Interior Gateway Protocols (RIP and OSPF)

What is Routing?

- To ensure information is delivered to the correct destination at a reasonable level of performance
- Forwarding
  - Given a forwarding table, move information from input ports to output ports of a router
  - Local mechanical operations
- Routing
  - Acquires information in the forwarding tables
  - Requires knowledge of the network
  - Requires distributed coordination of routers
Viewing Routing as a Policy

- Given multiple alternative paths, how to route information to destinations should be viewed as a policy decision
- What are some possible policies?
  - Shortest path (RIP, OSPF)
  - Most load-balanced
  - QoS routing (satisfies app requirements)
  - etc

Internet Routing

- Internet topology roughly organized as a two level hierarchy
- First lower level – autonomous systems (AS’s)
  - AS: region of network under a single administrative domain
- Each AS runs an intra-domain routing protocol
  - Distance Vector, e.g., Routing Information Protocol (RIP)
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Possibly others

- Second level – inter-connected AS’s
- Between AS’s runs inter-domain routing protocols, e.g., Border Gateway Routing (BGP)
  - De facto standard today, BGP-4
Why Need the Concept of AS or Domain?

- Routing algorithms are not efficient enough to deal with the size of the entire Internet
- Different organizations may want different internal routing policies
- Allow organizations to hide their internal network configurations from outside
- Allow organizations to choose how to route across multiple organizations (BGP)
- Basically, easier to compute routes, more flexibility, more autonomy/independence
Outline

- Two intra-domain routing protocols
- Both try to achieve the “shortest path” routing policy
- Quite commonly used

- OSPF: Based on Link-State routing algorithm
- RIP: Based on Distance-Vector routing algorithm

Intra-domain Routing Protocols

- Based on unreliable datagram delivery
- Distance vector
  - Routing Information Protocol (RIP), based on Bellman-Ford algorithm
  - Each neighbor periodically exchange reachability information to its neighbors
  - Minimal communication overhead, but it takes long to converge, i.e., in proportion to the maximum path length
- Link state
  - Open Shortest Path First (OSPF), based on Dijkstra’s algorithm
  - Each router periodically floods immediate reachability information to other routers
  - Fast convergence, but high communication and computation overhead
Routing on a Graph

- Goal: determine a “good” path through the network from source to destination
  - Good often means the shortest path
- Network modeled as a graph
  - Routers → nodes
  - Link → edges
    - Edge cost: delay, congestion level,…

Link State Routing (OSPF): Flooding

- Each node knows its connectivity and cost to a direct neighbor
- Every node tells every other node this local connectivity/cost information
  - Via flooding
- In the end, every node learns the complete topology of the network
- E.g. A floods message

A connected to B cost 2
A connected to D cost 1
A connected to C cost 5
Flooding Details

- Each node periodically generates Link State Packet (LSP) contains
  - ID of node created LSP
  - List of direct neighbors and costs
  - Sequence number (64 bit, assume to never wrap around)
  - Time to live
- Flood is reliable
  - Use acknowledgement and retransmission
- Sequence number used to identify *newer* LSP
  - An older LSP is discarded
  - What if a router crash and sequence number reset to 0?
- Receiving node flood LSP to all its neighbors except the neighbor where the LSP came from
- LSP is also generated when a link’s state changes (failed or restored)

Peterson & Davie

- P.285
- …[the crashed node] will eventually receive a copy of its own LSP with a higher sequence number,…
Link State Flooding Example

Diagram of a network with labeled nodes and links.
Link State Flooding Example
A Link State Routing Algorithm

Dijkstra’s algorithm
- Net topology, link costs known to all nodes
  - Accomplished via “link state flooding”
  - All nodes have same info
- Compute least cost paths from one node (‘source’) to all other nodes
- Repeat for all sources

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

---

Dijkstra’s Algorithm (A “Greedy” 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. \( D(v) = \min( D(v), D(w) + c(w,v) ) \);
       // new cost to \( v \) is either old cost to \( v \) or known
       // shortest path cost to \( w \) plus cost from \( w \) to \( v \)
   13. **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>A</td>
<td>D(B),p(B)</td>
<td>4,D</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></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>

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) = ∞;
   
   ...
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>4,E</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. \(D(v) = \min(D(v), D(w) + c(w,v))\);
13. **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>4,E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</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. \(D(v) = \min(D(v), D(w) + c(w,v))\);
13. **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>4,E</td>
<td></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>ADEBC</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 \[ D(v) = \min(D(v), D(w) + c(w,v)) \];
13 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>4,E</td>
<td></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>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</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 \[ D(v) = \min(D(v), D(w) + c(w,v)) \];
13 until all nodes in S;
Distance Vector Routing (RIP)

