Python Network Programming

David M. Beazley
http://www.dabeaz.com

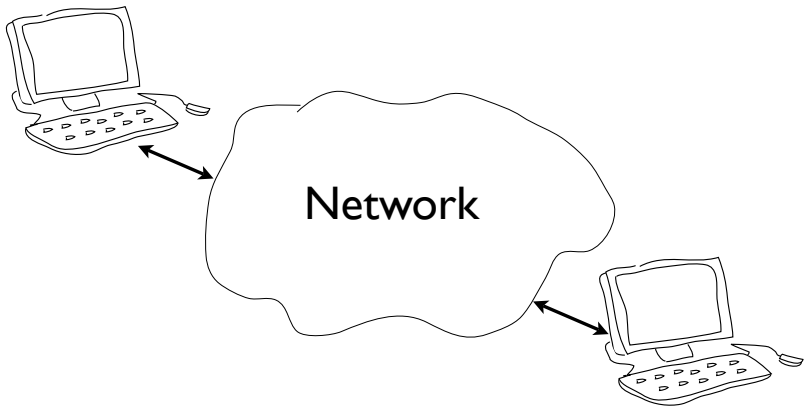
Edition: Thu Jun 17 19:49:58 2010

Section 1

Network Fundamentals

The Problem

- Communication between computers



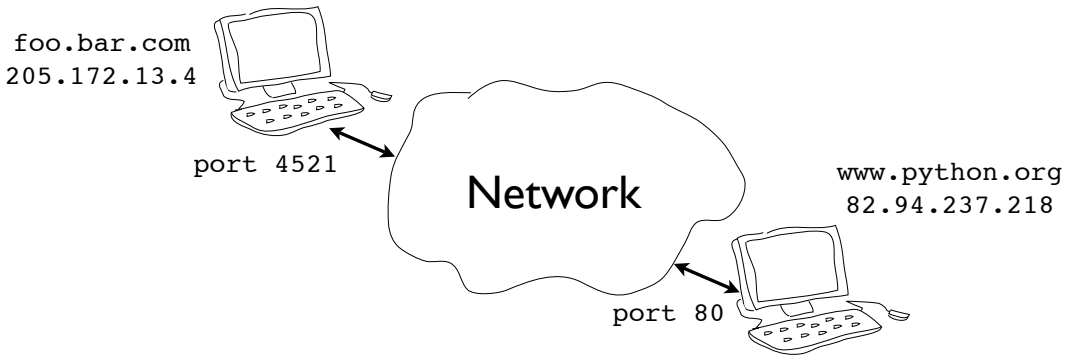
- It's just sending/receiving bits

Two Main Issues

- Addressing
  - Specifying a remote computer and service
- Data transport
  - Moving bits back and forth

Network Addressing

- Machines have a hostname and IP address
- Programs/services have port numbers



Standard Ports

- Ports for common services are preassigned

```
21          FTP
22          SSH
23          Telnet
25          SMTP (Mail)
80          HTTP (Web)
110         POP3 (Mail)
119         NNTP (News)
443         HTTPS (web)
```

- Other port numbers may just be randomly assigned to programs by the operating system

Using netstat

- Use 'netstat' to view active network connections

```
shell % netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
tcp        0      0 *:imaps                 *:*
tcp        0      0 *:pop3s                 *:*
tcp        0      0 localhost:mysql         *:*
tcp        0      0 *:pop3                  *:*
tcp        0      0 *:imap2                 *:*
tcp        0      0 *:8880                  *:*
tcp        0      0 *:www                   *:*
tcp        0      0 192.168.119.139:domain  *:*
tcp        0      0 localhost:domain        *:*
tcp        0      0 *:ssh                   *:*
...
```

- Note: Must execute from the command shell on both Unix and Windows

Connections

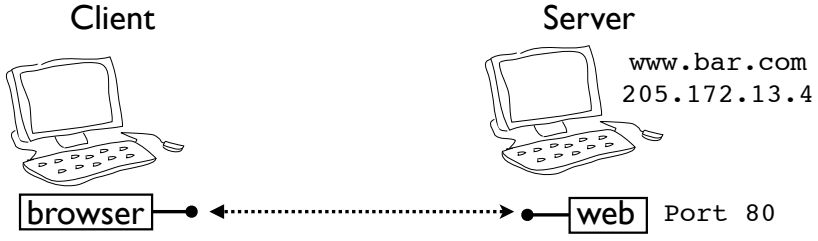
- Each endpoint of a network connection is always represented by a host and port #
- In Python you write it out as a tuple (host,port)

```
("www.python.org",80)
("205.172.13.4",443)
```

- In almost all of the network programs you'll write, you use this convention to specify a network address

Client/Server Concept

- Each endpoint is a running program
- Servers wait for incoming connections and provide a service (e.g., web, mail, etc.)
- Clients make connections to servers



Request/Response Cycle

- Most network programs use a request/ response model based on messages
- Client sends a request message (e.g., HTTP)

```
GET /index.html HTTP/1.0
```

- Server sends back a response message

```
HTTP/1.0 200 OK
Content-type: text/html
Content-length: 48823

<HTML>
...
```

