ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ ΙΙ

Υποχρεωτική Εργασία

Αραμπατζής Κωνσταντίνος 10809
Παπαδόπουλος Παναγιώτης 10697

Περιγραφή του κώδικα:

Στο αρχείο App.java το οποίο δόθηκε και ζητήθηκε να τροποποιηθεί. Για την δήλωση global μεταβλητών:

```
// TODO: Please define and initialize your variables here...

    // Declare the receiver's IP and port as class-level variables
    static String receiverAddressString = "ipAddress";  // change with correct address
    static String ownAddressString = "ipAddress";        // change with correct address
    static int receiverPort = 12346;                    // change with correct port number
    static int ownPort = 12345;                         // change with correct port number
    static public boolean outgoingCall = false;
    static public boolean incomingCall = false;
    static boolean activeCall = false;
    static boolean activeTest = false;


    // Local Recorder and Player initialization for microphone testing
    private static AudioRecorder audioRecorderLocal;
    private static AudioPlayer audioPlayerLocal;
```

Εδώ ορίζονται οι διευθύνσεις IP και οι αριθμοί θυρών του παραλήπτη και του αποστολέα. Επίσης γίνεται αρχικοποίηση των τεσσάρων boolean μεταβλητών που καθορίζουν την κατάσταση της εφαρμογής. Η κατάσταση που μπορεί να βρίσκεται η εφαρμογή ορίζεται ως ο συνδυασμός των παραπάνω μεταβλητών που αντιστοιχούν σε:

- outgoingCall: εξερχόμενη κλήση
- incomingCall: εισερχόμενη κλήση
- activeCall: ενεργή κλήση
- activeTest: ενεργή λειτουργία ελέγχου του μικροφώνου

Τέλος, δηλώνονται οι μεταβλητές για τον έλεγχο του μικροφώνου, για την καταγραφή και αναπαραγωγή του ήχου. Για την λειτουργία αυτή προστέθηκε νέο κουμπί αντίστοιχο με τα υπάρχοντα.

```
testMicButton = new JButton("Test mic - OFF");
.
.
.
add(testMicButton);
.
.
.
testMicButton.addActionListener(this);
```

Η επικοινωνία μεταξύ των χρηστών χωρίστηκε σε 3 είδη:

- Μηνύματα: μηνύματα τύπου κειμένου

- Σύνδεση: μηνύματα τύπου κειμένου για να ειδοποιήσουν τον χρήστη για την κατάσταση της ηχητικής κλήσης
- Ηχητικό μήνυμα: μηνύματα τύπου ήχου

Δημιουργήθηκε μια κλάση Sender για την αποστολή των μηνυμάτων, η οποία στέλνει ένα πακέτο με την κεφαλίδα και ένα επόμενο με το περιεχόμενο του μηνύματος. Ακολουθεί κώδικας με την υλοποίηση της αποστολής απλού μηνύματος. Αντίστοιχα λειτουργούν οι άλλες δυο περιπτώσεις.

```
public void sendMessage(String message) {
    try {
       // Convert the message to a byte array
        byte[] messageBytes = message.getBytes();
         // Create the header: "MSG " + 4-digit length
        String header = "MSG " + String.format("%04d", messageBytes.length);
        byte[] headerBytes = header.getBytes();

        // Send the header first
        DatagramPacket headerPacket = new DatagramPacket(headerBytes, headerBytes.length,
receiverAddress, receiverPort);
        socket.send(headerPacket);

        // Send the actual message data
        DatagramPacket messagePacket = new DatagramPacket(messageBytes, messageBytes.length,
receiverAddress, receiverPort);
        socket.send(messagePacket);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Προφανώς δημιουργήθηκε κλάση Receiver για να δέχεται τα μηνύματα και να καταλαβαίνει τις custom κεφαλίδες. Ακολουθεί κομμάτι της κλάσης που είναι υπεύθυνο για την λήψη και ερμήνευση των εισερχόμενων πακέτων.

