Chapter-3: Routing Protocols

Deep Medhi and Karthik Ramasamy

August 2007

http://www.NetworkRouting.net

Three terms: Routing Protocol, Routing Algorithm, and Routing Table

- A routing protocol refers to the mechanism and information communicated (and the format of information) between routing nodes so that actual user data can be forwarded
- A routing algorithm refers to an algorithm that may be used by a routing node based on the information received through a routing protocol to determine best paths to different destinations
- A routing table is a lookup table maintained at a routing node that provides lookup entry for each destination by identifying the outgoing link.
  - Forwarding table, conceptually no different than a routing table; however, it maintains information at the interface level (layer-2), so that when a packet arrives, it can be forwarded on the output interface

More …

- The framework of a routing algorithm has resulted in the development of an associated routing protocol
- However, the reverse need not be true
- For example, Dijsktra's shortest path algorithm has close relation to the link-state routing protocol; yet, a link-state routing protocol does not need to use Dijkstra's shortest path algorithm to determine the next hop.

Information Representation

- How to represent routing information
  - Depends on the specific routing protocol
  - For a distance vector protocol, a neighbor needs to communicate its distance to *j (at 2)*

Destination node, Distance Cost to Destination

$j=1, \overline{D}=1$	$j=2, \overline{D}=0$	$j=3, \overline{D}=2$	$j=4, \overline{D}=1$	$j=5, \overline{D}=3$	$j=6, \overline{D}=3$

  - In a link state protocol, a neighbor forwards the link cost of different links it is aware of:

Source node,Link-ID, link Cost

$1, 1 \rightarrow 2, 1$

Distance Vector Protocol: some basics

- A node relies on distance information it receives from its neighbors
- Compares them (by adding its costs to its neighbors) and select the best neighbor
- How often to communicate: protocol doesn't dictate
- The protocol does not need to know ahead of time how many nodes are in the network; in other words, through the information received periodically from its neighbors, which may contain a new node information, the receiving node can update the list of destination nodes.

DV protocol: basics (cont'd)

- Actual numbering of a node can be done through some addressing scheme outside the protocol (for example, IP addressing with RIP).
- For each destination *j* (from node *i*), the protocol maintains/updates the next hop, to be denoted as *Hij*.
- With the arrival of a distance vector message from a neighbor *k*, the protocol updates the cost to a destination if the currently stored next hop for this destination is also *k*.
- Steps indicated are not necessarily in any specific order (except for initialization).
- There is possibly a time wait between steps and within substeps of a step.
- Routing computation is a step within the protocol operation

Distance vector protocol (node *i*'s view)

Initialize:

- Node is configured with a unique node ID, say i
- Node i's distance vector to itself is set to zero, i.e., $\overline{D}_{ii} = 0$
- Initialize module that determines the link cost to its directly connected neighbor (either manually or based on measurements), i.e., d_{ik} for all neighbors k, and set the routing table entry to indicate the next hop for k as k itself, i.e., $H_{ik} = k$

Transmit mode:

- Transmit the most recently computed cost (distance) for all known destination nodes to all its neighbors k on a periodic basis

Receive mode:

- Node i receives a distance vector from its neighbor k

  a. If the distance vector contains a new destination node j', then a new node entry in the routing table is created and set $\overline{D}_{ij'} = \infty$

  b. The distance vector $\overline{D}^i_{kj}$ for each destination node j received from neighbor k at node i is temporarily stored

  c. If the currently stored next hop for a destination j is the same as k itself, then update the distance cost for this destination, i.e.,

  If ($H_{ij} = k$) then // if next hop is k

  $\overline{D}_{ij} = d_{ik} + \overline{D}^i_{kj}$

  Endif

DV protocol (cont'd)

- Route Computation

  For each destination j: // shortest path computation

  For all neighbors k (or, the last received neighbor k)

  Compute $temp = d_{ik} + \overline{D}^{i}_{kj}$

  If ($temp < \overline{D}_{ij}$) then

  $\overline{D}_{ij} = temp$ // update the new cost

  $H_{ij} = k$ // update next hop in the routing table

  Endif

Special Cases:

- If for any neighbor k, link i-k goes down, then

  Set $d_{ik} = \infty$

  If $H_{ij} = k$, then $\overline{D}_{ij} = \infty$

  Broadcast a distance vector to each neighbor

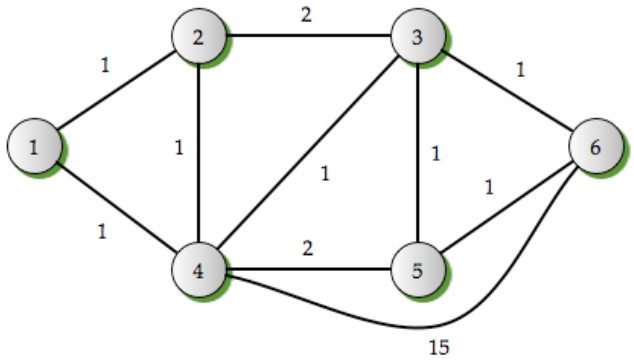
  Endif

- If for any neighbor k, link i-k is up again, then

  Update d_{ik} (fixed or dynamic)

  Broadcast a distance vector to each neighbor

  Endif



- At optimality, node 1 will
- Send the following DV
- For all destinations it knows
- (from 2)

$j=1, \overline{D}=1$	$j=2, \overline{D}=0$	$j=3, \overline{D}=2$	$j=4, \overline{D}=1$	$j=5, \overline{D}=3$	$j=6, \overline{D}=3$

- Node-1 receives DV from Node-4 as follows:

$j=1, \overline{D}=1$	$j=2, \overline{D}=1$	$j=3, \overline{D}=1$	$j=4, \overline{D}=0$	$j=5, \overline{D}=2$	$j=6, \overline{D}=2$

- Assume the table at node-1 to be as follows:

TABLE 3.1 Routing table information at node 1 (after receiving distance vector from node 2).

Destination Node	Cost	Outgoing Link
1	0	local
2	1	1-2
3	3	1-2
4	2	1-2
5	4	1-2
6	4	1-2

- Now, node-1 receives the following DV from node-4

$j=1, \overline{D}=1$	$j=2, \overline{D}=1$	$j=3, \overline{D}=1$	$j=4, \overline{D}=0$	$j=5, \overline{D}=2$	$j=6, \overline{D}=2$

- Now, node-1 compares the cost:

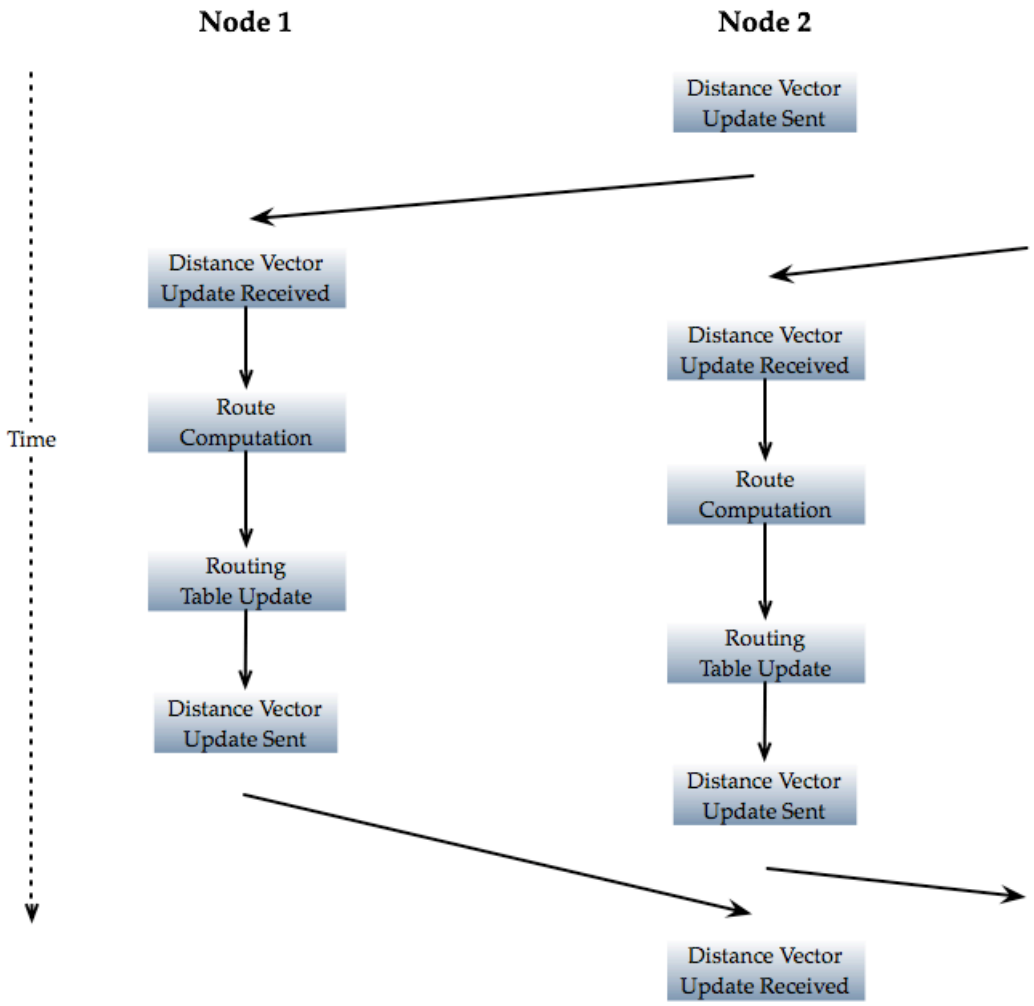
TABLE 3.2 Cost and routing able updating at node 1 (after receiving distance vector from node 4).

