ECS524
Application Networking
UDP and sockets

Prof. Steve Uhlig
steve.uhlig@qmul.ac.uk
Office: Eng202

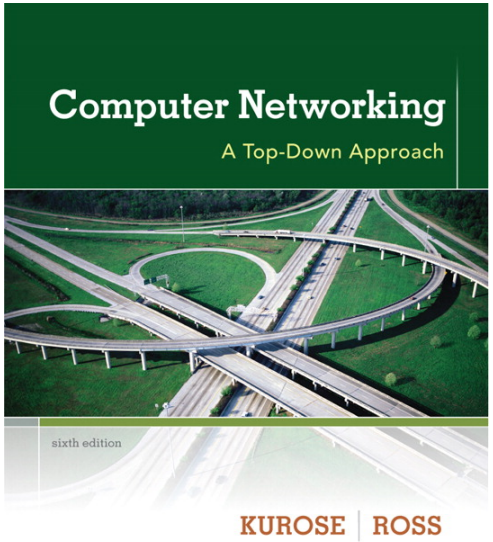
Dr. Felix Cuadrado
felix.cuadrado@qmul.ac.uk
Office: E205

Slides

Queen Mary University of London

Disclaimer:
Some of the slides' content is borrowed directly from those provided by the authors of the textbook. They are available from

http://www-net.cs.umass.edu/kurose-ross-ppt-6e



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

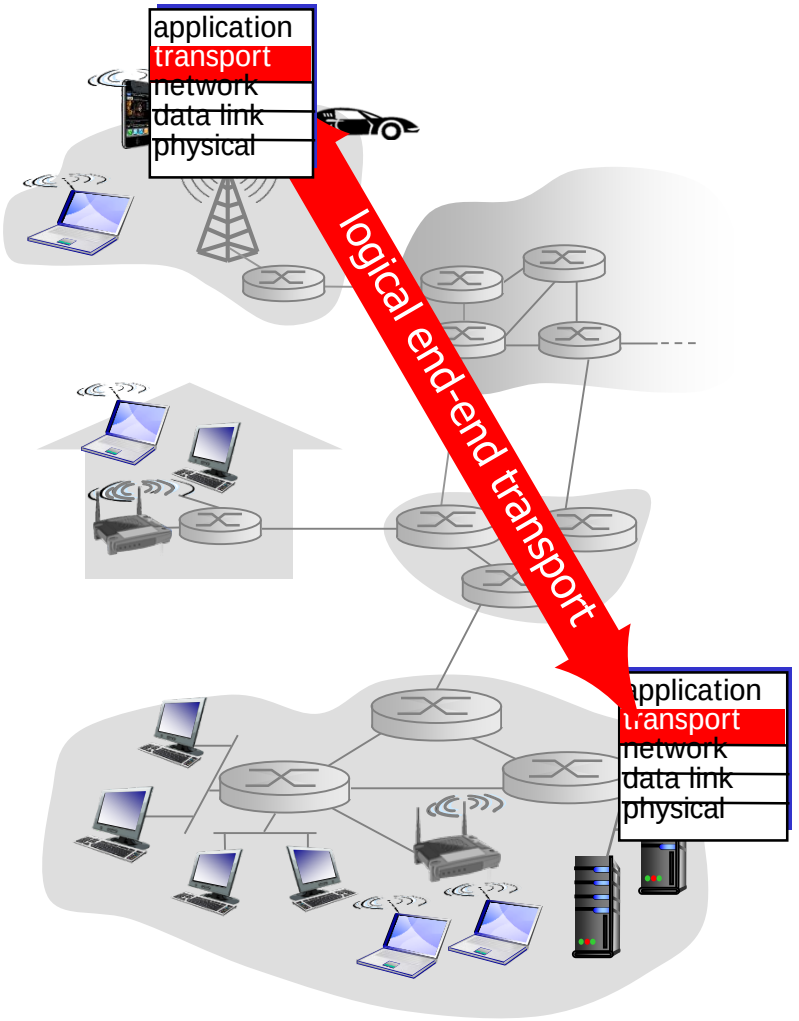
Application Networking

Queen Mary University of London

- **Transport Layer**
- UDP Sockets
- TCP Sockets

Transport services

- provide *logical communication*
  - between app processes running on different hosts
- transport protocols run in end systems
  - sender side: breaks app messages into *segments*, passes to network layer
  - receiver side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport service requirements: common apps

Queen Mary University of London

application	data loss	throughput	time sensitive
e-mail	no loss	elastic	no
file transfer	no loss	elastic	no
Web documents	no loss	elastic	no
interactive games	loss-tolerant	few kbps	yes, 100s of msec
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 10s of msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

Queen Mary University of London

TCP service:

***connection-oriented**:* setup required between client and server processes

***reliable** transport* between sending and receiving process

flow control: sender won't overwhelm receiver

congestion control: throttle sender when network overloaded

*does **not** provide:* timing, minimum throughput guarantee, security

UDP service:

***connectionless** service*

***unreliable** data transfer* between sending and receiving process

*does **not** provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

User Datagram Protocol

Queen Mary University of London

"no frills," "bare bones" Internet transport protocol

"best effort" service, UDP segments may be:

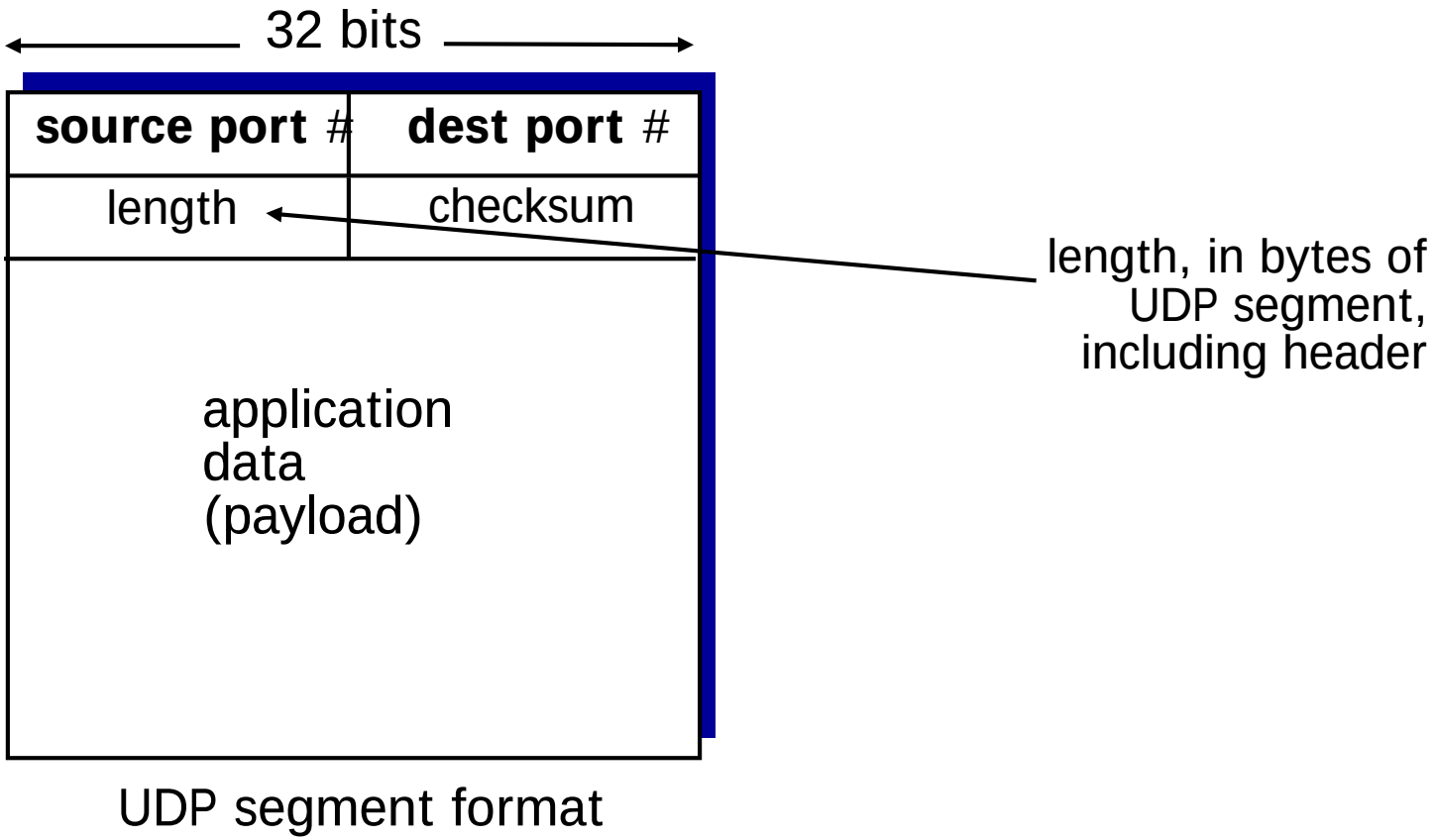
- lost
- delivered out-of-order to app

connectionless:

no handshaking between UDP sender, receiver

each UDP segment handled independently of others

UDP: segment header



UDP segment format

Why UDP?

