Report

Τσούσης Παναγιώτης
ΑΕΜ: 9590
pitsousis@ece.auth.gr

Statement of Originality:

Ο κώδικας που έχει αναπτυχθεί στην εργασία αυτή είναι κατά το μεγαλύτερο ποσοστό δικής μου έμπνευσης, όμως περιέχει και κομμάτια που πήρα από την εκφώνηση της εργασίας στη σελίδα 24, που έχουν σχέση με την αποστολή αιτημάτων στο server, λήψη πακέτων UDP και αναπαραγωγή ήχου.

Το πρωτόκολλο UDP:

Το UDP είναι ένα από τα βασικά πρωτόκολλα της Σουίτας Πρωτοκόλλων του Διαδικτύου. Σύμφωνα με το μοντέλο OSI, κατατάσσεται στο στρώμα της μεταφοράς.

Το UDP επιτρέπει σε εφαρμογές υπολογιστών να επικοινωνήσουν , μέσω μηνυμάτων που αποκαλούνται datagrams, με άλλους χρήστες στο πρωτόκολλο του διαδικτύου.

Σε γενικές γραμμές, το UDP είναι ένα απλό πρωτόκολλο επικοινωνίας, με τα ελάχιστα πρωτόκολλα. Δεν εξασφαλίζει, λοιπόν, αξιοπιστία κατά τη μεταφορά των πακέτων, όπως το πρωτόκολλο TCP , για παράδειγμα. Αυτό έχει ως συνέπεια πολλά πακέτα να μη μεταφέρονται σωστά. Το καλό που έχει, όμως, είναι ότι επειδή υπάρχει αυτή η έλλειψη ελέγχου ως προς την σωστή μεταφορά των δεδομένων, δεν γίνεται επαναποστολή χαμένων ή κατεστραμμένων πακέτων, που έχει ως συνέπεια τη μεγάλη (σε σχέση με το TCP, για παράδειγμα) ταχύτητα αποστολής πακέτων.

Η χρήση του πρωτοκόλλου UDP γίνεται σε εφαρμογές στις οποίες η γρήγορη μεταφορά των δεδομένων είναι πιο σημαντική από την εξασφάλιση

μεταφοράς όλων των πακέτων. Τέτοιες εφαρμογές είναι οι εφαρμογές audio και video streaming, που απαιτούν real-time, γρήγορη μεταφορά δεδομένων για τη συνεχή ροή του stream. Συνήθως, λόγω περιορισμών του ανθρώπινου σώματος, η έλλειψη πακέτων δεν γίνεται καν αντιληπτή από το χρήστη.

Source: https://en.wikipedia.org/wiki/User_Datagram_Protocol

Audio Streaming Protocols:

Οι εφαρμογές audio streaming επιτρέπουν την γρήγορη λήψη πακέτων ήχου από μια εφαρμογή audio streaming και τη σχεδόν άμεση αναπαραγωγή τους από αυτή. Για να μπορέσουν να λειτουργήσουν οι εφαρμογές αυτές, εκμεταλλεύονται τα streaming protocols, που δημιουργήθηκαν με το σκοπό αυτό. Τα βασικότερα από αυτά τα πρωτόκολλα είναι τα:

- Real-time Transport Protocol (RTP)
- RTP Control Protocol (RTCP)
- Real Time Streaming Protocol (RTSP)

Το πρωτόκολλο RTP είναι ένα πρωτόκολλο δικτύου, υπεύθυνο για την παράδωση audio και video σε δίκτυα πρωτοκόλλου Ίντερνετ (IP networks). Χρησιμοποιείται σε επικοινωνιακά και ψυχαγωγικά συστήματα όπως η τηλεφωνία, τηλεδιασκέψεις και υπηρεσίες διαδικτυακής τηλεοράσεως. Τυπικά, το πρωτόκολλο αυτό ‘τρέχει’ πάνω στο πρωτόκολλο UDP και σε συνεργασία με το πρωτόκολλο RTCP. Το RTP αναπτύχθηκε από την ομάδα Audio-Video Transport Working Group of the Internet Engineering Task Force (IETF) και κοινοποιήθηκε για πρώτη φορά το 1996 ως RFC 1889.

