Routing:

Routing is the process by which the forwarding table is built and maintained. The forwarding table contains information about the next hop for each destination.

Forwarding:

Forwarding is the process of moving packets from input to output based on the destination IP address and the incoming interface.

Forwarding V.S. Routing

Forwarding involves looking up the destination IP address in the forwarding table. If there is a match, the packet is forwarded to the next hop. Routing involves finding the lowest cost path between two nodes using routing algorithms.

Factors Affecting Routing

Factors affecting routing include:

- Topology
- Load
- Policy
- Cost
- Congestion
- Bandwidth
- Queueing delay
- Congestion control
- Resource allocation

Routing Algorithms

Routing algorithms are used to calculate the lowest cost path between two nodes. There are two main approaches:

- Link State Protocols (LS): You tell everyone about your neighbors.
- Distance-Vector Protocols (DV): You tell your neighbors what you know about everyone.

Two Main Approaches

Route Discovery Protocols:

- Link State Protocols (LS)
- Distance-Vector Protocols (DV)

Forwarding Examples

To forward unicast packets, a router uses the longest matching prefix in the forwarding table. To forward multicast packets, the router uses the longest and exact match algorithms.

A Router and Its Components

The components of a router include the processor, line cards, and interconnect (backplane, crossbar, etc.).

Factors Affecting Routing

Factors affecting routing include:

- Topology
- Load
- Policy
- Cost
- Congestion
- Bandwidth
- Queueing delay
- Congestion control
- Resource allocation
- Cost

Routing Algorithms

Routing algorithms are used to calculate the lowest cost path between two nodes. There are two main approaches:

- Link State Protocols (LS)
- Distance-Vector Protocols (DV)

Two Main Approaches

Route Discovery Protocols:

- Link State Protocols (LS)
- Distance-Vector Protocols (DV)
Distance Vector Protocols

Distributed Bellman-Ford Algorithm

E Receives D’s Routes

Distributed next hop computation

Distance Vector

Distributed Bellman-Ford Algorithm

send step:

upon receiving vectors from each of its neighbors, router
computes its own
distance to each neighbor

receive step:
each router starts with a vector of distances to all directly
attached networks
each router exchanges this current vector to all

Start Conditions:

then, for every network X, router finds that neighbor who
is closer to X than to any other neighbor
determine the cost of X to each neighbor
then, for every network X, router finds that neighbor

where w is a direct neighbor of Z


c(x,z) = c(x',z) + d_{x'}{z'}
d_{x,z}

Example - Initial Distances

Info at node

A B C D E
0 7 ~ ~ 1
7 0 1 ~ 8
~ 1 0 2 ~
~ ~ 2 0 2
1 8 ~ 2 0 1

Example - Initial Distances

Info at node

A B C D E
0 7 ~ ~ 1
7 0 1 ~ 8
~ 1 0 2 ~
~ ~ 2 0 2
1 8 ~ 2 0 1

c(D,E) = 2

Example - Initial Distances

Info at node

A B C D E
0 7 ~ ~ 1
7 0 1 ~ 8
~ 1 0 2 ~
~ ~ 2 0 2
1 8 ~ 2 0 1

E Receives D’s Routes

Distributed Bellman-Ford Algorithm

vector of distances to destinations

until all information exchanges

asynchronous, iterative

adapt the

Distributed next hop computation

employed in the area protocol

Distance Vector Protocols
A Receives B's Routes

A Updates Cost to C

A Receives E's Routes

A Updates Cost to C & D

Final Distances
The Bouncing Effect (a.k.a. Count to Infinity Problem)

Final Distances After Link Failure (Cont...)
This is known as the "count to infinity" problem.

### Observation 1:
- B sends routes to C
- C updates distance to A
- B sends routes to C

### Observation 2:
- But the implicit path from C to A includes itself.
- Adversely affects convergence

**How Are These Loops Caused?**

**Solution 1: Hold downs**

C eventually thinks B's route is gone, picks its own route.

B then selects C as next hop. C eventually thinks B's route is gone, picks the own route. If metric increases, delay propagating information.

In our example, B's metric increases.

**Solution 2:**

C picks B as next hop to A. C updates distance to A.

C eventually thinks B's route is gone, picks its own route.

