EXPERIMENT 1 - SIMPLE CLIENT-SERVER APPLICATION DEVELOPMENT USING PYTHON

AIM:

Develop a simple client-server application using Python language protocols to be verified with User Datagram Protocol (UDP) and Transmission Control Protocol (TCP).

SOFTWARE:

1. Oracle VM VirtualBox 6.1.38, Oracle Corporation
2. Ubuntu 22.04 (64-bit) Operating System
3. Microsoft Visual Studio Code, Microsoft Corporation
4. Python 3.10.7 (64-bit), Python Software Foundation

THEORY:

I. Key Terminologies:

1. Server – It is simply a computer that can be used to share resources with other computers based on request.
2. Client – It is the other computer(s) that request resources from the server.
3. Client-Server network: Clients can access resources and services from a central computer using a client-server network, which can be either a local area network (LAN) or a wide-area network (WAN), like the Internet.
4. Protocol – An established set of guidelines that govern how data is transferred between various devices connected to the same network is known as a network protocol. In essence, it enables interconnected devices to interact with one another despite any variations in their internal workings, organisational structures, or aesthetics.
5. IP Address – A device on the internet or a local network can be identified by its IP address, which is a special address. The rules defining the format of data delivered over the internet or a local network are known as "Internet Protocol," or IP.
6. Packet Fragmentation – For the resulting fragmentation to fit across a link with a smaller maximum transmission unit (MTU) than the original packet size, IP fragmentation is a technique that divides packets into smaller units of the Internet Protocol. The receiving host puts the pieces back together. Both the specifics of the technique for fragmentation and the broader architectural strategy for fragmentation differ between IPv4 and IPv6.
7. Socket – One endpoint of a two-way communication channel between two network-running programmes is a socket. For the TCP layer to recognise the application that data is intended to be transferred to, a socket is tied to a port number. A port number and an IP address make up an endpoint.

II. What is User Datagram Protocol (UDP)?

UDP decided to use the rudimentary method of marking each UDP packet with a pair of unsigned 16-bit port numbers in the range of 0 to 65,536 to distinguish between distinct talks.

The application at the destination IP address to which the communication should be sent is specified by the destination port, whereas the source port identifies the specific process or programme that submitted the packet from the source system.

III. How does UDP work?

A straightforward stateless protocol is UDP. Although this simplicity is excellent for straightforward applications, dependability issues arise. Delivering UDP packets across Network Address Translator (NAT) devices in particular is challenging. Let's review what NAT devices truly do so that you can see why.

As the Internet grew in use, it became impracticable to assign each host a different IP address. The NAT protocol was developed to address this issue by enabling a device on the edge of a network to advertise a publicly available, globally unique IP address for other hosts to connect to, and mapping that to an internal IP address that is distinct from the internal network. This makes it possible for numerous networks to share the same local IP address space.

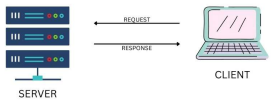

Figure 1 – Block Diagram of User Datagram Protocol (UDP)

NAT devices function by adhering to TCP's connection (and teardown) protocol and keeping a routing table for TCP-based connections. A new NAT routing entry is established when an application creates a TCP connection, and the NAT routing entry can be erased once the TCP connection has been terminated.

Since UDP has no connection protocol and no state, it is very difficult for NAT devices to maintain an accurate routing table for UDP datagrams. Any NAT device used between two hosts must be explicitly configured to port forward UDP traffic to connect to devices inside the network.

IV. Applications of UDP:

1. Used for straightforward request-response communication when data size is smaller and flow and error control are less of a concern. Because UDP provides packet switching, it is an appropriate protocol for multicasting.
2. Some routing update protocols, such as RIP, employ UDP (Routing Information Protocol). Usually applied to real-time applications that can't stand inconsistent delays in a message's different parts.
3. Following implementations use UDP as a transport layer protocol -
    - NTP (Network Time Protocol)

- DNS (Domain Name Service)
- BOOTP, DHCP
- NNP (Network News Protocol)
- Quote of the Day Protocol
- TFTP, RTSP, RIP.

