Distance Vector & RIP

Brad Smith
Administrativia

• How are the labs going?
• Opportunities
  – Cruzio... I’m waiting to hear back
  – NMO Software Development for Cisco Advanced Services... waiting for applications
  – Expect more from campus network operations group... network configuration & testing, Net Disco development, NFSEN
• This week
  – Link Layer lab due Wednesday, 4/24
  – Link-State Routing quiz Thursday, 4/25 (combine with Distance-Vector next Thursday)
• Next week
  – Project Proposal due Tuesday, 4/30!
  – OSPF lab due Wednesday, 5/1
  – Distance-Vector quiz Thursday, 5/2 (don’t forget reading!)
• Project deliverables
  – Paper describing what’s covered in the lab.
  – Lab
  – Answer key
Algorithm Classification

- **Distance-Vector**
  - 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**
  - 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)
Algorithm Classification

• **Distance-Vector**
  – 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**
  – 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)
How ensure *correct* routes?

- Recall requirement for correctness of routing protocol
  - Loop-free
  - Desired path characteristics
- Two strategies for ensuring correctness
  - Use identical algorithm for selecting paths
    - Share minimal topology information
    - Use identical path selection algorithm at all nodes
    - Used for IGP/Intra-domain routing
    - Use link-state or distance vector protocol
  - Use custom (private) algorithm for selecting paths
    - Share full path information
    - Use policy-specific path selection algorithm at each node
    - Used for EGP/Inter-domain routing
    - Use path-vector protocol
How ensure *correct* routes?

- Recall requirement for correctness of routing protocol
  - Loop-free
  - Desired path characteristics
- Two strategies for ensuring correctness
  - Use identical algorithm for selecting paths
    - Share minimal topology information
    - Use identical path selection algorithm at all nodes
    - Used for IGP/Intra-domain routing
    - Use link-state or distance vector protocol
  - Use custom (private) algorithm for selecting paths
    - Share full path information
    - Use policy-specific path selection algorithm at each node
    - Used for EGP/Inter-domain routing
    - Use path-vector protocol
Overview

• Bellman-Ford shortest path algorithm

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

• Using shortest-path algorithms in real networks
  – Destinations are **subnets**, not routers

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

• RIP details

• Quick DUAL (EIGRP) intro
Overview

• Bellman-Ford shortest path algorithm

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

• Using shortest-path algorithms in real networks
  – Destinations are *subnets*, not routers

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

• RIP details

• Quick DUAL (EIGRP) intro
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 $(R' = \emptyset)$
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 $j$
The Bellman-Ford Algorithm

**Bold** = this iteration, **Dashed** = rejected
Bellman-Ford vs. Dijkstra

**Figure 1.** Centralized Bellman-Ford Algorithm.

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

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

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

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)**
- Using shortest-path algorithms in real networks
  - Destinations are *subnets*, not routers
- DBF’s big weakness… counting-to-infinity
- RIP details
- Quick DUAL (EIGRP) intro
Translating to a Protocol

- DBF 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

```
\begin{figure}
\centering
\includegraphics[width=\textwidth]{bf_message_passing}
\end{figure}
```
DBF w/ Neighbor Tables

- Iterate with messages
- Neighbor tables
- Send updates
  - Reliably
  - Periodically
- How can we prove this works?
  - Centralized Bellman-Ford
- How handle update with worse current route?
  - Uses neighbor table

---

**protocol DBF1** run at node $i$

**Initialize**:

begin

1. $d_i \leftarrow 0; n_i \leftarrow i$
2. for each ($x \in V - \{i\}$)
   3. $d_x \leftarrow \infty ; n_x \leftarrow 0$
3. for each ($x \in N$)
   4. begin
       5. $d_x \leftarrow \omega_{ix}; n_x \leftarrow x$
       6. $Send/Update(d_x)$
   end

