Link State and Distance Vector Routing

EECS 489 Computer Networks

http://www.eecs.umich.edu/~zmao/eecs489

Z. Morley Mao
Tuesday Sept 21, 2004
IP Header

- Vers: IP versions
- HL: Header length (in 32-bits)
- Type: Type of service
- Length: size of datagram (header + data; in bytes)
- Identification: fragment ID
- Fragment offset: offset of current fragment (x 8 bytes)
- TTL: number of network hops
- Protocol: protocol type (e.g., TCP, UDP)
- Source IP addresses
- Destination IP address
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 a single administrative domain
- 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
Routing

- **Goal:** determine a “good” path through the network from source to destination
  - Good means usually the shortest path

- **Network modeled as a graph**
  - Routers → nodes
  - Link → edges
    - Edge cost: delay, congestion level,…
Interplay between routing and forwarding

routing algorithm

<table>
<thead>
<tr>
<th>header value</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

value in arriving packet’s header
Graph abstraction

Graph: $G = (N,E)$

$N = \text{set of routers} = \{u, v, w, x, y, z\}$

$E = \text{set of links} = \{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$

Remark: Graph abstraction is useful in other network contexts

Example: P2P, where $N$ is set of peers and $E$ is set of TCP connections
Graph abstraction: costs

- $c(x,x') =$ cost of link $(x,x')$
  - e.g., $c(w,z) = 5$
- cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

Question: What’s the least-cost path between $u$ and $z$?

Routing algorithm: algorithm that finds least-cost path
Routing Algorithm classification

Global or decentralized information?

Global:
- all routers have complete topology, link cost info
- “link state” algorithms

Decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

Static or dynamic?

Static:
- routes change slowly over time

Dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes
Outline

- Link State
  - Distance Vector
A Link State Routing Algorithm

Dijkstra’s algorithm

- Net topology, link costs known to all nodes
  - Accomplished via “link state flooding”
  - All nodes have same info
- Compute least cost paths from one node (‘source”) to all other nodes
- Iterative: after $k$ iterations, know least cost paths to $k$ closest destinations

Notations

- $c(i,j)$: link cost from node $i$ to $j$; cost infinite if not direct neighbors
- $D(v)$: current value of cost of path from source to destination $v$
- $p(v)$: predecessor node along path from source to $v$, that is next to $v$
- $S$: set of nodes whose least cost path definitively known
Link State Flooding Example
Link State Flooding Example
Link State Flooding Example
Link State Flooding Example
Dijsktra’s Algorithm

1 \textit{Initialization:}
2 \quad S = \{A\};
3 \quad \text{for all nodes } v
4 \quad \quad \text{if } v \text{ adjacent to } A
5 \quad \quad \quad \text{then } D(v) = c(A,v);
6 \quad \quad \text{else } D(v) = \infty;
7
8 \textit{Loop}
9 \quad \text{find } w \text{ not in } S \text{ such that } D(w) \text{ is a minimum;}
10 \quad \text{add } w \text{ to } S;
11 \quad \text{update } D(v) \text{ for all } v \text{ adjacent to } w \text{ and not in } S:
12 \quad \quad D(v) = \min( D(v), D(w) + c(w,v) );
13 \quad \quad \quad \text{// new cost to } v \text{ is either old cost to } v \text{ or known}
14 \quad \quad \quad \text{// shortest path cost to } w \text{ plus cost from } w \text{ to } v
15 \quad \text{until all nodes in } S;

\textbf{Notations}
- \( c(i,j) \): link cost from node \( i \) to \( j \); cost infinite if not direct neighbors
- \( D(v) \): current value of cost of path from source to destination \( v \)
- \( p(v) \): predecessor node along path from source to \( v \), that is next to \( v \)
- \( S \): set of nodes whose least cost path definitively known
Example: Dijkstra’s Algorithm

Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
--- | --- | --- | --- | --- | --- | --- |
0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
1
2
3
4
5

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>

Start at node A. For each step:

- **Loop**
  1. **Step 8**: Find the node w not in S such that D(w) is a minimum.
  2. **Step 9**: Add w to S.
  3. **Step 10**: Update D(v) for all v adjacent to w and not in S using the formula:
    
    \[ D(v) = \min( D(v), D(w) + c(w,v) ) \]

- **Step 11**: Repeat until all nodes are in S.

Diagram:

```
A --2-- B --3-- C
  |     |     |
  2    3    1
  |     |     |
  D --1-- E --2-- F
```

- **Minimizing Path**: Path from A to all other nodes.

