CSC 525:
Computer Networks

Intra-Domain Routing

- Objective
  - Compute the lowest-cost path to every destination within a network.
- Factors
  - Topology, delay, bandwidth, traffic load, policy
- Performance Metrics
  - Convergence time and loop-freedom
  - Router memory and routing messages
- Two major types:
  - Distance vector
  - Link state

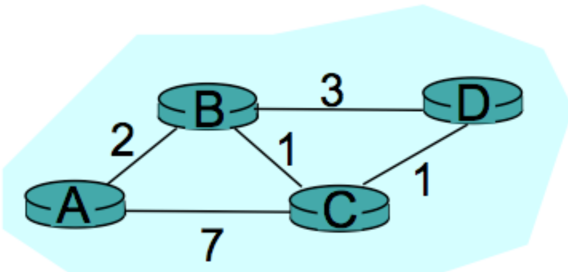
Basic Distance Vector Routing

- Based on Distributed Bellman-Ford (DBF) Algorithm
- Each node maintains one routing table, which lists
  - Destination, Cost, Next-Hop
  - Doesn’t know the entire path to the destination, doesn’t remember alternative paths, etc.
    - To save memory and processing
- Periodically advertise the routing table (i.e., sending out routing messages) to neighbors (i.e., directly connected routers)
- Update the routing table upon receiving neighbor’s messages

Basic Distance Vector Routing

- For node i, upon receiving a routing message [dest,cost(j,dest)] from node j, it updates the routing table as follows:
  - If NextHop(i,dest) = j then
    cost(i,dest) = cost(i,j) + cost(j, dest)
  - Else if cost(i,j) + cost(j,dest) < cost(i,dest) then
    cost(i,dest) = cost(i,j) + cost(j,dest)
    NextHop(i,dest) = j
  - Else do nothing

Initialization



- Each router is configured with its neighbors and the cost of direct links
- Don't know about other destinations, i.e., distance is infinity.

Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

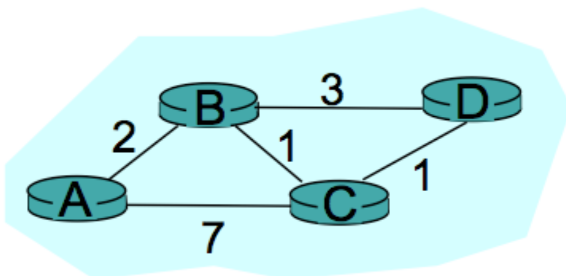
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

First Iteration (C → A)



- Node A receives node C' s routing table and updates its own accordingly.

Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	8	

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A, D) = D(A, C) + D(C,D) = 7 + 1 = 8

(D(C,A), D(C,B), D(C,D))

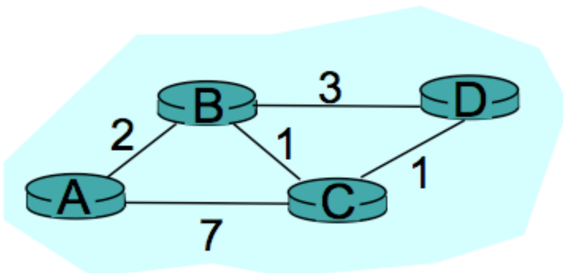
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

First Iteration (B → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A,D) = D(A,B) + D(B,D) = 2 + 3 = 5

D(A,C) = D(A,B) + D(B,C) = 2 + 1 = 3

Node C

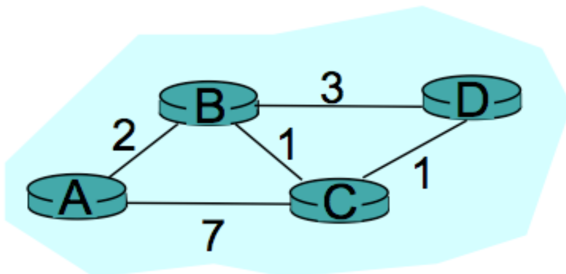
Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

- Node A receives node B's routing table and updates its own accordingly.