Το πρωτόκολλο RTCP τρέχει παράλληλα με το RTP και παρέχει στατιστικές πληροφορίες για μια συνεδρία RTP. Συμβάλει στην παράδωση και στο πακετάρισμα δεδομένων, αλλά δεν είναι υπεύθυνο για την μεταφορά τους. Μια εφαρμογή μπορεί να χρησιμοποιήσει τις στατιστικές πληροφορίες που παρέχει το πρωτόκολλο RTCP σχετικά με το Quality of Service (QoS) για να ρυθμίσει τις παραμέτρους του και τη λειτουργία του.

Το πρωτόκολλο RTSP είναι σχεδιασμένο για χρήση στα τηλεπικοινωνιακά συστήματα και παρέχει έλεγχο των streaming media servers. Οι χρήστες των media servers μπορούν να στείλουν εντολές όπως play, record και pause και έτσι έχουν έλεγχο πραγματικού χρόνου του media server.

Sources:

- https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
- https://en.wikipedia.org/wiki/RTP_Control_Protocol
- https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol
- https://en.wikipedia.org/wiki/Streaming_media

Σχόλια πάνω στις μετρήσεις

Σημείωση: Τα διαγράμματα και οι μετρήσεις φαίνονται στα αρχεία των sessions. Συνεπώς, για να έχουν κάποιο νόημα τα παρακάτω σχόλια, καλό είναι ο αναγνώστης να έχει ανοιχτό ένα τουλάχιστον αρχείο από τα sessions ταυτόχρονα με το παρόν.

Διάγραμμα G1:

Εδώ, εύκολα βλέπουμε ότι χρόνος απόκρισης κυμαίνεται από σχεδόν ακαριαίος (ελάχιστα ms), έως και περίπου 2000ms. Έχουμε μέση τιμή τα 1245-1337ms και διασπορά 241475-301277, που είναι σχετικά μεγάλες τιμές. Δε φαίνεται να ακολουθούν κάποιο συγκεκριμένο μοτίβο ως προς το χρόνο, είναι όμως εμφανές ότι οι πιο συχνές τιμές βρίσκονται απο τα 600ms και πάνω.

Διάγραμμα G2:

Η ρυθμαπόδοση του συστήματος για τα τελευταία 32 δευτερόλεπτα φαίνεται να μην είναι σταθερή, αλλά βρίσκεται πάνω απο τα 150 bit/sec και κάτω από τα 250 bit/sec.

Διάγραμμα G3:

Σε σύγκριση με τα πακέτα με καθυστέρηση, τα πακέτα χωρίς καθυστέρηση είναι πολύ πιο γρήγορα και πολύ πιο σταθερά ως προς το χρόνο απόκρισης. Οι τιμές τους είναι συνήθως λίγο κάτω από 250ms, όμως υπάρχουν και μερικές περιπτώσεις στις οποίες το ξεπερνάνε και μάλιστα κατά πολύ. Όπως και να έχει, κανένα πακέτο δεν κατάφερε να ξεπεράσει τα 1000ms.

Διάγραμμα G4:

Η ρυθμαπόδοση του συστήματος σε αυτό το σύστημα φαίνεται να είναι σχετικά σταθερή, με μερικές απότομες πτώσεις, αλλά συνήθως να κυμαίνεται γύρω από τα 1060 bit/sec περίπου.

Διάγραμμα G5:

Οι τιμές του χρόνου απόκρισης δεν είναι ξεκάθαρο αν ακολουθούν κάποια κατανομή. Όμως, αν αγνοήσουμε τις τιμές μικρότερες απο 400ms και υποθέσουμε ότι στα αριστερά θα μπορούσαν να υπάρξουν και άλλες τιμές, αν δεν τις ‘trimαρε’ το timeout της εφαρμογής μας, θα μπορούσαμε να κάνουμε την υπόθεση ότι ακολουθούν μια normal ή log-normal κατανομή.

Διάγραμμα G6:

Οι τιμές της ρυθμαπόδοσης φαίνεται να ακολουθούν επίσης την κανονική κατανομή.

Διάγραμμα G7:

Οι τιμές του χρόνου απόκρισης σίγουρα φαίνεται να ακολουθούν κάποιο μοτίβο, καθώς το διάγραμμα μοιάζει σαν να έχουμε υπέρθεση 2 laplace κατανομών γύρω από 2 κοντινές τιμές. Όμως, αυτή δεν είναι κάποια γνωστή μορφή, ώστε να μπορέσουμε να της δώσουμε κάποιο όνομα.

