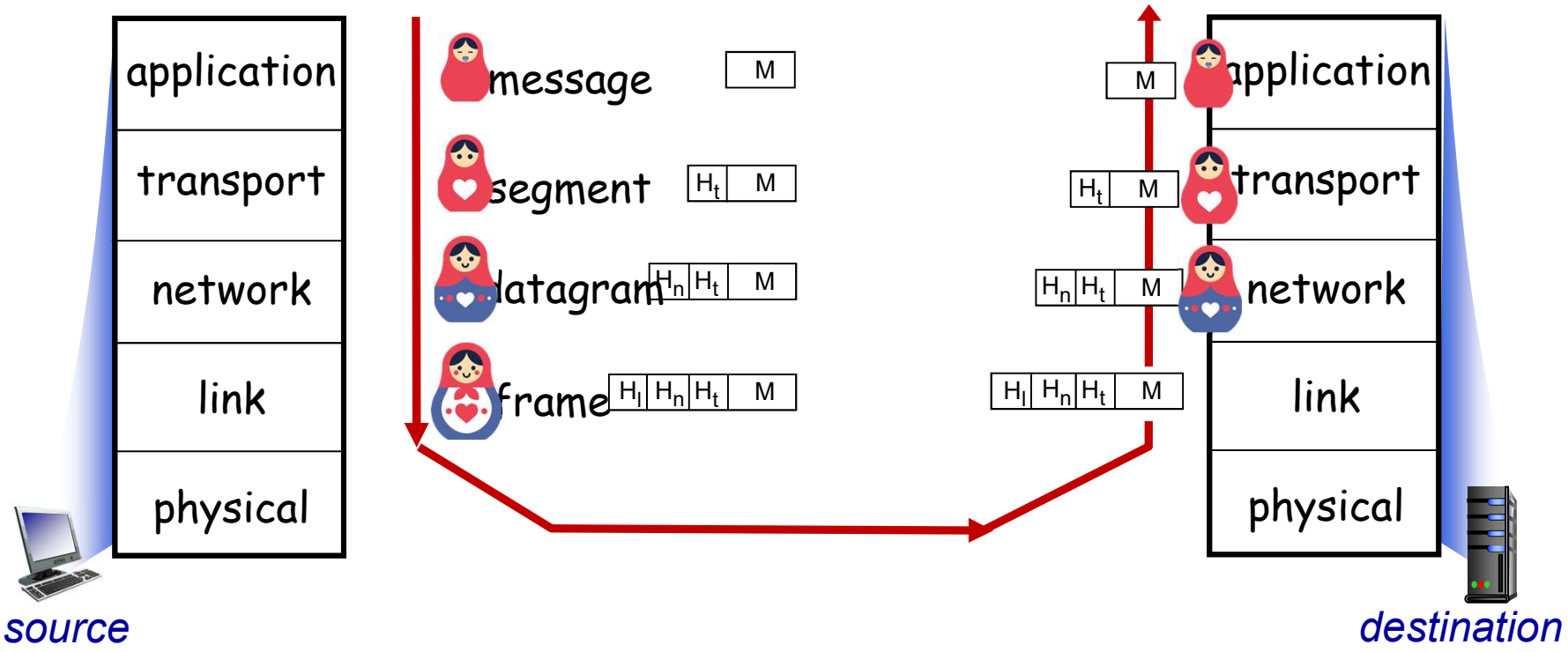
UDP Socket Programming

Contents

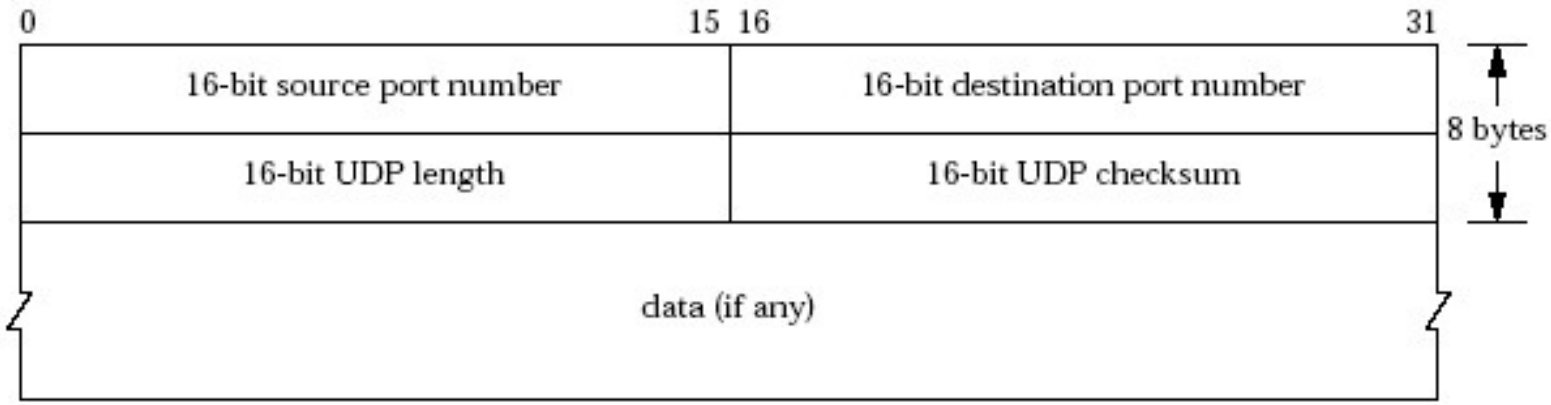
- UDP introduction
- Client/Server Model
- Socket class in Java
- UDP socket programming
- Examples
- Summary

