Ενσωματωμένα Συστήματα Πραγματικού Χρόνου
Τελική εργασία
Πετρίδης Περικλής - 8896

Εισαγωγή

Η εργασία αυτή είναι εμπνευσμένη από την δουλεία της καθηγήτριας Dr. Margaret Martonosi στο *ZebraNet*, το οποίο είναι ένα δίκτυο επικοινωνίας για την παρακολούθηση ζεβρών σε εθνικά πάρκα, του οποίου τα σημαντικά χαρακτηριστικά είναι η χαμηλή κατανάλωση ενέργειας και η μη-διαρκής σύνδεση κάθε συσκευής με τις υπόλοιπες.
Το κεντρικό νόημα λοιπόν είναι η υλοποίηση ενός επικοινωνιακού δικτύου με ενσωματωμένες συσκευές (στην προκειμένη περίπτωση raspberry pi με περιορισμένο OS) οι οποίες μπορούν να λειτουργούν για μεγάλες χρονικές περιόδους χωρίς επέμβαση, και χωρίς σφάλμα στην περίπτωση που δεν βρίσκουν άλλη συσκευή να συνδεθούν.

Λογική

Παραδοχές

Λόγω του ότι δεν βρήκα άτομα για να συνεργαστώ στην εργασία, χρησιμοποίησα επιπλέον 2 “ψεύτικους” κόμβους με την έννοια ότι έτρεξα τον κώδικα μου σε 3 διαφορετικές συσκευές, στο ίδιο WiFi, και έκανα ανάλυση στα δεδομένα και των 3 κόμβων για επαλήθευση σωστής λειτουργίας. Οι “τεχνητοί” κόμβοι χρησιμοποιούν τα (τυχαία) ΑΕΜ “6666” και “7777”.

Επιπλέον, αντί για 1 εώς 5 λεπτά ανά μήνυμα, παρήγαγα τα μηνύματα ανά περίπου 1 λεπτό ώστε να βγουν πιο αντιπροσωπευτικά στατιστικά και να υπάρχει μεγαλύτερη κίνηση στο δίκτυο.

Ανάλυση κώδικα

Παράμετροι

Λόγω των παραδοχών, στον κώδικα είναι “hard coded” οι στατικές IP και τα ΑΕΜ των υπολοίπων κόμβων, καθώς επίσης και τα μηνύματα που θα στέλνονται από κάθε κόμβο προς τους υπόλοιπους.

Main

Η main ουσιαστικά φροντίζει να παράξει ένα μήνυμα πριν ξεκινήσει όλη η διαδικασία, και έπειτα καλεί ένα νήμα pthread για την συνάρτηση *client και εκτελεί την συνάρτηση server(). Ουσιαστικά απλά ξεκινάει η παράλληλη εκτέλεση των δύο συναρτήσεων.

Server & Client

Χρησιμοποιείται ενα κλασικό παράδειγμα server & client με TCP Sockets σε C[1] που δημιουργεί listening socket στην θύρα 2288. Ο server μπαίνει σε ατέρμων βρόχο ώστε να δέχεται τις αιτήσεις του client. Όταν γίνει αίτηση, δημιουργείται νήμα για την υποδοχή και αποθήκευση του μηνύματος του client με την συνάρτηση *receive(), στην οποία περνάω ένα file descriptor. Η λούπα του server παραμένει κλειδωμένη λόγω mutex εώς ότου θέσουμε την τιμή του file descriptor στην *receive(). Αφού ξεκλειδωθεί το mutex, μπορεί να δεχθεί νέες συνδέσεις ο server().

Όσον αφορά τον buffer, ελέγχουμε αν έχει γεμίσει ή αν υπάρχουν διπλές εγγραφές και αν δεν υπάρχει πρόβλημα κλειδώνουμε το mutex του και αντιγράφεται το μήνυμα στον πίνακα και ξανα-ξεκλειδώνεται το mutex. Την εγγραφή στο msgs.txt την αναλαμβάνει η συνάρτηση logger().

