eecs489 Computer Networks

Lecture 16: Routing:
Link-State Algorithm

Link State Routing

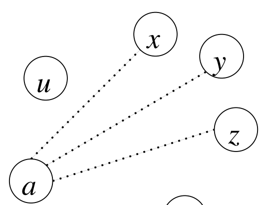
Observation: loop can be prevented if each node knows the actual network topology . . . theoretically

In link-state routing, each node:

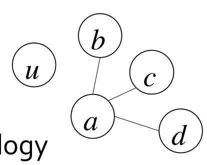
- keeps track of the state of its incident links
  - link state here means the cost of the link, ∞ = link down
- floods the network with the state of its links
- uses Dijkstra's Shortest Path First (SPF) algorithm to compute a shortest-path tree and construct the forwarding table

Link State Routing

What is advertised:



- DV: all nodes reachable from me, advertised to all neighbors



- LS: all my immediate neighbors, advertised to all nodes
  ⇒ thus each node has the complete topology

Example link-state routing protocols:

- Open Shortest-Path First (OSPF)
- Intermediate System–Intermediate System (IS-IS)

Dijkstra's Shortest Path First (SPF) Algorithm

A greedy algorithm for solving single-source shortest path problem

- assume non-negative edge weights
- even if we're only interested in the path from s to a single destination, d, we need to find the shortest path from s to all vertices in G (otherwise, we might have missed a shorter path)
- if the shortest path from s to d passes through an intermediate node u, i.e., $P = \{s, \ldots, u, \ldots, d\}$, then $P' = \{s, \ldots, u\}$ must be the shortest path from s to u

Dijkstra's SPF Algorithm

An iterative algorithm

- after k iterations, knows shortest path to k nodes

`spf`: a list of nodes whose shortest path is definitively known

- initially, `spf` $= \{s\}$ where s is the source node
- add one node with lowest path cost to `spf` in each iteration

`cost[v]`: current cost of path from source s to node v

- initially, `cost[v]` $= c(s, v)$ for all nodes v adjacent to s
- and `cost[v]` $= \infty$ for all other nodes v
- continually update `cost[v]` as shorter paths are learned

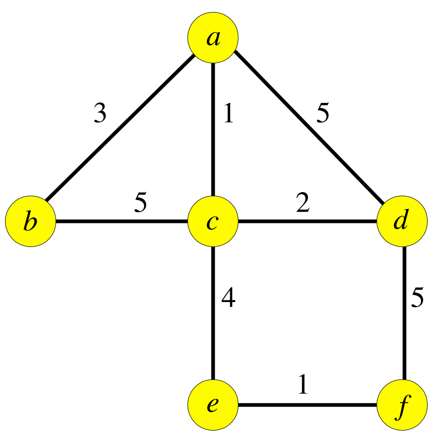
[Rexford]

Dijkstra's SPF Pseudocode

```
SPF(startnode s)
{ // Initialize
  table = createtable(|V|); // stores spf, cost, predecessor
  table[*].spf = false; table[*].cost = INFINITY;
  pq = createpq(|E|); // empty pq
  table[s].cost = 0;
  pq.insert(0, s); // pq.insert(cost, v)
  while (!pq.isempty()) {
    v = pq.getMin();
    if (!table[v].spf) { // not on sp tree
      table[v].spf = true;
      for each u = v.neighbors() {
        newcost = weight(u, v) + table[v].cost;
        if (table[u].cost > newcost) {
          table[u].cost = newcost;
          table[u].pred = v;
          pq.insert(newcost, u);
        }
      }
    }
  }
  extract SPF from table;
}
```

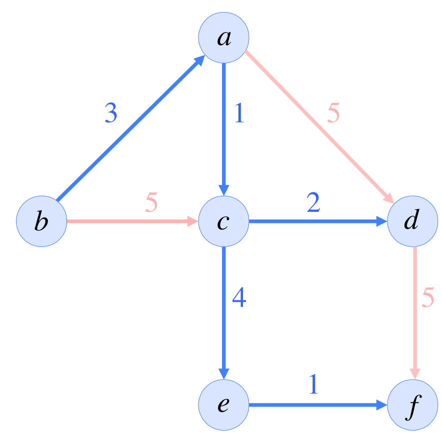
Dijkstra's SPF Example (init, $s=b$)

