DATA COMMUNICATOIN NETWORKING

Instructor: Ouldooz Baghban Karimi

Course Book: Computer Networking, A Top-Down Approach
By: Kurose, Ross

Course Overview

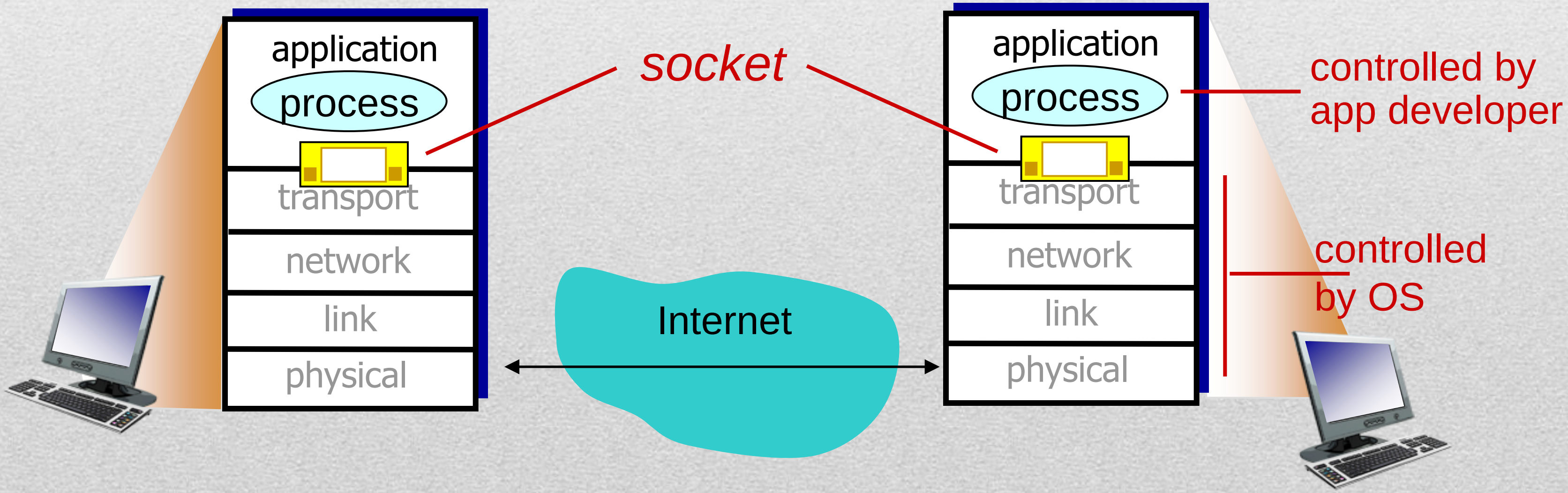
- **Basics of Computer Networks**
  - Internet & Protocol Stack
  - Application Layer
  - Transport Layer
  - Network Layer
  - Data Link Layer

- **Advanced Topics**
  - Case Studies of Computer Networks
  - Internet Applications
  - Network Management
  - Network Security

