Routing

Chapter 3

Forwarding versus Routing

– Forwarding:
  – to select an output port based on destination address and routing table
– Routing:
  – process by which routing table is built

MK MORGAN KAUFMANN 1

Routing

Chapter 3

- Forwarding table VS Routing table
  - Forwarding table
    - Used when a packet is being forwarded and so must contain enough information to accomplish the forwarding function
    - A row in the forwarding table contains the mapping from a network number to an outgoing interface and some MAC information, such as Ethernet Address of the next hop
  - Routing table
    - Built by the routing algorithm as a precursor to build the forwarding table
    - Generally contains mapping from network numbers to next hops

MK MORGAN KAUFMANN 2

Routing

(a)

Prefix/Length	Next Hop
18/8	171.69.245.10

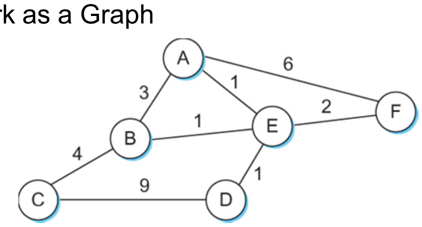
(b)

Prefix/Length	Interface	MAC Address
18/8	if0	8:0:2b:e4:b:1:2

Example rows from (a) routing and (b) forwarding tables

Routing

- Network as a Graph



- The basic problem of routing is to find the lowest-cost path between any two nodes
- The cost of a path equals the sum of the costs of all the edges that make up the path

Routing

Chapter 3

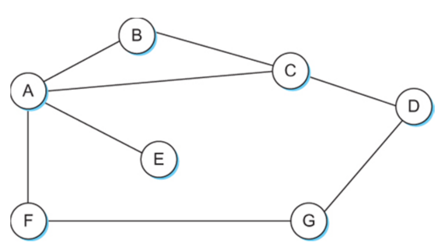
- For a simple network, we can calculate all shortest paths and load them into some nonvolatile storage on each node.
- Such a static approach has several shortcomings
  - It does not deal with node or link failures
  - It does not consider the addition of new nodes or links
  - It implies that edge costs cannot change
- What is the solution?
  - Need a distributed and dynamic protocol
  - Two main classes of protocols
    - Distance Vector
    - Link State

5

Distance Vector Routing Algorithm

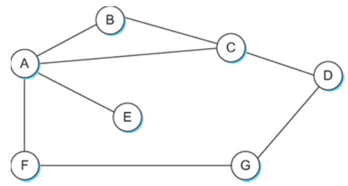
Chapter 3

- Each node constructs a one dimensional array (a vector) containing the "distances" (costs) to all other nodes and distributes that vector to its immediate neighbors
- Starting assumption is that each node knows the cost of the link to each of its directly connected neighbors



6

Distance Vector - Example

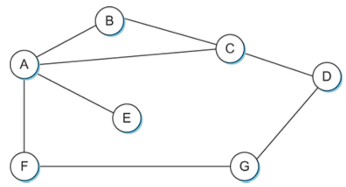


Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Initial distances stored at each node (global view)

Use hop count as the cost

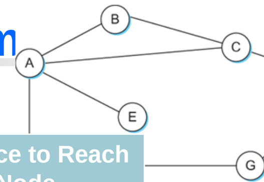
Distance Vector - Example



Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

Initial routing table at node A

Distance Vector - Exam



Chapter 3

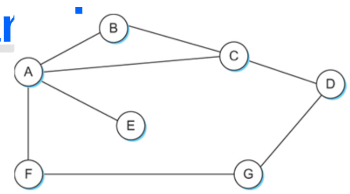
Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	∞
C	1	1	0	1	2	2	2
D	2	2	1	0	∞	2	1
E	1	2	2	∞	0	2	∞
F	1	2	2	2	2	0	1
G	2	∞	2	1	∞	1	0

Distances stored at each node after 1 update (global view)

9

Distance Vector - Exar



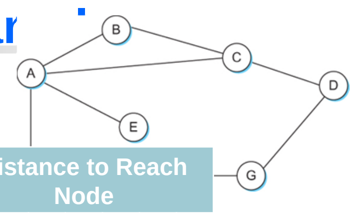
Chapter 3

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	∞
C	1	1	0	1	2	2	2
D	2	2	1	0	∞	2	1
E	1	2	2	∞	0	2	∞
F	1	2	2	2	2	0	1
G	2	∞	2	1	∞	1	0

Distances stored at each node after 1 update (global view)