End of First Iteration



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

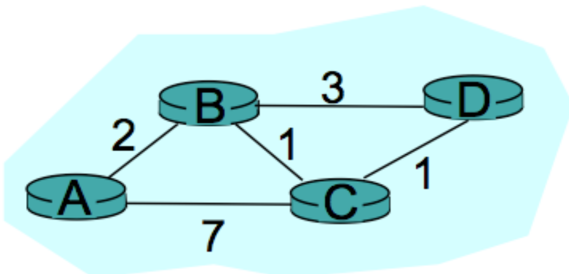
Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	5	B
B	3	B
C	1	C

End of Second Iteration



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

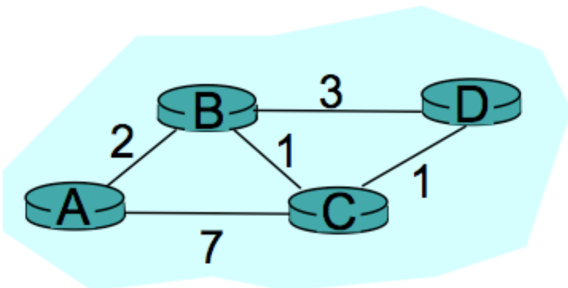
Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	5	B
B	3	B
C	1	C

End of Third Iteration



- Nothing changes after the updates: the routing algorithm has converged.

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

Node C

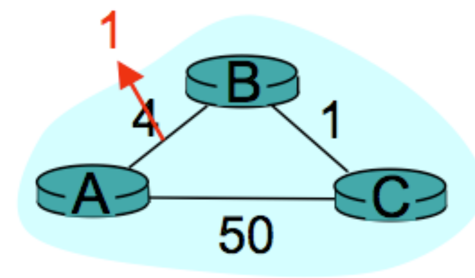
Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

