ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διπλωματική Εργασία

Web Εφαρμογή Καταγραφής Μετρήσεων Σε Ενσωματωμένα Συστήματα Με Χρήση Πρωτοκόλλου Επικοινωνίας Lora

---

Μαυρογιαννόπουλος Γεώργιος
Α.Μ. 235833

Επιβλέπων
Καθ. Μπούρας Χρήστος

Συνεπιβλέποντες
Γκάμας Απόστολος, Παπαχρήστος Νικόλαος

Μέλη Επιτροπής Αξιολόγησης
Καθηγητής – Βλάχος Κυριάκος
Επ. Καθηγήτρια – Παπαϊωάννου Ευαγγελία

Πάτρα, 2023

Στους Γ.Σ., Γ.Δ. & Δ.Δ.
Ο πρώτος δεν το ήθελε καθόλου και οι υπόλοιποι περισσότερο από εμένα.

Στη Ρ.Ψ.
που με βοήθησε να καταλάβω τι πρέπει να κάνω με τη παραπάνω πληροφορία.

Πρόλογος

Το **Διαδίκτυο των Πραγμάτων** (αγγλικά: **Internet of Things, IoT**) αποτελεί το δίκτυο επικοινωνίας πληθώρας συσκευών, οικιακών συσκευών, αυτοκινήτων καθώς και κάθε αντικειμένου που ενσωματώνει ηλεκτρονικά μέσα, λογισμικό, αισθητήρες και συνδεσιμότητα σε δίκτυο ώστε να επιτρέπεται η σύνδεση και η ανταλλαγή δεδομένων. Επινοήθηκε στα τέλη της δεκαετίας του ’90 από τον επιχειρηματία Kevin Ashton, μέλος της ομάδας που επινόησε τον τρόπο σύνδεσης αντικειμένων με το διαδίκτυο μέσω της ετικέτας RFID (Radio Frequency Identification) [1]. Πρόκειται για μία από τις 3 κορυφαίες τεχνολογικές εξελίξεις της τρέχουσας δεκαετίας μαζί με το Mobile Internet και την αυτοματοποίηση του knowledge work.

Συσκευές ή αντικείμενα που φέρουν ενσωματωμένους αισθητήρες μπορούν να διαμοιράζουν πληροφορία σε εφαρμογές που έχουν δημιουργηθεί ώστε να αντιμετωπίζουν συγκεκριμένα λάθη ή προβλήματα. Τέτοιου είδους λύσεις μπορούν να χρησιμοποιηθούν για τη γρήγορη και αποτελεσματική υλοποίηση, κατά τα άλλα, δύσκολων, χρονοβόρων ή ακόμα και επικίνδυνων διαδικασιών. Μερικά παραδείγματα είναι η μέτρηση υγρασίας σε αγροτικές εγκαταστάσεις, η μέτρηση θερμοκρασίας στα εσωτερικά μέρη ενός Η/Υ, ή η μέτρηση στροφών ανά λεπτό σε κινητήρες.

Η χρησιμότητα μίας τέτοιας τεχνολογίας δεν σταματά όμως εκεί. Οι άνθρωποι αποζητούν συνεχώς μεγαλύτερη αυτονομία από τις ηλεκτρονικές συσκευές που χρησιμοποιούν. Στις επιχειρήσεις, αυτό μπορεί να μεταφράζεται ως αποθήκευση ή παρακολούθηση/έλεγχο συνεχώς ροής δεδομένων (streaming data) ενώ στην καθημερινότητα ως αυτόματα ξυπνητήρια ανάλογα με το πρόγραμμα εβδομάδας του χρήστη ή «έξυπνα» ψυγεία με δυνατότητα εντοπισμού ελλείψεων στο εσωτερικό τους, καταγραφή αγαπημένων υλικών ή και αναζήτηση διαθεσιμότητας συστατικών για συγκεκριμένες συνταγές.

Βέβαια, στο συγκεκριμένο σημείο, πρέπει να εισαχθεί – σύμφωνα με πολλούς – το θέμα της ασφάλειας των δεδομένων που αποθηκεύονται σε τέτοιου είδους συσκευές. Σε πρόσφατη έρευνα του “Icontrol State of the Smart Home” [2], το 44% των ατόμων που ερωτήθηκαν

στην Αμερική, δήλωσαν ανησυχία για τα δεδομένα τους κατά τη χρήση τέτοιων συσκευών ή υπηρεσιών. Φυσικά δεν είναι ανεδαφικές τέτοιου είδους αμφιβολίες, αφού ήδη από το 2008, ερευνητές στο τομέα της ασφάλειας επέδειξαν τρόπο απομακρυσμένου ελέγχου βηματοδοτών. Αργότερα το ίδιο έγινε και σε αντλίες ινσουλίνης [3].

Επειδή λοιπόν, σε τέτοιου είδους συστήματα ελέγχου / παρακολούθησης, η πληροφορία που καταγράφεται μπορεί να είναι από αυστηρά προσωπική (θέματα υγείας – υγιεινής) μέχρι απολύτως άχρηστη (περίοδος λειτουργίας αυτόματου λαμπτήρα σε ένα έτος) και διάφοροι τεχνολογικοί τομείς εξελίσσονται ταυτόχρονα αλλά όχι με την ίδια ζωηρότητα, χρειάζεται προσοχή στον τύπο των δεδομένων που εισάγονται στους αισθητήρες αλλά και στις προληπτικές διαδικασίες οχύρωσης αυτών (των δεδομένων), κατά περίπτωση.

Σε κάθε περίπτωση, με τη παγίωση της τεχνολογίας αυτής στην καθημερινότητα - λαμβάνοντας πάντα υπόψιν τα προαναφερθέντα- οι συνέπειες θα είναι σίγουρα σημαντικές, καθώς θα δοθεί η δυνατότητα σε υπηρεσίες δημόσιου και ιδιωτικού τομέα να αλλάξουν τον τρόπο λειτουργίας τους, εκμεταλλευόμενες την συρρίκνωση μεγάλου όγκου πληροφορίας στο απολύτως απαραίτητο και σημαντικό μέρος του, μέσω της επιτυχούς διαρκούς επικοινωνίας πληθώρας τέτοιων συσκευών, δημιουργώντας νέες ευκαιρίες, υπηρεσίες καθώς και διευκολύνοντας την λήψη αποφάσεων αλλά και επιβεβαιώνοντας τη λήψη αυτών.

Περίληψη

Η εργασία αυτή παρουσιάζει τη διαδικασία τοποθέτησης και παραμετροποίησης ολοκληρωμένων συστημάτων με ενσωματωμένους αισθητήρες περιβαλλοντολογικών συνθηκών σε μια καίρια τοποθεσία μιας αληθινής εταιρίας πληροφορικής (δωμάτιο αρχειοθέτησης) χρησιμοποιώντας το πρωτόκολλο επικοινωνίας LoRa για μείωση διακοπών λειτουργίας λόγω προβλημάτων των παρόχων υπηρεσιών διαδικτύου, και συχνή (ανά 1 ώρα) ενημέρωση των μετρήσεων από το δωμάτιο.

Κατά την έναρξη του συγκεκριμένου έργου, υπήρχαν πολλές εναλλακτικές οι οποίες θα μπορούσαν να επιλεχθούν, όμως σημαντικοί παράγοντες της συγκεκριμένης περίπτωσης, μας οδήγησαν στην χρήση ασυρμάτων επικοινωνιών για IoT λύσεις.

Πιο συγκεκριμένα, είναι σημαντικό να σημειωθεί πως το συγκεκριμένο δωμάτιο δεν έχει σταθερή σύνδεση με το διαδίκτυο καθώς το σήμα Wi-Fi είναι ασθενές, δεν έχει έτοιμες υποδομές δικτύου (επιτοίχιες απολήξεις κλπ) για τη χρήση άλλων δικτυακών συσκευών. Τέλος, τελευταίο αλλά εξίσου σημαντικό θέμα που έπρεπε να λυθεί είναι η μέγιστη δυνατή ελαχιστοποίηση της χρήσης ηλεκτρικής ενέργειας, ώστε να συμβαδίζει με το ISO 27001 σχετικά με το περιβάλλον που διατηρεί η εν λόγω εταιρία.

Καταλήγοντας στη λύση που πραγματεύεται η παρούσα εργασία, παρακάμφθηκε επιτυχώς η απαίτηση σύνδεσης στο διαδίκτυο της συσκευής καταγραφής των συνθηκών που θα τοποθετούνταν στο δωμάτιο. Επίσης ελαχιστοποιήθηκε και η χρήση ηλεκτρικής ενέργειας λόγω της αυτόματης εναλλαγής μεταξύ των λειτουργιών αναμονής (sleep mode) κατά το χρονικό διάστημα που δεν χρειάζονται μετρήσεις και αφύπνισης (wake-up timer) για σύντομο χρονικό διάστημα κατά την παραγωγή και αποστολή των δεδομένων.

Αναλυτικότερα σχετικά με τη σύνδεση της συσκευής καταγραφής στο διαδίκτυο, δεν είναι απαιτητή καθώς η εν λόγω συσκευή αποστέλλει δεκαεξαδικά δεδομένα ασύρματα μέσω της ενσωματωμένης κεραίας που φέρει, χρησιμοποιώντας το πρωτόκολλο LoRa, σε άλλη συσκευή, σε διαφορετικό χώρο με σταθερή σύνδεση στο Διαδίκτυο. Η τελευταία λειτουργεί ως μεσάζων και μας επιτρέπει την παρακολούθηση από χειριστή και έλεγχο των μετρήσεων μέσω κάποιας web εφαρμογής, η οποία παρουσιάζεται στη συνέχεια της εργασίας.

Abstract

This paper presents the process of installing and configuring complete systems with integrated environmental sensors in a key location of a realistic IT company (archive room) using the LoRa communication protocol in order to overcome internet service outages due to problems of the ISPs, while frequently (every 1 hour) updating the environmental measurements from that room.

At the beginning of this project, there were many alternatives that could have been chosen, but important factors in this case led us to use wireless communication model solutions for IoT.

More specifically, it is important to note that this room did not have a stable internet connection as the Wi-Fi signal is weak, nor any network infrastructure in place (like ethernet wall sockets, etc.) that could make the use of usual network devices possible. Furthermore, the last but equally important issue that had to be solved was the best possible minimization of power usage, in order to keep up with ISO 27001 certificate's environmental guidelines, which is successfully maintained by the company.

In coming up with such a solution, there was no requirement for an internet connection regarding the device that was used to record the environmental conditions in the room. Also, the use of electricity was greatly reduced by implementing a method of switching between sleep mode during the time where no measurements were made and a wake-up timer for a short period of time during the production and dispatch of the required data.

In more detail about the connection of the recording device to the internet, it is not required as this device sends hexadecimal data wirelessly through its built-in antenna, using the LoRa protocol, to another device (gateway) in a different close-by location with a steady Internet connection that acts as a middleman, forwarding and re-shaping the data from the LoRa communication protocol to web-based requests, allowing us to monitor and control measurements through a web application, which is presented in the following paragraphs.

Περιεχόμενα

Λίστα Εικόνων

Λίστα Σχημάτων

Λίστα Πινάκων

1

Εισαγωγή

1.1 Ιστορική Αναδρομή

Ο όρος Internet εισάχθηκε για πρώτη φορά στις αρχές της δεκαετίας του 1980, όταν το ARPANET διασυνδέθηκε με το NSFNET και αποσκοπούσε στη περιγραφή του μεγαλύτερου μέχρι τότε ενοποιημένου δικτύου που χρησιμοποιούσε το πρωτόκολλο TCP/IP [4]. Αρχικά, όπως και τα υπόλοιπα παρόμοια δίκτυα που αναπτύχθηκαν, το internet είχε ως κύριο στόχο τη χρήση του από κυβερνητικούς παράγοντες και το προσωπικό αυτών. Ωστόσο το ενδιαφέρον για την εμπορική χρήση του Internet έγινε γρήγορα πρώτιστο θέμα συζητήσεων. Αυτό είχε ως αποτέλεσμα, την ίδια χρονιά, την ίδρυση και την ανάπτυξη πρώτων εταιριών Παροχής Υπηρεσιών Διαδικτύου (Internet Service Providers). Ο παγκόσμιος ιστός, εφευρέθη επίσης λίγο αργότερα από τον Τιμ Μπέρνερς Λι, με σκοπό τη διασύνδεση εγγράφων και άλλων δικτυακών πόρων που προσδιορίζονται μέσω Ενιαίων Αναγνωριστικών Πόρων (Uniform Resource Locators -URLs), χρησιμοποιώντας το Διαδίκτυο [5]. Για την πρόσβαση σε οποιοδήποτε τέτοιο πόρο, ο εκάστοτε χρήστης θα χρειαζόταν έναν περιηγητή διαδικτύου και την εγγραφή του σε κάποιον από τους τότε Internet Service Provider.

10 μόλις χρόνια αργότερα, παρουσιάστηκε από τον Κέβιν Άστον του MIT, ο όρος “Internet of Things” ως ένα σύστημα όπου το Διαδίκτυο συνδέεται με το φυσικό κόσμο μέσω πανταχού παρόντων αισθητήρων [6]. Ταυτόχρονα, ο Μαρκ Βάιζερ, Chief Technical Officer της Xerox PARC μιλούσε για την «Πανταχού Παρούσα Υπολογιστική», η οποία πίστευε ότι περιγράφει το τρίτο κύμα εξέλιξης της Πληροφορικής, μετά τα mainframe και τους

προσωπικούς υπολογιστές, όπου η τεχνολογία πλέον υποχωρεί, λειτουργώντας σιωπηλά και ακατάπαυστα στο υπόβαθρο της καθημερινότητας [7].

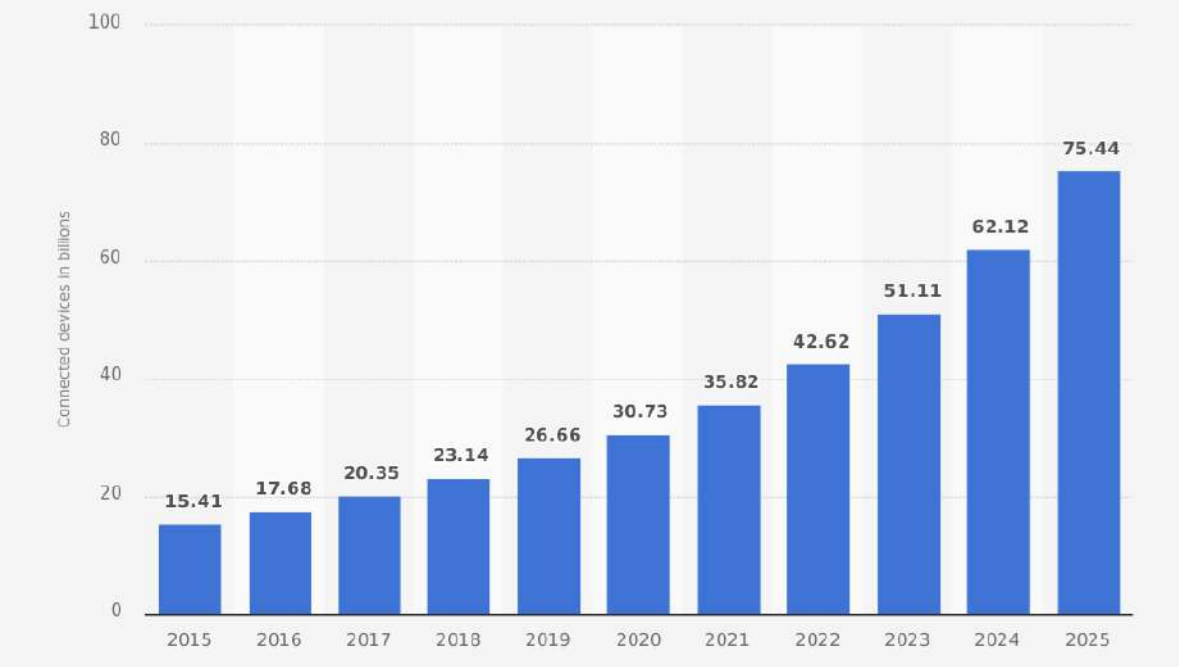
Γρήγορα, το όραμα του Κέβιν Άστον, επισημοποιήθηκε από τη Διεθνή Ένωση Τηλεπικοινωνιών, η οποία δημοσίευσε την πρώτη έκθεση για το Internet of Things το 2005 [8], αναφέροντας επιγραμματικά:

‘‘Το επόμενο βήμα στις επικοινωνίες "always on", στις οποίες νέες πανταχού παρούσες τεχνολογίες (όπως αναγνώριση ραδιοσυχνοτήτων και αισθητήρες) υπόσχονται έναν κόσμο δικτυωμένων και διασυνδεδεμένων συσκευών (π.χ., ψυγείο, τηλεόραση, όχημα, γκαράζ κ.λπ.) που παρέχουν σχετικές περιεχόμενο και πληροφορίες ανεξάρτητα από την τοποθεσία του χρήστη - προαναγγέλλοντας την αυγή μιας νέας εποχής, στην οποία το Διαδίκτυο (δεδομένων και ανθρώπων) αποκτά μια νέα διάσταση για να γίνει Διαδίκτυο των πραγμάτων.”

Το 2008, ο συνολικός αριθμός συνδεδεμένων συσκευών στο Internet ξεπέρασε για πρώτη φορά το πληθυσμό της Γης, δίνοντας αφορμή για τη καθιέρωση του πρώτου παγκόσμιου συνεδρίου για το IoT που έλαβε χώρα την ίδια χρονιά [9]. Εκεί παρουσιάστηκαν διάφορες εμπορικές και ακαδημαϊκές προτάσεις για την υλοποίηση και την περαιτέρω ανάπτυξη της ιδέας των διασυνδεδεμένων συσκευών. Σημαντικός επίσης παράγοντας στην ανάπτυξη της τεχνολογίας του IoT, ήταν και η εισαγωγή μικροελεγκτών ανοιχτού κώδικα στην αγορά (π.χ., Arduino) που κατέστησαν δυνατή την υλοποίηση τεράστιου αριθμού εφαρμογών για αυτοματισμούς σε οικιακές αλλά και επαγγελματικές εγκαταστάσεις.

Σύμφωνα με έρευνες που διεξήχθησαν, οι 4 πυλώνες της αλλαγής που οραματίστηκε προ ετών ο Μάρκ Βάιζερ, είναι το νέφος (cloud), η φορητότητα (mobility), τα μεγάλα δεδομένα (big data) και η κοινωνική δικτύωση (social networking). Οι πυλώνες αυτοί, επιτρέπουν στην τεχνολογία να παραμένει ζωντανή στο παρασκήνιο της ζωής των ανθρώπων και να λειτουργεί, τρόπον τινά, αυτόνομα, χωρίς να χρειάζεται ανθρώπινη συμμετοχή για την επεξεργασία δεδομένων [10]. Συνοπτικά δηλαδή, βοηθούν στην δημιουργία ενός εξειδικευμένου εμφωλευμένου οικοσυστήματος «βοηθών», ρόλος ο οποίος ανατίθεται στις συσκευές αυτές, ώστε να περιορίζεται στο ελάχιστο η ανθρώπινη ενασχόληση για την παραγωγή πληροφορίας και τη λήψη αποφάσεων σε επαναλαμβανόμενα, γνωστά σενάρια λειτουργίας. Το μόνο που μένει ως ανθρώπινη ευθύνη, είναι η αξιοποίηση της παραγόμενης πληροφορίας και ο έλεγχος των αυτοματοποιημένων λειτουργιών, σε υψηλό επίπεδο.

