

CO 223 - Computer Communication Networks I

Lab 05 - Socket Programming in Python II

Date - 01/10/2021

Reg No - E/18/397

Task 1

1)

```
server.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20001
5  buffersize = 1024
6  msgFromServer = "Hello UDP client"
7  bytesToSend = str.encode(msgFromServer)
8
9  # Create a datagram socket
10 UDPServerSocket = socket.socket(family=socket.AF_INET, type = socket.SOCK_DGRAM)
11
12 # Bind to address and IP
13 UDPServerSocket.bind((localIP, localPort))
14
15 print("UDP server up and listening")
16
17 while(True):
18     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
19     message = bytesAddressPair[0]
20     address = bytesAddressPair[1]
21     clientmsg = "message from client: " + message.decode()
22     clientIP = "client IP address: {}".format(address)
23
24     print(clientmsg)
25     print(clientIP)
26
27     # sending reply to client
28     UDPServerSocket.sendto(bytesToSend, address)
29
```

Figure 1.1: server code for task 1

```

client.py > ...
1  import socket
2
3  msgFromClient = "Hello UDP Server"
4  bytesToSend = str.encode(msgFromClient)
5  serverAddressPort = ("127.0.0.1",20001)
6  buffersize = 1024
7
8  # create UDP socket at client side
9  UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
10
11 # Send to server using created UDP socket
12
13 UDPClientSocket.sendto(bytesToSend,serverAddressPort)
14
15 msgFromServer = UDPClientSocket.recvfrom(buffersize)
16 msg = "Message from server: "+msgFromServer[0].decode()
17
18 print(msg)
19

```

Figure 1.2: client code for task 1

The screenshot shows two side-by-side terminal windows. The left window displays the output of running 'python server.py', which includes the message 'UDP server up and listening'. The right window shows the command prompt 'PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2>' with a cursor, indicating it is ready for the next command.

Figure 1.3: both terminals after running only server code

The screenshot shows two side-by-side terminal windows. The left window shows the output of running 'python client.py', which includes the message 'message from client: Hello UDP Server' and the client IP address. The right window shows the output of running 'python server.py', which includes the message 'Message from server: Hello UDP Client'.

Figure 1.4: Both terminal after running client code

2) Since UDP is connectionless or stateless protocol, server may respond to every request

3) Handling millions of requests depends on the load balancing. Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool. A load balancer is a reverse proxy. It presents a virtual IP address (VIP) representing the application to the client. The client connects to the VIP and the load balancer decides through its algorithms to send the connection to a specific application instance on a server. The load balancer continues to manage and monitor the connection for the entire duration.

Task 2

1)

```
task 2 > server1.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20001
5  buffersize = 1024
6  msgFromServer = "Hello UDP Client(from server 1)"
7  bytesToSend = str.encode(msgFromServer)
8
9  # Create a datagram socket
10 UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)
11
12 # Bind to address and IP
13 UDPServerSocket.bind((localIP,localPort))
14
15 print("UDP server 1 up and listening")
16
17 while(True):
18     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
19     message = bytesAddressPair[0]
20     address = bytesAddressPair[1]
21     clientmsg = "message from client: "+ message.decode()
22     clientIP = "client IP address: {}".format(address)
23
24     print(clientmsg)
25     print(clientIP)
26
27     # sending reply to client
28     UDPServerSocket.sendto(bytesToSend, address)
29
```

Figure 2.1: server 1 code

```
task 2 > server2.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20000
5  buffersize = 1024
6  msgFromServer = "Hello UDP Client(from server 2)"
7  bytesToSend = str.encode(msgFromServer)
8
9  # Create a datagram socket
10 UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)
11
12 # Bind to address and IP
13 UDPServerSocket.bind((localIP,localPort))
14
15 print("UDP server 2 up and listening")
16
17 while(True):
18     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
19     message = bytesAddressPair[0]
20     address = bytesAddressPair[1]
21     clientmsg = "message from client: "+ message.decode()
22     clientIP = "client IP address: {}".format(address)
23
24     print(clientmsg)
25     print(clientIP)
26
27     # sending reply to client
28     UDPServerSocket.sendto(bytesToSend, address)
29
```