**ReceiveUpdate**: $u^j_k$

begin

7. $d^j_k \leftarrow u^j_k$
8. $t_j \leftarrow \min\{d^j_x + \omega_{x} | x \in N\}$
9. $n_j \leftarrow \{x \in N | d^j_x + \omega_{x} = d^j_k\}$
10. if ($t_j \neq d^j_k$)
    11. then begin // No split-horizon, poison-reverse
        12. $d^j_k = t_j$
        13. Send/Update($d^j_k$)
    end

**LinkCostChange**: $l_{ik}$

begin

13. if ($l_{ik} < \omega_{ik}$)
14. then $T = V$; // $\omega_{ik}$ improves, recompute all routes
15. else $T = \{j | n_j = k\}$; // $\omega_{ik}$ worse, recompute routes using $k$ only
16. $\omega_{ik} \leftarrow l_{ik}$
17. for each ($j \in T$)
18. begin
19. $d \leftarrow \min\{d^j_x + \omega_{x} | x \in N\}$
20. if ($d < d^j_k$)
21. then begin
22. $d^j_k \leftarrow d; n_j \leftarrow \{x \in N | d^j_x + \omega_{x} = d\}$
23. $Send/Update(d^j_k)$
    end
end

**Update Timer Expires**:

24. for each ($j \in V$)
25. Send/Update($d^j_k$)

**Run at node $j$**

- $n_j$ – next hop to $j$
- $d^j_k$ – distance to $j$
- $d^k_j$ – nbr table for $j$ nbr $k$
- $u^k_j$ – update for $j$ nbr $k$
- $l_{ik}$ – update for link $i$-$k$

---

Figure 3. Distributed Bellman-Ford with Neighbor Tables.
DBF w/o Neighbor Tables

- Only remember best route
- This is the basis for RIP
- How handle update with worse current route?
  - Receive worse route from current neighbor... don’t know if other neighbor has better route.
  - Periodically send full routing table to all neighbors
  - Neighbor with better route will respond

```plaintext
protocol DBF2 run at node i
  Initialize:
  begin
  1  d_i ← 0; n_i ← i;
  2  for each {x ∈ V - {i}}:
  3      d_x ← ∞; n_x ← ∅;
  4  for each {x ∈ N}:
  5      d_x ← w_{ix}; n_x ← x;
  6      SendUpdate(d_x);
  end

ReceiveUpdate: u_j^k
  begin
  7  d ← u_j^k + w_{ik};
  8  if ((n_j = k) ∧ (d < d_j))
  9      then begin // Change if update better or from current next hop
  10     D_j ← u_j^k; d_j ← d; n_j ← k;
  11     SendUpdate(d_j);
  12      end
  13  if ((n_j ≠ k) ∧ (d > d_j))
  14      then SendUpdate(d_j); // Send update if u_j^k worse than current route?
  end

LinkCostChange: l_{ik}
  begin
  15  w_{ik} ← l_{ik};
  16  for each {j | n_j = k}:
  17     begin // Change all routes with link neighbor as next hop
  18     d_j ← D_j + w_{ik}; n_j ← k;
  19     SendUpdate(d_j);
  20     end
  end

Update Timer Expires:
  for each {j ∈ V}
  SendUpdate(d_j);
```

Run at node j
- n_j – next hop to j
- d_j – distance to j
- D_j – dist from nbr for j
- u_j^k – update for j nbr k
- l_{ik} – update for link i-k

Figure 4. Distributed Bellman-Ford without Neighbor Tables.
Review

• Distance-Vector protocols
  – Are IGPs
  – Implemented 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

• Link cost increases can also be handled with neighbor tables
Overview

• Bellman-Ford shortest path algorithm

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

• Using shortest-path algorithms in real networks
  – Destinations are subnets, not routers

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

• RIP details

• Quick DUAL (EIGRP) intro
Summary of following section…

• 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 (Net)</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: [Net, D(v,Net)] means "My cost to go to Net is D(v,Net)"
DV - Sending Updates

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_1</td>
<td>m</td>
<td>D(v,Net_1)</td>
</tr>
<tr>
<td>Net_2</td>
<td>n</td>
<td>D(v,Net_2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net_N</td>
<td>w</td>
<td>D(v,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**