10

Distance Vector - Exar



Chapter 3

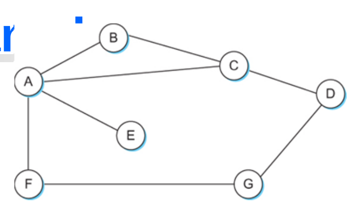
Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	∞
C	1	1	0	1	2	2	2
D	2	2	1	0	∞	2	1
E	1	2	2	∞	0	2	∞
F	1	2	2	2	2	0	1
G	2	∞	2	1	∞	1	0

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Distances stored at each node after 2 updates (global view)

MK 11

Distance Vector - Exar



Chapter 3

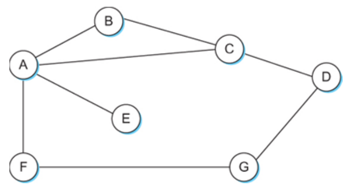
Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Distances stored at each node after 2 updates (global view)

MK 12

Distance Vector - Example

Chapter 3



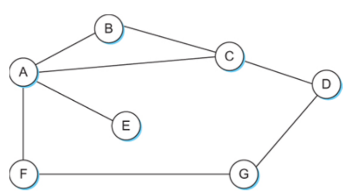
Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Final distances stored at each node (global view)

MK MORGAN KAUFMANN 13

Distance Vector - Example

Chapter 3



Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Final routing table at node A

MK MORGAN KAUFMANN 14

Distance Vector Routing Algorithm

Chapter 3

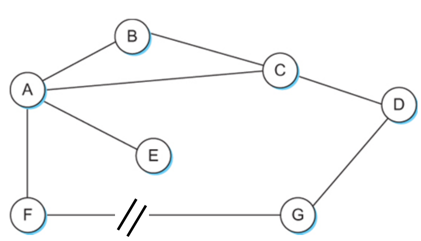
- The distance vector routing algorithm is sometimes called as Bellman-Ford algorithm
- Every T seconds each router sends its table to its neighbor each router then updates its table based on the new information
- Problems include fast response to good news and slow response to bad news. Also too many messages to update

MK MORGAN KAUFMANN 15

Distance Vector Routing Algorithm

Chapter 3

- When a node detects a link failure
  - F detects that link to G has failed
  - F sets distance to G to infinity and sends update to A
  - A sets distance to G to infinity since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A

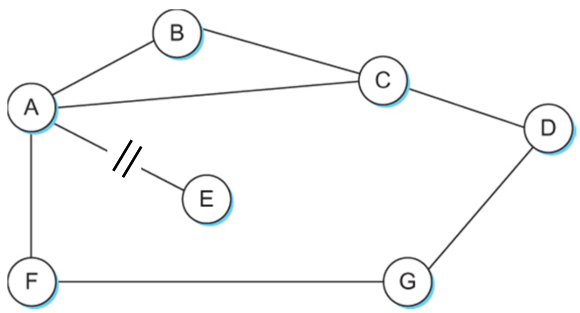


MK MORGAN KAUFMANN 16

Distance Vector Routing Algorithm

Chapter 3

- Slightly different circumstances can prevent the network from stabilizing
  - Suppose the link from A to E goes down
  - In the next round of updates, A advertises a distance of infinity to E, but B and C advertise a distance of 2 to E

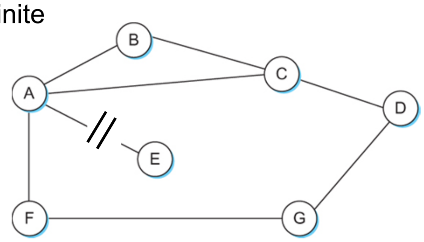


MK MORGAN KAUFMANN 17

Distance Vector Routing Algorithm

Chapter 3

- Depending on the exact timing of events, the following might happen
  - Node B, upon hearing that E can be reached in 2 hops from C, concludes that it can reach E in 3 hops and advertises this to A
  - Node A concludes that it can reach E in 4 hops and advertises this to C
  - Node C concludes that it can reach E in 5 hops; and so on.
  - This cycle stops only when the distances reach some number that is large enough to be considered infinite
    - **Count-to-infinity problem**

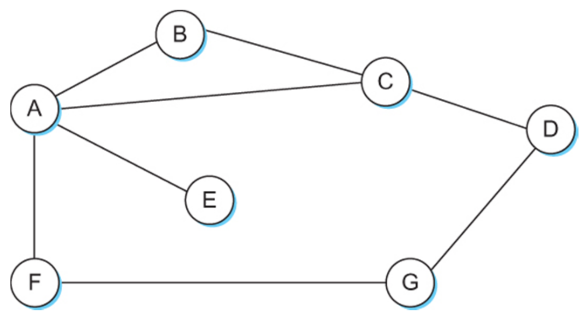


MK MORGAN KAUFMANN 18

Count-to-infinity Problem

Chapter 3

- Use some relatively small number as an approximation of infinity – i.e 16
- Assign a maximum number of hops to get across a certain network

