IOWA

CS3640

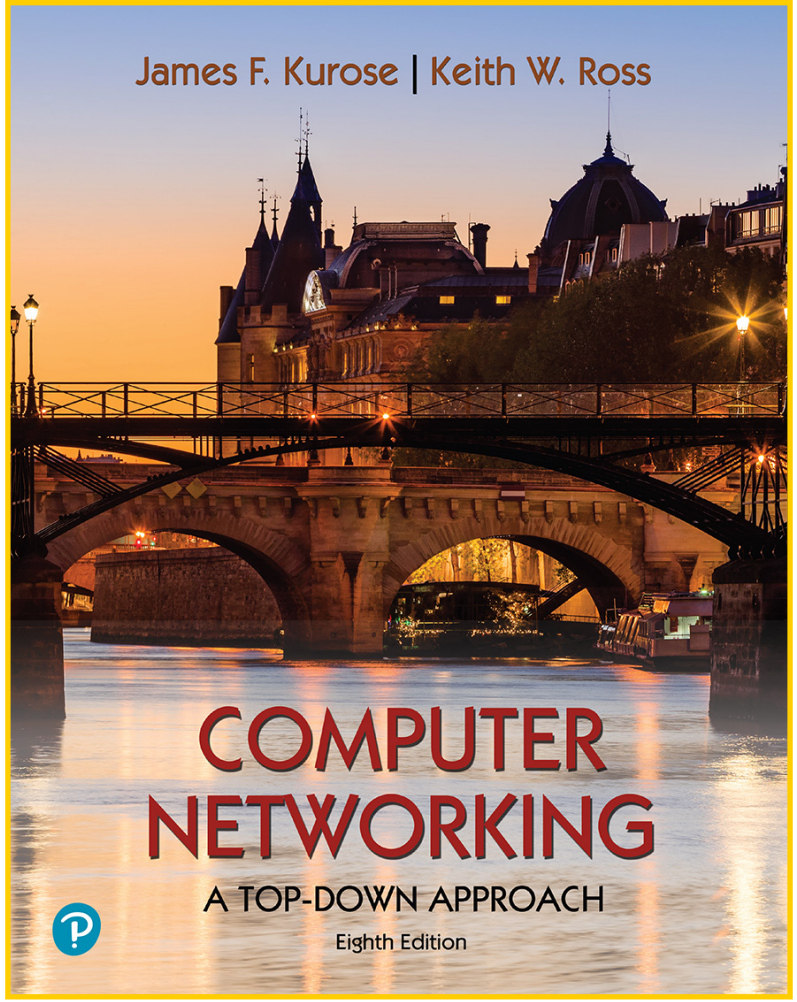
Application Layer (6): Socket Programming

Prof. Supreeth Shastri
Computer Science
The University of Iowa

Lecture goals

learn how to build client-server applications that communicate using sockets

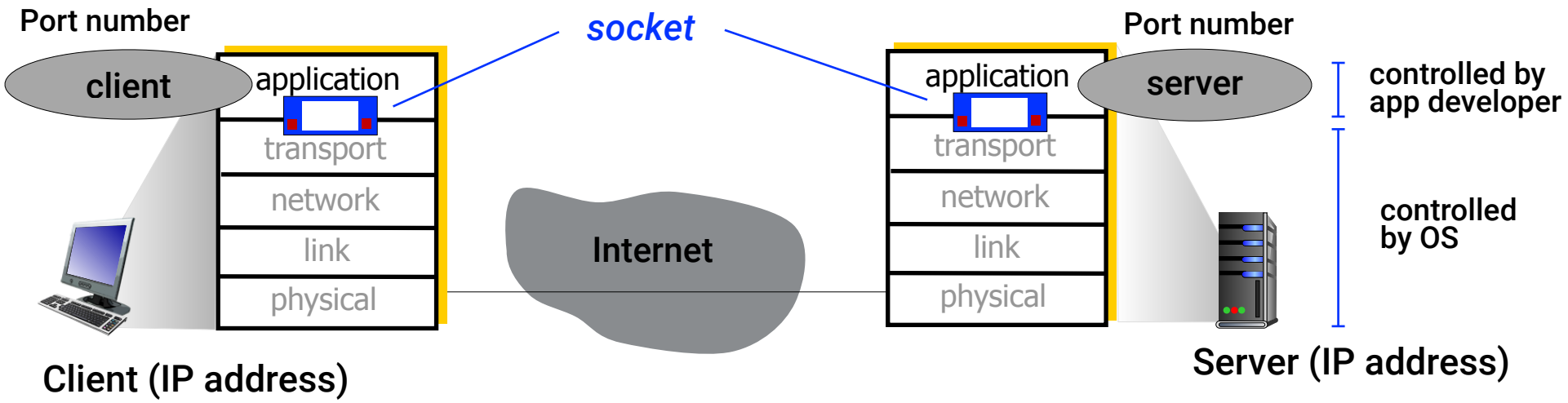
- *Socket programming*
- *UDP sockets*
- *TCP sockets*



Chapter 2.7

IOWA

Sockets: network programming API



Q: Do we have to use the socket APIs? Can't we create the whole packet and push it on to the Internet?

