• **Routing** is the process of creating and maintaining forwarding tables.

**Forwarding Table**

<table>
<thead>
<tr>
<th>Header value</th>
<th>Output Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xx</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
</tr>
<tr>
<td>11x</td>
<td>1</td>
</tr>
</tbody>
</table>

• **Forwarding** uses the table to determine the output link for each packet.
traceroute

- Routing Theory: Graphs and Overview
- Link State Algorithms
- Distance Vector Algorithms
Graph Abstraction

- Internet is composed of routers/hosts connected with links
- Can be modeled as a big graph
  - $G = (N, E)$
  - $N =$ Set of routers
  - $E =$ Set of links
Costs

- $c(N_1, N_2) =$ cost of the link between $N_1 \rightarrow N_2$
- ex: $c(w, z) = 5$, $c(x, z) = \infty$
- Cost could mean: 1 (hopcount), latency, congestion or inverse of bandwidth
- Cost of path $(N_1, N_2, .. N_p) =$
  
  $c(N_1, N_2) + c(N_2, N_3) + .. + c(N_{p-1}, N_p)$

![Diagram of network with labeled costs](image)
• What is the least-cost path between u and z?
• There are 17 different paths
• Routing Algorithm: find the least-cost path between any pairs of nodes
• When u forwards a packet bound for z:
  • Choose exit link with least-cost path
Algorithm Classifications

- **Global (“Link State” algorithms)**
  - All routers have complete topology information and all link costs

- **Decentralized (“Distance Vector” algorithms)**
  - Each router starts with just local knowledge
    - physically-connected neighbors
  - link costs to neighbors
  - Iterative process of computation, exchange of info with neighbors
traceroute

- Routing Theory: Graphs and Overview
- Link State Algorithms
- Distance Vector Algorithms
Link State Algorithms

• Use global knowledge: All routers know all
  • Connectivity, edge weights
• How do all routers learn? Flooding
  • Each node sends its link-state information on all directly connected links
  • Each node relays such information on all of its links, etc
After the Flood

• Each router calculates routes based on the link-state information
• Deterministic algorithm, so each router comes up with same answer
• Several algorithms exist: Dijkstra’s is most famous
Dijkstra’s Algorithm

• Iterative algorithm
• after $k$ steps, know the least cost path to $k$ closest locations

• Notation
• $c(x,y)$: link cost from node $x$ to $y$; $\infty$ if no direct link
• $D(v)$: current value of cost of path from source to node $v$
• $p(v)$: predecessor node along path from source to node $v$
• $N'$: set of nodes whose least-cost path is known
Initialization:

\[ N' = \{u\} \]

for all nodes \( v \)

if \( v \) adjacent to \( u \)

then \( D(v) = c(u,v) \)

else \( D(v) = \infty \)

Loop until \( N' \) contains all nodes

find \( w \) not in \( N' \) such that \( D(w) \) is a minimum

add \( w \) to \( N' \)

for all nodes \( v \) adjacent to \( w \) and \( \not\in N' \)

\[ D(v) = \min( D(v), D(w) + c(w,v) ) \]

/* new cost to \( v \) changes if path through \( w \) costs less */
### Example

<table>
<thead>
<tr>
<th>Step</th>
<th>( N' )</th>
<th>( D(v), p(v) )</th>
<th>( D(w), p(w) )</th>
<th>( D(x), p(x) )</th>
<th>( D(y), p(y) )</th>
<th>( D(z), p(z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( u )</td>
<td>( 2, u )</td>
<td>( 5, u )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 1, u )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>( ux )</td>
<td>( 2, u )</td>
<td>( 4, x )</td>
<td></td>
<td>( 2, x )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>2</td>
<td>( uxy )</td>
<td>( 2, u )</td>
<td>( 3, y )</td>
<td></td>
<td></td>
<td>( 4, y )</td>
</tr>
<tr>
<td>3</td>
<td>( uxyv )</td>
<td></td>
<td></td>
<td>( 3, y )</td>
<td></td>
<td>( 4, y )</td>
</tr>
<tr>
<td>4</td>
<td>( uxyvw )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 4, y )</td>
</tr>
<tr>
<td>5</td>
<td>( uxyvwz )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shortest Path Tree

• For a graph $G$ and a particular node $r$ ...

• ... the SPT($G, r$) is a tree...

• ... with the shortest path from the root to any other node $d$ in the graph

A SPT for Europe (all roads lead to Rome)
Dijkstra’s Results

• SPT \((G, u)\) and a forwarding table for \(u\)

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
</tr>
</tbody>
</table>
Complexity

• Algorithm complexity: n nodes
  • each iteration: check all nodes not in N
  • first iteration, check n nodes
  • 2nd iteration, check n-1 nodes ...
• Total is n(n+1)/2 comparisons ➞ O(n^2)
• more efficient implementations possible
  • Using a heap ➞ O(n log n)
traceroute

- Routing Theory: Graphs and Overview
- Link State Algorithms
- Distance Vector Algorithms
Distance-Vector Algs

• Still need to distribute local information
  • Message exchange rather than flooding
• Each node exchanges distance vector with neighboring nodes
  • 1d map of nodes to distances
Routing Algorithm

