Distance Vector & RIP

Brad Smith
Logistics

• Today
  • *Project proposals due!*

• Thursday
  – Distance-Vector routing exercise
  – Intra-Domain Routing quiz

• Sunday
  – OSPF lab due

• Next Tuesday
  – Spanning-Tree Protocol (STP) lecture

• Dordal readings
  • STP – Sections 2.5
Project Deliverables

• Proposal
  – Due today… e-mail (same time as lab reports)
  – A document (pdf) including
    • Describe topic you will develop a lab for
    • Draft outline of what you plan to include in the lab
    • What you need to investigate
  – Remember… links to command references are on web site…

• Deliverables
  – Presentation… 10 mins w/ some time for questions
  – Turn in… by the day of our final slot (Thursday, June 15th)
    • Slides from presentation
    • Paper describing
      – Technology covered in the lab
      – Lessons learned
    • Lab, answer key, netref content
Classifying Routing Protocols

• **Function:**
  – Intra-domain/Interior Gateway Protocol (IGP)
  – Inter-domain/Exterior Gateway Protocol (EGP)

• **Algorithm**... distinguished by information exchanged:
  – Distance-Vector
  – Link-State
  – Path-Vector
Functional Classification

• An **autonomous system** (AS) or **routing domain** is a region of the Internet that is administered by a single entity
  – UCSC’s network
  – IBM’s corporate network
  – AT&T’s ISP network

• Routing inside an AS
  – Focus is on performance
  – Popular protocols: RIP, OSPF, IS-IS
  – Called **intra-domain** or **internal gateway (IGP)** routing
  – Based on shortest-path routing (link-state and distance vector)

• Routing between ASs
  – Focus is on policy
  – Popular protocol: BGP
  – Called **inter-domain** or **external gateway (EGP)** routing
  – Based on path-vector routing
Algorithm Classification

• **Distance-Vector** – “sign-post”
  – Vectors of destination and distance sent to neighbors
  – “Tell your neighbors about the rest of the network”
  – Destination in terms of a network prefix
  – Distance in terms of a metric: hop count, delay, bandwidth
  – Use **Distributed Bellman-Ford** path selection algorithm
  – Popular protocol: Routing Information Protocol (RIP)

• **Link-State** – “map”
  – Flood description of your links (link state) to whole network
  – “Tell the rest of the network about your neighbors”
  – Links described by
    – End-point routers of subnet in internet
    – Cost of subnet: delay, bandwidth
  – Use **Dijkstra** path selection algorithm
  – Popular protocol: Open Shortest Path First (OSPF)

• **Path-Vector**
  – Routes advertised as full-paths
  – Paths described by sequence of ASs
  – Popular protocol is Border Gateway Routing Protocol (BGP)
Routing Algorithms

• **Distance-Vector** – “sign-post”
  – Vectors of destination and distance sent to neighbors
    • “Tell your neighbors about the rest of the network”
  – Destination in terms of a network prefix
  – Distance in terms of a metric: hop count, delay, bandwidth
  – Use Distributed Bellman-Ford path selection algorithm
  – Popular protocol: Routing Information Protocol (RIP)

• **Link-State** – “map”
  – Flood description of your links (link state)
    • “Tell the rest of the network about your neighbors”
  – Links described by
    • End-point routers of subnet in internet
    • Cost of subnet: delay, bandwidth
  – Use Dijkstra path selection algorithm
  – Popular protocol: Open Shortest Path First (OSPF)

• **Path-Vector**
  – Routes advertised as full-paths
  – Paths described by sequence of ASs
  – Popular protocol is Border Gateway Routing Protocol (BGP)
Link-State Protocols

• Are Interior-Gateway Protocols (IGPs)

• Exchange link-state information
  – Pair of routers connected by a subnet
  – Cost of subnet (hop count, delay, etc.)

• Conceptually, very simple...
Link-State Protocols

- Maintains a topology database of all the links it has heard of
  - Initialize with the subnets it is connected to.

