

# Chat-VoIP

Ioannis Michalainas, Maria Charisi

November 2024

## 1 Introduction

This report details the implementation of a project for the **Computer Networks II** course. The task involved creating an **end-to-end Chat and VoIP** application. The application facilitates **encrypted** messaging and voice communication over **UDP protocol**.

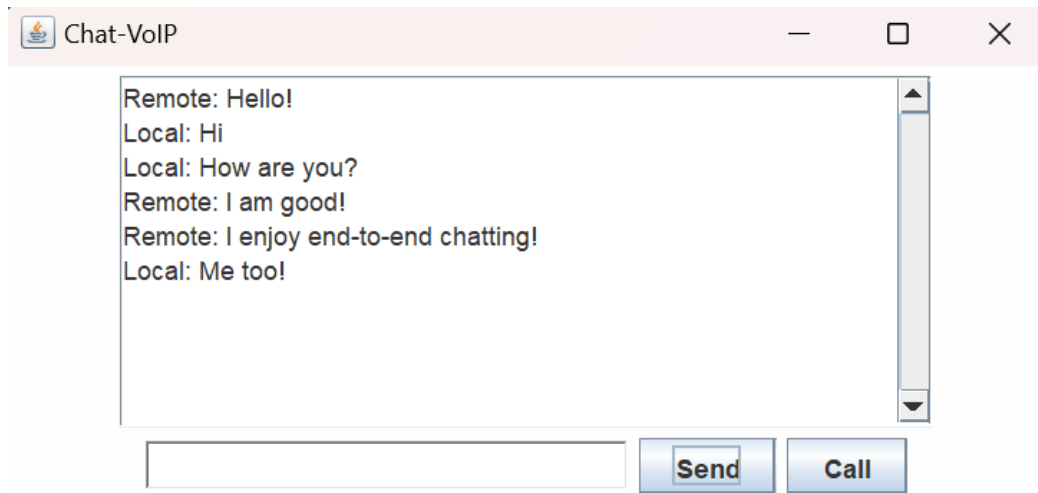


Figure 1: Chat Interface

## 2 Chat Implementation

The chat functionality is implemented using **UDP sockets**. A *dedicated thread* is used to listen for incoming messages on **port 12345**. Outgoing messages are encrypted and sent via a *separate thread* when the send button is pressed. Received messages are decrypted and displayed on the screen.

## 2.1 Receiving Messages

The following code snippet demonstrates the implementation of the listener thread for receiving messages:

```
1 // start a thread to listen for incoming messages
2 Thread receiveThread = new Thread(() -> {
3     DatagramSocket socket = null;
4     try {
5         // listen to port 12345 for inbound text
6         // messages
7         socket = new DatagramSocket(12345);
8         byte[] buffer = new byte[1024];
9
10        // continuously listen for incoming messages
11        while(true) {
12            DatagramPacket packet = new
13            DatagramPacket(buffer, buffer.length)
14            ;
15            socket.receive(packet);
16            String encryptedMessage = new String(
17            packet.getData(), 0, packet.getLength()
18            ());
19            String message = app.decryptMessage(
20            encryptedMessage);
21            textArea.append("Remote: " + message +
22            newline);
23        }
24    } catch(Exception ex) {
25        ex.printStackTrace();
26    } finally {
27        if(socket != null && !socket.isClosed()) {
28            socket.close();
29        }
30    }
31});
32receiveThread.start();
```

Listing 1: Receiving Messages

## 2.2 Sending Messages

When the send button is clicked, the following function is triggered, which encrypts and sends the message to the recipient.

```

1 private void handleSendButton() {
2     try {
3         // get user input
4         String message = inputTextField.getText();
5         String encryptedMessage = encryptMessage(
6             message);
7         byte[] buffer = encryptedMessage.getBytes();
8         InetAddress receiverAddress = InetAddress.
9             getByName("RECEIVER_IP"); // replace with
10                actual receiver IP
11         int port = 12345; // message port
12         DatagramPacket packet = new DatagramPacket(
13             buffer, buffer.length, receiverAddress,
14             port);
15
16         DatagramSocket socket = new DatagramSocket()
17             ;
18         socket.send(packet);
19         socket.close();
20
21         textArea.append("Local: " + message +
22             newline);
23         inputTextField.setText("");
24     } catch (Exception ex) {
25         ex.printStackTrace();
26     }
27 }

