Ατομική Διπλωματική Εργασία

ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΤΟΥ MQTT PROTOCOL ΣΕ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

Μαρία Μουσικού

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΤΟΥ MQTT PROTOCOL ΣΕ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

Μαρία Μουσικού

Επιβλέπων Καθηγητής

Αντρέας Πιτσιλλίδης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2017

Ευχαριστίες

Θα ήθελα να ευχαριστήσω το επιβλέπων καθηγητή μου Δρ. Ανδρέα Πιτσιλλίδη που μου ανάθεσε το συγκεκριμένο θέμα καθώς και για όλη την συνεργασία μας. Επίσης θα ήθελα να ευχαριστήσω το φοιτητή Μηχανικών Υπολογιστών Χρίστος Χατζηαντώνης για την πολύτιμη βοήθεια που μου πρόσφερε όσο αφορά την υλοποίηση δικτύου αισθητήρων. Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου για την στήριξη της.

Περίληψη

Το θέμα της διπλωματικής μου εργασίας αφορά την μελέτη της συμπεριφοράς του δικτύου ασύρματων αισθητήρων καθώς χρησιμοποιώ διάφορα σενάρια του πρωτόκολλο επικοινωνίας MQTT.

Τα δίκτυα αισθητήρων διαφέρουν από τα ασύρματα και ενσύρματα δίκτυα γι' αυτό και δημιουργείται η επιτακτική ανάγκη για μελέτη καινούργιων πρωτοκόλλων αφού τα υφιστάμενα δεν καλύπτουν τις ανάγκες των καινούργιων αυτών δικτύων. Τα δίκτυα αισθητήρων διαφέρουν από τα κοινά ασύρματα δίκτυα λόγω των διαφορετικών απαιτήσεων που έχουν από το δίκτυο και από το ότι οι κόμβοι του δικτύου έχουν διαφορετικά χαρακτηριστικά. Το ασύρματο δίκτυο αισθητήρων είναι επιρρεπές σε περισσότερα σφάλματα από τα άλλα δίκτυα, είναι αναξιόπιστο μπορεί ευκολά να χάσει την σύνδεση του και να επανέρθει μετά από λίγο. Συγχρόνως οι κόμβοι-αισθητήρες που συνδέονται πάνω στα συγκεκριμένα δίκτυα είναι πολύ μικρές συσκευές οι οποίες έχουν περιορισμένο αποθηκευτικό χώρο, περιορισμένη υπολογιστική ικανότητα όπως επίσης περιορισμένη ενέργεια. Συνεπώς γίνεται, εύκολα αντιληπτό ότι δημιουργείται η ανάγκη για την δημιουργία πρωτόκολλού όπου θα κάνει σωστή χρήση πόρων ούτω ώστε να καταναλώνει λιγότερη δυνατή μπαταρία και θα έχει μικρό bandwidth. Περισσότερα από αυτά τα χαρακτηριστικά τα διαθέτει το MQTT πρωτοκόλλου. Σκοπός της διπλωματικής είναι να επαληθεύσουμε ότι αυτά τα χαρακτηριστικά τα διαθέτει το συγκεκριμένο πρωτόκολλο.

Τέλος βλέπουμε ότι τελικά το MQTT πρωτόκολλο παρέχει αξιοπιστία έναντι αποτυχημένων κόμβων και υποστηρίζει τα πιο πολλά από τα ζητούμενα χαρακτηριστικά για να δουλεύει ομαλά ένα ασύρματο δίκτυο όπως ότι είναι ελαφρύ κάνει χρήση περιορισμένης ενέργειας.

Περιεχόμενα

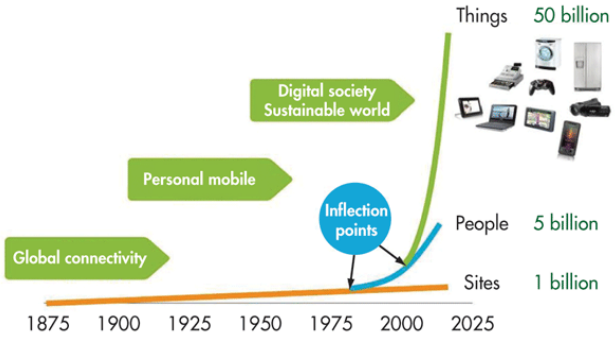
Κεφάλαιο 1

Εισαγωγή

1.1 Θέμα :

Στην διπλωματική μου εργασία ασχολήθηκα με ένα πρωτόκολλο επικοινωνίας, συγκεκριμένα το MQTT. Στην σημερινή εποχή η λέξη Internet of things είναι πολύ διαδεδομένη. Το Internet of things επιτρέπει σε συσκευές να συνδεθούν μεταξύ τους μέσω του διαδίκτυού και να αποστείλουν πληροφορίες μεταξύ τους. Η σύνδεση των συσκευών αυτών είναι πρακτική και χρήσιμη σε κάποιες περιπτώσεις. Σκοπός του να προάγει την αυτοματοποίηση διάφορων διαδικασιών χωρίς την εμπλοκή ανθρώπων.

Το internet of things έχει πολλές χρήσεις όπως στα έξυπνα σπίτια αλλά και σε πολλές άλλες περιπτώσεις που θα αναλυθούν αργότερα. Ενώ και στα προηγούμενα χρόνια υπήρχαν αυτές οι συσκευές πλέον υπάρχει ανάγκη για πιο φτηνές λύσεις. Ο πρώην διευθύνων σύμβουλος της Ericsson, Hans Vestburg ήταν από τους πρώτους που το ανέφερε σε παρουσίαση του 2010 στους μετόχους ότι το 2020 , 50 δισεκατομμύρια [Σχήμα 1.1] συσκευές θα συνδέονται μέσω του διαδικτύου. Συνεπώς όπως ειπώθηκε και πιο πάνω δημιουργείται η επιτακτική ανάγκη για καινούργιες συσκευές με καινούργιες προδιαγραφές.



Σχήμα 1.1: Στο σχήμα φαίνεται η εκθετική αύξηση των πραγμάτων που θα ενώνονται στο διαδίκτυο με το πέρασμα του χρόνου από το 1875 μέχρι 2025.

Στην διπλωματική μου εργασία ασχολήθηκα με της συσκευές Arduino και Raspberry Pi οι οποίες είναι φθηνές συσκευές που παίρνουν τα δεδομένα του αισθητήρα και τα αποστέλλουν μέσω του διαδικτύου. Αυτό γίνεται με την βοήθεια MQTT πρωτοκόλλου επικοινωνίας όπου είναι υπεύθυνό για την αποστολή δεδομένων μεταξύ των συσκευών Arduino και Raspberry Pi σε ένα δίκτυο αισθητήρων. Το συγκεκριμένο πρωτόκολλο έχει clients όπου είναι είτε publisher/Subscriber ή και τα δύο και ένα broker όπου λαμβάνει και στέλνει τα μηνύματα στους clients. Κάθε πελάτης που εκδίδει ένα μήνυμα προς το broker, περιλαμβάνει ένα θέμα στο μήνυμα. Κάθε πελάτης που θέλει να λαμβάνει μηνύματα με το συγκεκριμένο θέμα κάνει subscribe για το συγκεκριμένο θέμα. Ο broker όταν έρθει μήνυμα σε αυτό κάποιο θέμα το προωθεί στους clients που έκαναν subscriber με το συγκεκριμένο θέμα. Ως εκ τούτου, οι πελάτες δεν χρειάζεται να γνωρίζουν ο ένας τον άλλο, επικοινωνούν μόνο με το broker. Αυτή η αρχιτεκτονική είναι επεκτάσιμη, αν θέλω να προσθέσω περισσότερους clients ,γιατί δεν χρειάζεται να επικοινωνούν μεταξύ τους μόνο με το broker. Παρόλα αυτά ως προς την συμφόρηση δεν είναι επεκτάσιμη γιατί όλοι περνούν από το broker, αν χαλάσει ο broker τότε το δίκτυο πέφτει.

1.2 Σκοπός μελέτης διπλωματικής εργασίας:

Στόχος της διπλωματικής μου εργασίας ήταν η υλοποίηση και η μελέτη του συγκεκριμένου πρωτοκόλλου αν είναι αξιόπιστο και γρήγορο αν δηλαδή περιλαμβάνει όλα αυτά τα χαρακτηριστικά που ζητούν τα δίκτυα αισθητήρων.

1.3 Σημαντικότητα της μελέτης της διπλωματικής εργασίας:

Στις μέρες μας η τεχνολογία έχει αναπτυχθεί αρκετά. Οι αυτοματοποίηση των συσκευών θα είναι αναπόφευκτη. Ήδη πολλά πράγματα έχουν αρχίσει να αυτοματοποιούνται όπως κάποιες διαδικασίες του αυτοκινήτου ,το αυτόματο παρκάρισμα και αυτόματη προειδοποίηση για αποφυγή σύγκρουσης. Στο μέλλον προβλέπονται οι χρήσεις να είναι πολλές από την πιο απλή διαδικασία η καφετέρια να ανάβει μόνη της 10 λεπτά πριν ξυπνήσεις για να είναι έτοιμο ο καφές ζεστός στην πιο σοβαρή διαδικασία όπου για μια διάγνωση ιατρική δεν θα χρειάζεται γιατρός παρά μόνο ένα κομπιούτερ. Όλο αυτό οδήγησε στην ανάγκη για καινούργιες συσκευές ,καινούργια πρωτόκολλά όπου θα είναι πιο φτηνά για να μπορεί να τα αγοράζει και ένα μέσος άνθρωπος. Επίσης σημαντικό θα είναι η εύκολή εγκατάσταση για αυτές τις συσκευές, αθόρυβή λειτουργία ,η μικρή χρήση ενέργεια ,η γρήγορη απόκριση. Όλα αυτά θα μελετηθούν ώστε να βγάλουμε ένα σωστό συμπέρασμα κατά πόσο το συγκεκριμένο πρωτόκολλο ικανοποιεί κάποιες η όλες τις απαιτήσεις .

1.4 Σκελετός της διπλωματικής:

Στο 2 κεφάλαιο αναφέρω κάποιες σημαντικές πληροφορίες για το πρωτόκολλο μου, όπως ιστορικά γεγονότα και συγκεκριμένες λεπτομέρειες για το πρωτόκολλο. Επίσης στο 3 κεφάλαιο αναφέρω την μεθοδολογία που χρησιμοποίησα για το πρωτόκολλο καθώς κάνω και περιγραφή για τις συσκευές που χρησιμοποιώ .Στο 4 κεφάλαιο γίνεται μια ανάλυση διάφορων σεναρίων και γραφεί συμπερασμάτων. Τέλος γράφω το τελικό συγκεκριμένα.

Κεφάλαιο 2

Γενικό Πλαίσιο

2.1 Internet of things:

Μπαίνουμε σε μια καινούργια εποχή διαδικτυακών πραγμάτων. Βλέπουμε όλο και αυξανόμενη πανταχού παρουσία των αισθητήρων και των ενεργοποιητών όπου αλληλοεπιδρούν το ψηφιακό κόσμο με τον πραγματικό κόσμο[2].

Η έννοια του δικτύου των έξυπνων συσκευών είχε συζητηθεί από το 1982, το πανεπιστήμιο Carnegie Mellon University έφτιαξε μια coke machine η οποία μελετούσε κατά πόσο τα ποτά τα οποία φορτώθηκαν στην μηχανή ήταν κρύα καθώς και των αριθμό αυτών των ποτών που βρίσκονταν στην μηχανή[3]. Ο όρος IOT επινοήθηκε αργότερα από τον Kevin Aston της Procter & Gamble, later MIT's Auto-ID Center το 1999. Η ιδέα του Internet of things άρχισε σιγά σιγά να μπαίνει στην καθημερινότητα μας, όπου το 2013 η Global Standards Initiative on Internet of Things (IoT-GSI) ονόμασε το IOT ως «κοινωνία της πληροφορίας».

Κάποιοι από τους στόχους του IOT είναι να κάνει την ζωή μας πιο εύκολή καθώς και να επιλύσει προβλήματα της ζωής που οι άνθρωποι είναι δύσκολο η ακατόρθωτο να επιλύσουν καθώς ακόμα και να βελτιώσουν την ζωή του σύγχρονου ανθρώπου. Συνεπώς εισάγει καινοτόμους τρόπους στην ζωή μας.

Ας αναφέρουμε αρχικά τι είναι IOT, IOT αναφέρεται στην δικτύωση αντικειμένων οι οποίες συχνά είναι εξοπλισμένες με μια μορφή νοημοσύνης. Με απλά λόγια έχω συσκευές όπως αισθητήρες ενωμένους στο διαδίκτυο περισυλλέγω της πληροφορίες που μετρά ο αισθητήρας και είτε απλά εμφανίζω αυτές τις πληροφορίες στο διαδίκτυο είτε τους προσαρμόζω μια μορφή νοημοσύνης για να γίνει κάποια πράξη αυτόματα. Το IOT ήδη έχει αρχίσει να εμφανίζεται και στο μέλλον θα είναι σχεδόν σε όλες τις συσκευές. Θα χρησιμοποιείται από τα πιο απλά πράγματα μέχρι και τα πιο περίπλοκα πράγματα.

Παραδείγματα χρήσης του θα είναι στα αυτοκίνητα όπου πολλά πράγματα θα γίνουν αυτόματα ,όπως αυτόματη πλοήγηση, αποφυγή συγκρούσεων, έλεγχος ελαστικών, θα χρησιμοποιούνται πολλά είδη αισθητήρων. Επίσης θα χρησιμοποιείται και σε πιο σημαντικά θέματα όπως η γνωμάτευση ασθένειας από ένα κομπιούτερ και όχι γιατρό.
Επίσης σημαντική η ενσωμάτωση του IOT στα έξυπνα σπίτια όπου θα βοηθήσει στην ελαχιστοποίηση κατανάλωσης ενέργεια λόγω πολλών ηλεκτρονικών συσκευών. Είναι σημαντική εξέλιξη αφού θα αντικαταστήσει σε πολλούς τομείς τον ανθρώπινο παράγοντα όπου σε κάποια σημεία μειονεκτεί έναντι του υπολογιστή π.χ ο άνθρωπος έχει περιορισμένη προσοχή, ακρίβεια, περιορισμένο χρόνο.

2.2 Άλλα υπάρχοντα πρωτόκολλα για IOT:

COAP: Πρωτόκολλο εφαρμογής χρησιμοποιείται και αυτό σε δίκτυα αισθητήρων. Δουλεύει πάνω σε UDP δίκτυα και επιτρέπει στις συσκευές να ενωθούν πάνω στο διαδίκτυο. Χρησιμοποιείται από κάποιες εταιρίες κινητής τηλεφωνίας στην υπηρεσία sms επειδή έχει μικρό χρόνο latency και επίσης έχει χαμηλό κόστος[4].

AMQP : Advanced Message Queuing Protocol. Μεταφέρει δεδομένα σε δυαδική αναπαράσταση και συνδέει συστήματα διαφορετικών κατασκευαστών με επιτυχία. Είναι πιο δημοφιλή από άλλα σχετικά πρωτόκολλα και χρησιμοποιείται από μεγάλες εταιρίες για ανταλλαγή μμηνυμάτων. Χρησιμοποιείται από την JP Morgan από τη NASA και τη Google. Παρέχει ασφάλεια και αξιοπιστία

2.3 Τι είναι το MQTT protocol:

Το MQTT είναι πρωτόκολλο Client Server publish/subscribe messaging transport protocol. Το MQTT επιτρέπει στις συσκευές να στέλνουν δημόσια πληροφορίες σχετικά με ένα θέμα. Υπάρχουν πολλοί πελάτες που είναι συνδεδεμένοι με το broker. Αυτοί οι πελάτες είναι είτε publisher είτε subscriber είτε και τα δύο ταυτόχρονα. Ο publisher στέλνει ένα μήνυμα στο broker, το συγκεκριμένο μήνυμα έχει ένα θέμα. Οι subscribers εγγράφονται στο broker με ένα συγκεκριμένο θέμα. Όταν το μήνυμα σταλθεί στο broker , αυτός θα προωθήσει το μήνυμα σε όσους έχουν έκαναν subscriber με το συγκεκριμένο θέμα. Από αυτό καταλαβαίνουμε ότι οι clients δεν γνωρίζει ο ένας την ύπαρξη του άλλου. Ενώ ο broker γνωρίζει όλους τους πελάτες που είναι συνδεδεμένοι σε αυτόν.

2.3.1 Ιστορική Αναδρομή MQTT:

Message Queue Telemetry Transport. Ο Andy Stanford-Clark ο οποίος δούλευε στην IBM και ο Arlen Nipper ο οποίος δούλευε Arcom έχουν συγγράψει την πρώτη έκδοση του πρωτοκόλλου το 1999. Δημιουργήσαν αυτό το πρωτόκολλο με σκοπό την δημιουργία ενός πρωτοκόλλου το οποίο θα είχε ελάχιστη απώλεια μπαταρίας άρα χαμηλές απαιτήσεις σε ενέργεια καθώς και ελάχιστη απαίτηση σε bandwith, ώστε να ελέγχει την λειτουργία αγωγών πετρελαίου και φυσικού αερίου , δια μέσου δορυφόρου. Στόχος τους ήταν να κρατήσουν το πρωτόκολλο με μικρό κόστος. Το MQTT εξακολουθεί να είναι συνδεδεμένο με την IBM μέχρι σήμερα αλλά τώρα είναι ανοικτό πρωτόκολλο που εποπτεύεται από το OASIS(Organization for the Advancement of Structured Information Standards). Σήμερα η τρέχουσα τυπική έκδοση του OASIS είναι 3.1.1 και η έκδοση αυτή εγκρίθηκε στις 29 Οκτωβρίου 2014.

