To Autolab or Not to Autolab

- Autolab is supposed to make life easier
  - Give you confidence that you will credit for tests
- But autolab is an artificial execution environment
  - Isolated: no contact with other servers, file systems, ..
  - Strange timing: packet latencies can become very unpredictable and (possibly) unrealistic
  - It is very difficult (~impossible) to debug in autolab
- Recommendation: try to get tests to work in autolab but limit how much time you spend on it
  - You will get a chance to run the tests over Andrew, without loss of credit

To Take or Not to Take a Late Penalty

- Nobody likes to miss a deadline
  - But is happens, especially when you are overcommitted
- Put it in perspective: the penalty (in this course) is generally not a big deal
  - For assignment A: % of A x weight of A x 15%
  - CP2 of P1: ~0.75% of points for the course
- Think strategically:
  - Final project submissions: you can use a late day
  - The rest: move on and learn from the experience
- Biggest concern: cascading missed deadlines

To Use or Not to Use a Late Day

- You don't take late days
- At the end of the course, we look at what assignments you were late for and we use your late days so it maximizes your grade
  - You can also "suggest" what you think is optimal
  - It is to your advantage to keep track of this so you can make good decisions during the semester
Outline

- Routing intro
- Distance Vector
- Link State

Graph Model

- Represent each router as node
- Direct link between routers represented by edge
  - Symmetric links ⇒ undirected graph
- Edge “cost” \(c(x,y)\) denotes measure of difficulty of using link
  - delay, $\$, cost, or congestion level
- Task
  - Determine least cost path from every node to every other node
    - Path cost \(d(x,y) = \text{sum of link costs}\)

Ways to Compute Shortest Paths

- Centralized
  - Collect graph structure in one place
  - Use standard graph algorithm
  - Disseminate routing tables
- Link-state
  - Every node collects complete graph structure
  - Each computes shortest paths from it
  - Each generates its own routing table
- Distance-vector
  - No one has copy of graph
  - Nodes construct their own tables iteratively
  - Each sends information about its table to neighbors

Routes from Node A

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>5</td>
<td>E</td>
</tr>
</tbody>
</table>

- Set of shortest paths forms a tree
  - Shortest path spanning tree
- Solution is not unique
  - E.g., A-E-F-C-D also has cost 7
Routing Hierarchy

- IP packets must travel across domains – inter-domain routing
  - Primary role of IP
  - Based on CIDR prefix
- Must also travel through domains – intro-domain routing
  - Across subnets
  - Based on subnet ID or longer prefix

Outline

- Routing intro
- Distance Vector
- Link State

Distance-Vector Method

- Idea
  - At any time, have cost/next hop of best known path to destination
  - Use cost $\infty$ when no path known
- Initially
  - Only have entries for directly connected nodes

Distance-Vector Update

- Update(x,y,z)
  \[ d \leftarrow c(x,z) + d(z,y) \]
  - # Cost of path from x to y with first hop z
  - if $d < d(x,y)$
    - # Found better path
    - return $d,z$
  - else
    - # Existing cost / next hop
    - return $d(x,y), \text{nexthop}(x,y)$
Algorithm

- Bellman-Ford algorithm
- Repeat
  For every node x
  For every neighbor z
  For every destination y
  \( d(x, y) \leftarrow \text{Update}(x, y, z) \)
- Until converge
**Distance Vector: Link Cost Changes**

- Node detects local link cost change
- Updates distance table
- If cost change in least cost path, notify neighbors

### Table at Node Y

<table>
<thead>
<tr>
<th>Node</th>
<th>Distance to X via Y</th>
<th>Distance to Y via Y</th>
<th>Distance to Z via Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>6</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

### Table at Node Z

<table>
<thead>
<tr>
<th>Node</th>
<th>Distance to X via Z</th>
<th>Distance to Y via Z</th>
<th>Distance to Z via Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Y</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Z</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

**Distance Vector: Link Cost Changes**

- Good news travels fast
- Bad news travels slowly - “count to infinity” problem!

### Table at Node Y

<table>
<thead>
<tr>
<th>Node</th>
<th>Distance to X via Y</th>
<th>Distance to Y via Y</th>
<th>Distance to Z via Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>6</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

### Table at Node Z

<table>
<thead>
<tr>
<th>Node</th>
<th>Distance to X via Z</th>
<th>Distance to Y via Z</th>
<th>Distance to Z via Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Y</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Z</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

**Bad News Travels Slowly**

Is this a problem? Yes!
- After a path cost increases, it can take a very long time before paths stabilize, and
- During this process, the network has a routing loop

What is the cause?
- Nodes refuse to accept the up-to-date information, because they prefer the older, better cost
- Outdated information based on the older, lower path cost loops around the network

**Distance Vector: Split Horizon**

Problem: if Z routes through Y to get to X, it still advertises its path back to Y
- This serves no purpose and causes the loops

Solution: Z does not advertise its route back to Y

### Table at Node Y

<table>
<thead>
<tr>
<th>Node</th>
<th>Distance to X via Y</th>
<th>Distance to Y via Y</th>
<th>Distance to Z via Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>6</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