Figure 2.2: server 2 code

```

task 2 > client.py > ...
1  import socket
2
3  msgFromClient = "Helllo UDP Server"
4  bytesToSend = str.encode(msgFromClient)
5  serverAddressPort = ("127.0.0.1",20001)
6  serverAddressPort2 = ("127.0.0.1",20000)
7  buffersize = 1024
8
9  # create UDP socket at client side
10 UDPClietSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
11
12 # Send to server using created UDP socket
13 UDPClietSocket.sendto(bytesToSend,serverAddressPort)
14 UDPClietSocket.sendto(bytesToSend,serverAddressPort2)
15
16 # receiving and printing the messahge from 1 st server
17 msgFromServer = UDPClietSocket.recvfrom(buffersize)
18 msg1 = "Message from server 1: "+msgFromServer[0].decode()
19 print(msg1)
20
21 # receiving and printing the messagee from 2 nd server
22 msgFromServer = UDPClietSocket.recvfrom(buffersize)
23 msg2 = "Message from server 2: "+msgFromServer[0].decode()
24 print(msg2)
25

```

Figure 2.3: client code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		<pre> PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket progr amming lab 2\task 2> python server1.py UDP server 1 up and listening </pre>	<pre> PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket progr amming lab 2\task 2> python server2.py UDP server 2 up and listening </pre>

Figure 2.4: Terminals after running both servers

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE	
		<pre>PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\task 2> python server1.py UDP server 1 up and listening message from client: Hello UDP Server client IP address: ('127.0.0.1', 58819) █</pre>	<pre>PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\task 2> python server2.py UDP server 2 up and listening message from client: Hello UDP Server client IP address: ('127.0.0.1', 58819) █</pre>	<pre>PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\task 2> python client.py Message from server 1: Hello UDP Client (from server 1) Message from server 2: Hello UDP Client (from server 2) PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\task 2> █</pre>

Figure 2.5: Terminals after running client code

2) yes, it's possible.

- Socket specifies both server addresses to send. Since UDP is not connection oriented and UDP sockets are not classified to client socket and server socket, it is able to accept several concurrent connections from different remote hosts. Therefore, it's possible to create multiple servers which listens and communicate with a single client.

Task 3

1) since UDP is connectionless and only sending datagrams usage of `connect()` won't results anything. Like TCP there isn't any 3-way handshake. For datagram sockets, the `connect()` call specifies the peer for a socket.

2) As mentioned above `connect()` on UDP specifies the peer for a socket. Therefore, no need of specify the destination IP address and the port. Thus we can use `send()` instead of `sendto()` and we can use `recv()` instead of `recvfrom()` in the client code.

```
server3.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20001
5  buffersize = 1024
6  msgFromServer = "Hello UDP Client"
7  bytesToSend = str.encode(msgFromServer)
8
9  # Create a datagram socket
10 UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)
11
12 # Bind to address and IP
13 UDPServerSocket.bind((localIP,localPort))
14
15 print("UDP server up and listening")
16
17 while(True):
18     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
19     message = bytesAddressPair[0]
20     address = bytesAddressPair[1]
21     clientmsg = "message from client: {}".format(message)
22     clientIP = "client IP address: {}".format(address)
23
24     print(clientmsg)
25     print(clientIP)
26
27     # sending reply to client
28     UDPServerSocket.sendto(bytesToSend,address)
29
```

Figure 3.1: server code


```

client3.py > ...
1  import socket
2
3  msgFromClient = "Hello UDP Server"
4  bytesToSend = str.encode(msgFromClient)
5  buffersize = 1024
6
7  # create UDP socket at client side
8  UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
9  UDPClientSocket.connect(("127.0.0.1",20001))
10 # Send to server using created UDP socket
11
12 UDPClientSocket.send(bytesToSend)
13
14 msgFromServer = UDPClientSocket.recv(buffersize).decode()
15 msg = "Message from server: "+msgFromServer
16
17 print(msg)
18
19

```

Figure 3.2: client code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		<pre> n reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'. PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 3> python server3.py UDP server up and listening </pre>	<pre> n reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'. PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 3> </pre>

