

# Δίκτυα υπολογιστών 2 — Chat App

Τζανέτης Σάββας, Ζωίδης Βασίλης

Δεκέμβριος 2024

# Περιεχόμενα

<b>1</b>	<b>Εισαγωγή</b>	<b>2</b>
<b>2</b>	<b>Εφαρμογές Κώδικα</b>	<b>3</b>
2.1	Κωδικοποίηση κειμένου . . . . .	3
2.2	Αποστολή και παραλαβή μηνυμάτων . . . . .	4
2.3	Φωνητική επικοινωνία . . . . .	6
<b>3</b>	<b>Εργαλεία που χρησιμοποιήθηκαν</b>	<b>13</b>

# Εισαγωγή

Αυτή η αναφορά αποτελεί μέρος μιας εργασίας για το μάθημα **Δίκτυα Υπολογιστών 2** του τμήματος **Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών** του **Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης**. Ο στόχος της εργασίας είναι η δημιουργία μιας **Chat** και **VoIP** εφαρμογής χρησιμοποιώντας τη γλώσσα προγραμματισμού **Java** και τις περιλαμβανόμενες βιβλιοθήκες της. Η επικοινωνία θα επιτευχθεί χρησιμοποιώντας το **πρωτόκολλο UDP** με τη βιβλιοθήκη **java.net**.

Ο κώδικας για αυτήν την εργασία βρίσκεται σε αυτήν τη σελίδα **GitHub**.

# Εφαρμογές Κώδικα

## 2.1 Κωδικοποίηση κειμένου

Αυτή η υλοποίηση χρησιμοποιεί τη βιβλιοθήκη **javax.crypto** προκειμένου να **κρυπτογραφήσει/αποκρυπτογραφήσει** τα μηνύματα κειμένου που αποστέλλονται μέσω της εφαρμογής. Αυτό είναι ένα πολύ σημαντικό βήμα, καθώς εάν τα πακέτα που περιέχουν τα μηνύματα κειμένου δεν είναι κρυπτογραφημένα, το περιεχόμενό τους θα είναι **ορατό** σε οποιονδήποτε είναι συνδεδεμένο αυτήν τη στιγμή σε οποιοδήποτε από τα δίκτυα αποστολέα ή παραλήπτη, κάτι που μπορεί να παρατηρηθεί χρησιμοποιώντας το **Wireshark** και αυτό θα δείξουμε αργότερα στην αναφορά.

Προκειμένου να κρυπτογραφηθούν και να αποκρυπτογραφηθούν τέτοια πακέτα, απαιτείται ένα κοινό μυστικό **κλειδί** μεταξύ των δύο συνδεδεμένων πελατών. Για τους σκοπούς αυτής της υλοποίησης χρησιμοποιείται ένα στατικό προηγουμένως συμφωνημένο κλειδί, αλλά συνιστάται ιδιαίτερα η ανταλλαγή μυστικών κλειδιών χρησιμοποιώντας μεθόδους όπως οι μέθοδοι **Diffie-Hellman** ή **Double Ratchet Algorithm**.

Ο παρακάτω κώδικας δείχνει τις συναρτήσεις που χειρίζονται τη διαδικασία κρυπτογράφησης και αποκρυπτογράφησης για τα μηνύματα κειμένου:

```
private static String encryptMessage(String message) {
    try {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encrypted = cipher.doFinal(message.getBytes());
        return Base64.getEncoder().encodeToString(encrypted);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

```
private static String decryptMessage(String message) {
    try {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decrypted = cipher.doFinal(
            Base64.getDecoder().decode(message));
        return new String(decrypted);
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Error Decrypting");
        return null;
    }
}
```

## 2.2 Αποστολή και παραλαβή μηνυμάτων

Τα μηνύματα κειμένου αποστέλλονται χρησιμοποιώντας το πρωτόκολλο UDP. Δημιουργείται ένα αποκλειστικό νήμα για να "ακούσει" εισερχόμενα μηνύματα από μια συγκεκριμένη IP. Η θύρα που χρησιμοποιείται για την αποστολή και λήψη αυτών των μηνυμάτων είναι το **Port 12345** όπως προκαθορίζεται στις οδηγίες. Παρακάτω, είναι το απόσπασμα που δείχνει την υλοποίηση για το νήμα ακρόασης που χρησιμοποιείται για τη λήψη τυχόν μηνυμάτων κειμένου:

```
new Thread(() -> {
    byte[] buffer = new byte[1024];
    try {
        DatagramSocket textSocket = new DatagramSocket(
            Integer.parseInt(chatPort))
        {
            while (true) {
                DatagramPacket packet = new DatagramPacket(
                    buffer, buffer.length);
                textSocket.receive(packet);
                if (packet.getAddress().getHostAddress().equals(destIp))
                {
                    String message = new String(
                        packet.getData(), 0,
                        packet.getLength());
                    textArea.append(
                        "Received: " + decryptMessage(message) + newline);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}).start();
```

Η παρακάτω εικόνα δείχνει ένα παράθυρο **Wireshark**, που επαληθεύει ότι τα πακέτα κειμένου αποστέλλονται στο **Port 12345**. Οι δύο IP που χρησιμοποιούνται είναι τοπικές IP, καθώς είναι μια τοπική σύνδεση μεταξύ δύο υπολογιστών. Ωστόσο, με το σωστό firewall και port forwarding, είναι επίσης δυνατή μια σύνδεση μεταξύ πολλαπλών δικτύων.

No.	Time	Source	Destination	Protocol	Length	Info
221	30.150800	192.168.1.16	192.168.1.5	UDP	130	57867 → 12345 Len=88
504	62.979555	192.168.1.5	192.168.1.16	UDP	106	59279 → 12345 Len=64
4502	247.400770	192.168.1.5	192.168.1.16	UDP	66	63436 → 12345 Len=24

Figure 2.1: Package streams for text messages

Το ακόλουθο απόσπασμα κώδικα εκτελείται μόνο όταν πατηθεί το κουμπί **"Send"** ή το πλήκτρο **"Enter"** από τον χρήστη. Ένα πακέτο με το μήνυμα αποστέλλεται στον παραλήπτη μόνο εάν η συμβολοσειρά μέσα στο πεδίο κειμένου δεν είναι κενή (δηλαδή μόνο εάν έχει γραφτεί ένα πραγματικό μήνυμα).

```
// The "Send" button was clicked
String message = inputTextField.getText();
if (!message.trim().isEmpty()) {
    try {
        DatagramSocket textSocket = new DatagramSocket();
        InetAddress address = InetAddress.getByName(destIp);
        int port = Integer.parseInt(chatPort);
        byte[] buffer = encryptMessage(message).getBytes();
        DatagramPacket packet = new DatagramPacket(
            buffer, buffer.length,
            address, port);

        // Print the message locally
        textArea.append("Me: " + message + newline);
        inputTextField.setText("");

        //Send the message to the target
        textSocket.send(packet);
        textSocket.close();
        System.out.println("Message sent");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Η επόμενη εικόνα δείχνει τα περιεχόμενα ενός μεμονωμένου πακέτου, τόσο σε δεκαεξαδική μορφή όσο και σε μορφή κειμένου. Αποκαλύπτοντας έτσι την κεφαλίδα του πακέτου (**Network header**), καθώς και το κρυπτογραφημένο ωφέλιμο φορτίο του (**payload**). Εάν αποκρυπτογραφήσαμε τη συμβολοσειρά που εμφανίζεται χρησιμοποιώντας το κλειδί κρυπτογράφησης που χρησιμοποιήθηκε ("abcdefghigklmnop") με βάση το **Advanced Encryption Standard**, το τελικό μήνυμα θα ήταν **'Hello World'**.

```

> Frame 4502: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{AD8D424C-FB46-4A37-B9AC-0A331360BE16}, id 0
> Ethernet II, Src: ASUSTekCOMPU_16:95:df (fc:34:97:16:95:df), Dst: Intel_29:6f:62 (84:1b:77:29:6f:62)
> Internet Protocol Version 4, Src: 192.168.1.5, Dst: 192.168.1.16
> User Datagram Protocol, Src Port: 63436, Dst Port: 12345
▼ Data (24 bytes)
  Data: 784a376363735a675157727763534e324c55716a2b513d3d
  [Length: 24]

0000  84 1b 77 29 6f 62 fc 34  97 16 95 df 08 00 45 00  ..w)ob 4 .....E
0010  00 34 80 74 00 00 80 11  00 00 c0 a8 01 05 c0 a8  -4 t .....
0020  01 10 f7 cc 30 39 00 20  83 97 78 4a 37 63 63 73  ...09...xJ7ccs
0030  5a 67 51 57 72 77 63 53  4e 32 4c 55 71 6a 2b 51  ZgQWrwcs N2LUqj+Q
0040  3d 3d  ..==

