Session 6

Inter-Process Communication - 1

Course Leader: Jishmi Jos Choondal

Objectives

At the end of the session, the student will be able to

- Describe the Java API of the Internet protocols
- Explain external data representation and marshalling
- Discuss protocol support for Client-Server and Group Communications

Contents

- Java API for Internet protocols
- External data representation and marshalling
- Client-server communication
- Group communication

Introduction

- This session introduces the characteristics of interprocess communication in a distributed system both in its own right and as support for communication between distributed objects

- Presents Java interface to the TCP and UDP protocols with their Failure Models

- Discusses the construction of protocols to support the two communication patterns
    - Client server communication
    - Group communication

Role of Networks

- Distributed Systems use LAN, WAN and Internetworks for communication
- The performance, reliability, scalability, mobilty and quality of service characteristics of the underlying networks impact the behavior of distributed systems
- And hence affects their design

Types of Network

- Personal Area Networks (PANs)
- Local Area Networks (LANs)
- Wide Area Networks (WAN)
- Metropolitan Area Networks (MANs)
- Wireless Local Area Networks (WLANs)
- Wireless Metropolitan Area Networks (WMANs)
- Wireless Wide Area Networks (WWANs)
- Internetworks

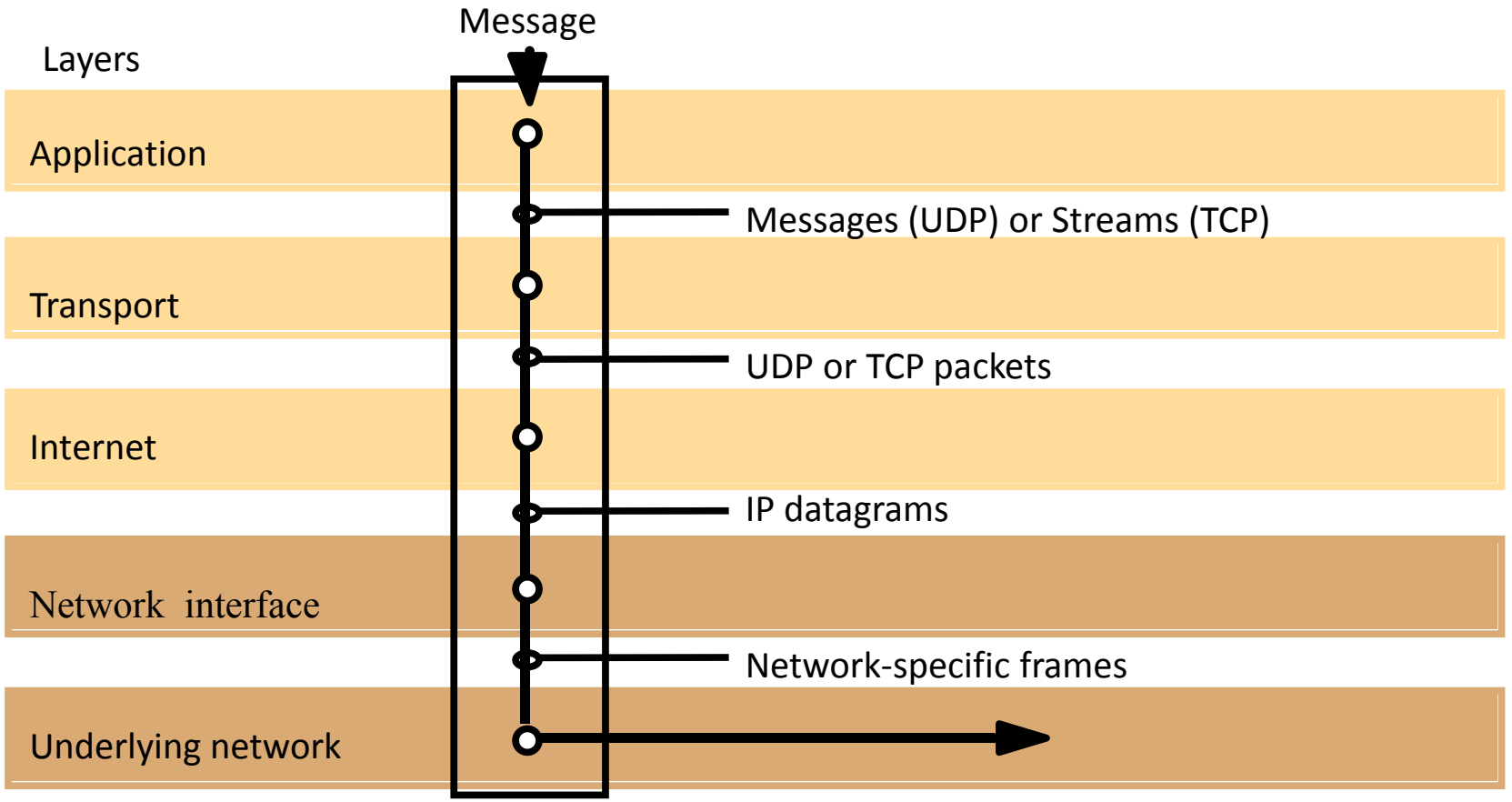
Protocols

- The term *protocol* is used to refer to a well-known set of rules and formats to be used for communication between processes in order to perform a given task.
- The definition of a protocol has two important parts to it:
  - a specification of the sequence of messages that must be exchanged;
  - a specification of the format of the data in the messages.
- A protocol is implemented by a pair of software modules located in the sending and receiving computers
- Each layer presents an interface to the layers above it that extends the properties of the underlying communication system.
- A layer is represented by a module in every computer connected to the network
- A complete set of protocol layers is referred to as a *protocol suite* or a *protocol stack*, reflecting the layered structure