- The exact format depends on the application

Using Telnet

- As a debugging aid, telnet can be used to directly communicate with many services

```
telnet hostname portnum
```

- Example:

type this and press return a few times →

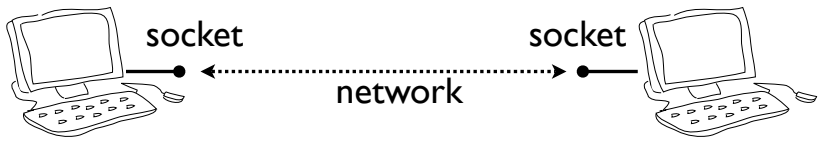
```
shell % telnet www.python.org 80
Trying 82.94.237.218...
Connected to www.python.org.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 31 Mar 2008 13:34:03 GMT
Server: Apache/2.2.3 (Debian) DAV/2 SVN/1.4.2
mod_ssl/2.2.3 OpenSSL/0.9.8c
...
```

Data Transport

- There are two basic types of communication
- Streams (TCP): Computers establish a connection with each other and read/write data in a continuous stream of bytes---like a file. This is the most common.
- Datagrams (UDP): Computers send discrete packets (or messages) to each other. Each packet contains a collection of bytes, but each packet is separate and self-contained.

Sockets

- Programming abstraction for network code
- Socket: A communication endpoint



- Supported by socket library module
- Allows connections to be made and data to be transmitted in either direction

Socket Basics

- To create a socket

```
import socket
s = socket.socket(addr_family, type)
```

- Address families

```
socket.AF_INET      Internet protocol (IPv4)
socket.AF_INET6     Internet protocol (IPv6)
```

- Socket types

```
socket.SOCK_STREAM  Connection based stream (TCP)
socket.SOCK_DGRAM   Datagrams (UDP)
```

- Example:

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
```


Socket Types

- Almost all code will use one of following

```
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s = socket(AF_INET, SOCK_DGRAM)
```

- Most common case: TCP connection

```
s = socket(AF_INET, SOCK_STREAM)
```

Using a Socket

- Creating a socket is only the first step

```
s = socket(AF_INET, SOCK_STREAM)
```

- Further use depends on application
- Server
  - Listen for incoming connections
- Client
  - Make an outgoing connection

TCP Client

- How to make an outgoing connection

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.connect(("www.python.org",80))        # Connect
s.send("GET /index.html HTTP/1.0\n\n")  # Send request
data = s.recv(10000)                    # Get response
s.close()
```

- s.connect(addr) makes a connection

```
s.connect(("www.python.org",80))
```

- Once connected, use send(),recv() to transmit and receive data
- close() shuts down the connection

TCP Server

- A simple server

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

- Send a message back to a client

```
% telnet localhost 9000
Connected to localhost.
Escape character is '^]'.
Hello 127.0.0.1
Connection closed by foreign host.
%
```

Server message

TCP Server

- Address binding

```
from socket import *
s = socket(AF_INET,SOCK_STR
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

binds the socket to a specific address

- Addressing

binds to localhost

```
s.bind(("",9000))
s.bind(("localhost",9000))
s.bind(("192.168.2.1",9000))
s.bind(("104.21.4.2",9000))
```

If system has multiple IP addresses, can bind to a specific address

TCP Server

- Start listening for connections

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

Tells operating system to start listening for connections on the socket

- s.listen(*backlog*)
- backlog is # of pending connections to allow
- Note: not related to max number of clients

TCP Server

- Accepting a new connection

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

Accept a new client connection

- s.accept() blocks until connection received
- Server sleeps if nothing is happening

TCP Server

- Client socket and address

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

Accept returns a pair (client_socket,addr)

`<socket._socketobject object at 0x3be30>`

This is a new socket that's used for data

`("104.23.11.4",27743)`

This is the network/port address of the client that connected

TCP Server

- Sending data

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

Send data to client

Note: Use the client socket for transmitting data. The server socket is only used for accepting new connections.

TCP Server

- Closing the connection

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

Close client connection

- Note: Server can keep client connection alive as long as it wants
- Can repeatedly receive/send data

TCP Server

- Waiting for the next connection

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",9000))
s.listen(5)
while True:
    c,a = s.accept()
    print "Received connection from", a
    c.send("Hello %s\n" % a[0])
    c.close()
```

Wait for next connection

- Original server socket is reused to listen for more connections
- Server runs forever in a loop like this