Η εταιρία Gartner προέβλεψε λανθασμένα το 2013, ότι θα υπάρχουν πάνω από 20 δισεκατομμύρια “things” στο Διαδίκτυο [11]. Αυτός ο αριθμός ξεπεράστηκε ήδη από το 2017 και η νέα πρόβλεψη είναι ότι το πλήθος θα φτάσει τις 75 δισεκατομμύρια συσκευές μέχρι το 2025, σύμφωνα με τη Morgan Stanley [12].



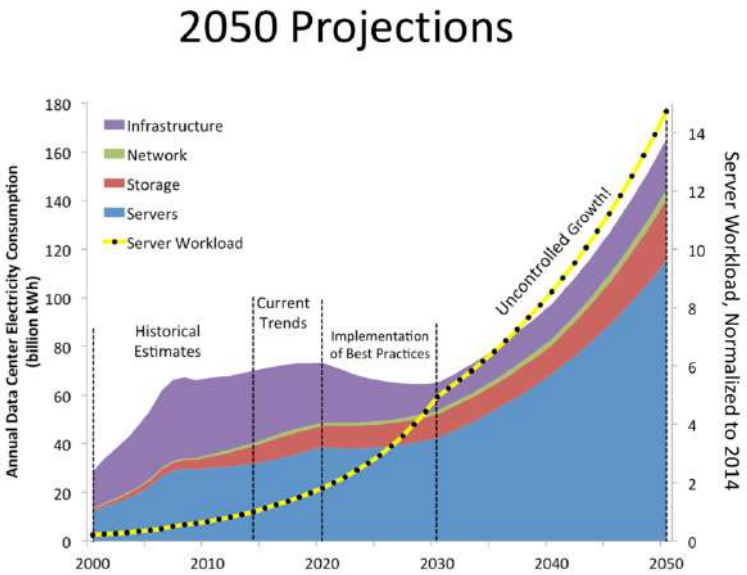
Σχήμα 1.α Ο αριθμός των things συνδεδεμένων στο Internet (Πηγή: Statista.com)

Η ραγδαία εκθετική αυτή αύξηση των συσκευών του Διαδικτύου, έφερε στην επιφάνεια προβλήματα τα οποία απειλούν την λειτουργικότητα του IoT, της οποία το κλειδί είναι η διασφάλιση ότι αυτά τα “things” ή τελικοί κόμβοι είναι πράγματι σε θέση να επικοινωνούν απρόσκοπτα με το Διαδίκτυο. Οποιοδήποτε δίκτυο υποστηρίζει μία τέτοια αχανή υποδομή θα πρέπει άλλωστε να έχει τη δυνατότητα να χειρίζεται τις όποιες κινήσεις (= επικοινωνίες) στο εσωτερικό του. Οι συσκευές IoT αυτή τη στιγμή, χρησιμοποιούν διάφορες τεχνολογίες για να υποστηρίξουν τις επικοινωνίες τους, αλλά σχεδόν καμία από αυτές δεν είναι πραγματικά ιδανική, σύμφωνα με τις σημερινές τεχνολογικές απαιτήσεις.

1.2 Στόχοι της Εργασίας

Σίγουρα, οι κατά περίπτωση αρμόδιοι έχουν φροντίσει ώστε ο ηλεκτρονικός εξοπλισμός που διατηρείται σε επαγγελματικές εγκαταστάσεις να είναι όσο το δυνατόν πιο αποκομμένος από φυσικά φαινόμενα ή/και καταστροφές. Παρολαυτά, δεν είναι λίγες οι φορές που η ευαισθησία του τελευταίου απέναντι σε φυσικά φαινόμενα έχει γίνει παραπάνω από αισθητή. Συνεπώς, η ανάγκη για συνεχή παρακολούθηση των περιβαλλοντολογικών συνθηκών σε εγκαταστάσεις τεχνολογίας είναι λίγο-πολύ αυταπόδεικτη. Συγκεκριμένοι λόγοι μπορεί να κυμαίνονται από νομικές υποχρεώσεις της εκάστοτε επιχείρησης (τήρηση νομικών εγγραφών σύμφωνα με το Ελληνικό δίκαιο) έως και παραγωγικότητας (π.χ., διατήρηση αρχείων ιστορικού όσον άφορα τις ενέργειες μιας επιχείρησης ή οργανισμού στην πάροδο του χρόνου). Σε κάθε περίπτωση όμως, μία σχετική εξασφάλιση σωστής πληροφόρησης όσον αφορά τις συνθήκες περιβάλλοντος σε μια τέτοια εγκατάστασης και η δυνατότητα πρόληψης ή έγκαιρης ενημέρωσης σε περιπτώσεις καταστροφών είναι παραπάνω από επιθυμητή, αν όχι αναγκαία για όλες τις επιχειρήσεις. Οι επιπτώσεις χρήσης τέτοιου εξοπλισμού δεν σταματούν όμως σε αυτό το σημείο. Σύμφωνα με πρόσφατη έρευνα του NRDC (Συμβούλιο Άμυνας Φυσικών Πόρων στις ΗΠΑ) αναμενόταν η κατανάλωση ηλεκτρικής ενέργειας στα διάφορα κέντρα δεδομένων (data centers) είχε ήδη αυξηθεί σε 140 δισεκατομμύρια κιλοβατώρες ετησίως το 2020, ποσό που ισούται με την συνολική παραγωγή 50 σταθμών παραγωγής ηλεκτρικής ενέργειας. Αυτό φυσικά μεταφράζεται άμεσα σε κόστος ύψους περίπου 13 δισεκατομμυρίων για τις αμερικάνικες επιχειρήσεις. Οι ενδείξεις αναφέρουν τέλος, ότι η αύξηση αυτή της κατανάλωσης ηλεκτρικής ενέργειας θα ακολουθήσει εκθετική πορεία και για τα επόμενα χρόνια. Άρα όλα τα παραπάνω, σε συνδυασμό με το ρόλο των διαχειριστών τέτοιων συστημάτων ή τέτοιων χώρων αποθήκευσης δεδομένων, να αλλάζει διαρκώς, καθώς προστίθενται ευθύνες στους αρμόδιους ενώ ταυτόχρονα διατηρούνται σταθερές οι απαιτήσεις για βέλτιστα επίπεδα λειτουργικότητας. Όλα τα παραπάνω αποτελούν σημαντικά σημάδια της αναγκαιότητας χρήσης νέων τεχνολογιών που μπορούν να έχουν την μέγιστη δυνατή απόδοση, με ελάχιστη χρήση πόρων (π.χ., ηλεκτρική κατανάλωση). Πέραν από το πνεύμα της εξέλιξης, η σωστή διαχείριση για την ισορροπία μεταξύ των παραγόντων αυτών (απόδοση και ενεργειακή κατανάλωση) δεν συνιστάται πλέον, αλλά

είναι αναπόσπαστο κομμάτι προκειμένου της σύγκλισης με τα νεότερα τεχνολογικά δεδομένα και τον τρόπο που αυτά διαμορφώνουν τη σύγχρονη κοινωνία.



Σχήμα 1.γ Γράφημα πρόβλεψης ετήσιας κατανάλωσης ηλεκτρικής ενέργειας σε κέντρα δεδομένων μέχρι το 2050

Πηγή: www.energy.gov

2

Ασύρματες Επικοινωνίες

2.1 Wi-Fi

Αποτελεί σίγουρα την ευρύτερα διαδεδομένη τεχνολογία ασύρματης επικοινωνίας καθώς χρησιμοποιείται από πληθώρα χρηστών σε όλο το κόσμο και είναι πλέον αναπόσπαστο κομμάτι της καθημερινότητας των αναπτυγμένων αλλά και των περισσότερων αναπτυσσόμενων χωρών. Αυτή η οικογένεια ασύρματων πρωτοκόλλων δικτύου, βασίζεται πάνω στο παγκοσμίως γνωστό σύνολο προτύπων IEEE 802.11, προκειμένου της διασύνδεσης συσκευών σε τοπικά δίκτυα (LAN) αλλά και την πρόσβαση των συσκευών αυτών, στο διαδίκτυο. Έχει σχεδιαστεί κατά τέτοιο τρόπο ώστε να λειτουργεί απρόσκοπτα σε συνδυασμό με τη τεχνολογία Ethernet, κατά την οποία η σύνδεση καλωδίου είναι απαιτητή. Συμβατές συσκευές μπορούν να συνδεθούν δικτυακά μέσω ασύρματων σημείων πρόσβασης μεταξύ τους, με συσκευές που χρησιμοποιούν καλωδίωση αλλά και στο Internet. Οι διάφορες εκδόσεις του WiFi καθορίζονται από τα διάφορα πρότυπα του πρωτοκόλλου 802.11 που καθορίζουν τις διαφορές στις συχνότητες, στις μέγιστες δυνατές

περιοχές κάλυψης διασύνδεσης αλλά και στις ταχύτητες αποστολής πληροφορίας που μπορούν να επιτευχθούν από τις συνδεδεμένες κατά τέτοιο τρόπο συσκευές [13]. Το Wi-Fi χρησιμοποιεί συχνότερα τις συχνότητες 2,4 GHz (120mm) και 5GHz (60mm). Αυτές οι δύο υποδιαιρούνται σε πολλά κανάλια, τα οποία μπορούν να μοιραστούν μεταξύ των δικτύων, έχοντας ως προϋπόθεση ότι μόνος ένας πομπός μπορεί να μεταδίδει τοπικά σε ένα κανάλι ανά δεδομένη χρονική περίοδο. Σίγουρα, το Wi-Fi σταδιακά εμφανίζεται παντού γύρω μας, αλλά καταναλώνει πολλή ενέργεια και εκπέμπει μεγάλο αριθμό δεδομένων, στοιχεία επιβαρυντικά για συσκευές που ενδεχομένως δεν έχουν σταθερή παροχή ηλεκτρικής ενέργειας (π.χ., συσκευές με μπαταρία) ή δεν μπορούν να στείλουν τόσο μεγάλο αριθμό δεδομένων λόγω περιορισμών σε επεξεργαστική ισχύ ή ανατεθειμένο μέγεθος μνήμης (RAM) – παράγοντες οι οποίοι επηρεάζουν σημαντικά την ταχύτητα αποστολής και λήψης δεδομένων σε ένα δίκτυο (throughput).

2.2 BLE – Bluetooth Low Energy

Το Bluetooth Low Energy, παρόμοιο με το ευρύτερα γνωστό Bluetooth, είναι μια τεχνολογία για ασύρματα προσωπικά δίκτυα που σχεδιάστηκε, αναπτύχθηκε και κυκλοφόρησε από την ομάδα Bluetooth SIG, στοχεύοντας την εφαρμογή του κυρίως στο τομέα της υγειονομικής περίθαλψης, φυσικής κατάστασης και στις βιομηχανίες ασφάλειας και οικιακής ψυχαγωγίας. Το αρχικό μοντέλο αναπτύχθηκε από την εταιρία Nokia το 2006, επονομαζόμενο ως Wibree και αργότερα ενσωματώθηκε στη τεχνολογία Bluetooth 4.0 ως Bluetooth LE [14]. Η συγκεκριμένη τεχνολογία χρησιμοποιεί τις ίδιες ραδιοσυχνότητες με το κλασικό Bluetooth (2.4 GHz) αλλά χρησιμοποιεί πιο απλό σύστημα διαμόρφωσης,

αποτελώντας σε γενικές γραμμές, συγγενή του προκάτοχού του με κύρια έμφαση στην ελαχιστοποίηση ενεργειακής κατανάλωσης, άρα και κόστους. Δυστυχώς όμως, ακόμα και οι συσκευές Bluetooth χαμηλής ενέργειας (Bluetooth Low Energy Devices) καταναλώνουν πολύ περισσότερη ενέργεια από αυτή που πραγματικά χρειάζεται.

2.3 GSM

Το Παγκόσμιο Σύστημα για Κινητές Επικοινωνίες (GSM) είναι ένα πρότυπο επικοινωνιών που αναπτύχθηκε για πρώτη φόρα από το Ευρωπαϊκό Ινστιτούτο Πρότυπων Τηλεπικοινωνιών (ETSI) για να περιγράψει πρωτόκολλα που αφορούν ψηφιακά κυψελοειδή δίκτυα 2^{ης} γενιάς [15]. Τέτοια πρωτόκολλα χρησιμοποιούνταν κατά κόρον στο παρελθόν από κινητές συσκευές και tablets. Έκανε την πρώτη του εμφάνιση στη Φινλανδία το Δεκέμβριο του 1991 και μέχρι το 2010, ήταν το παγκόσμια αποδεκτό πρότυπο για κινητές επικοινωνίες, με μερίδιο αγοράς άνω του 90%, λειτουργώντας σε περισσότερες από 193 χώρες. Περιεγράφηκε αρχικά ως ένα ψηφιακό δίκτυο για πλήρως αμφίδρομη φωνητική τηλεφωνία. Με τη πάροδο του χρόνου, επεκτάθηκε ώστε να συμπεριλαμβάνει μεταφορά δεδομένων μέσω πακέτων, χρησιμοποιώντας την υπηρεσία GPRS (General Packet Radio Service) και EDGE (Enhanced Data Rates for GSM Evolution). Στην συνέχεια βεβαίως, η τεχνολογία αυτή βρέθηκε να είναι παρωχημένη, με την ανάπτυξη των προτύπων 3^{ης} γενιάς (3G) και αργότερα των αντίστοιχων LTE Advanced 4^{ης} γενιάς (4G) και 5^{ης} γενιάς (5G) , τα οποία διαφοροποιήθηκαν ουσιαστικά από το πρότυπο GSM.

2.4 NB-IoT (Narrowband Internet of Things)

Στη προσπάθεια παγίωσης συγκεκριμένων προτύπων επικοινωνίας για το IoT, οι τεχνολογίες ξεκίνησαν να έχουν πιο εξειδικευμένη στόχευση ανάλογα με διάφορες ανάγκες. Ένα τέτοιο παράδειγμα είναι και το πρότυπο NB-IoT, τεχνολογίας χαμηλής ισχύος και μεγάλου εύρους (Low Power Wide Area Network - LPWAN) το οποίο αναπτύχθηκε ώστε να εφαρμοστεί σε ένα επίσης ευρύ φάσμα κινητών συσκευών και υπηρεσιών. Αναλυτικότερα, εστιάζει στην εσωτερική (indoor) κάλυψη συσκευών, το χαμηλό κόστος, την μεγάλη διάρκεια ζωής μπαταρίας και την υψηλή πυκνότητα συνδέσεων. Χρησιμοποιεί ένα υποσύνολο λειτουργιών του προτύπου LTE, περιορίζοντας το εύρος ζώνης στα 200kHz, διαμόρφωση OFDM για τις επικοινωνίες downlink και SC-FDMA για επικοινωνίες uplink. Προτείνεται για εφαρμογές IoT που απαιτούν συχνότερη επικοινωνία από ό,τι συνήθως, καθώς δεν έχει περιορισμό κύκλου λειτουργίας (δηλαδή περιορισμό του χρονικού ποσοστού κατά το οποίο μία συσκευή αποστέλλει ή μπορεί να λάβει δεδομένα), εξυπηρετώντας έτσι βέλτιστα το σκοπό του [16].

2.5 NFC (Near Field Communication)

Αξιοσημείωτη θέση στον χώρο επικοινωνίας των “πραγμάτων” έχει σίγουρα και το NFC, μια σειρά πρωτόκολλων επικοινωνίας μεταξύ δύο ηλεκτρονικών συσκευών («εκκινητής» και «στόχος» - “initiator” and “target”) σε απόσταση 4 εκατοστών, ή μικρότερη. Συσκευές που μπορούν να επικοινωνήσουν μέσω NFC χρησιμοποιούνται συνήθως ως ηλεκτρονικά έγγραφα ταυτότητας και κάρτες-κλειδιά (επί παραδείγματι για είσοδο σε κάποιο υπό επιτήρηση χώρο/δωμάτιο). Χρησιμοποιούνται επίσης ευρέως σε συστήματα ανέπαφων πληρωμών καθώς επιτρέπουν την συμπλήρωση πληροφοριών (στοιχείων κάρτας ή ειδικού πάσου) μέσω κινητού [17]. Το NFC χρησιμοποιείται επίσης για κοινή χρήση μικρών αρχείων, όπως π.χ., επαφές κινητού τηλεφώνου, λόγω της χαμηλής ταχύτητας μεταφοράς δεδομένων που προσφέρει (106 έως 424 kbit/s). Η ειδοποιός διαφορά με άλλα παρόμοια πρότυπα, είναι ότι αν μία από τις συσκευές που συμμετέχουν στην επικοινωνία είναι συνδεδεμένη στο διαδίκτυο, η δεύτερη μπορεί να διαμοιραστεί δεδομένα με δικτυακές υπηρεσίες (το σενάριο αυτό επί παραδείγματι, λαμβάνει χώρα κατά τη συναλλαγή μέσω ανέπαφης συναλλαγής – η κάρτα, που αναλαμβάνει το ρόλο του «στόχου» στην εν λόγω επικοινωνία - περιέχει τα απαραίτητα στοιχεία για την ολοκλήρωση της συναλλαγής και το EFT POS, «εκκινητής» ενεργοποιώντας ένα μαγνητικό πεδίο, ξεκινά τη μεταξύ τους επικοινωνία, λαμβάνει τα στοιχεία και τα αποστέλλει στη τράπεζα μαζί με το ποσό προς πληρωμή, μέσω internet, υλοποιώντας και ολοκληρώνοντας τη συναλλαγή). Η εν λόγω τεχνολογία άρα, χρησιμοποιεί την επαγωγική σύνδεση μεταξύ δύο κεραιών που υπάρχουν στις δύο συσκευές του παραδείγματος, οι οποίες μπορούν να επικοινωνούν προς μία ή και τις δύο κατευθύνσεις, κάνοντας χρήση της συχνότητας 13,56MHz στη διεθνή διαθέσιμη ζώνη ISM με το πρότυπο διασύνδεσης αέρα ISO /IEC 18000-3. Οι μέχρι τώρα αλλά και μελλοντικές εφαρμογές του NFC εκτιμάται ότι θα περιορίζονται στις ανέπαφες συναλλαγές, ανταλλαγή δεδομένων μικρού μεγέθους, και στην απλοποίηση εγκαταστάσεων πιο πολύπλοκων επικοινωνιών, όπως Wi-Fi (ανταλλαγή στοιχείων σύνδεσης κ.λ.π.)

2.6 Weightless

Το Weightless είναι ένα σύνολο προτύπων ανοιχτής ασύρματης τεχνολογίας LPWAN για την ανταλλαγή δεδομένων μεταξύ ενός σταθμού βάσης και χιλιάδων συσκευών γύρω από αυτόν. Το όνομα Weightless χρησιμοποιήθηκε ώστε να ανακλά την ανάλαφρη φύση του εν λόγω πρωτοκόλλου, αφού σχεδιάστηκε ώστε τα ενεργειακά έξοδα ανά μετάδοση πληροφορίας να ελαχιστοποιούνται για συσκευές που χρειάζεται να επικοινωνούν μόνο λίγα bytes δεδομένων. Σχεδιάστηκε και αναπτύχθηκε το 2011 από μία ομάδα μηχανικών στο Cambridge τουΗνωμένου Βασιλείου, βασιζόμενο στην τεχνολογία πολυπλεξίας διαίρεσης χρόνου (TDM) με αναπήδηση συχνότητας φάσματος εξάπλωσης (FHSS), στοχεύοντας στην μείωση της επίδρασης των παρεμβολών. Υποστηρίζει μεταβλητούς παράγοντες διάδοσης σε μια προσπάθεια αύξησης του εύρους επικοινωνίας (ελαττώνοντας έτσι τον ρυθμό αποστολής δεδομένων, όπως θα δούμε παρακάτω) ώστε να εξυπηρετεί και συσκευές χαμηλής ισχύος με χαμηλούς ρυθμούς μεταφοράς. Οι παραπάνω προδιαγραφές οριστικοποιήθηκαν το 2013, με την κυκλοφορία της έκδοσης 1.0 [18]. Δίκτυα Weightless έχουν αναπτυχθεί μέχρι στιγμής στη Κοπεγχάγη και τη Δανία, ωστόσο φαίνεται πως υπάρχει έλλειψη πελατειακού υλικού ώστε να χρησιμοποιηθεί μαζικά. Μέχρι στιγμής η πλειοψηφία των συσκευών που προβλέπεται να χρησιμοποιούν το συγκεκριμένο πρότυπο είναι κυρίως βιομηχανικού και ιατρικού τύπου, όπως έξυπνοι ηλεκτρικοί μετρητές, παρακολούθηση οχημάτων ή/και υγειονομικών μετρήσεων.