- Floods link-state updates describing its directly connected subnets, including any changes to these links.
  - “Tell the rest of the network about your neighbors”

- Participates in the flooding of link-state updates from other routers.

- On update of its topology database
  - Runs a shortest-path first (SPF) algorithm on the database to compute routes
    - Dijkstra is most efficient
  - Updates its forwarding table with any changes.
Characterizing Link State

**Link-State**

- # updates per link change?
  - One.

- How far propagate updates?
  - Flooded to all nodes.

- One update, global distribution.

- Scaling problems due to flooding

- *As we’ll see later, the characteristics of distance vector…*
  - *...are very different*
  - *...hint at a much better solution*
Distance-Vector
Algorithm Classification

• **Distance-Vector** – “sign-post”
  – Vectors of destination and distance sent to neighbors
    • “Tell your neighbors about the rest of the network”
  – Destination in terms of a network prefix
  – Distance in terms of a metric: hop count, delay, bandwidth
  – Use Distributed Bellman-Ford path selection algorithm
  – Popular protocol: Routing Information Protocol (RIP)

• **Link-State** – “map”
  – Flood description of your links (link state)
    • “Tell the rest of the network about your neighbors”
  – Links described by
    • End-point routers of subnet in internet
    • Cost of subnet: delay, bandwidth
  – Use Dijkstra path selection algorithm
  – Popular protocol: Open Shortest Path First (OSPF)

• **Path-Vector**
  – Routes advertised as full-paths
  – Paths described by sequence of ASs
  – Popular protocol is Border Gateway Routing Protocol (BGP)
Distance-Vector Protocols

- Are Interior-Gateway Protocols (IGPs)
- Exchange vectors of distances
  - Destination router
  - Cost of path to router (hop count, delay, etc.)
- Conceptually, quite a bit subtler…

- Need to understand underlying Bellman-Ford SPF algorithm to understand how distance-vector works…
Overview

• Bellman-Ford shortest path algorithm
• Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)
• DBF’s big weakness… counting-to-infinity
• RIP details
Overview

• Bellman-Ford shortest path algorithm

• Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)

• DBF’s big weakness… counting-to-infinity

• RIP details
Shortest-Path Routing

• Shortest-path spanning tree is computed for each router using a shortest-path first (SPF) algorithm.

• Internet routing is based on two SPF algorithms
  – Dijkstra – used in link-state protocols
  – Bellman-Ford – used in distance-vector protocols
SPF Algorithms

• Search the paths in a graph based on an increasing metric

• Dijkstra and Bellman-Ford differ by their metric
  – Dijkstra iterates on shortest path
  – Bellman-Ford iterates on hop count

• Each has their benefits
  – Dijkstra is very efficient
  – Bellman-Ford supports a distributed implementation... the basis for DV
Dijkstra Shortest-Path Algorithm

- Search paths, *by increasing path cost*, for best paths to all destinations. Terminate when path has been found for all destinations.

- Maintain two sets
  - Destinations for which shortest paths have been found.
    - Permanently labeled destinations $P$
    - Initialize with self
  - Destinations for which candidate shortest paths have been found.
    - Temporarily labeled destinations $T$
    - Initialize with my neighbors.

- Iterate
  - Move shortest path in $T$, say for destination $D$, to $P$
  - Add routes for $D$’s neighbors, that are extensions of the path to $T$, to $T$ if they are shorter than the current path in $T$ for each neighbor. The “relaxation” step.
  - Repeat until a route has been added to $P$ for all destinations
More formally...

