Next we study a routing algorithm that uses decentralized information:

- Distance Vector Routing

The Network Layer: Routing & Addressing

- Network layer services
- Routing algorithms
  - Least cost path computation algorithms
- Hierarchical routing
  - Connecting networks of networks
- IP Internet Protocol
  - Addressing
  - IPv6
- Routing on the Internet
  - Intra-domain routing
  - Inter-domain routing
Global routing: When a router is making a routing decision it is doing so based on complete knowledge of the network.

In link state routing each node maintains a graph representation of the entire network.

Decentralized routing: The calculation of least-cost path is done in a distributed, iterative manner and then the results are stored.

Called distance vector because a router never actually knows the path from a source to a destination (it only knows a vector towards the destination).

Distance vector routing differs from link state routing in three ways...

Note that the flooding portion of the link-state algorithm was also iterative, asynchronous, and distributed.

It was also self-terminating, however, this was easier to see because all nodes ever did was forward messages unmodified (or modified with later timestamps).

Iterative:
- Nodes exchange cost information until each node has the current route costs
- The algorithm is self-terminating — there’s no explicit stopping point

Asynchronous:
- Nodes need not exchange information and iterate in lock step
- Intermediate results may be inconsistent across nodes

Distributed:
- Each node communicates only with directly-attached adjacent nodes
- (But there is no flooding of cost information)
The principle data structure in the routing algorithm is the **distance table**.

The **distance vector algorithm** computes a distance table each time the cost of a link changes in the network.

This same formulation was also being used in link-state routing algorithm in **Dijkstra**'s algorithm, whenever a new node was added to set N, costs to its neighbors was updated using the above formulation.

**Computing the distance from E to C via D requires knowledge of D's distance table.** We'll see later how this information is learned.

Note that the minimum cost from E to A via D is a loop from E to D back to E.

Remember that in distance vector routing nodes do not have a map of the network (they do not know the graph) hence they can't detect routes that loop back to themselves.

E doesn’t know that D reaches A via E.

\[
D^E(X,Y) = \text{distance from } X \text{ to } Y \text{ via } Z \text{ as first hop} \\
= c(X,Z) + \min_w \{ D^E(Y,w) \} \\
\text{ } w = \text{neighbors of } Z
\]
The routing table is created from the distance table by taking the minimum cost entry in each row.

So when node E receives a packet bound for node B it will forward it to node D.

The distance vector algorithm is known as the Bellman-Ford algorithm after its inventors.

Note that the algorithm has a flooding-like aspect and indeed this is how link cost changes are propagated throughout the network.

However, unlike link state routing, the arrival of a message (and the subsequent processing of the message) may result in a large number of new messages being sent out.

Because the cost to one destination changes, the cost to lots of other destinations may change.

Thus distance vector routing can produce much more network traffic than link state routing.

Distance Vector Routing

Distance table example

<table>
<thead>
<tr>
<th>Destination</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

Routing Table

<table>
<thead>
<tr>
<th>Destination</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>D</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>D</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>D</td>
<td>2</td>
</tr>
</tbody>
</table>

- The distance table gives the routing table
  - Just take the minimum cost per destination

Distance Vector Routing

Algorithm

- Iterative, asynchronous: each local iteration caused by:
  - Local link cost change, or
  - Message from adjacent node that its least cost path to some destination has changed

- Distributed:
  - Each node notifies adjacent nodes only when its least cost path to some destination changes
  - Adjacent nodes then notify their adjacent nodes if this update changes a least cost path

Each node:

- wait for change in local link cost or message from adjacent node
- recompute distance table
- If least cost path to any destination has changed, notify adjacent nodes
Initially all we know is how to reach our immediate neighbors. Specifically, the only destinations we know of are the other nodes to which we are directly attached.

You wait until:
- You see a link cost change to a neighbor \( V \), or
- \( V \) tells us of a new minimum cost to a node \( Y \).

In the former case we update the cost of all destinations reachable via \( V \) by the delta in the cost of the link from \( X \) to \( V \).

In the latter case we only update one entry in our distance table.

We then compute our new routing table and if any entries have changed we send out the updated route cost information to all our neighbors.

Distance vector routing is more chaotic than link state routing.

Does the algorithm terminate? Does it result in nodes agreeing on least cost paths?
This is an example of building up the initial distance table (and routing table).

Each node starts out knowing the cost to each of its directly attached neighbors.

Each node sends its current minimum cost to each reachable node to each of its neighbors.

Consider the case of node X receiving the first updates from Y and Z.

Y tells X that it can reach X with a cost of 2 and Z with a cost of 1.

Z tells X that it can reach X with a cost of 7 and Y with a cost of 1.

Since Y is now reachable via Z, X computes the cost of a path to Y via Z.

