COMP 4621 Tutorial #2

Spring 2015

Outline

- HTTP: Concept & Programming
- TCP: Concept & Programming
- Practice & Q&A

Outline

- HTTP: Concept & Programming
- TCP: Concept & Programming
- Practice & Q&A

A Question

How to kill time in a boring tutorial?

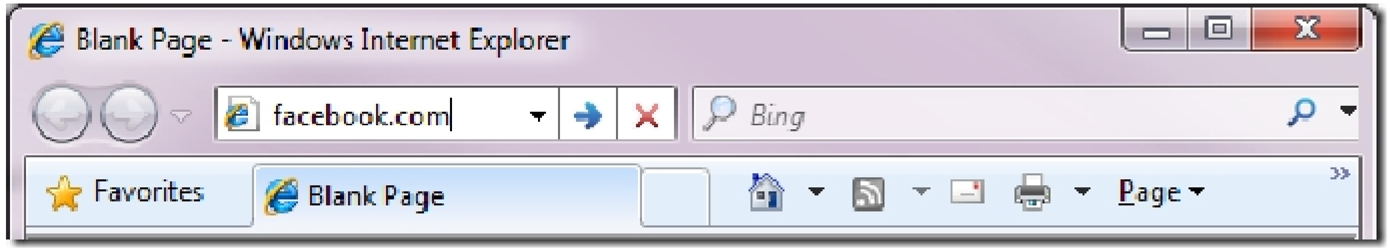
Time for internet surfing!

Before you really start to do it…

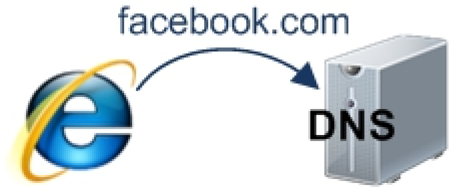
What really happens when you open a website, say, Facebook?

What really happens…

1. You enter a URL into the browser



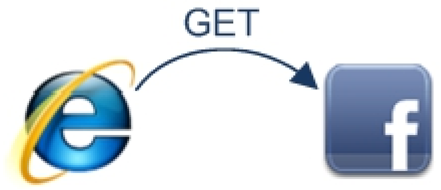
2. The browser looks up the IP address for the domain name

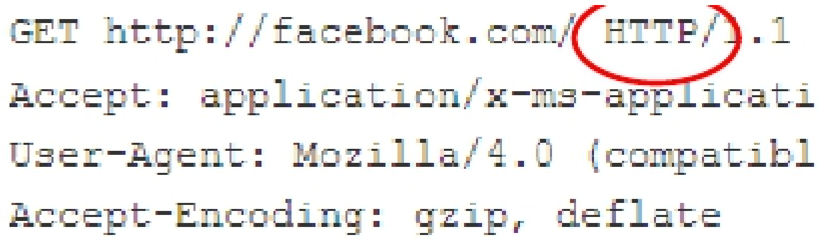


IP: 208.87.149.250

What really happens… (Cont.)

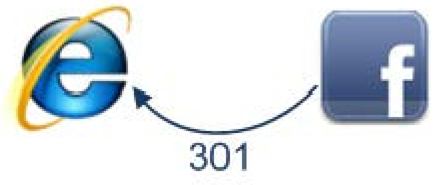
3. The browser sends a HTTP request to the web server

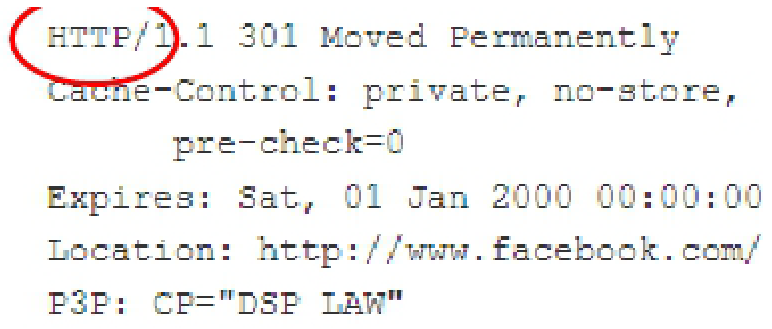




```
GET http://facebook.com/ HTTP/1.1
Accept: application/x-ms-applicati
User-Agent: Mozilla/4.0 (compatibl
Accept-Encoding: gzip, deflate
```