Services, Layering and Encapsulation

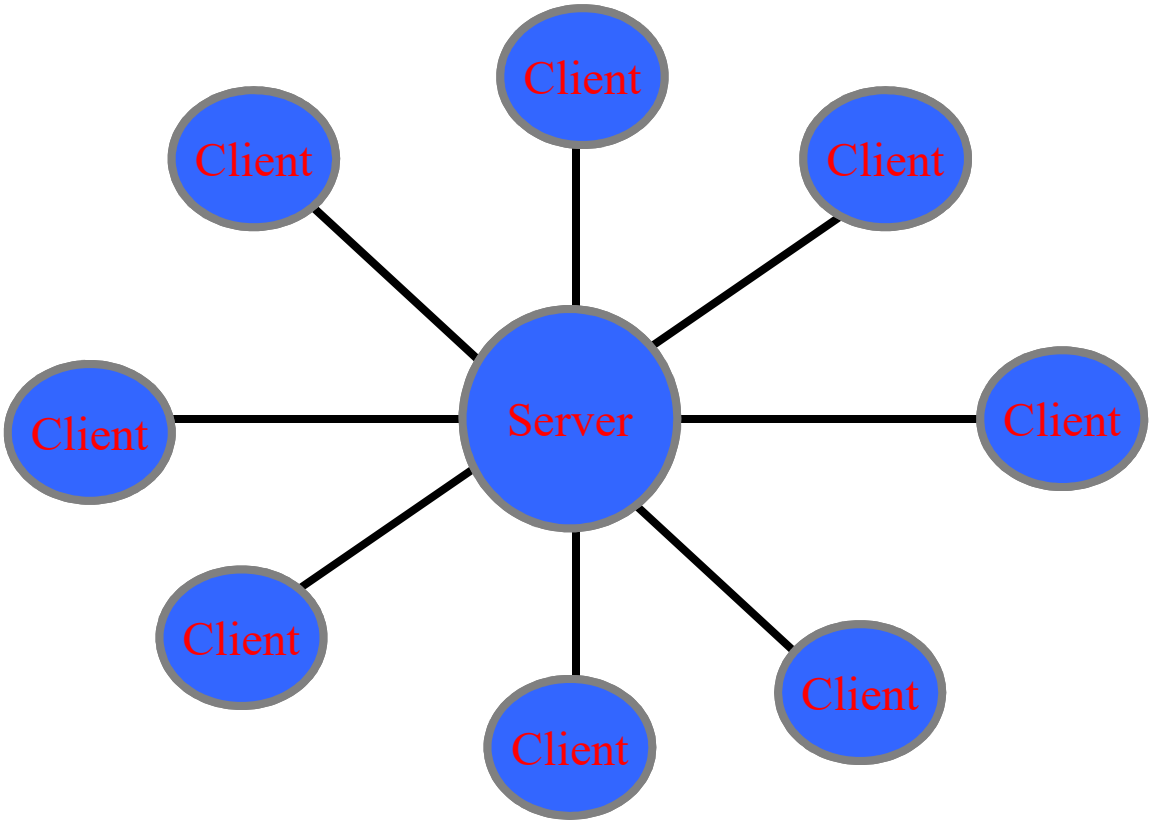


UDP

- Connectionless
- Packet can be **lost and disordered**
- Why do we use UDP ?



Client/Server Model



Client-server paradigm

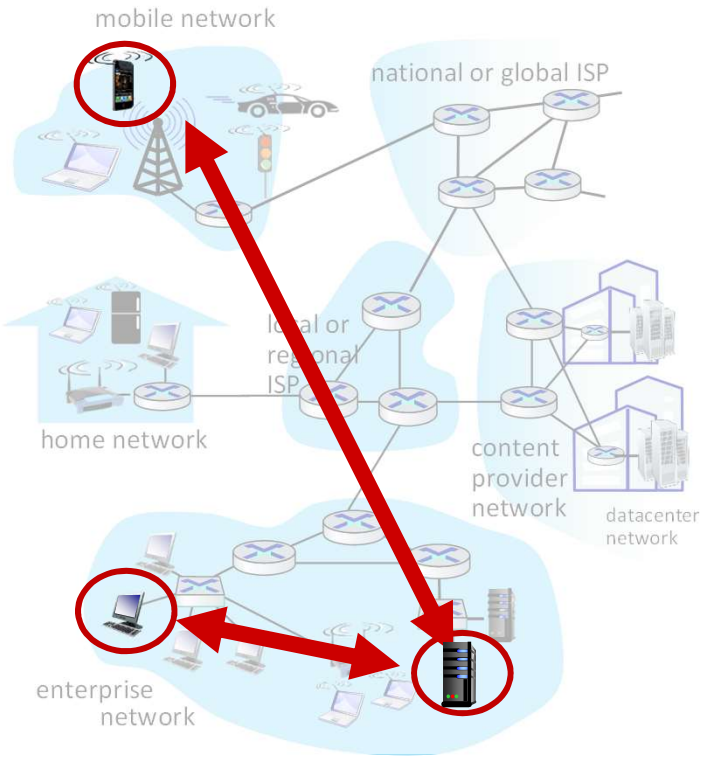
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