• Iterative: Each local iteration caused by ..
  • ... A change in local link cost
  • ... or DV update message from neighbor

• Distributed
  • Each node autonomously computes based on local knowledge ...
  • ... which, after “enough” iterations is communicated to the world
Convergence

• At each node:

  - \textit{wait for change}
  - \textit{recompute} estimates, based on change
  - if DV to any destination has changed, \textit{notify} neighbors

• Convergence: process of getting consistent information to all nodes
Bellman-Ford Eqn

- Define $d_x(y)$ as cost of the least-cost path from $x$ to $y$
- Bellman-Ford Equation says
  - $d_x(y) = \min_v \{c(x,v) + d_v(y)\}$
  - where $\min_v$ means the min for all neighbors $v$ of $x$

xkcd.com/69
B-F Example

- Neighbors of U:
  - $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$
  - $d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z) \}$
  - $d_u(z) = \min \{ 2 + 5, 1 + 3, 5 + 3 \} = 4$
- Node that achieves minimum is next hop in the shortest path
- $x$ goes in the forwarding table
Putting it all together

- Each node periodically sends its own distance vector estimates to neighbors.
- When a node $x$ receives a new DV estimate from a neighbor $v$, uses B-F:
  \[ D_x(y) \leftarrow \min_v \{ c(x,v) + D_v(y) \} \] for each $y \in N$.
- The estimate $D_x(y)$ converges to the actual $d_x(y)$ for minor, natural conditions.
Example

• At t=0
Example

• Node u receives DV from w, v, x

\[
d_u(w) = \min \{c(u,u) + d_u(w), c(u,v) + d_v(w), c(u,w) + d_w(w), c(u,x) + d_x(w)\}
\]
\[
d_u(w) = \min \{0 + 5, 2 + 3, 5 + 0, 1 + 3\}
\]
\[
d_u(w) = 4
\]
Example

• After 1 exchange
Example

• After 2 exchanges

<table>
<thead>
<tr>
<th>Node U</th>
<th>Node V</th>
<th>Node W</th>
<th>Node X</th>
<th>Node Y</th>
<th>Node Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

• After 3 exchanges

<table>
<thead>
<tr>
<th>Node U</th>
<th>Node V</th>
<th>Node W</th>
<th>Node X</th>
<th>Node Y</th>
<th>Node Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
</tbody>
</table>

Node U values: u=0, v=2, w=3, x=1, y=2, z=4
Node V values: u=2, v=0, w=3, x=2, y=3, z=5
Node W values: u=3, v=3, w=0, x=2, y=1, z=3
Node X values: u=1, v=2, w=2, x=0, y=1, z=3
Node Y values: u=2, v=3, w=1, x=1, y=0, z=2
Node Z values: u=4, v=5, w=3, x=3, y=2, z=0
Dynamics

• If a link cost changes
  • Node detects local link cost change, updates own forwarding table (recalculates DV)
• If DV changes, will notify neighbors
• “Good News travels fast” (one link-radius per exchange)
Dynamics (2)

• But: Bad news travels slowly

• If a link cost increases, can create a routing loop that slowly (and incorrectly) increases costs

• “Count to Infinity” problem

• Ex: If link c(x,y) changes to 60, Z still thinks there is a route to X of 5
In the Beginning

Oh, no! A change occurs

Finally, stability
Stabilization Techniques

• **Split horizon**

  • When a node sends a routing update to its neighbor, does not send those routes it learned from each neighbor back to that neighbor

• **Split horizon with poisoned reverse**

  • Nodes advertise a cost of $\infty$ for a destination to the neighbor it routes through to that destination
Example

• Since Z routes through Y to get to X, it tells Y that $D_Z(x) = \infty$
• $D_Z(x)$ is actually 5
• Now, Y calculates $D_Y(x) = \min\{60, 1 + \infty\}$
• Y will tell Z that $D_Y(x) = 60$, so Z will calculate $D_Z(x) = \min\{10, 1 + 60\}$
Can also redefine $\infty$

- Assume maximum costs to get to anywhere is $C$
- Do not let cost get $> C$ in calculation
- This bounds the time to “count to infinity”
- Unfortunately, no technique completely solves the “count to infinity” problem
- When routing loop contains 3+ nodes
Comparison

• Message Complexity
  • LS: with n nodes, E links, O(nE) messages sent
  • DV: exchange between neighbors only

• Speed of Convergence
  • LS: After correct message exchange
  • DV: Varies. May be routing loops
Comparison (2)

- **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 propagates thru network
Lesson Objectives

• Now, you should be able to:
  • describe the differences between global / decentralized and static / dynamic routing algorithms. Students should be able to describe different message complexity, convergence speeds, robustness and algorithm complexity
  • calculate a forwarding table using Dijkstra's algorithm (which may include identifying and using proper variables and terms). Intermediate results may be required, such as an SPT or table of variable values
You should be able to:

- use Bellman-Ford equations to calculate a forwarding table for a DV routing algorithm. Intermediate values may be required, which may require knowing variable names and terms.
- describe how DV algorithms operate to pass updates.
- describe DV instability problems, such as "Count to Infinity" and the associated stabilization techniques.
- analyze DV instability examples.