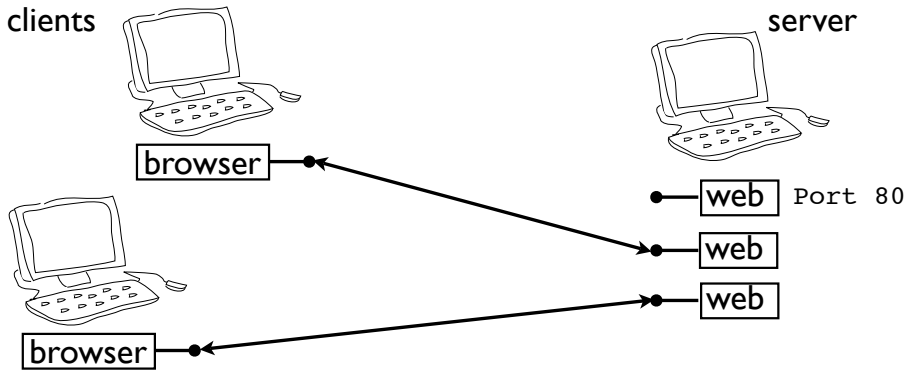
Raw Sockets

- If you have root/admin access, can gain direct access to raw network packets
- Depends on the system
- Example: Linux packet sniffing

```
s = socket(AF_PACKET, SOCK_DGRAM)
s.bind(("eth0",0x0800))   # Sniff IP packets

while True:
    msg,addr = s.recvfrom(4096)  # get a packet
    ...
```