Destination Node	Current Cost	New Possible Cost	Updated Cost (Update?)	Update Outgoing Link (If Any)
1	0	1 + 1	0 (No)	local
2	1	1 + 1	1 (No)	1-2
3	3	1 + 1	2 (Yes)	1-4
4	2	1 + 0	1 (Yes)	1-4
5	4	1 + 2	3 (Yes)	1-4
6	4	1 + 2	3 (Yes)	1-4

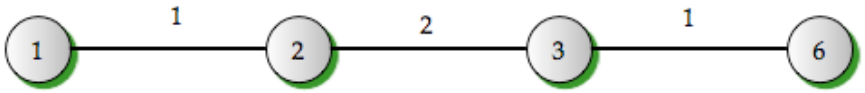
Some observations

- *The order of information as received matters:*
  - In the example discussed so far, we started by assuming that node 1 receives a distance vector from node 2 first *before* receiving a distance vector from node 4. Had node 1 received a distance vector from node 4 first, the entire routing table in the first round would have been different.
- *How often the distance vector information is disseminated matters:*
  - Assume for the sake of argument that a distance vector is disseminated by node 2 every minute while node 4 does it every 10 min. Clearly, this would make a difference to node 1 in terms of how quickly it would be able to arrive at the best routing table.
- *The instant when a node broadcasts the distance vector (after an event) matters:*
  - It is important to distinguish this item from the previous item. While the previous item discusses periodicity of the update interval, this one refers to whether a node should trigger an immediate update after a major event such as a failure of a link connected to it.
- *The instant when a routing computation is performed matters:*
  - Suppose a node receives a distance vector from a neighbor every 2 min while it performs the route computation ever 3min. Certainly, these update cycles have an impact on obtaining the best route in timely manner.
- *The instant when the routing table is updated matters:*
  - Suppose a node waits another 30 sec after performing a route computation before updating the routing table. This would impact the flow of user data.

Time line of different activities at different nodes



DV – slow convergence



Time: t= 0 sec: Nodes 1, 2, 3, and 6 are activated and the initial distance vector broadcast is sent.
Time: t= 1 sec: Routing tables at different nodes:

Node 1:

D-node	Cost	O-link
1	0	local
2	1	1-2

Node 2:

D-node	Cost	O-link
1	1	2-1
2	0	local
3	2	2-3

Node 3:

D-node	Cost	O-link
2	2	3-2
3	0	local
6	1	3-6

Node 6:

D-node	Cost	O-link
3	1	6-3
6	0	local

Time: t= 60 sec: Distance vector broadcast.
Time: t= 61 sec: Routing tables at different nodes:

Node 1:

D-node	Cost	O-link
1	0	local
2	1	1-2
3	3	1-2

Node 2:

D-node	Cost	O-link
1	1	2-1
2	0	local
3	2	2-3
6	3	2-3

Node 3:

D-node	Cost	O-link
1	3	3-2
2	2	3-2
3	0	local
6	1	3-6

Node 6:

D-node	Cost	O-link
2	3	6-3
3	1	6-3
6	0	local

Time: t= 120 sec: Distance vector broadcast.
Time: t= 121 sec: Routing tables at different nodes:

Node 1:

D-node	Cost	O-link
1	0	local
2	1	1-2
3	3	1-2
6	**4**	1-2

Node 2:

D-node	Cost	O-link
1	1	2-1
2	0	local
3	2	2-3
6	3	2-3

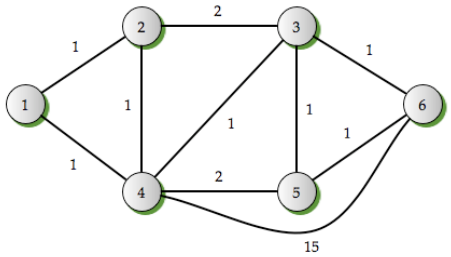
Node 3:

D-node	Cost	O-link
1	3	3-2
2	2	3-2
3	0	local
6	1	3-6

Node 6:

D-node	Cost	O-link
1	**4**	6-3
2	3	6-3
3	1	6-3
6	0	local

Routing Loop (DV): illustration



time, t_0 —converged state; a routing computation performed for all destinations.

time, t_1 —nodes 2 and 3 update their respective routing tables (based on the result of routing computation at time t_0).

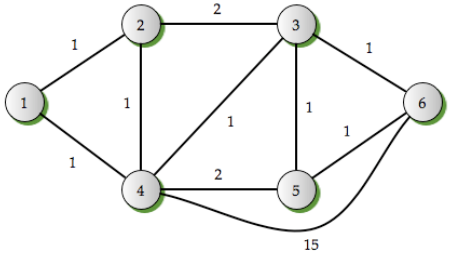
time, t_2 —link 3-6 fails.

time, t_3 —node 3 updates its routing table entry for destination node 6 by listing cost as ∞.

time, t_4 —node 2 sends distance vector to node 3.

time, t_5 —nodes 2 and 3 both perform a routing computation for all destinations.

time, t_6 —nodes 2 and 3 update their respective routing tables.



- Assume DV converges; at time t_1:

At node 2:

Destination Node	Cost	Outgoing Link
6	3	2-3

At node 3:

Destination Node	Cost	Outgoing Link
6	1	3-6

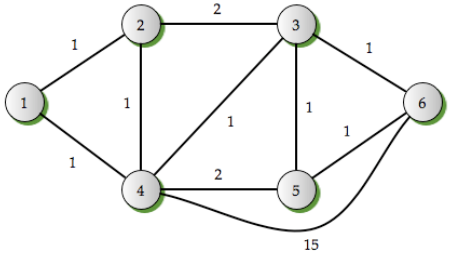
- Time t_2: link 3-6 fails
- Time t_3:Entry for destination-6 (at node 3)

Destination Node	Cost	Outgoing Link
6	∞	3-6

- Time t_4: node-3 receives DV from node-2

$j=1, \overline{D}=1$	$j=2, \overline{D}=0$	$j=3, \overline{D}=3$	$j=4, \overline{D}=1$	$j=5, \overline{D}=3$	$j=6, \overline{D}=3$

2



- Routing table at node 2 for destination 3 and 6:

Destination Node	Cost	Outgoing Link
3	2	2-3
6	3	2-3

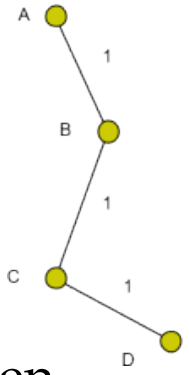
- At node 3, the following update occurs:

Destination Node	Current Cost	New Possible Cost	Updated Cost (update?)	Update Outgoing Link (if any)
6	∞	1 + 3	4 (yes)	3-2

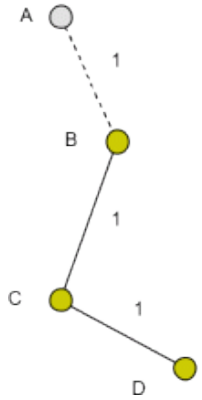
- Now, node 3 is pointing to node 2, and node 2 is pointing to node 3: loop!