- \( E \) is the set of edges.
- \( w_{ij} \) is the weight of the link between nodes \( i \) and \( j \).
- \( P \) and \( T \)... see previous slide.
- \( P \) and \( T \) entries are triples, \( <d, p, w> \):
  - \( d \) is the destination
  - \( p \) is the predecessor
  - \( w \) is the link weight

```plaintext
algorithm Dijkstra
begin
1 Push(\(<i, i, 0>, P>\);
2 for each \( \{(i, j) \in E\} \)
3 Insert(\(<j, i, w_{ij}>, T>\);
4 while \(|T| > 0\)
   begin
5 \(<x, p_x, d_x> \leftarrow \text{Min}(T)\);
6 DeleteMin(T);
7 Push(\(<x, p_x, d_x>, P>\);
8 for each \( \{(x, j) \in E\} \)
9   if \((T_j = \emptyset)\)
10      then Insert(\(<j, x, d_x + \omega_{xj}>, T>\)
11     else if \((d_x + \omega_{xj} < T_j.d_j)\)
12       then DecreaseKey(\(<j, x, d_x + \omega_{xj}>, T>\);
   end
end
```

**Figure 2.** Traditional Dijkstra Shortest-Path Algorithm.
Dijkstra in pictures

Push <dest = i, pred = i, dist = 0> onto T

while T is not Empty
    Pop <x, p, d> from T
    if P[x] has not been assigned
        P[x] = <x, p, d>
        for each n in Neighbors(x)
            Push <n, x, d + w(x,n)> onto T
Dijkstra... increasing *path cost*

Spring 2017

CE 151 - Advanced Networks 23
SPF Algorithms and Protocols

- Link-state uses Dijkstra...
  - ...because *Dijkstra is the most efficient SPF algorithm*

- Bellman-Ford is basis of distance vector...
  - ...because *BF supports a distributed implementation*
Centralized Bellman-Ford

• Breadth-first search of paths, *by increasing hop count*, for best paths to all destinations.

• Terminates when no improved path is found.

• Maintains set $R$ of nodes for which a route has been found

• Iterate
  – Look at all neighbors of nodes in $R$
  – If route through node in $R$ to neighbor is better than neighbor’s current route in $R$, update route in $R$
  – Repeat until no routes in R are improved

• $R$ contains routing table
Centralized Bellman-Ford

algorithm CBF run at node $i$
begin
1 $d_i \leftarrow 0; p_i \leftarrow i$;
2 for each $\{ x \in V - \{i\}\}$;
3 $d_x \leftarrow \infty; p_x \leftarrow \emptyset$;
4 $R \leftarrow \{i\}$;
5 do
6 $R' \leftarrow \{ j \mid x \in R, (x, j) \in E, d_j > d_x + \omega_{xj} \}$;
7 for each $\{ j \in R' \}$
8 begin
9 $d_j \leftarrow \text{Min}\{ d_x + \omega_{xj} \mid x \in R', (x, j) \in E \}$;
10 $p_j \leftarrow \{ x \mid (x, j) \in E, d_x + \omega_{xj} = d_j \}$;
11 end
12 $R \leftarrow R \cup R'$;
13 until ($R' = \emptyset$)
14 end

Figure 1. Centralized Bellman-Ford Algorithm.

- $V$ is the set of vertices.
- $E$ is the set of edges.
- $w_{ij}$ is the weight of the link between nodes $i$ and $j$.
- $R$ is the set of nodes we’ve found routes to
  - $d_j$ is the distance to node $j$
  - $p_j$ is the predecessor for node $j$
Bellman-Ford in pictures

Add \(<\text{dest} = i, \text{pred} = i, \text{dist} = 0>\) to \(T\)

\[
d \text{do} \\
R' = \text{EmptySet} \quad \text{// routes updated this iteration} \\
\text{for each } <x,p,d> \text{ in } R \\
\quad \text{for each } n \text{ in Neighbors}(x) \\
\quad \quad \text{if } <n,p',d'> \text{ is not in } R, \text{ or} \\
\quad \quad \quad \text{d} + w(x,n) < d' \\
\quad \quad \quad \text{then add } <n,x,d + w(x,n)> \text{ to } R' \\
\text{Update } R \text{ with } R' \\
\text{until } (R' == \text{EmptySet})
\]
The Bellman-Ford Algorithm

**Bold** = this iteration, **Dashed** = rejected

Spring 2017  CE 151 - Advanced Networks  28
Bellman-Ford vs. Dijkstra

What’s the difference in their iterative step..?