Socket fundamentals

Socket abstraction

- Originally defined in RFC 147 (in 1971) for ARPAnet
- First open implementation of sockets was by Berkeley in 1983
- After POSIX standardization, all operating systems have adapted Berkeley sockets as the de facto networking API

Socket types govern the exposed transport services

- **UDP sockets** for unreliable datagrams
- **TCP sockets** for reliable, flow- and congestion-controlled data streams
- **Raw sockets** for directly sending and receiving IP packets

Socket programming

Types of networking applications

- **Open**: conform to the rules laid out in the RFCs (or other standards). E.g., HTTP browsers and web servers that interoperate without being developed by same organization/developers
- **Proprietary**: applications whose behavior is not openly documented; may change over time without any notice. E.g., Skype and Zoom

A simple app for this lecture

- client reads a line of characters (data) from its keyboard and sends data to server
- server receives the data and converts characters to UPPERCASE
- server sends modified data back to client
- client receives modified data and displays line on its screen

Q: does this make our application open or proprietary?

Hands-on Socket Programming

TCP

UDP

Mechanics of UDP

Application viewpoint

UDP provides unreliable transfer for a group of bytes (“datagrams”) between client and server processes

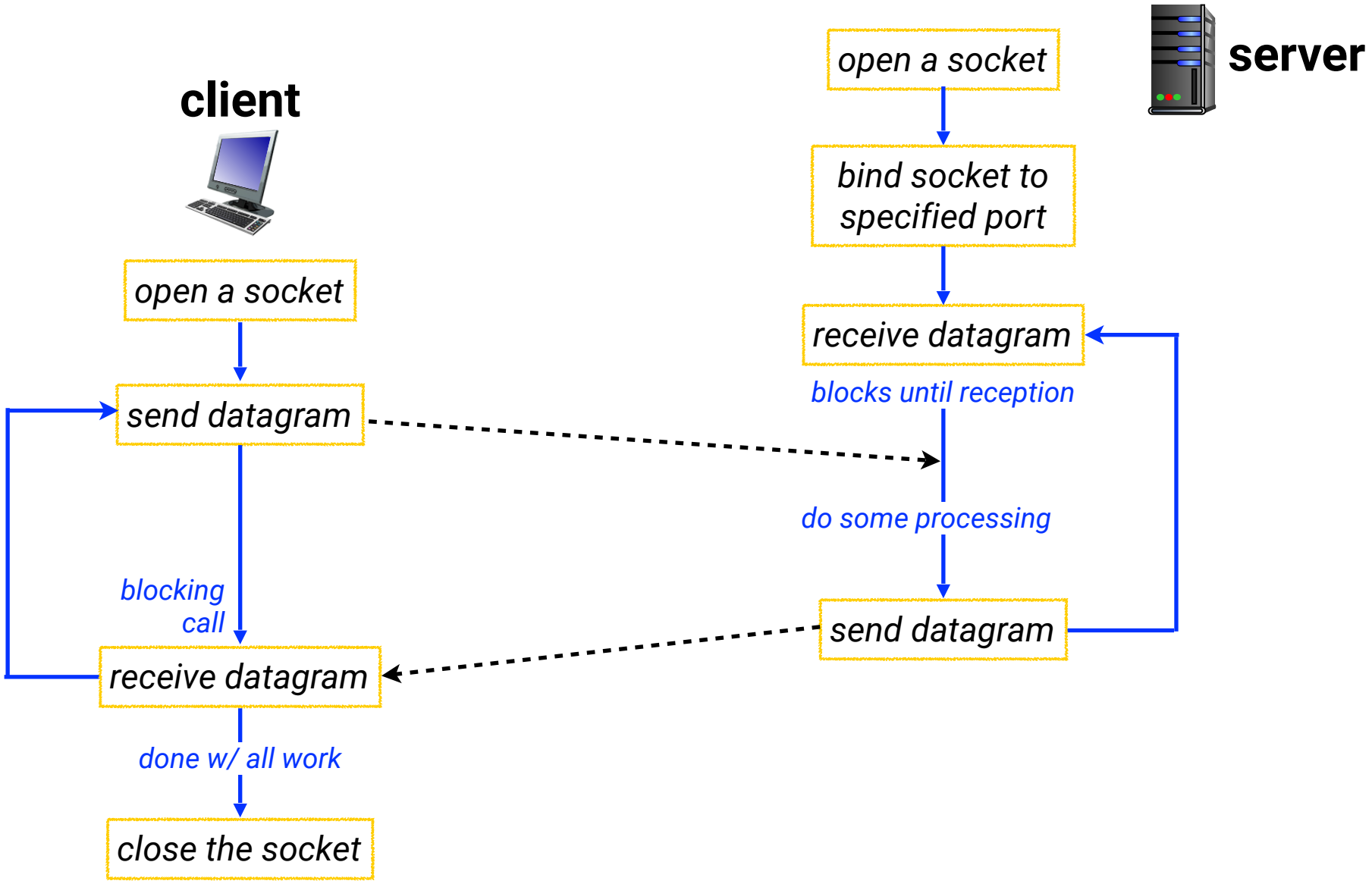
There is no “connection” between client and server

