Client-Server Chat Messaging and Voice Call Application

Table of Contents

1. Introduction

This project involves the development of a chat messaging and voice call system using a client-server model. The server mediates all communication between clients, enabling real-time messaging and voice communication.

2. Objective

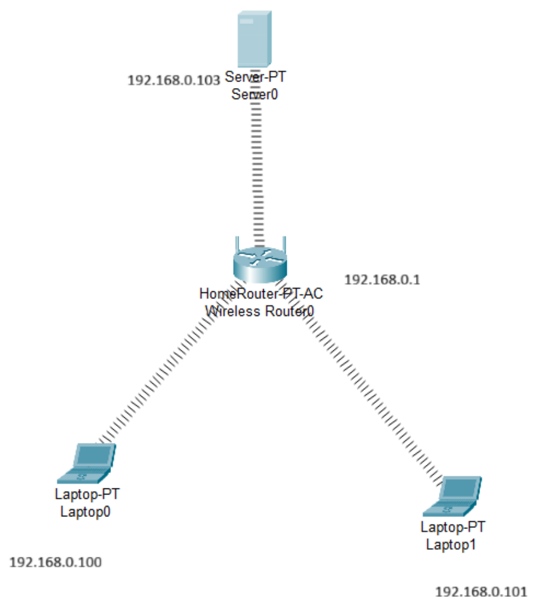
- Develop a functional client-server chat application.
- Implement real-time messaging and voice calls.
- Observe server-mediated communication and dynamic IP handling.

3. Equipment and Tools

- 3 Laptops (2 clients, 1 server)
- Python programming language
- NEDEUT Wi-Fi network for IP assignment
- Cisco Packet Tracer for diagram illustration

4. System Architecture

Client-Server Architecture diagram:



Server handles:
- Receiving and forwarding text messages
- Handling voice call streams
- Maintaining active client sessions

5. Subnet and Routing

- **Subnet mask used:** `/16` (255.255.0.0)
- **Network portion:** first two octets of the IP address.
- **Same-subnet communication:**
    - In a strict `/16` network, devices with different first two octets are technically in different subnets.
    - **In a real Wi-Fi network, the router bridges devices**, allowing them to communicate without manual routing.

- **Conclusion:**
  - No manual routing configuration was needed.
  - The server and clients communicated successfully via TCP sockets.

6. Features

- Real-time text messaging between clients
- **Voice call functionality**
- Centralized server to manage all communications
- Dynamic IP address handling
- Logs of client interactions
- Easy scalability to more clients

7. Implementation

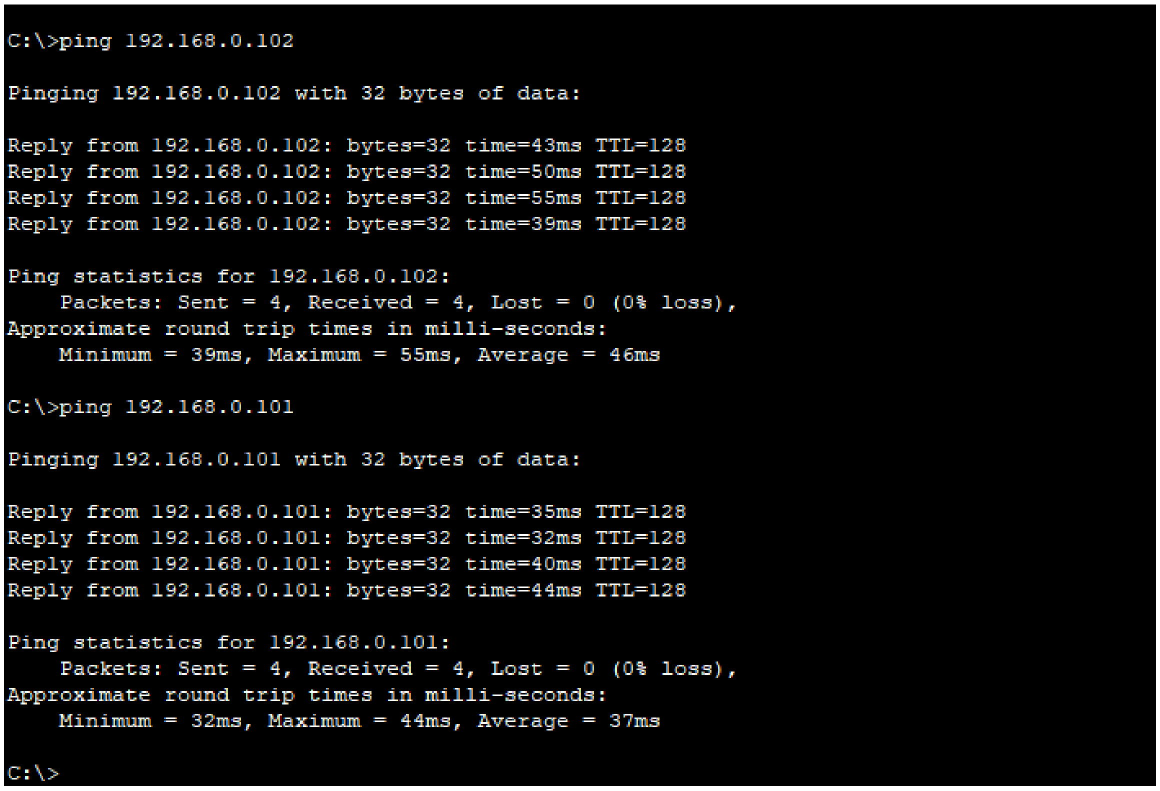
Step 1: Set up server to listen for client connections
Step 2: Implement client messaging module
Step 3: Implement voice call streaming module
Step 4: Test client-server communication
Step 5: Verify message delivery and voice call quality

8. Observations and Results

- Clients successfully exchange messages via the server
- Voice call works with minimal latency
- Dynamic IP addresses correctly assigned
- Server logs capture all communication events

PING TEST:

Pinging from laptop0 (192.168.0.100):



```
C:\>ping 192.168.0.102

Pinging 192.168.0.102 with 32 bytes of data:

Reply from 192.168.0.102: bytes=32 time=43ms TTL=128
Reply from 192.168.0.102: bytes=32 time=50ms TTL=128
Reply from 192.168.0.102: bytes=32 time=55ms TTL=128
Reply from 192.168.0.102: bytes=32 time=39ms TTL=128

Ping statistics for 192.168.0.102:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 39ms, Maximum = 55ms, Average = 46ms

C:\>ping 192.168.0.101

Pinging 192.168.0.101 with 32 bytes of data:

Reply from 192.168.0.101: bytes=32 time=35ms TTL=128
Reply from 192.168.0.101: bytes=32 time=32ms TTL=128
Reply from 192.168.0.101: bytes=32 time=40ms TTL=128
Reply from 192.168.0.101: bytes=32 time=44ms TTL=128

Ping statistics for 192.168.0.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 32ms, Maximum = 44ms, Average = 37ms

C:\>
```

9. Conclusion

Successfully built a Python-based client-server chat and voice call system. Demonstrates practical client-server communication, real-time messaging, and audio streaming.