Project Status

- Project 1a is over
  - Over ¼ of the class received perfect scores!!!
  - Unfortunately a non-trivial number do not pass any of the autograder tests…
  - We will release grades in the next 24-48 hours
  - There will be a mechanism to fix minor bugs
  - DON’T DROP IF YOU SUBMITTED; come see me to discuss if you’re struggling

- Project 1b will come out later today
  - As before, we will go over it in discussion WEDNESDAY—make sure to come
  - For your own mental health, read the (entire) specification before coding
Lecture 8 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 \textit{bits} on one port out \textit{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
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 collision domains 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
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.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>2</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
If $D$ generates a reply to $C$, it sends it on its LAN
- bridge sees frame from $D$
- bridge notes that $D$ is on port 2
- bridge knows $C$ on port 1 (from previous slide), so **selectively** forwards frame via port 1
Network Topology

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

- 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
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”
  - Add 1 to distance, send on ports that haven’t told us about a shorter path to the root
  - Don’t bother sending config to LANs with known better options

- Only forward packets on ports for which we’re on the shortest path to root
  (prunes edges to form tree)
For next time...

- Keep reading 3 – 3.1
- Take a look at Project 1b