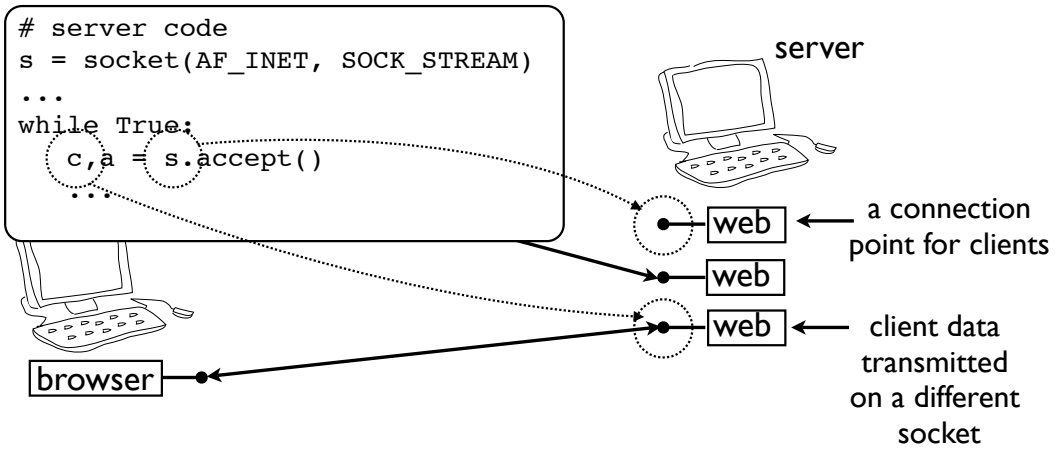

Sockets and Concurrency

- Servers usually handle multiple clients



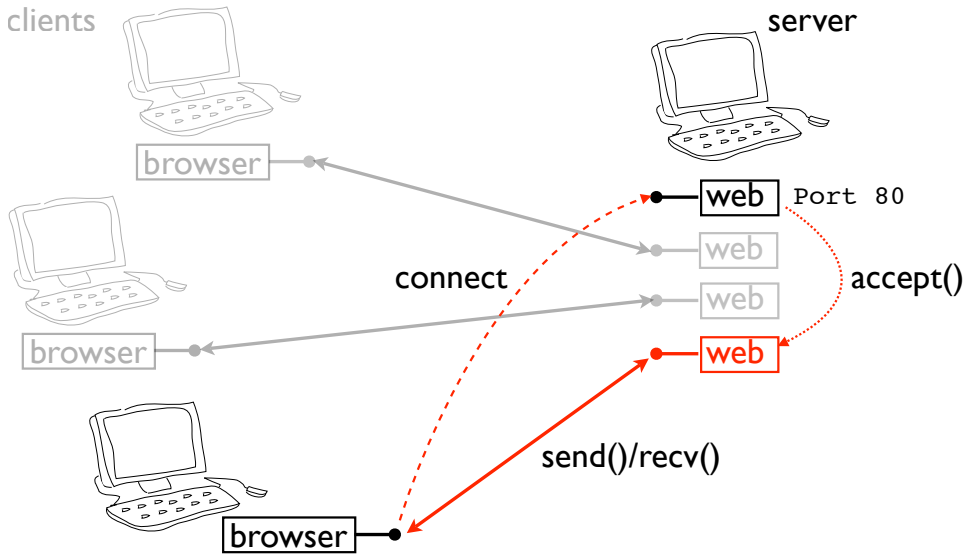
Sockets and Concurrency

- Each client gets its own socket on server



Sockets and Concurrency

- New connections make a new socket

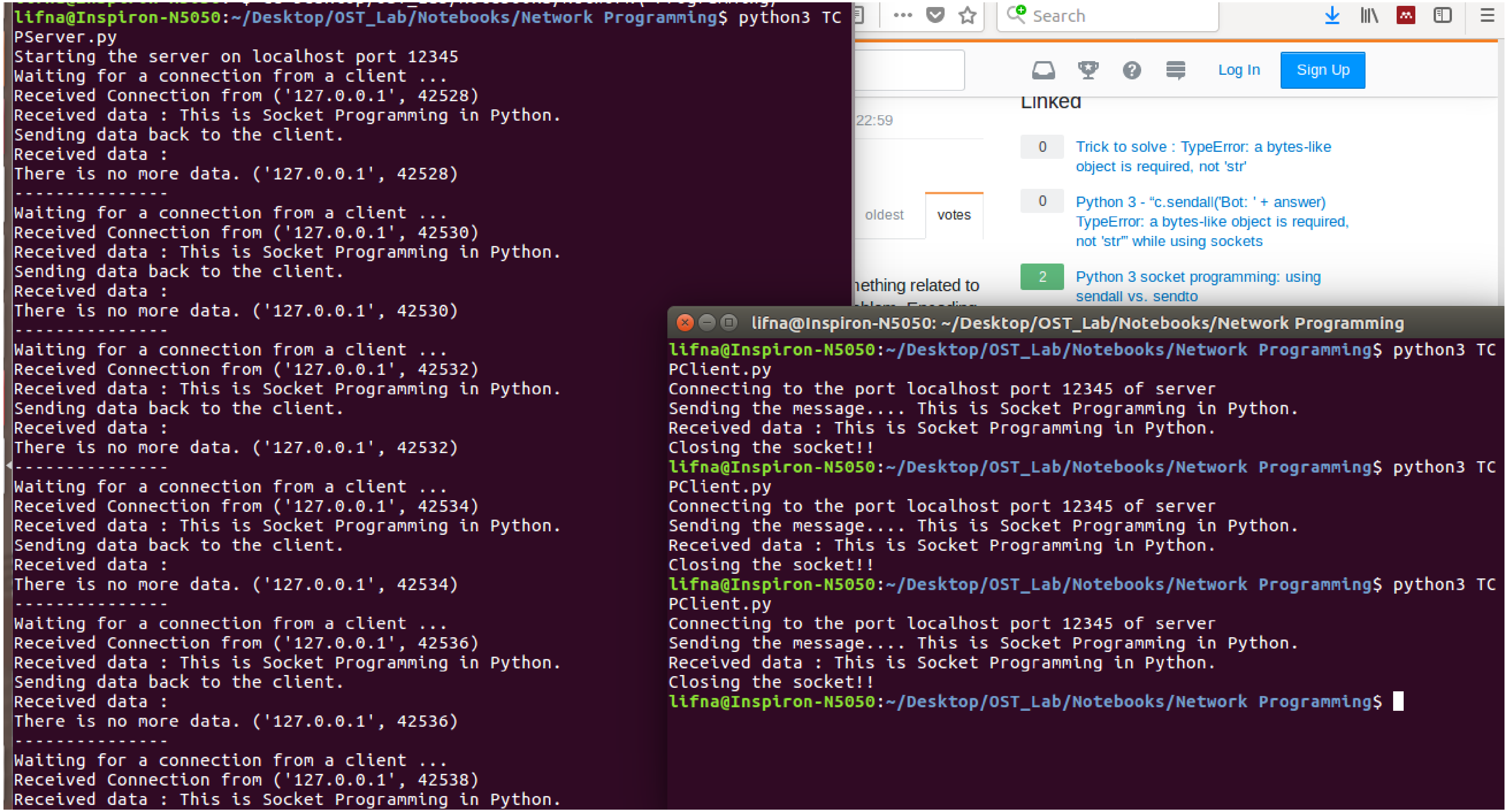


TCPServer.py

```
# TCP Server
# ==========
import socket

# Creatin a TCP/IP socket
serversock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# Binding the socket to the port
serveraddress = ('localhost',12345)

print("Starting the server on %s port %s" % serveraddress)
serversock.bind(serveraddress)

# Listening to the incoming connections
serversock.listen(1)

while True:
    # Waiting for a new connection
    print("Waiting for a connection from a client ... ")
    conn,clientaddress = serversock.accept()

            try:
                # Receiving the data fom the client and sending it again
                print('Received Connection from',clientaddress)

                while True:
                    data = conn.recv(1024)
                    print("Received data :", data.decode())
                    if data:
                        print("Sending data back to the client.")
                        conn.sendall(data)
                    else:
                        print("There is no more data.", clientaddress)
                        print("----------------")
                        break

            finally:
                # Closing the connection
                conn.close()
```

TCPClient.py

```
## TCP Client
## ==========
import socket

# Creating a TCP/ IP socket
clientsock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# Connecting the socket to the server's port
serveraddress = ("localhost",12345)

print("Connecting to the port %s port %s of server" % serveraddress)
clientsock.connect(serveraddress)

try:
        # Sending data
        senddata = ("This is Socket Programming in Python.")
        print("Sending the message.... %s" % senddata)
        clientsock.sendall(senddata.encode('utf-8'))

        # Receiving data
        recvdata = clientsock.recv(1024)
        print("Received data : %s" % recvdata.decode())

finally:
        # Closing the connection
        print("Closing the socket!!")
        clientsock.close()
```



lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 TCPServer.py
Starting the server on localhost port 12345
Waiting for a connection from a client ...
Received Connection from ('127.0.0.1', 42528)
Received data : This is Socket Programming in Python.
Sending data back to the client.
Received data :
There is no more data. ('127.0.0.1', 42528)