4. The Facebook server responds with a permanent redirect,why?

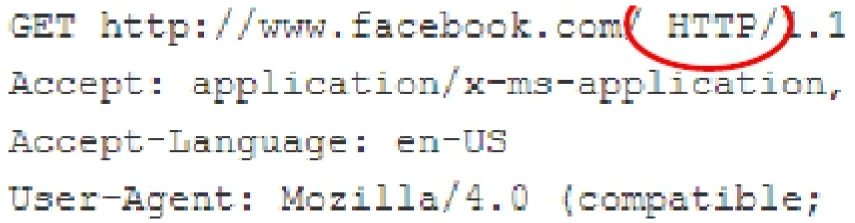




```
HTTP/1.1 301 Moved Permanently
Cache-Control: private, no-store,
       pre-check=0
Expires: Sat, 01 Jan 2000 00:00:00
Location: http://www.facebook.com/
P3P: CP="DSP LAW"
```

What really happens… (Cont.)

5. The browser follows the redirect

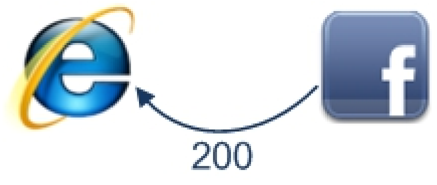


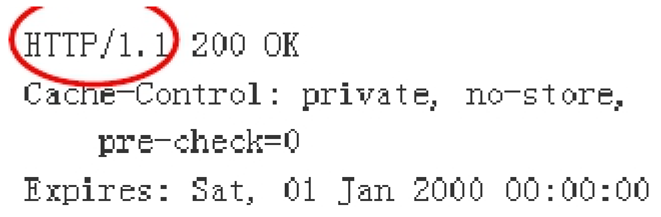
```
GET http://www.facebook.com/ HTTP/1.1
Accept: application/x-ms-application,
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible;
```

- 6. The server “handles” the request

What really happens… (Cont.)

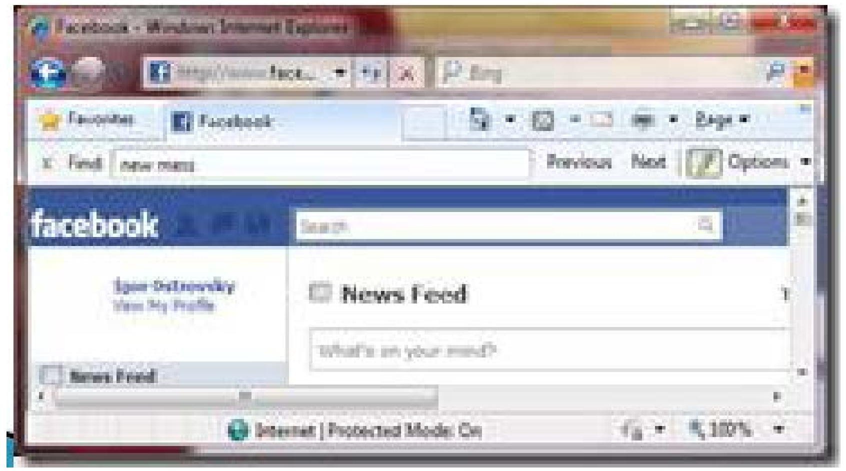
7. The server sends back a HTML response

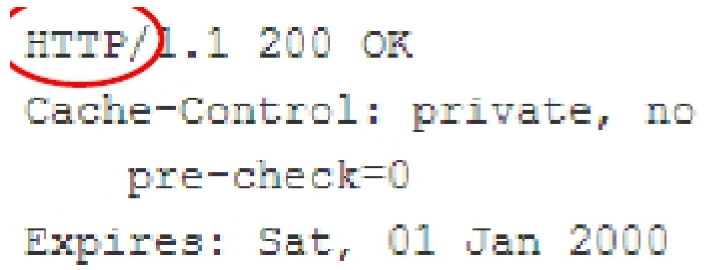




```
HTTP/1.1 200 OK
Cache-Control: private, no-store,
    pre-check=0
Expires: Sat, 01 Jan 2000 00:00:00
```