Counting-to-infinity

- Simpler example from
- http://wiki.uni.lu/secan-lab/Count-To-Infinity+Problem.html



B	C	D
1	2	3



B	C	D
3	2	3

B	C	D
3	4	3

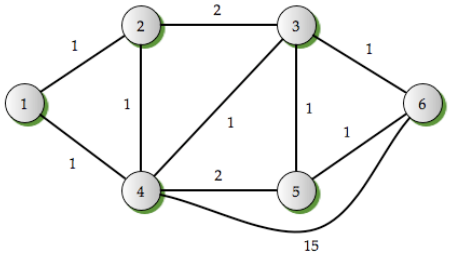
B	C	D
5	4	5

Counting-to-Infinity:

- Link(A, B) is broken.
- Router B observed it, but notes that router C has a route to A with 2 hops.
- What's the problem:
  - Router B doesn't know that C has router B as next-hop in its routing table on the route to A.
- So, B updates its routing table with new cost and let C know.
- From the second figure, see the new distances to A.
- In C's routing the route to A contains router B as next hop, so if B has increased its cost to A, then C is forced to do so. Thus, Router C increases his cost to A about B + 1 = 4.
- This continues until the cost reaches infinity. In the mean time, due to going back and forth, it also took several time steps before recognizing this.

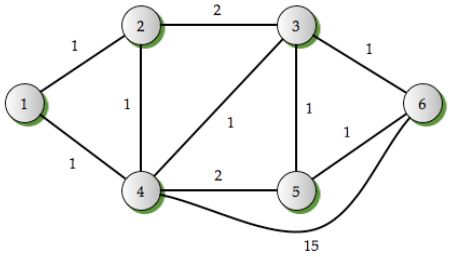
From: http://wiki.uni.lu/secan-lab/Count-To-Infinity+Problem.html

Counting-to -Infinity



- (Not a simple example☺; there are better examples, like the previous one; so skip this one).

time, t_1 —node 4 sends distance vector to node 5.
time, t_2 —links 3-6 and 5-6 both fail.
time, t_3 —node 5 informs node 4 (and node 3) that its cost to node 6 is ∞.
—node 3 informs node 4 (and node 5) that its cost to node 6 is ∞.
time, t_4 —node 4 performs a shortest path computation.
time, t_5 —node 4 receives a distance vector from node 1 indicating that its cost to node 6 is 3 (i.e., node 1 has not learned about the failures yet).
time, t_6 —node 4 performs a shortest path computation again.
time, t_7 —node 4 sends its distance vector to node 5 and node 3 (and others).
time, t_8 —node 3 updates their routing tables based on information from node 4.



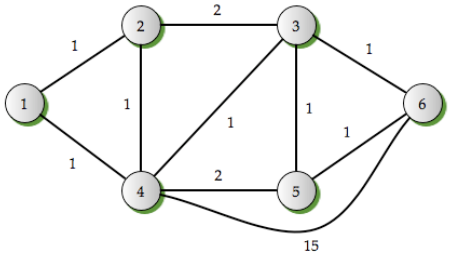
- At time t_1, table at node-4:

Destination Node	Cost	Outgoing Link
1	1	4-1
2	1	4-2
3	1	4-3
4	0	local
5	2	4-5
6	2	4-3

- So, broadcasts the following DV:

$j=1, \overline{D}=1$	$j=2, \overline{D}=1$	$j=3, \overline{D}=1$	$j=4, \overline{D}=0$	$j=5, \overline{D}=2$	$j=6, \overline{D}=2$

- Link failures occurs at time t_2; node 5 sets the cost to node 6 as ∞



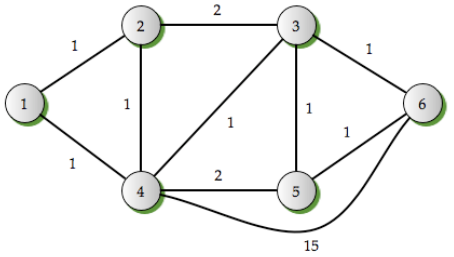
- At time t_3, node 5 broadcasts the following DV to node 4 and node 3:

$j=1, \overline{D}=3$	$j=2, \overline{D}=3$	$j=3, \overline{D}=1$	$j=4, \overline{D}=2$	$j=5, \overline{D}=0$	$j=6, \overline{D}=\infty$

- While node 3 broadcasts the following to node 4 and node 5:

$j=1, \overline{D}=2$	$j=2, \overline{D}=2$	$j=3, \overline{D}=0$	$j=4, \overline{D}=1$	$j=5, \overline{D}=1$	$j=6, \overline{D}=\infty$

- At time t_4, and after receiving the above DV from nodes 3 and 5, node 4 performs shortest path computation



- At node 4, a new DV from node 1 is received at t_5, which reports the best cost to node 6 to be4 3.
- At time t_6, node 4 performs shortest path computation and notices that for destination 6, it has

Destination Node	Current Cost	New Possible Cost	Updated Cost (update?)	Update Outgoing Link (if any)
6	∞	1 + 3	4 (Yes)	4-1

updates the routing table entry for destination 6.

NOW, trouble starts!

- At time t_7, node 4 sends the following DV to nodes 3 and 5:

$j=1, \overline{D}=1$	$j=2, \overline{D}=1$	$j=3, \overline{D}=3$	$j=4, \overline{D}=0$	$j=5, \overline{D}=2$	$j=6, \overline{D}=4$

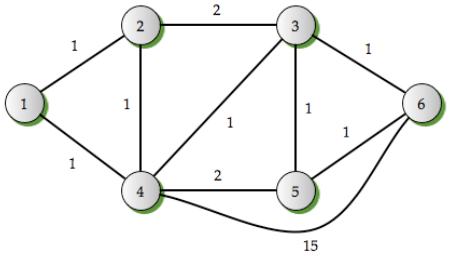
- On receiving this message, node 3 notices that node 6 is reachable via node 4 and the cost is 4
  - Node 3 updates the cost to 5(=1+4).
  - It sends DV to node 4 with this cost

- The following cycle starts:
  - Nod 4 sends DV to node 3 with cost for destination 6 as x
  - Node 3 computes the destination cost to node 6 as x+1
  - Node 3 sends DV to node 4 with cost for destination 6 as x+1
  - Node 4 updates its cost to node 6 as x+2 (=1+x+1)
  - Node 4 sends DV to node 3 with cost for destination node 6 as x+2
  - …
- Continued until cost is more than 15 (infinity).

Split Horizon

- A technique that helps speed up convergence
- Solve the counting to infinity problem in some instances
- Two variations:
  - Split horizon
  - Split horizon with Poisoned reverse
- Basic Idea:
  - When transmitting DV update on an outgoing link, send updates only for nodes for which this link is not on its routing table as outgoing link
    - Alternately, include the cost as *infinity* (Poisoned reverse)

Illustration (Split Horizon)



- Routing table at node 4:

Destination Node	Cost	Outgoing Link
1	1	4-1
2	1	4-2
3	1	4-3
4	0	local
5	2	4-5
6	2	4-3

- The basic DV will advertise the following:

$j=1,\overline{D}=1$	$j=2,\overline{D}=1$	$j=3,\overline{D}=1$	$j=4,\overline{D}=0$	$j=5,\overline{D}=2$	$j=6,\overline{D}=2$

- DV with split horizon will advertise
  - Node 3

$j=1,\overline{D}=1$	$j=2,\overline{D}=1$	$j=4,\overline{D}=0$	$j=5,\overline{D}=2$

  - Node 5

$j=1,\overline{D}=1$	$j=2,\overline{D}=1$	$j=3,\overline{D}=1$	$j=4,\overline{D}=0$	$j=6,\overline{D}=2$

- DV with Split horizon with Poisoned reverse:
  - Node 3:

$j=1,\overline{D}=1$	$j=2,\overline{D}=1$	$j=3,\overline{D}=\infty$	$j=4,\overline{D}=0$	$j=5,\overline{D}=2$	$j=6,\overline{D}=\infty$

Lessons from DV

- Most problems are due to timers
  - When an event (e.g. failure) occurs, should this be informed right away to neighbors *and* compute new shortest path computation?
    - This seems obvious (called "triggered update"), but it wasn't always done right away causing many of the problems.
    - However, you don't want to update always 'right away' either, especially if successive events occurs
      - Inject a 'hold-down' timer
    - Stability of the routing system should be addressed as well

