50.012 Networks

Lab 1 tutorial: Socket Programming & Proxy

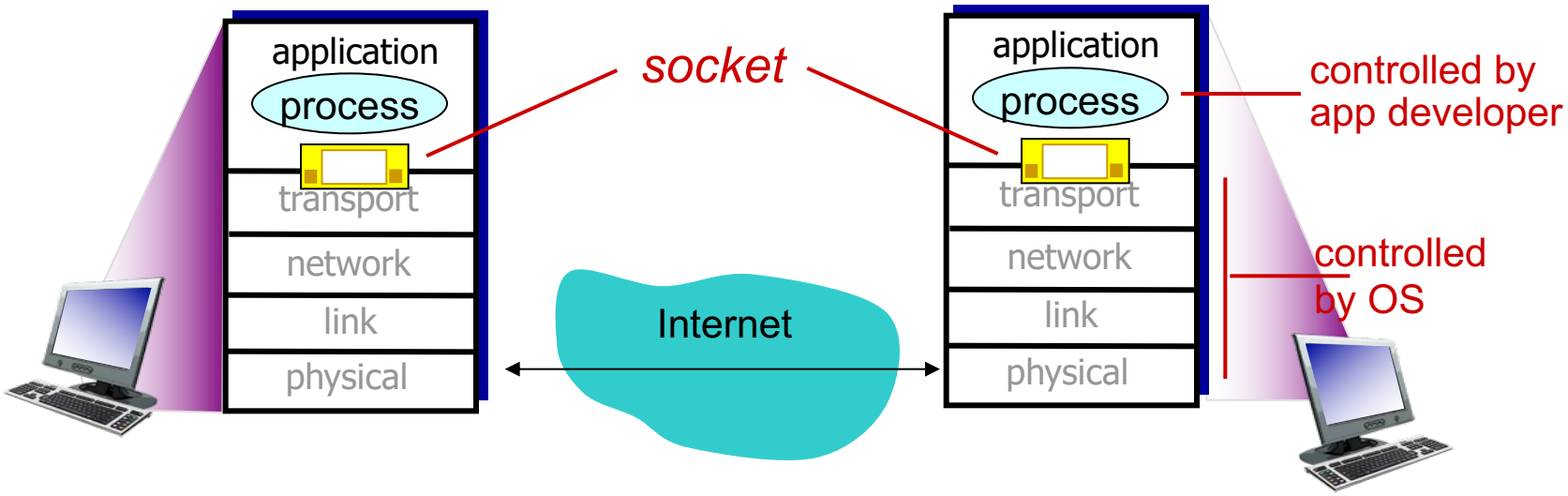
2023 Term 6

Assoc. Prof. CHEN Binbin

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Socket programming

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



Socket programming

Two socket types for two transport services:

- *UDP (User Datagram Protocol):* unreliable datagram
- *TCP (Transmission Control Protocol):* reliable, byte stream-oriented

We will use a simple network app as 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

