Intra-domain Routing

Outline

Introduction to Routing
Distance Vector Algorithm
Goal

• Build router forwarding tables in an internetwork using intra-domain routing protocols

• High level approach
  – Distributed Execution
  – Send messages about network links
  – Each router uses this info to compute routes
Forwarding versus Routing

- **Forwarding:**
  - to select an output port based on destination address and routing table
- **Routing:**
  - process by which routing table is built
Routing

• Forwarding table VS Routing table
  • Forwarding table
    • Used when a packet is being forwarded and so must contain enough information to accomplish the forwarding function
    • A row in the forwarding table contains the mapping from a network number to an outgoing interface and some MAC information, such as Ethernet Address of the next hop
  • Routing table
    • Built by the routing algorithm as a precursor to build the forwarding table
    • Generally contains mapping from network numbers to next hops
Routing

• Really, we are computing paths to networks
• Network as a Graph
  • Assume single admin. authority
  • Assume nodes are routers
  • Assume links have cost
    – This cost could be latency of the link, or a function of link bandwidth
    – Could also have a dynamic model for costs

• The basic problem of routing is to find the lowest-cost path between any two nodes
  • Where the cost of a path equals the sum of the costs of all the edges that make up the path
Routing Protocol Issues

- It may be simple to calculate least cost path if graph is static but…
- Links and routers go down
- Links and routers are added
- Traffic can cause links to overload
- How are costs calculated?
- Algorithm must be distributed in order to scale
- Rich area for research due to distributed, dynamic nature of the problem
- Different routers can have different routes at same time
Routing

• For a simple network, we can calculate all shortest paths and load them into some nonvolatile storage on each node.

• Such a static approach has several shortcomings
  • It does not deal with node or link failures
  • It does not consider the addition of new nodes or links
  • It implies that edge costs cannot change

• What is the solution?
  • Need a distributed and dynamic protocol
  • Two main classes of protocols
    • Distance Vector
    • Link State
Distance Vector

- Each node constructs a one dimensional array (a vector) containing the “distances” (costs) to all other nodes and distributes that vector to its immediate neighbors.
- Starting assumption is that each node knows the cost of the link to each of its directly connected neighbors.
Distance Vector

Initial distances stored at each node (global view)
Distance Vector

Initial routing table at node A
## Distance Vector

![Graph Diagram]

### Final routing table at node A

<table>
<thead>
<tr>
<th>Destination</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>F</td>
</tr>
</tbody>
</table>
Distance Vector

### Information Stored at Node

<table>
<thead>
<tr>
<th>Information Stored at Node</th>
<th>Distance to Reach Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
</tr>
</tbody>
</table>

Final distances stored at each node (global view)
Distance Vector

• The distance vector routing is based on the Bellman-Ford algorithm

• Every T seconds each router sends a list of distances to all the routers to its neighbor

• Each router then updates its table based on the new information

• Problems include fast response to good news and slow response to bad news. Also too many messages to update
Distance Vector

• When a node detects a link failure
  • F detects that link to G has failed
  • F sets distance to G to infinity and sends update to A
  • A sets distance to G to infinity since it uses F to reach G
  • A receives periodic update from C with 2-hop path to G
  • A sets distance to G to 3 and sends update to F
  • F decides it can reach G in 4 hops via A
• Slightly different circumstances can prevent the network from stabilizing
  – Suppose the link from A to E goes down
  – In the next round of updates, A advertises a distance of infinity to E, but B and C advertise a distance of 2 to E
  – Depending on the exact timing of events, the following might happen
    • Node B, upon hearing that E can be reached in 2 hops from C, concludes that it can reach E in 3 hops and advertises this to A
    • Node A concludes that it can reach E in 4 hops and advertises this to C
    • Node C concludes that it can reach E in 5 hops; and so on.
    • This cycle stops only when the distances reach some number that is large enough to be considered infinite
    – Count-to-infinity problem
Count-to-infinity Problem

• Use some relatively small number as an approximation of infinity

• For example, the maximum number of hops to get across a certain network is never going to be more than 16
Bellman Ford Algorithm

