Network Layer, Control Plane

- **Function:**
  - Set up routes within a single network

- **Key challenges:**
  - Distributing and updating routes
  - Convergence time
  - Avoiding loops

---

**Data Plane**

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

**Control Plane**

- RIP
- OSPF
- BGP
Internet Routing

- Internet organized as a two level hierarchy
- First level — autonomous systems (AS’s)
  - AS — region of network under a single administrative domain
  - Examples: Comcast, AT&T, Verizon, Sprint, etc.
Internet Routing

- Internet organized as a two level hierarchy
- First level – autonomous systems (AS’s)
  - AS – region of network under a single administrative domain
  - Examples: Comcast, AT&T, Verizon, Sprint, etc.
- AS’s use intra-domain routing protocols internally
  - Distance Vector, e.g., Routing Information Protocol (RIP)
  - Link State, e.g., Open Shortest Path First (OSPF)
Internet Routing

- Internet organized as a **two** level hierarchy
- First level – autonomous systems (AS’s)
  - AS – region of network under a single administrative domain
  - Examples: Comcast, AT&T, Verizon, Sprint, etc.
- AS’s use **intra-domain** routing protocols internally
  - Distance Vector, e.g., Routing Information Protocol (RIP)
  - Link State, e.g., Open Shortest Path First (OSPF)
- Connections between AS’s use **inter-domain** routing protocols
  - Border Gateway Routing (BGP)
  - De facto standard today, BGP-4
AS Example
AS Example

AS-1

AS-2

AS-3

Interior Routers

BGP Routers
Why Do We Need ASs?

- Routing algorithms are not efficient enough to execute on the entire Internet topology
Why Do We Need ASs?

- Routing algorithms are not efficient enough to execute on the entire Internet topology
- Different organizations may use different routing policies
Why Do We Need ASs?

- Routing algorithms are not efficient enough to execute on the entire Internet topology
- Different organizations may use different routing policies
- Allows organizations to hide their internal network structure
Why Do We Need ASs?

- Routing algorithms are not efficient enough to execute on the entire Internet topology
- Different organizations may use different routing policies
- Allows organizations to hide their internal network structure
- Allows organizations to choose how to route across each other (BGP)
Why Do We Need ASs?

- Routing algorithms are not efficient enough to execute on the entire Internet topology.
- Different organizations may use different routing policies.
- Allows organizations to hide their internal network structure.
- Allows organizations to choose how to route across each other (BGP).

  - Easier to compute routes.
  - Greater flexibility.
  - More autonomy/independence.
Goal: determine a “good” path through the network from source to destination

- What is a good path?
  - Usually means the shortest path
  - Load balanced
  - Lowest $$$ cost
Routing on a Graph

- Goal: determine a “good” path through the network from source to destination

- What is a good path?
  - Usually means the shortest path
  - Load balanced
  - Lowest $$$ cost

- Network modeled as a graph
  - Routers \(\rightarrow\) nodes
  - Link \(\rightarrow\) edges
    - Edge cost: delay, congestion level, etc.
Routing Problems

- Assume
  - A network with N nodes
  - Each node only knows
    - Its immediate neighbors
    - The cost to reach each neighbor
- How does each node learn the shortest path to every other node?
Intra-domain Routing Protocols
Intra-domain Routing Protocols

- **Distance vector**
  - Routing Information Protocol (RIP), based on Bellman-Ford
  - Routers periodically exchange reachability information with neighbors
Intra-domain Routing Protocols

- **Distance vector**
  - Routing Information Protocol (RIP), based on Bellman-Ford
  - Routers periodically exchange reachability information with neighbors

- **Link state**
  - Open Shortest Path First (OSPF), based on Dijkstra
  - Each network periodically *floods* immediate reachability information to all other routers
  - Per router local computation to determine full routes
Outline

- Distance Vector Routing
  - RIP
- Link State Routing
  - OSPF
  - IS-IS
Distance Vector Routing