4. The application layer can do some of the tasks through UDP –
    - Trace Route
    - Record Route
    - Timestamp
5. UDP takes a datagram from Network Layer, attaches its header, and sends it to the user. So, it works fast.
6. UDP is a null protocol if you remove the checksum field. For example,
    - Reduce the requirement for computer resources.
    - When using the Multicast or Broadcast to transfer.
    - The transmission of Real-time packets, mainly in multimedia applications.

V. Advantages of UDP:

1. It employs a tiny packet size and a tiny header (8 bytes). The UDP protocol requires less memory and less processing time since there are fewer bytes in the overhead.
2. It does not need a connection to be made and kept up. The absence of an acknowledgement field in UDP speeds up communication because there is no need to wait for an acknowledgement (ACK) or store data in memory until it is received.
3. For error detection, it employs a checksum on each packet. It can be applied to situations in which only one packet of data needs to be transmitted between the hosts.

VI. Disadvantages of UDP:

1. It is an unreliable, connectionless transport protocol. No windowing or mechanism exists to guarantee that data is received in the same sequence as it was transmitted.
2. It makes no use of error checking. As a result, if UDP finds a mistake in the received packet, it discreetly discards it.
3. There is no traffic management. As a result, congestion can be caused by several users sending enormous amounts of data via UDP, and nothing can be done to stop it.
4. Error recovery is only dealt with by the application layer. Application developers can therefore just ask the user to send the message again.
5. Routers may use UDP carelessly. They frequently discard UDP packets before TCP packets and do not retransmit a UDP datagram following a collision. There is no flow control and no acknowledgement for received data.

VII. What is Transmission Control Protocol (TCP)?

Transmission Control Protocol, or TCP, is a communications standard that enables computer hardware and software to exchange messages over a network. It is made to send packets across the internet and make sure that data and messages are successfully sent through networks.

TCP is one of the fundamental standards that the Internet Engineering Task Force (IETF) has specified as the guidelines for the internet (IETF). It provides end-to-end data delivery and is one of the most widely utilised protocols in digital network communications. Data is organised by TCP before being sent between a server and a client. It ensures the accuracy of the data transmitted via a network, before data transmission.

VIII. How does TCP work?

The TCP/IP paradigm divides the data into little bundles and then reassembles the bundles into the original message on the other end to ensure at each message reaches its target place intact. Instead of transmitting everything at once, delivering the information in smaller bundles of information makes it easier to retain efficiency.

When a message is divided into bundles, if one route is congested but the destination is the same, the bundles may go along other routes. For instance, when a user requests a web page on the internet, the server processes the request and delivers the user a sense in the form of an HTML page. The HTTP Protocol is the one that the server uses.

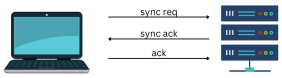

Figure 1 – Block Diagram of Transmission Control Protocol (TCP)

The TCP layer is then asked by HTTP to establish the necessary connection and transfer the HTML file. The data is now divided into individual packets by the TCP, which then sends them to the IP layer. The packets are subsequently transmitted via several pathways to their destination. When the transmission is complete and all packets have been received, the TCP layer in the user's system acknowledges.

IX. Applications of TCP:

1. Segment Numbering System - TCP keeps track of the segments being transmitted or received by allocating numbers to every one of them, which is one of its most noticeable features. Data bytes that need to be sent are given a specific byte number, and segments are given sequence numbers. Segments that have been received are given acknowledgement numbers.
2. Flow Control – The rate at which a sender delivers data is constrained by flow control. To guarantee dependable delivery, this is done. The sender is constantly informed by the receiver how much data can be received (using a sliding window)
3. Error Control – To ensure dependable data transport, TCP uses an error control technique. Byte-Oriented Error Control and Segment-Based Error Detection

and Control include, Corrupted Segment & Lost Segment Management, Out-of-Order Segments, Duplicate Segments, etc.