Queen Mary University of London

- Why use UDP instead of TCP?
  - no connection establishment (which can add delay)
  - simple: no connection state at sender, receiver
  - small header size
  - no congestion control: UDP transmits as fast as desired
  - Flexible: possible to implement some TCP mechanisms (or variants) at the application layer

- UDP is used by several applications:
  - streaming multimedia apps
    - (loss tolerant, rate sensitive)
  - DNS queries
  - SNMP (network management)

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

sender:

treat segment contents, including header fields, as sequence of 16-bit integers

checksum: addition (one's complement sum) of segment contents

sender puts checksum value into UDP checksum field

receiver:

compute checksum of received segment

check if computed checksum equals checksum field value:

NO - error detected

YES - no error detected. *But maybe errors nonetheless?* More later ….

Application Networking

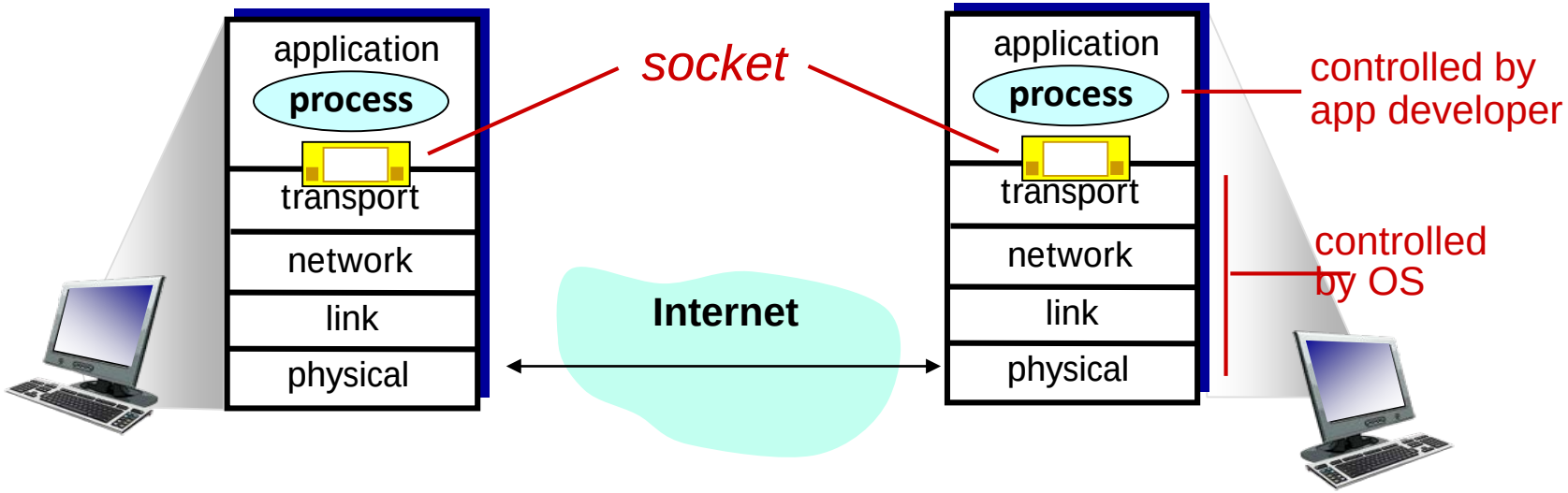
Queen Mary University of London

- Transport Layer
- **UDP Sockets**
- TCP Sockets

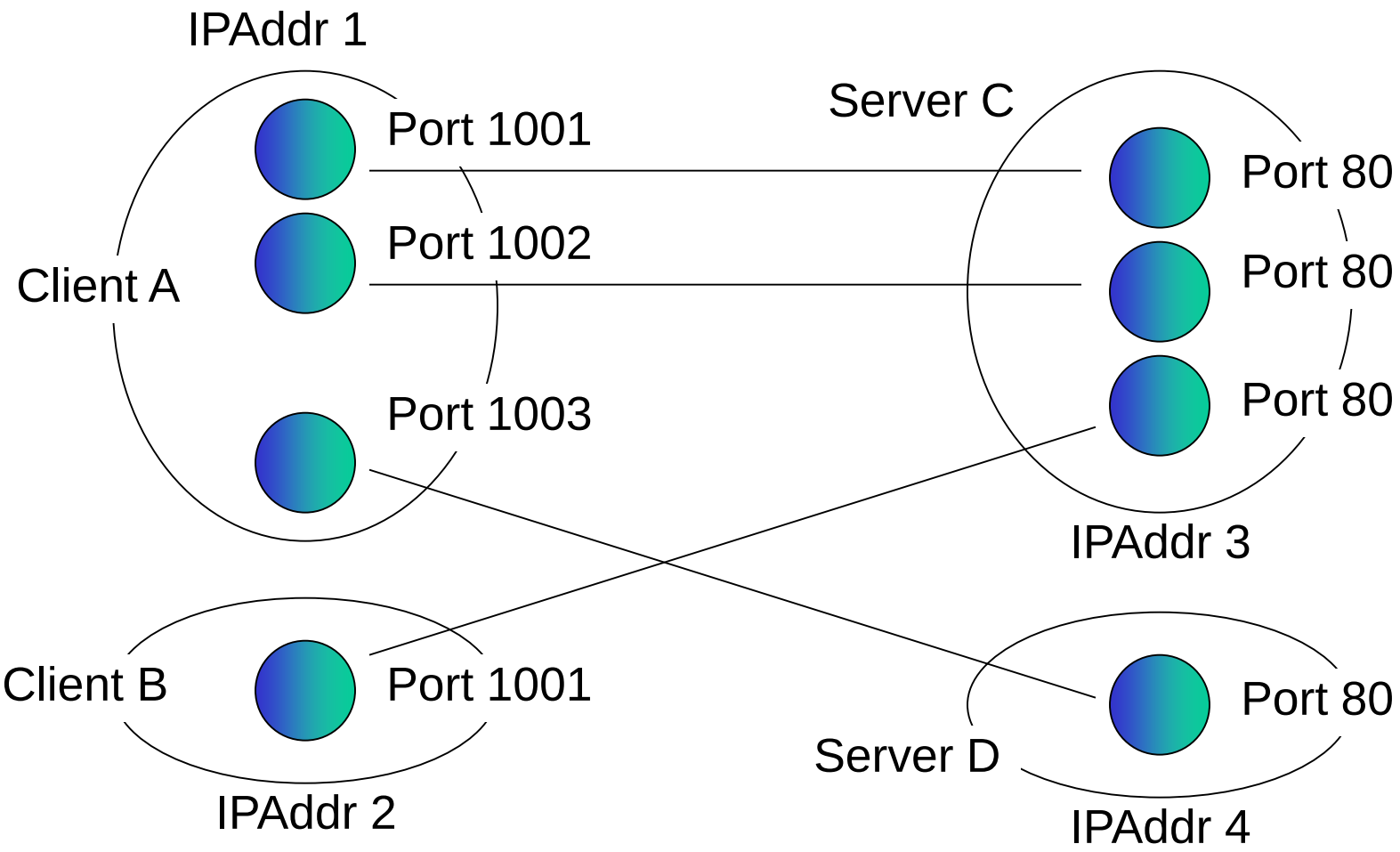
Socket programming

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

The API for applications to communicate across the network



Unambiguous Server Processing



UDP demultiplexing

Queen Mary University of London

created socket has host-local port #:

```
DatagramSocket mySocket1
= new DatagramSocket(12534);
```

datagram must specify

- destination IP address
- destination port #

when host receives UDP segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #

IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

Socket ports

Queen Mary University of London

- Logical resources managed by the operating system
- Sockets are always assigned a free port when created
- Each process on a networked host can be addressed remotely by the port number it is listening to
- TCP and UDP ports are independent
- For server-side applications, default port numbers are defined
  - HTTP -> 80
  - HTTPS -> 443
  - SMTP -> 25

Socket programming

Two socket types for two transport services:

UDP: unreliable datagram (message oriented)

TCP: reliable, byte stream-oriented

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

UDP Socket programming

Queen Mary University of London

UDP: no "connection" between client & server

no handshaking before sending data

1. sender explicitly attaches IP destination address and port # to each packet
2. receiver extracts sender IP address and port# from received packet