- CBF iterates on increasing hop count... “shortest n-hop path”
- Dijkstra iterates on increasing path cost... “next shortest path”

*Allows for distributed implementation of Bellman-Ford...*
Review

- The Bellman-Ford algorithm iterates on hop-count.
  - Considers “shortest n-hop path” at each iteration
  - This makes it adaptable to a distributed implementation
Overview

• Bellman-Ford shortest path algorithm

• **Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)**

• DBF’s big weakness… counting-to-infinity

• RIP details
Translating to a Protocol

• BF can be implemented in a distributed manner
  – Implement iterative step with messages
  – Each message received moves computation one hop further
BF with Message Passing

**Bold** = this iteration, **Dashed** = rejected
How this is Implemented

• Destinations are actually subnets (IP prefixes)

• Cost to reach destination...
  – Is 0 for directly connected routers
  – Is the sum of the costs of networks traversed to reach a directly connected router, otherwise.
View network as a graph

- **Networks typically represented as a network graph:**
  - nodes are connected by networks
    - network can be a link or a LAN
  - network interface has cost $c(v,w)$
  - networks are destinations
    - $\text{Net}(v,w)$ is an IP address of a network

- For ease of notation, we often replace the clouds between nodes by simple links.
Distance Vector – Messages

Routing Table of node v

<table>
<thead>
<tr>
<th>Dest</th>
<th>via (next hop)</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net</td>
<td>n</td>
<td>D(v,Net)</td>
</tr>
</tbody>
</table>

Nodes send messages to their neighbors which contain routing table entries

A message has the format: $[\text{Net}, D(v,\text{Net})]$ means "My cost to go to Net is $D(v,\text{Net})$"
DV - Sending Updates

RoutingTable of node \( v \)

<table>
<thead>
<tr>
<th>Dest</th>
<th>via ((\text{next hop}))</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net(_1)</td>
<td>m</td>
<td>(D(v,\text{Net}_1))</td>
</tr>
<tr>
<td>Net(_2)</td>
<td>n</td>
<td>(D(v,\text{Net}_2))</td>
</tr>
<tr>
<td>- - -</td>
<td>- - -</td>
<td>- - -</td>
</tr>
<tr>
<td>Net(_N)</td>
<td>w</td>
<td>(D(v,\text{Net}_N))</td>
</tr>
</tbody>
</table>

Periodically, each node \( v \) sends the content of its routing table to its neighbors:
DV – Router Initialization

• Suppose a new node v becomes active.
• The cost to access directly connected networks is zero:
  – D (v, Net(v,m)) = 0
  – D (v, Net(v,w)) = 0
  – D (v, Net(v,n)) = 0

RoutingTable

<table>
<thead>
<tr>
<th>Dest</th>
<th>via (next hop)</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net(v,m)</td>
<td>m</td>
<td>0</td>
</tr>
<tr>
<td>Net(v,w)</td>
<td>w</td>
<td>0</td>
</tr>
<tr>
<td>Net(v,n)</td>
<td>n</td>
<td>0</td>
</tr>
</tbody>
</table>
DV – Router Initialization

<table>
<thead>
<tr>
<th>Dest</th>
<th>via (next hop)</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net(v,m)</td>
<td>m</td>
<td>0</td>
</tr>
<tr>
<td>Net(v,w)</td>
<td>w</td>
<td>0</td>
</tr>
<tr>
<td>Net(v,n)</td>
<td>n</td>
<td>0</td>
</tr>
</tbody>
</table>

- New node v sends the routing table entry to all its neighbors:

  \[\text{[Net(v,n),0]} \quad \text{[Net(v,n),0]} \quad \text{[Net(v,m),0]}\]
DV – Router Initialization