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

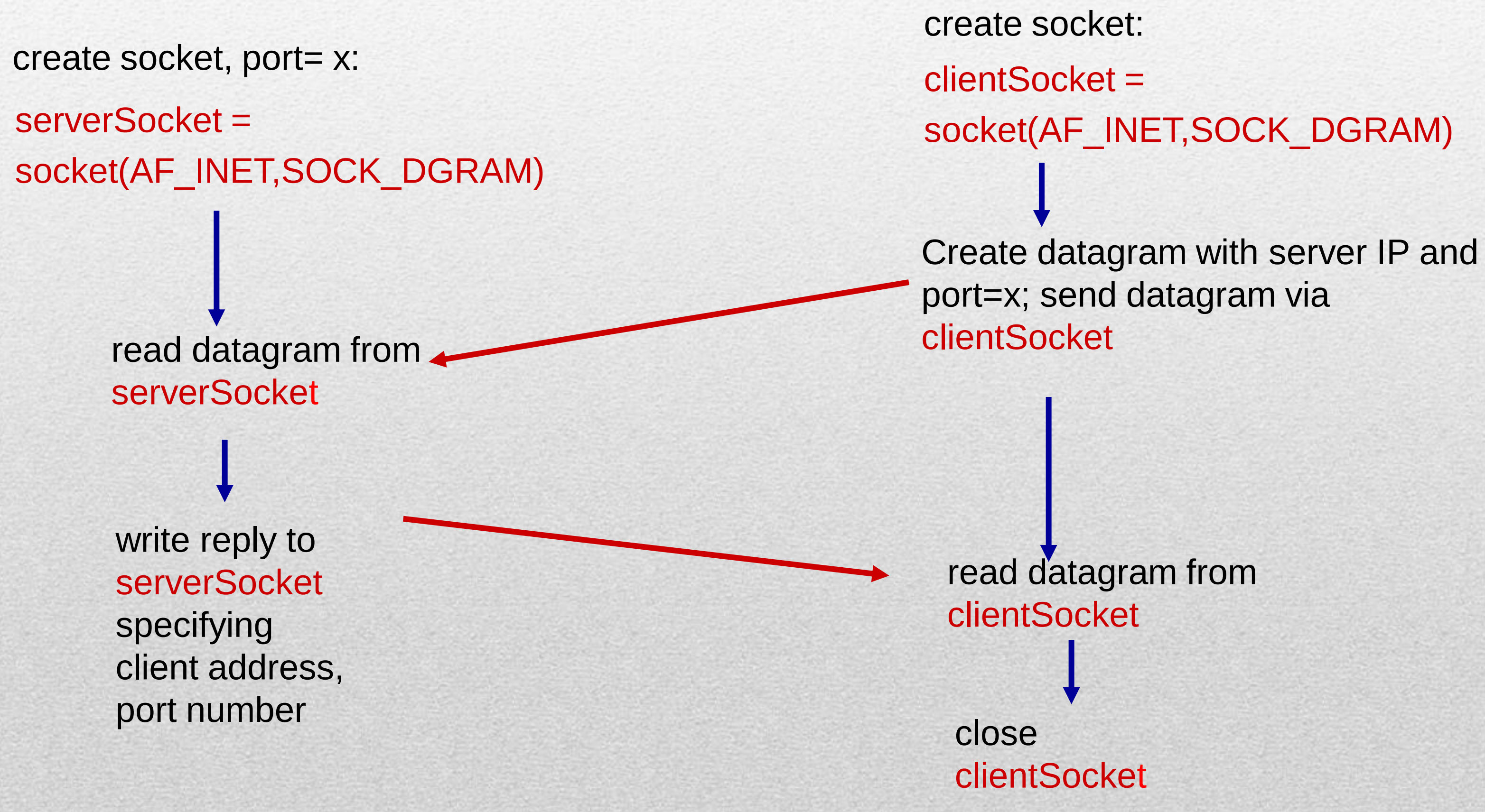
- **Two socket types for two transport services**
  - **UDP:** Unreliable datagram
  - **TCP:** Reliable, byte stream-oriented

- **Application Example**
  - Client reads a line of characters (data) from its keyboard and sends the data to the server.
  - The server receives the data and converts characters to uppercase.
  - The server sends the modified data to the client.
  - The client receives the modified data and displays the line on its screen.

Socket Programming with UDP

- **No connection between client and server**
  - No handshaking before sending data
  - Sender explicitly attaches IP destination address and Port # to each packet
  - Receiver extracts sender IP address and port # from received packets
- **Lost or out of order data at the receiver**
  - UDP provides UNRELIABLE transfer of groups of bytes (Datagrams) between client and server

Client/Server Socket Interaction: UDP



server (running on serverIP)

create socket, port= x:

serverSocket =

socket(AF_INET,SOCK_DGRAM)

read datagram from serverSocket

write reply to serverSocket specifying client address, port number

client

create socket:

clientSocket =

socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and port=x; send datagram via clientSocket