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

Minimal expectations

- transmitted data may be lost, may arrive out of order, and may overwhelm the receiver
- UDP does not monitor network congestion, nor does it have to respond to it

UDP Client Server Interaction



Coding up the UDP client

include Python's socket library →

create UDP socket for server →

get user keyboard input →

attach server name, port to message; send it 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(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

Coding up the UDP server

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 →

```
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)
```

Demo: UDP client-server

Mechanics of TCP

Application viewpoint

TCP provides a reliable, in-order byte-stream (“pipe”) between client and server processes

First, server must be setup to accept connections

- server process must be continuously running
- server must have created the socket that welcomes client’s contact

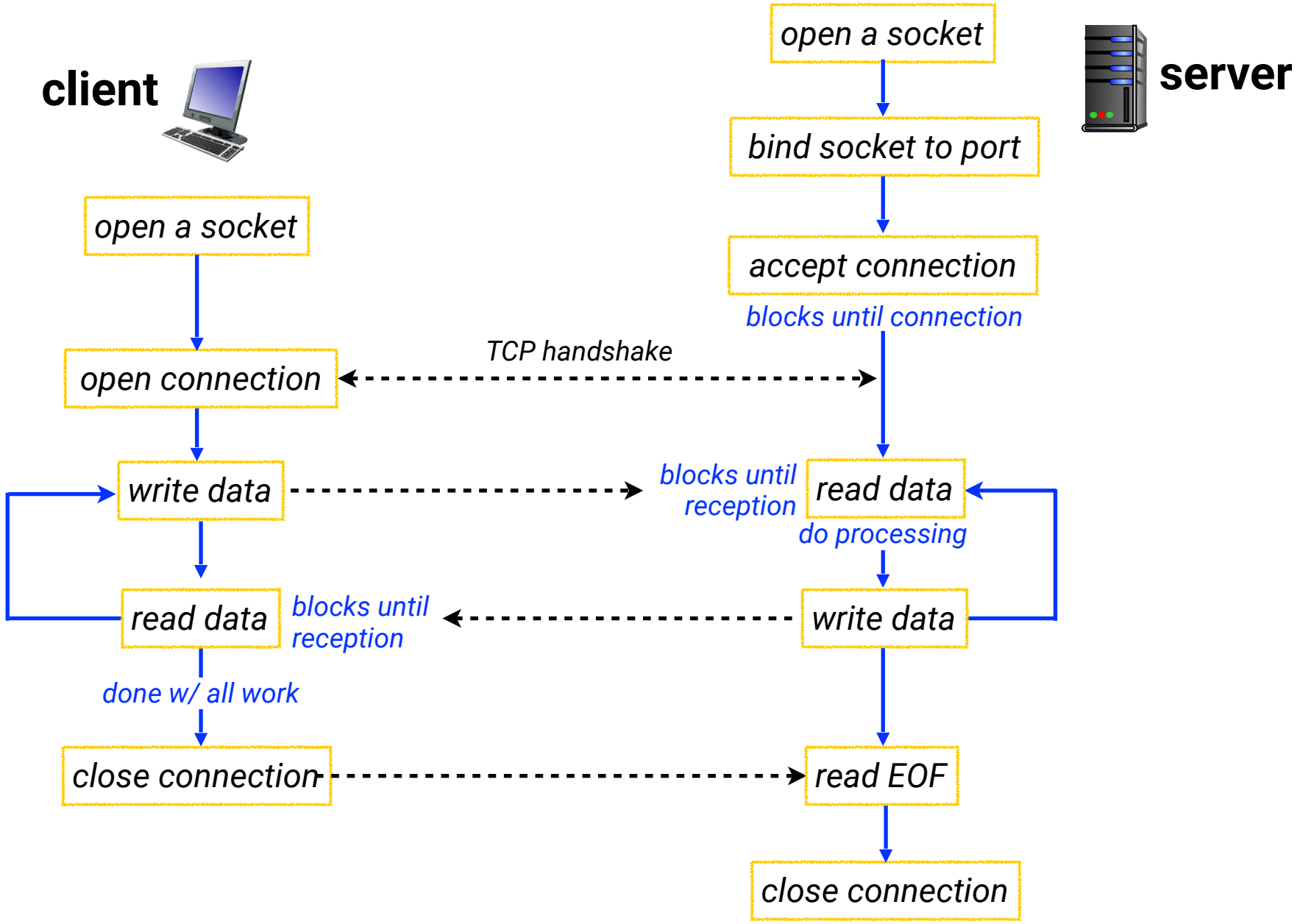
Then, a client can contact the server by