• Node v receives the routing tables from other nodes and builds up its routing table
DV – Updating Routing Tables

Suppose node $v$ receives a message from node $m$: $[\text{Net}, D(m,\text{Net})]$

$$[\text{Net}, D(m,\text{Net})] \rightarrow$$

Node $v$ updates its routing table and sends out further messages if the message reduces the cost of a route:

```plaintext
if  \((D(m,\text{Net}) + c(v,m) < D(v,\text{Net}))\) {
    D^{\text{new}}(v,\text{Net}) = D(m,\text{Net}) + c(v,m);
    Update routing table;
    send message \([\text{Net}, D^{\text{new}}(v,\text{Net})]\) to all neighbors
} 
```
DV – Updating Routing Tables

Before receiving the message:

Suppose \( D(m, Net) + c(v, m) < D(v, Net) \):
What if “??” was “m”... and

Suppose \( D(m, Net) + c(v,m) > D(v,Net) \):

- How deal with learning bad news?
  - Wait for periodic flood...
Review

• Distance-Vector protocols
  – Are IGPs
  – Implement a distributed routing model
  – Routers send vectors of distances to their neighbors
    • “Tell your neighbors about the rest of the world”
    • Distributed computation

• The DBF algorithm is used in RIP
  – Implements Bellman-Ford iterative step using routing updates
  – Remembers only the best route to each destination
  – Handles link cost increases using periodic updates
Assume: - link cost is 1, i.e., $c(v,w) = 1$
- all updates, updates occur simultaneously
- Initially, each router only knows the cost of connected interfaces

Example

<table>
<thead>
<tr>
<th>Net</th>
<th>via</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>t=0:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0.1.0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>10.0.2.0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>t=1:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0.1.0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>10.0.2.0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>10.0.3.0</td>
<td>10.0.2.2</td>
<td>1</td>
</tr>
<tr>
<td>t=2:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0.1.0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>10.0.2.0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>10.0.3.0</td>
<td>10.0.2.2</td>
<td>1</td>
</tr>
<tr>
<td>10.0.4.0</td>
<td>10.0.2.2</td>
<td>2</td>
</tr>
<tr>
<td>10.0.5.0</td>
<td>10.0.3.2</td>
<td>2</td>
</tr>
</tbody>
</table>

Spring 2017
CE 151 - Advanced Networks
Routing tables have converged!
“DV” script
Overview

• Bellman-Ford shortest path algorithm

• Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)

• **DBF’s big weakness… counting-to-infinity**

• RIP details
Counting to infinity

- What happens if...
  - ...after detecting link cost change
  - ...before sending update
  - ...receive update from upstream neighbor?
- It depends...

```
algorithm DBF2
    Initialize:
    begin
    1  \( d_s \leftarrow 0; n_s \leftarrow s \); 
    2  for each \( x \in V - \{s\} \);
    3  \( d_x \leftarrow \infty; n_x \leftarrow \emptyset \);
    4  for each \( x \in N \);
        begin
    5  \( d_x \leftarrow \omega_{sx}; n_x \leftarrow x \);
    6  SendUpdate(\( d_s \));
    end
    end

    ReceiveUpdate: \( u_k^i \)
    begin
    7  \( d \leftarrow \omega_{sk} + u_k^i \);
    8  if \((n_i = k) \vee (d < d_i)\) 
      then begin // Change if update better or from current next hop
    9      \( d_i \leftarrow d; n_i \leftarrow k \);
     10      SendUpdate(\( d_i \));
      end
    11    if \((n_i \neq k) \land (d > d_i)\)
    12      then SendUpdate(\( d_i \)); // Send update if \( u_k^i \) worse than current route?
    end

    LinkCostChange: \( l_{sk} \)
    begin
    14  \( w_{sk} \leftarrow l_{sk} \);
    15  for each \( \{i \mid n_i = k\} \)
    16      begin // Change all routes with source of update as next hop
    17      \( d_i \leftarrow \omega_{sk} + u_k^i; n_i \leftarrow k \);
     18      SendUpdate(\( d_i \));
      end
    end

    Update Timer Expires:
    begin
    16  for each \( \{i \in N\} \)
    17      SendUpdate(\( d_i \));
    end
```

Figure 4. Distributed Bellman-Ford without Neighbor Tables.
Counting to infinity

If B sends update first...