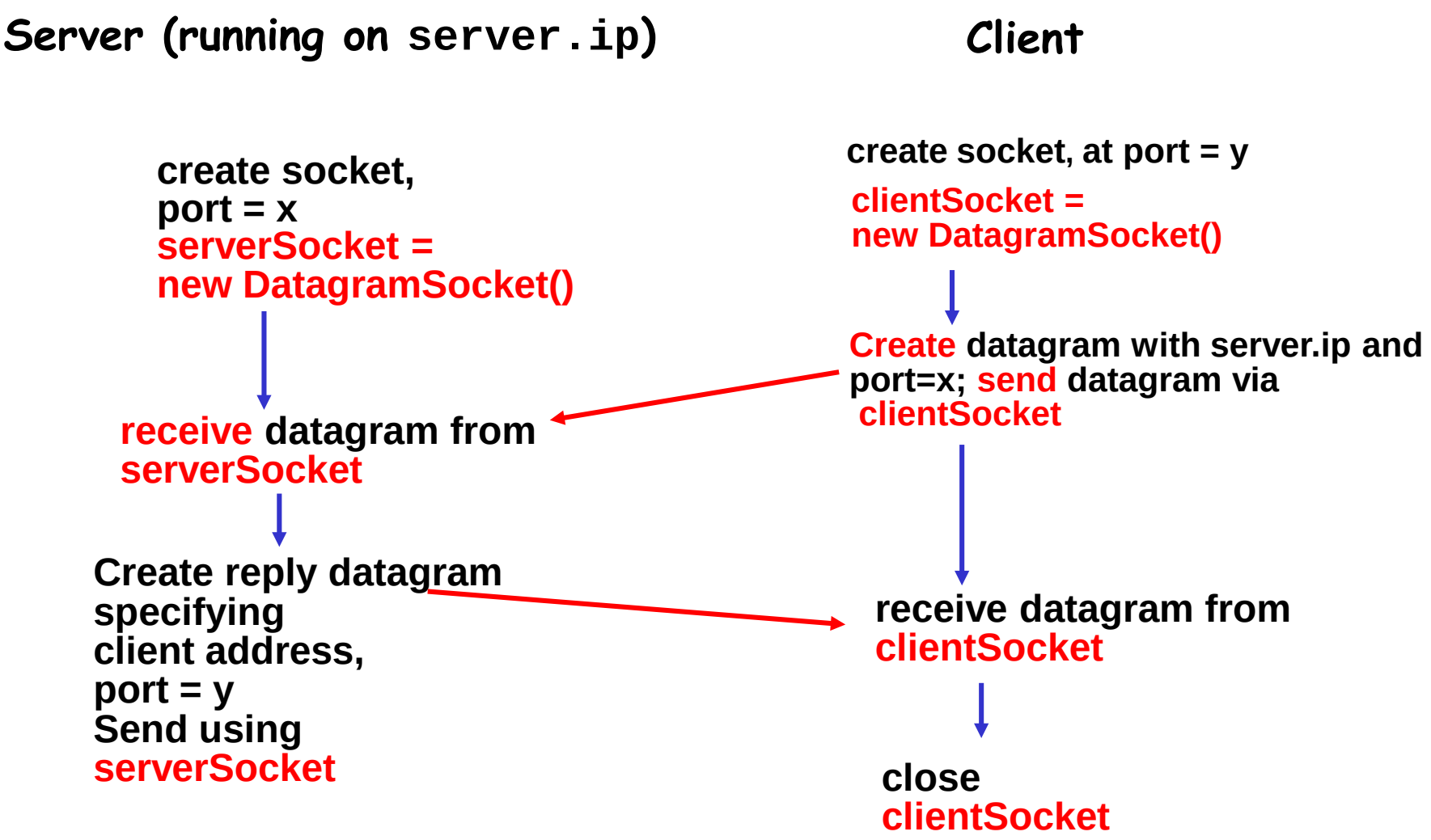
transmitted data may be lost or received out-of-order

Datagram Sockets

Queen Mary University of London

- DatagramSocket class allows to create both clients and servers using UDP
- UDP packets are sent individually (DatagramPacket object).
  - socket.send(), socket.receive()
- The server has to manually retrieve ip and port from the received packet to construct the reply

Client/server socket interaction: UDP



UDPClient.java

```
DatagramSocket clientSocket = new DatagramSocket();
InetAddress addr = InetAddress.getByName("hostname");

byte[] data = new byte[1024];
data = “this is ECS524”.getBytes();
DatagramPacket sendPacket = new DatagramPacket(data,
    data.length, addr, 9876);
clientSocket.send(sendPacket);
DatagramPacket receivePacket = new DatagramPacket(data,
   data.length);
clientSocket.receive(receivePacket);
String response = new String(receivePacket.getData());
System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
```

UDPServer.java

Queen Mary University of London

```
DatagramSocket serverSocket = new DatagramSocket(9876);

byte[] data = new byte[1024];
while(true) {
  DatagramPacket rcvPacket =
                  new DatagramPacket(data, data.length);
   serverSocket.receive(rcvPacket);
   String sentence = new String(rcvPacket.getData());
   InetAddress addr = rcvPacket.getAddress();
   int port = rcvPacket.getPort();
   String capitalizedSentence = sentence.toUpperCase();
   data = capitalizedSentence.getBytes();
   DatagramPacket sendPacket =
          new DatagramPacket(data, data.length, addr,port);
   serverSocket.send(sendPacket);
}
```