Ο client προσπαθεί να κάνει connect με τους υπόλοιπους κόμβους στην σειρά μέσω connect() στην γνώστη IP τους, όπου τους στέλνει μηνύματα μέσω της send().

[1] https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

sendMsgs() & produceMsg()

Η send() χρησιμοποιείται από τον client και, όπως λέει και το όνομα, στέλνει τα μηνύματα στον εκάστοτε παραλήπτη.
Για την send() ορίστηκαν οι πίνακες LastMsgsToOthers και LastMsgsToOthersIndex και έχουν το τελευταίο μήνυμα που στάλθηκε στον κάθε άλλο κόμβο και τον δείκτη των μηνυμάτων.
Πριν στείλω κάποιο μήνυμα, ελέγχω τον δείκτη του τελευταίου μηνύματος που έλαβε ο παραλήπτης, και αν δεν είναι το πιο πρόσφατο, τότε στέλνω και όλα τα νέα μηνύματα.

Η productMsg() ουσιαστικά δημιουργεί τα τυχαία μηνύματα με την μορφή που ζητούνται από την άσκηση. Επιλέγει τυχαία έναν παραλήπτη και ένα από τα μηνύματα της hardcoded λίστας. Εδώ επίσης επιλέγεται και ο τυχαίος χρόνος ανάμεσα στα διαδοχικά μηνύματα. Στην προκειμένη περίπτωση επέλεξα να στέλνω μηνύματα κάθε 30 εώς 90 δευτερόλεπτα.

Εκτέλεση

Για *compilation* μπορεί να χρησιμοποιηθεί το ίδιο makefile είτε για linux είτε για cross-compiling. Συγκεκριμένα, για κανονικό compilation για τοπική δοκιμή, χρησιμοποιείται απλώς το `$make`.
Για cross-compilation χρησιμοποιείται το: `$make TARGET=cross`.

Για *εκτέλεση* μπορεί να χρησιμοποιηθεί το run.sh το οποίο τερματίζει το εκτελέσιμο μετά απο 2 ώρες με χρήση kill command (TERM SIG).

Κατα την λειτουργία του εκτελέσιμου δημιουργούνται και γίνονται εγγραφές στα αρχεία **msgs.txt** και **msgTimes.txt**. Το αρχείο msgs.txt περιέχει όλα τα μηνύματα που έχουν σταλεί ή έχουν δεχθεί από την ενσωματωμένη συσκευή, στην μορφή που ζητείται από την άσκηση (AEM1_AEM2_Timestamp_TextMsg). Στο αρχείο msgTimes.txt κρατούνται οι χρόνοι μεταξύ των διαδοχικών μηνυμάτων που αποστέλλονται από την συσκευή για στατιστικούς λόγους. Ομολογουμένως θα μπορούσα να μην δημιουργώ δεύτερο αρχείο, αλλά να χρησιμοποιώ τα timestamps και το πεδίο Sender για να βγάλω τα στατιστικά, αλλά αφού το έκανα εξαρχής το άφησα. Στην ανάλυση με matlab δεν χρησιμοποιείται το msgTimes.txt αφού είναι πλεονασμός.

