Supplement IV.J: Networking Using Datagram Protocol

For Introduction to Java Programming
By Y. Daniel Liang

28.10 (Optional) Datagram Socket

Clients and servers that communicate via a stream socket have a dedicated point-to-point channel between them. To communicate, they establish a connection, transmit the data, and then close the connection. The stream sockets use TCP (Transmission Control Protocol) for data transmission. Since TCP can detect lost transmissions and resubmit them, transmissions are lossless and reliable. All data sent via a stream socket is received in the same order in which it was sent.

<Margin note: datagram>

In contrast, clients and servers that communicate via a datagram socket do not have a dedicated point-to-point channel. Data is transmitted using packets. Datagram sockets use UDP (User Datagram Protocol), which cannot guarantee that the packets are not lost, or not received in duplicate, or received in the order in which they were sent. A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

<Margin note: packet>

In an analogy, a stream socket communication between a client and a server is like a telephone connection with a dedicated link. A datagram communication is like sending a letter through the post office. Your letter is contained in an envelope (*packet*). If the letter is too large, it may be sent in several envelopes (packets). There is no guarantee that your letter will arrive or that the envelopes will arrive in the order they were sent. One difference is that the letter will not arrive in duplicate, whereas a datagram packet may arrive in duplicate.

Most applications require reliable transmission between clients and servers. In such cases, it is best to use stream socket network communication. Some applications that you write to communicate over the network will not require the reliable, point-to-point channel provided by TCP. In such cases, datagram communication is more efficient.

28.10.1 The DatagramPacket and DatagramSocket Classes

The java.net package contains two classes to help you write Java programs that use datagrams to send and receive packets over the network: DatagramPacket and

DatagramSocket. An application can send and receive DatagramPackets through a DatagramSocket.

28.10.1.1 The DatagramPacket Class

The DatagramPacket class represents a datagram packet. Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within the packet.

To create a DatagramPacket for delivery from a client, use the DatagramPacket(byte[] buf, int length, InetAddress host, int port) constructor. To create all other DatagramPackets, use the DatagramPacket(byte[] buf, int length) constructor, as shown in Figure 28.21. Once a datagram packet is created, you can use the getData and setData methods to obtain and set data in the packet.

******Same as Fig25.20 in intro6e p851***

<PD: UML Class Diagram>

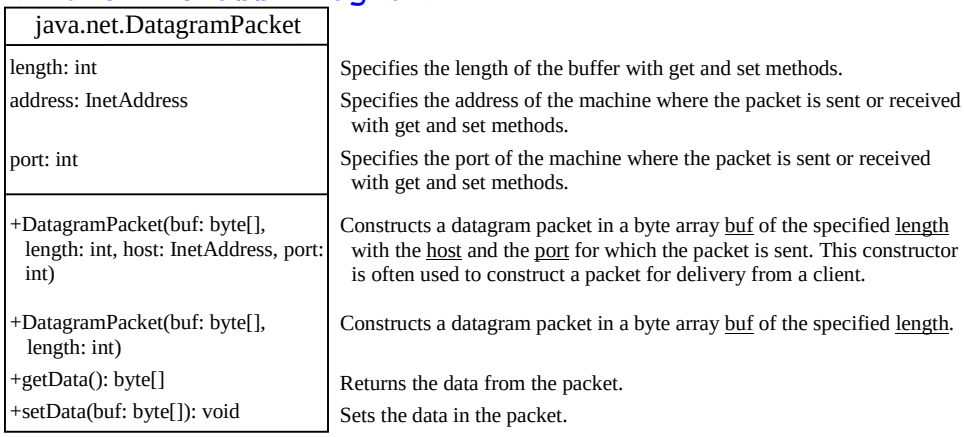

Figure 28.21

The DatagramPacket class contains the data and information about data.

28.10.1.2 DatagramSocket

The DatagramSocket class represents a socket for sending and receiving datagram packets. A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

<Margin note: construct datagram socket>

To create a server DatagramSocket, use the constructor DatagramSocket(int port), which binds the socket with the specified port on the local host machine.
To create a client DatagramSocket, use the constructor DatagramSocket(), which binds the socket with any available port on the local host machine.

<Margin note: send packet>
To send data, you need to create a packet, fill in the contents, specify the Internet address and port number for the receiver, and invoke the send(packet) method on a DatagramSocket.

<Margin note: receive packet>
To receive data, you have to create an empty packet and invoke the receive(packet) method on a DatagramSocket.

28.10.3 Datagram Programming
Datagram programming is different from stream socket programming in the sense that there is no concept of a ServerSocket for datagrams. Both client and server use DatagramSocket to send and receive packets, as shown in Figure 28.22.