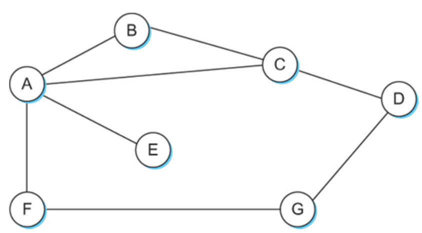


19

Count-to-infinity Problem

Chapter 3

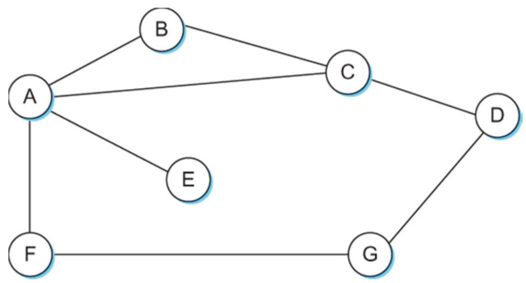
- One technique to improve the time to stabilize routing is called *split horizon*
  - When a node sends a routing update to its neighbors, it does not send those routes it learned from each neighbor back to that neighbor
  - For example, if B has the route (E, 2, A) in its table, then it knows it must have learned this route from A, and so whenever B sends a routing update to A, it does not include the route (E, 2) in that update



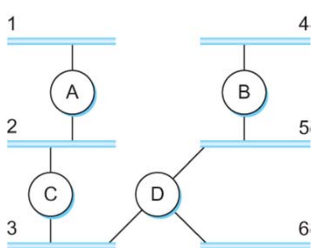
20

Count-to-infinity Problem

- In a stronger version of split horizon, called *split horizon with poison reverse*
  - B actually sends that route back to A, but it puts negative information in the route to ensure that A will not eventually use B to get to E
  - For example, B sends the route (E, ∞) to A



Routing Information Protocol (RIP)



Example Network running RIP

0	8	16 31
Command	Version	Must be zero
Family of net 1		Route Tags
Address prefix of net 1		
Mask of net 1		
Distance to net 1		
Family of net 2		Route Tags
Address prefix of net 2		
Mask of net 2		
Distance to net 2		

RIPv2 Packet Format

Built on Distance Vector Protocol
Simple to implement – Routers advertise cost (# hops) to a network
Infinity set to 16 – limited to small internetworks

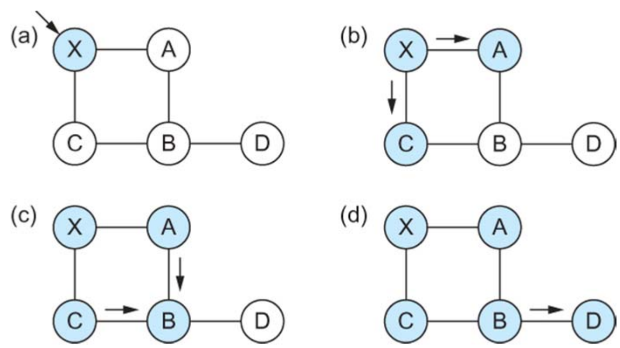
Link State Routing

Strategy: Send to all nodes (not just neighbors) information about directly connected links (not entire routing table).

- Link State Packet (LSP)
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet
- Reliable Flooding
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically; increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP; discard when TTL=0

Link State

Reliable Flooding



Flooding of link-state packets. (a) LSP arrives at node X; (b) X floods LSP to A and C; (c) A and C flood LSP to B (but not X); (d) flooding is complete

Shortest Path Routing

Chapter 3

- Dijkstra's Algorithm - Assume non-negative link weights
  - N: set of nodes in the graph
  - l(i, j): the non-negative cost associated with the edge between nodes i, j ∈N and l(i, j) = ∝ if no edge connects i and j
  - Let s ∈N be the starting node which executes the algorithm to find shortest paths to all other nodes in N
  - Two variables used by the algorithm
    - M: set of nodes incorporated so far by the algorithm
    - C(n) : the cost of the path from s to each node n
    - The algorithm

```
M = {s}
For each n in N – {s}
    C(n) = l(s, n)
while ( N ≠ M)
  M = M ∪ {w} such that C(w) is the minimum
                            for all w in (N-M)
  For each n in (N-M)
          C(n) = MIN (C(n), C(w) + l(w, n))
```


Shortest Path Routing

Chapter Subtitle

- In practice, each switch computes its routing table directly from the LSP's it has collected using a realization of Dijkstra's algorithm called the *forward search algorithm*
- Specifically each switch maintains two lists, known as **Tentative** and **Confirmed**
- Each of these lists contains a set of entries of the form (Destination, Cost, NextHop)

