Programming Assignment 1 for CSE232

# UDP Pinger

Palak Bhardwaj (2022344)
Yashovardhan Singhal (2022591)

16 September 2024

---

## Abstract :

This report provides an overview of **UDP** pinger server and client done as a part of the CN programming assignment 1.

**In part (a) -**

— The client sends 10 pings to the server. If the client doesn't receive a response in under 1 second, it is assumed that the packet is lost during transmission.

— The packets are sent to the local host, both the client and the server are hosted on a single device.

— The client calculates the round-trip time for each packet.

— The Minimum, Maximum, and Average RTTs are calculated at the end of all pings from the client.

— The packet loss rate is observed (as a percentage).

**In part (b) -**

— A UDP heartbeat application is made.

— IT is observed as to how many times the UDP Heartbeat packets are sent before you see 3 consecutive responses missing were found to be missing.

.

The initial server code was given in the program. A server socket is created by the server, which represents an end of a communication link ( UDP in this case) -

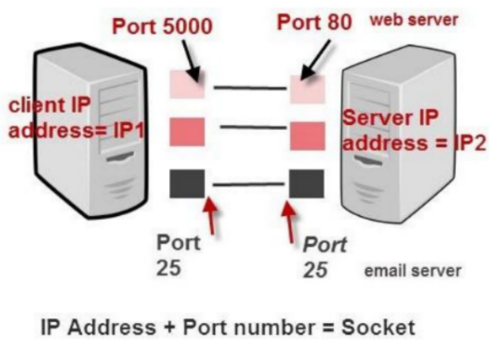


FIGURE 1 – A Socket specifies a server address and the port on which the server is listening ( From Socket Programming Tut1 )

Some of the Standard Socket types

```
/*Standard socket types */
#define  SOCK_STREAM             1 /*virtual circuit*/
#define  SOCK_DGRAM              2 /*datagram*/
#define  SOCK_RAW                3 /*raw socket*/
#define  SOCK_RDM                4 /*reliably-delivered message*/
#define  SOCK_CONN_DGRAM         5 /*connection datagram*/
```

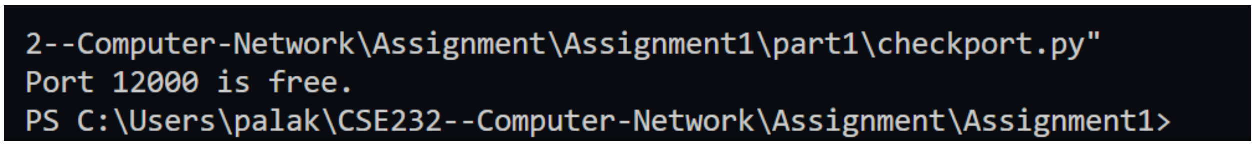
In the assignment application **SOCKDGRAM** Socket type is used for a UDP connection, it supports the bidirectional flow of data, which is not sequenced, reliable, or unduplicated. Each datagram is treated as independent, unlike in a **SOCKSTREAM** Socket type where the socket sets up a stream of data.

Once the Server socket is created, it binds to a specific port number , given 12000 in our code.

Below is a Python script to check if the port number 12000 is free or already in use.

```
from socket import *
    # Confirming that port 12000 is a free port for my device
def check_port(port):
    with socket(AF_INET, SOCK_STREAM) as serversocket:
        result = serversocket.connect_ex(('localhost', port))
        if result == 0:
            print(f"Port {port} is already in use.")
        else:
            print(f"Port {port} is free.")

# Example usage
check_port(12000)  # Replace 12000 with the port you want to check
```



```
2--Computer-Network\Assignment\Assignment1\part1\checkport.py"
Port 12000 is free.
PS C:\Users\palak\CSE232--Computer-Network\Assignment\Assignment1>
```

