Lecture 7 Overview

- Connecting links
  - Forwarding
  - Collision domains

- Bridging
  - Store and forward
  - Learning bridges
One link is not enough

- 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
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

If a host in EE sends a frame, who receives it?
A. Hosts in EE
B. Hosts in CS
C. Hosts in SE
D. All of the above
Still One Big Bus

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

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

- Requires performance homogeneity
  - Can’t connect 1 Gbps and 100 Gbps networks
Bridges

- **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 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?)

CSE 123 – Lecture 7: Bridging
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

LAN 1
- A
- B
- C
- D

LAN 2
- W
- X
- Y
- Z

Who should receive a frame sent from A to Z?
A. Only Z
B. W, X, Y, and Z
C. B, C, D, and Z
D. B, C, D, W, X, Y, and Z
Forwarding Tables

- 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 \( f.\text{dest} \) for output port /* know where to send it? */
  If \( f.\text{dest} \) 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>
Learning Example

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 $D$
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
Learning Bridges Recap

- Each bridge keeps a list mapping link-layer destination address to port number.

- 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 port x)

- If a frame arrives with a destination address not in the table, then send on all ports (except the one it came on).
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
An Issue: Cycles

- 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

How will a frame get from A to J?

A. B3, B2, B1, B4  
B. B5, B1, B7  
C. B5, B1, B4  
D. I don’t know

CSE 123 – Lecture 7: Bridging
Spanning Tree Algorithm

- Each bridge sends periodic configuration messages
  - (RootID, Distance to Root, BridgeID)
  - All nodes think they are root initially

- Each bridge updates route/Root upon receipt
  - Smaller root address is better
  - Select port with lowest cost to root as “root port”
    - To break ties, bridge with smaller address is better

- Rebroadcast new config to ports for which we’re “best”
  - Don’t bother sending config to LANs with better options
  - Add 1 to distance, send on ports that haven’t told us about a shorter path to the root

- Only forward packets on ports for which we’re on the shortest path to root
  (prunes edges to form tree)
Spanning Tree Example

- Sample messages to and from B3:
  1. B3 sends (B3, 0, B3) to B2 and B5
  2. B3 receives (B2, 0, B2) and (B5, 0, B5) and accepts B2 as root
  3. B3 sends (B2, 1, B3) to B5
  4. B3 receives (B1, 1, B2) and (B1, 1, B5) and accepts B1 as root
  5. B3 wants to send (B1, 2, B3 ) but doesn’t as its nowhere “best”
  6. B3 receives (B1, 1, B2) and (B1, 1, B5) again and again…
For next time…

- Keep reading 3 – 3.1