Lecture 5

Last

- **Forwarding**: data plane
    - Directing one data packet
    - Each router using local routing state
- **Routing**: control plane
    - Computing the forwarding tables that guide packets
    - Jointly computed by routers using a distributed protocol
- Two correctness conditions for routing state:
    - no deadends
    - no loops

Goal (for routing)

- v1: Find a path to a given destination
- v2: Find a **least cost path** to a given destination

Link costs, could represent

```
- propagation delay
- load
- cost
```

Least-cost path routing

- Given: router graph & link costs
- Goal: find least-cost path
    - from each source router
    - to each destination router

“Least Cost” Routes

- “Least cost” routes an easy way to avoid loops
    - No sensible cost metric is minimized by traversing the loop
- Least cost routes are also “destination-based”
    - i.e., do not depend on the source
- Least-cost paths form a spanning three

Approach 1: Link-state routing

Routing algorithm for source u

- Input: router graph & link costs
- Output: least cost path
    - from source router u
    - to every other router

Dijkstra’s algorithm

- Source router considers every other router
    - starting from next “closest” neighbor

- Checks whether it can improve paths
    - by using that router as an intermediate point
- Ends when all intermediaries have been considered

From routing algorithm to protocol

- Note: Dijkstra's is a **local** computation
    - computed by one node given complete network graph
- Possibilities
    - Option 1: a separate machine runs the algorithm
    - Option 2: every router runs the algorithm
- The internet currently uses Option 2

Link State Routing

- Every router knows its local "link state"
    - router u: "(u,v) with cost=2; (u,x) with cost=1"
- A router floods its link state to all other routers
    - does so periodically or when its link state changes
- Hence, every router learns the entire network graph
    - runs Dijkstra's locally to compute forwarding table
- OSPF is a link-state protocol (IETF RFC 2328 or 5340)
    - Berkeley runs OSPF internally!

Convergence

- All routers have consistent routing information
    - E.g., all nodes having the same link-state database
- Forwarding is consistent after convergence

- All nodes have the same link-state database
- All nodes forward packets on same paths

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
    - Time to detect failure
    - Time to flood link-state information
    - Time to re-compute forwarding tables
- Performance during convergence period?
    - Lost packets due to blackholes
    - Looping packets
    - Out-of-order packets reaching the destination

Link State Routing

- Are loops possible?
    - yes, until convergence
- Scalability
    - Messages: O(NE)
    - Computation time: $O(N^2)$
    - Convergence time: O(network diameter)
    - Entries in forwarding table: O(N)
- Do we have to use Dijkstra's? No.

Approach 2: Distance-vector routing

Experiment

- Your job: find the youngest person in the room
- Ground rules
    - You may not leave your seat, nor shout loudly across the class
    - You may talk with your immediate neighbors (hint: "exchange updates" with them)
- At the end of **5 minutes**, I will pick a victim and ask:
    - Who is the youngest person in the room? (name, date)
    - Which one of your neighbors first told you this information?

Distance-vector routing

- Input to each router
    - local link costs & neighbor messages
- Output of each router
    - least-cost path to every other router
- Distributed algorithm
- All routers run it "together"
    - each router runs its own instance
    - neighbors exchange and react to each other's message

Bellman-Ford algorithm

- All neighbors exchange information
    - Each router checks whether it can improve current paths by leveraging the new information
- Ends when no improvement is possible

Bellman-Ford equation

- $d_x(z) = \min_n\{ cost (x,n) + d_n(z) \}$
    - for all neighbors n
- Formalizes the following decision:
    - Pick as the next hop for destination z the neighbor that results in the least-cost path z

Problems with Bellman-Ford

- Routing loops
    - z routes through y, y routes through x
    - y loses connectivity to x
    - y decides to route through z
- Can take a very long time to resolve
    - Count-to-infinity scenario

Solution

- Poisoned reverse
    - if z routes to x through y, z advertises to y that its cost to x is infinite
    - y never decides to route to x through z
- Often avoids the count-to-infinity problem

Distance Vector Routing

- Are loops possible?
    - Yes, until convergence
    - Convergence slower than in Link-State

- Scalability
    - Requires fewer messages than Link-State
    - Update time on arrival of a new DV from neighbor: O(N)
    - Convergence time: O(network diameter)
    - Entries in forwarding table: O(N)
- Rip is a protocol that implements DV (IETF RFC 2080)