ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και υλοποίηση αστικού δικτύου LoRaWAN για εφαρμογές IoT

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Καλέμης

Επιβλέπων: Παναγιώτης Τσανάκας

Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και υλοποίηση αστικού δικτύου LoRaWAN για εφαρμογές IoT

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Καλέμης

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16η Ιουλίου 2024.

.....	.....	.....
Παναγιώτης Τσανάκας	Δημήτριος Σούντρης	Συμεών Παπαβασιλείου
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2024

……
Καλέμης Δημήτριος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Δημήτριος Καλέμης 2024
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς το συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν το συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Στην εποχή της ψηφιακής τεχνολογίας, η ραγδαία ανάπτυξη των επικοινωνιακών και πληροφοριακών συστημάτων έχει φέρει επανάσταση στην καθημερινή μας ζωή και στη λειτουργία των επιχειρήσεων. Οι έξυπνες συσκευές και οι καινοτόμες τεχνολογίες συνδέουν τα πάντα γύρω μας, επιτρέποντας τη δημιουργία ευφυών δικτύων που μπορούν να συλλέγουν και να αναλύουν δεδομένα σε πραγματικό χρόνο. Σε αυτό το δυναμικό περιβάλλον, το Διαδίκτυο των Πραγμάτων (IoT) ξεχωρίζει ως μία από τις πιο σημαντικές και πρωτοποριακές τεχνολογίες, επιτρέποντας τη σύνδεση και την επικοινωνία μεταξύ διάφορων συσκευών και συστημάτων. Το IoT δημιουργεί νέες ευκαιρίες για την αυτοματοποίηση, τη βελτίωση της απόδοσης και την εξοικονόμηση πόρων σε πολλούς τομείς. Οι εφαρμογές του κυμαίνονται από τις έξυπνες πόλεις και την αγροτική παραγωγή, μέχρι την παρακολούθηση της υγείας και την οικιακή αυτοματοποίηση. Μέσα σε αυτό το ευρύ πεδίο εφαρμογών, η ανάγκη για αξιόπιστες και αποδοτικές λύσεις επικοινωνίας καθίσταται πιο επιτακτική από ποτέ.

Σε αυτό το πλαίσιο, το πρωτόκολλο LoRa (Long Range) ξεχωρίζει ως μια λύση που παρέχει ασύρματη επικοινωνία μεγάλων αποστάσεων με χαμηλή κατανάλωση ενέργειας. Το LoRa έχει τη δυνατότητα να υποστηρίξει εφαρμογές IoT που απαιτούν ευρεία κάλυψη και αξιόπιστη μετάδοση δεδομένων, ακόμα και σε απομακρυσμένες ή δύσκολα προσβάσιμες περιοχές. Αυτή η τεχνολογία προσφέρει πλεονεκτήματα όπως η χαμηλή κατανάλωση ενέργειας, η μεγάλη εμβέλεια και η δυνατότητα σύνδεσης μεγάλου αριθμού συσκευών σε ένα δίκτυο.

Η παρούσα διπλωματική έχει ως στόχο να διερευνήσει και να υλοποιήσει ένα σύστημα IoT βασισμένο στην τεχνολογία LoRa, προκειμένου να επιτευχθεί η αποδοτική και αξιόπιστη διασύνδεση και διαχείριση συσκευών σε ένα ευρύ φάσμα εφαρμογών. Στο πλαίσιο αυτής της προσπάθειας, θα ενσωματωθεί και μια εφαρμογή ασφάλειας με τη χρήση PIR αισθητήρα (Passive Infrared Sensor) συνδεδεμένου στο δίκτυο LoRa. Ο αισθητήρας PIR θα χρησιμοποιηθεί για την ανίχνευση κίνησης, προσφέροντας μια πρακτική και αποδοτική λύση ασφάλειας για κατοικίες, επιχειρήσεις ή άλλες εγκαταστάσεις.

Με την υλοποίηση αυτής της εργασίας, θα αναλύσουμε τις αρχές λειτουργίας του LoRa, τα πλεονεκτήματα και τις προκλήσεις του, καθώς και πως μπορεί να στηθεί στην πράξη ένα τέτοιο δίκτυο. Επιπλέον, θα παρουσιάσουμε μια ολοκληρωμένη προσέγγιση για την ανάπτυξη ενός συστήματος IoT με LoRa στον τομέα της ασφάλειας, περιλαμβάνοντας το σχεδιασμό, την υλοποίηση και την αξιολόγηση της απόδοσής του.

Μέσα από αυτήν την προσέγγιση, φιλοδοξούμε να προωθήσουμε τη γνώση και την κατανόηση της τεχνολογίας LoRa, καθώς και να αναδείξουμε τη σημασία και τη δυναμική του IoT στη σύγχρονη κοινωνία και οικονομία.

Λέξεις κλειδιά

IoT, LoRa, LoRaWAN, Embedded systems, Flutter

Abstract

In the era of digital technology, the rapid development of communication and information systems has revolutionized our daily life and the operation of businesses. Smart devices and innovative technologies are connecting everything around us, enabling the creation of intelligent networks that can collect and analyse data in real time. In this dynamic environment, the Internet of Things (IoT) stands out as one of the most important and innovative technologies, enabling the connection and communication between different devices and systems. IoT creates new opportunities for automation, efficiency improvement and resource savings in many areas. Its applications range from smart cities and agricultural production to health monitoring and home automation. Within this wide range of applications, the need for reliable and efficient communication solutions is becoming more urgent than ever.

In this context, the LoRa (Long Range) protocol stands out as a solution that provides long-distance wireless communication with low power consumption. LoRa has the potential to support IoT applications that require wide coverage and reliable data transmission, even in remote or hard-to-reach areas. This technology offers advantages such as low power consumption, long range and the ability to connect a large number of devices in a network.

This thesis aims to investigate and implement an IoT system based on LoRa technology in order to achieve efficient and reliable interconnection and management of devices in a wide range of applications. As part of this effort, a security application using a Passive Infrared Sensor (PIR) connected to the LoRa network will be integrated. The PIR sensor will be used for motion detection, providing a practical and cost-effective security solution for homes, businesses or other facilities.

With the implementation of this work, we will analyze the operating principles of LoRa, its advantages and challenges, as well as how such a network can be set up in practice. Furthermore, we will present a comprehensive approach to deploy an IoT system with LoRa in the security domain, including its design, implementation and performance evaluation.

Through this approach, we aim to advance the knowledge and understanding of LoRa technology, as well as to highlight the importance and potential of IoT in modern society and economy.

Keywords

IoT, LoRa, LoRaWAN, Embedded systems, Flutter

Ευχαριστίες

Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες στον Καθηγητή κ. Παναγιώτη Τσανάκα για την εμπιστευτικότητα της παρούσας διατριβής και στον Υποψήφιο διδάκτορα κ. Κωνσταντίνο Αθανασιάδη για την πολύτιμη καθοδήγησή του σε όλη την εκπόνησή της.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου και τα αγαπημένα μου πρόσωπα που ήταν δίπλα μου καθ' όλη τη διάρκεια των σπουδών μου.

Καλέμης Δημήτριος,
Αθήνα, 16η Ιουλίου 2024

Περιεχόμενα

Ευρετήριο Εικόνων

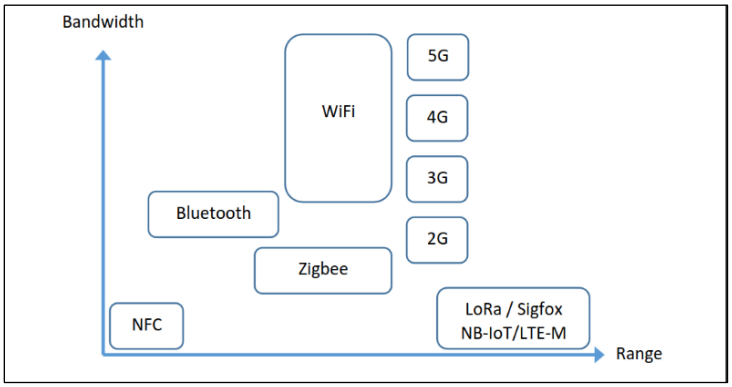
Κεφάλαιο 1: Θεωρητικό Πλαίσιο

1.1 Εισαγωγή

Εκτιμάται ότι υπάρχουν πάνω από 17 δισεκατομμύρια συσκευές συνδεδεμένες στο διαδίκτυο παγκοσμίως, καθώς και ότι ο αριθμός αυτός αναμένεται να διπλασιαστεί ως το 2030 [1]. Ο όγκος σε συνδυασμό με την ραγδαία αύξηση των συσκευών αυτών καθιστά αναμενόμενη την εξάπλωση του Internet of Things, το οποίο αφορά τη διαχείριση αυτών των συσκευών και τον τρόπο με τον οποίο αυτές συνδέονται στο διαδίκτυο. Ο όρος Internet of Things (IoT) ή Διαδίκτυο των Πραγμάτων (στα ελληνικά) δημιουργήθηκε στα τέλη της δεκαετίας του 1990 και αποδίδεται στον Kevin Ashton. Ο ίδιος αποτελεί έναν από τους ιδρυτές του Auto-ID center στο MIT και μέλος της ομάδας που ανακάλυψε τη μέθοδο να συνδέει αντικείμενα με το διαδίκτυο μέσω μιας ετικέτας RFID (ραδιοσυχνοτήτων). Το Internet of Things είναι μία έννοια στον χώρο της επιστήμης των υπολογιστών, η οποία περιγράφει το δίκτυο επικοινωνίας διαφόρων αντικειμένων, καθώς και την ανταλλαγή μηνυμάτων μεταξύ αυτών. Τα αντικείμενα αυτά, προκειμένου να εξυπηρετήσουν τον παραπάνω σκοπό, εμπεριέχουν ηλεκτρονικά μέσα, λογισμικό, αισθητήρες και συνδεσιμότητα σε δίκτυα. Η κεντρική ιδέα του Διαδικτύου των Πραγμάτων στηρίζεται σε ένα σύστημα υπολογιστών, το οποίο επικοινωνεί με μηχανικά ή ψηφιακά αντικείμενα και μπορεί να μεταφέρει δεδομένα μέσω διαδικτύου, δίχως την παρέμβαση κάποιου χρήστη ή με περιορισμένη την παρέμβαση αυτή [2].

Η ανάπτυξη του Διαδικτύου των Πραγμάτων έδωσε νέες δυνατότητες ενώ παράλληλα δημιούργησε καινούργιες ανάγκες. Οι συσκευές, δεν αρκεί να πραγματοποιούν τη λειτουργία τους σωστά, αλλά πρέπει να έχουν την απαραίτητη αξιοπιστία, ασφάλεια καθώς και την ελάχιστη δυνατή κατανάλωση κατά την επικοινωνία τους με το διαδίκτυο. Το γεγονός αυτό, οδήγησε στην ανάπτυξη διαφόρων τεχνολογιών ασύρματης επικοινωνίας με τις οποίες μια συσκευή μπορεί να συνδεθεί στο διαδίκτυο. Κάποιες από αυτές είναι η κινητή επικοινωνία (3G, 4G, 5G), μικρής εμβέλειας τεχνολογίες όπως τα bluetooth, Wi-Fi, NFC και μεγάλης εμβέλειας τεχνολογίες όπως τα LoRa, NB-Iot. Η κάθε μια τεχνολογία έρχεται με τα δικά της πλεονεκτήματα και μειονεκτήματα σε σχέση με την εμβέλεια, την ταχύτητα και τη κατανάλωση.

Στη παρούσα διπλωματική εργασία, μελετήθηκε το δίκτυο μεγάλης εμβέλειας LoRa και αναπτύχθηκε μια εφαρμογή.



Εικόνα 1: Protocols used in IoT

1.2 LoRa

1.2.1 Γενικά

Το θεωρητικό πλαίσιο για το κεφάλαιο “1.2 LoRa”, έχει βασιστεί στις πηγές [3], [4] και [5].

Η ασύρματη τεχνολογία LoRa (Long Range) αναπτύχθηκε από μια νεοφυή γαλλική εταιρεία, τη Cycleo, η οποία ανέπτυξε τη διαμόρφωση σήματος μεγάλης εμβέλειας (LoRa modulation). Το 2012, η Semtech εξαγόρασε τη Cycleo και πλέον έχει στη κατοχή της την πατέντα της διαμόρφωσης σήματος LoRa, η οποία είναι κλειστού κώδικα. Κατασκευαστές πλακετών, όπως η HopeRF και Microchip έχουν άδεια για κατασκευή μονάδων LoRa.

Το πρωτόκολλο LoRa (Long Range) είναι ένα ασύρματο πρωτόκολλο επικοινωνίας που χρησιμοποιείται για τη μετάδοση δεδομένων μεγάλης εμβέλειας με χαμηλή κατανάλωση ενέργειας. Ωστόσο, ο ρυθμός μετάδοσης δεδομένων είναι αρκετά περιορισμένος, συνεπώς δεν μπορεί να χρησιμοποιηθεί για μεταφορά μεγάλων πακέτων δεδομένων. Είναι σχεδιασμένο για εφαρμογές που απαιτούν αξιόπιστη ασύρματη επικοινωνία σε μεγάλες αποστάσεις, όπως οι αισθητήρες δικτύου Internet of Things, ασύρματα δίκτυα αισθητήρων, και άλλες εφαρμογές έξυπνων πόλεων και βιομηχανικών δικτύων. Επίσης χρησιμοποιείται και σε άλλους τομείς, όπως ο τομέας υγείας (παρακολούθηση περιβάλλοντος), η ασφάλεια (έξυπνος συναγερμός/φωτισμός) και η γεωργία (παρακολούθηση συνθήκες εδάφους).

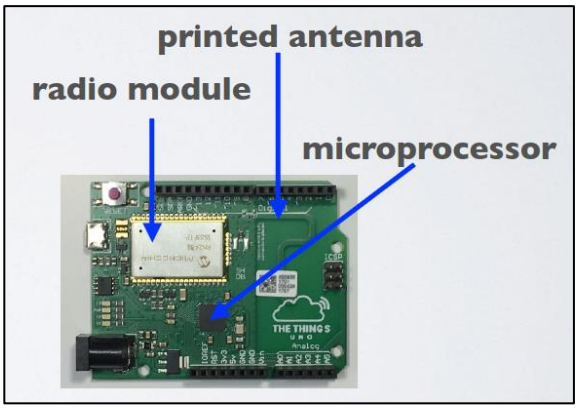
Το βασικό χαρακτηριστικό του πρωτοκόλλου LoRa είναι η μακρά εμβέλεια επικοινωνίας, η οποία μπορεί να φτάσει ακόμα και μερικά χιλιόμετρα, ανάλογα με τις συνθήκες του περιβάλλοντος. Η κάλυψη σε εσωτερικό χώρο εξαρτάται σε μεγάλο βαθμό από τον τύπο των υλικών του κτιρίου, ενώ σε εξωτερικό χώρο η εμβέλεια μπορεί να είναι σημαντικά πιο αυξημένη. Πιο συγκεκριμένα, σε κατοικημένη περιοχή, η επικοινωνία μπορεί να καλύπτει 2-5 χιλιόμετρα, σε επαρχία να καλύπτει 5-15 χιλιόμετρα, ενώ όταν υπάρχει ξεκάθαρη οπτική επαφή μπορεί να καλύψει και περισσότερο από 15 χιλιόμετρα. Το ρεκόρ απόστασης με διαμόρφωση Lora είναι 1336 χιλιόμετρα [3].

Το πρωτόκολλο LoRa λειτουργεί σε χαμηλές συχνότητες (συνήθως στα 868 MHz στην Ευρώπη ή 915 MHz στις Ηνωμένες Πολιτείες) και χρησιμοποιεί μια τεχνική διαμόρφωσης σήματος που ονομάζεται "Chirp Spread Spectrum" (CSS). Αυτή η τεχνική επιτρέπει στις συσκευές LoRa να μεταδίδουν δεδομένα μεγάλων αποστάσεων με χαμηλή ισχύ εκπομπής, επιτυγχάνοντας παράλληλα ανθεκτικότητα στις παρεμβολές και τις ανακλάσεις σήματος.

1.2.2 Βασική λειτουργία

Τα βασικά στοιχεία για το στήσιμο ενός δικτύου LoRa είναι:

- Πύλη (gateway): Υπολογιστικό σύστημα, το οποίο βρίσκεται μεταξύ πολλαπλών δικτύων, εφαρμογών ή συσκευών και μετατρέπει τις πληροφορίες και τα δεδομένα στην κατάλληλη μορφή ή πρωτόκολλο.
- Τελικός κόμβος (end node): Αποτελείται από ένα radio module με κεραία, το οποίο είναι υπεύθυνο για τη διαμόρφωση του σήματος και από ένα μικροεπεξεργαστή που διαχειρίζεται τα δεδομένα (πχ τα δεδομένα ενός αισθητήρα).



Εικόνα 2: The Things Uno (end node)

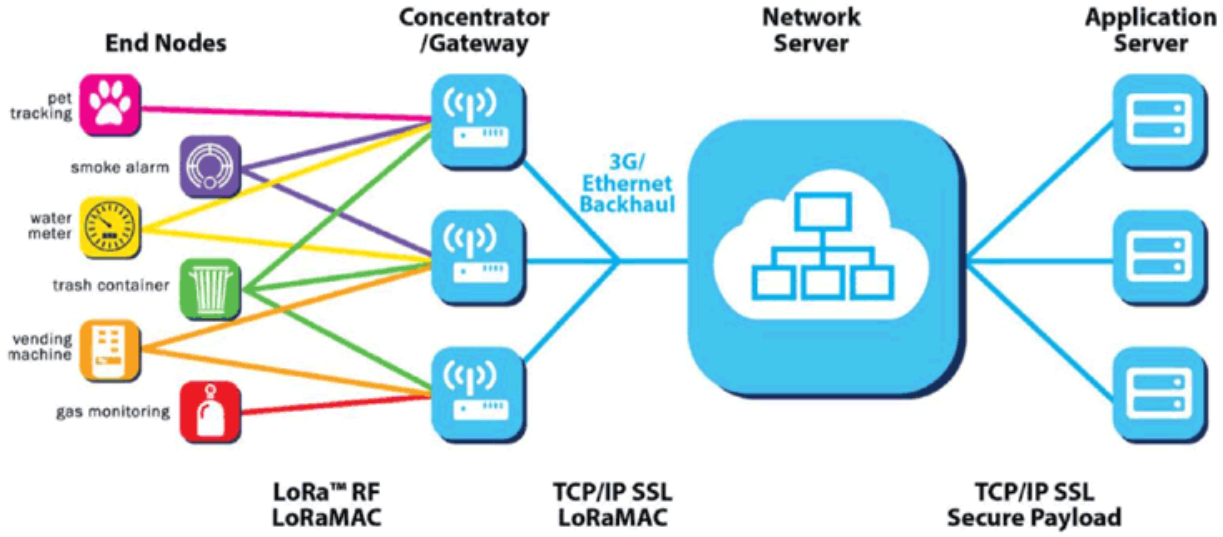
Εικόνα 3: The Things Gateway (gateway)

Τα end nodes συνήθως τροφοδοτούνται με μπαταρία και περιλαμβάνουν έναν ασύρματο πομποδέκτη, ο οποίος είναι υπεύθυνος τόσο για να στέλνει όσο και για να λαμβάνει μηνύματα. Από την άλλη, τα gateways συνήθως τροφοδοτούνται από κύρια τροφοδοσία και συνδέονται στο διαδίκτυο. Επιπλέον, πολλαπλά gateways μπορούν να λάβουν μηνύματα από τον ίδιο τελικό κόμβο, ενώ και να ακούν ταυτοχρόνως σε πολλές συχνότητες για εισερχόμενα σήματα. Γενικά, η επικοινωνία είναι αμφίδρομη, δηλαδή το gateway μπορεί να λάβει αλλά και να στείλει δεδομένα στον κόμβο και ο κόμβος αντίστοιχα. Η διαδικασία αποστολής μηνύματος από τον κόμβο στη πύλη ονομάζεται uplink, ενώ η αποστολή από την πύλη στον κόμβο downlink.

Όπως προαναφέρθηκε, το gateway συνδέεται στο διαδίκτυο και πιο συγκεκριμένα προωθεί τα πακέτα που λαμβάνει σε έναν διακομιστή δικτύου. Ο εξυπηρετητής αυτός, συλλέγει τα δεδομένα από όλες τις πύλες που επικοινωνούν μαζί του, φιλτράρει τα διπλά δεδομένα και αποφασίζει ποιο gateway έλαβε το καλύτερο σήμα (με βάση κάποιες μετρικές σήματος που αναλύονται παρακάτω). Έπειτα, προωθεί τα πακέτα στον κατάλληλο διακομιστή εφαρμογών (application server), όπου ο τελικός χρήστης μπορεί να τα δει.

Είναι σημαντικό να διευκρινιστεί πως το LoRa αναφέρεται στο ίδιο το σήμα που φέρει τα δεδομένα και στη τεχνική διαμόρφωση που εφαρμόζεται στο αρχικό σήμα, σε αντίθεση με το LoRaWAN, το οποίο αφορά το πρωτόκολλο επικοινωνίας και τον τρόπο με τον οποίο επικοινωνούν οι συσκευές εντός του δικτύου.

Παρακάτω φαίνεται η διαδικασία που αναφέρθηκε (Εικόνα 4):



Εικόνα 4: LoRaWAN flow

1.2.3 Κανόνες και Περιορισμοί

Το LoRa δραστηριοποιείται ζώνη ραδιοσυχνοτήτων ISM (Industrial, Scientific and Medical), η οποία δεν απαιτεί νόμιμη άδεια για χρήση και είναι διαθέσιμη διεθνώς[4]. Πιο αναλυτικά, στην Ευρώπη η ζώνη αυτή είναι στα 863-870MHz, ενώ στην Αμερική κυμαίνεται στο εύρος 902-928MHz. Σε άλλες γεωγραφικές περιοχές, όπως η Ασία, ο Καναδάς, η Αυστραλία και η Κίνα υπάρχουν διαφορετικές συχνοτικές μπάντες για ελεύθερη χρήση.

Συσκευές όπως φούρνοι μικροκυμάτων, ιατρικός εξοπλισμός κ.α. λειτουργούν στην ISM συχνοτική μπάντα. Η ζώνη έχει το πλεονέκτημα πως μπορεί να χρησιμοποιηθεί από όλους και πως δε χρειάζεται κάποια άδεια για τη χρήση της, όμως από την άλλη επιτρέπει μικρή ταχύτητα μετάδοσης καθώς και σημαντική παρεμβολή από άλλες συσκευές.

Εξαιτίας της μεγάλης χρήσης της ζώνης, έχουν εφαρμοστεί κανονισμοί που εφαρμόζουν σε όλους τους χρήστες της. Υπάρχουν πολλοί διεθνείς οργανισμοί που μεριμνούν για το συχνοτικό εύρος και εξασφαλίζουν την ασφαλή συνύπαρξη όλων των διαφορετικών ραδιοτεχνολογιών. Τους κανονισμούς έχει θέσει ο φορέας ETSI (European Telecommunications Standards Institute) στην Ευρώπη, ενώ αντίστοιχα στην Αμερική ο οργανισμός FCC (Federal Communications Commission). Η κάθε χώρα ξεχωριστά μπορεί να εφαρμόσει επιπλέον κανονισμούς βασιζόμενη και τηρώντας τους κανόνες που θέτει ο ETSI ή ο FCC.

Ειδικότερα, οι χρήστες της συχνοτικής ζώνης ISM της Ευρώπης (863MHz – 870MHz) πρέπει να τηρούν τους παρακάτω περιορισμούς:

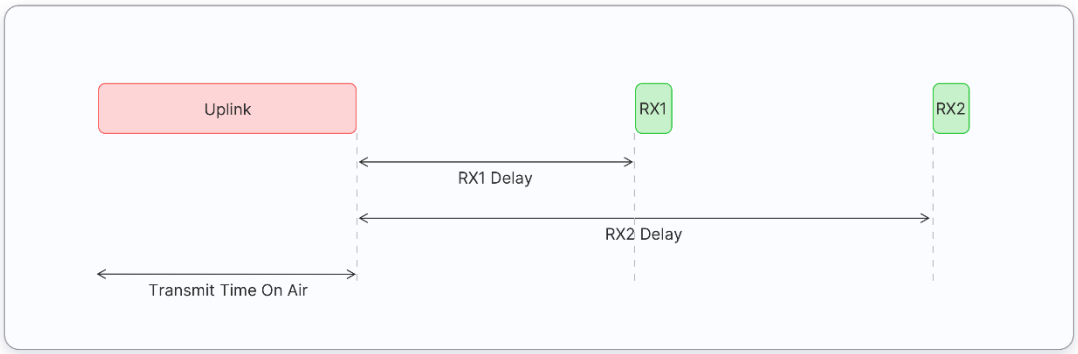
- Για το uplink, η μέγιστη ισχύς μεταφοράς σε Watt να μη ξεπερνά τα 25mW (14 dBm)
- Το duty cycle να είναι μεταξύ 0.1% και 1% αναλόγως το κανάλι (πχ για duty cycle 1%: για κάθε ένα δευτερόλεπτο που βρίσκεται το πακέτο στη μετάδοση - time on air - να περιμένει τουλάχιστον άλλα 99 δευτερόλεπτα για να στείλει το επόμενο πακέτο)
- Το κέρδος (gain) της κεραίας να μη ξεπερνά τα +2.15 dBi

Επιπλέον κανονισμοί μπορούν να επιβληθούν και από τον διαχειριστή δικτύου (πχ το The Things Network).

1.2.4 Τύποι συσκευών

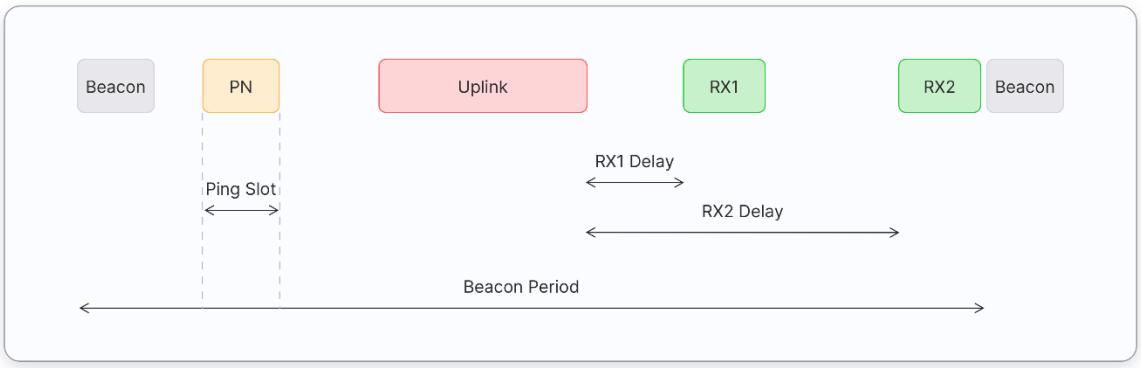
Οι προδιαγραφές του LoRaWAN προβλέπουν τρεις τύπου συσκευών:

a. A(ll): Οι τύπου A συσκευές τροφοδοτούνται με μπαταρίες και μετά από κάθε uplink προς τα gateways ανοίγουν δύο μικρά χρονικά παράθυρα λήψης για ενδεχόμενα downlinks. Οποιαδήποτε στιγμή ο πομπός μπορεί να στείλει σήμα και μετά από χρόνο RX1 Delay ή RX2 Delay θα μπορεί να λάβει μηνύματα για χρονικό διάστημα RX1 και RX2 αντίστοιχα. Το gateway μπορεί να απαντήσει στο πρώτο ή στο δεύτερο παράθυρο, αλλά όχι και στα δύο.



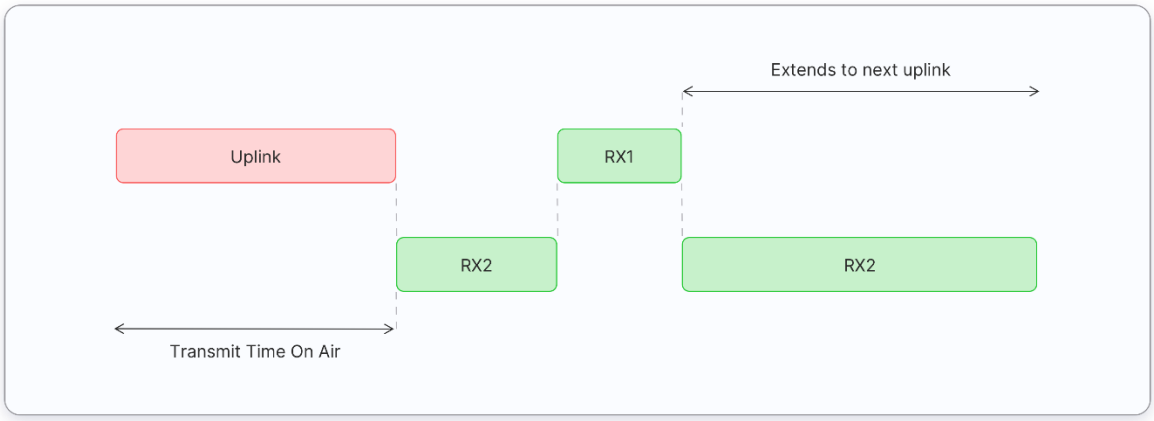
Εικόνα 5: Συσκευή Τύπου A

b. B(eacon): Οι τύπου B συσκευές είναι ίδιες με τις τύπου A και επιπρόσθετα ανοίγουν επιπλέον προγραμματισμένα παράθυρα λήψης για downlinks (PN παράθυρο στο διάγραμμα). Το end node λαμβάνει μηνύματα Beacon από το gateway για συγχρονισμό, επιτρέποντας στο gateway να γνωρίζει πότε ακούει το end node.



Εικόνα 6: Συσκευή Τύπυ B

c. C(ontinuous): Οι τύπου C συσκευές λειτουργούν όπως οι τύπου A, αλλά ακούν συνεχώς για downlinks. Συνεπώς, καταναλώνουν περισσότερη ενέργεια και συνήθως τροφοδοτούνται από κύριες πηγές ρεύματος.



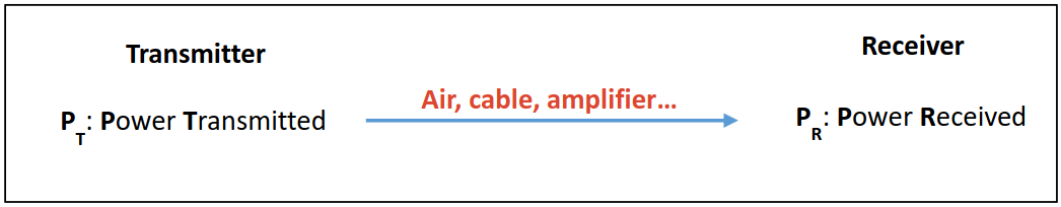
Εικόνα 7: Συσκευή Τύπου C

1.2.5 Μονάδα Decibel

Το decibel (dB) είναι μια μονάδα μέτρησης, η οποία χρησιμοποιείται για να εκφράσει την αναλογία δυνάμεων, ηχητικών πιέσεων ή άλλων πραγμάτων σε λογαριθμική κλίμακα. Η μονάδα αυτή εφευρέθηκε από τα εργαστήρια Bell και ορίζεται ως εξής:

$$A = 10 \times log_{10}\left(\frac{P_R}{P_T}\right) dB$$

Όπου για ηλεκτρικές δυνάμεις (δηλαδή Watts) έχουμε P_T την ισχύ εισόδου και P_R την ισχύ εξόδου.



Εικόνα 8: Μετάδοση μεταξύ πομπού και δέκτη

Αν το αποτέλεσμα σε dB είναι αρνητικός αριθμός (-), τότε έχουμε εξασθένιση του σήματος (attenuation), ενώ αν το αποτέλεσμα είναι θετικό (+), τότε πρόκειται για ενίσχυση του σήματος (gain). Εξασθένιση υπάρχει κατά τη μεταφορά με μέσο τον αέρα ή κάποιο καλώδιο και ενίσχυση επιτυγχάνεται με κάποιον ενισχυτή.

Για τη μετατροπή από Decibel σε αναλογία ισχύος, χρησιμοποιούμε την εξής σχέση:

$$\frac{P_R}{P_T} = 10^{\frac{A}{10}}$$

Η λογαριθμική κλίμακα χρησιμοποιείται προκειμένου να αποφευχθούν πολύ μεγάλοι ή πολύ μικροί αριθμοί (αναλογίες) και να απλοποιηθεί η υπολογιστική διαδικασία. Για παράδειγμα, όταν η ισχύς εξόδου είναι 1,000,000 φορές μεγαλύτερη από την ισχύ εισόδου, τότε το μέγεθος σε Decibel είναι μόλις 60dB. Αντίστοιχα, όταν το πηλίκο P_R/P_T είναι 0,000001 τότε το μέγεθος εκφρασμένο σε Decibel είναι -60dB.

Αν θέσουμε την ισχύ εισόδου σε 1mW (δηλαδή $P_T = 1mW$), τότε η αναλογία ισχύος εκφράζεται σε dBm. Πιο αναλυτικά:

$$A = 10 \times log_{10}\left(\frac{P_R}{1}\right) dBm$$

$$\frac{P_R}{1} = 10^{\frac{A}{10}} \rightarrow P_R = 10^{\frac{A}{10}} mW$$

1.2.6 Διάδοση σήματος

Διάδοση σήματος είναι ο τρόπος με τον οποίο τα ραδιοκύματα ταξιδεύουν στο χώρο (ή σε κάποιο μέσο διάδοσης). Ο τρόπος αυτός, μπορεί να επηρεάσει σε μεγάλο βαθμό την ισχύ του σήματος.

Στην περίπτωση της απευθείας οπτικής επαφής (direct line of sight), το σήμα μεταβαίνει από τον πομπό στο δέκτη χωρίς κανένα εμπόδιο. Με την αύξηση της απόστασης μεταξύ πομπού και δέκτη, το σήμα εξασθενεί. Η απώλεια αυτή είναι γνωστή ως Free Space Loss. Ωστόσο, ακόμα και αν υπάρχει απευθείας οπτική επαφή, αν υπάρχουν εμπόδια κοντά στην διαδρομή αυτή(μέσα στη ζώνη Fresnel), τα ραδιοκύματα μπορεί να ανακλαστούν στα εμπόδια αυτά και να φτάσουν με διαφορετική φάση από το αρχικό σήμα. Αυτό έχει ως επακόλουθο την εξασθένιση του αρχικού σήματος και την εισαγωγή θορύβου στον δέκτη.

Ακόμα, το σήμα μπορεί να ταξιδέψει μέσα από εμπόδια, αλλά εξασθενεί ειδικότερα στη περίπτωση που τα εμπόδια αυτά αποτελούνται από αγώγιμα υλικά. Συνεπώς, η ισχύς του σήματος μειώνεται περισσότερο όταν το σήμα αναγκάζεται να ταξιδέψει μέσα από κτίρια (τα οποία αποτελούνται από αγώγιμα υλικά) σε σχέση με όταν περνάει μέσα από δέντρα. Μεγάλα εμπόδια όπως βουνά αποκόπτουν την επικοινωνία εντελώς. Τέλος, παρατηρείται το φαινόμενο την ανάκλασης του σήματος σε κτίρια και της περίθλασης γύρω από αιχμηρά αντικείμενα.

Η απώλεια free space loss υπολογίζεται ως εξής:

$$L_{fs} = 32.45 + 20 \times \log(D) + 20 \times \log(f)$$

Όπου L_{fs} η απώλεια free space loss σε dB, D η απόσταση μεταξύ του gateway και του end node σε χιλιόμετρα και f η συχνότητα σε MHz.