```

Listing 2: Sending Messages

We can verify the text packet traffic on port 12345 with **Wireshark**, by using the correct filters.

udp && udp.dstport == 12345						
No.	Time	Source	Destination	Protocol	Length	Info
92	25.496420	192.168.100.10	192.168.100.5	UDP	48	33319 → 12345 Len=6
96	30.154501	192.168.100.5	192.168.100.10	UDP	44	54664 → 12345 Len=2
159	38.536670	192.168.100.5	192.168.100.10	UDP	54	53075 → 12345 Len=12
262	94.102789	192.168.100.10	192.168.100.5	UDP	52	39984 → 12345 Len=10
297	108.130992	192.168.100.10	192.168.100.5	UDP	70	34133 → 12345 Len=28
401	119.735084	192.168.100.5	192.168.100.10	UDP	49	63029 → 12345 Len=7

Figure 2: Packet Stream Analysis for Chat

By clicking on an individual package we can further inspect it and reveal its **Network/Internet headers** and its **payload**.

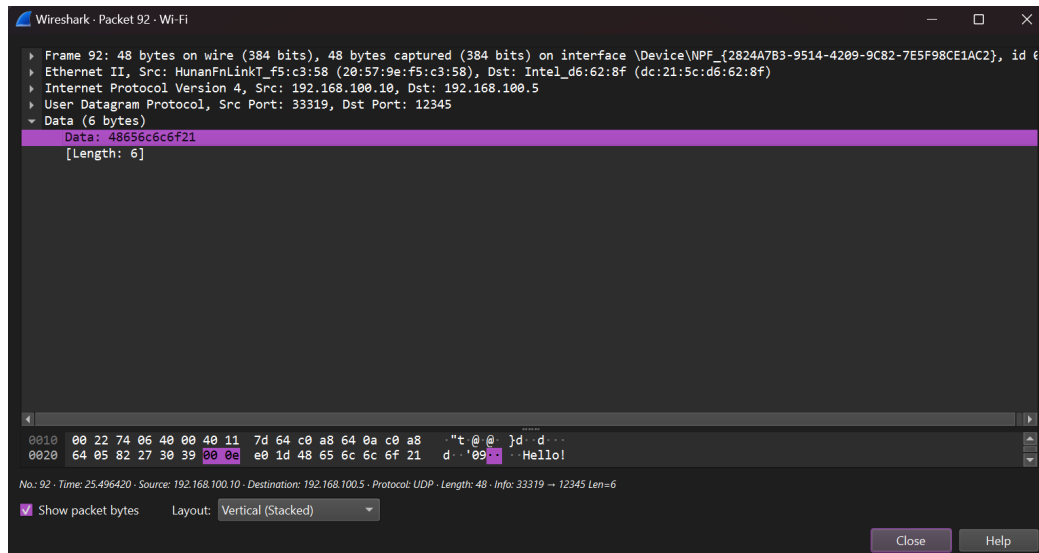


Figure 3: Individual Text Package

### 3 VoIP Implementation

The VoIP functionality allows *real-time* voice communication. This is achieved using the **PCM audio format** with an 8000 Hz sample rate, 8-bit sample size, and a monophonic channel. Two threads manage audio transmission and reception using UDP on **port 12346**.

