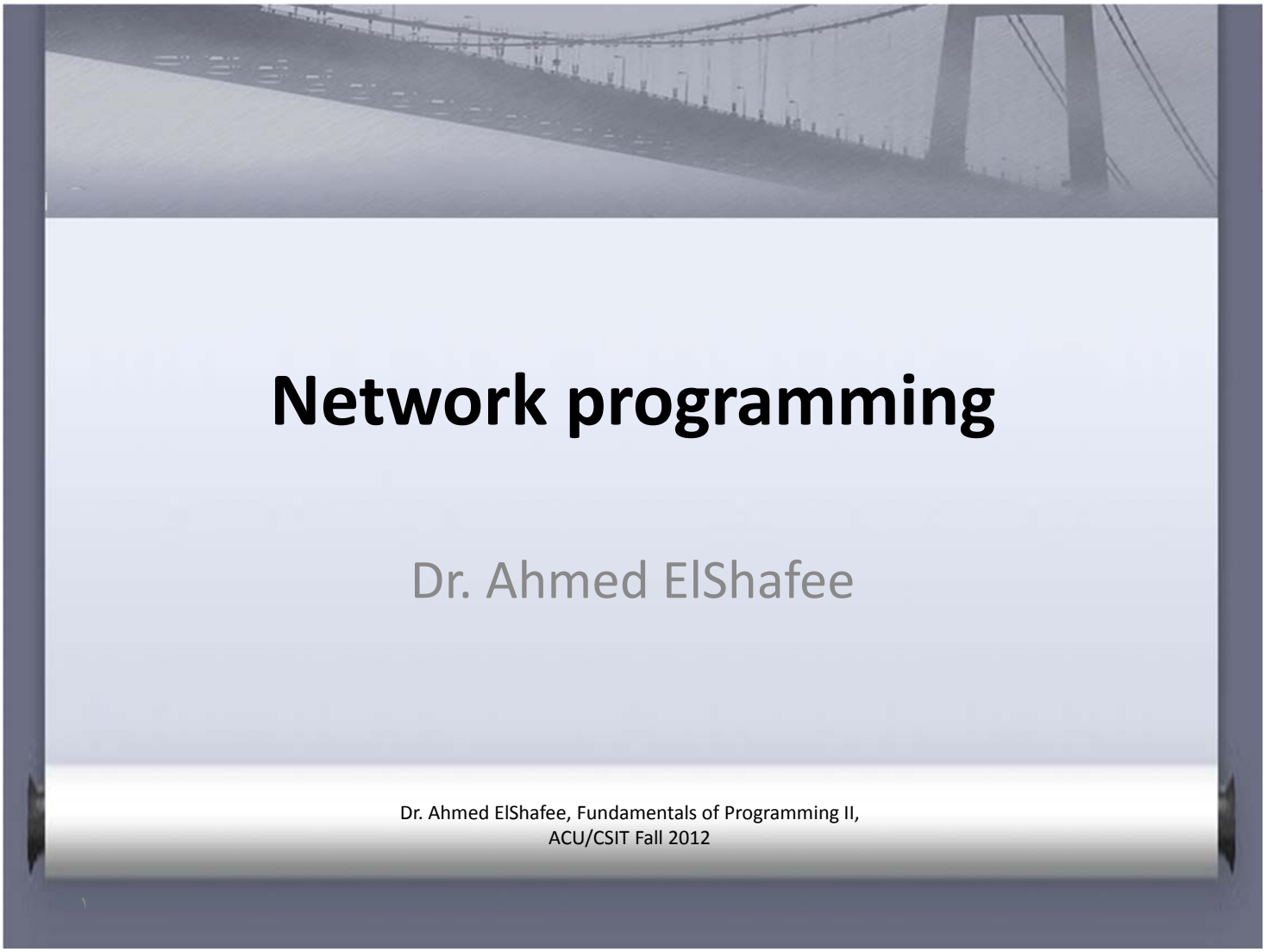


Network programming

Dr. Ahmed ElShafee

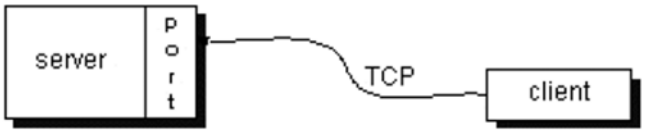
Agenda

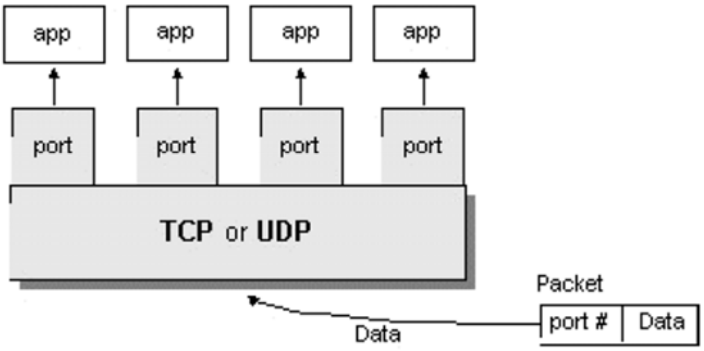
1. Client server programming
2. Email programming
3. Http protocol programming

Concepts of socket programming

Ports

- Ports are a virtual channels that enables applications to share the single network channel to exchange data.
- Port numbers are represented by 16-bit numbers. (0 to 65,535) The port numbers ranging from 0 - 1023 reserved for use by well known application
