COMPUTER COMMUNICATION NETWORKS

Department of Electronics and Communication Engineering

COMPUTER COMMUNICATION NETWORKS

SOCKET PROGRAMMING

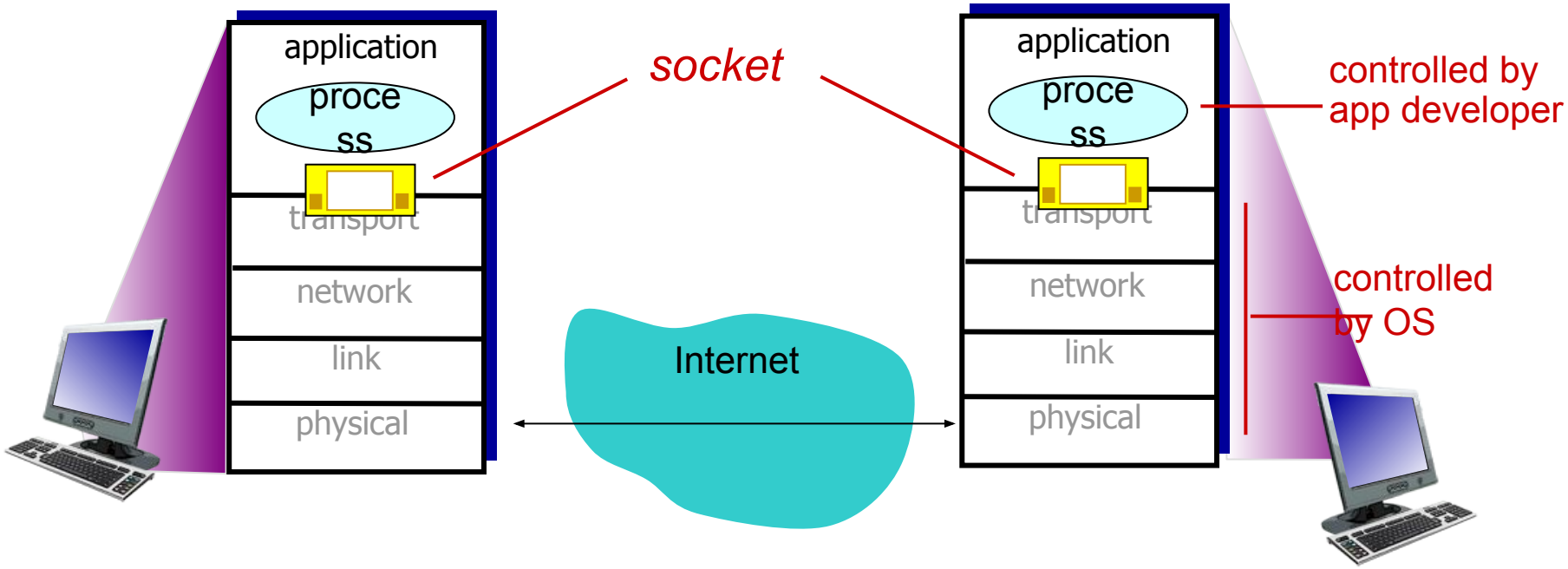
Dr. Arpita Thakre

Department of Electronics and Communication Engineering

SOCKET PROGRAMMING

goal: learn how to build client/server applications that communicate using sockets

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



Socket Programming

Two socket types for two transport services:

UDP: unreliable datagram

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.