8. The browser begins **rendering** the HTML

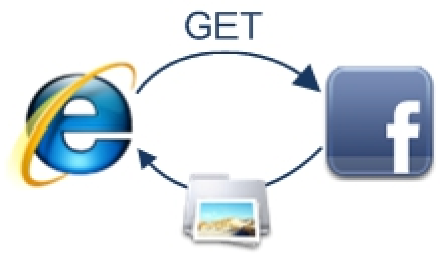


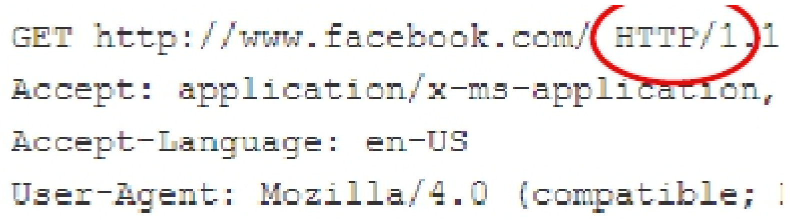


```
HTTP/1.1 200 OK
Cache-Control: private, no
    pre-check=0
Expires: Sat, 01 Jan 2000
```

What really happens… (Cont.)

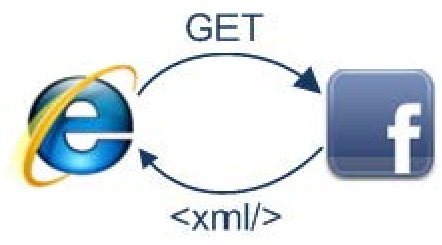
9. The browser sends requests for objects embedded in HTML

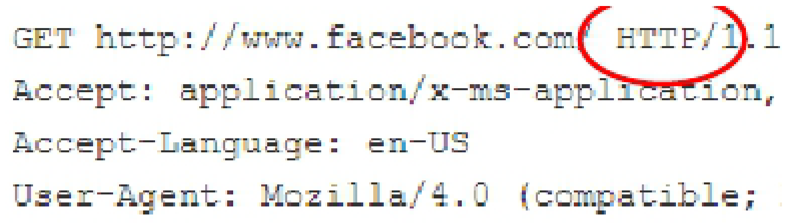




```
GET http://www.facebook.com/ HTTP/1.1
Accept: application/x-ms-application,
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible;
```

10. The browser sends further asynchronous (AJAX)





```
GET http://www.facebook.com/ HTTP/1.1
Accept: application/x-ms-application,
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible;
```

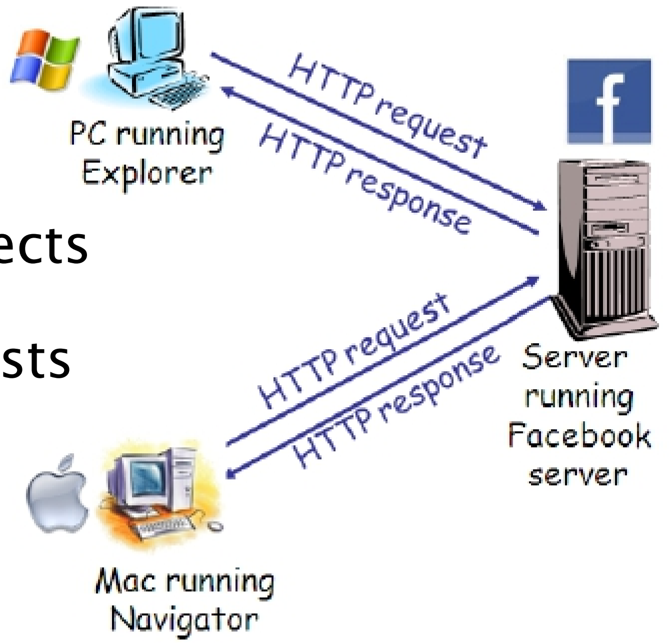
What really happens…

- 8/10 steps make use of HTTP.
- HTTP is important, especially when you want to kill time during the lab.

