

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

*ΕΝΣΩΜΑΤΩΜΕΝΑ
ΣΥΣΤΗΜΑΤΑ &
ΣΥΣΤΗΜΑΤΑ
ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ*
8ο ΕΞΑΜΗΝΟ

ΤΕΛΙΚΗ ΕΡΓΑΣΙΑ

ΕΙΣΗΓΗΤΗΣ: κ. Πιτσιάνης Ν.

Όνομα : ΘΑΝΑΣΗΣ ΧΑΡΙΣΟΥΔΗΣ

Α.Ε.Μ. : 9026

ΘΕΣΣΑΛΟΝΙΚΗ, 10 Οκτωβρίου 2019

Περιεχόμενα

1. Ζητούμενα / Στόχοι Εργασίας.....	3
2. Λογική Υλοποίησης.....	4
2.1. Πως δύναται η συσκευή να έλθει σε επικοινωνία με άλλες συσκευές.....	4
2.2. Τι γίνεται όταν δύο συσκευές συνδεθούν.....	5
2.3. Που αποθηκεύονται τα μηνύματα κατά τη διάρκεια ενός session.....	6
2.4. Αποφυγή Conflicts.....	8
3. Συνθήκες Υλοποίησης & Παραδοχές.....	9
3.1. Το setup.....	9
3.2. Η Λίστα των AEM.....	10
3.3. Συνθήκες Διεξαγωγής Sessions.....	10
3.3.1. Χώρος διεξαγωγής.....	10
3.3.2. Ημερομηνία & Χρόνος Διεξαγωγής.....	10
4. Αποτελέσματα.....	11
4.1. Συλλογή Αποτελεσμάτων.....	11
4.1.1. Συλλογή Στατιστικών.....	12
4.1.2. Καταγραφή Logs για την Δημιουργία Timeline.....	12
4.2. Παρουσίαση Αποτελεσμάτων.....	13
4.2.1. WebReport.....	13
4.3. Προβολή logs στο WebReport.....	14
4.3.1. Προβολή timeline ενός session.....	15
4.3.2. Προβολή stats του session.....	17
5. Ο κώδικας C.....	20
5.1. Οργάνωση(Clion) Project.....	20
5.2. Ο τύπος <i>Message</i>	20
5.3. Ο τύπος <i>InboxMessage</i>	21
5.4. Ο πίνακας (buffer) <i>Messages</i>	22
5.5. Ο πίνακας (buffer) <i>INBOX</i>	23
6. Παραδοτέα.....	24

1. Ζητούμενα / Στόχοι Εργασίας

Στην παρούσα εργασία ζητείται η δημιουργία εφαρμογής σε C η οποία να τρέχει σε ενσωματωμένες συσκευές (embedded systems) με σκοπό την επικοινωνία αυτών μέσω ενός ad-hoc δικτύου μεταξύ τους. Πρόκειται για ένα ασύρματο peer-to-peer ad-hoc δίκτυο στο οποίο δύο τυχαίες συσκευές – από εδώ και στο εξής θα γίνεται αναφορά σε αυτές ως **peers** – θα συνδέονται μεταξύ τους εφόσον βρίσκονται εντός εμβέλειας τα WiFi δίκτυά τους το ένα με το άλλο. Στόχος της επικοινωνίας είναι η μετάδοση μηνυμάτων από peer σε peer ώστε το μήνυμα να φτάσει στον τελικό του παραλήπτη ακόμη και εάν αυτός δεν είναι αρχικά εντός εμβέλειας με το δίκτυο του αποστολέα. Για την επίτευξη του στόχου αυτού, κάθε συσκευή όταν “συναντήσει” μια άλλη συσκευή “ανταλλάσσουν” όλα τα μηνύματα που “έχει η μία για την άλλη”. Τα εισαγωγικά εισάγονται καθώς αποφεύγεται η παράθεση τεχνικών λεπτομερειών στο συγκεκριμένο μέρος της αναφοράς – γίνεται εκτενής αναφορά σε αυτές σε επόμενα μέρη. Η όλη εκτέλεση γίνεται σε πραγματικό χρόνο και άρα κάθε συσκευή πρέπει σε πραγματικό χρόνο να διαχειρίζεται τη ροή μηνυμάτων μεταξύ αυτής και οποιασδήποτε άλλης με την οποία έλθει σε επικοινωνία.

2. Λογική Υλοποίησης

Η λογική της peer-to-peer επικοινωνίας που υλοποιήθηκε είναι κάθε συσκευή να είναι ένας peer ο οποίος τόσο θα ακούει για εισερχόμενες συνδέσεις από άλλες συσκευές όσο και θα αναζητά άλλες συσκευές για να συνδεθεί σε πραγματικό χρόνο. Ο πυρήνας της επικοινωνίας δεν μπορεί να είναι άλλος από τη χρήση (TCP) **sockets** στη C. Έτσι, για την υλοποίηση της λογικής του peer, κάθε συσκευή λειτουργεί ως server σε ένα νήμα (thread) ενώ λειτουργεί ως client σε ένα άλλο thread. Ασχέτως από το πως θα ανοίξει το (stream - TCP) socket μεταξύ των δύο συσκευών η κάθε συσκευή θα μεταδώσει σειριακά στην άλλη όλο τον buffer ενδιάμεσων μηνυμάτων της ενώ θα λάβει τον αντίστοιχο buffer της άλλης συσκευής (αυτή είναι μια χονδροειδής προσέγγιση· στην υλοποίηση σε C γίνονται έλεγχοι πριν την αποστολή του κάθε μηνύματος όπως και πριν την αποθήκευση του κάθε εισερχόμενου μηνύματος).

2.1. Πως δύναται η συσκευή να έλθει σε επικοινωνία με άλλες συσκευές

Για το σκοπό αυτό, ο κώδικας που υλοποιήθηκε εκτελεί τα ακόλουθα προκειμένου να καταστήσει δυνατή την επικοινωνία μεταξύ της συσκευής - peer - στην οποία εκτελείται και άλλων συσκευών - peers - του ad-hoc δικτύου:

- κατά την εκκίνηση, στη *main.c*, ανοίγει ένα thread το οποίο παράγει μηνύματα, το *producer_worker()*, με τυχαία καθυστέρηση στο διάστημα [60, 300] sec ή [1, 5] min. Η παραγωγή γίνεται σε έναν αέναο βρόχο και σταματάει μόνο όταν έρθει το σήμα *alarm* στο main thread (γίνεται χρήση της *pthread_cancel* για το σκοπό αυτό).

