Middlesex University London

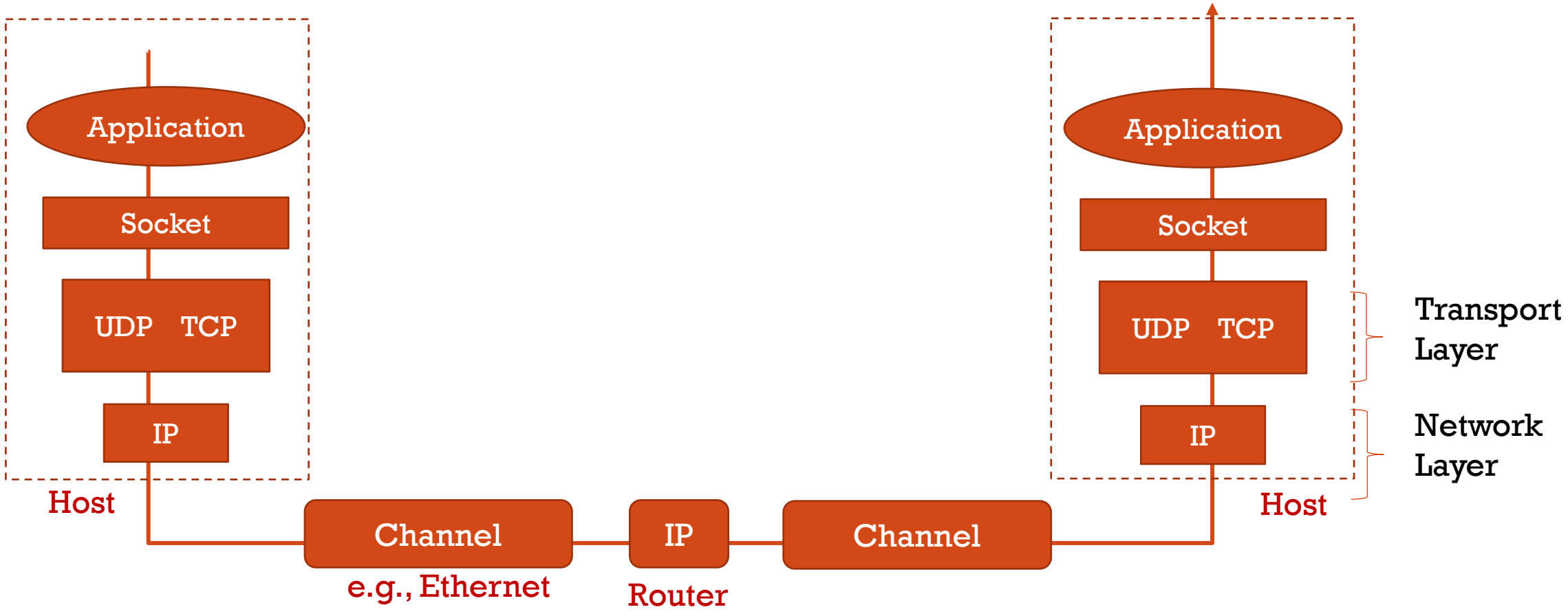
Lecture 17

Networking

TODAY:

- Network layers
- Client/ Server architecture
- Socket programming in Java
  - TCP socket networking
  - UDP datagram network
- Socket programming in Java one way
- Socket programming in Java two ways
- InetAddress
- UDP socket programming