- What is a distance vector?
  - Current best known cost to reach a destination
  - Idea: exchange vectors among neighbors to learn about lowest cost paths
Distance Vector Routing

- What is a distance vector?
  - Current best known cost to reach a destination
- Idea: exchange vectors among neighbors to learn about lowest cost paths

<table>
<thead>
<tr>
<th>Destination</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

- No entry for C
- Initially, only has info for immediate neighbors
  - Other destinations cost = \( \infty \)
- Eventually, vector is filled
What is a distance vector?
- Current best known cost to reach a destination
- Idea: exchange vectors among neighbors to learn about lowest cost paths

No entry for C
- Initially, only has info for immediate neighbors
  - Other destinations cost = $\infty$
- Eventually, vector is filled

Routing Information Protocol (RIP)
Distance Vector Routing Algorithm

1. Wait for change in local link cost or message from neighbor

2. Recompute distance table

3. If least cost path to any destination has changed, notify neighbors
### Distance Vector Initialization

1. **Initialization:**

2. for all neighbors $V$ do

3. if $V$ adjacent to $A$

4. $D(A, V) = c(A, V)$;

5. else

6. $D(A, V) = \infty$;

...
Distance Vector: 1st Iteration

... loop:
...

12. else if (update D(V, Y) received from V)

13. for all destinations Y do

14. if (destination Y through V)

15. D(A, Y) = D(A, V) + D(V, Y);

16. else

17. D(A, Y) = min(D(A, Y),

D(A, V) + D(V, Y));

18. if (there is a new min. for dest. Y)

19. send D(A, Y) to all neighbors

20. forever

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>∞</td>
<td></td>
</tr>
</tbody>
</table>

Node B

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>Next</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>Next</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>
Distance Vector: 1st Iteration

---

### Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>Next</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>Next</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>Next</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>

---

... loop:

12. else if (update $D(V, Y)$ received from $V$)
13. for all destinations $Y$ do
14. if (destination $Y$ through $V$)
16. else
17. $D(A, Y) = \min(D(A, Y), D(A, V) + D(V, Y))$;
18. if (there is a new min. for dest. $Y$)
19. send $D(A, Y)$ to all neighbors
20. forever
Distance Vector: 1\textsuperscript{st} Iteration

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>Next</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

\[
D(A, D) = \min(D(A, D), D(A, C) + D(C, D)) = \min(\infty, 7 + 1) = 8
\]

Node D

Loop:

7. \texttt{loop:}

11. else if (update \(D(V, Y)\) received from \(V\))

12. for all destinations \(Y\) do

13. if (destination \(Y\) through \(V\))

14. \(D(A, Y) = D(A, V) + D(V, Y)\) 

15. else 

16. \(D(A, Y) = \min(D(A, Y), D(A, V) + D(V, Y))\)

17. if (there is a new min. for dest. \(Y\))

18. send \(D(A, Y)\) to all neighbors

19. forever
Distance Vector: 1\textsuperscript{st} Iteration

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>C</td>
</tr>
</tbody>
</table>

Node B

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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

D(A, D) = \min(D(A, D), D(A, C) + D(C, D))
= \min(\infty, 7 + 1) = 8

Node D
Distance Vector: 1\textsuperscript{st} Iteration

\begin{itemize}
\item \textit{loop:}
\item \textit{else if} (update $D(V, Y)$ received from $V$)\textbf{do}
\item \textbf{for all} destinations $Y$ \textbf{do}
\item \textbf{if} (destination $Y$ through $V$)\textbf{then}
\item $D(A, Y) = D(A, V) + D(V, Y)$;
\item \textbf{else}
\item $D(A, Y) = \min(D(A, Y), D(A, V) + D(V, Y))$;
\item \textbf{end if}
\item \textbf{end for}
\item \textbf{if} (there is a new min. for dest. $Y$)\textbf{then}
\item send $D(A, Y)$ to all neighbors
\item \textbf{end if}
\item \textbf{forever}
\end{itemize}