Waiting for a connection from a client ...
Received Connection from ('127.0.0.1', 42530)
Received data : This is Socket Programming in Python.
Sending data back to the client.
Received data :
There is no more data. ('127.0.0.1', 42530)

Waiting for a connection from a client ...
Received Connection from ('127.0.0.1', 42532)
Received data : This is Socket Programming in Python.
Sending data back to the client.
Received data :
There is no more data. ('127.0.0.1', 42532)

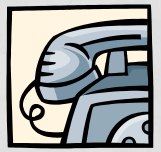
Waiting for a connection from a client ...
Received Connection from ('127.0.0.1', 42534)
Received data : This is Socket Programming in Python.
Sending data back to the client.
Received data :
There is no more data. ('127.0.0.1', 42534)

Waiting for a connection from a client ...
Received Connection from ('127.0.0.1', 42536)
Received data : This is Socket Programming in Python.
Sending data back to the client.
Received data :
There is no more data. ('127.0.0.1', 42536)

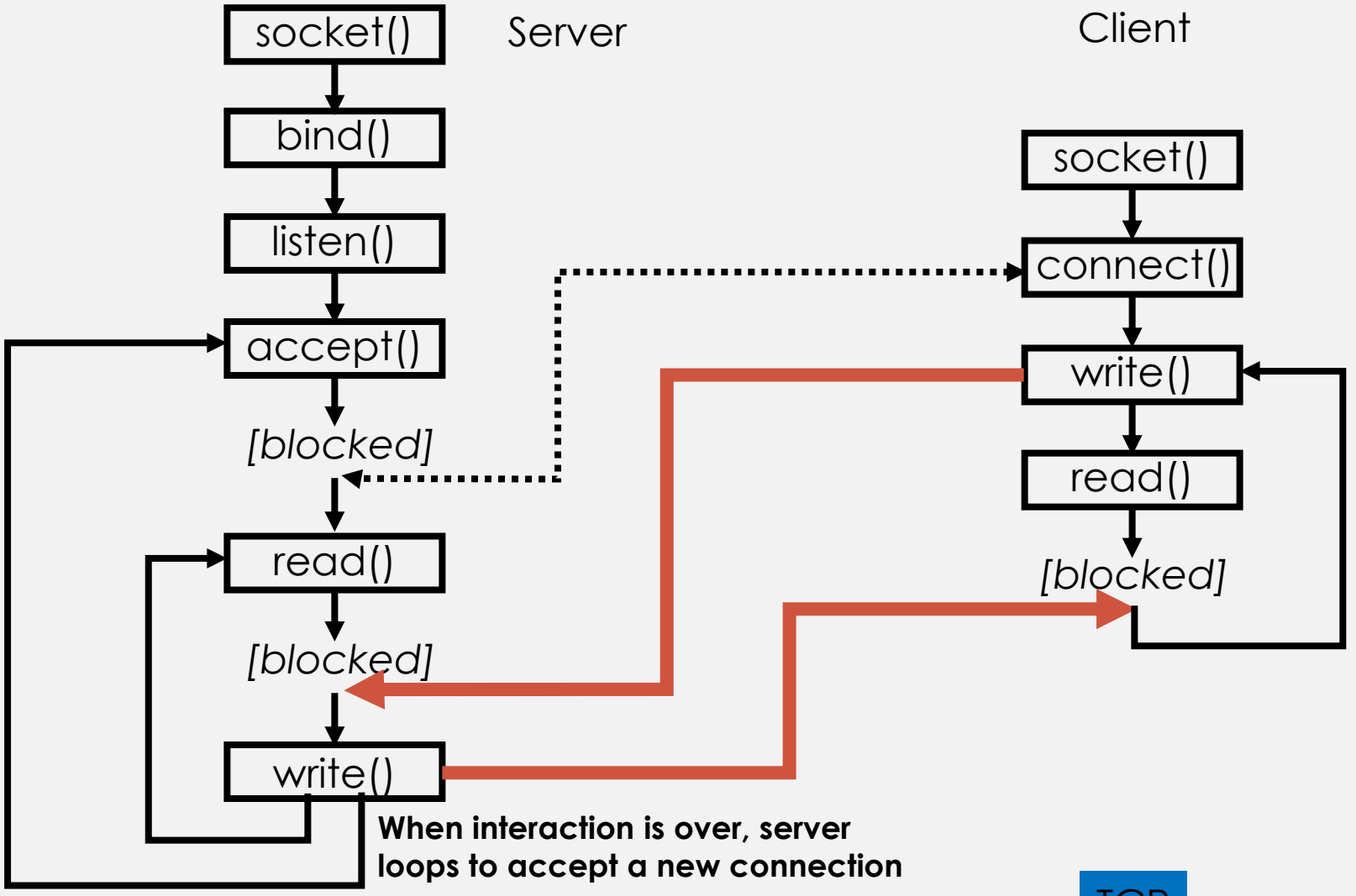
Waiting for a connection from a client ...
Received Connection from ('127.0.0.1', 42538)
Received data : This is Socket Programming in Python.
Search
Log In
Sign Up
Linked
22:59
0
Trick to solve : TypeError: a bytes-like object is required, not 'str'
oldest
votes
0
Python 3 - "c.sendall('Bot: ' + answer) TypeError: a bytes-like object is required, not 'str'" while using sockets
2
Python 3 socket programming: using sendall vs. sendto
lifna@Inspiron-N5050: ~/Desktop/OST_Lab/Notebooks/Network Programming
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 TCPClient.py
Connecting to the port localhost port 12345 of server
Sending the message.... This is Socket Programming in Python.
Received data : This is Socket Programming in Python.
Closing the socket!!
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 TCPClient.py
Connecting to the port localhost port 12345 of server
Sending the message.... This is Socket Programming in Python.
Received data : This is Socket Programming in Python.
Closing the socket!!
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 TCPClient.py
Connecting to the port localhost port 12345 of server
Sending the message.... This is Socket Programming in Python.
Received data : This is Socket Programming in Python.
Closing the socket!!
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$