2.3.2 Χαρακτηριστικά MQTT:

Είναι light weight protocol γιατί έχει μικρό overhead της γραμμής (δυαδικό πρωτόκολλο) γι’ αυτό μειώνει την επιβάρυνση του σύρματος. Η επιδίωξη του σχεδιασμού είναι να ελαχιστοποίηση το εύρος ζώνης του δικτύου και τις απαιτήσεις σε πόρους, καθώς και να χρησιμοποιεί την μπαταρία ελάχιστα αφού απευθύνεται σε κινητές εφαρμογές . Είναι σχεδιασμένο για συνδέσεις με απομακρυσμένες τοποθεσίες. Επίσης είναι αρκετά απλό

και εύκολό στο να υλοποιηθεί. Χρήση του MQTT πάνω στο TCP/IP. Επίσης είναι M2M πρωτόκολλο. Παρέχει κανάλι αμφίδρομης επικοινωνίας. Το MQTT είναι κατάλληλο για δίκτυα όπου είναι δυναμικά δηλαδή αλλάζουν συνεχώς και δεν έχουν σταθερή θέση οι client. Χρησιμοποιείται σε δίκτυα όπου θέλω να αποστείλω πληροφορίες σε πολλούς χρήστες (1 to many μηνύματα πρωτόκολλο) με το ίδιο θέμα όπου οι client μπορεί να βρίσκονται σε απομακρυσμένες περιοχές. Είναι κατάλληλο για δίκτυα με αισθητήρες. Είναι ασύγχρονο δεν απαιτεί την ταυτόχρονη σύνδεση subscriber με publisher.

2.3.3 Εφαρμογές που χρησιμοποιούν MQTT:

1) Facebook Messenger. Το πρωτόκολλο MQTT χρησιμοποιείται στο messenger για chat.

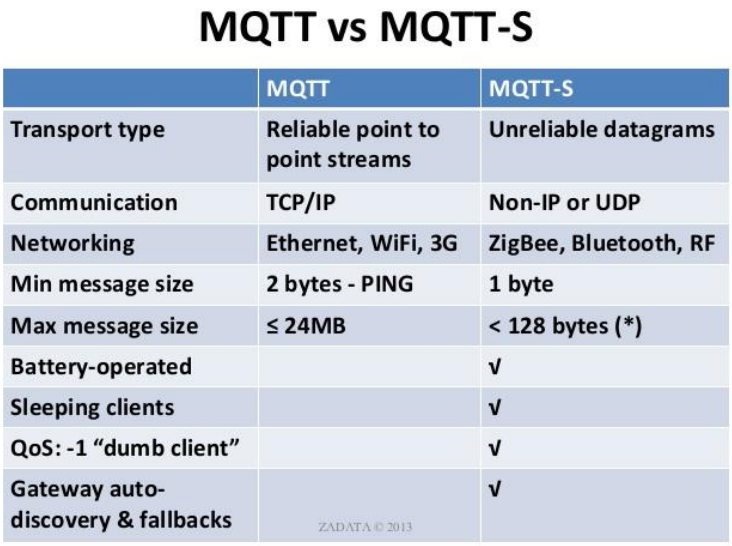
2) IECC Scalable Delta Rail's: Η τελευταία έκδοση του IECC Signaling Control System χρησιμοποιεί MQTT σε διάφορα μέρη του συστήματος για επικοινωνία.

3)The EVRYTHNG IoT platform χρησιμοποιεί MQTT στο M2M protocol για πολλές συσκευές τους.

4) On October 8, 2015, Amazon Web Services ανακοίνωσε την δημιουργία Amazon IoT βασισμένο στο MQTT.

2.4 Τι είναι το MQTT-SN protocol:

MQTT- Sensor Network. Είναι δίκτυα τα οποία αποτελούνται από αισθητήρες όπου στο κέντρο υπάρχει ένας μεσίτης. Οι αισθητήρες χρησιμοποιούνται για μέτρηση διάφορων φυσικό περιβαλλοντικών εκδηλώσεων θερμοκρασίας ,απόστασης, πίεσης, υγρασίας. Υπάρχει μια αύξηση στην χρήση αυτών των δικτύων λόγω της απλότητας ,του χαμηλού κόστους και εύκολης επέκτασης δικτύου. Είναι μια παραλλαγή του κύριου πρωτόκολλο που αποσκοπεί σε ενσωματωμένες συσκευές σε UDP δίκτυα, όπως το ZigBee.



Σχήμα 2.1: Στο σχήμα φαίνονται κάποιες διαφορές του MQTT με MQTT-S.

Μια σημαντική διάφορα του MQTT-SN είναι ότι είναι ακόμα πιο ελαφρύ πρωτόκολλο από το MQTT . Αρχικά το μικρότερο μέγεθος που μπορεί να σταλθεί ή το μέγιστο μέγεθος μηνύματος που μπορεί να σταλθεί είναι μικρότερο από του MQTT αυτό συμβαίνει γιατί ο subscriber ή ο publisher όταν θέλει να κάνει συνδρομή με το συγκεκριμένο θέμα η να στείλει μήνυμα με το συγκεκριμένο θέμα δεν στέλνει το θέμα αλλά ένα id του θέματος συνεπώς μειώνει κατά πολύ το μέγεθος του μηνύματος . Μια άλλη σημαντική διάφορα είναι ότι το δίκτυο υποστηρίζει πελάτες που κοιμούνται δηλαδή μπορεί να μην στέλνουν είτε να παραλαμβάνουν μηνύματα για ένα χρονικό διάστημα, όταν επανέλθουν σε σύνδεση τότε ο μεσίτης του στέλνει τα μηνύματα που δεν παρέλαβαν καθώς κοιμούνταν. Είναι πλεονέκτημα έναντι του MQTT γιατί συσκευές με περιορισμένη ισχύ μπορεί να καταστούν σε αναστολή, οι πελάτες μπορούν να έρθουν στο δίκτυο όταν χρειάζεται να στείλουν η να παραλάβουν μήνυμα. Επιπλέον το MQTT παρέχει 27 τύπους πακέτου ελέγχου συμπεριλαμβανομένων connect, publish, subscribe, unsubscribe, disconnect. Επίσης παρέχει ένα 4 Qos 0,1,2 όπου θα εξηγηθούν αργότερα και το -1 σε αντίθεση με το MQTT που παρέχει μόνο 3 Qos , το Qos -1 είναι για απλούς χρήστες αυτός ο πελάτης δεν ενδιαφέρεται για το κατά πόσον η πύλη είναι ενεργή και το αν παραδίδεται το μήνυμα. Συνεπώς το MQTT-SN είναι πιο ελαφρύ δίκτυο και καταναλώνει λιγότερη ενέργεια από το MQTT όμως είναι λιγότερο δημοφιλή και θα το βρεις σε ελάχιστες εφαρμογές αφού δεν είναι κατάλληλο για εφαρμογές iot δεν μπορεί να εγγραφτεί σε πολλά θέματα και έχει πολλές απώλειές μηνυμάτων αφού δουλεύει πάνω σε UDP[6].

2.5 MQTT security:

Δεν διαθέτει κρυπτογράφηση, έχει ελάχιστες δυνατότητες ελέγχου ταυτότητας ενσωματωμένες στο πρωτόκολλο για ασφαλή χρήση των πληροφοριών. Κάθε μήνυμα όπως εξηγήσαμε πιο πάνω αποτελείται από 2 bytes όπου το μήνυμα έχει την μορφή ενός απλού κειμένου. Εντούτοις υπάρχει δυνατότητα να χρησιμοποίησης για ασφάλεια το TLS/SSL το οποίο δεν κάνει το πρωτόκολλο ελαφρύ.

Κεφάλαιο 3

Μεθοδολογία

3.1 Παράμετροι του αλγόριθμου MQTT:

Το πρωτόκολλο έχει κάποιους συγκεκριμένους παραμέτρους όπου μπορεί να αλλαχτούν σύμφωνα με τις απαιτήσεις που έχει κάποιος από το δίκτυο του. Κάποιες παράμετροι είναι Qos του δικτύου, retain message ,persistent mode message. Όπου θα εξηγηθούν πιο κάτω.

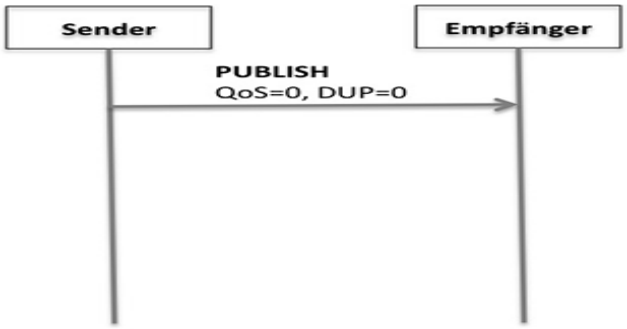
3.1.1 Qos : Quality of service:

Τα σύγχρονα δίκτυα έχουν απαιτήσεις στην ποιότητα εξυπηρέτησης. Η χρήση του Quality of service στα δίκτυα, βοηθά στην διαχείριση πόρων του δίκτυού μας. Συγχρόνως δίνει προτεραιότητα σε κάποιο client σε σχέση με τους άλλους προκειμένου να λειτουργήσει το δίκτυο μας αποτελεσματικά σύμφωνα με τις ανάγκες του κάθε clients. Στη περίπτωση μας έχουμε ένα δίκτυο με sensor όπου συλλέγει πληροφορίες από διάφορους sensors,όπου κάποια από αυτά τα δεδομένα είναι ευαίσθητα στο χρόνο όπως θερμοκρασία. Άρα καταλαβαίνουμε ότι θα πρέπει να κάποιοι κόμβοι να έχουν προτεραιότητα από το δίκτυο να δημοσιεύσουν τα δεδομένα τους, όπου μια καθυστέρηση θα δώσει λάθος αποτελέσματα αυτό γίνεται με το qos TCP. Επίσης το πρωτόκολλο MQTT παρέχει Qos με σκοπό την αξιοπιστία του δικτύου δηλαδή κατά πόσο είμαστε

σίγουρη ότι θα παραδοθεί το μήνυμα. Από όλα τα παραπάνω γίνεται φανερό ότι αν χρησιμοποιήσουμε σε ένα δίκτυο τις ιδιότητες του Qos θα έχουμε σωστή διαχείριση πόρων και το δίκτυο μας θα προσφέρει υψηλής ποιότητας υπηρεσίες.

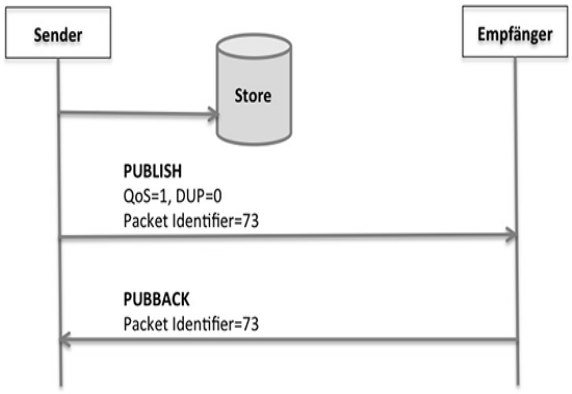
Στο πρωτόκολλο που μελετώ MQTT υπάρχουν 3 επίπεδα Qos:

Qos 0(το πολύ μια φορά): Το πρωτόκολλο κάνει προσπάθεια για να στείλει το μήνυμα αλλά υπάρχει πιθανότητα να μην παραδοθεί το μήνυμα. Η παράδοση του μηνύματος εξαρτάται από τις δυνατότητες του δικτύου και της σύνδεσης. Προτιμάται σε περιβάλλοντα όπου και να χαθεί ένα μήνυμα δεν επηρεάζει το αποτέλεσμα π.χ σε ένα αισθητήρα υγρασίας όπου στέλνει κάθε 10 δευτερόλεπτα αν χαθεί 1 μήνυμα δεν θα επηρεάσει το δίκτυο.



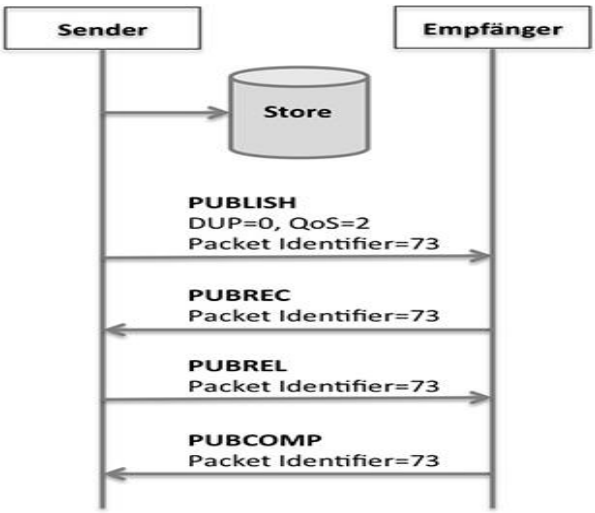
Σχήμα 3.1: Qos 0.

Qos 1(τουλάχιστον μια φορά): Σίγουρα το μήνυμα θα φτάσει στο παραλήπτη αλλά υπάρχει πιθανότητα να φτάσει περισσότερο από μια φορά. Χρησιμοποιείται σε περίπτωση που η αποστολή μηνύματος δεν στοιχίζει πολύ και είναι σημαντικό το μήνυμα να φτάσει π.χ σε συστήματα ασφάλειας .



Σχήμα 3.2: Qos 1.

Qos 2(ακριβώς μια φορά): Το μήνυμα σίγουρα θα φτάσει στο παραλήπτη ακριβώς μια φορά. Πρέπει να χρησιμοποιείται με φειδώ γιατί είναι πιο αργό σε σχέση με τα άλλα. Χρησιμοποιείται σε δίκτυα που δεν πρέπει να χαθεί έστω και ένα μήνυμα και ούτε να έχει διπλά αντίγραφα κάποιου μηνύματος.



Σχήμα 3.3: Qos 2.

Ο κάθε client μπορεί να κάνει subscribe,publish στο κάθε θέμα με διαφορετικό Qos , αφού είναι ανεξάρτητα μεταξύ τους. Όταν ένα μήνυμα κάνει publish με ένα συγκεκριμένο qos και ο πελάτης που παραλαμβάνει το μήνυμα με ένα διαφορετικό qos μικρότερο αυτό που ισχύει στην παράδοση είναι το qos του subscriber αν είναι μεγαλύτερος τότε ισχύει του publisher[7].

3.1.2 Retain message:

Όταν σταλθεί ένα μήνυμα από publisher και δεν υπάρχει κανένας subscriber με το συγκεκριμένο θέμα τότε το μήνυμα χάνεται. Όμως στην περίπτωση που το flag retain είναι true του publisher τότε το τελευταίο μήνυμα φυλάγεται από το broker ούτω ώστε όλοι οι subscriber του συγκεκριμένου θέματος να γνωρίζουν την τελευταία κατάσταση ,π.χ ένας αισθητήρας όπου αναφέρει αν η πόρτα είναι ανοικτή ή κλειστή ,αν το μήνυμα αναφέρει ίδια κατάσταση να ξέρει να μην κάνει οποιαδήποτε πράξη σπαταλώντας έτσι πόρους. Μόνο ένα μήνυμα διατηρείται από κάθε θέμα. Το επόμενο όπου θα δημοσιευτεί με retain flag είναι true θα αντικαταστήσει το τελευταίο.

Retain Message, Clean Session and QOS Table

Clean Session Flag	Retain Flag	Subsribe QOS	Publish QOS	Published Message Always Reeeived
True	False	0	0	No
True	False	0	1	No
True	False	1	0	No
True	False	1	1	No
False	False	0	0	No
False	False	0	1	No
False	False	1	0	No
False	False	1	1	Yes – All messages
True	True	0	0	Yes –Last Message only
True	True	0	1	Yes –Last Message only
True	True	1	0	Yes –Last Message only
True	True	1	1	Yes –Last Message only
False	True	0	0	Yes –Last Message only
False	True	0	1	Yes –Last Message only
False	True	1	0	Yes –Last Message only
False	True	1	1	Yes – All messages

Note: QOS 1 and QOS 2 produce same result. Therefore for QOS 1 in the table read 1 or 2.

Σχήμα 3.4: Πίνακας όπου δείχνει πώς να αρχικοποιήσω το κώδικα μου για να έχω το ανάλογο αποτέλεσμα που θέλω.