u	spf	cost	pred
a	F	∞	
b	F	0	–
c	F	∞	
d	F	∞	
e	F	∞	
f	F	∞	



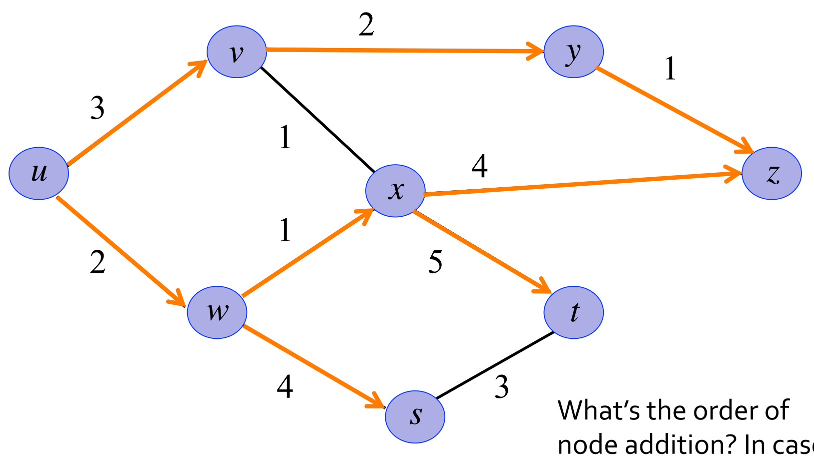
Dijkstra's SPF Example ($v=f$)

u	spf	cost	pred
a	T	3	b
b	T	0	–
c	T	4	a
d	T	6	c
e	T	8	c
f	T	9	e



Shortest path for $dest = f$: $f \leftarrow e \leftarrow c \leftarrow a \leftarrow b$

Dijkstra's SPF: Another Example

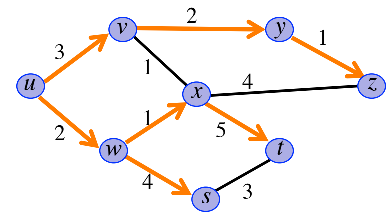


What's the order of node addition? In case of tie, assume fifo

[after Rexford]

Shortest-Path Tree

Shortest-path tree from u:



Forwarding table at u:

destination	link
v	(u, v)
w	(u, w)
x	(u, w)
y	(u, v)
s	(u, w)
z	(u, v)
t	(u, w)

[after Rexford]

Dijkstra's SPF Algorithm

Time complexity: given N nodes,

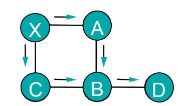
- each iteration: extract minHeap $O(\log|N|)$
- total $O(|N|\log|N|)$

Each neighbor of each node could also potentially be inserted into the minHeap once: $O(|E|\log|N|)$

Total: $O(|N|\log(|N|)+|E|\log(|N|)) = O(|E|\log|N|)$

- $|E| \geq |N| - 1$ for a connected graph

Flooding Link State



Flooding

- a node sends its link-state information out all of its links
- the next node forwards the link-state information on all of its links except the one the information arrived at

When to initiate flooding?

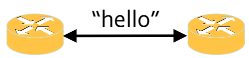
- topology change
  - link or node failure
  - link or node recovery
- configuration change
  - link cost change
- periodically
  - to refresh the link-state information (soft states)
  - typically (say) every 30 minutes
  - corrects for possible corruption of data

[Rexford]

How to Detect Down Link?

Beaconing

- send periodic "hello" messages in both directions
- detect a failure after a few missed "hellos"



Down link detection depends on timeout waiting for "hello" packets

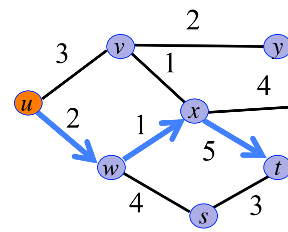
How often to send "hello" messages? Performance trade-offs

- detection speed
- overhead on link bandwidth and CPU
- likelihood of false detection

[Rexford]

Delay in Detecting Down Link

Meanwhile, undetected down link causes data packets to be sent into a "blackhole"



[Rexford]

Flooding Link State

Why flood? to get all the nodes in the network to converge to the new topology

Upon convergence, all nodes will have consistent routing information and can compute consistent forwarding:

- all nodes have the same link-state database
- all nodes forward packets on shortest paths
- the next router on the path forwards to the expected next hop

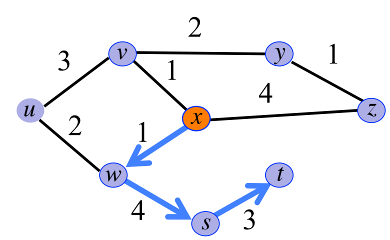
[Rexford]

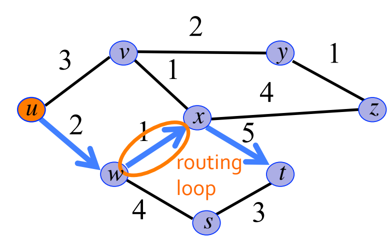
Sources of Convergence Delay

Delay in detecting network changes

Latency in link-state flooding results in inconsistent link-state database

- some routers know about failure before others
- computed shortest paths are no longer consistent
- can cause transient forwarding loops





[after Rexford]

What if Network Doesn't Converge?