Διάγραμμα G8:

Η ρυθμαπόδοση φαίνεται να παίρνει τιμές συγκεκριμένες μέσα σε ένα εύρος, με μικρό ποσοστό να βρίσκεται εκτός αυτών, αλλά δε φαίνεται να ακολουθεί κάποια κατανομή και πάλι.

Διάγραμμα R1:

Το SRTT φαίνεται να μεταβάλλεται πολύ και με γρήγορο ρυθμό, όπως και το SRTTVAR. Από την άλλη, το RTO δε φαίνεται να έχει τόσο μεγάλες μεταβολές. Βέβαια, για τα αποτελέσματα αυτά ευθύνονται, εννοείται, και οι συντελεστές που δώσαμε στις συναρτήσεις, σύμφωνα με το site https://tools.ietf.org/html/rfc2988

Διάγραμμα G9:

Το μόνο σχόλιο που μπορούμε να κάνουμε σχετικά με το διάγραμμα είναι ότι σίγουρα μοιάζει με τα διαγράμματα ήχου που πιθανότατα βρίσκουμε σε προγράμματα καταγραφής και αναπαραγωγής ήχου.

Διάγραμμα G10:

Εδώ άμεσα παρατηρούμε μια σχετικά ‘flat’ κυματομορφή, που παρουσιάζει υψηλή επαναληπτικότητα. Αυτό είναι λογικό, καθώς μιλάμε για μία καθαρή συχνότητα, που σημαίνει ότι παράγεται από σταθερές και περιοδικές διαφορές δειγμάτων.

Διάγραμμα G11:

Εδώ, η κυματομορφή μας αποκαλύπτει την κατανομή laplace των διαφορών, που δεν είναι τόσο απότομη βέβαια.

Διάγραμμα G12:

Στη συγκεκριμένη περίπτωση, ιδίως στο session 2, τα δείγματα ακολουθούν μια σχεδόν τέλεια κατανομή laplace.

Διάγραμμα G13:

Εδώ, βλέπουμε πως οι διαφορές των δειγμάτων με κωδικοποίηση AQ-DPCM μάλλον ακολουθούν μια απότομη κατανομή laplace.

Διάγραμμα G14:

Όπως και στις τιμές των διαφορών , είναι αναμμενόμενο και εμφανές πως πάλι, η κατανομή των δειγμάτων μοιάζει να είναι μια απότομη laplace.

Διαγράμματα G15/G16/G17/G18:

Στα διαγράμματα αυτά βλέπουμε πως η μέση τιμή και το βήμα δεν ακολουθούν κάποιο μοτίβο, αλλά για την ακρίβεια κυμαίνονται αρκετά. Βέβαια, η μέση τιμή ‘παίζει’ γύρω από το 0, ενώ το βήμα, αν και όχι με τόση σταθερότητα, βρίσκεται κοντά στο 400.

Διάγραμμα G19:

Από τα διαγράμματα, μπορούμε να δούμε ότι από κατάσταση ηρεμίας, αν αυξήσουμε το motor speed, το ύψος του IthakiCopter φτάνει μια μέγιστη τιμή. Όταν την πιάσει, τότε οι κινητήρες ρίχνουν σταδιακά την ταχύτητά τους, μέχρι το ύψος να πέσει πάλι στο ελάχιστο. Επίσης μπορούμε να δούμε την ελάχιστα μειωμένη τιμή της πίεσης του αέρα όταν το IthakiCopter βρίσκεται πιο ψηλά, όπως και την πτώση της θερμοκρασίας.

Διάγραμμα G20:

Εδώ μπορούμε να δούμε ότι τα 3 διαγράμματα έχουν σχεδόν ίδια μορφή, απλά σε άλλη κλίμακα. Είναι εμφανές όμως, πως η μεταβολή μιας παραμέτρου επηρεάζει αναλογικά τις άλλες παραμέτρους, πράγμα που σημαίνει, για παράδειγμα, πως η μεταβολή στο throttle, σημαίνει και σχεδόν

αναλογική μεταβολή στο engine RPM. Βέβαια, σε σχέση με τις άλλες μεταβλητές, το engine RMP φαίνεται να έχει και μια κλίση κατά πάνω. Αυτό μπορεί να σημαίνει πως όσο περισσότερο περνάει ο χρόνος, το engine RPM αυξάνει και από μόνο του, χωρίς απαραίτητα να αλλάζουν και οι υπόλοιπες μεταβλητές, που θα ήταν ισχυρή ένδειξη ότι η λειτουργία του κινητήρα γίνεται καλύτερη όταν το όχημα εχει ‘τρέξει’ για λίγο.