#### 3.1 Starting a Call

The following function starts the call by creating *separate threads* for audio sending and receiving:

```

1 private void handleCallButton() {
2     callActive = true;
3
4     Thread voiceThread = new Thread(() -> {
5         final TargetDataLine[] microphone = new
6             TargetDataLine[1];
7         final SourceDataLine[] speaker = new
8             SourceDataLine[1];
9         try {

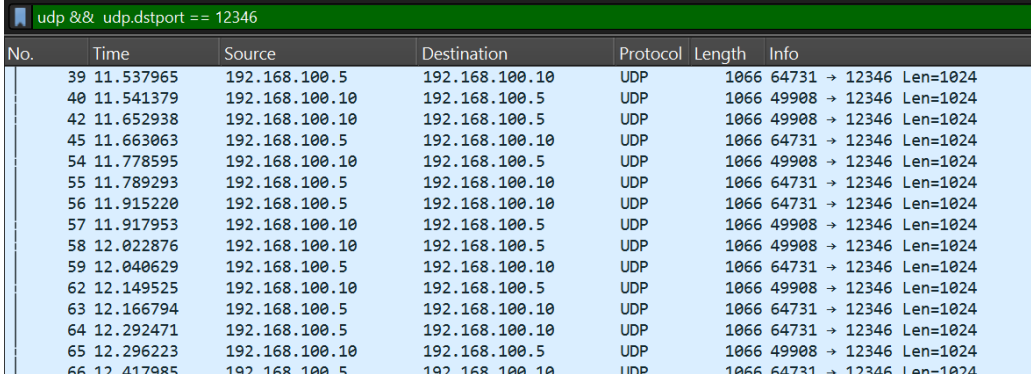
```

```
8      AudioFormat format = new AudioFormat
          (8000, 8, 1, true, true); // PCM
          format
9      DataLine.Info micInfo = new DataLine.
          Info(TargetDataLine.class, format);
10     DataLine.Info speakerInfo = new DataLine
          .Info(SourceDataLine.class, format);
11
12     microphone[0] = (TargetDataLine)
          AudioSystem.getLine(micInfo);
13     speaker[0] = (SourceDataLine)
          AudioSystem.getLine(speakerInfo);
14
15     microphone[0].open(format);
16     speaker[0].open(format);
17
18     microphone[0].start();
19     speaker[0].start();
20
21     sendSocket = new DatagramSocket();
22     receiveSocket = new DatagramSocket
          (12346);
23
24     InetAddress receiverAddress =
          InetAddress.getByName("RECEIVER_IP");
          // Replace with actual receiver IP
25     int port = 12346; // VoIP port
26     byte[] buffer = new byte[1024];
27
28     // Thread to capture and send audio
29     Thread sendThread = new Thread(() -> {
30         try {
31             while (callActive) {
32                 int bytesRead = microphone
                    [0].read(buffer, 0,
                    buffer.length);
33                 if (bytesRead > 0) {
34                     DatagramPacket packet =
                        new DatagramPacket(
                            buffer, bytesRead,
                            receiverAddress, port
                        );
35                     sendSocket.send(packet);
```

```
36         }
37     }
38     } catch (Exception ex) {
39         ex.printStackTrace();
40     } finally {
41         if (sendSocket != null && !
42             sendSocket.isClosed()) {
43             sendSocket.close();
44         }
45     }
46     sendThread.start();
47
48     // Continuously receive and play audio
49     while (callActive) {
50         DatagramPacket packet = new
51             DatagramPacket(buffer, buffer.
52                 length);
53         receiveSocket.receive(packet);
54         speaker[0].write(packet.getData(),
55             0, packet.getLength());
56     }
57     } catch (Exception ex) {
58         ex.printStackTrace();
59     } finally {
60         if (microphone[0] != null) {
61             microphone[0].close();
62         }
63         if (speaker[0] != null) {
64             speaker[0].close();
65         }
66     }
67 }
68
69 });
70 voiceThread.start();
71 }
```

Listing 3: VoIP Call Implementation

Once again, using **Wireshark** with the proper filters we can view the voice packet traffic on port 12346.



No.	Time	Source	Destination	Protocol	Length	Info
39	11.537965	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
40	11.541379	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
42	11.652938	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
45	11.663063	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
54	11.778595	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
55	11.789293	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
56	11.915220	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
57	11.917953	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
58	12.022876	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
59	12.040629	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
62	12.149525	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
63	12.166794	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
64	12.292471	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024
65	12.296223	192.168.100.10	192.168.100.5	UDP	1066	49908 → 12346 Len=1024
66	12.417985	192.168.100.5	192.168.100.10	UDP	1066	64731 → 12346 Len=1024

Figure 4: Packet Stream Analysis for VoIP

By clicking on an individual package we can further inspect it and reveal its **Network/Internet** headers and its **payload**.