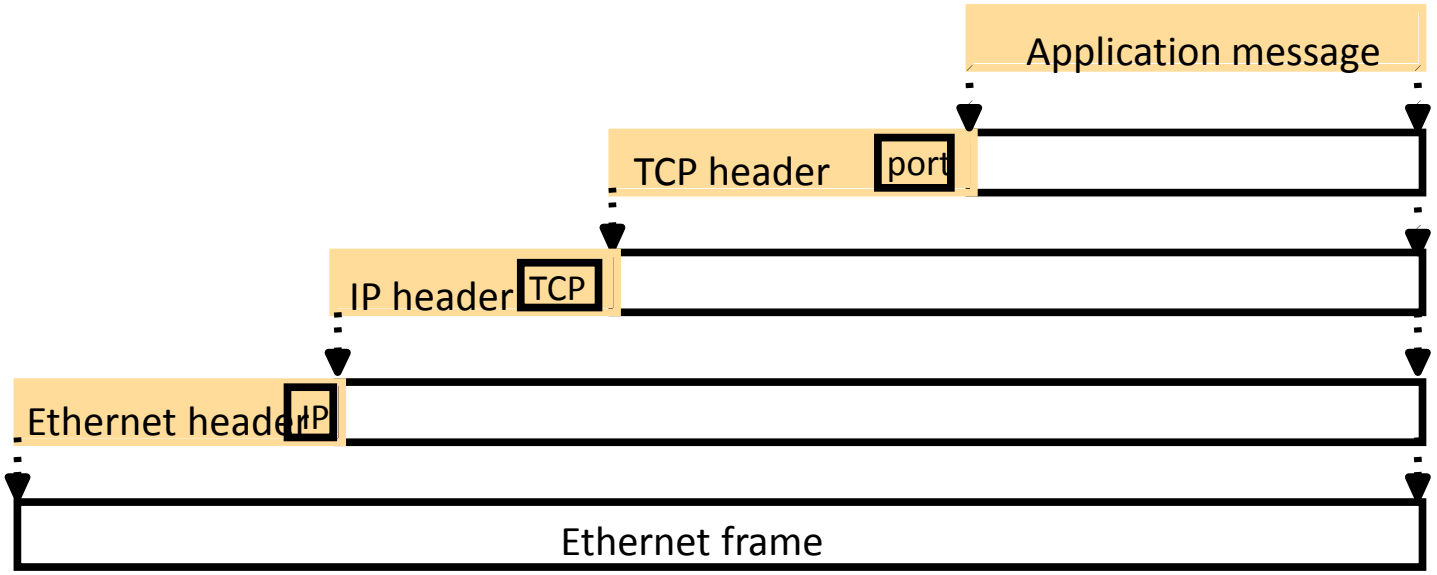
OSI protocol layers

Layer	*Description*	*Examples*
Application	Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service.	HTTP, FTP, SMTP, CORBA IIOP
Presentation	Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required.	Secure Sockets (SSL),CORBA Data Rep.
Session	At this level reliability and adaptation are performed, such as detection of failures and automatic recovery.	SIP
Transport	This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports attached to processes, Protocols in this layer may be connection-oriented or connectionless.	TCP, UDP
Network	Transfers data packets between computers in a specific network. In a WAN or an internetwork this involves the generation of a route passing through routers. In a single LAN no routing is required.	IP, ATM virtual circuits
Data link	Responsible for transmission of packets between nodes that are directly connected by a physical link. In a WAN transmission is between pairs of routers or between routers and hosts. In a LAN it is between any pair of hosts.	Ethernet MAC, ATM cell transfer, PPP
Physical	The circuits and hardware that drive the network. It transmits sequences of binary data by analogue signalling, using amplitude or frequency modulation of electrical signals (on cable circuits), light signals (on fibre optic circuits) or other electromagnetic signals (on radio and microwave circuits).	Ethernet base- band signalling, ISDN