- Discover more details on http://igoro.com/archive/what-really-happens-when-you-navigate-to-a-url/

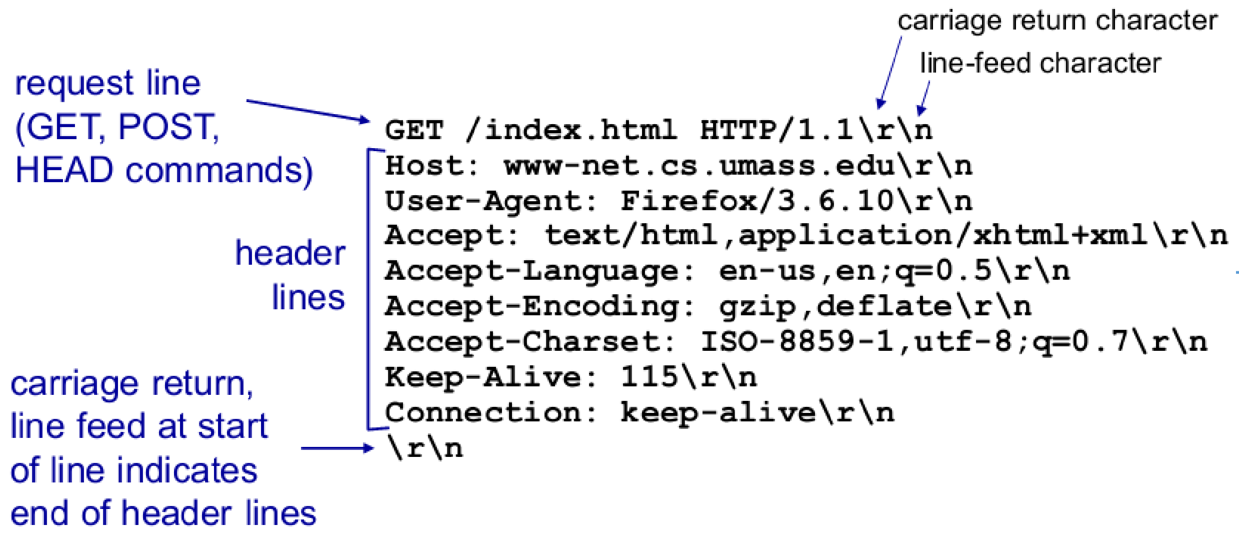
HTTP Overview

- HTTP: Hyper-Text Transfer Protocol
- Web's application layer protocol
- TCP-based, Client/server model
  - Client: browser that requests, receives, "displays" Web objects
  - Server: web server that sends objects in response to requests
- Status code
  - 404 – Not Found
  - 301 – Move permanently



HTTP Programming: Example

- In this example, we will:
  1. establishes a **TCP** connection with a standard HTTP server
  2. sends **HTTP request** to the server
  3. downloads the webpage to the user-end



carriage return character
line-feed character
request line (GET, POST, HEAD commands)
header lines
carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

```
GET / HTTP/1.1\r\n
Host: course.cse.ust.hk\r\n
```

HTTP Programming: Example

```
package lab2;
import java.io.*;
import java.net.*;
public class HttpClient {
		public static void main(String argv[]) throws Exception {
			// input url
			String urlWebPage;
			BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in) );
			System.out.println("Please input the URL of the webpage:");
			urlWebPage = inFromUser.readLine();

			// create socket
			System.out.println();
			Socket clientSocket = new Socket(urlWebPage, 80);
```

Read user input from keyboard

Create TCP socket

What is the difference with UDP socket?

HTTP Programming: Example

Prepare input & output stream

```
// prepare input/output stream
DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream() );
DataInputStream inFromServer = new DataInputStream(clientSocket.getInputStream() );

// attach output stream to a local file
File htmlFile = new File("index.html");
DataOutputStream outToFile = new DataOutputStream(new FileOutputStream(htmlFile) );
```

Sent HTTP req to server

```
// write HTTP req
outToServer.writeBytes("GET / HTTP/1.1\n");
outToServer.writeBytes("Host: " + urlWebPage + "\n");
outToServer.writeBytes("\n");
```

HTTP Programming: Example

Read from svr
Write to disk
Until ?