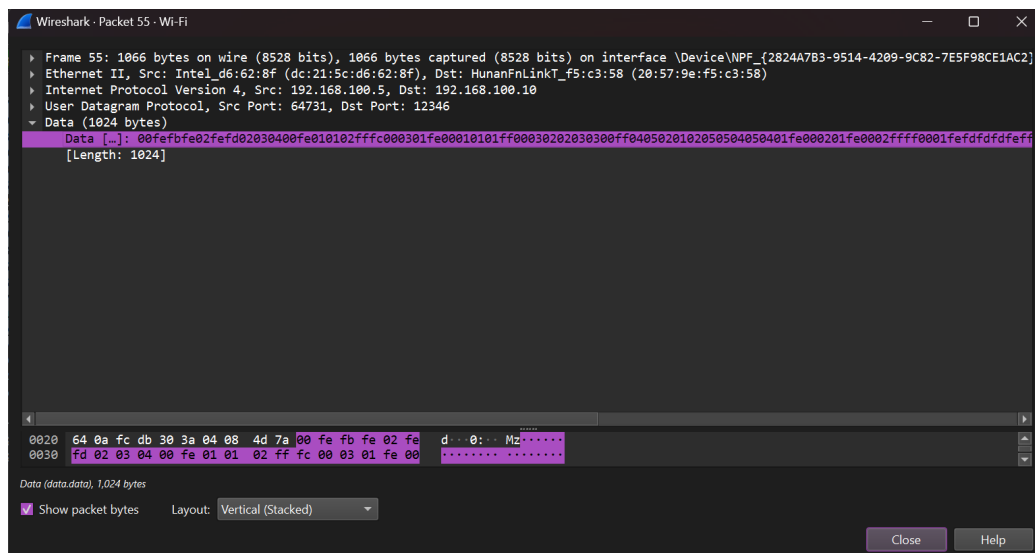


Figure 5: Individual Voice Package

## 4 Extras

### 4.1 Encryption

If the packet is not encrypted, its contents are visible to anyone (as observed using **Wireshark**). To improve security, AES encryption was implemented for the chat messages, ensuring confidentiality. However, hardcoding the encryption key poses a potential vulnerability. To further secure the chat,

the two users should securely exchange a shared key (using a method like the **Diffie-Hellman** method).

```

1 private String encryptMessage(String message) throws
   Exception {
2     Cipher cipher = Cipher.getInstance("AES");
3     cipher.init(Cipher.ENCRYPT_MODE, secretKey);
4     byte[] encryptedBytes = cipher.doFinal(message.
       getBytes());
5     return Base64.getEncoder().encodeToString(
       encryptedBytes);
6 }
7
8 private String decryptMessage(String
   encryptedMessage) throws Exception {
9     Cipher cipher = Cipher.getInstance("AES");
10    cipher.init(Cipher.DECRYPT_MODE, secretKey);
11    byte[] decryptedBytes = cipher.doFinal(Base64.
       getDecoder().decode(encryptedMessage));
12    return new String(decryptedBytes);
13 }

```

Listing 4: Encryption and Decryption

Here, we demonstrate the process of sending and receiving encrypted messaged using debug messages.

```

Received Encrypted Message: ZakMiteWcd0mU5bniq7Szw==
Starting decryption process...
Encrypted Message: ZakMiteWcd0mU5bniq7Szw==
Decoded Bytes: e?
??q?&S???
Decrypted Message: hi

```

Figure 6: Encrypted Chat

We see that now the packets appear encrypted in Wiresharks as well.

0020	64 05 a9 09 30 39 00 20 a4 a1 5a 61 6b 4d 69 74	d...09...ZakMit
0030	65 57 63 64 30 6d 55 35 62 6e 69 71 37 53 7a 77	ewcd0mU5 bniq7Szw

Figure 7: Encrypted Packet



## 5 Tools

- **Programming Language:** Java
- **IDE:** VSCode
- **Build Tool:** Maven
- **Version Control:** Git
- **Packet Analysis:** Wireshark