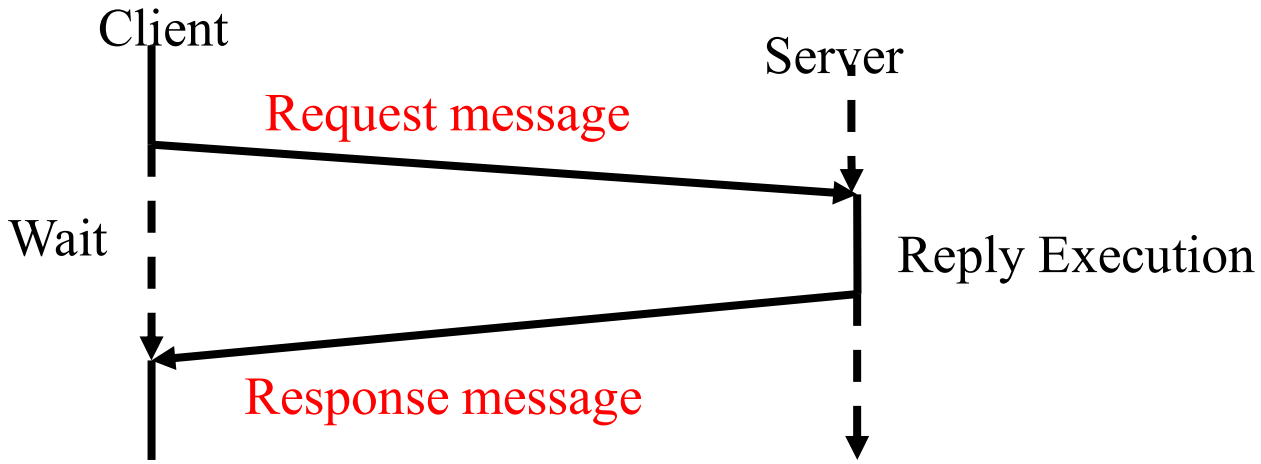
Client/Server Model

Server

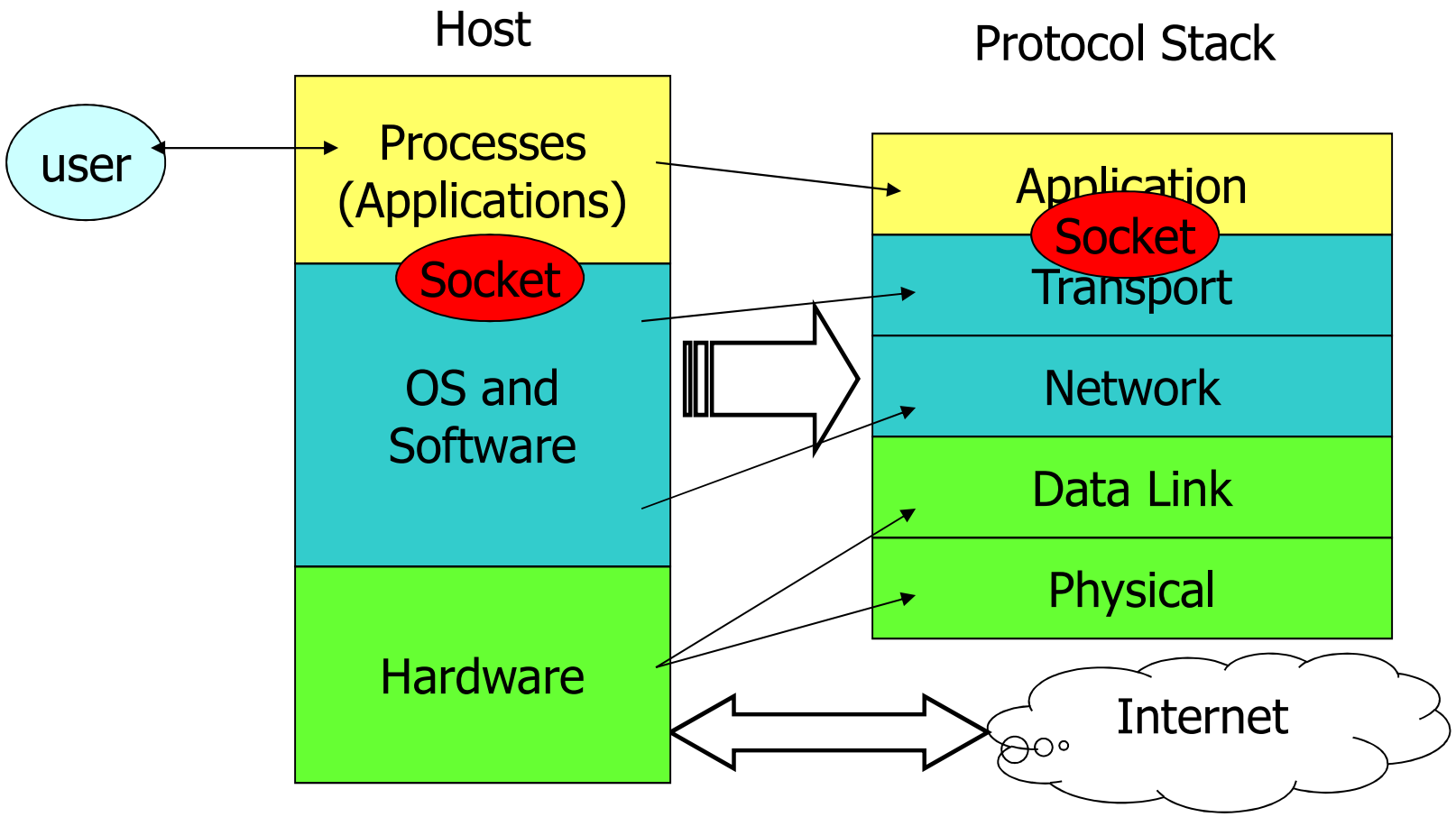
- Running before Client
- Waiting for request message from Client
- Sending response message to Client

