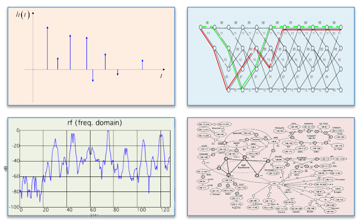


INTRODUCTION TO EECS II
DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2011
Lecture #21

- link-state routing
- routing around failures

Link-State Routing

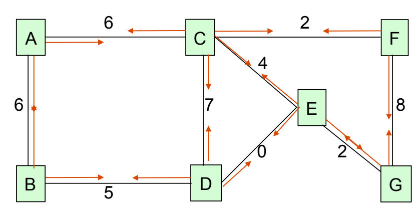
- Advertisement step
  - Send information about its links to its neighbors (aka **link state advertisement** or LSA):

    [node, seq#, [(nbhr1, linkcost1), (nbhr2, linkcost2), …]

  - Do it periodically (liveness, recover from lost LSAs)
- Integration
  - If seq# in incoming LSA > seq# in saved LSA for source node:
    update LSA for node with new seq#, neighbor list
    rebroadcast LSA to neighbors (→ **flooding**)
  - Remove saved LSAs if seq# is too far out-of-date
  - Result: Each node discovers current map of the network
- Build routing table
  - Periodically each node runs the same *shortest path algorithm* over its map (e.g., Dijkstra's alg)
  - If each node implements computation correctly and each node has the same map, then routing tables will be correct

LSA Flooding

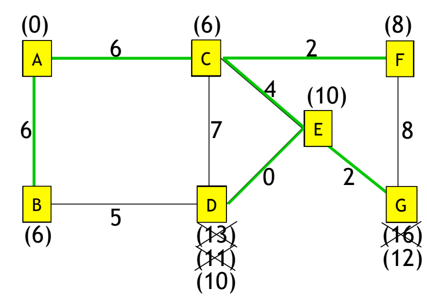
LSA: [F, seq, (G, 8), (C, 2)]



- Periodically originate LSA
- LSA travels each link in each direction
  - Don't bother with figuring out which link LSA came from
- Termination: each node rebroadcasts LSA exactly once
  - Use sequence number to determine if new, save latest seq
- Multiple opportunities for each node to hear any given LSA
  - Time required: number of links to cross network

Integration Step: Dijkstra's Algorithm (Example)

Suppose we want to find paths from A to other nodes



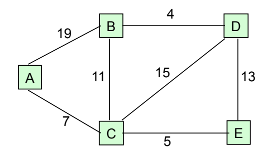
Dijkstra's Shortest Path Algorithm

- Initially
  - nodeset = [all nodes] = set of nodes we haven't processed
  - spcost = {me:0, all other nodes: ∞} # shortest path cost
  - routes = {me:--, all other nodes: ?} # routing table
- while nodeset isn't empty:
  - find u, the node in nodeset with smallest spcost
  - remove u from nodeset
  - for v in [u's neighbors]:
    - d = spcost(u) + cost(u,v) # distance to v via u
    - if d < spcost(v): # we found a shorter path!
      - spcost[v] = d
      - routes[v] = routes[u] (or if u == me, enter link from me to v)

Another Example

Finding shortest paths from A:

LSAs:
```
A: [(B,19), (C, 7)]
B: [(A,19), (C,11), (D, 4)]
C: [(A, 7), (B,11), (D,15), (E, 5)]
D: [(B, 4), (C,15), (E,13)]
E: [(C, 5), (D,13)]
```

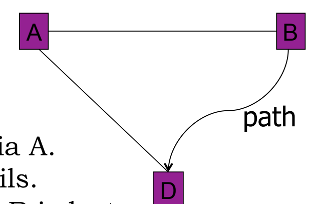


Step	*u*	*Nodeset*	*spcost*					*route*				
			A	*B*	*C*	*D*	*E*	*A*	*B*	*C*	*D*	*E*
0		[A,B,C,D,E]	0	∞	∞	∞	∞	--	?	?	?	?
1	A	[B,C,D,E]	0	19	7	∞	∞	--	L0	L1	?	?
2	C	[B,D,E]	0	18	7	22	12	--	L1	L1	L1	L1
3	E	[B,D]	0	18	7	22	12	--	L1	L1	L1	L1
4	B	[D]	0	18	7	22	12	--	L1	L1	L1	L1
5	D	[]	0	18	7	22	12	--	L1	L1	L1	L1

Failures

- Links and switches could fail
- Advertisements could get lost
- HELLO protocol
  - Detecting liveness of neighbors
- Routing loop
  - A sequence of nodes on forwarding path that has a cycle (so packets will never reach destination)
- Dead-end
  - Route does not actually reach destination
- Loops and dead-ends lead to *routes not being valid*

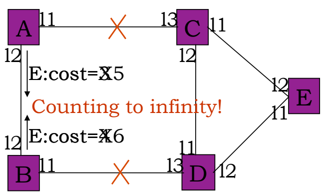
Routing Loop in Link-State Protocol



B to D is via A.
Link AD fails.
A's LSA to B is lost.
A now uses B to get to D.
But B continues to use A.
Routing loop!
Must wait for eventual arrival of correct LSAs to fix loop

But What About Distance-Vector: Pros and Cons

- + Simple protocol
- + Works well for small networks
- - Works only on small networks



Suppose link AC fails.
When A discovers failure, it sends E: cost = INFINITY to B.
B sends 'E: cost=2' to A
A installs E: cost=3.

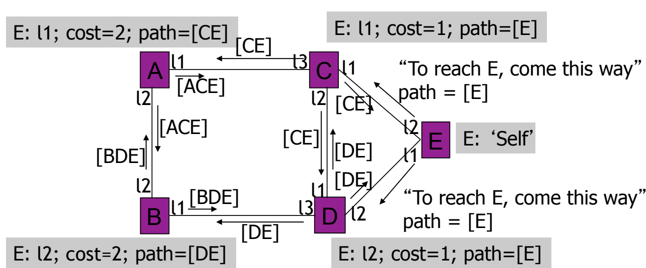
Now suppose link BD fails. B discovers it, then installs E: cost = INFINITY. Sends info to A, A installs E: cost = INFINITY.

But what if A had sent advert. to B before B sends advert to A?

Fixing "Count to Infinity" with Path Vector Routing

- Problem
  - Node C's route to A breaks, C sets cost to ∞
  - But at next round of advertisements, hears of lower-cost routes from neighbors, not know the neighbor's routes used C itself to get to A.
- Solution
  - In addition to reporting costs in advertisements, also report routing path as discovered incrementally by Bellman-Ford
  - Called "path-vector"
  - Modify Bellman-Ford update with new rule: nodes should ignore advertised routes that contain itself in the routing path
  - Pros: count-to-infinity "problem" is solved (routing tables eliminate routes to unreachable nodes more quickly)
  - Cons: advertisement overhead is larger

Path Vector Routing



- For each advertisement, run "integration step"
  - E.g., pick shortest, cheapest, quickest, etc.
- **Ignore advertisements with own address in path vector**
  - Avoids routing loops that "count to infinity"

Summary

- The network layer implements the "glue" that achieves connectivity
  - Does addressing, forwarding, and routing
- Forwarding entails a routing table lookup; the table is built using *routing protocol*
- DV protocol: distributes route computation; each node advertises its best routes to neighbors
  - Path-vector: include path, not just cost, in advertisement
- LS protocol: distributes (floods) neighbor information; centralizes route computation using shortest-path algorithm