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
\textbf{Dest.} & \textbf{Cost} & \textbf{Next} \\
\hline
$B$ & 2 & $B$ \\
$C$ & 7 & $C$ \\
$D$ & 8 & $C$ \\
\hline
\end{tabular}
\end{center}
Distance Vector: 1\textsuperscript{st} Iteration

... loop:
... else if (update \(D(V, Y)\) received from \(V\))
13. for all destinations \(Y\) do
14. if (destination \(Y\) through \(V\))
15. \(D(A, Y) = D(A, V) + D(V, Y)\);
16. else
17. \(D(A, Y) = \min(D(A, Y), D(A, V) + D(V, Y))\);
18. if (there is a new min. for dest. \(Y\))
19. send \(D(A, Y)\) to all neighbors
20. forever

![Diagram of the network with nodes A, B, C, and D, showing distances and next steps.]

\[D(A, C) = \min(D(A, C), D(A, B) + D(B, C)) = \min(7, 2 + 1) = 3\]
loop:

else if (update D(V, Y) received from V)

for all destinations Y do

if (destination Y through V)

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

else

D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));

if (there is a new min. for dest. Y)

send D(A, Y) to all neighbors

forever

D(A, C) = min(D(A, C), D(A, B) + D(B, C))
= min(7, 2 + 1) = 3
Distance Vector: 1\textsuperscript{st} Iteration

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>8</td>
<td>C</td>
</tr>
</tbody>
</table>

Node B

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>Next</th>
</tr>
</thead>
<tbody>
<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>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
</tbody>
</table>

D(A,D) = \min(D(A,D), D(A,B)+D(B,D))
= \min(8, 2 + 3) = 5
Distance Vector: 1st Iteration

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>Next</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>Next</th>
</tr>
</thead>
<tbody>
<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>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
</tbody>
</table>

D(A,D) = min(D(A,D), D(A,B)+D(B,D))
= min(8, 2 + 3) = 5

loop:
else if (update D(V, Y) received from V)
for all destinations Y do
if (destination Y through V)
D(A, Y) = D(A, V) + D(V, Y);
else
D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
if (there is a new min. for dest. Y)
send D(A, Y) to all neighbors
forever
Distance Vector: 1\textsuperscript{st} Iteration

... loop:

12. \textbf{else if} (update D(V, Y) received from V)
13. \hspace{1em} \textbf{for all} destinations Y \textbf{do}
14. \hspace{2em} \textbf{if} (destination Y through V)
15. \hspace{3em} D(A, Y) = D(A, V) + D(V, Y);
16. \hspace{1em} \textbf{else}
17. \hspace{2em} D(A, Y) = \min(D(A, Y),
\hspace{3em} D(A, V) + D(V, Y));
18. \hspace{1em} \textbf{if} (there is a new min. for dest. Y)
19. \hspace{2em} send D(A, Y) to all neighbors
20. forever

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

Node A

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

Node D

...
Distance Vector: End of 3rd Iteration

7. \textit{loop:} 

12. \textbf{else if} (update $D(V, Y)$ received from $V$) 
13. \textbf{for all} destinations $Y$ \textbf{do} 
14. \hspace{1em} if (destination $Y$ through $V$) 
15. \hspace{2em} $D(A, Y) = D(A, V) + D(V, Y)$; 
16. \hspace{1em} else 
17. \hspace{2em} $D(A, Y) =$ 
18. \hspace{3em} \hspace{1em} min($D(A, Y)$, 
19. \hspace{4em} $D(A, V) + D(V, Y)$); 
20. \hspace{1em} if (there is a new min. for dest. $Y$) 
21. \hspace{2em} send $D(A, Y)$ to all neighbors 
22. \hspace{1em} forever
Distance Vector: End of 3rd Iteration

Node A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>4</td>
<td>B</td>
</tr>
</tbody>
</table>