Για παράδειγμα για την Ευρώπη (f = 868MHz):

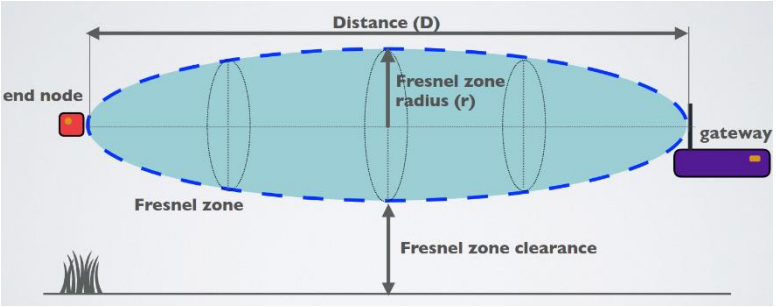
$$D = 1.00\ km, L_{fs} = 32.45 + 20 \times \log(1.00) + 20 \times \log(868) = 91dB$$

1.2.7 Ζώνη Fresnel

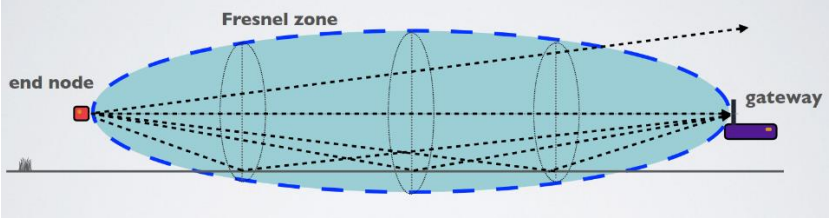
Η ζώνη Fresnel (Fresnel Zone) είναι μια ελλειπτική καμπύλη γύρω από την απευθείας οπτική επαφή μεταξύ του κόμβου και του gateway. Οποιοδήποτε εμπόδιο (κτίρια, δέντρα, έδαφος κλπ.) μέσα στον όγκο που ορίζει η καμπύλη αυτή μπορεί να εξασθενίσει το σήμα και ας υπάρχει οπτική επαφή ανάμεσα στις δύο συσκευές. Η μέγιστη ακτίνα της ζώνης Fresnel, η οποία βρίσκεται στα μισά της απόστασης κόμβου και πύλης δίνεται από τη σχέση:

$$r = 8.657 \times \sqrt{\frac{D}{f}}$$

Όπου r η ακτίνα της ζώνης Fresnel σε μέτρα, D η απόσταση κόμβου και πύλης σε χιλιόμετρα και f η συχνότητα σε GHz.



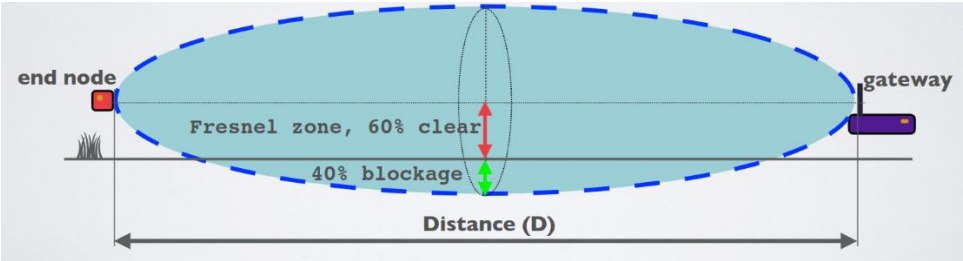
Εικόνα 9: Ζώνη Fresnel χωρίς εμπόδια



Εικόνα 10: Ζώνη Fresnel με εμπόδιο

Από τα πρώτο διάγραμμα φαίνεται πως όταν η Fresnel ζώνη είναι δίχως εμπόδια, το σήμα καταφθάνει χωρίς πρόβλημα, ενώ αντίθετα στη δεύτερη περίπτωση το σήμα καταφθάνει με διαφορά φάσης στο πομπό και είναι πιθανό τα πλάτη των διαφορετικών σημάτων να εξουδετερωθούν και να αλλοιωθεί ή ακόμη και να εξασθενίσει πλήρως το σήμα. Επομένως, είναι καλή πρακτική να τοποθετείται η πύλη σε όσο το δυνατόν πιο υψηλό σημείο, προκειμένου να αποφεύγονται όσο το δυνατόν περισσότερα εμπόδια μέσα στη ζώνη Fresnel.

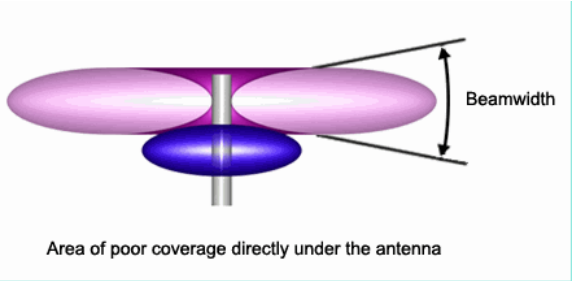
Σα γενικός κανόνας, η ζώνη αυτή θα πρέπει ιδανικά να είναι δίχως εμπόδια, όμως επειδή αυτό μπορεί στη πράξη να είναι μη υλοποιήσιμο, θεωρούμε πως τα εμπόδια στο 40% της ζώνης εισάγουν αμελητέα απώλεια σήματος και καταλήγουμε στο παρακάτω σχήμα:



Εικόνα 11: Ζώνη Fresnel με περιθώριο εμποδίων

Η παραδοχή αυτή, εξασφαλίζει μεγαλύτερη ευελιξία και πρακτικότητα σε πραγματικές εφαρμογές. Εν τέλει για τη βέλτιστη απόδοση μετάδοσης του σήματος είναι σημαντικό να εφαρμόζονται τα εξής:

- Η κεραία του gateway να τοποθετείται εξωτερικά και σε ψηλό σημείο (ώστε να αποφεύγονται τα εμπόδια στη ζώνη Fresnel)
- Τόσο η πύλη όσο και ο κόμβος να λειτουργούν βελτιστοποιημένα για τη συγκεκριμένη συχνότητα
- Η κεραία του gateway και του end node να παραμένει κάθετη και να είναι omnidirectional για κάλυψη μεγαλύτερης περιοχής

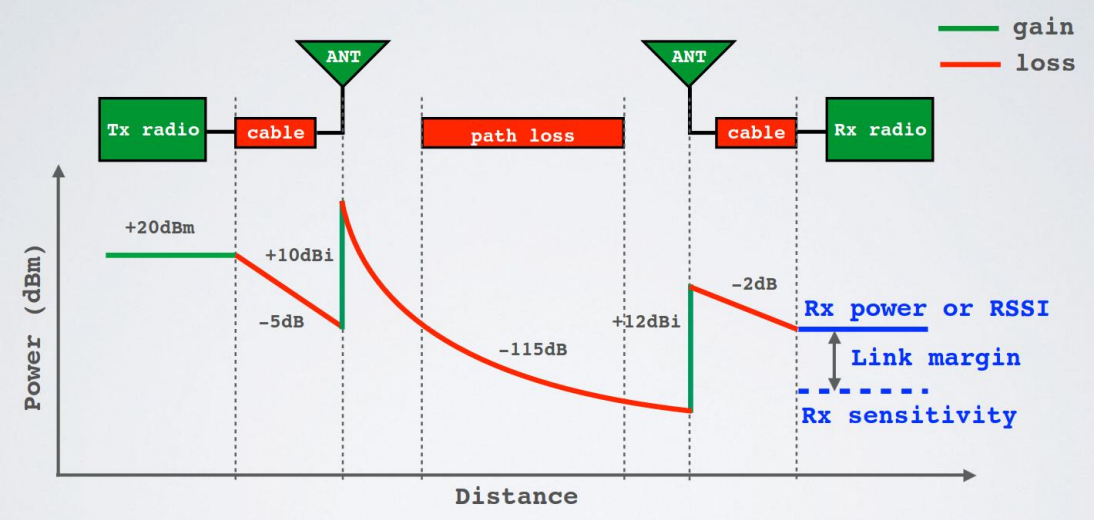


Εικόνα 12: Κάλυψη κεραίας omnidirectional

1.2.8 Link Budget, RSSI, SNR

Διαμόρφωση (modulation) σήματος είναι όταν το σήμα εισόδου επιβάλλεται σε ένα σήμα φορέα, ενώ η αντίθετη διαδικασία όπου ανακτάται το αρχικό σήμα ονομάζεται αποδιαμόρφωση (demodulation). Κάθε σήμα προκειμένου να σταλεί από τον πομπό στο δέκτη, διαμορφώνεται από τον πομπό για να αποσταλεί και αποδιαμορφώνεται από τον δέκτη για να ληφθεί ορθά το μήνυμα.

Στο παρακάτω διάγραμμα (εικόνα 13) απεικονίζεται ένα απλό σύστημα εκπομπού και δέκτη. Με κόκκινο χρώμα φαίνονται οι απώλειες, ενώ με πράσινο οι ενισχύσεις του σήματος σε dB. Αρχικά το σήμα από τον πομπό μεταφέρεται μέσω καλωδίου στην κεραία όπου και ενισχύεται. Έπειτα μεταφέρεται μέσω του αέρα (το μέσο μετάδοσης) όπου και εξασθενεί, ενώ όταν ληφθεί από την κεραία του δέκτη ενισχύεται και πάλι. Με $T_x\ power$ συμβολίζεται η ισχύς μετάδοσης, ενώ με $R_x\ power$ ή RSSI η ισχύς λήψης. $R_x\ sensitivity$ είναι το ελάχιστο δυνατό κατώφλι ισχύος που πρέπει να έχει το σήμα προκειμένου να γίνει αντιληπτό και να αποδιαμορφωθεί από τον δέκτη. Το link budget ορίζεται ως το άθροισμα όλων των ενισχύσεων και απωλειών από τον πομπό στο μέσο μετάδοσης και τελικά στον δέκτη. Αποτελεί ένα μέγεθος αξιολόγησης του συστήματος.



Εικόνα 13: Tx, Rx, Link margin

Η φόρμουλα που αποδίδει το link budget είναι ο παρακάτω:

$$Received\ Power = Transmitted\ power + Gains - Losses$$

Το link margin περιγράφει την διαφορά μεταξύ της ισχύος του ληφθέντος σήματος από την ευαισθησία του δέκτη, δηλαδή:

$$Link\ margin\ (dBm) = Received\ power\ (dBm) - Receiver\ sensitivity(dBm)$$

Το μέγεθος αυτό πρέπει να είναι θετικός αριθμός (Received power > Receiver sensitivity) και επιπλέον να είναι τουλάχιστον μερικά dB προκειμένου να αποδιαμορφωθεί με επιτυχία. Η διαμόρφωση LoRa είναι αρκετά ευαίσθητη και μπορεί να αντιληφθεί σήματα μέχρι και -148 dBm [8]. Ένα ακόμη χρήσιμο μέγεθος που χρησιμοποιείται για την αξιολόγηση ραδιοσυστημάτων είναι το maximum link budget το οποίο ορίζεται ως εξής:

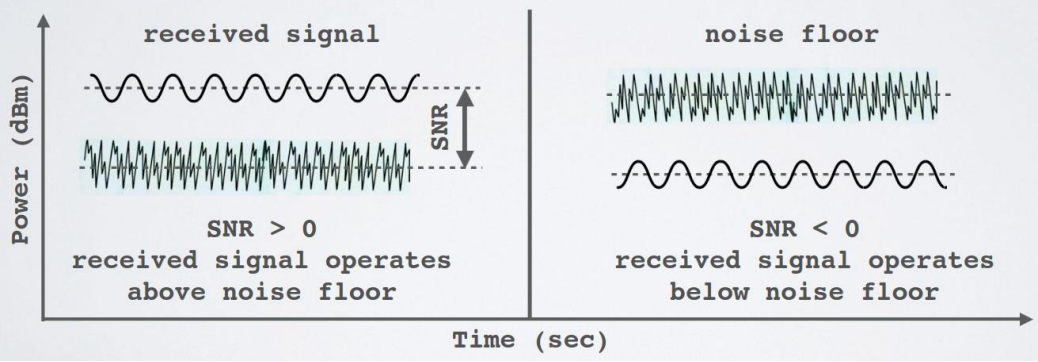
$$Maximum\ link\ budget = Maximum\ transmitter\ power - Lowest\ receiver\ sensitivity$$

Και περιγράφει τη διαφορά της μέγιστης ισχύος του σήματος από τον πομπό από την ελάχιστη ευαισθησία του δέκτη.

Επιπρόσθετα, ορίζονται τα μεγέθη RSSI και SNR για την αξιολόγηση του ληφθέντος σήματος. Το RSSI (Received Signal Strength Indicator) είναι η ισχύς του σήματος λήψης σε mW και μετράται σε dBm. Διαισθητικά, αποτελεί μια μετρική αξιολόγησης του πόσο καλά ακούει ένα σήμα ο δέκτης από τον πομπό. Το μέγεθος αυτό έχει αρνητική τιμή και όσο πιο κοντά είναι στο μηδέν, τόσο καλύτερο είναι το σήμα. Ενδεικτικές τιμές για καλό σήμα είναι κοντά στο -30dBm, ενώ ένα αδύναμο μπορεί να έχει τιμή κοντά στο -120dBm.

Το μέγεθος SNR (Signal-to-Noise Ratio), ορίζεται ως το πηλίκο της ισχύος του σήματος λήψης δια την ισχύ του επιπέδου του θορύβου. Το επίπεδο θορύβου αποτελείται από όλες τις μη επιθυμητές και παρεμβαλλόμενες πηγές σημάτων που μπορούν να διαφθείρουν το μεταδιδόμενο σήμα και να εξαναγκάσουν την επαναποστολή του. Το μέγεθος SNR μπορεί να λάβει θετικές και αρνητικές τιμές, αναλόγως με το αν το επίπεδο θορύβου βρίσκεται κάτω ή πάνω από το σήμα λήψης. Είναι σημαντικό

να επισημανθεί πως συνήθως το επίπεδο θορύβου αποτελεί το όριο ευαισθησίας του δέκτη, όμως το LoRa λειτουργεί και κάτω από το επίπεδο αυτό. Τυπικές τιμές για το Signal-to-Noise Ratio είναι μεταξύ -20dB και +10dB και όσο μεγαλύτερη είναι η τιμή αυτή, τόσο καλύτερο είναι το σήμα.



Εικόνα 14: SNR

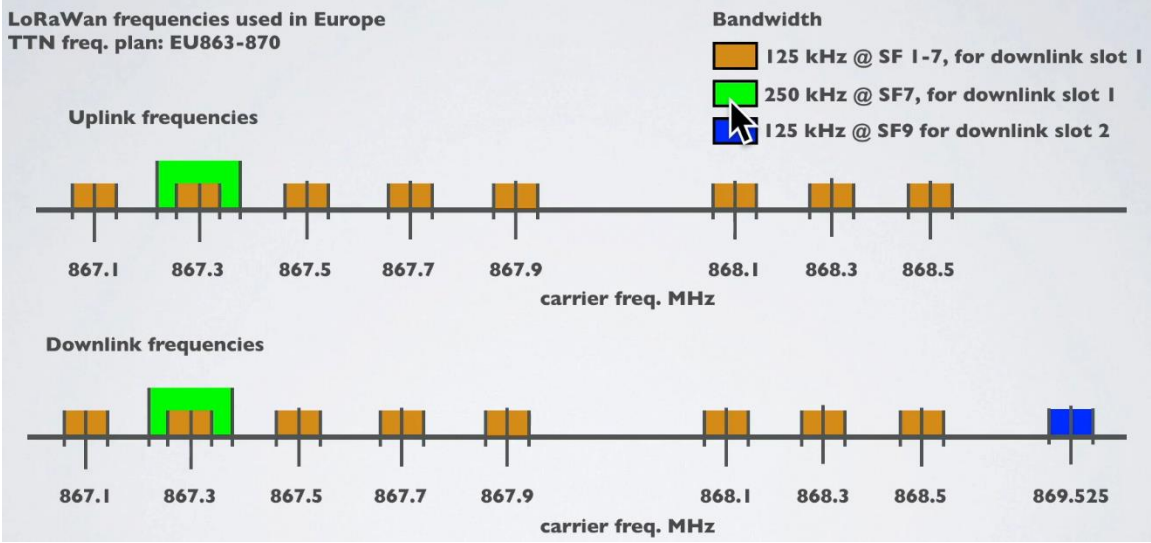
1.2.9 Συχνότητες φέροντος και εύρη ζώνης

Όπως έχει ήδη αναφερθεί, το LoRaWAN λειτουργεί στη συχνοτική ζώνη ISM. Πιο συγκεκριμένα, στη παρούσα εργασία αναλύεται η ζώνη EU863-870 ISM, αφού πάνω σε αυτή πραγματοποιήθηκε το πρακτικό κομμάτι. Περισσότερες πληροφορίες για αυτή και άλλες ζώνες βρίσκονται στην ιστοσελίδα του LoRa Alliance [9]. Εκεί περιέχονται μεταξύ άλλων τα εγκεκριμένα πλάνα συχνοτικών καναλιών για διάφορες γεωγραφικές περιοχές καθώς και οι αντίστοιχοι κανονισμοί. Στην Ευρώπη (EU863-870) είναι διαθέσιμα 9 κανάλια για uplinks, τα οποία αναγράφονται παρακάτω (Ως κανάλι ορίζεται μια ομάδα συχνοτήτων φέροντος σήματος με επιπλέον πληροφορίες, όπως το spreading factor ή το bandwidth range που αναλύονται στη συνέχεια):

Channel	Uplink frequency (MHz)	Spreading factor & Bandwidth range
0	868.1	SF7BW125 to SF7BW125
1	868.3	SF7BW125 to SF7BW125 and SF7BW250
2	868.5	SF7BW125 to SF7BW125
3	867.1	SF7BW125 to SF7BW125
4	867.3	SF7BW125 to SF7BW125
5	867.5	SF7BW125 to SF7BW125
6	867.7	SF7BW125 to SF7BW125
7	867.9	SF7BW125 to SF7BW125

Τα πρώτα τρία κανάλια (με πορτοκαλί χρώμα) πρέπει να εφαρμόζονται σε όλες τις συσκευές που χρησιμοποιούν το EU868MHz, ενώ οι υπόλοιπες μπορούν να οριστούν ελεύθερα από τον διαχειριστή του δικτύου. Εδώ, αυτές έχουν οριστεί με βάση το TTN (The Things Network).
Για τα downlinks τα ίδια κανάλια με τα uplinks χρησιμοποιούνται για το πρώτο παράθυρο λήψης (RX1), ενώ για το δεύτερο παράθυρο (RX2) υπάρχει ένα επιπλέον κανάλι. Τα παράθυρα εξηγούνται αναλυτικά στο κεφάλαιο 1.2.4 «Τύποι συσκευών». Πιο αναλυτικά για τα downlinks έχουμε:

Download Frequency (MHz)	Spreading factor & Bandwidth range
	Uplink channels 0-8 (RX1)
869.525	SF9BW125 (RX2 downlink only)



Εικόνα 15: Κανάλια Uplink και Downlink

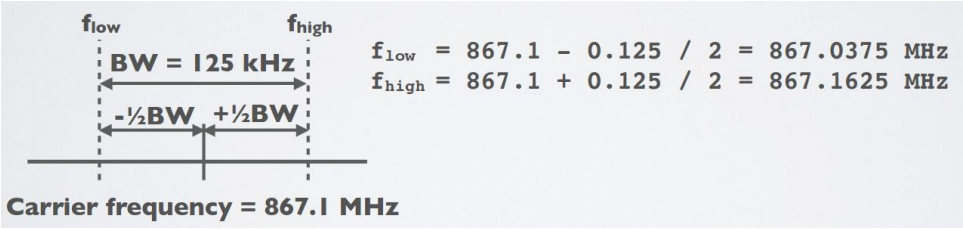
Σύμφωνα με τους κανονισμούς του ETSI, η ζώνη 863-870MHz χωρίζεται σε 5 υποζώνες: G, G1, G2, G3, G4. Κάθε υποζώνη έχει διαφορετικούς περιορισμούς σχετικά με το EIRP, το duty cycle και το εύρος ζώνης (bandwidth) του καναλιού. Πιο συγκεκριμένα:

Name	Band (MHz)	Limitations
G	863.0 - 868.0	ERP < 25mW, duty cycle < 1%
G1	868.0 – 868.6	ERP < 25mW, duty cycle < 1%
G2	868.7 – 869.2	ERP < 25mW, duty cycle < 0.1%
G3	869.4 – 869.65	ERP < 500mW, duty cycle < 10%
G4	869.7 – 870.0	ERP < 25mW, duty cycle < 1%

Η σχέσεις που συνδέουν τη συχνότητα φέροντος με το εύρος ζώνης δίνεται παρακάτω:

$$f_{low} = carrier\ frequency - \frac{bandwidth}{2}, f_{high} = carrier\ frequency - \frac{bandwidth}{2}$$

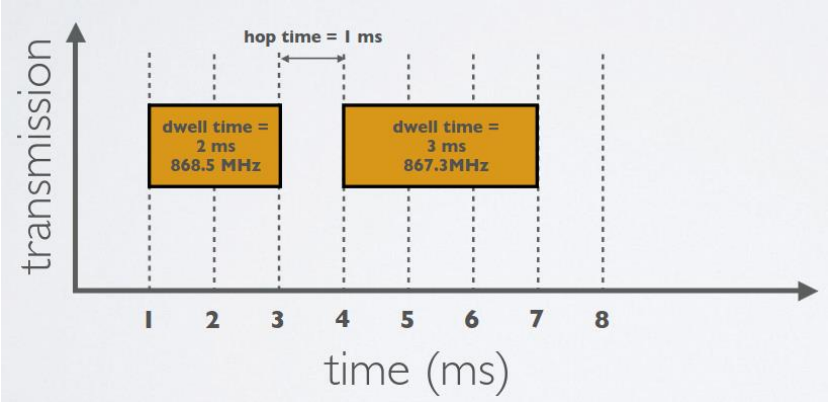
Δηλαδή για εύρος ζώνης 125kHz και συχνότητα φέροντος 867.1MHz έχουμε το εξής:



Εικόνα 16: Παράδειγμα εύρους ζώνης και συχνότητας φέροντος

Ένα end device δεν χρησιμοποιεί μόνο ένα κανάλι για να στέλνει μηνύματα, αλλά αξιοποιεί για κάθε μετάδοση διαφορετικό κανάλι με έναν ψευδο-τυχαίο τρόπο. Επομένως, στην Ευρώπη έχουμε 8 κανάλια επικοινωνίας για τα uplinks και 9 για τα downlinks. Το γεγονός αυτό δημιουργεί την ανάγκη να εισάγουμε ακόμα δύο μεγέθη: το Dwell time (χρόνος μετάδοσης) και Hop time (χρόνος αναπήδησης).

Ως χρόνος μετάδοσης εννοείται ο χρόνος που απαιτείται για την μετάδοση σε μια συχνότητα, ενώ χρόνος αναπήδησης είναι το χρονικό διάστημα που αναμένεται για την εναλλαγή του τελικού κόμβου από μια συχνότητα σε μια άλλη.

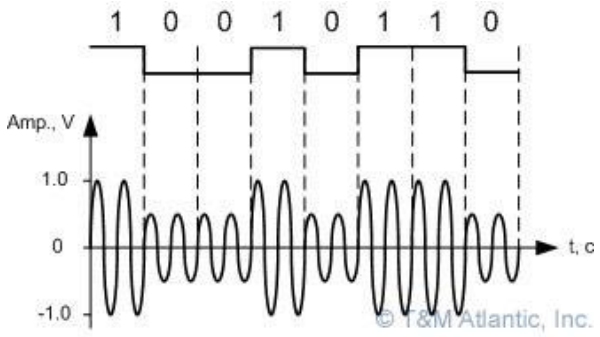


Εικόνα 17: Χρόνος μετάδοσης και χρόνος αναπήδησης

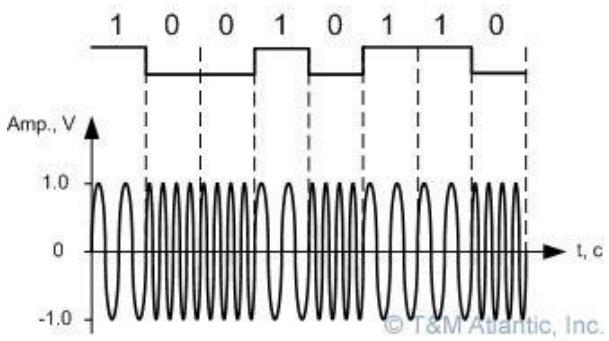
1.2.10 Τύποι διαμόρφωσης και CSS

Διαμόρφωση σήματος είναι η διαδικασία κωδικοποίησης ένας αναλογικού ή ψηφιακού σήματος εντός ενός φέροντος σήματος. Η διαμόρφωση πραγματοποιείται στον κωδικοποιητή του πομπού και είναι απαραίτητη προκειμένου να πραγματοποιηθεί η μετάδοση του σήματος, ενώ η αντίστροφη διαδικασία πραγματοποιείται στην πλευρά του δέκτη για να ληφθεί το αρχικό σήμα. Όπως έχει αναφερθεί, η επικοινωνία μεταξύ της πύλης και των τελικών κόμβων είναι αμφίδρομη, συνεπώς τόσο η πύλη όσο και οι τελικοί κόμβοι αναφέρονται και ως πομποδέκτες (transceivers) στο LoRaWAN.

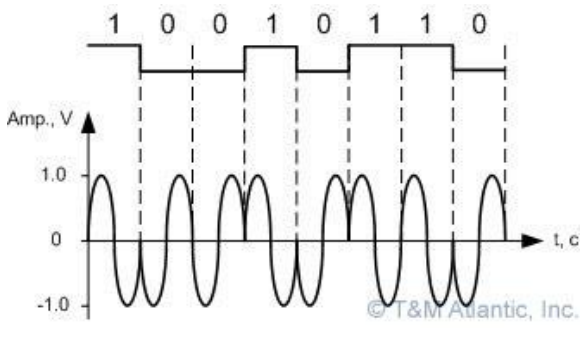
Όσον αφορά τα αναλογικά σήματα, υπάρχουν τρεις τύποι διαμόρφωσης: Διαμόρφωση Πλάτους ή Amplitude Modulation (AM), Διαμόρφωση Συχνότητας ή Frequency Modulation (FM) και Διαμόρφωση Φάσης ή Phase Modulation (PM). Από την άλλη, όταν η πληροφορία είναι σε ψηφιακή μορφή, οι αντίστοιχες διαμορφώσεις είναι οι εξής: Διαμόρφωση Μετατόπισης Πλάτους ή Amplitude Shift Keying (ASK), Διαμόρφωση Μετατόπισης Συχνότητας ή Frequency Shift Keying (FSK) και Διαμόρφωση Μετατόπισης Φάσης ή Phase Shift Keying (PSK).



Εικόνα 18: Aplitude Shift Keying (ASK)

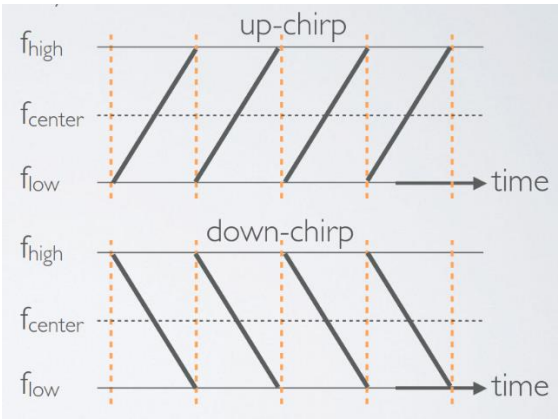


Εικόνα 19: Frequency Shift Keying (FSK)



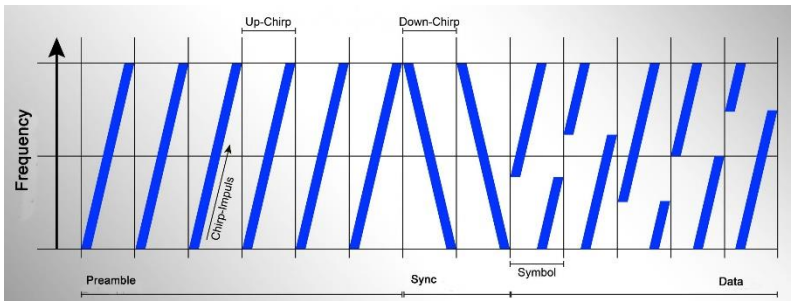
Εικόνα 20: Phase Shift Keyinh (PSK)

Πέρα από τις τρεις βασικές διαμορφώσεις, υπάρχουν και άλλοι τύποι διαμορφώσεων. Το LoRa είναι μια τέτοια περίπτωση ιδιόκτητης ευρέος φάσματος διαμόρφωση (spread spectrum modulation), η οποία βασίζεται στο Chirp Spread Spectrum (CSS). Η τεχνική αυτή, χρησιμοποιεί παλμούς γραμμικά αυξανόμενης συχνότητας (chirp pulses) για τη κωδικοποίηση της πληροφορίας. Όταν η συχνότητα αυξάνεται κατά τη διάρκεια του chirp, τότε γίνεται λόγος για up-chirp, ενώ στην αντίθετη περίπτωση για down-chirp.



Εικόνα 21: Up-chirp και Down-chirp

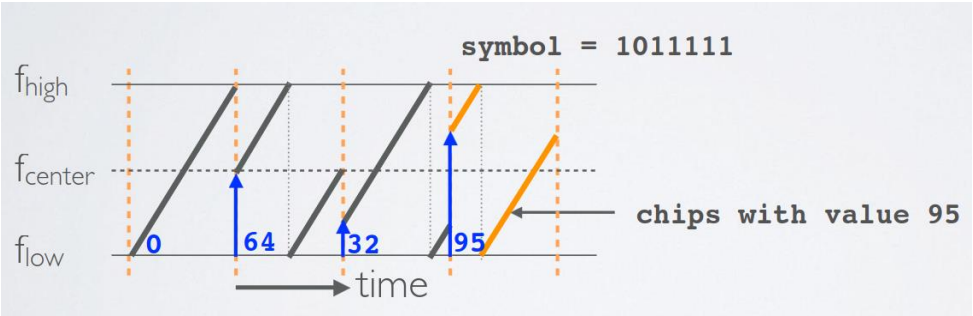
Τα chirp χρησιμοποιούνται ως σήματα φορείς, όπου κωδικοποιείται το αρχικό μήνυμα. Στο LoRa, η αρχή του μηνύματος επισημαίνεται με έναν αριθμό από up-chirps και έπειτα δύο down-chirps. Τα chirps μετατοπίζονται κυκλικά, δηλαδή την επόμενη χρονική στιγμή από τα σημεία που μεγιστοποιείται η συχνότητα (f_{max}), μεταπηδούν στην ελάχιστη συχνότητα (f_{min}). Επιπλέον, υπάρχουν μεταπηδήσεις στο πεδίο της συχνότητας (frequency jump), οι οποίες καθορίζουν την κωδικοποίηση των δεδομένων.



Εικόνα 22: Μήνυμα LoRa

Ένα σύμβολο (symbol) μπορεί να αναπαραστήσει ένα ή περισσότερα bits δεδομένων. Για παράδειγμα το σύμβολο 1011011 (δεκαδικό 91) μπορεί να κωδικοποιήσει 7 bit δεδομένων, δηλαδή μπορούμε να πούμε ότι έχει Spreading Factor 7 (SF7). Για αυτόν τον παράγοντα εξάπλωσης (SF) το σύμβολο μπορεί να λάβει 2^7τιμές (γενικά 2^{SF}), συνεπώς τιμές στο διάστημα 0 – 127. Η τιμή του συμβόλου αυτού είναι εκείνη που κωδικοποιείται σε up-chirp και μπορεί να σταλεί ως δεδομένο.

Το up-chirp χωρίζεται σε 2^{SF} διαβαθμίσεις ή chips. Για παράδειγμα, αν το σύμβολο είναι το 1011111 (δεκαδικό 95), ο αριθμός των bit που μπορούν να κωδικοποιηθούν είναι 7 όπως και προηγουμένως, δηλαδή SF=7 και το σήμα μπορεί να διαχωριστεί σε $2^{SF} = 2^7 = 128$ chips. Η πιθανή θέση από τις 128 διαβαθμίσεις που ξεκινάει το up-chirp καθορίζει την τιμή του συμβόλου που κωδικοποιεί.



Εικόνα 23: Διαμόρφωση LoRa με SF7

Ο spreading factor στη διαμόρφωση LoRa στη ζώνη ISM EU863-870 παίρνει ακέραιες τιμές από 7 έως 12. Ο παράγοντας αυτός καθορίζει τόσο τον αριθμό των bit που μπορούν να κωδικοποιηθούν όσο και τον αριθμό των chips. Αξίζει να επισημανθεί η διαφορά μεταξύ του chirp και του chip, από τη στιγμή που συχνά συγχέονται. Το chirp είναι ο παλμός από την ελάχιστη στη μέγιστη συχνότητα (up-chirp) ή το αντίστροφο (down-chirp), ενώ το chip είναι η κωδικοποίηση του συμβόλου με δεδομένο παράγοντα εξάπλωσης.

1.2.11 Διόρθωση σφαλμάτων και δείκτης κωδικοποίησης

Η διαδικασία διόρθωσης σφαλμάτων ή Forward Error Correction (FEC) στο LoRa προβλέπει την προσθήκη επιπλέον bits στο μεταδιδόμενο πακέτο. Αυτά τα πλεονάζοντα bit βοηθούν στην ανάκτηση των δεδομένων όταν εκείνα είναι διεφθαρμένα από παρεμβολές. Όσο περισσότερα επιπλέον bits προστίθενται στο μεταδιδόμενο σήμα, τόσο ευκολότερη γίνεται η διαδικασία διόρθωσης σε περίπτωση διεφθαρμένου αρχείου. Από την άλλη πλευρά, με την προσθήκη των bit διόρθωσης αυξάνεται ο όγκος του μεταδιδόμενου σήματος με συνέπεια την ανάγκη για περισσότερη ενέργεια για την πραγματοποίηση της αποστολής. Τη στιγμή που οι τελικοί κόμβοι συνήθως τροφοδοτούνται με μπαταρία είναι σημαντικό να ληφθεί υπόψη το μειονέκτημα αυτό και να βρεθεί η χρυσή τομή των bit διόρθωσης ανάλογα την εφαρμογή.

Ο δείκτης κωδικοποίησης (coding rate) αναφέρεται στην αναλογία των bit που περιέχουν πληροφορία σε σχέση με τα συνολικά bit (δηλαδή με την προσθήκη των bit διόρθωσης). Στη διαμόρφωση LoRa επιτρέπονται οι ρυθμοί κωδικοποίησης $4/5$, $4/6$, $4/7$ ή $4/8$ σύμφωνα με τη φόρμουλα $CR = 4/(4 + CR)$, για CR = 1, 2, 3 ή 4.

Coding Rate (CR)	$CR = 4/(4 + CR)$
1	$4/5$
2	$4/6$
3	$4/7$
4	$4/8$

1.2.12 Ρυθμός δεδομένων, συμβόλων και chip

Ο ρυθμός των chips ισοδυναμεί με το εύρος ζώνης (bandwidth) σε Hertz, δηλαδή τον αριθμό των παλμών ανά δευτερόλεπτο:

$$BW = R_c = chip\ rate\ (chips/sec)$$

Για παράδειγμα, για εύρος ζώνης 125kHz ($BW = R_c = 125{,}000\ chips/s$) και spreading factor 9 έχουμε: Κάθε σύμβολο αποτελείται από 9 bit πληροφορίας, το οποίο ισοδυναμεί με $2^{SF} = 2^9 = 512\ chips$. Επομένως, μπορούμε να υπολογίσουμε το ρυθμό μετάδοσης συμβόλων ως $\frac{125{,}000}{512} \cong 244$ σύμβολα ανά δευτερόλεπτο. Πιο γενικά, ο ρυθμός μετάδοσης συμβόλων υπολογίζεται ως εξής:

$$R_s\ (symbols/sec\ ) = \frac{BW}{2^{SF}} = \frac{R_c}{2^{SF}}$$

Όπου το Bandwidth (BW) σε Hz και το spreading factor (SF) να παίρνει ακέραιες τιμές στο εύρος 7-12. Από τις παραπάνω εξισώσεις είναι προφανές πως η μετάδοση των chip είναι πάντοτε μεγαλύτερη από αυτή των συμβόλων ($R_c > R_s$).

Για τον υπολογισμό του ρυθμού μετάδοσης δεδομένων (ή bit), έχουμε το παρακάτω:

$$R_b\ (bits/sec) = SF \times \frac{BW}{2^{SF}} \times \frac{4}{(4 + CR)}$$

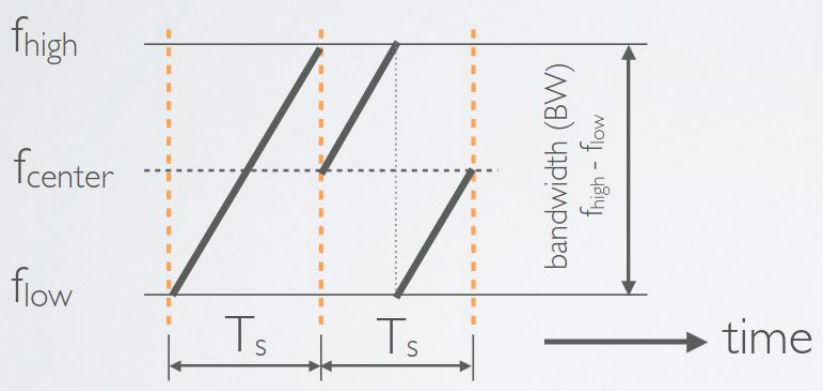
Όπου το Bandwidth (BW) σε Hz, το spreading factor (SF) παίρνει ακέραιες τιμές στο διάστημα 7-12 και το coding rate (CR) παίρνει ακέραιες τιμές στο εύρος 1-4.

