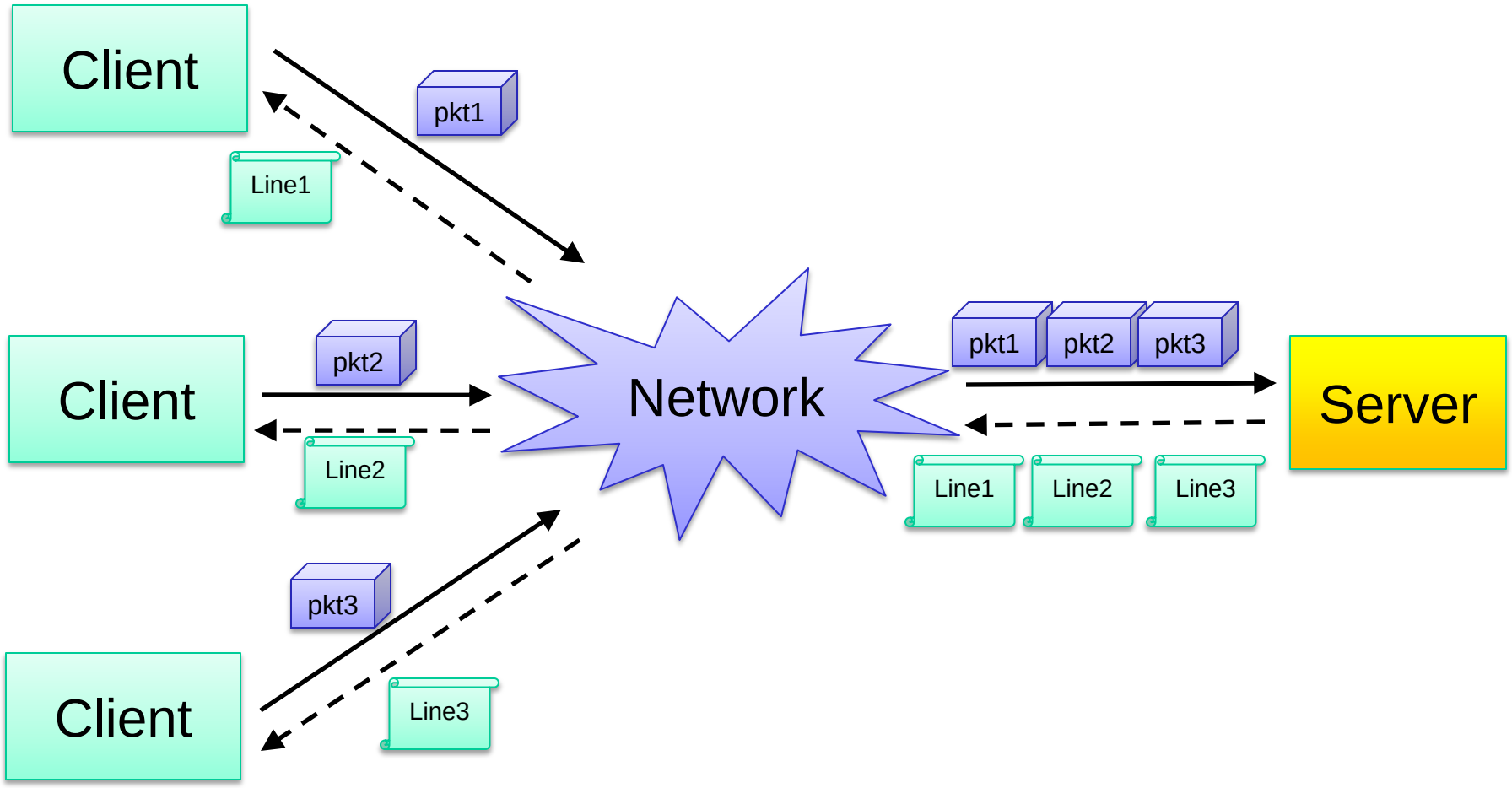
Università degli Studi di Bologna
Facoltà di Ingegneria

Principles, Models, and Applications for Distributed Systems M

Lab assignment 2 (worked-out)
Connectionless Java Sockets

Luca Foschini

Architecture



Specifications: Client

Develop the following C/S application.

The **client**, that requires as arguments server IP address and port, sends to the server a packet with **a text file name** and **the line number it wants to receive**. These parameters are asked to the user using the command prompt.

Note: the same packet carries both parameters

Why don't using two distinct packets?

Then, the client prints **prints the content of the reply packet received from the server**, that contains either the requested line or a negative response. To avoid that the client waits forever, if the server crashes or the reply packet is not delivered, the client should block waiting for an answer for at most 30 s.

To set the timeout use the **`setTimeout`** method on the socket itself (after its construction). This method allows to avoid indefinitely long waits when calling blocking methods of a socket: blocking methods will be unblocked after a time interval equal to the desired timeout.

Specifications: Server

The **server** receives client requests.

