

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ II

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 ερμηνευση 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 πλήρης 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 your-local-path/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 το περιεχόμενο του μηνύματος. Παρατίθενται στιγμιότυπα από το Wireshark που αναδεικνύουν την ιδιαιτερότητα αυτή.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.2.9	192.168.2.6	UDP	50	60905 → 12345 Len=8
2	0.000215	192.168.2.9	192.168.2.6	UDP	58	60905 → 12345 Len=16
3	11.615791	192.168.2.6	192.168.2.9	UDP	60	61886 → 12345 Len=8
4	11.615795	192.168.2.6	192.168.2.9	UDP	63	61886 → 12345 Len=21

Ανταλλαγή πακέτων κειμένου

The image shows a Wireshark packet capture of a UDP packet. The packet list on the left shows four packets. The selected packet (No. 2) is expanded, showing the following details:

- Frame 1: 50 bytes on wire (400 bits), 50 bytes captured (400 bits) on interface en0, id 0
- Ethernet II, Src: de:89:da:8d:57:90 (de:89:da:8d:57:90), Dst: RfLinkTechno_98:b9:72 (38:a2:8c:98:b9:72)
- Internet Protocol Version 4, Src: 192.168.2.9, Dst: 192.168.2.6
- 0100 = Version: 4
- 0101 = Header Length: 20 bytes (5)
- > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 36
- Identification: 0xc67 (53095)
- > 000. = Flags: 0x0
- ... 0 0000 0000 0000 = Fragment Offset: 0
- Time to Live: 64
- Protocol: UDP (17)
- Header Checksum: 0x2602 [validation disabled]
- [Header checksum status: Unverified]
- Source Address: 192.168.2.9
- Destination Address: 192.168.2.6
- [Stream index: 0]
- > User Datagram Protocol, Src Port: 60905, Dst Port: 12345
- Source Port: 60905
- Destination Port: 12345
- Length: 16
- Checksum: 0x6671 [unverified]
- [Checksum Status: Unverified]
- [Stream index: 0]
- [Stream Packet Number: 1]
- > [Timestamps]
- > UDP payload (8 bytes)
- > Data (8 bytes)
- Data: 4d53472030303136
- [Length: 8]

The packet bytes pane on the right shows the raw data of the packet, including the Ethernet II header, the IP header, and the UDP header and payload.

Πακέτο Κεφαλίδας

```
> Frame 2: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface en0, id 0
> Ethernet II, Src: de:89:da:0d:57:90 (de:89:da:0d:57:90), Dst: RfLinkTechno_98:b9:72 (38:a2:8c:98:b9:72)
> Internet Protocol Version 4, Src: 192.168.2.9, Dst: 192.168.2.6
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 44
    Identification: 0x74ad (29869)
  > 0000 .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x0004 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.2.9
    Destination Address: 192.168.2.6
    [Stream index: 0]
  > User Datagram Protocol, Src Port: 60905, Dst Port: 12345
    Source Port: 60905
    Destination Port: 12345
    Length: 24
    Checksum: 0x2273 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 2]
  > [Timestamps]
    UDP payload (16 bytes)
  > Data (16 bytes)
    Data: 48656d66720576f726c642074657374
    [Length: 16]
```

Πακέτο Δεδομένων

```
> Frame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface en0, id 0
> Ethernet II, Src: RfLinkTechno_98:b9:72 (38:a2:8c:98:b9:72), Dst: de:89:da:0d:57:90 (de:89:da:0d:57:90)
> Internet Protocol Version 4, Src: 192.168.2.6, Dst: 192.168.2.9
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 36
    Identification: 0x1183 (4483)
  > 0000 .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0xa3e6 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.2.6
    Destination Address: 192.168.2.9
    [Stream index: 0]
  > User Datagram Protocol, Src Port: 61886, Dst Port: 12345
    Source Port: 61886
    Destination Port: 12345
    Length: 16
    Checksum: 0x61a1 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
    [Stream Packet Number: 1]
  > [Timestamps]
    UDP payload (8 bytes)
  > Data (8 bytes)
    Data: 4053472030303231
    [Length: 8]
```

Πακέτο Κεφαλίδας Κειμένου

```
> Frame 4: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface en0, id 0
> Ethernet II, Src: RfLinkTechno_98:b9:72 (38:a2:8c:98:b9:72), Dst: de:89:da:0d:57:90 (de:89:da:0d:57:90)
> Internet Protocol Version 4, Src: 192.168.2.6, Dst: 192.168.2.9
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 49
    Identification: 0x1184 (4484)
  > 0000 .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0xa3d8 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.2.6
    Destination Address: 192.168.2.9
    [Stream index: 0]
  > User Datagram Protocol, Src Port: 61886, Dst Port: 12345
    Source Port: 61886
    Destination Port: 12345
    Length: 29
    Checksum: 0x31ce [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
    [Stream Packet Number: 2]
  > [Timestamps]
    UDP payload (21 bytes)
  > Data (21 bytes)
    Data: 48656d66720576f726c642074657374206261636b
    [Length: 21]
```