Τη διάρκεια του κάθε chip την υπολογίζουμε ως εξής:

$$T_c\ (\text{sec}) = 1/BW$$

Ενώ την διάρκεια συμβόλου αν πολλαπλασιάσουμε την διάρκεια του chip επί τον αριθμό των chips που αντιστοιχούν σε ένα σύμβολο, συνεπώς:

$$T_s\ (\text{sec}) = 2^{SF}/BW$$



Εικόνα 24: Διάρκεια συμβόλου

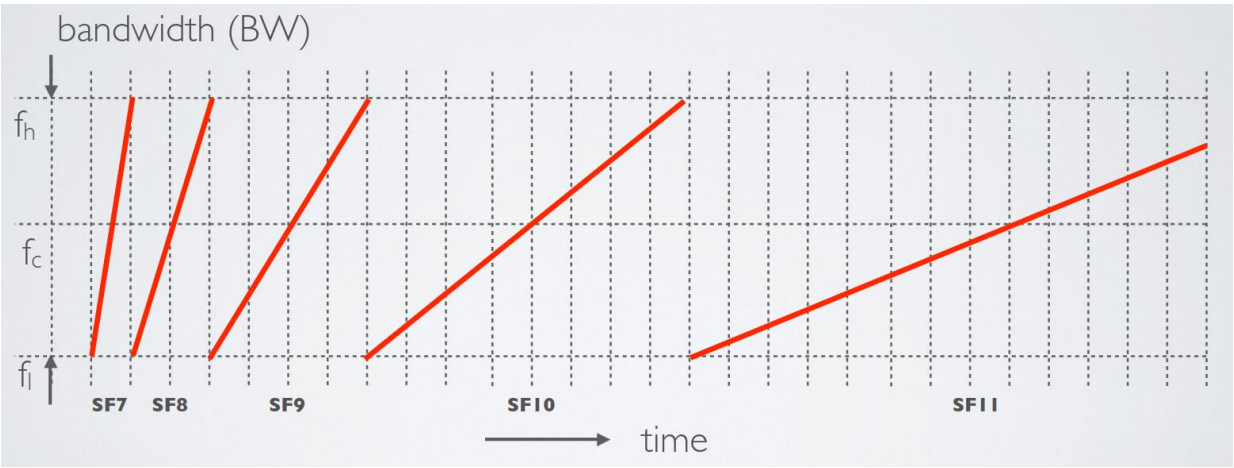
Γενικά ισχύουν οι παρακάτω παρατηρήσεις:

- Με την άυξηση του εύρους ζώνης, ο ρυθμός μετάδοσης των δεδομένων αυξάνεται, ενώ η διάρκεια του συμβόλου μειώνεται
- Με την αύξηση του spreading factor, ο ρυθμός μετάδοσης των δεδομένων μειώνεται, ενώ η διάρκεια του συμβόλου αυξάνεται

Αν το spreading factor αυξηθεί κατά 1, τότε:

- Η διάρκεια συμβόλου διπλασιάζεται
- Ο ρυθμός μετάδοσης bit σχεδόν υποδιπλασιάζεται
- Ο χρόνος μετάδοσης του μηνύματος ή Time on Air (ToA), αυξάνεται

Για ένα μήνυμα 10 byte δεδομένων, με εύρος ζώνης 125kHz, όταν το spreading factor είναι 7, τότε ο χρόνος μετάδοσης του μηνύματος είναι κοντά στα 40ms, ενώ για spreading factor 12 είναι γύρω στα 990ms. Στη διαμόρφωση LoRa, υψηλότερο spreading factor χρησιμοποιείται όταν το σήμα είναι αδύναμο ή υπάρχουν πολλές παρεμβολές. Όσο πιο μακριά βρίσκεται το gateway από το end device, τόσο πιο αδύναμο γίνεται το σήμα και επομένως τόσο μεγαλύτερο spreading factor απαιτείται.



Εικόνα 25: Χρόνος μετάδοσης και spreading factor

1.2.13 Όριο SNR και ευαισθησία δέκτη

Για κάθε spreading factor, υπάρχει ένα όριο SNR, το οποίο αν ξεπεραστεί, ο δέκτης δε θα μπορέσει να αποδιαμορφώσει το σήμα. Όσο μεγαλύτερη είναι η απόσταση μεταξύ πομπού και δέκτη τόσο μεγαλύτερο spreading factor απαιτείται για την επιτυχή αποδιαμόρφωση του σήματος. Στο παρακάτω πίνακα απεικονίζονται τα spreading factors μαζί με τα αντίστοιχα SNR limits:

Spreading Factor	Chips per symbol	SNR limit (dB)
7	128	-7.5
8	256	-10
9	512	-12.5
10	1024	-15
11	2048	-17.5
12	4096	-20

Παρατηρείται πως με την αύξηση του spreading factor κατά 1, το όριο SNR αλλάζει κατά -2.5 dB.

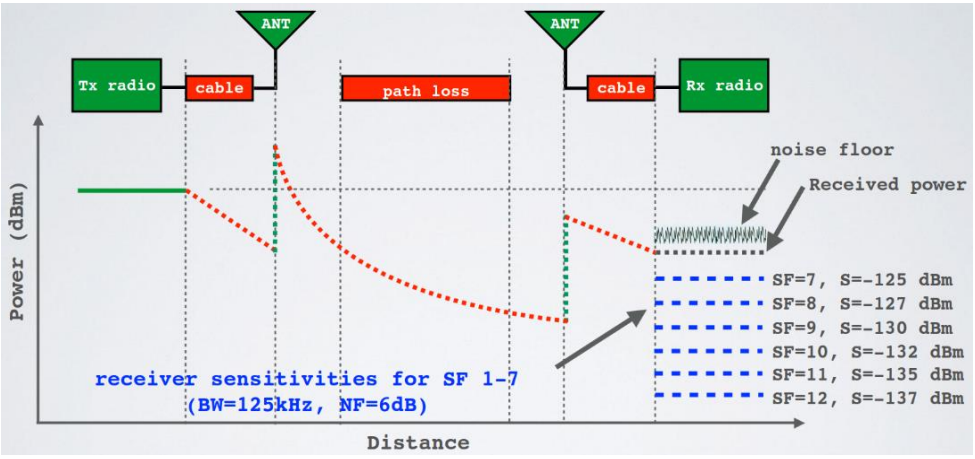
Για τον υπολογισμό της ευαισθησίας του δέκτη ισχύει η παρακάτω σχέση:

$$S = -174 + 10 \times log_{10}(BW) + NF + SNR_{limit}$$

Όπου S η ευαισθησία του δέκτη σε dBm, bandwidth (BW) σε Hz, Noise Figure (NF) του δέκτη σε dB και Signal-to-Noise limit (SNR_{limit}) σε dB. Το Noise Figure (NF) του δέκτη είναι προκαθορισμένο για δεδομένο hardware. Για τους LoRa πομποδέκτες SX1272 και SX1276, εκ των οποίων ο δεύτερος χρησιμοποιήθηκε για το πρακτικό μέρος της διπλωματικής εργασίας, ο παράγοντας NF είναι 6 dB.

Με τη χρήση της παραπάνω σχέσης και με δεδομένα εύρος ζώνης 125kHz και NF = 6 dB, υπολογίζουμε:

Spreading Factor	SNR_{limit} (dB)	S(ensitivity) (dBm)
7	-7.5	-125
8	-10	-127
9	-12.5	-130
10	-15	-132
11	-17.5	-135
12	-20	-137



Εικόνα 26: Spreading Factor και Όριο SNR

1.2.14 Δομή πακέτου και χρόνος μετάδοσης

Το πακέτο LoRa αποτελείται από 3 στοιχεία: Τον πρόλογο (preamble), την επικεφαλίδα (header) και το ωφέλιμο φορτίο (payload). Η επικεφαλίδα μπορεί να παραληφθεί, μειώνοντας τον χρόνο μετάδοσης, αν το μήκος του ωφέλιμου φορτίου, ο δείκτης κωδικοποίησης (coding rate) και η ύπαρξη του CRC (Cycle Redundancy Check) έχουν ρυθμιστεί τόσο στο πομπό όσο και στο δέκτη.

Εικόνα 27: Δομή πακέτου LoRa

- Preamble: Το χρησιμοποιεί ο δέκτης για να εντοπίσει την αρχή του πακέτου και αποτελείται συνήθως από 12.25 σύμβολα (8 για το EU868 + 4.25 που προσθέτει ο πομπός)
- Header: Παρέχει πληροφορίες για το μήκος του ωφέλιμου φορτίου σε bytes, τον δείκτη κωδικοποίησης και την ενδεχόμενη παρουσία ενός 16 bits ελέγχου του payload
- Payload: Περιέχει τα πραγματικά δεδομένα κωδικοποιημένα
- Payload CRC: Έλεγχος για την ορθή μεταφορά του ωφέλιμου φορτίου του πακέτου

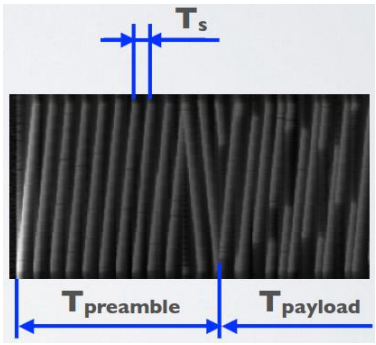
Όπως έχει προαναφερθεί, ο χρόνος που απαιτείται προκειμένου να φτάσει το σήμα από τον πομπό στο δέκτη, ονομάζεται χρόνος στον αέρα ή Time on Air (ToA). Ο χρόνος αυτός υπολογίζεται ως εξής:

$$ToA = T_{packet} = T_{preamble} + T_{payload}$$

Ο χρόνος μετάδοσης του προλόγου ($T_{preamble}$) προκύπτει:

$$T_{preamble} = (n_{preamble} + 4.25) \times T_s$$

Όπου το $n_{preamble}$ ισούται με 8 σύμβολα στο EU868 και T_s η διάρκεια του συμβόλου.



Εικόνα 28: Time on Air

Ο χρόνος μετάδοσης του payload μπορεί να βρεθεί με το παρακάτω:

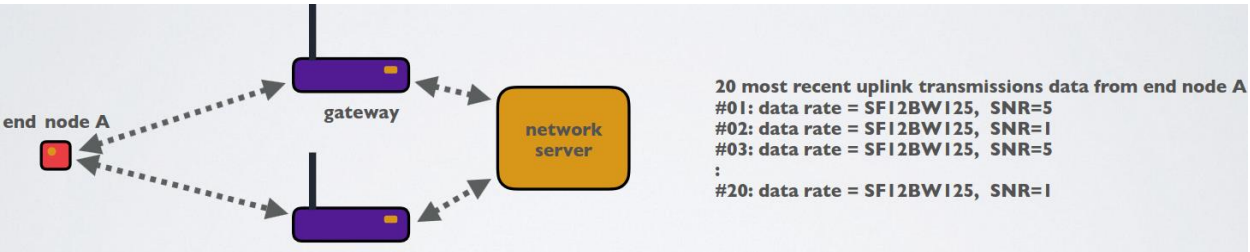
$$T_{payload} = T_s \times (8 + \max(ceil\left(\frac{8PL - 4SF + 28 + 16CRC - 20H}{4(SF - 2DE)}\right)(CR + 4), 0)$$

- T_s η διάρκεια συμβόλου σε δευτερόλεπτα (sec)
- PL (Payload) το ωφέλιμο φορτίο σε bytes
- SF (Spreading Factor) με τιμές 7 έως 12
- CRC (1 για ενεργοποιημένο και 0 για απενεργοποιημένο). Η προκαθορισμένη τιμή στο LoRaWAN είναι CRC=1
- H (Header), 1 για απενεργοποιημένο και 0 αντίθετα. Συνήθως στο LoRaWAN είναι H=0
- DE (LowDataRateOptimize), 1 για ενεργοποιημένο και 0 αντίθετα
- CR (Coding Rate). Η προκαθορισμένη τιμή στο LoRaWAN είναι CR=1

Για υψηλά spreading factors, όπου αυξάνεται ο χρόνος μετάδοσης του πακέτου, η επιλογή Low Data Rate Optimization μπορεί να ενεργοποιηθεί με σκοπό να αυξηθεί η ανθεκτικότητα της μετάδοσης από τις παρεμβολές άλλων σημάτων. Είναι σημαντικό να σημειωθεί πως για να λειτουργήσει η επιλογή αυτή, θα πρέπει να έχει ρυθμιστεί στον πομπό αλλά και στον δέκτη, καθώς και η διάρκεια συμβόλου να ξεπερνά τα 16ms. Επιπλέον, η επιλογή αυτή είναι διαθέσιμη για εύρος ζώνης 125kHz και spreading factor μεγαλύτερο ή ίσο του 11.

Το πρωτόκολλο LoRaWAN προβλέπει έναν μηχανισμό προσαρμογής των παραμέτρων αποστολής uplink των τελικών συσκευών. Πιο συγκεκριμένα, ο μηχανισμός ADR (Adaptive Data Rate), ελέγχει τον παράγοντα εξάπλωσης, το εύρος ζώνης και την ισχύ μετάδοσης. Μια συσκευή end node μπορεί να ζητήσει την ενεργοποίηση ή απενεργοποίηση της σημαίας ADR, η οποία όταν είναι πάνω επιτρέπει στον εξυπηρετητή δικτύου (network server) να ρυθμίσει τις παραμέτρους αποστολής του τελικού κόμβου.

Ο εξυπηρετητής δικτύου συλλέγει τα 20 πιο πρόσφατα uplinks από έναν τελικό κόμβο και ελέγχει τις τιμές RSSI και SNR, ενώ παράλληλα εκτιμά την πιθανότητα απώλειας πακέτου.



Εικόνα 29: Adaptive Data Rate

Από αυτά τα μηνύματα uplink, ο network server παίρνει τη μέγιστη τιμή SNR (η οποία ορίζεται ως $SNR_{measured}$) και τον αντίστοιχο ρυθμό μετάδοσης δεδομένων και έπειτα υπολογίζει το περιθώριο margin με τη χρήση του παρακάτω τύπου:

$$margin = SNR_{measured} - SNR_{limit} - margin_{default}$$

Για παράδειγμα, όταν έχουμε data rate SF12BW125, $SNR_{measured} = 5\ dB$, $SNR_{limit} = -20dB$, $margin_{default} = 10\ dB$, τότε υπολογίζεται το margin:

$$margin = 5 - (-20) - 10 = 15\ dB$$

Το περιθώριο 15 dB είναι αρκετά υψηλό, το οποίο σημαίνει πως ο πομπός καταναλώνει περισσότερη ενέργεια από ότι χρειάζεται για να στείλει το μήνυμα uplink με αποτέλεσμα την γρηγορότερη εξάντληση της μπαταρίας. Ο διακομιστής δικτύου υπολογίζει το περιθώριο και με άλλα spreading factors προκειμένου να μειώσει το margin. Επομένως για το προηγούμενο παράδειγμα και για spreading factor ίσο με 7 έχουμε το νέο περιθώριο:

$$margin = 5 - (-7.5) - 10 = 2.5\ dB$$

Το περιθώριο άλλαξε από 15dB σε 2.5dB, το οποίο υποδηλώνει μια βελτίωση σχετικά με την κατανάλωση ενέργειας. Ακόμη μπορεί να βελτιστοποιηθεί περαιτέρω με τη ρύθμιση της ισχύος αποστολής. Με την αποστολή του κατάλληλου μηνύματος downlink, ο τελικός κόμβος θα λάβει τις νέες ρυθμίσεις και θα ξεκινήσει να στέλνει τα μηνύματα καταναλώνοντας λιγότερη ενέργεια.

Κεφάλαιο 2: Τεχνολογικό Υπόβαθρο

Στο κεφάλαιο αυτό, αναλύονται όλα τα βήματα που έγιναν προκειμένου να στηθεί το δίκτυο LoRa καθώς και οι βασικές τεχνολογικές έννοιες που αναμένεται να γνωρίζει ο αναγνώστης προτού μεταβεί στα επόμενα κεφάλαια. Πιο συγκεκριμένα, γίνεται λόγος για το στήσιμο του διακομιστή δικτύου (network server), την τοποθέτηση και ρύθμιση των πυλών (gateways), τις τεχνολογικές μονάδες που χρησιμοποιήθηκαν για τη διαμόρφωση του σήματος και άλλα.

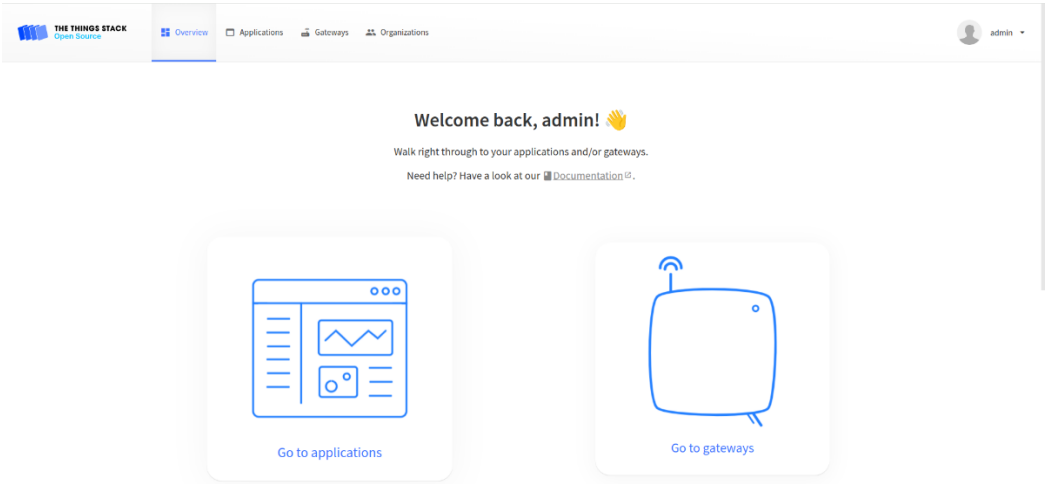
2.1 Network Server

2.1.1 The Things Stack

Το The Things Stack (TTN) είναι μια πλατφόρμα διαχείρισης δικτύων LoRaWAN, η οποία παρέχει λειτουργικότητες για τη δημιουργία, τη διαμόρφωση και τη διαχείριση τέτοιων δικτύων. Η πλατφόρμα αυτή, υποστηρίζει την ανάπτυξη δημόσιων αλλά και ιδιωτικών δικτύων LoRaWAN και παρέχει εργαλεία για τη διαχείριση συσκευών, τη δρομολόγηση μηνυμάτων, την παρακολούθηση δικτύου και άλλες λειτουργικότητες που είναι απαραίτητες για αυτά τα συστήματα. Στη παρούσα διπλωματική εργασία, στήθηκε ένα ιδιωτικό δίκτυο με την αξιοποίηση του The Things Stack Open Source.

Πιο αναλυτικά, σε σέρβερ του Πολυτεχνείου με λειτουργικό σύστημα Linux Ubuntu 22.04.4 στήθηκε ένα docker container με image του The Things Network από το docker hub. Περισσότερες πληροφορίες για το image μπορούν να βρεθούν στο [10], ενώ η έκδοση που χρησιμοποιήθηκε είναι η 3.27.1. Επιπλέον, με τη χρήση του caddy (open-source web server) γίνεται εσωτερική δρομολόγηση των http αιτήσεων μέσω reverse proxy και έτσι μπορούμε να έχουμε εκτεθειμένη μόνο την θύρα 80.

Ο TTN network server παρέχει διάφορα χρήσιμα services, όπως το MQTT στη θύρα 1883. Επίσης από το περιβάλλον χρήστη (UI), μπορούν να εγγραφούν gateways, να δημιουργηθούν εφαρμογές και να συνδεθούν end devices. Το περιβάλλον παρέχει ακόμα δυνατότητα δημιουργίας χρηστών με ρύθμιση δυνατοτήτων πάνω στον διακομιστή δικτύου.



Εικόνα 30: Περιβάλλον χρήστη TTN server

2.2 Gateway

2.2.1 Router

Τα routers που χρησιμοποιήθηκαν για τις πύλες είναι MikroTik Knot LR8 kit. Υποστηρίζουν πολλά πρωτόκολλα δικτύου, όπως: 2.4 GHz Wi-Fi, Bluetooth, LoRa και Ethernet με PoE-in και PoE-out. Τα gateways συνδέονται στο διαδίκτυο μέσω ethernet για να μπορέσουν να επικοινωνήσουν με τον network server. Για τη ρύθμιση του δρομολογητή, συνδεόμαστε μέσω Wi-Fi στο τοπικό του δίκτυο, το οποίο ανοίγει αυτόματα κατά τη λειτουργία του. Από το πάνελ ελέγχου μπορούν να γίνουν διάφορες ρυθμίσεις τόσο για το LoRa όσο και για τα υπόλοιπα πρωτόκολλα.

Αρχικά, για τη σύνδεση του router στον network server του LoRa, απαιτείται η δήλωση του server στο πάνελ ελέγχου. Όπως φαίνεται παρακάτω, ο εξυπηρετητής μας (lorawan-motion) δηλώνεται μαζί με τα αντίστοιχα up ports και down ports:

6 items

		Name	Address	Up port	Down port
-		TTN-EU	eu.mikrotik.thethings.industries	1700	1700
-		TTN-US	us.mikrotik.thethings.industries	1700	1700
-		TTS Cloud (au1)	au1.cloud.thethings.industries	1700	1700
-		TTS Cloud (eu1)	eu1.cloud.thethings.industries	1700	1700
-		TTS Cloud (nam1)	nam1.cloud.thethings.industries	1700	1700
-		lorawan-motion	lorawan-motion.dslab.ece.ntua.(	1700	1700

Εικόνα 31: Συνδεδεμένοι Servers

Με την επιτυχή σύνδεση στον διακομιστή, ο δρομολογητής μπορεί να προωθεί τα μηνύματα που λαμβάνει από τα end devices μέσω του πρωτοκόλλου LoRa στον network server και ο τελικός χρήστης να έχει πρόσβαση σε αυτά μέσω http στη θύρα 80.

Αξίζει να σημειωθεί, πως μέσω του μενού του router μπορούμε να δούμε τα κανάλια του LoRa που αναλύθηκαν με λεπτομέρεια στο κεφάλαιο 1 και αφορούν τη ζώνη EU868 όπως αναμένεται:

		#	Device	Type	Freq. (MHz)	Bandwidth	
D		0	mikrotik-x8j8	MSF	868.100	125 kHz	
D		1	mikrotik-x8j8	MSF	868.300	125 kHz	
D		2	mikrotik-x8j8	MSF	868.500	125 kHz	
D		3	mikrotik-x8j8	MSF	867.100	125 kHz	
D		4	mikrotik-x8j8	MSF	867.300	125 kHz	
D		5	mikrotik-x8j8	MSF	867.500	125 kHz	
D		6	mikrotik-x8j8	MSF	867.700	125 kHz	
D		7	mikrotik-x8j8	MSF	867.900	125 kHz	
D		8	mikrotik-x8j8	LoRa	868.300	250 kHz	
D		9	mikrotik-x8j8	FSK	868.800	125 kHz	

Εικόνα 32: Κανάλια LoRa

2.2.2 Πανκατευθυντική Κεραία

Η κεραία που επιλέχθηκε για τις πύλες είναι η 868 omni κεραία της MikroTik με ενίσχυση 6.5 dBi. Η κεραία είναι πανκατευθυντική, το οποίο σημαίνει πως δε χρειάζεται να τοποθετηθεί προσανατολισμένη προς τις συσκευές πομπούς, αλλά καλύπτει ομοιόμορφα την εμβέλεια γύρω από την περιοχή τοποθέτησής της. Όπως εξηγήθηκε στο κεφάλαιο 1, η κεραία οφείλει να τοποθετηθεί κατακόρυφα προκειμένου να καλύψει τη μέγιστη δυνατή εμβέλεια και ταυτόχρονα να βρίσκεται κατά προτίμηση σε υψηλό και εξωτερικό σημείο.

Είναι σημαντικό να τοποθετηθεί όσο το δυνατόν μικρότερο σε μήκος καλώδιο από το router στην κεραία καθώς το σήμα φθίνει γρήγορα μέσω του καλωδίου και επίσης να χρησιμοποιηθεί ο ελάχιστος αριθμός μετασχηματιστών.

Εικόνα 33: Gateway (router and antenna)

2.3 Υλικό τελικού κόμβου

2.3.1 Διαμορφωτής LoRa SX1276

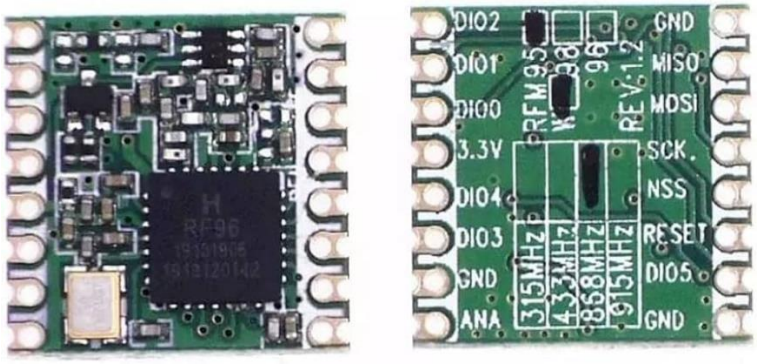
Το τσιπ που χρησιμοποιείται στους πομπούς για τη διαμόρφωση του LoRa σήματος είναι RFM95W της εταιρείας HopeRF. Αυτό ενσωματώνει το τσιπ SX1276, το οποίο χρησιμοποιείται ευρέως στα συστήματα LoRa. Σύμφωνα με το φύλλο δεδομένων του, καταγράφονται τα παρακάτω χαρακτηριστικά:

Chip	Frequency range MHz	RF output power dBm	Max Link Budget dBm	Max Rx Sensitivity dBm	Remark
SX1276	137-1020	20	168	-148	Support all major sub GHz ISM bands

Ενώ για την υλοποίηση πάνω στο RFM95W έχουμε:

Chip	Frequency range MHz	RF output power dBm	Max Link Budget dBm	Max Rx Sensitivity dBm	Compatible With	Chip Marking	Available in Frequency MHz
RFM95W	137-1020	20	168	-148	SX1276	RF96	868,915

Το τσιπ είναι συμβατό με το SX1276 και λειτουργεί στη ζώνη EU863-870 ISM.



Εικόνα 34: RFM95W

2.3.2 Μικροελεγχτής/ Μικροεπεξεργαστής

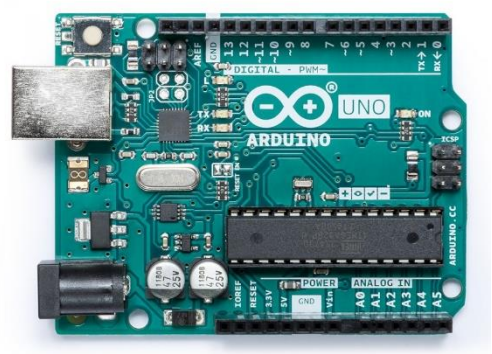
Στην πρώτη έκδοση του τελικού κόμβου χρησιμοποιήθηκε για μικροελεγχτής ένα Arduino Uno, ενώ για τη δεύτερη έκδοση και για λόγους που θα αναλυθούν παρακάτω αξιοποιήθηκε ένας μικροεπεξεργαστής esp32 NodeMCU-32S.

a. Arduino Uno

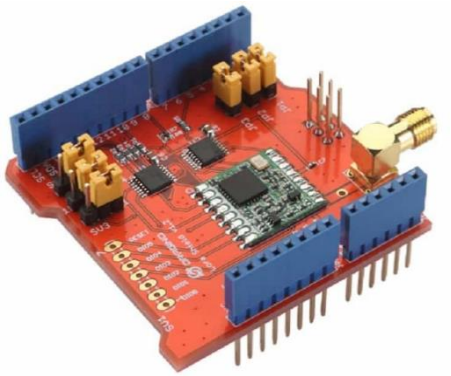
Το Arduino Uno είναι ένα από τα πιο δημοφιλή μοντέλα μικροελεγκτών (microcontrollers) από την οικογένεια των Arduino. Αποτελεί έναν μικροελεγκτής ανοιχτού κώδικα (open-source), ο οποίος χρησιμοποιείται ευρέως για πρωτότυπες κατασκευές, πειραματισμούς και εφαρμογές εκπαίδευσης.

Στον πυρήνα του περιλαμβάνει έναν μικροελεγκτή ATmega328P, ο οποίος διαθέτει ψηφιακές και αναλογικές εισόδους/εξόδους, καθώς και διάφορα άλλα χαρακτηριστικά. Περιλαμβάνει μια σειριακή θύρα USB για τη σύνδεση με υπολογιστή, καθιστώντας το εύκολο στη χρήση και τον προγραμματισμό.

Επιλέχθηκε αρχικά για την ευκολία στον προγραμματισμό του, ενώ παράλληλα σημαντικό ρόλο έπαιξε η διαθεσιμότητα έτοιμων shields με ενσωματωμένο το RFM95W. Πιο συγκεκριμένα, έγινε χρήση του Dragino LoRa shield για 868MHz, το οποίο απεικονίζεται παρακάτω. Η συσκευή αυτή “κουμπώνει” πάνω στο Arduino Uno και διευκολύνει την διαδικασία προγραμματισμού του SX1276, από τη στιγμή που δεν απαιτούνται ηλεκτροκολλήσεις ή συνδέσεις για την λειτουργία του. Το Dragino Shield έρχεται μαζί με κεραία στα +2 dBi, η οποία συνδέεται στην αντίστοιχη υποδοχή.



Εικόνα 35: Arduino Uno



Εικόνα 36: Dragino shield

b. NodeMCU – Esp32

Το ESP32 NodeMCU είναι μια δημοφιλής πλακέτα ανάπτυξης που βασίζεται στον μικροελεγκτή ESP32 από την Espressif Systems. Αποτελεί μια πλακέτα που χρησιμοποιείται ευρέως για πρωτότυπες κατασκευές, έρευνα και εφαρμογές IoT (Internet of Things).

Το ESP32 NodeMCU περιλαμβάνει τον μικροελεγκτή ESP32 με ενσωματωμένο Wi-Fi και Bluetooth, που το καθιστούν ιδανικό για ασύρματες εφαρμογές δικτύωσης και επικοινωνίας. Έχει επίσης ψηφιακές και αναλογικές εισόδους/εξόδους, καθιστώντας το κατάλληλο για ποικίλες εφαρμογές αισθητήρων και ελέγχου.

Προτιμήθηκε για τη δεύτερη έκδοση του end device έναντι του Arduino, από τη στιγμή που διαθέτει περισσότερες δυνατότητες, όπως η λειτουργία βαθύ ύπνου και η δυνατότητα διακοπής (interrupt) από όλα τα pins του τσιπ. Επιπλέον, σε αντίθεση με το Arduino Uno, το οποίο λειτουργεί στα 5V, το esp32 λειτουργεί στα 3.3V, το οποίο το καθιστά πιο αποδοτικό ενεργειακά. Από την άλλη πλευρά, η σύνδεση του RFM95W απαιτούσε περισσότερη διαδικασία, αφού έπρεπε να πραγματοποιηθούν κολλήσεις σε αντίθεση με την περίπτωση του dragino shield.

Και στις δύο υλοποιήσεις, ο προγραμματισμός έγινε στο περιβάλλον του Arduino IDE και ο κώδικας είναι σε γλώσσα C. Οι υλοποιήσεις αυτές θα αναλυθούν παρακάτω στο κεφάλαιο αρχιτεκτονικής και υλοποίησης της εφαρμογής.

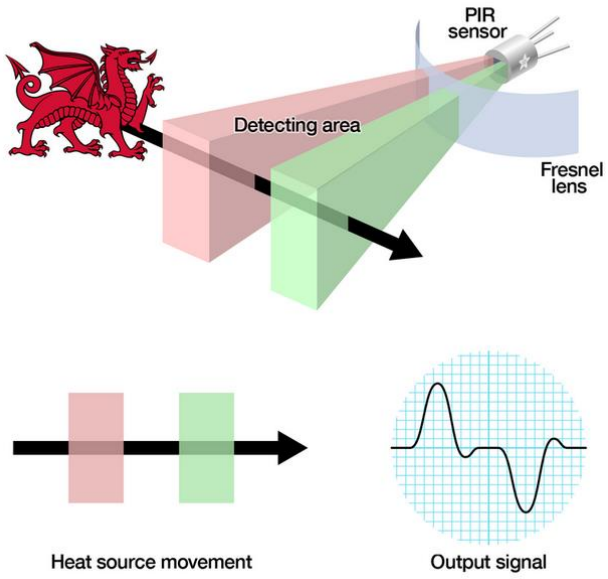
Εικόνα 37: NodeMCU - Esp32

2.3.3 Αισθητήρες

Οι αισθητήρες χωρίζονται σε δύο βασικές κατηγορίες: Τους ενεργητικούς (active) και τους παθητικούς (passive). Οι ενεργητικοί αισθητήρες χρησιμοποιούν ενέργεια για να δημιουργήσουν ένα σήμα εξόδου. Αυτή η ενέργεια μπορεί να παρέχεται από εσωτερικές πηγές, όπως μπαταρίες ή τροφοδοτικά κυκλώματα. Παραδείγματα τέτοιων αισθητήρων είναι επιταχυνσιόμετρα, γυροσκόπια, μέτρησης ποιότητας αέρα κ.α. Από την άλλη πλευρά, οι παθητικοί αισθητήρες δεν απαιτούν εξωτερική παροχή ενέργειας, αφού λαμβάνουν την ενέργεια από το περιβάλλον. Οι αισθητήρες αυτοί μετατρέπουν τη μεταβολή σε μια φυσική ποσότητα (όπως φως, θερμότητα ή ήχος) σε ένα ηλεκτρικό σήμα χωρίς ενεργή συμμετοχή. Παραδείγματα παθητικών αισθητήρων περιλαμβάνουν αισθητήρες κίνησης, ηλεκτρομαγνητικών πεδίων και υπέρυθρων ακτινοβολιών.

a. PIR Sensors

Η βασική λειτουργία του PIR αισθητήρα, ο οποίος ανήκει στην κατηγορία των παθητικών αισθητήρων φαίνεται στην παρακάτω εικόνα:



Εικόνα 38: Λειτουργία Αισθητήρα PIR

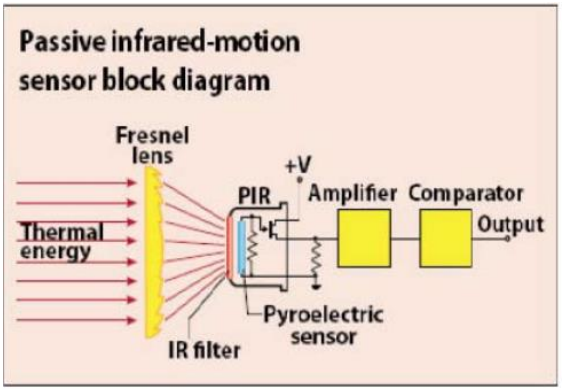
Ο παθητικός αισθητήρας υπέρυθρων (PIR - Passive Infrared Sensor) έχει δύο υποδοχές, η κάθε μία από τις οποίες είναι κατασκευασμένη από ένα ειδικό υλικό που είναι ευαίσθητο στο υπέρυθρο φως. Ο φακός που χρησιμοποιείται εδώ δεν παίζει πολύ μεγάλο ρόλο και γι 'αυτό βλέπουμε ότι οι δύο υποδοχές μπορούν να "δουν" πέρα από μια απόσταση (αναλόγως την ευαισθησία του αισθητήρα). Όταν ο αισθητήρας είναι αδρανής, και οι δύο υποδοχές ανιχνεύουν το ίδιο ποσό υπέρυθρου φωτός, το οποίο εκπέμπεται από το δωμάτιο, τους τοίχους ή το εξωτερικό περιβάλλον. Όταν ένα ζεστό σώμα, όπως ένας άνθρωπος ή ένα ζώο, περνά από κοντά, πρώτα αλληλοεπιδρά με μισό από τον PIR αισθητήρα, προκαλώντας μια θετική διαφορική μεταβολή μεταξύ των δύο μισών. Όταν το ζεστό σώμα

απομακρύνεται από την περιοχή ανίχνευσης, συμβαίνει το αντίστροφο, συνεπώς ο αισθητήρας δημιουργεί αρνητική διαφορική μεταβολή. Αυτές οι μεταβολές μεταφράζονται σε παλμούς και είναι τελικά αυτά που ανιχνεύονται.