2.7 Z-Wave

Το Z-Wave είναι ένα άλλο πρωτόκολλο ασύρματων επικοινωνιών, σχεδιασμένο ειδικά για οικιακό αυτοματισμό. Χρησιμοποιεί ραδιοκύματα χαμηλής ενέργειας για επικοινωνία από συσκευή σε συσκευή σε μια υποδομή δικτύου πλέγματος (meshnet). Έτσι μπορεί και προσφέρει έλεγχο των οικιακών συσκευών που συνδέονται μέσω του πρωτοκόλλου αυτού, όπως για παράδειγμα, λαμπτήρες, συναγερμούς, θερμοστάτες, παράθυρα, κλειδαριές και αυτόματο άνοιγμα εισόδων (π.χ., γκαράζ). Όπως τα περισσότερα τέτοια πρωτόκολλα, προσφέρει δυνατότητα διαχείρισης μέσω του Διαδικτύου από κινητό τηλέφωνο, tablet ή Η/Υ. Αναπτύχθηκε από την Δανικής προέλευσης εταιρία Zensys το 1999 η οποία αγοράστηκε από τη Sigma Designs το 2008 [19]. Είναι σχεδιασμένο για την αξιόπιστη μετάδοση μικρών πακέτων δεδομένων με ρυθμό 100Kbits/sec με χαμηλή καθυστέρηση. Σε αντίθεση, με άλλα συστήματα ασύρματου LAN που έχουν σχεδιαστεί για υψηλό ρυθμός μεταφοράς δεδομένων, το συγκεκριμένο προτείνεται για εφαρμογές ελέγχου μεταξύ αισθητήρων και διακοπτών. Λόγω της χρήσης δικτύου πλέγματος, η επικοινωνία μεταξύ των κόμβων του δικτύου μπορεί να είναι άμεση ή έμμεση. Αν οι τερματικοί κόμβοι βρίσκονται σε κοντινή απόσταση μεταξύ τότε, επικοινωνούν απευθείας, ενώ σε μακρινή απόσταση χρησιμοποιούν ενδιάμεσους κόμβους για την επικοινωνία (μέχρι 4 μεταπηδήσεις μεταξύ κόμβων). Η διαμόρφωση που χρησιμοποιείται είναι FSK (frequency-shift keyring) με κωδικοποίηση Manchester. Στην Ευρώπη, λειτουργεί στα 908,42MHz, συχνότητα που χρησιμοποιούν μερικά ασύρματα τηλέφωνα και άλλες συσκευές της αγοράς αλλά αποφεύγει παρεμβολές με τα πρωτόκολλα Wi-Fi και Bluetooth που λειτουργούν στην γνωστότερη περιοχή των 2,4GHz [19]. Η μέγιστη δυνατή απόσταση για απευθείας επικοινωνία είναι περίπου 100 μέτρα, μεταξύ κόμβων ενώ ο μέγιστος αριθμός συσκευών συνδεδεμένων στο ίδιο δίκτυο, χωρίς την εισαγωγή network bridge, είναι 232. Ο τρόπος διασύνδεσης των συσκευών σε ένα τέτοιο δίκτυο, πρέπει να ακολουθήσει τη διαδικασία υιοθέτησης από τον ελεγκτή του συστήματος, ώστε να δοθεί στη συσκευή μια

συγκεκριμένη ταυτότητα κόμβου εντός μια συγκεκριμένης ταυτότητας δικτύου. Συσκευές που ανήκουν σε διαφορετικά δίκτυα, δεν μπορούν να επικοινωνήσουν μεταξύ τους. Αναφορικά με την ασφάλεια, η συγκεκριμένη τεχνολογία αναφέρεται ότι φέρει τους πλέον σύγχρονους τρόπους κρυπτογράφησης κατά την αποστολή και λήψη πακέτων μεταξύ δικτυακών κόμβων και η μόνη αδυναμία του συστήματος είναι κατά τη διαδικασία σύζευξης με μία συσκευή, λόγω της προς τα πίσω συμβατότητας με παλαιότερα συστήματα, που διατήρησε ο κατασκευαστής.

2.8 Zigbee

Κοντά στις λειτουργίες και τα χαρακτηριστικά του Z-Wave είναι το νεότερο (2004) Zigbee πρωτόκολλο επικοινωνίας, το οποίο επίσης αφορά δίκτυα πλέγματος συσκευών χαμηλής ισχύος με χαμηλούς ρυθμούς μεταφοράς. Συγκεκριμένα για την ολοκλήρωση της πιστοποίησης, οι συσκευές που χρησιμοποιούν το εν λόγω πρωτόκολλο θα έχουν διάρκεια μπαταρίας τουλάχιστον 2 έτη. Χρησιμοποιείται, όπως ο προκάτοχός του, σε εφαρμογές οικιακού αυτοματισμού αλλά και σε δίκτυα ασύρματων αισθητήρων, συστήματα ελέγχου βιομηχανικού τύπου, συλλογή ιατρικών δεδομένων και ελέγχους σημάτων κινδύνου όπως καπνού ή εισβολής σε συγκεκριμένες περιοχές ή χώρους. Έχει τη δυνατότητα, όπως το Z-Wave να μεταφέρει πληροφορία χρησιμοποιώντας ενδιάμεσους κοντινούς κόμβους ως μεσάζοντες, με κάλυψη απόστασης από 10 έως 100 μέτρα ανάλογα με τις περιβαλλοντικές

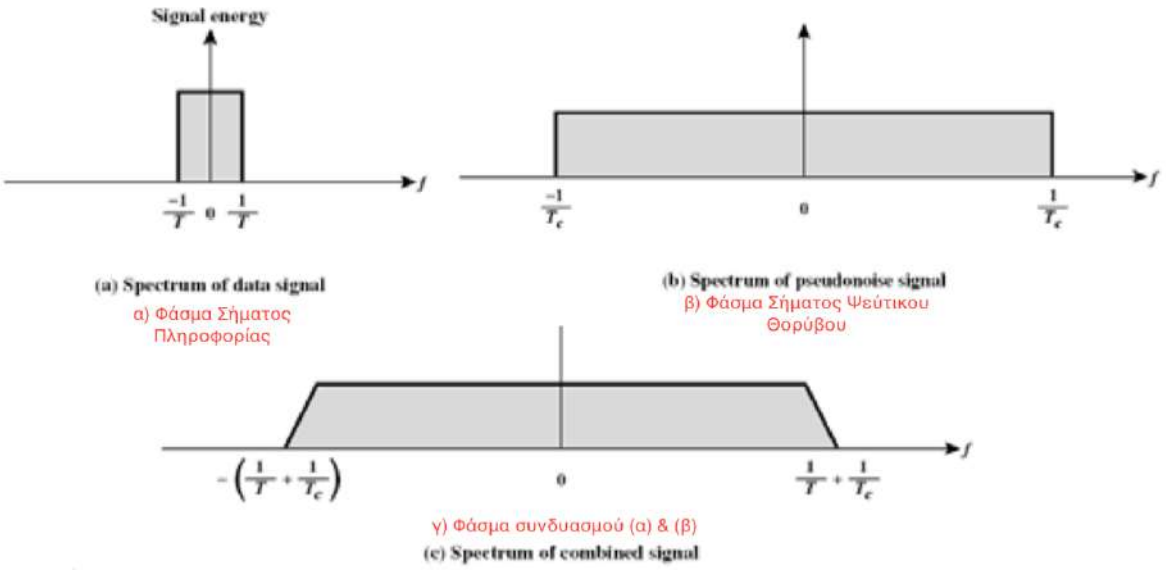
συνθήκες. Η συμμετρική κρυπτογράφηση που χρησιμοποιείται κατά την επικοινωνία είναι με χρήση κλειδιών 128-bit ενώ ο ορισμένος ρυθμός μεταφοράς είναι 250 Kbits/sec, βελτιστοποιημένος για διαλείπουσες μεταδόσεις δεδομένων από και προς αισθητήρες ή συσκευές εισόδου [20].

3 Τεχνολογία LoRa

3.1 LoRa

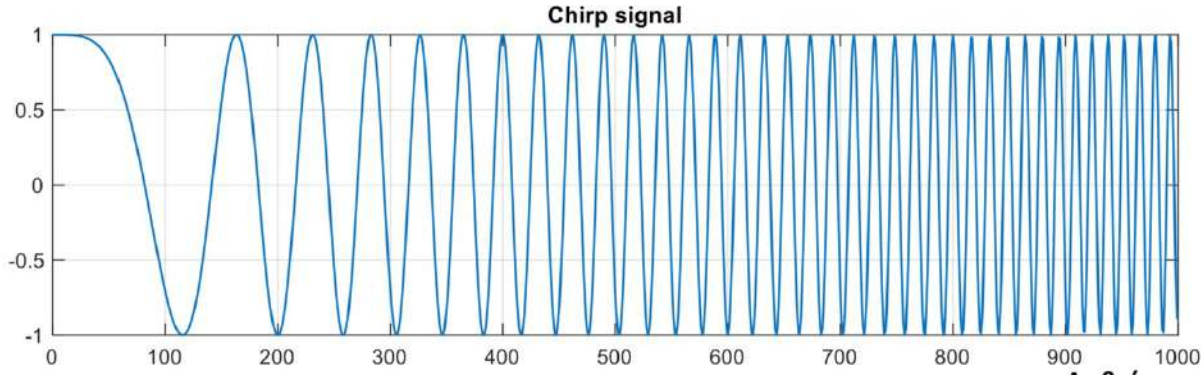
Ακολουθώντας τις ανάγκες του 21ου αιώνα, το 2009, δύο φίλοι από τη Γαλλία, ο Nicolas Sornin και ο Olivier Seller στόχευσαν στη δημιουργία μιας τεχνολογίας διαμόρφωσης χαμηλής ισχύος και μεγάλης εμβέλειας. Παρά την όποια αντίσταση και δυσκολίες, όπως συμβαίνει στις περισσότερες επαναστατικές τεχνολογίες συνέχισαν την ιδέα τους για την εκμετάλλευση της τεχνολογίας CSS (Chip Spread Spectrum modulation technology), που γινόταν έως τότε χρήση, κυρίως στη ναυτιλιακή βιομηχανία τα ραντάρ και την αεροπορία, για την αποστολή δεδομένων μέσω αυτής [21].

Συγκεκριμένα είναι η συνένωση των εννοιών της διασποράς φάσματος, δηλαδή του τρόπου με τον οποίο η ενέργεια ενός σήματος που καταλαμβάνει κάποιο σχετικά περιορισμένο φάσμα συχνοτήτων, ανακατανέμεται εσκεμμένα σε πολύ μεγαλύτερο φασματικό εύρος με σκοπό την ασφάλεια αποφυγής υποκλοπών - μεγαλύτερη αντοχή σε παρεμβολές, και του σήματος ολίσθησης συχνότητας (chirp). Το τελευταίο αποτελεί πρακτικά ένα ημιτονοειδές σήμα με μη σταθερές παραμέτρους ημιτόνων του οποίου η συχνότητα μεταβάλλεται γραμμικά ή εκθετικά.



Εικόνα Ε1: Παράδειγμα Διασποράς Φάσματος Ευθείας Ακολουθίας

$$x[n] = \cos(a_0 n^2)$$

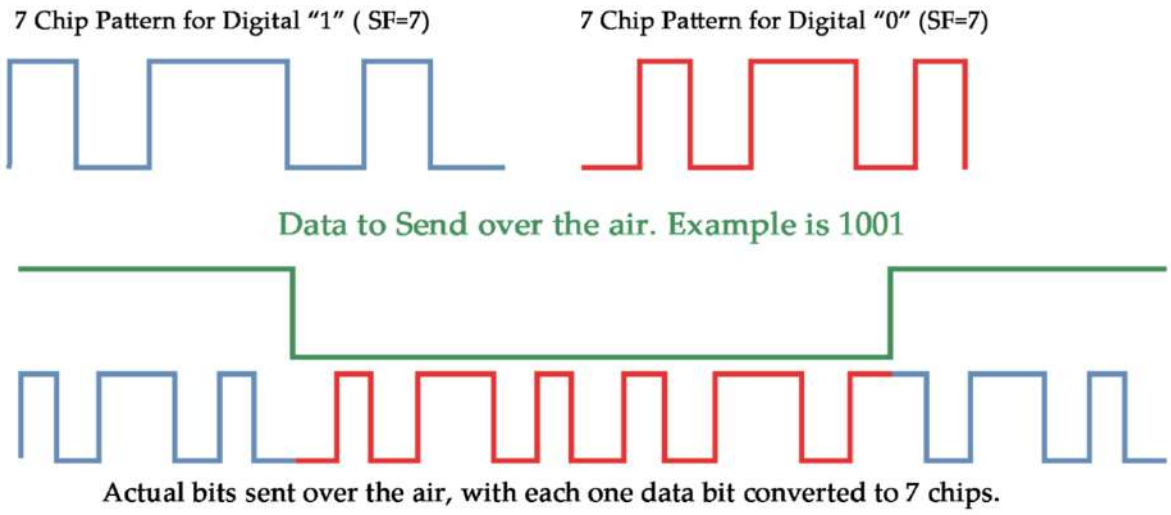


Εικόνα Ε2: Γραφική Αναπαράσταση δειγμάτων σήματος “chirp” διακριτού χρόνου

Η τεχνολογία CSS διαχωρίζεται από τις παραδοσιακές τεχνικές διασποράς φάσματος καθώς χρησιμοποιεί όλο το εκχωρημένο εύρος ζώνης για την μετάδοση ενός σήματος, κάνοντας το ισχυρό στο να διοχετεύει θόρυβο και να μην «ξεθωριάζει» ακόμη και όταν λειτουργεί με πολύ χαμηλή ισχύ. Επιπλέον, η συγκεκριμένη μέθοδος δεν προσθέτει ψευδό-τυχαία στοιχεία στο σήμα για να το διακρίνει από το θόρυβο του καναλιού αλλά βασίζεται στη γραμμική φύση του σήματος. Εξαιρετικά παρόμοιες κυματομορφές χρησιμοποιούνται και σε εξοπλισμό ραντάρ για την μέτρηση αποστάσεων. Αντίστοιχα και στο φυσικό κόσμο, από δελφίνια και νυχτερίδες για την αναζήτηση τροφής και επικοινωνία.

Πεπεισμένοι για την ορθότητα της ιδέας αυτή, η εταιρία Semtech αποκτά των μικρή γαλλική εταιρία Cycleo και συνεργαζόμενη με την αρχική ομάδα, χρησιμοποιώντας πλέον τους περισσότερους πόρους της ίδιας, δημοσιοποιεί τον Ιανουάριο του 2015 την πρώτη εκδοχή του πρωτοκόλλου LoRa. Η εκδοχή αυτή του πρωτοκόλλου λειτουργεί σε ζώνες ραδιοσυχνοτήτων, χαμηλότερες των GHz (στα 868 MHz συγκεκριμένα για την Ευρώπη), επιτρέποντας μεταδόσεις μεγάλης εμβέλειας (10+ χιλιόμετρα σε ιδανικές συνθήκες) με χαμηλή κατανάλωση ισχύος. Λόγω της μεγάλης εμβέλειας, οι ρυθμοί μεταφοράς δεδομένων που επιτυγχάνονται, κυμαίνονται μεταξύ 0,3 Kbits και 27Kbits/sec ανάλογα με τον παράγοντα διάδοσης.

Τα δεδομένα που διαμοιράζονται με το πρωτόκολλο LoRa, περιέχονται στο εσωτερικό ημιτονοειδών παλμών γραμμικά μεταβαλλόμενης συχνότητας, όπου τα λογικά ‘1’ και ‘0’ μεταφράζονται με σήματα αντίθετων φάσεων, όπως στο παρακάτω σχήμα [22]:



Εικόνα Ε3: Ακολουθία bits κωδικοποιημένη στη διαμόρφωση LoRa

Παράγοντας Διάδοσης – Spreading Factor

Ο παράγοντας διάδοσης (spreading factor) αποτελεί ένα από τα κύρια χαρακτηριστικά του LoRa, καθώς επιτρέπει προσαρμοστικές βελτιστοποιήσεις των επιπέδων ισχύος και της ταχύτητας μεταφοράς δεδομένων ενός κόμβου ανάλογα με το σενάριο που χρησιμοποιείται. Η διαμόρφωση LoRa έχει συνολικά 6 διαφορετικούς παράγοντες διάδοσης (SF7 έως SF12) ανάλογα με την απόσταση που χρειάζεται να διανύσει επιτυχώς το παραγόμενο σήμα.

Πιο συγκεκριμένα, όσον αφορά τους παράγοντες διάδοσης, ή αλλιώς συντελεστές διασποράς SF, αποτελούν πρακτικά τον λόγο μεταξύ του ρυθμού συμβόλων $\mathbf{R_S}$ και του ρυθμού ολίσθησης συχνότητας $\mathbf{R_C}$, σύμφωνα με τον τύπο:

$$SF = log_2 (R_c/R_s)$$

Εξίσωση 1.1

Η χρήση του παράγοντα διάδοσης SF αφορά την διαδικασία της διαμόρφωσης (modulation) του κάθε συμβόλου που πρόκειται να σταλεί σε ένα τερματικό κόμβο, πριν ξεκινήσει η μετάδοση. Κάθε μοναδικό σύμβολο διαμορφώνεται με $\mathbf{2^{SF}}$ ολισθήσεις συχνότητας (chirps).

Ο ρυθμός ολίσθησης συχνότητας (αλλιώς **chirp rate**) είναι η πρώτη παράγωγος της συχνότητας $\mathbf{F_S}$ των ημιτονοειδών παλμών που χρησιμοποιούνται για την μετάδοση δεδομένων στο LoRa πρωτόκολλο και εμπεριέχει στο εσωτερικό του τα μεγέθη του παράγοντα διάδοσης και του εύρους ζώνης, αναλυτικότερα:

$$Chirp\ rate = BW \times R_s = \frac{BW^2}{2^{SF}}$$

Εξίσωση 1.2

Το αντίστροφο του παραπάνω, ονομάζεται διάρκεια συμβόλου LoRa και συμβολίζεται με $\mathbf{T_S}$. Συνεπώς όταν το SF αυξάνεται κατά μία μονάδα, η διάρκεια συμβόλου διπλασιάζεται, εάν χρησιμοποιείται καθορισμένο εύρος ζώνης.

Όσο αυξάνεται η διάρκεια του συμβόλου, τόσο πιθανότερη είναι και η πιθανότητα επιτυχούς μεταφοράς της πληροφορίας αφού το διαμορφωμένο σήμα αποκτά ανθεκτικότητα σε παρεμβολές και θόρυβο με αποτέλεσμα η χρήση υψηλότερου παράγοντα διάδοσης να ισούται με την διεύρυνση της ακτίνας κάλυψης του εκάστοτε τηλεπικοινωνιακού LoRa συστήματος. Χρειάζεται προσοχή όμως σε κάθε περίπτωση, καθώς όσο περισσότερα σύμβολα αποστέλλονται, τόσο μεγαλύτερη είναι η πιθανότητα αποστολής λαθών ή συγκρούσεων μηνυμάτων. Για το λόγο αυτό, προτείνεται η αποστολή μηνυμάτων μικρής έκτασης ακόμα και όταν επιλέγεται μεγάλος παράγοντας διάδοσης.