Πακέτο Περιεχομένου Κειμένου

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

Παρατίθενται εικόνες από το Wireshark των πακέτων που αφορούν τη σύνδεση των δυο συσκευών καθώς και η επικοινωνία των δύο συσκευών.

```
> Frame 1: 50 bytes on wire (400 bits), 50 bytes captured (400 bits) on interface en0, id 0
> Ethernet II, Src: de:8b:da:8d:57:98 (de:8b:da:8d:57:98), Dst: ff:ff:ff:ff:ff:ff (38:a2:8c:98:b9:72)
> Internet Protocol Version 4, Src: 192.168.2.9, Dst: 192.168.2.6
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 36
    Identification: 0xb765 (46949)
  > 0000 .... = Flags: 0x0
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0x3e04 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.2.9
  Destination Address: 192.168.2.6
  [Stream index: 0]
  > User Datagram Protocol, Src Port: 51407, Dst Port: 12345
    Source Port: 51407
    Destination Port: 12345
    Length: 16
    Checksum: 0xbdf63 [unverified]
    [Checksum status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 1]
  > [Timestamps]
  > UDP payload (8 bytes)
  > Data (4 bytes)
    Data: 434f4e4330303034
    [Length: 8]
```

Πακέτο Κεφαλίδας Σύνδεσης

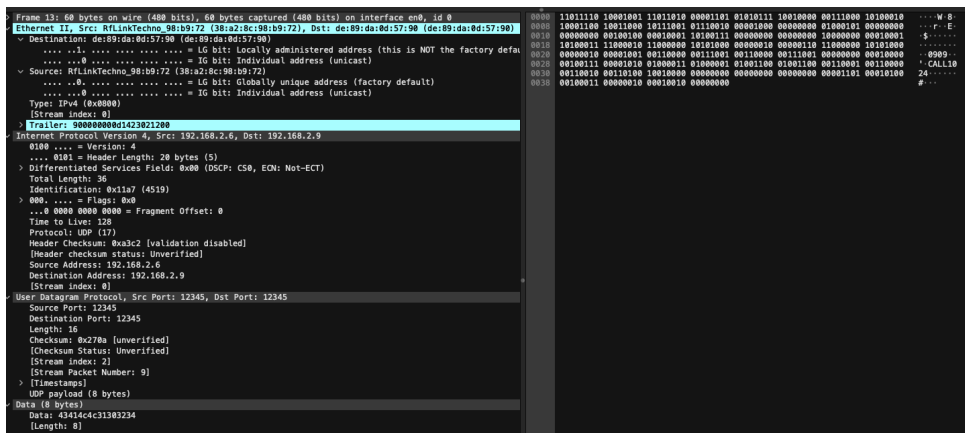
```
> Frame 2: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface en0, id 0
> Ethernet II, Src: de:8b:da:8d:57:98 (de:8b:da:8d:57:98), Dst: ff:ff:ff:ff:ff:ff (38:a2:8c:98:b9:72)
> Internet Protocol Version 4, Src: 192.168.2.9, Dst: 192.168.2.6
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 32
    Identification: 0x60de (26334)
  > 0000 .... = Flags: 0x0
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0x8e8f [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.2.9
  Destination Address: 192.168.2.6
  [Stream index: 0]
  > User Datagram Protocol, Src Port: 51407, Dst Port: 12345
    Source Port: 51407
    Destination Port: 12345
    Length: 12
    Checksum: 0xb19f [unverified]
    [Checksum status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 2]
  > [Timestamps]
  > UDP payload (4 bytes)
  > Data (4 bytes)
    Data: 63616d63
    [Length: 4]
```