Ο αισθητήρας υπέρυθρων (IR) φιλοξενείται σε μια αεροστεγή μεταλλική θήκη για να βελτιώσει την αντοχή στο θόρυβο, τη θερμοκρασία και την υγρασία. Υπάρχει ένα παράθυρο κατασκευασμένο από υλικό που διαπερνά την υπέρυθρη ακτινοβολία (συνήθως επικαλυμμένο με πυριτικό κυανίου) που προστατεύει το στοιχείο ανίχνευσης. Πίσω από το παράθυρο βρίσκονται οι δύο ισορροπημένοι αισθητήρες, οι οποίοι δημιουργούν τις διαφορικές μεταβολές που αναφέρθηκαν παραπάνω.

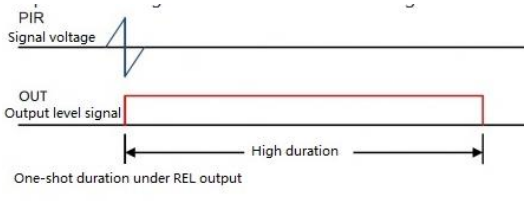
Συνήθως θέλουμε να έχουμε μια περιοχή ανίχνευσης που είναι μεγαλύτερη από αυτή που φαίνεται στην εικόνα 38. Για να το επιτευχθεί αυτό, γίνεται χρήση ενός απλού φακού, όπως αυτοί που βρίσκονται σε μια φωτογραφική μηχανή: οι φακοί συμπυκνώνουν μια μεγάλη περιοχή (όπως ένα τοπίο) σε μια μικρή (σε φιλμ ή ένα αισθητήρα CCD). Στη πράξη χρησιμοποιούνται φακοί Fresnel, οι οποίοι επιτυγχάνουν την αύξηση της περιοχής ανίχνευσης με μικρό κόστος.

Εικόνα 40: Αισθητήρας PIR με φακό Fresnel

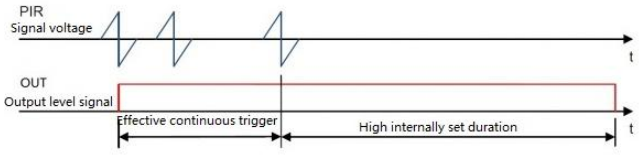


Εικόνα 39: Απλοποιημένο κύκλωμα PIR

Στο υλοποίηση των end devices έγινε χρήση ενός HC-SR501 PIR αισθητήρα, οποίος λειτουργεί στα 5V και είναι θετικής λογικής, δηλαδή όταν ανιχνεύει κίνηση δίνει λογικό 1 στην έξοδό του για μερικά δευτερόλεπτα (περίπου 2-3 δευτερόλεπτα). Επιπλέον, χρησιμοποιήθηκε ένας SEN0171-DFRobot με παρόμοια λειτουργία, που λειτουργεί στα 3.3V για περισσότερη εξοικονόμηση ενέργειας. Η απόσταση που ανιχνεύουν και οι δύο αισθητήρες είναι γύρω στα 7 μέτρα, ενώ ακολουθήθηκε η λογική της επαναπυροδότησης (retriggering), δηλαδή η επίτρεψη ανίχνευσης επιπλέον κινήσεων στο διάστημα που είναι ενεργοποιημένος ήδη ο αισθητήρας. Η λειτουργία αυτή φαίνεται και στην παρακάτω εικόνα:



Εικόνα 41: Χωρίς επαναπυροδότηση

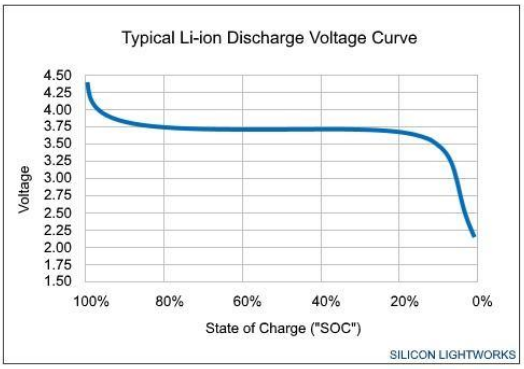


Εικόνα 42: Με απαναπυροδότηση

b. Μετρητής μπαταρίας

Οι μπαταρίες 3.7V λιθίου (li-ion) χρησιμοποιούνται σε πολλές εφαρμογές Internet of Things. Στο end device χρησιμοποιήθηκαν επαναφορτιζόμενες μπαταρίες 18650 με ονομαστική τάση 3.7V και χωρητικότητα 3350mAh. Πιο συγκεκριμένα, ο πομπός τροφοδοτείται από 2 τέτοιες μπαταρίες σε παράλληλη σύνδεση, προκειμένου να διατηρηθεί η τάση στα 3.7V αλλά να διπλασιαστεί η χωρητικότητα στα 6700mAh.

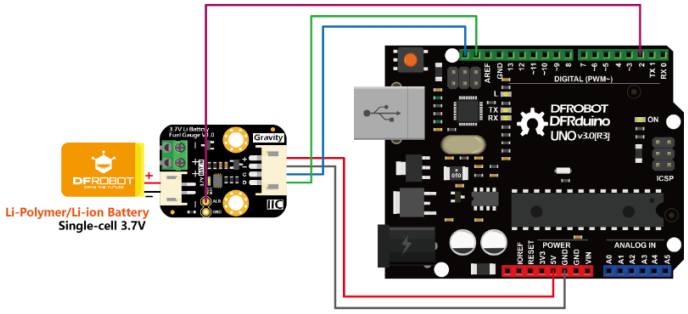
Ωστόσο, η ισχύς και η τάση της μπαταρίας λιθίου είναι εξαιρετικά μη γραμμική, επομένως η μπορούμε μόνο να κρίνουμε κατά προσέγγιση αν η μπαταρία είναι πλήρης ή σχεδόν άδεια ανάλογα με την τάση της. Με άλλα λόγια, η απλή λύση της μέτρησης τάσης της μπαταρίας με ένα διαιρέτη τάσης αποτελούμενο από δύο αντιστάσεις της τάξης των 10kΩ, μπορεί να μας δώσει ακριβής μέτρηση για την τάση της μπαταρίας, αλλά όχι για το πόσο ακόμα θα διαρκέσει μέχρι να χρειαστεί επαναφόρτιση. Επιπλέον, ένα τέτοιο κύκλωμα καταναλώνει και περισσότερο ρεύμα, άρα μειώνει το διάρκεια αυτονομίας του τελικού κόμβου.



Εικόνα 44: Πτώση τάσης Li-Ion μπαταρίας

Εικόνα 43: Μπαταρία Li-Ion

Έχοντας τα παραπάνω υπόψη, χρησιμοποιήθηκε στο end device ένας μετρητής μπαταρίας, ο Gravity 3.7V Lithium Battery Fuel Gauge. Αυτός ο μετρητής χρησιμοποιεί τη διεπαφή Gravity I2C και λειτουργεί σε εξαιρετικά χαμηλό ρεύμα, ενώ η παρακολούθηση της σχετικής κατάστασης φόρτισης (SOC) της μπαταρίας σε πραγματικό χρόνο γίνεται μέσω του κατοχυρωμένου αλγορίθμου Maxim. Με τον αλγόριθμο αυτό παίρνουμε ακριβές μετρήσεις για την τάση και την υπολειπόμενη ισχύ της μπαταρίας. Το κύκλωμα αυτό, ενσωματώνει επίσης τη λειτουργία ειδοποίησης χαμηλής ισχύος μπαταρίας. Όταν η ισχύς της μπαταρίας πέφτει κάτω από το καθορισμένο όριο, το pin ALR δημιουργεί μια πτώση τάσης (από 3.3V σε τάσης γείωσης) για να προκαλέσει την εξωτερική διακοπή του μικροελεγκτή.



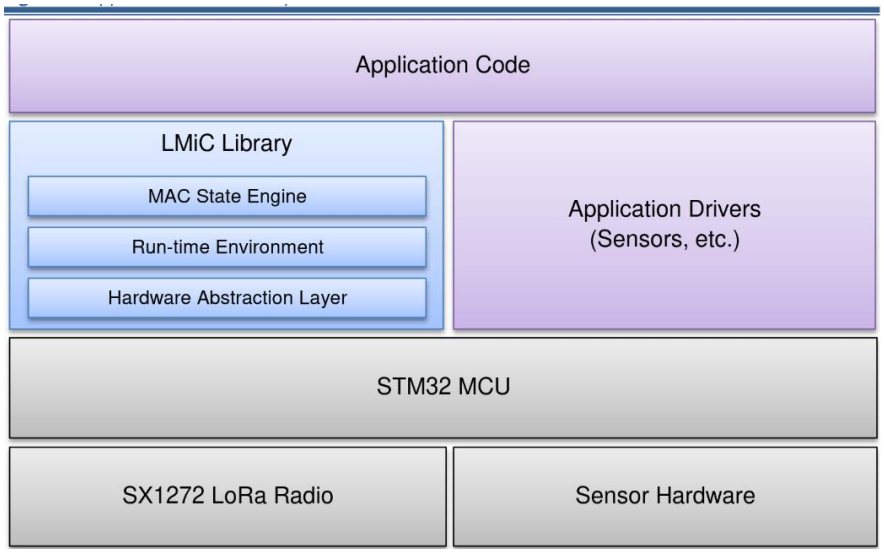
Εικόνα 45: Σύνδεση Battery Gauge

2.4 Λογισμικό τελικού κόμβου

2.4.1 Βιβλιοθήκη IBM LMIC

Για τον προγραμματισμό των end devices χρησιμοποιήθηκε βιβλιοθήκη IBM LMIC (LoRaWAN-MAC-in-C), ελαφρώς διαμορφωμένη για να τρέχει στο περιβάλλον του Arduino. Επιτρέπει την χρήση και τον προγραμματισμό του πομποδέκτη SX1276 και είναι συμβατή με το module HopeRF που χρησιμοποιήθηκε. Περισσότερες πληροφορίες για τη βιβλιοθήκη μπορούν να βρεθούν στο github repo [11].

Αρχικά την εισήγαγαν στο Arduino οι Matthijs Kooijman και Thomas Telkamp και ονομαζόταν Arduino LMIC, αλλά σήμερα έχει μετονομαστεί σε MCCI Arduino LMIC και υποστηρίζει πολλές περιοχές (regions).



Εικόνα 46: Στοιχεία end node

Η βιβλιοθήκη LMIC εμπεριέχει συναρτήσεις API, run-time συναρτήσεις, συναρτήσεις επανάκλησης (callback functions), καθώς και μια καθολική δομή δεδομένων. Με την επικεφαλίδα "**lmic.h**" στο Arduino script μπορούμε να αποκτήσουμε πρόσβαση σε όλα τα παραπάνω εργαλεία.

1) Προγραμματιστικό μοντέλο και διεργασίες

Πιο συγκεκριμένα, η βιβλιοθήκη παρέχει ένα απλό event-based προγραμματιστικό μοντέλο, όπου όλα τα γεγονότα του πρωτοκόλλου LoRa διαχειρίζονται από τη συνάρτηση **onEvent**(). Επιπλέον, προκειμένου να αποφευχθεί η ανάγκη για συγχρονισμό ή διακοπές, η βιβλιοθήκη έχει ενσωματωμένο ένα περιβάλλον run-time, το οποίο διαχειρίζεται τις ουρές και τις διεργασίες.

Το μοντέλο για όλον τον κώδικα της εφαρμογής τρέχει σε μια διεργασία (job), η οποία εκτελείται στο βασικό νήμα (thread) από τη συνάρτηση run-time **os_runloop**(). Αυτές οι διεργασίες, είναι γραμμένες ως κανονικές συναρτήσεις σε C και μπορούν να εκτελεστούν από άλλες run-time συναρτήσεις. Για την

διαχείριση των jobs, υπάρχει μια ειδική δομή, η δομή osjob_t, η οποία έχει περιέχει ένα αναγνωριστικό για την κάθε διεργασία και άλλες πληροφορίες. Είναι σημαντικό να σημειωθεί, πως οι διεργασίες πρέπει να διαρκούν λίγο χρόνο, προκειμένου να λειτουργεί αποδοτικά ο κώδικας. Επομένως, είναι καλή πρακτική να χρησιμοποιούνται μόνο για την ενημέρωση καταστάσεων ή τον προγραμματισμό άλλων δράσεων, οι οποίες με τη σειρά τους θα καλέσουν νέα jobs ή γεγονότα callback.

2) Κύριος βρόγχος (Main Event Loop)

Η εφαρμογή για να ξεκινήσει, αρκεί να αρχικοποιήσει το περιβάλλον run-time με τη συνάρτηση **os_init**() και έπειτα να εκτελέσει τον προγραμματιστή διεργασιών **os_runloop**(), ο οποίος δεν επιστρέφει. Προκειμένου να φορτώσουν οι δράσεις του πρωτοκόλλου και να παραχθούν γεγονότα, πρέπει να προηγηθεί το στήσιμο μιας αρχικής διεργασίας. Αυτό επιτυγχάνεται με τη συνάρτηση **os_setCallback**().

```
void main () {
    osjob_t initjob;

    // initialize run-time env
    os_init();
    // setup initial job
    os_setCallback(&initjob, initfunc);
    // execute scheduled jobs and events
    os_runloop();
    // (not reached)
}
```

Εικόνα 47: Κύριος βρόγχος

Η συνάρτηση **initfunc**() αρχικοποιεί την MAC διεύθυνση και σηματοδοτεί την πρόσβαση στο δίκτυο. Η συνάρτηση αυτή, επιστρέφει αμέσως και αμέσως μετά θα κληθεί η **onEvent**() callback συνάρτηση από τον προγραμματιστή για τα γεγονότα EV_JOINING (προσπάθεια σύνδεσης στο δίκτυο), EV_JOINED (σύνδεση στο δίκτυο) ή EV_JOIN_FAILED (αποτυχία σύνδεσης στο δίκτυο).

```
// initial job
static void initfunc (osjob_t* j) {
    // reset MAC state
    LMIC_reset();
    // start joining
    LMIC_startJoining();
    // init done - onEvent() callback will be invoked...
}
```

Εικόνα 48: Συνάρτηση initfunc()

3) The Συναρτήσεις run-time

Οι συναρτήσεις run-time όπως προαναφέρθηκε χρησιμοποιούνται για τον έλεγχο του περιβάλλοντος run-time. Αυτό περιλαμβάνει την αρχικοποίηση, τον προγραμματισμό και την εκτέλεση των διεργασιών. Οι συναρτήσεις αυτές παρουσιάζονται παρακάτω:

1. void **os_setCallback** (osjob_t* job, osjobcb_t cb)

   Προετοιμάζει μια διεργασία για να τρέξει άμεσα. Μπορεί να καλεστεί οποιαδήποτε στιγμή, ακόμη και από συνάρτηση διαχείρισης διακοπής (interrupt handler), όταν για παράδειγμα μια νέα τιμή αισθητήρα είναι διαθέσιμη.

2. void **os_setTimedCallback** (osjob_t* job, ostime_t time, osjobcb_t cb)

   Προετοιμάζει μια διεργασία για να τρέξει σε προγραμματισμένη χρονική στιγμή. Όμοια με την παραπάνω, μπορεί να εκτελεστεί οποιαδήποτε στιγμή.

3. void **os_clearCallback** (osjob_t* job)

   Ακυρώνει μια διεργασία. Μια προηγουμένως προγραμματισμένη διεργασία αφαιρείται από τον χρονοπρογραμματιστή και τις ουρές εκτέλεσης. Το job αναγνωρίζεται από τη μοναδική διεύθυνσή του στη δομή osjob_t. Αν η διεργασία δεν έχει προσδιοριστεί ή δεν έχει προγραμματιστεί πριν την κλήση της **clearCallback()**, τότε εκείνη δεν έχει καμία επίδραση.

4. void **os_runloop** ()

   Εκτελεί run-time jobs από τις ουρές εκτέλεσης. Αποτελεί τη βασική συνάρτηση διαχείρισης των γεγονότων. Δεν επιστρέφει κάποια τιμή και οφείλει να εκτελείται στο βασικό νήμα.

5. ostime_t **os_getTime** ()

   Επιστρέφει την απόλυτη ώρα συστήματος (absolute system time).

4) Συναρτήσεις Callback

Οι συναρτήσεις callback επιστρέφουν πληροφορίες σχετικά με την εφαρμογή και χρησιμοποιούνται για να στείλουν γεγονότα στην εφαρμογή. Αυτές περιλαμβάνουν τις παρακάτω:

- void **os_getDevEui** (u1_t* buf)

  Με την κλήση της συνάρτησης αυτής, δίνεται ως όρισμα ο κωδικός EUI της συσκευής και αντιγράφεται στον buffer. Ο κωδικός συσκευής EUI είναι μήκους 8 bytes και αποθηκεύεται σε μορφή little-endian, δηλαδή από το λιγότερο σημαντικό byte στο περισσότερο σημαντικό.

- void **os_getDevKey** (u1_t* buf)

  Με την κλήση αυτής της συνάρτησης, δίνεται ως όρισμα το κρυπτογραφημένο κλειδί εφαρμογής για τη συσκευή και αντιγράφεται στον buffer. Το κλειδί αυτό έχει μήκος 16 bytes.

- void **os_getArtEui** (u1_t* buf)

Με την κλήση της συνάρτησης αυτής, δίνεται ως όρισμα ο κωδικός EUI της εφαρμογής και αντιγράφεται στον buffer. Ο κωδικός εφαρμογής EUI είναι μήκους 8 bytes και αποθηκεύεται σε μορφή little-endian, δηλαδή από το λιγότερο σημαντικό byte στο περισσότερο σημαντικό.

- void **onEvent** (ev_t ev)

Η υλοποίηση αυτής της συνάρτησης προβλέπει την αντίδραση σε συγκεκριμένα γεγονότα και την πυροδότηση νέων δράσεων με βάση τα γεγονότα και την κατάσταση (state) της δομής LMIC. Η συνάρτηση διαχειρίζεται το γεγονός για το οποίο ενδιαφέραται και έπειτα αναθέτει στο LMIC API τις περεταίρω δράσεις. Παρακάτω παρουσιάζονται τα πιθανά γεγονότα:

- **EV_JOINING**

  Η συσκευή έχει αρχίσει τη προσπάθεια σύνδεσης στο δίκτυο

- **EV_JOINED**

  Η συσκευή έχει συνδεθεί με επιτυχία στο δίκτυο και είναι έτοιμη για ανταλλαγή δεδομένων

- **EV_JOIN_FAILED**

  Η συσκευή απέτυχε να συνδεθεί στο δίκτυο (αφού ξαναπροσπαθήσει)

- **EV_REJOIN_FAILED**

  Η συσκευή δε συνδέθηκε σε νέο δίκτυο, αλλά είναι ακόμα συνδεδεμένη στο παλιό δίκτυο

- **EV_TXCOMPLETE**

  Τα δεδομένα τα οποία ετοίμασε η συνάρτηση LMIC_setTxData() έχουν αποσταλεί και το downstream των δεδομένων έχει ληφθεί. Αν απαιτήθηκε επιβεβαίωση λήψης, τότε εκείνη έχει ληφθεί

- **EV_RXCOMPLETE**

  Τα δεδομένα downstream έχουν ληφθεί

- **EV_SCAN_TIMEOUT**

  Μετά την κλήση της LMIC_enableTracking() δεν έχει ληφθεί beacon στο χρονικό παράθυρο που προβλέπεται. Η ανίχνευση (tracking) θα πρέπει να επανεκκινήσει

- **EV_BEACON_FOUND**

  Μετά την κλήση της συνάρτησης LMIC_enableTracking() το πρώτο beacon έχει ληφθεί μέσα στο ορισμένο χρονικό παράθυρο

- **EV_BEACON_TRACKED**

  Το επόμενο beacon έχει ληφθεί την αναμενόμενη χρονική στιγμή

- **EV_BEACON_MISSED**

Δε λήφθηκε κάποιο beacon την αναμενόμενη χρονική στιγμή

- **EV_LOST_TSYNC**

Το beacon χάνεται επανειλημμένα και ο χρονικός συγχρονισμός έχει χαθεί. Η ανίχνευση (pinging) πρέπει να επανεκκινήσει

- **EV_RESET**

Επαναφορά της συνεδρίας (session). Η σύνδεση στο δίκτυο θα αποκατασταθεί αυτόματα και η συσκευή θα λάβει νέα συνεδρία

- **EV_LINK_DEAD**

Δε λήφθηκε επιβεβαίωση από τον εξυπηρετητή δικτύου για μεγάλη χρονική περίοδο. Οι μεταδόσεις από τη συσκευή είναι δυνατές, αλλά η λήψη τους αβέβαιη

5) Η δομή LMIC

Αντί να μεταφέρονται πολλαπλές παράμετροι μπρος και πίσω μεταξύ των API και Callback συναρτήσεων, πληροφορίες σχετικά με την κατάσταση του πρωτοκόλλου είναι προσβάσιμες από μια καθολική δομή, την LMIC, η οποία παρουσιάζεται παρακάτω. Όλα τα πεδία, εκτός από εκείνα που ρητά αναγράφεται, είναι διαθέσιμα μόνο για ανάγνωση (read-only) και δεν ενδείκνεται η τροποποίησή τους.

```
struct lmic_t {
    u1_t        frame[MAX_LEN_FRAME];
    u1_t        dataLen;    // 0 no data or zero length data, >0 byte count of data
    u1_t        dataBeg;    // 0 or start of data (dataBeg-1 is port)

    u1_t        txCnt;
    u1_t        txrxFlags;  // transaction flags (TX-RX combo)

    u1_t        pendTxPort;
    u1_t        pendTxConf; // confirmed data
    u1_t        pendTxLen;
    u1_t        pendTxData[MAX_LEN_PAYLOAD];

    u1_t        bcnChnl;
    u1_t        bcnRxsyms;
    ostime_t    bcnRxtime;
    bcninfo_t   bcninfo;    // Last received beacon info

    …
    …
};
```

Εικόνα 49: Δομή LMIC

Η παραπάνω αναπαράσταση της δομής δεν περιέχει όλα τα πεδία της δομής, από τη στιγμή που τα περισσότερα από αυτά είναι για εσωτερική μόνο χρήση (internal usage). Τα πιο σημαντικά πεδία για εξέταση κατά την λήψη (γεγονότα EV_RXCOMPLETE ή EV_TXCOMPLETE) είναι το txrxFlags για πληροφορίες κατάστασης, καθώς και τα frame[ ] και dataLen / dataBeg για το ωφέλιμο φορτίο λήψης της εφαρμογής. Για την μετάδοση, τα πιο σημαντικά πεδία είναι τα pendTxData[ ], pendTxLen και pendTxConf, τα οποία χρησιμοποιούνται ως ορίσματα στην API συνάρτηση LMIC_setTxData(), η οποία προετοιμάζει την αποστολή δεδομένων.

Για τα γεγονότα EV_RXCOMPLETE και EV_TXCOMPLETE, το πεδίο txrxFlags περιέχει πληροφορίες σχετικά με τη μετάδοση. Πιο συγκεκριμένα:

For the EV_TXCOMPLETE event the fields have the following values:

Received frame	LMIC.txrxFlags						LMIC.dataLen	LMIC.dataBeg
	ACK	NACK	PORT	DNW1	DNW2	PING		
nothing	0	0	0	0	0	0	0	0
empty frame	x	x	0	x	x	0	0	x
port only	x	x	1	x	x	0	0	x
port+payload	x	x	1	x	x	0	x	x

For the EV_RXCOMPLETE event the fields have the following values:

Received frame	LMIC.txrxFlags						LMIC.dataLen	LMIC.dataBeg
	ACK	NACK	PORT	DNW1	DNW2	PING		
empty frame	0	0	0	0	0	1	0	x
port only	0	0	1	0	0	1	0	x
port+payload	0	0	1	0	0	1	x	x

Εικόνα 50: LMIC.txrxFlags

6) Συναρτήσεις API

Η βιβλιοθήκη LMIC παρέχει ένα σύνολο από συναρτήσεις API για τον έλεγχο της κατάστασης MAC και τηνπυροδότηση δράσεων πρωτοκόλλου. Οι πιο βασικές από αυτές είναι:

- void **LMIC_reset** ()

  Επαναφορά της κατάστασης MAC. Η συνεδρία και οι εκκρεμές μεταφορές δεδομένων θα απορριφθούν.

- bit_t **LMIC_startJoining** ()

  Ξεκινά αμέσως τη σύνδεση στο δίκτυο. Θα καλεστεί αυτόματα από άλλες συναρτήσεις API σε περίπτωη που δεν έχει καθιερωθεί ακόμα κάποια συνεδρία. Τα γεγονότα EV_JOINING και EV_JOINED ή EV_JOIN_FAILED θα παραχθούν με την κλήση της

- void **LMIC_tryRejoin** ()

Ελέγχει αν υπάρχουν άλλα δίκτυα που μπορεί να συνδεθεί η συσκευή. Η συνεδρία του τρέχοντος δικτύου διατηρείται αν δε βρεθεί νέο δίκτυο. Τα γεγονότα EV_JOINED ή EV_JOIN_FAILED θα παραχθούν κατά την κλήση της

- void **LMIC_setSession** (u4_t netid, devaddr_t devaddr, u1_t* nwkKey, u1_t* artKey)

Θέτει τη συνεδρία με βάση τις παραμέτρους. Για την ένταξη στην ήδη υπάρχουσα συνεδρία, τα πεδία LMIC.seqnoUp και LMIC.seqnoDn πρέπει να αποκατασταθούν στις τελευταίες τους τιμές

- bit_t **LMIC_setupBand** (u1_t bandidx, s1_t txpow, u2_t txcap)

Δημιουργεί μια νέα μπάντα με συγκεκριμένη ισχύ μετάδοσης και duty cycle (1/txcap)

- bit_t **LMIC_setupChannel** (u1_t channel, u4_t freq, u2_t drmap, s1_t band)

Δημιουργεί ένα νέο κανάλι στην δοσμένη ζώνη με τη δοσμένη συχνότητα και επιτρέπει τα data rates που ορίζεται από το όρισμα data rate bitmask

- void **LMIC_disableChannel** (u1_t channel)

Καθιστά ανίκανο το συγκεκριμένο κανάλι

- void **LMIC_setDrTxpow** (dr_t dr, s1_t txpow)

Θέτει το data rate και την ισχύ μετάδοσης

- void **LMIC_setTxData** ()

Προετοιμάζει τα δεδομένα για αποστολή την επόμενη δυνατή χρονική στιγμή.

- void **LMIC_clrTxData** ()

Αφαιρεί τα δεδομένα που είχαν προετοιμαστεί για αποστολή

- bit_t **LMIC_enableTracking** (u1_t tryBcnInfo)

Επίτρεψη του εντοπισμού beacons. Η τιμή 0 δηλώνει το ξεκίνημα για σκανάρισμα beacons άμεσα

- void **LMIC_disableTracking** ()

Απενεργοποίηση των beacons

- void **LMIC_setPingable** (u1_t intvExp)

Επιτρέπει την λήψη downstream δεδομένων

- void **LMIC_stopPingable** ()

Σταματά την λήψη downstream δεδομένων

2.4.2 OTAA και ABP

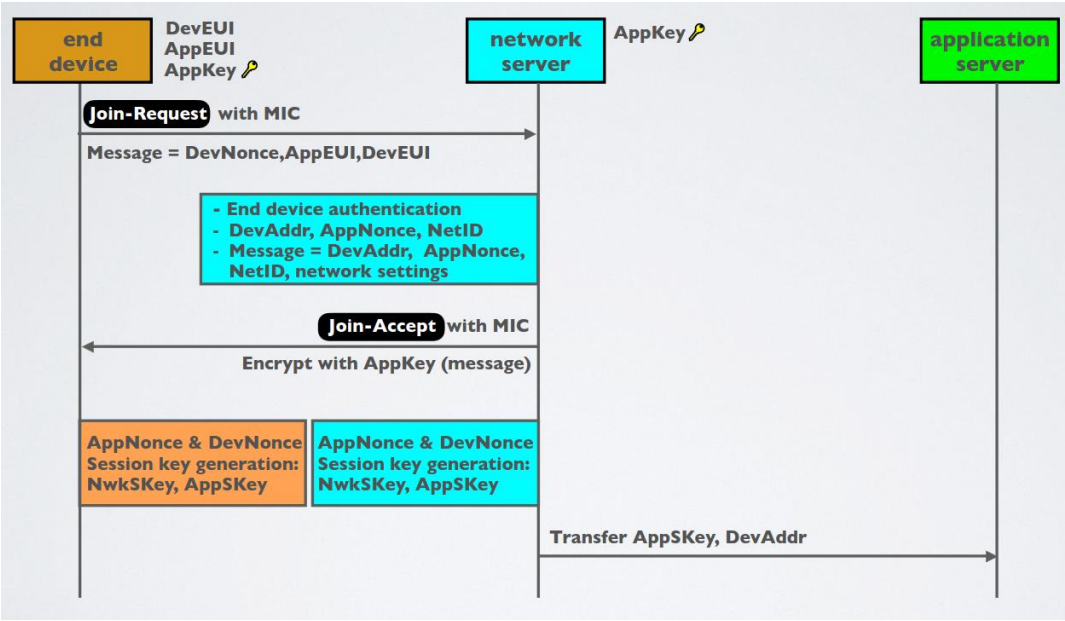
Οι όροι OTAA (Over The Air Activation) και ABP (Activation By Personalization) είναι δύο μέθοδοι ενεργοποίησης για το end device, το οποίο οφείλει να να ενσωματώσει μια από αυτές προκειμένου να συνδεθεί στον διακομιστή δικτύου.

1) OTAA

Συνήθως χρησιμοποιείται αυτή η μέθοδος ενεργοποίησης, από τη στιγμή που παρέχει τον πιο ασφαλή τρόπο σύνδεσης του τελικού κόμβου με τον εξυπηρετητή δικτύου. Πριν την ενεργοποίηση, πρέπει να γνωρίζουμε τις παραμέτρους DevEUI, AppEUI και AppKey του end device. Ο σέρβερ θα πρέπει να γνωρίζει και να αποθηκεύει το ίδιο AppKey για την πραγματοποίηση της σύνδεσης. Τα αρχικά EUI σημαίνουν Extended Unique Identifier και αποτελούν έναν 64-bit αριθμό, ο οποίος χρησιμοποιείται συχνά για την ταυτοποίηση στοιχείων δικτύου. Ο αριθμός αυτός, αντιστοιχεί σε ένα μοναδικό end device (αντίστοιχα όπως οι διευθύνσεις MAC), ενώ κάποιες συσκευές έχουν κατά την κατασκευή τους τέτοιον αριθμό.

Το AppEUI αναγνωρίζει μονοσήμαντα τον εξυπηρετητή εφαρμογής και έχει παρόμοια λειτουργία με τη θύρα (port) στα δίκτυα.

Το AppKey αποτελεί ένα AES (Advanced Encryption Standard) συμμετρικό κλειδί 128 bit (συχνά αναφέρεται ως root key) και αξιοποιείται για την δημιουργία του MIC (Message Integrity Code), το οποίο εξασφαλίζει την ακεραιότητα του μηνύματος.



Εικόνα 51: OTAA activation

Παρακάτω ακολουθεί αναλυτική επεξήγηση της παραπάνω εικόνας:

Το end device δημιουργεί το DevNonce, το οποίο αποτελεί έναν αριθμό που παράγεται τυχαία και αποτρέπει την πολλαπλή αίτηση εισόδου (Join-Request). Επιπλέον, ο τελικός κόμβος κατασκευάζει το μήνυμα που περιέχει το DevNonce, το AppEUI και το DevEUI και έπειτα με τη χρήση του AppKey κατασκευάζεται το MIC. Με την πραγματοποίηση του Join_Request, αποστέλεται στον εξυπηρετητή δικτύου το μήνυμα μαζί με το MIC. Ο εξυπηρετητής διαβάζει το DevNonce προκειμένου να αποφανθεί αν είναι νέα συσκευή ή έχει ήδη πραγματοποιήσει σύνδεση σε αυτόν.

Τα περιεχόμενα του μηνύματος δεν είναι κρυπτογραφημένα με βάση το AppKey. Ο σέρβερ με βάση το μήνυμα και το δικό του AppKey (το οποίο οφείλει να είναι το ίδιο με το AppKey της τελικής συσκευής), υπολογίζει το δικό του MIC και μόνο στη περίπτωση που ταυτίζεται με εκείνο του end device πραγματοποιεί τη σύνδεση. Αν ταυτοποιηθεί ένας τελικός κόμβος, τότε ο διακομιστής επιστρέφει τις παρακάτω τιμές:

- **DevAddr**: Αντιστοιχεί το DevEUI σε μια μικρότερη διεύθυνση του εσωτερικού δικτύου, η οποία είναι 32 bit (το DevEUI είναι 64 bit). Αυτό συμβάλλει στη μείωση των μεταφερόμενων πακέτων, αφού αυτά εμπεριέχουν ένα overhead πρωτοκόλλου. Η διεύθυνση αυτή μοιάζει με την IP ενός client.
- **AppNonce**: Αριθμός που παράγεται με τυχαίο τρόπο
- **NetID**: Αναγνωριστικό δικτύου

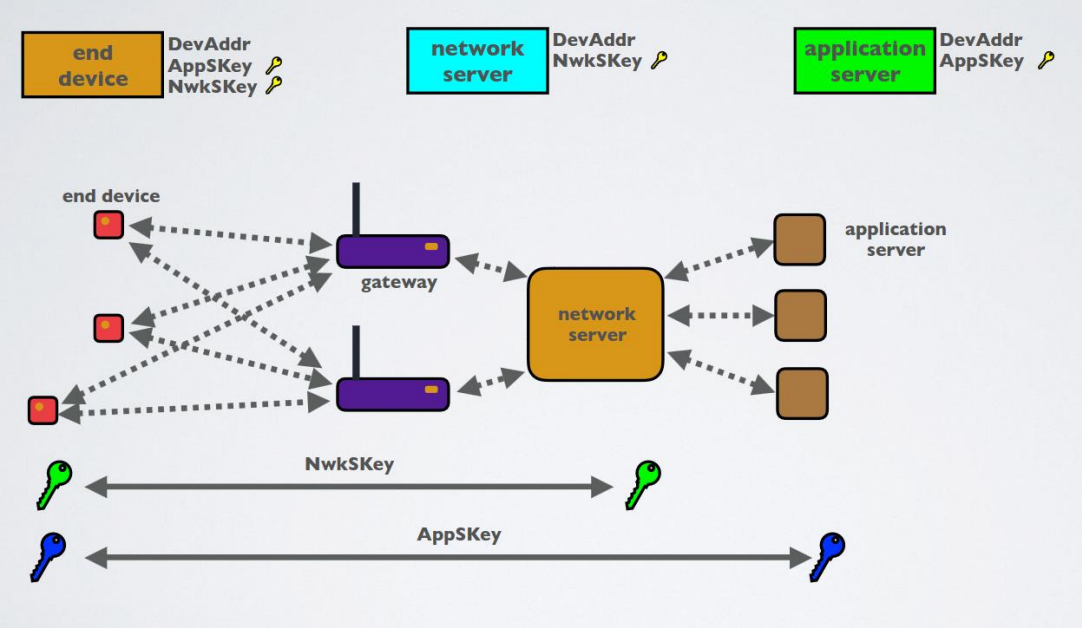
Τέλος κατασκευάζει και επιστρέφει ένα μήνυμα που περιέχει τα 3 παραπάνω μαζί με τις ρυθμίσεις δικτύου (network settings). Οι ρυθμίσεις αφορούν την λήψη (data rates για λήψη), τη καθυστέρηση λήψης (χρόνος μεταξύ αποστολής και λήψης), τα κανάλια συχνοτήτων και άλλα. Το μήνυμα που αποστέλει γίνεται με κρυπτογράφηση με βάση το AppKey.

Μετά την εγκαθίδρυση της σύνδεσης, ο σέρβερ και ο τελικός κόμβος μοιράζονται τα ίδια AppNonce και DevNonce. Με βάση τις δύο αυτές τιμές, παράγονται δύο κλειδιά για τη συγκεκριμένη συνεδρία (session): το Network Session Key (NwkSKey) και το Application Session Key (AppSKey). Το δεύτερο μαζί με τη διεύθυνση συσκευής μεταφέρονται στον εξυπηρετητή εφαρμογής (application server).

Το NwkSKey χρησιμοποιείται από το end device και τον διακομιστή για τον υπολογισμό και επαλήθευση της ακεραιότητας των μηνυμάτων. Επιπλέον, χρησιμοποιείται για την κρυπτογράφηση και αποκρυπτογράφηση του ωφέλιμου φορτίου του μηνύματος. Το AppSKey χρησιμοποιείται για την εξασφάλιση επικοινωνίας end-to-end μεταξύ του σέρβερ και των συσκευών. Αξιοποιείται για την κρυπτογράφηση των μηνυμάτων προς και από τον εξυπηρετητή. Όμως δεν εξασφαλίζει την ακεραιότητα του μηνύματος, από τη στιγμή που ο application server μπορεί να αλλοιώσει το περιεχόμενο του μηνήματος πριν τη μετάδοση.

