6. Routing

6.1 Introduction

LANs:
- Bus or ring topology
- Each packet is sent to all terminals.
- By evaluation of destination address: Destination terminal reads packet, other terminals ignore it.
- No routing function needed; unique addressing scheme is sufficient.

LANs interconnected by bridges:
- *Stupid* bridge (promiscuous mode): Copy all packets from on LAN to the other respecting the (possibly different) medium access schemes.
- *Self-learning* bridge: Observe traffic in both LANs, to learn addresses of terminals located on either side. Copy packets only to other LAN, if destination terminal located there or if its location is unknown.

Tasks in routing:
1) Gathering information on the network (terminals, switches, and their interconnections).
2) Search a route for a connection or a datagram.
3) Controlling the switching matrix; i.e. execution of the packet forwarding from an incoming line to the correct outgoing line (to the next node towards the packet’s destination).

Examples:
1) Information about network (graph)
2) Routes to other nodes: Routes starting from node A
3) Routing tables for control of switches:

<table>
<thead>
<tr>
<th>Switch of node A:</th>
<th>Destination</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next node</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>E</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Switch of node B:</th>
<th>Destination</th>
<th>A</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next node</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>A</td>
<td>D</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Switch of node C:</th>
<th>Destination</th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next node</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>G</td>
<td></td>
</tr>
</tbody>
</table>

Basically: How does a switch work?

When packets on several incoming lines shall be delivered to the same outgoing line at the same time: **Competition!**
- Buffer needed to delay some of the packets.
- Where to place the buffer (incoming line, crossing point, outgoing line)?
- Minimize number of buffer places, because memory chips consume electric energy, produce heat, require space for cooling.
- Large number of buffer places to avoid buffer overflows (packet loss).
Chapter 6: Routing

- Maximize throughput, avoid blocking
- Minimize packet delay

Classification of routing algorithms:
- Datagram (packet switching) versus connection-oriented switching
- Static versus adaptive
  - Static: Routes between switches/terminals are determined once (possibly at network setup).
  - Adaptive: New routes are determined frequently or on demand, taking into account the actual traffic load, delay, and buffer queue length on all individual links.
- Centralized versus distributed algorithm
  - Centralized: Bellmann-Ford algorithm, Dijkstra algorithm
  - Distributed: Ford-Fullerson algorithm, distributed Bellmann-Ford algorithm
- Support of p2p, multicast/p2mp, broadcast connections
  - Multicast /broadcast can be useful for distributing network status information in adaptive routing.
- Address resolution methods, network topology: Routing tables with entries for all existing terminals are impossible to handle in large networks (e.g. telephone networks, Internet)
  - Hierarchical network structure
  - Name server (IP: Domain Name Server, UMTS/GSM Home Location Register, Visiting Location Register)

6.2 Examples

Routing in telephone networks:
- Connection-oriented
- Hierarchical:
  - country code  +49
  - city code    731
  - company     505
  - phone       4816
- Static:
  - Routing tables are set up at installation
  - Constant bandwidth
  - Fixed number of connections per physical link
  - Variation of traffic load is not an issue
- Redundancy concept:
  - Switch components are doubled
  - Pre-configured alternative routes
Chapter 6: Routing

Routing in computer networks:
- Datagram switching
- Partly hierarchical:
  - Backbone, WAN, interconnection of routers/gateways
  - subnets
  - LAN
- Highly dynamic traffic load (adaptive routing)
- Decentralized algorithms

6.3 Basics from Graph Theory

Graph, Directed Graph (Digraph):

\[ N : \text{Set of nodes} \]
\[ E : \text{Set of edges} \quad E \subseteq N \times N \]
Edge \( e = (m,n) \)
- Node \( m \) is connected to node \( n \) (in graph),
- There is a link from node \( m \) to node \( n \) (in digraph)

\( d(e) \): Weight of an edge \( e \in E \) (can correspond to delay, traffic load, buffer queue length,…)

Example:

\[
\begin{array}{cccccc}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} \\
\text{A} & 2 & 1 & 2 & \\
\text{B} & 1 & 1 & & \\
\text{C} & 3 & & & \\
\text{D} & 1 & & & \\
\text{E} & 2 & & & \\
\end{array}
\]

\[ N = \{ABC,DE\} \]
\[ E = \{(AD),(AE),(AC),(BA),(BC),(CA),(DA),(EB)\} \]
\[ d(E) = \{1,2,2,1,3,1,2\} \]
Chapter 6: Routing

Incidence matrix:

<table>
<thead>
<tr>
<th></th>
<th>(A,D)</th>
<th>(B,C)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Path: Sequence of edges such that:
- Graph: any two successive edges have one node in common
- Digraph: ending node of any edge (except the last) is starting node of next edge.

Cycle: Path, whose starting node of the first edge equals the ending node of the last edge, and all the edges are distinct.

Example:

![Graph with nodes A, B, C, D, E and edges labeled with weights 1, 2, 3]

Path: ((A, E), (E, B))
Cycle: ((A, C), (C, B), (B, A))

Hamiltonian cycle: Contains all nodes of the graph.

Tree: Graph without cycles and exactly one path between any pair of nodes. \(|N| = |E_{tree}| + 1\)

Complete tree: Contains all nodes of the graph.

