What is Routing?

Routing implements the core function of a network:

It ensures that information accepted for transfer at a source node is delivered to the correct set of destination nodes, at reasonable levels of performance.

Internet Routing

- Internet organized as a two level hierarchy
- First level – autonomous systems (AS’s)
  - AS – region of network under a single administrative domain
  - AS’s run an intra-domain routing protocols
    - Distance Vector, e.g., Routing Information Protocol (RIP)
    - Link State, e.g., Open Shortest Path First (OSPF)
  - Between AS’s runs inter-domain routing protocols, e.g., Border Gateway Routing (BGP)
    - De facto standard today, BGP-4

Example
Forwarding vs. Routing

- **Forwarding:** “data plane”
  - Directing a data packet to an outgoing link
  - Individual router using a forwarding table
- **Routing:** “control plane”
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router creating a forwarding table

Routing requires knowledge of the network structure
- Centralized global state
  - Single entity knows the complete network structure
  - Can calculate all routes centrally
  - Problems with this approach?
- Distributed global state
  - Every router knows the complete network structure
  - Independently calculates routes
  - Problems with this approach?
- Distributed no-global state
  - Every router knows only about its neighboring routers
  - Independently calculates routes
  - Problems with this approach?

Know Thy Network

- Link State Routing
  - E.g. Algorithm: Dijkstra
  - E.g. Protocol: OSPF
- Distance Vector Routing
  - E.g. Algorithm: Bellman-Ford
  - E.g. Protocol: RIP

Modeling a Network

- Modeled as a graph
  - Routers ⇒ nodes
  - Link ⇒ edges
    - Possible edge costs
      - delay
      - congestion level
- Goal of Routing
  - Determine a “good” path through the network from source to destination
  - Good usually means the shortest path

Link State: Control Traffic

- Each node floods its local information to every other node in the network
- Each node ends up knowing the entire network topology → use Dijkstra to compute the shortest path to every other node
Dijsktra’s Algorithm

1  Initialization:
2    S = \{A\};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6      else D(v) = \infty;
7
8  Loop
9    find w not in S such that D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent to w and not in S:
12      if D(w) + c(w,v) < D(v) then
13        D(v) = D(w) + c(w,v); p(v) = w;
14      until all nodes in S;

Notation

- \(c(i,j)\): link cost from node \(i\) to \(j\); cost infinite if not direct neighbors; \(\geq 0\)
- \(D(v)\): current value of cost of path from source to destination \(v\)
- \(p(v)\): predecessor node along path from source to \(v\), that is next to \(v\)
- \(S\): set of nodes whose least cost path definitively known

Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start S</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>\infty</td>
<td>\infty</td>
</tr>
</tbody>
</table>

Example: Dijkstra’s Algorithm

1  Initialization:
2    S = \{A\};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6      else D(v) = \infty;
7
Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>AD</td>
<td></td>
<td>4,D</td>
<td>2,D</td>
<td></td>
</tr>
</tbody>
</table>

Loop
8. find w not in S s.t. D(w) is a minimum;
9. add w to S;
10. update D(v) for all v adjacent to w and not in S;
12. if D(w) + c(w,v) < D(v) then
13. D(v) = D(w) + c(w,v); p(v) = w;
14. until all nodes in S;

Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>AD</td>
<td></td>
<td>4,D</td>
<td>2,D</td>
<td></td>
</tr>
</tbody>
</table>

Loop
8. find w not in S s.t. D(w) is a minimum;
9. add w to S;
10. update D(v) for all v adjacent to w and not in S;
12. if D(w) + c(w,v) < D(v) then
13. D(v) = D(w) + c(w,v); p(v) = w;
14. until all nodes in S;
### Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start S</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>4,D</td>
<td>2,D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3,E</td>
<td></td>
<td>4,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loop

9. find w not in S s.t. D(w) is a minimum;
10. add w to S;
11. update D(v) for all v adjacent to w and not in S;
12. if D(w) + c(w,v) < D(v) then
13. D(v) = D(w) + c(w,v); p(v) = w;
14. until all nodes in S;

