COMP 445
Data Communications & Computer networks
Winter 2022

Application Layer – Part 4

- ✓ Socket programming
  - ✓ Sockets with UDP
  - ✓ Sockets with TCP

Learning objectives

- To describe the way open and proprietary network applications are created
- To explain how sockets are established and the differences when using sockets with UDP and sockets with TCP
- To use the socket API to build network applications

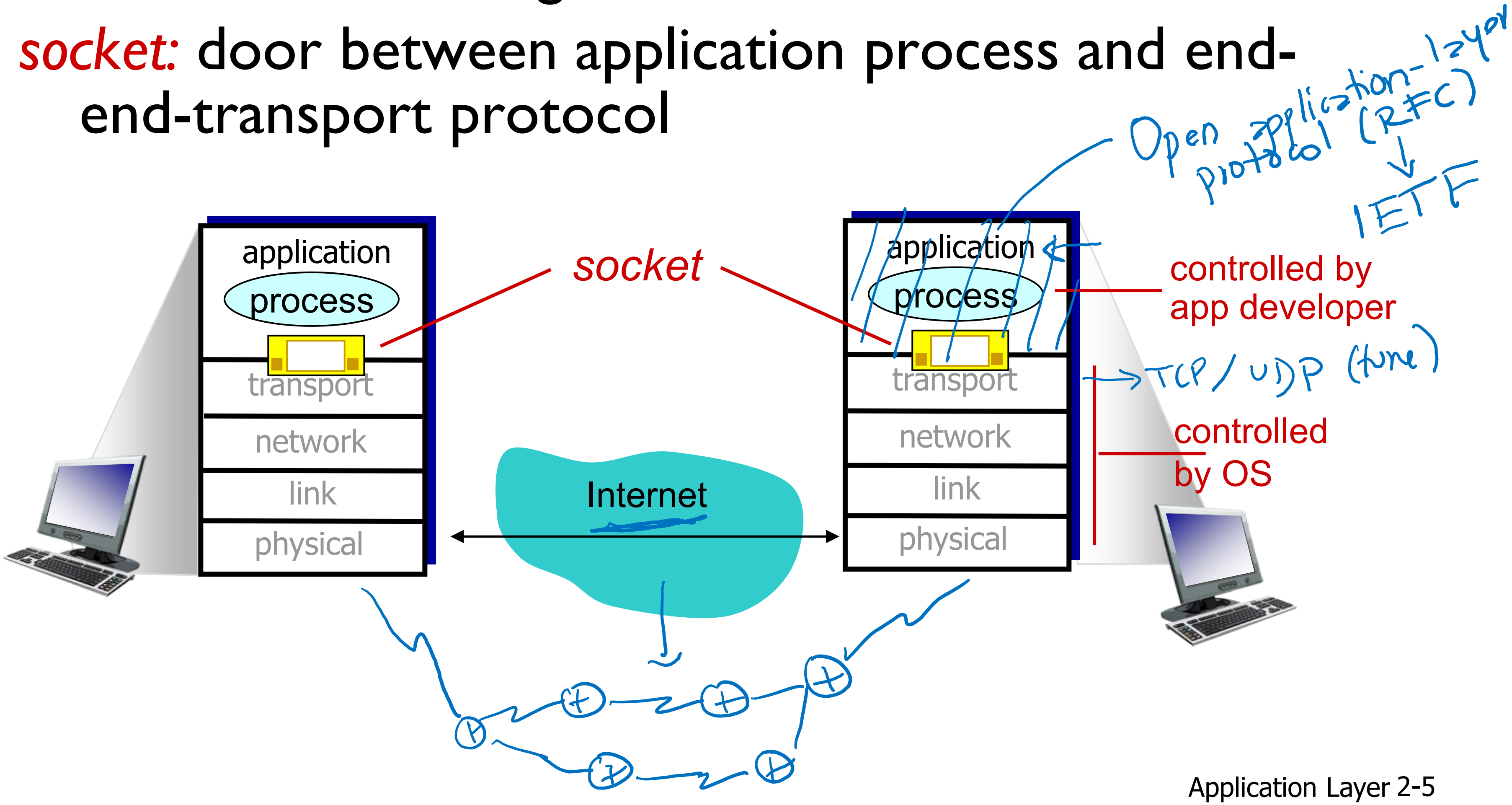
Application Layer – Part 4

- ✓ Socket programming
  - ✓ Sockets with UDP
  - ✓ Sockets with TCP

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 (one-shot)
- *TCP:* reliable, byte stream-oriented

Application Example:

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

Socket programming *with UDP*

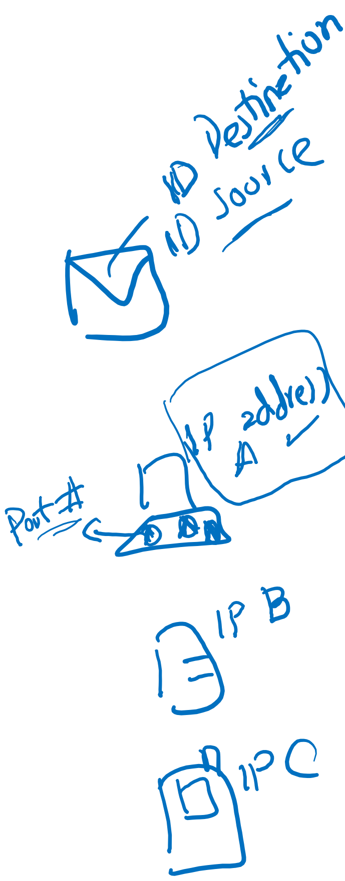
UDP: no "connection" between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

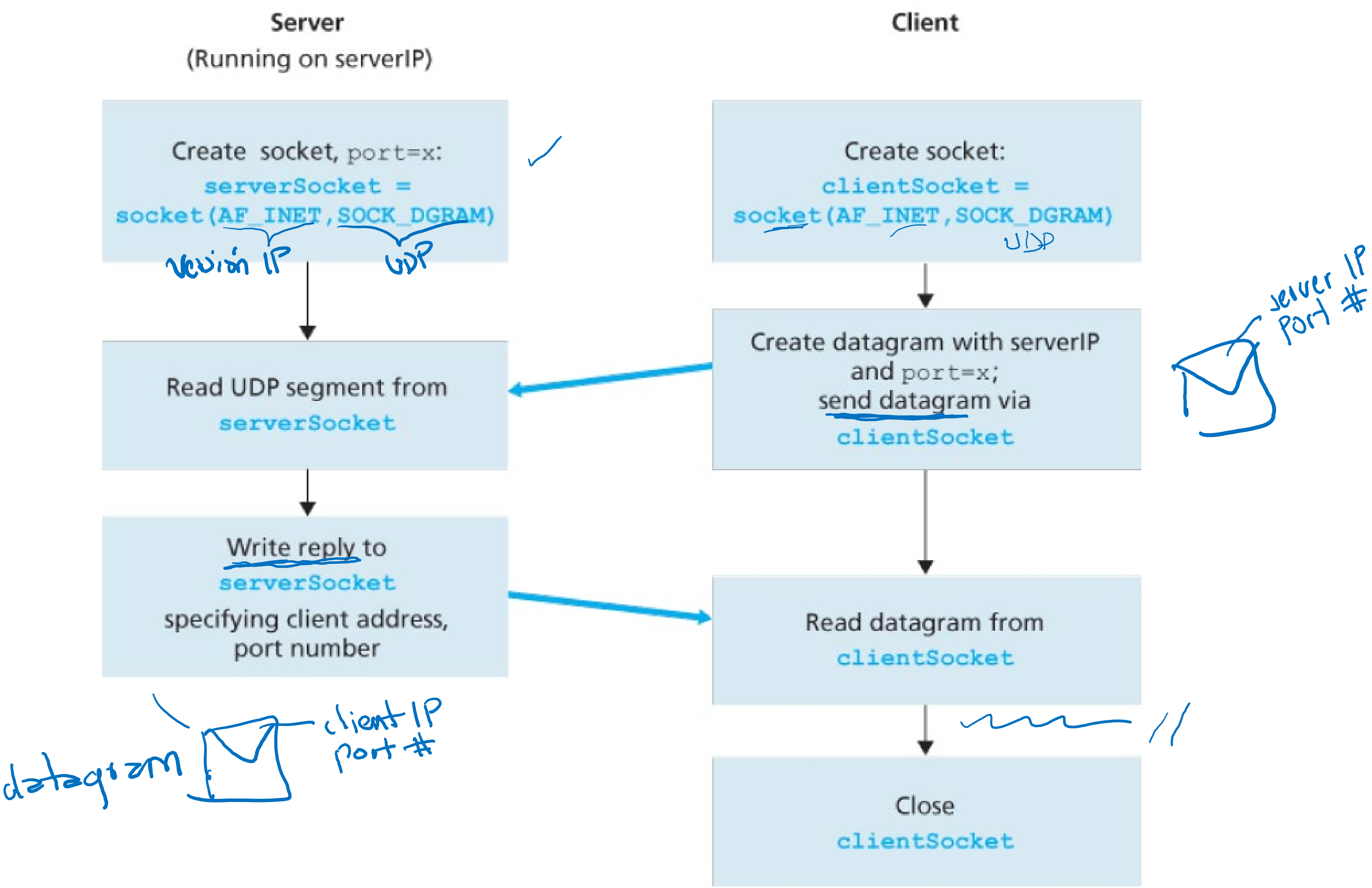
UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