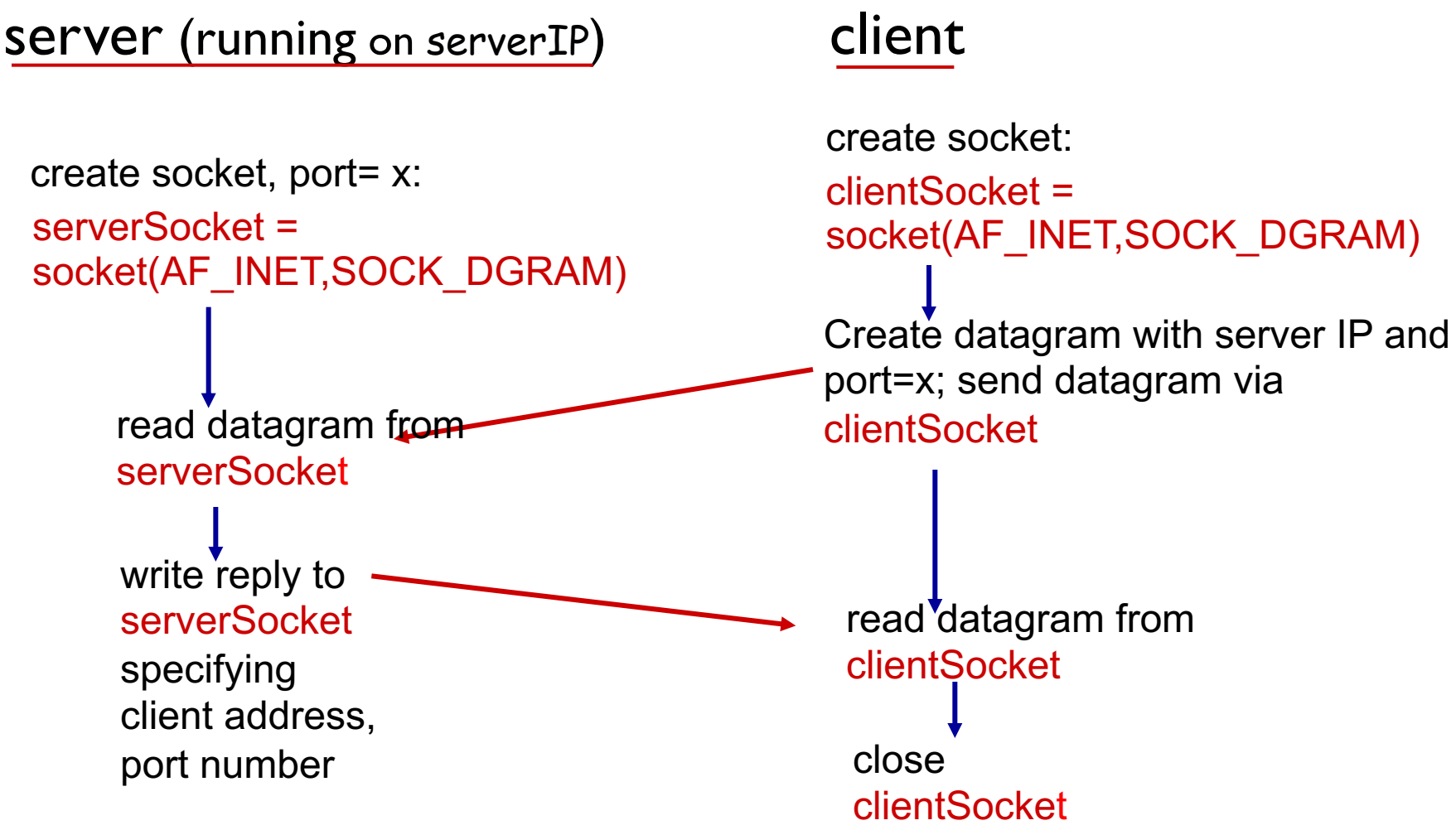
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