******Same as Fig25.21 in intro6e p852***

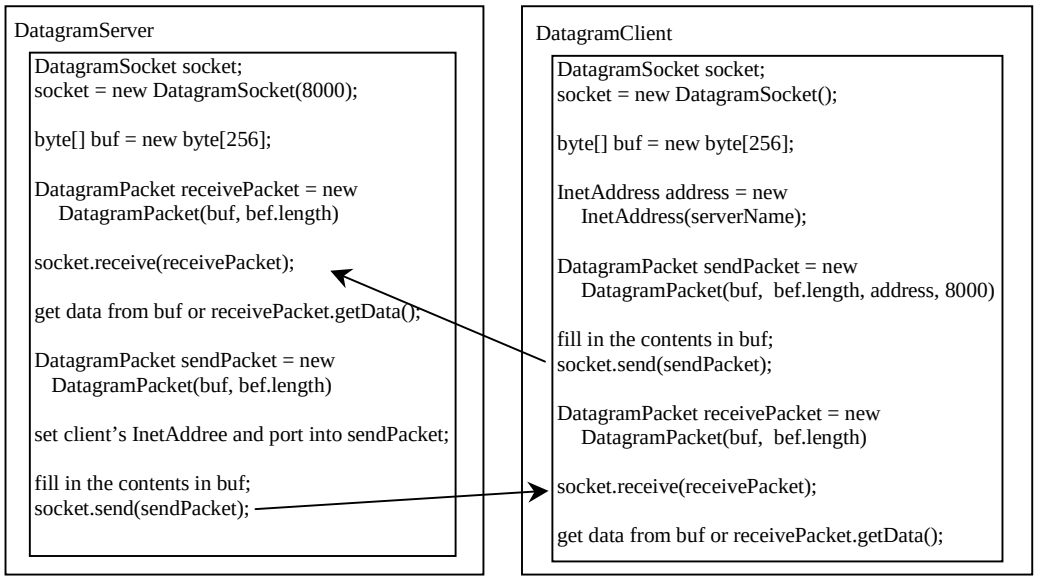

Figure 28.22
The programs send and receive packets via datagram sockets.

Normally, you designate one application as the server and create a DatagramSocket with the specified port using the

constructor DatagramSocket(port). A client can create a DatagramSocket without specifying a port number. The port number will be dynamically chosen at runtime. When a client sends a packet to the server, the client's IP address and port number are contained in the packet. The server can retrieve it from the packet and use it to send the packet back to the client.

To demonstrate, let us rewrite the client and server programs in Listings 28.1 and 28.2 using datagrams rather than socket streams. The client sends the radius to a server. The server receives this information, uses it to find the area, and then sends the area to the client.

Listing 28.15 gives the server, and Listing 28.16 gives the client. A sample run of the program is shown in Figure 28.23.

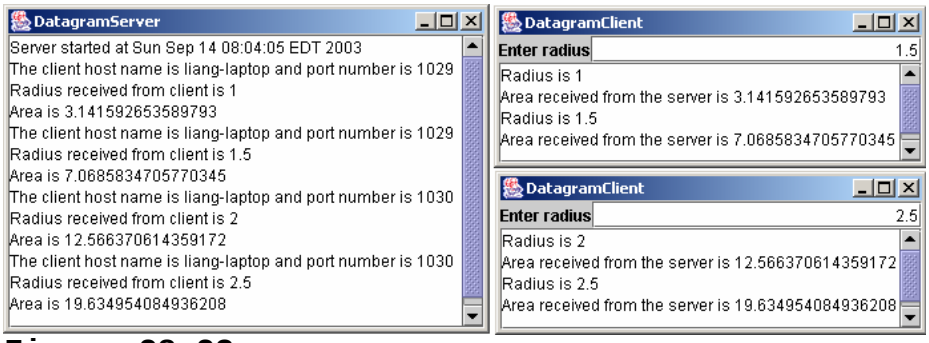

Figure 28.23
The server receives a radius from a client, computes the area, and sends the area to the client. The server can serve multiple clients.

Listing 28.15 DatagramServer.java

PD: Please add line numbers in the following code
<Margin note line 19: create UI>
<Margin note line 30: datagram socket>
<Margin note line 34: incoming packet>
<Margin note line 38: outgoing packet>
<Margin note line 46: receive packet>
<Margin note line 59: packet address >
<Margin note line 62: send packet>

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;

public class DatagramServer extends JFrame {
  // Text area for displaying contents
  private JTextArea jta = new JTextArea();

  // The byte array for sending and receiving datagram packets
  private byte[] buf = new byte[256];
```

```
  public static void main(String[] args) {
    new DatagramServer();
  }

  public DatagramServer() {
    // Place text area on the frame
    setLayout(new BorderLayout());
    add(new JScrollPane(jta), BorderLayout.CENTER);

    setTitle("DatagramServer");
    setSize(500, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true); // It is necessary to show the frame here!

    try {
      // Create a server socket
      DatagramSocket socket = new DatagramSocket(8000);
      jta.append("Server started at " + new Date() + '\n');

      // Create a packet for receiving data
      DatagramPacket receivePacket =
        new DatagramPacket(buf, buf.length);

      // Create a packet for sending data
      DatagramPacket sendPacket =
        new DatagramPacket(buf, buf.length);

      while (true) {
        // Initialize buffer for each iteration
        Arrays.fill(buf, (byte)0);