2) ABP

Σε αυτή τη μέθοδο, δεν υπάρχουν αιτήσεις και αποδοχές σύνδεσης (Join-Request και Join-Accept). Επιπλέον, ο τελικός κόμβος δεν αποθηκεύει τα DevEUI, AppEUI και AppKey και επομένως ούτε ο σέρβερ αποθηκεύει το AppKey. Η τελική συσκευή εμπεριέχει τα AppSKey και NwkSKey τα οποία μοιράζεται με τους εξυπηρετητές εφαρμογής και δικτύου αντίστοιχα. Επιπλέον, αποθηκεύει τη διεύθυνση συσκευής DevAddr. Η επικοινωνία γίνεται με τους διακομιστές απευθείας και χρησιμοποιούνται τα παραπάνω κλειδιά για κρυπτογράφηση.



Εικόνα 52: ABP activation

Όλες οι συσκευές LoRaWAN υποχρεούνται να κρυπτογραφούν το ωφέλιμο φορτίο και την επικεφαλίδα των πακέτων με Advanced Encryption Standard (AES) αλγόριθμο, με χρήση κλειδιών 128 bit. Το πρωτόκολλο προσφέρει δύο επίπεδα ασφάλειας:

a. Στο στρώμα δικτύου, η ακεραιότητα του μηνύματος εξασφαλίζεται από το Message Integrity Code με τη χρήση του NwkSKey. Το ωφέλιμο φορτίο είναι κρυπτογραφημένο από άκρη σε άκρη (end-to-end encryption)
b. Στο επίπεδο εφαρμογής, το ωφέλιμο φορτίο κρυπτογραφείται με την αξιοποίηση του κλειδιού AppSKey. Ομοίως έχουμε κρυπτογράφηση από άκρη σε άκρη

Κεφάλαιο 3: Αρχιτεκτονική & Υλοποίηση Συστήματος

3.1 Arduino και Dragino shield

Στη πρώτη έκδοση του συστήματος χρησιμοποιήθηκε για end device ένα Arduino Uno σε συνδυασμό με ένα dragino shield, το οποίο έχει ενσωματωμένο το sx1276 module. Ο τελικός κόμβος τροφοδοτείται με τάση 5 Volt και χρησιμοποιεί την ενεργοποίηση OTAA για τη σύνδεση με τον σέρβερ. Επιπλέον στη συσκευή αυτή συνδέεται και ο HC-SR501 PIR αισθητήρας για ανίχνευση κίνησης. Η λειτουργία του end device είναι να στέλνει ένα μήνυμα κατάστασης κάθε TX_INTERVAL (σταθερά που ορίζεται στον κώδικα) δευτερόλεπτα στον εξυπηρετητή δικτύου, ενώ παράλληλα να στέλνει ένα έκτακτο μήνυμα αν ο αισθητήρας εντοπίσει κίνηση.

Όπως φαίνεται και στην παρακάτω εικόνα, ορίζονται στον τελικό κόμβο οι τιμές AppEUI, DevEUI και AppKey σε μορφή little endian ή big endian και έπειτα αντιγράφονται στον buffer. Αυτό είναι απαραίτητο από τη στιγμή που η μέθοδος ενεργοποίησης είναι η OTAA.

```
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8]={ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]={ 0x56, 0x34, 0x12, 0x90, 0x78, 0x56, 0x34, 0x12 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
static const u1_t PROGMEM APPKEY[16] = { 0xFE, 0x2A, 0x47, 0x43, 0x4C, 0x36, 0xD5, 0x5F, 0x20, 0x44, 0x97, 0x14, 0x11, 0x21, 0xD5, 0x19 };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
```

Εικόνα 53: Ορισμός AppEUI, DevEUI και AppKey

Στην περίπτωση επιτυχούς σύνδεσης στον διακομιστή δικτύου, η onEvent() συνάρτηση θα μεταβεί στην περίπτωση EV_JOINED, όπου και θα τυπώσει στην οθόνη μέσω του Serial τα χαρακτηριστικά της συνεδρίας (nwkKey και artKey).

```
case EV_JOINED:
    Serial.println(F("EV_JOINED"));
    ev_join = true;
    {
      u4_t netid = 0;
      devaddr_t devaddr = 0;
      u1_t nwkKey[16];
      u1_t artKey[16];
      LMIC_getSessionKeys(&netid, &devaddr, nwkKey, artKey);
      Serial.print("netid: ");
      Serial.println(netid, DEC);
      Serial.print("devaddr: ");
      Serial.println(devaddr, HEX);
      Serial.print("AppSKey: ");
      for (size_t i=0; i<sizeof(artKey); ++i) {
        if (i != 0)
          Serial.print("-");
        printHex2(artKey[i]);
      }
      Serial.println("");
      Serial.print("NwkSKey: ");
      for (size_t i=0; i<sizeof(nwkKey); ++i) {
              if (i != 0)
                      Serial.print("-");
              printHex2(nwkKey[i]);
      }
      Serial.println();
    }
```

Εικόνα 54: Case EV_JOINED

Η συνάρτηση loop(), η οποία εκτελείται συνεχώς, εκτελεί τα παρακάτω. Αρχικά, θέτει μια συγκεκριμένη τιμή για το Spreading Factor, ενώ όσο η συσκευή δεν έχει συνδεθεί στον network server τρέχει τη βασική συνάρτηση os_runloop_once(), επιχειρώντας την επίτευξη σύνδεσης. Από τη στιγμή που εκείνη επιτευχθεί, η loop() πραγματοποιεί σε κάθε επανάληψη έλεγχο για το αν πρέπει να στείλει μήνυμα κατάστασης, δηλαδή αν έχουν περάσει τα ορισμένα δευτερόλεπτα από την προηγούμενη αποστολή

μηνύματος. Στη περίπτωση που πρέπει να στείλει εκτελεί και πάλι την os_runloop_once() για την πραγματοποίηση της αποστολής.

Για την υλοποίηση της αποστολής έκτακτου μηνύματος, αξιοποιείται το pin inputPin, στο οποίο συνδέεται η έξοδος του PIR αισθητήρα. Ο συγκεκριμένος αισθητήρας είναι θετικής λογικής, δηλαδή δίνει έξοδο λογικό 1 (εδώ 5V) όταν εντοπίσει κίνηση. Επομένως σε κάθε επανάληψη της βασικής συνάρτησης γίνεται ο έλεγχος της τιμής του inputPin και όταν εντοπιστεί τιμή HIGH στέλνεται μήνυμα εντοπισμού κίνησης στον σέρβερ.

```
void loop() {
    LMIC_setDrTxpow(DR_SF7, 14); // start at SF10

    while (!ev_join) {
      os_runloop_once();
    }

    unsigned long currentMillis = millis();
    // Check if it's time to update the counter
    if (currentMillis - previousMillis >= interval) {
      // Save the last time the counter was updated
      previousMillis = currentMillis;
      Serial.println("STATUS MESSAGE");
      while (millis() - currentMillis <= 1000) {
        os_runloop_once();
      }
    }

    val = digitalRead(inputPin);  // read input value
    while (val == HIGH) {
      val = digitalRead(inputPin);  // read input value
      digitalWrite(ledPin, HIGH);  // turn LED ON
      os_runloop_once();
      // Reset status message timer if an emergency message is sent
      previousMillis = millis();
    }
    digitalWrite(ledPin, LOW);  // turn LED ON
}
```

Εικόνα 55: Arduino loop

Η συνάρτηση do_send(osjob_t* j) προετοιμάζει το μήνυμα πριν αυτό σταλεί. Εκεί γίνεται ξανά ο έλεγχος του inputPin προκειμένου να σταλεί μήνυμα εντοπισμού κίνησης ή μήνυμα κατάστασης ανάλογα την τιμή του. Εδώ χρησιμοποιείται επίσης και ο μορφοποιητής CayenneLPP, ο οποίος διευκολύνει τη διαχείριση του ωφέλιμου φορτίου που πρόκειται να σταλεί. Τέλος, η LMIC_setTxData2 προετοιμάζει το περιεχόμενο για αποστολή και εκτυπώνεται και το αντίστοιχο μήνυμα.

Η υλοποίηση αυτή μπορεί να καλύπτει τις λειτουργικές ανάγκες τις εφαρμογής, όμως έχει κάποια βασικά προβλήματα. Η κατανάλωση της συσκευής είναι πολύ υψηλή, αφού στην κανονική λειτουργία έχουμε κατανάλωση 16mA, ενώ κατά την αποστολή μηνύματος η κατανάλωση φτάνει τα 65mA. Δηλαδή με μια μπαταρία λιθίου 3350mAh η αναμενόμενη διάρκεια λειτουργίας της συσκευής (αν θεωρήσουμε πως στέλνει σπάνια μηνύματα, δηλαδή λειτουργεί κατά κανόνα στα 16mA) υπολογίζεται ως εξής:

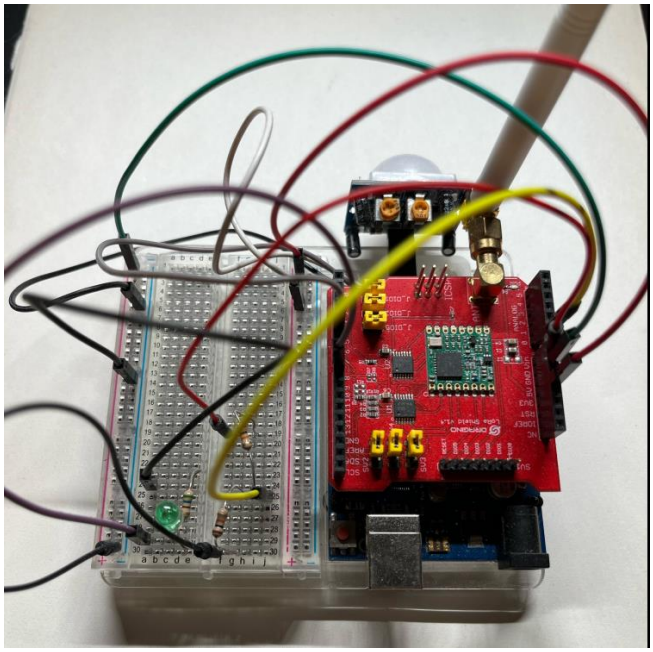
$$Battery\ Life\ (hours) = \frac{3350mAh}{16mA} \cong 209\ hours$$

Συνεπώς η αναμενόμενη διάρκεια λειτουργίας της συσκευής είναι 209 ώρες ή περίπου 8 ημέρες.

```
void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare upstream data transmission at the next possible time

        val = digitalRead(inputPin);  // read input value
        if (val == HIGH) {
          digitalWrite(ledPin, HIGH);  // turn LED ON
        } else {
          digitalWrite(ledPin, LOW);  // turn LED ON
        }

        lpp.reset(); // clear CayenneLPP buffer
        lpp.addDigitalOutput(1, val);

        LMIC_setTxData2(1, lpp.getBuffer(), lpp.getSize(), 0);
        //===
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}
```

Εικόνα 56: Arduino do_send

Από τη στιγμή που η αυτονομία των συστημάτων Internet Of Things είναι ένα από τα θεμιτά χαρακτηριστικά, είναι φανερό πως η υλοποίηση αυτή δεν καλύπτει πλήρως τις μη λειτουργικές ανάγκες. Οι λόγοι που υπάρχει αυξημένοι κατανάλωση ενέργειας οφείλονται τόσο στις ενεργειακές απαιτήσεις του αισθητήρα, ο οποίος λειτουργεί στα 5 Volt, όσο και στην υλοποίηση από πλευράς κώδικα του μικροελεγκτή. Για το πρώτο ζήτημα, μπορούμε να επιλέξουμε έναν μικροελεγχτή και έναν αισθητήρα με μικρότερη τάση λειτουργίας, ενώ για το δεύτερο να κάνουμε βελτιώσεις στον κώδικα. Επομένως, υλοποιήθηκε και η δεύτερη έκδοση του τελικού κόμβου σε esp32 λαμβάνοντας υπόψη τις παραπάνω παρατηρήσεις.



Εικόνα 57: Arduino και Dragino Shield

3.2 Esp32 και SX1276

Για την επίλυση των παραπάνω ζητημάτων, επιλέχθηκε το esp32 ως μικροεπεξεργαστής, ο οποίος έχει τάση λειτουργίας τα 3.3 Volt, ενώ ταυτόχρονα έχει τη δυνατότητα να εισέλθει σε κατάσταση βαθύ ύπνου (deep sleep) εξοικονομώντας ενέργεια [12]. Οι δυνατές καταστάσεις ύπνου συνοψίζονται στον παρακάτω πίνακα:

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Εικόνα 58: Esp32 sleep modes

Ενώ οι ενεργειακές απαιτήσεις της καθεμίας από αυτές φαίνονται παρακάτω:

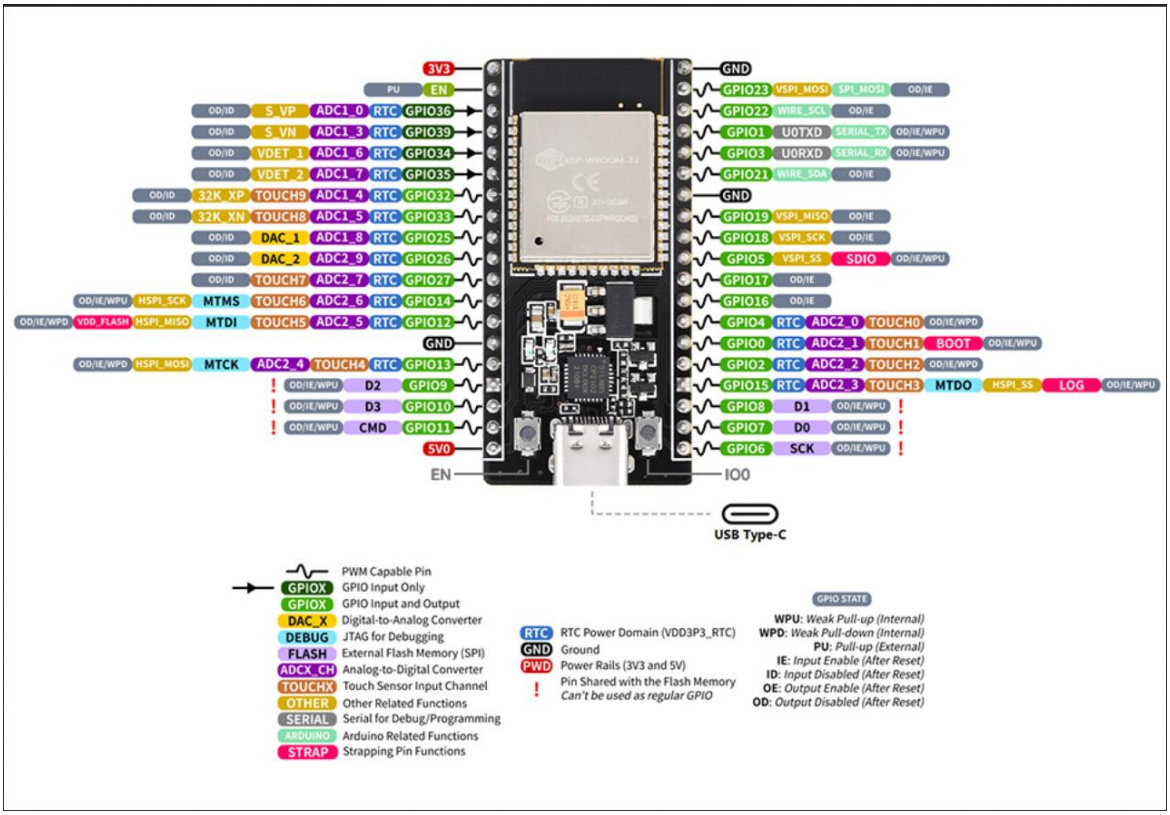
Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	Please refer to Table 10 for details.
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA
		Normal speed 80 MHz: 20 mA ~ 25 mA
		Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μA

Εικόνα 59: Esp32 consumption per mode

Η κατάσταση που επιλέχθηκε στη παρούσα υλοποίηση είναι η Deep-sleep για όσο χρόνο ο τελικός κόμβος δεν μεταδίδει ή λαμβάνει μήνυμα. Θεωρητικά η κατανάλωση θα μπορούσε να έχει φτάσει τα 10μA όπως φαίνεται στο διάγραμμα, αλλά στην κατάσταση βαθύ ύπνου είχαμε κατανάλωση 4mA, γεγονός που οφείλεται στη κατανάλωση τόσο του αισθητήρα και του μετρητή της μπαταρίας, όσο και στο LoRa module. Καταφέραμε σε σχέση με τη πρώτη υλοποίηση 4 φορές μικρότερη κατανάλωση, συνεπώς και τετραπλάσια αυτονομία μπαταρίας.

Παρατηρούμε πως στη κατάσταση Deep-sleep λειτουργούν μόνο το RTC timer και RTC memory (RTC = Real Time Clock). Η πρώτη λειτουργία δίνει τη δυνατότητα ενός μετρητή, ο οποίος μπορεί να αφυπνίζει τον μικροελεγχτή και να τον επαναφέρει στην κανονική του λειτουργία μετά από ένα ορισμένο χρονικό διάστημα. Από την άλλη, το RTC memory επιτρέπει την αποθήκευση τιμών ακόμα και κατά τη διάρκεια εισόδου του esp32 σε βαθύ ύπνο, γεγονός που θα φανεί χρήσιμο καθώς εκεί θα αποθηκεύσουμε τα δεδομένα συνεδρίας μετά την πραγματοποίηση της σύνδεσης με τον διακομιστή δικτύου.

Το esp32 πρέπει να επανέλθει στην κανονική του λειτουργία όταν θέλουμε να στείλουμε ένα μήνυμα στον σέρβερ. Η μετάβαση αυτή πραγματοποιείται με την αλλαγή της τάσης σε ένα GPIO pin του ελεγχτή. Ένα μεγάλο πλεονέκτημα του esp32 σε σχέση με το Arduino uno είναι πως όλα του τα pins (εκτός εκείνα της τροφοδοσίας) είναι GPIO pins, το οποίο επιτρέπει την εύκολη επέκταση του τελικού κόμβου να υποστηρίξει και άλλους αισθητήρες, αφού θα υπάρχουν διαθέσιμα interrupt pins. Τα pins του esp32 φαίνονται παρακάτω:



Εικόνα 60: Esp32 pins

3.2.1 Ανάλυση Κώδικα

Όπως και προηγουμένως, η μέθοδος ενεργοποίησης είναι OTAA και δίνονται και πάλι τα AppEUI, DevEUI, AppKey. Έπειτα ορίζονται οι RTC μεταβλητές, οι οποίες παραμένουν ακόμα και μετά την είσοδο σε βαθύ ύπνο, διότι κρίνεται σημαντικό να κρατούν την προηγούμενη τιμή τους κατά την είσοδό τους στην κανονική λειτουργία. Συγκεκριμένα, αποθηκεύουμε το αντικείμενο LMIC το οποίο έχει εξηγηθεί παραπάνω και το χρειαζόμαστε για αποθήκευση των κλειδιών συνεδρίας, το χρονικό διάστημα μεταξύ αποστολής μηνυμάτων κατάστασης (timeToSleep), το spreading factor και τέλος το χρονικό διάστημα για το οποίο μας ενδιαφέρει να λαμβάνουμε μηνύματα ανίχνευσης κίνησης.

```
RTC_DATA_ATTR lmic_t RTC_LMIC;

RTC_DATA_ATTR int timeToSleep;
RTC_DATA_ATTR int spreadingFactor;
RTC_DATA_ATTR int hourFrom;
RTC_DATA_ATTR int hourTo;
RTC_DATA_ATTR int minuteFrom;
RTC_DATA_ATTR int minuteTo;
```

Εικόνα 61: RTC Data

Επιπλέον έχουμε και δύο βοηθητικές συναρτήσεις για να εκτυπώνεται στην οθόνη ο λόγος διακοπής της λειτουργίας βαθύ ύπνου και επαναφοράς στην κανονική λειτουργία. Οι συναρτήσεις αυτές είναι σημαντικές για λόγους αποσφαλμάτωσης καθώς και επαλήθευσης της σωστής λειτουργίας της εφαρμογής. Υπάρχουν πέντε δυνατοί τρόποι αφύπνισης του μικροεπεξεργαστή οι οποίοι συνοψίζονται παρακάτω και διαχειρίζονται αναλόγως από τις συναρτήσεις:

1. **EXT0**: Εξωτερική διακοπή pin (αλλαγή τάσης pin από λογικό 0 σε 1 ή από λογικό 1 σε 0)
2. **EXT1**: Πολλαπλή εξωτερική διακοπή pin (ίδια με την EXT0 αλλά για πολλά pins)
3. **TIMER**: Διακοπή μετά από ορισμένο χρονικό διάστημα
4. **TOUCHPAD**: Με τη χρήση των touch pins
5. **ULP**: Με τη χρήση του ULP co-processor

```
int get_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : return 1;
    case ESP_SLEEP_WAKEUP_EXT1 : return 2;
    case ESP_SLEEP_WAKEUP_TIMER : return 3;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : return 4;
    case ESP_SLEEP_WAKEUP_ULP : return 5;
    default : return -1;
  }
}

void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
  }
}
```

Εικόνα 62: Εμφάνιση λόγου αφύπνισης

Στη συγκεκριμένη εφαρμογή, χρησιμοποιήθηκαν οι διακοπές EXT0 για όταν ο αισθητήρας εντοπίσει κίνηση και ο TIMER για την αποστολή μηνύματος κατάστασης κάθε κάποιο ορισμένο χρονικό διάστημα. Συνεπώς έχουμε τη συνάρτηση GoDeepSleep(), η οποία επισυνάπτει τις θεμιτές διακοπές, μια στο κατάλληλο pin που συνδέεται η έξοδος του διακόπτη (εδώ GPIO_NUM_33) και μια στον RTC μετρητή για *timeToSleep* δευτερόλεπτα:

```
void GoDeepSleep()
{
    Serial.println(F("Go DeepSleep"));
    PrintRuntime();
    Serial.flush();
    //esp_sleep_enable_timer_wakeup(TX_INTERVAL * 1000000);
    //esp_sleep_enable_ext0_wakeup(GPIO_NUM_32,1); // smoke sensor
    // attachInterrupt(digitalPinToInterrupt(GPIO_NUM_32), handleSmokeInterru
    esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,1); // pir sensor
    esp_sleep_enable_timer_wakeup(timeToSleep * uS_TO_S_FACTOR);
    esp_deep_sleep_start();
}
```

Εικόνα 63: Συνάρτηση Deep-sleep

Ακριβώς μετά την μετάβαση στην κανονική λειτουργία, καλείται αυτόματα η συνάρτηση setup(), η οποία αρχικοποιεί το esp32. Εδώ, αρχικοποιείται στο Serial port προκειμένου να μπορούμε να βλέπουμε τα θεμιτά μηνύματα στην οθόνη για πιο εύκολη αποσφαλμάτωση. Στη συνέχεια, αρχικοποιείται ο μετρητής μπαταρίας και η κατάσταση MAC, επομένως απαλείφονται όλες οι συνεδρίες και τα δεδομένα που περίμεναν να σταλούν. Είναι σημαντικό να σημειωθεί πως σε περίπτωση που είχε ήδη ενεργοποιηθεί προηγουμένως το LMIC object (δηλαδή είχε ήδη προηγηθεί σύνδεση στον σέρβερ), το αντικείμενο LMIC_RTC φορτώνεται από την RTC μνήμη και δεν δημιουργείται από την αρχή. Αυτό εξασφαλίζει την επανασύνδεση του κόμβου με τα ίδια κλειδιά συνεδρίας (NwkSKey και AppSKey). Επομένως, κάθε φορά που το end device μπαίνει σε κατάσταση βαθύ ύπνου και επαναφέρεται λόγω κάποιου interrupt επικοινωνεί κατευθείαν με τον εξυπηρετητή δικτύου. Αυτό έχει ως αποτέλεσμα την επίτευξη της ίδιας απόκρισης με προηγουμένως, αλλά ταυτόχρονα την εξοικονόμηση μπαταρίας. Τέλος τυπώνεται ο λόγος αφύπνισης από τον βαθύ ύπνο.

```
void setup() {
    Serial.begin(115200);
    Serial.println(F("Starting DeepSleep test"));

    if(timeToSleep == 0) {
      timeToSleep = 600;
    }
    if(spreadingFactor == 0) {
      spreadingFactor = 7;
    }

    // Battery Gauge
    Wire.begin(21, 22);
    FuelGauge.begin();

    // pinMode(GPIO_NUM_32, INPUT);
    // LMIC init
    os_init();

    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();

    if (RTC_LMIC.seqnoUp != 0)
    {
        LoadLMICFromRTC();
    }

    LoraWANDebug(LMIC);

    print_wakeup_reason();

    // Start job (sending automatically starts OTAA too)
    do_send(&sendjob);
}
```

Εικόνα 64: Esp32 συνάρτηση setup()

Η συνάρτηση do_send(osjob_t* j) είναι υπεύθυνη για την προετοιμασία και αποστολή του ωφέλιμου φορτίου του μηνύματος. Κατά την κλήση της, εάν δεν πραγματοποιείται κάποια άλλη διεργασία αποστολής ή λήψης, επαναφέρει τον buffer. Έπειτα ελέγχει τον λόγο αφύπνισης από το βαθύ ύπνο και θέτει το πρώτο πεδίο του buffer ως 1 σε περίπτωση που εντόπισε κίνηση. Στην αντίθετη περίπτωση, που προέκυψε διακοπή από τον μετρητή και αναμένεται να αποσταλεί μήνυμα κατάστασης, το αντίστοιχο πεδίο λαμβάνει την τιμή 0. Συνεπώς, το ωφέλιμο φορτίο περιέχει την πληροφορία της αιτίας της διακοπής και εκείνη μεταφέρεται στον εξυπηρετητή δικτύου.

Επιπλέον, η μέσα στη συνάρτηση αυτή, πραγματοποιείται η ανάγνωση της υπολειπόμενης μπαταρίας από τον μετρητή μπαταρίας. Από εκείνον παίρνουμε τόσο την υπολειπόμενη μπαταρία σε ποσοστό, όσο και την τάση της μπαταρίας σε Volt. Οι δύο αυτές τιμές ενσωματώνονται σε δύο ξεχωριστά πεδία του buffer. Τέλος η συνάρτηση do_send(osjob_t* j) προετοιμάζει τα δεδομένα για αποστολή (τον buffer με τα τρία πεδία που αναλύθηκαν) και τα αποστέλει την επόμενη δυνατή χρονική στιγμή.

```
void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        lpp.reset(); // clear CayenneLPP buffer

        // if woke up by pir sensor
        if(get_wakeup_reason()==1) {
          lpp.addDigitalInput(1, 1);  // 1 == motion detected
        } else {
          lpp.addDigitalInput(1, 0);  // 0 == status message
        }

        // read battery gauge
        float percentage = FuelGauge.percent();
        float voltage = FuelGauge.voltage() / 1000;

        Serial.print("Battery percentage: ");
        Serial.println(percentage);

        Serial.print("Battery voltage: ");
        Serial.println(voltage);

        lpp.addAnalogInput(2, percentage);
        lpp.addAnalogInput(3, voltage);

        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, lpp.getBuffer(), lpp.getSize(), 0);
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}
```

Εικόνα 65: Esp32 συνάρτηση do_send()

Όπως ήδη έχει αναφερθεί και στο θεωρητικό πλαίσιο του πρωτοκόλλου LoRaWAN, τα end nodes αποτελούν τόσο πομπούς όσο και δέκτες. Στην συγκεκριμένη υλοποίηση, στις απαιτήσεις της εφαρμογής είναι η ρύθμιση κάποιων λειτουργιών του τελικού κόμβου από τον σέρβερ. Επομένως, είναι αναμενόμενη η χρήση των downlinks. Από πλευράς κώδικα, το σημείο διαχείρισης των downlinks γίνεται εντός της συνάρτησης onEvent(ev_t ev) και συγκεκριμένα στο γεγονός EV_TXCOMPLETE, το οποίο σηματοδοτεί την ολοκλήρωση μετάδοσης ενός μηνύματος (περιλαμβάνει και την ολοκλήρωση αναμονής των παραθύρων λήψης).

Το αντικείμενο LMIC περιλαμβάνει το πεδίο dataLen, το οποίο περιέχει το μήκος του μηνύματος λήψης από τον σέρβερ σε bytes. Επομένως, αν το LMIC.dataLen είναι διάφορο του μηδενός, ο τελικός κόμβος έχει λάβει μήνυμα downlink. Από το αντικείμενο LMIC βρίσκουμε επίσης το fPort (Frame Port), το οποίο αποτελεί το κανάλι στο οποίο στάλθηκε το downlink και μπορεί να πάρει τιμές από το 1 έως το 223. Συνήθως χρησιμοποιείται για να διαχωρίσει τα μηνύματα downlink μεταξύ τους.

Στη συγκεκριμένη υλοποίηση, διαχειριζόμαστε μόνο τα μηνύματα από τον σέρβερ με fPort τιμή 1. Αρχικά, δημιουργείται ένας δείκτης, ο οποίος δείχνει στη πρώτη θέση του πίνακα LMIC.frame που περιέχει ωφέλιμη πληροφορία. Στη θέση αυτή (με counter == 0) αναμένεται ο χρόνος αναμονής του τελικού κόμβου μεταξύ δύο μηνυμάτων κατάστασης σε λεπτά. Το περιεχόμενο αυτό που δείχνει ο δείκτης πολλαπλασιάζεται επί 60 για να πάρουμε τον χρόνο μεταξύ δύο διαδοχικών μηνυμάτων κατάστασης σε δευτερόλεπτα. Έπειτα ο μετρητής counter αυξάνεται κατά 1 και ο δείκτης δείχνει στην επόμενη θέση του πίνακα. Για counter == 1, έχουμε την επόμενη θέση του πίνακα, στην οποία έχει ληφθεί από το downlink μήνυμα το επιθυμητό spreading factor. Επομένως, η τιμή του spreading factor γίνεται ίση με εκείνη που δείχνει ο δείκτης και έτσι ο τελικός κόμβος λαμβάνει την νέα τιμή από τον εξυπηρετητή δικτύου. Αφού προηγηθούν όλα τα παραπάνω, ο τελικός κόμβος μπορεί να εισέλθει σε κατάσταση ύπνου και συνεπώς με το πέρας της παραπάνω διαδικασίας θέτουμε την μεταβλητή GOTO_DEEPSLEEP σε true.

```
case EV_TXCOMPLETE:
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
    if (LMIC.txrxFlags & TXRX_ACK)
      Serial.println(F("Received ack"));
    if (LMIC.dataLen) {
      Serial.print(F("Received "));
      Serial.print(LMIC.dataLen);
      Serial.println(F(" bytes of payload"));

      //------ Added ----------------
      fPort = LMIC.frame[LMIC.dataBeg - 1];
      Serial.print("DATABEG FPORT: ");
      Serial.println(fPort);
      if(fPort == 1) {
        uint8_t *ptr = LMIC.frame + LMIC.dataBeg;
        int counter = 0;
        while(counter < 2) {
          if(counter == 0) {    //status interval
            timeToSleep = (*ptr) * 60;
            Serial.print("Time to sleep: ");
            Serial.println(timeToSleep);
          }
          if(counter == 1) {
            spreadingFactor = *ptr;
            Serial.print("Spreading Factor: ");
            Serial.println(spreadingFactor);
          }
          counter += 1;
          ptr++;
        }
      }
    }
    GOTO_DEEPSLEEP = true;
    break;
```

Εικόνα 66: Esp32 περίπτωση EV_TXCOMPLETE

Παρακάτω απεικονίζεται η βασική συνάρτηση loop(), η οποία εκτελείται συνεχώς και καλεί τις υπόλοιπες συναρτήσεις, η λειτουργία των οποίων αναλύθηκε προηγουμένως. Η συνάρτηση αρχικά ελέγχει την τιμή του spreading factor και αναλόγως θέτει την τιμή του data rate και της ισχύς μετάδοσης. Το spreading factor δεν επηρεάζει την ισχύ μετάδοσης, για αυτό και η τιμή της παραμένει η ίδια για όλες τις περιπτώσεις. Έπειτα εκτελείται η os_runloop_once(), η οποία όπως έχει αναφερθεί εκτελεί run-time jobs από τις ουρές εκτέλεσης και αποτελεί τη βασική συνάρτηση διαχείρισης των γεγονότων. Τέλος, πραγματοποιείται ο έλεγχος για κρίσιμες διεργασίες, κατά τη διάρκεια των οποίων δε θα ήταν θεμιτό να εισέλθει ο μικροελεγχτής σε κατάσταση βαθύ ύπνου. Σε περίπτωση που δεν υπάρχουν τέτοιες κρίσιμες διεργασίες και η σημαία GOTO_DEEPSLEEP είναι true, τότε μπορεί το esp32 να μπει με ασφάλεια σε κατάσταση ύπνου. Ειδάλλως, τυπώνει ένα μήνυμα προειδοποίησης ότι υπάρχουν κρίσιμες διεργασίες στην ουρά εκτέλεσης και περιμένει 2 δευτερόλεπτα πριν ελέγξει και πάλι αν τελείωσαν οι διεργασίες αυτές.

```
void loop()
{
    static unsigned long lastPrintTime = 0;

    // set spreading factor
    switch(spreadingFactor) {
      case 7:
        LMIC_setDrTxpow(DR_SF7, 14);
        break;
      case 8:
        LMIC_setDrTxpow(DR_SF8, 14);
        break;
      case 9:
        LMIC_setDrTxpow(DR_SF9, 14);
        break;
      case 10:
        LMIC_setDrTxpow(DR_SF10, 14);
        break;
      case 11:
        LMIC_setDrTxpow(DR_SF11, 14);
        break;
      case 12:
        LMIC_setDrTxpow(DR_SF12, 14);
        break;
    }

    os_runloop_once();

    const bool timeCriticalJobs = os_queryTimeCriticalJobs(ms2osticksRound((TX_INTERVAL * 1000)));
    if (!timeCriticalJobs && GOTO_DEEPSLEEP == true && !(LMIC.opmode & OP_TXRXPEND))
    {
        Serial.print(F("Can go sleep "));
        LoraWANPrintLMICOpmode();
        SaveLMICToRTC(TX_INTERVAL);
        GoDeepSleep();
    }
    else if (lastPrintTime + 2000 < millis())
    {
        Serial.print(F("Cannot sleep "));
        Serial.print(F("TimeCriticalJobs: "));
        Serial.print(timeCriticalJobs);
        Serial.print(" ");

        LoraWANPrintLMICOpmode();
        PrintRuntime();
        lastPrintTime = millis();
    }
}
```

Εικόνα 67: Esp32 συνάρτηση loop()