```
public ReceivedData receiveData() throws Exception {
    byte[] header = new byte[8]; // Fixed header size (4 bytes for type and 4 bytes for length)
    DatagramPacket packet = new DatagramPacket(header, header.length);
    socket.receive(packet);

    // Read and parse the header
    String type = new String(header, 0, 4); // "MSG " or "CALL" or "CONN"
    int length = Integer.parseInt(new String(header, 4, 4).trim()); // Length of the data
    // Receive the payload data
    byte[] payload = new byte[length];
    packet = new DatagramPacket(payload, payload.length);
    socket.receive(packet);

    // Return the data based on the type
    if (type.equals("MSG ")) {
        String message = new String(payload);
        return new ReceivedData("message", message, null);
    } else if (type.equals("CALL")) {
        return new ReceivedData("audio", null, payload);
    } else if (type.equals("CONN")) {
        String message = new String(payload);
        return new ReceivedData("connection", message, null);
    } else {
        throw new Exception("Unknown data type: " + type);
    }
}
```

Στην main συνάρτηση του αρχικού αρχείου δηλώνονται άλλες τοπικές μεταβλητές που είναι υπεύθυνες για την κίνηση των πακέτων και την καταγραφή και αναπαραγωγή ήχου.

```
// Create a DatagramSocket to listen on the specified port
DatagramSocket socket = new DatagramSocket(ownPort);

// Create the receiver
Receiver receiver = new Receiver(socket);

// Create the sender
Sender sender = new Sender(socket, InetAddress.getByName(receiverAddressString), receiverPort);

// Audio format configuration
```

```
AudioFormat audioFormat = new AudioFormat(16000, 16, 1, true, false);

// Create and start audio sender and receiver
AudioRecorder audioRecorderCall = new AudioRecorder(sender, audioFormat, activeCall, false);
AudioPlayer audioPlayerCall = new AudioPlayer(audioFormat, activeCall, false);

// Start threads for sending and receiving audio
new Thread(audioRecorderCall).start();
new Thread(audioPlayerCall).start();

// Microphone check
audioRecorderLocal = new AudioRecorder(audioFormat, activeTest);
new Thread(audioRecorderLocal).start();

audioPlayerLocal = new AudioPlayer(audioFormat, activeTest, true);
new Thread(audioPlayerLocal).start();
```

Στην ήδη υπάρχουσα while loop η οποία “ακούει” συνεχώς για εισερχόμενα πακέτα, με βάση την κεφαλίδα ερμηνεύεται αντίστοιχα το μήνυμα.

```
if (data.getType().equals("message")) {
    // Handle received message
    textArea.append(receiverPort + ": " + data.getMessage() + newline);

} else if (data.getType().equals("connection")) {
    switch (data.getMessage()) {
    case "call":
        incomingCall = true;
        textArea.append(receiverPort + " started a call. " + newline);
        break;
    case "answer":
        incomingCall = false;
        outgoingCall = false;
        activeCall = true;
        textArea.append(receiverPort + " accepted the call. " + newline);
        break;
    case "end-call":
        incomingCall = false;
        outgoingCall = false;
        activeCall = false;
        textArea.append(receiverPort + " ended the call. " + newline);
        break;
    default:
        textArea.append(receiverPort + ": " + data.getMessage() + newline);
        break;
    }

} else if (data.getType().equals("audio")) {
    // Handle received audio
    byte[] audioData = data.getAudio();
    audioRecorderCall.setActive(activeCall);
    audioPlayerCall.setAudioData(audioData);
}
```

Όταν πατηθεί το κουμπί αποστολής μηνύματος δημιουργείται ένα socket το οποίο χρησιμοποιεί η κλάση Sender για να προσθέσει την κεφαλίδα και εμφανίζεται και το μήνυμα στην μεριά του αποστολέα. Παρακάτω φαίνεται ο κώδικας:

```
if (!inputTextField.getText().isEmpty()) {
    try {
        DatagramSocket socket = new DatagramSocket(); // Creates a socket for sending
        Sender sender = new Sender(socket, InetAddress.getByName(receiverAddressString),
receiverPort);

        String message = inputTextField.getText();
        sender.sendMessage(message);
        textArea.append("You: " + message + newline);
        socket.close(); // Close the socket
        inputTextField.setText("");
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}
```

Αντίστοιχα όταν πατηθεί το κουμπί κλήσης:

```
else if (e.getSource() == callButton) {
```

```
    try {
        DatagramSocket socket = new DatagramSocket(); // Creates a socket for sending
        Sender sender = new Sender(socket, InetAddress.getByName(receiverAddressString),
receiverPort);

        // First outgoing Call
        if (!outgoingCall && !incomingCall && !activeCall) {
            // Start the client for initiating a call
            sender.sendConnectionMessage("call");

            // Locally handle outgoing call
            outgoingCall = true;
            textArea.append("You started a call." + newline);

        // Answer incoming call
        } else if (incomingCall) {
            // Send the appropriate message on the other end
            sender.sendConnectionMessage("answer");

            // Locally handle incoming call
            incomingCall = false;
            activeCall = true;
            textArea.append("You accepted the call." + newline);

        // End Call
        } else if (activeCall) {
            // Send the appropriate message on the other end
            sender.sendConnectionMessage("end-call");

            // Locally handle cancelling the call
            activeCall = false;
            outgoingCall = false;
            incomingCall = false;
            textArea.append("You ended the call." + newline);
        }
        socket.close(); // Close the socket

    } catch (Exception exception) {
        exception.printStackTrace();
    }
}
```

Και τέλος όταν πατηθεί το κουμπί ελέγχου του μικροφώνου που καταγράφει τοπικά τη φωνή σε έναν Buffer και τον αναπαράγει:

```
else if (e.getSource() == testMicButton) {
    // Open/close microphone
    activeTest = !activeTest;
    System.out.println("Mic status: " + activeTest);

    audioRecorderLocal.setActive(activeTest);
    audioPlayerLocal.setActive(activeTest);

    if(activeTest) {
        textArea.append("Microphone Active" + newline);
    } else {
        textArea.append("Microphone Inactive" + newline);
    }
}
```

Μετά από κάθε λήψη εισερχόμενης δυάδας πακέτων εκτελείται μία συνάρτηση η οποία ανανεώνει την κατάσταση της εφαρμογής όσον αφορά τις ηχητικές κλήσεις.

```
public static void buttonLogic(){

    if(outgoingCall){
        callButton.setText("Waiting...");
        callButton.setEnabled(false);
        testMicButton.setEnabled(false);
    }

    if (incomingCall) {
        callButton.setText("Accept Call");
    }

    if(activeCall){
        callButton.setText("End Call");
        callButton.setEnabled(true);
```

```
        testMicButton.setEnabled(false);
    }

    if (!incomingCall && !outgoingCall && !activeCall) {
        callButton.setText("Call");
        callButton.setEnabled(true);
        testMicButton.setEnabled(true);
    }

    // if mic test is active, call button does not work
    if (activeTest) {
        testMicButton.setText("Test mic - ON");
        callButton.setEnabled(false);
    }

    // If mic test is inactive, call button works
    if(!activeTest) {
        testMicButton.setText("Test mic - OFF");
    }
}
```

Άλλες κλάσεις που υλοποιήθηκαν είναι οι AudioRecorder και AudioPlayer για την καταγραφή και αναπαραγωγή ήχου αντίστοιχα, καθώς και ο Buffer που αναφέρθηκε προηγουμένως. Ο πλήρες κώδικας φαίνεται στο συμπιεσμένο .zip αρχείο.

Η εφαρμογή εκτελείται με την παρακάτω εντολή σε λειτουργικό σύστημα Windows:

```
cd C:\your-local-path\src\main\java && javac
com\cn2\communication\*.java && java com.cn2.communication.App
```

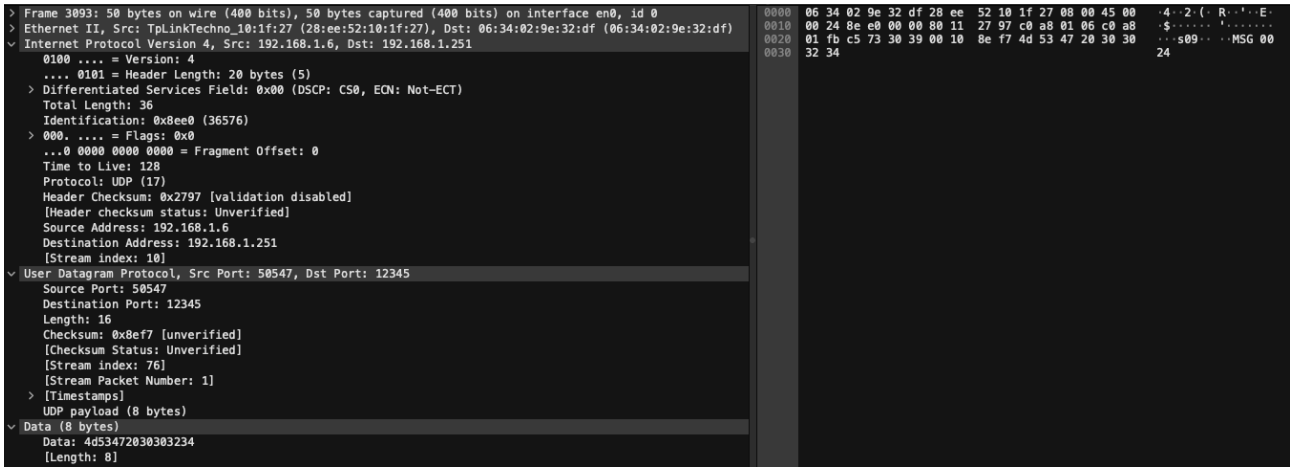
ή την παρακάτω εντολή σε λειτουργικό σύστημα Unix:

```
cd /Users/panagiwths/Desktop/assignments/computer-networks/src/main/java
&& javac com/cn2/communication/*.java && java com.cn2.communication.App
```

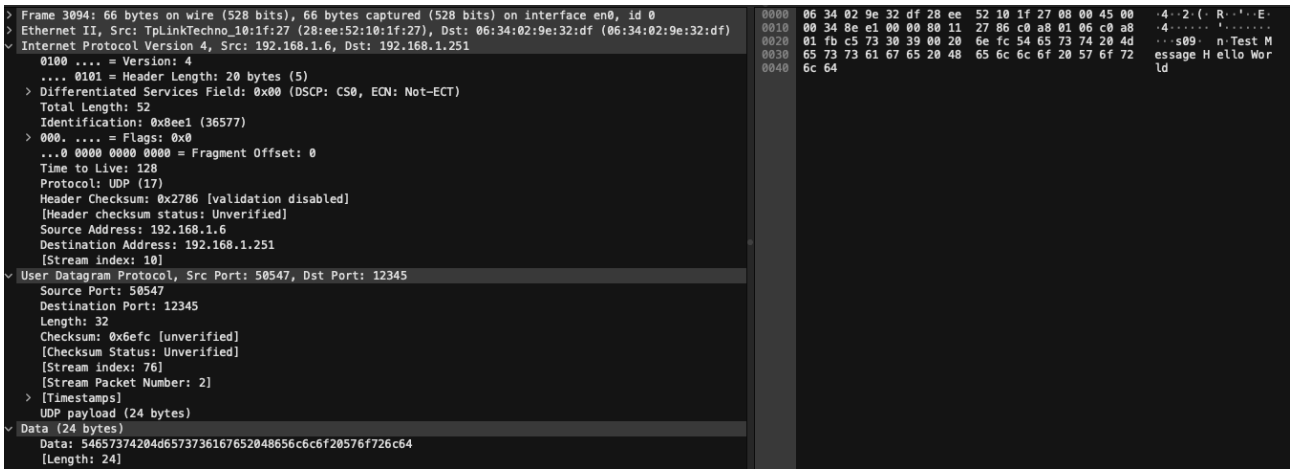
Ανταλλαγή Μηνυμάτων Κειμένου

Η σύνδεση των υπολογιστών έγινε τοπικά στις διευθύνσεις IP που φαίνονται στις ακολουθούμενες εικόνες και στις θύρες με αριθμό 12345 στον εκάστοτε υπολογιστή.

Όπως αναφέρθηκε παραπάνω, για κάθε μήνυμα αποστέλλονται δύο πακέτα, ένα με το είδος μηνύματος (πακέτο κεφαλίδας) και ένα με με το περιεχόμενο του μηνύματος. Παρατίθενται στιγμιότυπα από το Wireshark που αναδεικνύουν την ιδιαιτερότητα αυτή.



Πακέτο κεφαλίδας



Πακέτο μηνύματος

Το payload στο Wireshark συνοδεύεται από headers και μετα-δεδομένα, τα οποία μπορεί να εμφανίζονται ως μη έγκυροι χαρακτήρες όταν δεν αποκωδικοποιούνται σωστά. Αυτό συμβαίνει επειδή το ωφέλιμο φορτίο δεδομένων (payload) δεν είναι πάντα καθαρά αναγνώσιμο κείμενο, αλλά περιλαμβάνει πληροφορίες απαραίτητες για τη διαχείριση της επικοινωνίας.

Τα headers του πρωτοκόλλου, όπως αυτά του IP ή του UDP, περιέχουν δεδομένα που δεν είναι συμβατά με τις συνηθισμένες μορφές κωδικοποίησης κειμένου (π.χ., ASCII ή UTF-8). Αυτά τα δεδομένα, όταν εμφανίζονται σε μορφή raw binary, μεταφράζονται σε άκυρους χαρακτήρες.

Επιπλέον, τα bytes συμπλήρωσης (padding) που χρησιμοποιούνται για να φτάσουν τα πακέτα στο απαιτούμενο μήκος μπορούν επίσης να δημιουργήσουν την ψευδαίσθηση άκυρων χαρακτήρων. Τέλος, η ασυμβατότητα ανάμεσα στο format του payload και τον τρόπο αποκωδικοποίησης του Wireshark μπορεί να ενισχύσει το φαινόμενο.

Για την ανάλυση αυτών των χαρακτήρων, απαιτείται προσαρμογή της διαδικασίας αποκωδικοποίησης ή εξαγωγή του payload σε εξωτερικά εργαλεία που μπορούν να απομονώσουν και να αναλύσουν το περιεχόμενο σε διαφορετικά επίπεδα.

Ανταλλαγή Μηνυμάτων Ήχου

Fragmentation - Κατακερματισμός

Πραγματοποιήθηκε δοκιμή με την αποστολή πακέτων μεγέθους 2048 bytes μεταξύ δύο συσκευών, όπου παρατηρήθηκε κατακερματισμός για πακέτα που υπερέβαιναν τα 1480 bytes. Αυτό το φαινόμενο προκύπτει διότι το μέγεθος των πακέτων ξεπερνά το καθορισμένο Maximum Transmission Unit (MTU), το οποίο για το συγκεκριμένο δίκτυο έχει οριστεί στα 1480 bytes. Ως αποτέλεσμα, το πακέτο διασπάται σε μικρότερα τμήματα στο επίπεδο του πρωτοκόλλου IP, σύμφωνα με τους μηχανισμούς κατακερματισμού.

Κατά την ανάλυση στο Wireshark, παρατηρήθηκε ότι το πεδίο Fragment Offset αυξάνεται κατά 1480 bytes για κάθε νέο τμήμα, αποτυπώνοντας τη θέση του τμήματος στο αρχικό πακέτο. Επιπλέον, το πεδίο More Fragments (MF) ήταν ενεργοποιημένο στο πρώτο τμήμα και όχι στο τελευταίο, υποδεικνύοντας ότι υπάρχουν επιπλέον τμήματα που ανήκουν στο ίδιο πακέτο.

Η διαδικασία επανασύνθεσης των τμημάτων ολοκληρώθηκε επιτυχώς, επιτρέποντας την ανακατασκευή του αρχικού πακέτου στο επίπεδο εφαρμογής. Παρόλα αυτά, η διαδικασία κατακερματισμού εισάγει καθυστερήσεις λόγω της ανάγκης επανασύνθεσης. Επιπλέον, η πιθανότητα απώλειας τμημάτων αυξάνει το ρίσκο αποτυχίας της μεταφοράς, καθώς η απώλεια ενός τμήματος απαιτεί την επαναμετάδοση ολόκληρου του πακέτου.

Η ανάλυση αυτή καταδεικνύει τη σημασία της σωστής διαχείρισης του MTU και υπογραμμίζει τη χρησιμότητα τεχνικών, όπως το Path MTU Discovery (PMTUD), για τη βελτιστοποίηση της αποδοτικότητας του δικτύου.