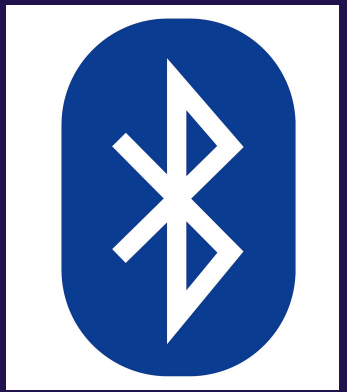
OOADI lecture 6
Networking

Federico Chiariotti - fchi@es.aau.dk
Jimmy Jessen Nielsen - jjn@es.aau.dk

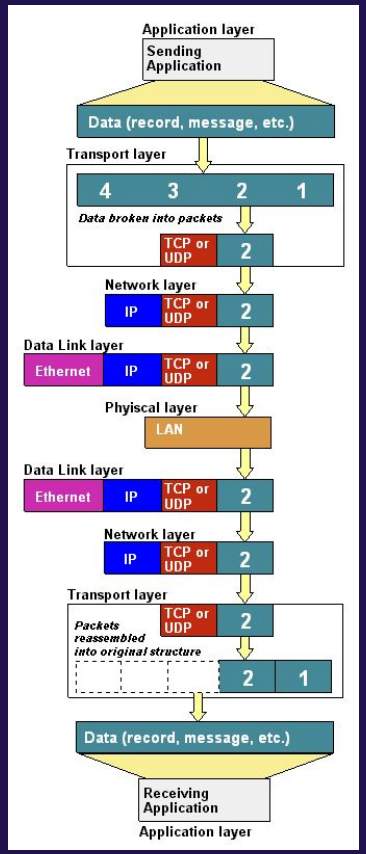
The communication problem

Data have to go through multiple links, using different technologies. How do we transmit without knowing all the relevant protocols?



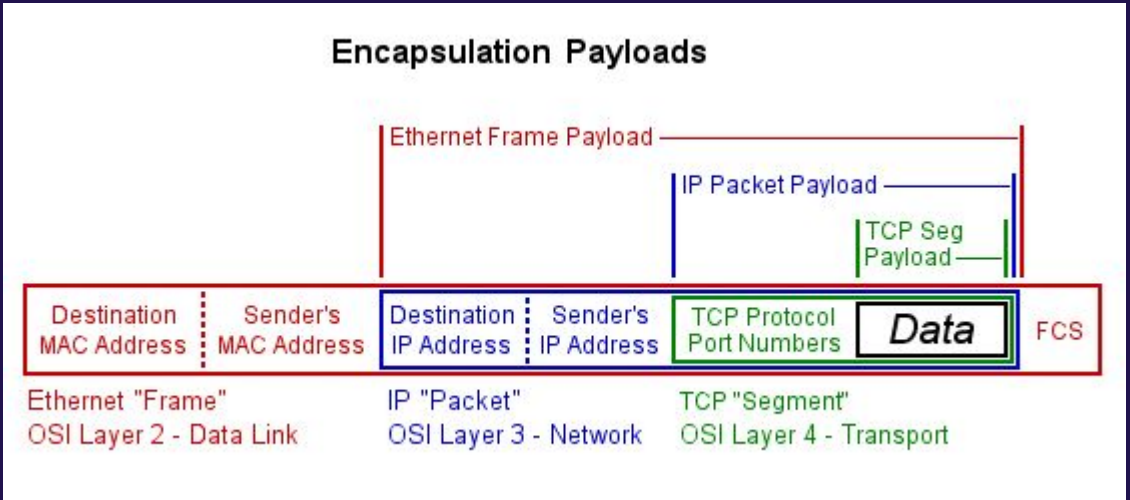
The networking stack

- Layer 1: Physical
- Layer 2 : Medium Access/Logical Link Control
- Layer 3: Network
- Layer 4: Transport
- Layer 5: Session
- Layer 6: Presentation
- Layer 7: Application



Encapsulated communications

Data from each layer is **encapsulated** as payload of the packet of the lower layer. The kernel handles layers 1-4



Sockets

Sockets hide away the details of the lower layers by presenting a pre-defined interface

What is a socket?