Επιπλέον, οι έξι διαφορετικοί παράγοντες διάδοσης παρέχουν κάποιο είδος ορθογωνικότητας στη μετάδοση δεδομένων που λαμβάνει χώρα στην ίδια συχνότητα. Αυτή η ιδιότητα ορθογωνικότητας μπορεί να εξηγηθεί αφού ο διαφορετικός παράγοντας διάδοσης θα σημαίνει διαφορετική ολίσθηση συχνότητας (chirp) στο χρόνο. Δεδομένου ότι η τεχνολογία CSS χρησιμοποιεί συχνότητες με γραμμική μεταβολή στη πάροδο του χρόνου, όταν σχεδιάζει συχνότητα ενάντια στο χρόνο, ο ρυθμός ολίσθησης θα φαίνεται στη κλίση της καμπύλης. Επομένως, θεωρητικά, διαφορετικός παράγοντας διάδοσης θα παράγει

διαφορετικές κλίσεις που παρέχουν αυτή την ιδιότητα ορθογωνικότητας [22]. Επί παραδείγματι, χρησιμοποιώντας υψηλότερους SF (συγκεκριμένα SF2) σε κόμβους πιο μακριά από μία πύλη και μικρότερους (SF1) σε χρήστες πιο κοντά στη πύλη, έχουμε τα εξής:

- SF1 < SF2 διανέμεται σε χρήστες με απόσταση r από τη πύλη, όπου r < d < R
- SF2 διανέμεται σε χρήστες σε απόσταση r > d, όπου d < r < R

Σε αυτό το σενάριο, οι χρήστες με συντελεστή διασποράς SF1 δεν θα έχουν ποτέ παρεμβολές στην επικοινωνία τους από κόμβους που χρησιμοποιούν τον SF2 αφού οι τελευταίοι βρίσκονται ντετερμινιστικά σε μεγαλύτερες αποστάσεις και άρα η ισχύς σήματος που λαμβάνεται από την πύλη θα είναι πάντα χαμηλότερη.

Λόγω όλων των παραπάνω και προκειμένου της επίτευξης επικοινωνίας μεταξύ πομπού και δέκτη, είναι λογικό να χρειάζεται ο ίδιος παράγοντας διάδοσης να είναι γνωστός και στους δύο.

LoRa Spreading Factors (125 kHz bw)

Spreading Factor	Chips/symbol	SNR limit	Time-on-air (10 byte packet)	Bitrate
7	128	-7.5	56 ms	5469 bps
8	256	-10	103 ms	3125 bps
9	512	-12.5	205 ms	1758 bps
10	1024	-15	371 ms	977 bps
11	2048	-17.5	741 ms	537 bps
12	4096	-20	1483 ms	293 bps

Εικόνα Ε4: Παράγοντες Διάδοσης LoRa και επηρεαζόμενα μεγέθη

Τεχνικές Κωδικοποίησης

Επιπλέον, για τη διασφάλιση της αξιοπιστίας κατά την μεταφορά δεδομένων, χρησιμοποιούνται κάποιες συγκεκριμένες τεχνικές κωδικοποίησης οι οποίες παρατίθενται επιγραμματικά παρακάτω:

Data Whitening: Τα δεδομένα μετασχηματίζονται γραμμικά σε μια προσπάθεια τυχαιοποίησης αυτών. Στην αρχική τους μορφή περιέχουν ένα είδος αυτοσυσχέτισης μεταξύ τους, η οποία σταματά να ισχύει κάνοντας χρήση μίας whitening ακολουθίας (random sequence) που δημοσιοποιείται τόσο στο δέκτη όσο και στο πομπό του μηνύματος με αποτέλεσμα την ομοιόμορφη κατανομή της σε όλο το εύρος ζώνης του καναλιού.

Gray Indexing: Τα κωδικοποιημένα σύμβολα αναδιατάσσονται ώστε δύο διαδοχικά να μην έχουν διαφορά παραπάνω από 1 bit. Μέσω της διαδικασίας αυτής αποτρέπονται σφάλματα κατά την ανάκτηση των συμβόλων και αυξάνονται οι πιθανότητες διόρθωσης τέτοιων σφαλμάτων από το κώδικα του καναλιού που χρησιμοποιείται.

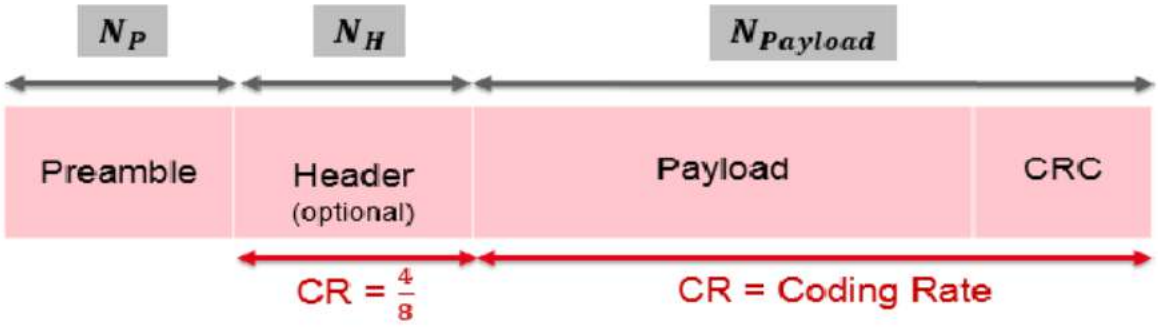
Forward Error Correction (FEC): Κατά την επικοινωνία LoRa, η διόρθωση και ο έλεγχος σφαλμάτων που χρησιμοποιείται είναι η προς τα εμπρός διόρθωση με τη μορφή κώδικα Hamming, επηρεάζοντας τον ρυθμό bit του σήματος της πληροφορίας.

Interleaving: Ουσιαστικά αφορά τη χρήση ενός μοντέλου που αναδιατάσσει τα bits πληροφορίας στην έξοδο του κωδικοποιητή FEC, με σκοπό την αποφυγή λαθών κατά ριπές (burst errors). Ένα τέτοιο σύστημα είναι απαραίτητο καθώς τέτοιου είδους σφάλματα παρουσιάζονται ομαδικά σε συγκεκριμένα χρονικά διαστήματα και άρα πρέπει να μην συσχετίζονται χρονικά ώστε να μην μεταφέρονται στο πραγματικό σήμα. Συγκεκριμένα, στην εξεταζόμενη περίπτωση, το αποτέλεσμα είναι κάθε λέξη πληροφορίας που παράγεται είναι μετατοπισμένη ή έχει περιστραφεί κατά έναν αυθαίρετο αριθμό bits ενώ τα ίδια τα δυαδικά ψηφία εντός της λέξης είναι ανεστραμμένα.

Μορφή Πακέτου LoRa

Η δομή ενός πακέτου LoRa περιγράφει πρακτικά το τρόπο της επικοινωνίας. Στην αρχή του περιέχει το προοίμιο (preamble) το οποίο φροντίζει για τον συγχρονισμό μεταξύ του πομπού και του δέκτη. Μετά το προοίμιο, ακολουθεί μία προαιρετική κεφαλίδα (header) η οποία φέρει σε συγκεκριμένη κωδικοποιημένη μορφή το μέγεθος του ωφέλιμου φορτίου

προς μετάδοση και πληροφορίες σχετικά με τον τρόπο επικοινωνίας. Ύστερα ακολουθεί η πληροφορία ή payload προς μετάδοση ενώ στο τέλος υπάρχει δυνατότητα επίσης προαιρετικής προσθήκης ενός κυκλικού ελέγχου απόρριψης (Cyclic Redundancy Check – CRC) για τη πληροφορία αυτή. Τέλος, τέτοια πακέτα μπορούν να αναδιαμορφωθούν εάν ο δέκτης είναι εξαρχής ενημερωμένος για τα περιεχόμενα της κεφαλίδας, ορίζοντας την ύπαρξή της μη απαιτητή, χαμηλώνοντας τον χρόνο μετάδοσης δεδομένων από τον πομπό στο δέκτη [22].



Εικόνα Ε5: Μορφή Πακέτου LoRa

Γνωρίζοντας την παραπάνω μορφή πακέτου, μπορούμε να ορίσουμε ένα νέο μέγεθος, μέσω εξισώσεων, την Ενέργεια ανά χρήσιμο bit (E_{bit}) το οποίο αποτελεί μία πολύ σημαντική μέτρηση καθώς μας επιτρέπει την αξιολόγηση της κατανάλωσης ενέργειας στους κόμβους που χρησιμοποιούνται. Ορίζεται ως εξής:

$$E_{bit} = \frac{E_{Total}}{8 \cdot PL} = \frac{Pcons(P_{Tr}) \cdot T_{Packet}}{8 \cdot PL},$$

Εξίσωση 2.1

Όπου:

PL = το μέγεθος του ωφέλιμου φορτίου (payload size)

E_{Total} = συνολική κατανάλωση ενέργειας (total consumed energy)

$P_{cons}(P_{Tr})$ = Συνολική κατανάλωση ισχύος, εξαρτώμενη από την ισχύ μετάδοσης

Αναδιαμορφώντας τον παραπάνω τύπο, κάνοντας χρήση διάφορων εξισώσεων καταλήγουμε σε ένα αρκετά ενδιαφέρον συμπέρασμα. Πιο συγκεκριμένα, έχοντας ως δεδομένα τα παρακάτω [23]:

Transmission Power (dBm)	Power Consumption (mW)
20	412.5
17	297
13	92.4
7	95.4

Και

Και

$$T_{Payload} = N_{Payload} \cdot T_{Symbol}$$

Όπου

N_P = Αριθμός Συμβόλων του Προοίμιου (Preamble Symbol Number)

$N_{Payload}$ = Αριθμός Συμβόλων ωφέλιμου φορτίου και,

$T_S = T_{Symbol} = \frac{2^{SF}}{BW}$

Καταλήγουμε στον εξής τύπο για την E_{bit}:

$$E_{bit} = \frac{Pcons(P_{Tr}) \cdot (N_{Payload} + N_P + 4.25) \cdot 2^{SF}}{8 \cdot PL \cdot BW}$$

Το οποίο μας οδηγεί στο συμπέρασμα, πως ο παράγοντας διάδοσης είναι ανάλογος της ενέργειας που καταναλώνεται ανά χρήσιμο bit, άρα όσο μεγαλύτερος είναι ο παράγοντας διάδοσης που χρησιμοποιείται τόσο περισσότερη ενέργεια καταναλώνουμε για την επικοινωνία μέσω LoRa [23]. Ως παράδειγμα, παρατίθεται ο σχετικός πίνακας τιμών μεταξύ ισχύς μετάδοσης και ενεργειακής κατανάλωσης για το μοντέλο πομποδέκτη LoRa SX1272 [23]:

Πίνακας 1: Σχέση Ισχύος – Ενέργειας του πομποδέκτη LoRa SX1272

3.2 LoRaWAN

Δεδομένου όμως ότι η επικοινωνία LoRa, ορίζει το κατώτερο φυσικό επίπεδο, τα ανώτερα στρώματα δικτύωσης έλειπαν, περιορίζοντάς την επικοινωνία στην βασική της λειτουργία, δηλαδή την μετατροπή ραδιοσυχνοτήτων σε bits πληροφορίας. Εκεί εισέρχεται η έννοια του LoRaWAN, το οποίο είναι υπεύθυνο για τη διαχείριση της επικοινωνίας μεταξύ των πυλών (gateways) και των τερματικών κόμβων (nodes) ως πρωτόκολλο δρομολόγησης. Έχει δημιουργηθεί από μία ομάδα εταιριών και ερευνητικών κέντρων, ονόματι LoRa Alliance, με σκοπό την επικοινωνία με τη βέλτιστη δυνατή ενεργειακή κατανάλωση στους τελικούς κόμβους του.

Στην ουσία αποτελεί ένα δίκτυο από συσκευές που χρησιμοποιούν τη παραπάνω διαμόρφωση LoRa, και την αναπτύσσουν ώστε η πληροφορία ξεκινώντας από τις τερματικές συσκευές, να φθάνει να σκιαγραφείται μέσω dashboards ή ιστοσελίδες εμφάνισης δεδομένων στο cloud [24].

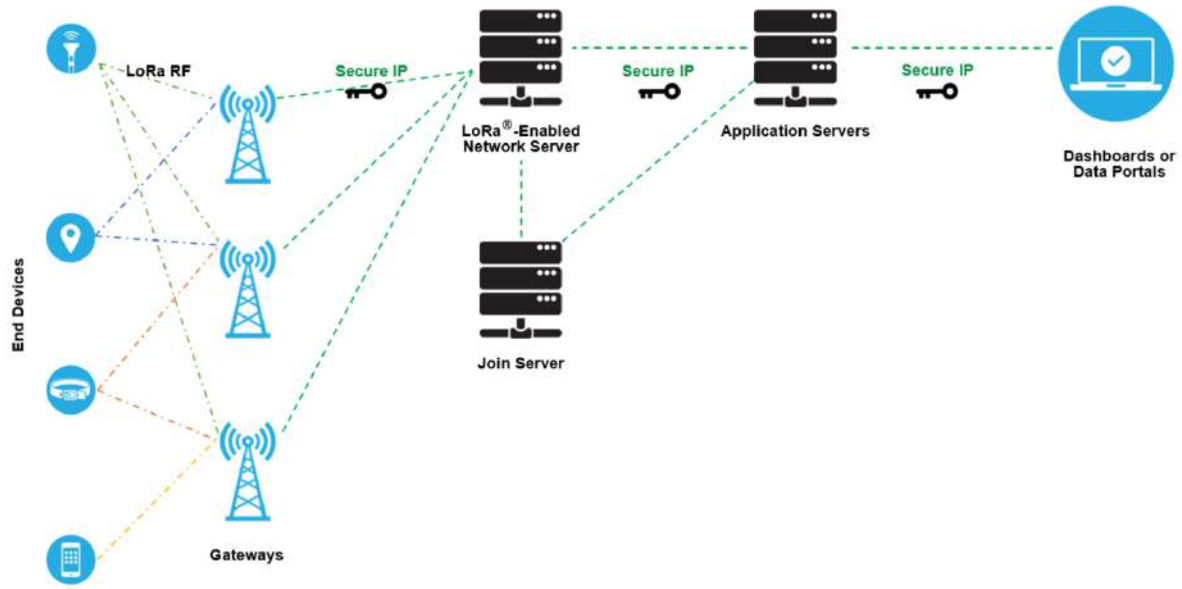
LoRaWAN Gateway

Πιο συγκεκριμένα, μία πύλη LoRaWAN (LoRaWAN **Gateway**) λαμβάνει διαμορφωμένα μηνύματα από μία τερματική συσκευή, η οποία μπορεί να είναι ένας αισθητήρας ή ένας ενεργοποιητής (actuator), και τα προωθεί σε ένα LoRaWAN δικτυακό server (LNS) μέσω του διαδικτύου. Η οποιαδήποτε επικοινωνία μεταξύ της gateway και του LNS μπορεί να επιτευχθεί μέσω Wi-Fi, ethernet ή 4G/5G. Οι πύλες LoRaWAN λειτουργούν εξ ολοκλήρου στο φυσικό επίπεδο και, στην ουσία, δεν είναι τίποτα άλλο παρά το πρόγραμμα προώθησης ράδιο-σήματος LoRa. Ελέγχουν μόνο την ακεραιότητα των δεδομένων κάθε εισερχόμενου μηνύματος RF LoRa. Εάν η ακεραιότητα δεν είναι ανέπαφη, δηλαδή, εάν ο CRC είναι λανθασμένος, το μήνυμα θα αγνοηθεί. Εάν διορθωθεί, η πύλη θα την προωθήσει στο LNS, μαζί με ορισμένα μεταδεδομένα που περιλαμβάνουν το επίπεδο λήψης RSSI (Received

Signal Strength Indication) του μηνύματος καθώς και μια προαιρετική χρονική σήμανση. Για downlinks LoRaWAN, μια πύλη εκτελεί αιτήματα μετάδοσης που προέρχονται από το LNS χωρίς καμία ερμηνεία του ωφέλιμου φορτίου. Δεδομένου ότι πολλές πύλες μπορούν να λάβουν το ίδιο μήνυμα LoRa RF από μία συσκευή, το LNS εκτελεί από-αντιγραφή δεδομένων και διαγράφει όλα τα αντίγραφα. Με βάση τα επίπεδα RSSI των ίδιων μηνυμάτων, ο διακομιστής δικτύου επιλέγει συνήθως την πύλη που έλαβε το μήνυμα με το καλύτερο RSSI κατά τη μετάδοση ενός μηνύματος downlink, επειδή αυτή η πύλη είναι εκείνη που βρίσκεται πλησιέστερα στην εν λόγω τελική συσκευή [25].

Διακομιστής Δικτύου LoRaWAN (LoRaWAN Network Server - LNS)

Ο διακομιστής δικτύου LoRaWAN (LNS) διαχειρίζεται ολόκληρο το δίκτυο, ελέγχει δυναμικά τις παραμέτρους δικτύου για να προσαρμόσει το σύστημα σε συνεχώς μεταβαλλόμενες συνθήκες και δημιουργεί ασφαλείς συνδέσεις AES 128-bit για τη μεταφορά και των δύο δεδομένων από άκρο σε άκρο (από την τελική συσκευή LoRaWAN στους τελικούς χρήστες της όποιας εφαρμογής στο Cloud) καθώς και για τον έλεγχο της ροής δεδομένων από την



Εικόνα Ε6: Μια τυπική υλοποίηση ενός LoRaWAN δικτύου

τελική συσκευή LoRaWAN στο LNS (και πίσω). Ο διακομιστής δικτύου διασφαλίζει την αυθεντικότητα κάθε αισθητήρα στο δίκτυο και την ακεραιότητα κάθε μηνύματος. Σημειώνεται εδώ ότι ο διακομιστής δικτύου δεν μπορεί να δει ή να αποκτήσει πρόσβαση

στα δεδομένα του server εφαρμογών ο οποίος είναι υπεύθυνος για τον ασφαλή χειρισμό, διαχείριση και την ερμηνεία δεδομένων εφαρμογής αισθητήρων. Ο τελευταίος παράγει επίσης όλα τα ωφέλιμα φορτία downlink επιπέδου εφαρμογής στις συνδεδεμένες τελικές συσκευές. Ενδιάμεσα εισέρχεται η έννοια του διακομιστή συμμετοχής (Join Server), ο οποίος πρακτικά διαχειρίζεται τη διαδικασία ενεργοποίησης over-the-air για την προσθήκη τερματικών συσκευών στο δίκτυο LoRaWAN. Περιέχει τις πληροφορίες που απαιτούνται για την επεξεργασία των πλαισίων αιτήματος συμμετοχής στην σύνδεση uplink και τη δημιουργία των πλαισίων συμμετοχής της σύνδεσης downlink. Σηματοδοτεί στον διακομιστή δικτύου ποιος διακομιστής εφαρμογών θα πρέπει να είναι συνδεδεμένος με την τελική συσκευή και εκτελεί παραδόσεις κλειδιών κρυπτογράφησης περιόδου λειτουργίας δικτύου και εφαρμογής. Επικοινωνεί το κλειδί συνεδρίας δικτύου της συσκευής στον διακομιστή δικτύου και το κλειδί περιόδου λειτουργίας εφαρμογής στον αντίστοιχο διακομιστή εφαρμογών [25].

