Welcome to this course! My name is Jasleen Kaur and I'm the instructor for this course. We'll spend today's class talking about the course outline, the course requirements, and your first homework. But before we do that, I'd like for us all to spend some time and introduce ourselves. So if each of you can say 3 things: (1) what is your name, (2) what is your major and year, and (3) what is it that you expect to learn from this course.

So let me begin: my name is Jasleen and I conduct research in the design and analysis of networks and distributed systems. And I'm looking forward to a semester-worth of exchanging ideas with all of you.

How many have NOT had socket programming?

What do you think is the "Internet"?

Forwarding vs. Routing

Local vs. Distributed

Both datagram and virtual-circuit based networks need to know how to construct forwarding tables.

Forwarding:

» Select an output port based on destination address and routing table
» Simple and well-defined process performed locally at a given node
» Forwarding table used while forwarding packets
  ℹ eg: (network number, interface id, MAC address)

Routing:

» Building the forwarding/routing table
» A complex, distributed algorithm problem
» Routing table may be a precursor to a forwarding table
  ℹ eg: (network number, nextHopIP, CostToDestination)

Why do we need two tables?

There are at least two ways to think about the Internet (and also about what we will cover in this course):

The first is in terms of the components that make up the Internet. So this would be a nuts-and-bolts view in which we can talk about several acronyms and terms that you may have come across, such as the:

The second way to think about it is the one that is more common, which is in terms of the services and applications that run over the Internet. So all of us do web-browsing, exchange emails, share files.

The important point to note is that these applications we care about do not operate solely by themselves. There are numerous services that all work together to provide a seamless view that an application is doing everything by itself.

— In this class we expose some of the seams. The services that enable such applications are all part of the Internet. So we will spend time talking about how these services work?
Intra-domain Routing

Formulation

- Intra-domain routing \(\Rightarrow\) ~ 100 routers

- Given:
  - **Graph**: where nodes are routers and edges are links
  - **Cost**: associated with each link

- Find:
  - **Lowest-cost path** between any two nodes

- Requirements:
  - Self-healing, traffic-sensitive, scalable

Each node:

- Constructs a vector of distances to all other nodes
- Distance vector
- Distributes to immediate neighbors

Neighbors use the distributed information to update their distance vectors.

This distributed exchange-update-exchange should lead to globally consistent distance vectors (and routing tables).

**Distance-vector Routing**

**Basic Idea**

- Each node:
  - Constructs a vector of distances to all other nodes
    - Distance vector
  - Distributes to immediate neighbors

- Neighbors use the distributed information to update their distance vectors

- This distributed exchange-update-exchange should lead to globally consistent distance vectors (and routing tables)
Distance Vector Routing

Distance table data structure

- Each node has its own table with a...
  - Row for each possible destination
  - Column for each directly-attached adjacent node (neighbor)
- Each table entry gives cost to reach destination via that adjacent node
  - Distance = Cost

\[ D^t(Y,Z) = \text{distance from } X \text{ to } Y \text{ via } Z \text{ as first hop} \]
\[ = c(X,Z) + \min_w \{D^t(Y,w)\} \]
\[ w = \{\text{neighbors of } Z\} \]

Distance Vector Routing

Distance table example

<table>
<thead>
<tr>
<th></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>

Distance Vector Routing

Distance table example

<table>
<thead>
<tr>
<th></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>

\[ D^t(C,D) = c(E,D) + \min_w \{D^t(C,w)\} \]
\[ = 2 + 2 = 4 \]

\[ D^t(A,D) = c(E,D) + \min_w \{D^t(A,w)\} \]
\[ = 2 + 3 = 5 \]

\[ D^t(A,B) = c(E,B) + \min_w \{D^t(A,w)\} \]
\[ = 8 + 6 = 14 \]

A loop?!

Loop!
Distance Vector Routing

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

Distance Table Routing 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.

Note: 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 Algorithm**

**Example**

Time [comp move] → 1

\[
D^0(Y, Z) = c(X, Z) + \min_w \{D^2(Y, w)\} = 7 + 1 = 8
\]

\[
D^0(Z, Y) = c(X, Y) + \min_w \{D^2(Z, w)\} = 2 + 1 = 3
\]

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.)
Distance Vector Algorithm

**Link cost changes**

- When a node detects a local link cost change:
  - The nodes updates its distance table
  - If the least cost path changes, the node notifies its neighbors

"Good news travels fast"

<table>
<thead>
<tr>
<th>Distance Table</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D^i )</td>
<td>( t_0 )</td>
</tr>
<tr>
<td>( X ) ( Y )</td>
<td>( t_1 )</td>
</tr>
<tr>
<td>( X ) ( Z )</td>
<td>( t_2 )</td>
</tr>
<tr>
<td>( X ) ( Y )</td>
<td>( t_3 )</td>
</tr>
</tbody>
</table>

**How long does convergence take?**

- Good news travels fast, but…
- "Bad news" travels slow!
- The "count to infinity" problem

Routing Loop!

Does it Terminate?

- 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 and forth until the cost of the route through X becomes the minimum cost route.
- In the meantime:
  - Buffers may overflow,
  - \( \text{ttl} \)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.
The Count to Infinity Problem
The “poisoned reverse” technique

- If Z routes through Y to get to X:
  » Then Z tells Y that Z’s distance to X is infinite

Initialization…

The Count to Infinity Problem
The “poisoned reverse” technique

- If Z routes through Y to get to X:
  » Then Z tells Y that Z’s distance to X is infinite

- (Will this completely solve the problem?)
  » How about delaying advertising alternate routes after link failures?

Poisoned reverse only helps in the case where routing loops involve only two nodes.

- 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 thorough 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 it’s route to X (to avoid a loop with Z).

Algorithm terminates