Initially, all up:

A goes down:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
</tr>
</tbody>
</table>

...all goes fine.
Counting to infinity

But, if C sends update first...

Initially, all up: 1 2 3 4

A goes down: 3 2 3 4  (after 1 exchange)
Counting to infinity

But, if C sends update first...

Initially, all up: 1 2 3 4
A goes down: 3 2 3 4 (after 1 exchange)
3 4 3 4 (after 2 exchanges)
Counting to infinity

But, if C sends update first...

Initially, all up: 1 2 3 4
A goes down: 3 2 3 4  (after 1 exchange)
 3 4 3 4  (after 2 exchanges)
 5 4 5 4  (after 3 exchanges)
Counting to infinity

But, if C sends update first...

Initially, all up:
A goes down:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

(after 1 exchange)
(after 2 exchanges)
(after 3 exchanges)
(after 4 exchanges)
Counting to infinity

But, if C sends update first...

Initially, all up:
A goes down:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

(after 1 exchange)
(after 2 exchanges)
(after 3 exchanges)
(after 4 exchanges)
(after 5 exchanges)
Counting to infinity

But, if C sends update first...

<table>
<thead>
<tr>
<th></th>
<th>B → 7 → C → 7 → D → 7 → E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Initially, all up:

A goes down: (after 1 exchange) (after 2 exchanges) (after 3 exchanges) (after 4 exchanges) (after 5 exchanges) (after 6 exchanges)

...things don’t go so well.

Counting to infinity is a timing problem!

∞
More generally,

- More generally, can be seen as problem of coordinating updates across a subtree
- Only want to accept updates from another subtree.
Counting to infinity solutions

- **Triggered updates**: send update immediately on change to forwarding table; speeds up everything (including counting-to-infinity)

- **Hold-down timer**: If a destination is identified as unreachable
  - Freeze route as unreachable
  - Ignore any other updates with distance same or worse to current
  - Accept any router better than current
  - Hold time should be long enough for all routers to get bad news

- **Split horizon**: don’t advertise route on interface it was learned from; resolves loops with 2 routers

- **Poisoned reverse**: advertise distance of $\infty$ on interface that route was learned

- **Use small value for $\infty$ (16)**.

- **TTL** field in IP header.
Split Horizon w/ Poisoned Reverse

C sends $\infty$ update to B (from whom it learned about A)...

Initially, all up:  1  2  3  4  
A goes down:  $\infty$  2  3  4  (after 1 exchange)
Split Horizon w/ Poisoned Reverse

C sends $\infty$ update to B (from whom it learned about A)...

Initially, all up:

A goes down:

```
1    2    3    4
infinity  2    3    4  (after 1 exchange)
infinity  infinity  3    4  (after 2 exchanges)
```
Split Horizon w/ Poisoned Reverse

C sends $\infty$ update to B (from whom it learned about A)...

Initially, all up:

A goes down:

\[
\begin{array}{c}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} \\
1 & \infty & \infty & 3 & 4 \\
\infty & 2 & \infty & 4 & \text{(after 1 exchange)} \\
\infty & \infty & 3 & 4 & \text{(after 2 exchanges)} \\
\infty & \infty & \infty & 4 & \text{(after 3 exchanges)}
\end{array}
\]
Split Horizon w/ Poisoned Reverse

C sends $\infty$ update to B (from whom it learned about A)…

Initially, all up:
A goes down:

```
A  B  C  D  E
1  2  $\infty$ 4
infinity 2  3  4
infinity infinity 3  4
infinity infinity infinity 4
infinity infinity infinity infinity
```

(after 1 exchange)
(after 2 exchanges)
(after 3 exchanges)
(after 4 exchanges)

...things go much better.

