Application Layer
Socket Programming

Prof. Anja Feldmann, Ph.D.

Thorben Krüger, MSc

(Based on slide deck of Computer Networking, 7th ed., Jim Kurose and Keith Ross.)

Application Layer Protocols

- HTTP
- DNS
- Email (SMTP/IMAP/POP3)
- *Your own protocol?*

Data exchange between hosts: General

- How do we get two hosts to exchange arbitrary data?
  - Without trying to use HTTP or SMTP or IMAP

Sockets!

What are sockets?

- Abstract representation of a network connection on application level
  - Corresponding API provided by the host‘s OS
  - OS responsible for actual data transmission
  - Application responsible for content
- Makes sending data to a connected remote host similar to simply writing data to a file
  - Receiving data is similar to reading from a file

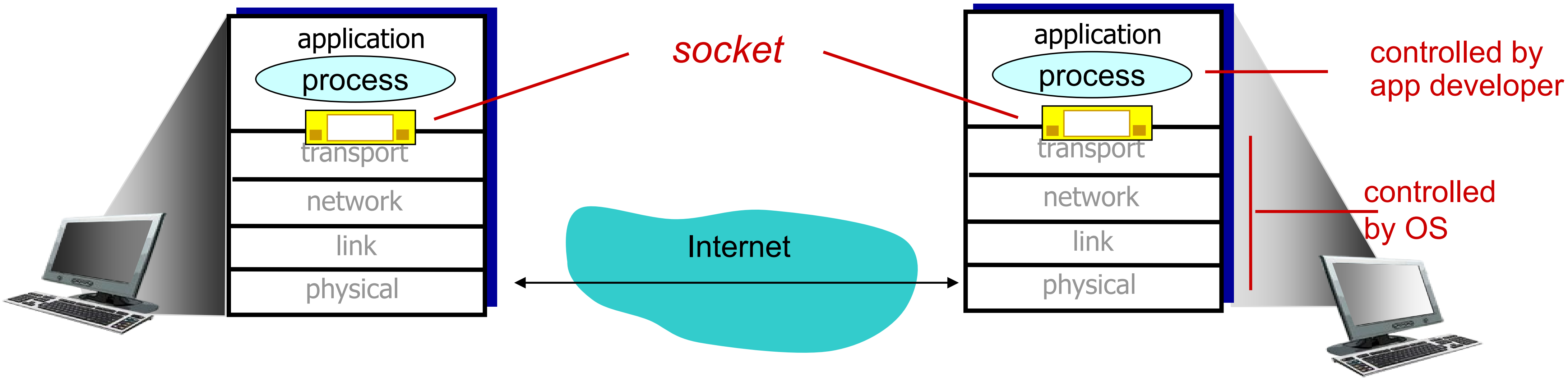
How do Sockets releate to Applications?

- Browsers, Webservers
  - Use sockets to speak HTTP
- Mailservers, Mailclients
  - Use sockets to speak SMTP/IMAP/POP3
- Peer 2 Peer Apps
  - Use sockets to speak, e.g., Bittorrent

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 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