4. Congestion Control – TCP takes into account the level of congestion in the network. Congestion level is determined by the amount of data sent by a sender. It assigns an IP address to each computer on the network and a domain name to each site, making each device site distinct over the network. Open Protocol, is not owned by any organisation or individual.

X. Advantages of TCP:

1. This protocol is dependent; it offers an error checking system and a recovery mechanism; it controls flow; it ensures that data reaches its destination in the precise sequence in which it was sent and it proves flow control.
2. TCP is designed for wide-area networks, therefore tiny networks with limited resources may have trouble with their scale. Additionally, because TCP runs on multiple layers, the network's speed may be slowed down. This means that it can only represent the TCP/IP suite of protocols.
3. Its nature is not generic. This means that it can only represent the TCP/IP suite of protocols. For instance, a Bluetooth connection is not compatible. Since their creation, around 30 years ago, there have been no changes.

XI. Disadvantages of TCP:

1. TCP is a complicated model to set up and manage and control. The shallow/overhead of TCP is higher than IPX. Replacing protocol in TCP is not easy.
2. It does not separate the concept of service, interface, and protocols. So, it is not suitable to describe new technologies in the new network. In this model, the transport layer does not guarantee the delivery of packets.
3. It has no clear operations from its services, interfaces, and protocols. It is not generic in nature. So, it fails or loses to represent any protocol stack other than the TCP/IP suite.
4. It was originally designed and implemented for a wide area network. It is not optimized for small networks like LAN and PAN. TCP can't be used for broadcast and multicast connections.
5. All the work is being done by your OS, so if there are bugs in your OS, then you will face many problems like problems surfing and downloading from the net.

SOURCE CODE:

UDP Server

```
import socket # Low-level networking interface

if __name__ == "__main__":

    localIP = "127.0.0.1" # IP address of the local host
    localPort = 20001 # Port Number
    bufferSize  = 1024 # Byte Size

    UDPServerSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM) # Create a datagram socket, belonging to IPv4 family
and following connection-less UDP protocol
    UDPServerSocket.bind((localIP, localPort)) # Bind the socket with IP
address and port number
    print("\nUDP Server Up & Listening...")

    # Listen for incoming datagrams:-
    while(True):
        data, address = UDPServerSocket.recvfrom(bufferSize) # Receive the
address to send the data back
        data = data.decode("UTF-8")
        print(f"Client: {data}")

        data = data.upper()
        data = data.encode("UTF-8")
        UDPServerSocket.sendto(data, address) # Sending a reply to client
```

UDP Client

```
import socket # Low-level networking interface

if __name__ == "__main__":

    localIP = "127.0.0.1" # IP address of the local host
    localPort = 20001 # Port Number
    serverAddressPort = (localIP, localPort)
    bufferSize = 1024 # Byte Size

    UDPClientSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM) # Create a UDP datagram socket at the client side,
belonging to IPv4 family and following connection-less UDP protocol

    while(True):
        data = input("\nEnter data: ")
        data = data.encode("UTF-8") # Encode the string
```

```
        UDPClientSocket.sendto(data, serverAddressPort) # Sending a reply to
server

        data, address = UDPClientSocket.recvfrom(bufferSize) # Receive the
address to send the data back
        data = data.decode("UTF-8") # Decode the bytes
        print(f"Server: {data}")
```