Node B

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Cost</th>
<th>Next</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>

- Nothing changes, algorithm terminates
- Until something changes...
7. loop:
8. wait (link cost update or update message)
9. if (c(A,V) changes by d)
10. for all destinations Y through V do
12. else if (update D(V, Y) received from V)
13. for all destinations Y do
14. if (destination Y through V)
15. D(A,Y) = D(A,V) + D(V,Y);
16. else
17. D(A,Y) = min(D(A,Y), D(A,V) + D(V,Y));
18. if (there is a new minimum for destination Y)
19. send D(A,Y) to all neighbors
20. forever

Node B

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
7. loop:
8. wait (link cost update or update message)
9. if (c(A, V) changes by d)
10. for all destinations Y through V do
11. \(D(A, Y) = D(A, Y) + d\)
12. else if (update \(D(V, Y)\) received from V)
13. for all destinations Y do
14. if (destination Y through V)
15. \(D(A, Y) = D(A, V) + D(V, Y)\);
16. else
17. \(D(A, Y) = \min(D(A, Y), D(A, V) + D(V, Y))\);
18. if (there is a new minimum for destination Y)
19. send \(D(A, Y)\) to all neighbors
20. forever

Node B

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
7. loop:
8. \textbf{wait} (link cost update or update message)
9. \textbf{if} (c(A,V) changes by \(d\))
10. \textbf{for all} destinations \(Y\) through \(V\) do
11. \(D(A,Y) = D(A,Y) + d\)
12. \textbf{else if} (update \(D(V,Y)\) received from \(V\))
13. \textbf{for all} destinations \(Y\) do
14. \textbf{if} (destination \(Y\) through \(V\))
15. \(D(A,Y) = D(A,V) + D(V,Y);\)
16. \textbf{else}
17. \(D(A,Y) = \min(D(A,Y), D(A,V) + D(V,Y));\)
18. \textbf{if} (there is a new minimum for destination \(Y\))
19. \textbf{send} \(D(A,Y)\) to all neighbors
20. \textbf{forever}
7. loop:
8.   wait (link cost update or update message)
9.   if (c(A, V) changes by d)
10.   for all destinations Y through V do
12.   else if (update D(V, Y) received from V)
13.     for all destinations Y do
14.       if (destination Y through V)
15.         D(A, Y) = D(A, V) + D(V, Y);
16.       else
17.         D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
18.   end
19. else
20.   end

Link Cost Changes, Algorithm Starts
7. \textit{loop:}
8. \textbf{wait} (link cost update or update message)
9. \textbf{if} \(c(A,V)\) changes by \(d\)
10. \textbf{for all} destinations \(Y\) through \(V\) \textbf{do}
    11. \(D(A,Y) = D(A,Y) + d\)
12. \textbf{else if} (update \(D(V,Y)\) received from \(V\))
13. \textbf{for all} destinations \(Y\) \textbf{do}
    14. \textbf{if} (destination \(Y\) through \(V\))
        15. \(D(A,Y) = D(A,V) + D(V,Y);\)
    16. \textbf{else}
        17. \(D(A,Y) = \min(D(A,Y), D(A,V) + D(V,Y));\)
18. \textbf{if} (there is a new minimum for destination \(Y\))
19. \textbf{send} \(D(A,Y)\) to all neighbors
20. \textbf{forever}

---

**Link Cost Changes, Algorithm Starts**

<table>
<thead>
<tr>
<th>Time</th>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\begin{tabular}{</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>\begin{tabular}{</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>\begin{tabular}{</td>
<td>c</td>
</tr>
</tbody>
</table>