3.1.3 Persistent Session and Queuing Messages

Όταν client κάνει subscribe πάνω σε ένα θέμα με Qos 1,2 τότε σημαίνει ότι θέλει να του σταλθούν όλα τα μηνύματα με το συγκεκριμένο θέμα. Υπάρχει πιθανότητα ο συγκεκριμένος client να αποσυνδεθεί για ένα μικρό χρονικό διάστημα ή μεγάλο, από κάποιο σφάλμα. Με την επίμονη συνεδρία διασφαλίζω ότι όταν ο client ξανασυνδεθεί τα μηνύματα που στάλθηκαν καθώς ήταν αποσυνδεδεμένος θα τα παραλαμβάνει και δεν θα

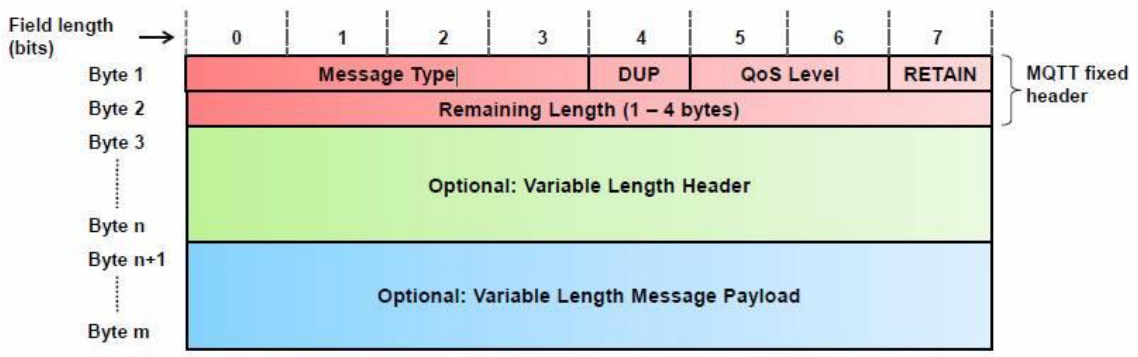
χρειαστεί να ξαναεγγραφτεί πάλι ο client στο συγκεκριμένο θέμα όπου θα ήταν σπατάλη πόρων . Έτσι όταν εγγραφτεί για πρώτη φορά κατά την επίμονη συνεδρία αποθηκεύει όλες τις πληροφορίες σχετικά με το client και το broker. Η κάθε συνεδρία έχει αναγνωριστικό το clientid του κάθε client, όπου κατά την σύνδεση του συγκεκριμένου client εξετάζει με το clientid αν υπάρχει στην επίμονη συνεδρία. Στην συνεδρία αποθηκεύονται Ο broker φυλάει τα μηνύματα σε μια queuqe και όταν ο client συνδεθεί τα στέλνει σε αυτό. Πώς εγγράφεται ένας subscriber σε επίμονη συνεδρία, με την λέξη clean session: true σημαίνει Non durable(temporary) queuqe θα δημιουργηθεί όταν ο πελάτης προσθέτει συνδρομή Qos 1. Αυτή η ουρά θα διαγραφεί όταν ο client αποσυνδεθεί για κάποιο λόγο. Όταν clean session: false θα δημιουργηθεί durable queque όπου θα διατηρεί τα μηνύματα ακόμα και μετά την αποσύνδεση του client. Τα μηνύματα στην ουρά παραμένουν μέχρι να ξανασυνδεθεί ο client αν δεν ξανασυνδεθεί τότε είναι αποθηκευμένα μέχρι να φτάσει στο όριο μνήμης του λειτουργικού συστήματος.

3.2 Αλγοριθμική περιγραφή κώδικα:

Το πρωτόκολλο είναι υπεύθυνό για την αποστολή μηνυμάτων ανάμεσα σε ένα δίκτυο. Στο δίκτυο υπάρχουν κόμβοι ένας broker και raspberry pi. Οι κόμβοι στην προκειμένη περίπτωση είναι sensor που ενώνονται με wifi connection(esp8266). Οι κόμβοι και ο broker επικοινωνούν μεταξύ τους στέλνοντας ο ένας πακέτα ελέγχου στον άλλο και αντίστροφα. Υπάρχουν 14 διαφορετικοί τύποι μηνυμάτων[Σχήμα 3.5] τα οποία έχουν μια συγκεκριμένη μορφή [Σχήμα 3.6].

Control packet	Direction of flow	Description
CONNECT	Client to Server	Client request to connect to Server
CONNACK	Server to Client	Connect acknowledgment
PUBLISH	Client to Server or Server to Client	Publish message
PUBACK	Client to Server or Server to Client	Publish acknowledgment
PUBREC	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	Client to Server	Client subscribe request
SUBACK	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	Client to Server	Unsubscribe request
UNSUBACK	Server to Client	Unsubscribe acknowledgment
PINGREQ	Client to Server	PING request
PINGRESP	Server to Client	PING response
DISCONNECT	Client to Server	Client is disconnecting

Σχήμα 3.5: Στο σχήμα φαίνονται οι 14 διαφορετικοί τύποι πακέτων ελέγχου που στέλνονται μεταξύ του client και του broker.



Σχήμα 3.6: Γενική μορφή μηνυμάτων MQTT.

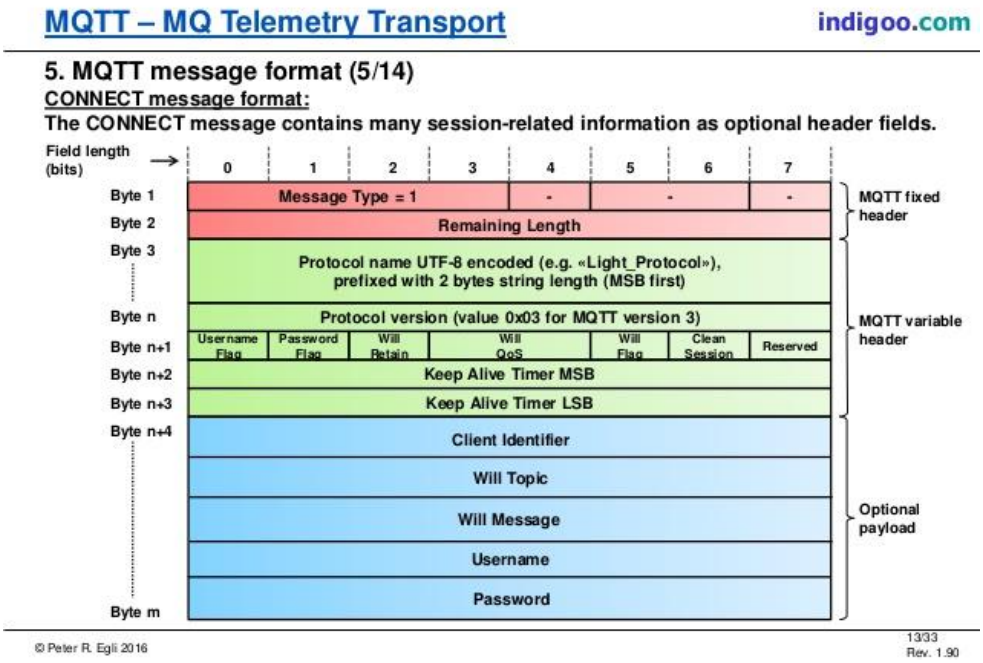
Το μήνυμα αποτελείται από 3 μέρη .Το επίπεδο Fixed header[Σχήμα 3.7], υπάρχει σε όλα τα MQTT μηνύματα. Ενώ το επίπεδο Variable header και Payload υπάρχουν σε ορισμένα μηνύματα MQTT. Το fixed header είναι 2 bytes. Παρόλο που τα δύο επίπεδα προσθέτουν επιπλέον φόρτο στο μήνυμα εντούτοις χρειάζονται για να σταλθεί σωστά το μήνυμα και για να αποφευχθούν συγκρούσεις[5].

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Σχήμα 3.7: Fixed header.

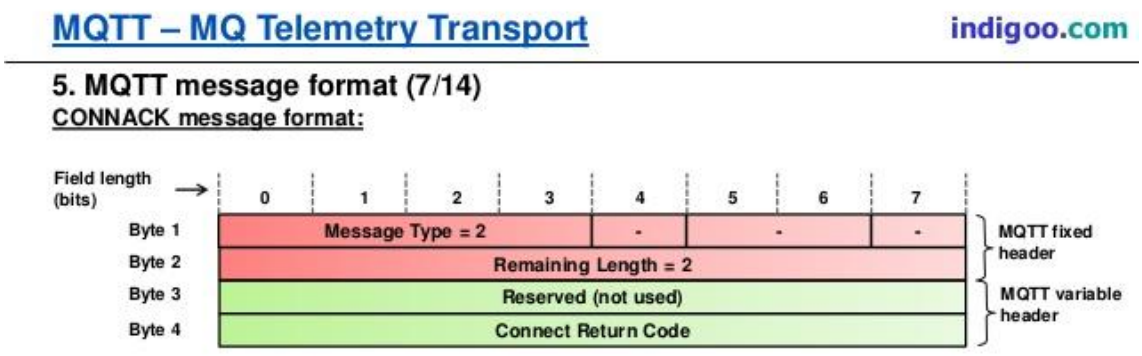
Στο πρώτο byte στέλνονται τα χαρακτηριστικά του μηνύματος. Αρχικά στα πρώτα 4 bits δηλώνεται ο τύπος του μηνύματος. Υπάρχουν 16 πιθανοί τύποι μηνύματος[Σχήμα 3.5]. Η σημαία DUP δείχνει αν το μήνυμα είναι διπλότυπο. Στα επόμενα 2 bits δηλώνεται ο τύπος υπηρεσίας και τέλος αν επιθυμούμε το μήνυμα να διατηρηθεί retain.Στο δεύτερό byte 2 βρίσκεται Remaining Length είναι ο αριθμός των bytes που έχουν μείνει με κωδικοποιημένο τρόπο.

Αρχικά για να συνδεθούν οι κόμβοι με το broker αποστέλλουν ένα πακέτο ελέγχου τύπου connect[Σχήμα 3.8] και ο broker απαντά σε αυτό το πακέτο με πακέτο ελέγχου τύπου connack[Σχήμα 3.9]. Αν η απάντηση είναι 0 η σύνδεση είναι επιτυχής τότε ο sensor τώρα έχει συνδεθεί με το broker. Η σύνδεση είναι M2M επειδή συνδέονται 2 μηχανές και επικοινωνούν μεταξύ τους.



Σχήμα 3.8 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος connect.

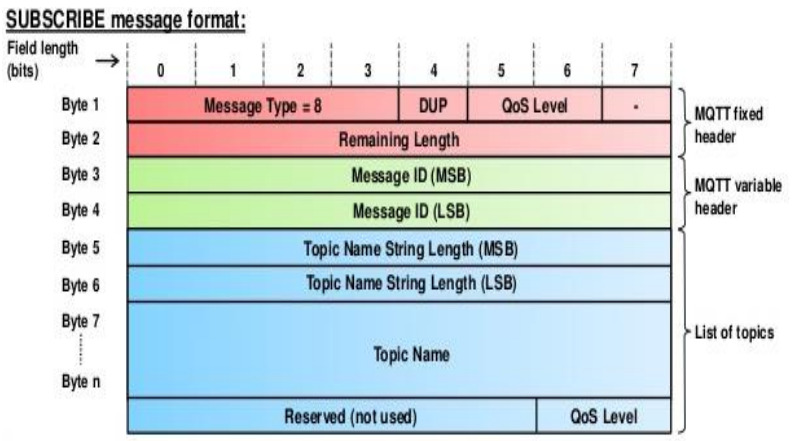
Το μήνυμα connect αποτελείται και από τα 3 μέρη. Στο επίπεδο του fixed header το Message type του είναι 1(0001). Επίσης δεν παρέχει τα άλλα flags. Στο επίπεδο variable header και optional δίνονται επιπλέον πληροφορίες για το πρωτόκολλο και το client. Λόγω του ότι η σύνδεση είναι ασύγχρονη ονομάζεται half-open δημιουργείται το πρόβλημα ότι αν κάποιος αποσυνδεθεί και ο άλλος δεν το γνωρίζει θα συνεχίσει να στέλνει μηνύματα και θα περιμένει ανταπόκριση την οποία δεν θα πάρει ποτέ. Υπάρχουν δυο περιπτώσεις που μπορώ να χρησιμοποιήσω. Πρώτη περίπτωση αναφέρεται σε μήνυμα όπου στέλνεται στους ενδιαφερόμενους client από το broker όταν ένας client σταματήσει την λειτουργία του απότομα, είναι ένα μήνυμα που συμφωνείται από πριν π.χ μπορεί να είναι το μήνυμα ”offline”(will message). Δεύτερή περίπτωση είναι μετά από κάποιο καθορισμένο χρόνο ο οποίο ο broker δεν στέλνει κάποιο μήνυμα μπορώ να του στείλω PINREG[Σχήμα 3.17] για να δω αν είναι ζωντανός ή όχι(keep alive timer).Αν είναι ζωντανός θα απαντήσει με το μήνυμα PINGRESP[Σχήμα 3.17].



Σχήμα 3.9: Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος connack.

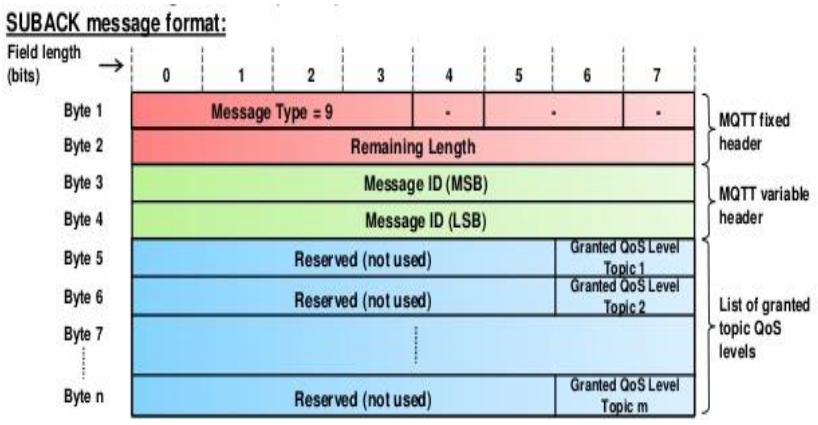
Το μήνυμα connack αποτελείται από 2 μέρη το fixed header όπου υπάρχει σε όλα τα μηνύματα και το variable header. Στο variable header τα πρώτα 8 bits(1 byte) είναι κρατημένο για μελλοντική χρήση. Το επόμενο byte είναι η απάντηση που θα δοθεί στο connect message δηλαδή στην προσπάθεια ενός πελάτη να συνδεθεί. Υπάρχουν 256 διαφορετική συνδυασμοί όπου αποτελούν την απάντηση. Η απάντηση: 0: Η σύνδεση ήταν επιτυχής ,1: Η σύνδεση απορρίφτηκε λόγω unacceptable protocol version,2: Η σύνδεση απορρίφτηκε λόγω indentifier reject, 3: Η σύνδεση απορρίφτηκε λόγω server unavailable, 4: Η σύνδεση απορρίφτηκε λόγω bad username or password, 5: Η σύνδεση απορρίφτηκε λόγω client not authorized, 6-22 : κρατημένα για μελλοντική χρήση.

Οι clients μπορεί να είναι είτε publisher είτε subscriber είτε και τα δύο. Δηλαδή κάποιοι θέλουν να στέλνουν μηνύματα με συγκεκριμένο θέμα και κάποιοι να παραλαμβάνουν μηνύματα με συγκεκριμένο θέμα. Στην προκειμένη περίπτωση subscribers είναι clients στο node.js και publishers είναι sensor. Όταν ένας πελάτης κάνει εγγραφεί σε ένα συγκεκριμένο θέμα στέλνει πακέτο ελέγχου με τύπο subscribe[Σχήμα 3.10]. Επίσης μπορεί να διαγράψει την συγκεκριμένη εγγραφή με το πακέτο ελέγχου unsubscribe[Σχήμα 3.12]. Ο broker απαντά στο client με το μήνυμα τύπου suback[Σχήμα 3.11] στο μήνυμα subscribe και με μήνυμα τύπου unsuback[Σχήμα 3.13]. Βλέπουμε ότι ένα client μπορεί να γραφτεί σε περισσότερα από ένα θέματα και στέλνονται ταυτόχρονα με ένα πακέτο ελέγχου.



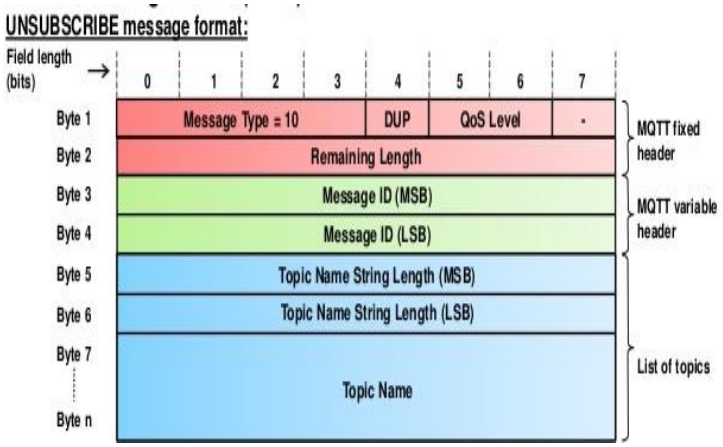
Σχήμα 3.10 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος subscribe.

Το μήνυμα subscribe αποτελείται από το fixed header,variable header, payload. Στο επίπεδο fixed υπάρχουν το message type και όλα τα flag εκτος retain που βρίσκεται μόνο στο publish message. Υπάρχει το message id στην περίπτωση Qos 1 ,2. Τέλος υπάρχουν τα θέματα και το μήκος των θεμάτων που θα εγγραφτεί πάνω καθώς και το Qos.



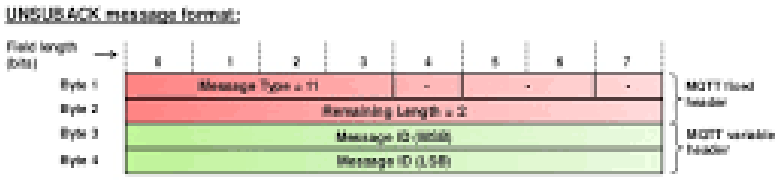
Σχήμα 3.11 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος suback.

Το μήνυμα suback αποτελείται από το fixed header το οποίο περιέχει το message type και το remaining length. Στο επίπεδο variable header υπάρχει το message id. Τέλος υπάρχει λίστα με το qos level του κάθε topic.



Σχήμα 3.12 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος unsubscribe.

Το μήνυμα unsubscribe έχει την ίδια μορφή με το subscribe με μόνη διαφορά ότι δεν υπάρχει το Qos.