A general question

- Can we avoid loops in a distance vector environment?
  - First understand why/when looping occurs
    - It's a transient behavior when an unusual event occurs
  - Thus, instead of being reactionary after an event, be judicious (sort of check with neighbors first before doing a 'change')
    - Means need to rely on additional info

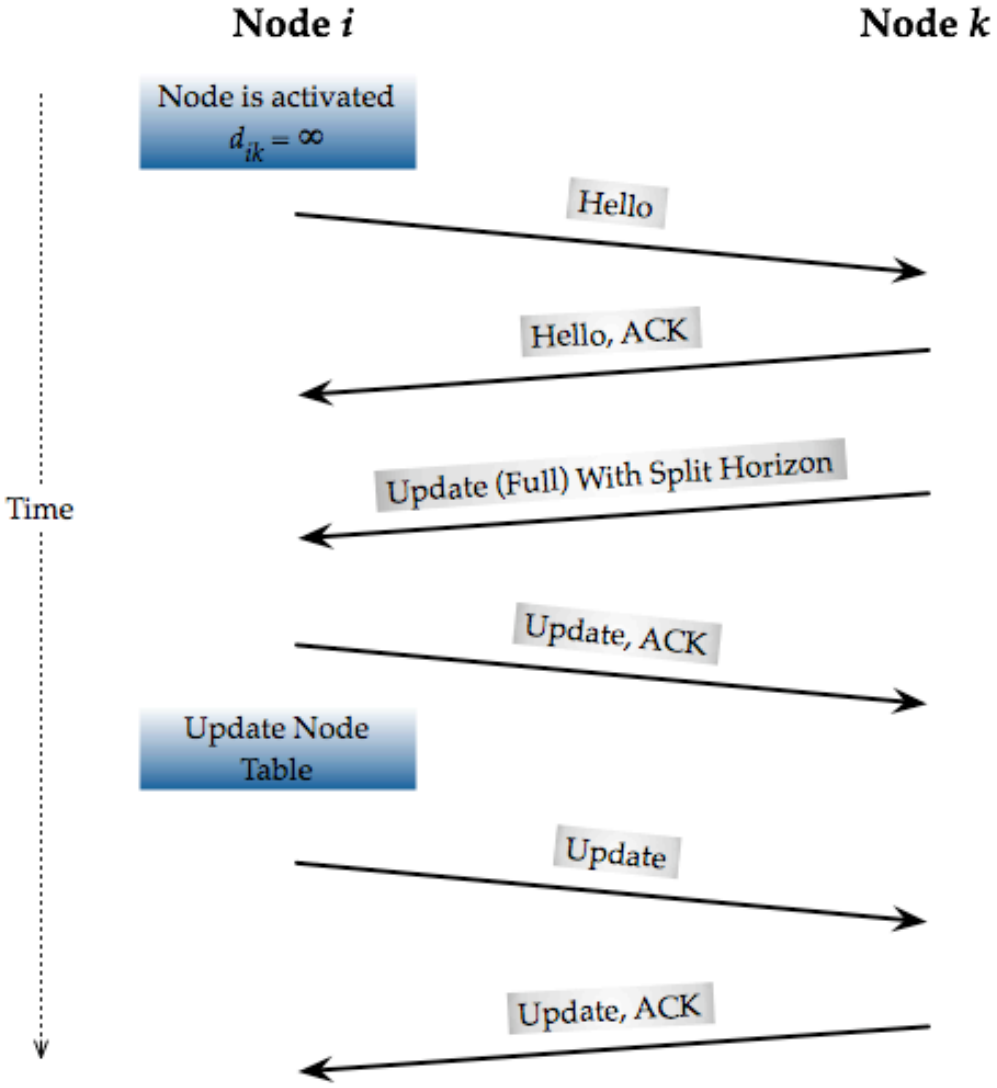
Distance Vector based on Diffusing Computation with Coordinated Update

- Can be labled as loop-free distance vector protocol
- Also known as DUAL (Diffusing Update) Algorithm
- Was initially implemented by Cisco in EIGRP (Enhanced interior Gateway Routing Protocol)

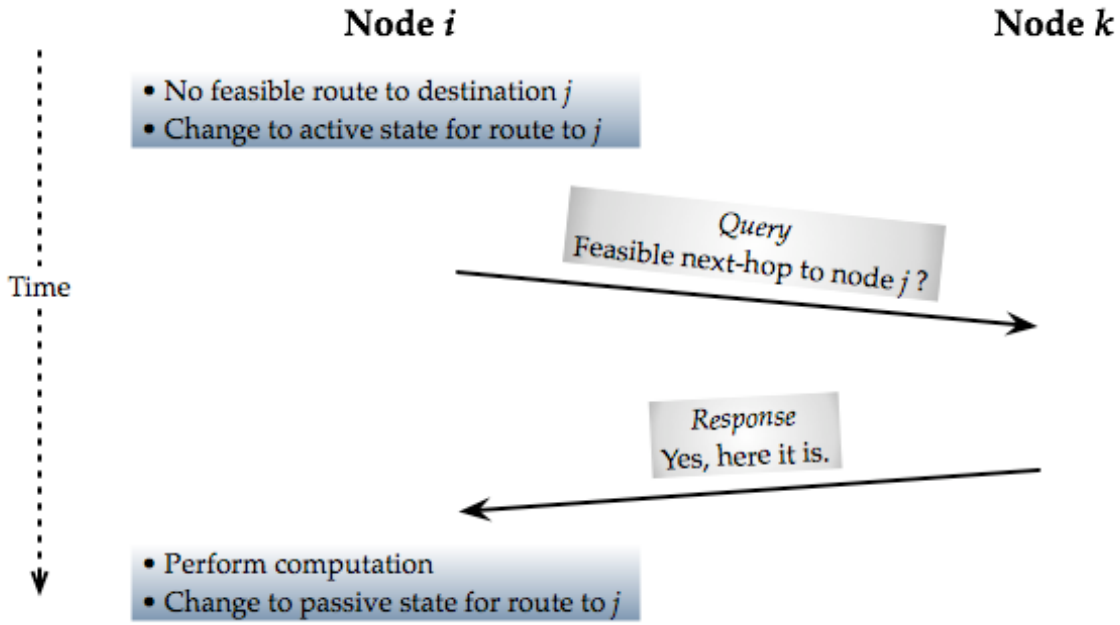
Loop-free DV: basic idea

- Has five message types: hello, ack, query, update, reply
- DV announcement includes message type
  - [Dest Node, Message Type, NextHop, Distance]
- Node state: active, passive