Application Networking

Queen Mary University of London

- Transport Layer
- UDP Sockets
- **TCP Sockets**

TCP Socket programming

Queen Mary University of London

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

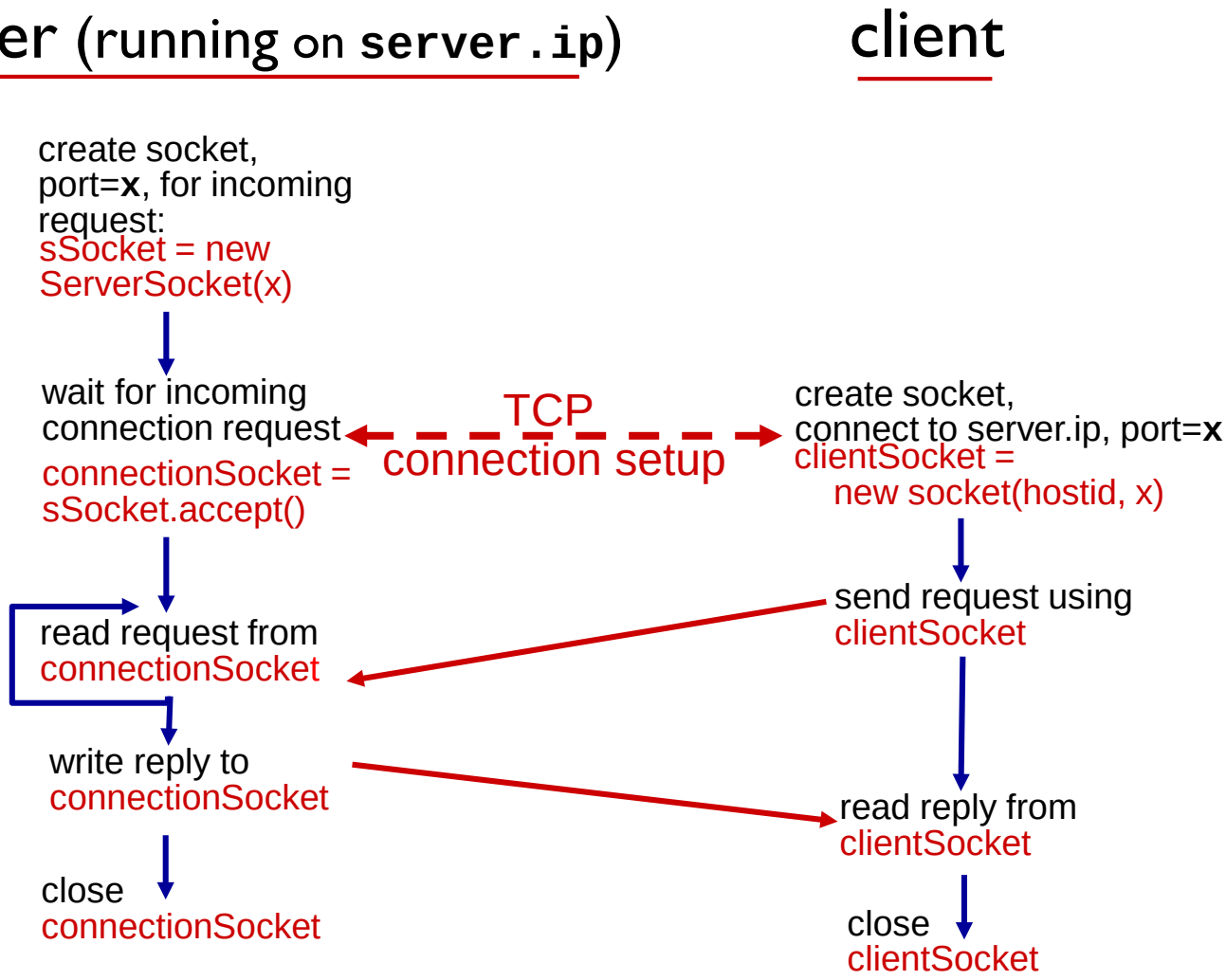
when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client

- 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

Client/server socket interaction: TCP



TCP Socket creation and configuration

Queen Mary University of London

- Sockets are the connection interface between a client and a server
  - Client: ip address, port number
  - Server: ip address, port number
- A client Socket specifies on creation the server host and the destination port
  - new Socket(“eecs.qmul.ac.uk”, 80)
  - Server sockets will get the client host and port from the connection request

Socket configuration: Local Port

Queen Mary University of London

Every socket must **bind** to a local port

- A socket must define local and destination addresses
- Typically local port is **automatically selected** at the client
- Server sockets define the **specific port** number to be used

```
ServerSocket httpServer = new ServerSocket(80);
```

Sending Data through a Socket

Queen Mary University of London

Once a socket connection is established it can send and receive data through the input and output streams