Χαρακτηριστικά LoRaWAN δικτύου

1) Ανάθεση Συσκευής στο Δίκτυο

Για λόγους ασφάλειας, ποιότητας της επικοινωνίας, χρέωσης και άλλων αναγκών, οι συσκευές πρέπει να τεθούν σε λειτουργία και να ενεργοποιηθούν στο δίκτυο κατά την εκκίνησή τους. Η διαδικασία θέσης σε λειτουργία ευθυγραμμίζει με ασφάλεια κάθε συσκευή και το δίκτυο σε σχέση με τις βασικές παραμέτρους παροχής (όπως είναι τα αναγνωριστικά, τα κλειδιά κρυπτογράφησης και οι τοποθεσίες διακομιστών).
Η προδιαγραφή LoRaWAN επιτρέπει δύο τύπους ενεργοποίησης: **Over-the-Air Activation** (OTAA) (προτιμάται) και **Activation by Personalization** (ABP). Ο Πίνακας παρακάτω δείχνει τα διαφορετικά χαρακτηριστικά καθενός από αυτούς τους τύπους ενεργοποίησης.

Over-the-Air Activation (OTAA)	Activation by Personalization (ABP)
• Device manufacturers autonomously generate essential provisioning parameters • Secure keys (session-long and derived) can be renewed regularly • Devices can store multiple "identities" to dynamically and securely switch networks and operators during its lifetime • High-grade, tamper-proof security options are available	• A simplified (less secure) commissioning process • IDs and Keys are personalized at fabrication • Devices become immediately functional upon powering up; the Join procedure is skipped • Devices are tied to a specific network/service; the NetID is a portion of the device network address

Πίνακας 2: Διαφορές ενεργοποίησης με ABP και OTAA

2) Ασφάλεια Δικτύου

Υπάρχουν δύο βασικά στοιχεία για την ασφάλεια ενός δικτύου LoRaWAN: η διαδικασία ανάθεσης (παραπάνω) και ο έλεγχος ταυτότητας μηνυμάτων. Η διαδικασία ανάθεσης καθιερώνει αμοιβαίο έλεγχο ταυτότητας μεταξύ μιας τελικής συσκευής και του δικτύου LoRaWAN στο οποίο είναι συνδεδεμένη. Μόνο εξουσιοδοτημένες συσκευές επιτρέπεται να συμμετέχουν στο δίκτυο. Τα μηνύματα LoRaWAN MAC και εφαρμογής είναι πιστοποιημένα από προέλευση, προστατεύονται στην ακεραιότητά τους και κρυπτογραφούνται από άκρο σε άκρο (δηλαδή, από την τελική συσκευή στον διακομιστή εφαρμογών και το αντίστροφο). Αυτά τα χαρακτηριστικά ασφαλείας διασφαλίζουν ότι:

a. Η κίνηση στο δίκτυο δεν έχει αλλάξει
b. Μόνο νόμιμες συσκευές είναι συνδεδεμένες στο δίκτυο LoRaWAN
c. Δεν είναι δυνατή η υποκλοπή της κίνησης του δικτύου (eavesdropping)
d. Δεν είναι δυνατή η καταγραφή και η αναπαραγωγή της κυκλοφορίας δικτύου

Ο βασικός τρόπος λειτουργίας ενός δικτύου LoRaWAN είναι σαν ένα δίκτυο – αστέρι. Η πύλη (gateway) "μιλά" με όλους τους κόμβους του δικτύου και αντίστροφα. Σε όρους επικοινωνιών, η σχέση αυτή θεωρείται ασύμμετρη, καθώς όλοι οι κόμβοι μπορούν να επικοινωνήσουν με τη πύλη ταυτόχρονα αλλά η τελευταία δεν έχει τη δυνατότητα λήψης πληροφορίας από όλους.

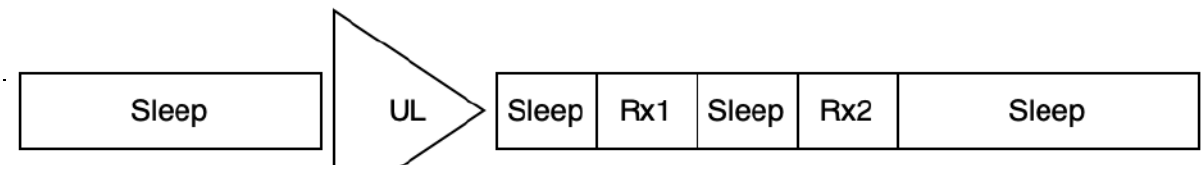
Αναλυτικότερα, φέρνουμε ως παράδειγμα το σενάριο 4 πυλών και 1 κόμβου στο δίκτυο: Ο κόμβος εκπέμπει “τυφλά” σε όλο ράδιο-φάσμα, πληροφορία και όποια πύλη βρίσκεται σε απόσταση επικοινωνίας, τη λαμβάνει και τη προωθεί. Αυτό έχει ως θετικό ότι η πιθανότητα μετάδοσης πληροφορίας αυξάνεται σε σενάρια αδύναμων συνδέσεων καθώς αν ο κόμβος εκπέμψει 4 μηνύματα, επί παραδείγματι, και μόνο ένα καταφέρει να σταλεί, η επικοινωνία ολοκληρώνεται κανονικά. Συνήθως, κατά τη διαδικασία επικοινωνίας μεταξύ κόμβου και πύλης δεν υπάρχει λειτουργία επιβεβαίωσης λήψης μηνύματος (acknowledgements) καθώς αν μπει η πύλη σε αυτή τη διαδικασία, σταματά να “ακούει” το δίκτυο για νέες προσπάθειες επικοινωνίας. Τα συστήματα LoRaWAN, διαθέτουν πολλά διαφορετικά κανάλια λήψης πληροφορίας ώστε να μπορούν να δεχθούν ταυτόχρονα έως και 8 μηνύματα.

Με περισσότερες από 500 εταιρείες μέλη, η LoRa Alliance είναι μία από τις ταχύτερα αναπτυσσόμενες τεχνολογικές συμμαχίες. Όντας μια κοινότητα καινοτόμων, η LoRa Alliance δεσμεύεται να τυποποιήσει δίκτυα χαμηλής ισχύος (LPWANs). Για το σκοπό αυτό, η ομάδα παρέχει δωρεάν τις Προδιαγραφές LoRaWAN.Η προδιαγραφή βασίζεται σε ανοιχτά πρότυπα και παρέχει πιστοποιημένη διαλειτουργικότητα.

3.3 Κλάσεις Συσκευών στο LoRaWAN δίκτυο

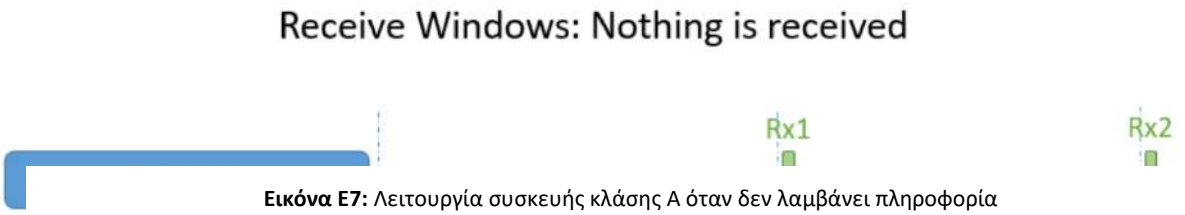
Συσκευές κλάσης Α

Σε αυτήν την περίπτωση, η τελική συσκευή περνά τον περισσότερο χρόνο της σε κατάσταση αδράνειας, (δηλαδή σε κατάσταση αναστολής λειτουργίας). Όταν υπάρχει μια αλλαγή στο περιβάλλον που σχετίζεται με ό, τι έχει προγραμματιστεί να παρακολουθεί η συσκευή, ξυπνά και ξεκινά μια σύνδεση, μεταδίδοντας τα δεδομένα σχετικά με την αλλαγή κατάσταση στο δίκτυο (Tx). Στη συνέχεια, η συσκευή ακούει μια απόκριση από το δίκτυο, συνήθως για ένα δευτερόλεπτο (αν και αυτή η διάρκεια είναι διαμορφώσιμη). Εάν δεν λαμβάνει downlink κατά τη διάρκεια αυτού του παραθύρου λήψης (Rx1), επιστρέφει για λίγο σε κατάσταση αναμονής, εκκινώντας ξανά λίγο αργότερα, και πάλι ακούγοντας μια απάντηση (Rx2). Εάν δεν ληφθεί απάντηση κατά τη διάρκεια αυτού του δεύτερου παραθύρου Rx, η συσκευή επιστρέφει στον ύπνο μέχρι την επόμενη φορά που θα έχει δεδομένα για αναφορά. Η καθυστέρηση μεταξύ Rx1 και Rx2 διαμορφώνεται με όρους



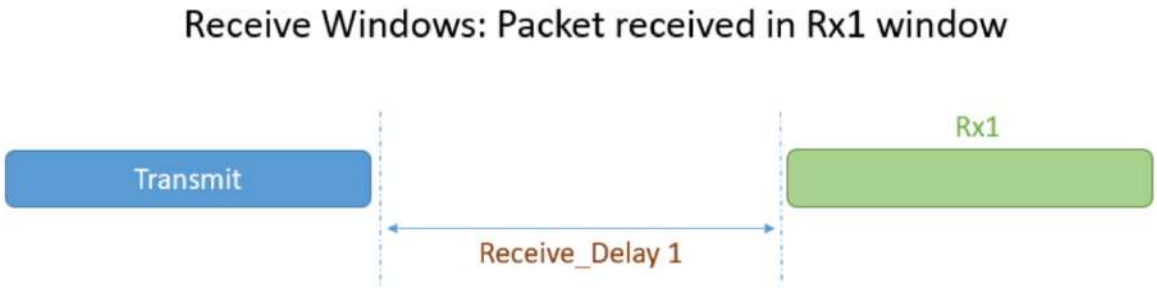
Σχήμα 2: Αναπαράσταση λειτουργία συσκευής κλάσης Α σε ένα δίκτυο LoRaWAN

καθυστέρησης από το τέλος της μετάδοσης uplink. Σημειώνεται ότι δεν υπάρχει τρόπος η εφαρμογή της τελικής συσκευής να ξυπνήσει μια συσκευή κλάσης Α. Λόγω αυτού του περιορισμού, οι συσκευές κλάσης Α δεν είναι κατάλληλες για ενεργοποιητές (actuators).

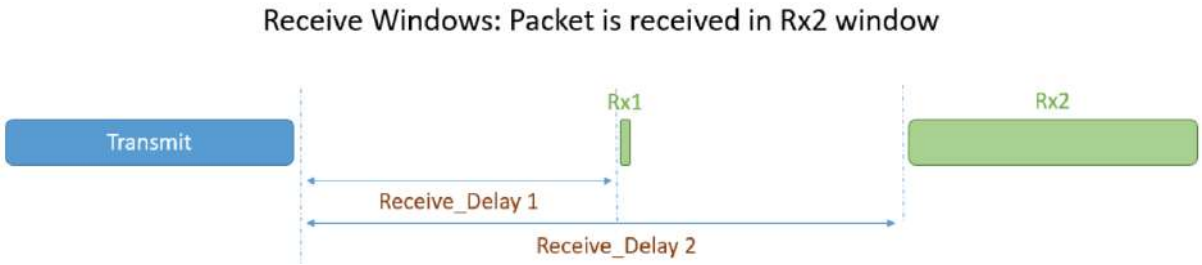


Εικόνα Ε7: Λειτουργία συσκευής κλάσης Α όταν δεν λαμβάνει πληροφορία

Εικόνα Ε8: Λειτουργία συσκευής κλάσης Α όταν λαμβάνει πληροφορία στο παράθυρο επικοινωνίας RX1



Όπως παρατηρείται από τις παραπάνω εικόνες Ε7 και Ε8, οι συσκευές τύπου Α ξεκινούν πάντα τη διαδικασία επικοινωνίας με το δίκτυο όταν έχουν κάποια πληροφορία να μεταδώσουν ή εφόσον έχουν προγραμματιστεί να στέλνουν κάποιο μήνυμα τύπου "hello" στο δίκτυο ανά περιοδικά χρονικά διαστήματα. Εξάγεται τέλος, εύκολα το συμπέρασμα, ότι εφόσον το uplink packet κατά τη διαδικασία UL παραδοθεί επιτυχώς, τότε είναι εγγυημένη η δυνατότητα του δικτύου να στείλει πακέτο επικοινωνίας πίσω στη συσκευή, εφόσον



Εικόνα Ε9: Λειτουργία συσκευής κλάσης Α όταν λαμβάνει πληροφορία στο παράθυρο επικοινωνίας RX2

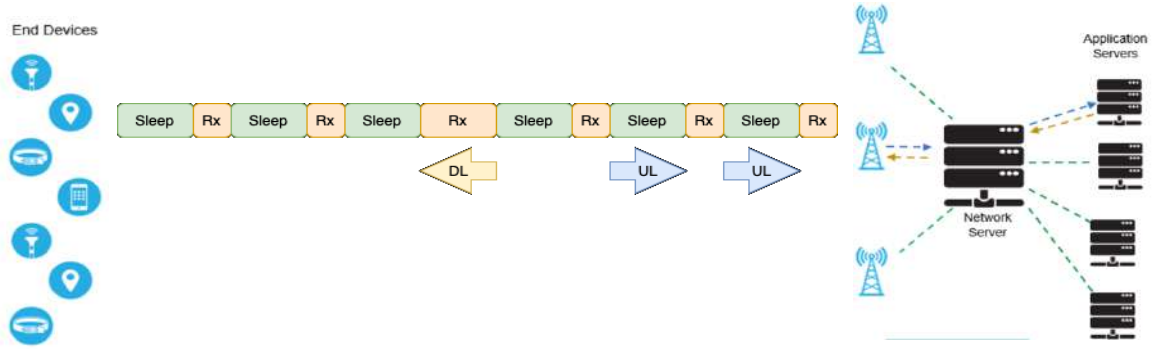
χρειάζεται.

Σημειώνεται ότι σε αυτό το σενάριο η συσκευή δεν θα δοκιμάσει να στείλει πληροφορία στο δίκτυο (uplink) εκτός εάν ικανοποιούνται μία από τις εξής συνθήκες: Λήψη της απόκρισης από το δίκτυο κατά τη διάρκεια του παράθυρου επικοινωνίας Rx1 ή εφόσον ολοκληρωθεί επιτυχώς η επικοινωνία με το δίκτυο κατά το δεύτερο παράθυρο επικοινωνίας Rx2.

Συσκευές κλάσης Β

Η βελτίωση της λειτουργίας Class A, LoRaWAN Class B προσφέρει τακτικές προγραμματισμένες, σταθερού χρόνου ευκαιρίες για μια τελική συσκευή να λαμβάνει downlinks από το δίκτυο, καθιστώντας τις τελικές συσκευές Class B κατάλληλες τόσο για αισθητήρες παρακολούθησης όσο και για ενεργοποιητές. Όλες οι τελικές συσκευές που βασίζονται σε LoRa ξεκινούν σε κατάσταση κλάσης Α. Ωστόσο, οι συσκευές που έχουν προγραμματιστεί με μια στοίβα κατηγορίας Β κατά τη διάρκεια της κατασκευής μπορούν να μεταβούν σε λειτουργία κλάσης Β από κάποια εφαρμογή. Τέτοιες τελικές συσκευές σε λειτουργία κλάσης Β παρέχουν τακτικά προγραμματισμένα παράθυρα λήψης, πέραν εκείνων που ανοίγουν κάθε φορά που αποστέλλεται ένας ανερχόμενος σύνδεσμος τύπου Α.

Για να λειτουργήσει ο τρόπος επικοινωνίας της κλάσης Β, απαιτείται μία διαδικασία που ονομάζεται “beaconing”. Κατά τη διάρκεια της διαδικασίας αυτής, ένας συγχρονισμένος “beacon” εκπέμπεται περιοδικά από το δίκτυο μέσω των πυλών-gateways. Η τελική συσκευή λαμβάνει επίσης περιοδικά αυτά τα σήματα δικτύου έτσι ώστε να μπορεί να ευθυγραμμίσει την εσωτερική αναφορά χρονισμού με το δίκτυο. Συνεπώς, οι συσκευές χρησιμοποιούν beacons για την παραγωγή και την ευθυγράμμιση των εσωτερικών ρολογιών τους με το δίκτυο. Η διαδικασία αυτή δεν χρειάζεται να συμβαίνει συνεχώς, εάν η συσκευή είναι ήδη ευθυγραμμισμένη. Στις περισσότερες περιπτώσεις αρκεί επανευθυγράμμιση αρκετές φορές την ημέρα, με ελάχιστη επίδραση στην διάρκεια ζωής της μπαταρίας. Με βάση την αναφορά χρονισμού του beacon, οι τελικές συσκευές μπορούν να ανοίγουν τα παράθυρα λήψης (ping slots) περιοδικά. Οποιαδήποτε από αυτές τις υποδοχές ping μπορεί να χρησιμοποιηθεί από την υποδομή δικτύου για να ξεκινήσει μια επικοινωνία κάτω ζεύξης. Προκειμένου ένα δίκτυο LoRaWAN να υποστηρίζει συσκευές κλάσης Β, όλες οι πύλες-gateways LoRaWAN σε αυτό το δίκτυο πρέπει να έχουν ενσωματωμένη Πηγή χρονισμού GPS, έτσι ώστε όλοι ο κόμβοι του δικτύου να μπορούν να συγχρονιστούν με τον ακριβή συγχρονισμό των φάρων.

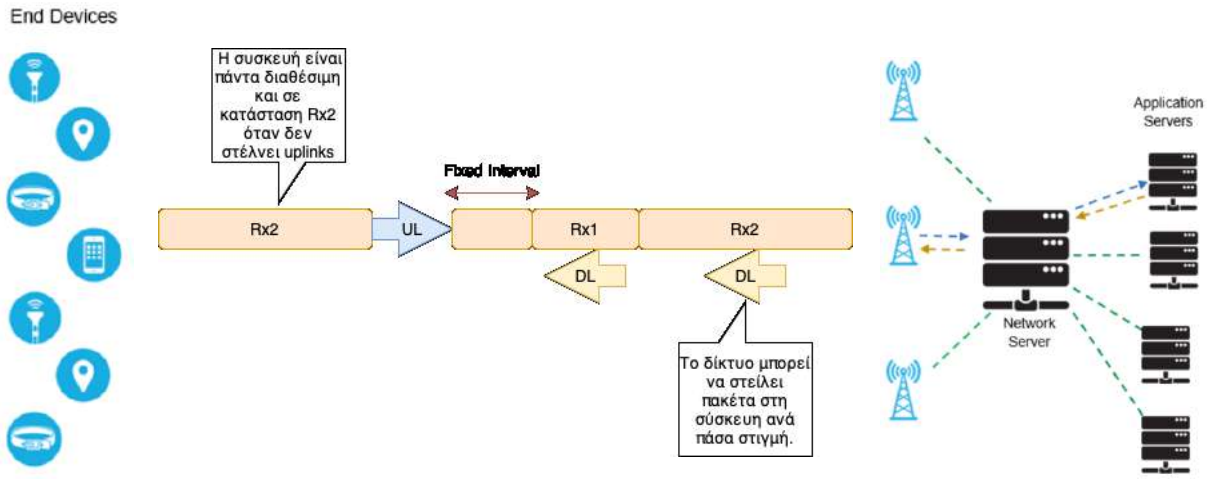


Εικόνα Ε10: Λειτουργία συσκευής κλάσης Β στο δίκτυο με περιοδικά ενεργοποιημένα παράθυρα επικοινωνίας DL αλλά και τυχαία uplink packets προς το δίκτυο

Συσκευές κλάσης C

Η τάξη Γ είναι πάντα "ενεργοποιημένη". Δηλαδή, δεν εξαρτώνται από την ισχύ της μπαταρίας. Οι συσκευές της κατηγορίας C περιλαμβάνουν πράγματα όπως φώτα του δρόμου, ηλεκτρικά μέτρα κ.λπ. Ως αποτέλεσμα, προσφέρουν τη χαμηλότερη καθυστέρηση για επικοινωνία από το διακομιστή σε μια τελική συσκευή.

