Beyond HTTP

• Up to this point we have been dealing with software tools that run on browsers and communicate to a server that generates files that can be interpreted by the browser.

• What HTTP “is”: it forms elements (.html, .xml) with programs (processes) running on the server (e.g. java) and on clients (e.g. javaScript), cookies).
  – HTTP is a stateless protocol: the browser makes a simple request, the server responds and the connection is immediately closed.
  – Once the connection is broken, the server forgets all about the request and moves to handle the next request.

• May wish to establish more “persistent” connections between Browser and Server (e.g. TCP (Transmission Control Protocol, 100% reliability, echoes back), UTP (User Datagram Protocol, just throws it out there)) TCP/IP Tutorial Book Example: handout 6
Assignment 5: Beyond HTTP
Setting up persistent connections

• Create a system that accesses departure gate information from two (2) airlines
  – GateKeeper.java controls the flow of the app
  – GateListener.java listens to a UDP port for gate information
  – GateInformationBoard.java displays the gate info

Each airline maintains their own departure gate information on their own computers.
  carmonAir’s machine is named orf-3p180f1.princeton.edu
  (alt. caladbolg.princeton.edu)
  vanderFly’s machine is named kornhauser-laptop.princeton.edu
  (alt. balmung.princeton.edu)

Each is listening on port 8189 for registration information which includes machine name and port number that directs the outbound UDP gate data
Your machine needs to be listening for those messages at that port
Machines in Assignment 5

orf-3p180f1.princeton.edu (Prof K’s Desktop)

yourMachine

8189

Register & Accept

kornhauser-laptop.princeton.edu
Using Sockets to implement a more persistent Client–Server connection

- RMI (Remote Method Invocation): lets you easily access methods in remote java objects including the ability to send and receive general java objects.

- The java DataBase Connectivity (JDBC) API provides access to a wide variety of relational databases using a standard interface
Implementing a Client:

1. Create a Socket object
2. Create an output stream that can be used to send information to the Socket
3. Create an input stream to read the response from the server
4. Do I/O with the input and output streams
5. Close the socket when done!
Create a Socket object

- A socket is an object corresponding to a network connection
- A client connects to an existing server that is listening on a numbered network “port” for a connection
  - Standard way to making a socket connection is to supply a host name and a port # as follows:
    - Socket client = new Socket (“hostname”, portNum);
  - If you have an IP address (e.g. 198.4.155.21), then you can turn it into an InetAddress object by calling: getByName:
    - InetAddress address = InetAddress.getByName (“IP Address”);
    - Socket client = new Socket (address, portNumber);
This is then a connection oriented socket (TCP)
In Java 1.1: Socket client = new DatagramSocket (“hostname”, portNum);
Create an output stream that can be used to send info to the socket

- Can use any of a variety of OutputStream processes available for files
  - In Java 1.0 PrintStream of System.out
  - PrintStream out = new PrintStream (client.getOutputStream());
Create an inputStream (read data from server)

• **Use DataInoutStream**
  
  - DataInputSteam in = new DataInputSteam (client.getInputStream());

• **The input stream usually needs to be buffered:**
  
  - BufferedInputSteam buffered = new BufferedInputSteam (client.getInputStream());
  
  - DataInputSteam in = new DataInputSteam (buffered);
Comments

- A simple email client would send something to a server but not read anything back.
  - In this case, since you do not want to read anything back, then you don’t need an inputStream (step 3)

- In other cases you may want to read data from a server w/o sending anything first.
  - e.g. a network “clock” is simply pumping out the time at a certain port.
  - You then don’t need to create an output stream (step 2)

- If you do want to do I/O with input and output streams then realize:
  - readChar and readLine are “blocking” methods: they do not return until data is available
  - readFully “blocks” until all of the data is available
  - Need to convert byte arrays into strings;
    - String fullstring = new String (byteArray.o);
    - String partialString = new String(byteArray.o, tartIndex, numBytes);
Close Socket when done!!
(otherwise you may remain registered and may not be able to “re-register”)

• Client.clos();

(This will close the socket. That is, if you use the kill button on the date display windows, that will kill the connection. If you simply (abort) the command window that you used to execute Gatekeeper, that will NOT necessarily close your connection. When you try to register again, you may get a “can’t register” error message.)