For sending data in Java, a wrapper Stream can be used

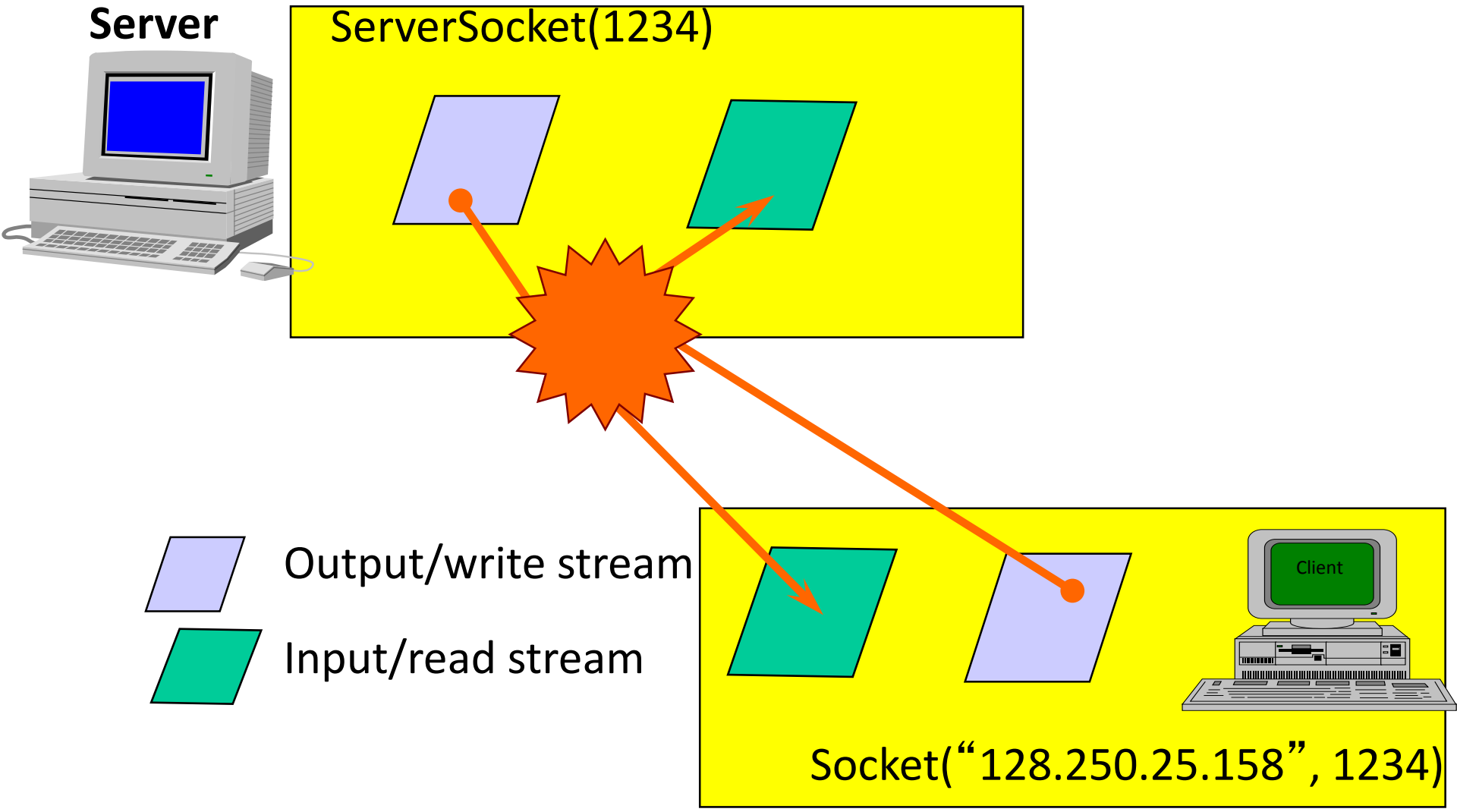
```
os = new DataOutputStream(socket.getOutputStream());
os.writeBytes("Hello\n");
os.flush();
```

The input stream allows to read data from the other side

```
is = new DataInputStream(socket.getInputStream() );
String input = is.readLine();
```