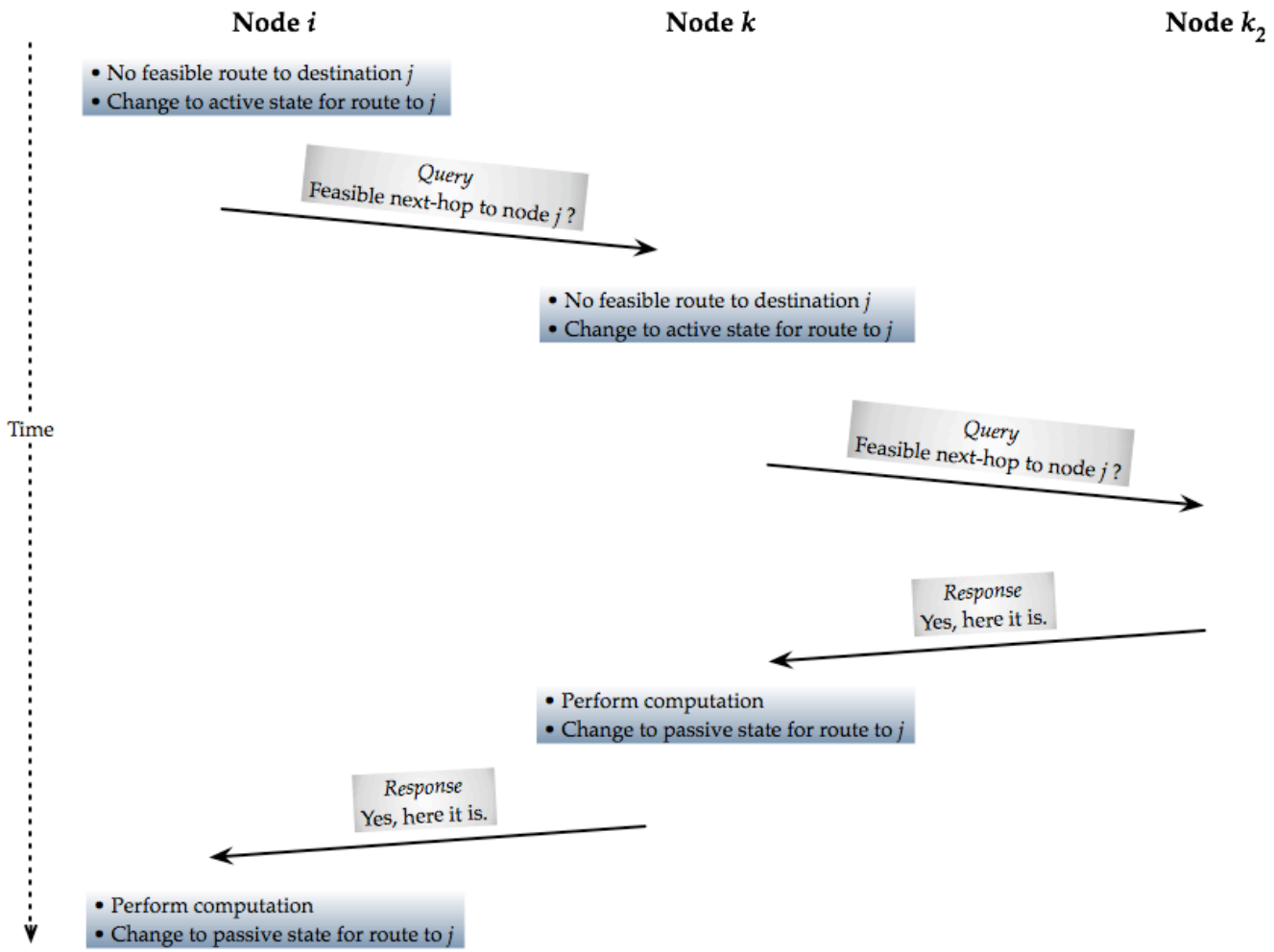
Loop-free DV: when a node is activated



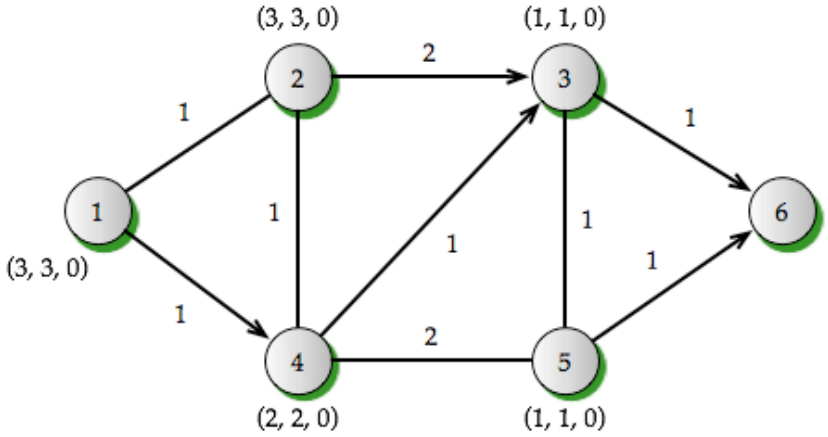
Loop-free DV: Request/Response handling



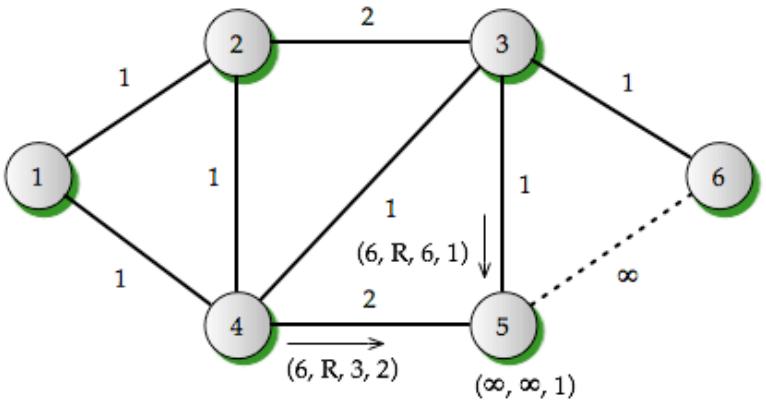
Cont' d


Node i
Node k
Node k2
• No feasible route to destination j
• Change to active state for route to j
Query
Feasible next-hop to node j ?
• No feasible route to destination j
• Change to active state for route to j
Time
Query
Feasible next-hop to node j ?
Response
Yes, here it is.
• Perform computation
• Change to passive state for route to j
Response
Yes, here it is.
• Perform computation
• Change to passive state for route to j

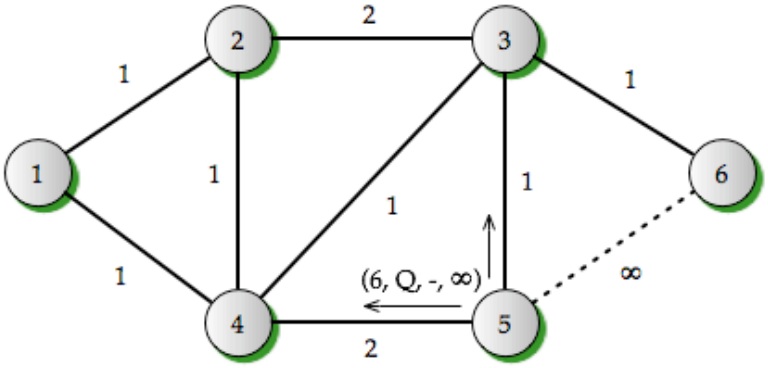
Message := [Dest Node, Message Type, NextHop, Distance]
Node := (distance, working distance, state)



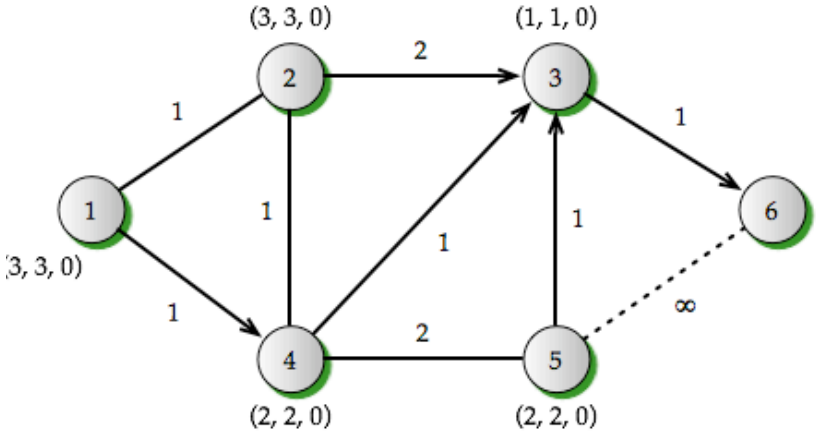
Converged State (before link failure)



Step 2



Step 1



Converged State (after link failure)

Loop-free distance vector protocol

Initialization:

- Node i initializes itself in a passive state with an infinite distance for all its known neighbors and zero distance to itself
- Initiate hello protocol with neighbor k and update value d_{ik}
- Receive update message from all neighbors k, and update node and routing table

Node i in passive mode:

- Node i detects change in a link cost/status to neighbor k that changes $\overline{D}ij$ for some j

  If (this is due to link failure) then $d_{ik} = \infty$, $\overline{D}^i_{kj} = \infty$
- Check for a feasible next hop already in the node table (i.e., a neighbor k in the node table that satisfied $\overline{D}_{kj} < \overline{D}_{ij}$ prior to the link status change)
- If (Node i finds one or more feasible next hops, $k \in \overline{N}_{ij}$) then

  // initiate local computation (much like D-BF)

  If (there is a feasible k such that $d_{ik} + \overline{D}_{kj} < d_{i,H_{ij}} + \overline{D}^i_{H_{i,j},j}$, that is, cost through this k is better than the current next hop H_{ij}) then

  Set k as the new next hop, i.e., $H_{ij} = k$

  If ($\overline{D}^i_{H_{i,j},j} < \overline{D}_{ij}$) then $\overline{D}_{ij} = \overline{D}^i_{H_{i,j},j}$