Πακέτο Περιεχομένου Σύνδεσης

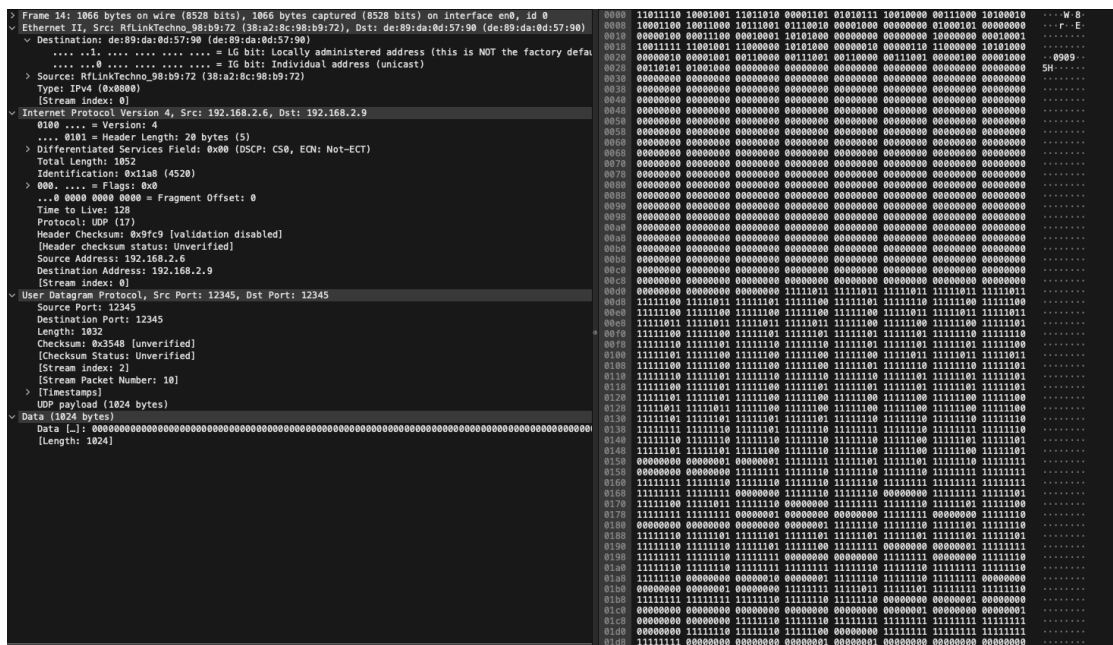
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.2.9	192.168.2.6	UDP	50	51407 → 12345 Len=8
2	0.000006	192.168.2.9	192.168.2.6	UDP	46	51407 → 12345 Len=4
3	3.849021	192.168.2.6	192.168.2.9	UDP	60	56923 → 12345 Len=8
4	3.849023	192.168.2.6	192.168.2.9	UDP	60	56923 → 12345 Len=8
5	4.114752	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
6	4.114785	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
7	4.245206	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
8	4.245239	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
9	4.284382	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
10	4.284384	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
11	4.505610	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
12	4.505654	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
13	4.552156	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
14	4.552159	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
15	4.762971	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
16	4.762994	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
17	4.808980	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
18	4.808983	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
19	5.019050	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
20	5.019077	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
21	5.069700	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
22	5.069703	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
23	5.279408	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
24	5.279440	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
25	5.329019	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
26	5.329021	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
27	5.539774	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
28	5.539804	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
29	5.593160	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
30	5.593162	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
31	5.800122	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
32	5.800169	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
33	5.855385	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
34	5.855388	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
35	5.930433	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
36	5.930462	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
37	5.991197	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
38	5.991199	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
39	6.188191	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
40	6.188225	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
41	6.254215	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
42	6.254217	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
43	6.448541	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
44	6.448608	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024
45	6.515661	192.168.2.6	192.168.2.9	UDP	60	12345 → 12345 Len=8
46	6.515664	192.168.2.6	192.168.2.9	UDP	1066	12345 → 12345 Len=1024
47	6.706323	192.168.2.9	192.168.2.6	UDP	50	12345 → 12345 Len=8
48	6.706353	192.168.2.9	192.168.2.6	UDP	1066	12345 → 12345 Len=1024

Ανταλλαγή Πακέτων μεταξύ των δύο συσκευών κατά τη διάρκεια ηχητικής κλήσης

Συγκεκριμένα, για ένα πακέτο με δεδομένα ηχητικής κλήσης και το αντίστοιχο για την κεφαλίδα:



Πακέτο Κεφαλίδας Ήχου



Πακέτο Περιεχομένου Ήχου

Η επικοινωνία είναι ασύγχρονη, αμφίδρομη και μη διακοπτόμενη.

- Η επικοινωνία είναι ασύγχρονη καθώς δεν απαιτείται και οι δύο συμμετέχοντες να είναι ταυτόχρονα ενεργοί. Τα μηνύματα μπορούν να αποστέλλονται και να λαμβάνονται ανεξάρτητα από τον χρόνο παράδοσής τους.
- Η επικοινωνία είναι επίσης αμφίδρομη, επειδή και οι δύο συμμετέχοντες έχουν τη δυνατότητα να στέλνουν και να λαμβάνουν δεδομένα είτε ταυτόχρονα είτε διαδοχικά. Αυτό υποστηρίζεται από τη χρήση socket APIs, που επιτρέπουν στα δύο μέρη να λειτουργούν τόσο ως πελάτης όσο και ως διακομιστής.
- Τέλος, η επικοινωνία χαρακτηρίζεται ως μη διακοπτόμενη, καθώς διασφαλίζεται η συνεχής ροή δεδομένων χωρίς παύσεις ή απώλεια κρίσιμων τμημάτων.

Η διαδικασία επανασύνθεσης των τμημάτων ολοκληρώθηκε επιτυχώς μέσω του Wireshark, 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.

*** Τα πλήρη αρχεία με τα καταγεγραμμένα πακέτα βρίσκονται στο συμπιεσμένο .zip αρχείο.**