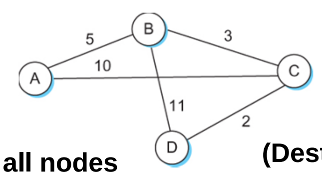
Shortest Path Routing-Forward Search

Chapter 3

- The algorithm
  1) Initialize the **Confirmed** list with an entry for myself; this entry has a cost of 0
  2) For the node just added to the **Confirmed** list in the previous step, call it node **Next**, select its LSP
  3) For each neighbor (Neighbor) of **Next**, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor
     a) If Neighbor is currently on neither the **Confirmed** nor the **Tentative** list, then add (Neighbor, Cost, Nexthop) to the **Tentative** list, where Nexthop is the direction I go to reach Next
     b) If Neighbor is currently on the **Tentative** list, and the Cost is less than the currently listed cost for the Neighbor, then replace the current entry with (Neighbor, Cost, Nexthop) where Nexthop is the direction I go to reach Next
     c) If Neighbor on **Confirmed** list, then ignore it
  4) If the **Tentative** list is empty, stop. Otherwise, pick the entry from the **Tentative** list with the lowest cost, move it to the **Confirmed** list, and return to Step 2.

Shortest Path Routing-Forward Search

Chapter 3



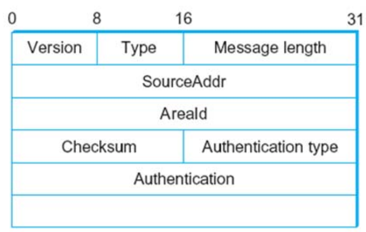
D has LSP's of all nodes

(Destination, Cost, NextHop)

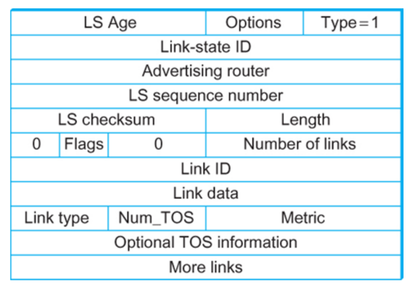
Step	Confirmed	Tentative	Comments
1	(D,0,–)		Since D is the only new member of the confirmed list, look at its LSP.
2	(D,0,–)	(B,11,B) (C,2,C)	D's LSP says we can reach B through B at cost 11, which is better than anything else on either list, so put it on Tentative list; same for C.
3	(D,0,–) (C,2,C)	(B,11,B)	Put lowest-cost member of Tentative (C) onto Confirmed list. Next, examine LSP of newly confirmed member (C).
4	(D,0,–) (C,2,C)	(B,5,C) (A,12,C)	Cost to reach B through C is 5, so replace (B,11,B). C's LSP tells us that we can reach A at cost 12.
5	(D,0,–) (C,2,C) (B,5,C)	(A,12,C)	Move lowest-cost member of Tentative (B) to Confirmed, then look at its LSP.
6	(D,0,–) (C,2,C) (B,5,C)	(A,10,C)	Since we can reach A at cost 5 through B, replace the Tentative entry.
7	(D,0,–) (C,2,C) (B,5,C) (A,10,C)		Move lowest-cost member of Tentative (A) to Confirmed, and we are all done.

Open Shortest Path First (OSPF)

Chapter 3



OSPF Header Format



OSPF Link State Advertisement

29

Distance-Vector vs Link-State

Chapter 3

- **Distance-Vector(RIP):**
  - Each node talks with its neighbors only
  - Sends all information it knows - known distances to other nodes
  - Speed of convergence is slower than LS
  - Stabilization may not occur – count to infinity
  - Simple algorithm
- **Link-State (OSPF):**
  - Each node talks to all other nodes
  - Sends what it knows for sure - State of its directly connected links
  - Stabilizes quickly and it responds rapidly to network changes
  - Low traffic generation
  - Storage at each node is large
  - Uses reliable flooding of packets

30

Metrics – Cost of Links

Chapter 3

- Assign 1 to all links – hop count
- Latency – take into account delay of the link
- Capacity – what is BW of each link
- Current Load – increase cost as load increases
- Queue length (average value between updates)

- Metrics are fixed by administrators – not dynamically changing due to stability issues.

Summary

Chapter 3

- We have looked at some of the issues involved in building scalable and heterogeneous networks by using switches and routers to interconnect links and networks.
- To deal with heterogeneous networks, we have discussed in details the service model of Internetworking Protocol (IP) which forms the basis of today's routers.
- We have discussed in details two major classes of routing algorithms
  - Distance Vector
  - Link State