**Routing Table**

<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:

  - \([\text{Net}(v,n), 0] \)
  - \([\text{Net}(v,w), 0] \)
  - \([\text{Net}(v,m), 0] \)

---

Spring 2013

CE 151 - Advanced Networks
DV – Router Initialization

- Node v receives the routing tables from other nodes and builds up its routing table
Suppose node v receives a message from node m: $[\text{Net}, D(m, \text{Net})]$

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

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

Before receiving the message:

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

\[
\text{RoutingTable}
\begin{array}{|c|c|c|}
\hline
\text{Dest} & \text{via} & \text{cost} \\
\hline
\text{Net} & ?? & D(v, Net) \\
\hline
\end{array}
\]

\[
\text{RoutingTable}
\begin{array}{|c|c|c|}
\hline
\text{Dest} & \text{(next hop)} & \text{cost} \\
\hline
\text{Net} & m & D^{new}(v, Net) \\
\hline
\end{array}
\]
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>$10.0.1.0$</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>$10.0.2.0$</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

$t=0:$

<table>
<thead>
<tr>
<th>Net</th>
<th>via</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

$t=1:$

<table>
<thead>
<tr>
<th>Net</th>
<th>via</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10.0.1.0$</td>
<td>$10.0.2.0$</td>
<td>1</td>
</tr>
<tr>
<td>$10.0.2.0$</td>
<td>$10.0.3.0$</td>
<td>0</td>
</tr>
<tr>
<td>$10.0.3.0$</td>
<td>$10.0.4.0$</td>
<td>1</td>
</tr>
</tbody>
</table>

$t=2:$

<table>
<thead>
<tr>
<th>Net</th>
<th>via</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10.0.1.0$</td>
<td>$10.0.2.0$</td>
<td>1</td>
</tr>
<tr>
<td>$10.0.2.0$</td>
<td>$10.0.3.0$</td>
<td>0</td>
</tr>
<tr>
<td>$10.0.3.0$</td>
<td>$10.0.4.0$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Net</th>
<th>via</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10.0.1.0$</td>
<td>$10.0.2.0$</td>
<td>1</td>
</tr>
<tr>
<td>$10.0.2.0$</td>
<td>$10.0.3.0$</td>
<td>0</td>
</tr>
<tr>
<td>$10.0.3.0$</td>
<td>$10.0.4.0$</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

Routing tables have converged!
Review

• For shortest-path routing in real networks...

• The subnet is the destination

• The subnet is specified by an IP prefix

• The cost of directly connected networks is 0

• The cost of a link is the cost for a router to forward a packet onto that link
  – In effect it is the cost for a packet to transit a router onto that link... kind of strange
Overview

• Bellman-Ford shortest path algorithm
• Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)
• Using shortest-path algorithms in real networks
  – Destinations are subnets, not routers
• DBF’s big weakness… counting-to-infinity
• RIP details
• Quick DUAL (EIGRP) intro
Counting to infinity

- What happens if...
  - ...after detecting link cost change
  - ...before sending update
  - (i.e. after “ReceiverUpdate” event but before “UpdateTimerExpires”)
  - ...receive update from upstream neighbor?

- It depends...

algorithm DBF2

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

ReceiveUpdate: $u^k_i$
begin
7 $d \leftarrow \omega_{sk} + u^k_i$
8 if $((n_i = k) \lor (d < d_i))$
9  then begin // Change if update better or from current next hop
10    $d_i \leftarrow d; n_i \leftarrow k$
11    $SendUpdate(d_i)$;
12  end
13 if $((n_i \neq k) \land (d > d_i))$
14  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    $d_x \leftarrow \omega_{sk} + u^k_i; n_x \leftarrow k$
18    $SendUpdate(d_i)$;
19  end
20 end

Update Timer Expires:
begin
21 for each $\{i \in N\}$
22 $SendUpdate(d_i)$;
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>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>infinity</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>infinity</td>
<td>infinity</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
<td>4</td>
</tr>
<tr>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
<td>infinity</td>
</tr>
</tbody>
</table>

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

...all goes fine.
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>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...things don’t go so well.