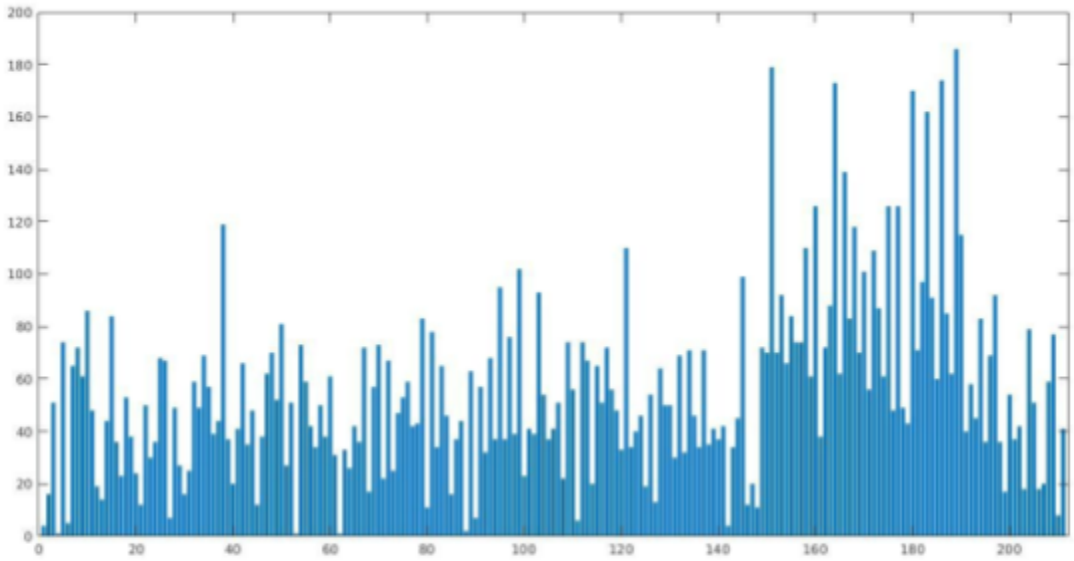
Αφού περάσουν οι 2 ώρες τα αρχεία msgs.txt και msgTimes.txt είναι πλέον έτοιμα. Αφού μαζευτούν τα 6 αρχεία (2 από κάθε κόμβο), χρησιμοποιείται το *shell script* **msg-to-csv.sh** το οποίο αντιγράφει το αρχείο, αλλάζει την κατάληξη σε *.csv* και αντικαθιστά τις κάτω παύλες (“_”) με υποδιαστολές (“,”), ώστε να γίνουν απευθείας εισαγωγή από το matlab.

Αποτελέσματα

Μερικά ενδιαφέροντα αποτελέσματα παρουσιάζονται εδώ. Όλα τα αποτελέσματα βγήκαν από το script **analyse_msgs.m** στον φάκελο ***matlab***.

Timestamps

Τα timestamps του node 8896 φαίνεται στο παρακάτω σχήμα. Φαίνεται πως ακόμα και με προκαθορισμένος χρόνος μεταξύ μηνυμάτων (30 - 90 δευτερόλεπτα) , πάλι υπάρχει ξεχωριστό και απρόβλεπτο delay.



Χρόνος μεταξύ διαδοχικές αποστολές για τον κόμβο 8896, σε δευτερόλεπτα

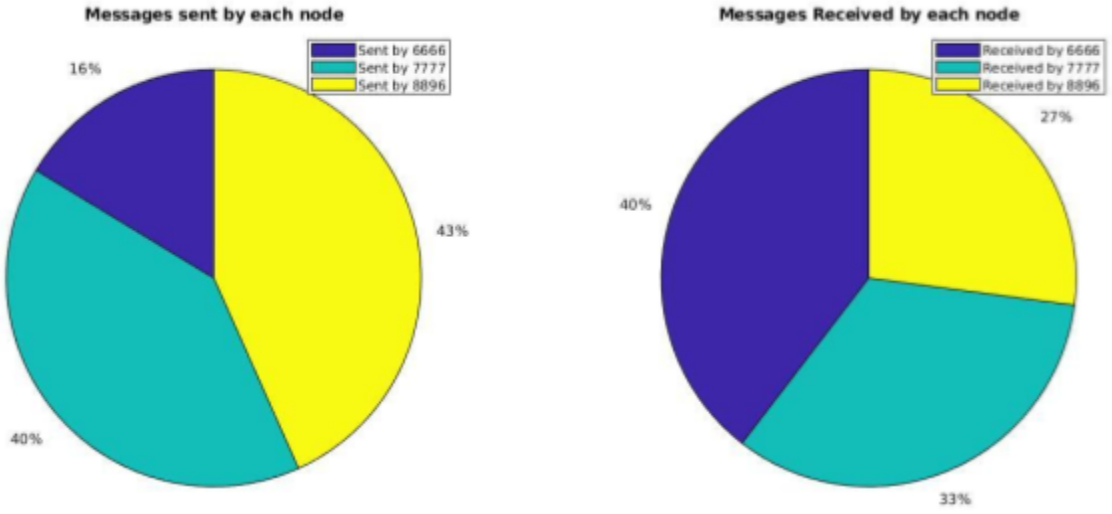
Μερικά στατιστικά για τα timestamp:
Mean = 55.1185
Median = 50
Standard devation = 34.3056

