Shortest-Path Routing

EE 122: Intro to Communication Networks
Fall 2007 (WF 4-5:30 in Cory 277)
Vern Paxson
TAs: Lisa Fowler, Daniel Killebrew & Jorge Ortiz
http://inst.eecs.berkeley.edu/~ee122/
Materials with thanks to Jennifer Rexford, Ion Stoica, and colleagues at Princeton and UC Berkeley

Announcements
• Guest lecturer next Wednesday:
  Brighten Godfrey

• I will have office hours as usual next Friday, but not next Wednesday
  – Send email for an appointment Mt/Tu/Th
• Section next week will cover Spanning Tree algorithm (plus questions re routing algorithms)
  – Do attend (as usual!)

Goals of Today’s Lecture
• Routing vs. forwarding
  – “Control plane” vs. “Data plane”
• Link-state routing (Dijkstra’s algorithm)
  – Suitable as Interior Gateway Protocol (IGP)
    o i.e., used within a single ISP
• Topology change - detection & convergence
• Distance-vector routing (Bellman-Ford)
  o Suitable as Exterior Gateway Protocol (EGP)
    o Though really need something more
      • BGP - next lecture

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

Why Does Routing Matter?
• We need good end-to-end performance
  – Find the shortest/best path
    o 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, 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 correctly
  – 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?
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 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**

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**

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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</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>

1. Initialiation:
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

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 | 2A | 5A | 1A | 0 | 0 |
1 | AD | 4D | 2D | | | |
2 | ADE | 3E | | 4E | | |
3 | ADEBC | | | | | |

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. 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>2A</td>
<td>5A</td>
<td>1A</td>
<td>5C</td>
<td>5C</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>2A</td>
<td>5A</td>
<td>1A</td>
<td>5C</td>
<td>5C</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3E</td>
<td>4D</td>
<td>4</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td>3E</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>E</td>
</tr>
</tbody>
</table>

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
  - Note: need to remember previous msgs & suppress duplicates!
  - 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 K&R 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

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
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

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
  - Churn: having to keep recomputing the forwarding table as new information comes in

- 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

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

![Loop!]

A and D think that this is the path to C
E thinks that this is the path to C
Summary of Link-State Routing

• Each router keeps track of its incident links
  – Whether the link is up or down
  – The cost on the link
• Each router broadcasts the link state
  – To give every router a complete view of the graph
• Each router runs Dijkstra’s algorithm
  – To compute the shortest paths
  – ... and construct the forwarding table
• Example protocols
  – Open Shortest Path First (OSPF)
  – Intermediate System – Intermediate System (IS-IS)

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: “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

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 = \( D_{Z}(X,Y) \)
• 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)

Initialization:
1. Initialization:
2. for all neighbors V do
3. if V is adjacent to A
4. D(A, V) = c(A, V);
5. else
6. D(A, V) = ∞;
7. send D(A, V) to all neighbors
8. forever
9. if (c(A, V) changes by ±d) then case 1
10. for all destinations Y that go through V do
11. if (c(A, Y) changes by ±d) then case 2
12. if (update D(V, Y) from V) then case 2
13. if (there is a new minimum for destination Y)
14. execute:
15. forever

A

Example: Distance Vector Algorithm

Node A
Dest. | Cost | NextHop
---|---|---
B | 2 | B
C | 7 | C
D | ∞ | D

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

Note: for simplicity in this lecture the examples show only the shortest distances to each destination

Example: Distance Vector Algorithm

Node A
Dest. | Cost | NextHop
---|---|---
B | 2 | B
C | 7 | C
D | ∞ | D

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

Example: 1st Iteration (C → A)

Node A
Dest. | Cost | NextHop
---|---|---
A | 2 | A
C | 1 | C
D | ∞ | D

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

D(A, D) = D(A, C) + D(C, D) = 7 + 1 = 8
(D(C, A), D(C, B), D(C, D))

Example: 1st Iteration (C → A)

Node A
Dest. | Cost | NextHop
---|---|---
A | 2 | A
C | 1 | C
D | ∞ | D

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

D(A, D) = D(A, C) + D(C, D) = 7 + 1 = 8
(D(C, A), D(C, B), D(C, D))

Not an improvement
Example: End of 2nd Iteration

Node A

Dest. | Cost | NextHop
--- | --- | ---
B | 2 | B
C | 3 | B
D | 5 | B

Node B

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

Distance Vector:

Example: End of 3rd Iteration

Node A

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

Node B

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

Distance Vector: Link Cost Changes

Example: End of 2nd Iteration

Node A

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

Node B

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

Distance Vector: Count to Infinity Problem

Node B

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

Node C

Dest. | Cost | NextHop
--- | --- | ---
A | 4 | A
C | 5 | B
D | 2 | C

Distance Vector: Poisoned Reverse

Node A

Dest. | Cost | NextHop
--- | --- | ---
B | 60 | A
D | 50 | A

Node B

Dest. | Cost | NextHop
--- | --- | ---
A | 60 | A
C | 1 | C

Node C

Dest. | Cost | NextHop
--- | --- | ---
A | 51 | C
C | 1 | C

Node D

Dest. | Cost | NextHop
--- | --- | ---
A | 51 | C
C | 1 | C
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 is 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