Clear up

```
// prepare to buffer for recv
int recv = 0;
int bufferSize = 1024;
byte[] buffer = new byte[bufferSize];
// recv from svr
while ( (recv = inFromServer.read(buffer, 0, bufferSize)) != -1) {
        outToFile.write(buffer, 0, recv); // write to file
}
// flush and close stream
outToFile.flush();
outToFile.close();
//close socket
clientSocket.close();
System.out.println("The web page has been downloaded as index.html.");
}}
```

HTTP Programming: Practice

- run this example by yourself to download a webpage to your local disk. See what is included in that response
  - mkdir lab2, create HttpClient.java
  - javac HttpClient.java // compile the java file;
  - java HttpClient // run the client program;

- Try “facebook.com” and “course.cse.ust.hk”, what is the difference?

Outline

- HTTP: Concept & Programming
- TCP: Concept & Programming
- Practice & Q&A

Internet Protocol Stack

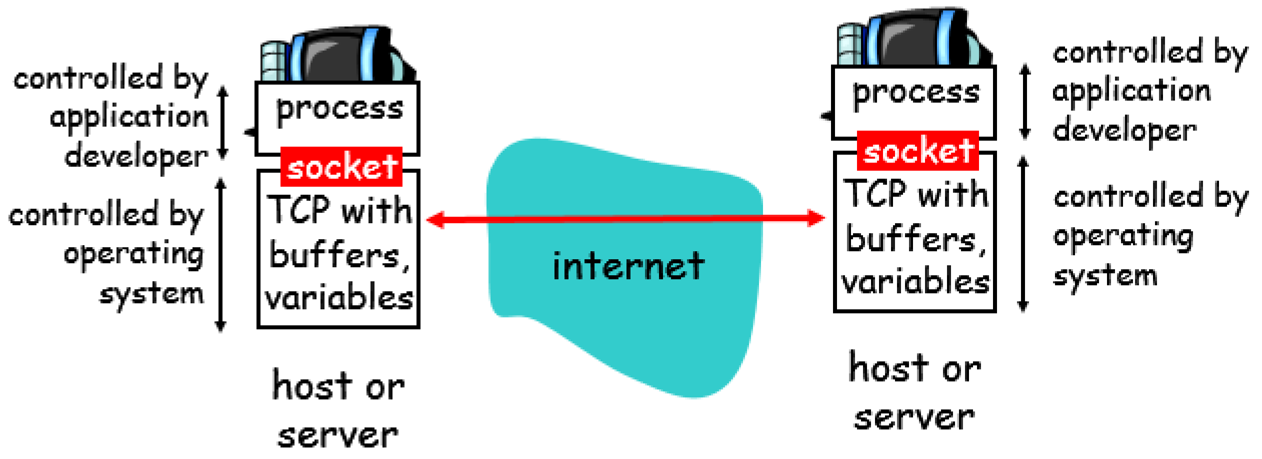
Application	HTTP, FTP, SMTP
Transport	TCP, UDP
Network	IP
Link	Ethernet, PPP
Physical	Bits “on the wire”

TCP Overview

- TCP: Transmission Control Protocol
- Point-to-point: One sender, one receiver
- Connection-oriented, reliable, in-order byte-stream
  - handshake
  - Congestion control
  - Flow control
  - Re-transmission
- Lots of applications rely on TCP: HTTP, FTP, SMTP,…