Client

- Running after Server
- Sending request message to Server
- Waiting for response message from Server



Socket


Host
Protocol Stack
user
Processes
(Applications)
Socket
OS and
Software
Hardware
Application
Socket
Transport
Network
Data Link
Physical
Internet

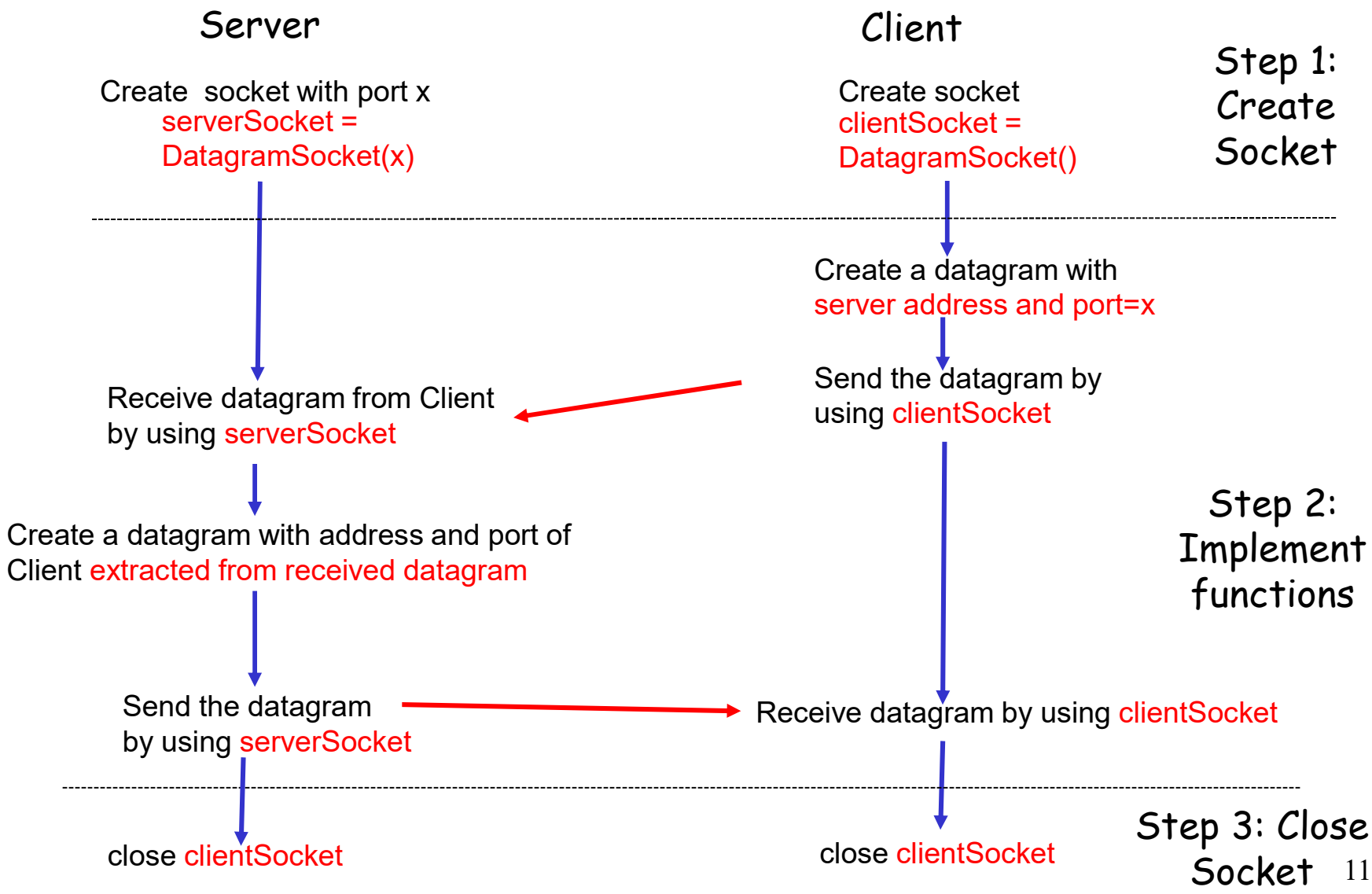
Socket class

- package java.net
- TCP
  - Socket
  - ServerSocket
- UDP
  - DatagramPacket
  - DatagramSocket

UDP Socket programming

- No hand-shaking method
- Sender will attach IP address and port of Receiver into group of small bytes of data => datagram
- After receiving datagram, Receiver will remove IP address and port and get data
- In short, UDP provides a way to transmit a group of bytes ("datagram") without any guarantee between sender and receiver.
- In UDP programming, all kinds of data have to be converted to a group of bytes before transmission.

Flowchart of data communication



Example 1

- Write client program connecting to server program by **UDP**. Client sends a string, which is inputted from keyboard, to server. Server will convert all letters of this string to upper-case letters and send back to Client. Client will print the data received from Server to Console.

UDPClient.java

```
try{
   DatagramSocket cl = new DatagramSocket();

   Scanner keyboard = new Scanner(System.in);
   System.out.println("Please input a string:");
   String st= keyboard.nextLine();

   byte buff[] = st.getBytes();
   InetAddress addsv = InetAddress.getByName("localhost");
   DatagramPacket p = new DatagramPacket(buff,buff.length,addsv,1234);