To determine path A → C (say), work backward from C via p(v)
The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the forwarding table

<table>
<thead>
<tr>
<th>Destination</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>(A,B)</td>
</tr>
<tr>
<td>C</td>
<td>(A,D)</td>
</tr>
<tr>
<td>D</td>
<td>(A,D)</td>
</tr>
<tr>
<td>E</td>
<td>(A,D)</td>
</tr>
<tr>
<td>F</td>
<td>(A,D)</td>
</tr>
</tbody>
</table>

Complexity

- How much processing does running the Dijkstra algorithm take?
- Assume a network consisting of N nodes
  - Each iteration: need to check all nodes, w, not in S
  - \( N(N+1)/2 \) comparisons: \( O(N^2) \)
  - More efficient implementations possible: \( O(N \log(N)) \)

Obtaining Global State

- Flooding
  - Each router sends link-state information out its links
  - The next node sends it out through all of its links
    - except the one where the information arrived
    - Note: need to remember previous msgs & suppress duplicates!

Flooding the Link State

- Reliable flooding
  - Ensure all nodes receive link-state information
  - Ensure all nodes use the latest version
- Challenges
  - Packet loss
  - Out-of-order arrival
- Solutions
  - Acknowledgments and retransmissions
  - Sequence numbers
  - Time-to-live for each packet
When to Initiate Flooding

- Topology change
  - Link or node failure
  - Link or node recovery
- Configuration change
  - Link cost change
  - See next slide for hazards of dynamic link costs based on current load
- Periodically
  - Refresh the link-state information
  - Typically (say) 30 minutes
  - Corrects for possible corruption of the data

Oscillations

- Assume link cost = amount of carried traffic

Distance Vector Routing

- Each router knows the links to its immediate neighbors
  - Does not flood this information to the whole network
- Each router has some idea about the shortest path to each destination
  - E.g.: Router A: “I can get to router B with cost 11 via next hop router D”
- Routers exchange this information with their neighboring routers
  - Again, no flooding the whole network
- Routers update their idea of the best path using info from neighbors

5 Minute Break

Questions Before We Proceed?
Information Flow in Distance Vector

Bellman-Ford Algorithm

- **INPUT:**
  - Link costs to each neighbor
  - *Not* full topology
- **OUTPUT:**
  - Next hop to each destination and the corresponding cost
  - Does *not* give the complete path to the destination

Bellman-Ford - Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X ⇒ best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$

Bellman-Ford - Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X ⇒ best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$

Smallest distance in row Y = shortest Distance of A to Y, $D(A, Y)$
Bellman-Ford - Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X ⇒ best known distance from X to Y, via Z as next hop = \( D_Z(X,Y) \)
- Each local iteration caused by:
  - Local link cost change
  - Message from neighbor
- Notify neighbors only if least cost path to any destination changes
  - Neighbors then notify their neighbors if necessary

Distance Vector Algorithm (cont’d)

1. Initialization:
   - \( c(i,j) \): link cost from node i to j
   - \( D_{ij}(A,V) \): cost from A to V via Z
   - \( D(A,V) \): cost of A’s best path to V

2. for all neighbors V do
   - if V adjacent to A
     - \( D(A,V) = c(A,V) \)
   - else
     - \( D(A,V) = \infty \)
   - send \( D(A,Y) \) to all neighbors

3. loop:
   - wait (until A sees a link cost change to neighbor V /* case 1 */
   - or until A receives update from neighbor V) /* case 2 */
   - if (\( c(A,V) \) changes by \( \pm d \)) /* case 1 */
     - for all destinations Y that go through V do
       - \( D_V(A,Y) = D_V(A,Y) \pm d \)
   - else if (update \( D_V(A,Y) \) received from V) /* case 2 */
     - if (new minimum for destination Y)
       - send \( D(A,Y) \) to all neighbors