Figure 3.3: both terminals after running server code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		<pre> n reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'. PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 3> python server3.py UDP server up and listening message from client: b'Hello UDP Server' client IP address: ('127.0.0.1', 60269) </pre>	<pre> n reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'. PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 3> python client3.py Message from server: Hello UDP Client PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 3> </pre>

Figure 3.4: both terminals after running client code

Task 4

1)

```
server.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20001
5  buffersize = 1024
6  msgFromServer = "Hello UDP Client"
7  bytesToSend = str.encode(msgFromServer)
8
9  # Create a datagram socket
10 UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)
11
12 # Bind to address and IP
13 UDPServerSocket.bind((localIP,localPort))
14
15 print("UDP server up and listening")
16
17 while(True):
18     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
19     message = bytesAddressPair[0]
20     address = bytesAddressPair[1]
21     clientmsg = "Number from client: {}".format(message.decode())
22     clientIP = "client IP address: {}".format(address)
23
24     print(clientmsg)
25     print(clientIP)
26
27     # sending reply to client
28     UDPServerSocket.sendto(bytesToSend, address)
29
```

Figure 4.1.1: server code

```

client.py > ...
1  import socket
2
3  serverAddressPort = ("127.0.0.1",20001)
4  buffersize = 1024
5
6  # create UDP socket at client side
7  UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
8
9  # getting number from user
10 msgFromClient = input("Enter a Number: ")
11 bytesToSend = str.encode(msgFromClient)
12
13 # Send to server using created UDP socket
14 UDPClientSocket.sendto(bytesToSend,serverAddressPort)
15
16 msgFromServer = UDPClientSocket.recvfrom(buffersize)
17 msg = "Message from server: "+msgFromServer[0].decode()
18
19 print(msg)
20

```

Figure 4.1.2: client code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> python server.py UDP server up and listening █	PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> █

Figure 4.1.3: both terminals after running server code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> python server.py UDP server up and listening Number from client: 99 client IP address: ('127.0.0.1', 50361) █	PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> python client.py Enter a Number: 99 Message from server: Hello UDP Client PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> █

Figure 4.1.4: both terminals after user sending a number

2)

```
server2.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20001
5  buffersize = 1024
6
7  # Create a datagram socket
8  UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)
9
10 # Bind to address and IP
11 UDPServerSocket.bind((localIP,localPort))
12
13 print("UDP server up and listening")
14
15 while(True):
16     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
17     message = bytesAddressPair[0]
18     address = bytesAddressPair[1]
19
20     clientmsg = "Number from client: {}".format(message.decode())
21
22     print(clientmsg)
23
24     # sending reply to client
25     UDPServerSocket.sendto(message, address)
```

Figure 4.2.1: server code

```

client2.py > ...
1  import socket
2
3  serverAddressPort = ("127.0.0.1",20001)
4  buffersize = 1024
5
6  # create UDP socket at client side
7  UDPCliientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
8
9  # getting number from user
10 n =int(input("Enter a Number: "))
11
12 # loop to send numbers to the server
13 for i in range(0,n):
14     bytesToSend = str.encode(str(i+1))
15
16     # Send to server using created UDP socket
17     UDPCliientSocket.sendto(bytesToSend,serverAddressPort)
18
19 # loop to recceive messages from the server
20 while (True):
21     msgFromServer = UDPCliientSocket.recvfrom(buffersize)
22     msg = "Number from server: "+msgFromServer[0].decode()
23
24     print(msg)
25
26

```

Figure 4.2.2: client code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> python server2.py UDP server up and listening █	PS D:\my files\Current\CO 223 Computer Communication Network s\lab\Socket programming lab 2\Task 4> █

Figure 4.2.3: both terminals after running server code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 4> python server2.py UDP server up and listening Number from client: 1 Number from client: 2 Number from client: 3 Number from client: 4 Number from client: 5 Number from client: 6 Number from client: 7 Number from client: 8 █	PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 4> python client2.py Enter a Number: 8 Number from server: 1 Number from server: 2 Number from server: 3 Number from server: 4 Number from server: 5 Number from server: 6 Number from server: 7 Number from server: 8 █