Θερμοκρασία στο σταθμό:

Εδώ πρέπει να σημειωθούν οι υποψίες μου για την ελλατωματική λειτουργία του σταθμού, καθώς πήρα μία μέτρηση Δεκέμβριο στις 3 το πρωί και μου έβγαλε +21 C. Τόσο είχε, όμως, το δωμάτιο από το οποίο πήρα τη μέτρηση, μέσα στο οποίο λειτουργούσε κλιματιστικό.

Source code report

1. Σημειώσεις:

Ο κώδικας έχει αναπτυχθεί με θεμέλιο τον κώδικα που δώθηκε στην εκφώνηση της εργασιας, στη σελίδα 24. Αυτό περιλαμβάνει το setup των αντικειμένων DatagramSocket και DatagramPacket, την αποστολή ενός request προς το σέρβερ Ιθάκη, την λήψη των πακέτων που αποστέλονται προς το χρήστη και το setup για την αναπαραγωγή ήχου.

Ο κώδικας δε θα αναλυθεί γραμμή προς γραμμή, αλλά από μια πιο γενική σκοπιά, ώστε να γίνει κατανοητό πως λειτουργεί, χωρίς όμως να κουράσει (όχι πολύ, τουλάχιστον) τον αναγνώστη.

Συνάρτηση main():

Στις αρχικές σειρές της main ορίζουμε δίαφορες μεταβλητές, όπως τα ports και τους βασικούς κωδικούς που δίνονται από το server, αλλά και μερικές επιπλέον ώστε να καθορίσουμε τις παραμέτρους μας για τα διάφορα ζητούμενα και να δημιουργήσουμε τους τελικούς κωδικούς που θα στείλουμε. Τέτοιες μεταβλητές είναι, για παράδειγμα, το echo_run_time, που

καθορίζει για πόσα λεπτά θα λαμβάνουμε πακέτα echo από το server, καθώς και η mode που καθορίζει πόσα πακέτα ήχου θα λάβουμε.

Ο χρήστης θα πρέπει να βάλει τους σωστούς κωδικούς και ports όπως δίνονται από το server, καθώς και να αλλάξει τις παραμέτρους σε αυτά που θέλει, ώστε να λειτουργήσει σωστά η εφαρμογή.

Στις επόμενες γραμμές, καλούμε τις διάφορες συναρτήσεις που θέλουμε να τρέξουν. Τα ονόματά τους είναι αρκετά περιγραφικά, ώστε να μη χρειάζεται περαιτέρω εξήγηση το τι ακριβώς κάνει η κάθε μια, αν και η εξήγηση θα δωθεί παρακάτω. Το μόνο που θα σημειωθεί είναι ότι οι συναρτήσεις start_main_graphics() και end_main_graphics() δεν κάνουν τίποτα πέρα από την εκτύπωση μερικών string στην κονσόλα στην αρχή και στο τέλος της main, με σκοπό να κάνουν την εργασία πιο εμφανίσιμη, αλλά και να βοηθήσουν με ένα πιο διαισθητικό τρόπο τον προγραμματιστή να διαπιστώσει τη σωστή λειτουργία του προγράμματός του.

Συνάρτηση stringToTextFile(String name, String input):

Σκοπός της συνάρτησης αυτής είναι η διευκόλυνση του output αρχείων txt.

Ο χρήστης ορίζει το όνομα που θέλει να δώσει στα αρχεία καθώς και τη μεταβλητή String που περιέχει τα δεδομέναε που θέλει να κάνει output και η συνάρτηση αυτή φροντίζει να εκτελέσει τη διαδικασία της εξαγωγής των δεδομένων σε αρχείο txt.