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



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))
```

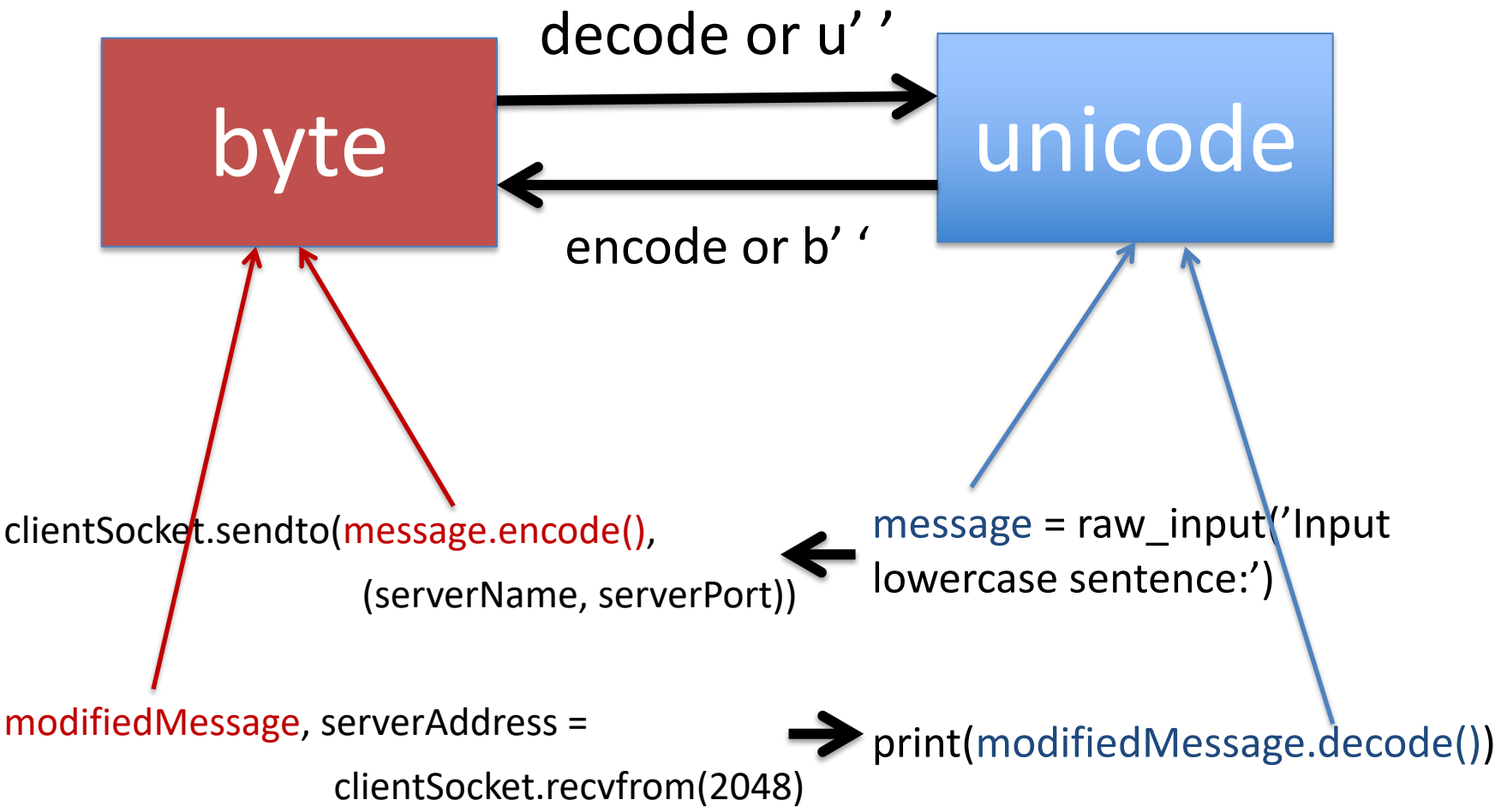
read reply characters from socket into string →

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

print out received string and close socket →