- services such as HTTP and FTP and other system services.





Sockets

You can reach required service via its network and port IDs. what then?

a) If you are a client

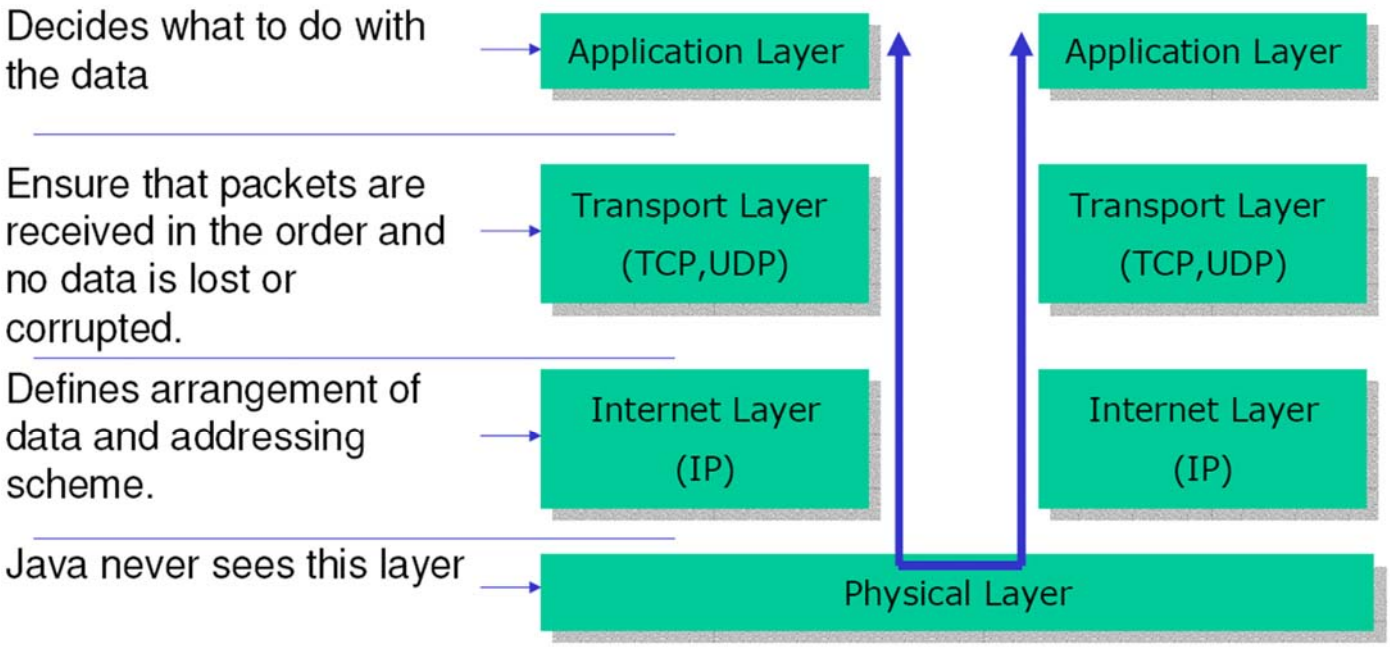
- you need an API that will allow you to send messages to that service and read replies from it

b) If you are a server

- you need to be able to create a port and listen at it.
- you need to be able to read the message comes in and reply to it.

