

COLLEGE OF ENGINEERING, TRIVANDRUM

EXPERIMENT 9

---

# Network Programming Lab

---

*Author:*  
Alan Anto

*Registration Number :*  
TVE16CS09

March 27, 2019

Contents

<b>1</b>	<b>Client-Server Communication using UDP</b>	<b>2</b>
1.1	AIM . . . . .	2
1.2	Theory . . . . .	2
1.2.1	UDP Protocol . . . . .	2
1.2.2	Socket Programming . . . . .	4
1.3	Program . . . . .	5
1.3.1	Server Side . . . . .	5
1.3.2	Client Side . . . . .	6
1.4	Output . . . . .	7
1.4.1	Server . . . . .	7
1.4.2	Client . . . . .	7
1.5	Result . . . . .	7

# 1 Client-Server Communication using UDP

## 1.1 AIM

To implement a Client-Server communication using Socket Programming and UDP as trans- port layer protocol

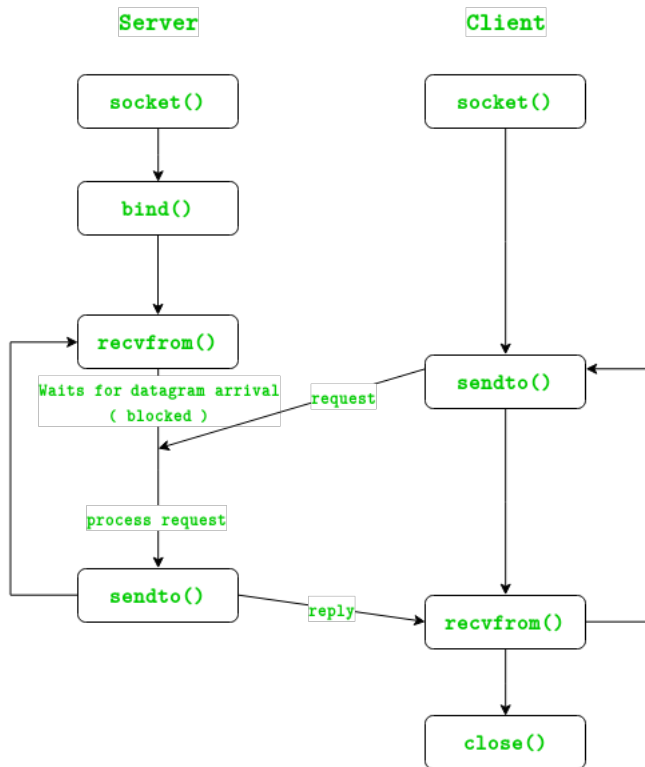
## 1.2 Theory

### 1.2.1 UDP Protocol

UDP stands for User Datagram Protocol. The UDP protocol works similarly to TCP, but it throws all the error-checking stuff out.

When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet, it will just continue sending the next packets. There is no guarantee that all the packets are delivered and there is no way to ask for a packet again if it misses out, but losing all this overhead means the computers can communicate more quickly.

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.



UDP Server:

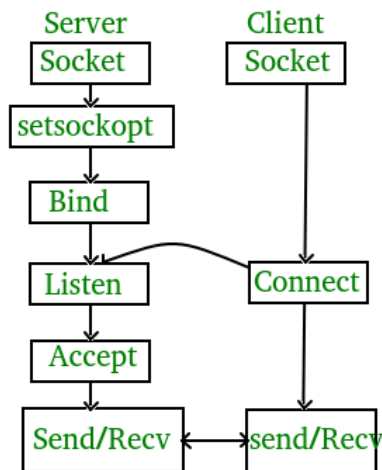
- Create UDP socket.
- Bind the socket to server address.
- Wait until datagram packet arrives from client.
- Process the datagram packet and send a reply to client.
- Go back to Step 3.

UDP Client:

- Create UDP socket.
- Send message to server.
- Wait until response from server is received.
- Process reply and go back to step 2, if necessary.
- Close socket descriptor and exit.

### 1.2.2 Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



- Socket creation:

```
int sockfd = socket(domain, type, protocol)
```

– sockfd: socket descriptor, an integer (like a file-handle)

– domain: integer, communication domain e.g., AF\_INET (IPv4 protocol) , AF\_INET6 (IPv6 protocol)

– type: communication type

SOCK\_STREAM: TCP(reliable, connection oriented) SOCK\_DGRAM: UDP(unreliable, connectionless)

– protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

- Bind:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure).