TCP CHARACTERISTICS

- Connection-oriented
  - Two endpoints of a virtual circuit
- Reliable
  - Application needs no error checking
- Stream-based
  - No predefined blocksize
- Processes identified by port numbers
- Services live at specific ports



CONNECTION-ORIENTED SERVICES



CONNECTION-ORIENTED SERVER

- The socket module
  - Provides access to low-level network programming functions.
  - Example: A server that returns the current time

```
# Time server program
from socket import *
import time
s = socket(AF_INET, SOCK_STREAM)          # Create TCP socket
s.bind(("",8888))                         # Bind to port 8888
s.listen(5)                               # Start listening
while 1:
  client,addr = s.accept()                # Wait for a connection
  print "Got a connection from ", addr
  client.send(time.ctime(time.time()))    # Send time back
  client.close()
```

- Notes:
  - Socket first opened by server is not the same one used to exchange data.
  - Instead, the accept() function returns a new socket for this ('client' above).
  - listen() specifies max number of pending connections.

ex1.py

CONNECTION-ORIENTED CLIENT

- Client Program
  - Connect to time server and get current time

```
# Time client program
from socket import *
s = socket(AF_INET,SOCK_STREAM)      # Create TCP socket
s.connect(("127.0.0.1",8888))        # Connect to server
tm = s.recv(1024)                    # Receive up to 1024 bytes
s.close()                            # Close connection
print "The time is", tm
```

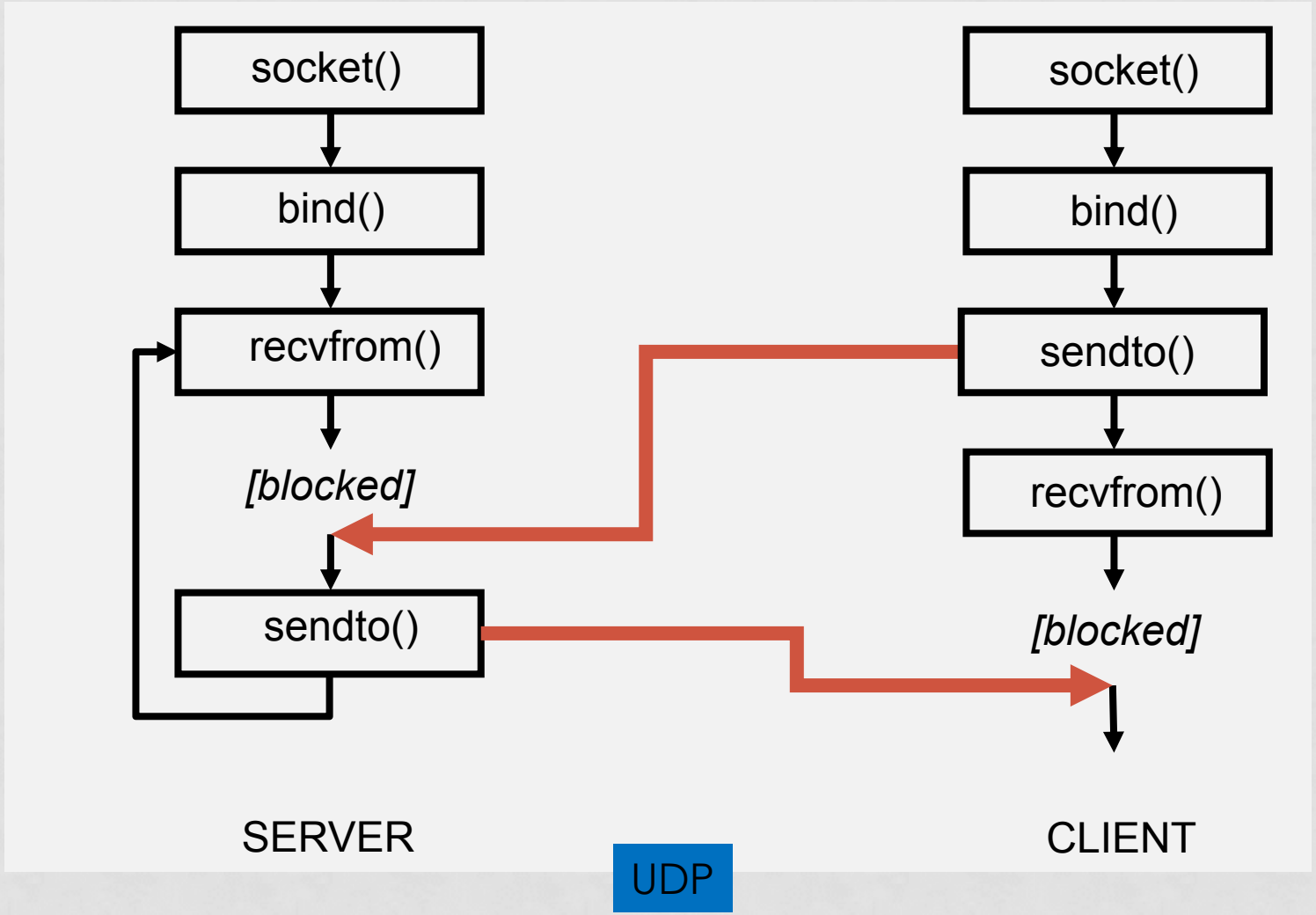
- Key Points
  - Once connection is established, server/client communicate using send() and recv().
  - Aside from connection process, it's relatively straightforward.
  - Of course, the devil is in the details.
  - And are there ever a LOT of details.