TCP Server

```
import socketserver # Simplifies the task of writing network servers
bufferSize = 1024 # Byte Size

# The TCP Server class for demonstration:-
class Handler_TCPServer(socketserver.BaseRequestHandler):

    # We need to implement the Handle method to exchange data with TCP
client:-
    def handle(self):

        self.data = self.request.recv(bufferSize).strip() # TCP socket
connected to the client for receiving data from the socket
        print("\n{} sent: ".format(self.client_address[0]) + str(self.data))
        self.request.sendall("Acknowledgment from TCP Server!\n".encode()) #
Send back ACK for data arrival confirmation

if __name__ == "__main__":

    HOST, PORT = "127.0.0.1", 9999 # IP Address of the local host and port
number
    tcp_server = socketserver.TCPServer((HOST, PORT), Handler_TCPServer) #
Initialize the TCP server object and bind it to the localhost on the 9999 port
    tcp_server.serve_forever() # Activate the TCP server

    # To abort the TCP server, press Ctrl + C.
```

TCP Client

```
import socket # Low-level networking interface

host_ip, server_port = "127.0.0.1", 9999 # IP Address of the local host and
port number
bufferSize = 1024 # Byte Size
data = input("\nEnter data: ")
tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create a UDP
datagram socket at the client side, belonging to IPv4 family and following TCP
protocol
```

```
try: # Some Code...

    # Establish connection to TCP server and exchange data:-
    tcp_client.connect((host_ip, server_port)) # Connect to a remote socket
address
    tcp_client.sendall(data.encode()) # Send data to the socket
    received = tcp_client.recv(bufferSize) # Receive data from the socket

finally: # Some Code... (ALWAYS EXECUTED)
    tcp_client.close() # Mark the socket closed

print ("Bytes Sent: {}".format(data))
print ("Bytes Received: {}\n".format(received.decode()))
```

SCREENSHOTS OF INPUTS AND OUTPUTS:

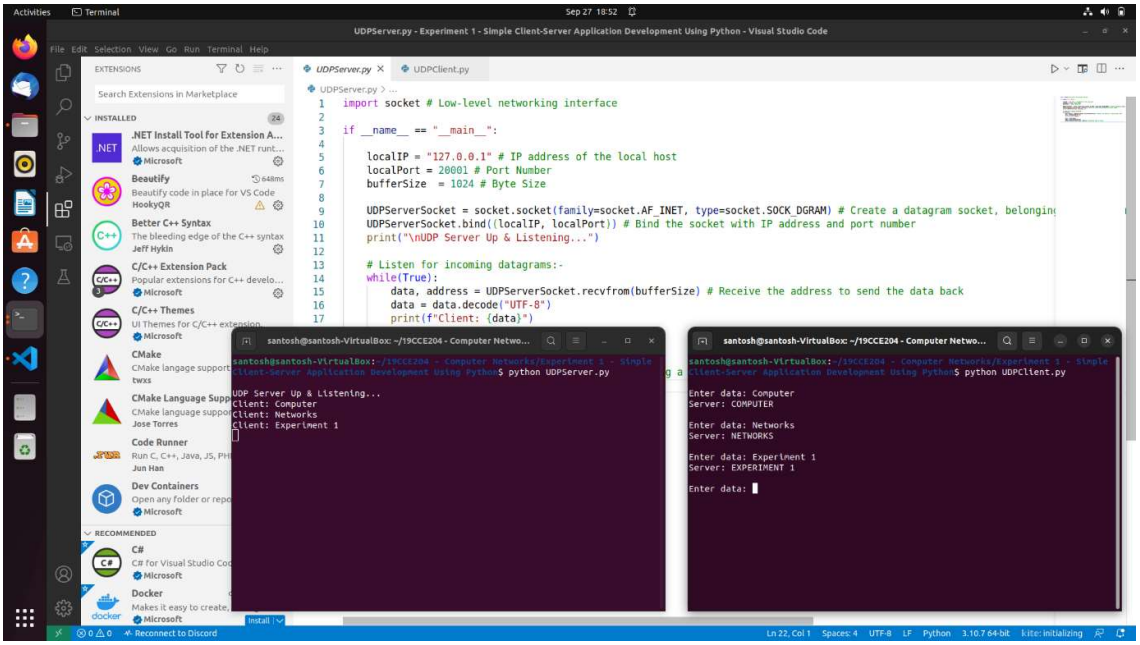

Figure 1 - User Datagram Protocol (UDP)