- επίσης κατά την εκκίνηση της *main.c*, ανοίγει ένα άλλο thread, το *polling_worker()* βασική εργασία του οποίου είναι αναζήτηση σε μορφή polling άλλων server sockets με IP διεύθυνση που σχηματίζεται από μία προκαθορισμένη λίστα με AEM (όπως ζητείται στην εκφώνηση της εργασίας) – πχ. για λίστα AEM = {ΑΒΓΔ, ΕΖΗΘ, ΙΚΛΜ, ΝΞΟΠ}, ο polling worker “ψάχνει” (= προσπαθεί να συνδεθεί με) σε έναν αέναο βρόχο server sockets τα οποία έχουν γίνει bound στις IP διευθύνσεις “10.0.ΑΒ.ΓΔ”, “10.0.ΕΖ.ΗΘ”, “10.0.ΙΚ.ΛΜ”, “10.0.ΝΞ.ΟΠ” αντίστοιχα για κάθε AEM από τη λίστα.
- τέλος, στο main thread υλοποιείται το server socket το οποίο “ακούει” για εισερχόμενες συνδέσεις στη διεύθυνσή IP της εκάστοτε συσκευής (η οποία ορίζεται φυσικά βάσει του AEM), *listening_worker()*.

Πριν την εκκίνηση των threads ορίζεται ένα timeout σε μορφή σήματος alarm για την περάτωση της επικοινωνίας μετά από χρονική διάρκεια που δίνεται ως command line argument σε seconds.

2.2. Τι γίνεται όταν δύο συσκευές συνδεθούν

Βασικό στοιχείο της υλοποίησης είναι ότι όταν η επικοινωνία ξεκινήσει είτε επειδή ο polling worker “βρήκε” server socket να ακούει, ή επειδή ο listening worker δέχθηκε εισερχόμενη σύνδεση, τότε καλείται ένας ενιαίος κώδικας, ο *communication_worker()*. Σε αυτόν, γίνεται τόσο η σειριακή αποστολή του buffer ενδιάμεσων μηνυμάτων – στον κώδικα υπάρχει η μεταβλητή *MESSAGES_BUFFER* για να κρατάει τον buffer αυτόν – (με έλεγχο εάν κάθε μήνυμα δεν έχει αποσταλλεί στη

συγκεκριμένη συσκευή στο παρελθόν), όσο και η παραλαβή μηνυμάτων από την άλλη συσκευή και η αποθήκευση αυτών **μετά από έλεγχο** είτε στον buffer ενδιάμεσων μηνυμάτων ή στο inbox - στον κώδικα υπάρχει η μεταβλητή *INBOX* για να κρατάει τα μηνύματα που προορίζονταν για εκείνη. Οι συναρτήσεις που είναι υπεύθυνες για την επικοινωνία μεταξύ των συσκευών βρίσκονται στο *communication.h*.

2.3. Που αποθηκεύονται τα μηνύματα κατά τη διάρκεια ενός session

Αρχικά, τόσο στον κώδικα όσο και στο WebReport (εφαρμογή που δημιουργήθηκε για την προβολή των logs - γίνεται αναφορά παρακάτω) τα μηνύματα διαχωρίζονται στους παρακάτω τύπους / κατηγορίες:

- *produced*: μηνύματα που έχουν παραχθεί από τον producer worker του κάθε peer, με αποστολέα το AEM του peer και παραλήπτη κάποιο τυχαίο AEM από τα υπόλοιπα της λίστας
- *received*: μηνύματα τα οποία ελήφθησαν κατά την επικοινωνία του peer με κάποιον άλλον peer
- *transmitted*: μηνύματα τα οποία απεστάλησαν κατά την επικοινωνία του peer με κάποιον άλλον peer (τα μηνύματα που παράχθηκαν (*produced*), σε επόμενες χρονικές στιγμές πιθανότατα θα αποτελέσουν μηνύματα αυτής της κατηγορίας καθώς θα εισαχθούν στον buffer για μετάδοση)
- *inbox*: τα μηνύματα αυτά είναι τα μηνύματα της κατηγορίας *received*, δηλ. μηνύματα που ελήφθησαν κατά την επικοινωνία του peer με κάποιον άλλον peer, τα

οποία όμως προορίζονταν για τον peer που τα έλαβε και άρα προωθούνται στο inbox του

Οι πίνακες (buffers) στους οποίους δύνανται να αποθηκευτούν τα μηνύματα των παραπάνω κατηγοριών είναι οι εξής δύο:

1. *MESSAGES_BUFFER*: πίνακας ενδιάμεσης αποθήκευσης. Εδώ, αποθηκεύονται τα μηνύματα τα οποία είτε έχουν παραχθεί από τον peer ή έχουν ληφθεί από κάποιον άλλον peer (σε μεταξύ τους επικοινωνία) χωρίς ωστόσο να προορίζονταν για αυτόν (για τον peer που έλαβε). Πρόκειται για έναν κυκλικό buffer προκαθορισμένου μήκους 2000 μηνυμάτων, κάθε μήνυμα του οποίου αποστέλλεται, μετά από έλεγχο, σε κάθε νέο peer όταν ξεκινήσουν να επικοινωνούν. Ο έλεγχος αφορά εάν το μήνυμα έχει αποσταλεί στο παρελθόν στον εκάστοτε peer και εάν το μήνυμα έχει αποσταλεί στο παρελθόν στον peer-recipient – σε αυτή τη περίπτωση το μήνυμα δεν αποστέλλεται σε κανέναν άλλο peer.
2. *INBOX*: πίνακας μόνιμης αποθήκευσης. Εδώ αποθηκεύονται τα μηνύματα τα οποία έχουν ληφθεί κατά τη διάρκεια της επικοινωνίας με άλλους peers και τα οποία προορίζονταν για αυτόν τον peer. Όταν βρεθεί ένα τέτοιο μήνυμα, τότε αυτό δεν τοποθετείται στο *MESSAGES_BUFFER* αλλά μόνο στο *INBOX*, έτσι ώστε να μην αποσταλεί ξανά σε άλλους peers. Για κάθε μήνυμα που μπαίνει στο *INBOX* κρατείται ένα πρόσθετο πεδίο που δείχνει ποιος peer μετέδωσε το μήνυμα στον peer-recipient (συνήθως αυτός

διαφέρει από τον peer-sender – αυτόν δηλαδή που παρήγαγε το μήνυμα).