NETWORK LAYERS - TCP/IP

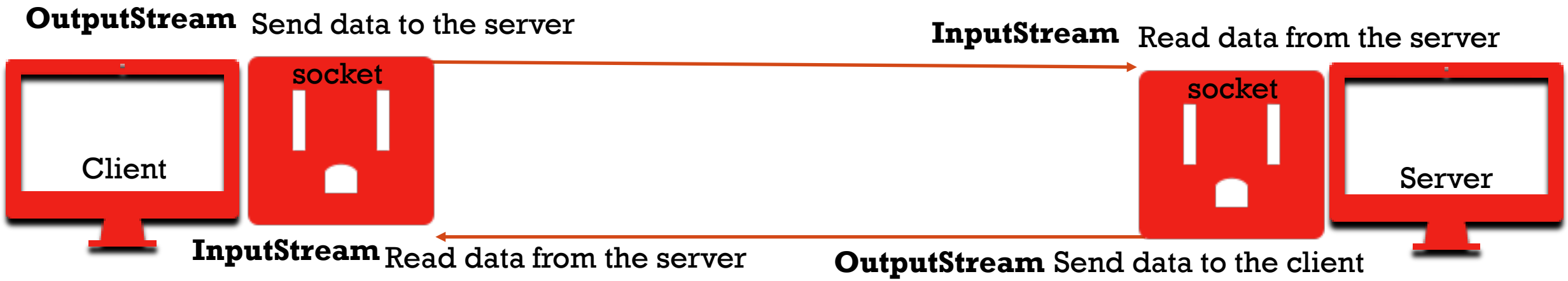


TCP/IP Socket in Java Practical guide for programmer K L. Calvert, M J. Donahoo

SOCKET

- A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a **port number** so that the TCP layer can identify the application that data is destined to be sent to.
- An endpoint is a combination of an **IP address** and a **port number**.
- **IP** - Unique address on the local network (for example: 127.0.0.1)
- **Port Number** – Unique id. Number on your computer link to your program (Ports range from 1 – 65535.) from 1 to 1023 are reserve for well known services.
- **Packet** – Unit of data send from one computer to another
- Higher level protocols:
  - **TCP** Transmission control protocol: provide reliable two-way connected communication
  - **UDP** User datagram protocol: connecting may be unreliable

CLIENT AND SERVER



CLIENT/SERVER BASIC STRUCTURE

- **Client-** (The **Socket** class represents a socket client.)
- 1. The client initiates connection to a server specified by hostname/IP address and port number.
- 2. Send data to the server using an OutputStream.
- 3. Read data from the server using an InputStream.
- 4. Close the connection.

- **Server** – (The **ServerSocket** class is used to implement a server program)
- 1. Create a server socket and bind it to a specific port number
- 2. Listen for a connection from the client and accept it. This results in a client socket is created for the connection.
- 3. Read data from the client via an InputStream obtained from the client socket.
- 4. Send data to the client via the client socket's OutputStream.
- 5. Close the connection with the client.

ONE WAY - CLIENT (EXAMPLE)

```
14
    public class ClientSocket_ {
16
17      public static void main(String[] args) throws IOException {
18  //1. The client initiates connection to a server specified by hostname/IP address and port numbe
19          System.out.println("Client ready!");
20          Socket socket = new Socket("localhost", 9999);
21
22  //2. Send data to the server using an OutputStream.
23          OutputStreamWriter os = new OutputStreamWriter (socket.getOutputStream());
24          PrintWriter out = new PrintWriter (os);
25          out.println("Hello world!");
26          out.flush();
27      }
28  }
```

ONE WAY – SERVER

```java
public class ServerSocket {
    public static void main(String[] args) throws IOException {
        System.out.println("Server Ready!");
    //1. Create a server socket and bind it to a specific port number
        ServerSocket ss = new ServerSocket(9999);
    // 2. Listen for a connection from the client and accept it.
        System.out.println("Server is waiting for client request");
        Socket socket = ss.accept();
        System.out.println("Client connected");

    //3. Read data from the client via an InputStream obtained from the client socket.
        BufferedReader br = new BufferedReader (new InputStreamReader(socket.getInputStream()));
        String str = br.readLine();
        System.out.println("ClientData: "+str);
    }

}
```

TWO WAYS – CLIENT

```
14
public class ClientSocket {
16
17      public static void main(String[] args) throws IOException {
18  //1. The client initiates connection to a server specified by hostname/IP address and port number.
19          System.out.println("Client ready!");
            Socket socket = new Socket("localhost", 9999);
21
22  //2. Send data to the server using an OutputStream.
23          OutputStreamWriter os = new OutputStreamWriter (socket.getOutputStream());
24          PrintWriter out = new PrintWriter (os);
25          out.println("Hello world!");
26          out.flush();
27
28   //3. Read data from the server using an InputStream.
29          BufferedReader br = new BufferedReader (new InputStreamReader(socket.getInputStream()));
30          String str = br.readLine();
31          System.out.println("Server reply: "+ str);
32
33     //4. Close the connection.
34         socket.close();
35      }
36  }
37
```

