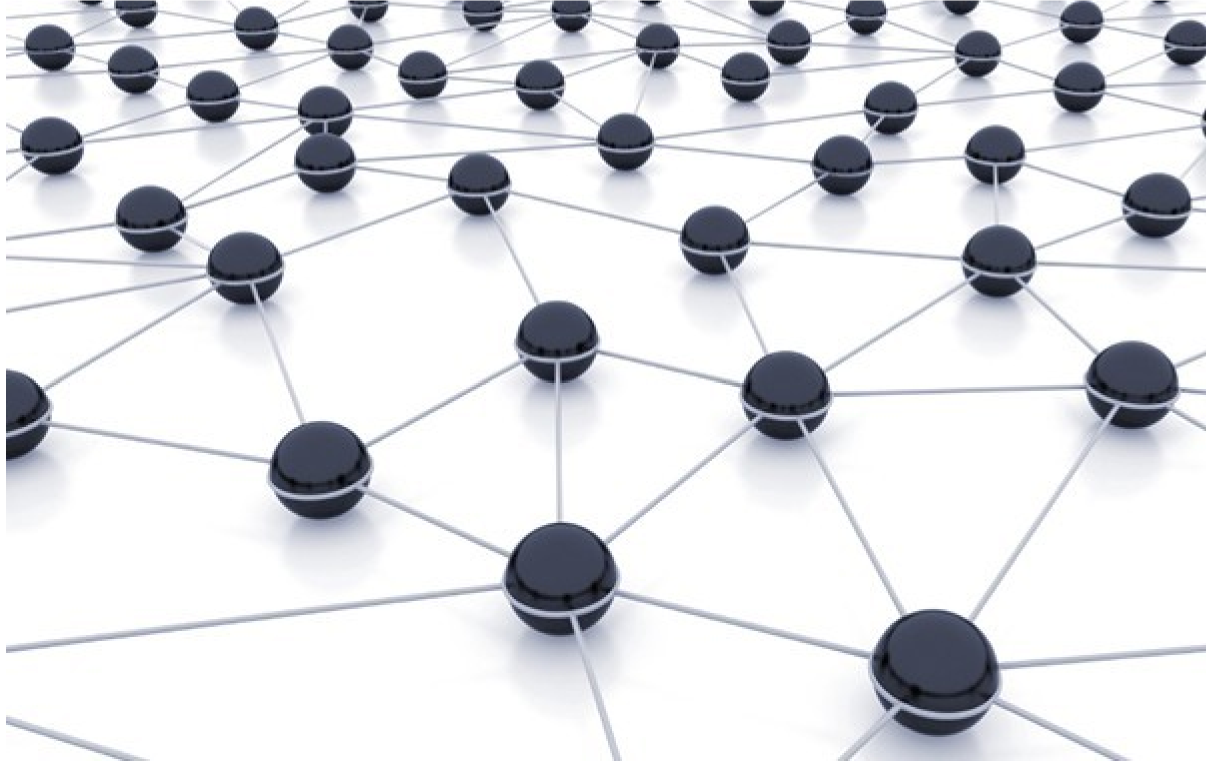
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ



rtes_final

ΤΕΛΙΚΗ ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ:

ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

ΧΡΗΣΤΟΣ ΧΟΥΤΟΥΡΙΔΗΣ ΑΕΜ:8997
cchoutou@ece.auth.gr
+30 697 8894932

1. Στόχος

Στόχος της παρούσας εργασίας είναι η δημιουργία ενός mesh network με ενσωματωμένες συσκευές και η αυτοματοποιημένη ανταλλαγή μηνυμάτων μεταξύ τους. Τα μηνύματα δημιουργούνται από τις συσκευές και ταξιδεύουν μέσα στο δίκτυο μέχρι να φτάσουν στον τελικό προορισμό τους. Για το λόγο αυτό όταν μια συσκευή παράγει ένα μήνυμα για μία άλλη το στέλνει σε όλες τις συσκευές που είναι κοντά της με την “ευχή” ότι και αυτές θα το προωθήσουν στις συσκευές που είναι κοντά τους και έτσι κάποια στιγμή θα φτάσει στον προορισμό του.