   cl.send(p);

   byte buff2[] = new byte[256];
   DatagramPacket l = new DatagramPacket(buff2,buff2.length);
   cl.receive(l);
   String data = new String(l.getData()).trim();
   System.out.println("Data from Server:"+data);

   cl.close();
 } catch(IOException e){}
```

Create a UDP Socket

Convert to array of Bytes

Create an InetAddress with IP address of Server

Send packet to Server

Create a packet with port of Server

Get data from received packet

UDPServer.java

```
try{
    DatagramSocket sv = new DatagramSocket(1234);

    byte buff1[] = new byte[256];
    DatagramPacket q = new DatagramPacket(buff1, buff1.length);
    sv.receive(q);

    String data = new String(q.getData()).trim();
    String kq = data.toUpperCase();
    byte buff2[] = new byte[256];
    buff2 = kq.getBytes();
    InetAddress addcl = q.getAddress();
    int portcl = q.getPort();
    DatagramPacket k = new DatagramPacket(buff2,buff2.length,addcl,portcl);
    sv.send(k);

    sv.close();
  } catch(IOException e) {}
```

Create a UDP Socket with port 1234

Receive a packet from Client

Get data from received packet

Get InetAddress and port of Client from received packet

Create a packet included address and port of Client

Example 2

- Write client program connecting to server program by **UDP**. Client sends an integer, which is inputted from keyboard, to server. After receiving, Server will send a string "even" or "odd", depending on the integer. Client prints the string received from Server to Console.