  Send update, $\widetilde{D}_{ij}$, to all its neighbors

  Endif
- If (Node i cannot find a feasible next hop) then

  Set $\overline{D}_{ij} = \widetilde{D}_{ij} = d_{ih} + \overline{D}^i_{hj}$ where h is the current next hop for node j

  Set feasible distance to D^i_{hj}

  // initiate diffusing computation

  Change state to node j to active state, freeze any change for destination j, and set action flag to 1

  Send query to all neighbors $k \in \mathcal{N}_i$

  Endif

Node i in receiving mode:

Node i in passive mode:

Receive update message:

Update node table, determine new next hop, routing table, and update message

Receive Query:

If (feasible next hop exists) respond with $\widetilde{D}_{ij}$

If ($\overline{D}_{ij}$ changes) send update to neighbors

If (no feasible next hop exists)

change to active mode, $R = 1$ and send query to other neighbors

Node i in active mode:

If (response received from neighbors k to all queries) then

Change to passive mode ($R = 0$)

Reset $\widetilde{D}_{ij}$

Update node table, determine new next hop, update routing table

Endif

Loop-free DV: summary

- Avoids loop
- Introduces states at each node
- Issues:
  - Chatty mode when a failure occurs
    - Can generate a lot of chit-chat messages
      - Problematic in a low-capacity link as such messages consume a lot of bandwidth
  - Stuck in Active (SIA) condition
    - Possible in a multi-event situation
    - A node remains stuck in Active even after holddown timer expires – a difficult problem to resolve

Link State Routing Protocol

- Simple view:
  - Each node generates cost of its outgoing links
  - This information is communicated to other nodes
    - Communication can be "in-band" – typically through a mechanism known as flooding
      - Commonly used in Data Networks such as Internet
      - Specifics for particular protocols such as OSPF or IS-IS will be discussed later
    - Communication can be "out-of-band" – typically through a dedicated circuit or another communication medium
      - Common for routing update in PSTN

Link state protocol & Dijkstra's algorithm

- Often there's confusion/misinterpretation that they are the same – they aren't
- LSP talks about how information about links ("link cost") are communicated
- Dijkstra's algorithm discusses how to compute shortest paths if link costs are known
- Often, a LSP protocol uses Dijkstra's algorithm at each node to compute shortest paths
- However, it's *not necessary* for a LSP to use Dijkstra's algorithm to compute paths
- Finally, link information is not limited to just one type of information; it may contain multiple attributes: cost, bandwidth, delay, and so on

Link state protocol (with flooding)

- Flooding is accomplished through hop-by-hop dissemination
  - The originating node transmits the link state of its outgoing links to its neighboring nodes
    - Known as 'link state advertisement'
  - The neighboring nodes, in turn, forward this link state information to their neighbors, and so on
  - This way, after some time, all nodes in the network find out the link state information of various links

What's in a link state advertisement (LSA)

- An LSA contains the following:

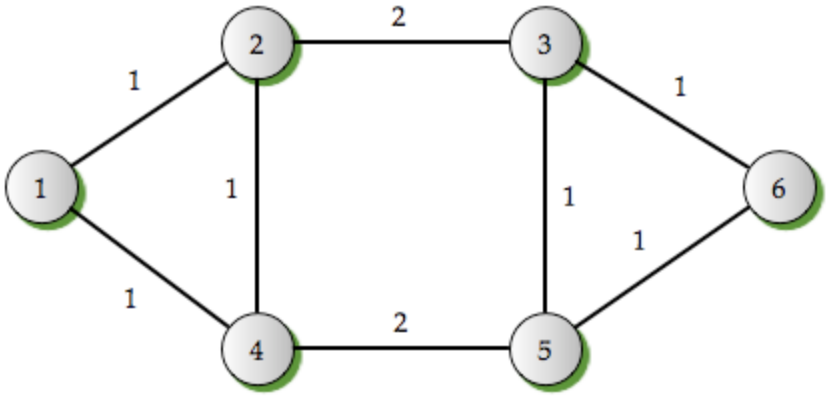
Source node, Link ID, Link Cost

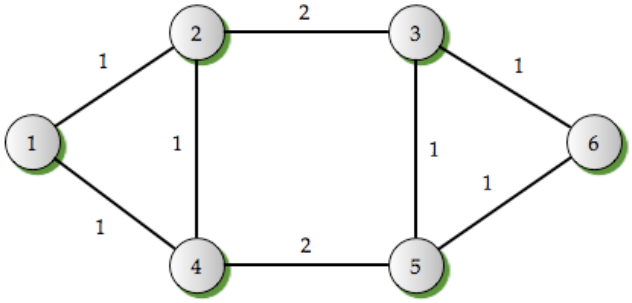
- Example:

$i = 1$, Link=$1 \rightarrow 2$, $d_{12} = 1$

  - Simply as:

$1, 1 \rightarrow 2, 1$





- There's a problem with previous representation!
- If the cost of a link changes (e.g., due to failure), how does a receiving node know that it is receiving the latest information?
- Consider

time t_0: LSA $\boxed{1, 1\rightarrow 2, 1}$ is generated at node 1 and is sent to node 2 and node 4.
time t_1: LSA $\boxed{1, 1\rightarrow 2, 1}$ is forwarded by node 2 to node 4.
time t_2: $1\rightarrow 2$ fails; node 1 generates the new LSA $\boxed{1, 1\rightarrow 2, \infty}$ to node 4.
time t_3: LSA $\boxed{1, 1\rightarrow 2, 1}$ is received at node 4 from node 2.

LSA format: issues

- Need basic information

  Source node, Link ID, Link Cost

- The above can't resolve timing; so add time stamping

  Source Node, Link ID, Link Cost, Time stamp

- Typically, done using a sequence number

  Source Node, Link ID, Link Cost, Sequence Number

- However, sequence number comes from a finite-bit space; so aging is added for safeguard

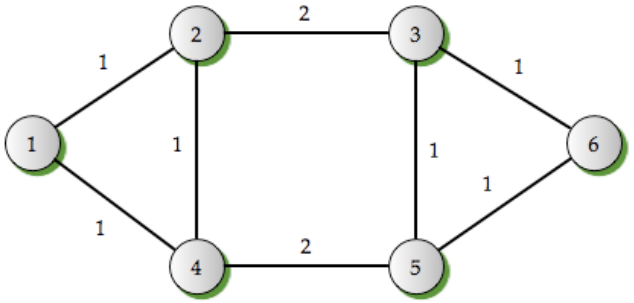
  Source Node, Link ID, Link Cost, Sequence Number, Age

A few things about LSAs

- Sequence number problem can be addressed by a concept call lollypop sequence numbering (see p. 87)
- LSAs for different links can be combined to send a Link State Update (LSU)

1, 1→4, 1, 1, 59	2, 2→4, 1, 1, 59	4, 4→5, 2, 1, 60

(last field: age counter)



Workings of Link State Protocol

- Need to address how to handle special cases
  - A node/link going down; or coming up
  - How to handle sequence number (when it comes up)

LSP: three subprotocols

- Hello protocol
  - When a node is first activated
  - Also periodically invoked to check if the link to neighbor is operational
  - It can have both information push or pull
- Resynchronization protocol
  - For use after recovery from a link or a node failure
    - Link-state information synchronization
- Link state advertisement (normal)
  - The basic flooding mechanism discussed above
  - Each node maintains a link state database
- Details of its working in Figure 3.11 (next slide)

Initialization:
- Use hello protocol to establish neighbor relation and learn about links from neighbors

Link State Advertisement (normal):
- Generate LSA periodically (before the expiration of the age counter), incrementing the sequence number from the last time, and set the age field to the maximum value for this LSA; start the timer on the age field

Receive (new):
- Receive LSA about a new link $l \rightarrow m$ from neighboring node k
- Update the link state database and send LSA for this link to all neighbors (except k)
- Start the timer for decrementing the age counter

Receive (normal):
- Receive LSA about known link $l \rightarrow m$ from neighboring node k
- Compare the sequence number to check if I have the most recent LSA for this link
- If yes, send LSA for this link back to neighboring node k
- If not, decrement the age counter, store the recent LSA for link $l \rightarrow m$ in the link state database, and forward it on all other outgoing links to the rest of the neighbors (except for k)
- If it is the same sequence number and the age difference is small, do nothing; if the age difference is large, store the most recent record
- Start the timer for decrementing the age counter

Compute:
- Perform route computation using the local copy of the link state database
- Update the routing table

Special Cases:
- Link failure: set the link cost to ∞ and send LSA immediately to neighbors
- Link recovery: perform link state database resynchronization
- Node recovery: perform link state database resynchronization; alternately, flush records and perform hello protocol
- (Action mode when the age counter reaches 0 for a link ID):
  (1) Do not consider this link in route computation
  (2) Inform neighbors through advertisement stating that the age field is zero
  (3) Delete this link entry from the database
- (Receive mode with age 0): accept this LSA. This would set the age to zero, and thus, perform 'action mode when age counter reaches 0.' If the record is already deleted, ignore this advertisement.