2. Παραδοτέα

Η υλοποίηση της εφαρμογής είναι σε C. Τα επισυναπτόμενα αποτελούνται από:

- **report_rtes_Choutouridis_8997.pdf**: Το παρόν.
- Ένα **Makefile** για την μεταγλώττιση του κώδικα και το ανέβασμα της εφαρμογής στο Pi.
- Ένας κατάλογος **src/** με τον κώδικα της εφαρμογής. *Σημείωση: Απαιτείται εγκατεστημένος cross compiler για το raspberry pi zero.*
- Ένας υποκατάλογος **doc/html/** με **την τεκμηρίωση του κώδικα** όπως αυτή έχει παραχθεί από τα σχόλια με το εργαλείο **doxygen**. Το αρχείο ρυθμίσεων του doxygen είναι στον root με το όνομα Doxyfile. Η πλοήγηση στην τεκμηρίωση μπορεί να γίνει ανοίγοντας το αρχείο ***doc/html/index.html***

3. Σχεδίαση

Καθώς η επικοινωνία θα πρέπει να γίνεται με το πρωτόκολλο TCP/IP η εφαρμογή πρέπει να υλοποιήσει sockets. Ακόμα θα πρέπει να είναι σε θέση να λαμβάνει συνδέσεις μεσώ socket στην προεπιλεγμένη πόρτα ενώ ταυτόχρονα να μπορεί να εκτελεί τις υπόλοιπες λειτουργίες. Αυτό θα μπορούσε να υλοποιηθεί με χρήση socket σε non-blocking mode, αλλά κάτι τέτοιο θα αύξανε την πολυπλοκότητα του προγράμματος. Για το σκοπό αυτό αποφασίσαμε να χρησιμοποιήσουμε blocking sockets και να χωρίσουμε την λειτουργία του προγράμματος σε περισσότερα threads. Έτσι το πρόγραμμα χωρίστηκε:

- Στο **listener**, ο οποίος αναλαμβάνει να δέχεται μηνύματα.
  Αυτός βρίσκεται συνεχώς σε **blobking mode** αναμένοντας καινούριες συνδέσεις από τις υπόλοιπες συσκευές. Σε κάθε καινούρια σύνδεση κάνει ανάγνωση του μηνύματος , έλεγχο της ορθότητάς του, έλεγχο διπλοτύπων, αποθήκευση στην ζητηθείσα λίστα και έπειτα επανέρχεται ξανά σε κατάσταση αναμονής για το επόμενο μήνυμα.
- Στον **seeker**, ο οποίος αναλαμβάνει να αναζητά τις κοντινές συσκευές.
  Αυτός κατά βάση βρίσκεται σε κατάσταση **sleep** και περιοδικά ξυπνά και προσπαθεί να λάβει απάντηση με την εντολή **ping** από όλες τις συσκευές της λίστας. Για τις συσκευές που απαντούν καταγράφει τόσο το γεγονός ότι είναι κοντά, αλλά και τις χρονικές στιγμές για τις οποίες αυτό συμβαίνει
- Στον **client**, ο οποίος αναλαμβάνει να δημιουργεί και να αποστέλλει τα μηνύματα. Τόσο αυτά που δημιούργησε όσο και αυτά που δέχτηκε αλλά χρειάζεται να επαναπροωθήσει σε περίπτωση που δεν έχουν αυτόν ως αποδέκτη.

Να σημειώσουμε εδώ πως η παραπάνω παραλληλία θα μπορούσε να γίνει και με διαφορετικά processes. Η επιλογή των **POSIX threads** βασίστηκε στο γεγονός ότι είναι ελαφρύτερα και κυρίως στο ότι όλα έχουν πρόσβαση στα ίδια δεδομένα. Έτσι εξασφαλίζουμε την εύκολη κοινή πρόσβαση στις δομές δεδομένων του προγράμματος.

Ακόμα θα πρέπει να πούμε πως τον περισσότερο χρόνο το λειτουργικό σύστημα κρατά και τα τρία thread μπλοκαρισμένα. Τα δύο γιατί το ζητούν μέσω της ***sleep()*** και το τρίτο γιατί αναμένει σύνδεση από κάποια συσκευή. Με αυτό τον τρόπο επιτυγχάνουμε χαμηλή κατανάλωση ενέργειας, κάτι που είναι σημαντικό για ένα ενσωματωμένο σύστημα.