TWO WAYS - SERVER

```java
15
public class ServerSocket {
17      public static void main(String[] args) throws IOException {
18          System.out.println("Server Ready!");
19      //1. Create a server socket and bind it to a specific port number
            ServerSocket ss = new ServerSocket(9999);
21      // 2. Listen for a connection from the client and accept it.
22          System.out.println("Server is waiting for client request");
23          Socket socket = ss.accept();
24          System.out.println("Client connected");
25
26      //3. Read data from the client via an InputStream obtained from the client socket.
27          BufferedReader br = new BufferedReader (new InputStreamReader(socket.getInputStream()));
28          String str = br.readLine();
29          System.out.println("ClientData: "+str);
30
31      //4. Send data to the client via the client socket's OutputStream.
32          OutputStreamWriter os = new OutputStreamWriter (socket.getOutputStream());
33          PrintWriter out = new PrintWriter (os);
34          out.println("Hello from server!");
35          out.flush();
36          System.out.println("Data sent");
37
38        //5. Close the connection with the client.
39
40          ss.close();
41      }
42
43  }
```

TCP

Transmission control protocol

- Check the GITHub repository for example of TCPClient and TCPServer with some additional very useful functionalities as well as other examples available:

Check related files

UDP CLIENT

```java
public class UDPClient {
    public static void main(String[] args) throws SocketException, UnknownHostException, IOException {

    // DatagramSocket to send and receive DatagramPackets.
        DatagramSocket ds = new DatagramSocket();

        //to  send data----------------------------------------
        int i =8; //data to send
        //need to  e change into binary format
        byte [] b = String.valueOf(i).getBytes();
        //InetAddress returns the address of the localhost.
        InetAddress ia = InetAddress.getLocalHost();

        //the packet would have 4 parameters: data in bite, data length, IP address, Port number
        DatagramPacket dp = new DatagramPacket(b,b.length,ia,9999);
        //send(DatagramPacket p): sends a datagram packet.
        ds.send(dp);

        //to accept the data back----------------------------------

        //temp array for reciving binary data
        byte[] b1= new byte [1024];
        //when reciving data the packet would have only 2 parameters, does not require any more IP or po
        DatagramPacket dp1 = new DatagramPacket(b1,b1.length);
        ds.receive(dp1);
        String str = new String (dp1.getData()).trim();

        System.out.println("the result is: "+ str);
    }

}
```

UDP SERVER

```
17  */
18  public class UDPServer {
19      public static void main(String[] args) throws SocketException, IOException {
20
21          //to recive data
22          DatagramSocket ds = new DatagramSocket(9999);
23
24          //bite array for the recived data
25          byte [] b1 = new byte[1024];
26
27          DatagramPacket dp = new DatagramPacket(b1, b1.length);
28          ds.receive(dp);
29          //to fetch the data
30          String str = new String (dp.getData(),0,dp.getLength());
31          //to perform operation
32          int num = Integer.parseInt(str);
33          int result = num *num;
34
35
36          //send data back to client
37          byte [] b2 = String.valueOf(result).getBytes();
38
39          InetAddress ia = InetAddress.getLocalHost();
40
41          DatagramPacket dp1 = new DatagramPacket(b2,b2.length,ia,dp.getPort());
42          ds.send(dp1);
43
44      }
45
46  }
```