Συνάρτηση bytesToInt(byte h, byte l):

Η συνάρτηση αυτή απλά παίρνει το ‘high’ (msb) byte και το ‘low’ (lsb) ενός δεδομένου που έχουμε ‘σπάσει’ σε 2 bytes και ανασυνθέτει τον ακέραιο αριθμό που είχαμε ‘σπάσει’.

Συνάρτηση get_echo(String requestCode,int sPort,int cPort,int time_mins):

Σκοπός της συνάρτησης, είναι να πάρει λάβει τα πακέτα echo από το server, να κάνει την επεξεργασία που επιθυμούμε και να βγάλει τα αντίστοιχα output αρχεία.

Παίρνει ως ορίσματα το request code για τα echo (with/without delay), το server listening port, το client listening port και το χρόνο για τον οποίο θα λαμβάνουμε πακέτα.

Αρχικά (σειρά 62) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους και στη συνέχεια ορίζει τις μεταβλητές που θα χρειαστούν για να τρέξει η συνάρτηση σωστά. Οι μεταβλητές δε θα σχολιαστούν, υπάρχουν, όμως, σχόλια στον κώδικα που τις εξηγούν σχετικά σύντομα.

Έπειτα (σειρά 111), τρέχει το loop στο οποίο γίνεται η λήψη πακέτων, ελέγχοντας σε κάθε loop το χρόνο που έχει περάσει, ώστε να τρέξει για το χρόνο που ζητήσαμε. Εδώ, είναι απαραίτητο να σχολιαστεί πως τρέχουμε για όσα λεπτά μας έδωσε ο χρήστης, αλλά προσθέτουμε και τα timeouts που έχουμε. Αυτό γίνεται για να λαμβάνει ο χρήστης αριθμό πακέτων ανάλογα με την ταχύτητα του server και όχι ανάλογα με το πόσο καλή είναι η σύνδεσή του, καθώς τα UDP πακέτα τείνουν να ‘χάνονται’ αρκετά συχνά στο ‘δρόμο’, μετά από παρατηρήσεις. Ταυτόχρονα, μέσα στο loop γίνεται και η αποθήκευση του χρόνου απόκρισης του server για κάθε πακέτο σε ένα arraylist.

Μετά τη λήψη (σειρά 137), ακολουθούν οι υπολογισμοί για τη ρυθμαπόδοση τα τελευταία 32 δευτερόλεπτα (για κάθε δευτερόλεπτο μετά τα 32sec λήψης). Αυτό γίνεται με βάση τα αποθηκευμένα response time με μια μέθοδο κινητών δεικτών πάνω στο arraylist.

Μετά (σειρά 179), υπολογίζουμε την πιθανότητα εμφάνισης κάθε τιμής response time, βάσει των δεδομένων που είχαμε συλλέξει. Αυτό γίνεται κάνοντας sort το αρχικό μας arraylist και μετρώντας πόσες φορές υπάρχει το κάθε στοιχείο.

Ακολουθούν (σειρά 202) οι υπολογισμοί των σηναρτήσεων SRTT, RTO και SRTTVAR, με μια απλή προγραμματιστική υλοποίηση αυτών που δίνονται από τις σημειώσεις του μαθήματος στη σελίδα 60, αλλά και από τη σελίδα https://tools.ietf.org/html/rfc2988 .

Έπειτα (σειρα 235), γίνεται ο υπολογισμός της μέσης τιμής και της διασποράς των response time των πακέτων που λάβαμε.

Τέλος (σειρά 264), μέσω της συνάρτησης stringToTextFile(), γίνεται output αρχείων txt με όλα τα δεδομένα που λάβαμε και υπολογίσαμε. Η δίακριση μεταξύ των κωδικών EXXXX και E0000 γίνεται ώστε να δώσουμε διαφορετικό όνομα στα αρχεία, αν πρόκειται για πακέτα με καθυστέρηση απο το server ή χωρίς.

Συνάρτηση get_Temp(String requestCode,int sPort,int cPort):

Η συνάρτηση αυτή έχει ως σκοπό να πάρει τη θερμοκρασία απο το σταθμό Τ00 (καθώς οι άλλοι δε λειτουργούν) και να βγάλει ως output το αρχείο με τη θερμοκρασία του.

Παίρνει ως ορίσματα το request code για τα θερμοκρασία, το server listening port και το client listening port.