- creating a TCP socket and specifying server’s IP address and port number
- when client creates socket, client’s TCP establishes a connection to server TCP

On the server side

- when contacted by a client, server TCP spawns a new socket to manage communications with that client
- this feature allows the server to talk with multiple clients

TCP Client Server Interaction



Coding up the TCP client

create TCP socket for server

perform TCP handshake

No need to attach server name, port

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

Coding up the TCP server

create TCP welcoming socket

server begins listening for incoming TCP requests

server waits on incoming requests, new socket created on return

read and write bytes from/to socket (no IP addr/port as in UDP)

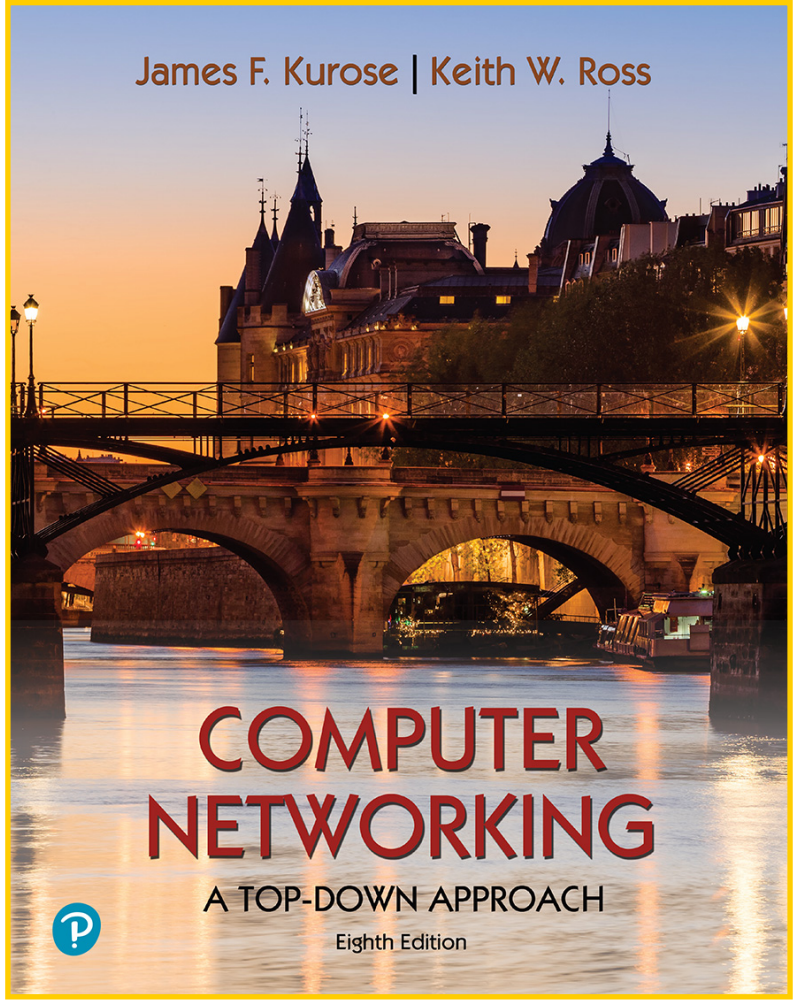
close connection to this client (but *not* welcoming socket)

```
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()
```

Next lecture

understand the principles and organization of the transport layer

- *Services*
- *Internet's transport protocols*
- *Multiplexing and demultiplexing*



Chapter 3.1 - 3.2

IOWA

Spot Quiz (ICON)