Minimal tree: Complete tree, with minimum sum of edge weights over all complete trees (There can exist more than one minimal tree in a given graph).

Source tree

Sink tree


6.4 Routing algorithms

6.4.1 Flooding

Principle: Forward a packet to all neighboring nodes except the node which sent the packet.

Advantages:
- All nodes are reached (broadcast).
- All possible paths are used in parallel, including the shortest path from source to any destination.
- Any node needs to know only his incoming and outgoing lines, but nothing else about the network.

Drawback: High network load!

Problems:
- Cycles: Packet will be forwarded in cycle forever.
- Same packet received on several lines – forward each of them?

Counter measures:
- Hop counter: Packet is deleted after certain number of hops. Requires appropriate counter setting.
- Packet carries source node’s address and a sequence number. Switching node can recognize already known packets and discard them.

Applications:
- Distribution of network status info for other, more elaborate routing algorithms (connectivity, traffic load, link failures,…)
- For concurrent update of data bases in all network nodes.
- In military networks with extreme robustness demands.

6.4.2 Complete tree and Hamiltonian cycle

Principle:
- Determine a complete source tree (with source node as root), or a Hamiltonian cycle from the graph of the network.
- Forward the packet along the tree or the cycle (preferably in both directions).
Example:

Complete tree

![Complete tree diagram]

Hamiltonian cycle

![Hamiltonian cycle diagram]

Advantages:
- All nodes reached (broadcast)
- Significantly lower network load than with Flooding.
- No problems with cycles.

Drawbacks:
- Network graph must be known (contrary to Flooding).
- New complete tree/Hamiltonian cycle must be determined, when network topology changes.
- All nodes must know the complete tree/Hamiltonian cycle.

### 6.4.3 Dijkstra's algorithm

Basis for routing algorithm of ARPANET since 1979.

Shortest path algorithm: Determines shortest path from source node A to all other nodes.

**Length \( \Lambda \) of a path**: sum of the edge weights of all edges in the path

\[
\Lambda = \sum_{e \in \text{path}} d(e)
\]
Chapter 6: Routing

Preliminaries:
- Define to sets
  - \( M = \{ \text{all nodes to which shortest path from A is known} \} \)
  - \( N = \{ \text{all other nodes} \} \)
- Set \( \Lambda_A = 0 \)
- For all other nodes \( y \in N \Rightarrow \Lambda_y = \infty \)
- \( d((X, Y)) > 0 \) for all edges between nodes \( X, Y \).
- \( d((X, Y)) = \infty \) if no edge exists between \( X, Y \).

Algorithm:
- For all nodes \( Y \in N: \Lambda_Y := \min_{X \in M} \{ \Lambda_X + d(X, Y), \Lambda_Y \} \)
- Find the node \( Y' \) such that \( \Lambda_{Y'} := \min_{Y \in N} \{ \Lambda_Y \} \)
- \( M := M \cup Y' \) and \( N := N / Y' \)
- Repeat until \( N = \{ \} \)

Note: \( \Lambda_X \leq \Lambda_Y \) for \( X \in M \) and \( Y \in N \).

Example:

```
<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1(A)</td>
<td>\infty</td>
<td>4(A)</td>
<td>3(A)</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>2</td>
<td>3(B)</td>
<td>3(B)</td>
<td>3(A)</td>
<td>\infty</td>
<td>\infty</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>5(D)</td>
<td>6(D)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>6(D)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(-)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3(B)</td>
<td>3(B)</td>
<td>3(A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5(D)</td>
<td>6(D)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6(D)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
6.4.4 Bellman-Ford algorithm

Basis for the first routing algorithm of ARPANET

Preliminary: \(d((X,Y)) > 0, \quad d((X,Y)) = \infty\) if no edge exists between node X and Y

Algorithm:

Initialization: \(\Lambda_x := \infty\) for all nodes except starting node A; \(M_0 := \{A\}\)

Iteration 1: \(M_1 := \{\text{all nodes which are 1 hop away from A}\}\)

For all \(X \in M_1\):

\[\Lambda_x := \min_{Y \in M_0} \{\Lambda_y + d(X,Y), \Lambda_x\}\]

i-th Iteration: \(M_i := \{\text{all nodes which are i hops away from A}\}\)

For all \(X \in M_i\):

\[\Lambda_x := \min_{Y \in M_{i-1}} \{\Lambda_y + d(X,Y), \Lambda_x\}\]

-> iterate over all hops

Example:

```
Start: \(M_0 := \{A\}, \quad \Lambda = (0)\)

Iteration 1: \(M_1 := \{B, D, E\}, \quad \Lambda = (1(A),4(A),3(A))\)

Iteration 2: \(M_2 := \{D, C, F, G\}, \quad \Lambda = (3(B),3(B),6(D),7(D))\)

Iteration 3: \(M_3 := \{E, F, G\}, \quad \Lambda = (3(A),5(D),6(D))\)

Iteration 4: \(M_4 := \{E, F\}, \quad \Lambda = (3(A),5(D))\)

Iteration 5: \(M_5 := \{E\}, \quad \Lambda = (3(A))\)
```