B then selects C as next hop.
B does not advertise route to C
works for two node loops
Split horizon
other solutions
Poisoned reverse
does not work for loops with more nodes
When link breaks C makes B unreachable
B reports B's cost 2 to A, who reports new cost 4 and reports that to B. C thinks B is reachable through A, so reports C's cost 5 to A. A reports new cost 6 to B. B reports new cost 7 to C. C reports cost 8...

Example where split horizon fails

Distance Vector in Practice

Computing implicit paths

Avoiding the bouncing effect

One way of doing this:

Propagation of path information may be stale
only way to fix this
BGP does it this way
If a router sees itself in path, it rejects the path

Not transient loops are still possible
Does bouncing effect avoid loops?

Loop freedom at every instant

Why? Because implicit path information may be stale
why not transient loops?
Ensure that you have up-to-date information by explicitly
propagating path information
Split horizon/poison reverse
propagate for each destination not only the cost but also its predecessor
To reduce the space requirements

Avoiding the bouncing effect

Computing implicit paths

Distance Vector in Practice

Example where split horizon fails

Other solutions

Distance Vector in Practice

Example where split horizon fails

Other solutions

Distance Vector in Practice

Example where split horizon fails

Other solutions

Distance Vector in Practice

Example where split horizon fails

Other solutions
**Step 1:**
Each node broadcasts its state to all other nodes.
Each node assumes to know the state of links to its neighbors.

**Problem:** Sequence number may wrap around.

**Solution:**
- Treat space as circular, continue after wrap around.
- If LSP is the most recent LSP, it is saved in the database, and a copy is sent on all links.
- When node receives LSP from node:
  - Otherwise, discard LSP.

**Sequence Number Space Issues**
Problem: sequence number may wrap around.
Solution: treat space as circular, continue after wrap around:
- $A < B$ and $A-B < N/2$,
- $A < B$ and $B-A < N/2$.

**Reliable Flooding**
- Node outputs LSP on all links.
- Time to live (TTL), sequence number, list of neighbors, and link cost.
- Node ID.
- Periodically, each node creates a Link State Packet (LSP).

**Link State Packets (LSPs)**
- Node outputs LSP on all links.
- Time to live (TTL), sequence number, list of neighbors and link cost.
- Node ID.
- Periodically, each node creates a Link State Packet.

**Building Blocks**
- Dijkstra's Shortest Path Tree (SPT) Algorithm.
- Sequence number issues.
- Reliable broadcast mechanism.
- Flooding.

**Link State Algorithms**
- Other nodes from global state.
- Each node assumes to know the state of links to its neighbors.
- Each node broadcasts its state to all neighbors.
- Each node computes shortest path to all neighbors.

**Basic Steps**
- Each node broadcasts its state to all neighbors.
- Each node computes shortest path to all neighbors.
- Each node outputs LSP on all links.
- Time to live (TTL), sequence number, list of neighbors, and link cost.
- Node ID.
- Periodically, each node creates a Link State Packet.
A failed router and comes up but does not remember the last sequence number it used before it crashed.

**Problem:** Router Failure

**Computer Communications - CSCI 551**

Copyright © William C. Cheng

New LSPs may be ignored if they have lower sequence number. The LSP is re-flooded once TTL = 0. Nodes periodically decrement the age (TTL) of stored LSPs.

**One Solution:** LSP Aging

**Computer Communications - CSCI 551**

Copyright © William C. Cheng

LSPs expire when TTL reaches 0. Rebooted router waits until all LSPs have expired. Trade-off between frequency of LSPs and router wait after reboot.

**Is Aging Still Needed?**

Aging ensures that old state is eventually flushed out of the network. New rule: Newly rebooted router discovers its seq num before it crash.

**Lollipop Operation**

Divide sequence space \( N \) into 3 spaces:
- **Negative space:** \( -N/2 - 0 \)
- **Positive space:** \( 0 \) to \( N/2 - 1 \)
- **Lollipop sequence space**

A Better Solution

The number 0.

- Positive space: \( 0 \) to \( N/2 - 1 \)
- Negative space: \( N/2 \) to \( -N/2 \)
- Lollipop sequence space

Aging ensures that old state is eventually flushed out of the network. New rule: Newly rebooted router always stays with oldest seq num (-N/2).

**Lollipop Operation (Cont...)**

Newly booted router discovers its seq num before it crash.

**Is Aging Still Needed?**

Newly booted router discovers its seq num before it crash.

**Aging**

When seq number becomes positive, wrap round as before.