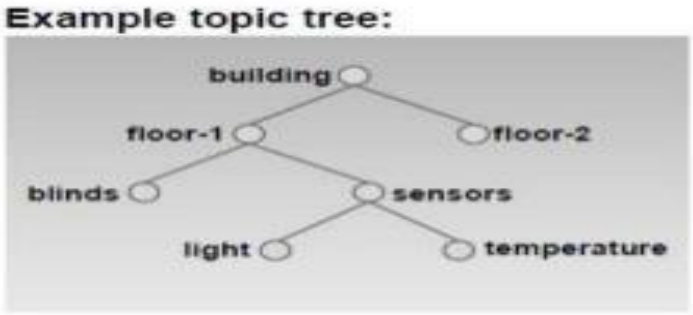


Σχήμα 3.13: Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος unsuback.

Στο μήνυμα unsuback message απλά περιέχει το message type, remaining length και το message id του θέματος που θέλω να μην είμαι πλέον γραμμένος .

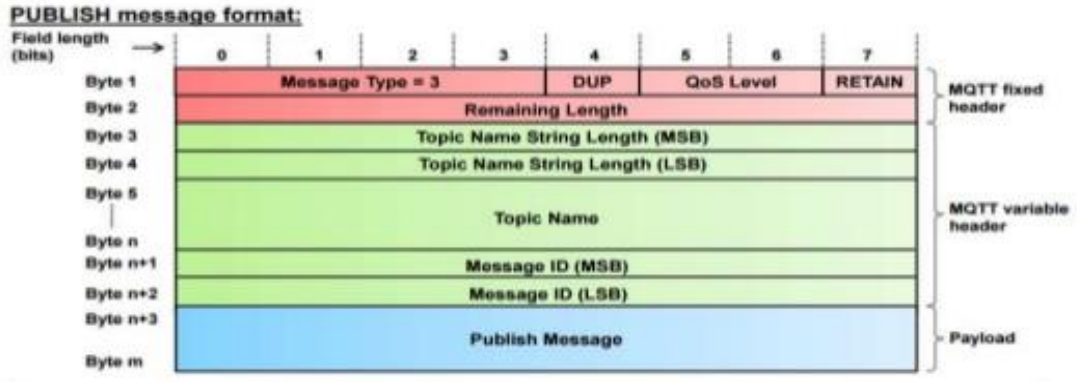
Στην περίπτωση που ένας client θέλει να στείλει ένα μήνυμα, στέλνει πακέτου ελέγχου τύπου publish[Σχήμα 3.15]. Το μήνυμα πρέπει να έχει ένα συγκεκριμένο θέμα. Κάθε θέμα δημιουργεί μια ουρά δυναμικά στην βάση δεδομένων mongo.

Topics of MQTT:



Σχήμα 3.14: Στο σχήμα φαίνεται πώς δημιουργείται ένα δέντρο από topic.

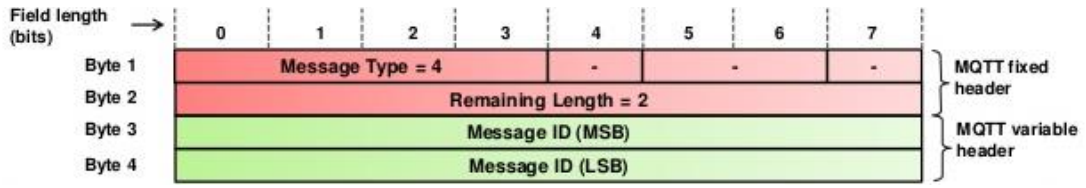
Ο broker δέχεται οπουδήποτε θέμα, το οποίο είναι 1 χαρακτήρας ή και περισσότερους. Το κάθε θέμα αποτελεί τεχνικά μια ουρά από μηνύματα στο broker. Το θέμα αποτελείται από χαρακτήρες string. Επίσης χρησιμοποιούνται και τα σύμβολα /,+,# . Το σύμβολο / ξεχωρίζει το level[Σχήμα 3:14 παράδειγμα buildings\floor-1\blinds]. Το σύμβολο + συμβολίζει οποιοδήποτε 1 level[Σχήμα 3:14 παράδειγμα buildings\floor-1\+ το + μπορεί να αντιπροσωπεύει το sensors. Το σύμβολο # μπαίνει στο τέλος και συμβολίζει 1 η και περισσότερα level[Σχήμα 3:4 παράδειγμα buildings\floor-1\# .Το # αντιπροσωπεύει είτε blinds είτε sensors είτε sensors\light].



Σχήμα 3.15 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος publish.

Το μήνυμα Publish αποτελείται και από τα 3 μέρη. Στο επίπεδο fixed header, message type είναι 3 (0011) , DUP και Qos level και RETAIN θα αρχικοποιηθούν αναλόγως. Στο επίπεδο variable header φαίνεται το θέμα του μηνύματος καθώς και το μήκος του. Όπως επίσης και το message id στην περίπτωση που το Qos είναι 1 ή 2 , για να δοθεί απάντηση στο συγκεκριμένο μήνυμα .Τέλος στο επίπεδο payload είναι το περιεχόμενο του μηνύματος που θέλουμε να στείλουμε.

Στην περίπτωση που Qos 1 τότε ο αποστολέας περιμένει την απάντηση PUBACK για να καταλάβει ότι το μήνυμα στάλθηκε στην περίπτωση που δεν υπάρχει απάντηση ξαναστέλνει και κάνει το flag DUP 1. Στην περίπτωση που Qos 2 τότε ο αποστολέας περιμένει την απάντηση PUBREC[Σχήμα 3.16] θα απαντήσει ξανά με το μήνυμα PUBREL και θα πάρει την απάντηση PUBCOMP.



Σχήμα 3.16 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος puback,pubrec,pubrel.

Τα μηνύματα αυτά έχουν το ίδιο μορφότυπο δηλαδή αποτελούνται από 2 μέρη. Στο fixed header το μόνο που είναι διαφορετικό είναι το message type, όπου κάθε μήνυμα έχει διαφορετικό αριθμό. Τέλος στο variable header απλά βρίσκεται το message id.

Τέλος αν ένας client θέλει να αποσυνδεθεί στέλνει μήνυμα τύπου disconnect[3.17].

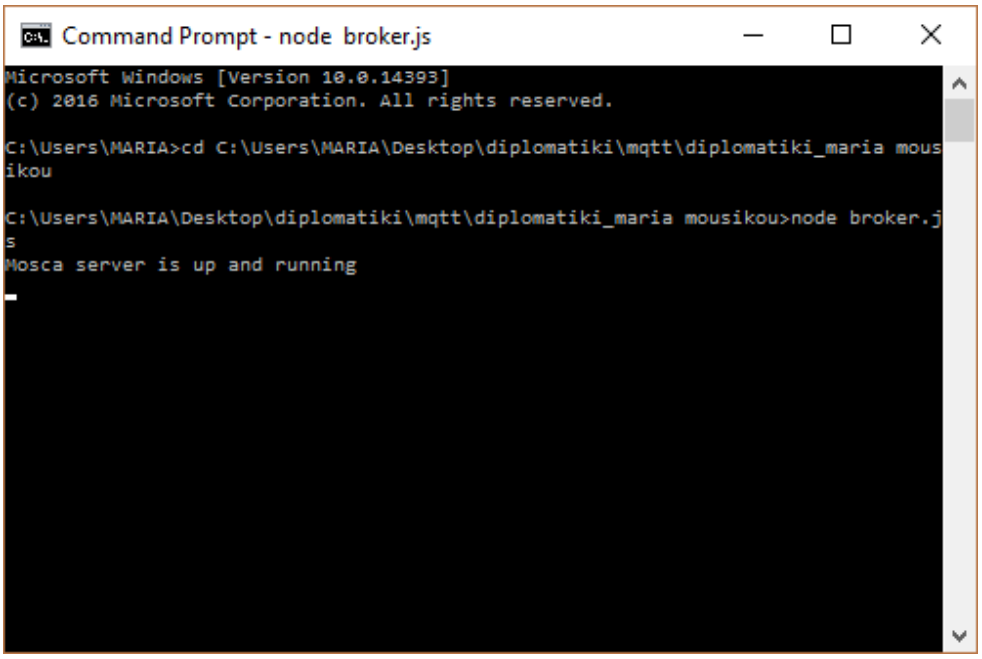
Σχήμα 3.17 : Στο σχήμα φαίνεται η μορφή συγκεκριμένου μηνύματος disconnect, pingreq, pingresp.

Τα μηνύματα disconnect, pingreq, pingresp αποτελείται μόνο από το επίπεδο fixed header. Remaining Lenght =0 , αφού δεν υπάρχουν τα άλλα επίπεδα.

3.3 Μεθοδολογία:

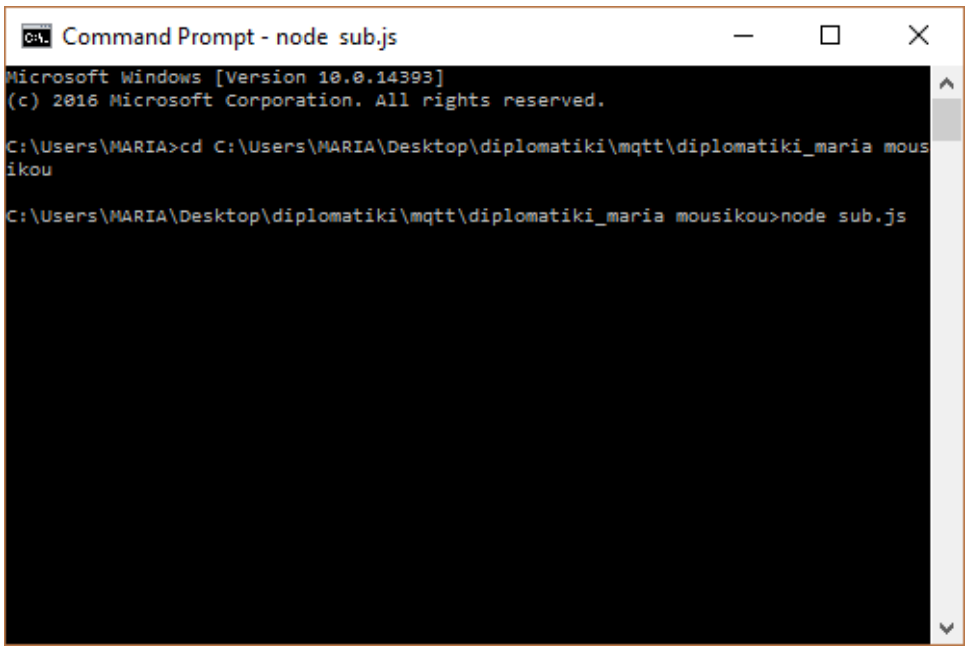
Στην διπλωματική μου ασχολήθηκα με την πειραματική μελέτη του συγκριμένου πρωτοκόλλου. Αρχικά δημιούργησα το δίκτυο με την βοήθεια node.js. Για να φτιάξω το δίκτυο επέλεξα το mosca server όπου θα είναι ό κεντρικός κόμβος του δικτύου.

1. Αρχικά τρέχω το mosca. Υλοποιήσαμε το πρωτόκολλο επικοινωνίας MQTT χρησιμοποιώντας mosca server. Για να υλοποιήσουμε το mosca server χρησιμοποιήσαμε το open source του mosca server από το github σε περιβάλλον node.js, γράφοντας σε γλώσσα javascript.

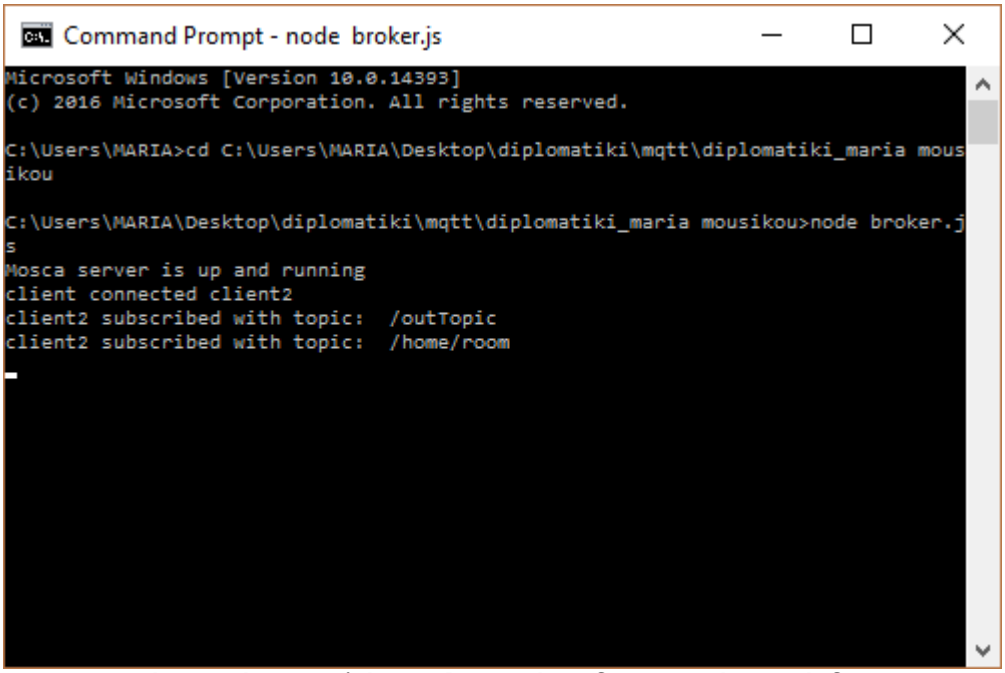


Σχήμα 3.18 : Στο σχήμα φαίνεται ότι τρέχουμε το server mosca.

2. Επίσης τρέχω ένα client όπου θα είναι subscriber που θα κάνει subscribe 2 topic. Στην συνέχεια θα δημιουργήσω το δίκτυο των αισθητήρων όπου οι αισθητήρες είναι publisher στέλνουν μήνυμα στο mosca και αυτός προωθά τα μηνύματα στο subscriber.



Σχήμα 3.19 : Τρέχει ο πελάτης με το αναγνωριστικό client2.



Σχήμα 3.20. Φαίνεται ότι ο πελάτης client2 έχει δημιουργήσει σύνδεση με το mosca server.

3.4 Μετρήσεις:

Τις διάφορες μετρήσεις τις πείρα είτε με την βοήθεια του wireshark είτε με την βοήθεια τις πλατφόρμας node.js, με την εισαγωγή βιβλιοθηκών από την npm.

Επίσης μπορούσα να βρω διάφορες μετρήσεις γράφοντας ένα ανάλογο κώδικα. Π.χ για να υπολογίσω το latency υπολόγιζα το χρόνο που το μήνυμα δημιουργείται στο publisher (χρόνο υπολογισμένος με συνάρτηση που μετρά milliseconds από 1/1/1970) και υπολογίζω και το χρόνο που το μήνυμα έφτασε στο subscriber και βρίσκω το latency του.

$$Total\ Latency\ = \sum_{k=1}^{n}(Recv\ Time(k) - Sent\ Time(k))..$$

$$Average\ Latency = Total\ Latency\ /\ Total\ Packets\ Received\ .$$

Πρόβλημα:

1. Ο sensor δεν περιέχει ρολόι

2.Έπρεπε να βρούμε μια χρονική μέτρηση όπου θα μπορούσε να υπολογιστεί και για το esp8266 και για το subscriber για να μπορούμε να πάρουμε μετρήσεις.

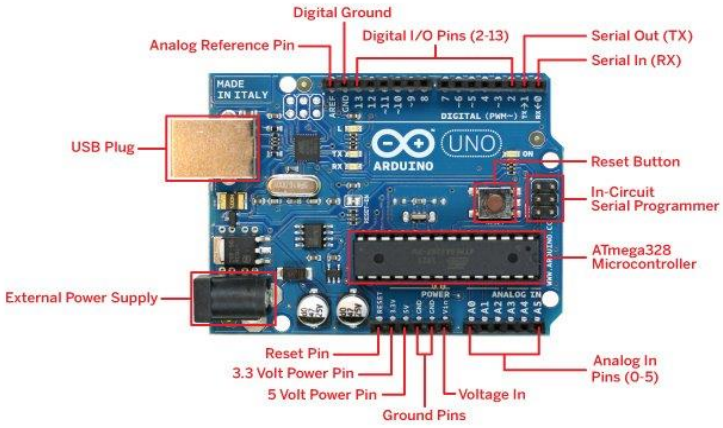
Λύση:

Για sensors παίρνουμε χρόνο seconds από 1/1/1970 μέσω time protocol[Παράρτημα],και για το subscriber μέσω συνάρτησης new Date().getTime()) όπου αντίστοιχα μου δίνει το χρόνο τώρα σε διαφορά από το 1/1/1970. Για να μπορώ να κάνω τις πράξεις.

Κεφάλαιο 4

Υλοποίηση

4.1 Συσκευές και λογισμικό που χρησιμοποιήθηκε:



Σχήμα 4.1: Arduino uno.

Ονομάζεται uno λόγω της έκδοσης λογισμικό Arduino (ide) 1.0. Είναι ένας μικροέλεγκτης μονής πλακέτας δηλαδή μια απλή μητρική πλακέτα ανοικτού κώδικα με ενσωματωμένο μικροελεγκτής και εισόδους και εξόδους η οποία μπορεί να προγραμματιστεί με τη γλώσσα wiring(c++) .Διαθέτει 14 ψηφιακές εισόδους/εξόδους. Στην διπλωματική μου χρησιμοποίησα την είσοδο TX μετάδοση στοιχείου ,RX λήψη στοιχείων ,RESET κάνει reset το κώδικα,USB PLUG το Arduino ενώνεται με θύρα usb πάνω στο υπολογιστή για να μπορεί να γίνει upload ο κώδικας μας, EXTERNAL POWER SUPPLY μπορούμε να συνδέσουμε με ένα καλώδιο usb η τροφοδοτικό με

εναλλασσόμενο ρεύμα 5V. Το Arduino έρχεται προγραμματισμένο με ένα bootloader που επιτρέπει στους χρήστες του να ανεβάσουν απευθείας κώδικα χωρίς οποιαδήποτε άλλη επεξεργασία από πριν. Είναι σημαντικό να αναφερθεί ότι το Arduino παρέχει επιπλέον ασφάλεια, περιέχει πολύπλεξη που προστατεύει τις θύρες υπολογιστή μας[1].

