CHAPTER 2
APPLICATION LAYER
SOCKET PROGRAMMING

Abdullah Yasin Nur

CHAPTER 2: OUTLINE

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

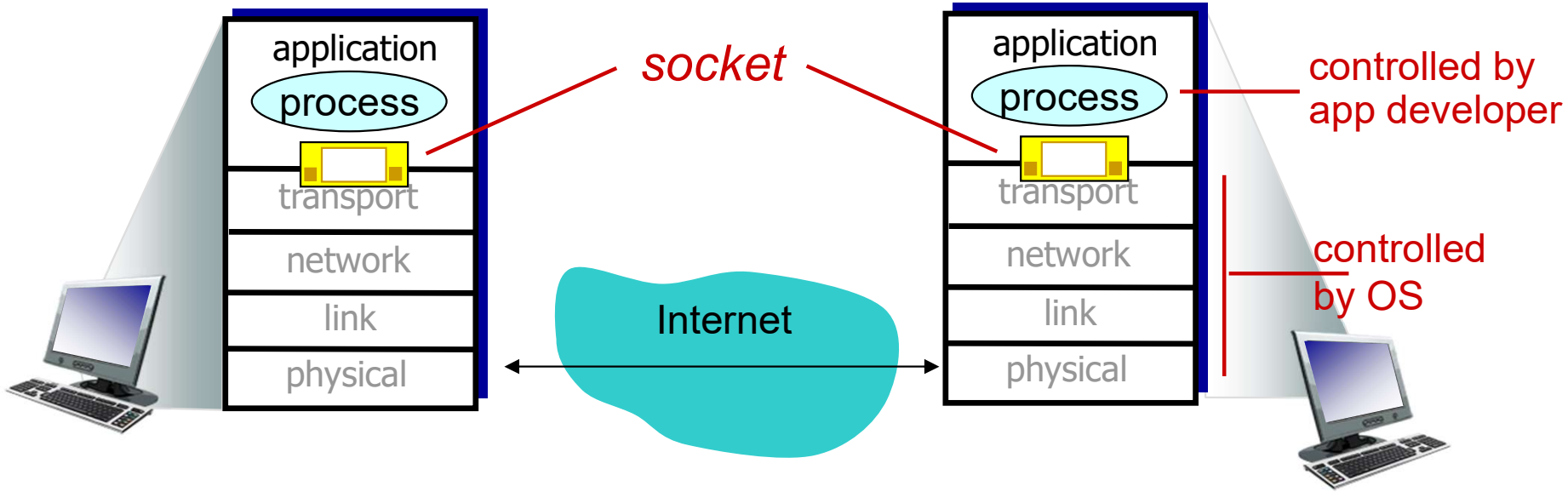
2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

SOCKET PROGRAMMING

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



SOCKET PROGRAMMING

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

SOCKET PROGRAMMING *WITH TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

JAVA SOCKET PROGRAMMING

- Java networking package (java.net)
- Establishing Socket Connections with TCP
  - Socket serverSct = new ServerSocket(port); //Server side
    - Argument: TCP Port - Port number can be from 0 to 65535)
  - Socket clientSct = new Socket(ServerAddress, port); //Client Side
    - Argument: ServerAddress is the IP address of the server.
    - If you run your code in your computer and not sharing your application with outside of your network, you can use "localhost"
- java.io package
  - Gives us input and output streams to write to and read from while communicating

SOCKET PROGRAMMING EXAMPLE

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and prints the data
3. The application terminates when the client sends "over" message to the server

Server Side

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

public class MyServer {

    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;

    public MyServer(int port) {
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");
            System.out.println("Waiting client");
            socket = server.accept();
            System.out.println("Client accepted");

            in = new DataInputStream(
                    new BufferedInputStream(socket.getInputStream()));

            String line = "";
            while (!line.equals("over")) {
                try {
                    line = in.readUTF();
                    System.out.println(line);
                } catch (IOException i) {
                    System.out.println("Error " + i.getMessage());
                }
            }

            System.out.println("Connection closed");
            socket.close();
            in.close();
        } catch (Exception e) {
            System.out.println("Error here " + e.getMessage());
        }

    }

    public static void main(String args[])
    {
        MyServer server = new MyServer(5000);

    }
}
```

Server Side

```
try {
    server = new ServerSocket(port);
    System.out.println("Server started");
    System.out.println("Waiting client");
    socket = server.accept();
    System.out.println("Client accepted");
```

Line 1 - Create a server socket with a port number

- Client will use this port number to connect to the server

Line 4 – Server listens the port, if a client sends a request, server will use server.accept() part to accept the client

Server Side

```
in = new DataInputStream(
        new BufferedInputStream(socket.getInputStream()));

String line = "";
while (!line.equals("over")) {
    try {
        line = in.readUTF();
        System.out.println(line);
    } catch (IOException i) {
        System.out.println("Error " + i.getMessage());
    }
}
```

- We use input stream for communication
- While loop continues until the client sends “over” message
- line = in.readUTF() is the value received from the client

Server Side

```
            System.out.println("Connection closed");
            socket.close();
            in.close();
        } catch (Exception e) {
            System.out.println("Error here " + e.getMessage());
        }

}
```

If the user enters “over”, server’s job is done. Server close the socket and input stream before termination.

Server Side

```
    public static void main(String args[])
    {
        MyServer server = new MyServer(5000);

    }
}
```

Main function to run the server. It takes port number as an argument.

Client Side

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

public class MyClient {

    private Socket socket = null;
    private DataInputStream input = null;
    private DataOutputStream out = null;

    public MyClient(String address, int port) {
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");
            input = new DataInputStream(System.in);
            out = new DataOutputStream(socket.getOutputStream());
        } catch (Exception e) {
            System.out.println("error " + e.getMessage());
        }

        String line = "";
        while (!line.equals("over")) {
            try {
                line = input.readLine();
                out.writeUTF(line);
            } catch (IOException i) {
                System.out.println(i);
            }
        }
        try {
            input.close();
            out.close();
            socket.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        MyClient client = new MyClient("localhost", 5000);
    }
}
```

Client Side

```
public MyClient(String address, int port) {
    try {
        socket = new Socket(address, port);
        System.out.println("Connected");
        input = new DataInputStream(System.in);
        out = new DataOutputStream(socket.getOutputStream());
    } catch (Exception e) {
        System.out.println("error " + e.getMessage());
    }
```

- Client connects to the server over given address and port number.
- The address is server's IP address
- In client side, we need to create input stream and output stream
  - To get info from the user
  - To communicate with the server

Client Side

```
        String line = "";
        while (!line.equals("over")) {
            try {
                line = input.readLine();
                out.writeUTF(line);
            } catch (IOException i) {
                System.out.println(i);
            }
        }
        try {
            input.close();
            out.close();
            socket.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }
```

- Read from user until the user types over
- out.writeUTF will send the message to the server.
  - Remember, server uses readUTF code to listen

Client Side

```
    public static void main(String args[])
    {
        MyClient client = new MyClient("localhost", 5000);
    }
}
```

- Main function for client
- Arguments:
  - Server IP address
    - If your client and your server is in the same network, use “localhost.
    - Otherwise, give full IP address of the server,

      e.g. "127.0.0.1"
  - Server port number