Αρχικά (σειρά 292) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους.

Έπειτα (σειρές 306-307), γίνεται η αποστολή του request και η λήψη του.

Τέλος (σειρά 310), γίνεται το output των δεδομένων σε αρχείο txt.

Αντικείμενο MusicSamples:

Το αντικείμενο αυτό έχει ως μοναδικό σκοπό τη διευκόλυνση της επιστροφής των πινάκων των δειγμάτων και διαφορών ήχου και της μέσης τιμής και βήματος από τις συναρτήσεις DPCM_decoder και AQ_DPCM_decoder στις συναρτήσεις get_dpcm_audio και

get_aq_dpcm_audio. Με άλλα λόγια, ο σκοπός του είναι μόνο οργανωτικός και δεν έχει κάποια άλλη πρακτική αξία.

Συνάρτηση_get_dpcm_audio(String requestCode,int sPort,int cPort,int sound_packets):

Η συνάρτηση αυτή έχει ώς σκοπό τη λήψη των πακέτων, το output των δειγμάτων και των διαφορών ήχου, καθώς και την αναπαραγωγή του ήχου που λάβαμε.

Παίρνει ως ορίσματα το request code για το DPCM audio, το server listening port, το client listening port και τον αριθμό των πακέτων που θέλουμε να λάβουμε.

Αρχικά (σειρά 324) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους και στη συνέχεια ορίζει τις μεταβλητές που θα χρειαστούν για να τρέξει η συνάρτηση σωστά.

Μετά (σειρά 350), στέλνουμε το αίτημα του DPCM audio στο server και ξεκινάμε το loop για τη λήψη των πακέτων. Κάθε πακέτο χρειάζεται αποκωδικοποίηση, το οποίο γίνεται μέσω της συνάρτησης DPCM_decoder, η οποία επιστρέφει ένα αντικείμενο MusicSamples, που μας επιτρέπει να έχουμε πρόσβαση στους αποκωδικοποιημένους πίνακες. Τα δείγματα μπαίνουν στη σωστή θέση στον audio buffer και τροποποιούνται οι μεταβλητές που θα χρειαστεί να κάνουμε output.

Στη συνέχεια (σειρες 376 – 377), γίνεται το output των αρχείων txt που περιέχουν τα δεδομένα μας.

Τέλος (σειρές 385-391), γίνεται η αναπαραγωγή του ήχου που λάβαμε.

Συνάρτηση_DPCM_decoder(byte[] dpcm_bytes, int mean, int step):

Η συνάρτηση αυτή έχει ώς σκοπό την αποκωδικοίηση ενός πακέτου που είναι κωδικοποιημένο κατά DPCM και την επιστροφή της αποκωδικοποίησης του μέσω του αντικειμένου MusicSamples.

Αρχικά (σειρά 396), ορίζουμε τις μεταβλητές που θα χρειασούμε:

- int mean → μέσος όρος τιμών δειγμάτων
- int step → βήμα
- byte[] buffer_out → πίνακας που θα περιέχει τις τελικές τιμές των δειγμάτων μετά την αποκωδικοποίηση
- int[] deltas → πίνακας που θα περιέχει τις τιμές των διαφορών
- int MSnib → Most significant Nibble ενός byte
- int LSNib → Least significant Nibble ενός byte
- int current_byte → το byte που επεξεργαζόμαστε στο loop
- int index → δείκτης που δείχνει που να βάλουμε το κάθε στοιχείο της διαφοράς

Έπειτα (σειρά 408), στο loop γίνεται η επεξεργασία του κάθε byte ώστε να γεμίσουμε τον πίνακα των διαφορών με τις διαφορές των δειγμάτων. Η επεξεργασία γίνεται με bitwise operators και σύμφωνα με τη μαθηματική περιγραφή της εκφώνησης.

Μετά (σειρά 422), εφόσον γνωρίζουμε ότι ο μέσος όρος των δειγμάτων είναι 0, με μαθηματικά μπορούμε να υπολογίζουμε και το πρώτο δείγμα. *(Βέβαια, η προσέγγιση είναι θεωρητική και δεν βρέθηκε σε βιβλίο, εκφώνηση ή στο διαδίκτυο. Η εξήγηση είναι περιττή για κάτι τόσο μικρό και κάπως μακροσκελής για να μπει στην εργασία, αλλά σε περίπτωση που απαιτηθεί, θα δωθεί.)*

Στη συνέχεια (σειρά 433), με ένα loop κάνουμε αποκωδικοποίηση των διαφορών σε δείγματα και τα βάζουμε στον πίνακα buffer_out.