1 for v in V:
2   v.dist = infinity
3   v.p = None
4 source.distance = 0
5 for i from 1 to |V| - 1:
6   for (u, v) in E:
7     relax(u, v)

// Detect negative weight cycles
8 for (u, v) in E:
9   if v.dist > u.dist + w(u, v):
10      print "A negative weight cycle exists"

relax(u, v):
1   if v.dist > u.dist + w(u, v):
2       v.dist = u.dist + w(u, v)
3       v.p = u
In a stronger version of split horizon, called *split horizon with poison reverse*,
- B sends that back route to A, but it puts negative information in the route
to ensure that A will not eventually use B to get to E
- For example, B sends the route \((E, \infty)\) to A
**Count-to-infinity Problem**

<table>
<thead>
<tr>
<th>Dst.</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dst.</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>2</td>
</tr>
</tbody>
</table>

**Initial State**

- **B** to **D** link goes down

- **B** receives **A**’s advertisement before **B** can advertise to **A**.
- **B** finds shorter route to **S** via **A**.

<table>
<thead>
<tr>
<th>Dst.</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dst.</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

- **B** advertises updated table to **A**.
- **A** registers change in path length to **S** via **B**.
- **A** accordingly updates no. of hops to **S**.

<table>
<thead>
<tr>
<th>Dst.</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dst.</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

- **A** advertises updated table to **B**.
- **B** registers change in path length to **S** via **A**.
- **B** accordingly updates no. of hops to **S**.

This back-and-forth of advertisements between **A** and **B** continues => path lengths to **S** keep increasing => count-to-infinity
Split Horizon with Poison Reverse

- When a node sends a routing update to its neighbors, it does not send those routes it learned from each neighbor back to that neighbor.

- Old advertisement from $A$ to $B$ without split horizon/poison reverse.

- Split horizon with poison reverse.
  - Advertisement from $A$ to $B$ has infinite path length to $S$.
  - This prevents $B$ from using that path.
Routing Information Protocol (RIP)

Example Network running RIP

RIPv2 Packet Format
Intradomain Routing Summary

• Intradomain routing protocols determine how forwarding tables are maintained in routers
  – Least cost algorithms

• Distance vector routing
  – Algorithm based on building forwarding table by distributing vector of distances to neighbors
  – Completely distributed and based only on knowledge of immediate neighbors
  – Known to converge under static conditions
  – Count to infinity problem
  – Limited network diameter
A bit more on DV and Poisoned Reverse
When is Distance Vector Good: Good News Travels Quickly

- When costs decrease, network converges quickly
When is it Bad: Bad News Travels Slowly

Note also that there is a forwarding loop between $y$ and $z$. 
It Gets Worse

- **Question**: How long does this continue?
- **Answer**: Until z’s path cost to x via y is greater than 50.
“Solution”: Poison Reverse

- If z routes through y to get to x, z advertises infinite cost for x to y
- Does poison reverse always work?
Does Poison Reverse Always Work?
Use in RIP

Routing Table
1.1.1.0 /24   Fa0/0  0
192.168.12.0 /24   Fa1/0  0

Routing Table
192.168.12.0 /24   Fa0/0  0
192.168.23.0 /24   Fa1/0  0

Routing Table
192.168.23.0 /24   Fa0/0  0
3.3.3.0 /24   Fa1/0  0
Use in RIP

Routing Table
1.1.1.0 /24  Fa0/0  0
192.168.12.0 /24  Fa1/0  0
192.168.23.0 /24  Fa1/0  0

Routing Table
192.168.12.0 /24  Fa0/0  0
192.168.23.0 /24  Fa1/0  0

Routing Table
192.168.23.0 /24  Fa0/0  0
3.3.3.0 /24  Fa1/0  0

Routing Table
192.168.12.0 /24  Fa0/0  0
192.168.23.0 /24  Fa1/0  0
1.1.1.0 /24  Fa0/0  1
3.3.3.0 /24  Fa1/0  1

Routing Table
192.168.23.0 /24  Fa0/0  0
3.3.3.0 /24  Fa1/0  0
1.1.1.0 /24  Fa0/0  2
RIP details

Interval 30 seconds intervals to exchange distance vectors

Uses UDP for messaging

Operates on port 520