Source Routing

Idea: Include the entire route to be followed in the header
Nodes must know the network topology in advance

Possible Header Implementations

Header entering switch
- D C B A
- D C B A
- Ptr D C B A

Header leaving switch
- A D C B
- D C B
- Ptr D C B A

(a) (b) (c)

Header has variable length
Used in both datagram and VC switching (connection setup)
May be “loose” or “strict” routing
Bridges

Bridges are **special switches** that interconnect Ethernet networks. They act as **simple nodes** on each Ethernet. Early bridges forwarded all packets between the two networks.

![Diagram of a bridge connecting three networks](image)

**Bridges Vs. Repeaters**

Repeaters are **analog amplifiers** of the signal:
- Monitor the existence of analog signal on the line
- Amplify and repeat the signal
- Physical layer function, no message parsing

Bridges "repeat" frames rather than signals:
- They forward received frames among different LANs
- Message parsing is performed to decide whether to fwd the packet or not

**Limitations of Repeaters**

All hosts connected via repeaters belong to the same collision domain:
- Every bit is sent everywhere
- So, aggregate throughput is limited
  - E.g., three departments each get 10 Mbps independently
    - ... and then connect via a hub and must share 10 Mbps

Multiple LAN technologies not compatible:
- No message parsing – analog process
- No interoperability between different rates and formats
  - E.g., 10 Mbps Ethernet and 100 Mbps Ethernet

Limited geographical distances and # of nodes
Advantages of Bridges

Facilitate breaking of network into isolated collision domains
Message parsing may avoid unnecessary packet forwarding

Collision domain 1

Collision domain 2

Bridge Forwarding Table

Not all packets need to be forwarded. E.g., packets from A to B need not be forwarded on port 2
Bridges maintain a forwarding table

<table>
<thead>
<tr>
<th>Host</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>Z</td>
<td>2</td>
</tr>
</tbody>
</table>

Forwarding Algorithm

- Look up Destination Address
- Destination not found
  - broadcast
  - Destination found
    - Same as incoming port
    - Different port
  - Forward designated port
  - Discard
  - Flood all ports but incoming one
Problem with Flooding

Some Preliminaries on Graphs

A graph is said to be connected, if there is a path from any node to any other node.
A graph is \( k \)-connected if the degree of each vertex is at least \( k \). Alternatively, if removal of any \( k-1 \) edges does not leave the graph disconnected.

A graph of \( N \) vertices is fully connected if each node has a degree \( (N-1) \) (directly connected network).

Number of edges: \( \frac{N(N-1)}{2} \)

(A) Cyclic Graphs

- **Cycle**: a path with the same first and last vertex
- **Cyclic graph**: A graph that contains at least one cycle
- **Acyclic graph**: A graph with no cycles
- **Tree**: An acyclic connected graph (spanning tree, spans all vertices)
- **Forest**: A disconnected acyclic graph (a graph with many trees)

Mapping Extended LAN to a Graph

Bridges and LANs become vertices, ports become edges.

Goal: Make a tree that spans only LANs (red vertices)
Spanning Tree Algorithm (1)
Each bridge has a unique identifier
Pick bridge with smallest ID, make it the root of the tree

Spanning Tree Algorithm (2)
Compute shortest path (in terms of hops) from each bridge to root

Spanning Tree Algorithm (3)
Each LAN remains connected to the bridge with shortest path to the root. In case of a tie use smallest ID
Message Exchange
Initially, all bridges consider themselves as roots
Broadcast on all ports ID, root bridge, and distance to root
If bridge hears a message from another bridge with smaller ID it adjusts root/distance, and forwards
Ex., Let (Y, d, X) denote (root, distance, bridge ID)
B3 receives (B2, 0, B2)
B3 accepts B2 as root
B3 adds one on its distance to B2, and fwd to B5 (B2, 1, B3)
B2 accepts B3 as root and updates B3 with (B1, 1, B2)
B5 accepts B1 as root and updates B3 with (B1, 1, B3)
B3 accepts B1 as root, and note that both B2, B5 are closer to root in both ports. Hence, B3 blocks all ports