Intra-domain Routing
Andreas Terzis
Outline

• Internet Intra-Domain Routing
  - Distance Vector
  - Shortest Path
Forwarding Table

- terzis@peregrine(~)> netstat -nr

Routing Table:
<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Ref</th>
<th>Use</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.220.13.0</td>
<td>128.220.13.85</td>
<td>U</td>
<td>3</td>
<td>8438</td>
<td>le0</td>
</tr>
<tr>
<td>224.0.0.0</td>
<td>128.220.13.85</td>
<td>U</td>
<td>3</td>
<td>0</td>
<td>le0</td>
</tr>
<tr>
<td>default</td>
<td>128.220.13.1</td>
<td>UG</td>
<td>0</td>
<td>58348</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>UH</td>
<td>0</td>
<td>668</td>
<td>lo0</td>
</tr>
</tbody>
</table>

- How are entries inserted in fwd table?
  - Statically (route add, route delete)
  - ICMP Redirects
  - Dynamic Routing Protocols
ICMP Redirects

- Default router informs host that better router exists
  - How does it know that?
- Host adds a new entry in it’s routing table
  - Security concerns?
What is Routing?

• Routing is the core function of a network

• It ensures that
  • Information accepted for transfer at a source node
  • Is delivered to the correct set of destination nodes, at reasonable levels of performance.
Internet Routing

• Internet organized as a two level hierarchy
• First level – autonomous systems (AS’s)
  – AS – region of network under single administrative control
• AS’s run an intra-domain routing protocols
  – Distance Vector, e.g., Routing Information Protocol (RIP)
  – Link State, e.g., Open Shortest Path First (OSPF)
• Between AS’s runs inter-domain routing protocols, e.g., Border Gateway Routing (BGP)
  – De facto standard today, BGP-4
Example

AS-1

AS-2

AS-3

Interior router

BGP router
Intra-domain Routing Protocols

- Based on unreliable datagram delivery
- Distance vector
  - Routing Information Protocol (RIP), based on Bellman-Ford
  - 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
  - Each network periodically floods immediate reachability information to other routers
  - Fast convergence, but high communication and computation overhead
Distance Vector Routing Algorithm

- Iterative: continues until no nodes exchange info
- Asynchronous: nodes need *not* exchange info/iterate in lock step
- 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 ➔ best known distance from X to Y, via Z as next hop (remember this !)
- *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 nodes $V$ 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 $V$
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             $D(A, Y) = D(A, V) + d$
13          else if (update $D(V, Y)$ received from $V$)
14             /* shortest path from $V$ to some $Y$ has changed */
15             $D(A, Y) = D(A, V) + D(V, Y)$;
16       if (there is a new minimum for destination $Y$)
17           send $D(A, Y)$ to all neighbors
18       forever
Example: Distance Vector Algorithm

**Initialization:**

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

### Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>( \infty )</td>
<td>-</td>
</tr>
</tbody>
</table>

### Node B

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>

### Node C

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

### Node D

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>( \infty )</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
</tbody>
</table>
Example: 1\textsuperscript{st} Iteration (C \rightarrow A)

7 \textit{loop:}

... 

13 \textbf{else if} (update D(V, Y) received from V)
14 \hspace{1em} D(A, Y) = D(A, V) + D(V, Y);
15 \textbf{if} (there is a new min. for destination Y)
16 \hspace{1em} \textbf{send} D(A, Y) to all neighbors
17 \hspace{1em} \textbf{forever}
**Example: 1^{st} Iteration (B→A, C→A)**

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>B</td>
</tr>
</tbody>
</table>

Node B

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

Node D

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

7 \textit{loop:}

... 