```

Figure 2.2: Individual Text Package

## 2.3 Φωνητική επικοινωνία

Αυτή η εφαρμογή java, εκτός από την αποστολή μηνυμάτων κειμένου, έχει και δυνατότητα φωνητικής επικοινωνίας σε πραγματικό χρόνο (**VoIP**). Αυτό επιτυγχάνεται επίσης χρησιμοποιώντας το **Πρωτόκολλο UDP** και πιο συγκεκριμένα χρησιμοποιώντας ρυθμό δειγματοληψίας **44100 Hz**, με μέγεθος δειγματοληψίας **16 bit** σε **στερεοφωνικό** κανάλι .

Η διαδικασία της φωνητικής επικοινωνίας είναι λίγο πιο περίπλοκη σε σύγκριση με την υλοποίηση μηνυμάτων κειμένου. Πρώτον, δημιουργείται ένα ξεχωριστό νήμα για το χειρισμό των αιτημάτων κλήσεων σε ένα ξεχωριστό **requests port**, "**12347**". Αυτό γίνεται για να ειδοποιηθεί ο παραλήπτης για το αίτημα κλήσης και να του δοθεί η επιλογή είτε να **απορρίψει** είτε να **αποδεχτεί** την κλήση. Εάν η κλήση απορριφθεί, χρησιμοποιώντας το ίδιο port, αποστέλλεται ένα μήνυμα απόρριψης στον χρήστη που ξεκίνησε την κλήση, ειδοποιώντας τον ότι το αίτημα κλήσης απορρίφθηκε.

Τα αποσπάσματα κώδικα για το request-listening thread και τη λειτουργία αποστολής αυτών των μηνυμάτων αιτήματος κλήσης φαίνονται παρακάτω:

```

new Thread(() -> {
    byte[] buffer = new byte[1024];
    try (DatagramSocket requestsSocket =
        new DatagramSocket(
            Integer.parseInt(requestsPort)))
    {
        while (true) {
            DatagramPacket packet = new DatagramPacket(
                buffer, buffer.length);
            requestsSocket.receive(packet);
            if (packet.getAddress().getHostAddress().equals(destIp))
            {
                String message = new String(
                    packet.getData(), 0,
                    packet.getLength());
                handleCallMessages(message);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}).start();