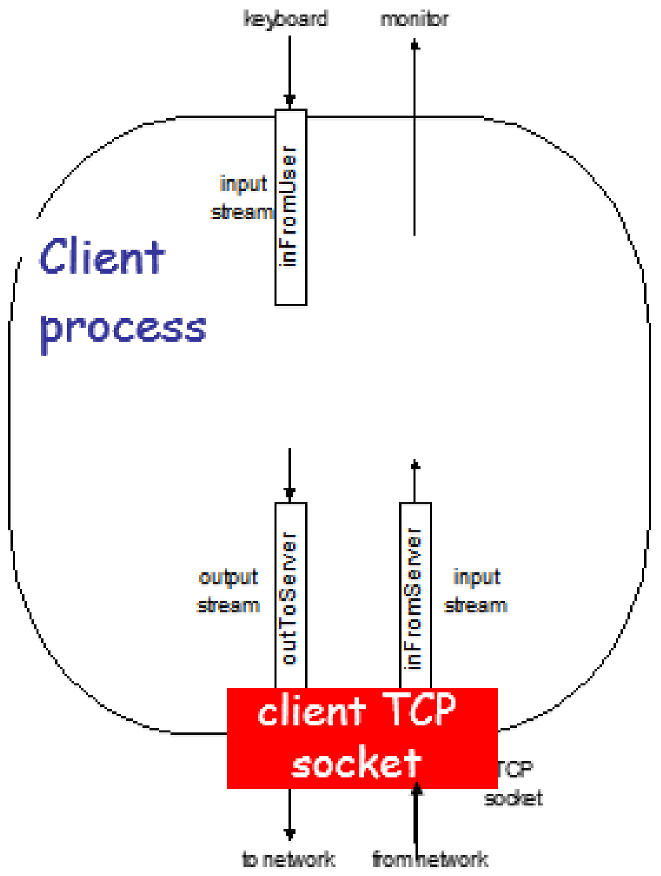
TCP Programming: Socket

- TCP: reliable transfer **Protocol** of bytes from one process to another.
- Network Socket: an **endpoint** of an inter-process communication flow across a computer network.
- Socket API: an **API** provided by the operating system, that allows application programs to control and use network sockets.