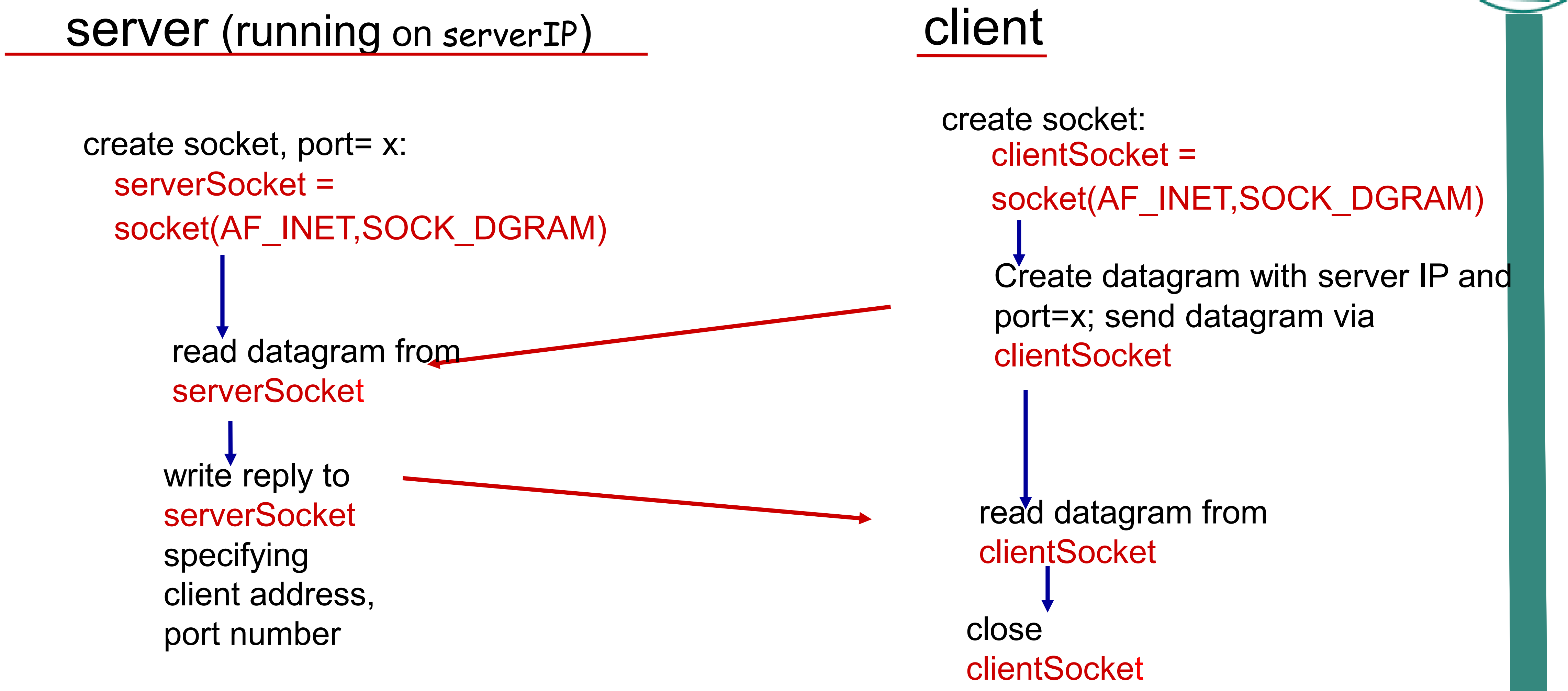
Socket programming *with UDP*

- UDP: No “connection” between client & server
  - No handshaking before sending data
  - Sender explicitly attaches IP dst address and port # to each packet
  - Receiver extracts src 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)

client

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

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

```
from socket import *
serverName = ‘hostname’
serverPort = 12000
```

create UDP socket for server

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

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

print out received string and close socket

```
print modifiedMessage.decode()
clientSocket.close()
```

Example app: UDP Server

Python UDPServer

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)
```

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:

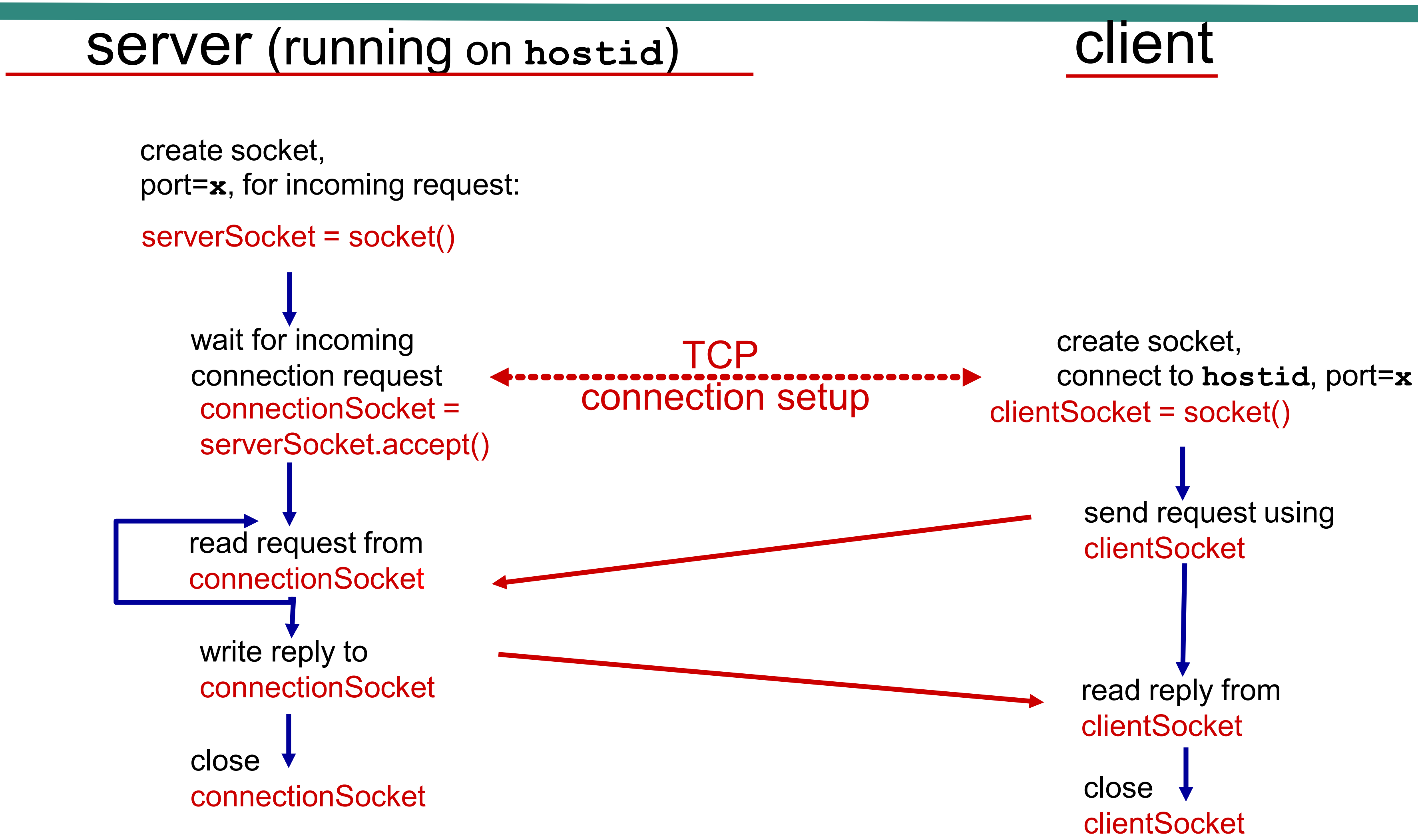
- Creating TCP socket, specifying IP address, port number of server process
- *When client creates socket:* Client TCP establishes connection to server TCP

- 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

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.encode())
modifiedSentence = clientSocket.recv(1024)
print (‘From Server:’, modifiedSentence.decode())
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 True:
    connectionSocket, addr = serverSocket.accept()

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

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
- CDNs
- Socket programming:

  TCP, UDP sockets