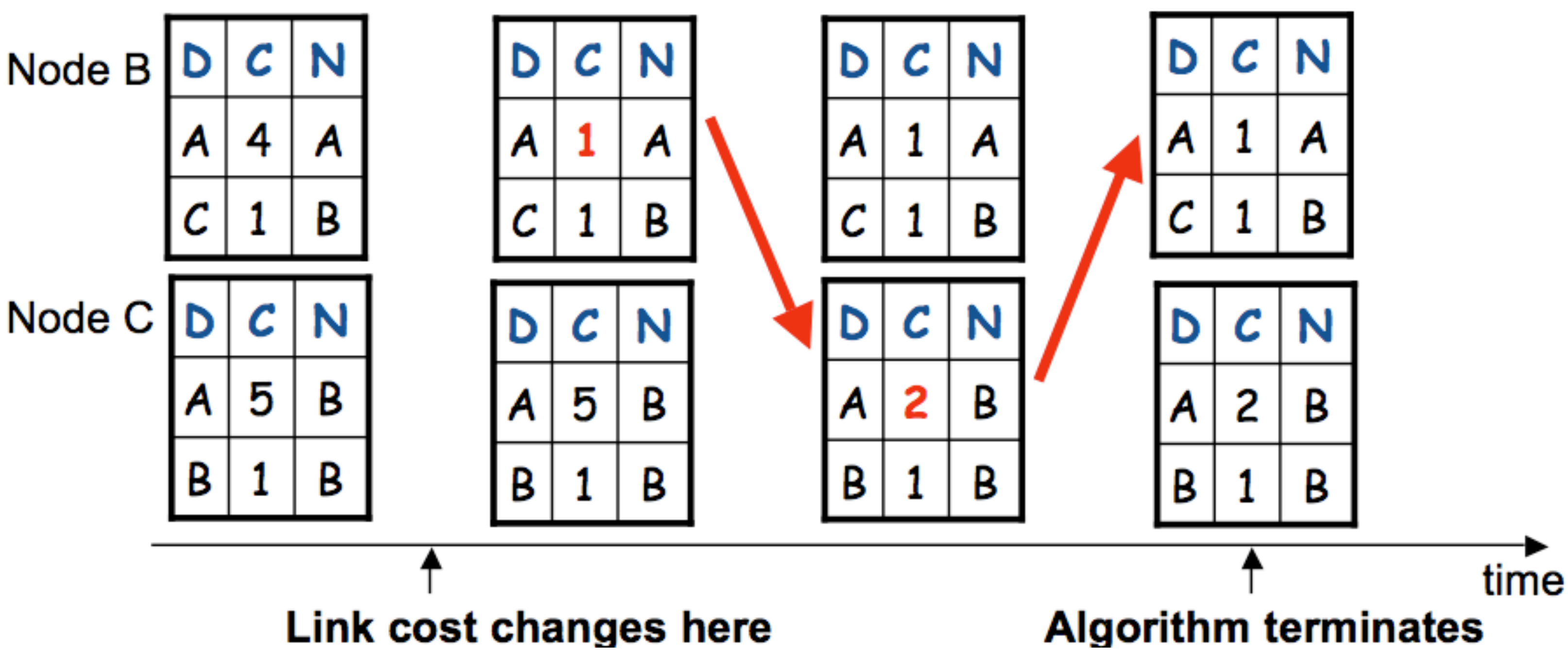
Node D

Dest.	Cost	NextHop
A	5	B
B	3	B
C	1	C

Link Changes

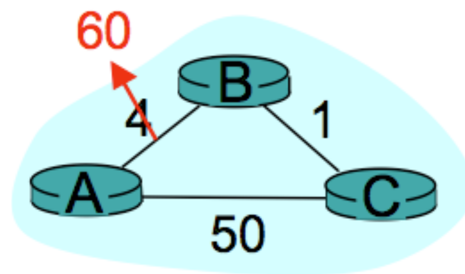
- Good news travels fast

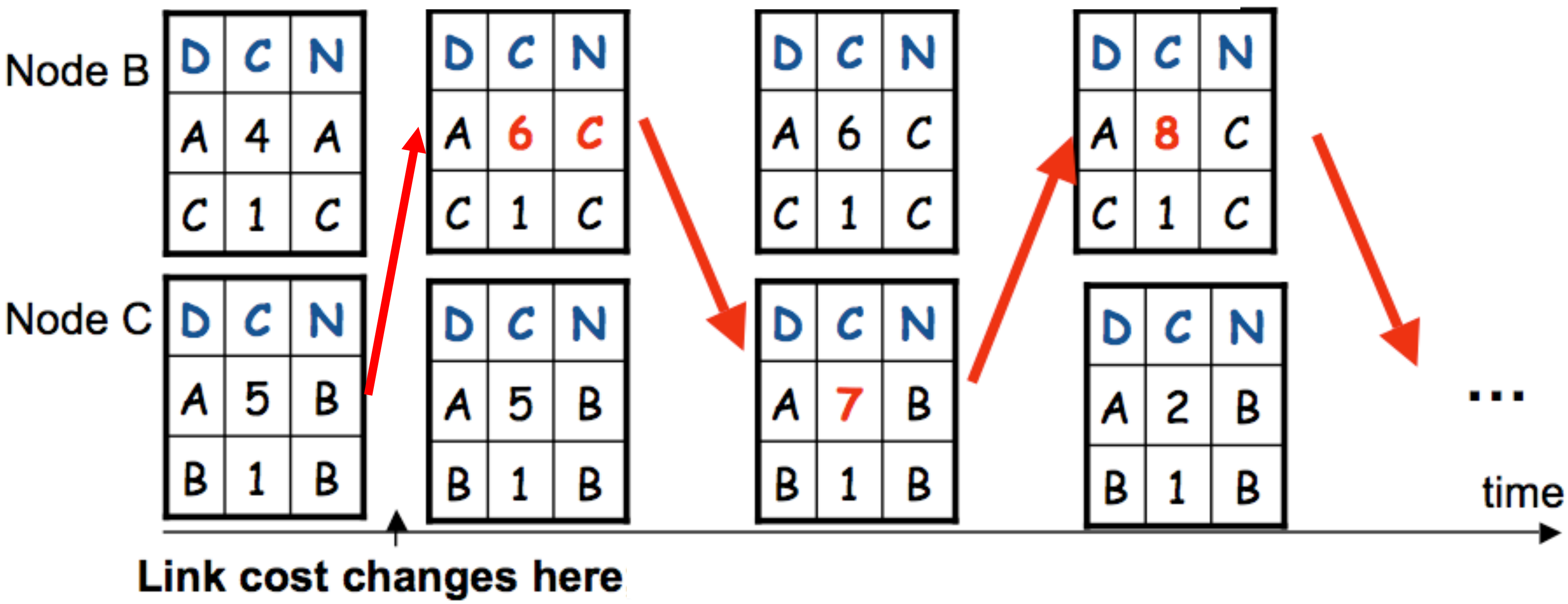




The Bouncing Effect

- Bad news travels slowly
  - a two-node loop





Count-to-infinity

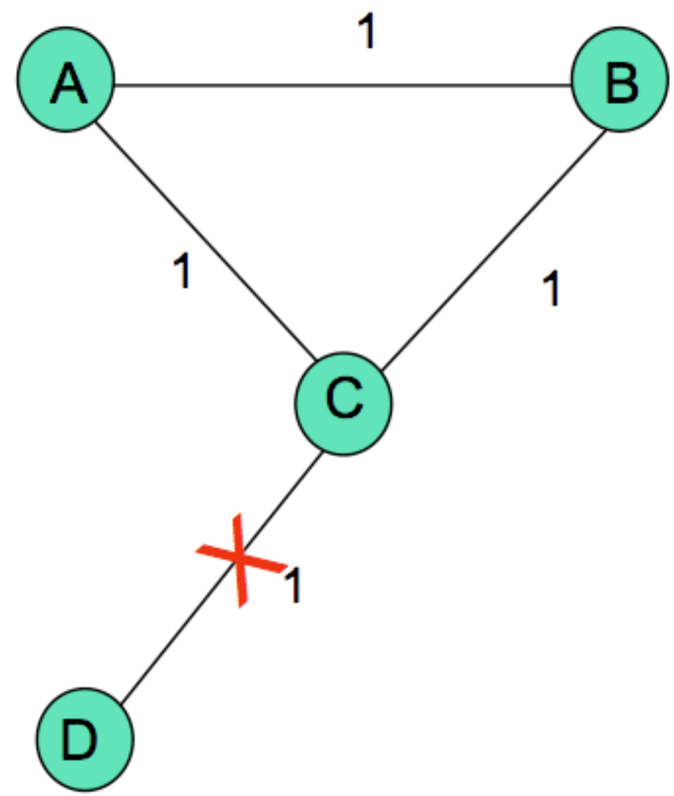
- If both link A-B and link A-C fail, the two-node loop between B and C will keep going
  - Until the upper limit of link cost is reached
- Long convergence time
- Routing loop may cause congestion, which in turn may cause the loss of routing messages, further delaying the convergence.

Split Horizon and Poison Reverse

- Split Horizon:
  - If C uses B as the next hop to reach A, C will **not** announce its route to B.
- Poison Reverse
  - C announces to B that its cost to reach A is infinity
- Simple, but only eliminate *two-node* loops, not loops with more than 2 nodes.