2.4. Αποφυγή Conflicts

Όπως φαίνεται από την ενότητα 2.1, επειδή τα client & server threads για κάθε peer τρέχουν παράλληλα, είναι πιθανό το client thread ενός peer να “βρει” το server thread ενός άλλου – ξεκινώντας έτσι ένα νέο event επικοινωνίας μεταξύ τους – αλλά πριν ολοκληρωθεί η επικοινωνία αυτών των threads, το client thread του άλλου peer να “βρει” το server thread του ενός. Κάτι τέτοιο θα οδηγούσε σε αστάθεια μεταξύ των sockets και πιθανώς σε χάσιμο μηνυμάτων (να γραφτούν στον buffer μηνύματα πάνω σε άλλα που μόλις γράφτηκαν). Για την προστασία από κάτι τέτοιο η εφαρμογή κρατάει μια λίστα με τις ενεργές κάθε φορά συνδέσεις – μεγέθους όσο και η λίστα με τα AEM. Έτσι την στιγμή που δύο αντίλογα threads δύο διαφορετικών peers ξεκινήσουν να επικοινωνούν, σηκώνεται σε κάθε peer μία σημαία στο index του AEM του απέναντι peer, η οποία πέφτει στην λήξη της επικοινωνίας μεταξύ τους. Αυτό αποτρέπει την ταυτόχρονη σύνδεση αντίλογων threads μεταξύ δύο peers και ως εκ τούτου και τον κίνδυνο απώλειας μηνυμάτων.

Περισσότερες λεπτομέρειες παρέχονται με μορφή σχολίων στον κώδικα C (βλ. `server.h` > `device_exists()`, `device_push()`, `device_remove()`).

3. Συνθήκες Υλοποίησης & Παραδοχές

Στον παραδοτέο φάκελο δίνονται τα logs τα δύο τελευταία επιτυχή sessions εκτέλεσης. Παρακάτω, δίνονται το setup που χρησιμοποιήθηκε σε amφότερα τα sessions καθώς και οι συνθήκες διεξαγωγής τους.

3.1. To setup

Το setup που χρησιμοποιήθηκε για την εκτέλεση των sessions της παραπάνω εργασίας αποτελείται από:

- 2 x Raspberry Pi Zero W (with headers)
- 1 x PC, Dell <Μοντέλο>, το οποίο συνδέθηκε στο ασύρματο ad-hoc δίκτυο των Raspberry Pi

Το laptop τρέχει Ubuntu 18.04 και για τη σύνδεση του στο ad-hoc δίκτυο έπρεπε να δημιουργηθεί μια νέα ασύρματη σύνδεση με Manual Configuration:

Address	Netmask	Gateway
10.0.0.1	8 (255.0.0.0)	-

Στο σημείο αυτό θέλω να ευχαριστήσω το συνάδελφο και φίλο, Αργύρη Παπουδάκη (του οποίου το ΑΕΜ έχει το 2ο Raspberry Pi Zero - 8600) για την ευγενική χορηγία του δικού του Raspberry Pi Zero αλλά και του laptop του (καθώς στο δικό μου κάθε προσπάθεια να συνδεθώ στο ασύρματο ad-hoc δίκτυο των raspberries έπεσε στο κενό), καθώς χωρίς αυτή τη συνεισφορά θα είχε δαπανηθεί πολύ περισσότερος χρόνος στο στήσιμο του setup.

3.2. Η Λίστα των AEM

Για τους σκοπούς της εργασίας, η προκαθορισμένη λίστα με τα AEM έχει οριστεί ως ακολούθως:

```
// Sorted list of AEMs
static const uint32_t CLIENT_AEM_LIST[] = {
    0001, 7051, 8001, 8011, 8032, 8600, 8723, 8859,
    8869, 8888, 8998, 8999, 9005, 9026, 9028, 9999
};
```

όπου υπάρχει τόσο το AEM του συγγραφέα, **9026**, όσο και του laptop (το οποίο λειτούργησε ως η 3η ενσωματωμένη συσκευή), **0001**, καθώς και του συναδέλφου του οποίου χρησιμοποιήθηκε το Raspberry Pi Zero (ο ίδιος δεν υλοποίησε την εργασία), **8600**. Επιλέχθηκαν

3.3. Συνθήκες Διεξαγωγής Sessions

3.3.1. Χώρος διεξαγωγής

Τόσο το πρώτο όσο και το δεύτερο session έγιναν στη βιβλιοθήκη του Α.Π.Θ. Το setup αποτελούνταν που χρησιμοποιήθηκε φαίνεται στην ενότητα 3.1.

3.3.2. Ημερομηνία & Χρόνος Διεξαγωγής

Το πρώτο session ξεκίνησε την Τετάρτη 9/10/2019 στις 20:11:48 και διήρκεσε 7200.0026 sec (είχε ζητηθεί χρόνος εκτέλεσης 7200sec = 2hrs). Το δεύτερο session ξεκίνησε την Πέμπτη 10/10/2019 στις 18:31:09 και διήρκεσε 7200.0025 sec (είχε ζητηθεί χρόνος εκτέλεσης 7200sec = 2hrs).