**Algorithm Terminates**
7. **loop:**
8. **wait** (link cost update or update message)
9. if (c(A,V) changes by \(d\))
10. **for all** destinations \(Y\) through \(V\) do
11. \(D(A,Y) = D(A,Y) + d\)
12. else if (update \(D(V,Y)\) received from \(V\))
13. **for all** destinations \(Y\) do
14. if (destination \(Y\) through \(V\))
15. \(D(A,Y) = D(A,V) + D(V,Y)\);
16. else
17. \(D(A,Y) = \min(D(A,Y), D(A,V) + D(V,Y))\);
18. if (there is a new minimum for destination \(Y\))
19. send \(D(A,Y)\) to all neighbors
20. forever

Good news travels fast

<table>
<thead>
<tr>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>D C N</td>
<td>D C N</td>
</tr>
<tr>
<td>A 4 A</td>
<td>A 1 A</td>
</tr>
<tr>
<td>C 1 B</td>
<td>C 1 B</td>
</tr>
<tr>
<td>A 5 B</td>
<td>A 5 B</td>
</tr>
<tr>
<td>B 1 B</td>
<td>B 1 B</td>
</tr>
<tr>
<td>A 2 B</td>
<td>A 2 B</td>
</tr>
<tr>
<td>B 1 B</td>
<td>B 1 B</td>
</tr>
</tbody>
</table>
Count to Infinity Problem

Node B

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
Count to Infinity Problem

Node B

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Time

A → B: 60
B → C: 1
B → A: 50
Node B knows $D(C, A) = 5$
However, B does not know the path is $C \rightarrow B \rightarrow A$
Thus, $D(B,A) = 6$!
Node B knows $D(C, A) = 5$
However, B does not know the path is $C \rightarrow B \rightarrow A$
Thus, $D(B, A) = 6$!
Count to Infinity Problem

Node B

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
Count to Infinity Problem

Node B
- A: 4
- C: 1

Node C
- A: 5
- B: 1

Time
1
50
60

Flow:
- A: 6 → C
- C: 1 → B
- A: 7 → B
Count to Infinity Problem

Bad news travels slowly

Node B

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Node C

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
Poisoned Reverse

- If C routes through B to get to A
  - C tells B that $D(C, A) = \infty$
  - Thus, B won’t route to A via C
Poisoned Reverse

- If C routes through B to get to A
  - C tells B that $D(C, A) = \infty$
  - Thus, B won’t route to A via C

---

<table>
<thead>
<tr>
<th>Node B</th>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node C</th>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
Poisoned Reverse

- If C routes through B to get to A
  - C tells B that \( D(C, A) = \infty \)
  - Thus, B won’t route to A via C

Node B

- Time: 50
  - Node B:
    - \( D \): 4
    - \( C \): 1
    - \( N \): A

- Time: 60
  - Node B:
    - \( D \): 60
    - \( C \): 1
    - \( N \): A

Node C

- Time: 50
  - Node C:
    - \( D \): 5
    - \( C \): 1
    - \( N \): B

- Time: 60
  - Node C:
    - \( D \): 5
    - \( C \): 1
    - \( N \): B
Poisoned Reverse

- If C routes through B to get to A
  - C tells B that $D(C, A) = \infty$
  - Thus, B won’t route to A via C
Poisoned Reverse

- If C routes through B to get to A
  - C tells B that \( D(C, A) = \infty \)
  - Thus, B won’t route to A via C

<table>
<thead>
<tr>
<th>Node</th>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node B</td>
<td>A</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>Node C</td>
<td>A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>D</th>
<th>C</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>A</td>
<td>60</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>50</td>
<td>A</td>
<td>50</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
If C routes through B to get to A

C tells B that $D(C, A) = \infty$

Does this completely solve this count to infinity problem?

NO
Multipath loops can still trigger the issue
Outline

- Distance Vector Routing
  - RIP
- Link State Routing
  - OSPF
  - IS-IS
Link State Routing

- Each node knows its connectivity and cost to direct neighbors
Link State Routing

- Each node knows its connectivity and cost to direct neighbors
- Each node tells every other node this information
Link State Routing

- Each node knows its connectivity and cost to direct neighbors
- Each node tells every other node this information
Link State Routing

- Each node knows its connectivity and cost to direct neighbors
- Each node tells every other node this information
Link State Routing

- Each node knows its connectivity and cost to direct neighbors
- Each node tells every other node this information
Each node knows its connectivity and cost to direct neighbors
Each node tells every other node this information
Each node learns complete network topology
Each node knows its connectivity and cost to direct neighbors.