UDPClient.java

```
try{
    DatagramSocket cl = new DatagramSocket();
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Please input an integer:");
    int x= keyboard.nextInt();
    byte buff[] = String.valueOf(x).getBytes();
    InetAddress addsv = InetAddress.getByName("localhost");
    DatagramPacket p = new DatagramPacket(buff,buff.length,addsv,1234);
    cl.send(p);

    byte buff2[] = new byte[256];
    DatagramPacket l = new DatagramPacket(buff2,buff2.length);
    cl.receive(l);
    String data = new String(l.getData()).trim();
    System.out.println("Result:"+data);
    cl.close();
 } catch(IOException e){}
```

UDPServer.java

```
try{
    DatagramSocket sv = new DatagramSocket(1234);

    byte buff[] = new byte[256];
    DatagramPacket q = new DatagramPacket(buff, buff.length);
    sv.receive(q);

    String data = new String(q.getData()).trim();
    int x = Integer.parseInt(data);
     String result=(x%2==0)? "even number" : "odd number";

    byte buff2[] = new byte[256];
    buff2 = result.getBytes();
    InetAddress addcl = q.getAddress();
    int portcl = q.getPort();
    DatagramPacket k = new DatagramPacket(buff2,buff2.length,addcl,portcl);
    sv.send(k);

    sv.close();
  } catch(IOException e) {}
```

Example 3

- Write client program connecting to server program by **UDP**. Client sends two integers, which are inputted from keyboard, to server. Server will calculate the result = number1 – number2 and send the result back to Client. Client will print the result received from Server to Console.

Some important notes

- Both client and server use DatagramSocket
- All kinds of data have to be converted to bytes before transmission.
- Both IP address and port of receiver (server or client) are included in datagram.
- In UDP programming, Client usually send a datagram to Server firstly because only **Client know the address and port of Server**.
- We make sure that we receive the correct data we want. Data can be lost and disordered.

Methods of DatagramSocket

- public void send(DatagramPacket dp) throws IOException
- public void receive(DatagramPacket dp) throws IOException
- public void close( )
- public int getLocalPort( )
- public InetAddress getLocalAddress( )
- public void connect(InetAddress host, int port)
- public int getPort( )
- public InetAddress getInetAddress( )
- public InetAddress getRemoteSocketAddress( )

DatagramPacket

- Maximum length of datagram packet is 65,507 bytes
- Constructors
  - public DatagramPacket(byte[] buffer, int length)
  - public DatagramPacket(byte[] buffer, int offset, int length)
- Other constructors
  - `public DatagramPacket(byte[] data, int length, InetAddress destination, int port)`
  - `public DatagramPacket(byte[] data, int offset, int length, InetAddress destination, int port)`
  - `public DatagramPacket(byte[] data, int length, SocketAddress destination, int port)`
  - `public DatagramPacket(byte[] data, int offset, int length, SocketAddress destination, int port)`

Other methods of DatagramPacket

- Address
  - public InetAddress getAddress( )
  - public int getPort( )
  - public SocketAddress getSocketAddress( )
  - public void setAddress(InetAddress remote)
  - public void setPort(int port)
  - public void setAddress(SocketAddress remote)

Other methods of DatagramPacket

- Data
  - `public byte[] getData( )`
  - `public int getLength( )`
  - `public int getOffset( )`
  - `public void setData(byte[] data)`
  - `public void setData(byte[] data, int offset, int length )`
  - `public void setLength(int length)`

DatagramSocket

- Constructors
  - public DatagramSocket( ) throws SocketException
  - public DatagramSocket(int port) throws SocketException
  - public DatagramSocket(int port, InetAddress interface) throws SocketException
  - public DatagramSocket(SocketAddress interface) throws SocketException
  - (protected DatagramSocket(DatagramSocketImpl impl) throws SocketException)

Example

```
java.net.*;
public class UDPPortScanner {
  public static void main(String[] args) {

    for (int port = 1024; port <= 65535; port++) {
      try {
            // checking if a port is being used?
            //if yes, throw the exception
            DatagramSocket server = new DatagramSocket(port);
            server.close( );
      }
      catch (SocketException ex) {
        System.out.println("Port:" + port + " is being used");
      } // end try
    } // end for
  }
}
```

Supporting many clients simultaneously

- Can you rewrite your program to allow a server support many UDP clients simultaneously?
  - Why?
  - How?

Summary

- If we use UDP for transmission, the data can be lost and disordered.
- DatagramSocket is used for both client and server.
- Client/Server program has 3 steps.
- We need to make sure the received data is the one we want.

Homework

- Please finish all assignments
- Next week is TCP Socket.