Example of 3-node loop

- C marks D unreachable and reports to A and B.
- Assume A learns it first
  - A now thinks the best path to D is through B
  - A reports infinity to B
  - A reports a route of cost = 3 to C
- C thinks D is reachable through A at cost = 4, and reports to B.
- B reports cost=5 to A
- …



RIP

- Use hop count as link cost, max is 16 (=infinity)
- Soft state: send updates every 30 seconds
  - Even if nothing has changed
- Time-Out Stale Routes
  - If 90 seconds elapse no update for NextHop(i,dest)

    cost(i,dest) = 16

    NextHop(i,dest) = none
- Triggered Updates
  - If route changes, send update immediately
  - But also ensure at least 5 second interval between triggered updates

IGRP/EIGRP

- Cisco's proprietary implementation of distance vector routing
- Use composite metrics for link cost
  - Delay, bandwidth, reliability, load
  - Tunable parameters
- Loop avoidance
  - Path hold-down
  - Route poisoning (poison reverse)

Loop-Free Path-Finding Algorithm

- Eliminate routing loops in distance vector
  - Even the temporary loops caused by message delay
- Much more overhead and much more complex protocol than the basic distance vector routing

Tables

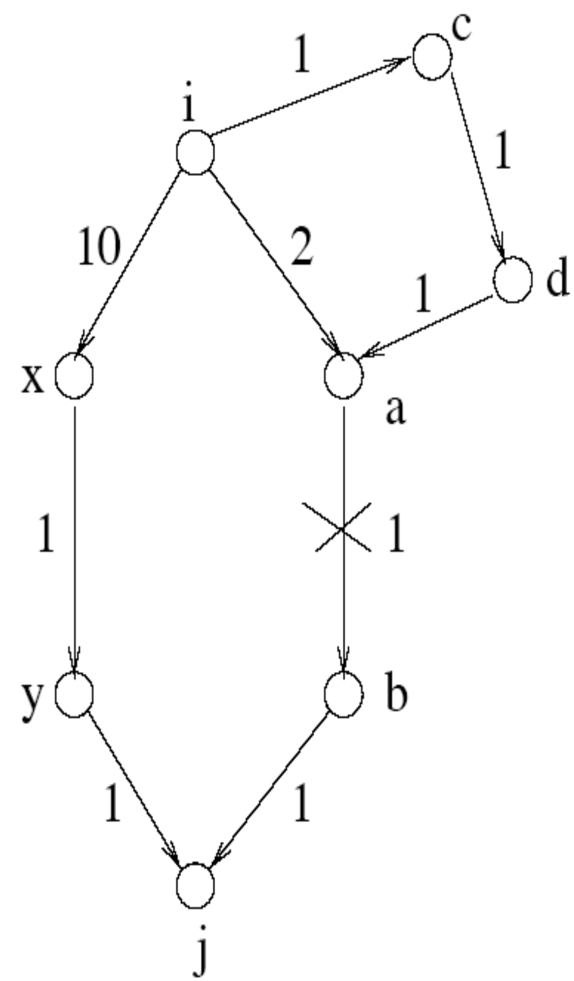
- Each router maintains three tables
  - Link cost table
    - link, cost
  - Routing table
    - dest, cost, successor (next hop)
    - and predecessor (next-to-last hop), marker
    - Can derive the entire path
  - Distance table
    - dest, neighbor, cost, predecessor
    - Can derive alternate paths

Using Path Information

- The entire path can be derived recursively using predecessor information.
  - A … C D
  - A … B C
  - A B
  - Thus [A B C D]
- Use the entire path to detect loops
  - If A uses B as the next hop to reach D, check B's path to D, and if B's path contains A, then there is a loop.