4. Υλοποίηση

Η υλοποίηση του προγράμματος έχει γίνει σε διαφορετικά αρχεία:

- **core.c/h**. Περιέχει βασικές λειτουργίες πάνω στα δεδομένα του προγράμματος καθώς και βοηθητικά εργαλεία.
- **client.c/h**. Περιέχει τις συναρτήσεις που αναλαμβάνουν την αναζήτηση συσκευών, την δημιουργία και την αποστολή μηνυμάτων.
- **Listener.c/h**. Περιέχει τις συναρτήσεις που αναλαμβάνουν την αποδοχή μηνυμάτων, τον έλεγχο της ορθότητας τους και την αποθήκευση.
- **msg_impl.h**. Περιέχει τις δηλώσεις όλων των δομών δεδομένων που χρησιμοποιούνται από το πρόγραμμα
- **main.c**. Περιέχει την συνάρτηση main καθώς και την λειτουργία ανάγνωσης των παραμέτρων εισόδου.

4.1 Δομές δεδομένων

Πριν ξεκινήσουμε να αναλύουμε τις λεπτομέρειες των λειτουργιών του προγράμματός καλό θα ήταν να περιγράψουμε τις δομές αναπαράστασης των δεδομένων στο πρόγραμμά μας. Κατά τη σχεδίαση έγινε μια προσπάθεια να προσομοιάσουμε ελαφρώς την αντικειμενοστραφή λογική, τουλάχιστον για τους τύπους δεδομένων που χρησιμοποιούμε. Γιαυτό και πολλές συναρτήσεις που αφορούν δεδομένα παίρνουν ως όρισμα ένα δείκτη σε αντικείμενο του συγκεκριμένου τύπου δεδομένων. Αυτός ο δείκτης είναι παρόμοιος με τον **this** pointer άλλων γλωσσών προγραμματισμού.

<u>Δομή devList_t</u>

Σε αυτό τον τύπο αποθηκεύουμε το ΑΕΜ της κάθε συσκευής, καθώς και επιπρόσθετες πληροφορίες όπως μια σημαία για το αν η συσκευή είναι κοντά (δηλαδή αν απαντάει στο ping) και ποια ήταν η πρώτη και η τελευταία χρονική στιγμή που βρέθηκε η συσκευή κοντά. Στην εφαρμογή δημιουργούμε ένα πίνακα από τέτοια αντικείμενα με μέγεθος όσο η δοθείσα λίστα με τα ΑΕΜ και χρησιμοποιούμε αυτόν για λίστα. Συνδεόμενες λειτουργίες με αυτό τον τύπο είναι:

- **devList_init (devList_t* devList):** Που λειτουργεί "ως constructor" και αρχικοποιεί τη λίστα
- **devList_getIter (devAEM_t dev):** Που αναζητάει μέσα στη λίστα και επιστρέφει έναν iterator (δηλαδή τη θέση) του ΑΕΜ στη λίστα.

<u>Δομή cMsg_t</u>

Αυτός ο τύπος αποτελεί μια αναπαράσταση του μηνύματος. Περιέχει τα τέσσερα πεδία του μηνύματος όπως αυτά αναφέρονται στην εκφώνηση, δηλαδή τον **αποστολέα (from)**, τον **αποδέκτη (to)**, το **χρονικό στιγμιότυπο (ts)** και το **κείμενο (text)**. Βασικές συνδεόμενες λειτουργίες με αυτό τον τύπο είναι:

- **cMsg_make (cMsg_t* msg):** Που κατασκευάζει ένα μήνυμα για ένα τυχαίο ΑΕΜ από την λίστα
- **cMsg_parse (cMsg_t* cMsg, char_t* rawMsg, size_t size):** Που κατασκευάζει ένα μήνυμα από ένα εισερχόμενο κείμενο.
- **cMsg_serialize (cMsg_t* msg, char_t* buffer):** Που μετατρέπει ένα μήνυμα σε κείμενο για αποστολή.
- **cMsg_equal (cMsg_t* m1, cMsg_t* m2):** Που ελέγχει ισότητα μεταξύ δύο μηνυμάτων.

