Shortest-Path Routing
Reading: Sections P&D 4.2

What we have learnt so far ...
• Sending IP packets from source to destination through a series of routers
  – Router looks up destination IP address in a table
  – Forwards the packet to the corresponding next hop
• Sending Ethernet frames from source to destination through a series of switches
  – Switch looks up destination MAC address in a table
  – Forwards the frame on the corresponding port
• What important task did we NOT talk about?
  – How are these forwarding tables filled with information?
  – Routing
• What about self-learning in switches?

What is Routing?
• A famous quotation from RFC 791
  “A name indicates what we seek.
  An address indicates where it is.
  A route indicates how we get there.”
  -- Jon Postel

Why Does Routing Matter?
• We need good end-to-end performance
  – Find the shortest/best path
    • Propagation delay, throughput, packet loss
• Ensure efficient use of network resources
  – Balance traffic over the routers and links
  – Avoid congestion by directing traffic to lightly-loaded links
• Withstand disruptions
  – Failures, maintenance, and load balancing
  – Limit packet loss and delay during changes

Know Thy Network
• 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 only local state
    – Every router knows only about its neighboring routers
    – Independently calculates routes
    – Problems with this approach?

Modeling a Network
• Modeled as a graph
  – Routers → nodes
  – Link → edges
    • Edge cost
    • delay
    • congestion level
• Goal of Routing
  – Determine a “good” path through the network from source to destination
  – Good means usually the shortest path
Link State Routing

- Each router has a complete picture of the network
- How does each router get the global state?
  - Each router reliably floods information about its neighbors to every other router (more later)
- Each router independently calculates the shortest path from itself to every other router
  - Dijkstra’s Shortest Path Algorithm

Dijkstra’s Shortest Path Algorithm

- Named after Edsger W. Dijkstra (1930-2002)
- INPUT
  - Net topology, link costs known to all nodes
- OUTPUT
  - Least cost paths from one node (“source”) to all other nodes

Notation

- \( c(i,j) \): link cost from node \( i \) to \( j \); cost infinite if not direct neighbors
- \( D(v) \): 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>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>2</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>3</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>4</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>5</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

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 \( D(v) = \infty \)

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>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>2</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>3</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>4</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>5</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

8. 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. \( D(v) = \min(D(v), D(w) + c(w,v)) \)
13. until all nodes in \( S \)
Example: Dijkstra’s Algorithm
Step start S D(B),p(B) D(C),p(C) D(D),p(D) D(E),p(E) D(F),p(F)
0 A 2,A 5,A 1,A ∞ ∞
1 AD 4,D 2,A ∞
2 ADE 3,E 2,D 4,E
3 ADEB
4 ADEBC
5

![Diagram of Dijkstra's Algorithm](image)

Steps:
1. Find w not in S s.t. D(w) is a minimum;
2. Add w to S;
3. Update D(v) for all v adjacent to w and not in S:
   - D(v) = min( D(v), D(w) + c(w,v) );
4. Until all nodes in S;

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>

- Why have separate routing and forwarding tables?

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))
Oscillations

• Assume link cost = amount of carried traffic

A D C B
1+e 0 e 1
initially
... recompute routing ... recompute ... recompute

• How can you avoid oscillations?

Obtaining Global State

• Flooding
  – Each router sends link-state information out through its links
  – The next node sends it out through all of its links
    • except the one where the information arrived

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

Transient Disruptions

• Inconsistent link-state database
  – Some routers know about failure before others
  – The shortest paths are no longer consistent
  – Can cause transient forwarding loops

A and D think that this is the path to C
E thinks that this is the path to C

Scaling Link-State Routing

• Overhead of link-state routing
  – Flooding link-state packets throughout the network
  – Running Dijkstra’s shortest-path algorithm
  – Becomes unscalable when 100s of routers
• Introducing hierarchy through “areas”
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 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

Bellman-Ford Algorithm

- Named after Richard Bellman and Ford
- INPUT
  - Link costs to each neighbor
- 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
- 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
- Each node:
  - wait for (change in local link cost or msg from neighbor)
  - recompute distance table
  - if least cost path to any dest has changed, notify neighbors

Distance Vector Algorithm (cont’d)

```
1 Initialization:
2 for all neighbors V do
3   if V adjacent to A
4     D(A, V) = c(A, V);
5   else
6     D(A, V) = ∞;
7   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,Y) changes by d)
11     for all destinations Y through V do
12       D(A,Y) = D(A,Y) + d
13 else if (update D(V, Y) received from V)
14     if (shortest path from V to some Y has changed Y)
15       D(A,Y) = D(A,V) + D(V, Y);
16     if (there is a new minimum for destination Y)
17       send D(A, Y) to all neighbors
18     forever
```

