PCET's
PIMPRI CHINCHWAD UNIVERSITY
Department of CSE - Artificial Intelligence & Data Science

VoIP (Voice over IP) Simulation using Socket Programming

DCCN Mini Project Report

Submitted by:
Rajkunvar Mohite 26
Swarangi Kothawade 34
Suraj Madane 50
Jay Godse 49

Under the guidance of:
Dr. Manisha Khadse

Department of CSE AI & DS
PCET's PIMPRI CHINCHWAD UNIVERSITY

November 13, 2025

Abstract

This mini-project implements a simple VoIP simulation that demonstrates real-time audio capture, packetization, transmission over UDP sockets, and playback. The implementation uses socket programming and audio libraries to achieve near real-time voice communication between two machines on the same network. The goal is to illustrate practical concepts in DCCN: latency, packet loss, buffering, and the trade-offs between TCP and UDP for voice traffic.

Contents

1 Introduction

Voice over IP (VoIP) is a technique for delivering voice communications over Internet Protocol (IP) networks. Modern systems convert analog voice to digital packets, transmit them over a network, and reconstruct audio at the receiver. Typical applications include voice calls and conferencing systems such as WhatsApp, Skype, and Zoom.

2 Objectives

- Design and implement a basic real-time voice communication system using socket programming.
- Demonstrate sending and receiving audio packets between two hosts using UDP.
- Evaluate key challenges such as latency, packet loss, and synchronization.
- Provide a modular codebase that can be extended (e.g., codec integration, encryption, GUI).

3 System Architecture

The system follows a simple pipeline: capture → encode/packetize → transmit via UDP → receive → decode/playback.

Microphone Input → Encoder / Packetizer → UDP Socket (Sender) → Netwo

Figure 1: High-level architecture of the VoIP simulation

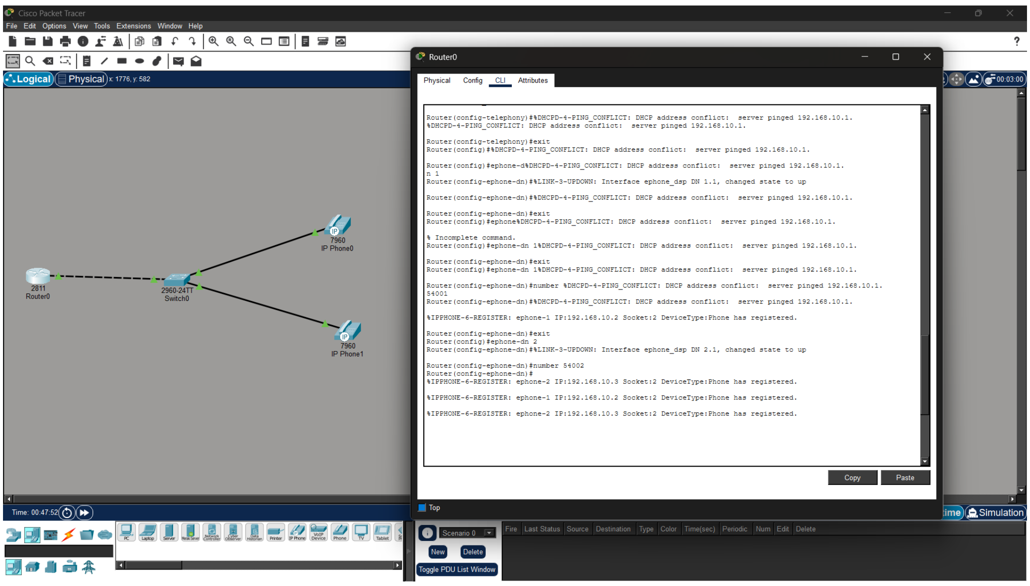

Figure 2: VoIP network topology in Cisco Packet Tracer

4 Modules

4.1 Audio Capture & Processing

- Capture audio using `PyAudio` (or platform-specific APIs).
- Use PCM encoding at a sensible sample rate (e.g., 16000 Hz or 44100 Hz) and a small frame size (e.g., 1024 bytes).

4.2 Socket Communication

- Use UDP sockets to transmit raw audio frames to conserve latency.
- Implement send and receive threads to allow full-duplex communication.

4.3 Playback

- Buffer incoming frames in a small jitter buffer to smooth packet arrival variations.
- Immediately write frames to the audio output stream for low delay.

4.4 User Interface (Optional)

A minimal GUI can be made using Tkinter or PyQt with buttons to Connect, Start Call, End Call, and show connection status.

5 Sample Implementation (Python)

Below is a compact example illustrating sender and receiver logic. This is a starting point for an Overleaf appendix or code listing.