The **Socket** and **ServerSocket** are the Java client and server classes to do this.

Network Layers

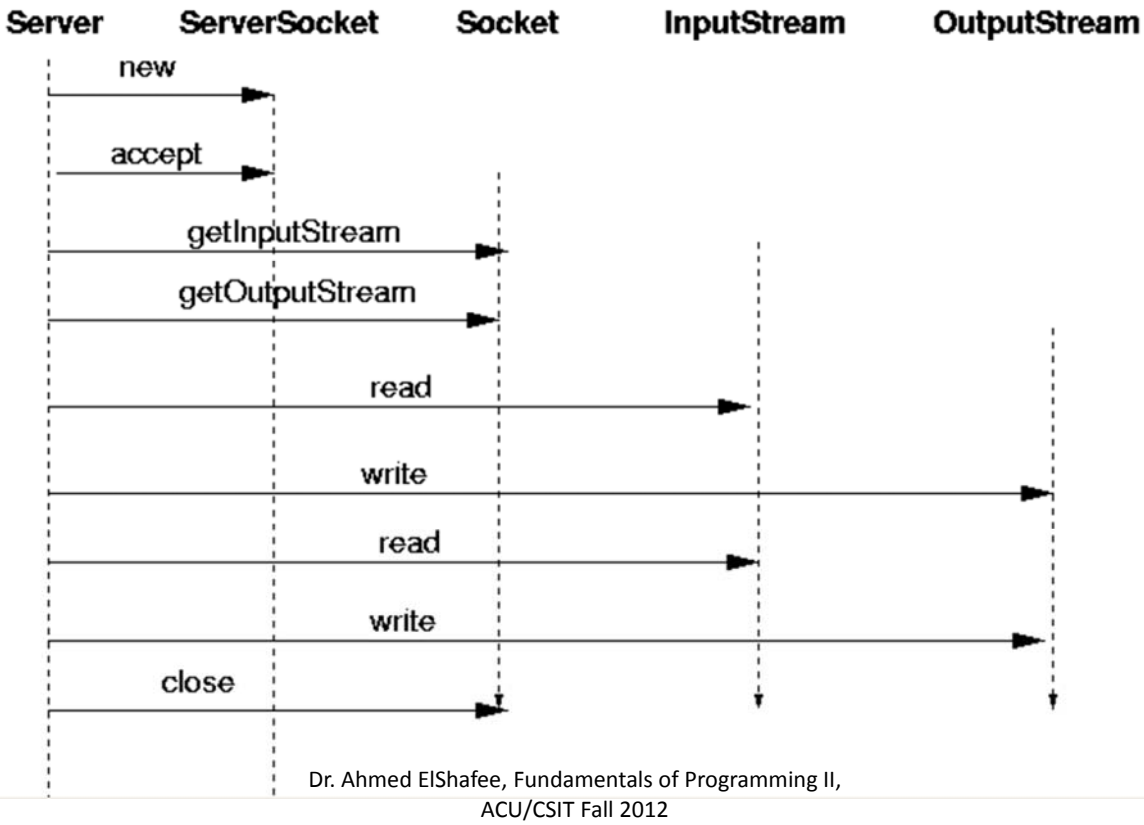


TCP and UDP

Java only supports TCP (Transmission Control Protocol), UDP (User Datagram Protocol) and application layer protocols built on top of these.

TCP/IP client/server

TCP Server



Simple Server example

function	code
Create server socket	s = new ServerSocket(int PORT);
Accepts incoming connection	Socket incoming = = s.accept();
Create reader object form accepted connection object as a data source	BufferedReader reader = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
Create print stream object to write data to socket as a data source	PrintStream out = new PrintStream(incoming.getOutputStream());
Read line from socket	String str = reader.readLine();
Write line to socket	out.println(str);

Simple client socket

function	code
Create remote socket	echoSocket = new Socket(IP, port);
Create print writer object to write chars to remote socket	PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
Create buffered reader object to read chars from local socket	BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
Write line to remote socket	out.println(”line”);
Read line from local socket	String str=in.readLine()

Lecture1501

- Check lab manual

Lecture1502

- Check lab manual

Email protocols

IMAP (Internet Message Access Protocol) –

Is a standard protocol for accessing e-mail from your local server.