<u>Δομή msg_t</u>
Αυτός ο τύπος αποτελεί μια ποιο λειτουργική αναπαράσταση του μηνύματος καθώς περιέχει:

- **cMsg:** Το μήνυμα
- **sender:** Η συσκευή που έστειλε το μήνυμα (όχι αυτή που το δημιούργησε).
- **recipients[]**: Έναν πίνακας μεγέθους όσο και η δοθείσα λίστα με τα AEM που αποτελείται από σημαίες οι οποίες εκφράζουν αν το AEM έχει το εν λόγο μήνυμα. Η σύνδεση με το AEM γίνεται σε ευθεία αναλογία με τον πίνακα **devList[].** Αυτό σημαίνει πχ ότι η σημαία *recipients[3]* αναφέρεται στο AEM *devList[3].dev.*

Με αυτόν το τύπο δημιουργούμε τον msgList_t που είναι και η καρδιά της εφαρμογής μας.

<u>Δομή msgList_t</u>
Αυτός ο τύπος αποτελείται από

- **m[]**: Ένα πίνακα 2000 θέσεων τύπου **msg_t** τον οποίο χρησιμοποιούμε σαν ring buffer
- **last, first**: Δύο iterators που δείχνουν στο πρώτο και στο τελευταίο στοιχείο του ring buffer.
- **size**: Το τρέχον μέγεθός της δομής, δηλαδή τα αποθηκευμένα μηνύματά μέχρι τη δεδομένη χρονική στιγμή.

Αυτή η δομή δημιουργεί στην ουσία μία αναπαράσταση δύο διαστάσεων.

1. Κινούμενοι στον **m[*]** προσπελάζουμε τα διαφορετικά μηνύματα. Αυτή είναι κατά μία έννοια η ***χρονική διάσταση***[1]
2. Κινούμενοι στον **m[].recipients[*]** προσπελάζουμε τους διαφορετικούς αποδέκτες του κάθε μηνύματος. Αυτή η διάσταση είναι κοινή με αυτή στο **devList** και κατά μία έννοια είναι η ***χωρική διάσταση***.

Βασικές συνδεόμενες λειτουργίες με αυτό τον τύπο είναι:

- **msgList_init (msgList_t* msgList):** Που κατασκευάζει μια λίστα μηνυμάτων.
- **msgList_preInc (mIter_t* it):** Πρώτα αυξάνει και μετά επιστρέφει ένα iterator στη διάσταση **m[]**
- **msgList_preDec (mIter_t* it):** Πρώτα μειώνει και μετά επιστρέφει ένα iterator στη διάσταση **m[]**
- **msgList_begin (msgList_t* this):** Επιστρέφει ένα iterator στο πρώτο στοιχείο του ring buffer m[].
- **msgList_last (msgList_t* this):** Επιστρέφει ένα iterator στο τελευταίο στοιχείο του ring buffer m[].
- **msgList_size (msgList_t* this):** Επιστρέφει το τρέχον μέγεθος του m[], δηλαδή τον αριθμό μηνυμάτων που είναι αποθηκευμένα στον m[].
- **msgList_find (msgList_t* this, msg_t* msg):** Αναζητά ένα μήνυμα στον m[]
- **msgList_add (msgList_t* this, msg_t* msg):** Προσθέτει ένα μήνυμα στον ring buffer m[].