Works for simple, **but not all**, routing loops.
Split Horizon w/ Poisoned Reverse

- Still risk of loops...
- Don’t want updates in alternative branch of subtree to corrupt forwarding tables...
  - Split Horizon not triggered...
Review

• Counting-to-infinity
  – Is a timing problem
    • After detecting path cost increase...
    • Receive update from downstream neighbor...
    • Before send update.
    • Result is bad data get’s propagated...
  – Solutions
    • Small value for infinity (16)
    • Triggered updates – send update immediately on changing forwarding table; minimizes window for receiving bad update
    • Hold-down timer – if destination becomes unreachable, freeze as unreachable until bad (dependent) data is purged
    • Split-horizon - don’t advertise route on interface it was learned from; resolves loops with 2 routers
    • Poison-reverse - advertise distance of $\infty$ on interface that route was learned from;
    • TTL in IP header
Review... from reading

• What information is exchanged in DV protocols?
• What assumptions are required for the correctness of RIP?
• What causes RIP to converge so slowly?
• How does split-horizon w/ poison-reverse work?
• What CtoI situations does split-horizon w/ poison-reverse prevent?
• What CtoI situations does triggered-updates help with.
Characterizing Distance-Vector

• How many updates generated per link change?
  – # of destinations that currently use next hops on that link.

• How far do updates propagate?
  – Only as far as they’re needed.

• Subject to counting-to-infinity
Comparing with Link-State

**Distance-Vector**
- # updates per link change?
  - # destinations that currently use next hops on that link.
- How far propagate updates?
  - Only as far as they’re needed.
- *Lot’s of updates, limited distribution.*
- Subject to counting-to-infinity

**Link-State**
- # updates per link change?
  - One.
- How far propagate updates?
  - Flooded to all nodes.
- *One update, global distribution.*
- Scaling problems due to flooding

- Is there an opportunity for improvement?
  - Yes!
  - Send link-state updates only as far as they’re needed!
  - Called “*link-vector*”
Link-Vector

• Maintain partial graph of network
  – Union of links in neighbor’s out-trees
  – Guaranteed to contain shortest paths!
• Send graph changes to neighbors
  – Forward link states...
  – ...only as far as they’re needed!
• Currently not deployed.

**Link-Vector**

• # updates per link change?
  – One.
• How far propagate updates?
  – Only as far as they’re needed.
• *Have your cake and eat it to!* 

**Link-State**

• # updates per link change?
  – One.
• How far propagate updates?
  – Flooded to all nodes.
• Scaling problems due to flooding
Review

• Characterizing Distance-Vector
  – A link-cost change event results in updates for all destinations that use that link for their next hop.
  – Updates propagate only as far as they’re needed.
  – I.e. “lot’s of updates, limited distribution”
  – Problem: subject to counting-to-infinity

• Link-vector only shares forwarding (vs full routing) table with neighbors
  – Based on observation that best paths are in this subset of the topology(!)

• Link-vector combines strengths of LS and DV
  – LS: one update, global distribution
  – DV: many updates, limited distribution
  – LV: one update, limited distribution!
Overview

- Bellman-Ford shortest path algorithm
- Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)
- DBF’s big weakness… counting-to-infinity
- RIP details
RIP - History