3.2.2 Ο τελικός κόμβος στην πράξη

- Εκκίνηση και σύνδεση στον εξυπηρετητή δικτύου

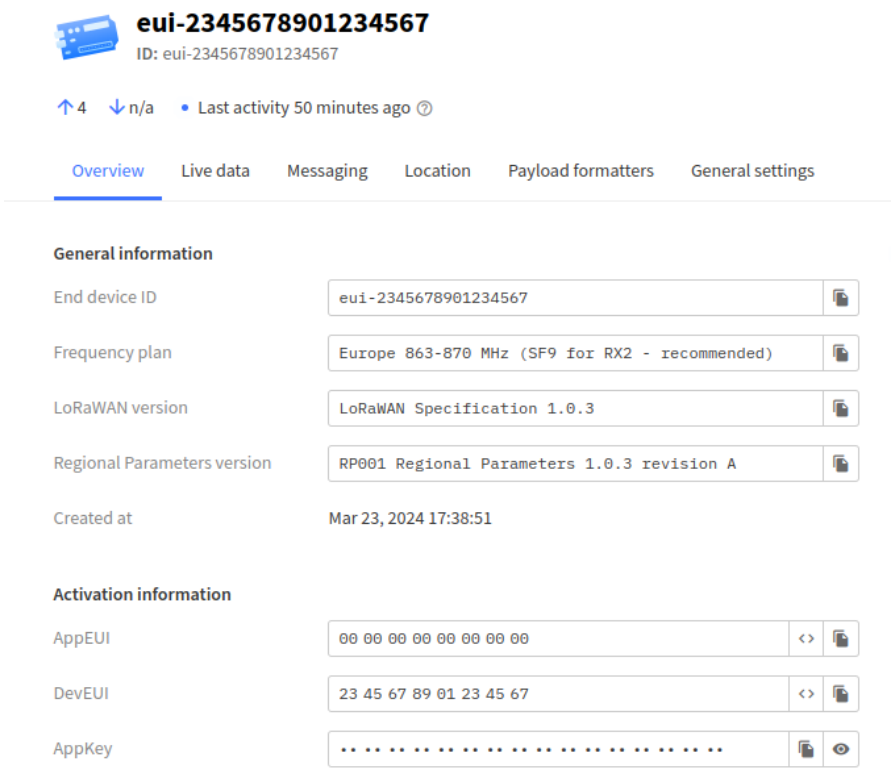
Αφού εξηγήθηκε ο κώδικας αναλυτικά στην παραπάνω ενότητα, είναι σημαντικό να δειχθεί και στην πράξη, καθώς έτσι θα γίνει πιο κατανοητή η λειτουργία του τελικού κόμβου. Παρακάτω απεικονίζεται η σειριακή έξοδος του end device από το εργαλείο Serial Monitor του Arduino IDE τη στιγμή που ρευματοδοτείται για πρώτη φορά.

```
170974: EV_TXSTART
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_JOINING OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 4 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_JOINING OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 6 seconds
492032: EV_JOINED
netid: 0
devaddr: F16378
AppSKey: D7-DE-52-E0-44-DC-D1-7E-02-FC-BE-7B-AA-18-29-36
NwkSKey: F2-58-D2-BB-95-6C-80-27-CB-5B-63-CD-C8-C6-52-03
492778: EV_TXSTART
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 8 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 10 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 12 seconds
812202: EV_TXCOMPLETE (includes waiting for RX windows)
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_POLL OP_RNDTX Runtime: 14 seconds
877743: EV_TXSTART
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXRXPEND OP_NEXTCHNL Runtime: 16 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXRXPEND OP_NEXTCHNL Runtime: 18 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXRXPEND OP_NEXTCHNL Runtime: 20 seconds
1257427: EV_TXCOMPLETE (includes waiting for RX windows)
Can go sleep LMIC.opmode: OP_RNDTX OP_NEXTCHNL Save LMIC to RTC
Reset CFG_LMIC_EU_like band avail
Go DeepSleep
Runtime: 20 seconds
```

Εικόνα 68: Σειριακή έξοδος τελικού κόμβου κατά την εκκίνηση

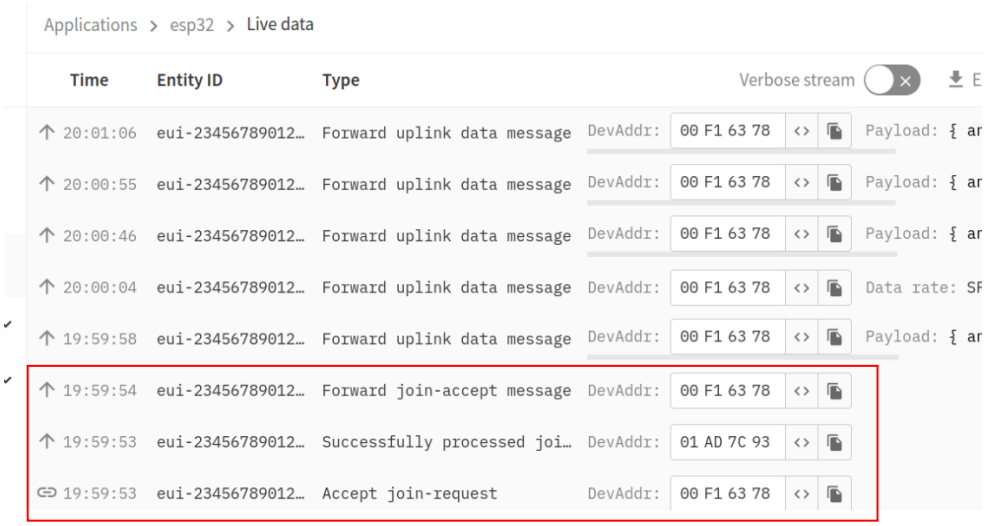
Αρχικά, η τελική συσκευή επιχειρεί να επικοινωνήσει με τον εξυπηρετητή δικτύου με τη μέθοδο ενεργοποίησης OTAA. Όπως ήδη δείχθηκε σε επίπεδο κώδικα, η τελική συσκευή έχει καθορισμένα DevEUI, AppEUI και AppKey. Αυτά δηλώνονται και στον σέρβερ προκειμένου να πραγματοποιηθεί η εγγραφή της συσκευής. Επομένως, η τελική συσκευή αφού ξεκινήσει την επικοινωνία περιμένει για επιβεβαίωση από τον εξυπηρετητή και αφού την λάβει τυπώνει το μήνυμα EV_JOINED, το οποίο υποδηλώνει επιτυχή σύνδεση. Μαζί με το μήνυμα επιτυχούς σύνδεσης στο γεγονός EV_JOINED της συνάρτησης onEvent() τυπώνονται στην οθόνη τα netid, devaddr και τα δύο σημαντικά κλειδιά της συνεδρίας AppSKey και NwkSKey. Ο τελικός κόμβος αφού συνδεθεί, αποθηκεύει στη μνήμη RTC το αντικείμενο LMIC, το οποίο περιέχει μεταξύ άλλων τα κλειδιά συνεδρίας και μόνο τότε μπορεί να εισέλθει σε κατάσταση βαθύ ύπνου όπου και το κάνει τυπώνοντας το μήνυμα “Go DeepSleep”. Είναι φανερό πως η συσκευή ελέγχει κάθε 2 δευτερόλεπτα αν υπάρχουν κρίσιμες διαδικασίες σε εκτέλεση, αλλά από τη στιγμή που δεν έχει γίνει αποθήκευση του αντικειμένου LMIC η μετάβαση σε κατάσταση deep sleep δεν είναι επιτρεπτή. Τέλος αναγράφεται και η διάρκεια λειτουργίας του αισθητήρα όσο είναι στην κανονική του λειτουργία (εδώ λειτούργησε για 20 δευτερόλεπτα πριν μπει σε κατάσταση ύπνου).

Στην παρακάτω εικόνα (εικόνα 69) φαίνεται η συσκευή στον σέρβερ, μαζί με τις πληροφορίες για την μέθοδο ενεργοποίησης (AppEUI, DevEUI, AppKey) καθώς και άλλες γενικές πληροφορίες.



Εικόνα 69: Πληροφορίες συσκευής από την πλευρά του σέρβερ

Επιπλέον, η διαδικασία σύνδεσης της τελικής συσκευής φαίνεται και από την πλευρά του εξυπηρετητή. Η μέθοδος ενεργοποίησης OTAA και η χειραψία δύο σταδίων που αναλύθηκε προηγουμένως, απεικονίζεται και στα μηνύματα (logs) του σέρβερ κάτω από τη συγκεκριμένη εφαρμογή.



Εικόνα 70: Μηνύματα σύνδεσης από την πλευρά του σέρβερ

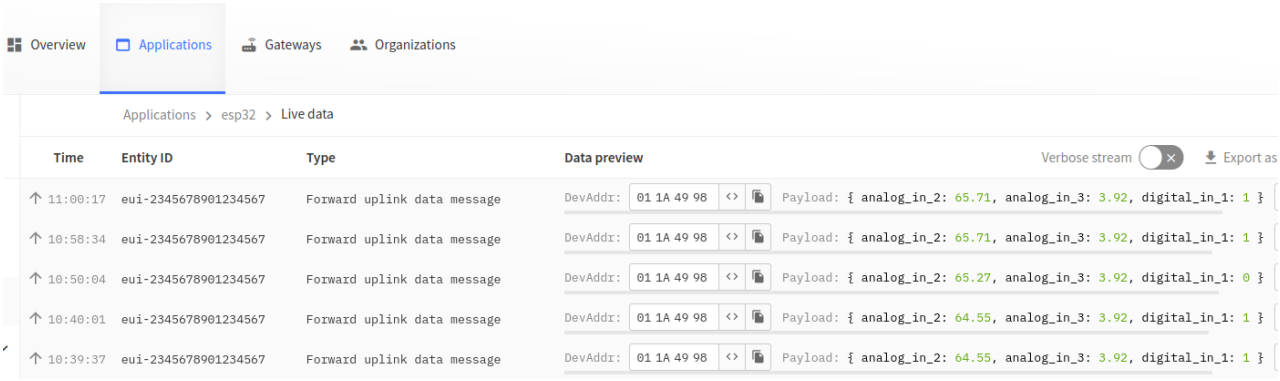
- Διακοπή και αποστολή uplink

Παραπάνω αναλύθηκε η διαδικασία σύνδεσης στον εξυπηρετητή δικτύου και πως αυτή αναπαραστάται τόσο μέσω της σειριακής επικοινωνίας στην οθόνη, όσο και από την πλευρά του εξυπηρετητή δικτύου. Αντίστοιχα, θα εξηγηθεί η περίπτωση που έχουμε διακοπή στην τελική συσκευή και συνεπώς αποστολή μηνύματος uplink στον σέρβερ. Από το Serial Monitor με την πραγματοποίηση μιας διακοπής, βλέπουμε την ακριβή ημερομηνία και ώρα πραγματοποίησής της. Έπειτα φαίνεται ότι το αντικείμενο LMIC φορτώνεται από την μνήμη RTC και τυπώνονται κάποια από τα χαρακτηριστικά του. Όπως έχει αναφερθεί και στην εξήγηση του κώδικα αναγράφεται ο λόγος αφύπνισης και επίσης το ποσοστό υπολοιπόμενης μπαταρίας καθώς και η τάση της. Με τα γεγονότα EV_TXSTART ξεκινάει η μετάδοση του μηνύματος uplink, ενώ με το γεγονός EV_TXCOMPLETE ολοκληρώνεται. Τέλος το αντικείμενο LMIC αποθηκεύεται και πάλι στη μνήμη RTC και ο μικροελεγχτής εισέρχεται σε deep sleep.

```
ets Jul 29 2019 12:21:46

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
Starting DeepSleep test
Wire.begin()
Load LMIC from RTC


LMIC.opmode: OP_RNDTX OP_NEXTCHNL
LMIC.seqnoUp = 4
LMIC.globalDutyRate = 0 osTicks, 0 sec
LMIC.globalDutyAvail = 0 osTicks, 0 sec
LMICbandplan_nextTx = 0 osTicks, 0 sec
os_getTime = 3425 osTicks, 0 sec
LMIC.txend = 8981
LMIC.txChnl = 3
Band    avail           avail_sec       lastchnl        txcap
0       0               0               0               1000
1       0               0               11              100
2       0               0               5               10
3       0               0               0               0


Wakeup caused by external signal using RTC_IO
Battery percentage: 0.05
Battery voltage: 3.33
4882: EV_TXSTART
Packet queued
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 2 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 4 seconds
Cannot sleep TimeCriticalJobs: 1 LMIC.opmode: OP_TXDATA OP_TXRXPEND OP_NEXTCHNL Runtime: 6 seconds
385532: EV_TXCOMPLETE (includes waiting for RX windows)
Can go sleep LMIC.opmode: OP_RNDTX OP_NEXTCHNL Save LMIC to RTC
Reset CFG_LMIC_EU_like band avail
Go DeepSleep
Runtime: 6 seconds
```

Εικόνα 71: Σειριακή έξοδος τελικού κόμβου κατά τη διακοπή

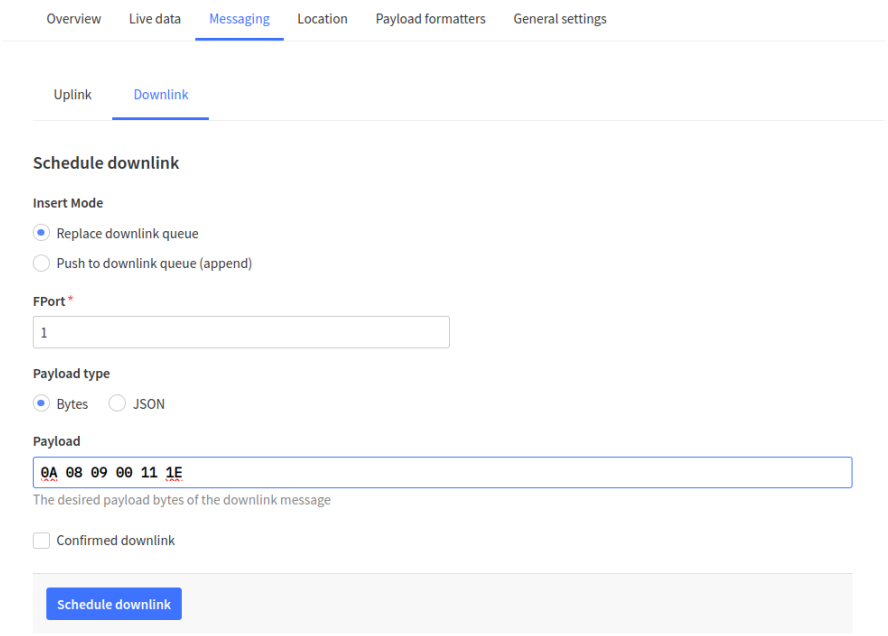
Τα μηνύματα uplink φαίνονται και στον network server. Πιο συγκεκριμένα, βλέπουμε την ώρα άφιξης, το id της συσκευής, τον τύπο του μηνύματος, το ωφέλιμο φορτίο και πολλές άλλες πληροφορίες αν πατήσουμε πάνω σε κάποιο από αυτά. Αξίζει να σημειωθεί πως το analog_in_2 αντιστοιχεί στο ποσοστό της υπολειπόμενης μπαταρίας και το analog_in_3 στην τάση της. Από την άλλη, το digital_in_1 είναι 1 όταν εντοπίστηκε κίνηση από τον pir αισθητήρα, ενώ είναι 0 στην περίπτωση μηνύματος κατάστασης. Παρατηρούμε ότι πράγματι έχουμε digital_in_1 = 0 μόλις 10 λεπτά μετά το προηγούμενο μήνυμα, από τη στιγμή που η προεπιλογή χρόνου αναμονής μεταξύ οποιουδήποτε μηνύματος και μηνύματος κατάστασης είναι 10 λεπτά.



Εικόνα 72: Μηνύματα uplink στον σέρβερ

- Λήψη μηνύματος Downlink

Η ρύθμιση του spreading factor, του χρόνου μεταξύ μηνύματος και μηνύματος κατάστασης καθώς και το ωράριο λειτουργίας του αισθητήρα είναι μεταβλητές οι οποίες μπορούν να ρυθμιστούν από τον σέρβερ και να μεταβούν στο end device. Η διαδικασία αυτή γίνεται στον εξυπηρετητή δικτύου και απεικονίζεται στην εικόνα 73.



Εικόνα 73: Αποστολή μηνύματος downlink από τον σέρβερ

Η λειτουργία εισαγωγής που επιλέγεται είναι η αντικατάσταση της ουράς μηνυμάτων downlink προκειμένου να κρατήσουμε τη τελευταία ενημέρωση που δίνεται από τον εξυπηρετητή δικτύου και να μην αποστέλλονται όλες οι ενδιάμεσες αλλαγές στον τελικό κόμβο. Όπως αναφέρθηκε και παραπάνω, ορίζουμε το FPort = 1 και θεωρούμε ότι οι αλλαγές που αφορούν τις ρυθμίσεις της συσκευής ορίζονται εκεί. Δεν επεξεργαζόμαστε στο επίπεδο του μικροελεγχτή μηνύματα downlink σε άλλα frame ports. Όσον αφορά το payload, αυτό δίνεται σε bytes και αναπαρίσταται σε δεκαεξαδική μορφή.

Το πρώτο byte ορίζει τον χρόνο αναμονής από το τελευταίο μήνυμα μέχρι την αποστολή μηνύματος κατάστασης. Εδώ, έχουμε το δεκαεξαδικό 0A, το οποίο αντιστοιχεί στο δεκαδικό 10, συνεπώς κάθε 10 λεπτά θα λαμβάνουμε status μήνυμα. Το δεύτερο byte αντιστοιχεί στην τιμή του spreading factor και συγκεκριμένα εδώ δίνεται να ισούται με 08 δεκαεξαδικό, δηλαδή δεκαδικό 8. Τα δύο επόμενα bytes ορίζουν την ώρα έναρξης λειτουργίας του pir αισθητήρα (το πρώτο byte για την ώρα και το δεύτερο για τα λεπτά). Αντίστοιχα τα δύο τελευταία bytes ορίζουν την ώρα λήξης αντίστοιχα. Στο συγκεκριμένο downlink μήνυμα, το ωράριο λειτουργίας του αισθητήρα είναι 9:00 – 17:30. Με την επιλογή του κουμπιού “Schedule downlink”, την επόμενη στιγμή που θα επικοινωνήσει ο αισθητήρας με τον σέρβερ, ο αισθητήρας θα παραλάβει και τις ενημερώσεις στα προαναφερθέντα πεδία.

Από τη σειριακή έξοδο βλέπουμε πως μετά το γεγονός ολοκλήρωσης της μετάδοσης EV_TXCOMPLETE και αναμονής των παραθύρων λήψης, παραλάβαμε το μήνυμα downlink. Πιο συγκεκριμένα, λάβαμε 6 bytes ωφέλιμου φορτίου στο frame port 1. Τυπώνονται οι νέες τιμές Time to sleep και spreading factor και ενσωματώνονται στη λειτουργία του τελικού κόμβου.

```
1035733: EV_TXCOMPLETE (includes waiting for RX windows)
Received 6 bytes of payload
DATABEG FPORT: 1
Time to sleep: 600
Spreading Factor: 8
Can go sleep LMIC.opmode: OP_RNDTX OP_NEXTCHNL Save LMIC to RTC
Reset CFG_LMIC_EU_like band avail
Go DeepSleep
Runtime: 16 seconds
```

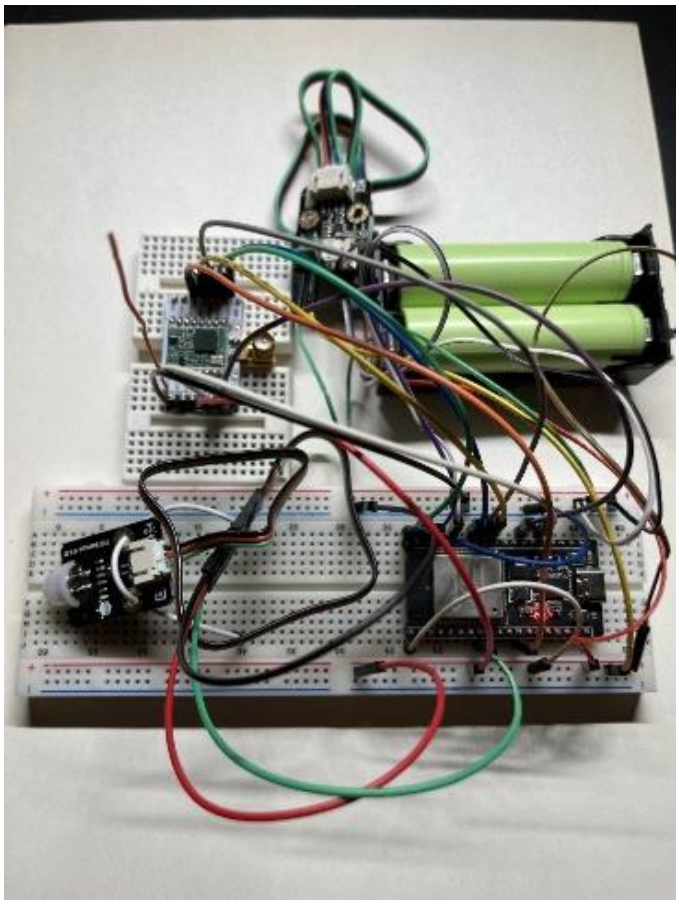
Εικόνα 74: Σειριακή έξοδος τελικού κόμβου κατά τη λήψη downlink

Σε σχέση με την πρώτη υλοποίηση, η κατανάλωση της συσκευής είναι πιο χαμηλή, από τη στιγμή που εκμεταλλευτήκαμε την υλοποίηση με interrupts. Με τη χρήση του ESP32 έχουμε κατανάλωση 5mA, ενώ κατά την αποστολή μηνύματος η κατανάλωση φτάνει τα 65mA. Δηλαδή με μια μπαταρία λιθίου 3350mAh, αναμένουμε υπετριπλάσια διάρκεια λειτουργίας της συσκευής (αν θεωρήσουμε πως στέλνει σπάνια μηνύματα, δηλαδή λειτουργεί κατά κανόνα στα 5mA) σε σχέση με πριν. Επομένως, υπολογίζουμε όπως προηγουμένως:

$$Battery\ Life\ (hours) = \frac{3350mAh}{5mA} \cong 670\ hours$$

Συνεπώς η αναμενόμενη διάρκεια λειτουργίας της συσκευής είναι 670 ώρες ή περίπου 28 ημέρες.

Ο τελικός κόμβος, του οποίου η λειτουργία εξηγήθηκε αναλυτικά, φαίνεται παρακάτω συνδεδεμένος σε μπαταρίες 3.3V:



Εικόνα 75: Esp32 end node circuit

3.3 Backend

3.3.1 Βάση δεδομένων Firebase

Η Firebase είναι μια πλατφόρμα για διαδικτυακές εφαρμογές, με εργαλεία και υποδομές, με τα οποία μπορεί να βοηθήσει τους προγραμματιστές να αναπτύξουν εφαρμογές υψηλής ποιότητας. Είναι μια Backend-as-a-Service (BaaS) πλατφόρμα ανάπτυξης εφαρμογών που παρέχει backend services όπως realtime database, cloud storage, authentication, crash reporting, Machine Learning, remote configuration, και hosting για τα στατικά αρχεία.

Αποθηκεύει τα δεδομένα σε JavaScript Object Notation (JSON) και δεν χρησιμοποιεί ερώτημα για την εισαγωγή, ενημέρωση, διαγραφή ή προσθήκη δεδομένων μιας και μπορεί να αποτελέσει backend ενός συστήματος που χρειάζεται μια βάση δεδομένων για την αποθήκευση. Η εταιρεία ιδρύθηκε το 2011 από τον Andrew Lee και James Tamplin. Η Firebase είναι μια πραγματικού χρόνου (real-time) βάση δεδομένων, η οποία παρέχει ένα API που επιτρέπει στους προγραμματιστές να αποθηκεύουν και να συγχρονίζουν δεδομένα μεταξύ πολλαπλών clients. Με τη πάροδο του χρόνου, έχει επεκτείνει τη σειρά των προϊόντων της σε σημείο να έχει γίνει πλέον μια πλήρης σουίτα για την ανάπτυξη εφαρμογών. Η εταιρεία εξαγοράστηκε από την Google τον Οκτώβριο του 2014 και πρόσθεσε ένα σημαντικό αριθμό νέων χαρακτηριστικών το Μάιο του 2016, στο συνέδριο Google I/O [14].

Παρακάτω αναλύονται οι πιο σημαντικές υπηρεσίες που μπορεί να προσφέρει η Firebase:

- **Firebase Authentication:** Οι περισσότερες εφαρμογές πρέπει να γνωρίζουν την ταυτότητα ενός χρήστη. Η γνώση της ταυτότητας ενός χρήστη επιτρέπει σε μια εφαρμογή να αποθηκεύει με ασφάλεια τα δεδομένα του χρήστη στο cloud και να παρέχει την ίδια εξατομικευμένη εμπειρία σε όλες τις συσκευές του. Το Firebase Authentication παρέχει υπηρεσίες backend, εύχρηστα SDK και έτοιμες βιβλιοθήκες UI για την πιστοποίηση ταυτότητας των χρηστών στην εφαρμογή. Υποστηρίζει τον έλεγχο ταυτότητας χρησιμοποιώντας κωδικούς πρόσβασης, αριθμούς τηλεφώνου, δημοφιλείς παρόχους ομοσπονδιακής ταυτότητας όπως η Google, το Facebook και το Twitter και πολλά άλλα. Το Firebase Authentication ενσωματώνεται στενά με άλλες υπηρεσίες της Firebase και αξιοποιεί βιομηχανικά πρότυπα όπως το OAuth 2.0 και το OpenID Connect, ώστε να μπορεί να συντονιστεί εύκολα με το προσαρμοσμένο backend.

- **Firebase Realtime Database:** Η βάση δεδομένων Firebase Realtime Database είναι μια βάση δεδομένων που φιλοξενείται στο cloud. Τα δεδομένα αποθηκεύονται ως JSON και συγχρονίζονται σε πραγματικό χρόνο σε κάθε συνδεδεμένο πελάτη. Όταν κατασκευάζονται εφαρμογές πολλαπλών υπηρεσιών με τα SDKs για πλατφόρμες Apple, Android και JavaScript, όλοι οι πελάτες μοιράζονται μια κοινή περίπτωση βάσης δεδομένων Realtime και λαμβάνουν αυτόματα ενημερώσεις με τα νεότερα δεδομένα.

- **Firebase Cloud Firestore:** Η Firestore είναι μια υπηρεσία βάσης δεδομένων που παρέχεται από την Google ως μέρος της πλατφόρμας Firebase. Είναι μια NoSQL βάση δεδομένων που σχεδιάστηκε ειδικά για την ανάπτυξη εφαρμογών στον τομέα του cloud και της ιστοσελίδας. Η Firestore προσφέρει πληθώρα χαρακτηριστικών και πλεονεκτημάτων, συμπεριλαμβανομένων:

1) Ευελιξία σχεδίασης: Η Firestore χρησιμοποιεί μοντέλο δεδομένων βασισμένο σε συλλογές (collections) και έγγραφα (documents), παρέχοντας έναν ευέλικτο τρόπο αποθήκευσης και ανάκτησης δεδομένων.

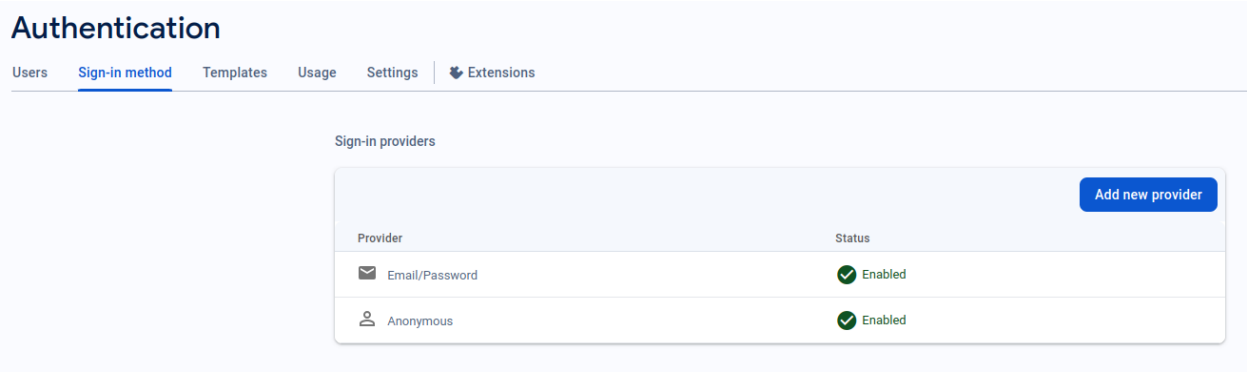
2) Σύγχρονος συγχρονισμός: Η Firestore υποστηρίζει συγχρονισμό σε πραγματικό χρόνο, επιτρέποντας την άμεση ενημέρωση των δεδομένων σε πραγματικό χρόνο σε όλες τις συσκευές που συνδέονται.

3) Σωρευτικά ερωτήματα: Μπορείτε να εκτελέσετε πολύπλοκα ερωτήματα για την ανάκτηση δεδομένων από την Firestore χωρίς να χρειάζεται να φροντίζετε για τη δομή της βάσης δεδομένων.

4) Αυθεντικοποίηση και ασφάλεια: Η Firestore ενσωματώνει την πιστοποίηση χρηστών και διαχείριση των δικαιωμάτων πρόσβασης, προσφέροντας έτσι ασφάλεια για τα δεδομένα.

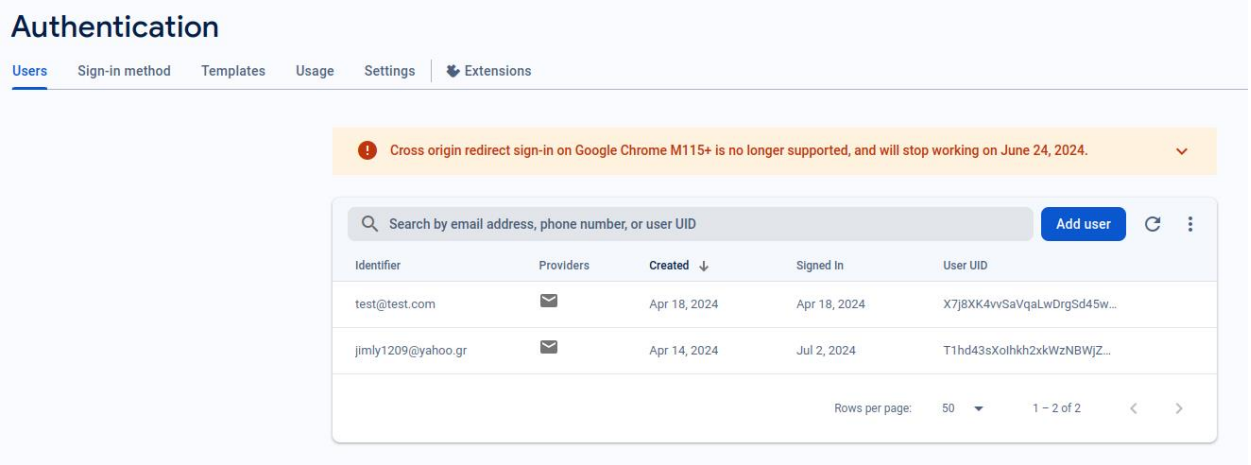
5) Ευελιξία πλατφόρμας: Η Firestore είναι διαθέσιμη για πολλές πλατφόρμες, συμπεριλαμβανομένων των ιστοσελίδων, των κινητών εφαρμογών (Android και iOS) και των εφαρμογών στον τομέα του cloud.

Στη παρούσα εφαρμογή, δημιουργήθηκε μια βάση δεδομένων πραγματικού χρόνου (real time database), από τη στιγμή που υπάρχουν βιβλιοθήκες που ικανοποιούν τις απαιτήσεις μια εφαρμογής κινητού τηλεφώνου (mobile application). Όπως ήδη αναφέρθηκε, η firebase παρέχει υπηρεσίες ασφάλειας και πρόσβασης των δεδομένων. Εδώ, ήταν θεμιτό για λόγους ευκολίας να επιτρέψουμε την ανώνυμη σύνδεση στη βάση (χωρίς credentials), καθώς επίσης και την σύνδεση με email.



Εικόνα 76: Firebase μέθοδοι πιστοποίησης

Υπάρχει η δυνατότητα προβολής όλων των εγγεγραμμένων χρηστών καθώς και η δημιουργία νέων από το πάνελ της firebase. Η δημιουργία νέων χρηστών καθώς και η σύνδεση ήδη εγγεγραμμένων στη βάση θα μπορεί να γίνει και μέσω του app και θα αναλυθεί αργότερα στην αναφορά.



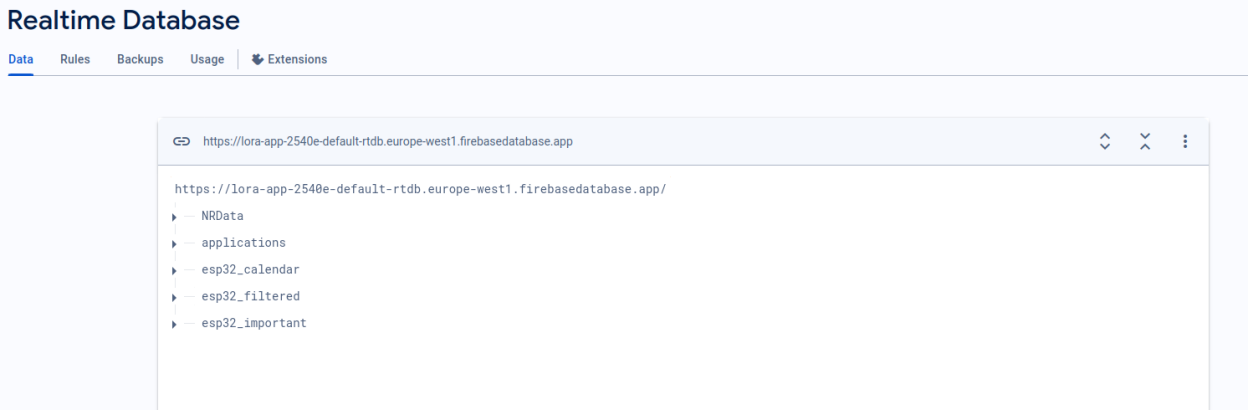
Εικόνα 77: Firebase λίστα χρηστών

Στη βάση δεδομένων πραγματικού χρόνου φτιάχτηκαν διάφορα collections, προκειμένου να αποθηκευτούν στα κατάλληλα σημεία οι πληροφορίες της εφαρμογής. Για παράδειγμα στη συλλογή “applications”, αποθηκεύονται οι πληροφορίες για τα συνδεδεμένα end nodes. Φαίνεται στο όνομα της εφαρμογής, το device_id, το ωράριο λειτουργίας, το spreading factor και η αναμονή μεταξύ μηνυμάτων κατάστασης σε λεπτά. Η αποθήκευση αυτών των πληροφοριών στη δυναμική αυτή βάση δεδομένων, θα φανεί χρήσιμη για την λειτουργία του mobile app.

Εικόνα 78: Firebase συλλογή applications

Παρακάτω φαίνονται όλες οι συλλογές της εφαρμογής. Στην “NRData”, αποθηκεύονται τα μηνύματα από την υλοποίηση του Arduino + dragino shield, στην “esp32_calendar” και “esp32_filtered”, τα μηνύματα από τον τελικό κόμβο esp32, ενώ στην “esp_important” τα μηνύματα του esp32 στα οποία

εντοπίστηκε κίνηση. Με την παραπάνω υλοποίηση υπάρχει μεν πολλαπλή αποθήκευση των ίδιων δεδομένων, αλλά διευκολύνεται κατά πολύ ο προγραμματισμός της εφαρμογής κινητού τηλεφώνου.



Εικόνα 79: Firebase collections

3.3.2 Node-RED

Το Node-RED είναι ένα εργαλείο ανάπτυξης flow-based, αρχικά αναπτυγμένο από την IBM, για τη διασύνδεση υλικών συσκευών, APIs και διαδικτυακών υπηρεσιών με απλό και άμεσο τρόπο. Παρέχει έναν πρόγραμμα επεξεργασίας βασισμένο σε περιηγητή (browser), που καθιστά εύκολη τη σύνθεση ροών χρησιμοποιώντας την ευρεία γκάμα των κόμβων που υπάρχουν στην εργαλειοθήκη και που μπορούν να αναπτυχθούν στη στιγμή.

Τα βασικά χαρακτηριστικά του συνοψίζονται παρακάτω:

- **Προγραμματισμός Βασισμένος στη Ροή**: Το Node-RED χρησιμοποιεί ένα οπτικό περιβάλλον για τη δημιουργία εφαρμογών συνδέοντας κόμβους, ο καθένας από τους οποίους αντιπροσωπεύει ένα κομμάτι λειτουργικότητας, με τρόπο που μοιάζει με διάγραμμα ροής.
- **Ενσωμάτωση με IoT**: Χρησιμοποιείται συνήθως για εφαρμογές IoT, επιτρέποντας την εύκολη ενσωμάτωση συσκευών, αισθητήρων και APIs.
- **Επεκτασιμότητα**: Οι χρήστες μπορούν να δημιουργήσουν τους δικούς τους κόμβους χρησιμοποιώντας JavaScript, επεκτείνοντας τις δυνατότητες του Node-RED.
- **Βιβλιοθήκη Κόμβων**: Διαθέτει μια πλούσια βιβλιοθήκη προκατασκευασμένων κόμβων για ένα ευρύ φάσμα λειτουργιών και υπηρεσιών που μπορούν εύκολα να ενσωματωθούν σε έργα.
- **Ανάπτυξη**: Τρέχει σε διάφορες πλατφόρμες, συμπεριλαμβανομένων των τοπικών υπολογιστών, των διακομιστών και των περιβαλλόντων cloud, καθώς και σε συσκευές όπως το Raspberry Pi.