Counting to infinity is a timing problem!

infinity
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:

<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>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>infinity</td>
<td>infinity</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>

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>C</td>
<td>$\infty$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

(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 upstream subtree to corrupt forwarding tables...
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

<table>
<thead>
<tr>
<th>Distance-Vector</th>
<th>Link-State</th>
</tr>
</thead>
<tbody>
<tr>
<td># updates per link change?</td>
<td># updates per link change?</td>
</tr>
<tr>
<td>– # destinations that currently use next hops on that link.</td>
<td>– One.</td>
</tr>
<tr>
<td>How far propagate updates?</td>
<td>How far propagate updates?</td>
</tr>
<tr>
<td>– Only as far as they’re needed.</td>
<td>– Flooded to all nodes.</td>
</tr>
<tr>
<td>Lot’s of updates, limited distribution.</td>
<td>One update, global distribution.</td>
</tr>
<tr>
<td>Subject to counting-to-infinity</td>
<td>Scaling problems due to flooding</td>
</tr>
</tbody>
</table>

- **Which would you use for content routing?**
- 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)
- Using shortest-path algorithms in real networks
  - Destinations are subnets, not routers
- DBF’s big weakness… counting-to-infinity
- RIP details
- Quick DUAL (EIGRP) intro
**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

- IP header
- UDP header
- RIP Message

<table>
<thead>
<tr>
<th>Command</th>
<th>Version</th>
<th>Set to 00...0</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>response</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| address family | Set to 00.00 |

| 32-bit address |

| Unused (Set to 00...0) |
| Unused (Set to 00...0) |

| metric (1-16) |

Up to 24 more routes (each 20 bytes)

1: RIPv1

- 1: request
- 2: response
- 2: for IP
- 0...0: request full routing table

Address of destination

Cost (measured in hops)

One RIP message can have up to 25 route entries

Up to 24 more routes (each 20 bytes)
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**

2: RIPv2

- **Command**
- **Version**
- **Set to 00.00**
- **address family**
- **route tag**
- **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 a response message

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

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

![RIPv2 Message Diagram]

<table>
<thead>
<tr>
<th>Command</th>
<th>Version</th>
<th>Set to 00.00</th>
<th>Authentication Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffff</td>
<td></td>
<td></td>
<td>Password (Bytes 0 - 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Password (Bytes 4 - 7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Password (Bytes 8 - 11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Password (Bytes 12 - 15)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Up to 24 more routes (each 20 bytes)</td>
</tr>
</tbody>
</table>

2: plaintext password
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

• None... mostly to help with lab
Overview

• Bellman-Ford shortest path algorithm
• Distributed Bellman-Ford (DBF) and the Routing Information Protocol (RIP)
• Using shortest-path algorithms in real networks
  – Destinations are subnets, not routers
• DBF’s big weakness… counting-to-infinity
• RIP details

• Quick DUAL (EIGRP) intro
DUAL – solution to C-to-I

- Basis of Cisco’s EIGRP
- Coordinates updates across a subtree
- Defines conditions that allow switching to path that *increases* distance to destination... “feasible condition (FC).”
- On topology change, if no available paths satisfy FC, initiate a “*diffusing computation.*”
  - Freeze route for destination
  - Send *Query* to all neighbors
  - Update routing table when all queries received
- Node responds to query with it’s best *feasible path*
  - It has a path that meets the feasible condition
  - It completes a diffusing computation
- *Ensures updates are coordinated to avoid loops! Prefix solution? Sequence # solution?*
Characterizing DUAL

- How many updates generated per link change?
  - Same as distance-vector.
- How far do updates propagate?
  - Same as distance-vector.
- Additional signaling?
  - Query/Response for diffusing computation in subtree.
- Not subject to counting-to-infinity

- Revolutionary advancement in routing.
- Only proprietary routing protocol widely deployed in Internet.
- Query/Response adds overhead... leaves door open for improvements:)!
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
The End!

• Spanning Tree Protocol next week.