

CN_II Chat and VoIP over UDP 2024

Ομάδα ΑΕ

Χρήστος Μάριος Περδίκης 10075 cperdikis@ece.auth.gr

Αιμιλία Παλάσκα 10453 aimiliapm@ece.auth.gr

Το παρόν έγγραφο είναι η αναφορά της εργασίας του μαθήματος Δίκτυα Υπολογιστών II. Κληθήκαμε να υλοποιήσουμε μια εφαρμογή VoIP και Chat σε Java η οποία θα χρησιμοποιεί το πρωτόκολλο UDP. Συγκεκριμένα υλοποιήσαμε τον κώδικα αποστολής και λήψης πακέτων κειμένου και φωνής. Ακολουθεί η περιγραφή του κώδικα που γράψαμε.

1 Ανταλλαγή μηνυμάτων κειμένου

Λαμβάνουμε datagrams κειμένου στη θύρα `text_dest_port = 26557` και στέλνουμε από τη θύρα `text_dest_port = 26557`. Το μέγεθος του payload κάθε datagram είναι 1024 bytes, αλλά όπως θα δούμε παρακάτω υποστηρίζουμε να σταλούν μηνύματα μεγαλύτερου payload.

1.1 Receive

Ο μηχανισμός λήψης μηνυμάτων υλοποιείται στην συνάρτηση `receiveText`. Για να εγγυηθούμε ότι η λήψη πακέτων δεν θα περιορίσει την υπόλοιπη λειτουργικότητα της εφαρμογής, δηλαδή την αποστολή μηνυμάτων και την δυνατότητα κλήσης, η συνάρτηση αυτή ανατίθεται σε ένα Thread, το οποίο ξεκινάει από την `main` και εκτελεί μία αέναη λούπα. Μέσα σε αυτήν λαμβάνονται πακέτα ορισμένου μεγέθους (1024 bytes) και εμφανίζονται στην διεπαφή, έχοντας σαν αρχή το "Friend:" για να γίνει αντιληπτό πότε πρόκειται για εισερχόμενο μήνυμα. Θεωρούμε ότι όλα τα πακέτα θα είναι το πολύ 1024 bytes, διότι η διαδικασία αποστολής μηνυμάτων διαχειρίζεται τον χωρισμό μεγάλων συμβολοσειρών σε πολλαπλά πακέτα. Στην διάρκεια των δοκιμών μας δεν παρατηρήθηκε άφιξη των πακέτων με λάθος σειρά, όταν αυτά προέρχονταν από χωρισμό μεγαλύτερου payload.

1.2 Send

Αρχικά ελέγχουμε αν το `inputTextField` είναι άδειο, στην οποία περίπτωση αγνοούμε το πάτημα του κουμπιού Send και δεν στέλνουμε τίποτα. Αν δεν είναι άδειο, τότε ξεκινάμε τη διαδικασία αποστολής ενός udp datagram.

Αφότου αποθηκεύσουμε το `input_text` στη μεταβλητή `payload` σε μορφή bytes, υπολογίζουμε πόσες φορές χωράει το 1024 στο μήκος του `payload`. Στη γενική περίπτωση, θα σταλούν σε αριθμό `multiplier + 1` datagrams, όπου όλα εκτός του τελευταίου θα έχουν μέγεθος payload 1024 bytes και το τελευταίο θα έχει μέγεθος payload `modulo` bytes (`multiplier` είναι το πηλίκο και `modulo` το υπόλοιπο της διαίρεσης `payload.length/1024`).

Για παράδειγμα, αν έχουμε ένα μήνυμα σε μέγεθος 2050 bytes, το μήνυμα θα χωριστεί και θα σταλεί με δύο datagrams με payload μήκους 1024 bytes και ενός ακόμα datagram με payload μήκους $2050 \% 1024 = 2$ bytes. Αν πάλι στείλουμε ένα μήνυμα μεγέθους 256 bytes, θα στείλουμε ένα payload μεγέθους 256 bytes. Προφανώς τα τελικά datagrams θα είναι λίγο μεγαλύτερα λόγω της προσθήκης του udp header.