Ο εξυπηρετητής Node-RED στήθηκε σε τοπικό σέρβερ με τη χρήση docker container. Το image που χρησιμοποιήθηκε είναι από το docker hub και μπορεί να βρεθεί στην παραπομπή [13]. Προηγήθηκε η δημιουργία ενός εικονικού δίσκου, προκειμένου να μην χάνονται οι ρυθμίσεις και τα αρχεία όταν γίνονται αναβαθμίσεις του docker image:

```
$ docker volume create nodered_data
```

Στη συνέχεια ακολούθησε η δημιουργία του αρχείου docker-compose.yaml:

```
version: "3.7"
services:
  node-red:
    image: nodered/node-red:latest
    container_name: nodered
    restart: always
    environment:
      - TZ=Europe/Athens
        # ports:
        #- "1880:1880"
    volumes:
      - nodered_data:/data

volumes:
  nodered_data:
    external: true
    name: nodered_data

networks:
  default:
    name: nginx-proxy-manager_default
    external: true
```

Έγινε χρήση του δικτύου ενός άλλου docker image ως reverse proxy και για αυτό δεν είναι εκτεθειμένη κάποια πόρτα του node-red container. Το docker-compose.yaml του δεύτερου docker image, με τη χρήση του οποίου ανοίγουν οι πόρτες 80, 81 και 443 φαίνεται παρακάτω:

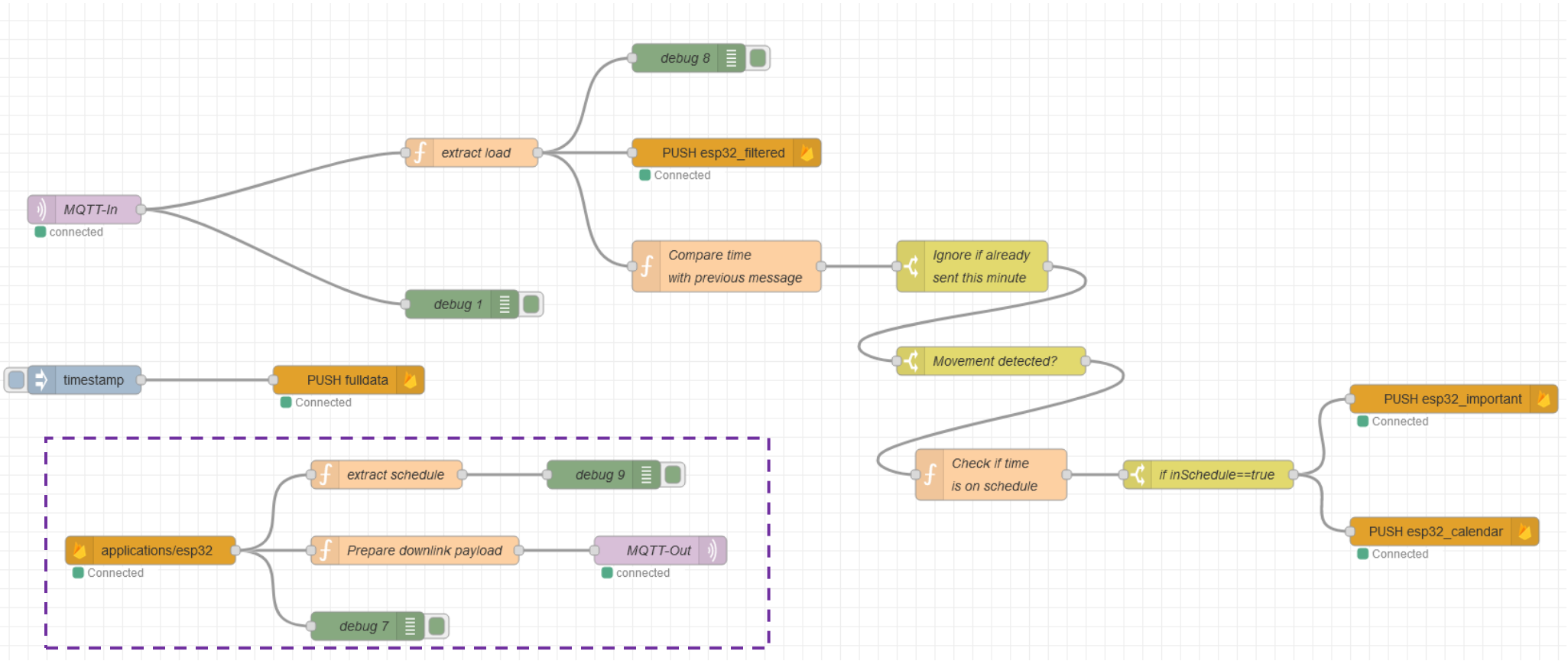
```
version: '3'
```

```
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    container_name: npm
    restart: unless-stopped
    ports:
      - '80:80'
      - '81:81'
      - '443:443'
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
```

Μέσα από περιβάλλον χρήσης του nginx proxy manager και μέσω των υπηρεσιών ddns και letsencrypt προβάλλεται η υπηρεσία node-red στο browser, ενώ παράλληλα ασφαλίζεται με credentials για τον χρήστη στον nginx server.

Με την εισαγωγή των στοιχείων πιστοποίησης στον node-RED server, ο χρήστης εισέρχεται στο βασικό περιβάλλον ανάπτυξης block-based κώδικα. Στην παρακάτω εικόνα (εικόνα 80), απεικονίζεται η λειτουργία του εξυπηρετητή. Το block “MQTT-in”, λαμβάνει από τον the things stack server μηνύματα (τα οποία έχει λάβει μέσω του πρωτοκόλλου LoRa από τα end devices) και εξάγει τις χρήσιμες πληροφορίες του μηνύματος. Έπειτα αποθηκεύει στη βάση δεδομένων την εξαγόμενη πληροφορία και πραγματοποιεί κάποιους επιπλέον ελέγχους για το μήνυμα προκειμένου να αποφανθεί για την σημαντικότητά του. Αυτά τα blocks ελέγχου είναι γραμμένα σε javascript και εξετάζουν αν το μήνυμα έχει εντοπισμό κίνησης και αν βρίσκεται εντός του ορισμένου ωραρίου λειτουργίας από τον χρήστη. Αν ισχύουν οι προϋποθέσεις αυτές, το μήνυμα αποθηκεύεται και σε επιπλέον collections της βάσης δεδομένων όπως φαίνεται και από την εικόνα.

Στο μωβ διακεκομμένο πλαίσιο, περιέχεται η περίπτωση αποστολής downlink μέσω του mobile app, το οποία όπως θα αναλυθεί παρακάτω ανανεώνει τη συλλογή applications/esp32 και ενημερώνει τους listeners ότι πραγματοποιήθηκε κάποια ενημέρωση στη συγκεκριμένη συλλογή της βάσης. Επομένως, ο σέρβερ node-RED που ακούει για αλλαγές στο συγκεκριμένο collection, εξάγει τα επιθυμητά δεδομένα από τη βάση και δημιουργεί ένα νέο μήνυμα το οποίο αποστέλλεται με πρωτόκολλο MQTT στον the things server, ο οποίος με τη σειρά του το διαμορφώνει και το αποστέλλει στην κατάλληλη τελική συσκευή με πρωτόκολλο LoRa. Τα “debug” blocks βοηθούν στον εντοπισμό σφαλμάτων, εφόσον τυπώνουν την έξοδο σε μια σελίδα του node-RED server.



Εικόνα 80: Node-RED diagram esp32

3.4 Mobile Application

Η ανάπτυξη της εφαρμογής έγινε σε κινητά τηλέφωνα, καθώς οι mobile εφαρμογές είναι σχεδιασμένες για να είναι εύκολες στη χρήση, με φιλικές διεπαφές χρήστη που επιτρέπουν ακόμη και στους μη τεχνικούς χρήστες να διαχειρίζονται και να παρακολουθούν τις IoT συσκευές τους. Θεωρήθηκε απαραίτητη η πρόσβαση στην εφαρμογή από οπουδήποτε και οποτεδήποτε από τη στιγμή που πρόκειται για εντοπισμό κίνησης. Οι ενημερώσεις και ειδοποιήσεις που θα λαμβάνει ο χρήστης είναι θεμιτό να γίνονται σε πραγματικό χρόνο, συνεπώς το mobile application διασφαλίζει την απαίτηση αυτή, από τη στιγμή που ο χρήστης έχει συνεχώς πάνω του το κινητό τηλέφωνο.

3.4.1 Flutter

Το Flutter SDK είναι ένα εργαλείο που δημιουργήθηκε από την Google και βοηθά τους προγραμματιστές να δημιουργούν εφαρμογές που λειτουργούν σε διαφορετικές συσκευές, όπως τηλέφωνα και υπολογιστές. Επιτρέπει στους προγραμματιστές να χρησιμοποιούν τον ίδιο κώδικα για διαφορετικά λειτουργικά συστήματα όπως το Android και το iOS. Αυτό διευκολύνει τη δημιουργία εφαρμογών που εκτελούνται ομαλά σε όλες τις συσκευές. Το Flutter χρησιμοποιεί μια γλώσσα προγραμματισμού που ονομάζεται Dart, η οποία είναι καλή για τη γρήγορη εκτέλεση των εφαρμογών. Επιτρέπει επίσης στους προγραμματιστές να βλέπουν γρήγορα τις αλλαγές που κάνουν στην εφαρμογή τους χωρίς να χρειάζεται να περιμένουν ώρα για να ενημερωθεί. Συνολικά, το Flutter είναι ένα χρήσιμο εργαλείο για τη δημιουργία εφαρμογών που λειτουργούν καλά σε πολλές διαφορετικές συσκευές.

Το Flutter έχει πολλά εργαλεία που βοηθούν τις εφαρμογές να λειτουργούν ομαλά και να φαίνονται μοντέρνες. Αυτά τα εργαλεία ονομάζονται γραφικά στοιχεία (widgets) και μπορούν να χρησιμοποιηθούν για τη δημιουργία όλων των διαφορετικών τμημάτων μιας εφαρμογής, όπως κουμπιά, κινούμενα σχέδια και διατάξεις. Το κύριο μέρος μιας εφαρμογής Flutter είναι επίσης ένα widget, που ονομάζεται root widget. Ο τρόπος που λειτουργεί η εφαρμογή βασίζεται στον αντιδραστικό προγραμματισμό (reactive programming), όπου η εφαρμογή ενημερώνει μόνο τα μέρη που έχουν αλλάξει αντί να αναδημιουργεί ολόκληρο το γραφικό περιβάλλον που βλέπει ο χρήστης. Τα γραφικά στοιχεία είναι οργανωμένα σε μια ιεραρχία, με κάθε γραφικό στοιχείο να λαμβάνει τις πληροφορίες του από τον γονέα του. Αυτή η δομή φτάνει μέχρι το κύριο γραφικό στοιχείο της εφαρμογής.

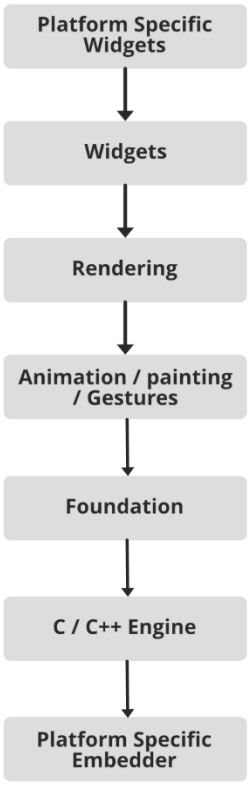
Τα βασικά χαρακτηριστικά της αρχιτεκτονικής Flutter συνοψίζονται παρακάτω:

Χειρονομίες (gestures): Τα γραφικά στοιχεία Flutter υποστηρίζουν την αλληλεπίδραση των χρηστών μέσω του ειδικού γραφικού στοιχείου GestureDetector. Το GestureDetector είναι ένα από τα μη ορατά γραφικά στοιχεία που ανιχνεύει τις κινήσεις του χρήστη (tap, swipe κ.λπ.) στο γραφικό στοιχείο, ενώ πολλά βασικά widgets το υποστηρίζουν.

Έννοια κατάστασης (concept of state): Τα γραφικά στοιχεία Flutter υποστηρίζουν την έννοια της κατάστασης, παρέχοντας ένα ειδικό widget που ονομάζεται StatefulWidget. Προκειμένου να μπορέσει ένα widget να υποστηρίξει τη λογική κατάστασης πρέπει να δημιουργηθεί από ένα StatefulWidget. Αντιθέτως, αν δε μας ενδιαφέρει η δυναμική ανανέωση του περιεχομένου του γραφικού στοιχείου,

τότε μπορούμε να του κληρονομήσουμε τα χαρακτηριστικά του StatelessWidget. Το StatefullWidget θα αναδημιουργηθεί, όποτε αλλάζει η εσωτερική του κατάσταση (state). Στο flutter, αυτή η διαδικασία είναι βελτιστοποιημένη, αφού εντοπίζονται οι διαφορές μεταξύ του παλαιού και του νέου γραφικού στοιχείου διεπαφής χρήστη και επανεμφανίζονται μόνο τις απαραίτητες αλλαγές.

Επίπεδα (layers): Η πιο σημαντική ιδέα του Flutter είναι η οργάνωσή του σε ξεκάθαρα επίπεδα μειωμένης πολυπλοκότητας. Κάθε επίπεδο συνδέεται με το αμέσως επόμενο επίπεδο, ενώ το πρώτο από αυτά στην ιεραρχία είναι το widget, το οποίο εξαρτάται από την πλατφόρμα (Android, iOS κ.λπ.). Το επόμενο στρώμα είναι εκείνο που περιλαμβάνει όλα τα γραφικά στοιχεία (widgets) του Flutter. Ακολουθεί το επίπεδο απόδοσης (rendering layer), το οποίο είναι ένα από τα δομικά στοιχεία χαμηλού επιπέδου και απεικονίζει τα πάντα στην εφαρμογή. Τα επίπεδα συνεχίζονται μέχρι τον βασικό κώδικα της πλατφόρμας.



Εικόνα 81: Flutter architecture of application

Το flutter δίνει πολλές δυνατότητες στον προγραμματιστή, όπως ένα ευρύ φάσμα πακέτων τρίτων που επιτρέπουν στον χρήστη να επεκτείνει τις λειτουργίες της εφαρμογής του. Επιπλέον, εκείνος χρειάζεται να γράψει μόνο μία κοινή βάση κώδικα για όλες τις πλατφόρμες (Android, iOS, Windows κ.λπ.). Το Flutter απαιτεί λιγότερους ελέγχους (testing), λόγω της κοινής βάσης κώδικα, αρκεί να δημιουργήσουμε αυτόματα tests κοινά για όλες τις πλατφόρμες. Η απλότητα του Flutter το καθιστά εξαιρετική επιλογή για γρήγορη ανάπτυξη εφαρμογών, ενώ παράλληλα η προσαρμοστικότητα και η επεκτασιμότητά του το καθιστούν ισχυρό εργαλείο. Τέλος, ο προγραμματιστής έχει πλήρη έλεγχο χάρη στα widgets και τη δομή που το flutter του παρέχει.

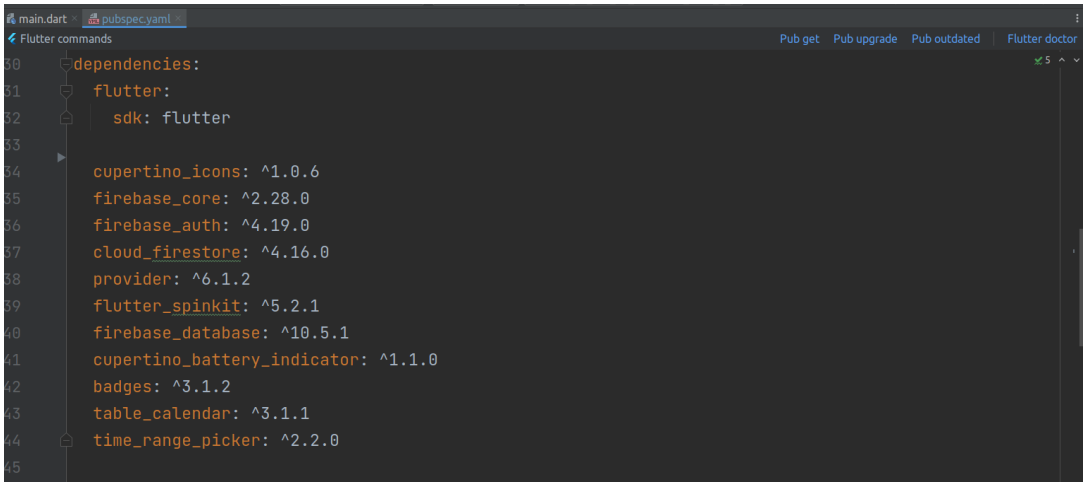
3.4.2 Android Studio

Η ανάπτυξη διεξάγεται στο Android Studio, το οποίο παρέχει στους προγραμματιστές ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) σχεδιασμένο ειδικά για εφαρμογές Android και iOS. Αρχικά, το IDE ενσωματώνει το Flutter SDK με λειτουργίες που διευκολύνουν την ολοκλήρωση κώδικα, τη διόρθωση σφαλμάτων και την αναδιαμόρφωση, επιτρέποντας στους προγραμματιστές να γράφουν καθαρό, διατηρήσιμο κώδικα. Παρέχει μια τεράστια συλλογή API βιβλιοθηκών και εργαλείων που μπορούν να χρησιμοποιηθούν για τη δημιουργία πλούσιων σε χαρακτηριστικά και ελκυστικών εφαρμογών. Πολύ σημαντική είναι η δυνατότητα προσομοίωσης της λειτουργίας της εφαρμογής σε προσομοιωτές, τόσο για Android όσο και για iOS, καθώς και σε ένα ευρύ φάσμα μοντέλων για κινητά, καθένα από τα οποία έχει τις δικές του διαστάσεις και δυνατότητες.

Όλες αυτές οι ιδιότητες, εκτός από την εξοικονόμηση χρόνου, βοηθούν τους προγραμματιστές να δομήσουν σωστά τα έργα τους, να αποφύγουν περιττά λάθη και να χρησιμοποιήσουν τα απαραίτητα σχόλια για να βοηθήσουν άλλους προγραμματιστές να ενταχθούν στις εφαρμογές τους και να τις αναπτύξουν σύμφωνα με όλα τα απαραίτητα στον κώδικα τους.

3.4.3 Flutter Packages

Ένα άλλο πολύ σημαντικό χαρακτηριστικό που αξίζει να αναφερθεί είναι τα χιλιάδες «πακέτα» (plugins) που διαθέτει το flutter. Πέρα από τα βασικά στοιχεία που μπορούν να αξιοποιήσουν οι προγραμματιστές στην «αρχική» έκδοση του Flutter, είναι πολύ πιθανό ότι για να λειτουργήσει η εφαρμογή, θα χρειαστεί να προσθέσει επιπλέον λειτουργίες υλοποιημένες από τρίτους. Για να προστεθούν στην εφαρμογή, αρκεί να εγκατασταθούν οι απαραίτητες βιβλιοθήκες. Όλα τα διαθέσιμα «πακέτα» που έχουν εγκατασταθεί και μπορούν να χρησιμοποιηθούν βρίσκονται στο Pub.dev. Το Pub.dev είναι ο διαχειριστής πακέτων για τη γλώσσα προγραμματισμού Dart, η οποία περιλαμβάνει επαναχρησιμοποιήσιμες βιβλιοθήκες για προγράμματα Flutter, AngularDart και regular dart. Για να ενεργοποιηθεί ένα "πακέτο", απλώς εισάγεται στο αρχείο “pubspec.yaml” το όνομά του στην ενότητα εξαρτήσεων (dependencies) και δίπλα του εισάγεται η επιθυμητή έκδοση. Έπειτα, με την επιλογή “Pub get” εγκαθίσταται το νέο πακέτο και μπορεί να γίνει η χρήση των βιβλιοθηκών του.



Εικόνα 82: pubspec.yaml dependencies

3.4.4 Κώδικας Dart και App Screens

- main.dart

Το αρχείο “main.dart” περιέχει την main() συνάρτηση, η οποία εκτελείται κατά την εκκίνηση της εφαρμογής. Η συνάρτηση αυτή, αρχικοποιεί τη βάση firebase και αναμένει τη σύνδεση της εφαρμογής σε αυτή. Μόνο μετά την ολοκλήρωση της σύνδεσης εκτελείται η runApp(), η οποία εμφανίζει το root widget MyApp(). Όπως φαίνεται και από τον ορισμό του, το γραφικό στοιχείο MyApp() είναι ένα StatelessWidget, επομένως δεν έχει κατάσταση.

Γίνεται χρήση της συνάρτησης build(), η οποία επιστρέφει StreamProvider<User?>.value{} και έχει παιδί το γραφικό στοιχείο MaterialApp(). Γενικώς, το StreamProvider χρησιμοποιείται για να «ακούει» για νέα δεδομένα (streams) που έρχονται ή πηγαίνουν στη βάση. Εδώ, γίνεται χρήση του StreamProvider για να δούμε αν έχει συνδεθεί ο χρήστης ή όχι στην εφαρμογή, προκειμένου να του εμφανίζεται η κατάλληλη οθόνη.

Το όρισμα home έχει την τιμή Wrapper() widget, το οποίο προβάλλει είτε την οθόνη της πιστοποίησης ή την βασική οθόνη της εφαρμογής, ανάλογα αν έχει ολοκληρωθεί η σύνδεση (ή εγγραφή) του χρήστη.

main.dart

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return StreamProvider<User?>.value(
      value: AuthService().user,
      initialData: null,
      child: MaterialApp(
        theme: ThemeData(
          primaryColor: Colors.cyan, // Background color for app bars, toolbars, etc.
          scaffoldBackgroundColor: Colors.white, // Background color for scaffolds (e.g., background color of pages)
        ),  // ThemeData
        home: Wrapper(),
        ),  // MaterialApp
    );  // StreamProvider.value
  }
```

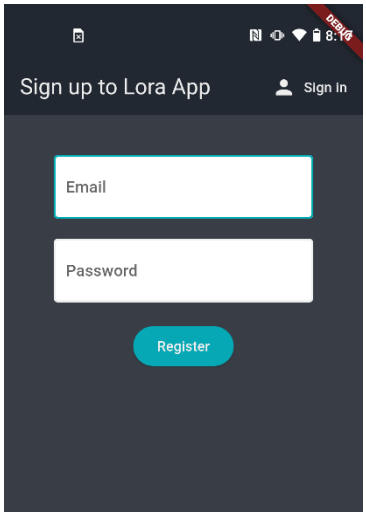
Εικόνα 83: main.dart

- authenticate.dart

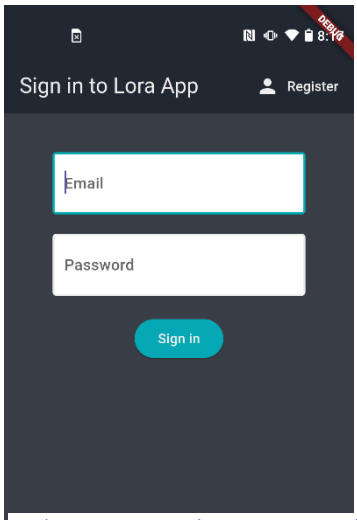
Η οθόνη πιστοποίησης μπορεί να επιστρέφει δύο widgets, ανάλογα αν ο χρήστης επιθυμεί να συνδεθεί ή να κάνει εγγραφή. Στην πρώτη περίπτωση, επιστρέφεται το SignIn() γραφικό στοιχείο, ενώ στη δεύτερη το Register(). Το widget Authenticate() αυτή τη φορά είναι StatefulWidget, επομένως μπορούμε να αλλάξουμε την κατάστασή του με τη συνάρτηση setState(), η οποία όπως φαίνεται παρακάτω αντιστρέφει την Boolean μεταβλητή showSignIn προκειμένου να εναλλάσσονται οι δύο οθόνες κατά την εκτέλεσή της.

```dart
class Authenticate extends StatefulWidget {
  const Authenticate({super.key});

  @override
  State<Authenticate> createState() => _AuthenticateState();
}

class _AuthenticateState extends State<Authenticate> {

  bool showSignIn = true;
  void toggleView() {
    setState(() {
      showSignIn = !showSignIn;
    });
  }

  @override
  Widget build(BuildContext context) {
    if (showSignIn) {
      return SignIn(toggleView: toggleView);
    } else {
      return Register(toggleView: toggleView);
    }
  }
}
```

Εικόνα 84: authenticate.dart



Εικόνα 85: Σύνδεση στην εφαρμογή

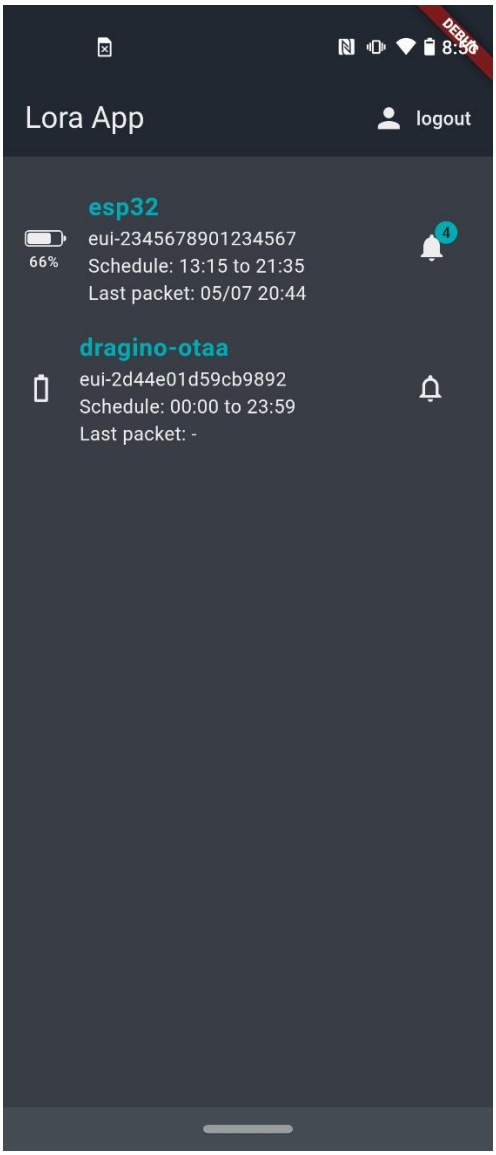


Εικόνα 86: Εγγραφή στην εφαρμογή

Στις οθόνες σύνδεσης και εγγραφής υπάρχει η δυνατότητα εναλλαγής από τη μια λειτουργία στην άλλη με την επιλογή του πάνω δεξιά κουμπιού. Τόσο η εγγραφή όσο και η σύνδεση έρχονται με ελέγχους στα στοιχεία που εισάγει ο χρήστης. Πιο συγκεκριμένα, γίνεται έλεγχος για έγκυρη μορφή email και για μήκος κωδικού προκειμένου να ενισχυθεί η ασφάλεια του συστήματος. Σε περίπτωση μη τήρησης των κριτηρίων αυτών, ο χρήστης ενημερώνεται με κατάλληλο μήνυμα.

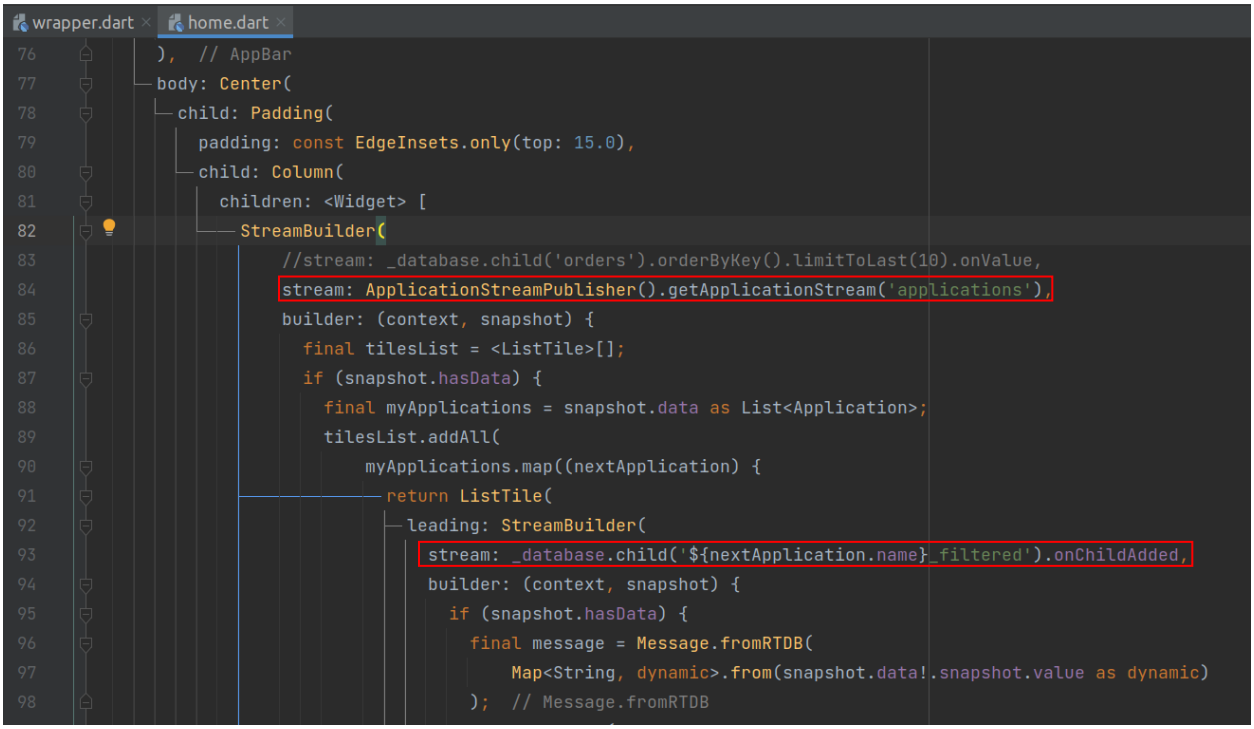
- home.dart

Το widget αυτό, αποτελεί τη βασική οθόνη της εφαρμογής, από όπου ο χρήστης μπορεί να δει όλες τις συνδεδεμένες συσκευές (end devices), ενώ παράλληλα και κάποιες βασικές πληροφορίες για αυτές. Επιπλέον, μπορεί να δει τις ειδοποιήσεις της κάθε συσκευής καθώς και να αποσυνδεθεί από την εφαρμογή.

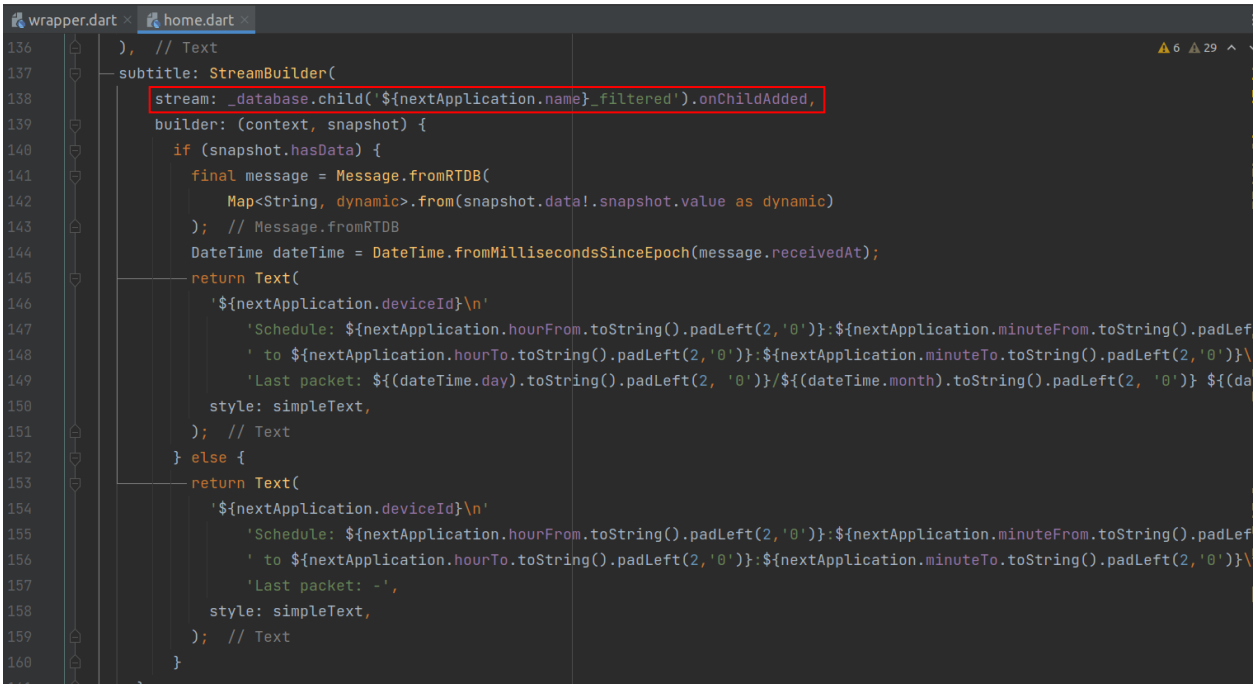


Εικόνα 87: Κεντρική οθόνη εφαρμογής

Πιο συγκεκριμένα, από την κεντρική οθόνη μπορούμε να δούμε το ποσοστό υπολειπόμενης μπαταρίας, τον κωδικό συσκευής dev-eui, την ώρα λήψης τελευταίου πακέτου από την συσκευή καθώς και το ωράριο λειτουργίας της συσκευής (εδώ ωράριο εντοπισμού κίνησης).



Εικόνα 88: home.dart λίστα συσκευών

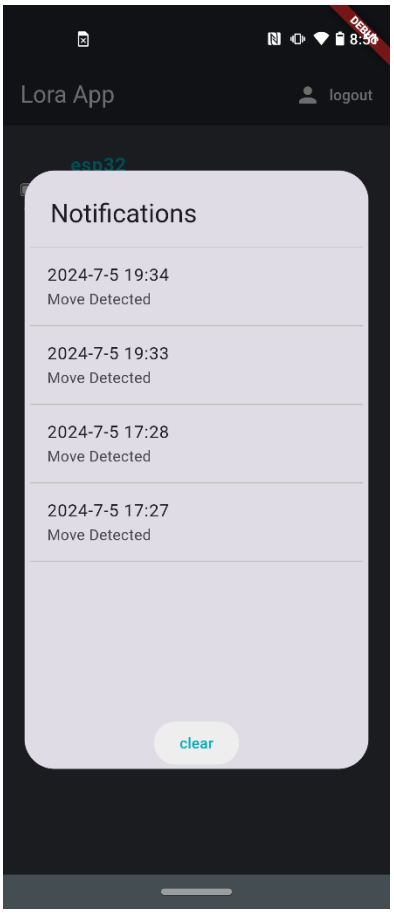


Εικόνα 89: home.dart πληροφορίες συσκευής

Από πλευρά κώδικα, όλα τα δεδομένα που απεικονίζονται στην αρχική οθόνη αποτελούν widget που κτίζονται με βάση τις αλλαγές που συμβαίνουν στη βάση δεδομένων σε πραγματικό χρόνο. Το ίδιο το Home() γραφικό στοιχείο είναι StatelessWidget, παρόλα αυτά από τη στιγμή που έχουμε StreamBuilder widgets (πρακτικά listeners στη βάση) μπορούμε να ανανεώνουμε δυναμικά το περιεχόμενο.

Όπως φαίνεται στις παραπάνω εικόνες, το γραφικό στοιχείο StreamBuilder(), παίρνει ως όρισμα ένα stream, το οποίο αντιστοιχεί σε μια συγκεκριμένη συλλογή της real-time βάσης. Με το που προκύψει κάποια αλλαγή στη συγκεκριμένη συλλογή, τότε η συνάρτηση που ορίζεται στο όρισμα “builder” του StreamBuilder εκτελείται και επανεμφανίζει το πλέον ανανεωμένο περιεχόμενο. Η βασική ιδέα είναι ότι δεν χρησιμοποιείται μνήμη του κινητού τηλεφώνου για αποθήκευση μεταβλητών ή τιμών, αλλά όλα προέρχονται από τη βάση δεδομένων και απλώς απεικονίζονται με φιλικό τρόπο προς στο χρήστη στο user interface της εφαρμογής.

Από τη βασική οθόνη της εφαρμογής, υπάρχει η δυνατότητα μετάβασης στις ειδοποιήσεις με απλό tap στο εικονίδιο ειδοποιήσεων.



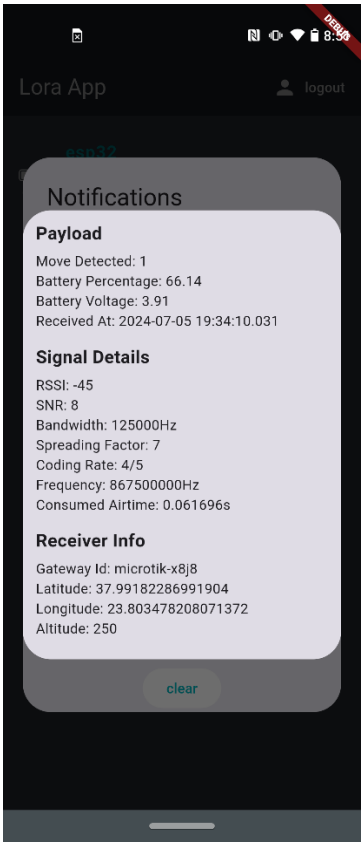
Εικόνα 91: Pop-up ειδοποιήσεων