Προτιμάται από πολλούς λόγω του ότι είναι φτηνή σχετικά συσκευή ,προγραμματίζεται ευκολά και περιέχει πολλές έτοιμες βιβλιοθήκες.

Σχήμα 4.2: Raspberry pi

Το raspberry pi είναι ένας πλήρης υπολογιστής με μέγεθος πιστωτικής κάρτας. Περιέχει τετραπύρηνο επεξεργαστή. Υπολογιστή ισχύς συγκεντρώνεται σε τόσο λίγο χώρο και με τόσο χαμηλό κόστος.

Node js

Node js είναι μια πλατφόρμα ανάπτυξης λογισμικού χτισμένο σε περιβάλλον javascript. Χρησιμοποιεί ασύγχρονη επικοινωνία. Έχει πολλές υλοποιημένες βιβλιοθήκες που μπορείς να χρησιμοποίησης.

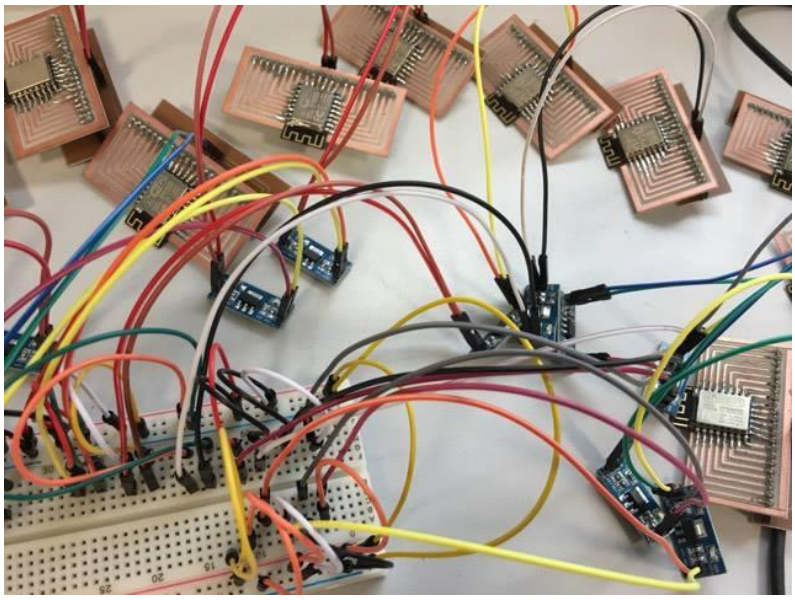
Esp8266

Χαμηλού κόστους chip όπου με την χρήση πλήρης στοίβας TCP/IP επιτρέπει τις συσκευές να συνδεθούν στο δίκτυο μέσω wifi.

Σχήμα 4.3 sensor.

Ηλεκτρονικό στοιχείο όπου ανιχνεύει τις αλλαγές σε κάποιους τομείς στο περιβάλλον και μεταδίδει αυτές τις πληροφορίες κάπου άλλου. Οι αισθητήρες συνδέουν πληροφορίες από φαινόμενα του πραγματικού κόσμου και μετατρέποντας σε αυτές σε τέτοια μορφή για να μπορούν να επεξεργαστούν και να αποθηκεύσουν. Υπάρχουν διάφοροι τύποι αισθητήρων όπως υγρασία θερμοκρασία πίεσης[Σχήμα 4.3]. Είναι μικρού μεγέθους ,χωρίς ιδιαίτερη υπολογιστική δύναμη, μη ανανεώσιμη ενέργεια συνήθως.

4.2 Διαδικασία δημιουργίας πραγματικού δικτύου:

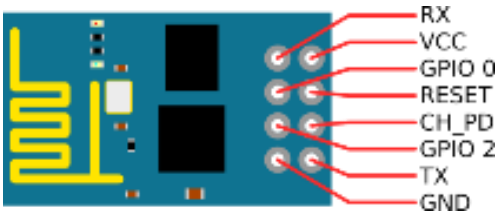


Σχήμα 4.4. Φωτογραφία από το δίκτυο αισθητήρων που δημιούργησα.

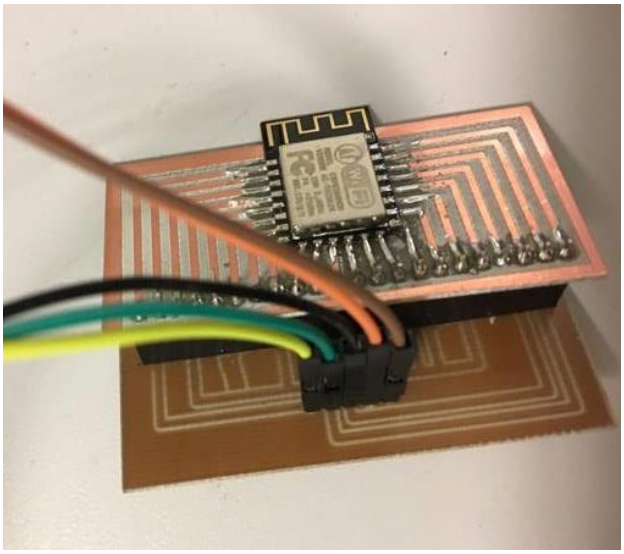
Δημιουργήσαμε διάφορους πελάτες είτε εικονικούς στην πλατφόρμα node.js είτε πραγματικούς.

Πώς δημιουργώ πραγματικούς πελάτες:

Διαδικασία για ένα πελάτη: Ενώνω τους sensor μου το wifi module esp8266, το οποίο παρέχει σύνδεση με το internet και το arduino.[Σχήμα 4.6]. Κάθε client χρειάζεται 3.3V για να συνεχίσει να δουλεύει.



Σχήμα 4.5: Στην εικόνα απεικονίζεται ένα esp8266.



Σχήμα 4.6. Στην εικόνα απεικονίζεται ένα esp8266 ενωμένο με πλακέτα.

Πώς συνδέεται esp8266 με το Arduino [Σχήμα 4.5]:

GND, GPIO 0 ενώνεται με το – από το usb cable παίρνω από 5 V από τη θύρα του υπολογιστή.

VCC, CH_PD ενώνεται με το + από το usb cable παίρνω από 5 V από τη θύρα του υπολογιστή.

TX ενώνεται με το TX του Arduino

RX ενώνεται με το RX του Arduino

GPIO 2 ενώνεται ο sensor πάνω στο esp8266[8].

Με την βοήθεια του arduino ανεβάζω το κώδικα που θα τρέχει σε κάθε αισθητήρα ,κάθε αισθητήρας συνδέεται με το rasperry pi μέσω του κώδικα στο arduino. Το arduino για να τρέξει χρειάζεται 5V. Όταν προγραμματιστεί ο client τότε βγάζω τα σύρματα που ενώνονται στο Arduino και πλέον βάζω να παίρνει ρεύμα από το power supply.

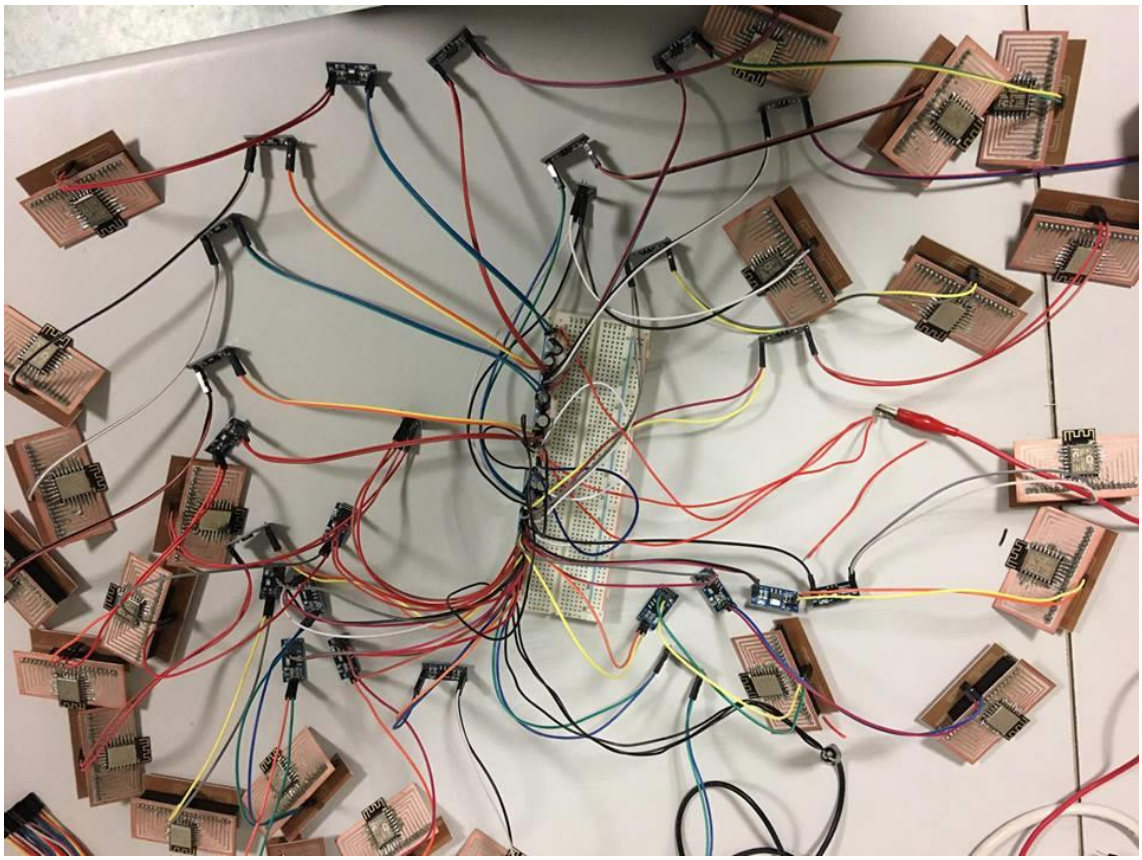
Προβλήματα:

Δεν μπορούσαμε να είχαμε ενωμένο το esp8266 απευθείας με το power supply γιατί ο esp8266 χρειάζεται ακριβώς 3.3v για να μπορεί να προγραμματιστεί. Όταν ενώναμε το power supply με το convert από 5v to 3.3v υπήρχαν κάποιες απώλειες έτσι δεν μπορούσε να προγραμματιστεί.

Λύση:

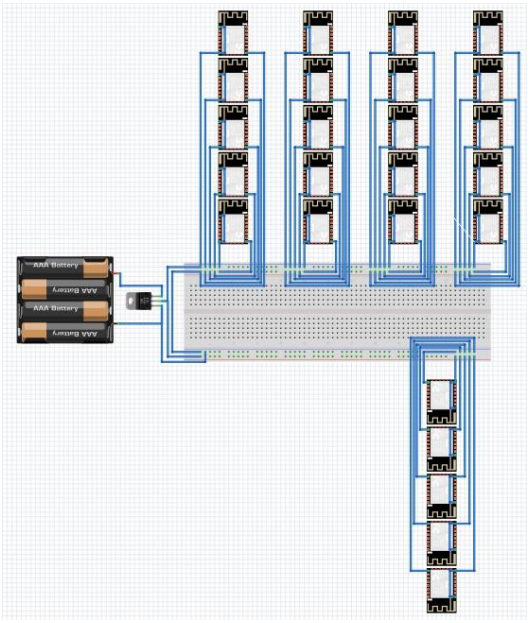
Ο esp8266 αρχικά για να προγραμματιστεί σε κώδικα στο arduino έπρεπε να ήταν ενωμένο με usb cable από το υπολογιστή για να παίρνει ακριβώς 3.3v για να μπορεί να προγραμματιστεί και μετά όταν προγραμματίζονταν το esp8266 το βγάζαμε από το arduino και το ενώναμε στο convert 5v to 3.3v που έπαιρνε ρεύμα από power supply

Διαδικασία για πολλούς πελάτες:



Σχήμα 4.7 Στην εικόνα απεικονίζεται το δίκτυο αισθητήρων

Όταν ανεβάσω το κώδικα τότε βγάζω το client από το Arduino και το ενώνω με το convert 5v σε 3.3v ρεύμα που παίρνω από το power supply. Χρησιμοποιώ power supply 10 A,5V,50 W όπου βάζω παράλληλα sensors[Σχήμα 4.8] για να παίρνουν όλοι τα ίδια volts και να δουλεύει.



Σχήμα 4.8 Αρχιτεκτονική δικτύου αισθητήρων.

Σχήμα 4.9 Στην εικόνα απεικονίζεται power supply 10 A,5V,50 W

Πρόβλημα:

Κάθε τύπος esp8266 που χρησιμοποιείται για να προγραμματιστεί ενώνει κάποια διαφορά σύρματα όπως είπαμε πιο πάνω. Εμείς χρησιμοποιήσαμε το τύπο esp8266-12F όπου για να ενωθεί χρειάζεται 8 σύρματα. Αυτό έκανε πιο δύσκολη την ένωση πολλών sensors στο δίκτυο.

Λύση:

Φτιάξαμε με την μηχανή πλακέτες οι οποίες τα 8 σύρματα που έπρεπε να ενωθούν έγιναν 4 ούτω ώστε όταν ενωθούν πολλά sensor να έχουμε οικονομία από σύρματα, για να αποφευχθούν συμφόρηση συρμάτων και για οικονομία από λεφτά.

Πρόβλημα:

Καθώς πρόσθετα sensors υπήρξε πρόβλημα ότι έκαναν restart τα esp8266 λόγω του ότι όταν προσθέταμε ένα πάνω στο δίκτυο στιγμαία έπαιρνε όλο το ρεύμα από το τροφοδοτικό, το τροφοδοτικό δεν είχε την δυνατότητα να μας το παρέχει συνεπώς δεν παρείχε το συγκεκριμένο ρεύμα που χρειάζονταν τα άλλα έτσι έκαναν restart

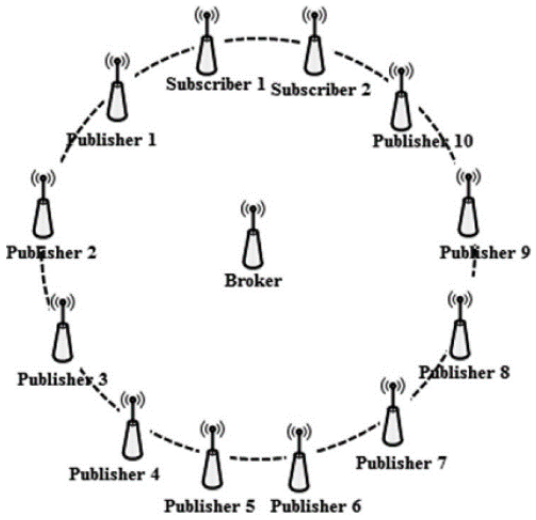
Λύση:

Προσθέσαμε πυκνωτές πάνω στο δίκτυο για να φιλτράρει την συγκεκριμένη στιγμή επειδή το τροφοδοτικό μας δεν είχε την δυνατότητα να μας παρέχει τόσο ψηλό ρευμά.

Κεφάλαιο 5

Ανάλυση Αποτελεσμάτων

5.1 Topology of MQTT:



Σχήμα 5.1 Τοπολογία του mqtt σε δίκτυο αισθητήρων.

Η τοπολογία που χρησιμοποιεί το MQTT είναι star topology. Στην συγκεκριμένη τοπολογία υπάρχουν nodes και ο broker όπου στην περίπτωση μας είναι το raspberry pi.

Τα node είναι συνδεδεμένα με το broker απευθείας. Ο broker είναι υπεύθυνος για όλες τις λειτουργίες, όπως οι μεταφορά δεδομένων .

Πλεονεκτήματα:

Το συγκεκριμένο δίκτυο είναι εύκολο να επεκταθεί ,δηλαδή μπορώ με ευκολία να προσθέσω ή να αφαιρέσω κόμβους.

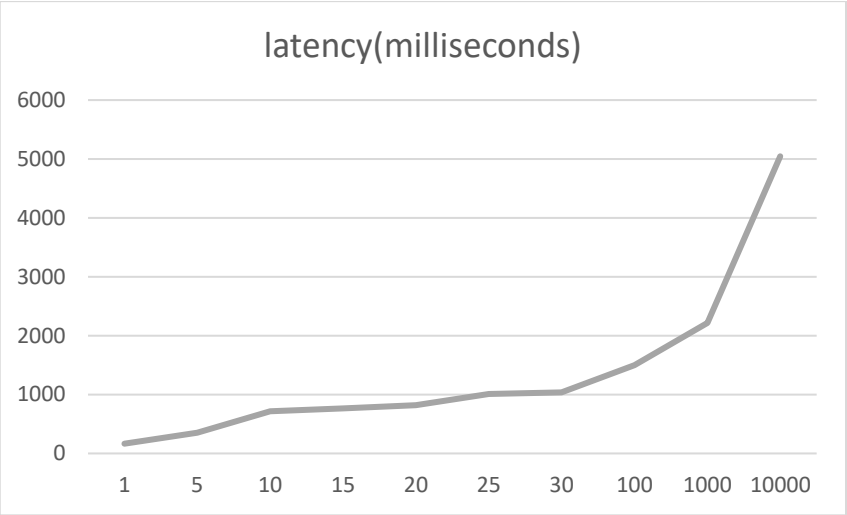
Μειονεκτήματα:

Όλα τα μηνύματα περνούν από το broker έτσι δημιουργεί μεγάλη συμφόρηση αν όλοι οι client θέλουν να στείλουν ταυτόχρονα, καθώς και όταν ο broker αποτύχει θα αποτύχει όλο το δίκτυο. Παρόλαυτα το MQTT έχει ουρά όπου κρατά τα μηνύματα για μικρό χρονικό διάστημα μέχρι να επανέλθει ο broker σε λειτουργία.