It **tries to open the requested file and replies to the request**.

If the file does not exist or it is not possible to satisfy the request, it builds an error message to be returned to the client.

Finally, it **replies** to the client: if the file does not exist or it is not long enough to contain the requested line, it replies with the error message; else it sends to the client a packet with the desired line.

NOTE: the server has been implemented as **sequential** (single-process), that runs an **endless loop** repeating the following steps:

1. Wait for a request;
2. Construct reply;
3. Send reply.

DatagramPacket class

This class **represents incoming and outgoing UDP packets** that are sent and received using Datagram sockets.

To build a datagram packet and associate it to a destination IP, the constructor requires:

- the data payload (the `ibuf` byte array of `ilength` bytes)
- the destination IP address;
- the destination port.

```
public DatagramPacket(byte ibuf[], int ilength,
                      InetAddress iaddr, int iport)
```

To build a datagram and associate to it the content received from a socket, it is sufficient to specify the data buffer and its length:

```
public DatagramPacket(byte ibuf[], int ilength)
```

DatagramPacket allows to get and set many parameters:

```
getAddress(),    setAddress(InetAddress addr)
getPort(),       setPort(int port)
getData(),       setData(byte[] buf), etc.
```

InetAddress class

This class represents Internet addresses, abstracting their representation (numeric values or logic names) → **portable and transparent**

It has no constructors and provides static methods:

```
public static InetAddress getByName(String hostname);
```

Determines host IP address given a DNS host name. Host name can either be a machine name, such as “www.unibo.it”, or a textual representation of its IP address. If the host is null then returns an InetAddress representing the loopback interface.

```
public static InetAddress[] getAllByName(String hostname);
```

Returns an array of host IP addresses given a DNS host name; useful for logic names that have many IP addresses.

```
public static InetAddress getLocalHost();
```

Returns the InetAddress of the local machine. If the local machine is not connected to a network, it returns the loopback address: 127.0.0.1

All these methods may throw a **`UnknownHostException`** if they are unable to get the IP address of the host.

Solution sketch: Client 1/2

1. Create a socket, set options and build a DatagramPacket:

```
DatagramSocket socket =  new DatagramSocket();
socket.setTimeout(30000);
byte[] buf = new byte[512];
DatagramPacket packet =
    new DatagramPacket(buf, buf.length, addr, port);
```

2. Ask the user for input:

```
BufferedReader stdIn =
  new BufferedReader(new InputStreamReader(System.in));
System.out.print("\n^D(Unix)/^Z(Win) to exit, "+
  "insert the target filename:");
while ((filename=stdIn.readLine())!=null){
  System.out.print("Line number? ");
  lineNumber = Integer.parseInt(stdIn.readLine());
  …
}
```

Solution sketch: Client 2/2

3. Construct the data buffer that wraps the user request:

```
ByteArrayOutputStream boStream = new ByteArrayOutputStream();
DataOutputStream doStream = new DataOutputStream(boStream);
doStream.writeInt(lineNumber);
doStream.writeUTF(filename);
doStream.flush();
byte[] data = boStream.toByteArray();
```

4. Set the buffer as packet payload and send it to the server:

```
packet.setData(data);
socket.send(packetOUT);
```

5. Initialize incoming packet and wait for an answer:

```
packet.setData(buf);
socket.receive(packet);
```

6. Extract data from response packet:

```
ByteArrayInputStream biStream =
  new ByteArrayInputStream(packet.getData(),0,packet.getLength());
DataInputStream diStream = new DataInputStream(biStream);
String response = diStream.readUTF();
```

Solution sketch: Server 1/2

1. Create the socket and a DatagramPacket:

```
DatagramSocket socket = new DatagramSocket(port);
byte[] buf = new byte[512];
DatagramPacket packet =
                new DatagramPacket(buf,buf.length);
```

2. Initialize packet and wait for incoming request:

```
packet.setData(buf);
socket.receive(packet);
```

3. Extract parameters from incoming packet:

```
ByteArrayInputStream biStream =
  new ByteArrayInputStream(packet.getData(),
                           0,packet.getLength());
DataInputStream diStream = new DataInputStream(biStream);
linenumber = diStram.readInt();
filename = diStream.readUTF();
```

Solution sketch: Server 2/2

4. Prepare response:

```
String line = LineUtility.getLine(filename,
  linenumber);
ByteArrayOutputStream boStream =
  new ByteArrayOutputStream();
DataOutputStream doStream =
  new DataOutputStream(boStream);
doStream.writeUTF(line);
doStream.flush();
data = boStream.toByteArray();
```

5. Attach data to packet and send it:

```
packet.setData(data, 0, data.length);
socket.send(packet);
```

Remember to call close() for all unused sockets!