read datagram from clientSocket

close clientSocket

Example App: UDP Client

Python UDPClient

include Python’s socket library

create UDP socket for server

get user keyboard input

Attach server name, port to message; send into socket

read reply characters from socket into string

print out received string and close socket

```
from socket import *
serverName = ‘hostname’
serverPort = 12000
clientSocket = socket(socket.AF_INET,
                                      socket.SOCK_DGRAM)
message = raw_input(’Input lowercase sentence:’)
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress =
                                      clientSocket.recvfrom(2048)
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

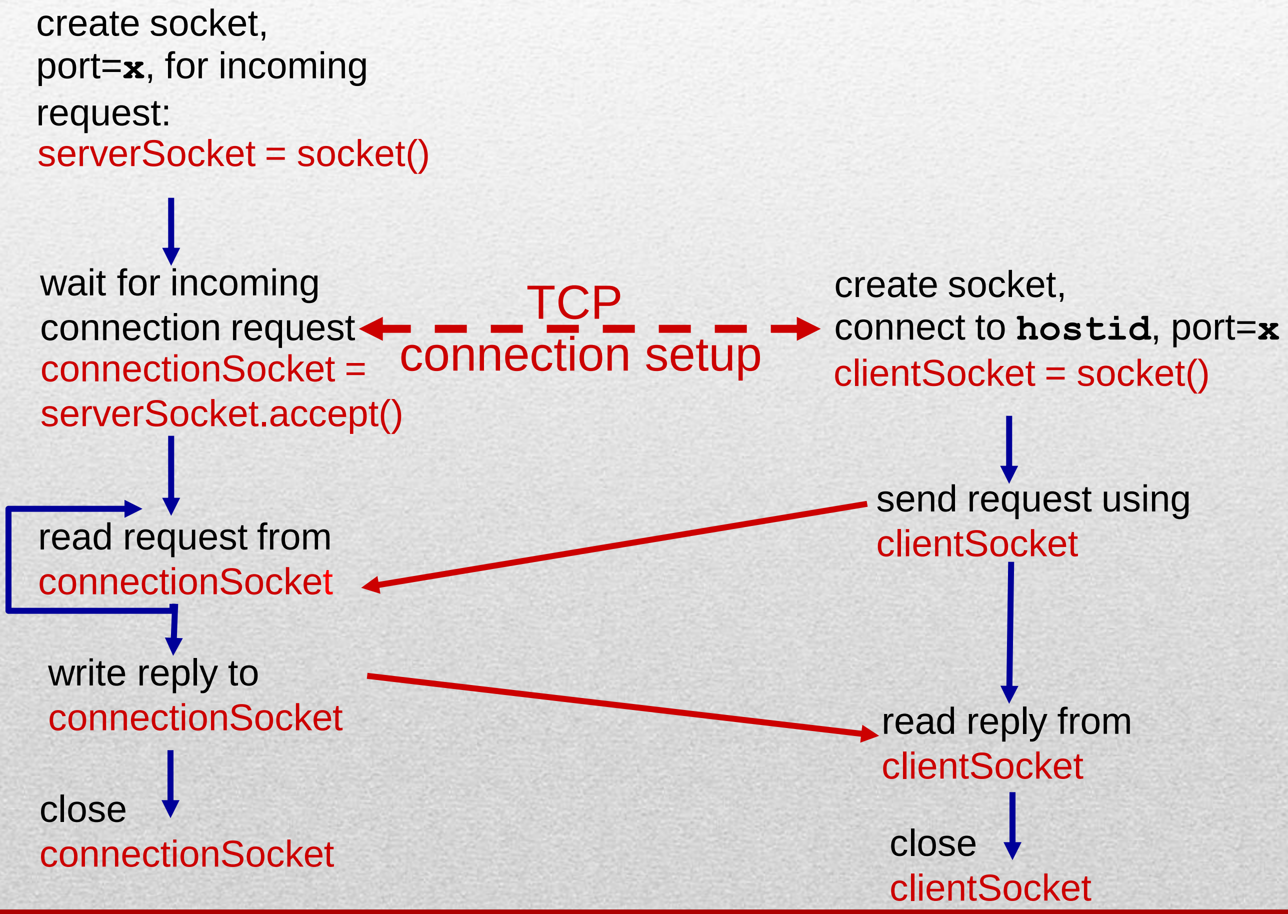
- **Server**
  - When contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

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

Client/Server Socket Interaction: TCP

server (running on **hostid**)

client



create socket,
port=**x**, for incoming
request:
serverSocket = socket()

wait for incoming
connection request
connectionSocket =
serverSocket.accept()

TCP
connection setup

create socket,
connect to **hostid**, port=**x**
clientSocket = socket()

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

read reply from
clientSocket

close
connectionSocket

close
clientSocket

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)

Application Layer

- **Application Architectures**
  - P2P
  - Client-Server
- **Application Service Requirements**
  - Reliability
  - Bandwidth
  - Delay
- **Specific Protocols**
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- **Socket Programming**

Protocols

- **Control/data Messages**
  - In-band
  - Out-of-band
- **State**
  - Stateless
  - Stateful
- **Reliability**
  - Reliable
  - Unreliable
- **Implementation**
  - Centralized
  - Distributed
- **Message Exchange**
  - Format
    - Header
    - Data
  - Client/Server