```

```

private static void sendCallMessages(String message) {
    try {
        DatagramSocket socket = new DatagramSocket();
        InetAddress address = InetAddress.getByName(destIp);
        byte[] buffer = message.getBytes();
        DatagramPacket packet = new DatagramPacket(
            buffer, buffer.length, address,
            Integer.parseInt(requestsPort));

        socket.send(packet);
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```



Συνεπώς, απαιτείται και μια **συνάρτηση για τον χειρισμό όλων των αιτημάτων κλήσης** (για παράδειγμα "CALL\_REJECTED") έτσι ώστε, να πραγματοποιούνται οι σωστές λειτουργίες ανάλογα με τις ενέργειες των χρηστών. Για παράδειγμα, εάν η κλήση τερματιστεί από οποιονδήποτε από τους χρήστες, ένα μήνυμα "CALL\_END" θα σταλεί στον άλλο χρήστη, με αποτέλεσμα να ληφθεί από αυτή τη συνάρτηση και να εκτελέσει την **endCall()**. Η **endCall()** με την σειρά της κλείνει τυχόν χρησιμοποιημένα sockets ηχείων και μικροφώνου προκειμένου να αποτραπούν οι διαρροές δεδομένων. Στην αντίθετη περίπτωση, που η κλήση έχει γίνει αποδεκτή ("CALL\_ACCEPTED"), και οι δύο χρήστες θα εκτελέσουν τις **startAudioCommunication()** για την αποστολή των δεδομένων ήχου από το μικρόφωνο και **startAudioReception()** για την ακρόαση πακέτων σε ξεχωριστό νήμα και εξαγωγή του περιεχομένου τους σε ένα ηχείο.

Τα αποσπάσματα για όλες αυτές τις λειτουργίες φαίνονται παρακάτω:

```
private static void handleCallMessages(String message) {
    switch(message) {
        case "CALL_REQUEST":
            int response = JOptionPane.showConfirmDialog(
                null, "Do you want to accept the call?",
                "Incoming call", JOptionPane.YES_NO_OPTION);
            if (response == JOptionPane.YES_OPTION) {
                sendCallMessages("CALL_ACCEPTED");
                startAudioCommunication();
                startAudioReception();
                callInProgress = true;
                callButton.setText("End");
            } else
                sendCallMessages("CALL_REJECTED");
            break;
        case "CALL_ACCEPTED":
            startAudioCommunication();
            startAudioReception();
            callInProgress = true;
            callButton.setText("End");
            break;
        case "CALL_REJECTED":
            textArea.append("Call rejected by the recipient" + newline);
            break;
        case "CALL_END":
            endCall();
            textArea.append("Call ended by the other user" + newline);
            break;
        default:
            break;
    }
}
```

```

private static void endCall() {
    callInProgress = false;
    if (voiceSenderSocket != null && !voiceSenderSocket.isClosed()) {
        voiceSenderSocket.close();
    }
    if (voiceReceiverSocket != null && !voiceReceiverSocket.isClosed()) {
        voiceReceiverSocket.close();
    }
    if (microphone != null && microphone.isOpen()) {
        microphone.close();
    }
    if (speaker != null && speaker.isOpen()) {
        speaker.close();
    }
    callButton.setText("Call");
}

```

```

public static void startAudioCommunication() {
    new Thread(() -> {
        try {
            // Start capturing and sending audio data
            AudioFormat format = new AudioFormat(44100, 16, 2,
                                                    true, true);
            DataLine.Info info = new DataLine.Info(
                                    TargetDataLine.class, format);
            microphone = (TargetDataLine) AudioSystem.getLine(info);
            microphone.open(format);
            microphone.start();

            byte[] buffer = new byte[1024];
            voiceSenderSocket = new DatagramSocket();
            InetAddress address = InetAddress.getByName(destIp);

            while (callInProgress) {
                int bytesRead = microphone.read(buffer, 0,
                                                  buffer.length);
                DatagramPacket packet = new
                    DatagramPacket(buffer, bytesRead,
                                    address,
                                    Integer.parseInt(
                                        voicePort));
                voiceSenderSocket.send(packet);
            }
        } catch (IOException | LineUnavailableException ex) {
            ex.printStackTrace();
        } finally {
            if (voiceSenderSocket != null && !voiceSenderSocket.isClosed()) {
                voiceSenderSocket.close();
            }
            if (microphone != null && microphone.isOpen()) {
                microphone.close();
            }
        }
    }).start();
}

```

```

public static void startAudioReception() {
    new Thread(() -> {
        try {
            // Start receiving and playing audio data
            AudioFormat format = new AudioFormat(44100, 16, 2,
                                                true, true);