5.2 Μέτρα απόδοσης Δικτύων

Δύο σημαντική παράγοντες που κάνουν το σύστημα αποδοτικό είναι να είναι αξιόπιστο δηλαδή να παραλαμβάνει όλα τα μηνύματα ,να είναι γρήγορο και το κόστος να μην είναι πολύ μεγάλο. Δύο μετρήσεις όπου μπορεί να μας ορίσει την επίδοση του δικτύου είναι το PDR, PDR είναι το ποσοστό επιτυχημένων μηνυμάτων που φτάνουν τελικά στο προορισμό τους σε σχέση με αυτά που δημιουργήθηκαν. Επιπλέον και ο χρόνος που καθυστερεί να φτάσει ένα μήνυμα στο προορισμό του μπορεί να μας δείξει πόσο φορτώθηκε το σύστημα. Κάθε σενάριο έχει εκτελεστεί 5 φορές.(Χωρίς persistent mode).

Retain=false. Στις συγκεκριμένες γραφικές μελετούμε την συμπεριφορά τους καθώς αυξάνεται ο κόμβων.

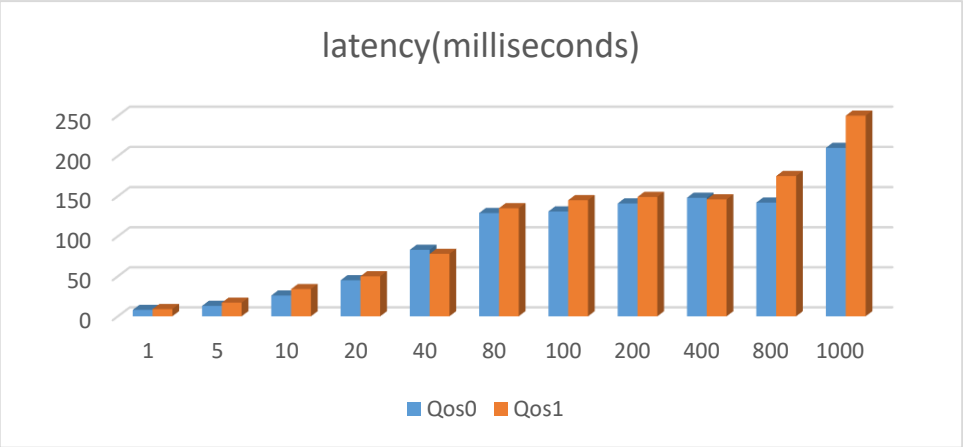


Σχήμα 5.2.

Η αύξηση της καθυστέρησης από κόμβο σε κόμβο είναι γραμμική για μερικές δεκάδες sensor ,για πιο μεγάλο αριθμό η αύξηση γίνεται εκθετική. Είναι λογικό αφού υπάρχει μια κεντρική μονάδα στο σύστημα που είναι ο server. Αν θέλουν όλοι οι κόμβοι που είναι ενωμένοι πάνω του να στείλουν μήνυμα τότε η συμφόρηση του δικτύου γίνεται μεγάλη γιαυτό και η σχέση της καθυστέρησης αυξάνεται εκθετικά. Επίσης γνωρίζουμε ότι δουλεύει πάνω σε TCP όπου το TCP δουλεύει με τριπλή χειραψία πριν την μεταφορά δεδομένων. Γνωρίζουμε ότι τα ασύρματα δίκτυα είναι επιρρεπής σε λάθη σε περίπτωση που υπάρχει λάθος μετάδοση το TCP αντιλαμβάνεται ότι υπάρχει λάθος λόγω συμφόρησης ενώ μπορεί να είναι λόγω μεταφοράς κάποιου λάθος bit τότε το TCP μειώνει το ρυθμό το bits που φτάνουν στο παραλήπτη και τότε αυξάνεται η καθυστέρηση . Συνεπώς το συγκεκριμένο πρωτόκολλο για 10000 δεν είναι κατάλληλο για latency του.

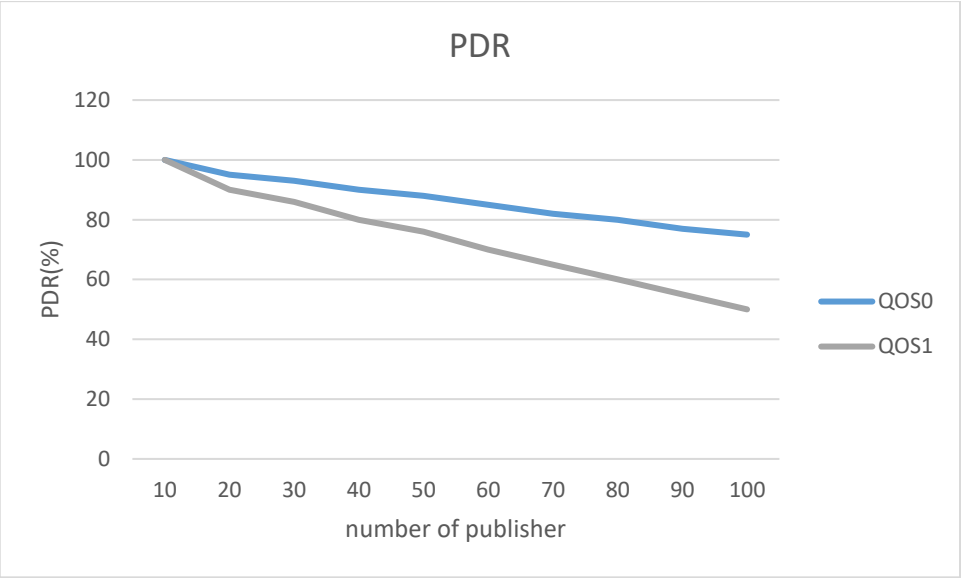
5.3 Χαρακτηριστικά του δικτύου που επηρεάζουν την απόδοση δικτύου

Όπως ειπώθηκε πιο πάνω από κάποιες μετρήσεις μπορούμε να δούμε την απόδοση του δικτύου μας. Όμως πλέον οι διαδικτυακές εφαρμογές έχουν συγκεκριμένες απαιτήσεις εξυπηρέτησης Qos. Το συγκεκριμένο πρωτόκολλο MQTT παρέχει QOS 0 ,1, 2 όπου τα έχουμε εξηγήσει πιο πάνω. Στις πιο κάτω γραφικές θα δούμε κατά πόσο επηρεάζει η αλλαγή του Qos στην απόδοση του δικτύου μας.

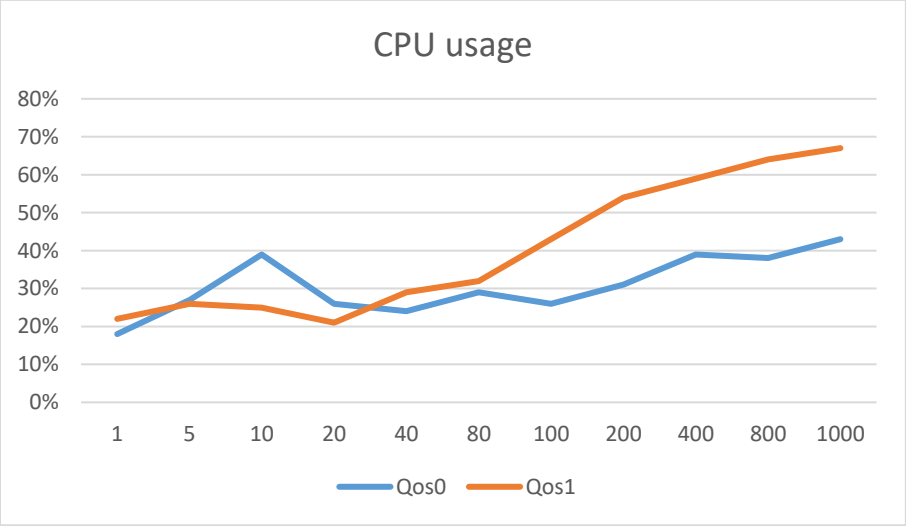


Σχήμα 5.3 Σχέση latency με Qos όταν αυξάνονται οι clients.

Όπως αναμέναμε από την θεωρία όταν το Quality of service είναι μεγαλύτερο στην συγκεκριμένη γραφική Qos 0 σε σύγκριση με Qos 1 ο χρόνος που καθυστερεί να φτάσει ένα μήνυμα είναι μεγαλύτερος λόγω του ότι έγκειται ότι το μήνυμα θα φτάσει τουλάχιστον μια φορά. Όταν σταλθεί το μήνυμα και δεν πάρει απαντήσει ο broker τότε θα ξαναστείλει το μήνυμα. Συνεπώς καταλαβαίνουμε ότι η γραμμή φορτώνεται περισσότερο στην δεύτερη περίπτωση γι' αυτό και τα μηνύματα αργούν να φτάσουν στο subscriber αλλά όπως βλέπουμε η διαφορά είναι πολύ μικρή ,μερικά milliseconds. Η γραμμή στην δεύτερη περίπτωση φορτώνεται περισσότερο αφού θα υπάρχουν retransmissions στην περίπτωση που δεν σταλθεί το μήνυμα να το ξαναστείλει αφού πρέπει να υπάρχει τουλάχιστον μια φορά το μήνυμα και στην γραμμή θα σταλθούν τα μηνύματα όπου είναι η απάντηση 1 στο publish. Το Qos 2 δεν το παρέχει ο server mosca. Όμως ξέρουμε ότι για Qos 2 λόγω του 4 handshake έχουμε μεγαλύτερη καθυστέρηση από άκρο σε άκρο και έχει PRD 100%.Παρόλο που είναι TCP μπορεί να υπάρξουν packet loss λόγω του αν χαθεί η σύνδεση του TCP για κάποιο λόγο.[Σχήμα 5.4]



Σχήμα 5.4 PDR.



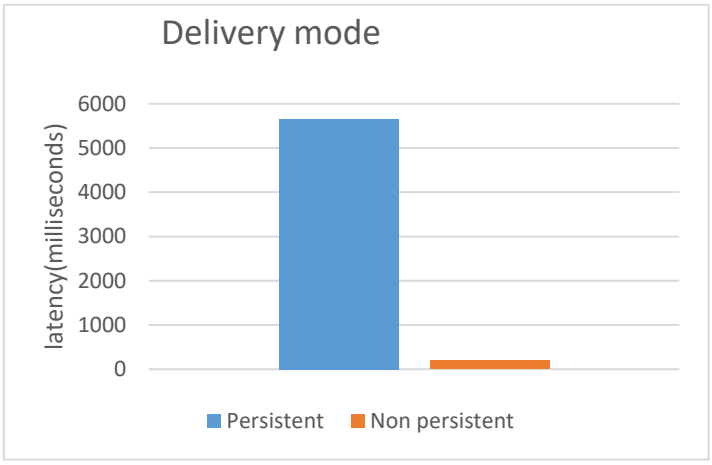
Σχήμα 5.5.

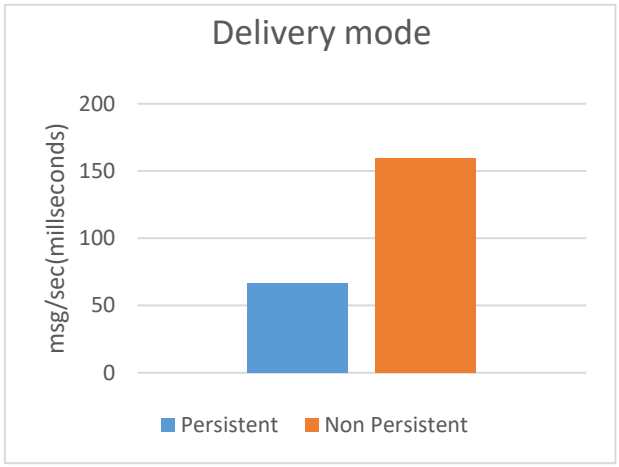
Όπως αναμέναμε γίνεται περισσότερη χρήση cpu όταν έχω μεγαλύτερο Qos για τους λόγους που εξηγήσαμε πιο πάνω.

Durable-Non durable subscriptions

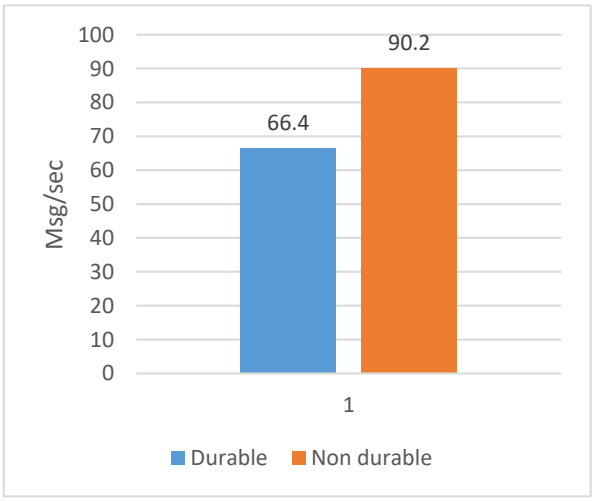
Στο πιο κάτω παράδειγμα θα χρησιμοποιήσω Number of connections 1001 όπου 1000 είναι publisher και ένας subscriber. Για το Persistent mode πρέπει να χρησιμοποιήσουμε

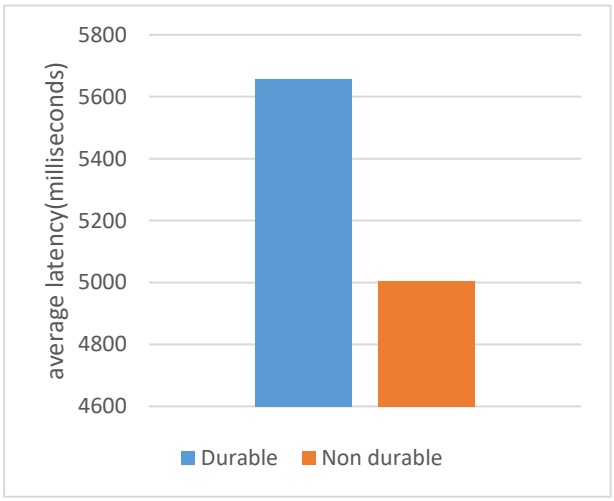
στο αλγόριθμο μας Retain flag false και Qos του publisher και του subscriber 1 και τέλος για durable subscriber clean subscriber να είναι false για να διατηρούνται οι ουρές.





Σχήμα 5.4 και 5.5:Γραφικές εικονικού περιβάλλοντος.



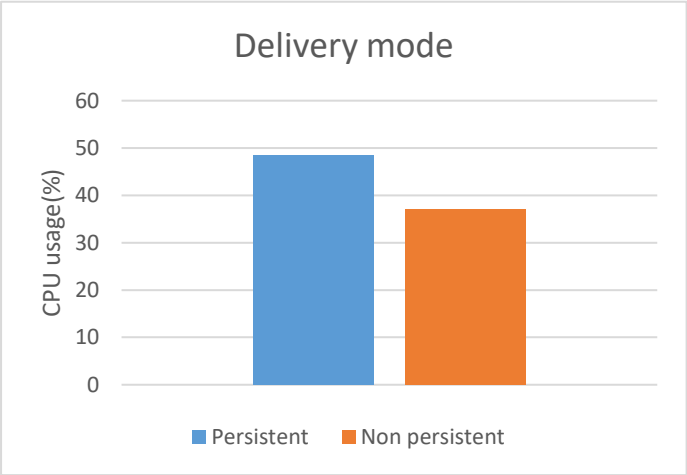


Σχήμα 5.6 και Σχήμα 5.7: Γραφικές εικονικού περιβάλλοντος

Από την γραφικές [Σχήμα 5.4] και [Σχήμα 5.5] βλέπουμε ότι η παράδοση μηνύματος από το broker είναι πιο αργή στην Persistent mode επειδή θέλουμε οπωσδήποτε το μήνυμα να φτάσει στο subscriber έτσι ο broker κάνει επιπλέον επεξεργασία στα μηνύματα. Ξέρουμε ότι το throughput και το latency εξαρτάται από την διαδικασία που κάνει ένα

μήνυμα να φτάσει στο προορισμό του. Στην συγκεκριμένη περίπτωση στο χρόνο προθέτονται και οι χρόνοι όπου ο broker πρέπει να επιβεβαιώσει ότι παρέλαβε το μήνυμα. Επιπλέον όπως γνωρίζουμε το latency είναι αντίθετο του throughput δηλαδή όσο πιο αργό είναι το latency τόσο είναι μικρό. Με αποτέλεσμα στις γραφικές [Σχήμα 5.6] και [Σχήμα 5.7] βλέπουμε ότι όταν υπάρχει επίμονο subscription ο broker αποθηκεύει ουρές από μηνύματα για το συγκεκριμένο subscriber ούτω ώστε όταν ξανασυνδεθεί μετά από κάποια αποτυχία σύνδεσης να παραλάβει όλα τα μηνύματα, γιαυτό και η επεξεργασία του broker γίνεται πιο αργή.

Συμπέρασμα βλέπουμε ότι η διαφορά με persistent mode non peristen mode του broker είναι αρκετά μεγάλη για τους λόγους αυτούς, ενώ durable και non durable η διαφορά είναι πιο μικρή γιατί είδη τα μηνύματα τοποθετούνται στην ουρά και μετά στέλνονται ενώ non persistent όχι και είναι μεγάλος αριθμός ενώ το durable .