- If \( a > 0 \) and \( 0 < b < a \), \( a < b \) and \( b < N/2 \).
- If \( a > 0 \) and \( b > a \), \( a > b \) and \( a - b > N/4 \).

Router comes up and starts with \( N/2 \). Then \( N/2 + 1 \), \( N/2 + 2 \), etc.

- If \( a < 0 \) and \( b < 0 \).

If router \( R1 \) gets older LSP from router \( R2 \), \( R1 \) informs \( R2 \) of the sequence number in \( R1 \)'s LSP.

- Suppose \( R1 \) is new, then \( R1 \) sends an LSP and \( R1 \) comes up.

**Babier Solution**

When seq number becomes positive, wrap round as before.

- If \( a > 0 \) and \( 0 < b < a \), \( a < b \) and \( b < N/2 \).
- If \( a > 0 \) and \( b > a \), \( a > b \) and \( a - b > N/4 \).

Aging ensures that old state is eventually flushed out of the network.

**Problem:** Router Failure

**Is Aging Still Needed?**

Newly booted router discovers its seq num before it crash.

**Aging**

Number.

- New LSP may be ignored if they have lower sequence number. Repeat number if received before it.

A failing router and comes up does not remember the last LSPs.
SPT = \{a\}

**Example**

<table>
<thead>
<tr>
<th>Step</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>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Dijkstra's Algorithm**

1. Initialize all nodes with infinite distance except the source node.
2. Set the distance to the source node as 0.
3. For each node in the graph:
   - If the node is not in the SPT and its distance is finite, find the adjacent node with the minimum distance from the SPT and set the distance to the new node.
   - For all adjacent nodes to the new node, update the distance if it is smaller than the current distance.
4. Repeat step 3 until all nodes are in the SPT.

**Example**

- Initial distances: D(b) = 2, D(c) = 3, D(d) = 4, D(e) = 3, D(f) = 1
- After step 1: D(b) = 2, D(c) = 3, D(d) = 4, D(e) = 3, D(f) = 1
- After step 2: D(b) = 2, D(c) = 3, D(d) = 4, D(e) = 3, D(f) = 1
- After step 3: D(b) = 2, D(c) = 3, D(d) = 4, D(e) = 3, D(f) = 1
- After step 4: D(b) = 2, D(c) = 3, D(d) = 4, D(e) = 3, D(f) = 1

**SPT Algorithm**

- Find the minimum distance node not in the SPT.
- Add the node to the SPT and update the distances of its adjacent nodes.
- Repeat until all nodes are in the SPT.
Example

Flooding:

1) Use Dijkstra's shortest path algorithm to compute distances to all destinations
2) Install <destination, nexthop> pairs in forwarding table
3) Re-distribute LSA to all neighbors

Limited by Dijkstra computation overhead, space requirements
With consistent LSDBs, all nodes compute consistent loop-free paths

In LS, nodes must compute consistent routes independently
In DV, routes are computed relative to other nodes

Limited by Dijkstra computation overhead, space requirements
With consistent LSDBs, all nodes compute consistent loop-free paths

LS: fast
Convergence speed:
DV: fast with triggered updates

LS: small
Msg size:
DV: potentially large

LS can broadcast incorrect/corrupted LSP
Localized problem

DV can advertise incorrect paths to all destinations
incorrect calculation can spread to entire network
must protect against LSDB corruption

In LS, nodes must compute consistent routes independently
In DV, routes are computed relative to other nodes

Packet from C→A may loop around BDC
What Makes Routing Hard?

Routing and business/policy issues
- Distributed computation (hard to debug)
- No clear winners
- Convergence time
- Corrupted host can get all routes

LS v.s. DV (Cont...)
- Reliability and robustness
- Scalability to many nodes
- Dealing with changes
- Routing and aggregation
- Some changes (this goes down) should be dealt with ASAP

LS risks:
- Bottom line: no clear winner, but we see more frequent use of LS in the Internet

Scalability to Big Networks
- Approaches
  - Key idea: want to know about some networks far away
  - Two approaches:
    - Area hierarchy
    - Semi-manual aggregation

Internet today is on the order of 1 million networks
Example Area Hierarchy

Routing table at 11.2.1:
- Networks (11.2.1/24, etc.)
- Sub-AS's (11.1, etc.)
- AS's (11, 12, 13)

In this area hierarchy, is this the only way?