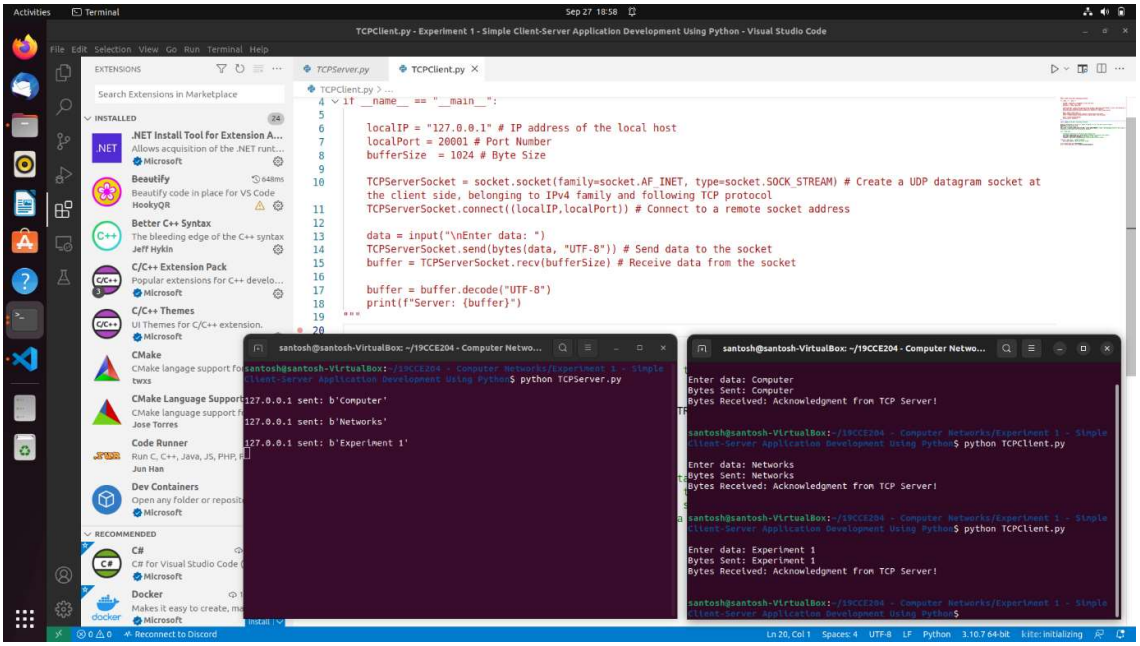

Figure 2 - Transmission Control Protocol (TCP)

CONCLUSION:

Thus, developed a simple client-server application using Python language protocols to be verified with UDP and TCP and all simulation results were verified successfully.

REFERENCES:

1. Foundations of Python Network Programming, Brandon Rhodes, John Goerzen, Apress Berkeley, CA, Third Edition, 2014 (Softcover ISBN: 978-1-4302-5854-4, eBook ISBN: 978-1-4302-5855-1)
2. Socket Module Python Documentation – https://docs.python.org/3/library/socket.html
3. Socket Server Module Python Documentation – https://docs.python.org/3/library/socketserver.html
4. UDP - Client and Server Example Programs In Python – https://pythontic.com/modules/socket/udp-client-server-example

CONTRIBUTION:

1. Narendran S [CB.EN.U4CCE20036] – Theory Documentation + Code Review
2. T Narun [CB.EN.U4CCE20037] – Topic Research + Code Debugging/Testing
3. Pabbathi Greeshma [CB.EN.U4CCE20040] – Code Work for UDP
4. Santosh [CB.EN.U4CCE20053] – Code Work for TCP