Ουσιαστικά, αν λάβουμε υπόψιν μας και τον κώδικα που λαμβάνει κείμενο, η εφαρμογή μας τεμαχίζει τα πολύ μεγάλα κείμενα σε περισσότερα μικρότερα διαδοχικά μηνύματα.

1.3 Παράδειγμα ανταλλαγής μηνυμάτων μέσω Wireshark

Ακολουθούν ανταλλαγές μηνυμάτων κειμένου μεταξύ δύο υπολογιστών στο ίδιο δίκτυο οι οποίες καταγράφηκαν με το πρόγραμμα Wireshark. Οι διευθύνσεις IPv4 των δύο υπολογιστών ήταν 192.168.100.22 και 192.168.100.13.

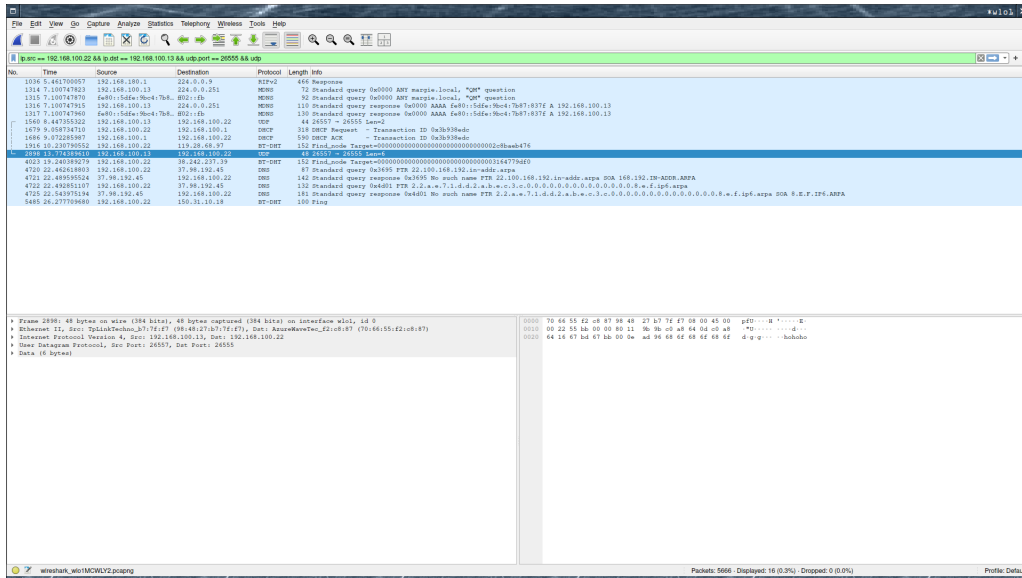


Figure 1: Καταγραφή μηνύματος μικρού μήκους στο Wireshark

Στην εικόνα 1 βλέπουμε την αποστολή ενός μηνύματος με κείμενο “hohoho”. Εφόσον έχουμε κείμενο μήκους 6 bytes, το datagram payload έχει και αυτό μέγεθος 6 bytes.

Στις εικόνες 2a, 2b και 2c στείλαμε τον ίδιο τον πηγαίο κώδικα μέσω της εφαρμογής για να δοκιμάσουμε τη συμπεριφορά της σε πολύ μεγάλα κείμενα (> 1024 bytes). Βλέπουμε ότι το κείμενο χωρίζεται σε πολλά datagrams με μέγεθος payload 1024 bytes μέχρι το τελευταίο datagram στο οποίο στέλνεται ουσιαστικά.