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



Client/server socket interaction: UDP



Server
(Running on serverIP)

Create socket, port=x:
`serverSocket = socket(AF_INET,SOCK_DGRAM)`

Read UDP segment from
`serverSocket`

Write reply to
`serverSocket`
specifying client address, port number

Client

Create socket:
`clientSocket = socket(AF_INET,SOCK_DGRAM)`

Create datagram with serverIP and port=x;
send datagram via
`clientSocket`

Read datagram from
`clientSocket`

Close
`clientSocket`

Example app: UDP client

Python UDPClient

include Python's socket library →

```
from socket import *
serverName = ‘hostname’
```

hostname → (DNS)

IP address

```
serverPort = 12000
```

create UDP socket for server →

```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```

UDP

get user keyboard input →

```
message = raw_input(’Input lowercase sentence:’)
```

Attach server name, port to message; send into socket →

```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```

bytes

read reply characters from socket into string →

```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```

print out received string and close socket →

bytes

```
print modifiedMessage.decode()
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 True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket

UDP

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

bytes

(IP client , Port # client)

Application Layer – Part 4

- ✓ Socket programming
  - ✓ Sockets with UDP
  - ✓ Sockets with TCP

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:

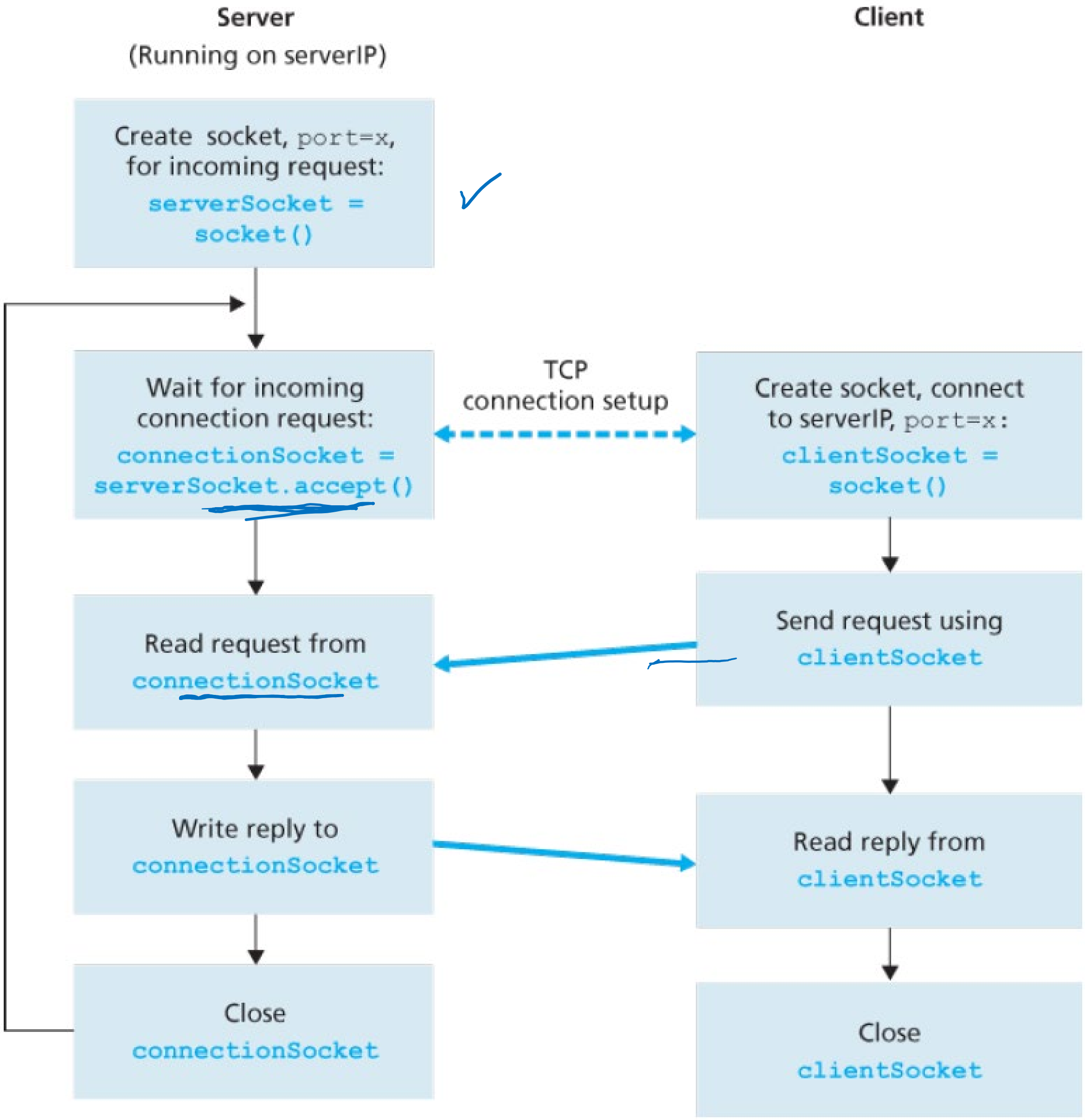
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- 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 (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 serverIP)