4. Αποτελέσματα

Αποτελέσματα πέρα από αυτά που εμφανίζονταν στην κονσόλα κατά τη διάρκεια εκτέλεσης και από το γεγονός της επιτυχούς περάτωσης της δίωρης επικοινωνίας σε κάθε session δεν υπάρχουν. Αυτό που υπάρχει όμως και αξίζει να παρουσιασθεί είναι:

- **στατιστικά της επικοινωνίας** για κάθε μία από τις συμμετέχουσες συσκευές (εφόσον υπάρχει πρόσβαση σε αυτές; στις συσκευές της εργασίας υπάρχει σε όλες πρόσβασης αφού ελέγχονται από το συγγραφέα)
- **timeline** με τα συμβάντα που συνέβησαν σε κάθε session καθώς και με τα μηνύματα που αντάλλαξαν οι συσκευές κατά τη διάρκεια του κάθε event, **ως προς τον χρόνο**

4.1. Συλλογή Αποτελεσμάτων

Για την καταγραφή των logs, όπου κρίθηκε απαραίτητο στον κώδικα, εισήχθησαν εντολές για την εγγραφή logs σε ένα αρχείο για κάθε συμμετέχουσα συσκευή και κάθε session εκτέλεσης. Τα αρχεία αυτά είναι **τύπου JSON** ώστε να μπορούν εύκολα μετά το πέρας της επικοινωνίας να παρσαριστούν και να προβληθούν με web τεχνολογίες. Έτσι, για κάθε session δημιουργείται ένα αρχείο, *session1.json*, το οποίο προβάλλεται μετά μέσω μικρής εφαρμογής που αναπτύχθηκε στη JavaScript για το σκοπό αυτό.

4.1.1. Συλλογή Στατιστικών

Τα στατιστικά που κρατήθηκαν είναι τα εξής (παραθέτονται τα αντίστοιχα `fields` από τον τύπο `MessageStats` που δημιουργήθηκε για την αποθήκευση των στατιστικών):

- *produced*: πόσα μηνύματα παράχθηκαν από τη συσκευή (από τον `producer worker`)
- *received*: πόσα μηνύματα έλαβε η συσκευή κατά την διάρκεια της επικοινωνίας (συμπ. και διπλότυπων)
- *received_for_me*: πόσα μηνύματα έλαβε τα οποία είχαν ως παραλήπτη την ίδια τη συσκευή (και άρα πήγαν στο INBOX)
- *transmitted*: πόσα μηνύματα μετέδωσε η συσκευή κατά την διάρκεια της επικοινωνίας
- *transmitted_to_recipient*: πόσα μηνύματα μετέδωσε η συσκευή στον τελικό τους παραλήπτη (να συνδέθηκε με τη συσκευή της οποίας το AEM είναι στο πεδίο `recipient` του αντίστοιχου μηνύματος)

4.1.2. Καταγραφή Logs για την Δημιουργία Timeline

Για την δημιουργία του `timeline` εισάγεται η έννοια του `event`. Αυτό ξεκινάει όταν η συσκευή ξεκινήσει την επικοινωνία της με μία άλλη συσκευή και ολοκληρώνεται όταν η λήξη αυτή η επικοινωνία. Με χρήση `locks` φροντίζουμε ώστε τα `events` να καταγράφονται με την ίδια σειρά με την οποία συμβαίνουν για πιστή απεικόνιση του τι συνέβη κατά τη διάρκεια ενός `session`. Οι συναρτήσεις που γράφουν στο JSON αρχείο (καθώς και στη κονσόλα) βρίσκονται στο `log.h`.

4.2. Παρουσίαση Αποτελεσμάτων

Το πιο ενδιαφέρον κομμάτι ίσως πέρα από τη συγγραφή του κώδικα είναι η παρουσίαση των κρατηθέντων αποτελεσμάτων. Μέρος αυτής θα παρατεθεί ως screenshots, θα ήταν ωστόσο βασικό να εκτελεστεί το συνοδευτικό script (*WebReport/server.sh*) για προβολή στον browser της web εφαρμογής.

4.2.1. WebReport

Δημιουργήθηκε, λοιπόν μία εφαρμογή σε JS-HTML-CSS η οποία κάνει τα εξής:

- διαβάζει το αρχείο JSON για τη συσκευή με το AEM, XXXX και το session νούμερο Y
- προβάλλει (α) το timeline ή (β) ένα σύνολο στατιστικών μαζί με την τελική μορφή του MESSAGES_BUFFER και του INBOX για την συγκεκριμένη συσκευή συσκευή και το συγκεκριμένο session

Για είσοδο στην εφαρμογή απαιτούνται τα εξής:

1. μετάβαση στο φάκελο WebReport:

```
cd <path παραδοτέου φακέλου>/report/WebReport
```

2. εκτέλεση του Python script *serve.py*:

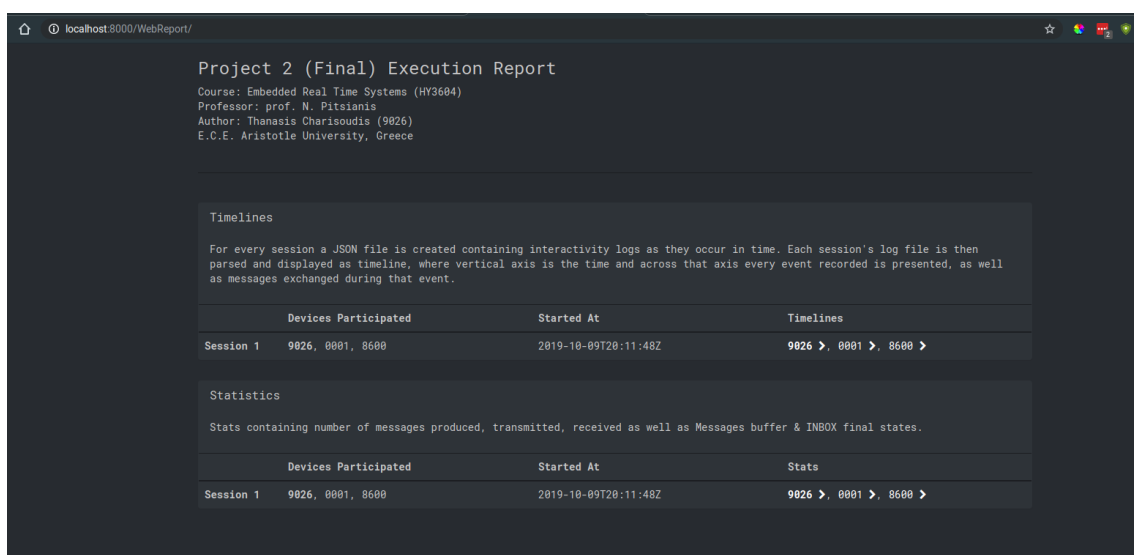
```
python server.py
```

(στην ουσία καλεί το Python module *SimpleHttpServer* το οποίο δίνεται μαζί με τη python, για να ξεκινήσει έναν απλό HTTP server στο τρέχων directory και κατόπιν ανοίγει στον προεπιλεγμένο browser το url της εργασίας: <http://localhost:9026>)

Για τα session που παραδίδονται, sessions **1** και **2** για τις συσκευές με AEM **0001** (PC), **9026** (συσκευή Raspberry Pi Zero -RPZ- του συγγραφέα) και **8600** (συσκευή RPZ συναδέλφου) έχουν δημιουργηθεί τα αντίστοιχα JSON αρχεία και έχουν μεταφερθεί στους φακέλους `/report/WebReport/Sessions`.