5.1 Requirements

- Python 3.x
- `pyaudio`
- `socket`
- `threading`

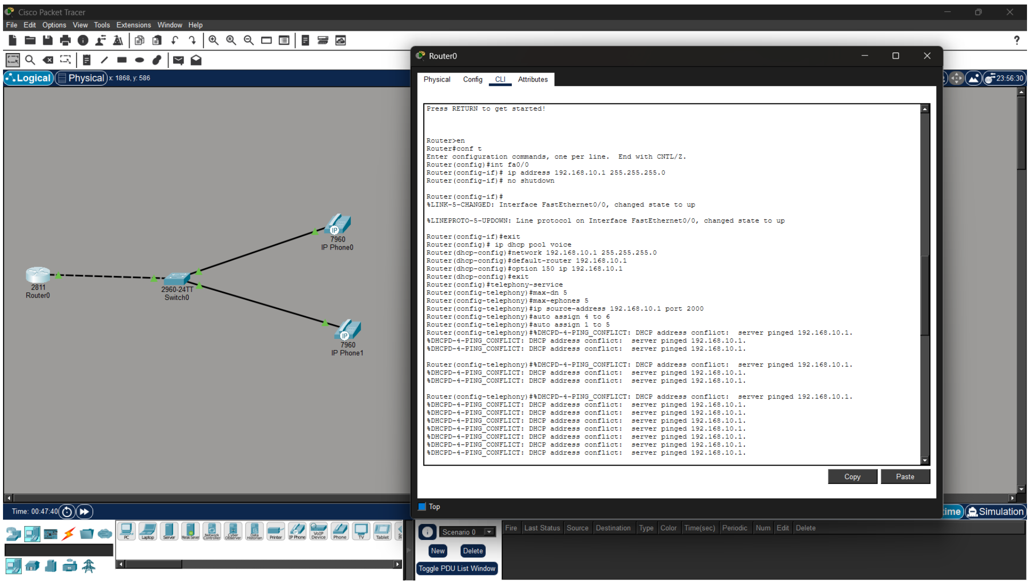

Figure 3: Router configuration and VoIP setup in Cisco Packet Tracer

5.2 Code (Simple Full-duplex UDP)

Listing 1: Full-duplex UDP VoIP (simplified)

```
# run this on both machines (adjust TARGET_IP on one side)
import socket
import pyaudio
import threading
CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
TARGET_IP = '192.168.1.2'  # change to peer IP
TARGET_PORT = 50005
LOCAL_PORT = 50005
p = pyaudio.PyAudio()
# open output stream (playback)
out_stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE,
   output=True, frames_per_buffer=CHUNK)
# open input stream (microphone)
in_stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE,
   input=True, frames_per_buffer=CHUNK)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("", LOCAL_PORT))
def send_audio():
    while True:
        try:
            data = in_stream.read(CHUNK, exception_on_overflow=
               False)
            sock.sendto(data, (TARGET_IP, TARGET_PORT))
        except Exception as e:
            print('Send error:', e)
            break
```

```python

def receive_audio():
    while True:
        try:
            packet, addr = sock.recvfrom(4096)
            out_stream.write(packet)
        except Exception as e:
            print('Receive error:', e)
            break
send_thread = threading.Thread(target=send_audio, daemon=True)
recv_thread = threading.Thread(target=receive_audio, daemon=True)
send_thread.start()
recv_thread.start()
send_thread.join()
recv_thread.join()
p.terminate()
```

6 Challenges and Considerations

- **Latency:** Keep frame sizes small (e.g., 512–2048 samples) and avoid excessive buffering.
- **Packet Loss:** UDP can drop packets resulting in audio gaps; adding simple packet-loss concealment improves quality.
- **Jitter:** Introduce a jitter buffer (small queue) to smooth arrival times; tune size to balance latency vs. smoothness.
- **Echo and Duplexing:** Full-duplex may cause echo; use hardware or software echo cancellation for cleaner calls.
- **Codecs:** Integrating codecs like G.711 or Opus reduces bandwidth and improves perceived quality.

7 Extensions (Extra Marks)

- Add a text chat channel multiplexed on the same socket or on a separate TCP channel.
- Add end-to-end encryption (e.g., SRTP or simple AES encryption of payloads).
- Implement a central server that handles multiple clients and mixes audio for conference calls.
- Integrate Opus codec (via libopus) for better compression and quality.

8 Testing and Results

Describe how you tested: two machines connected on the same LAN or two terminals on one machine with loopback. Measure round-trip latency using timestamps or by subjective listening. Note observable packet loss and perceived audio quality.