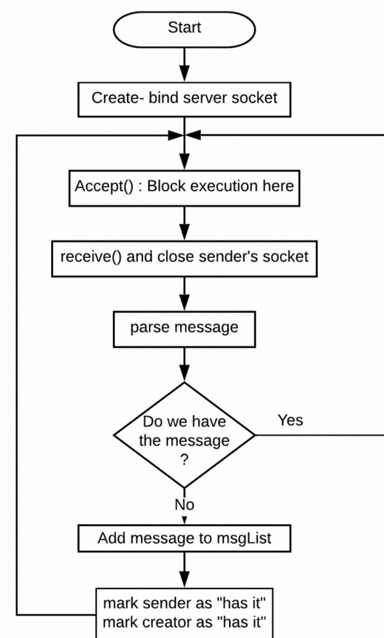
Οι παραπάνω συναρτήσεις λειτουργούν ως **αφαίρεση για την δημιουργία ενός κυκλικού buffer.** Ένας εξοικειωμένος με την C++ αναγνώστης ίσως διαπιστώσει ότι λείπει η συνάρτηση ***end()*** η οποία δείχνει μία θέση μετά το τελευταίο στοιχείο. Δυστυχώς όταν ο buffer φτάσει στο μέγιστό του μέγεθος τότε αυτή η θέση είναι η θέση του πρώτου στοιχείου. Γιαυτό το λόγο η συνάρτηση *end()* δεν υλοποιήθηκε και όλες οι προσπελάσεις στη διάσταση m[] γίνονται εκκινώντας από το *begin()* ή το *last()* και εκτελώντας *size()* επαναλήψεις.
Για παράδειγμα:

```
mIter_t it = msgList_begin (&msgList);    // get a message iterator
size_t size= msgList_size(&msgList);      // get current msgList size
for (size_t i=0 ; i<size ; ++i, msgList_preInc (&it)) { // Increase iterator
   // use msgList[it]
}
```

[1] Το γεγονός ότι ο m είναι κυκλικός buffer καθιστά και τον χρόνο "κυκλικό" σε αυτό το περιεχόμενο!!

4.2 Listener

Με υλοποιημένα όλα τα παραπάνω, η δημιουργία ενός listener ο οποίος θα δεχόταν ένα μήνυμα θα έψαχνε να δει αν υπάρχει στη λίστα, θα το πρόσθετε αν όχι και θα σημείωνε πως ο αποστολέας και ο δημιουργός του μηνύματος, ήδη έχουν το μήνυμα, ήταν πολύ εύκολη υπόθεση. Το μόνο κομμάτι που λείπει είναι η δημιουργία ενός socket, η σύνδεσή του με την πόρτα που υποδεικνύεται από την εκφώνηση και μετά η συνεχής αποδοχή συνδέσεων και δεδομένων από τις εξωτερικές συσκευές. Αυτή ακριβώς είναι και η λειτουργία του η οποία υλοποιείται από τις συναρτήσεις ***listen_handler ()*** και ***listener().*** Στο διάγραμμα δίπλα φαίνεται σε απλοποιημένη μορφή η λειτουργία αυτών των συναρτήσεων. Να σημειώσουμε εδώ πως οι συναρτήσεις αυτές αποκτούν πρόσβαση στις δομές **devList** και **msgList** οι οποίες είναι πώροι που διαμοιράζονται με τα άλλα δύο thread**.** Για το λόγο, για την πρόσβασή σε αυτά τα δεδομένα, γίνεται χρήση **pthread_mutex**.



4.3 Seeker

Η λειτουργία του seeker είναι αρκετά απλή. Βρίσκεται μόνιμα σε ένα ατέρμονα βρόχο όπου αρχικά ζητά από το λειτουργικό μπλοκάρει την εκτέλεση του thread για ένα μικρό χρονικό διάστημα. Έπειτα εκτελεί μέσω της συνάρτησης **system()** την εντολή **ping** για κάθε μία συσκευή από την λίστα με τα ΑΕΜ. Αν πάρει απάντηση ενημερώνει την devList[] ότι η εν λόγο συσκευή είναι κοντά. Ακόμα στην περίπτωση που λάβει απάντηση σημειώνει και τη χρονική στιγμή σε δύο μέλη της devList. Στην πρώτη απάντηση στο begin και σε κάθε απάντηση στο last. Με αυτό τον τρόπο γνωρίζουμε πότε εντοπίστηκε πρώτη φορά μια συσκευή και πότε ήταν η τελευταία φορά που είχαμε επικοινωνία με αυτή. Να σημειώσουμε και εδώ πως η πρόσβαση στο devList προστατεύεται από τη χρήση **pthread_mutex**.

