

Supplement

J

Network Programming Using Datagrams

J.1 Introduction

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 are received in the same order in which they were sent.

In contrast, clients and servers that communicate via a datagram socket do not have a dedicated point-to-point channel. Data are transmitted using packets. Datagram sockets use UDP (User Datagram Protocol), which cannot guarantee that the packets are not lost, or 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.

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 postal 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 arrive in the order it was 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.

Chapter 18, "Networking," introduces network programming using stream sockets. This supplement introduces datagram programming.

J.2 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.

J.2.1 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. Multiple packets sent from one machine to another may be routed differently and may arrive in any order. Packet delivery is not guaranteed.

To create a DatagramPacket, use the following two constructors:

- public DatagramPacket(byte[] buf, int length, InetAddress host, int port)
Constructs a datagram packet in a byte array buf of the specified length. It also specifies the host and the port for which the packet is sent. This constructor is often used to construct a packet for delivery.
- public DatagramPacket(byte[] buf, int length)
Constructs a datagram packet in a byte array buf of the specified length. This constructor is often used to construct a packet for receiving.

The DatagramPacket class provides the following methods to get and set data, host address, and port number in the packet.

- public byte[] getData()
Returns the data buffer.
- public void setData(byte[] buf)
Sets the data buffer for the packet.
- public int getLength()
Returns the length of the data in the packet. This is not the length of the buffer.
- public InetAddress[] getAddress()
Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
- public void setAddress(InetAddress iaddr)
Sets the IP address of the machine to which the datagram is being sent.
- public int getPort()
Returns the port number on the remote host to which the datagram is being sent or from which the datagram was received.
- public void setPort(int port)
Sets the port number on the remote host to which the datagram is being sent.

J.2.2 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.

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.

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`.

To receive data, create an empty packet and invoke the `receive(packet)` method on a `DatagramSocket`.

J.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 J.1.

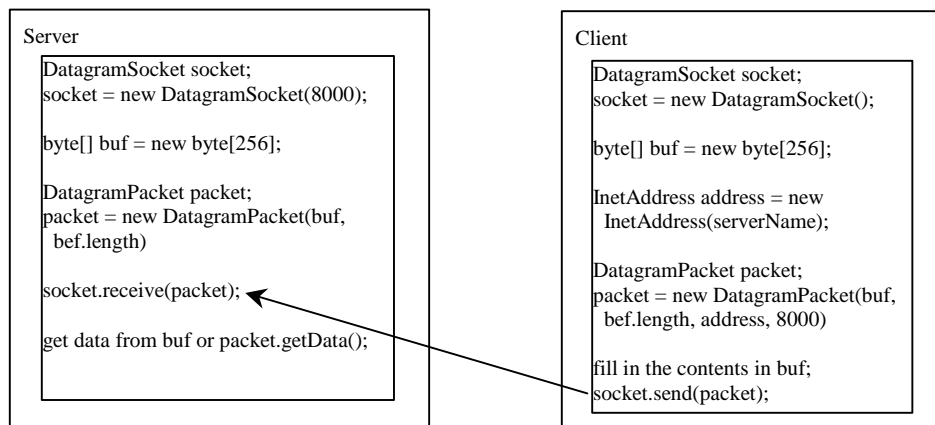


Figure J.1

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 Example 21.1, "A Client/Server Example," using datagrams.

Example J.1

Client/Server Programming Using Datagrams

Problem

Example 21.1 presents a client program and a server program using socket streams. The client sends radius to a server. The server receives the data, uses them to find the area, and then sends the area to the client. Rewrite the program using datagram sockets.

Solution

The server and client programs follow. A sample run of the program is shown in Figure J.2.

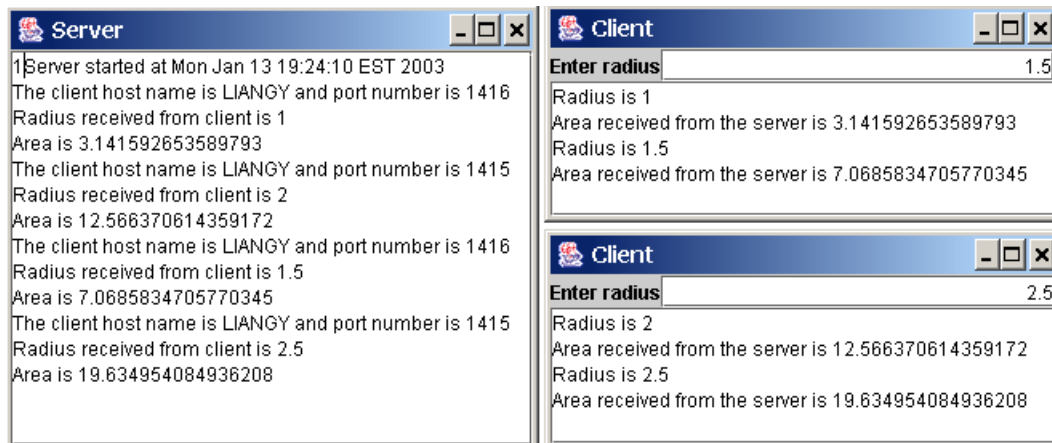


Figure J.2

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

*****PD: Please add line numbers in the following code*****

```

1 // Server.java: The server accepts data from the client, processes it
2 // and returns the result back to the client using datagram packet
3 package supplementj;
4
5 import java.io.*;
6 import java.net.*;
7 import java.util.*;
8 import java.awt.*;
9 import java.awt.event.*;
10 import javax.swing.*;
11
12 public class Server extends JFrame {
13     // Text area for displaying contents
14     private JTextArea jta = new JTextArea();
15
16     // The byte array for sending and receiving datagram packets
17     private byte[] buf = new byte[256];
18
19     public static void main(String[] args) {
20         new Server();
21     }
22
23     public Server() {
24         // Place text area on the frame
25         getContentPane().setLayout(new BorderLayout());
26         getContentPane().add(new JScrollPane(jta), BorderLayout.CENTER);
27
28         setTitle("Server");
29         setSize(500, 300);
30         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31         setVisible(true); // It is necessary to show the frame here!
32     }
33 }

```

```

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();
}
}

```

*****PD: Please insert a separate line**

*****PD: Please add line numbers in the following code*****

```

// Client.java: The client sends the input to the server and receives
// result back from the server using datagram socket
package supplementj;

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

public class Client extends JFrame implements ActionListener {
    // 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 Client();
}

public Client() {
    // 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);

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

    jtf.addActionListener(this); // Register listener

    setTitle("Client");
    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();
    }
}

public void actionPerformed(ActionEvent e) {
    String actionCommand = e.getActionCommand();
    if (e.getSource() instanceof JTextField) {
        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();
        }
    }
}
}

```

Review

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 35 in Server.java). No DatagramSocket can be created again on the same port. The client creates a DatagramSocket on an available port (Line 59 in

Client.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 61-62 in Client.java). The DatagramPacket contains the server address and port number. The client creates another DatagramPacket named receivePacket (Line 63), 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. Upon pressing 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 78-79 in Client.java). The server receives the packet (Line 51 in Server.java), extracts the data from the byte array buf, and computes the area (Lines 59-60 in Server.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 64-67). The client receives the packet (Line 82) and displays the result in the text area.

The data in the packet are 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 66 in Server.java and Line 78 in Client.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 59 in Server.java and Line 87 in Client.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.