Τέλος (σειρά 437), δημιουργούμε το αντικείμενο MusicSamples με την κατάλληλη αρχικοποίηση, ώστε να μπορέσει να ‘μεταφέρει’ τις διαφορές και τα δείγματα εκτός της συνάρτησης και το επιστρέφουμε σαν αποτέλεσμα της συνάρτησης.

Συνάρτηση_get_aq_dpcm_audio(String requestCode,int sPort,int cPort,int sound_packets):

Η συνάρτηση αυτή έχει ώς σκοπό τη λήψη των πακέτων, το output των δειγμάτων και των διαφορών ήχου, καθώς και την αναπαραγωγή του ήχου που λάβαμε.

Παίρνει ως ορίσματα το request code για το AQ_DPCM audio, το server listening port, το client listening port και τον αριθμό των πακέτων που θέλουμε να λάβουμε.

Αρχικά (σειρά 445) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους και στη συνέχεια ορίζει τις μεταβλητές που θα χρειαστούν για να τρέξει η συνάρτηση σωστά.

Μετά (σειρά 475), στέλνουμε το αίτημα του AQ-DPCM audio στο server και ξεκινάμε το loop για τη λήψη των πακέτων. Κάθε πακέτο χρειάζεται αποκωδικοποίηση, το οποίο γίνεται μέσω της συνάρτησης AQ_DPCM_decoder, η οποία επιστρέφει ένα αντικείμενο MusicSamples, που μας επιτρέπει να έχουμε πρόσβαση στους αποκωδικοποιημένους πίνακες και μεταβλητές. Τα δείγματα μπαίνουν στη σωστή θέση στον audio buffer και τροποποιούνται οι μεταβλητές που θα χρειαστεί να κάνουμε output.

Στη συνέχεια (σειρες 506-509), γίνεται το output των αρχείων txt που περιέχουν τα δεδομένα μας.

Τέλος (σειρές 517-523), γίνεται η αναπαραγωγή του ήχου που λάβαμε.

Συνάρτηση_AQ_DPCM_decoder(byte[] dpcm_bytes, int mean, int step):

Η συνάρτηση αυτή έχει ώς σκοπό την αποκωδικοίηση ενός πακέτου που είναι κωδικοποιημένο κατά DPCM και την επιστροφή της αποκωδικοποίησης του μέσω του αντικειμένου MusicSamples.

Αρχικά (σειρά 528), ορίζουμε τις μεταβλητές που θα χρειασούμε:

- int mean → μέσος όρος τιμών δειγμάτων
- int step → βήμα

- byte[] sound_bytes → περιέχει τα bytes από το πακέτο πυο αφορούν αποκλειστικά τις διαφορές των δειγμάτων
- byte[] buffer_out → πίνακας που θα περιέχει τις τελικές τιμές των δειγμάτων μετά την αποκωδικοποίηση
- byte[] delta → πίνακας που θα περιέχει τις τιμές των διαφορών
- int MSnib → Most significant Nibble ενός byte
- int LSNib → Least significant Nibble ενός byte
- int current_byte → το byte που επεξεργαζόμαστε στο loop

Έπειτα (σειρά 544), στο loop γίνεται η επεξεργασία του κάθε byte ώστε να γεμίσουμε τον πίνακα των διαφορών με τις διαφορές των δειγμάτων. Η επεξεργασία γίνεται με bitwise operators και σύμφωνα με τη μαθηματική περιγραφή της εκφώνησης. Σημειώνεται ότι πλέον η κάθε διαφορά που προκύπτει είναι μεγαλύτερη από ένα byte, για αυτό και χρειάζεται 2 διαδοχικές θέσεις για να αποθηκευτεί σε έναν πίνακα τύπου byte.

Στη συνέχεια (σειρά 561), με ένα loop κάνουμε αποκωδικοποίηση των διαφορών σε δείγματα, τα οποία επίσης έχουν μέγεθος 2 byte και έτσι πρέπει να τα ‘σπάσουμε’ και να τα βάλουμε σε 2 διαδοχικές θέσεις στον πίνακα buffer_out.