FIGURE 2 – Availability of port 12000

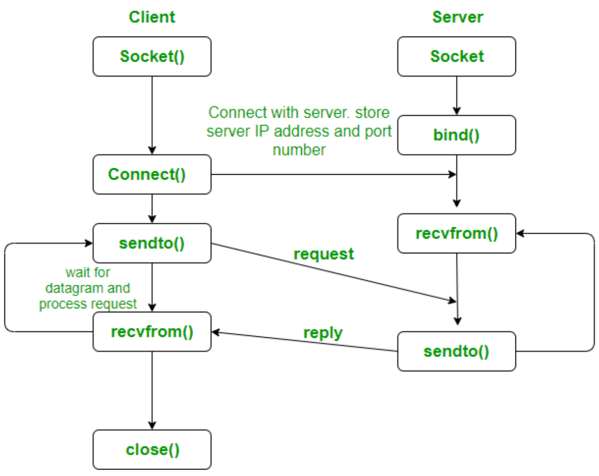

Client
Server
Socket()
Socket
bind()
Connect with server. store server IP address and port number
Connect()
sendto()
recvfrom()
request
wait for datagram and process request
recvfrom()
reply
sendto()
close()

FIGURE 3 – UDP connection Overview

This is a overview of how our code send packets using UDP.
**Client Side**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**Socket() :** The client creates a socket using the socket() system call. This socket is used for communication and for connection with the server.
**Connect() :** The client initiates a connection to the server. It stores the server's IP address and port number, which is required to communicate with the server.
**Sendto() :** The client sends a request (datagram) to the server using the sendto() function. This datagram is sent to the IP address and port of the server. The request be a query, command, or data the client wants the server to process. In our case, the message is sent.
**Recvfrom() :** After sending the request, the client waits for a reply from the server using recvfrom(). This function listens for incoming data from the server. When a reply is received, the client processes it.
**Close() :** Once the communication is complete, the client closes the connection using the close() function, freeing up resources.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**Server Side**

**Socket() :** The server creates its own socket using socket(). This listens for incoming communication from clients.
**Bind() :** The server binds its socket to a specific IP address and port number using the bind() function. This allows the server to listen for requests on a known address and port.
**Recvfrom() :** The server waits for requests from clients using recvfrom(). When a request is received, it processes the client's message.
**Sendto()** : After processing the client's request, the server sends a reply using sendto(). The reply is sent to the client's IP address and port from which the request originated.

**Recvfrom()** (Loop) : The server typically runs in a loop, continually waiting for requests from clients using recvfrom(). It processes each request and sends back a response.

## 1 Server - Part(a)

Since the port 12000, is free, it can be used by the client to send request.

```
serverSocket.bind(('', 12000))
# This commands binds the server socket to the port 12000
# The '' indicates that the socket listens to all the network interfaces on that port
```

The Server Socket binds to the port 12000, so that the socket can listen to UDP packets on that port. The server receives the message along with the address from where the message comes.

### 1.1 Functioning

— The server responds to the message if the random number generated by the server is less than 4.
— If the random number generated is greater than or equal to 4, the packet is not dropped.
— The server responds and sends the message back on the localhost.

## 2 Client - Part(a)

### 2.1 Inital Set up

```
client_socket = socket(AF_INET,SOCK_DGRAM)
#The client socket is created on the client side
```

AFINET means the IPv4 Address Family is used SOCKDGRAM means a UDP connection

```
server_addr = ("localhost",12000)
#Server address is specified as localhost, and the same port number 12000 is used
```

### 2.2 Settimeout()

The time after which the packet is assumed to be dropped is 1 second, hence, setimeout() is used

```
socket.settimeout(value)
#Sets a timeout on blocking socket operations.
```

If a non-zero value is given, subsequent socket operations raise a timeout exception if the timeout period value has elapsed before the operation has completed. This exception is caught in try-except clause to check when the packet is dropped.

The message is sent to the server via the client socket and the message is received via the same client socket.

### 2.3 Calculations

```
client_socket.sendto(message_sent.encode(), server_addr)
message_received, address = client_socket.recvfrom(4096)
# 4096 is the buffer size of the message
# Message is encoded from string to byte while sending
```

This helps in calculation of the **Round Trip Time** as a start counter is started before sending the message and the counter is stopped after receiving the message.

```
start_time = perf_counter()
current_time = ctime()
message_sent = "Ping_Number: " + ping_number + " Time: " + str(current_time)
client_socket.sendto(message_sent.encode(), server_addr)
message_received, address = client_socket.recvfrom(4096)
end_time = perf_counter()
Round_Trip_Time = end_time - start_time
```

perfcounter() returns the value (in fractional seconds) of a performance counter, i.e. a clock with the highest available resolution to measure a short duration. The the difference between the results of two calls is valid and perfoms as a high resoultion timer.

