Lecture 7: Bridging

CSE 123: Computer Networks
Alex C. Snoeren
Lecture 7 Overview

- Connection establishment

- Connecting links
  - Forwarding
  - Collision domains

- Bridging
  - Store and forward
  - Learning bridges
Connection Establishment

- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - Most important: sequence number space in each direction
  - Lots of other parameters: e.g., the window size

- Handshake protocols: setup state between two oblivious endpoints
  - Need to deal with delayed and reordered packets
Two-way handshake?

Active participant (client)  Passive participant (server)

SYN, SequenceNum = x
SYN, SequenceNum = y
+data

What one of the following could cause a problem?
A. The SYN gets lost
B. The ACK gets lost
C. Neither
D. Both

CSE 123 – Lecture 7: Bridging
Two-way handshake?

Active participant (client)

Old SYN, SequenceNum = x

New SYN, SequenceNum = q

SYN, SequenceNum = y

Passive participant (server)

Delayed old SYN

Rejected

CSE 123 – Lecture 7: Bridging
Three-Way Handshake

- Opens both directions for transfer

![Diagram of Three-Way Handshake]

- \( \text{Active participant (client)} \)
  - SYN, SequenceNum = \( x \)
  - SYN + ACK, SequenceNum = \( y \)
  - ACK, Acknowledgment = \( x + 1 \)

- \( \text{Passive participant (server)} \)
  - ACK, Acknowledgment = \( y + 1 \)

CSE 123 – Lecture 7: Bridging
Connection Teardown

- Orderly release by sender and receiver when done
  - Delivers all pending data and “hangs up”

- Cleans up state in sender and receiver

- TCP provides a “symmetric” close
  - Both sides shutdown independently

Why a symmetric close?
A. Need to ensure no data loss
B. Data transfer is bi-directional
C. Both ends need to know when they can “forget” about connection
D. All of the above
TCP Connection Teardown

Web server

FIN

ACK

TIME_WAIT

... 

CLOSED

Web browser

FIN

ACK

CLOSED
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?)
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

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 \( f\.dest \) 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>
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
  - (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 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)
For next time...

- Keep reading 3 – 3.1