How do Switches Maintain Routing Tables?

Distance-vector (Bellman-Ford)

Distance Vector (Bellman-Ford)

$D_i^h \triangleq$ shortest distance to some destination s.t. path length $\leq h$ hops

$G=(V,E)$, $|V|=N$ nodes

Init $D_{dest}^0 = 0$ $\quad D_{i \neq dest}^0 = \infty$

for $h = 1, 2, 3, \ldots, N-1$

$D_i^h = \min_{k \in Neighbors(i)} \{ D_k^{h-1} + w_{ik} \}$ $\quad \forall i$

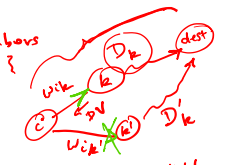
$O(N^3)$

18

How do Switches Maintain Routing Tables?

Distance-vector (Bellman-Ford)

Periodically, send your neighbors Distance vector $\rightarrow$ { (dest, distance) }



1) unknown destination $\Rightarrow$ take k as next hop

2) k is offering a shorter distance $\Rightarrow$ switch to k as next hop

3) k is my current next hop $\Rightarrow$ update my own distance

Routing table

Dest	Next Hop	Dist Cost

19

How do Switches Maintain Routing Tables?

Distance-vector (Bellman-Ford)

- Each node keeps in its routing table its current best distance (cost) to each destination node: (Destination, Cost, NextHop)
- Initially, the routing table of each node contains its neighbor nodes and distances to them
- Each node sends a copy of its routing information {(Destination, Cost)} to each of its neighbor nodes
  - periodically (on the order of several seconds)
  - whenever its table changes (called ***triggered*** update)
- Upon receiving a message from a neighbor J, a node K updates its routing table as follows:
  - If node J knows a shorter path to some destination, node K updates its corresponding entry with J as the new next-hop
  - If node J reports a distance for an unknown destination, node K adds a new entry for it with J as the next-hop
  - If node K is using node J as its next-hop to some destination node and J reports a change in its distance to it, K updates the corresponding entry

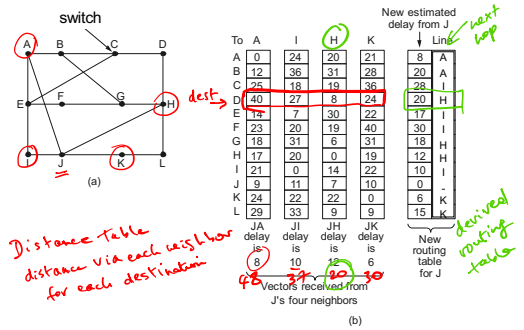
20

Route Calculation in Distance Vector

```
void mergeRoute (Route *new) {
    int i;
    for (i = 0; i < numRoutes; ++i)  {
        if (new->Dest == rt[i].Dest) {
                if (new->Cost + 1 < rt[i].Cost)  // min-hop routing
                        break;
                else if (new->NextHop == rt[i].NextHop)
                        break;
                else
                        return;
        }
    }
    rt[i] = *new;
    rt[i].TTL = MAX_TTL;
    ++rt[i].Cost;
    if (i == numRoutes)
        ++numRoutes;
}
```


21

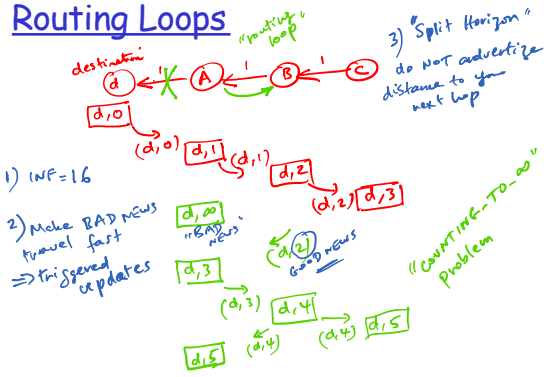
DV Routing



Distance table
distance via each neighbor
for each destination

22

Routing Loops

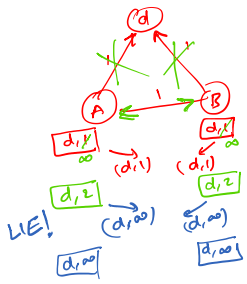


23

Routing Loops

4) "Split Horizon with Poison Reverse"

No routing loops involving 2 nodes



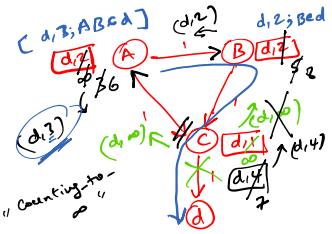
24

Routing Loops

PATH VECTOR

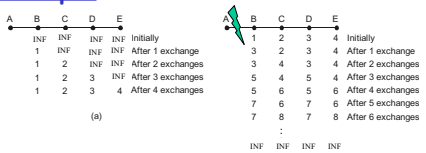
Advertise ∞ to your neighbor if it's on the path

append node id in routing message



25

Routing Loops



A	B	C	D	E	
INF	INF	INF	INF	INF	Initially
1	INF	INF	INF	INF	After 1 exchange
1	2	INF	INF	INF	After 2 exchanges
1	2	3	INF	INF	After 3 exchanges
1	2	3	4	INF	After 4 exchanges

(a)

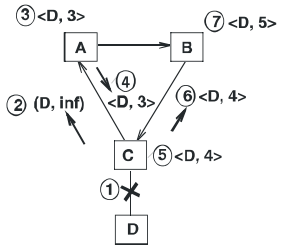
A	B	C	D	E	
1	2	3	4		Initially
3	2	3	4		After 1 exchange
3	4	3	4		After 2 exchanges
5	4	5	4		After 3 exchanges
5	6	5	6		After 4 exchanges
7	6	7	6		After 5 exchanges
7	8	7	8		After 6 exchanges
:					
INF	INF	INF	INF		

(b)

- Heuristics to break routing loops
  - set infinity to maximum distance/cost
  - split horizon: a node does not advertize to its next-hop
  - split horizon with poison reverse: a node advertizes to its next-hop a distance of infinity

26

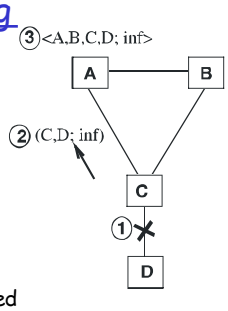
Routing Loops can still happen!



- link CD goes down
- loop forms involving A, B and C

27

Path Vector Routing



- loops can be completely avoided

28

Distance-Vector versus Link-State

Distance-Vector:

- Easy to implement
- Larger routing update messages: message size is proportional to the number of nodes in the network
- Slow to converge: route computation is distributed
- Loops/count-to-infinity may happen
- If link changes don't affect shortest path, no message exchange

29

Distance-Vector versus Link-State

Link-State:

- Smaller routing update messages: message size depends on the number of neighbors a node has
- Converges quickly: route computation is centralized
- A node stores a complete view of the network
- Any link change requires a broadcast

Both have strengths and weaknesses.
One or the other is used in almost every network