            DataLine.Info info = new DataLine.Info(
                                SourceDataLine.class, format);
            speaker = (SourceDataLine) AudioSystem.getLine(info);
            speaker.open(format);
            speaker.start();

            voiceReceiverSocket = new DatagramSocket(Integer.parseInt(
                                                voicePort));

            byte[] buffer = new byte[1024];

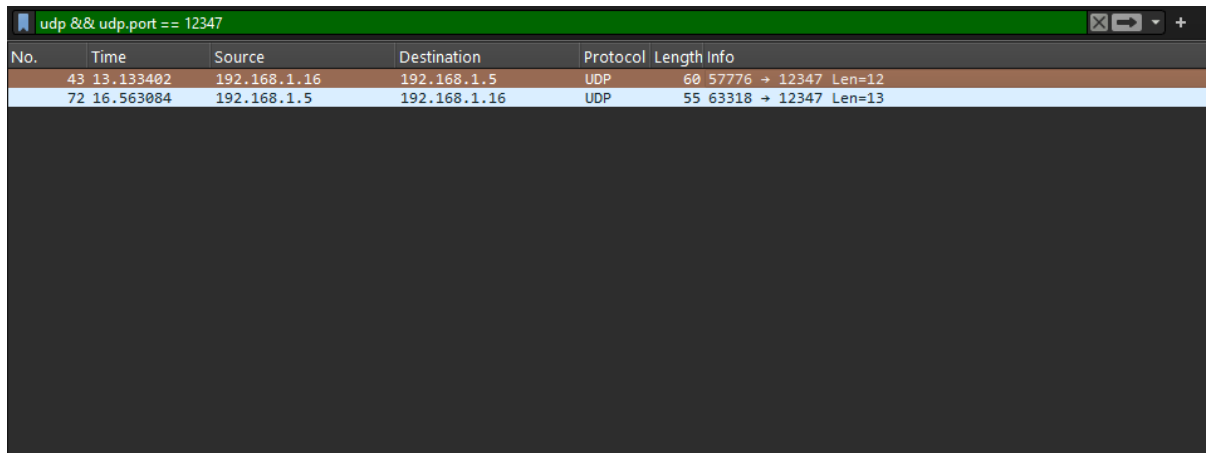
            while (callInProgress) {
                DatagramPacket packet=new DatagramPacket(buffer,
                                                            buffer.length);