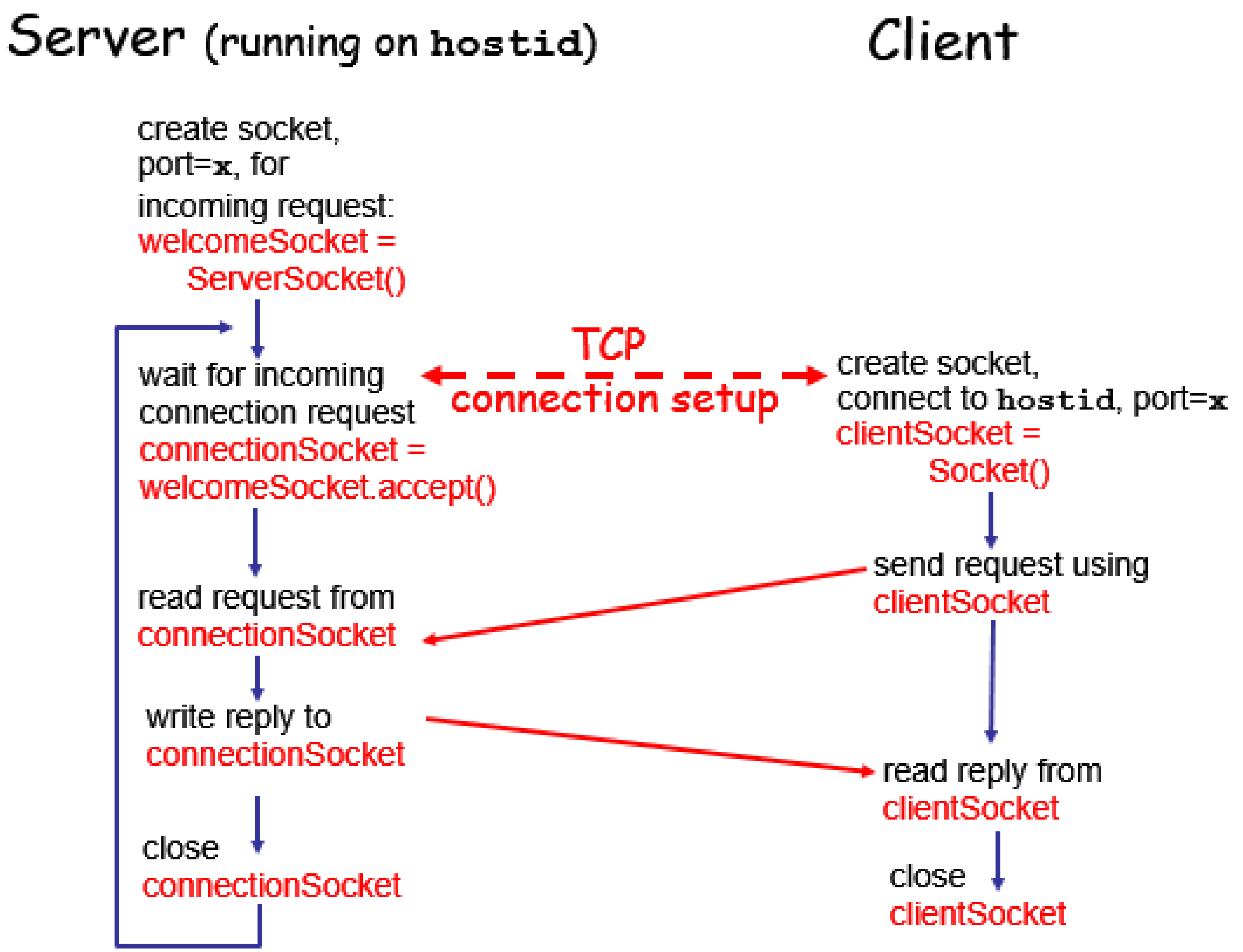

controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

TCP Programming: Stream

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



TCP Programming: Steps


Server (running on hostid)
Client
create socket, port=x, for incoming request:
welcomeSocket = ServerSocket()
TCP connection setup
wait for incoming connection request
connectionSocket = welcomeSocket.accept()
create socket, connect to hostid, port=x
clientSocket = Socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

TCP Programming: Example 1

- In this example, we will:

1) client reads line from standard input, sends to server via socket.
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket

TCP Programming: Example 1 (Server)

```
package lab2;
import java.io.*;
import java.net.*;

public class TCPServer_msg {
		public static void main(String argv[]) throws Exception {

				String clientSentence;
				String capitalizedSentence;

				// create svr socket on port 6789
				int nPort = 6789;
				ServerSocket welcomeSocket = new ServerSocket(nPort);
```

Create a Server socket, listen on port 6789

TCP Programming: Example 1 (Server)

Keep listening on port 6789

Prepare input & output stream

Read from client, To upper case, Send back to client

```
while (true) {
            System.out.println("msg svr is listening on"+nPort+"...");
            // block until a new connection is accepted
            Socket connectionSocket = welcomeSocket.accept();
            // get input/output stream
            BufferedReader inFromClient = new BufferedReader( new InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new DataOutputStream( connectionSocket.getOutputStream());
            // read from client
            clientSentence = inFromClient.readLine();
            System.out.println("receive from client: " + clientSentence);
            // change to upper case
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            // write back to client
            outToClient.writeBytes(capitalizedSentence);
}}}
```

TCP Programming: Example 1 (Client)

```
package lab2;
import java.io.*;
import java.net.*;

public class TCPClient_msg {
        public static void main(String argv[]) throws Exception {
                String sentence;
                String modifiedSentence;

                // prepare input stream from keyboard
                BufferedReader inFromUser = new BufferedReader(new InputStreamReader( System.in));

                // create sockets
                Socket clientSocket = new Socket("localhost", 6789);
```

Prepare input stream from keyboard

Create TCP socket

TCP Programming: Example 1 (Client)

Prepare input & output stream

Read from user input, Send to svr, Then wait to response

```
// get input/output stream
DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream() ) );
// read from standard input
sentence = inFromUser.readLine();
// send to svr
outToServer.writeBytes(sentence + '\n');
// block until receive from server
modifiedSentence = inFromServer.readLine();
// print out received msg
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
} }
```

TCP Programming: Practice 1

- Try this example on your machine
    - Note: svr need to be started before client try to connect to it

- Questions:
    - What is the difference btw this example and UDP example we did in last lab?
    - Can multiple clients connect to this server simultaneously?
        - If yes, how to identify each connection?

TCP Programming: Example 2

- In this example, we will:
  - creates a file and input some content
  - start the TCPClient and input the file’s name
  - transfer the content of the file to the server and save it in a file.