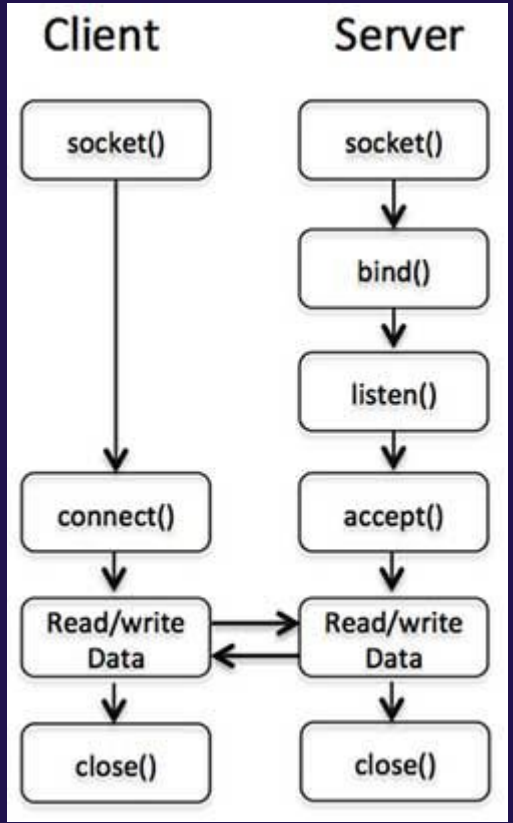
- Layer 4 interface (transport)
- The application can pass it data or messages, and it handles communication
- TCP, UDP, or raw IP (to implement your own protocol)
- Identified by IP and port of source and destination

The Berkeley model

- Created in 1983 for the BSD operating system, entered the UNIX specs
- Most programming languages use the Berkeley model
- Simple API between the kernel-space protocol stack and applications

Berkeley socket methods

- socket(): create the socket
- bind(): reserve the port
- listen(): announce willingness to accept connection requests
- connect(): try to connect to the given address
- accept(): block execution until connection
- send(): send data through the open socket
- receive(): receive data from the open socket
- close(): release the connection



Java: TCP and UDP sockets

TCP sockets are implemented in java.net.Socket:

- Connection procedure is needed
- Reliable transmission
- Congestion control

UDP sockets are implemented in java.net.DatagramSocket:

- No connection needed (you can just send and receive packets)
- Unreliable transmission
- No congestion control (you need to limit send rate!)

The ServerSocket class

The java.net.ServerSocket class allows you to create a server that can deal with multiple clients at once

- On accept(), the server creates a new socket to handle the connection
- If you use threading, the server can handle multiple connections
- The thread ends when the connection is closed

Example: echo server

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

public class EchoServer {
       public static void main(String[] args) throws IOException {
              if (args.length != 1) {
                     System.err.println("Usage: java EchoServer <port number>");
                     System.exit(1);
              }
              int portNumber = Integer.parseInt(args[0]);
              try {
                     ServerSocket serverSocket = new ServerSocket(Integer.parseInt(args[0]));
                     Socket clientSocket = serverSocket.accept();
                     PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                    BufferedReader in = new BufferedReader(new  InputStreamReader(clientSocket.getInputStream()));
                     String inputLine;
                     while ((inputLine = in.readLine()) != null) {
                     out.println(inputLine);
                     }
              } catch (IOException e) {
                     System.out.println("Exception caught when trying to listen on port "
                     + portNumber + " or listening for a connection");
                     System.out.println(e.getMessage());
              }
       }
}
```

Connection setup

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

public class EchoServer {
       public static void main(String[] args) throws IOException {
              if (args.length != 1) {
                     System.err.println("Usage: java EchoServer <port number>");
                     System.exit(1);
              }
              int portNumber = Integer.parseInt(args[0]);
              try {
                     ServerSocket serverSocket = new ServerSocket(Integer.parseInt(args[0]));
                     Socket clientSocket = serverSocket.accept();
                     PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                    BufferedReader in = new BufferedReader(new  InputStreamReader(clientSocket.getInputStream()));
                     String inputLine;
                     while ((inputLine = in.readLine()) != null) {
                     out.println(inputLine);
                     }
              } catch (IOException e) {
                     System.out.println("Exception caught when trying to listen on port "
                     + portNumber + " or listening for a connection");
                     System.out.println(e.getMessage());
              }
       }
}
```

Input and output streams

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

public class EchoServer {
       public static void main(String[] args) throws IOException {
              if (args.length != 1) {
                     System.err.println("Usage: java EchoServer <port number>");
                     System.exit(1);
              }
              int portNumber = Integer.parseInt(args[0]);
              try {
                     ServerSocket serverSocket = new ServerSocket(Integer.parseInt(args[0]));
                     Socket clientSocket = serverSocket.accept();
                     PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                    BufferedReader in = new BufferedReader(new  InputStreamReader(clientSocket.getInputStream()));
                     String inputLine;
                     while ((inputLine = in.readLine()) != null) {
                     out.println(inputLine);
                     }
              } catch (IOException e) {
                     System.out.println("Exception caught when trying to listen on port "
                     + portNumber + " or listening for a connection");
                     System.out.println(e.getMessage());
              }
       }
}
```

Example: echo client