Τέλος (σειρά 571), δημιουργούμε το αντικείμενο MusicSamples με την κατάλληλη αρχικοποίηση, ώστε να μπορέσει να ‘μεταφέρει’ τις διαφορές, τα δείγματα, τη μέση τιμή και το βήμα εκτός της συνάρτησης και το επιστρέφουμε σαν αποτέλεσμα της συνάρτησης.

Συνάρτηση get_image(String requestCode,int sPort,int cPort):

Σκοπός της συνάρτησης είναι να κάνει output μια εικόνα jpeg από την κάμερα με τον αντίστοιχο κωδικό.

Παίρνει ως ορίσματα το request code για τα θερμοκρασία, το server listening port και το client listening port.

Αρχικά (σειρά 583) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους.

Στη συνέχεια (σειρά 587), ορίζει ένα αντικείμενο OutputStream για την εξαγωγή της εικόνας.

Μετά (σειρα 608), στέλνει το request για την εικόνα που θέλουμε (το οποίο διαφοροποιείται ανάλογα με την κάμερα από την οποία θέλουμε να κάνουμε λήψη) και στη συνέχεια (σειρά 615), με ένα while loop ‘ταίζει’ στο OutputStream τα δεδομένα που μας έρχονται, μέχρι να πάψουν να έρχονται δεδομένα. Έτσι γίνεται η εξαγωγή της εικόνας.

Συνάρτηση get_ithakicopter_telemetry():

Η συνάρτηση αυτή έχει ως σκοπό τη λήψη των πακέτων από την εφαρμογή του Ithakicopter.

Επειδή το client listening port είναι fixed και δε χρειάζεται να στείλουμε κωδικό, δε χρειάζονται ορίσματα.

Αρχικά (σειρά 637) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους.

Στη συνέχεια, ορίζει τις μεταβλητές που θα χρειαστούμε για να κάνουμε λήψη και επεξεργασία των πακέτων.

Έπειτα (σειρά 649), έχουμε το loop λήψης των πακέτων για 200 iterations, μέσα στο οποίο γίνεται και η επεξεργασία τους για το output.

Τέλος (σειρά 672), γίνεται το output του αρχείου txt με τα δεδομένα που πήραμε.

Συνάρτηση get_OBD(String requestCode,int sPort,int cPort,int time_mins):

Η συνάρτηση αυτή έχει ως σκοπό τη λήψη και το output δεδομένων σχετικά με τη λειτουργία του οχήματος OBD.

Παίρνει ως ορίσματα το request code για το OBD-II, το server listening port, το client listening port και το χρόνο για τον οποίο θα λαμβάνουμε πακέτα.

Αρχικά (σειρά 677) , ξεκινάει τα αντικείμενα DatagramSocket και DatagramPacket με τις σωστές παραμέτρους και στη συνέχεια ορίζει τις μεταβλητές που θα χρειαστούν για να τρέξει η συνάρτηση σωστά. Οι μεταβλητές δε θα σχολιαστούν, διότι τα ονόματά τους είναι αρκετά επεξηγηματικά από μόνα τους.

Έπειτα (σειρά 708), τρέχει το loop μέσα στο οποίο γίνεται η αποστολή του κάθε ένα κωδικού που χρειάζεται για τη λήψη της κάθε μεταβλητής. Μετά τη λήψη μίας παραμέτρου, γίνεται η σωστή επεξεργασία για να λάβουμε τη χρήσιμη, για εμάς πληροφορία, σύμφωνα με τους τύπους που δώθηκαν στην εκφώνηση.

Τέλος (σειρά 789), γίνεται το output του αρχείου txt με τα δεδομένα μας.

Συνάρτηση sleep(int ms):

Η συνάρτηση αυτή κάνει ‘pause’ το πρόγραμμά μας για τη χρονική διάρκεια που ορίσαμε. Κύριος σκοπός της ήταν να βοηθήσει στο debugging του προγράμματος.

Συναρτήσεις start/end_xxxx_graphics():

Βρίσκονται στο τέλος του προγράμματος και δεν έχουν λειτουργικό σκοπό, είναι μόνο για να γίνει ομορφότερο το πρόγραμμα.