Αποστολές και λήψεις κάθε κόμβου

Ο **πίνακας δραστηριότητας** , δηλαδή πόσα μηνύματα έστειλε κάθε κόμβος προς κάθε άλλον είναι:

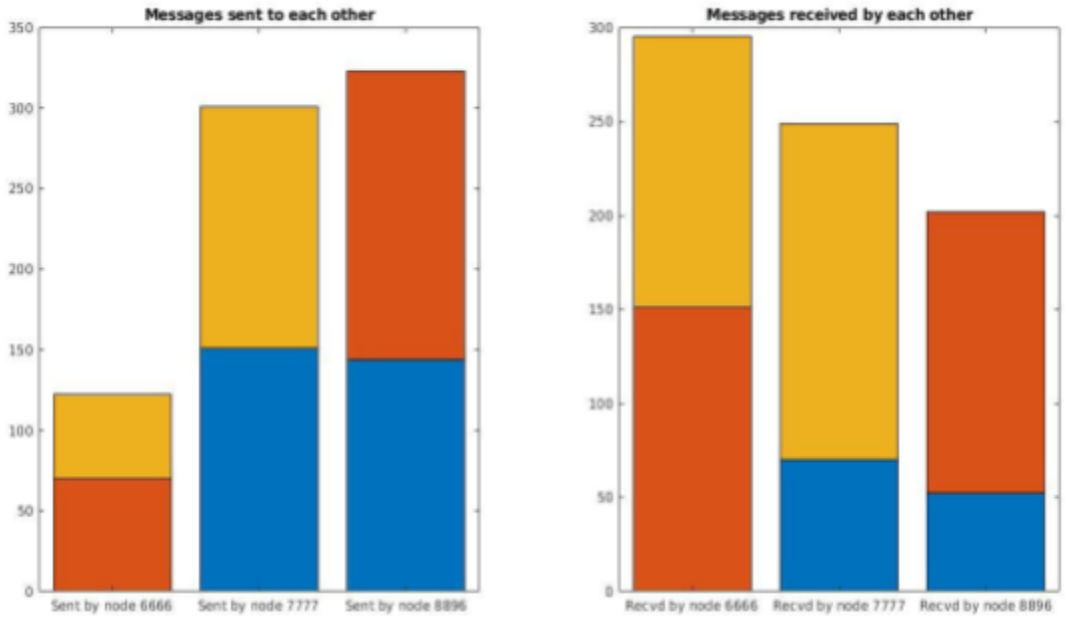
	Προς 6666	Προς 7777	Προς 8896
Από 6666	0	70	52
Από 7777	151	0	150
Από 8896	144	179	0

Στο παρακάτω διάγραμμα στις **πίτες** φαίνεται το σύνολο των μηνυμάτων που έστειλε και πήρε κάθε κόμβος ξεχωριστά.



Αριστερά: μηνύματα που αποστάλθηκαν από κάθε κόμβο,
Δεξια: μηνύματα που ελήφθησαν από κάθε κόμβο

Τέλος, στα **ραβδογράμματα** φαίνεται και ο διαχωρισμός των μηνυμάτων, δηλαδή ποιος στέλνει που και πόσα.