LineUtility

```
public class LineUtility {

// Static method that finds and returns the n-th line of the specified file
   static String getLine(String filename, int linenumber)
   { String line = null;
     BufferedReader in = null;
    try { in = new BufferedReader(new FileReader(filename)); }
    catch (FileNotFoundException e)
     { e.printStackTrace(); return linea = "File not found"; }
    try
     { for (int i=1; i<=linenumber; i++)
       { line = in.readLine();
          if ( line == null)
          { line = "Line not found"; in.close(); return line; }
       }
     }
     catch (IOException e)
       { e.printStackTrace(); return line = "Line not found"; }
     return line;
   }
}
```

Line Client 1/3

```
public class LineClient {
 public static void main(String[] args)
 { InetAddress addr=null; int port = -1;
  try
  {if (args.length == 2)
    {addr = InetAddress.getByName(args[0]);
     port = Integer.parseInt(args[1]);
    }
   else{ System.out.println("Usage: java LineClient addr port");
         System.exit(1);
   }   }
  catch(UnknownHostException e){...}
  DatagramSocket socket=null; DatagramPacket packet = null;
  byte[] buf = new byte[512];

  try{
    socket = new DatagramSocket(); socket.setSoTimeout(30000);
    packet = new DatagramPacket(buf, buf.length, addr, port);
  }
  catch (SocketException e) { e.printStackTrace();System.exit(2);}
```

Line Client 2/3

// user interaction

```
BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
System.out.print("\n(^D(Unix)/^Z(Win) to exit) insert a filename");
try{
  ByteArrayOutputStream boStream = null; DataOutputStream doStream = null;
  byte[] data = null; String filename = null; int linenumber = -1;
  String response = null;
  ByteArrayInputStream biStream = null; DataInputStream diStream = null;

  while ((filename=stdIn.readLine())!=null){
   try{
    System.out.print("Line number? ");
     lineNumber = Integer.parseInt(stdIn.readLine());
}
  catch (Exception e) {
    System.out.println("Error while asking for input: ");
    e.printStackTrace(); System.out.print("\n^D(Unix)/... ");
    continue;
  }
```

Line Client 3/3

```
try{   // construct and send request
    boStream = new ByteArrayOutputStream();
    doStream = new DataOutputStream(boStream);
    doStream.writeInt(linenumber);
    doStream.writeUTF(filename);
    data = boStream.toByteArray();
   packet.setData(data); socket.send(packet);
 } catch (IOException e){...}
try{   // initialize and receive packet
   packet.setData(buf); socket.receive(packet);
  } catch (IOException e){ /* same as before */ }
 try{
   biStream = new ByteArrayInputStream(packet.getData(),
                                       0,packet.getLength());
   diStream = new DataInputStream(biStream);
  response = diStream.readUTF();
  }
 catch (IOException e){... }
 System.out.print("\n^D(Unix)/^Z(Win...");
}
System.out.print("\n^D(Unix)/^Z(Win...");
} //while
} catch(Exception e){...}
socket.close();
System.out.println("LineClient: terminating..."); }}
```

Line Server 1/3

```
public class LineServer {
// Unrestricted port range: 1024-65535
private static final int DEFAULTPORT = 4445;

public static void main(String[] args)
{  DatagramSocket socket = null;
   DatagramPacket packet = null;
   byte[] buf = new byte[512];

 // Check input arguments: 0 or 1 argument (port number)
 if ((args.length == 0)) { port = DEFAULTPORT; }
 else if (args.length == 1) {
   try {
     port = Integer.parseInt(args[0]);
     if (port < 1024 || port > 65535) {  // check port
          System.out.println("Usage: java LineServer [serverPort>1024]");
          System.exit(1);
     }
   } catch (NumberFormatException e) { … }
 } else {
   System.out.println("Usage: java LineServer [serverPort>1024]");
   System.exit(1);
 }
```

Line Server 2/3

```
try
 {socket = new DatagramSocket(port);
    packet = new DatagramPacket(buf, buf.length);
 }
 catch (SocketException e) {...}

 try
 { String filename = null;
   int linenumber = -1;
   ByteArrayInputStream biStream = null;
   DataInputStream diStream = null;
   ByteArrayOutputStream boStream = null;
   DataOutputStream doStream = null;
   String line = null;
   byte[] data = null;

while (true)
{
try { packet.setData(buf);
      socket.receive(packet); }
     catch(IOException e){ ... continue; }
```

Line Server 3/3

```
try
 { biStream = new ByteArrayInputStream( packet.getData(),0
,packet.getLength());
    diStream=new DataInputStream(biStream);
    linenumber = diStream.readInt();
    filename = diStream.readUTF();
} catch(Exception e){/* same as before */}

try
  { String line =  LineUtility.getLine(filename, linenumber);
    boStream = new ByteArrayOutputStream();
    doStream = new DataOutputStream(boStream);
    doStream.writeUTF(line);
    data = boStream.toByteArray();
    packet.setData(data); socket.send(packet);
  } catch(IOException e){/* same as before */}
 }  // while
} catch(Exception e){...}
System.out.println("LineServer: terminating..");
socket.close();
}  // main
}  // class
```