4.3. Προβολή logs στο WebReport

Εφόσον τρέξουμε το `serve.py`, ανοίγει ο προεπιλεγμένος browser στην ακόλουθη σελίδα:



Project 2 (Final) Execution Report

Course: Embedded Real Time Systems (HY3604)
Professor: prof. N. Pitsianis
Author: Thanasis Charisoudis (9026)
E.C.E. Aristotle University, Greece

Timelines

For every session a JSON file is created containing interactivity logs as they occur in time. Each session's log file is then parsed and displayed as timeline, where vertical axis is the time and across that axis every event recorded is presented, as well as messages exchanged during that event.

	Devices Participated	Started At	Timelines
Session 1	9026, 0001, 8600	2019-10-09T20:11:48Z	9026 > , 0001 > , 8600 >

Statistics

Stats containing number of messages produced, transmitted, received as well as Messages buffer & INBOX final states.

	Devices Participated	Started At	Stats
Session 1	9026, 0001, 8600	2019-10-09T20:11:48Z	9026 > , 0001 > , 8600 >

Από εκεί επιλέγουμε εάν θέλουμε να δούμε το timeline ή τα στατιστικά για το session γ (γ -οστή γραμμή του κάθε πίνακα; ο πρώτος περιέχει τα timelines ενώ ο δεύτερος περιέχει τα στατιστικά επικοινωνίας για κάθε συσκευή που συμμετείχε στο session). Όλες οι συσκευές που έλαβαν μέρος στο κάθε session εμφανίζονται χωρισμένες με comma στη τελευταία στήλη του κάθε πίνακα (με έντονα εμφανίζεται η συσκευή του συγγραφέα). Για κάθε συσκευή υπάρχει σύνδεσμος που δείχνει στη σελίδα που προβάλλεται το timeline ή τα στατιστικά χρήσης για αυτήν και το αντίστοιχο session.

4.3.1. Προβολή timeline ενός session

Για την προβολή του timeline πηγαίνουμε στο σχετικό uri:

```
/timeline/?session=<sessionIndex>&device=<deviceAEM>
```

όπου sessionIndex ο αριθμός του session στο παραδοτέο (στην παρούσα έκδοση θα είναι 1 ή 2), ενώ deviceAEM το AEM της συσκευής της οποίας θέλουμε να δούμε το timeline.

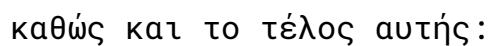
Στην σελίδα εμφανίζεται ένα κάθετος άξονας στην κορυφή του οποίου εμφανίζεται σε ένα κουτάκι πότε ξεκίνησε το session το AEM της συσκευής και την πραγματική διάρκεια του session (σε κάθε session η ζητούμενη διάρκεια είναι 7200sec). Για κάθε event υπάρχει μια οριζόντια γραμμή που αναφέρει τον τύπο του event, πότε προέκυψε (ξεκίνησε) ενώ κάτω από αυτήν υπάρχουν όλα τα μηνύματα που ανταλλάχθηκαν ή παράχθηκαν κατά τη διάρκεια του event αυτού.

Οι τύποι events είναι οι ακόλουθοι:

- *production*: event κατά το οποίο παράγεται ένα νέο μήνυμα από τον producer worker και αποθηκεύεται στον MESSAGES_BUFFER
- *connection*: event κατά το οποίο ξεκίνησε η επικοινωνία μεταξύ του peer και κάποιου άλλου peer - συσκευής. Τα μηνύματα που ανταλλάχθηκαν παραθέτονται από κάτω, με ενδείξεις "Message Transmitted" και "Message Received" ανάλογα με το εάν το εκάστοτε μήνυμα στάλθηκε ή ληφθηκε.

Πατώντας στο κουμπί "Details" στο κουτί του κάθε μηνύματος βλέπουμε και τα metadata του μηνύματος, όπως το εάν

Ακολουθως φαίνεται η αρχή της προβολής του timeline για τη συσκευή 9026 και το session 1. Το url της σελίδας είναι: <http://localhost:9026/timeline/?session=1&device=9026>



4.3.2. Προβολή stats του session

Για την προβολή του timeline πηγαίνουμε στο σχετικό uri:

```
/stats/?session=<sessionIndex>&device=<deviceAEM>
```