4.4 Client

Η λειτουργία του client είναι αντίστοιχη. Επίσης σε ένα ατέρμονα βρόχο ζητά από το λειτουργικό να μπλοκάρει την εκτέλεση του thread. Αυτή τη φορά όμως για τυχαίο χρονικό στο διάστημα του 1 με 5 λεπτών. Έπειτα δημιουργεί ένα καινούργιο μήνυμα για ένα τυχαίο ΑΕΜ από τη λίστα και το προσθέτει στη msgList. Στη συνέχεια εκτελεί ένα διπλό βρόχο για κάθε μήνυμα στη **msgList** και για κάθε συσκευή στον πίνακα **recipients[].** Σε κάθε επανάληψη αν:

1. Η συσκευή έχει εντοπιστεί και είναι κοντά
2. Δεν έχει το μήνυμα και
3. Ο αποδέκτης του μηνύματος δεν είναι η δικιά μας συσκευή

τότε καλεί την *sendMsg()* για να στείλει το μήνυμα στη συσκευή. Αν η *sendMsg()* επιτύχει να στείλει το μήνυμα τότε σημειώνουμε τον αποδέκτη ως "έχει το μήνυμα".
Η ***sendMsg()*** ακολουθεί την αντίστοιχη λογική με την *listen()* με την διαφορά ότι κάνουμε χρήση non-blocking ***connect()*** με **timeout** καθώς και ***send()*** με timeout. Ο λόγος είναι η περίπτωση να συνδεθούμε σε μια συσκευή και πριν προλάβουμε να τελειώσουμε την επικοινωνία, η συσκευή να κλείσει ή να βγει εκτός εμβέλειας. Φυσικά και εδώ η πρόσβαση στο devList και στο msgList προστατεύεται από τη χρήση **pthread_mutex**.

5 Μετρήσεις

Ο στόχος της υλοποίησης ήταν η επικοινωνία των συσκευών, με σκοπό την ανταλλαγή μηνυμάτων και η εξαγωγή στατιστικών από μετρήσεις κατά τη λειτουργία. Ένα ιδανικό σενάριο λειτουργίας θα ήταν η δημιουργία ενός μεγάλου mesh δικτύου στο οποίο οι συσκευές δεν θα είχαν πρόσβαση σε όλες τις υπόλοιπες. Δυστυχώς για την παρούσα εργασία το raspberry μας πήρε μέρος σε 2 μετρήσεις αλλά και τις δύο φορές όλες οι συσκευές ήταν κοντά μεταξύ τους συνεχώς.

5.1 Στατιστικά

Τα στατιστικά που εξάγαμε και τα οποία στην εφαρμογή τηρούνται στη δομή **stats_t** αφορούν:

- **totalMsg:** Τον συνολικό αριθμό των μηνυμάτων που δημιούργησε ή έλαβε η συσκευή συμπεριλαμβανομένων και αυτών που ήταν διπλότυπα.
- **duplicateMsg**: Τον αριθμό των διπλοτύπων μηνυμάτων που έλαβε η συσκευή.
- **myMsg**: Τον αριθμό των μηνυμάτων που δημιούργησε η δική μας συσκευή.
- **forMeMsg**: Τον αριθμό των μηνυμάτων που είχαν στο πεδίο του αποδέκτη την δική μας συσκευή.
- **inDirectMsg**: Τον αριθμό των μηνυμάτων που είχαν στο πεδίο του αποδέκτη τη δική μας συσκευή και στάλθηκαν κατευθείαν σε εμάς από τον δημιουργό.
- **outDirectMsg**: Τον αριθμό μηνυμάτων που δημιούργησε η δική μας συσκευή για κάποιον αποδέκτη και στάλθηκαν απευθείας στον αποδέκτη.
- **avMsgSize**: Το μέσο μέγεθος του κειμένου που είχαν όλα τα μη διπλότυπα μηνύματα που πέρασαν από τη συσκευή.
- **avMsgTime**: Τον μέσο χρόνο που χρειάστηκαν τα μηνύματα που είχαν ως αποδέκτη τη δική μας συσκευή, από τη στιγμή που δημιουργήθηκαν μέχρι τη στιγμή που τα λάβαμε.