IMAP is a client/server protocol in which e-mail is received and held for you by your Internet server.

As this requires only a small data transfer this works well even over a slow connection such as a modem.

Only if you request to read a specific email message will it be downloaded from the server.

You can also create and manipulate folders or mailboxes on the server, delete messages etc.

Email protocols (2)

The **POP (Post Office Protocol 3)** protocol provides a simple, standardized way for users to access mailboxes and download messages to their computers.

When using the POP protocol all your eMail messages will be downloaded from the mail server to your local computer.

You can choose to leave copies of your eMails on the server as well.

Email protocols (3)

The **SMTP (Simple Mail Transfer Protocol)** protocol is used by the Mail Transfer Agent (MTA) to deliver your eMail to the recipient's mail server.

The SMTP protocol can only be used to send emails, not to receive them.

Depending on your network / ISP settings, you may only be able to use the SMTP protocol under certain conditions

Email protocols (4)

The **HTTP protocol** is not a protocol dedicated for email communications, but it can be used for accessing your mailbox.

Also called web based email, this protocol can be used to compose or retrieve emails from an your account.

Hotmail is a good example of using HTTP as an email protocol.

Email

- SMTP E-mail is sent by socket communication with port 25 on a computer system.
- Open a socket connected to port 25 on some system, and speak “mail protocol” to the daemon at the other end.

Email (2)

Using telnet

telnet 192.168.1.225

<<220 192.168.1.2 ESMTP (Code-Crafters Ability Mail Server 2.52)

HELO 192.168.1.2

<<250 192.168.1.2

MAIL FROM: user001@192.168.1.2

<<250 Email address accepted. <user001@192.168.1.2>

RCPT TO: user002@192.168.1.2

<< 250 Email address accepted. <user002@192.168.1.2>

Email (3)

DATA

<<354 Please send the data and end with a <CRLF>.<CRLF>.

SUBJECT: email subject<ret><ret>

Body of the test email

.

<<250 Mail accepted and queued for delivery

quit

Lecture1503

Check lab manual

Http protocol programming

The URL class provides methods for retrieving data from a URL:

public InputStream openStream( ) throws IOException

public URLConnection openConnection( ) throws IOException

public Object getContent( ) throws IOException

Procedure to use the methods:

1) Create an URL object

e.g. URL u = new URL("http://www.iist.unu.edu");

2) Open an InputStream object directly from the URL object

e.g. InputStream in = u.openStream( );

3) Or open an URLConnection object from the URL object and then get an InputStream object from the URLConnection object .

e.g. URLConnection uc = u.openConnection( );

InputStream in = uc.getInputStream( );

4) In either case, you will have an InputStream. What's followed is the normal I/O procedure for getting data.

5) Don't forget to put the try catch block for catching the *MalformedURLException* and *IOException*.

What is the difference between using the **openStream** and **openConnection** method?

1) **openStream** method only give you the access to the raw data and cannot detect the encoding information.

2) **openConnection** method opens a socket to the specified URL and returns a URLConnection object.

3) The **URLConnection** object gives you access to everything sent by the server. You can access all the metadata specified by the protocol such as the scheme. The URLConnection class also lets you write data to as well as read from a URL.

The following methods are used to access the header fields and the contents after the connection is made to the remote object:

1)getContent

2)getHeaderField

3)getInputStream

4)getOutputStream

Certain header fields are accessed frequently. The methods:

1) getContentEncoding

2) getContentLength

3) getContentType

4) getDate

5) getExpiration

6) getLastModifed

```
        URL site = new URL(args[0]);
        URLConnection sc = site.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader
    (sc.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
        System.out.println(inputLine);
        in.close(); }}
```

```
java urlconnectionreader1 http://www.draelshafee.net
```



```
java urlconnectionreader1 http://www.draelshafee.net
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1"/>
<meta name="description" content="description"/>
<meta name="keywords" content="keywords"/>
<meta name="author" content="author"/>
<link rel="stylesheet" type="text/css" href="default.css" media="screen"/>
<title>Dr. Ahmed ElShafee</title>
</head>
<body>
```

Lecture1504

- Check lab manual

Lecture1505

- Check lab manual

Thanks,

See you next Lecture, isA