SOCKET PROGRAMMING WITH UDP

UDP: no "connection" between client & server

no handshaking before sending data

sender explicitly attaches IP destination address and port # to each packet

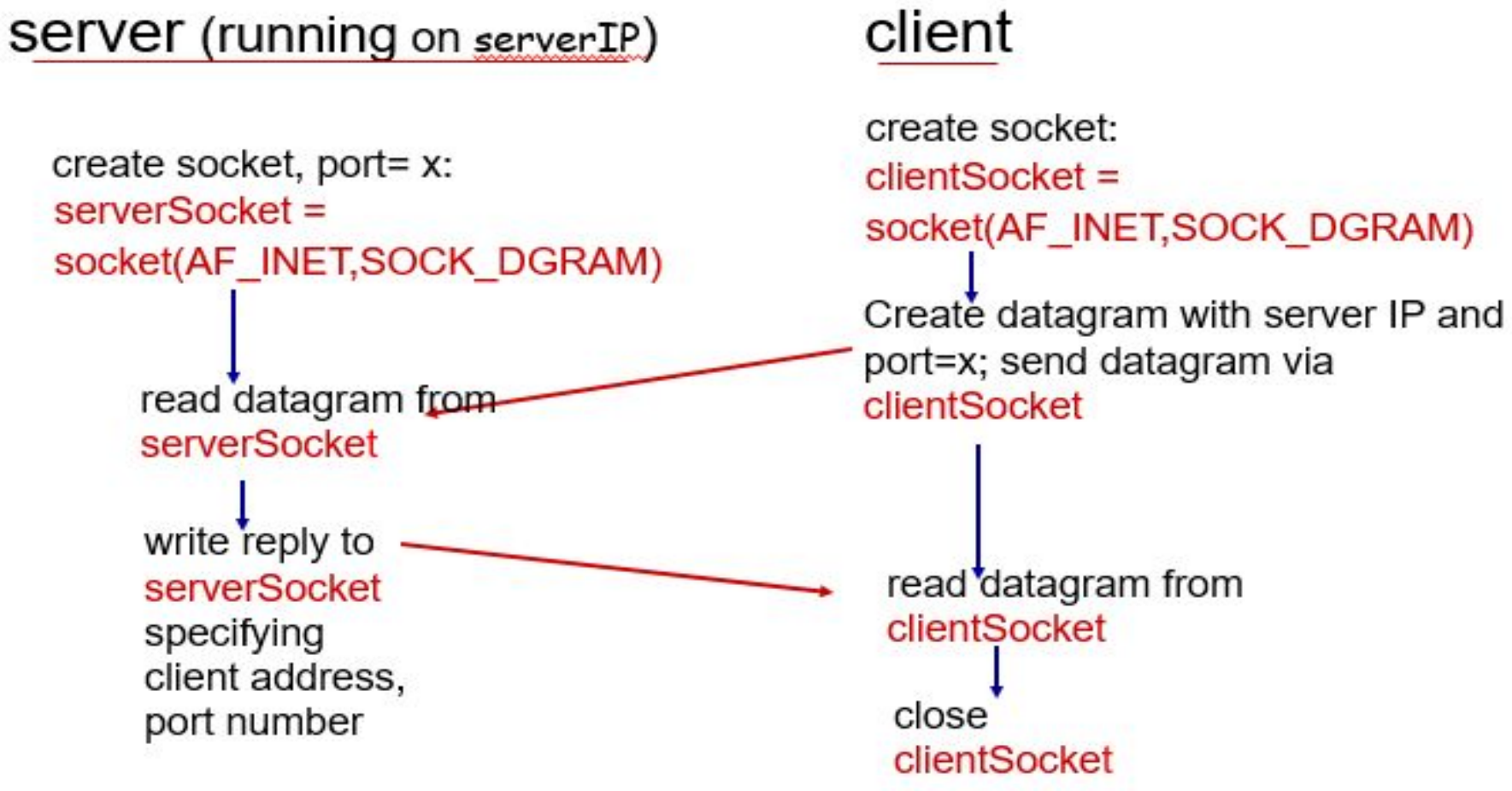
rcvr extracts sender IP address and port# from received packet

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

Application viewpoint:

UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



Example app: UDP client

Python UDPClient

include Python's socket library →
```
from socket import *
```

```
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server →
```
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
```

get user keyboard input →
```
message = raw_input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket →
```
clientSocket.sendto(message,(serverName, serverPort))
```

