UDP: User Datagram Protocol

• Generally faster than TCP.
• Less reliable than TCP:
  – No guarantee that messages ever arrive.
  – No guarantee that messages arrive in order.
  – Guaranteed correctness of packets that do arrive.
• Applications: VOIP, Music/Video Streaming, Multiplayer Games.
Analogy

- TCP: Phone Company.

- UDP: Post Office.
UDP Classes

- **DatagramPacket**:
  - Stores destination host and port.
  - Stores data to be transmitted.

- **DatagramSocket**:
  - Bound to a local port.
  - Supports `send` and `receive` of `DatagramPacket`s.

- Clients and servers both use these classes.
  - Servers use agreed ports.
  - Clients use anonymous ports.
Sending

1. Prepare a message string.
2. Convert message to byte array.
3. Construct `DatagramPacket` from message byte array, host address and port number.
4. Construct a `DatagramSocket` bound to agreed (server) or anonymous (client) port.
5. Invoke the `DatagramSocket`’s `send` method on the `DatagramPacket`. 
Scanner userEntry = new Scanner(System.in);
String message = userEntry.nextLine();
InetAddress host = InetAddress.getLocalHost();
int PORT = 1234;
DatagramPacket outPacket;
outPacket = new DatagramPacket(message.getBytes(),
message.length(),
host,
PORT);
DatagramSocket datagramSocket = new DatagramSocket();
datagramSocket.send(outPacket);
Receiving

1. Construct a `DatagramSocket` bound to an agreed (server) or anonymous (client) port.
2. Prepare an empty byte array.
3. Construct `DatagramPacket` from empty byte array.
5. Extract the message string and remote host address and port from `DatagramPacket`. 
int PORT = 1234;
DatagramSocket datagramSocket = new DatagramSocket(PORT);
byte[] buffer = new byte[256];
DatagramPacket inPacket;
inPacket = new DatagramPacket(buffer, buffer.length);
datagramSocket.receive(inPacket);
String messageIn;
messageIn = new String(inPacket.getData(),
                          0,
                          inPacket.getLength());
InetAddress clientAddress = inPacket.getAddress();
int clientPort = inPacket.getPort();
Choosing the Packet Size

• Unreliable Network:
  – Packets are often lost.
  – Small packet size is better.

• Reliable Network:
  – Efficient to send lots of data at a time.
  – Large packet size is better.

• Many implementations don’t support packet sizes above 8K = 8182 bytes.
UDP Connections

• A **DatagramSocket** can normally receive a message from any host/port at any time.

• This may lead to confusion in the case of a publicly advertized UDP server.

• UDP supports a limited kind of connection.

• **DatagramSocket connect** method:
  – Accepts a host address and a port number.
  – Subsequently the socket will discard messages it receives from other hosts or ports.
UDPEchoClient
UDPEchoServer

• Essentially the same as the TCP versions.
• But using UDP classes and methods.
• Problem with UDPEchoClient:
  – What happens if a packet is lost?
  – Program should handle it gracefully.
UDPEchoClientAlt

- Client program uses separate threads:
  - `SenderThread` for sending packets.
  - `ReceiverThread` for receiving packets.
- Sending and receiving threads use the same socket.
- Client program does not wait for response from server before sending next message.
UDPEchoServerAlt

- Abstract **UDPServer** class:
  - Sets up a **DatagramSocket**.
  - Repeatedly:
    1. Receives a **DatagramPacket** on the socket.
    2. Passes it to the abstract **respond** method.
- Derived class implements **respond** method to send message back to client.
Multicasting

• Facility for sending packets to many hosts with a single send operation.
• Uses `MulticastSocket` and `DatagramPacket` classes.
• Hosts join a multicast group associated with a multicast address.
• Sender makes a `DatagramPacket` addressed to the multicast address.
• Sender sends the datagram via a `MulticastSocket`. 
Multicast Addressing

- IP addresses from 225.0.0.0 to 238.255.255.255 are multicast addresses.
- No host machines are assigned to these addresses.
- Some of these addresses are reserved for use by routers.
- A user may set up a multicast associated with any other address in this range.
- Some multicast addresses have standard names and uses:
  - all-systems.mcast.net 224.0.0.1 (Multicasting to all hosts on any reachable network.)
  - experiment.mcast.net 224.0.1.20 (Multicasting to all hosts on the local subnetwork.)
“Private” Multicasting

- The 239.0.0.0/8 range is a locally administered address space with local or organizational scope.

- It may be used by anyone, without concern for address collisions, for private multicast domains.
Time To Live (TTL)

• Each packet is associated with an integer `TimeToLive` (TTL) value.

• TTL is (roughly) the number of routers through which the packet is allowed to pass.

• TTL is normally decremented by one each time it arrives at a router.

• When TTL reaches zero, the packet is not transmitted any further.

• TTL thus limits the propagation range of multicast packets.
Multicasting Versus Simple UDP

One Sent: Many Received

Two Sent: Two Received, etc.
Security in Multicasting

• On a public part of the Internet:
  – No authentication
  – No access control.

• On a private network: Authentication and access control may be provided by the network firewall.
MulticastSender

MulticastSocket ms = new MulticastSocket();
ms.setTimeToLive(TTL);

The sender program initializes a **MulticastSocket** on an anonymous port. It also sets the time-to-live value for the packets to be sent through the socket.
String message = "Multicast message.";
byte[] data = message.getBytes();
String addressName = "all-systems.mcast.net";
InetAddress ia = InetAddress.getByName(addressName);
int port = 1234;
DatagramPacket dp;
    dp = new DatagramPacket(data, data.length, ia, port);
ms.send(dp);

The program makes a **DatagramPacket** and fills it with a message along with the address and port number of a multicasting group. Then it sends the packet through the **MulticastSocket**.
MulticastSniffer

```java
int port = 1234;
MulticastSocket ms = new MulticastSocket(port);
String addressName = "all-systems.mcast.net";
InetAddress group = InetAddress.getByName(addressName);
ms.joinGroup(group);
```

The sniffer program initializes a `MulticastSocket` on a designated port integer. It then joins that socket to the group indicated by an `InetAddress` object.
byte[] buffer = new byte[8192];
while (true) {
    DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
    ms.receive(dp);
    String s = new String(dp.getData(),0,dp.getLength());
    System.out.println(s);
}

The sniffer program then uses the MulticastSocket like an ordinary DatagramSocket to receive data from the multicast group.
Multicast Messenger

- Nearly the same as MulticastSender.
- Allows user to enter messages from keyboard.
- Multicasts user’s message to whole multicasting group.
Multicast Chat Room

1. Log into a machine in the Asprey Lab.
2. Download examplesreleased.zip to your desktop.
3. Unzip the contents.
4. Start NetBeans
5. Open and run: MulticastSniffer
6. Open and run: MulticastMessenger
7. Send messages in messenger pane.
8. Read messages in sniffer pane.
9. Behave yourself!