2 VoIP

2.1 Receive

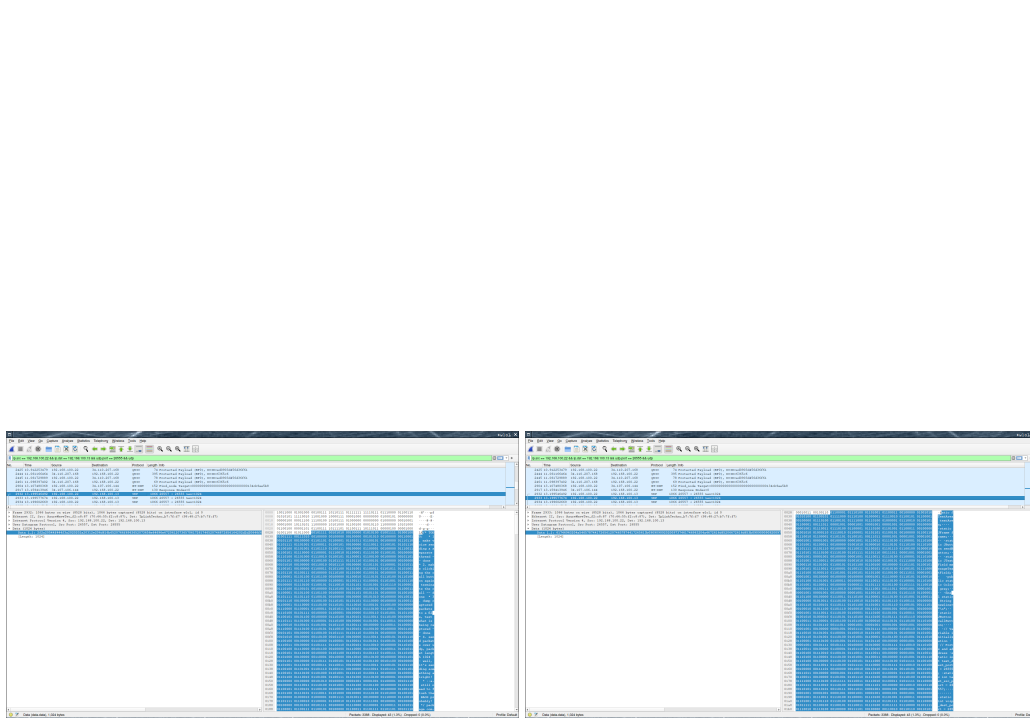
2.2 Send

Αντίστοιχα με λήψη και αποστολή κειμένου λαμβάνουμε datagrams φωνής στη θύρα `voice_dest_port = 26567` και στέλνουμε από τη θύρα `voice_src_port = 26565`. Αν πατήσουμε το κουμπί Call και δεν βρισκόμαστε σε κλήση, εκτός από το thread λήψης πακέτων φωνής, γίνεται εκκίνηση του thread καταγραφής και αποστολής φωνής. Αν είμαστε ήδη σε κλήση, τότε τα δύο threads τερματίζουν ομαλά και η κλήση τελειώνει. Είναι δυνατόν η ξαναεπαρκεκκίνηση της κλήσης αν πατηθεί το κουμπί Call ξανά.

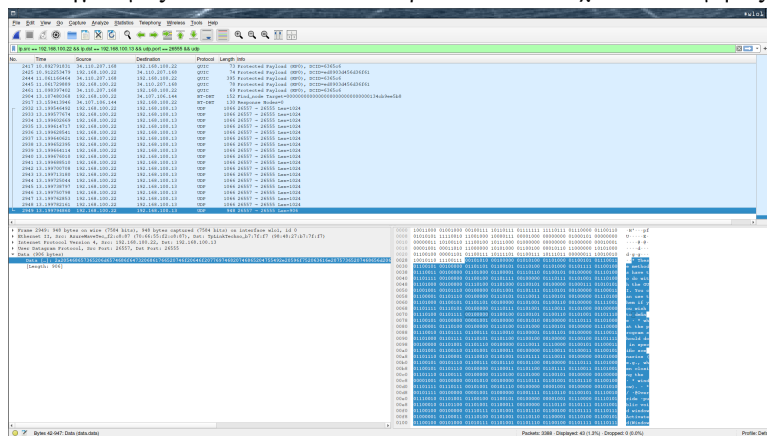
Στο ξεχωριστό thread αναλαμβάνουμε την καταγραφή και την αποστολή ήχου. Με τις κατάλληλες ρυθμίσεις στον υπολογιστή του χρήστη μπορεί να γίνει καταγραφή φωνής μέσω του μικροφώνου αλλά και καταγραφή των ήχων του συστήματος όπως π.χ. μουσικής. Αφού αρχικοποιήσουμε το socket και την διεύθυνση του προορισμού, θέτουμε το `AudioFormat` μας να είναι 8kHz μονοφωνικό PCM με 8-bits για κάθε sample. Αν είχαμε παραπάνω από ένα byte για κάθε sample, η διάταξη μας θα ήταν big endian.