Each node tells every other node this information.

Each node learns complete network topology.
Link State Routing

- Each node knows its connectivity and cost to direct neighbors
- Each node tells every other node this information
- Each node learns complete network topology
- Use Dijkstra to compute shortest paths
Flooding Details

- Each node periodically generates Link State Packet
  - ID of node generating the LSP
  - List of direct neighbors and costs
  - Sequence number (64-bit, assumed to never wrap)
  - Time to live
Flooding Details

- Each node periodically generates Link State Packet
  - ID of node generating the LSP
  - List of direct neighbors and costs
  - Sequence number (64-bit, assumed to never wrap)
  - Time to live

- Flood is reliable (ack + retransmission)
Flooding Details

- Each node periodically generates Link State Packet
  - ID of node generating the LSP
  - List of direct neighbors and costs
  - Sequence number (64-bit, assumed to never wrap)
  - Time to live
- Flood is reliable (ack + retransmission)
- Sequence number “versions” each LSP
Flooding Details

- Each node periodically generates Link State Packet
  - ID of node generating the LSP
  - List of direct neighbors and costs
  - Sequence number (64-bit, assumed to never wrap)
  - Time to live

- Flood is reliable (ack + retransmission)

- Sequence number “versions” each LSP

- Receivers flood LSPs to their own neighbors
  - Except whoever originated the LSP
Flooding Details

- Each node periodically generates Link State Packet
  - ID of node generating the LSP
  - List of direct neighbors and costs
  - Sequence number (64-bit, assumed to never wrap)
  - Time to live

- Flood is reliable (ack + retransmission)
- Sequence number “versions” each LSP
- Receivers flood LSPs to their own neighbors
  - Except whoever originated the LSP
- LSPs also generated when link states change
Dijkstra'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. \( \ldots \)
Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Start S</th>
<th>→B</th>
<th>→C</th>
<th>→D</th>
<th>→E</th>
<th>→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>
</tbody>
</table>

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:
12. $D(v) = \min(D(v), D(w) + c(w,v))$;
13. *until all nodes in S*;
## Dijkstra’s Algorithm

### Step-by-Step Table

<table>
<thead>
<tr>
<th>Step</th>
<th>Start S</th>
<th>⇒B</th>
<th>⇒C</th>
<th>⇒D</th>
<th>⇒E</th>
<th>⇒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>
</tbody>
</table>

### Diagram

![Dijkstra's Algorithm Diagram]

### 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:
12. \( D(v) = \min( D(v), D(w) + c(w,v) ) \);
13. until all nodes in S;
## Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Start S</th>
<th>→B</th>
<th>→C</th>
<th>→D</th>
<th>→E</th>
<th>→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></td>
<td>4, E</td>
</tr>
</tbody>
</table>

...  

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:
12. \( D(v) = \min( D(v), D(w) + c(w,v) ) \);
13. until all nodes in S;

---

**Diagram:**

[Diagram showing the network and the steps of Dijkstra's Algorithm]
# Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Start S</th>
<th>(\rightarrow B)</th>
<th>(\rightarrow C)</th>
<th>(\rightarrow D)</th>
<th>(\rightarrow E)</th>
<th>(\rightarrow 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></td>
<td>2, D</td>
<td>(\infty)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td></td>
<td>3, E</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>
</tbody>
</table>

... 

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:
12. \(D(v) = \min(D(v), D(w) + c(w,v))\);
13. until all nodes in S;
**Dijkstra's Algorithm**

<table>
<thead>
<tr>
<th>Step</th>
<th>Start S</th>
<th>→B</th>
<th>→C</th>
<th>→D</th>
<th>→E</th>
<th>→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>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...  