All the packets being successfully sent are stored in a list. Hence the number of datagrams sent are represented by the length of this list. All the other calculations are also done based on this time calculation.

```
sum_RTT = sum(RTT_list)
avg_RTT = sum_RTT/len(RTT_list)
# len of RTT_list represents the total number of datagrams sent
max_RTT = max(RTT_list)
min_RTT = min(RTT_list)
perc_RTT_loss = (lost_package/10)*100
# lost package is the number of packets that went in timeout
```

### 2.4 Repetition time

```
sleep (repetition _ time)
// This assigns a constant value time between consecutive pings
repitition time = 2
```

The repetition time ensures the client waits for 2 second between sending pings.
Repitition time should be less than timeout time.

## 3 Server - Part(b)

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', 12000))
while True:
    rand = random.randint(0, 10)
    message, address = serverSocket.recvfrom(1024)
    time_received = message[- 26:].decode()
    date_format = '%Y-%m-%d %H:%M:%S.%f'
    time_enrypted = datetime.strptime(time_received,date_format)
    print("Time difference ", datetime.now()-time_enrypted)
    if rand < 4:
        continue
    serverSocket.sendto(message, address)
```

The UDP heartbeat server calculates the time taken by the server to respond to the client message. The time at which the message was sent is taken from the message itself as the format of the message is given as **Ping no Message time**.

The message at which the server responds to the message is calculated using datetime.now() and this is used to calculate the time difference.

The client sends the message in the format -

```
4  Time: 2024-09-16 23:52:24.144551
```

The server receives the message and makes use of the time in the message received to calculate time taken to respond to the message, the time duration is then sent as a message to the client. The server sends back the message as the calculated time difference.

## 4 Client - Part(b)

```
try :
    while(packet_misses<3):
        try:
            current_time = datetime.now()
            message_sent = str(ping_count) + " "  + " Time: " + str(current_time)
            client_socket.sendto(message_sent.encode(), server_addr)
            message_received, address = client_socket.recvfrom(4096)
            packet_misses=0
        except timeout:
                print(" Requested Time out for : ", ping_number)
                lost_package+=1
                packet_misses+=1
        ping_count+=1
 finally:
    print("Total packets sent before application stopped : ", ping_count)
    client_socket.close()
```

The UDP heartbeat client receives the message from the UDP server which has the time difference in it. The UDP client calculates if there are 3 consecutive message time out observed, in such a case, the server is assumed to be stop working. The package lost counter is set to 0 whenever a message is received successfully before timeout. The total number of packets sent before 2 consecutive losses is observed.

## 5 Additional Cases - When server and client are on two different devices

To perform this, In client part instead of LocalHost, the address to be added is the Local IPv4 address of the device where server is hosted.

```
server_addr = ("192.168.43.62",12000)
Local IP Address is 192.168.43.62
```

When we send packets from the client to the server, they are routed through the local network infrastructure (like a switch, router, or wireless access point).
For this to happen, the client must know the actual IP address of the server on that network. Using the local IP ensures the data packets are routed to the correct destination on the same network.

## 6 Assumptions

### 6.1 Part (a)

— In average RTT calculation, only those packets are considered, those which were able to be delivered successfully to the server.
— RTT is not calculated for the packets which were dropped, and hence, not taken in any min max or avg RTT calculations.
— This message format assumed to be from the client side -

```
Ping_Number: 9 Time: Tue Sep 17 08:45:53 2024
```

— Modules such as time module are used for time calculations.
— Buffer size for the received message is assumed to be 4096.
— The Socket type and port number are taken from the given server code, for the client as well.
— It is assumed when clients and server are on the same network, while checking for different devices.
— Reptition time is taken to be a small value less than the timeout time.

### 6.2 Part(b)

— The message sent by the server contains only the time difference calculated in receiving the message and replying to the message.
— The server is assumed to be closed by the client after receiving 3 consecutive packet time out messages.
— The timeout time is taken to be 1 second as given in the question.

## 7 References

https ://www.ibm.com/docs/en/aix/7.1 ?topic=protocols-socket-types
https ://docs.python.org/3/library/socket.html
https ://docs.python.org/3/library/time.html
SocketProgamming-1-Tut1 and Tut2, CSE232
https ://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/