Ανοίγουμε ένα `TargetDataLine` με `buffer_size = 1024` και ξεκινάμε την καταγραφή ήχου σε ένα while loop το οποίο τερματίζει μόνο όταν πατηθεί το κουμπί Call. Κάθε φορά διαβάζουμε $1024/5 = 204$ bytes από τον εσωτερικό buffer του `TargetDataLine`, έτσι σιγουρευόμαστε ότι δεν θα έχουμε ούτε buffer overflows ούτε underflows. Τα δεδομένα ήχου γίνονται append στο `ByteArrayOutputStream out`, το οποίο μετατρέπουμε στο byte array `audio_data` όταν θέλουμε να τα χρησιμοποιήσουμε.

Όταν το `audio_data` γεμίσει επαρκώς, τότε στέλνουμε ένα πακέτο. Αυτό γίνεται σειριακά με την καταγραφή ήχου, στο ίδιο thread. Αν επιθυμούσαμε μικρότερο latency θα μπορούσαμε να είχαμε ξεχωριστά threads για την καταγραφή ήχου και την αποστολή πακέτων. Όταν το `audio_data` συμπληρώσει 1024 bytes, θα στείλουμε το πρώτο πακέτο το οποίο θα περιέχει τα πρώτα 1024 bytes του `audio_data`. Όταν συμπληρώσει $2 * 1024 = 2048$ bytes, θα στείλουμε 1024 bytes με offset $1 * 1024$ bytes από την αρχή του `audio_data`. Όμοια στα $3 * 1024 = 3072$



(α) Πρώτο datagram από πολλά που στάλθηκαν για (β) Δεύτερο datagram. Όλα μέχρι και το να στείλουμε τον πηγαίο μας κώδικα!



(c) Τελευταίο datagram, μικρότερο σε μέγεθος από όλα τα προηγούμενα!

Figure 2: Καταγραφή αποστολής πολύ μεγάλου κειμένου με Wireshark

bytes θα στείλουμε 1024 bytes με offset $2 * 1024$ bytes από την αρχή του *audio_data* κλπ.

Όταν το κουμπί Call πατηθεί όσο η κλήση βρίσκεται σε εξέλιξη φροντίζουμε να κλείσουμε τα *TargetDataLine* και *DatagramSocket* μας για να μην έχουμε resource leaks. Υποθέτουμε ότι οι κλήσεις χρησιμοποιώντας αυτήν την εφαρμογή θα έχουν διάρκεια μερικών λεπτών, καθώς αφήνουμε το *out* να μεγαλώνει ανεξέλεγκτα. Θεωρητικά αν περάσει επαρκής χρόνος τα *out* και *audio_data* θα γίνουν τόσο μεγάλα ώστε να καταλάβουν όλη τη μνήμη του συστήματος (και λογικά θα τερματιστεί η εφαρμογή μας από το λειτουργικό σύστημα του υπολογιστή στον οποίο τρέχει).

2.3 Παράδειγμα VoIP μέσω Wireshark

3 Συμπεράσματα;

Χρειάζεται αυτό το section; Ο Τσακ Νόρις είπε “έλα μου ντε” και ο Ντε ήρθε...