read reply characters from socket into string →
```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

print out received string and close socket →
```
print modifiedMessage
clientSocket.close()
```

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Socket programming with TCP

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

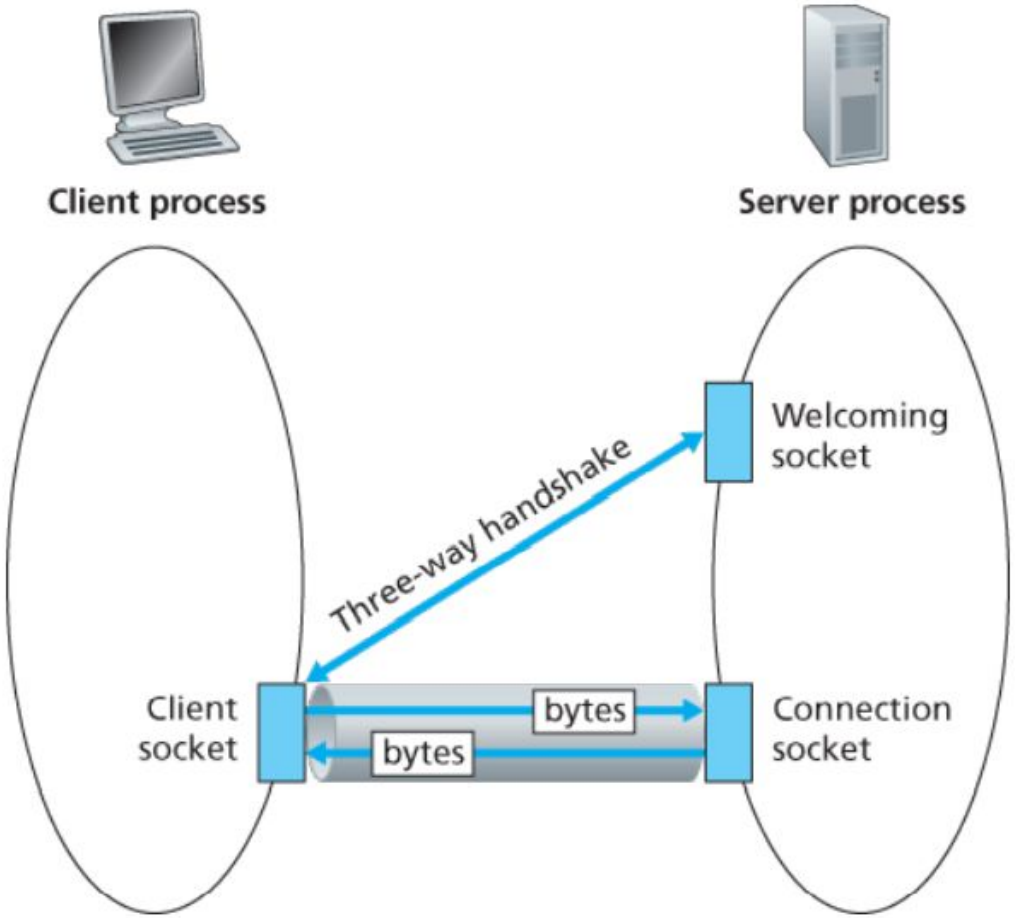
when client creates socket: client TCP establishes connection to server TCP

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.

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

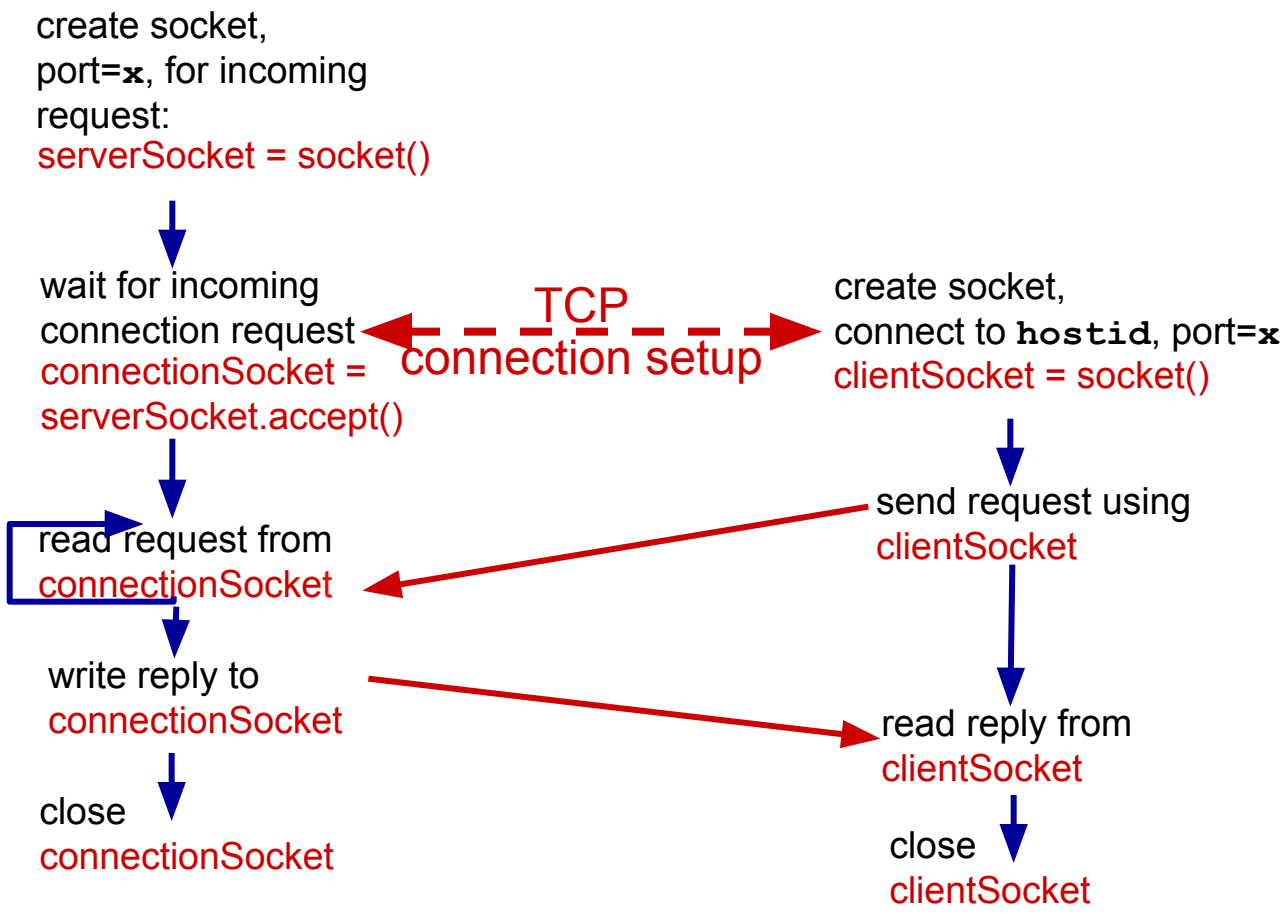
Socket Programming



The `TCPServer` process has two sockets

Client/server socket interaction: TCP

Client/server socket interaction: TCP



Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

THANK YOU

Dr. Arpita Thakre

Department of Electronics and Communication Engineering