4. forever

Example: 1st Iteration (C ⇒ A)

```
Node A  B  C  A  C  D
B 2  8  A 2  =  =
C 7  =  =
D 8  =  =
```

```
Node B  A  C  D
A 2  =  =
C 1  =  =
D 3  =  =
```

```
Node C  A  B  D  A  B  C
A 7  =  =  A  =  =
B 1  =  =  B 3  =
D 9  =  =  C 1  =
```

```
Node D  A  B  C  D
A 2  =  =
C 1  =  =
D 3  =  =
```

```
D_{ij}(A,B) = D_{ij}(A,C) + D(C,B) = 7 + 1 = 8
D_{ij}(A,D) = D_{ij}(A,C) + D(C,D) = 7 + 1 = 8
```

```
D_{ij}(A,C) = D_{ij}(A,B) + D(B,C) = 2 + 1 = 3
D_{ij}(A,D) = D_{ij}(A,B) + D(B,D) = 2 + 3 = 5
```

Example: 1st Iteration (B ⇒ A)

```
Node A  B  C  A  C  D
B 2  8  A 2  =  =
C 3  7  =  =
D 5  8  =  =
```

```
Node B  A  C  D
A 2  =  =
C 1  =  =
D 3  =  =
```

```
Node C  A  B  D  A  B  C
A 7  =  =  A  =  =
B 1  =  =  B 3  =
D 9  =  =  C 1  =
```

```
Node D  A  B  C  D
A 2  =  =
B 1  =  =
C 3  =  =
```

```
D_{ij}(A,B) = D_{ij}(A,C) + D(C,B) = 7 + 1 = 8
D_{ij}(A,D) = D_{ij}(A,C) + D(C,D) = 7 + 1 = 8
```
Example: End of 1st Iteration

All nodes knows the best two-hop paths

Example: 2nd Iteration (A  B)

End of 2nd Iteration

All nodes knows the best three-hop paths

Example: End of 3rd Iteration

End of 2nd Iteration: Algorithm Converges!
**Distance Vector: Link Cost Changes**

Loop:
8. wait (until A sees a link cost change to neighbor V)
9. or until A receives update from neighbor V)
10. if (D(A,V) changes by ad) /* = case 1 */
11. for all destinations Y that go through V do
12. \( D(A,Y) = D(A,Y) + d \)
13. else if (update D(V,Y) received from V) /* = case 2 */
14. \( D(A,Y) = D(A) + D(V,Y) \)
15. if (there is a new minimum for destination Y)
16. send D(A,Y) to all neighbors
17. forever

**Distance Vector: Poisoned Reverse**

- If B routes through C to get to A:
  - B tells C its (B's) distance to A is infinite (so C won't route to A via B)
  - Will this completely solve count to infinity problem?

**Routing Information Protocol (RIP)**

- Simple distance-vector protocol
  - Nodes send distance vectors every 30 seconds
  - ... or, when an update causes a change in routing
- Link costs in RIP
  - All links have cost 1
  - Valid distances of 1 through 15
  - ... with 16 representing infinity
  - Small "infinity" ⇒ smaller "counting to infinity" problem
- RIP is limited to fairly small networks
  - E.g., campus
Link State vs. Distance Vector

Per-node message complexity:
- LS: $O(e)$ messages
  - $e$: number of edges
- DV: $O(d)$ messages, many times
  - $d$: node’s degree

Complexity/Convergence
- LS: $O(N \log N)$ computation
  - Requires global flooding
- DV: convergence time varies
  - Count-to-infinity problem

Robustness: what happens if router malfunctions?
- LS:
  - Node can advertise incorrect link cost
  - Each node computes only its own table
- DV:
  - Node can advertise incorrect path cost
  - Each node’s table used by others; errors propagate through network

Summary
- Routing is a distributed algorithm
  - Different from forwarding
  - React to changes in the topology
  - Compute the shortest paths
- Two main shortest-path algorithms
  - Dijkstra link-state routing (e.g., OSPF, IS-IS)
  - Bellman-Ford distance-vector routing (e.g., RIP)
- Convergence process
  - Changing from one topology to another
  - Transient periods of inconsistency across routers
- Next time: BGP
  - Reading: K&R 4.6.3