ex2.py

UDP CHARACTERISTICS

- Also datagram-based
  - Connectionless, unreliable, can broadcast
- Applications usually message-based
  - No transport-layer retries
  - Applications handle (or ignore) errors
- Processes identified by port number
- Services live at specific ports
  - Usually below 1024, requiring privilege

CONNECTIONLESS SERVICES



UDP VERSUS TCP

UDP v/s TCP		
Characteristics/ Description	**UDP**	**TCP**
General Description	Simple High speed low functionality "wrapper" that interface applications to the network layer and does little else	Full-featured protocol that allows applications to send data reliably without worrying about network layer issues.
Protocol connection Setup	Connection less data is sent without setup	Connection-oriented; Connection must be Established prior to transmission.
Data interface to application	Message base-based is sent in discrete packages by the application.	Stream-based; data is sent by the application with no particular structure
Reliability and Acknowledgements	Unreliable best-effort delivery without acknowledgements	Reliable delivery of message all data is acknowledged.
Retransmissions	Not performed. Application must detect lost data and retransmit if needed.	Delivery of all data is managed, and lost data is retransmitted automatically.
Features Provided to Manage flow of Data	None	Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms
Overhead	Very Low	Low, but higher than UDP
Transmission speed	Very High	High but not as high as UDP
Data Quantity Suitability	Small to moderate amounts of data.	Small to very large amounts of data.

SIMPLE CONNECTIONLESS SERVER

```
from socket import socket, AF_INET, SOCK_DGRAM
s = socket(AF_INET, SOCK_DGRAM)
s.bind(('127.0.0.1', 11111))
while 1:
    data, addr = s.recvfrom(1024)
    print "Connection from", addr
    s.sendto(data.upper(), addr)
```

- How much easier does it need to be?

Note that the *bind()* argument is a two-element tuple of address and port number

ex3.py

SIMPLE CONNECTIONLESS CLIENT

```
from socket import socket, AF_INET, SOCK_DGRAM
s = socket(AF_INET, SOCK_DGRAM)
s.bind(('127.0.0.1', 0))      # OS chooses port
print "using", s.getsockname()
server = ('127.0.0.1', 11111)
s.sendto("MixedCaseString", server)
data, addr = s.recvfrom(1024)
print "received", data, "from", addr
s.close()
```

- Relatively easy to understand?

