hop, D</i>
2	$R_2, 0$



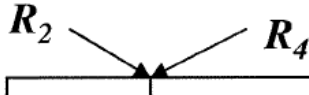
<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 2$
2	$R_2, 0$
3	$R_3, 1$
4	$R_4, 1$



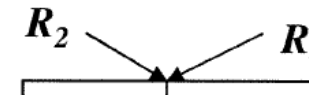
<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 2$
2	$R_2, 0$
3	$R_3, 1$
4	$R_4, 1$

@ R_3

<i>Netid</i>	<i>Next Hop, D</i>
3	$R_3, 0$



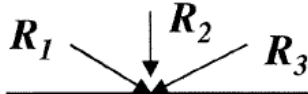
<i>Netid</i>	<i>Next Hop, D</i>
2	$R_2, 1$
3	$R_3, 0$
4	$R_4, 2$




<i>Netid</i>	<i>Next Hop, D</i>
1	$R_2, 3$
2	$R_2, 1$
3	$R_3, 0$
4	$R_4, 2$

@ R_4

<i>Netid</i>	<i>Next Hop, D</i>
4	$R_4, 0$



<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 2$
2	$R_2, 1$
3	$R_3, 2$
4	$R_4, 0$

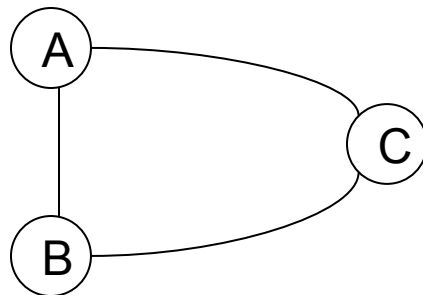


<i>Netid</i>	<i>Next Hop, D</i>
1	$R_1, 2$
2	$R_2, 1$
3	$R_3, 2$
4	$R_4, 0$

Problem#4: Description



- Suppose that nodes in the network shown in this Figure participate in link-state routing, and C receives contradictory LSPs:
 - One from A arrives claiming the A-B link is down, but one from B arrives claiming the AB link is up.
- a) How could this happen?
- b) What should C do? What can C expect?





Problem#4: Solution



- a) This could happen if the link changed recently and one of the two LSPs was old.
- b) If flooding is working properly, and if A and B do in fact agree on the state of the link, then eventually (rather quickly) whichever of the two LSPs was old would be updated by the same sender's newer version, and reports from the two sides of C would again agree.