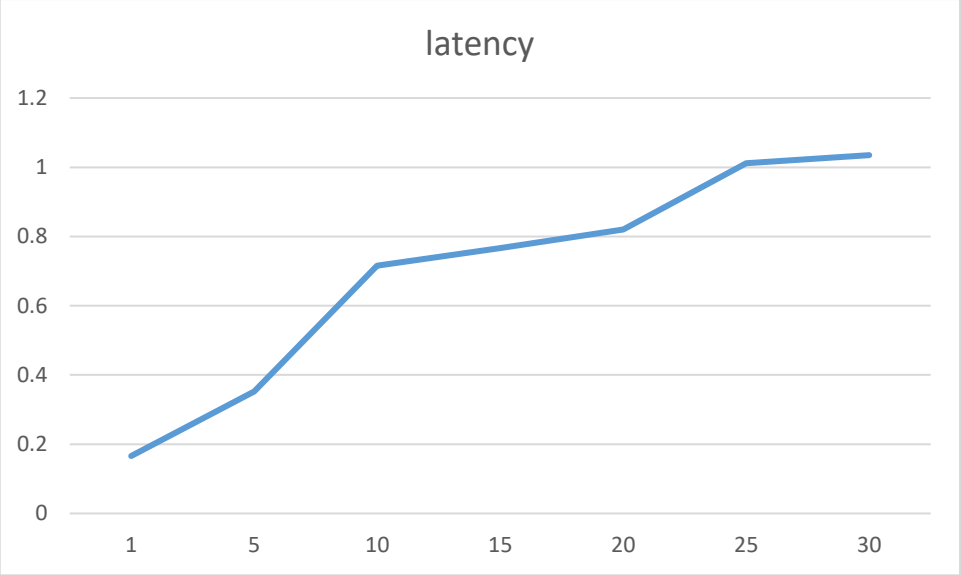


Σχήμα 5.8 Γραφική εικονικού περιβάλλοντος

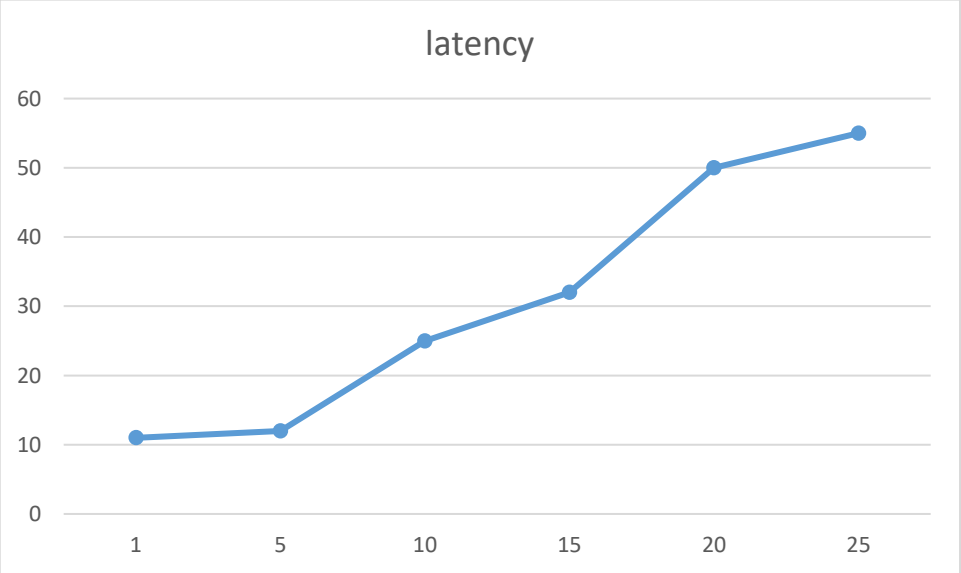
Ως αποτέλεσμα όλων αυτών που είπαμε πιο πάνω καταλαβαίνουμε ότι ο φόρτος της γραμμής είναι μεγαλύτερος στην περίπτωση που έχω persistent mode επειδή ήδη η γραμμή είναι φορτισμένη από τα μηνύματα τώρα έρχεται να προστεθεί και ο φόρτος από τα μηνύματα που στέλνει ο broker ότι παρέλαβε τα μηνύματα ούτω ώστε να διασφαλίσει ότι το μήνυμα οπωσδήποτε θα σταλθεί.

5.4 Σύγκριση εικονικού περιβάλλοντος με πραγματικό περιβάλλον

Ξέρουμε ότι στο εικονικό περιβάλλον δεν μετριούνται πολλοί παράγοντες όπου υπάρχουν στο πραγματικό περιβάλλον.

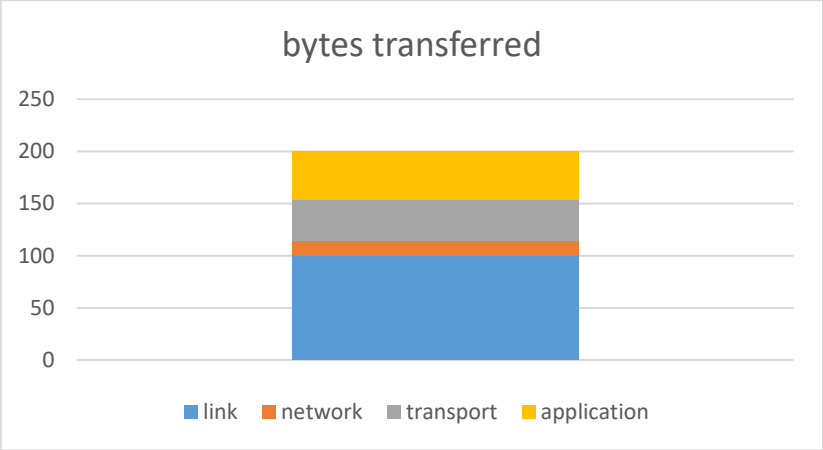


Σχήμα 5.9. Στο σχήμα φαίνεται η σχέση latency καθώς αυξάνεται ο αριθμός clients.



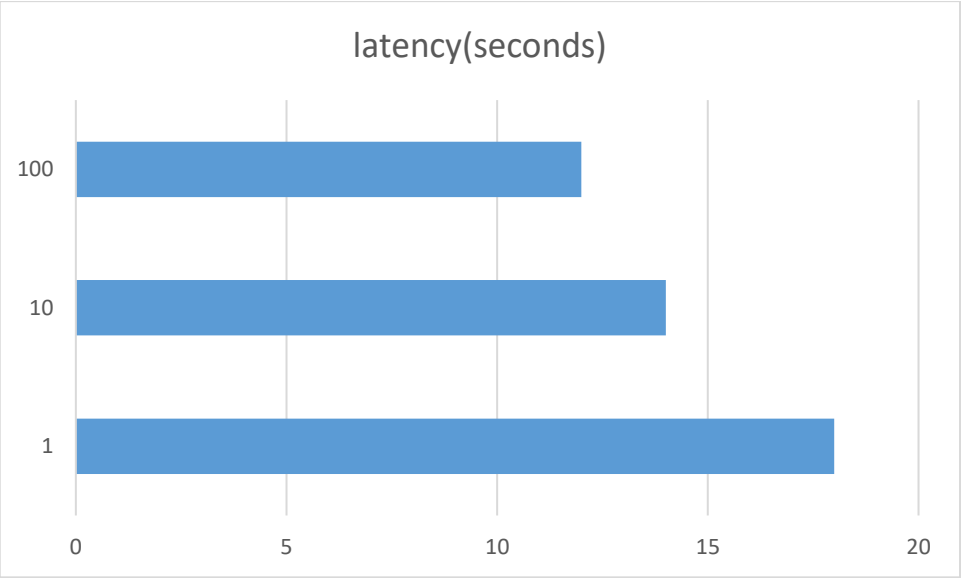
Σχήμα 5.10 Στο σχήμα φαίνεται η σχέση latency καθώς αυξάνεται ο αριθμός sensor.(πραγματικό περιβάλλον)

Όσο αυξάνονται οι κόμβοι πάνω στο δίκτυο βλέπουμε και στις δύο περιπτώσεις ότι αυξάνεται. Συνεπώς έρχεται να επαληθεύσει ότι οι μετρήσεις που πήραμε μέχρι στιγμής αντικατοπτρίζουν και σε πραγματικές καταστάσεις. Η καθυστέρηση από κόμβο σε κόμβο μεγαλώνει γραμμικά και στις δύο περιπτώσεις το οποίο είναι εξαιρετικά σημαντικό γιατί στα δίκτυα αισθητήρων μας ενδιαφέρει πάρα πολύ ο χρόνος που παραλαμβάνουμε τις μετρήσεις γιατί αν οι πληροφορίες φτάσουν αργά μπορεί να είναι αργά και να αντιμετωπιστεί μια κατάσταση. Επίσης το γεγονός ότι στο πραγματικό περιβάλλον το χρόνο αποστολής του μηνύματος των παίρνουμε από ένα πακέτο που στέλνεται από time protocol αυξάνει το χρόνο παραλαβής μηνυμάτων. Παράδειγμα στο σύστημα ασφάλειας αν έρθει στο σπίτι κάποιος κλέφτης και ανοίξει την πόρτα ,θέλουμε σε ελάχιστα δευτερόλεπτα να πάει το σήμα στην αστυνομία για να φτάσει σπίτι μας. Στον πραγματικό περιβάλλον



Σχήμα 5.11 Σχήμα από πραγματικό περιβάλλον

Ξέρουμε ότι το overhead του δικτύου είναι πολύ μικρό γιατί το πρωτόκολλο είναι δυαδικό πρωτόκολλο, και το μέγεθος των μηνυμάτων είναι μικρό .Το μικρότερο μήνυμα είναι 2 bytes και το μεγαλύτερο μήνυμα είναι 256 bytes. Γιαυτό το και το overhead πάνω στο ωφέλιμο φορτίο είναι αρκετά μικρό. Είναι πολύ σημαντικό στο ότι φορτώνει την γραμμή ελάχιστα έτσι το latency είναι μικρό.



Σχήμα 5.12. Σχήμα από πραγματικό περιβάλλον.

Όπως βλέπουμε στο σχήμα αν στέλνουμε κάθε 1 seconds μήνυμα τότε η συμφόρηση θα μεγαλώσει και τότε ο ρυθμός εισερχόμενων μηνυμάτων θα μειωθεί και η καθυστέρηση από κόμβο σε κόμβο θα αυξηθεί, αφού είναι λογικό θα στέλνει συνεχώς μηνύματα αρά ο φόρτος της γραμμής θα αυξηθεί.

Κεφάλαιο 6

Συμπεράσματα

6.1 Συμπεράσματα

Συμπερασματικά με τα πιο πάνω βλέπουμε ότι καθώς αυξάνονται οι χρήστες τότε η γραμμή φορτώνεται και η καθυστέρηση αυξάνεται και ο ρυθμός εισερχόμενων bits μειώνεται.

Επίσης βλέπουμε ότι οι πραγματικές μετρήσεις διαφέρουν από τις μετρήσεις σε εικονικό περιβάλλον. Στο εικονικό περιβάλλον οι καταστάσεις είναι πιο εξειδικευμένες ενώ σε πραγματικό υπάρχουν πολλές άλλοι παράμετροι που επηρεάζουν. Στο δικό μου δίκτυο επηρέαζε το γεγονός ότι οι sensor μου ήταν αρκετά κοντά έτσι προκαλούσε παρεμβολές στα ηλεκτρομαγνητικά κύματα το οποίο αλλοίωνε το αποτέλεσμα, επίσης και η ποιότητα των συρμάτων και των συσκευών επηρέαζε την απόδοση του δικτύου επειδή χρησιμοποιούμε συσκευές η οποίες είναι φτηνές έχουν πιο λίγες παροχές από αυτές που είναι πιο ακριβές καλύτερη ποιότητα.

Όλο και περισσότερες εταιρίες θα χρησιμοποιούν το IOT, και ο αναλυτής συστήματος θα έρθει σε θέση να αποφασίσει πιο πρωτόκολλο επικοινωνίας θα χρησιμοποιήσει. Προσωπικά θα πρότεινα το MQTT επειδή καταναλώνει ελάχιστη ενέργεια και περιορισμένο ευρό ζώνης, έχει χαμηλό χρόνο αύξησης καθυστέρησης από κόμβο σε κόμβο όταν οι κόμβοι που είναι στο δίκτυο είναι μερικές δεκάδες. Όμως δεν θα το πρότεινα για χιλιάδες χρήστες γιατί αυξάνει την συμφόρηση στα ύψη αφού είναι κεντροποιημένο δίκτυο. Όπως βλέπουμε το δίκτυο παρέχει διάφορες επιλογές και αυτές οι επιλογές έχουν τις ανάλογες επιπτώσεις στην επίδοση, αν η εταιρία θέλει τα μηνύματα που στέλνονται να λαμβάνονται οπωσδήποτε τότε χρησιμοποιώ το qos όμως θα μου αυξήσει την καθυστέρηση παραλαβής μηνυμάτων αρά

πρέπει να αποφασίσω ανάμεσα qos 1,qos 2. Καλό είναι να βρεθεί μια μέση λύση δηλαδή να είναι και αποδοτικό αλλά και γρήγορο και να στέλνεται ούτε πολύ γρήγορα το μήνυμα ούτε πολύ αργά. Όμως παίζει σημαντικό ρόλο και το σύστημα όπου θέλουμε να χρησιμοποιούμε δηλαδή αν είναι για θερμοκρασία να επιλέξω κάτι ενδιάμεσο αν είναι για σύστημα ασφάλειας τότε το μήνυμα να στέλνεται πάρα πολύ γρήγορα.

Επίσης ένα μεγάλο αρνητικό είναι ότι το mqtt δεν παρέχει κρυπτογράφηση ασφάλεια αλλά μπορείς να χρησιμοποιήσεις το TTLS/SSL για να προσθέσεις ασφάλεια όμως έρχεται να αυξήσει το overhead όπου πριν ήταν πάρα πολύ μικρό.

6.2 Επίλογος

Όπως αναφέρει και το όνομα του δηλαδή το MQTT(Message Queue Telemetry Transport) είναι κατάλληλο για telemetry data. Δηλαδή για δεδομένα που προέρχονται από αισθητήρες. Η τηλεμετρία είναι αυτοματοποιημένη διαδικασία επικοινωνίας με τις μετρήσεις και άλλα δεδομένα που συλλέγονται από απομακρυσμένα ή απρόσιτα σημεία η οπουδήποτε σημεία και μεταδίδονται στο εξοπλισμό για παρακολούθηση.

6.3 Μελλοντική Δουλειά

Μια εισήγηση δική μου βλέποντας ότι το πρωτόκολλο δεν προσφέρει αρκετή ασφάλεια και επειδή άποψη μου είναι ότι είναι ένας σημαντικός παράγοντας αφού στο iot δόθηκε το όνομα κοινωνία της πληροφορίας θα ήταν να μελετηθεί κατά πόσο μπορεί το πρωτόκολλο να προσφέρει θέμα ασφάλειας χωρίς να επιβαρύνεται όμως το πρωτόκολλο και να παραμένει ελαφρύ. Θεωρώ ότι είναι από τους πιο σημαντικούς παράγοντες σε ένα δίκτυο αφού όπως είπαμε το IOT θα είναι παντού σε λίγα χρόνια ,συνεπώς όλοι μας θέλουμε να διασφαλίσουμε τα προσωπικά μας δεδομένα από την πλευρά των έξυπνών σπιτιών αλλά και από θέμα αποτροπής κάποια κακόβουλης πράξης που θα χρησιμοποιήσει τα δεδομένα για κακό σκοπό η αποτροπή κάποιων παρεμβολών στην αυτοματοποίηση πράξεων με αποτέλεσμα λανθασμένων αποτελεσμάτων. Παράδειγμα λανθασμένη ιατρική γνωμάτευση με αποτέλεσμα χρήση κάποια μη αναγκαίας εγχείρηση με αποτέλεσμα το κακό του ασθενή. Επίσης θα μπορεί να γίνει πρόσθεση και άλλων sensors. Μπορεί να γίνει με την πρόσθεση ενός power supply πιο πολλών volts, αλλά

δεν το συνιστώ γιατί θα ήταν επικίνδυνο ,αυτό που θα συνιστούσα θα ήταν η πρόσθεση και άλλων power supply και οι πρόσθεση των sensor εκεί.

Βιβλιογραφία

1.Χαρακτηριστικά του Arduino από την επίσημη ιστοσελίδα.

Available: https://www.arduino.cc/en/main/arduinoBoardUno.

2. Brown, Eric (13 September 2016). "Who Needs the Internet of Things?" *Available:* https://www.linux.com/news/who-needs-internet-things

3. *"The "Only" Coke Machine on the Internet"* Available: https://www.cs.cmu.edu/~coke/history_long.txt

4. "Constrained Application Protocol (CoAP) RFC 7252" The Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7252.