```
print('From Server:', modifiedMessage.decode())
clientSocket.close()
```

Conversion between byte & unicode str in Python 3



https://nedbatchelder.com/text/unipain/unipain.html#1

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

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:

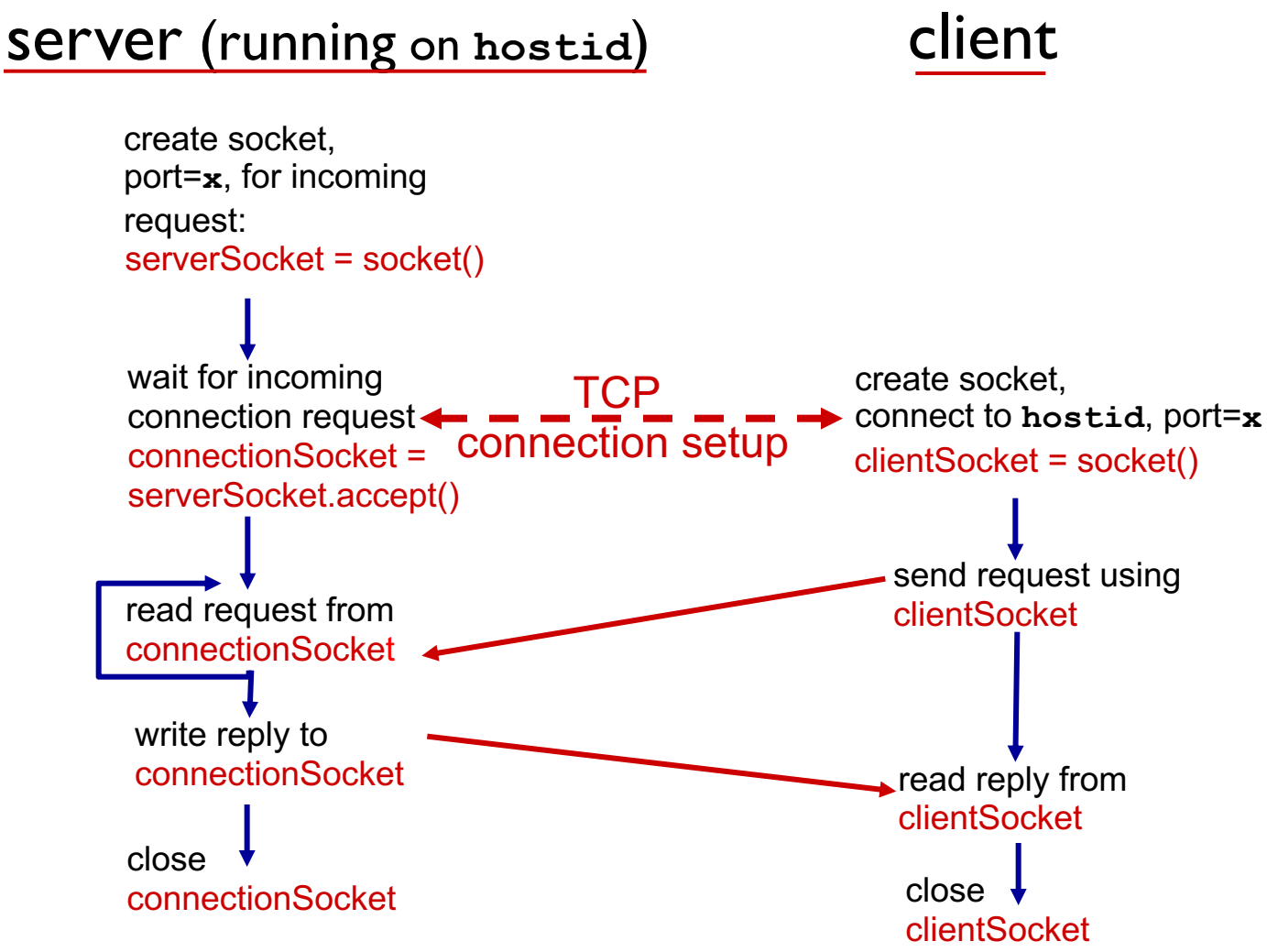
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- 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

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()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

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

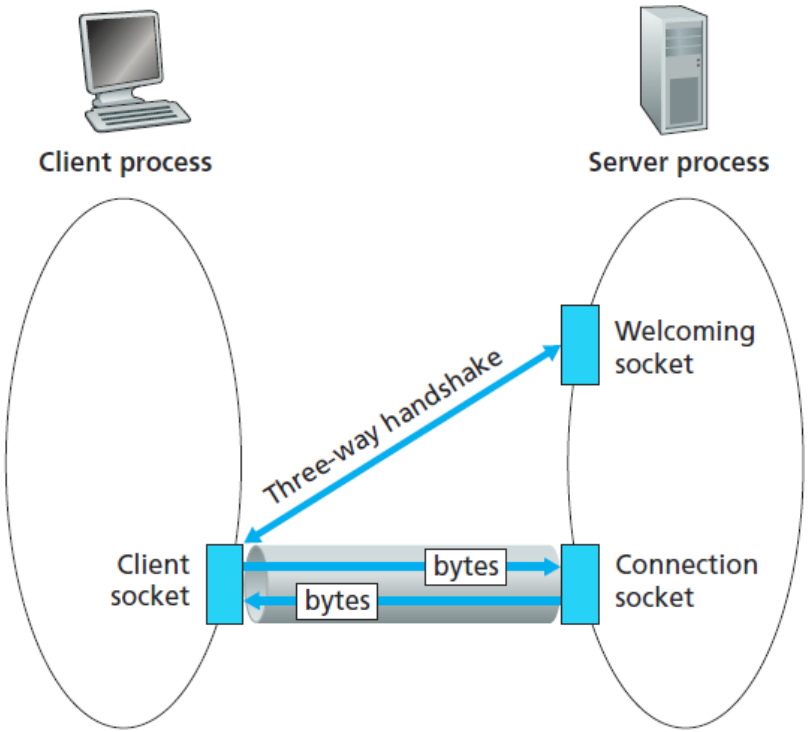
read bytes from socket (but not address as in UDP)

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