Οι συσκευές τελικής κλάσης C εφαρμόζουν τα ίδια δύο παράθυρα λήψης με τις συσκευές κλάσης A, αλλά δεν κλείνουν το παράθυρο Rx2 μέχρι να στείλουν την επόμενη μετάδοση πίσω στο διακομιστή. Επομένως, μπορούν να λάβουν μια κατερχόμενη σύνδεση στο παράθυρο Rx2 σχεδόν ανά πάσα στιγμή. Ένα μικρό παράθυρο στη συχνότητα Rx2 και στον ρυθμό δεδομένων ανοίγεται επίσης μεταξύ του τέλους της μετάδοσης και της έναρξης του παραθύρου λήψης Rx1.



Εικόνα E11: Λειτουργία συσκευής κλάσης C στο δίκτυο

3.4 Ασφάλεια LoRaWAN (ABP vs OTAA)

Δεδομένου ότι η ενσωμάτωση ασφάλειας είναι εξαιρετικά σημαντική για οποιοδήποτε ασύρματο δίκτυο, το LoRaWAN χρησιμοποιεί δύο επίπεδα ασφάλειας, ένα για το επίπεδο δικτύου και ένα για το επίπεδο εφαρμογής. Η ασφάλεια επιπέδου δικτύου διασφαλίζει την αυθεντικότητα της συσκευής στο δίκτυο. Η ασφάλεια επιπέδου εφαρμογής διασφαλίζει ότι ο διαχειριστής δικτύου δεν έχει πρόσβαση στα δεδομένα εφαρμογής του τελικού χρήστη. Μια τερματική συσκευή (κόμβος δικτύου) πρέπει να ενεργοποιηθεί προτού επικοινωνήσει μέσω του δικτύου LoRaWAN. Υπάρχουν δύο μέθοδοι ενεργοποίησης στα δίκτυα LoRaWAN: OTAA (Over The Air Activation) και ABP (Activation by Personalization).

Μέθοδος Ενεργοποίησης OTAA

Αυτή η μέθοδος OTAA βασίζεται στη συνεργασία μεταξύ μηνυμάτων Over-the-Air για αιτήσεις συμμετοχής και αποδεκτές συμμετοχής. Για κάθε φορητή συσκευή παρέχονται ένα DevEUI 64-bit, ένα AppEUI 64-bit και ένα AppKey 128-bit. Το DevEUI είναι ένα μοναδικό, καθολικό αναγνωριστικό για τη συσκευή που είναι συγκρίσιμο με τη διεύθυνση MAC μιας συσκευής TCP / IP. Το AppKey χρησιμοποιείται για την κρυπτογραφική σχεδίαση του αιτήματος συμμετοχής. Το AppKey χρησιμοποιείται όταν ο κόμβος δικτύου στέλνει ένα μήνυμα αίτησης συμμετοχής. Ο κόμβος δικτύου στέλνει ένα μήνυμα αίτησης συμμετοχής που αποτελείται από το AppEUI και το DevEUI. Στέλνει επίσης ένα DevNonce - μια μοναδική, τυχαία παραγόμενη τιμή των 2 bytes που χρησιμοποιείται για την αποτροπή επιθέσεων επανάληψης (replay attacks). Αυτές οι τρεις τιμές αναγνωρίζονται μέσω του AppKey της συσκευής με MIC 4-byte (Κωδικός ακεραιότητας μηνυμάτων). Ο διακομιστής δέχεται μόνο αιτήματα συμμετοχής από συσκευές με γνωστές τιμές DevEUI και AppEUI και ελέγχει το MIC αντιστοιχίζοντας το με το AppKey [26].

Όταν ο διακομιστής αποδέχεται το αίτημα συμμετοχής, αποκρίνεται στη συσκευή με ένα μήνυμα αποδοχής συμμετοχής. Οι διακομιστές εφαρμογών και δικτύου υπολογίζουν τα δύο κλειδιά 128-bit του κόμβου δικτύου: το κλειδί περιόδου λειτουργίας εφαρμογής (AppSKey) και το κλειδί σύνδεσης δικτύου (NwkSKey). Αυτά υπολογίζονται με βάση τις τιμές που αποστέλλονται από τον κόμβο δικτύου στο μήνυμα αίτησης συμμετοχής. Επιπλέον, ο διακομιστής εφαρμογών δημιουργεί τη δική του τιμή nonce: AppNonce. Αυτή είναι μια άλλη μοναδική, τυχαία παραγόμενη τιμή [26].

Η απόκριση join-accept περιέχει το AppNonce, ένα NetID και μια διεύθυνση της τερματικής συσκευής (DevAddr) μαζί με δεδομένα διαμόρφωσης για καθυστερήσεις RF (RxDelay) και τα κανάλια που θα χρησιμοποιηθούν (CFList). Το NetID είναι σημαντικό για συσκευές κλάσης B LoRaWAN. Είναι μια τιμή 3 byte που επιτρέπει στους κόμβους δικτύου να αναγνωρίζουν ότι ένα μήνυμα έχει σταλεί από μια εσωτερική πύλη δικτύου. Επομένως, πρέπει να έχει ένα μοναδικό αναγνωριστικό για το αντίστοιχο LoRaWAN. Η διεύθυνση της τερματικής συσκευής (DevAddr) στην απόκριση σύνδεσης-αποδοχής είναι ένα αναγνωριστικό 32-bit που είναι μοναδικό στο δίκτυο.

Είναι δυνατή η διάκριση μεταξύ τελικών συσκευών που έχουν ήδη συνδεθεί στο δίκτυο μέσω της διεύθυνσης της συσκευής. Αυτό επιτρέπει στους διακομιστές δικτύου και εφαρμογών να χρησιμοποιούν τη σωστή κρυπτογράφηση και να ερμηνεύουν σωστά τα δεδομένα.

Όταν τα δεδομένα λαμβάνονται πίσω, κρυπτογραφούνται με το AppKey. Στη συνέχεια, ο κόμβος δικτύου χρησιμοποιεί το AppKey για να αποκρυπτογραφήσει τα δεδομένα και αντλεί το AppSKey και το NwkSKey από το μήνυμα σύνδεσης-αποδοχής χρησιμοποιώντας την τιμή AppNonce. [26]

Μέθοδος Ενεργοποίησης ABP

Αυτή η μέθοδος ABP διαφέρει από την OTAA, επειδή οι κόμβοι δικτύου αποστέλλονται μαζί με το DevAddr και τα δύο κλειδιά περιόδου λειτουργίας (NwkSKey και AppSKey), τα οποία είναι μοναδικά στον κόμβο δικτύου. Εφόσον οι κόμβοι δικτύου έχουν ήδη τις απαιτούμενες πληροφορίες και κλειδιά, μπορούν να ξεκινήσουν την επικοινωνία με τον διακομιστή δικτύου χωρίς να χρειάζεται να ανταλλάξουν προηγουμένως μηνύματα σύνδεσης. Μόλις ένας κόμβος δικτύου ενταχθεί σε ένα LoRaWAN - είτε μέσω OTAA είτε ABP - όλα τα μελλοντικά μηνύματα κρυπτογραφούνται με συνδυασμό διαφορετικών πλήκτρων:

Κλειδί σύνδεσης δικτύου (NwkSKey) - ένας μηχανισμός ασφαλείας του επιπέδου δικτύου. Αυτό το κλειδί είναι μοναδικό για κάθε τελική συσκευή και είναι κοινόχρηστο μεταξύ της τελικής συσκευής και του διακομιστή δικτύου. Το κλειδί σύνδεσης δικτύου διασφαλίζει την ακεραιότητα των μηνυμάτων κατά την επικοινωνία και διασφαλίζει την επικοινωνία από τις τελικές συσκευές στον διακομιστή δικτύου [26].

Application Session Key (AppSKey) - Αυτό το κλειδί είναι υπεύθυνο για την κρυπτογράφηση των δεδομένων χρήστη από άκρη σε άκρο (συσκευή σε συσκευή). Είναι επίσης ένα κλειδί AES 128-bit που είναι μοναδικό για κάθε συσκευή και είναι κοινόχρηστο μεταξύ της συσκευής και του διακομιστή εφαρμογών. Το κλειδί περιόδου λειτουργίας εφαρμογής κρυπτογραφεί και αποκρυπτογραφεί μηνύματα δεδομένων εφαρμογής και ασφαλίζει τα δεδομένα χρήστη της εφαρμογής. [26]

Αυτά τα δύο κλειδιά περιόδου λειτουργίας (NwkSKey και AppSKey) είναι μοναδικά για κάθε συσκευή και περίοδο σύνδεσης. Εάν η συσκευή ενεργοποιείται δυναμικά (OTAA), αυτά τα πλήκτρα δημιουργούνται ξανά κάθε φορά που ενεργοποιείται. Εάν η συσκευή ενεργοποιηθεί στατικά (ABP), αυτά τα πλήκτρα παραμένουν μέχρι να αλλάξουν χειροκίνητα. [26]

Πλεονεκτήματα των λειτουργιών ασφαλείας του προτύπου LoRaWAN

- Τα κλειδιά και τα πιστοποιητικά για κάθε περίοδο λειτουργίας μεταφέρονται δυναμικά μεταξύ μιας συσκευής και των διακομιστών δικτύου και εφαρμογών. Καθώς οι συσκευές επανέρχονται στα δίκτυα, τα κλειδιά περιόδου λειτουργίας τους αλλάζουν περιοδικά. Αυτό είναι σημαντικό για την αποφυγή πιθανών κινδύνων, όπως πλαστογράφηση, μετριασμός κ.λπ.
- Οι δυναμικά ενεργοποιημένες συσκευές (OTAA) χρησιμοποιούν το κλειδί εφαρμογής (AppKey) κατά τη διαδικασία ενεργοποίησης για να αντλήσουν τα δύο κλειδιά Συνεδρίας. Στο δίκτυο, η τιμή τους έχει οριστεί στο προεπιλεγμένο AppKey, το οποίο στη συνέχεια χρησιμοποιείται για την ενεργοποίηση όλων των συσκευών. Συνιστάται η εφαρμογή ενός μεμονωμένου AppKey για κάθε συσκευή. Ο αποφασιστικός παράγοντας είναι ότι σε ολόκληρη τη διαδικασία ασφαλείας τα κλειδιά δεν αποστέλλονται ποτέ χρησιμοποιώντας τη μέθοδο over-the-air. Μόνο τα μέρη του υπολογισμού που λείπουν ανταλλάσσονται και από τις δύο πλευρές. Η δημιουργία κλειδιών με την παρεμπόδιση της εναέριας κυκλοφορίας είναι επομένως εξαιρετικά περίπλοκη.
- Η παρουσία ενός στοιχείου υλικού για την αποθήκευση διαπιστευτηρίων ασφαλείας στη συσκευή είναι επιπλέον επωφελής, καθώς η εξαγωγή κλειδιών με αντίστροφη μηχανική ή σάρωση αναμνήσεων συσκευών γίνεται πολύ πιο δύσκολη. Επιπλέον, η ασφαλής χρήση εκκίνησης διασφαλίζει θεωρητικά την ακεραιότητα του υλικολογισμικού της συσκευής. Συνιστάται μολαταύτα πάντοτε ένα επιπλέον επίπεδο κρυπτογράφησης σε επίπεδο εφαρμογής και θα πρέπει να εφαρμόζεται εάν είναι απαραίτητο.
- Για να αποφευχθούν επιθέσεις επανάληψης (replay attacks), είναι σημαντικό να ενεργοποιηθεί η σάρωση μετρητή uplink / downlink

στο δίκτυο. Η σύνδεση ασύρματου δικτύου εξακολουθεί να αφήνει την πιθανότητα κακόβουλης λήψης και αποθήκευσης μηνυμάτων, ωστόσο, η κρυπτογράφηση των μηνυμάτων καθιστά αδύνατη την ανάγνωσή τους χωρίς το AppSKey. Η ανταλλαγή μηνυμάτων σε επίπεδο δικτύου δεν είναι επίσης δυνατή χωρίς το NwkSKey, καθώς η επαλήθευση MIC δεν θα ήταν επιτυχής σε αυτήν την περίπτωση. Αυτές οι αποκαλούμενες "επιθέσεις επανάληψης" μπορούν να εντοπιστούν και να αποκλειστούν από μετρητές πλαισίων. Εάν επί παραδείγματι, μια συσκευή είναι ενεργοποιημένη, οι δύο μετρητές πλαισίων (FCntUp και FCntDown) ρυθμίζονται στο 0. Κάθε φορά που η συσκευή μεταδίδει ένα μήνυμα ανερχόμενης ζεύξης (uplink message), το FCntUp αυξάνεται. Αντίστοιχα κάθε φορά που ο διακομιστής δικτύου μεταδίδει ένα μήνυμα κατερχόμενης ζεύξης (downlink message), το FCntDown αυξάνεται. Εάν η συσκευή ή το δίκτυο λάβει ένα μήνυμα του οποίου ο αριθμός πλαισίου είναι χαμηλότερος από ό,τι στην τελευταία εγγραφή, το μήνυμα αγνοείται.

Αδυναμίες των λειτουργιών ασφαλείας του προτύπου LoRaWAN

- Τα κρυπτογραφημένα μηνύματα έχουν το ίδιο μήκος με το κλειδί.
- Η μέθοδος ABP ορίζει και τα δύο κλειδιά περιόδου λειτουργίας μόνιμα, γεγονός που οδηγεί σε αδυναμίες.
- Η αποθήκευση των κλειδιών LoRaWAN (AppKey, NwkSKey και AppSKey) είναι ένα θεμελιώδες πρόβλημα. Υπάρχουν αρκετές ευπάθειες όσον αφορά τη διαχείριση του κύριου κύκλου ζωής, τη δημιουργία κλειδιών συνεδρίας, την αποθήκευση και τη μεταφορά, που απαιτούν προσεκτική ανάπτυξη και εφαρμογή. Με γνωστές επιθέσεις καναλιού, τα κλειδιά είναι δυνατόν να ανακτηθούν από τη μνήμη της συσκευής. Προτείνεται για το λόγο αυτό, όλα τα κλειδιά της συσκευής να είναι κατάλληλα ασφαλισμένα, π.χ., με κρυπτογράφηση με κύρια κλειδιά, για τη μείωση του κινδύνου αποκάλυψής τους.
- Μια άλλη αδυναμία είναι οι διαδικασίες αναγνώρισης και σύνδεσης. Κάθε πύλη στέλνει τακτικά beacons (μεμονωμένο αναγνωριστικό) στον διακομιστή. Αυτό το αναγνωριστικό μπορεί εύκολα να ανακτηθεί από εισβολείς. Εάν το αναγνωριστικό είναι γνωστό, η πύλη μπορεί να παραβιαστεί, για

παράδειγμα με μια κακόβουλη πύλη που στέλνει το αναγνωριστικό της πιο συχνά από την πραγματική.

Συμπέρασμα και προοπτικές

Το τρέχον πρότυπο LoRaWAN 1.0.x περιλαμβάνει ήδη σχετικά υψηλό επίπεδο ασφάλειας. Οι κίνδυνοι και τα σενάρια επίθεσης που αναφέρονται παραπάνω είναι αρκετά δικαιολογημένα και πρέπει να αξιολογούνται ξεχωριστά για κάθε περίπτωση χρήσης.
Οι αλλαγές στην προδιαγραφή LoRaWAN 1.1 (Οκτώβριος 2017) επηρεάζουν τις τελικές συσκευές καθώς και τις πύλες και τους διακομιστές. Οι πιο σημαντικές αλλαγές είναι στην επικοινωνία μεταξύ τερματικής συσκευής και πύλης καθώς και στην επικοινωνία μεταξύ των διακομιστών. [27]
Για παράδειγμα, ο μετρητής πλαισίου Uplink (FCntUp) συμπεριλήφθηκε σε κρυπτογραφημένη μορφή σε μηνύματα επιβεβαίωσης (ACK Downlinks). Με τη βοήθεια αυτής της νέας δυνατότητας, η τερματική συσκευή μπορεί να αναγνωρίσει ποια από τα μηνύματά της έχουν επιβεβαιωθεί από το διακομιστή δικτύου. Περαιτέρω βελτιώσεις ασφαλείας περιλαμβάνουν τον σαφή διαχωρισμό μεταξύ διακομιστή δικτύου, διακομιστή σύνδεσης και διακομιστή εφαρμογών και την εισαγωγή πρόσθετων κλειδιών όπως NwkKey, NwkSEncKey, SNwkSIntKey, FNwkSIntKey και AppSKey. Αυτά τα νέα κλειδιά είναι ιδιαίτερα σημαντικά για το νέο Handover Roaming, το οποίο επιτρέπει την προώθηση ενεργών μηνυμάτων από τελικές συσκευές μέσω ξένων δικτύων LoRaWAN στο οικιακό δίκτυο της τελικής συσκευής. Ιδιαίτερη προσοχή δόθηκε επίσης στους "διακομιστές σύνδεσης τρίτων μερών (third party join servers)". Αυτούς τους διακομιστές σύνδεσης δεν τους διαχειρίζονται πλέον χειριστές των διακομιστών δικτύου, αλλά παρέχονται από έναν αξιόπιστο τρίτο οργανισμό. Η διαχείριση όλων των κλειδιών που σχετίζονται με την ασφάλεια των κινητών συσκευών γίνεται στους διακομιστές σύνδεσης, είτε από τον διακομιστή δικτύου που είναι υπεύθυνος για τη μεταφορά των μηνυμάτων είτε από τον διακομιστή εφαρμογών που είναι υπεύθυνος για την επεξεργασία των μηνυμάτων. Ο σαφής διαχωρισμός αυτών των ευθυνών εμποδίζει τον χειριστή του δικτύου να διαβάσει το περιεχόμενο (ωφέλιμο φορτίο - payload) των μηνυμάτων LoRaWAN.

4

Σενάριο Υλοποίησης Εργασίας

4.1 Παρουσίαση Προβλήματος

Η συγκεκριμένη υλοποίηση αφορά την επίτευξη επικοινωνιών συγκεκριμένου σκοπού ανάμεσα σε 2 διαφορετικά κτίρια της ίδιας επιχείρησης με πραγματική απόσταση μεταξύ τους λίγο περισσότερο από ένα οικοδομικό τετράγωνο και παρεμβαλλόμενα κτίρια ανάμεσα. Η πρώτη εγκατάσταση αποτελεί χώρο αποθήκευσης αρχείων και επίδειξης εξοπλισμού. Η δεύτερη είναι πρακτικά το κέντρο λειτουργιών της επιχείρησης όπου στεγάζεται το προσωπικό, ο ενεργός εξοπλισμός και τα τμήματά της. Η διαφορά μεταξύ των δύο είναι ότι στο πρώτο δεν υπάρχει δικτυακή υποδομή ώστε να υπάρχει επικοινωνία αλλά απαιτήθηκε παρακολούθηση των περιβαλλοντολογικών συνθηκών εντός αυτού, λόγω πιστοποίησης ISO ασφάλειας δεδομένων που διατηρεί η εν λόγω επιχείρηση. Συγκεκριμένα, στο χώρο αυτό αποθηκεύονται αποδεικτικά δελτία τεχνικών εργασιών και νόμιμα παραστατικά που παγιώνουν τις όποιες συναλλαγές της εταιρίας, σύμφωνα με την κείμενη ελληνική νομοθεσία και τις συμβάσεις που διατηρεί. Σημαντικό αίτημα αποτέλεσε επίσης η διατήρηση ιστορικού των μετρήσεων για έλεγχο αυτών καθ’ όλη τη διάρκεια της ημερολογιακής ημέρας ώστε να υπάρχει η δυνατότητα συλλογής στοιχείων σε περίπτωση ανάδυσης καταστροφικού σεναρίου.