Example: Distance Vector Algorithm

<table>
<thead>
<tr>
<th>Node</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

Note: for simplicity in this lecture examples we show only the shortest distances to each destination
Example: 1st Iteration (C → A)

Example: 1st Iteration (B→A, C→A)

Example: End of 1st Iteration

Example: End of 2nd Iteration

Example: End of 3rd Iteration

Distance Vector: Link Cost Changes

Node A

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
B | 2 | B | A | 2 | A
C | 3 | B | C | 1 | C
D | 4 | B | D | 2 | C

Node B

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
B | 2 | B | A | 2 | A
C | 3 | B | C | 1 | C
D | 4 | B | D | 2 | C

Node C

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
A | 3 | B | A | 3 | B
B | 1 | B | B | 3 | B
D | 1 | D | C | 1 | C

Node D

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
A | 2 | B | A | 4 | C
B | 1 | B | B | 2 | C
D | 1 | D | C | 1 | C

Note:
- D(A, D) = D(A, B) + D(B, D) = 2 + 3 = 5
- D(A, C) = D(A, B) + D(B, C) = 2 + 1 = 3

Example: End of 3rd Iteration

Node A

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
B | 2 | B | A | 2 | A
C | 3 | B | C | 1 | C
D | 4 | B | D | 2 | C

Node B

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
B | 2 | B | A | 2 | A
C | 3 | B | C | 1 | C
D | 4 | B | D | 2 | C

Node C

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
A | 3 | B | A | 4 | C
B | 1 | B | B | 2 | C
D | 1 | D | C | 1 | C

Node D

Dest. | Cost | NextHop | Dest. | Cost | NextHop
--- | --- | --- | --- | --- | ---
A | 2 | B | A | 4 | C
B | 1 | B | B | 2 | C
D | 1 | D | C | 1 | C

Distances:
- D(A, D) = 5
- D(A, C) = 3
- D(A, B) = 2

Message Order:
Distance Vector: Count to Infinity Problem

- If A routes through B to get to C:
  - A tells B (C’s) distance to A is infinite
  - B won’t route to A via C

- Will this completely solve count to infinity problem?

Distance Vector: Poisoned Reverse

- If C routes through B to get to A:
  - C tells B (C’s) distance to A is infinite
  - B won’t route to A via C

- Will this completely solve count to infinity problem?

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: $O(n^2)$ computation
  - DV: convergence time varies
    - may be routing loops
    - 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; error propagates through network

Conclusions

- 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)
  - 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 and policy
  - Reading: Section 4.3.3, 4.3.4

Are we done?

- No. We still have to take care of:
  - Scaling to Internet size
  - Routing policy issues

- Next Lecture
  - Internet scale routing by Prof. Scott Shenker

Backup Slides

- To be covered if time permits
- Refer to textbook for more information about these topics
When to Initiate Flooding

• **Topology change**
  – Link or node failure
  – Link or node recovery

• **Configuration change**
  – Link cost change

• **Periodically**
  – Refresh the link-state information
  – Typically (say) 30 minutes
  – Corrects for possible corruption of the data

Convergence

• **Getting consistent routing information to all nodes**
  – E.g., all nodes having the same link-state database

• **Consistent forwarding after convergence**
  – All nodes have the same link-state database
  – All nodes forward packets on shortest paths
  – The next router on the path forwards to the next hop

Convergence Delay

• **Time elapsed before every router has a consistent picture of the network**

• **Sources of convergence delay**
  – Detection latency
  – Flooding of link-state information
  – Shortest-path computation
  – Creating the forwarding table

• **Performance during convergence period**
  – Lost packets due to blackholes and TTL expiry
  – Looping packets consuming resources
  – Out-of-order packets reaching the destination

• **Very bad for VoIP, online gaming, and video**

Reducing Convergence Delay

• **Faster detection**
  – Smaller hello timers
  – Link-layer technologies that can detect failures

• **Faster flooding**
  – Flooding immediately
  – Sending link-state packets with high-priority

• **Faster computation**
  – Faster processors on the routers
  – Incremental Dijkstra algorithm

• **Faster forwarding-table update**
  – Data structures supporting incremental updates

Detecting Topology Changes

• **Beaconing**
  – Periodic “hello” messages in both directions
  – Detect a failure after a few missed “hellos”

• **Performance trade-offs**
  – Detection speed
  – Overhead on link bandwidth and CPU
  – Likelihood of false detection