TCP/IP layers



Encapsulation in a message transmitted via TCP over an Ethernet

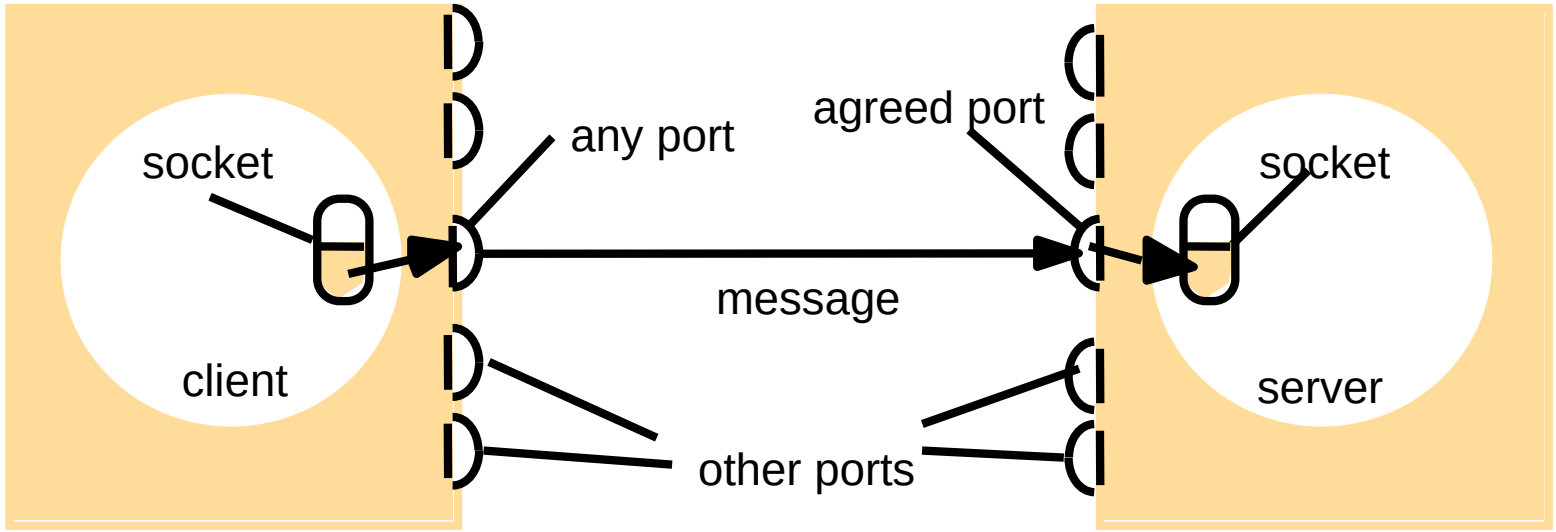


Internetwork Protocol Layers

- There are two transport protocols – TCP (Transport Control Protocol) and UDP (User Datagram Protocol).
- TCP is a reliable connection-oriented protocol, and UDP is a datagram protocol that does not guarantee reliable transmission.
- The Internet Protocol is the underlying 'network' protocol of the Internet virtual network – that is, IP datagrams provide the basic transmission mechanism for the Internet and other TCP/IP networks.
- The tags in the headers are the protocol types for the layers above, needed for the receiving protocol stack to correctly unpack the packets.
- In the TCP layer, the receiver's port number serves a similar purpose, enabling the TCP software component at the receiving host to pass the message to a specific application-level process.

The API for the Internet Protocols

- Sockets
  - IPC consists of transmitting a message between a socket in one process and a socket in another process



The Java API for Internet Addresses

- Java provides a class, *InetAddress*, that represents Internet Addresses
- Users of this class refer to computers by Domain Name Service hostnames
- Instances of InetAddress that contain Internet Addresses can be created by calling a static method of InetAddress, giving hostname as argument

UDP datagram Communication

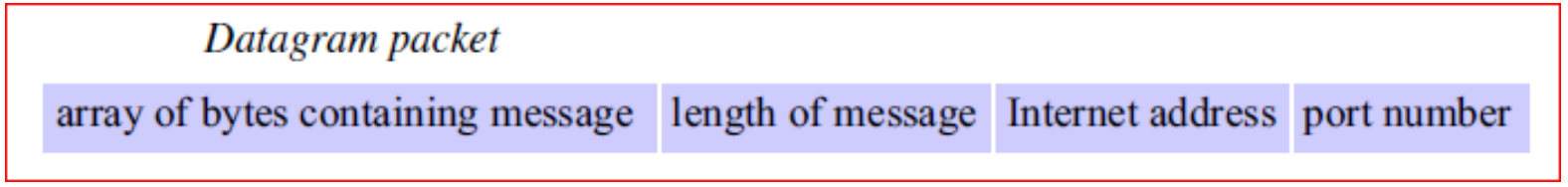
- A datagram sent by UDP is transmitted from a sending process to a receiving process without acknowledgement or retries.
- If a failure occurs, the message may not arrive.
- A datagram is transmitted between processes when one process *sends* it and another *receives* it.
- send or receive messages a process must first create a socket bound to an Internet address of the local host and a local port.
- A server will bind its socket to a *server port* – one that it makes known to clients so that they can send messages to it

The API for the Internet Protocols

- Java API provides communication for UDP datagrams by means of two classes
  - DatagramPacket
  - DatagramSocket

The API for the Internet Protocols

- DatagramPacket
  - It provides a constructor contains array of bytes containing message, length of message, Internet Address and Port Number
  - This class provides another constructor for use when receiving a message and its arguments are an array of bytes in which to receive message and length of the array
  - The message can be retrieved from the DatagramPacket by means of *getData*
  - The methods *getPort* and *getAddress* access the port and Internet Address



The API for the Internet Protocols

- DatagramSocket provides methods that include
  - *send and receive* are for transmitting datagrams between a pair of sockets
  - The argument of *send* is an instance of DatagramPacket containing message and its destination
  - The argument of *receive* is an empty DatagramPacket in which to put message, its length and its origin
  - The method *setSoTimeout* allows a timeout to be set
  - The method *connect* used for connecting it to a particular remote port and Internet address