FIGURE 3.11 Link state protocol from the point of view of node i (with in-band hop-by-hop communication for flooding).

Failure (Link 4-5 already failed)

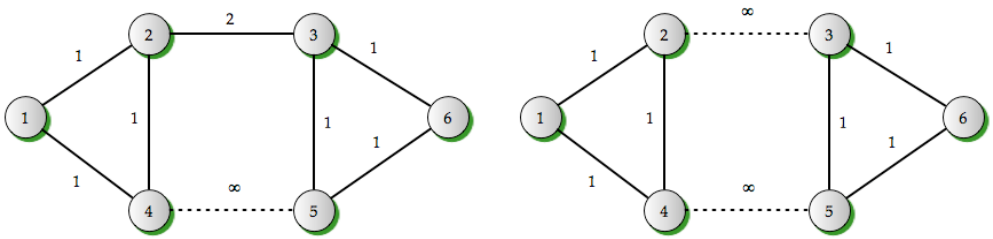


TABLE 3.3 Link state database as viewed before and after failure of link 2-3.

(a) Before Failure of Link 2-3 (as Seen by Every Node)			(b) After Failure of Link 2-3 (as Seen by Nodes 1, 2, 4)			(c) After Failure of Link 2-3 (as Seen by 3, 5, 6)		
Link-ID	Cost	Seq #	Link-ID	Cost	Seq #	Link-ID	Cost	Seq #
1→2	1	1	1→2	1	1	1→2	1	1
2→1	1	1	2→1	1	1	2→1	1	1
1→4	1	1	1→4	1	1	1→4	1	1
4→1	1	1	4→1	1	1	4→1	1	1
2→4	1	1	2→4	1	1	2→4	1	1
4→2	1	1	4→2	1	1	4→2	1	1
2→3	2	1	2→3	∞	2	2→3	2	1
3→2	2	1	3→2	2	1	3→2	∞	2
4→5	∞	2	4→5	∞	2	4→5	∞	2
5→4	∞	2	5→4	∞	2	5→4	∞	2
3→6	1	1	3→6	1	1	3→6	1	1
6→3	1	1	6→3	1	1	6→3	1	1
3→5	1	1	3→5	1	1	3→5	1	1
5→3	1	1	5→3	1	1	5→3	1	1
5→6	1	1	5→6	1	1	5→6	1	1
6→5	1	1	6→5	1	1	6→5	1	1

- After the second failure, each isolated will evolve over time (cost change, sequence number etc)
- Now consider link 4-5 is recovered
  - Node 4 will generate LSA for 4->5
  - Node 5 will generate LSA for 5->4
  - Seems that should be enough
- Well, Node 4 and 5 need to do more: exchange info about all the other links in their database so that they can be synchronized
  - Age field become important since it can be used to mark a link as 'stale' when it expires and will be deleted from the link state database
  - So when new information exchange, it' ll re-learn about the links that were purged from the database

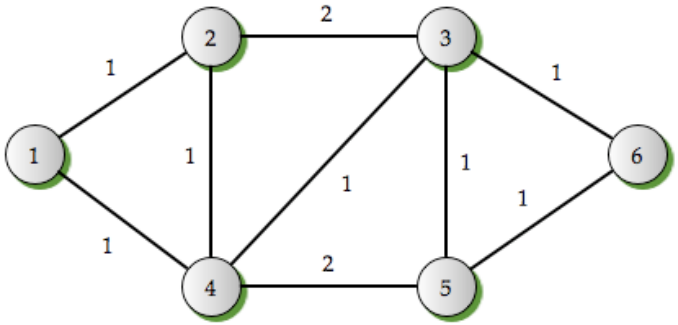
Link state protocol: recap

- In-band hop-by-hop communication accomplished through flooding
- Because flooding, a node may find out about a link's cost from two different neighbors
  - Need to then address how to find out who is providing the most recent information
  - Accomplished through sequence number field
  - Also, age field is added for safeguard

Path Vector Routing Protocol

- This protocol is similar to a distance vector protocol
- The main difference is that node *j*, instead of announcing just the distance to a destination *j* (D_{kj}) to its neighboring node *i* as in a distance vector protocol, in this protocol, node *k* will announce the entire path to a destination P_{kj} to node *i*
  - Advantage is that it helps to avoid looping

- Node 3:

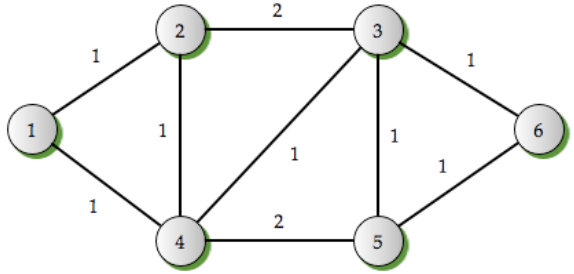


$$\overline{D}_{33} = 0, \quad \mathcal{P}_{33}^2 \equiv (3)$$
$$\overline{D}_{34} = 1, \quad \mathcal{P}_{34}^2 \equiv (3, 4)$$
$$\overline{D}_{35} = 1, \quad \mathcal{P}_{35}^2 \equiv (3, 5)$$
$$\overline{D}_{36} = 1, \quad \mathcal{P}_{36}^2 \equiv (3, 6).$$

Destination Node, Cost, Number of Nodes in the Path; Node List of the Path | ...

$j = 3, \overline{D}_{33} = 0$, Number of Nodes = 1; $\mathcal{P}_{33}^2 \equiv (3)$

3, 0, 1; 3 | 4, 1, 2; 3, 4 | 5, 1, 2; 3, 5 | 6, 1, 2; 3, 6



- Initial path table at node 2:

Destination	Cost	Path
1	1	(2, 1)
4	1	(2, 4)

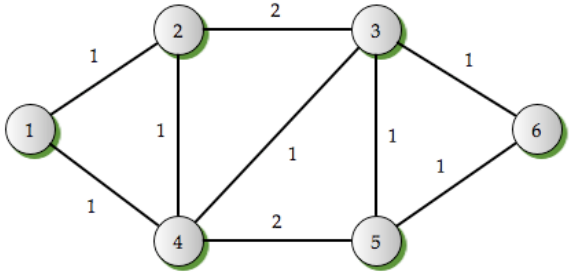
- On receiving path vector form node 3, till update to:

Destination	Cost	Path
1	1	(2, 1)
3	2	(2, 3)
4	1	(2, 4)
5	3	(2, 3, 5)
6	3	(2, 3, 6)

- Node 4 sends PV:

1, 1, 2; 2, 1 | 2, 0, 1; 2 | 3, 2, 2; 2, 3 | 4, 1, 2; 2, 4 | 5, 3, 3; 2, 3, 5 | 6, 3, 3; 2, 3, 6

PV: Link failure



- Consider path table at each source just for destination 6:

From Node	To Destination	Cost	Path Table Entry
1	6	3	(1, 4, 3, 6)
2	6	3	(2, 3, 6)
3	6	1	(3, 6)
4	6	2	(4, 3, 6)
5	6	1	(5, 6)

- Now link 3-6 fails.
- Node 3 generates an [illegible] neighbors regarding node 6
- On receiving it, node 2 will inform nodes 1n and about unreachability
- In the mean time, node 5, on receiving unreachable message from node 3, will realize the need to inform node 3 that it can reach 6.