UDP

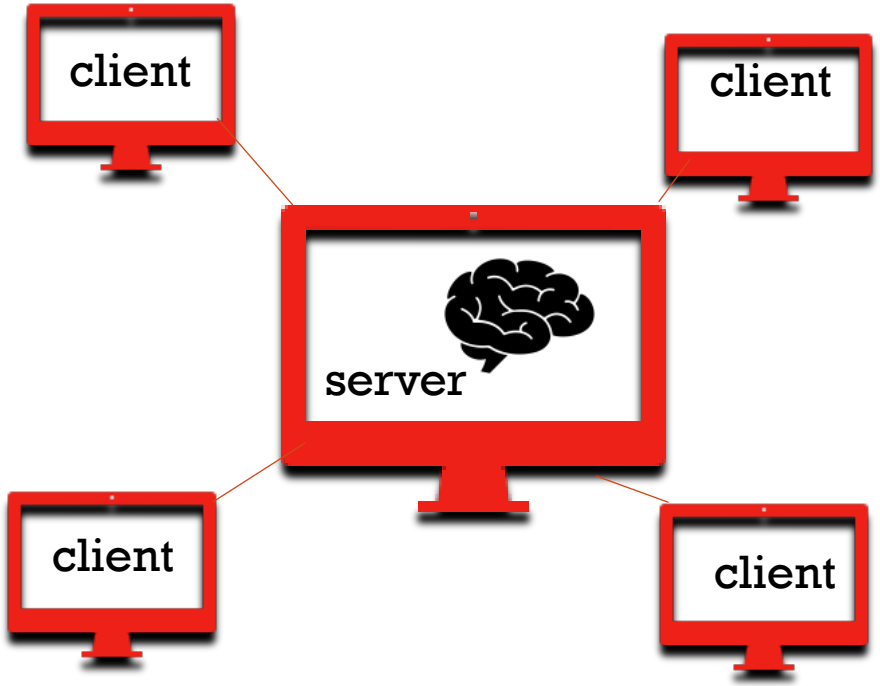
User datagram protocol

- For example of SimpleUDPExampleClient and SimpleUDPExampleServer

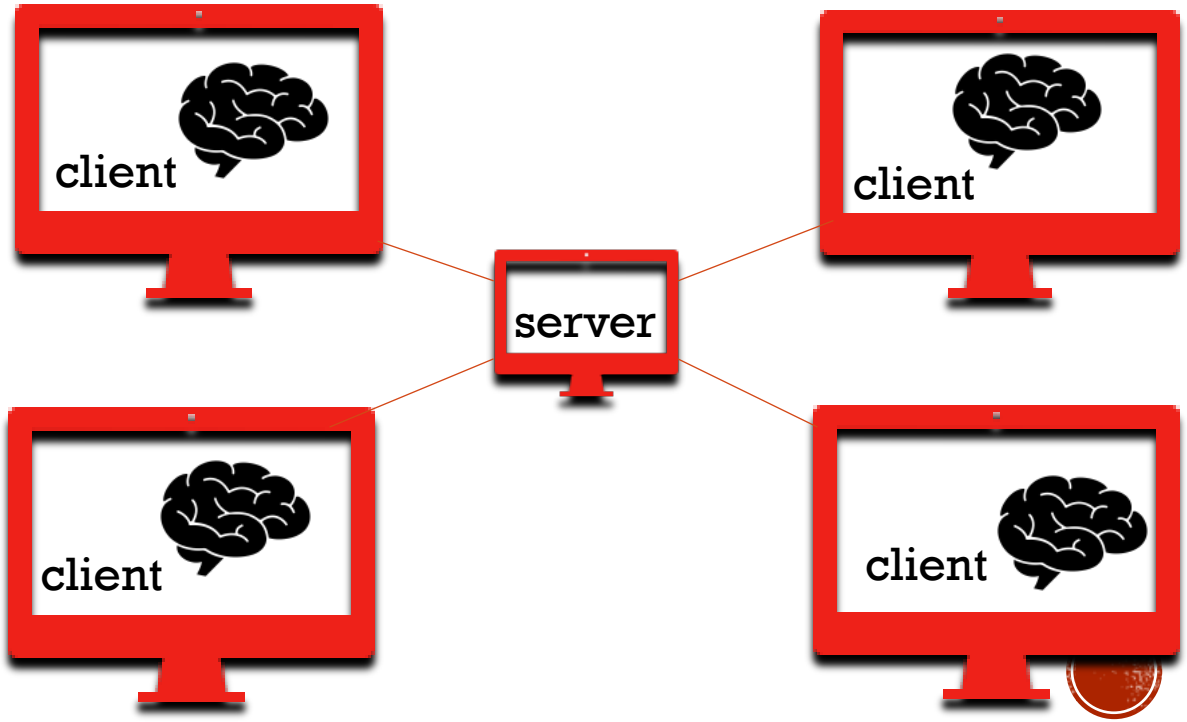
  Check related files

THIN AND FAT CLIENTS ARCHITECTURE

- **Thin** clients rely on a central computer for processing



- **Fat** clients do most of the processing themselves



MORE ABOUT CLIENT/SERVER