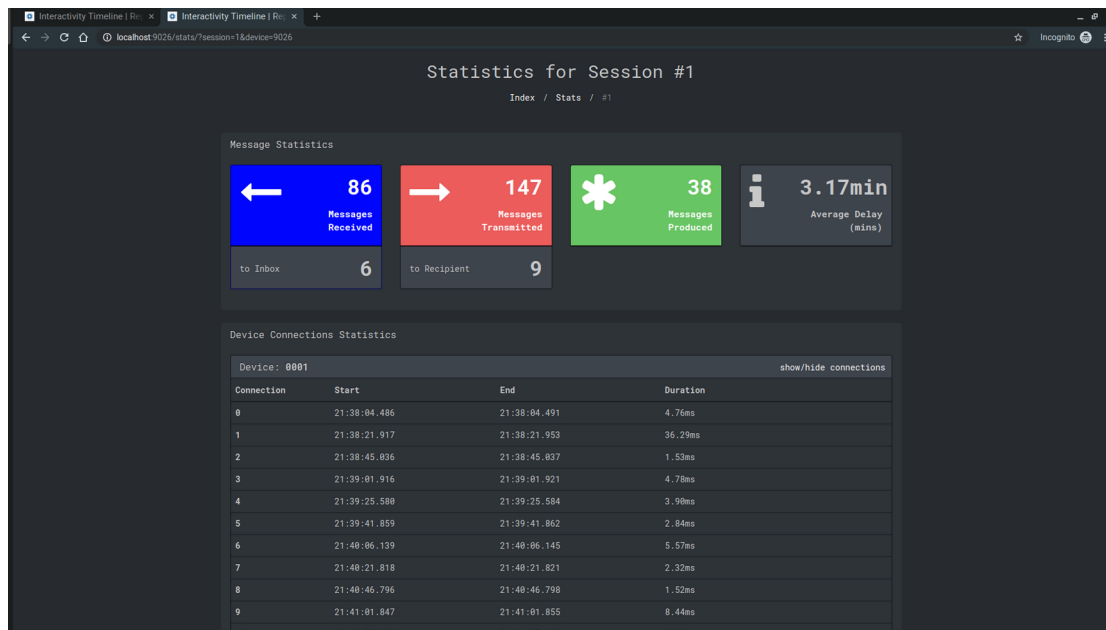
όπου sessionIndex ο αριθμός του session στο παραδοτέο (στην παρούσα έκδοση θα είναι 1 ή 2), ενώ deviceAEM το AEM της συσκευής της οποίας θέλουμε να δούμε το timeline.

Στην σελίδα εμφανίζονται τρία (3) sections:

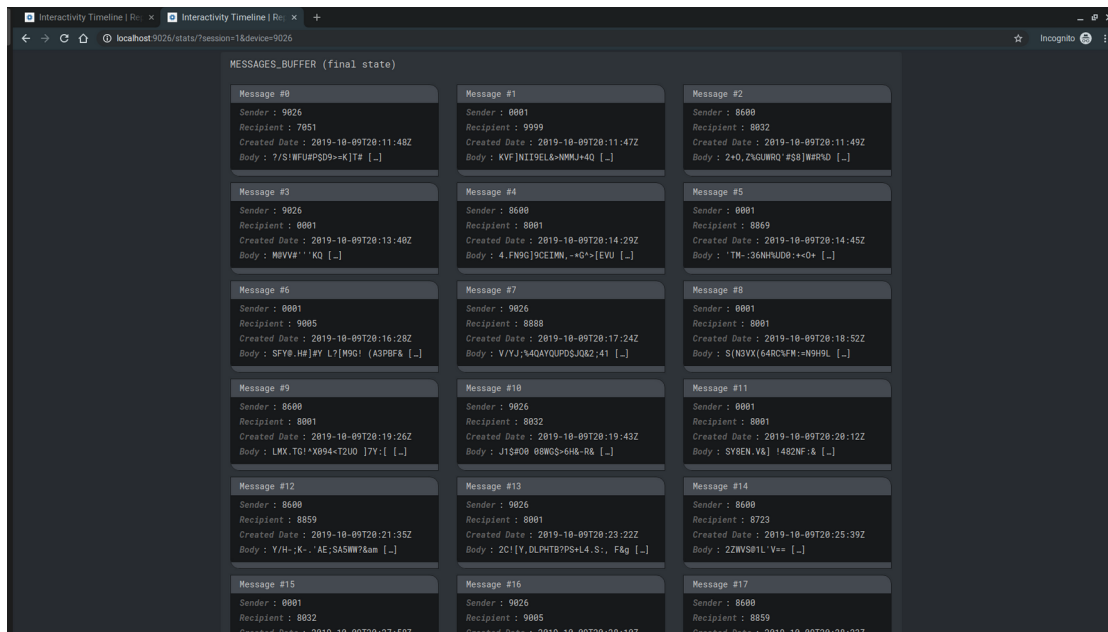
1. "Message Statistics", το οποίο περιέχει τα ακόλουθα στατιστικά χρήσης – επικοινωνίας:
 - *Messages Received*: πόσα μηνύματα λήφθηκαν συνολικά στο session από τη συσκευή
 - *Messages Transmitted*: πόσα μηνύματα στάλθηκαν συνολικά στο session από τη συσκευή
 - *Messages Produced*: πόσα μηνύματα παράχθηκαν συνολικά στο session από τη συσκευή
 - *Average Delay (mins)*: μέση καθυστέρηση παραγωγής μηνυμάτων σε λεπτά
2. "Device Connections Statistics", το οποίο περιέχει στατιστικά των συνδέσεων με άλλες συσκευές. Για κάθε συσκευή με την οποία συνδέθηκε δημιουργείται ένας πίνακας κάθε γραμμή του οποίου είναι μια σύνδεση ενώ οι στήλες λένε πότε ξεκίνησε, τελείωσε και πόσο διήρκεσε η σύνδεση αυτή. Στο τέλος του κάθε πίνακα δίνεται και η μέση διάρκεια σύνδεσης με την συσκευή.

3. "MESSAGES_BUFFER (final state)", το οποίο περιέχει όλα τα μηνύματα που υπήρχαν στον ενδιαμέσο πίνακα MESSAGES_BUFFER όταν τελείωσε το session
4. "INBOX (final state)", το οποίο περιέχει όλα τα μηνύματα που υπήρχαν στον πίνακα INBOX όταν τελείωσε το session