```
import java.io.*;
import java.net.*;
public class EchoClient {
	public static void main(String[] args) throws IOException {
		if (args.length != 2) {
			System.err.println("Usage: java EchoClient <host name> <port number>"); System.exit(1);
		}
		String hostName = args[0];
		int portNumber = Integer.parseInt(args[1]);
		try {
			Socket echoSocket = new Socket(hostName, portNumber);
			PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
			BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
			BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
			String userInput;
			while ((userInput = stdIn.readLine()) != null) {
				out.println(userInput);
				System.out.println("echo: " + in.readLine());
			}
		} catch (UnknownHostException e) {
			System.err.println("Don't know about host " + hostName); System.exit(1);
		} catch (IOException e) {
			System.err.println("Couldn't get I/O for the connection to " + hostName); System.exit(1);
		}
	}
}
```

UDP datagrams

UDP is not connection-oriented: datagrams are exchanged without any contrl packets

- The socket does not need to be bound
- The packet needs to be pre-formed (input and output are not streams)

Example: quote client

```
import java.io.*;
import java.net.*;
import java.util.*;

public class QuoteClient {
       public static void main(String[] args) throws IOException {
              if (args.length != 1) {
                     System.out.println("Usage: java QuoteClient <hostname>");
                     return;
              }
              // get a datagram socket
              DatagramSocket socket = new DatagramSocket();
              // send request
              byte[] buf = new byte[256];
              InetAddress address = InetAddress.getByName(args[0]);
              DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
              socket.send(packet);
              // get response
              packet = new DatagramPacket(buf, buf.length);
              socket.receive(packet);
              // display response
              String received = new String(packet.getData(), 0, packet.getLength());
              System.out.println("Quote of the Moment: " + received);

              socket.close();
       }
}
```

Example: quote server

```
import java.io.*;
import java.net.*;
import java.util.*;
public class QuoteServerThread extends Thread {
        protected DatagramSocket socket = null;
        protected BufferedReader in = null;
        protected boolean moreQuotes = true;

        public QuoteServerThread() throws IOException {
                this("QuoteServerThread");
        }

        public QuoteServerThread(String name) throws IOException {
                super(name);
                socket = new DatagramSocket(4445);
                try {
                        in = new BufferedReader(new FileReader("one-liners.txt"));
                } catch (FileNotFoundException e) {
                        System.err.println("Could not open quote file. Serving time instead.");
                }
        }
        ...
```

Example: quote server

```
public void run() {
        while (moreQuotes) {
                try {
                        byte[] buf = new byte[256];
                        // receive request
                        DatagramPacket packet = new DatagramPacket(buf, buf.length);
                        socket.receive(packet);
                        // figure out response
                        String dString = null;
                        if (in == null)
                                dString = new Date().toString();
                        else
                                dString = getNextQuote();
                        buf = dString.getBytes();
                        // send the response to the client at "address" and "port"
                        InetAddress address = packet.getAddress();
                        int port = packet.getPort();
                        packet = new DatagramPacket(buf, buf.length, address, port);
                        socket.send(packet);
                        } catch (IOException e) {
                                e.printStackTrace();
                                moreQuotes = false;
                        }
                }
        socket.close();
}
```

Example: quote server

```
        protected String getNextQuote() {
                String returnValue = null;
                try {
                        if ((returnValue = in.readLine()) == null) {
                                in.close();
                                moreQuotes = false;
                                returnValue = "No more quotes. Goodbye.";
                        }
                } catch (IOException e) {
                        returnValue = "IOException occurred in server.";
                }
                return returnValue;
        }
}
```

```
import java.io.*;

public class QuoteServer {
        public static void main(String[] args) throws IOException {
                new QuoteServerThread().start();
        }
}
```

Exercises

1. Change the EchoServer class so that it can support multiple clients
2. Write a simple chat application in which client and server can both write and receive messages
3. Write the same class using UDP
4. Write a UDP echo server that can listen on multiple ports
5. Write a file transfer application: if the client connects, the server sends it a given file through the socket and the client buffers it and saves it.