The same happens for a path to Z via Y.

In the first step nodes Y and Z would also be computing new tables based on the information received from the other nodes.

— After this step nodes X and Z now have new lower cost paths to each other hence they tell all the other nodes about this.

— This new information doesn’t cause any changes to the distance table and hence no changes to any minimum cost path.

Here the algorithm terminates after three steps (taken asynchronously and independently by each node).

— (You'll complete this example as a homework problem.)
The more common case is running the Bellman-Ford algorithm in response to link cost changes. In this example, node X’s routing table is also changing but we don’t show this. X would have new lower cost paths to Y and Z as a result of the link cost change.

It turns out that the Bellman-Ford algorithm has surprisingly different performance when a link cost change makes a link more expensive. Eventually Y and Z will have consistent routes to X, however in the interim packets will be stuck in a routing loop bouncing back and forth between Y and Z. They route back & forth until the cost of route through X becomes the minimum cost route.

In the meantime:

- Buffers may overflow,
- Ttl’s may expire,
- Traffic load increases.

In the count-to-infinity problem the increment by which counting is done is a function of the costs of the links between nodes in the loop. So the routes may actually converge rather quickly in practice. But the number of messages exchanged is also a function of the number of nodes in the loop.

Also note that the fact that X’s distance table isn’t shown doesn’t help things because no one is going to route through X until the nodes in the routing loop realize that the minimum cost path is through X.
Initially Y and Z only have direct routes to X.

Y and Z exchange routes and both fill in their distance tables.

Z recognizes that its cheapest route to X is now via Y.

Since Z routes to X via Y it realizes that if it advertises a route to X a routing loop will be formed.

Z thus lies to Y and tells Y that its cost to X is infinite.

—

Y then thinks it can get to X via Z.

—

So technically the distance tables are now inconsistent.

Y thinks there is no route to X via Z.

When the cost of the link from Y to X changes, Y tells all of its neighbors.

When Y advertises a new route to X (the old route with a new cost), Z realizes that its new minimum cost route is now direct through X.

—

Z stops lying to Y about its route to X.

—

Y updates its distance table and realizes its cheapest route to X is via Z.

—

Y thus lies to Z about its route to X (to avoid a loop with Z).

—

Poisoned reverse only helps in the case where routing loops involve only two nodes.
Routing loops don’t occur much in link-state, because everyone has up-to-date information within a flooding interval.

Count-to-infinity never occurs in link state, because each router has complete topology.

Oscillations can occur in distance-vector also, when a load-sensitive cost-metric is used.

Robustness:
- What happens if there are failures?
  - 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
- Errors propagate through network

Robustness: Damage is more limited in the case of link state routing because routers only advertise the cost of links (not routes).

In distance-vector routing, one error (one wrongly advertised route) can bring down the entire Internet.

This finishes the material on least-cost path computation algorithms.

Link state or distance vector simply won’t scale to the global Internet.

Next we briefly look at hierarchical routing. This is how routing is actually done on the Internet and we’ll go into this in more detail in a future lecture.

The Network Layer: Routing & Addressing

- Network layer services
- Routing algorithms
  - Least cost path computation algorithms
- Hierarchical routing
  - Connecting networks of networks
- IP Internet Protocol
  - Addressing
    - IPv6
- Routing on the Internet
  - Intra-domain routing
  - Inter-domain routing
The Network Layer: Routing & Addressing

Hierarchical routing

- The theory of routing: relatively simple algorithms with manageable shortcomings

- Critical assumptions:
  - All routers are identical
  - The network is "flat"

- The reality: Routing is dominated by issues of scale
  - The Internet has 100 million hosts!
  - Can’t store all host destinations in routing tables!
  - Routing table exchange would swamp links!
  - We must route to networks, not hosts

- Routing also dominated by issues of administrative autonomy
  - The Internet is a network of networks — each network owner may want to control routing in its own network

Hierarchical Routing

Gateway routers

- Aggregate routers into regions, "autonomous systems" (AS)

- All routers inside same AS run same routing protocol among themselves
  - "Intra-AS" routing protocol
  - Routers in different AS can run different intra-AS routing protocol

Gateway routers

- Special routers in AS
- Run intra-AS routing protocol with all other routers inside AS
- Responsible for routing to destinations outside AS
  - Also run inter-AS routing protocol with gateway routers in adjacent AS
The solution to all computer science problems is to add a level of indirection. —

In this case we create a hierarchy.

Note that the link from A.c to A.a is a logical and not a physical link.

- Gateways:
  - Perform inter-AS routing among themselves
  - Perform intra-AS routing with other routers in their AS

- We’ll examine specific inter-AS and intra-AS Internet routing protocols shortly