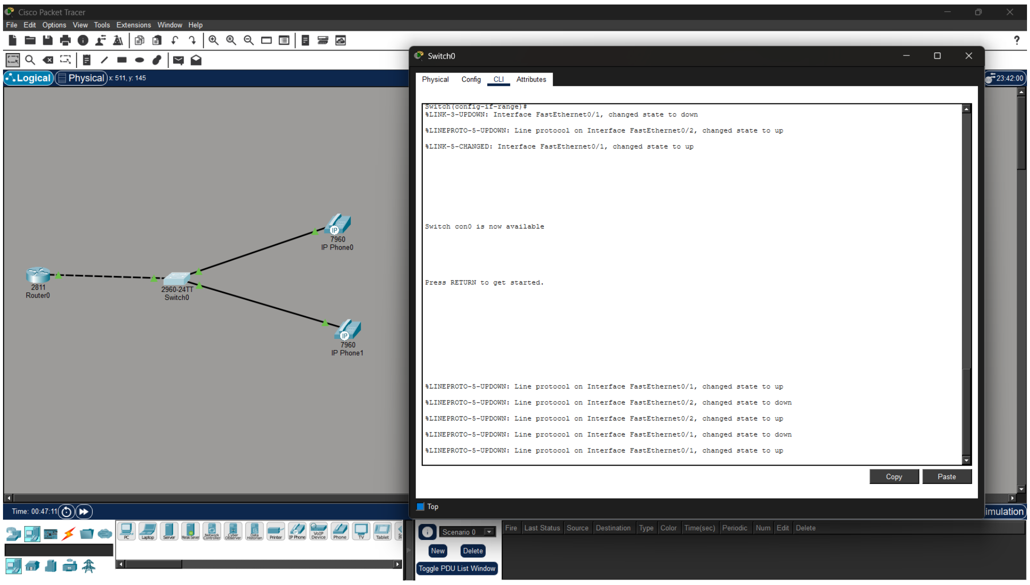

Figure 4: Switch configuration and active link status during VoIP simulation

9 Conclusion

This project demonstrates fundamental concepts of real-time audio streaming over IP networks using socket programming. It provides a platform to experiment with QoS, codecs, and network impairments in a controlled environment.

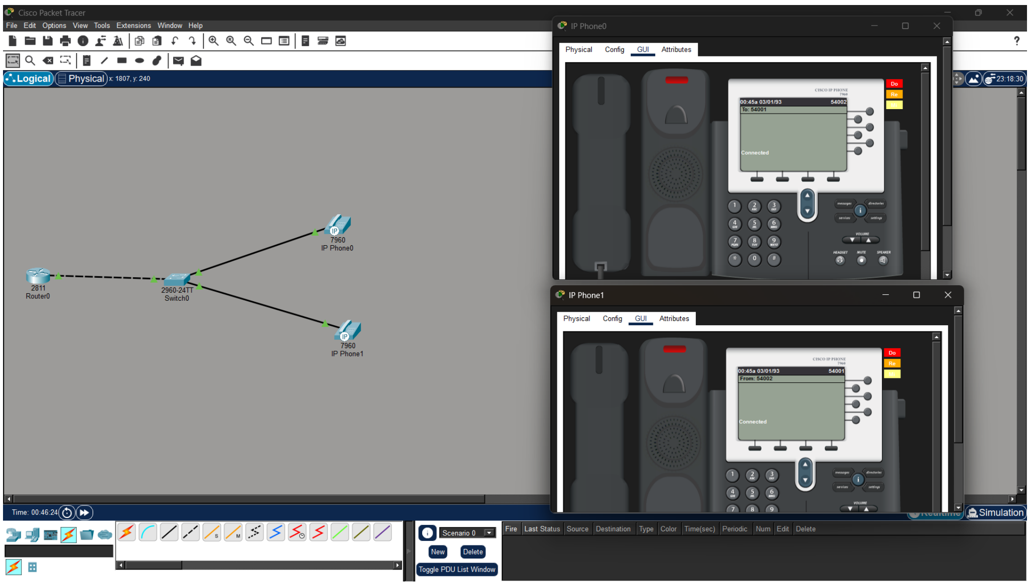

Figure 5: Successful VoIP call established between two IP phones in Cisco Packet Tracer

Appendix: Notes

- For production-grade VoIP, consider using existing libraries and protocols (RTP/RTCP, SIP, SRTP) and well-tested codecs (Opus).

- Always handle exceptions and close audio streams and sockets gracefully in your final implementation.

References

[1] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. (2003). RTP: A Transport Protocol for Real-Time Applications. RFC 3550.

[2] Valin, J.-M., Terriberry, T., Maxwell, G. (2012). Opus — Interactive Speech and Audio Codec. https://www.opus-codec.org