Να σημειώσουμε εδώ πως για το τελευταίο στατιστικό απαιτείται όλες οι συσκευές να είναι συγχρονισμένες. Αυτό το επιτύχαμε και στις δύο μετρήσεις εκτελώντας από τον υπολογιστή-host το:

```
#!/bin/bash
STAMP="$(date +'%d %b %Y %T %Z')"
ssh root@192.168.0.1 'date -s "'${STAMP}'"'
```

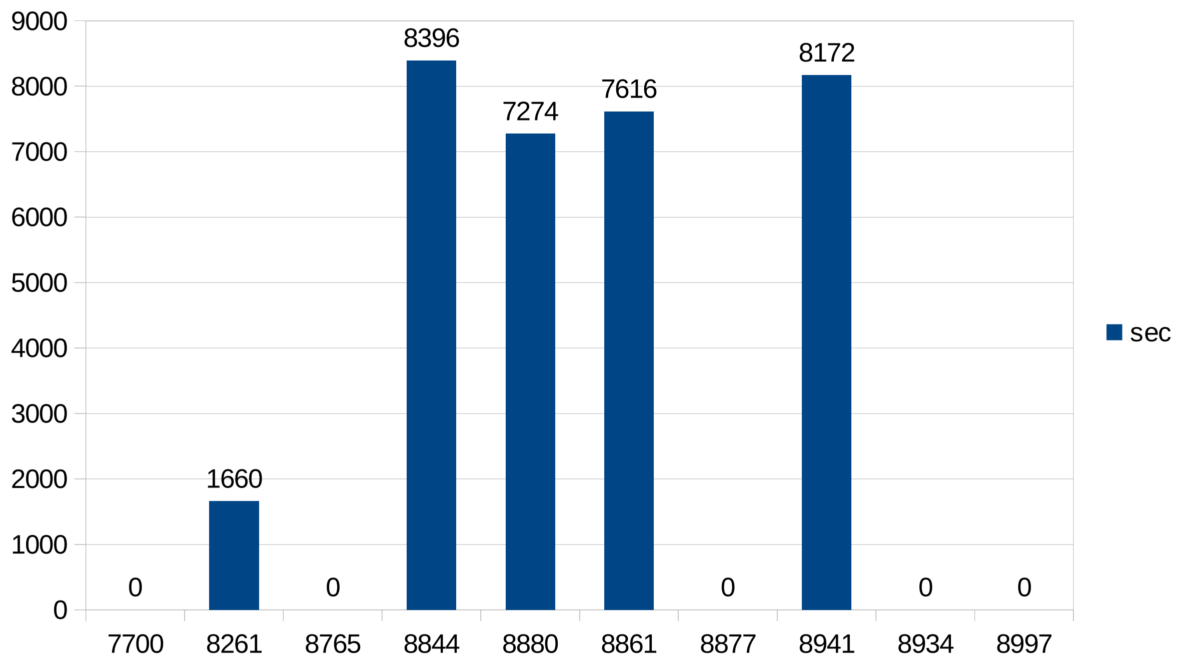
Στο παρακάτω διάγραμμα φαίνονται τα στατιστικά που εξάγαμε μετά από 2 ώρες και 20 λεπτά λειτουργίας.

5.2 Συσκευές-εμβέλεια

Κατά τη διάρκεια των μετρήσεων καταγράψαμε τις συσκευές που ήταν εντός εμβέλειας.

Συσκευή (AEM)	Χρόνος πρώτης εύρεσης	Χρόνος τελευταίας επικοινωνίας	Διάρκεια
7700	0	0	0
8261	1570210285	1570211945	1660
8765	0	0	0
8844	1570209288	1570217684	8396
8880	1570209288	1570216562	7274
8861	1570209288	1570216904	7616
8877	0	0	0
8941	1570209289	1570217461	8172
8934	0	0	0
8997	-	-	-

Στο παρακάτω διάγραμμα φαίνεται η διάρκεια κατά την οποία είχαμε επικοινωνία με της συσκευές



6 Συμπεράσματα

Μπορεί η παρούσα εφαρμογή να μην είναι η ιδανική, παρόλα αυτά ήταν ικανή να πάρει μέρος σε δύο measurement party. Η υλοποίησή της για την συσκευή raspberry pi ήταν σχετικά εύκολη, πράγμα που μας οδηγεί στο συμπέρασμα πως η εν λόγο πλατφόρμα είναι πολύ καλή για τέτοιου είδους εφαρμογές αφού συνδυάζει μικρό μέγεθος, χαμηλή κατανάλωση, μεγάλη επεξεργαστική ισχύς και αρκετή RAM μαζί με την ευελιξία της ασύρματης σύνδεσης.