Αρχικά αξιολογήθηκαν αρκετά σενάρια υλοποίησης για την κάλυψη της παραπάνω ανάγκης. Μία από αυτές ήταν και η καλωδιακή σύνδεση των δύο κτιρίων μέσω οπτικής ίνας, η οποία απορρίφθηκε λόγω κόστους τόσο της διασύνδεσης όσο και του απαιτούμενου εξοπλισμού. Το ίδιο συνέβη και με τη λύση της δημιουργίας νέας σύνδεσης του κτιρίου στο διαδίκτυο, μέσω παρόχων, η οποία επίσης υπερκαλύπτει τις ανάγκες που χρειάζεται να ικανοποιηθούν και τέλος δεν αφαίρει μεγάλο τμήμα των παραμετροποιήσεων που απαιτούνται για την ορθή λειτουργία της επικοινωνίας μεταξύ των δύο κτιρίων.

Στρεφόμενοι σε ασύρματες λύσεις, η κατάληξη στη χρήση της τεχνολογίας IoT ήταν μονόδρομος. Δεδομένης της απόστασης μεταξύ των δύο κτιρίων, της τοπολογίας που παρεμβάλλεται, των «πράσινων στόχων» που διατηρεί η εν λόγω επιχείρηση, τη λειτουργική σταθερότητα που απαιτείται σε ένα τέτοιο εγχείρημα και την προσπάθεια ελαχιστοποίησης του ανθρώπινου παράγοντα όσον αφορά τη χρήση της υποδομής, έγινε η πρώτη επαφή με τις τεχνολογίες LoRa και LoRaWAN.

Μέσω αυτής, οι αισθητήρες θερμοκρασίας και υγρασίας θα εγκαθίστανται εντός του χώρου αρχειοθέτησης για τη μέτρηση - συλλογή δεδομένων και θα έστελναν τα τελευταία, ασύρματα, απευθείας στη πύλη (gateway) η οποία θα βρισκόταν στο 2^{ο} κτίριο με σταθερή σύνδεση στο Διαδίκτυο με σκοπό τη προώθηση τους σε έναν διακομιστή δικτύου προκειμένου τον έλεγχο και της διαχείρισης αυτών. Ύστερα όλη αυτή η κίνηση δεδομένων θα προωθούνταν σε έναν διακομιστή εφαρμογών με σκοπό την επεξεργασία και τον έλεγχο αυτών από τον ανθρώπινο παράγοντα.

Οι μετρήσεις δεν χρειαζόταν να είναι περισσότερο συχνές από μία ανά ώρα, έτσι ώστε να υπάρχει μία ακριβής αλλά όχι αναίτια λεπτομερής αναφορά για τις περιβαλλοντικές συνθήκες εντός του χώρου. Εδώ έπαιξε σημαντικό ρόλο η λειτουργία sleep των κόμβων-αισθητήρων όταν δεν χρειάζεται να παίρνουν μετρήσεις, με αποτέλεσμα να μειώνεται δραστικά η κατανάλωση ισχύος όταν δεν είναι απαιτητή, μειώνοντας σημαντικά το ενεργειακό ίχνος της υποδομής.

Τέλος, με τη χρήση τεχνολογιών LoRaWAN επιτυγχάνεται η ικανοποίηση ίσως της σημαντικότερης ιδιοτροπίας του σεναρίου αυτού, που είναι η αποφυγή πολύπλοκων και χρονοβόρων προσεγγίσεων, οι οποίες υπερκαλύπτουν τις ανάγκες της επιχείρησης. Αυτό είναι άλλωστε και το κύριο έμπρακτο προτέρημα της τεχνολογίας IoT.

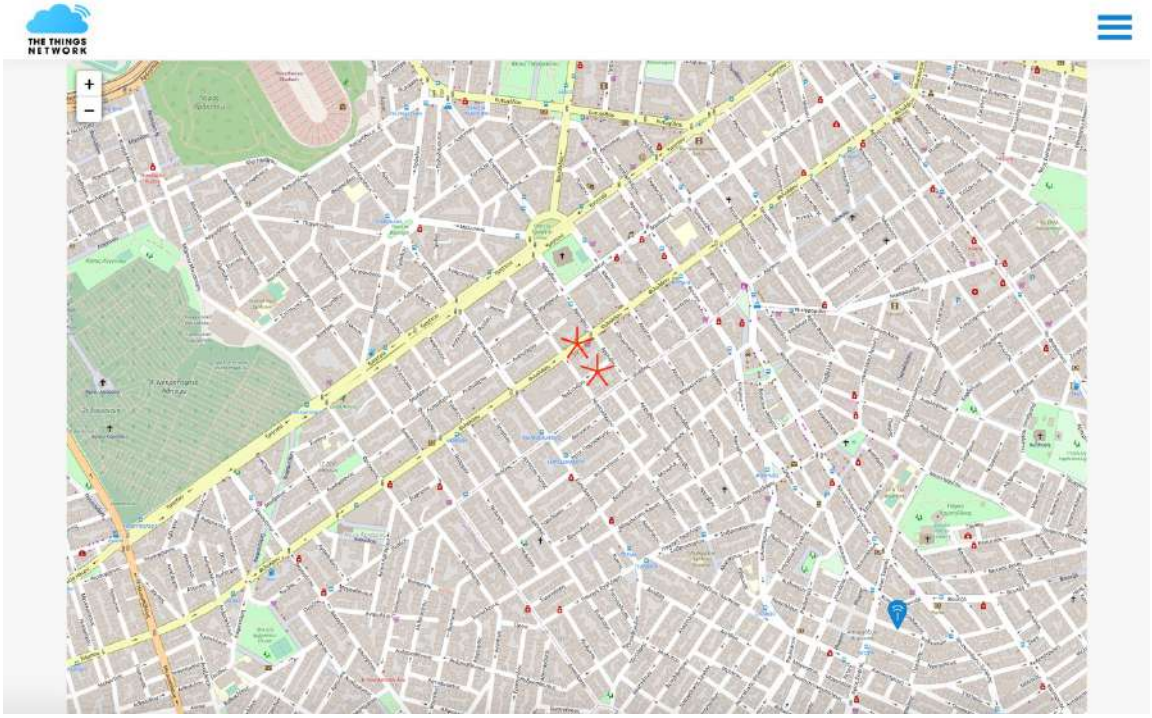
4.2 Παρουσίαση Εξοπλισμού και Προγραμμάτων

Στις παρακάτω υπό-παραγράφους, περιγράφεται αναλυτικά η υποδομή όσον αφορά τον υλικοτεχνολογικό εξοπλισμό (Hardware) και τα προγράμματα (Software) που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας με σκοπό την πρόταση χρήσης τους, στο παραπάνω σενάριο.

4.2.1 Gateway

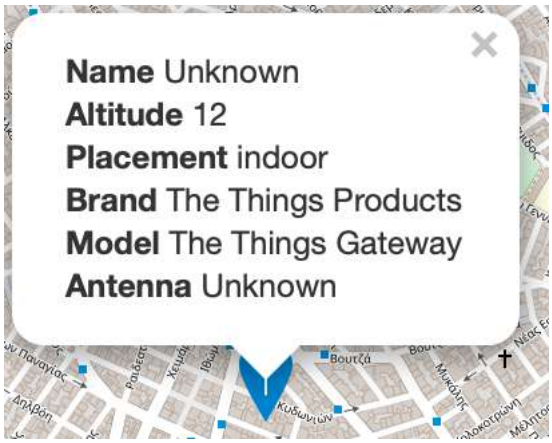
Η λειτουργία της πύλης ή αλλιώς, σταθμού βάσης ενός δικτύου LoRa είναι σαφής και συγκεκριμένη, ενώ κάθε σταθμός βάσης καλύπτει μια καθορισμένη περιοχή του δικτύου. Ένα gateway, είναι πρακτικά υπεύθυνο για τη λήψη και την προώθηση δεδομένων από τους τερματικούς κόμβους (αισθητήρες) στον δίκτυο LoRaWAN, και το αντίστροφο.

Χρησιμοποιώντας κανείς την ιστοσελίδα https://www.thethingsnetwork.org/map έχει τη δυνατότητα να αναζητήσει κοντινούς σε εκείνον σταθμούς βάσεις που έχουν ήδη σταθεί και παραμετροποιηθεί ώστε να τους κάνει χρήση, χωρίς κόστος. Σε περίπτωση όμως που βρίσκεται εκτός της περιοχής κάλυψης θα πρέπει να εγκαταστήσει δική του πύλη, όπως στη δική μας περίπτωση:



Εικόνα Ε12: Χάρτης κάλυψης LoRaWAN στη περιοχή των κτιρίων (κόκκινες ενδείξεις), με την κοντινότερη πύλη να εικονίζεται με τη μπλε αντίστοιχη ένδειξη.

Αναλυτικότερα στη δική μας περίπτωση, ένας σταθμός βάσης σαν αυτόν που εικονίζεται θα μπορούσε να καλύπτει την περιοχή για την οποία έχουμε ενδιαφέρον, αλλά φαίνεται πως η συγκεκριμένη είναι εσωτερικού χώρου και ο δημιουργός της δεν έχει επιλέξει να δημοσιοποιήσει πολλές πληροφορίες για αυτή.



Με αυτά τα δεδομένα, θεωρήθηκε βέλτιστο να κατασκευάσουμε και να χρησιμοποιήσουμε δική μας πύλη, μονού καναλιού, περιορίζοντας κόστη και έχοντας πλήρη έλεγχο της λειτουργίας της.

Hardware Εξοπλισμός για το Gateway:

- Pycom LoPy 4
- Pycom Expansion Board 3.0
- F :

Εικόνα Ε13: Pycom LoPy 4

3.0

Εικόνα Ε15: Pycom LoRa/Sigfox Antenna Kit

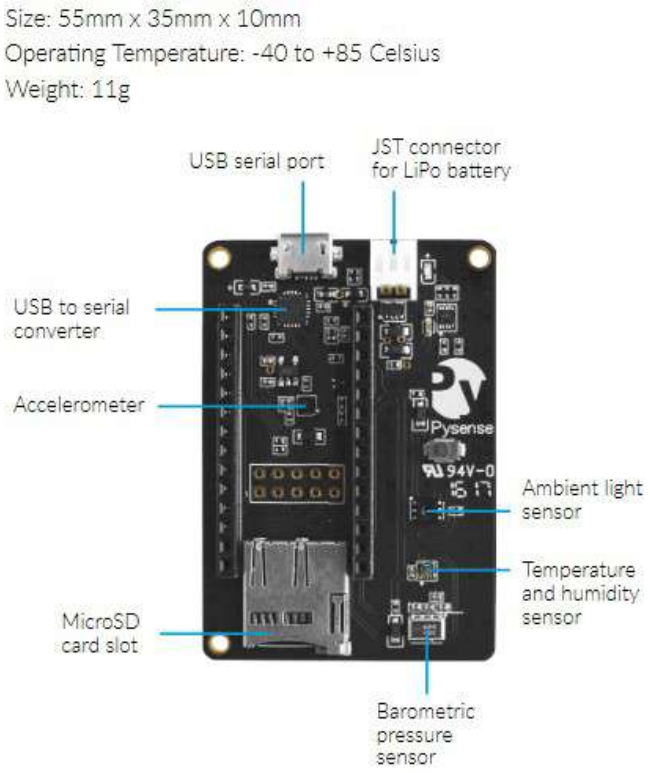
Λόγω του ότι δεν χρησιμοποιήθηκε καθετοποιημένη συσκευή LoRaWAN Gateway (το μέσο κόστος αυτής κυμαίνεται περίπου στα 1.000 Ευρώ) χρειάστηκε να γίνουν κάποιες αλλαγές στο κώδικα (Micropython) του LoPy 4 προκειμένου της επιτυχούς σύνδεσής του με το The Things Network (TTN), το οποίο στη περίπτωση μας αναπαριστά τον network server, τον join server αλλά και τον application server. Συγκεκριμένα, η πύλη που θα ρυθμίσουμε αναφέρεται ως nano gateway λόγω του ότι ενεργοποιείται σε αυτή μόνο ένα κανάλι επικοινωνίας και όλα τα υπόλοιπα παραμένουν ανενεργά. Ο κώδικας Nano-Gateway χωρίζεται σε 3 αρχεία, **main.py**, **config.py** και **nanogateway.py**. Αυτά χρησιμοποιούνται για τη διαμόρφωση και τον καθορισμό του τρόπου σύνδεσης της πύλης σας στο δίκτυο και πώς μπορεί να λειτουργήσει κατά τη προώθηση πακέτων σε και από αυτό. Αναλυτικότερα, το αρχείο **main.py** καλείται κατά την εκκίνηση της συσκευής και καλεί τα υπόλοιπα δύο αρχεία ώστε να γίνουν το αρχικό setup του nano gateway, μόλις τελειώσει η διαδικασία διαμόρφωση, ξεκινά η λειτουργία της πύλης. Το αρχείο **config.py** περιέχει τις συγκεκριμένες ρυθμίσεις για το διακομιστή και το δίκτυο στο οποίο θέλουμε να συνδεθούμε, ανάλογα με τη περιοχή λειτουργίας, αυτές οι ρυθμίσεις διαφέρουν. Τέλος το αρχείο **nanogateway.py** περιέχει τη βιβλιοθήκη, η οποία ελέγχει όλη τη δημιουργία πακέτων και τη προώθηση των δεδομένων LoRa.

Εικόνα Ε16: Τελική εικόνα του NanoGateway μετά την σύνδεση των επιμέρους υλικών

4.2.2 End Device - Sensors

Στην συγκεκριμένη εργασία, όσον αφορά τους αισθητήρες – τερματικές συσκευές του δικτύου LoRa, χρησιμοποιήθηκε η πλακέτα PySense για την συλλογή δεδομένων θερμοκρασίας, υγρασίας και λοιπών άλλων περιβαλλοντικών συνθηκών. Αναλυτικότερα, η συγκεκριμένη πλακέτα φέρει αισθητήρα φωτός περιβάλλοντος, αισθητήρα βαρομετρικής πίεσης, παρέχει μέτρηση υγρασίας, θερμοκρασίας ενώ επίσης περιέχει επιταχυνσιόμετρο 12 bit. Για πρόσβαση στη πλακέτα χρησιμοποιείται η θύρα USB στην άκρη της, στην οποία συνδεόμαστε σειριακά. Τέλος, έχει τη δυνατότητα προσθήκης MicroSD κάρτας για την αποθήκευση δεδομένων.

Επιλέχθηκε η συγκεκριμένη πλακέτα λόγω της πληθώρας δυνατοτήτων καθώς επίσης και για την εξαιρετικά χαμηλή κατανάλωση ενέργειας που επιτρέπει (1uA σε λειτουργία deep sleep) [28].



Εικόνα Ε17: Κάτοψη πλακέτας PySense

Για την αποστολή δεδομένων από τους αισθητήρες προς την πύλη gateway με διασύνδεση LoRa, χρησιμοποιήθηκε το module FiPy της Pycom και μια κεραία LoRa/Sigfox σαν αυτή που αναφέρεται παραπάνω.

Αναλυτικότερα το FiPy φέρει τον διπύρηνο επεξεργαστή Espressif ESP32, ο οποίος έχει αφεθεί ολοκληρωτικά ελεύθερος ώστε να τρέξει την εφαρμογή του εκάστοτε χρήστη. Επιπλέον φέρει έναν επιπλέον συνεπεξεργαστή ULP που μπορεί να παρακολουθεί GPIOs, τα κανάλια ADC και να ελέγχει τα περισσότερα εσωτερικά περιφερειακά κατά την λειτουργία deep sleep, καταναλώνοντας μόνο 25uA. Όσον αφορά τις εισόδους/εξόδους, περιέχονται 2x UART και 2x SPI για σειριακή επικοινωνία με άμεση πρόσβαση μνήμης (DMA) για όλα τα περιφερειακά, καθώς επίσης και μέχρι 22 GPIO ακροδέκτες, των οποίων η συμπεριφορά ρυθμίζεται μέσω του όποιου λογισμικού. Σχετικά με την ασφάλεια, είναι συμβατό με αλγόριθμους κρυπτογράφησης SHA, MD5, DES και AES. Όσον αφορά τη μνήμη, φέρει RAM 4MB, μία μνήμη flash των 8MB, και υποστηρίζει Python multithreading. Η τάση εισόδου κυμαίνεται από 3.3V μέχρι 5.5V, ικανή να παράγει 400mA. Τέλος, όσον αφορά την επικοινωνία LoRa, λειτουργεί στις συχνότητες 868MHz (Ευρώπη) και στα 915MHz (Αμερική, Αυστραλία και Νέα Ζηλανδία) κάνοντας εφικτή την επικοινωνία ενός κόμβου δικτύου (για το οποίο το κάνουμε χρήση) με την πύλη-gateway, σε απόσταση έως και 40 χιλιομέτρων [29].