UDP Client Sends a Message and Gets a Reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
                // args give message contents and server hostname
                DatagramSocket aSocket = null;
                  try {
                                aSocket = new DatagramSocket();
                                byte [] m = args[0].getBytes();
                                InetAddress aHost = InetAddress.getByName(args[1]);
                                int serverPort = 6789;
                                DatagramPacket request = new DatagramPacket(m, m.length(), aHost,
serverPort);
                                aSocket.send(request);
                                byte[] buffer = new byte[1000];
                                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                                aSocket.receive(reply);
                                System.out.println("Reply: " + new String(reply.getData()));
                  }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                  }catch (IOException e){System.out.println("IO: " + e.getMessage());}
                }finally {if(aSocket != null) aSocket.close();}
    }
}
```

UDP Server Repeatedly Receives a Request and Sends it back to the Client

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]){
        DatagramSocket aSocket = null;
            try{
                    aSocket = new DatagramSocket(6789);
                    byte[] buffer = new byte[1000];
                    while(true){
                        DatagramPacket request = new DatagramPacket(buffer,
buffer.length);
                       aSocket.receive(request);
                       DatagramPacket reply = new DatagramPacket(request.getData(),
                               request.getLength(), request.getAddress(),
request.getPort());
                       aSocket.send(reply);
                    }
                }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
               }catch (IOException e) {System.out.println("IO: " + e.getMessage());}
        }finally {if(aSocket != null) aSocket.close();}
    }
}
```

The API for the Internet Protocols

- Java API provides communication for TCP streams by means of two classes
  - ServerSocket
  - Socket

The API for the Internet Protocols

- ServerSocket
  - This class is intended for use by a server to create a socket at a server port for listening for *connect* requests from clients.
  - Its *accept* method gets a *connect* request from the queue, or if the queue is empty , it blocks until one arrives.
  - The result of executing *accept* is an instance of *Socket* for giving access to streams for communicating with client

The API for the Internet Protocols

- Socket
  - This class is intended for use by a pair of processes with a connection
  - The client uses a constructor to create not only a socket associated with a local port but also connects it to the specified remote computer and port number
  - The Socket class provides methods *getInputStream* and *getOutputStream* for accessing the two streams associated with it

TCP Client

```
import java.net.*;
import java.io.*;
public class TCPClient {
        public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
            try{
                    int serverPort = 7896;
                    s = new Socket(args[1], serverPort);
                    DataInputStream in = new DataInputStream( s.getInputStream());
                    DataOutputStream out =
                            new DataOutputStream( s.getOutputStream());
                    out.writeUTF(args[0]);          // UTF is a string encoding
                    String data = in.readUTF();
                    System.out.println("Received: "+ data) ;
            }catch (UnknownHostException e){
                            System.out.println("Sock:"+e.getMessage());
            }catch (EOFException e){System.out.println("EOF:"+e.getMessage());
            }catch (IOException e){System.out.println("IO:"+e.getMessage());}
        }finally {if(s!=null) try {s.close();}catch (IOException
e){System.out.println("close:"+e.getMessage());}
        }
    }
}
```

TCP Echo Server

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
            try{
                    int serverPort = 7896;
                    ServerSocket listenSocket = new ServerSocket(serverPort);
                    while(true) {
                            Socket clientSocket = listenSocket.accept();
                            Connection c = new Connection(clientSocket);
                    }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
    }
}

// continues on the next slide
```

Continued

```
class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
            try {
                    clientSocket = aClientSocket;
                    in = new DataInputStream( clientSocket.getInputStream());
                    out =new DataOutputStream( clientSocket.getOutputStream());
                    this.start();
             } catch(IOException e)
{System.out.println("Connection:"+e.getMessage());}
        }
        public void run(){
            try {                                       // an echo server
                    String data = in.readUTF();
                    out.writeUTF(data);
            } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
            } catch(IOException e) {System.out.println("IO:"+e.getMessage());}
            } finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
        }
}
```

Summary

In this session

- We have presented a overview of computer networking with reference to the communication requirements of distributed system
- We have presented the Java interface to the TCP and UDP prtocols

Questions

Thank you

- https://www.youtube.com/watch?v=YY0SvestyaQ