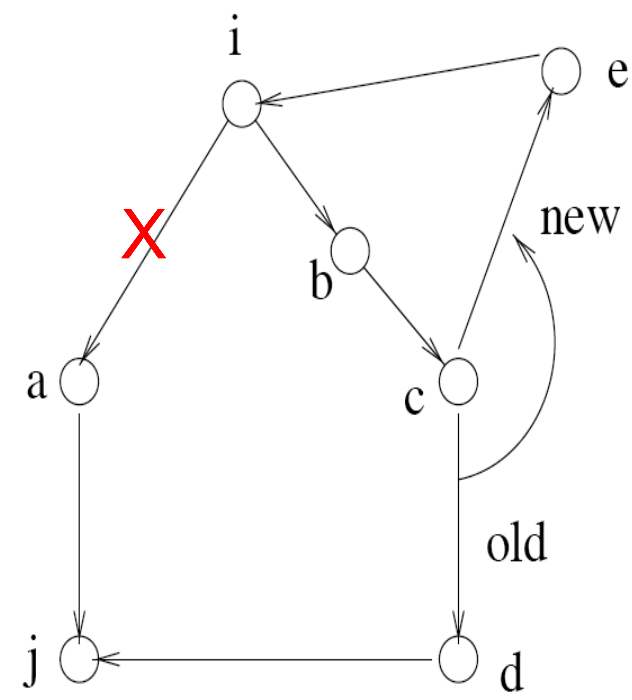
Updating Distance Table

- Make path information from *all* neighbors consistent with the latest update
  - update both best path and alternative paths
- When link *a-b* fails and node i receives an update from *a* stating *abj* is gone, node i updates its distance table entry for [a, j] **and** entry for [c, j], even there's no update from c yet.



Temporary Loops

- Temporary loops due to delay of routing messages
- Solution:
  - Find out whether a loop is possible before adopting a new successor.
    - How?
  - If a loop is possible, synchronize with neighbors to make sure loop will not happen before making any changes.
    - How?



Detecting Potential Looping

- *Feasible distance* is the minimum distance to the destination during recent time.
- *Feasibility Condition*:

  To use neighbor E as the successor to destination j

  - E must have the minimum distance among all neighbors, **and**
  - *E's distance is less than the current feasible distance*.
- If this condition is met, it's guaranteed no loop, i.e.., safe to switch path to E. Otherwise, loops are possible.
- Upon receiving an update, check feasibility condition. If satisfied, adopt the new successor, otherwise synchronize with neighbors first.