Αριστερά: Αποστολές από κάθε κόμβο στους άλλους 2
Δεξιά: Λήψεις μηνυμάτων κάθε κόμβου από τους άλλους 2

Έλεγχος Ορθότητας

Ο έλεγχος ορθότητας είναι πολύ απλός και λογικός. Θα πρέπει θεωρητικά κάθε μήνυμα που παράχθηκε από τους 3 κόμβους να έχει μεταδοθεί επιτυχώς σε τουλάχιστον έναν άλλον (αν πάει απευθείας στον παραλήπτη-στόχο). Επομένως, ελέγχω ότι **κάθε εγγραφή** στα αρχεία **msgs.txt** θα **πρέπει** να υπάρχει **τουλάχιστον 2 φορές ανάμεσα στα 3 αρχεία**.
Τον έλεγχο ορθότητας τον υλοποιώ στο matlab ως “post processing”, χρησιμοποιώντας και τα 3 αρχεία που παρήγαγαν οι 3 κόμβοι. Αυτό γίνεται στον φάκελο **matlab** με το **check.m**.

Συνοπτικά τα αποτελέσματα ορθότητας:

Συνολική επιτυχία μετάδοσης μεταξύ των 3 κόμβων**:**

86.46% ή **645/746** επιτυχή μηνύματα

Επιμέρους ποσοστά επιτυχίας για κάθε κόμβο για αποστολή και παραλαβή μηνυμάτων**:**

	Επιτυχία αποστολής	**Επιτυχία Παραλαβής**
Node “6666”	**87.767%**	**87.76%**
Node “7777”	**89.27%**	**83.13%**
Node “8896”	**82.08%**	**88.75%**

Συμπεράσματα

Βλέπουμε από τον έλεγχο ορθότητας ότι έχουμε λίγες απώλειες μηνυμάτων. Παρόλα αυτά, η κύρια λειτουργία η οποία είναι η ανταλλαγή μηνυμάτων με οικονομικό τρόπο έχει επιτευχθεί. Επίσης πρέπει να σημειωθεί ξανά ότι ο φόρτος ανα κόμβο αυξήθηκε από τα 5 λεπτά που ζητούσε η άσκηση σε περίπου 1 λεπτό. Δυστυχώς δεν πρόλαβα να δω τα στατιστικά για φόρτο εργασίας στα 5 λεπτά ανά μήνυμα, αλλά η με απλή λογική (και θεωρία ουρών, για $\lambda_2 = 5\lambda_1$), θεωρώ ότι τα αποτελέσματα θα ήταν ακόμα καλύτερα.

Το κύριο επόμενο βήμα είναι η βελτιστοποίηση του δικτύου ώστε να μειωθούν οι απώλειες. Αυτό κατά πάσα πιθανότητα μπορεί να επιτευχθεί με βελτίωση του κώδικα.

Τέλος, δοκιμάζοντας το δίκτυο για n=3 κόμβους και σε αυξημένη παραγωγή μηνυμάτων , θεωρώ ότι τα αποτελέσματα είναι αρκετά αντιπροσωπευτικά της λειτουργίας της ενσωματωμένης συσκευής στον χώρο του πανεπιστημίου ταυτόχρονα με συσκευές άλλων φοιτητών.

Σχόλια

Κάποιες *ασυνήθιστες* δυσκολίες που αντιμετώπισα:

1) Συντονισμός των 3 συσκευών μόνος μου, ειδικά όταν άλλαζα κάτι στον κώδικα.
2) Διαχείριση των δεδομένων και από τις 3 συσκευές. Χρειαζόταν ιδιαίτερη προσοχή στα conflict αφού όλα τα αρχεία έβγαιναν με το ίδιο όνομα.
3) Ανάλυση των δεδομένων με matlab. Δεν περίμενα ότι θα μπορούσα να βγάλω τόσες πληροφορίες από 3 κόμβους που ανταλλάσσουν μηνύματα για 2 ώρες. Δυστυχώς έπρεπε να σταματήσω σε κάποιο σημείο την ανάλυση λόγω χρόνου.

Κώδικας:
Ο κώδικας βρίσκεται στο https://github.com/PericlesPet/espx-final
Αν και δεν έγραψα README, προσπάθησα να καλύψω ότι χρειάζεται στο report. Κυρίως χρησιμοποιείται το makefile, τα δύο shell scripts και τα 2 matlab scripts.