Lecture 7 Overview

- Connecting links
  - Forwarding
  - Collision domains

- Bridging
  - Store and forward
  - Learning bridges
Often called a **Local Area Network (LAN)**
- Link is multiplexed across time
- Frames have a link-layer header with addresses of source and destination of hosts on the LAN

One shared link (a **bus**) limits scale in terms of:
- Distance (e.g., 2500 m for Ethernet)
- Performance (Capacity shared across all nodes)

A better alternative is to have multiple busses
- Each bus is of a limited size, scale, number of hosts, etc.

Need the ability to connect multiple busses together
- In other words move frames from one wire to another
Hubs/Repeaters

- Physical layer device
  - One “port” for each LAN
  - Repeat received bits on one port out all other ports
Hub Advantages

- Hubs can be arranged into hierarchies
  - Ethernet: up to four hubs between any pair of nodes

- Most of LAN continues to operate if “leaf” hub dies

- Simple, cheap

CSE 123 – Lecture 7: Bridging
Still One Big Bus

- Single collision domain
  - No improvement in max throughput
  - Average throughput < as # of nodes increases
  - Why?

- Still limited in distance and number of hosts
  - Collision detection requirements
  - Synchronization requirements

- Requires performance homogeneity
  - Can’t connect 10 Mbps and 100 Mbps networks
● **Store and forward** device
  - Data-link layer device
  - Buffers entire packet and *then* rebroadcasts it on other ports

● Creates *separate* collision domains
  - Uses link-layer MAC protocol for access to each LAN (i.e., acts like a host on that LAN)
  - Can accommodate different speed interfaces (issues?)
  - Separate CDs improves throughput (why?)

● Can significantly improve performance
  - Not all frames go everywhere. (Why did they with a hub?)
Selective Forwarding

- Only rebroadcast a frame to the LAN where its destination resides
  - If A sends packet to X, then bridge must forward frame
  - If A sends packet to B, then bridge shouldn’t
Need to know “destination” of frame
- Destination address in frame header (48bit in Ethernet)

Need know which destinations are on which LANs
- One approach: statically configured by hand
  » Table, mapping address to output port (i.e. LAN)
- But we’d prefer something automatic and dynamic...

Simple algorithm:
Receive frame \( f \) on port \( q \)
Lookup \( f\.dest \) for output port /* know where to send it? */
If \( f\.dest \) found
then if output port is \( q \) then drop /* already delivered */
  else forward \( f \) on output port;
else flood \( f \);
/* forward on all ports but the one where frame arrived*/
Learning Bridges

- Eliminate manual configuration by learning which addresses are on which LANs

- Basic approach
  - If a frame arrives on a port, then associate its source address with that port
  - As each host transmits, the table becomes accurate

- What if a node moves? Table aging
  - Associate a timestamp with each table entry
  - Refresh timestamp for each new packet with same source
  - If entry gets too stale, remove it

<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>D</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>2</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>Y</td>
<td>3</td>
</tr>
<tr>
<td>Z</td>
<td>2</td>
</tr>
</tbody>
</table>
Suppose $C$ sends frame to $D$ and $D$ replies back with frame to $C$

- $C$ sends frame, bridge has no info about $D$, so floods to both LANs
  - bridge notes that $C$ is on port 1
  - frame ignored on upper LAN
  - frame received by $D$
Learning Example

- D generates reply to C, sends
  - bridge sees frame from D
  - bridge notes that D is on port 2
  - bridge knows C on port 1, so **selectively** forwards frame via port 1

CSE 123 – Lecture 7: Bridging
Each bridge keeps a list mapping link-layer destination address to port number

- (what are the directions to this destination?)

This list is populated by looking at the source address of each packet it receives on a given port and entering those values in the table

- (if a packet from A came from port x, then packets to A should be sent on part x)

If a packet arrives with a destination address not in the table, then send on all ports (except the one it came on)

Simple, automatic, self healing
Network Topology

- Linear organization
  - Inter-bridge hubs (e.g. CS) are single points of failure
  - Unnecessary transit (e.g. EE<->SE must traverse CS)

- Backbone/tree
  - Can survive LAN failure
  - Manages all inter-LAN communication
  - Requires more ports
Learning works well in tree topologies

But trees are fragile
- Net admins like redundant/backup paths

How to handle Cycles?
- Where should $B1$ forward packets destined for LAN $A$?
Spanning Tree

- Spanning tree uses *subset* of bridges so there are no cycles
  - Prune some ports
  - Only **one** tree

- Q: How do we find a spanning tree?
  - Automatically!
  - Elect root, find paths
For next time…

- Read P&D 3.2 – 3.2.4