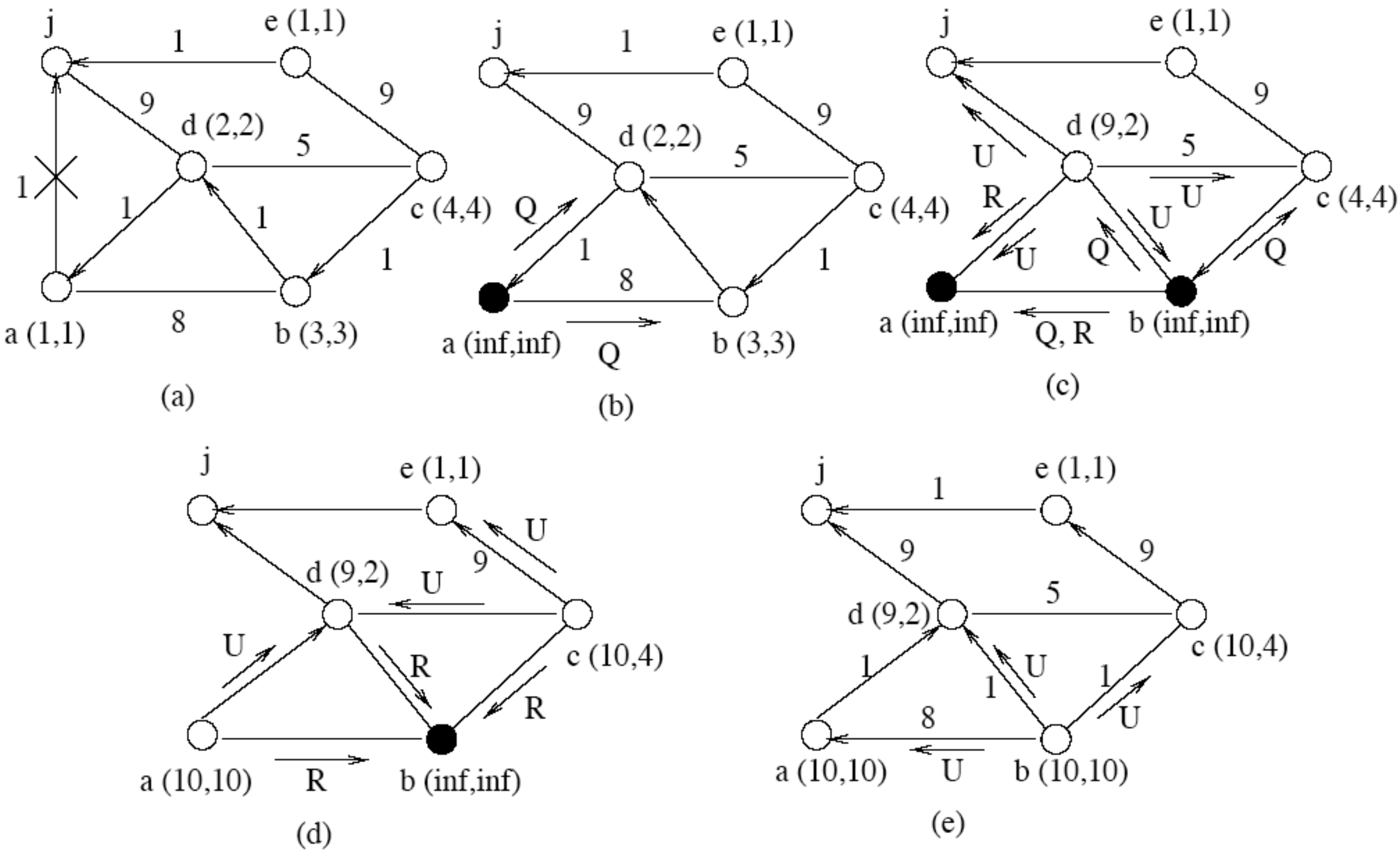
Routing updates

- Two modes
  - Passive: not in danger of looping
    - Has a feasible successor, or has determined that no feasible successor exists.
    - Report distance to neighbors (i.e., normal updates)
  - Active: in danger of looping
    - Start searching for a feasible successor by sending queries with distance to destination set to infinity (i.e., freeze data forwarding).

Routing Updates

- Upon receiving an update
  - If find a feasible successor, update the table accordingly
    - Send reply and updates (I've found my path!)
  - Otherwise, become active: set the distance to infinity, send queries to neighbors.
    - I'm in doubt, let me ask my neighbors.
  - Become passive after receiving all queries have been replied

Example


j
e (1,1)
1
9
d (2,2)
5
1
1
1
c (4,4)
1
a (1,1)
8
b (3,3)
(a)
j
e (1,1)
1
9
9
d (2,2)
5
Q
1
c (4,4)
1
8
a (inf,inf)
Q
b (3,3)
(b)
j
e (1,1)
9
d (9,2)
U
5
R
U
U
U
Q
c (4,4)
Q
a (inf,inf)
Q, R
b (inf,inf)
(c)
j
e (1,1)
U
9
d (9,2)
U
U
R
c (10,4)
R
a (10,10)
R
b (inf,inf)
(d)
j
e (1,1)
1
9
9
5
d (9,2)
c (10,4)
1
U
1
1
U
8
a (10,10)
U
b (10,10)
(e)

Summary

- Basic Distance Vector Routing
  - Pros: distributed path computation, simple, low overhead, good for small networks
  - Cons: count-to-infinity, routing loops, for large or complex topologies
- Loop-Free Path-Finding Algorithm
  - Use predecessor to derive entire paths
  - Remember alternative paths
  - Use the path information to detect loops and update table
  - Use feasibility condition to restrict successor adoption
  - During convergence (query/reply) period, lock up the routes (set to unreachable), synchronize with neighbors before adopting new successors.