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 \( S \):  
12. \( D(v) = \min( D(v), D(w) + c(w,v) ) \);  
13. **until all nodes in \( S \);**
Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Start S</th>
<th>→B</th>
<th>→C</th>
<th>→D</th>
<th>→E</th>
<th>→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>ADE</td>
<td></td>
<td></td>
<td>3, E</td>
<td></td>
<td>4, E</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;
OSPF vs. IS-IS

- Two different implementations of link-state routing

OSPF

IS-IS
OSPF vs. IS-IS

- Two different implementations of link-state routing
- Favored by companies, datacenters
OSPF vs. IS-IS

- Two different implementations of link-state routing

- Favored by companies, datacenters
- More optional features
## OSPF vs. IS-IS

- **Two different implementations of link-state routing**

### OSPF
- Favored by companies, datacenters
- More optional features
- Built on top of IPv4
  - LSAs are sent via IPv4
  - OSPFv3 needed for IPv6

### IS-IS
- Favored by ISPs
- Less “chatty”
  - Less network overhead
  - Supports more devices
- Not tied to IP
  - Works with IPv4 or IPv6
Different Organizational Structure

- Organized around overlapping areas
- Area 0 is the core network
Different Organizational Structure

- Organized around overlapping areas
- Area 0 is the core network
Different Organizational Structure

- Organized around overlapping areas
- Area 0 is the core network

OSPF

- Area 0
- Area 1
- Area 2
- Area 3
- Area 4

IS-IS
Different Organizational Structure

**OSPF**
- Organized around overlapping areas
- Area 0 is the core network

**IS-IS**
- Organized as a 2-level hierarchy
Different Organizational Structure

**OSPF**
- Organized around overlapping areas
- Area 0 is the core network

**IS-IS**
- Organized as a 2-level hierarchy

Area 0 is the core network in both OSPF and IS-IS, but their organizational structures differ. OSPF is organized around overlapping areas, whereas IS-IS is organized as a 2-level hierarchy.
Different Organizational Structure

**OSPF**
- Organized around overlapping areas
- Area 0 is the core network

**IS-IS**
- Organized as a 2-level hierarchy

![OSPF Diagram](image)

![IS-IS Diagram](image)
Different Organizational Structure

**OSPF**
- Organized around overlapping areas
- Area 0 is the core network

**IS-IS**
- Organized as a 2-level hierarchy
- Level 2 is the backbone
## Link State vs. Distance Vector

<table>
<thead>
<tr>
<th></th>
<th>Link State</th>
<th>Distance Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Message Complexity</strong></td>
<td>$O(n^2 e)$</td>
<td>$O(d n k)$</td>
</tr>
<tr>
<td><strong>Time Complexity</strong></td>
<td>$O(n \log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Convergence Time</strong></td>
<td>$O(1)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td>• Nodes may advertise incorrect link costs</td>
<td>• Nodes may advertise incorrect path cost</td>
</tr>
</tbody>
</table>

$n =$ number of nodes in the graph  
$d =$ degree of a given node  
$k =$ number of rounds
### Link State vs. Distance Vector

<table>
<thead>
<tr>
<th></th>
<th>Link State</th>
<th>Distance Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Message Complexity</strong></td>
<td>$O(n^2e)$</td>
<td>$O(d<em>n</em>k)$</td>
</tr>
<tr>
<td><strong>Time Complexity</strong></td>
<td>$O(n \log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Convergence Time</strong></td>
<td>$O(1)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td>- Nodes may advertise</td>
<td>- Nodes may advertise</td>
</tr>
<tr>
<td></td>
<td>incorrect link costs</td>
<td>incorrect path costs</td>
</tr>
</tbody>
</table>

- Which is best?
- In practice, it depends.
- In general, link state is more popular.