5. MQTT Version 3.1.1 Plus Errata 01 Andrew Banks IBM Rahul Gupta IBM Available:http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

6. Andy, Stanford-Clark. "MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2" Available: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

7. Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, Hongtaek Ju Dept. of Computer Engineering Keimyung University Daegu, Republic of Korea "Correlation Analysis of MQTT Loss and Delay According to QoS Level" Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6496715

8. ILYAS BAIG CHIKTAY MUZAMIL SALAHUDDIN DALVI, "HOME AUTOMATION USING ARDUINO WIFI MODULE ESP8266" Available: http://www.aiktcdspace.org:8080/jspui/bitstream/123456789/1558/1/PE0112.pdf

9. Vasileios Karagiannis1, Periklis Chatzimisios1, Francisco Vazquez-Gallego2, Jesus Alonso-Zarate 1/ CSSN Research Lab, Department of Informatics, Alexander TEI of Thessaloniki, Greece "A Survey on Application Layer Protocols for the Internet of Things" Available: https://www.researchgate.net/profile/Periklis_Chatzimisios/publication/303192188_A_survey_on_application_layer_protocols_for_the_Internet_of_Things/links/577b656608ae213761c9d91d.pdf

10. Σχετική Βιβλιογραφία:

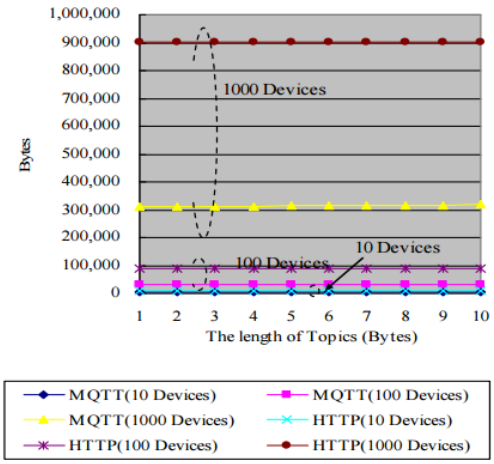
A) Design and Implementation of Push Notification System Based on the MQTT Protocol.

Table 2 Effect of the power and Network data consumption

Time	Power consumption	Network data consumption	Loss rate
12 hours	10%	20KB	0.3%
24 hours	20%	40KB	1%

Σκοπός της μελέτης τους ήταν η κατανάλωση ενέργειας με χρήση MQTT σε αποστολή μηνυμάτων σε smartphone, μέτρησαν την κατανάλωση και την αξιοπιστία του συστήματος σε 12 ώρες και μετά σε 24 ώρες. Έβγαλαν συμπέρασμα ότι καταναλώνεται πολύ λίγη ενέργεια και έχει πολύ ψηλή ακρίβεια.

B) Transfer Protocols of Tiny Data Blocks in IoT and their Performance Evaluation
http://ieeexplore.ieee.org/abstract/document/7845442/



Στο συγκεκριμένο άρθρο αναφέρονται οι διαφορές μεταξύ του πρωτοκόλλου MQTT με HTTP. Ένα από τα συμπεράσματα που έβγαλαν είναι ότι το bandwith του mqtt με payload=0 είναι μεγαλύτερο καθώς αυξάνεται ο αριθμός των devices αυξάνεται και στα δύο πρωτόκολλα. Το mqtt έχει μικρότερο bandwith σε σχέση με το http συνεπώς το mqtt είναι πιο ελαφρύ πρωτόκολλο. Μάλιστα φαίνεται ότι η διάφορα τους σε bytes είναι αρκετά μεγάλη.

Παράρτημα:

Κώδικας:

Για broker.js. ο κώδικας έχει παρθεί από το github mosca(μαυρό χρώμα). Όμως έχει τροποποιηθεί και έχουν προσθέτει κάποια πράγματα για να πάρουμε τις μετρήσεις για να γίνει η ανάλυση του δικτύου, σε πραγματικά δεδομένα (μπλε χρώμα).

```
var mosca = require('mosca');
var cpuAverage = require('./cpu_usage.js');
var monitor = require('monitor');
const tls = require('tls');


//initialize
const metrics={
        //number of msg received
        num_Received_msg:0,
        //number of msg published
        num_published_msg:0,
        latency:0,
        latencyAvg:0,
        efficiency:0,
        time_received_msg:[],
        first:0,
        num_client:0,
        last:0,
        startMeasure:0,
        endMeasure:0,
        p_loss:0,
        pososto:0,
        max_l:0,
        min_l:0,
```

```
}
var i=0;
var ascoltatore = {

  type: 'mongo',                                =>προσθήκη χρήσης της βάσης δεδομένων
  url: 'mongodb://127.0.0.1:27017/mqtt',        => για να φυλάγονται δεδομένα που δεν
  pubsubCollection: 'ascoltatori',              =>σταλθεί ακόμα.
  mongo: {}

};

var settings = {
   port: 1883,                                   =>σταθερό port για το mqtt
          persistence: {
       factory: mosca.persistence.Mongo,
       url: 'mongodb://127.0.0.1:27017/mqtt'
    },

backend: ascoltatore
};


var server = new mosca.Server(settings);



server.on('ready',setup);

// fired when the mqtt server is ready
function setup() {

        //Grab first CPU Measure
```

```
//apothikeuetai I xrisi cpu prin na xenisi  I diadikasia apostolic minimatwn
metrics.startMeasure = cpuAverage();
  initialize();
  console.log('Mosca server is up and running');
  if(i==0){
//apothikeuetai I xroniki stigma opou ftani to proto minima
    metrics.first=new Date().getTime();
}
};

//fired when client is connected
server.on('clientConnected', function(client) {
    console.log('client connected', client.id);
            metrics.num_client=metrics.num_client+1;
});

server.on('publish',function(){
//console.log('Publish message', start);

});
// fired when a message received at server
```

//στην συγκεκριμένη συνάρτηση ,όπου ενεργοποιείται όταν έρθει ένα καινούργιο μήνυμα στο server,γίνονται κάποιες μετρήσεις,latency και αριθμός μηνυμάτων που έχουν παραληφθεί. Για να χρησιμοποιηθούν πιο κάτω και να βρεθούν και άλλες μετρήσεις

```
server.on('published', function(packet, client) {
            const msg = packet.topic.toString();

            var top=msg.split('|').splice(0,1);
            var top_1=packet.topic;
            if((top_1=='/home/room')||(top_1=='/outTopic')){
                        //console.log(packet.qos);
```

```
                metrics.num_published_msg++;

        }
        //count message received in consumer
        if(top=='server'){

                metrics.last=new Date().getTime();

                var l=Math.abs(msg.split('|').splice(-1));
                if(i==0){

                metrics.min_l=l;
                }
                if(l>metrics.max_l){
                        metrics.max_l=l;
                }

                if((l<metrics.min_l)&&(i>=1)){
                        metrics.min_l=l;
                }
            metrics.time_received_msg[i]=l;
      i++;
            metrics.latency=Math.floor(l)+metrics.latency;
                metrics.num_Received_msg++;

        }

});

// fired when a client subscribes to a topic
server.on('subscribed', function(topic, client) {
  console.log(client.id,'subscribed with topic: ', topic);
});
```

```
// fired when a client subscribes to a topic
server.on('unsubscribed', function(topic, client) {
  console.log('unsubscribed : ', topic);
});

server.published = function(packet, client, callback) {
  callback(null);
};

// fired when a client is disconnected
server.on('clientDisconnected', function(client) {
  console.log('clientDisconnected : ', client.id);
});


close_server();                          =>φωνάζω την συνάρτηση για να υπολογίσει
                                         =>μετρήσεις.
var msg='hello';
var x='quit\n';

function close_server (){
process.stdin.resume();
  process.stdin.setEncoding('utf8');
  var util = require('util');

  process.stdin.on('data', function (text) {          =>όταν γράφω στο node.js
   // console.log('received data:', util.inspect(text));    =>την λέξη quit τότε ο server
    if (text === 'quit\r\n') {                        =>κλείνει για να μην πάρει άλλα
      done();                                          =>μηνύματα και υπολογίζονται
    }                                                   =>οι μετρήσεις
  });
};
```

```
function efficiency_x(){                                    =>υπολογίζεται το PDR
   metrics.efficiency=(metrics.num_Received_msg-
metrics.p_loss)*100/metrics.num_published_msg;
}
function mps(){                                             =>υπολογίζεται το throughput

        var timespan=0;
        var th=0;
        var n=metrics.num_Received_msg;

        timespan=(metrics.last-metrics.first)/1000;

        //th=1/th;
        if((timespan==0)&&(metrics.num_Received_msg==1)){
                th=1;
        }
        else{
                th=Math.floor(n/timespan);
                console.log("th",th,"n",n,"timespan",timespan)
        }
        return th;

}
function throughput_x(){
        var th=0;
        var n=metrics.num_Received_msg;

        th=(metrics.last-metrics.first)/1000;
        //sending overhead
        var son=th/n;
        var stn=0;
```

```
        if((son==0)&&(metrics.num_Received_msg==1)){
                stn=1;
        }
        else
                stn=Math.floor(1/son);
        //console.log("stn",stn);
        return stn;
}
function packet_lost(){

        var p_l=Math.floor((metrics.num_client-1)*metrics.pososto*0.5/100);
        return p_l;
}

function initialize(){                  =>μόλις υπολογιστούν όλες οι μετρήσεις τότε
   metrics.num_Received_msg=0; =>μηδενίζονται οι πράξεις για την χρήση την
        //number of msg published     =>επόμενη φορά
        metrics.num_published_msg=0;
        i=0;
        metrics.latency=0;
        metrics.latencyAvg=0;
        metrics.efficiency=0;
        metrics.first=0;
        metrics.last=0;
        metrics.pososto=0;

}
 function done() {      =>στο τέλος  τυπώνονται
        var ef=0;
        var t=throughput_x();
        var mespersecond=mps();
        metrics.latencyAvg=Math.floor((1.0*metrics.latency)/(1.0*metrics.num_Receiv
ed_msg));
```

```
        console.log("metrics.latency kai metrics.num_Received_msg");
       console.log(metrics.latency);
       console.log(metrics.num_Received_msg);
        //5 decimal
        var n = metrics.latencyAvg.toFixed(5);



        metrics.endMeasure  = cpuAverage();
        var idleDifference = metrics.endMeasure.idle - metrics.startMeasure.idle;
   var totalDifference = metrics.endMeasure.total - metrics.startMeasure.total;

   //Calculate the average percentage CPU usage
   var percentageCPU = 100 - ~~(100 * idleDifference / totalDifference);

   //Output result to console
   console.log(percentageCPU + "% CPU Usage.");
        metrics.pososto=(metrics.num_client-1)*0.5/100;//10%an ine qos=0
        console.log("metrics.pososto",metrics.pososto);
        //metrics.pososto=0;
        metrics.p_loss=packet_lost();
        efficiency_x();

        console.log('number of clients which connected',metrics.num_client);
        console.log('Total Message Sent:', metrics.num_published_msg);
        console.log('Total Message Received:', metrics.num_Received_msg-
Math.floor((metrics.num_client-1)*metrics.pososto*0.5/100));
        console.log("averagelatency",n,"seconds");
        console.log("max latency",metrics.max_l,"seconds");
        console.log("min latency",metrics.min_l,"seconds");
        console.log("message per second",mespersecond);
        console.log("received throughput is",t,"message per second");
        console.log("efficiency",metrics.efficiency,"%");
        console.log("packet_lost",metrics.p_loss);
```

```
        console.log('close server');
        initialize();
   process.exit();
 };
```

Subscriber:

Το μήνυμα καθώς έχει δημιουργηθεί από το publisher αποθηκεύεται και ο χρόνος που το μήνυμα αποστέλνεται και μόλις παραληφθεί δημιουργείται ο χρόνος που το μήνυμα έχει παραληφθεί και έτσι υπολογίζεται το latency.

```
const mqtt = require('mqtt')
const client = mqtt.connect('mqtt://10.16.21.179',{clientId:'client2',clean:false});

const metrics={

        count:0,
};

client.on('connect', function() {//when connected
client.subscribe('/outTopic',{qos:1},function(){
});
  client.subscribe('/home/room',{qos:1},function(){
  });
});
//client listen message
client.on('message', (topic, message) => {
if((topic=='/home/room')||(topic=='/outTopic')){

        //time which received msg
  var received_time = 0;


  const msg = message.toString();
```

```
substring = "hellox";

var sent_time=0;
if(msg.includes(substring)==true){
        console.log("clients of node.js");
        received_time=Math.floor(new Date().getTime());
        var sent_time=Math.floor(msg.split('|').splice(-1));
        //var table=msg.split('|');
       //console.log(table);
        //sent_time=table[2];
        console.log("sent_time");
   console.log(sent_time);
        console.log("receive time");
        console.log(received_time);

}
else {
        console.log("sensor");
        received_time=Math.floor(((new Date().getTime())/1000)+1);
        var table=msg.split(',');
   var n=table[1].split(':');
   var teliko=n[1].split('}');
        console.log("receive");
        console.log(received_time);
        console.log("sent");
   console.log("d",teliko[0]);
        sent_time=teliko[0];
}

var latency=received_time-sent_time;
console.log("latency");
console.log(latency);
```

```
  var packet='hi';
  //time from 1/1/1970 to seconds
  const topic1=['server',(latency)].join('|');
  client.publish(topic1,"received");
  metrics.count=metrics.count+1;
  //console.log([topic,msg.split('|').splice(0,1)].join(":"),'seconds',metrics.count);

}
});

 client.on('clientDisconnected', function() {
        console.log('Connection closed');
        console.log("message pu stalthikan"+count);
});
```

Publisher

=>Προσθήκη βιβλιοθηκών.

```
#include <ArduinoJson.h>
#include <DHT.h>
#include <WiFiUdp.h>
//#include <time.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.
=>το esp8266 ενώνεται με το  raspberry
#define wifi_ssid "smarthome"
#define wifi_password "raspberry"
#define MQTTQOS0
#define MQTTQOS1
#define MQTTQOS2
#define mqtt_server "10.16.21.179"
```

```
#define DHTTYPE DHT11
#define DHTPIN 14
unsigned int localPort = 2390;      // local port to listen for UDP packets

/* Don't hardwire the IP address or we won't get the benefits of the pool.
 *  Lookup the IP address for the host name instead */
IPAddress timeServer(129, 6, 15, 28); // time.nist.gov NTP server
IPAddress timeServerIP; // time.nist.gov NTP server address
const char* ntpServerName ="time.nist.gov";

const int NTP_PACKET_SIZE = 48; // NTP time stamp is in the first 48 bytes of the message

byte packetBuffer[ NTP_PACKET_SIZE]; //buffer to hold incoming and outgoing packets

// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp;
WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN,DHTTYPE,11);

float temp=0;
float hum=0;
long lastMsg = 0;
char msg[50];
int value = 0;
unsigned long secsSince1900=0;


void setup_wifi() {

  delay(10);
```

```
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(wifi_ssid);

  WiFi.begin(wifi_ssid, wifi_password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }


  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  udp.begin(localPort);
  Serial.println("Starting UDP");
  udp.begin(localPort);
  Serial.print("Local port: ");
  Serial.println(udp.localPort());

}


void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
```

```
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += 25;
    //String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}



void setup() {
  dht.begin();
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}
//διαβάζει μέτρηση από το sensor, κάθε τύπου sensor  έχει  δικό του τρόπο για να πάρει δεδομένα ,άρα άλλως κώδικας.
bool getTempHum() {
  hum = dht.readHumidity();          // Read humidity (percent)
  temp = dht.readTemperature();     // Read temperature as celsius
  // Check if any reads failed and exit early (to try again).
  if (isnan(hum) || isnan(temp)) {
    hum=0;
    temp=0;
```

```
    Serial.println("Failed to read from DHT sensor!");
    return false;
  }
  return true;
}
```

//παίρνω το χρόνο σε seconds από το 1970 για να μπορώ να επεξεργαστώ με το χρόνο που παίρνω στο subscriber που είναι γραμμένο σε node .js

```
// send an NTP request to the time server at the given address
unsigned long sendNTPpacket(IPAddress& address)
{
  Serial.println("sending NTP packet...");
  // set all bytes in the buffer to 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  udp.beginPacket(address, 123); //NTP requests are to port 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
}

void loop() {
```

```
  char buffer[256];
  StaticJsonBuffer<256> jsonBuffer;


  if (!client.connected()) {
    reconnect();
  }
  client.loop();
WiFi.hostByName(ntpServerName, timeServerIP);

  sendNTPpacket(timeServerIP); // send an NTP packet to a time server
  // wait to see if a reply is available
  delay(1000);

  int cb = udp.parsePacket();
  if (!cb) {
    Serial.println("no packet yet");
  }
  else {
    Serial.print("packet received, length=");
    Serial.println(cb);
    // We've received a packet, read the data from it
    udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet into the buffer

    //the timestamp starts at byte 40 of the received packet and is four bytes,
    // or two words, long. First, esxtract the two words:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // combine the four bytes (two words) into a long integer
    // this is NTP time (seconds since Jan 1 1900):
    unsigned long secsSince1900 = (highWord << 16 | lowWord);
```

```
  Serial.print("Seconds since Jan 1 1900 = " );
  Serial.println(secsSince1900);

  // now convert NTP time into everyday time:
  Serial.print("Unix time = ");
  // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
  const unsigned long seventyYears = 2208988800UL;
  // subtract seventy years:
  unsigned long epoch = (secsSince1900 - seventyYears)-13;
  // print Unix time:
  Serial.println(epoch);


  // print the hour, minute and second:
  Serial.print("The UTC time is ");       // UTC is the time at Greenwich Meridian (GMT)
  Serial.print((epoch  % 86400L) / 3600); // print the hour (86400 equals secs per day)
  Serial.print(':');
  if ( ((epoch % 3600) / 60) < 10 ) {
    // In the first 10 minutes of each hour, we'll want a leading '0'
    Serial.print('0');
  }
  Serial.print((epoch  % 3600) / 60); // print the minute (3600 equals secs per minute)
  Serial.print(':');
  if ( (epoch % 60) < 10 ) {
    // In the first 10 seconds of each minute, we'll want a leading '0'
    Serial.print('0');
  }
  Serial.println(epoch % 60); // print the second

//poso tha perimeni gia na stili message
 delay(1000);
 long t1 = millis();
 if (t1 - lastMsg > 2000) {
```

```
    lastMsg = t1;
    JsonObject& root = jsonBuffer.createObject();
    root["client_25"] =25;//(double)temp;
    root["time"]=(int)epoch;
    char* name = NULL;
    root.printTo(buffer,sizeof(buffer));
//to minima ginete publish.
    client.publish((char*) "/outTopic", buffer, false);
    }

  }
}
```