### Table at Node Z

<table>
<thead>
<tr>
<th>Node</th>
<th>Distance to X via Z</th>
<th>Distance to Y via Z</th>
<th>Distance to Z via Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Y</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Z</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>
Poison Reverse Failures

- Split horizon does not help!
- Especially bad if a link goes down: “Count to infinity”
- Solution:
  - Make “infinity” smaller
  - Force the cost “infinity” to all interfaces and wait
  - Helps network converge faster

Routing Information Protocol (RIP)

- Earliest IP routing protocol (1982 BSD)
- Current standard is version 2 (RFC 1723)
- Features
  - Every link has cost 1
  - “Infinity” = 16 - Limits network diameter to 15 hops
- Routers exchange different types of updates
  - Initial: asks for copy of table for every neighbor when it starts
  - Uses it to iteratively generate own table
  - Periodic: sends copy of its table to each neighbor every 30 sec
  - Neighbors use it to iteratively update their tables
  - Triggered: send copy of entry to neighbors when entry changes
  - Except for one causing update (split horizon rule)
  - Neighbors use it to update their tables

Outline

- Routing intro
- Distance Vector
- Link State

Link State Protocol Concept

- Every node gets complete copy of graph
  - Every node “floods” network with data about its outgoing links
- Every node computes routes to every other node
  - Using single-source, shortest-path algorithm
- Process performed whenever needed
  - When connections die / reappear
Sending Link States by Flooding

- X Wants to Send Information
  - Sends on all outgoing links
- When Node Y Receives Information from Z
  - Send on all links other than Z

Dijkstra’s Algorithm

- Given
  - Graph with source node \( s \) and edge costs \( c(u,v) \)
  - Determine least cost path from \( s \) to every node \( v \)
- Shortest Path First Algorithm
  - Traverse graph in order of least cost from source

Dijkstra’s Algorithm: Concept

- Node Sets
  - Done
    - Already have least cost path to it
  - Horizon:
    - Reachable in 1 hop from node in Done
  - Unseen:
    - Cannot reach directly from node in Done
- Label
  - \( d(v) = \) path cost from \( s \) to \( v \)
- Path
  - Keep track of last link in path

Dijkstra’s Algorithm: Initially

- No nodes done
- Source in horizon
Dijkstra’s Algorithm: Initially

- \( d(v) \) to node A shown in red
- Only consider links from done nodes

Dijkstra’s Algorithm

- Select node \( v \) in horizon with minimum \( d(v) \)
- Add link used to add node to shortest path tree
- Update \( d(v) \) information

Dijkstra’s Algorithm

- Repeat…

Dijkstra’s Algorithm

- Update \( d(v) \) values
- Can cause addition of new nodes to horizon
Dijkstra’s Algorithm

- Final tree shown in green

Link State Characteristics

- With consistent LSDBs**, all nodes compute consistent loop-free paths
- Can still have transient loops

OSPF Routing Protocol

- Open standard created by IETF
- Shortest-path first
  - Another name for Dijkstra’s algorithm
- Replaced RIP
  - RIP is dated, given today’s requirements
  - OSPF has fast convergence when configuration changes
  - OSPF can scale to very large networks using “areas”

OSPF Reliable Flooding

- Transmit link state advertisements
  - Originating router
    - Typically, minimum IP address for router
  - Link ID
    - ID of router at other end of link
  - Metric
    - Cost of link
  - Link-state age
    - Incremented each second
    - Packet expires when reaches 3600
  - Sequence number
    - Incremented each time sending new link information
Flooding Issues

- When should it be performed
  - Periodically
  - When status of link changes
    - Detected by connected node
- What happens when router goes down & back up
  - Sequence number reset to 0
  - Other routers may have entries with higher sequence numbers
  - Router will send out LSAs with number 0
  - Will get back LSAs with last valid sequence number p
  - Router sets sequence number to p+1 & resends

Areas: Scaling to Larger Networks

- Within area: Each node has routes to every other node
- Outside area: Each node has routes for other top-level areas only
  - Inter-area packets are routed to nearest border router
  - Constraint: no path between two sub-areas of an area can exit that area
  - May no longer have shortest path routes

Comparison of LS and DV Algorithms

<table>
<thead>
<tr>
<th>Message complexity</th>
<th>Space requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LS:</strong> with n nodes, E links, O(nE) messages</td>
<td><strong>LS:</strong> Node can advertise incorrect link cost</td>
</tr>
<tr>
<td><strong>DV:</strong> exchange between neighbors only</td>
<td><strong>Each node computes its own table</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speed of Convergence</th>
<th>Robustness: router malfunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LS:</strong> Relatively fast</td>
<td><strong>LS:</strong> Node can advertise incorrect link cost</td>
</tr>
<tr>
<td>Complex computation, but can forward before computation</td>
<td>Each node computes its own table</td>
</tr>
<tr>
<td>may have transient loops</td>
<td><strong>DV:</strong> Node can advertise incorrect path cost</td>
</tr>
<tr>
<td><strong>DV:</strong> convergence time varies</td>
<td>Each node’s table used by others (error propagates)</td>
</tr>
<tr>
<td>may have routing loops</td>
<td>Faster with triggered updates</td>
</tr>
<tr>
<td>count-to-infinity problem</td>
<td></td>
</tr>
</tbody>
</table>