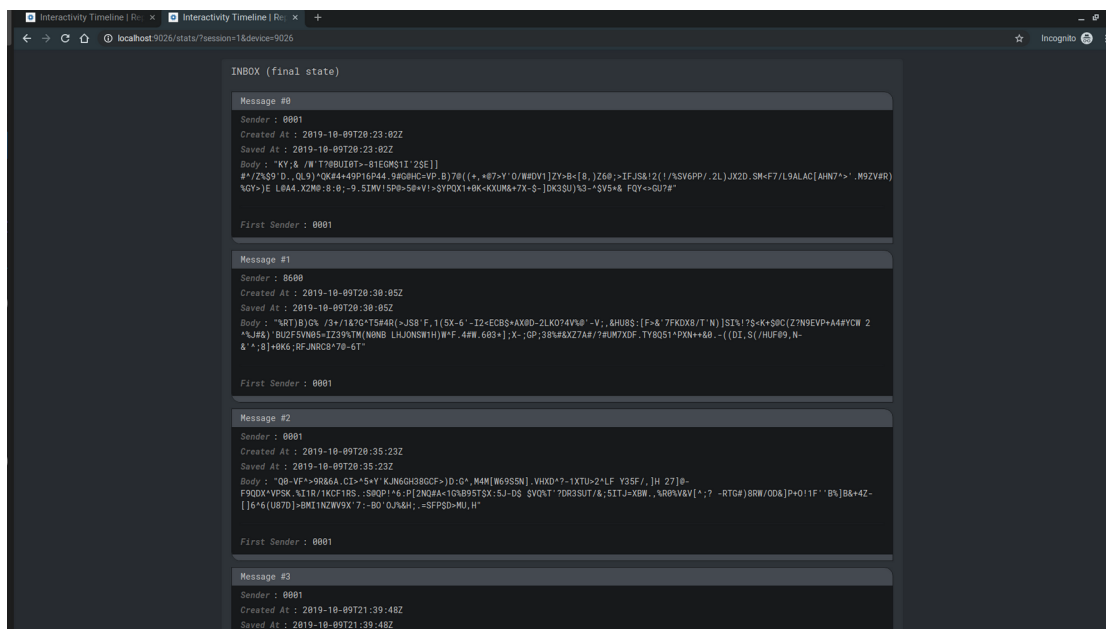
Ακολούθως φαίνεται η αρχή της προβολής του stats για τη συσκευή 9026 και το session 1. Το url της σελίδας είναι: <http://localhost:9026/stats/?session=1&device=9026>



ενώ η αρχή του section "MESSAGES_BUFFER (final state)" φαίνεται ακολούθως:



Τέλος, η αρχή του section “INBOX (final state)” φαίνεται
ακολουθώς:



Για ολοκληρωμένη εικόνα των logs παρακαλώ να τρέξετε το
server.py μέσα από το φάκελο WebReport για προβολή των
παραπάνω στον browser.

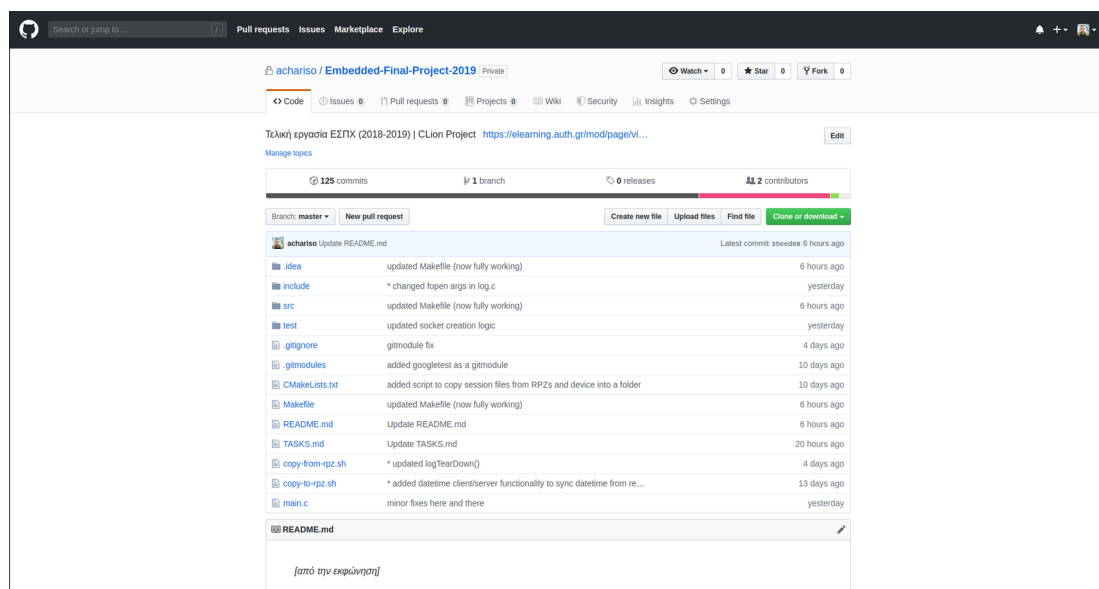
5. Ο κώδικας C

5.1. Οργάνωση(Clion) Project

Ο κώδικας οργανώθηκε στα εξής subdirs:

- *include*: περιέχονται όλα τα header files
- *src*: περιέχονται όλα τα αρχαία πηγαίου κώδικα C
- *test*: περιέχονται τα unit tests του κώδικα καθώς και ένα git submodule με τον κώδικα της βιβλιοθήκης google/googletest (για unit testing/mocking στη C)

Ένα screenshot από το [repository στο GitHub](#) παρατίθεται ακολούθως:



Ακολούθως παραθέτονται βασικοί τύποι που δημιουργήθηκαν στη C για την παραιτέρω κατανόηση της υλοποίησης της εφαρμογής.

5.2.0 τύπος *Message*

Πηρύνας της υλοποίησής μας είναι ο τύπος που δημιουργήθηκε για να κρατάει ένα μήνυμα είτε αυτό είναι παραχθέν από τη

συσκευή είτε είναι από επικοινωνία με άλλη συσκευή. Ορίζεται ως εξής:

```
typedef struct message_t {
    // Necessary fields
    uint32_t sender;                // AEM αποστολέα:      uint32
    uint32_t recipient;            // AEM παραλήπτη:    uint32
    uint64_t created_at;           // Χρόνος δημιουργίας: uint64 (Linux timestamp-10 digits)
    char body[MESSAGE_BODY_LEN];   // Κείμενο μηνύματος: ASCII[256]

    // Metadata
    uint8_t transmitted;           // If the message was actually transmitted from this device
    uint8_t transmitted_devices[CLIENT_AEM_LIST_LENGTH]; // Boolean array, 1 if i-th device has
                                                                // received the message, 0 otherwise
    uint8_t transmitted_to_recipient;
} Message;
```

Όπως φαίνεται, πέρα από τα βασικά πεδία, όπως το AEM του αποστολέα και του παραλήπτη ή το σώμα του μηνύματος, υπάρχουν κάποια πεδία, τα "metadata". Αυτά είναι:

- *transmitted*: εάν το μήνυμα έχει μεταδοθεί σε προηγούμενη επικοινωνία. Χρησιμοποιείται έτσι ώστε όταν πρόκειται να διαγραφεί ένα μήνυμα από τον κυκλικό buffer να αναζητηθεί π.χ. το πρώτο το οποίο έχει μεταδοθεί
- *transmitted_devices*: σε ποιες συσκευές έχει μεταδοθεί το μήνυμα σε προηγούμενη επικοινωνία (αποτελείται από ισάριθμα με το αριθμό AEM της λίστας αναζήτησης στοιχεία με τιμές 0/1)
- *transmitted_to_recipient*: εάν το μήνυμα έχει μεταδοθεί στον τελικό του παραλήπτη (πεδίο recipient) στο παρελθόν

5.3.0 τύπος *InboxMessage*

Ένας παράγωγος τύπος του τύπου Message είναι ο τύπος *InboxMessage*. Απο τα πεδία metadata δεν έχει κανένα, έχει όμως ένα δικό του: τον αποστολέα που έδωσε αυτό το μήνυμα; αυτός δεν είναι απαραίτητο να είναι αυτός που παρήγαγε το

μήνυμα καθώς αυτό μπορεί να έφτασε από κάποια άλλη διαδρομή στη συσκευή. Ο τύπος δίνεται ακολούθως:

```
typedef struct inbox_message_t {  
    // Necessary fields  
    uint32_t sender;           // AEM αποστολέα:      uint32  
    uint64_t created_at;       // Χρόνος δημιουργίας:  uint64 (Linux timestamp-10 digits)  
    uint64_t saved_at;         // Χρόνος αποθήκευσης:  uint64 (Linux timestamp-10 digits)  
    char body[MESSAGE_BODY_LEN]; // Κείμενο μηνύματος:  ASCII[256]  
  
    // Metadata  
    uint32_t first_sender;      // AEM της συσκευής που μετέδωσε το μήνυμα  
} InboxMessage;
```

Όπως φαίνεται, το AEM του παραλήπτη παραλείπεται καθώς νοείται το AEM της συσκευής που τρέχει. Πέρα από τα βασικά πεδία, όπως το AEM του αποστολέα ή το σώμα του μηνύματος, υπάρχουν ένα ακόμη πεδίο ως “metadata”. Αυτό είναι:

- *first_sender*: ο πρώτος αποστολέας που έδωσε το μήνυμα αυτό που “μπήκε” στο inbox. Κάθε επόμενη παραλαβή του ίδιου μηνύματος θα οδηγήσει στην απόρριψή του (καμία ενέργεια)

5.4.0 πίνακας (buffer) Messages

Επίσης στον πυρήνα της υλοποίησής μας είναι ο κυκλικός buffer Messages στον οποίο γίνεται η ενδιάμεση αποθήκευση των μηνυμάτων κατά την διάρκεια ενός session επικοινωνίας πραγματικού χρόνου. Ορίζεται ως εξής:

```
Message MESSAGES_BUFFER[ MESSAGES_SIZE ];
```

(η μεταβλητή **MESSAGES_SIZE** έχει οριστεί σε 2000 μηνύματα)

Βασική ιδέα όπως περιγράφηκε πιο πάνω η εισαγωγή στον buffer (μετά από έλεγχο) μηνύματα των οποίων ο παραλήπτης διαφέρει από το AEM της συσκευής που λαμβάνει το μήνυμα. Για το σκοπό αυτό υπάρχει η επιλογή να γίνεται “**τυφλή**” εισαγωγή – χωρίς έλεγχο των μηνυμάτων, δηλ. όταν ο buffer γεμίσει απλώς ξεκινάει να γράφει στην αρχή – ή να γίνεται

έλεγχος ώστε όταν γεμίσει ο buffer να γράφονται νέα μηνύματα σε θέσεις **μηνυμάτων που έχουν αποσταλλεί μόνο** (εκτός και εάν αυτό δεν είναι δυνατό). Η επιλογή αυτή βρίσκεται μεταξύ άλλων στο *includes/conf.h*:

```
#ifndef MESSAGES_PUSH_OVERRIDE_POLICY
#define MESSAGES_PUSH_OVERRIDE_POLICY "blind" // "sent_only", "blind"
#endif
```

Η ορθότητα με την οποία δουλεύει η παραπάνω λογική ελέγχεται στα unit tests που έχουν αναπτυχθεί για τις συναρτήσεις του server.h, test/final_tests/ServerTest.cpp, και συγκεκριμένα στα test:

```
TEST_F(ServerTest, MessagesPushSimple)
```

και

```
TEST_F(ServerTest, MessagesPushFull)
```

5.5.0 πίνακας (buffer) *INBOX*

Στον πίνακα αυτό αποθηκεύονται (μετά από έλεγχο) όλα τα μηνύματα τα οποία ελήφθησαν κατά τη διάρκεια της επικοινωνίας και τα οποία είχαν ως παραλήπτη το AEM της συσκευής που έλαβε το αντίστοιχο μήνυμα. Ορίζεται ως εξής:

```
InboxMessage INBOX[ INBOX_SIZE ];
```

(η μεταβλητή **INBOX_SIZE** έχει οριστεί σε 1000 μηνύματα)

6. Παραδοτέα

Στον παραδοτέο φάκελο βρίσκονται τα εξής sub-directories:

- **clion**: περιέχεται ο πηγαίος κώδικας της εφαρμογής (είναι ένα JetBrains® Clion® project). Έχει αφαιρεθεί ο φάκελος `.git` που περιέχει τα `git trees` ενώ έχει προστεθεί `Makefile` για compilation με `make`
- **report**: εκεί βρίσκεται και το παρών pdf (όπως και το αρχείο `odt` από το οποίο έχει παραχθεί). Μέσα στον φάκελο αυτό βρίσκεται ο φάκελος **WebReport** στον οποίο έχει αναπτυχθεί η εφαρμογή προβολής των logs (που περιγράφηκε παραπάνω) και στον οποίο βρίσκονται και τα JSON αρχεία log από τα sessions που εκτελέστηκαν (`/report/WebReport/Sessions/XXXX/sessionY.json` – XXXX είναι το AEM της συσκευής, Y ο δείκτης του session)