Figure 4.2.4: both terminal after entering a number by client

3) Same code used as the above

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		Number from client: 16144 Number from client: 16145 Number from client: 16146 Number from client: 16147 Number from client: 16148 Number from client: 16149 Number from client: 16150 Number from client: 16151 Number from client: 16152 Number from client: 16153 Number from client: 16154 Number from client: 16155 Number from client: 16156 Number from client: 16159 Number from client: 16175 Number from client: 16189 Number from client: 16219 Number from client: 16236 Number from client: 16249 Number from client: 16260 Number from client: 16284 Number from client: 16294 Number from client: 16306 Number from client: 16316 Number from client: 16339	Number from server: 16144 Number from server: 16145 Number from server: 16146 Number from server: 16147 Number from server: 16148 Number from server: 16149 Number from server: 16150 Number from server: 16151 Number from server: 16152 Number from server: 16153 Number from server: 16154 Number from server: 16155 Number from server: 16156 Number from server: 16159 Number from server: 16175 Number from server: 16189 Number from server: 16219 Number from server: 16236 Number from server: 16249 Number from server: 16260 Number from server: 16284 Number from server: 16294 Number from server: 16306 Number from server: 16316 Number from server: 16339

Figure 4.3.1: both terminals after entering a larger n value (after 16156 packet loss can see)

- UDP isn't reliable protocol. The UDP does not require a connection, and it will not resend data packets if there are errors. On every UDP socket, there's a "socket send buffer" that we put packets into. So, if we have a network card that's too slow or something, it's possible that it will not be able to send the packets as fast as we put them in. Therefore, packets will loss.

4)

```
server4.py > ...
1  import socket
2
3  localIP = "127.0.0.1"
4  localPort = 20001
5  buffersize = 1024
6
7  # Create a datagram socket
8  UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)
9
10 # changing the boffer size
11 UDPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 200000)
12
13 # Bind to address and IP
14 UDPServerSocket.bind((localIP,localPort))
15
16 print("UDP server up and listening")
17
18 while(True):
19     bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
20     message = bytesAddressPair[0]
21     address = bytesAddressPair[1]
22
23     clientmsg = "Number from client: {}".format(message.decode())
24
25     print(clientmsg)
26
27     # sending reply to client
28     UDPServerSocket.sendto(message, address)
29
```

Figure 4.4.1: server code

```

client4.py > ...
1  import socket
2
3  serverAddressPort = ("127.0.0.1",20001)
4  buffersize = 1024
5
6  # create UDP socket at client side
7  UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
8
9  # getting number from user
10 n =int(input("Enter a Number: "))
11
12 # loop to send numbers to the server
13 for i in range(0,n):
14     bytesToSend = str.encode(str(i+1))
15
16     # Send to server using created UDP socket
17     UDPClientSocket.sendto(bytesToSend,serverAddressPort)
18
19 # loop to receive messages from the server
20 while (True):
21     msgFromServer = UDPClientSocket.recvfrom(buffersize)
22     msg = "Number from server: "+msgFromServer[0].decode()
23
24     print(msg)
25

```

Figure 4.4.2: client code

PROBLEMS	OUTPUT	TERMINAL	DEBUG CONSOLE
		Number from client: 19989	Number from server: 19989
		Number from client: 19990	Number from server: 19990
		Number from client: 19991	Number from server: 19991
		Number from client: 19992	Number from server: 19992
		Number from client: 19993	Number from server: 19993
		Number from client: 19994	Number from server: 19994
		Number from client: 19995	Number from server: 19995
		Number from client: 19996	Number from server: 19996
		Number from client: 19997	Number from server: 19997
		Number from client: 19998	Number from server: 19998
		Number from client: 19999	Number from server: 19999
		Number from client: 20000	Number from server: 20000

Figure 4.4.3: end of the both terminals after entering n as 20000

- By using `socket.setsockopt()` we can increase receive buffer size. By increasing buffer size, we can reduce UDP packet loss. That's why our code is working fine without showing any data loss.