- sendto

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen)
```

- sockfd – File descriptor of socket
- buf – Application buffer containing the data to be sent
- len – Size of buf application buffer
- flags – Bitwise OR of flags to modify socket behaviour
- dest\_addr – Structure containing address of destination
- addrlen – Size of dest\_addr structure

- recvfrom

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen)
```

- sockfd – File descriptor of socket
- buf – Application buffer in which to receive data
- len – Size of buf application buffer
- flags – Bitwise OR of flags to modify socket behaviour
- src\_addr – Structure containing source address is returned
- addrlen – Variable in which size of src\_addr structure is returned

## 1.3 Program

### 1.3.1 Server Side

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

port=8080

s.bind((' ', port))
```

```
msgFromServer      = "Hello UDP Client "

bytesToSend        = str.encode(msgFromServer)

while True:
    msg, addr=s.recvfrom(1024)
    print('Got Connection from', addr)
    print('Message from Client: ', msg)
    s.sendto(bytesToSend, addr)
    print("Thanks for connecting")
    exit(0)
```

### 1.3.2 Client Side

```
// Client side implementation of UDP client-server model

import socket
import sys

msgFromClient      = "Hello UDP Server\n"

bytesToSend        = str.encode(msgFromClient)

s= socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

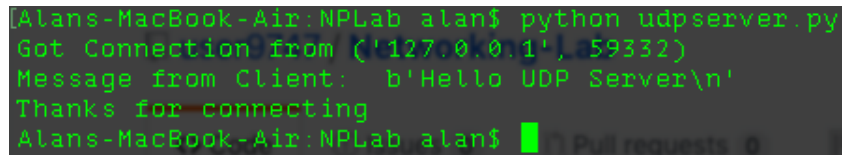
port = 8080

s.sendto(bytesToSend, ('127.0.0.1', port))
print("Sending message to ", port)
print("\n")
msgFromServer=s.recvfrom(1024)
# msg = "Message from Server:  {}".format(msgFromServer[0])

print(" Message from Server: ", msgFromServer[0])
print("\n ")
s.close()
```

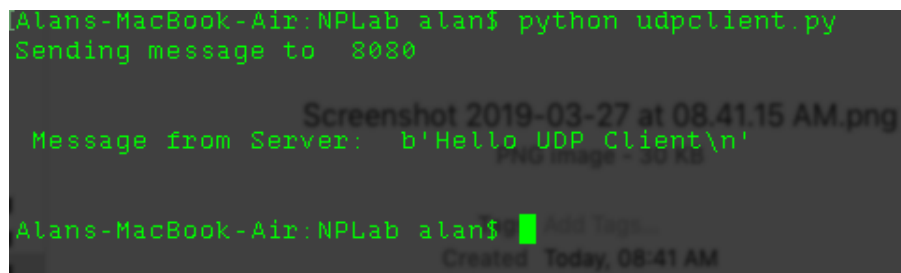
## 1.4 Output

### 1.4.1 Server

A terminal window on a MacBook Air showing the output of a Python UDP server. The prompt is 'Alans-MacBook-Air:NPLab alan\$'. The output is: 'python udpserver.py', 'Got Connection from ('127.0.0.1', 59332)', 'Message from Client: b\'Hello UDP Server\\n\'', 'Thanks for connecting', and 'Alans-MacBook-Air:NPLab alan\$'.

```
Alans-MacBook-Air:NPLab alan$ python udpserver.py
Got Connection from ('127.0.0.1', 59332)
Message from Client: b'Hello UDP Server\n'
Thanks for connecting
Alans-MacBook-Air:NPLab alan$
```

### 1.4.2 Client

A terminal window on a MacBook Air showing the output of a Python UDP client. The prompt is 'Alans-MacBook-Air:NPLab alan\$'. The output is: 'python udpclient.py', 'Sending message to 8080', 'Message from Server: b\'Hello UDP Client\\n\'', and 'Alans-MacBook-Air:NPLab alan\$'.

```
Alans-MacBook-Air:NPLab alan$ python udpclient.py
Sending message to 8080
Message from Server: b'Hello UDP Client\n'
Alans-MacBook-Air:NPLab alan$
```

## 1.5 Result

A Client-Server communication program has been implemented in python language using libraries for Socket Programming and UDP as transport layer Protocol. The program was compiled and excuted using python compiler for python 3.7.0 in MacOS Mojave.

The server code creates a UDP server and it will be running inside an infinite while loop. The server will be receiving message send from the UDP client. The socket is bind to port 8080. Similarly the client also sends message to the server and it is printed in the server. Then it prints the message that is sent from the server and the socket is closed.