Both Streams are available until the Socket is closed in either side

Java Sockets communication streams



How a Server Socket Accepts Connections

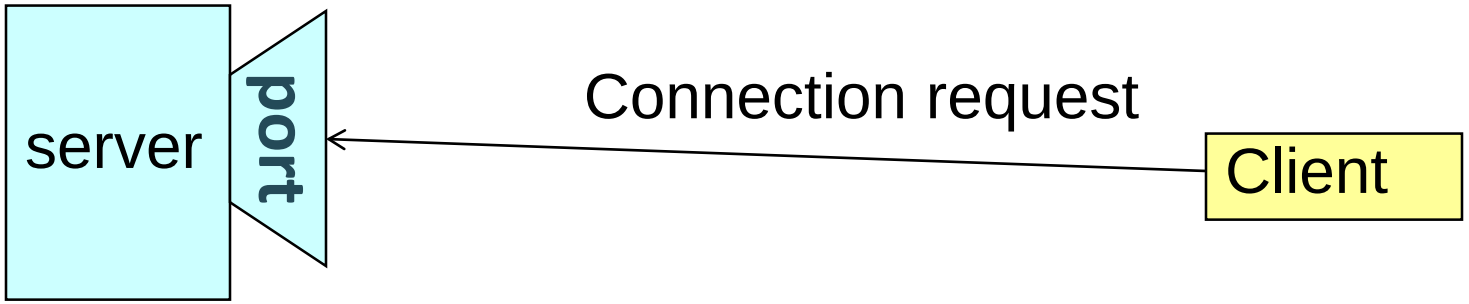
Queen Mary University of London

- A ServerSocket accepts client connections through the Accept operation.
  - The method is a blocking call, waiting until a connection request arrives
  - Creates a new socket when a connection request arrives
    - Socket cSock = serverSocket.accept()
  - When a connection request arrives the server can either handle the requests **iteratively** or **concurrently**

TCP Socket Communications Sequence (I)

Queen Mary University of London

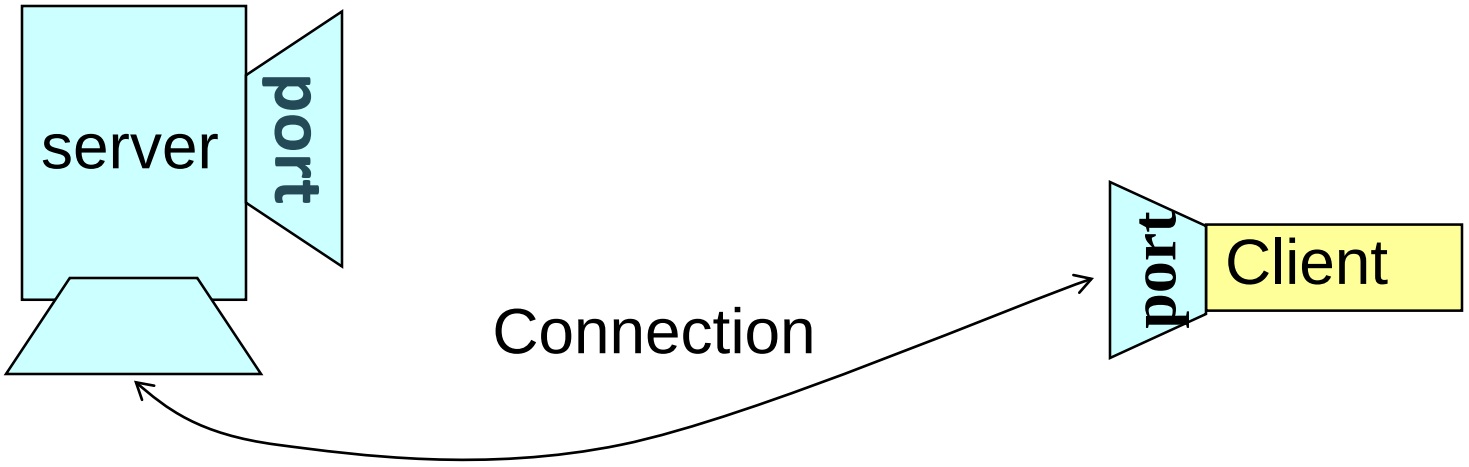
A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.



TCP Socket Communications Sequence (II)

Queen Mary University of London

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket to accept that client (bound to the source ip / port). Now it can continue listening to the original socket for connection requests while serving the connected client.



Simple TCP Socket Client

Queen Mary University of London

```
// Open your connection to a server, at port 1234
Socket s1 = new Socket("ecs524.eecs.qmul.ac.uk",1234);
// Get file handle from the socket and read input
InputStream s1In = s1.getInputStream();
DataInputStream dis = new DataInputStream(s1In);
String st = new String (dis.readUTF());
System.out.println(st);
// When done, just close the connection and exit
dis.close();
s1In.close();
s1.close();
```

Simple TCP Socket Server

Queen Mary University of London

```
// Register service on port 1234
ServerSocket s = new ServerSocket(1234);
while(true) { //forever {
   Socket s1=s.accept(); // Wait and accept a connection
   // Get a communication stream associated with the socket
   OutputStream s1out = s1.getOutputStream();
   DataOutputStream dos = new DataOutputStream (s1out);
   // Send a string!
   dos.writeUTF("Hi there");
   // Close the connection, but not the server socket
   dos.close();
   s1out.close();
   s1.close();
}
```

TCP demultiplexing

Queen Mary University of London

- TCP connection identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request