Sockets & Physical Layer

CS 640, 2015-01-27

Announcements
● Project #1 released

Outline
● Recap: Life of a Packet
● Sockets
● Network Structure
● Packet Switching
● Multiplexing
● Encoding

Life of a Packet
● Communication abstractions
  ○ Application-to-application -- exchange of requests/responses and data
  ○ Processes-to-process -- reliable communication channel for a stream of bytes; identify specific processes using ports
  ○ Host-to-host -- identify specific hosts using common numeric addressing scheme
● Encapsulation
  ○ Information for each layer is put in a header which is appended to the front of the packet
  ○ Encapsulate as you go down towards physical
  ○ Decapsulate as you go up towards app
● Questions?

Sockets
● Need an interface for applications on end-hosts to set up and send/receive data over end-to-end communication channels
  ○ Berkeley sockets interface -- originally provided by BSD 4.1 in about 1982
  ○ Java interface is slightly different from the C/C++ interface
● Java methods
  ○ Socket()
    ■ Create a new socket
    ■ Socket is similar to a file descriptor
    ■ Uses TCP underneath to provide reliable delivery of a stream of bytes; data is automatically broken into packets by the network stack in the OS
    ■ To send individual packets (unreliably), use DatagramSocket, which uses UDP underneath
  ○ bind(SocketAddress bindpoint)
    ■ Associates a network layer (IP) address and port with the socket
    ■ This is the address and port used on this host
  ○ connect(SocketAddress endpoint)
- Sets up the socket to communicate with a specific remote host
- When using a Socket (i.e., TCP), a handshake occurs to establish the reliable communication channel
- Specify the address and port used on the remote host
- Not called when using a DatagramSocket (i.e., UDP)
  - `accept()`
    - Wait for an incoming connection request
    - Only used with TCP
    - Use with a ServerSocket() rather than just a Socket()
    - When a connection is accepted, a Socket object for that specific connection is returned; can call accept on ServerSocket again to accept another connection from a different host/port
  - `read()` (from InputStream)
    - `receive(DatagramPacket p)`
      - With TCP, read bytes sent by the other side
      - With UDP, receive a packet sent by the other side; since UDP is connectionless, packet could have come from anyone -- call getAddress() and getPort()
  - `write` (to OutputStream)
    - `send(DatagramPacket p)`
      - With TCP, write bytes to send to the other send
      - With UDP, send a packet to the other side; since UDP is connectionless, you need to provide the address and port for the remote host when constructing the packet
  - `close()`
    - Close the socket
    - With TCP, terminate the connection
Example

Client

4. c = new Socket()
5. c.connect(SocketAddress remote)
6. c.getOutputStream().write()
9. c.getInputStream().read()
10. c.close()

Server

1. s = new ServerSocket()
2. s.bind(SocketAddress local)
3. Socket c = s.accept()
7. c.getInputStream().read()
8. c.getOutputStream().write()
11. c.close()
12. s.close()
Network Structure

- Building blocks -- nodes & links
- Point-to-point -- simplest network; two hosts are directly connected; not scalable

- Switched network -- multiple hosts connected to a switch; switches connected to each other

- Network of networks -- multiple switched networks connected to each other by routers
  - Each network (i.e. cloud) is administered by a particular organization
  - View as a hierarchy
    - CS network
    - Campus network contains CS, Math, DoIT
    - WiscNet contains UW-Madison, UW-Milwaukee
    - ...
    - Internet
- Physical structure of a portion of the campus network -- https://stats.net.wisc.edu
Circuit vs. Packet Switched Networks

- Circuit switching
  - **What is circuit switching?**
  - Used by the telephone network
  - Dedicated path between 2 nodes -- path is a sequence of dedicated links
  - Signals establish path (or circuit) between nodes
  - Send stream of bits over established path
  - **Disadvantages**
    - Number of parallel links determines the maximum number of nodes that communicate at one time -- e.g., telephone network is designed for the expected highest volume of calls

- Packet switching
  - Used by computer networks
  - Shared path between 2 nodes -- path is sequence of shared links (last link is dedicated)
  - Communicate with discrete groups of bits -- packets
  - Headers (Ethernet & IP) on each packet have information on source & destination
    - Used by network to forward packet to its destination
    - No need to pre-establish path (note that this is different from establishing a logical communication channel between two processes using TCP)
  - **Advantages**
    - Improved scalability -- do not need dedicated links for each pair of communicating nodes
    - Improved robustness -- as long as there is at least one path, everyone can use this path, although they'll only be able to send a little data because the path is shared
  - **Disadvantages**
    - No guarantee on capacity -- sharing means you may get to send less packets sometimes and more packets at other times
Multiplexing

● **How do we share a link (i.e., a channel) between multiple communicating nodes?**

● Frequency division multiplexing (FDM)
  ○ Divide channel into sub channels using frequency
  ○ Each pair of communicating nodes are assigned a sub channel
  ○ Used by DSL -- telephone calls use one frequency range and data uses a different frequency range

● Time division multiplexing (TDM)
  ○ Divide channel into time slices
  ○ Each pair of communicating nodes are assigned a set of time slices

● Problems
  ○ Frequency or time slice is wasted when communicating hosts do not have any data to send -- e.g., while waiting for server to respond with web page
  ○ Need to know ahead of time that maximum number of communicating hosts so channel can be divided into enough pieces

● Statistical multiplexing
  ○ Divide channel into time slices, like with TDM
  ○ Nodes can transmit on demand, rather than being allocated a set of time slices
  ○ Nodes can send more when other nodes have less to send
  ○ If all nodes always have data to send and we cycle through them in round robin and the link is effectively shared in the same way as TDM
**Encoding**

- **How do we represent bits on the physical wire? -- i.e., What signal should we send for a specific sequence of bits?**
  - Two issues
    - What signal do we use for 0 & 1?
    - How do we know when the signal for one bit ends and another begins?
  - Non-return to zero (NRZ)
    - Low voltage for 0 and high voltage for 1
    - Use a separate signal for clock
      - Disadvantage: require two wires -- one for bit signal and one for clock signal
      - Signal edges are an opportunity to synchronize clocks, but we need enough edges for such synchronization -- this cannot happen if we have long sequences of 0s or 1s
  - Non-return to zero inverted (NRZI)
    - Same voltage for 0 and change voltage for 1
    - No separate clock signal
      - Disadvantage: long sequences of 0s can cause clock to become unsynchronized
  - Manchester encoding
    - Low to high for 0 and high to low for 1
    - Transitions for every bit provide clock signal
      - Disadvantage: requires signal to be sensed at twice the rate
  - 4-bit/5-bit (4B/5B)
    - Use 5-bits to encode every 4-bits of data -- bit sequences chosen so there are lots of 1 to 0 and 0 to 1 transitions
    - Disadvantage: 20% of bits are “wasted”