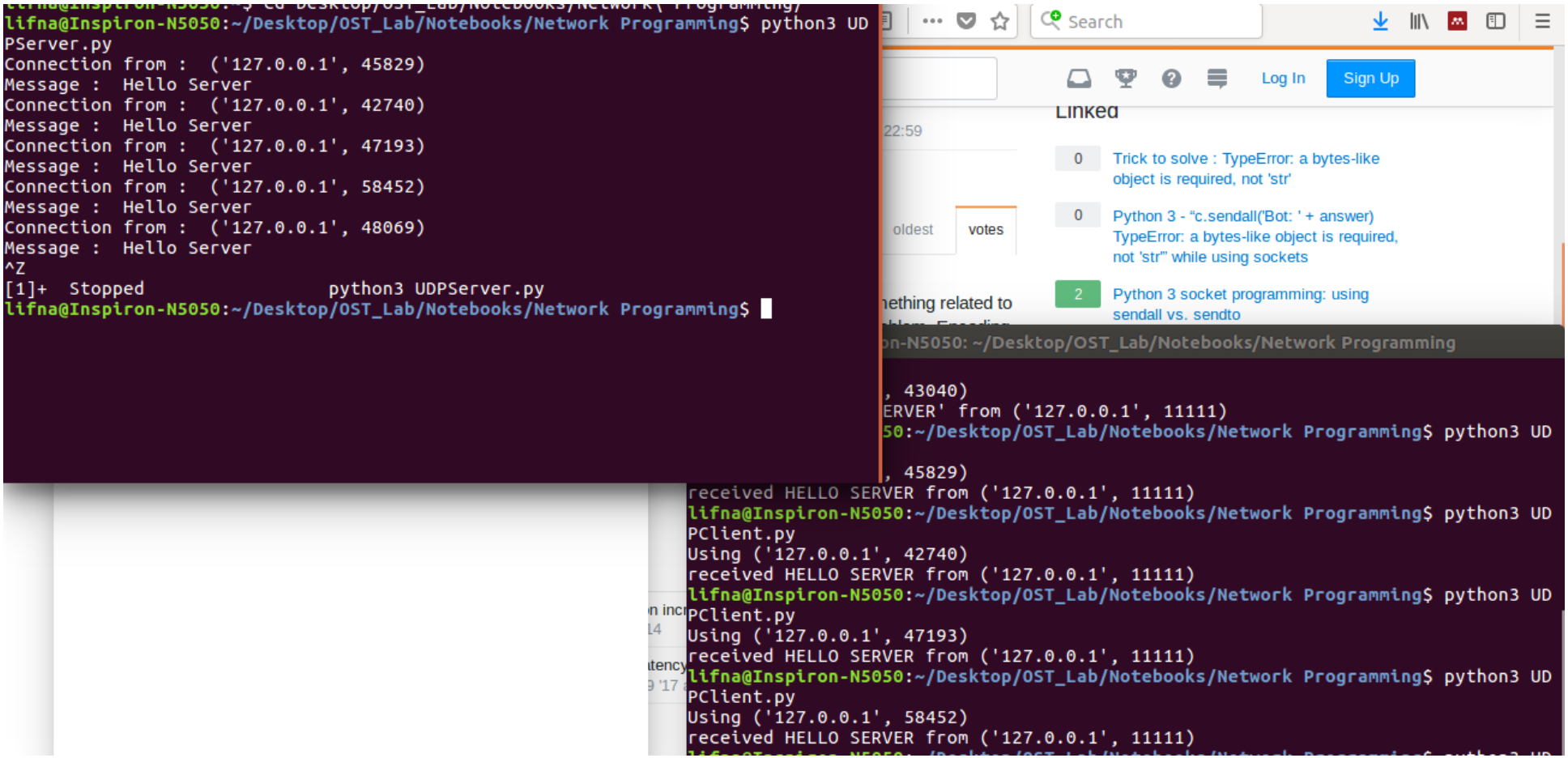
ex4.py

SOCKET METHODS

- socket methods

```
s.accept()                  # Accept a new connection
s.bind(address)             # Bind to an address and port
s.close()                   # Close the socket
s.connect(address)          # Connect to remote socket
s.fileno()                  # Return integer file descriptor
s.getpeername()             # Get name of remote machine
s.getsockname()             # Get socket address as (ipaddr,port)
s.getsockopt(...)           # Get socket options
s.listen(backlog)           # Start listening for connections
s.makefile(mode)            # Turn socket into a file object
s.recv(bufsize)             # Receive data
s.recvfrom(bufsize)         # Receive data (UDP)
s.send(string)              # Send data
s.sendto(string, address)   # Send packet (UDP)
s.setblocking(flag)         # Set blocking or nonblocking mode
s.setsockopt(...)           # Set socket options
s.shutdown(how)             # Shutdown one or both halves of connection
```

- Comments
  - There are a huge variety of configuration/connection options.
  - You'll definitely want a good reference at your side.


lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 UD
PServer.py
Connection from : ('127.0.0.1', 45829)
Message : Hello Server
Connection from : ('127.0.0.1', 42740)
Message : Hello Server
Connection from : ('127.0.0.1', 47193)
Message : Hello Server
Connection from : ('127.0.0.1', 58452)
Message : Hello Server
Connection from : ('127.0.0.1', 48069)
Message : Hello Server
^Z
[1]+ Stopped   python3 UDPServer.py
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$
Search
Log In
Sign Up
Linked
22:59
0
Trick to solve : TypeError: a bytes-like object is required, not 'str'
0
Python 3 - "c.sendall('Bot: ' + answer) TypeError: a bytes-like object is required, not 'str'" while using sockets
2
Python 3 socket programming: using sendall vs. sendto
oldest
votes
nething related to
on-N5050: ~/Desktop/OST_Lab/Notebooks/Network Programming
, 43040)
ERVER' from ('127.0.0.1', 11111)
50:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 UD
, 45829)
received HELLO SERVER from ('127.0.0.1', 11111)
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 UD
PClient.py
Using ('127.0.0.1', 42740)
received HELLO SERVER from ('127.0.0.1', 11111)
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 UD
PClient.py
Using ('127.0.0.1', 47193)
received HELLO SERVER from ('127.0.0.1', 11111)
lifna@Inspiron-N5050:~/Desktop/OST_Lab/Notebooks/Network Programming$ python3 UD
PClient.py
Using ('127.0.0.1', 58452)
received HELLO SERVER from ('127.0.0.1', 11111)