Sockets for Interprocess Communication

TCP/IP recap

- **IP**:
  
  Every host has a unique IP address (routers use this)

  IP is "best effort" — messages (packets) can be lost, reordered, duplicated

- TCP provides FIFO stream abstraction over IP
  handles:
  - reordering packets at receiver
  - retransmission of missing packets
Reasoning:

Simplest: Sender waits for ACK of packet n before sending packet n+1

Problem: Latency bounds throughput
Send batches to mask latency

Round-trip latency \[ \rightarrow [ \text{dead time} \rightarrow [ \text{ACK 3} \rightarrow [ \rightarrow 4 \rightarrow [ \rightarrow 5 \rightarrow [ \rightarrow 6 \rightarrow [ \rightarrow 7 \rightarrow [ \rightarrow 8 \rightarrow [ \rightarrow 9 \rightarrow \text{Sender} \]\]\]\]\]\]\]\]\]\]

Idea: Send packet 4 as soon as get ACK for packet 1
Keep pipe full — some packets always in transit
Solution: Sliding window

Sender: in transit

RCV

received

waiting to be sent

window

# of packets = window size

Sender waits & decreases size of window if:

- window is full
- haven't yet received ack for "oldest" packet in the window
- retransmit for missing ACK (timers)
- gradually ramp back up

Receiver: continuous ACK of largest seq. # in contiguous (no gaps)
Java supports TCP streams with Sockets

- Socket - connection between 2 processes
  ```
  Socket s
  s.getInputStream()
  s.getOutputStream()
  
  Can wrap these w/ any of the java.io wrappers.
  ```

- How to establish connection?
  Asymmetric

  Player 1
  be "ready" to accept a connection
  SERVER

  Player 2
  find player 1 & ask for a connection
  CLIENT
Server
Make itself available on the designated port.
accept() is a blocking call and returns a new Socket.

Client
new Socket (Server-IP, port-num);
needs to know:
- Server's IP address
- port number
public class SquareServer {

  ServerSocket ss = new ServerSocket(10450);
  Socket s = ss.accept(); // BLOCKING CALL
  DataInputStream in = new DataInputStream(s.getInputStream());
  double d = in.readDouble();
  DataOutputStream out = new DataOutputStream(s.getOutputStream());
  out.writeDouble(d * d);
  s.close();
  ss.close();

  Could put in a loop

  port number [10,000 - 30,000]
On client side

```java
public class SquareClient {
    String serverIPaddress = "localhost";
    int port = 10450;
    Socket s = new Socket(serverIPaddress, port);
    DataOutputStream out =
        new DataOutputStream(s.getOutputStream());
    out.writeInt(3);
    DataInputStream in =
        new DataInputStream(s.getInputStream());
    System.out.println(in.readDouble());
}
```

Client  

```
--------X
```

Server
Problem:
low level
lots of set-up

Better:
/ Appl. Developer model
* hide connection setup
* support concurrent clients (some clients may be slow or block forever)

MSG

Server

MSG

Client A

Client B
Design a client/server appl. framework:  

Asynchronous model

What should client developers do?

- define message types \[\text{classes } \Rightarrow \text{object streams}\]
- GUI for user (sitting on top)
- way to process messages received

API design choice

\[\begin{align*}
\rightarrow 1 & \text{ active loop to read incoming messages in app:} \\
& \Rightarrow \text{Message receive()} \\
\rightarrow 2 & \text{ react as we're told about messages (event-driven)} \\
& \Rightarrow \text{void messageReceived(Message m)}
\end{align*}\]

- active behavior of client (including sending messages) 
  (optional, but allow & support
  Runnable client objects)
- send(Message m) — sends m to the server (nonblocking)
API design choice

- way to process messages received
  \[ \Rightarrow \begin{align*}
  &\Rightarrow \text{active loop to read incoming messages in app:} \\
  &\Rightarrow \quad \text{Message receive()}; \\
  &\Rightarrow \quad \text{void messageReceived (Message m)};
  \end{align*} \]

\[\begin{align*}
0 &\quad \text{Client sends m, client waits for reply} \\
\hphantom{0} &\quad \text{or Client sends m, client waits for reply}
\end{align*} \]

\[\begin{align*}
1 &\quad \text{Blocking/Reactive} \\
2 &\quad \text{Meanwhile, client handles incoming messages from server}
\end{align*} \]

If both are available, using both simultaneously would be awkward: Don’t know whether blocking call or msg handler should get the next msg.
Server side:
Server developer will provide:
  - mechanism for handling each client
    - should be thread safe
    - ClientHandler (run in their own threads)

  - need a way to create ClientHandler objects
    (factory method)
  - way of handling messages received
    - sending messages
    - active? (Runnable?)
    - Server active computation (or just passive?)