```
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 sockets



Client IP: open, client port: open
Server IP: Y.Y.Y.Y Server port: 80

Client IP: X.X.X.X, client port: a
Server IP: Y.Y.Y.Y Server port: 80

Comparison: UDP vs. TCP socket

- Socket establishment

UDP client

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

TCP client

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
```

UDP server

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
```

TCP server

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
```

Comparison: UDP vs. TCP socket

- Use of socket

UDP client

```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
modifiedMessage, serverAddress =
                clientSocket.recvfrom(2048)
```

TCP client

```
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
```

UDP server

```
 message, clientAddress =
serverSocket.recvfrom(2048)
serverSocket.sendto(message, clientAddress)
```

TCP server

```
connectionSocket, addr = serverSocket.accept()
message = connectionSocket.recv(1024)
connectionSocket.send(message)
```

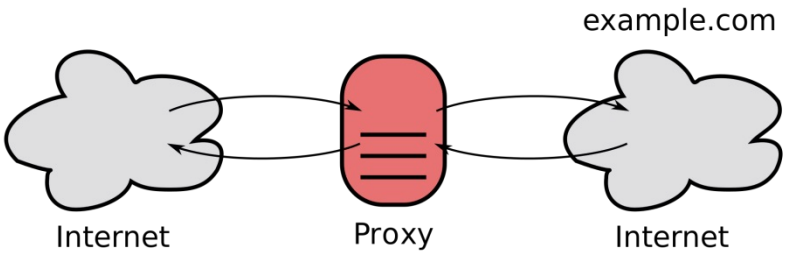
Proxy

Web Proxies

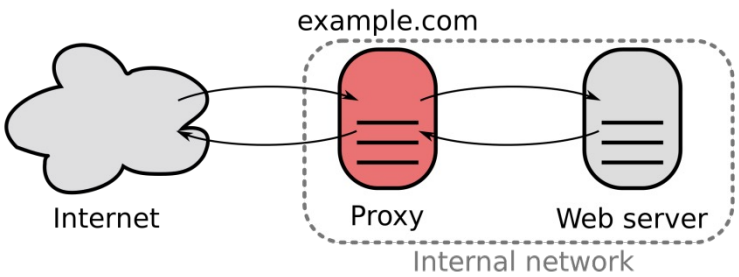
- Proxy: an entity authorized to act on behalf of another
  - An intermediate server that is performing requests for us
- A caching proxy keeps copies of resources for the client
  - E.g., results of HTTP GET queries
  - Results of non-idempotent operations (e.g., POST) are not cached
- These cached results are served to subsequent queries
  - These clients do not have to be the same as original clients
  - As long as GET was requesting the same resource

Types of Proxies

- Forward (Open) Proxies
  - Content accelerators: by reducing delay and load on outgoing connections
  - Content filters / access control (or flip it: bypass filtering)
  - Content logging and eavesdropping (or flip it: accessing services anonymously)
- Reverse Proxies can also cache queries in front of servers
  - Application firewall
  - TLS acceleration
  - Distribute the load, A/B testing, and multivariate testing
  - Accelerators: Cache / compression



An open proxy



A reverse proxy