---

Mao F04 20
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></td>
<td>2,D</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>2,D</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3,E</td>
<td></td>
<td>4,E</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></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

8. **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>∞</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>ADEBC</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 \textit{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>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
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;
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: \( O(n*\log(n)) \)
Oscillations

- Assume link cost = amount of carried traffic

- How can you avoid oscillations?
Outline

- Link State
  - Distance Vector
Distance Vector Routing Algorithm

- Iterative: continues until no nodes exchange info
- Asynchronous: nodes need *not* exchange info/iterate in lock steps
- 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 neighbors $V$ do
3      if $V$ adjacent to $A$
4         $D(A, V) = c(A, V)$;
5      else
6         • $D(A, V) = 8$
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                    /* 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

1 *Initialization*: 
2   for all neighbors V do 
3     if V adjacent to A 
4        D(A, V) = c(A, V); 
5     else 
6        D(A, V) = 8; 
7     NextHopCostDest.

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>8</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>8</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>

Mao F04 31
Example: 1st Iteration (C → A)

- **Node A**
  - Dest.: B, Cost: 2, NextHop: B
  - Dest.: C, Cost: 7, NextHop: C
  - Dest.: D, Cost: 8, NextHop: C

- **Node B**
  - Dest.: A, Cost: 2, NextHop: A
  - Dest.: C, Cost: 1, NextHop: C
  - Dest.: D, Cost: 3, NextHop: D

- **Node C**
  - Dest.: A, Cost: 7, NextHop: A
  - Dest.: B, Cost: 1, NextHop: B
  - Dest.: D, Cost: 1, NextHop: D

- **Node D**
  - Dest.: A, Cost: 8, NextHop: -
  - Dest.: B, Cost: 3, NextHop: B
  - Dest.: C, Cost: 1, NextHop: C

7 loop:

- else if (update D(V, Y) received from V)
- D(A, Y) = D(A, V) + D(V, Y);
- if (there is a new min. for destination Y)
- send D(A, Y) to all neighbors
- forever
Example: 1\textsuperscript{st} Iteration (B→A, C→A)

\begin{align*}
D(A, D) &= D(A, B) + D(B, D) = 2 + 3 = 5 \\
D(A, C) &= D(A, B) + D(B, C) = 2 + 1 = 3
\end{align*}

7 \textbf{loop:}

\begin{align*}
&\quad\ldots \\
13 &\text{else if } (\text{update } D(V, Y) \text{ received from } V) \\
14 &\quad D(A, Y) = \min(D(A, V), D(A, V) + D(V, Y)) \\
15 &\text{if } (\text{there is a new min. for destination } Y) \\
16 &\quad \text{send } D(A, Y) \text{ to all neighbors} \\
17 &\quad \text{forever}
\end{align*}
Example: End of 1\textsuperscript{st} Iteration

7 \textit{loop:}

\textbf{...}

13 \textbf{else if} (update $D(V, Y)$ received from $V$) \textbf{then}

14 $D(A, Y) = D(A, V) + D(V, Y)$;

15 \textbf{if} (there is a new min. for destination $Y$) \textbf{then}

16 \textbf{send} $D(A, Y)$ to all neighbors

17 \textbf{forever}
Example: End of 2\textsuperscript{nd} Iteration

7 \textit{loop}:

\text{...}

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 \textbf{forever}
Example: End of 3\textsuperscript{rd} Iteration

7 \textbf{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{2em} \textbf{send} $D(A, Y)$ to all neighbors
17 \textbf{forever}

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”

Node B

\[
\begin{array}{ccc}
A & 4 & A \\
C & 1 & B \\
\end{array}
\]

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

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

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

Node C

\[
\begin{array}{ccc}
A & 5 & B \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
A & 5 & B \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
A & 2 & B \\
B & 1 & B \\
\end{array}
\]

\[
\begin{array}{ccc}
A & 2 & B \\
B & 1 & B \\
\end{array}
\]
Distance Vector: Count to Infinity
Problem

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

“bad news travels slowly”

Link cost changes here; recall that B also maintains shortest distance to A through C, which is 6. Thus D(B, A) becomes 6!
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?

**Link cost changes here:** B updates $D(B, A) = 60$ as C has advertised $D(C, A) = 8$

**Algorithm terminates**
Look at this example

- Initially, both A and B have a distance 2 to D, C has a distance 1
- Assume CD goes down
- Using split horizon, A and B tell C that they cannot get to D
- C concludes D is unreachable, reports to A and B
- A hears that B has a path of length 2 to D
- B concludes it can get to D via A in 3 hops
- A and B set their distance to D to 4
- Count to infinity??
Link State vs. Distance Vector

Per node message complexity
- LS: $O(n^2)$ with $O(n \cdot e)$ messages
- DV: $O(d)$ messages; where $d$ is node’s degree

Complexity
- LS: $O(n^2)$ with $O(n \cdot e)$ messages
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?
- LS:
  - node can advertise incorrect link cost
  - each node computes only its own table
- DV:
  - node can advertise incorrect path cost
  - each node’s table used by others; error propagate through network