UDP/IP in Java

Based on Java Network Programming and Distributed Computing
UDP Advantages

• Less overhead (no connection establishment)
• More efficient (no guaranteed delivery)
• Real-time applications (no error checking or flow-control)
  • E.g., weather, time, video, audio, games
• Data reception from more than one machine
Internet Addresses

• `java.net.InetAddress` class

• You get an address by using static methods:

  ```java
  ad = InetAddress.getByName(hostname);
  myAddress = InetAddress.getLocalHost();
  ```
Printing Internet Addresses

• You get information from an InetAddress by using methods:
  
  ad.getHostName();
  ad.getHostAddress();

• Example.
UDP Sockets Programming

• Sending/Receiving data.
  • `java.net.DatagramPacket` class

• Creating UDP sockets.
  • Client
  • Server
  • `java.net.DatagramSocket` class
Creating a UDP packet

// to receive data from a remote machine
DatagramPacket packet =
    new DatagramPacket(new byte[256], 256);

// to send data to a remote machine
DatagramPacket packet =
    new DatagramPacket(new byte[128], 128,
                        address, port);
Creating UDP sockets

• A UDP socket can be used both for reading and writing packets.

• Write operations are asynchronous; however, read operations are blocking.

• Since there is no guaranteed delivery, a single-threaded application could stall.
Creating UDP Sockets

// A client datagram socket:
DatagramSocket clientSocket =
    new DatagramSocket();

// A server datagram socket:
DatagramSocket serverSocket =
    new DatagramSocket(port);
Listening for UDP Packets

// create datagram packet
   ...

// create datagram server socket
   ...

boolean finished = false;
while ( ! finished ) {
    serverSocket.receive (packet);
    // process the packet
}

serverSocket.close();
Processing UDP Packets

ByteArrayInputStream bin =
    new ByteArrayInputStream(
    packet.getData() );
DataInputStream din =
    new DataInputStream(bin);

// read the contents of the packet
Sending UDP Packets

// create datagram packet
  . . .

// create datagram client socket
  . . .

boolean finished = false;
while ( ! finished ) {
  // write data to packet buffer
  clientSocket.send (packet);
  // see if there is more to send
}

Netprog 2002  TCP/IP
Sending UDP packets

• When you receive a packet, the ip and port number of the sender are set in the DatagramPacket.

• You can use the same packet to reply, by overwriting the data, using the method:

  ```java
  packet.setData(newbuffer);
  ```
Non-blocking I/O receiving UDP packets

• You can set a time-out in milliseconds to determine how long a read operation blocks, before throwing an exception.
  • `socket.setSoTimeout(duration);`

• If the duration given in milliseconds is exceeded, an exception is thrown:
  • `java.io.InterruptedIOException`
Typical UDP client code

- Create UDP socket to contact server (with a given hostname and service port number)
- Create UDP packet.
- Call `send(packet)`, sending request to the server.
- Possibly call `receive(packet)` (if we need a reply).
Typical UDP Server code

• Create UDP socket listening to a well known port number.
• Create UDP packet buffer
• Call `receive(packet)` to get a request, noting the address of the client.
• Process request and send reply back with `send(packet)`.
Debugging

- Debugging UDP can be difficult.
- Write routines to print out addresses.
- Use a debugger.
- Include code that can handle unexpected situations.
Asynchronous Errors

• What happens if a client sends data to a server that is not running?
  • ICMP “port unreachable” error is generated by receiving host and send to sending host.
  • The ICMP error may reach the sending host after send() has already returned!
  • The next call dealing with the socket could return the error.