- **Late 1960s**: Distance Vector protocols were used in the ARPANET
- **Mid-1970s**: XNS (Xerox Network system) routing protocol is the precursor of RIP in IP (and Novell’s IPX RIP and Apple’s routing protocol)
- **1982**: Release of routed for BSD Unix
- **1988** RIPv1 (RFC 1058)
  - Classful routing
- **1993** RIPv2 (RFC 1388)
  - Adds subnet masks with each route entry
  - Allows classless routing
- **1998** Current version of RIPv2 (RFC 2453)
### RIPv1 Packet Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP header</td>
<td></td>
</tr>
<tr>
<td>UDP header</td>
<td></td>
</tr>
<tr>
<td>RIP Message</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>1: request</td>
</tr>
<tr>
<td></td>
<td>2: response</td>
</tr>
<tr>
<td>Version</td>
<td>Set to 00...0</td>
</tr>
<tr>
<td>Address family</td>
<td>Set to 00.00</td>
</tr>
<tr>
<td>32-bit address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unused (Set to 00...0)</td>
</tr>
<tr>
<td></td>
<td>Unused (Set to 00...0)</td>
</tr>
<tr>
<td>Metric (1-16)</td>
<td></td>
</tr>
<tr>
<td>Up to 24 more routes (each 20 bytes)</td>
<td></td>
</tr>
</tbody>
</table>

One RIPv1 message can have up to 25 route entries.

- **2:** for IP
- **0…0:** request full routing table

Address of destination

Cost (measured in hops)
RIPv2

• RIPv2 is an extends RIPv1:
  – Subnet masks are carried in the route information
  – Authentication of routing messages
  – Route information carries better next-hop address if it exists
  – Exploites IP multicasting

• Extensions of RIPv2 are carried in unused fields of RIPv1 messages
RIPv2 Packet Format

- IP header
- UDP header
- RIPv2 Message

<table>
<thead>
<tr>
<th>Command</th>
<th>Version</th>
<th>Set to 00.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>address family</td>
<td>route tag</td>
<td></td>
</tr>
</tbody>
</table>

- IP address
- Subnet Mask
- Next-Hop IP address
- metric (1-16)

Up to 24 more routes (each 20 bytes)

Used to carry information from other routing protocols (e.g., autonomous system number)

Subnet mask for IP address

Identifies a better next-hop address on the same subnet than the advertising router, if one exists (otherwise 0....0)
RIP Messages

• This is the operation of RIP in *routed*. Runs on UDP port 520.

• Two types of messages:
  – *Request messages*
    • used to ask neighboring nodes for an update
  – *Response messages*
    • contains an update
RIP Protocol

- **Initialization**: Send a request packet (command = 1, address family=0..0) on all interfaces:
  - RIPv1 uses broadcast if possible,
  - RIPv2 uses multicast address 224.0.0.9, if possible

- **Request received**: Routers that receive above request send their entire routing table

- **Response received**: Update the routing table

- **Regular routing updates**: Every 30 seconds, send all or part of the routing tables to every neighbor in an response message

- **Triggered Updates**: Whenever the metric for a route change, send entire routing table.
RIP Security

• Issue: Sending bogus routing updates to a router
• RIPv1: No protection
• RIPv2: Simple authentication scheme

<table>
<thead>
<tr>
<th>Command</th>
<th>Version</th>
<th>Set to 00.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffff</td>
<td></td>
<td>Authentication Type</td>
</tr>
<tr>
<td>Password (Bytes 0 - 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password (Bytes 4 - 7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password (Bytes 8 - 11)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password (Bytes 12 - 15)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Up to 24 more routes (each 20 bytes)

2: plaintext password

IP header | UDP header | RIPv2 Message

32 bits
RIP Problems

• RIP takes a long time to stabilize
  – Even for a small network, it takes several minutes until the routing tables have settled after a change

• RIP has all the problems of distance vector algorithms, e.g., count-to-Infinity
  – RIP uses split horizon to avoid count-to-infinity

• The maximum path in RIP is 15 hops
Review?

• Bellman-Ford is basis for distance-vector protocols.
• Distributed Bellman-Ford protocols subject to counting-to-infinity
• Counting-to-infinity is a timing problem
• DUAL algorithm solves this timing problem
  – Feasible Condition
  – Diffusing computation (Query/Response)
• Distance-vector based protocols
  – Generate multiple updates per link cost change
  – Distribute updates only as far as needed
• DUAL adds overhead for diffusing computation
Questions?