```dart
home.dart
child: Icon(
    Icons.notifications,
  color: Color.fromRGBO(238, 238, 238, 1.0),
), // Icon
onPressed: () {
  showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          backgroundColor: Color.fromRGBO(238, 238, 238, 1.0),
          contentPadding: EdgeInsets.fromLTRB(5.0, 0.0, 5.0, 0.0),
          insetPadding: EdgeInsets.fromLTRB(20.0, 0.0, 20.0, 0.0),
          title: Text('Notifications'),
          content: SizedBox(
              width: 400.0,
              height: 500.0,
              child: ApplicationImportant(applicationName: nextApplication.name)),
        ); // AlertDialog
```

Εικόνα 90: home.dart pop-up ειδοποιήσεων

Με τη μετάβαση στο pop-up ειδοποιήσεων, ο χρήστης έχει τη δυνατότητα να δει ποιες χρονικές στιγμές και ημερομηνίες εντοπίστηκε κάποια κίνηση από τον pir αισθητήρα, ενώ μπορεί να καθαρίσει το ιστορικό των ειδοποιήσεων με την επιλογή “clear”.

Όπως φαίνεται και στον παραπάνω κώδικα, το εικονίδιο “notifications” όταν πατηθεί, εκτελείται το onPressed όρισμα, δηλαδή η ορισμένη ανώνυμη συνάρτηση, η οποία επιστρέφει το AlertDialog (pop-up window) γραφικό στοιχείο.



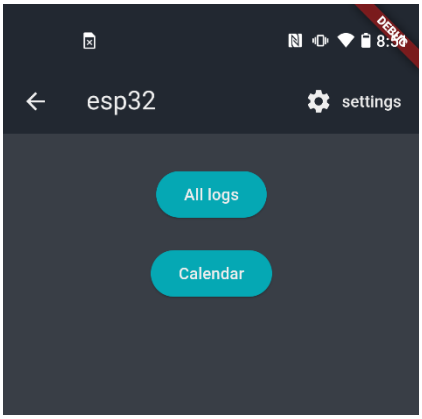
Εικόνα 92: Πληροφορίες ειδοποίησης

Με long press πάνω σε μια συγκεκριμένη ειδοποίηση ανοίγει ένα δεύτερο παράθυρο pop-up, που περιέχει περισσότερες πληροφορίες για το συγκεκριμένο μήνυμα. Πιο αναλυτικά, έχουμε πληροφορίες για το ωφέλιμο φορτίο (ακριβή ώρα λήψης, ποσοστό και τάση μπαταρίας), για το σήμα λήψης (τιμές RSSI, SNR, Consumed Airtime κλπ) και για την κεραία λήψης (Gateway Id, τοποθεσία).

Πέρα από τις ειδοποιήσεις, ο χρήστης μέσω της αρχικής οθόνης μπορεί να επιλέξει κάποια συγκεκριμένη συσκευή από τη λίστα προκειμένου να εισέλθει στο μενού της συγκεκριμένης συσκευής. Αυτό γίνεται με πάτημα πάνω στην επιθυμητή συσκευή.

- application_homepage.dart

Το γραφικό στοιχείο ApplicationHomepage είναι Stateless και περιέχει τις βασικές επιλογές για τη συσκευή που επιλέχθηκε. Πιο αναλυτικά, περιέχει δύο κουμπιά που παραπέμπουν σε δύο άλλες οθόνες και μια επιλογή ρυθμίσεων πάνω δεξιά.



Εικόνα 93: μενού εφαρμογής

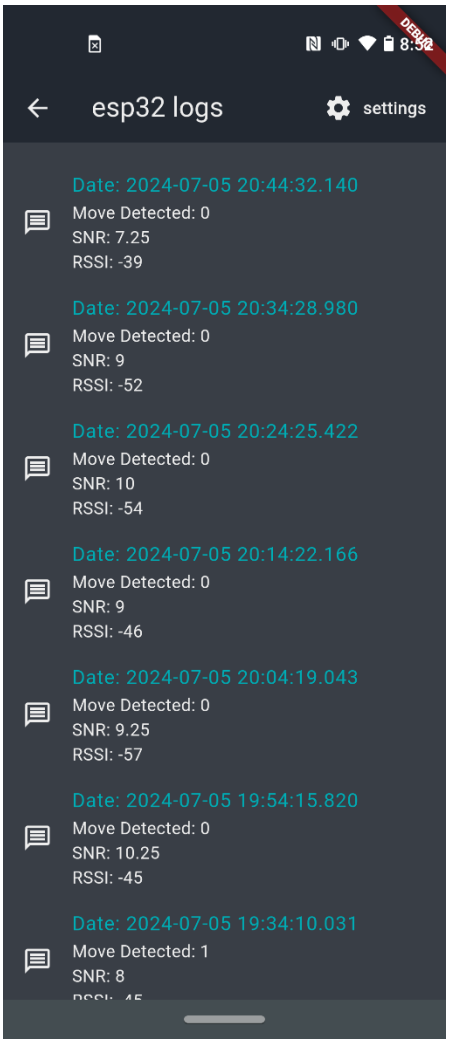
- message_details.dart

Με την επιλογή “All logs” μεταβαίνουμε σε μια οθόνη που περιέχει μια λίστα με όλα τα μηνύματα που έχει στείλει το end device και είναι αποθηκευμένα στη βάση. Με επιλογή πάνω σε κάποια συγκεκριμένο μήνυμα, μπορούμε να λάβουμε μια λεπτομερή περιγραφή του μηνύματος παρόμοια με τις πληροφορίες ειδοποίησης που δείχθηκαν παραπάνω.

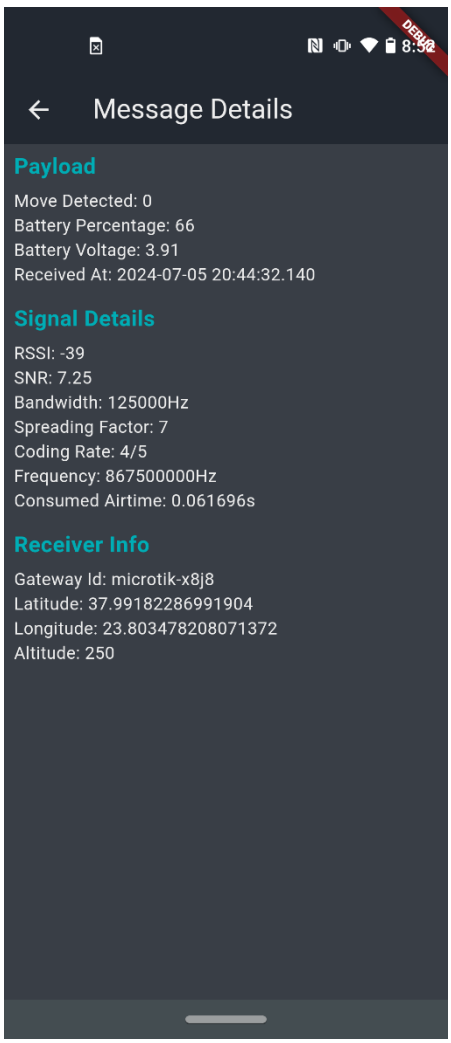
- calendar.dart

Από την άλλη, με την επιλογή “Calendar”, μπορούμε να μεταβούμε σε ένα ημερολόγιο που περιέχει αναλυτικά τις μέρες και ώρες που εντοπίστηκε κίνηση από τον pir αισθητήρα. Το ημερολόγιο αυτό, όπως και όλα τα άλλα στοιχεία της εφαρμογής, ενημερώνονται σε πραγματικό χρόνο και κρατούν τα δεδομένα στη βάση μέχρι ο χρήστης να επιλέξει να τα διαγράψει. Συνεπώς, το ημερολόγιο αποτελεί ένα φιλικό προς το χρήστη περιβάλλον, το οποίο κρατάει αναλυτικό ιστορικό με ευανάγνωστο τρόπο.

Το πακέτο που χρησιμοποιήθηκε για το ημερολόγιο είναι το table_calendar και συγκεκριμένα η έκδοση 3.1.1 [15]. Το πακέτο αυτό, παρέχει πλήρως τροποποιήσιμο περιβάλλον χρήστη, ενώ παράλληλα υποστηρίζει πολλαπλές μορφές (μηνιαίο, εβδομαδιαίο ημερολόγιο). Ο χρήστης μπορεί να επιλέξει μια συγκεκριμένη μέρα ή ακόμη και ένα διάστημα ημερών και να δει αναλυτικά πότε εντοπίστηκν κίνησεις από τον αισθητήρα. Προκειμένου να αξιοποιηθούν οι έτοιμες βιβλιοθήκες για το εργαλείο αυτό, αρκεί να εισάγουμε στο “pubspec.yaml” αρχείο την εξάρτηση table_calendar: ^3.1.1 και να ενημερώσουμε το Pub.dev με την επιλογή “Pub get” όπως έχει εξηγηθεί και παραπάνω.



Εικόνα 94: Λίστα μηνυμάτων

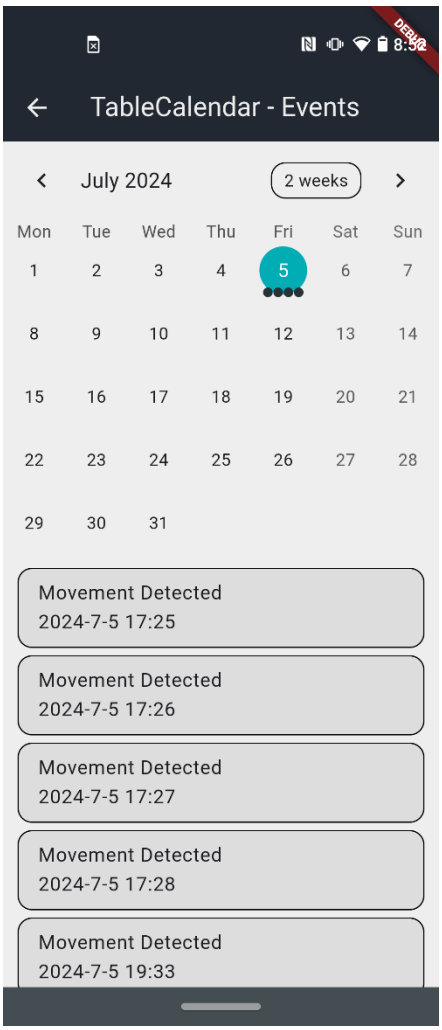


Εικόνα 95: Πληροφορίες μηνύματος

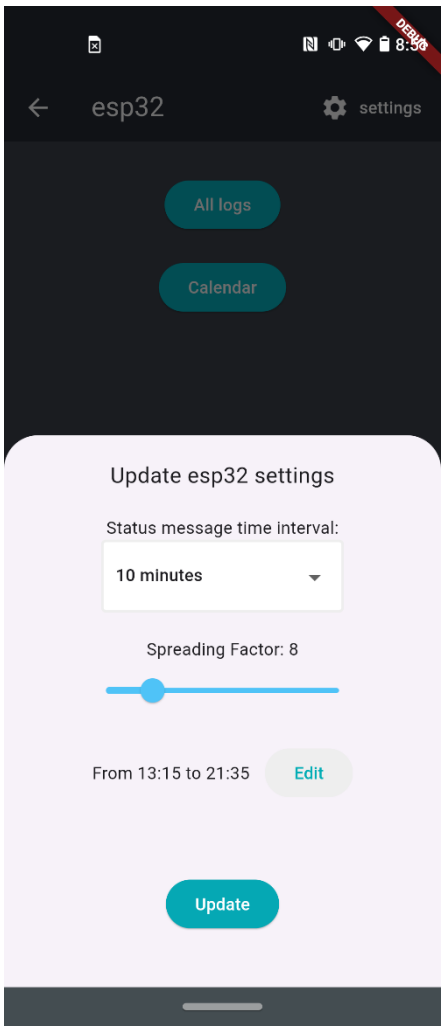
- downlink_settings.dart

Με την επιλογή “settings” πάνω δεξιά στο μενού συσκευής, μπορούμε να ρυθμίσουμε το χρόνο μεταξύ μηνυμάτων κατάστασης, το spreading factor καθώς και το ωράριο λειτουργίας του end device. Οι επιλογές αποθηκεύονται στη συλλογή “applications” της βάσης δεδομένων και μπορούν τόσο να διαβαστούν όσο και να ενημερωθούν από την εφαρμογή.

Στην πραγματικότητα, όταν πραγματοποιείται μια αλλαγή στις ρυθμίσεις του end device μέσω της εφαρμογής, αλλάζει η αντίστοιχη εγγραφή στην real-time βάση. Έπειτα ο Node-RED σέρβερ, ο οποίος ακούει για αλλαγές στη συγκεκριμένη συλλογή, αντλεί και επεξεργάζεται τις αλλαγές και στέλνει κατάλληλο μήνυμα στον the things server μέσω πρωτοκόλλου MQTT. Ο εξυπηρετητής δικτύου λαμβάνει το μήνυμα και προετοιμάζει το downlink για να σταλεί την επόμενη δυνατή στιγμή. Από τη στιγμή που οι συνδεδεμένες συσκευές είναι τύπου A, το downlink μήνυμα θα σταλεί την επόμενη φορά που θα λάβουμε μήνυμα από το end device και εκείνη τη στιγμή θα λάβει και τις τελευταίες ενημερώσεις που όρισε ο χρήστης μέσω της εφαρμογής.



Εικόνα 96: Ημερολόγιο μηνυμάτων



Εικόνα 97: Ρυθμίσεις downlink

downlink_settings.dart

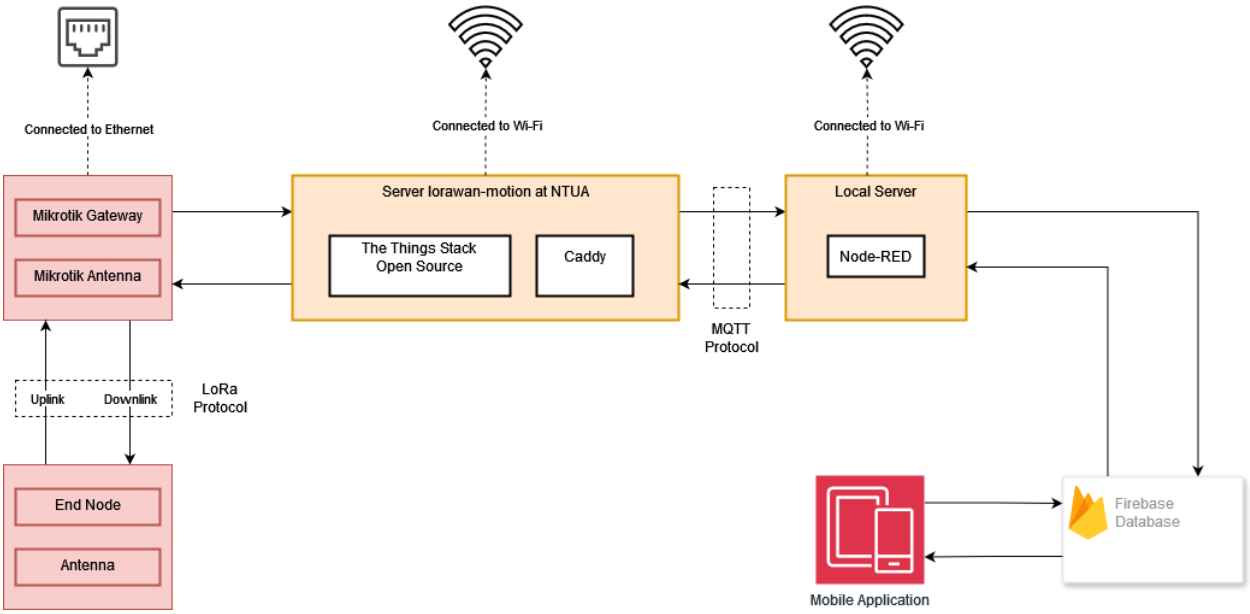
```
137        ElevatedButton(
138          style: ButtonStyle(
139            backgroundColor: MaterialStateProperty.all(Color.fromRGBO(0, 173, 181, 1.0)),
140          ),  // ButtonStyle
141          child: const Text(
142            'Update',
143            style: TextStyle(color: Colors.white),
144          ),  // Text
145          onPressed: () async {
146            if(_formKey.currentState!.validate()) {
147              final dbRef = _database.child('applications/${currentApplication.name}');
148              await dbRef.update({
149                'status_interval': _statusInterval ?? currentApplication.statusInterval,
150                'spreading_factor':  _spreadingFactor ?? currentApplication.spreadingFactor,
151                'hour_from': _hourFrom ?? currentApplication.hourFrom,
152                'minute_from': _minuteFrom ?? currentApplication.minuteFrom,
153                'hour_to': _hourTo ?? currentApplication.hourTo,
154                'minute_to': _minuteTo ?? currentApplication.minuteTo,
155              });
156              Navigator.pop(context);
157            }
158          },
```

Εικόνα 98: downlink.dart ενημέρωση ρυθμίσεων

Κεφάλαιο 4: Σύνοψη

4.1 Σφαιρική εικόνα

Η εφαρμογή που αναπτύχθηκε στη παρούσα διπλωματική εκμεταλλεύεται πολλές και διαφορετικές τεχνολογίες προκειμένου να εκμεταλλευτεί τα προτερήματα της κάθε τεχνολογίας, διατηρώντας ταυτόχρονα μια φιλική εμπειρία προς τον χρήστη. Παρακάτω φαίνεται ένα διάγραμμα που εξηγεί σε υψηλό επίπεδο (high level) τη συνεργασία των διαφόρων στοιχείων της εφαρμογής και την ανταλλαγή μηνυμάτων μεταξύ τους.



Εικόνα 99: High level overview of project

Η εφαρμογή μπορεί να αποτελέσει μια σύγχρονη και μοντέρνα λύση σε συστήματα ασφαλείας, από τη στιγμή που ενημερώνει σε πραγματικό χρόνο τον χρήστη στο κινητό του τηλέφωνο. Ο τελικός κόμβος έχει τη δυνατότητα να τοποθετηθεί σε οποιοδήποτε σημείο, δίχως να υπάρχει η ανάγκη για ηλεκτρική τροφοδοσία ή σύνδεση Wi-Fi. Επομένως, με το πρωτόκολλο LoRa μπορούμε να έχουμε ασύρματο εντοπισμό ύποπτων κινήσεων σε υπόγεια, ταράτσες, πάρκινγκ και απομακρυσμένες περιοχές, όπως δοκιμάστηκε και στην πράξη στο campus του Εθνικού Μετσόβιου Πολυτεχνείου. Σε αντίθεση με άλλα πρωτόκολλα μακριάς εμβέλειας, πετύχαμε μεγάλη αυτονομία και εξοικονόμηση ενέργειας.

Το mobile app που αναπτύχθηκε, είναι απλό στη χρήση και παρέχει τις απαραίτητες δυνατότητες σε έναν απλό χρήστη τόσο να λάβει μηνύματα όσο και να ρυθμίσει τα end devices. Συνεπώς, δε χρειάζεται κάποια εξειδικευμένη γνώση στο δίκτυο LoRa ή πρόσβαση στους servers προκειμένου ο τελικός χρήστης να λάβει τα μέγιστα από την εφαρμογή.

4.2 Δοκιμές εμβέλειας

4.2.1 Δημοσιεύσεις και δοκιμές τρίτων

Ένα ίσως από τα πιο σημαντικά ζητήματα της εφαρμογής είναι η κάλυψη όσο το δυνατόν μεγαλύτερης εμβέλειας επικοινωνίας μεταξύ της κεραίας και του end node. Στη θεωρία, το LoRa πρωτόκολλο μπορεί να μεταδώσει πληροφορία σε απόσταση πολλών χιλιομέτρων (>10 χιλιόμετρα). Παρόλα αυτά, σε μια πραγματική εφαρμογή, όπως στη περίπτωσή μας που το δίκτυο LoRa στήθηκε εντός του campus του Εθνικού Μετσόβιου Πολυτεχνείου, τα πράγματα διαφέρουν.

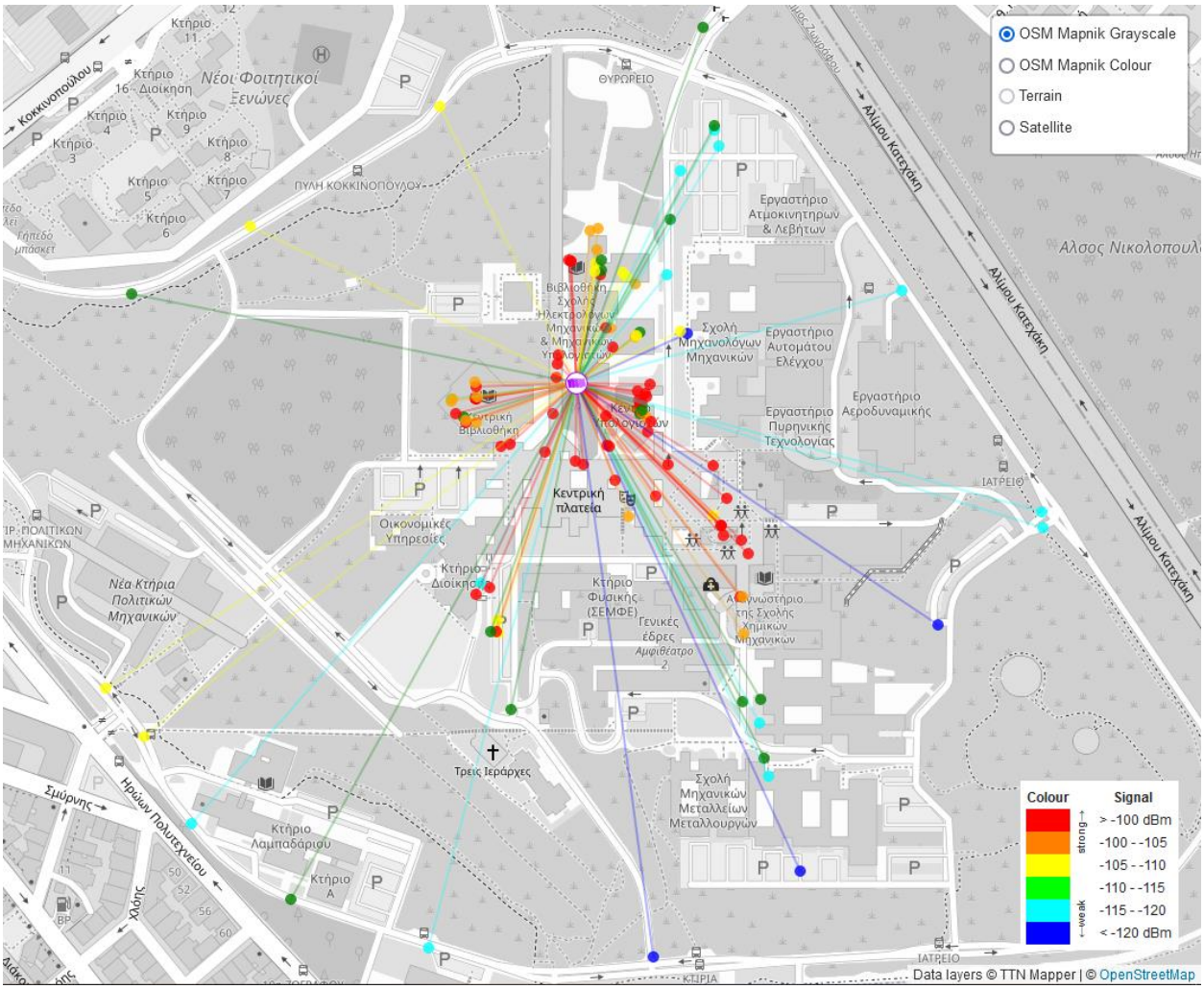
Έχουν πραγματοποιηθεί πολλές δοκιμές για την πραγματική εμβέλεια του πρωτοκόλλου LoRa. Μια από αυτές (πηγή [16]) συμπέρανε πως μπορούμε να έχουμε αξιόπιστη ανταλλαγή μηνυμάτων μεταξύ εξυπηρετητή και τελικής συσκευής σε απόσταση το πολύ 5 χιλιομέτρων. Στη συγκεκριμένη δοκιμή, η κεραία λήψης τοποθετήθηκε σε ταράτσα κτιρίου (περίπου 10 μέτρα υψόμετρο) και ήταν 12 dBi. Η ευαισθησία του gateway ήταν -139 dBm και η συχνοτική μπάντα στα 868 MHz, ενώ ο καιρός χωρίς αέρα και βροχή στους 25 βαθμούς Κελσίου. Άλλες δοκιμές έδειξαν πως μέχρι σε απόσταση 3 χιλιομέτρων μπορούμε να έχουμε καλή αξιοπιστία, λαμβάνοντας υπόψη μη ιδανικές συνθήκες όπως καιρικά φαινόμενα ή ενδιάμεσα εμπόδια.

Σύμφωνα με μια δημοσίευση [17], σε κλειστούς χώρους (πχ μέσα σε κτίρια) η εμβέλεια της διαμόρφωσης LoRa πέφτει δραματικά. Εμπόδια, όπως ξύλινες πόρτες ή τοίχοι από τούβλα μπορούν να μειώσουν πολύ την ισχύ του σήματος και συνεπώς την εμβέλεια ακόμα και σε πολύ μικρές αποστάσεις.

4.2.2 Δοκιμές στην εφαρμογή

Προκειμένου να αποφανθούμε για την εμβέλεια της δικής μας εφαρμογής με τα χαρακτηριστικά και τις ιδιαιτερότητές της δεν αρκεί μόνο η στήριξη στη θεωρία και σε δημοσιεύσεις άλλως, αλλά κρίνεται απαραίτητη η δοκιμή με το δικό μας εξοπλισμό. Με την αξιοποίηση της εφαρμογής TTN Mapper [18], πραγματοποιήθηκαν τεστ εμβέλειας του end node. Το mobile app αυτό χρησιμοποιεί το GPS του κινητού τηλεφώνου προκειμένου να αντιλαμβάνεται τη θέση του end node κάθε φορά που αποστέλλεται ένα μήνυμα. Πιο αναλυτικά, ο χρήστης της εφαρμογής έχει μαζί του το end device μαζί με το κινητό του τηλέφωνο και κάθε φορά που πραγματοποιείται αποστολή μηνύματος LoRa από το end node, επισυνάπτεται στο μήνυμα η τοποθεσία του GPS του κινητού τηλεφώνου. Συνεπώς με βάση την γεωγραφική τοποθεσία και τα χαρακτηριστικά του σήματος SNR και RSSI μπορεί να δημιουργηθεί ένας χάρτης που απεικονίζει τόσο την εμβέλεια όσο και την ποιότητα των μηνυμάτων. Ακόμα, στην εφαρμογή ορίζεται η θέση της κεραίας λήψης, δηλαδή του gateway.

Παρακάτω, απεικονίζεται η εμβέλεια του esp32 end node (που έχει αναλυθεί αναλυτικά παραπάνω) στο campus του Εθνικού Μετσόβιου Πολυτεχνείου. Ο παρακάτω χάρτης μπορεί να βρεθεί και στο σύνδεσμο https://ttnmapper.org/devices/?device=dragino-otaa&startdate=2023-12-27&enddate=2023-12-27&gateways=on&lines=on&points=on



Εικόνα 100: Εμβέλεια κόμβου esp32

4.3 Προτάσεις για μελλοντικές επεκτάσεις

Παρακάτω παρουσιάζονται κάποιες προτάσεις και βελτιώσεις που μπορούν να υλοποιηθούν σε μεταγενέστερο στάδιο, προκειμένου η εφαρμογή να γίνει πιο αποδοτική και χρήσιμη για τον τελικό χρήστη:

- Επίτευξη μεγαλύτερης αυτονομίας μπαταρίας με την αφαίρεση «περιττών» στοιχείων του μικροελεγχτή, όπως leds, voltage regulators κλπ. Ακόμη, θα μπορούσε μελλοντικά να δημιουργηθεί συγκεκριμένη πλακέτα για την εξυπηρέτηση του συγκεκριμένου σκοπού μειώνοντας δραματικά τις ενεργειακές απαιτήσεις
- Τοποθέτηση μεγαλύτερης πιο ισχυρής κεραίας στον πομπό, έτσι ώστε να έχουμε αξιόπιστη επικοινωνία σε μεγαλύτερες αποστάσεις και εντός κτιρίων. Για το σκοπό αυτό, θα μπορούσαμε επίσης να προσθέσουμε και περισσότερα gateways

- Υλοποίηση συστήματος εντοπισμού γεωγραφική θέσης χωρίς τη χρήση GPS, αλλά με τη μέθοδο τριγωνισμού και μετρήσεις time-on-air των μηνυμάτων. Προκειμένου να πραγματοποιηθεί το χαρακτηριστικό αυτό, θα πρέπει να έχουμε τουλάχιστον 3 gateways
- Ενσωμάτωση ηλιακών panel στην τελική συσκευή, έτσι ώστε να έχουμε πλήρη αυτονομία ενέργειας
- Ενσωμάτωση νέων αισθητήρων στο δίκτυο για περιβαλλοντική παρακολούθηση (ποιότητα αέρα και κλιματικών συνθηκών), για έξυπνη διαχείριση πόρων (έξυπνος φωτισμός, διαχείριση απορριμμάτων, θέσεις πάρκινγκ)
- Επέκταση της εφαρμογής σε πιο γενικού σκοπού. Κάθε φοιτητής του Πολυτεχνείου με κατάλληλα credentials να μπορεί να έχει πρόσβαση μέσω της εφαρμογής στο δίκτυο LoRa και να μπορεί εύκολα να συνδέσει (register) δικές του συσκευές, δίχως την ανάγκη για πρόσβαση στον The Things Server

Κεφάλαιο 5: Κατακλείδα

Σκοπός της παρούσας διπλωματικής ήταν ο σχεδιασμός και η υλοποίηση αστικού δικτύου LoRaWAN για εφαρμογές IoT στο campus του Εθνικού Μετσόβιου Πολυτεχνείου. Επιπλέον, στόχος ήταν η ανάπτυξη εφαρμογής κινητού τηλεφώνου που να επιτρέπει στο χρήστη να βλέπει τα μηνύματα που λαμβάνουν τα gateways σε πραγματικό χρόνο καθώς και να λαμβάνει αντίστοιχες ειδοποιήσεις για αυτά. Πιο συγκεκριμένα, η εφαρμογή που αναπτύχθηκε εντάσσεται στη κατηγορία εφαρμογών ασφάλειας, από τη στιγμή που εντοπίζει κινήσεις ανθρώπων σε κάποιο συγκεκριμένο χώρο ένα ορισμένο χρονικό διάστημα.

Η υλοποίηση της εφαρμογής αυτής με το πρωτόκολλο LoRa λύνει πολλά από τα προβλήματα που υπάρχουν σε αντίστοιχες εφαρμογές, Αρχικά η εμβέλειά του, επιτρέπει την τοποθέτηση των συσκευών-αισθητήρων μακριά από τα gateways, ενώ παράλληλα μπορούμε να έχουμε μόνο ένα gateway για πολλές συσκευές. Το πετυχαίνουμε αυτό, έχοντας ταυτόχρονα τη δυνατότητα και για αμφίδρομη επικοινωνία. Επιπλέον, η αυτονομία που πετυχαίνουμε με τη διαμόρφωση LoRa είναι πολύ μεγάλη σε σχέση με άλλα πρωτόκολλα, συνεπώς η αντικατάσταση των μπαταριών χρειάζεται να γίνεται πιο σπάνια. Βασικό χαρακτηριστικό της τελικής εφαρμογής είναι επεκτασιμότητα που έχει, αφού μετά το στήσιμο του δικτύου LoRa και την ανάπτυξη της εφαρμογής, μπορούν πολύ εύκολα να συνδεθούν με παρόμοιο τρόπο κι άλλες συσκευές εμπλουτίζοντας το δίκτυο.

Παρακάτω αναφέρονται τα βασικά στοιχεία των όσο επιτεύχθηκαν από την παρούσα διπλωματική και υλοποίηση:

- Εκτενής μελέτη και κατανόηση της διαμόρφωσης LoRa
- Στήσιμο The Things Open Source Server
- Προγραμματισμός esp32 και arduino μικροελεγχτών
- Αξιοποίηση PIR αισθητήρα για ανίχνευση κίνησης, battery gauge για καθορισμό υπολειπόμενης τάσης μπαταρίας και άλλων αισθητήρων (dht22, αισθητήρας καπνού)
- Αποστολή μηνυμάτων από τα end devices προς τα gateways
- Αποστολή μηνυμάτων από τα gateways προς τα end devices
- Στήσιμο backend server και χρήση firebase βάσης δεδομένων
- Ανάπτυξη εφαρμογής κινητού τηλεφώνου Cross Platform (Android/iOS)

Η κεντρική συμβολή της παρούσας διπλωματικής είναι το στήσιμο του αστικού δικτύου LoRaWAN στην Πολυτεχνειούπολη καθώς και η ανάπτυξη μιας βασικής μεν αλλά πλήρως λειτουργικής εφαρμογής. Η εργασία αυτή μπορεί να αποτελέσει βάση για πιο σύνθετες εφαρμογές στον τομέα Internet of Things, αφού συνδυάζει πολλές σύγχρονες τεχνολογίες. Αποτελεί μια καινοτομία στον τομέα της ασφάλειας, από τη στιγμή που τα συνήθη συστήματα εντοπισμού κίνησης περιορίζονται σε μικρές αποστάσεις από την πηγή τροφοδοσίας και παράλληλα δεν υποστηρίζουν δυνατότητα ειδοποίησης του τελικού χρήστη σε άμεσο χρόνο, σε αντίθεση με την πρόταση της παρούσας διπλωματικής, η οποία λύνει τα προβλήματα αυτά.

Βιβλιογραφία

[1] Duarte, F. (2024, February 19). *Number of IOT devices (2024)*. Exploding Topics. https://explodingtopics.com/blog/number-of-iot-devices

[2] Internet of things: Τι είναι και πώς αλλάζει τη ζωή μας. https://fsdet.dmst.aueb.gr/index.php/2023/02/15/internet-of-things/

[3] Hinz, B. (2022, February 18). *Free lorawan book*. LoRa Alliance®. https://lora-alliance.org/lorawan-news/free-lorawan-book/

[4] Lie, R. *Lorawan*. Mobilefish.com - LoRa/LoRaWAN tutorial. https://www.mobilefish.com/developer/lorawan/lorawan_quickguide_tutorial.html

[5] *Lorawan®*. The Things Network. https://www.thethingsnetwork.org/docs/lorawan/

[6] MrT0b0r. (2024, May 2). *Another record breaking transmission for Lorawan*. Hackster.io. https://www.hackster.io/news/another-record-breaking-transmission-for-lorawan-0cca5f6cd032#:~:text=2,had%20a%20LoRaWAN%20sensor%20attached.

[7] Wikimedia Foundation. (2024, April 11). *ISM Radio Band*. Wikipedia. https://en.wikipedia.org/wiki/ISM_radio_band

[8] *SX1276*. smtc. (n.d.). https://www.semtech.com/products/wireless-rf/lora-connect/sx1276

[9] *Rp2-1.0.3 LoRaWAN® regional parameters*. LoRa Alliance®. (2022, November 19). https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/

[10] Docker. https://hub.docker.com/r/thethingsnetwork/lorawan-stack

[11] Mcci-Catena. *Mcci-Catena/Arduino-lmic: Lorawan-MAC-in-C library, adapted to run under the arduino environment*. GitHub. https://github.com/mcci-catena/arduino-lmic

[12] *Sleep modes*. ESP. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html

[13] Hub.docker.com. (n.d.). https://hub.docker.com/r/nodered/node-red

[14] C. Khawas and P. Shah, "Application of Firebase in Android App Development-A Study," Int J Comput Appl, vol. 179, no. 46, pp. 49–53, Jun. 2018, doi: 10.5120/ijca2018917200.

[15] *Table_calendar: Flutter Package*. Dart packages. (2024, June 9). https://pub.dev/packages/table_calendar

[16] *What is the real range of Lora?*. Yosensi. (n.d.). https://yosensi.io/posts/what_is_the_real_range_of_lora/

[17] R. Muppala, A. Navnit, S. Poondla and A. M. Hussain, "Investigation of Indoor LoRaWAN Signal Propagation for Real-World Applications," 2021 6th International Conference for Convergence in Technology (I2CT), Maharashtra, India, 2021, pp. 1-5, doi: 10.1109/I2CT51068.2021.9418173.

[18] Meijers, J. (n.d.). TTN coverage. https://ttnmapper.org/heatmap/