[cont'd]

- Now, node 3 generate a NEW message to let its neighbors know
- On receiving this new message, node 2 will update its entry to node 6 as:

Destination	Cost	Path Table Entry
6	4	(2, 3, 5, 6)

- Node 4 will its entry to node 6 as:

Destination	Cost	Path Table Entry
6	3	(4, 3, 5, 6)

Initialization:

- Node i is activated and exchanges "hello" message with neighbors; table exchange updates are performed to find paths for all known destination nodes j

Node i in the receiving mode:

Announcement / Advertisement Message:

- Receive a path vector $\mathcal{P}_{kj}^{i}$ from neighbor k regarding destination node j
- If (destination node j is not previously in routing and path table)
  create a new entry and continue
- If (for destination node j there is already an entry for the previously announced path from node k in the path table)
  Replace the old path by the new path $\mathcal{P}_{kj}^{i}$ in the path table
- Update candidate path cost: $\widetilde{D}_{ij} = d_{ik} + \overline{D}_{kj}^{i}$
  If ($\widetilde{D}_{ij} < \overline{D}_{ij}$) then
  Mark best path as $i \rightarrow \mathcal{P}_{kj}^{i}$
  Update the routing table entry for destination j
  else
  For destination j, identify neighbor $\overline{k}$ that results in minimum cost
  over all neighbors, i.e., $\overline{D}_{ij} = d_{i\overline{k}} + \overline{D}_{\overline{k}j}^{i} = \min_{k \in \mathcal{N}_i}\{d_{ik} + \overline{D}_{kj}^{i}\}$
  Mark the best path through this new neighbor as $i \rightarrow \mathcal{P}_{\overline{k}j}^{i}$
  Update the routing table entry for destination j
  Node i in *sending* mode: send the new best path to all its neighbors
  Endif

Withdrawal Message:

- If (a withdrawal message is received from a neighbor for a particular destination j) then
  Mark the corresponding entry as "unreachable"
  If (there is a next best path in the path table for this destination)
  Advertise this to the rest of the neighbors
  Endif

Special Cases: (node i lost communication with neighbor k ["link failure"]):

- For those destinations j for which the best path was through k, mark the path as "unreachable" in the path table (and also routing table)
- If (another path is available in cache for the same destination through another neighbor k')
  Advertise/announce this to all the remaining neighbors
- If (there is no other path available in cache for destination j)
  Send a message to all neighbors that path to node j is "withdrawn"

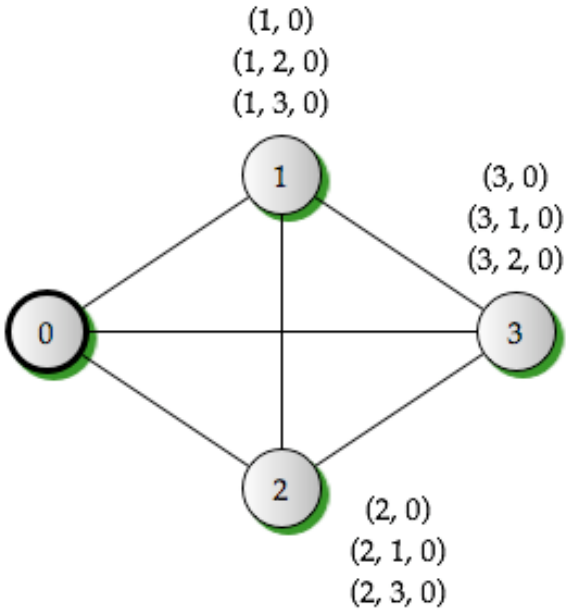
FIGURE 3.15 Path vector protocol with path caching (node i's view).

PV: failure, with path caching

- The previous example assumes that a node maintains only the one (best) path to a destination
- Suppose it does store additional paths (say, non-shortest) to a destination that it has learned from other neighboring nodes
- Idea would be that if there's a failure, it can quickly switch to the next one in the cache
  - It's a good idea; but not always!

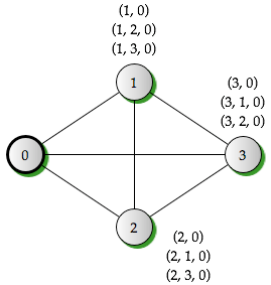
LV: with node failure (in the presence of path caching)

- Consider the following topology, with paths cached at each node noted:



[Example: due to Jen Rexford]

Node 0 goes down



- Node 2 sees that path (2, 0) is no longer available and crosses it off. By inspecting paths cached in its routing table, it then switches to (2, 1, 0), which is least as the next preferred path.

  Node 2 informs both node 1 and node 3 about withdrawal of path (2, 0).
  - [sidenote: assume that AS 2 has not heard any route withdrawal from AS 1 and AS 3 yet]

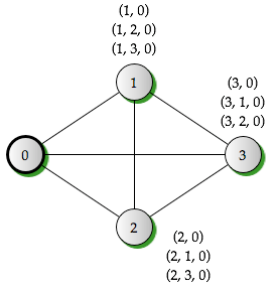
- Node 1 recognizes that path (1, 0) is no longer available and crosses it off.

  Node 1 switches to its next preferred path (1, 2, 0).

  Node 1 receives withdrawal of path (2, 0) from node 2, before it has time to inform others about (1, 0).

  Node 1 switches to (1, 3, 0) and advertises this path to node 2.
  - [sidenote: No advertisement of (1,3,0) from AS 1 to AS 3]

[cont'd]



- Node 2 receives the advertisement (1, 3, 0) from node 1 and recognizes that the preferred path of node 1 to node 0 is *no* longer (1, 0); thus, node 1 strikes off (2, 1, 0).

  Node 2 compares the last path in its preferred list (2, 3, 0) to the newly advertised path (1, 3, 0) received from node 1, i.e., compare (2, 3, 0) with (2, 1, 3, 0), and switches to (2, 3, 0) since this is preferred over (2, 1, 3, 0).

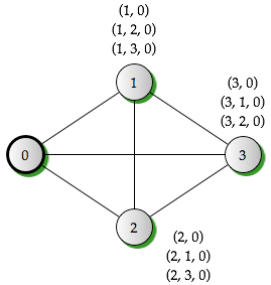
- Node 3 recognizes that path (3, 0) is no longer available and crosses it off.

  Node 3 receives withdrawal of path (1, 0) from node 1 and withdrawal of path (2, 0) from node 2.

  Node 3 thus realizes that (3, 1, 0) and (3, 2, 0) are no longer available and crosses them off.

  Node 3 informs node 1 and node 2 that path (3, 0) is no longer available.

[cont'd]



- Upon receiving withdrawal of path (3, 0) from node 3, node 2 realizes that path (2, 3, 0) is no longer available, thus, it switches to (2, 1, 3, 0), since this is the only path remaining in its table.

- Upon receiving withdrawal of path (3, 0) from node 3, node 1 realizes that (1, 3, 0) is no longer available and thus inform node 2 that path (1, 3, 0) is no longer available.

  - [Sidenote: (1,3,0) is now dead (after receiving the withdrawal from AS 3) and withdraws it from AS 2

- Upon receiving withdrawal of path (1, 3, 0) from node 3, node 2 finally realizes that it no longer has any path to node 0.
  - [sidenote: AS 2 no longer has any route]

PV: comment

- Path vector described so far is generic
  - Considered both caching and without caching
  - Cost not necessarily limited to hop-count
- An important PV protocol is BGP-4
  - We'll cover this later
  - Uses path caching
  - Cost is limited to hop-count (but a hop may be repeated in the PV announcement to increase path cost)
- Node failure
  - Sounds like a rare case
    - Not so; BGP-4 faces this situation often since the node-0 represents an address block – more about this later

ARPANET routing & metrics

- Good to know for historic significance
- "Old" ARPAnet routing:
  - Distance vector protocol (in fact, in it's form, ARPAnet folks invented DV protocol)
  - Link metric based on queue length
    - Packets going from one short queue to another
    - Routing oscillations
    - Looping problems
    - Link cost didn't consider link speed
- Late 1970s: Link metric computation changed
  - First one
    - Delay measured every past 10 sec; report any cost change; report anyway every 50 sec
      - Is delay by itself a good indicateor
    - Works well for low to moderate load, not for highly loaded links
  - Second one: "revised" metric
    - Measured delay is mapped to utilization (see Eq. (3.6.2))
    - Utilization is smoothed over two time window

Summary

- Discussed three classes of routing protocols
  - Distance Vector, Link State, Path Vector
    - Difficulty of running a distributed protocol, looping issues, convergence properties, and so on
  - Their relation to routing algorithms
- Do not discuss how link cost ("metrics") is determined – it's assumed to be given.
- A brief review of ARPA routing metrics
  - Other metrics to be discussed later