Create socket, port=x, for incoming request:

`serverSocket = socket()`

Wait for incoming connection request:

`connectionSocket = serverSocket.accept()`

Read request from `connectionSocket`

Write reply to `connectionSocket`

Close `connectionSocket`

TCP connection setup

Client

Create socket, connect to serverIP, port=x:

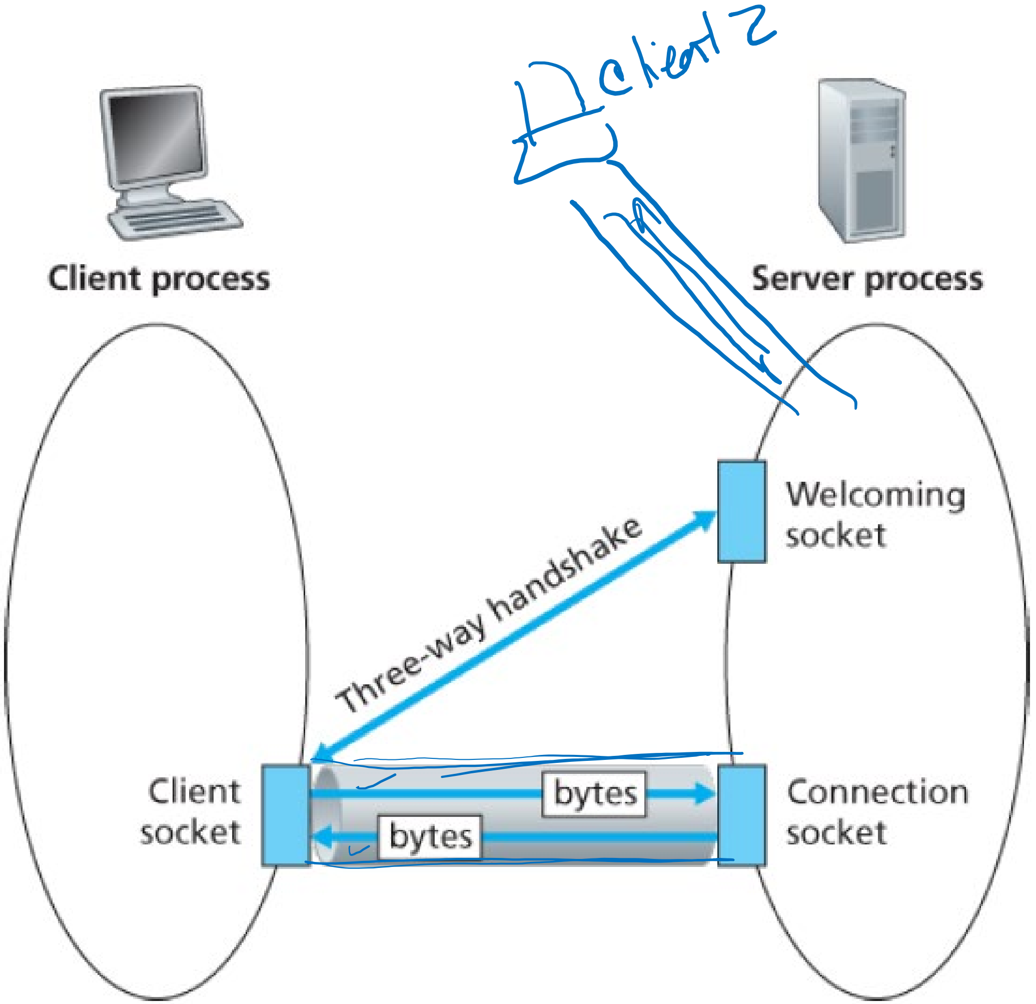
`clientSocket = socket()`

Send request using `clientSocket`

Read reply from `clientSocket`

Close `clientSocket`

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.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

IPv4

TCP

bytes

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 True:
     connectionSocket, addr = serverSocket.accept()

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

TCP

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)

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP ✓
  - unreliable, datagrams: UDP ✓

- specific protocols:
  - HTTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent
- video streaming, CDNs
- socket programming:

  TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

important themes:

- control vs. messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- "complexity at network edge"

References

Figures and slides are taken/adapted from:

- Jim Kurose, Keith Ross, “Computer Networking: A Top-Down Approach”, 7th ed. Addison-Wesley, 2012. All material copyright 1996-2016 J.F Kurose and K.W. Ross, All Rights Reserved