TCP Programming: Example 2 (Server)

```
package lab2;
import java.io.*;
import java.net.*;

public class TCPServer_file {
		public static void main(String[] argv) throws Exception {
				// create svr socket on specified port
				int PORT = 9876;
				ServerSocket serverSocket = new ServerSocket(PORT);
				System.out.println("Server is running.....");

				// client socket to recv
				Socket socket = null;
```

Create a server socket on port 9876

TCP Programming: Example 2 (Server)

Block until a connection is established

Read 1st line from client, try to parse user name and file name from it

```
while (true) {
        //block until new connection is accepted
        socket = serverSocket.accept();
        // prepare buff to recv
        int recv = 0;
        int bufferSize = 1024;
        byte[] buffer = new byte[bufferSize];
        // create input stream from accepted socket
        DataInputStream fileStream = new DataInputStream( new BufferedInputStream(socket.getInputStream()));
        // get user name and file name from 1st line
        String fileMsg = fileStream.readLine();
        int userNameIndex = fileMsg.indexOf(':');
        String userName = fileMsg.substring(0, userNameIndex);
        String fileName = fileMsg.substring(userNameIndex + 1);
        System.out.println(fileName);
        System.out.println("New connection accepted from: " + username + "," + socket.getInetAddress() + ":" + socket.getPort());
```

TCP Programming: Example 2 (Server)

Create a file and attach it to an output stream

While read from client, write to file

Clear up & close

```
// create new file to write(filename: username_port_filename)
File recvFile = new File(userName + "_" + socket.getPort() + "_" + fileName);
// attach this file to an output stream
DataOutputStream outputFile = new DataOutputStream( new FileOutputStream(recvFile));
// continue to read from input stream until end
while ((recv = fileStream.read(buffer, 0, bufferSize)) != -1) {
            outputFile.write(buffer, 0, recv); // write to file
}
// flush and close streams
outputFile.flush();
outputFile.close();
socket.close();

System.out.println("File " + fileName + " from user "  + userName + " has been transfered!");
} } }
```

TCP Programming: Example 2 (Client)

```
package lab2;
import java.io.*;
import java.net.*;

public class TCPClient_file {
        public static void main(String argv[]) throws Exception {
                // create input stream from standard input
                BufferedReader inFromUser = new BufferedReader(new InputStreamReader( System.in));
                System.out.println("Input the ip you want to send to:");
                // get ip address
                String ip = inFromUser.readLine();
                InetAddress IPAddress = InetAddress.getByName(ip);
                System.out.println("Connecting to " + IPAddress.toString());

                // create a socket
                int nPort = 9876;
                Socket clientSocket = new Socket(ip, nPort);
```

Read ip from user input

Create a socket with dst port 9876

TCP Programming: Example 2 (Client)

Get user name & file to transmit

```
// get system user name
String username = System.getProperty("user.name");

// get file to transfer
System.out.println("Input the name of the file you want to transfer:");
String fileName = inFromUser.readLine();
```

Prepare input & output stream

```
// attach file stream to socket output stream
DataOutputStream outToServer = new DataOutputStream(
                    new BufferedOutputStream(clientSocket.getOutputStream()));
DataInputStream fileInputStream = new DataInputStream(
                    new BufferedInputStream(new FileInputStream(fileName)));
```

Send user name & file name to svr

```
// write 1st line(format=username:filename)
outToServer.writeBytes(username + ":" + fileName + "\n");
outToServer.flush(); // send to svr
```

TCP Programming: Example 2 (Client)

keep reading from file and send to svr until the end of file

Clear up

```
// read from file and send to svr
int bufferSize = 1024;
byte[] buffer = new byte[bufferSize];
int read = 0;
if (fileInputStream != null) {
        while ((read = fileInputStream.read(buffer)) != -1) {
                outToServer.write(buffer, 0, read);
        }
        // flush and close socket
        outToServer.flush();
        fileInputStream.close();
        clientSocket.close();
        System.out.println("The file " + fileName + " has been tranfered!");
} } }
```

TCP Programming: Practice 2

- Run the sample code by yourself
    - Compile with javac
    - Run with java
    - Note
        - The IP address of local machine is 127.0.0.1
        - Svr need to be started before client

Outline

- HTTP: Concept & Programming
- TCP: Concept & Programming
- Practice & Q&A

Download links

- https://www.dropbox.com/sh/lokjgr6xbf6filc/AAAZaqVNLVI19RD6LtYcrOAHa?dl=0