        // Receive radius from the client in a packet
        socket.receive(receivePacket);
        jta.append("The client host name is " +
          receivePacket.getAddress().getHostName() +
          " and port number is " + receivePacket.getPort() + '\n');
        jta.append("Radius received from client is " +
          new String(buf).trim() +  '\n');

        // Compute area
        double radius = Double.parseDouble(new String(buf).trim());
        double area = radius * radius * Math.PI;
        jta.append("Area is " + area + '\n');

        // Send area to the client in a packet
        sendPacket.setAddress(receivePacket.getAddress());
        sendPacket.setPort(receivePacket.getPort());
        sendPacket.setData(new Double(area).toString().getBytes());
        socket.send(sendPacket);
      }
    }
    catch(IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

Listing 28.16 DatagramClient.java

PD: Please add line numbers in the following code

<Margin note line 34: create UI>

<Margin note line 54: datagram socket>

<Margin note line 56: outgoing packet>

<Margin note line 58: incoming packet>

<Margin note line 74: send packet>

<Margin note line 77: receive packet >

```
import java.io.*;
```

```
import java.net.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DatagramClient extends JFrame {
  // Text field for receiving radius
  private JTextField jtf = new JTextField();

  // Text area to display contents
  private JTextArea jta = new JTextArea();

  // Datagram socket
  private DatagramSocket socket;

  // The byte array for sending and receiving datagram packets
  private byte[] buf = new byte[256];

  // Server InetAddress
  private InetAddress address;

  // The packet sent to the server
  private DatagramPacket sendPacket;

  // The packet received from the server
  private DatagramPacket receivePacket;

  public static void main(String[] args) {
    new DatagramClient();
  }

  public DatagramClient() {
    // Panel p to hold the label and text field
    JPanel p = new JPanel();
    p.setLayout(new BorderLayout());
    p.add(new JLabel("Enter radius"), BorderLayout.WEST);
    p.add(jtf, BorderLayout.CENTER);
    jtf.setHorizontalAlignment(JTextField.RIGHT);

    setLayout(new BorderLayout());
    add(p, BorderLayout.NORTH);
    add(new JScrollPane(jta), BorderLayout.CENTER);

    jtf.addActionListener(new ButtonListener()); // Register listener

    setTitle("DatagramClient");
    setSize(500, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true); // It is necessary to show the frame here!

    try {
      // get a datagram socket
      socket = new DatagramSocket();
      address = InetAddress.getByName("localhost");
      sendPacket =
        new DatagramPacket(buf, buf.length, address, 8000);
      receivePacket = new DatagramPacket(buf, buf.length);
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
  }

  private class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      try {
        // Initialize buffer for each iteration
        Arrays.fill(buf, (byte)0);

        // send radius to the server in a packet
```

```
        sendPacket.setData(jtf.getText().trim().getBytes());
        socket.send(sendPacket);

        // receive area from the server in a packet
        socket.receive(receivePacket);

        // Display to the text area
        jta.append("Radius is " + jtf.getText().trim() + "\n");
        jta.append("Area received from the server is "
          +  Double.parseDouble(new String(buf).trim()) + '\n');
      }
      catch (IOException ex) {
        ex.printStackTrace();
      }
    }
  }
}
```

Since datagrams are connectionless, a DatagramPacket can be sent to multiple clients, and multiple clients can receive a packet from the same server. As shown in this example, you can launch multiple clients. Each client sends the radius to the server, and the server sends the area back to the client.

The server creates a DatagramSocket on port 8000 (line 30 in DatagramServer.java). No DatagramSocket can be created again on the same port. The client creates a DatagramSocket on an available port (line 55 in DatagramClient.java). The port number is dynamically assigned to the socket. You can launch multiple clients simultaneously, and each client's datagram socket will be different.

The client creates a DatagramPacket named sendPacket for delivery to the server (lines 57-58 in DatagramClient.java). The DatagramPacket contains the server address and port number. The client creates another DatagramPacket named receivePacket (line 59), which is used for receiving packets from the server. This packet does not need to contain any address or port number.

A user enters a radius in the text field in the client. When the user presses the Enter key on the text field, the radius value in the text field is put into the packet and sent to the server (lines 74-75 in DatagramClient.java). The server receives the packet (line 48 in DatagramServer.java), extracts the data from the byte array buf, and computes the area (lines 54-55 in DatagramServer.java). The server then builds a packet that contains the area value in the buffer, the client's address, and the port number, and sends the packet to the client (lines 60-63). The client receives the packet (line 77 in DatagramClient.java) and displays the result in the text area.

The data in the packet is stored in a byte array. To send a numerical value, you need to convert it into a string and then store it in the array as bytes, using the getBytes() method in the String class (line 60 in DatagramServer.java and line 72 in DatagramClient.java). To convert the array into a number, first convert it into a string, and then convert it into a number using the static parseDouble method in the Double class (line 53 in DatagramServer.java and line 81 in DatagramClient.java).

> NOTE: The port numbers for the stream socket and the datagram socket are not related. You

can use the same port number for a stream socket and a datagram socket simultaneously.