Other sources of convergence delay
- time complexity of shortest-path computation
- time complexity of forwarding table update

Performance when network is not in convergence
- packets lost due to blackholes and TTL expiry
- looping packets consuming resources
- packets reach destination out-of-order

[after Rexford]

Ways to Reduce Convergence Delay

Faster link down detection
- smaller hello timers
- link-layer technologies that can detect failures

Faster flooding
- flood immediately
- send link-state packets with high-priority

Faster SPF computation
- faster processors on the routers
- incremental Dijkstra's algorithm

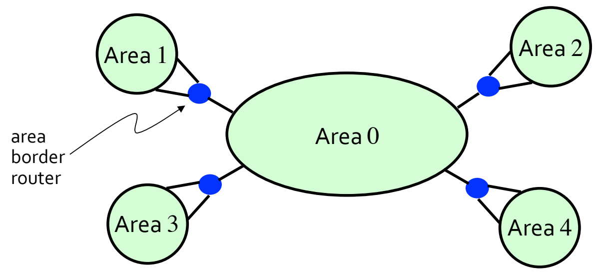
Faster forwarding-table update
- data structures that support incremental updates

[after Rexford]

Scaling Link-State Routing

Scalability-limiting overheads of link-state routing:
- the need to flood link-state packets throughout the network
- time complexity of Dijkstra's SPF algorithm

Break-up the overheads by isolating parts of the network into a hierarchy of "areas"



[after Rexford]

OSPF (Open Shortest Path First)

Open: non-proprietary

Uses link state algorithm
- not loop free due to delay in topology propagation
- link state dissemination by flooding requires reliable transmission:
  - all nodes must receive all link-state information and
  - they must recognize the latest version (complicated!)
- carried in OSPF messages directly over IP (rather than TCP or UDP)

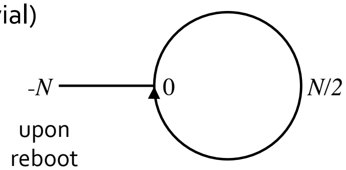
OSPF (Open Shortest Path First)

Maintaining LS database consistency is hard:

- how to determine which LS is newer?
- challenges: packet loss, out-of-order arrival, node reboots (complicated!)

Solutions:

- acknowledgments and retransmissions
- lollipop sequence numbers (not trivial)
- time-to-live for each packet



[after Rexford]

OSPF (Open Shortest Path First)

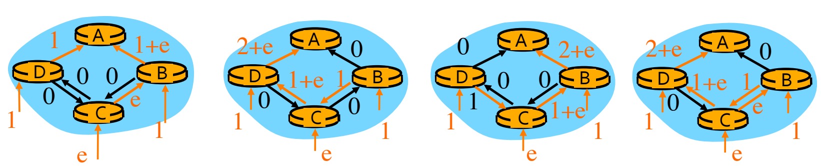
Advance features (not in RIP):

- security: all OSPF messages are authenticated (to prevent fake advertisement)
- equal-cost multi-path (ECMP) routing allowed: popular for data center networks (only one path in RIP)
- for each link, multiple cost metrics for different TOS (e.g., satellite link cost can be set to "low" for best effort traffic, but set to "high" for real time traffic)
- integrated unicast and multicast support:
  - multicast OSPF (MOSPF) uses same topology data base as OSPF
- hierarchical OSPF support in large domains

Oscillation in SPF

Oscillation is possible, e.g., if link cost is computed as link load

Example: traffic destined for A



initially | recompute routing | recompute | recompute

Comparison of LS and DV Routing

Message complexity

LS: with N nodes, E links, $O(NE)$ messages sent

DV: exchange between neighbors only

Speed of Convergence

LS: relatively fast

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect link cost
- each node computes its own table

DV:

- node can advertise incorrect path cost
- each node's table used by others (error propagates)

Similarities of LS and DV Routing

Both are shortest-path based routing

- minimizing cost metric (link weights) a common optimization goal
  - routers share a common view as to what makes a path “good” and how to measure the “goodness” of a path

Due to shared goal, commonly used inside an organization

- RIP and OSPF are mostly used for intradomain routing
- e.g., AT&T uses OSPF

But the Internet is a “network of networks”

- how to stitch together the many networks when the networks may not share common goals
- and may not want to share information