Today’s Lecture: 6

Application
Transport
Network (IP)
Link
Physical
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
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 \(\rightarrow\) nodes
  - Link \(\rightarrow\) edges
  - Edge cost: delay, congestion level,…
Outline

- Link State
  - Distance Vector

Link State: Control Traffic

- Each node floods its local information to every other node in the network
- Each node ends up knowing the entire network topology → use Dijkstra to compute the shortest path to every other node
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
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 \( S \):
      \[ D(v) = \min(D(v), D(w) + c(w,v)) \]
      // new cost to \( v \) is either old cost to \( v \) or known
      // shortest path cost to \( w \) plus cost from \( w \) to \( v \)
   12. until all nodes in \( S \);

Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start</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>

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 \( D(v) = \infty \);
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>2,D</td>
<td>4,E</td>
<td>4,E</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>2,D</td>
<td>4,E</td>
<td>4,E</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td></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></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>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
  - \( n^2 \) comparisons: \( O(n^2) \)
  - More efficient implementations possible: \( O(n \log(n)) \)
Oscillations

- Assume link cost = amount of carried traffic

```
A -> D: 1 + e
D -> B: 0
B -> C: 0
C -> A: 1 + e
```
Initially

```
A -> D: 1 + e
D -> B: e
B -> C: 1 + e
C -> A: 1 + e
```
... recompute routing

```
A -> D: 2 + e
D -> B: 0
B -> C: 0
C -> A: 2 + e
```
... recompute

- How can you avoid oscillations?

Outline

- Link State
  - Distance Vector
Distance Vector: Control Traffic

- When the routing table of a node changes, the node sends its table to its neighbors
- A node updates its table with information received from its neighbors

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. $D(A, V) = \infty$;
      7. loop:
      8. wait (until $A$ sees a link cost change to neighbor $V$ or until $A$ receives update from neighbor $V$)
      9. if $D(A, V)$ changes by $d$
      10. for all destinations $Y$ through $V$ do
      12. else if (update $D(V, Y)$ received from $V$)
      14. else if (there is a new minimum for destination $Y$)
      15. send $D(A, Y)$ to all neighbors
      16. 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. $D(A, V) = \infty$;

Example: 1st Iteration ($C \rightarrow A$)

7. loop:
   13. else if (update $D(V, Y)$ received from $V$)
   15. if (there is a new min. for destination $Y$)
   16. send $D(A, Y)$ to all neighbors
   17. forever
Example: 1st 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>

\[
D(A,D) = D(A,B) + D(B,D) = 2 + 3 = 5 \quad D(A,C) = D(A,B) + D(B,C) = 2 + 1 = 3
\]

### 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>

loop:

13 else if (update \( D(V,Y) \) received from \( V \))
14 \( D(A,Y) = \min(D(A,V), D(A,V) + D(V,Y)) \);
15 if (there is a new min. for destination \( Y \))
16 send \( D(A,Y) \) to all neighbors
17 forever

Example: End of 1st Iteration

### 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>2</td>
<td>C</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>3</td>
<td>B</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>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
</tbody>
</table>

loop:

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

```
loop:

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 min. for destination Y)
16     send D(A, Y) to all neighbors
17 forever
```

Example: End of 3rd Iteration

```
loop:

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 min. for destination Y)
16     send D(A, Y) to all neighbors
17 forever
```

Nothing changes → algorithm terminates
Distance Vector: Link Cost Changes

7 \textbf{loop:}
8 \textbf{wait} (until \( A \) sees a link cost change to neighbor \( V \))
9 \textbf{or} until \( A \) receives update from neighbor \( V \)
10 \textbf{if} \( D(A, V) \) changes by \( \delta \)
11 \textbf{for all} destinations \( Y \) through \( V \) \textbf{do}
12 \( D(A, Y) = D(A, Y) + \delta \)
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 minimum for destination \( Y \))
16 \textbf{send} \( D(A, Y) \) to all neighbors
17 \textbf{forever}

Node B
\begin{tabular}{ccc}
\hline
  A \& 4 \& A \\
  C \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  A \& 1 \& A \\
  C \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}

Node C
\begin{tabular}{ccc}
\hline
  A \& 5 \& B \\
  B \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  A \& 5 \& B \\
  B \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}

Link cost changes here
Algorithm terminates

“good news travels fast”

Distance Vector: Count to Infinity Problem

7 \textbf{loop:}
8 \textbf{wait} (until \( A \) sees a link cost change to neighbor \( V \))
9 \textbf{or} until \( A \) receives update from neighbor \( V \)
10 \textbf{if} \( D(A, V) \) changes by \( \delta \)
11 \textbf{for all} destinations \( Y \) through \( V \) \textbf{do}
12 \( D(A, Y) = D(A, Y) + \delta \)
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 minimum for destination \( Y \))
16 \textbf{send} \( D(A, Y) \) to all neighbors
17 \textbf{forever}

Node B
\begin{tabular}{ccc}
\hline
  A \& 4 \& A \\
  C \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  A \& 6 \& C \\
  C \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  A \& 8 \& C \\
  C \& 1 \& B \\
\hline
\end{tabular}

Node C
\begin{tabular}{ccc}
\hline
  A \& 5 \& B \\
  B \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  A \& 7 \& B \\
  B \& 1 \& B \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  D \& C \& N \\
\hline
\end{tabular}
\begin{tabular}{ccc}
\hline
  A \& 2 \& B \\
  B \& 1 \& B \\
\hline
\end{tabular}

Link cost changes here: recall from slide 24 that B also maintains shortest distance to A through C, which is 6. Thus \( D(B, A) \) becomes 6!

“bad news travels slowly”
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?

    ![Diagram of network with nodes A, B, and C showing link costs and updates]

Node B

- \[ D(B, A) = 60 \]
- \[ D(C, A) = \text{infinite} \]

Node C

- \[ D(C, C) = \text{infinite} \]
- \[ D(C, A) = 60 \]

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

Algorithm terminates

Link State vs. Distance Vector

Per node message complexity

- **LS**: \( O(n^2) \) messages; \( n \) – number of nodes; \( e \) – number of edges
- **DV**: \( O(d) \) messages; where \( d \) is node’s degree

Complexity

- **LS**: \( O(n^2) \) with \( O(n^2) \) 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