- What is a distance vector?
  - Current best known cost to get to a destination
- Idea: Exchange distance vectors among neighbors to learn about lowest cost paths

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
</tr>
</tbody>
</table>

Note no vector entry for C itself

At the beginning, distance vector only has information about directly attached neighbors, all other dests have cost $\infty$

Eventually the vector is filled

Distance Vector Routing Algorithm

- Iterative: continues until no nodes exchange info
- Asynchronous: nodes need not exchange info/iterate in lock steps
- Distributed: each node communicates only with directly-attached neighbors
- Each router maintains
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row $Y$ and column $Z$ of node $X \Rightarrow$ best known distance from $X$ to $Y$, via $Z$ as next hop

*Note: for simplicity in this lecture examples we show only the shortest distances to each destination*
Distance Vector Routing

- Each local iteration caused by:
  - Local link cost change
  - Message from neighbor: its least cost path change from neighbor to destination

- Each node notifies neighbors only when its 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) = \infty;
7         loop:
8         wait (until A sees a link cost change to neighbor V or until A receives update from neighbor V)
9       if (D(A,V) changes by d)
10          for all destinations Y through V do
11             D(A,Y) = D(A,Y) + d
12       else if (update D(V, Y) received from V)
13           /* shortest path from V to some Y has changed */
14             D(A,Y) = D(A,V) + D(V, Y);
15       if (there is a new minimum for destination Y)
16         send D(A, Y) to all neighbors
17     forever
Example: Distance Vector Algorithm

**Initialization:**

1. for all neighbors \( V \) do
2. if \( V \) adjacent to \( A \)
3. \( D(A, V) = c(A, V) \);
4. else
5. \( D(A, V) = \infty \);
6. \( D(A, V) = \infty \);

...
Example: 1st Iteration (B→A, C→A)

Node A

\[ D(A,D) = D(A,B) + D(B,D) = 2 + 3 = 5 \]

\[ D(A,C) = D(A,B) + D(B,C) = 2 + 1 = 3 \]

Node B

Node C

Node D

Example: End of 1st Iteration

Node A

Node B

Node C

Node D
### Example: End of 2\textsuperscript{nd} Iteration

<table>
<thead>
<tr>
<th>Node A</th>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node B</th>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

### Example: End of 3\textsuperscript{rd} Iteration

<table>
<thead>
<tr>
<th>Node A</th>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node B</th>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node C</th>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node D</th>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

Nothing changes $\rightarrow$ algorithm terminates
**Distance Vector: Link Cost Changes**

<table>
<thead>
<tr>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 4 A</td>
<td>A 1 A</td>
</tr>
<tr>
<td>C 1 B</td>
<td>C 1 B</td>
</tr>
</tbody>
</table>

**Algorithm:**

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,V) 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 D(A,Y) = D(A,V) + D(V,Y);
15 if (there is a new minimum for destination Y)
16 send D(A,Y) to all neighbors
17 forever

<table>
<thead>
<tr>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 5 C</td>
<td>A 2 B</td>
</tr>
<tr>
<td>B 1 B</td>
<td>B 1 B</td>
</tr>
</tbody>
</table>

“good news travels fast”

**Distance Vector: Count to Infinity Problem**

<table>
<thead>
<tr>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 4 A</td>
<td>A 6 C</td>
</tr>
<tr>
<td>C 1 B</td>
<td>C 1 B</td>
</tr>
</tbody>
</table>

**Algorithm:**

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,V) 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 D(A,Y) = D(A,V) + D(V,Y);
15 if (there is a new minimum for destination Y)
16 send D(A,Y) to all neighbors
17 forever

<table>
<thead>
<tr>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 5 B</td>
<td>A 7 B</td>
</tr>
<tr>
<td>B 1 B</td>
<td>B 1 B</td>
</tr>
</tbody>
</table>

“bad news travels slowly”

Link cost changes here; recall that B also maintains shortest distance to A through C, which is 6. Thus D(B,A) becomes 6!
### Distance Vector: Poisoned Reverse

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

**Link cost changes here:** B updates $D(B, A) = 60$ as C has advertised $D(C, A) = \infty$

**Algorithm terminates**

---

### Link State vs. Distance Vector

**Per node message complexity**
- **LS:** $O(e)$ messages; $n$ – number of nodes; $e$ – number of edges
- **DV:** $O(d)$ messages; where $d$ is node’s degree

**Complexity**
- **LS:** $O(n^2)$ with $O(n^e)$ messages
- **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 propagate through network
Oscillations

- Assume link cost = amount of carried traffic

- How can you avoid oscillations?

Reference

- T. S. Eugene Ng Slides on Interior Gateway Protocols, Rice University