13 \textbf{else if} (update D(V, Y) received from V)
14 \hspace{1em} D(A, Y) = \min(D(A, V), D(A, V) + D(V, Y))
15 \textbf{if} (there is a new min. for destination Y)
16 \hspace{1em} \textbf{send} D(A, Y) to all neighbors
17 \textbf{forever}
Example: End of 1\textsuperscript{st} Iteration

7 loop:

\[\ldots\]

13 \textbf{else if} (update $D(V, Y)$ received from $V$) \\
14 $D(A, Y) = D(A, V) + D(V, Y)$; \\
15 \textbf{if} (there is a new min. for destination $Y$) \\
16 \textbf{send} $D(A, Y)$ to all neighbors \\
17 \textbf{forever}
Example: End of 2\textsuperscript{nd} Iteration

\begin{itemize}
\item loop:
\item \ldots
\item 13 \textbf{else if} (update \(D(V, Y)\) received from \(V\))
\item 14 \(D(A, Y) = D(A, V) + D(V, Y)\);
\item 15 \textbf{if} (there is a new min. for destination \(Y\))
\item 16 \textbf{send} \(D(A, Y)\) to all neighbors
\item 17 \textbf{forever}
\end{itemize}

Nothing changes \(\rightarrow\) algorithm terminates
Distance Vector: Link Cost Changes

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

“good news travels fast”
Distance Vector: Count to Infinity

Problem

Loop:
wait (until A sees a link cost change to neighbor V
or until A receives update from neighbor V)
if (D(A, V) changes by d)
for all destinations Y through V do
D(A, Y) = D(A, Y) + d;
else if (update D(V, Y) received from V)
D(A, Y) = D(A, V) + D(V, Y);
if (there is a new minimum for destination Y)
send D(A, Y) to all neighbors
forever

Link cost changes here; recall from slide 23 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?

Node B

\[
\begin{array}{ccc}
\text{Node B} & \text{D} & \text{C} & \text{N} \\
& A & 4 & A \\
A & 60 & A \\
& C & 1 & C \\
\end{array}
\]

Node C

\[
\begin{array}{ccc}
\text{Node C} & \text{D} & \text{C} & \text{N} \\
A & 5 & B \\
& A & 5 & B \\
& B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 60 & A \\
C & 1 & C \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 51 & C \\
C & 1 & C \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 51 & C \\
C & 1 & C \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 50 & A \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 50 & A \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 50 & A \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 50 & A \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
& D & C & N \\
A & 50 & A \\
B & 1 & B \\
\end{array}
\]

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

Algorithm terminates

\[\text{time} = 19\]
Outline

• Distance Vector
• Link State
Link State Flooding Example
Link State Flooding Example
Link State Flooding Example
Link State Flooding Example
Dijsktra’s 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 \[ D(v) = \min( D(v), D(w) + c(w,v) ); \]
13 \[ \text{// new cost to } v \text{ is either old cost to } v \text{ or known} \]
14 \[ \text{// shortest path cost to } w \text{ plus cost from } w \text{ to } v \]
15 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>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></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>

### Initialization:
1. $S = \{A\}$;
2. For all nodes $v$
3. If $v$ adjacent to $A$
4. Then $D(v) = c(A,v)$;
5. Else $D(v) = \infty$;

...
### 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></td>
<td>4,D</td>
<td></td>
<td>2,D</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>

**Diagram**

- **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></td>
<td>2,D</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3,E</td>
<td></td>
<td></td>
<td>4,E</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>

... *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></td>
<td>2,D</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3,E</td>
<td></td>
<td></td>
<td>4,E</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></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>4,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3,E</td>
<td></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:
   
   \[ D(v) = \min( D(v), D(w) + c(w,v) ) ; \]

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>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>4,D</td>
<td>2,D</td>
<td>$\infty$</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**

8. find w not in S s.t. D(w) is a minimum;
9. add w to S;
10. update D(v) for all v adjacent to w and not in S:
11. \( D(v) = \min( D(v), D(w) + c(w,v) ) \);
12. until all nodes in S;
Complexity

- Assume a network consisting of \( n \) nodes
  - Each iteration: need to check all nodes, \( w \), not in \( S \)
  - \( n*(n+1)/2 \) comparisons: \( O(n^2) \)
  - More efficient implementations possible: \( O(n*\log(n)) \)