                voiceReceiverSocket.receive(packet);
                speaker.write(packet.getData(), 0, packet.getLength());
            }
        } catch (IOException | LineUnavailableException ex) {
            ex.printStackTrace();
        } finally {
            if (voiceReceiverSocket!=null &&
                !voiceReceiverSocket.isClosed())
            {
                voiceReceiverSocket.close();
            }
            if (speaker != null && speaker.isOpen()) {
                speaker.close();
            }
        }
    }).start();
}

```



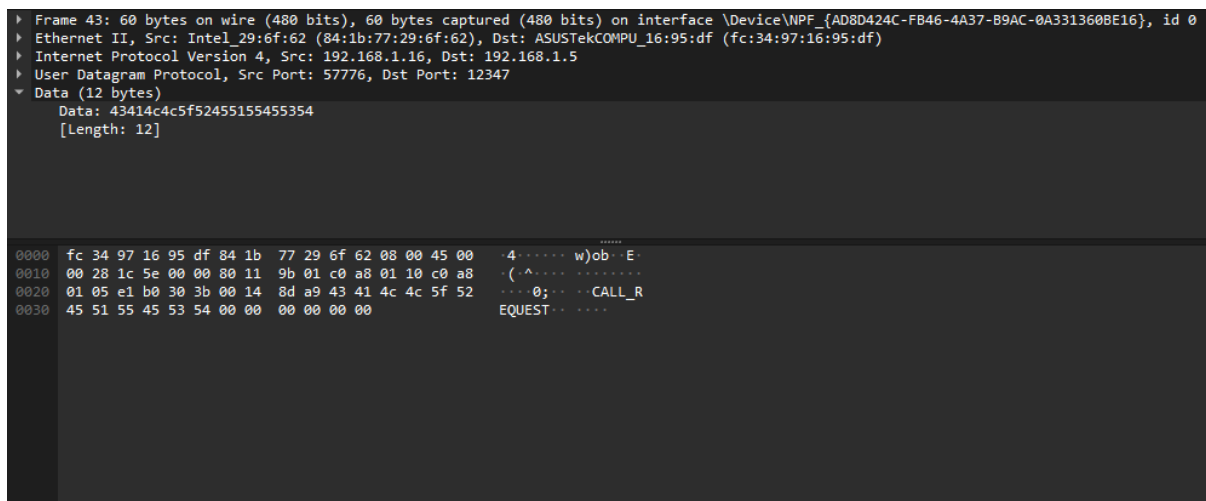
Μπορούμε επίσης να δούμε τα αιτήματα που αποστέλλονται στη θύρα **port 12347** που χειρίζεται αιτήματα κλήσεων, απορρίψεις και τερματισμούς όπως εξηγήσαμε παραπάνω:



The image shows a Wireshark packet list with the filter 'udp && udp.port == 12347'. It displays two UDP packets. The first packet (No. 43) is from 192.168.1.16 to 192.168.1.5, length 60, with info '57776 → 12347 Len=12'. The second packet (No. 72) is from 192.168.1.5 to 192.168.1.16, length 55, with info '63318 → 12347 Len=13'.

No.	Time	Source	Destination	Protocol	Length	Info
43	13.133402	192.168.1.16	192.168.1.5	UDP	60	57776 → 12347 Len=12
72	16.563084	192.168.1.5	192.168.1.16	UDP	55	63318 → 12347 Len=13

Figure 2.5: Requests Packages List



The image shows the details and hex dump for packet 43. The details pane shows: Frame 43: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF\_{AD8D424C-FB46-4A37-B9AC-0A331360BE16}, id 0; Ethernet II, Src: Intel\_29:6f:62 (84:1b:77:29:6f:62), Dst: ASUSTekCOMPU\_16:95:df (fc:34:97:16:95:df); Internet Protocol Version 4, Src: 192.168.1.16, Dst: 192.168.1.5; User Datagram Protocol, Src Port: 57776, Dst Port: 12347; Data (12 bytes): 43414c4c5f52455155455354 [Length: 12]. The hex dump shows the raw bytes of the packet, including the Ethernet II header, IP header, and UDP header.

```

Frame 43: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{AD8D424C-FB46-4A37-B9AC-0A331360BE16}, id 0
Ethernet II, Src: Intel_29:6f:62 (84:1b:77:29:6f:62), Dst: ASUSTekCOMPU_16:95:df (fc:34:97:16:95:df)
Internet Protocol Version 4, Src: 192.168.1.16, Dst: 192.168.1.5
User Datagram Protocol, Src Port: 57776, Dst Port: 12347
Data (12 bytes)
  Data: 43414c4c5f52455155455354
  [Length: 12]

0000  fc 34 97 16 95 df 84 1b 77 29 6f 62 08 00 45 00  4-----w)ob--E
0010  00 28 1c 5e 00 00 80 11 9b 01 c0 a8 01 10 c0 a8  (-^-----
0020  01 05 e1 b0 30 3b 00 14 8d a9 43 41 4c 4c 5f 52  ....0;...CALL_R
0030  45 51 55 45 53 54 00 00 00 00 00 00 00 00 00 00  EQUEST.....

```

Figure 2.6: Individual Request Packet

# Εργαλεία που χρησιμοποιήθηκαν

Τα εργαλεία που χρησιμοποιήθηκαν για αυτήν την υλοποίηση είναι:

- Η γλώσσα προγραμματισμού **Java**.
- Η βιβλιοθήκη **java.net** για networking, καθώς και η βιβλιοθήκη **javax.sound** για λήψη και αναπαραγωγή ήχου.
- Η βιβλιοθήκη **javax.crypto** για κρυπτογράφηση και αποκρυπτογράφηση των πακέτων.
- **Maven** για build automation και project management.
- **GitHub** για version control.