Hardware Εξοπλισμός για το End-Device:

- Pycom FiPy (Model 1.0)
- Pycom PySense V1.0
- Pycom LoRa/Sigfox Antenna Kit

Εικόνα Ε18: Pycom FiPy (Model 1.0)

Εικόνα Ε19: Pycom LoRa/Sigfox Antenna Kit

Εικόνα Ε20: Τελική εικόνα του End Device μετά την σύνδεση των επιμέρους υλικών

4.2.3 Χρήση του The Things Network (TTN)

Στη συγκεκριμένη υλοποίηση, οι πύλες αποτελούν τη γέφυρα μεταξύ συσκευών και του The Things Network. Οι συσκευές δηλαδή χρησιμοποιούν δίκτυα χαμηλής ισχύος όπως το LoRaWAN για σύνδεση στο Gateway, ενώ το Gateway χρησιμοποιεί δίκτυα υψηλού εύρους ζώνης όπως WiFi, Ethernet ή Cellular για σύνδεση στο The Things Network. Όλες οι πύλες που είναι προσβάσιμες από μια συσκευή θα λαμβάνουν τα μηνύματα της συσκευής και θα τα προωθούν στο The Things Network. Το δίκτυο θα αντιγράφει τα μηνύματα και θα επιλέγει την καλύτερη πύλη για προώθηση όλων των μηνυμάτων που βρίσκονται στην ουρά για κατερχόμενη σύνδεση. Σχετικά με τη σύνδεση μεταξύ της πύλης και του TTN, ρυθμίζεται μέσω packet forwarder bridges και το πρωτόκολλο επικοινωνίας ονομάζεται Semtech UDP Protocol. Οι προωθητές πακέτων συνδέονται στο The Things Network μέσω του στοιχείου του δρομολογητή (router – τον οποίο επιλέγουμε κατά τη διάρκεια του registration της πύλης στο TTN, βάσει της περιοχής). Αυτό το στοιχείο είναι υπεύθυνο για το χειρισμό δεδομένων πύλης και κατάστασης και για τη μεταφορά δεδομένων στο υπόλοιπο δίκτυο. Για ανερχόμενες συνδέσεις (uplinks), ο δρομολογητής λαμβάνει συνδέσμους από gateways,

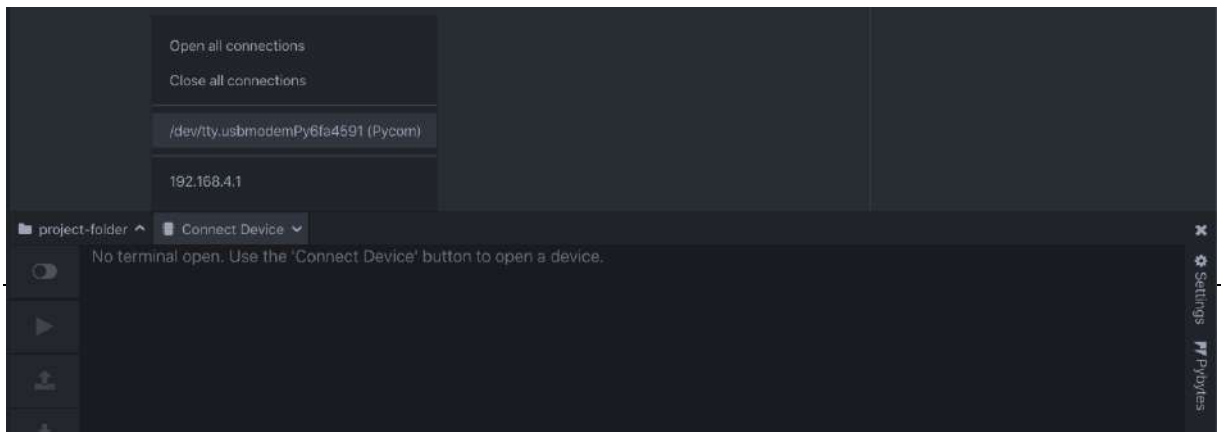
αναλύει το περιεχόμενο ανερχόμενης ζεύξης για ωφέλιμο φορτίο και ενεργοποιήσεις MAC, υπολογίζει νέα παράθυρα κατερχόμενης ζεύξης, βρίσκει έναν μεσίτη για να μεταδώσει το πακέτο και να μεταδώσει τον ανερχόμενο σύνδεσμο. Για μηνύματα κατάστασης, ο δρομολογητής τα χρησιμοποιεί για να παρακολουθεί τα ενεργά gateways. Για κατερχόμενες συνδέσεις (downlinks), ο δρομολογητής λαμβάνει μια κατερχόμενη ζεύξη, περιμένει να υπάρχει διαθέσιμη πύλη κοντά στη συσκευή, δημιουργεί επιλογές κατερχόμενης ζεύξης από τις επιλογές πύλης και τη μεταδίδει.

Σχετικά με το Semtech UDP πρωτόκολλο που χρησιμοποιείται, είναι ιστορικά, το πρώτο πρωτόκολλο πύλης που αναπτύχθηκε για LoRaWAN. Χτίστηκε από τον Semtech, η οποία το διατηρεί ακόμα. Με αυτό το πρωτόκολλο, τα uplinks, οι καταστάσεις και τα downlinks ανταλλάσσονται σε ψευδό-JSON μορφή, μέσω UDP, μεταξύ της πύλης και του διακομιστή δικτύου. Λόγω της απλότητας των μηνυμάτων και του πρωτοκόλλου, είναι εύκολο να αναπαραχθεί αυτό το πρωτόκολλο, για δοκιμές ή για bootstrap [30].

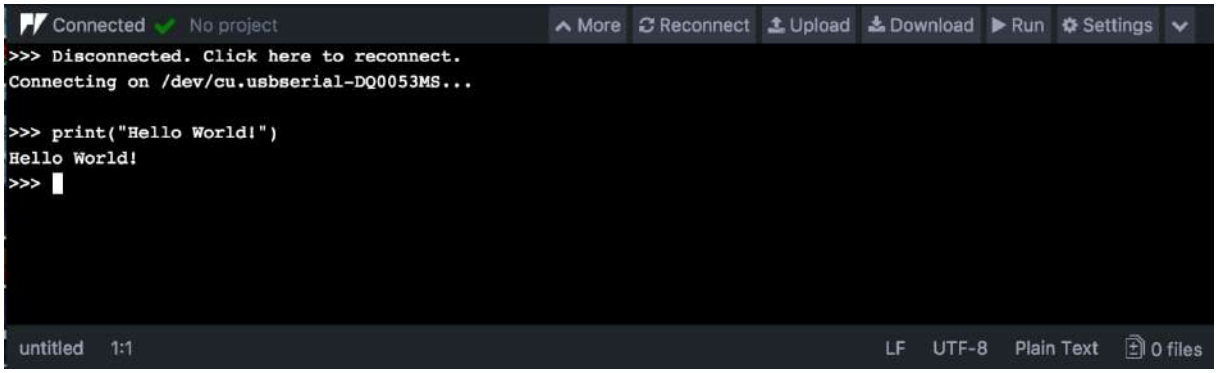
Όσον αφορά την πιστοποίηση ταυτότητας του gateway για σωστή επικοινωνία, γίνεται σε 2 μέρη, τόσο στο κώδικα της συσκευής όσο και κατά την διαδικασία registration του gateway στο TTN.

4.2.4 Χρήση Προγραμματιστικής Γλώσσας και IDE για την ρύθμιση των συσκευών

Για τον προγραμματισμό των συσκευών, επιλέχθηκε ο Atom editor και το plugin Pymakr. Ο editor κατέβηκε και εγκαταστάθηκε από την επίσημη σελίδα https://atom.io/ και το συγκεκριμένο plugin μπορεί να προστεθεί μέσα από την εφαρμογή, ως installable package. Ως εναλλακτική θα μπορούσε να χρησιμοποιηθεί το πιο ευρέως γνωστό Visual Studio Code, η οποία φέρει το ίδιο extension ως Pymakr VSCode Extension. Η επιλογή των IDE αφορά τη συμβατότητα με το συγκεκριμένο plugin, το οποίο επιτρέπει απευθείας σύνδεση στη συσκευή όταν συνδέεται μέσω usb σε οποιοδήποτε τερματικό.



Εικόνα Ε21: Επιλογή συνδεδεμένων συσκευών από τον Atom IDE

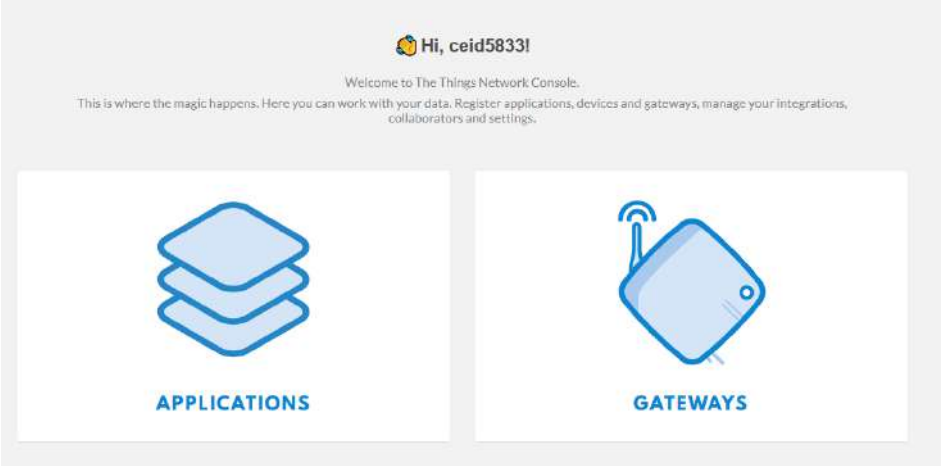


Εικόνα Ε22: Επιτυχής σύνδεση συσκευής στον Atom IDE

Η γλώσσα προγραμματισμού που χρησιμοποιείται για την παραμετροποίηση των συσκευών είναι η MicroPython, η οποία είναι μια εφαρμογή λογισμικού που είναι σε μεγάλο βαθμό συμβατή με το Python 3, γραμμένη σε C και είναι βελτιστοποιημένη για εκτέλεση σε έναν μικροελεγκτή. Αποτελεί έναν πλήρη μεταγλωττιστή και runtime που εκτελείται στο υλικό του μικροελεγκτή. Επίσης παρουσιάζεται στο χρήση με διαδραστική προτροπή (REPL) για άμεση εκτέλεση υποστηριζόμενων εντολών. Περιλαμβάνει μια συλλογή βασικών βιβλιοθηκών Python, καθώς και λειτουργικές μονάδες που δίνουν στον προγραμματιστή πρόσβαση σε υλικό χαμηλού επιπέδου. Δημιουργήθηκε αρχικά από τον Αυστραλό προγραμματιστή και φυσικό Damien George, μετά από μια επιτυχημένη εκστρατεία που υποστηρίχθηκε από το Kickstarter το 2013. Ενώ η αρχική εκστρατεία Kickstarter κυκλοφόρησε το MicroPython με έναν πίνακα ανάπτυξης με υποστήριξη STM32F4 "pyboard", το MicroPython υποστηρίζει μια σειρά αρχιτεκτονικών βασισμένων σε ARM. Οι θύρες που υποστηρίζονται στην κύρια γραμμή είναι ARM Cortex-M (πολλές πλακέτες STM32, TI CC3200 / WiPy, Teensy board, Nordic nRF series, SAMD21 and SAMD51), ESP8266, **ESP32**, 16bit PIC, Unix, Windows, Zephyr και JavaScript [31].

4.2.5 Εγγραφή Gateway και end nodes στη πλατφόρμα TTN

Ως πρώτη ενέργεια μετά την αναβάθμιση των firmware όλων των συσκευών στη τελευταία έκδοση είναι η εγγραφή αυτών στην πλατφόρμα TTN. Πλοηγούμαστε στην επίσημη σελίδα https://console.thethingsnetwork.org/ και εφόσον φτιάξουμε λογαριασμό, μας εμφανίζεται μία σελίδα όπως η παρακάτω:



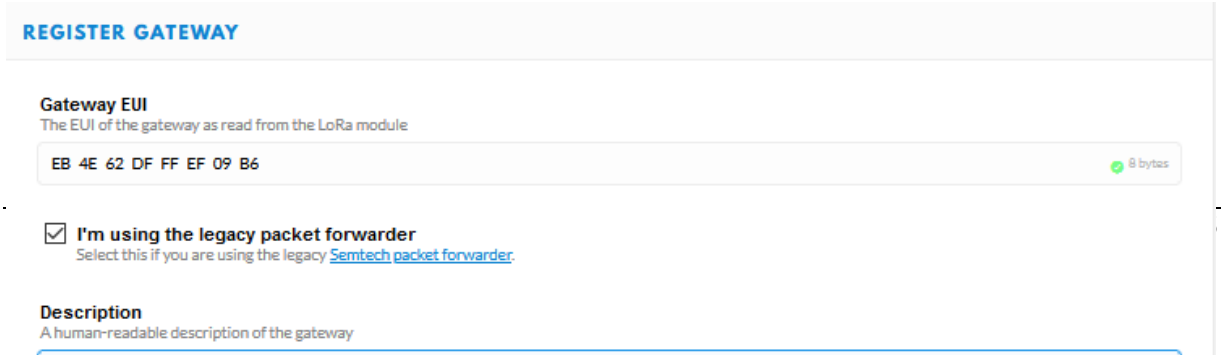
Εικόνα E23: Web σελίδα δήλωσης συσκευών στο TTN

Εκεί επιλέγουμε, καταρχήν, το παράθυρο gateways και ύστερα την επιλογή register gateway. Η φόρμα που εμφανίζεται στη συνέχεια περιέχει τα στοιχεία που πρέπει να συμπληρωθούν για την αναγνώριση της συσκευής από το TTN. Το **"Gateway EUI"** είναι ένα μοναδικό για κάθε συσκευή αναγνωριστικό, το οποίο αναφέρεται και παραπάνω, που αποτελεί με άλλα λόγια τη διεύθυνση MAC της συσκευής. Για να εξαχθεί αυτή η πληροφορία με τρόπο ώστε να τη δέχεται το TTN, πρέπει να γραφεί συγκεκριμένο κομμάτι κώδικα, χρησιμοποιώντας την βιβλιοθήκη **ubinascii**.

```
import machine
import ubinascii

WIFI_MAC = ubinascii.hexlify(machine.unique_id()).upper()
# Set  the Gateway ID to be the first 3 bytes of MAC address + 'FFFE' +
last 3 bytes of MAC address
```

```
GATEWAY_ID = WIFI_MAC[:6] + "FFFE" + WIFI_MAC[6:12]
```

Ακολουθώντας με ένα print statement, λαμβάνουμε το μοναδικό ID της συσκευής που δημιουργήθηκε χρησιμοποιώντας την διεύθυνση MAC, η οποία είναι ενσωματωμένη στο προσαρμογέα δικτύου Wi-Fi σε όλες τις Pycom συσκευές. Η έξοδος της εντολής print μας αποφέρει ένα αποτέλεσμα σαν **b3e41dfffef08b5d** το οποίο, εφόσον προσθέσουμε το πρόθεμα "eui-" στο εμπρός μέρος, συμπληρώνουμε στην προαναφερόμενη φόρμα. Το επόμενο που πρέπει να συμπληρώσουμε είναι το "**Description**" της συσκευής που είναι πρακτικά ένα friendly name το οποίο μπορεί να πάρει όποια τιμή επιλέξουμε. Ύστερα, σειρά έχει το **"Frequency Plan"** το οποίο επιλέγεται σύμφωνα με την περιοχή στην οποία βρίσκεται το εκάστοτε gateway. Στη δική μας περίπτωση επιλέγεται το **Europe – 868MHz.** Τέλος, μένει η επιλογή του **"Router"** στο TTN, μέσω του οποίου θα γίνεται η προώθηση των πακέτων, ο οποίος επιλέγεται και πάλι βάσει περιοχής, οπότε θα χρησιμοποιηθεί ο **ttn-router-eu**. Προαιρετικά, και σε περίπτωση που η συσκευή δεν περιέχει GPS δέκτη, μας δίνεται η επιλογή να κάνουμε pin την τοποθεσία της σε χάρτη.



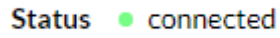
Εικόνα Ε24: Συμπληρωμένη φόρμα register gateway στο TTN

Ως δεύτερη ενέργεια, θα πρέπει να εισάγουμε τις πληροφορίες αυτές, μαζί με τις απαραίτητες για την σύνδεση της πύλης στο διαδίκτυο μέσω Wi-Fi, στη συσκευή. Στο αρχείο **config.py** που αναφέρεται και παραπάνω, εισάγουμε τις πληροφορίες της εικόνας στις κατάλληλες μεταβλητές, καθώς επίσης και τα στοιχεία σύνδεσης σε ένα εντός εμβέλειας, Wi-Fi δίκτυο.

```
import machine
import ubinascii

WIFI_MAC = ubinascii.hexlify(machine.unique_id()).upper()
# Set  the Gateway ID to be the first 3 bytes of MAC address + 'FFFE' +
last 3 bytes of MAC address
```

```
GATEWAY_ID = WIFI_MAC[:6] + "FFFE" + WIFI_MAC[6:12]

SERVER = 'router.eu.thethings.network'
PORT = 1700

NTP = "0.gr.pool.ntp.org"
NTP_PERIOD_S = 3600

WIFI_SSID = 'Obi-Wlan-Kenobi'
WIFI_PASS = 'insert_WiFi_password_here'

# for EU868
LORA_FREQUENCY = 868100000
LORA_GW_DR = "SF7BW125" # DR_5
LORA_NODE_DR = 5

# for US915
# LORA_FREQUENCY = 903900000
# LORA_GW_DR = "SF10BW125" # DR_0
# LORA_NODE_DR = 0
```

Μετά την αποθήκευση και το upload των ρυθμίσεων στο LoPy gateway, συνδεόμαστε στο TTN. Για την επιβεβαίωση της επιτυχούς σύνδεσης, εάν επιστρέψουμε στην καρτέλα θα πρέπει πλέον το gateway να φαίνεται “connected” με το The Things Network.



Εικόνα Ε25: Κατάσταση επιτυχούς σύνδεσης του gateway στο TTN

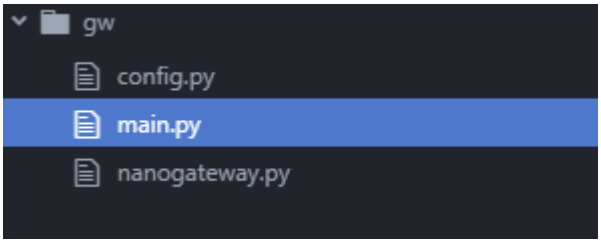
5

Πειράματα

5.1 Τι χρησιμοποιήθηκε στο κώδικα

Το σύνολο του κώδικα των συσκευών χωρίζεται σε φακέλους οι οποίοι αλληλεπιδρούν μεταξύ τους για να δοθεί το τελικό αποτέλεσμα. Η αρχική συλλογή βιβλιοθηκών πάρθηκε από το Github repository της Pycom https://github.com/pycom/pycom-libraries/archive/master.zip στο οποίο έγιναν όλες οι απαραίτητες τροποποιήσεις σύμφωνα με τα ζητούμενα του σεναρίου που πραγματεύεται αυτή η εργασία.

Όσον αφορά το gateway, το σύνολο του κώδικα που χρησιμοποιείται βρίσκεται σε 3 αρχεία, **main.py**, **config.py** και **nanogateway.py.** Στην **main.py** δηλώνονται τα βασικά στοιχεία που χρειάζεται η πύλη για τη σύνδεση με το TTN, για να δέχεται πακέτα από συσκευές και να τα προωθεί καθώς επίσης και για τη σύνδεσή της με internet μέσω Wi-Fi. Τα στοιχεία αυτά δηλώνονται στην συνάρτηση **“nanogw”** και λαμβάνονται από τα υπόλοιπα δύο αρχεία που την συνοδεύουν.



Εικόνα Ε26: Εκτελέσιμα αρχεία κώδικα gateway

Επίσης στο αρχείο αυτό βρίσκεται και η εντολή **“nanogw.start()”** η οποία σηματοδοτεί την αρχική λειτουργίας της συσκευής ως πύλη LoRa. Στο **config.py** είναι εκεί που λαμβάνουν χώρα οι περισσότερες αλλαγές καθώς περιέχει όλα τα απαραίτητα στοιχεία σύνδεσης στο internet ανάλογα με το περιβάλλον στο οποίο εισάγεται το gateway, όπως τα στοιχεία σύνδεσης Wi-Fi, ο server του TTN με τον οποίο θα επικοινωνεί, έναν NTP server για τον συγχρονισμό των δεδομένων καθώς επίσης και την συχνότητα λειτουργίας LoRa. Το **nanogateway.py** αναλαμβάνει όλη τη λειτουργία της συσκευής ως LoRa gateway, ακολουθώντας συγκεκριμένη σειρά ενεργειών. Αρχικά, προσπαθεί να συνδεθεί στο Wi-Fi με τα στοιχεία που του έχουν εισαχθεί και να λάβει συγχρονισμό από τον ntp server. Στην συνέχεια, εφόσον επιτύχει συνδέεται με τον router του TTN και δημιουργεί ένα UDP Socket. Αφού το δημιουργήσει, η πρώτη του ενέργεια είναι να αποστείλει απευθείας πακέτο για τον έλεγχο της ορθής επικοινωνίας στον router και ύστερα να δημιουργήσει τα κατάλληλα alarms που περιέχουν τις περιόδους αποστολής και λήψης πακέτων από και προς αυτόν. Επιπλέον, ξεκινά ο σχετικός UDP Listener ώστε να μπορεί να λαμβάνει downlink πακέτα, ενώ ακριβώς μετά αρχικοποιεί την επικοινωνία LoRa με την συχνότητα, τον ρυθμό μετάδοσης δεδομένων και τα λοιπά στοιχεία που του έχουν οριστεί.

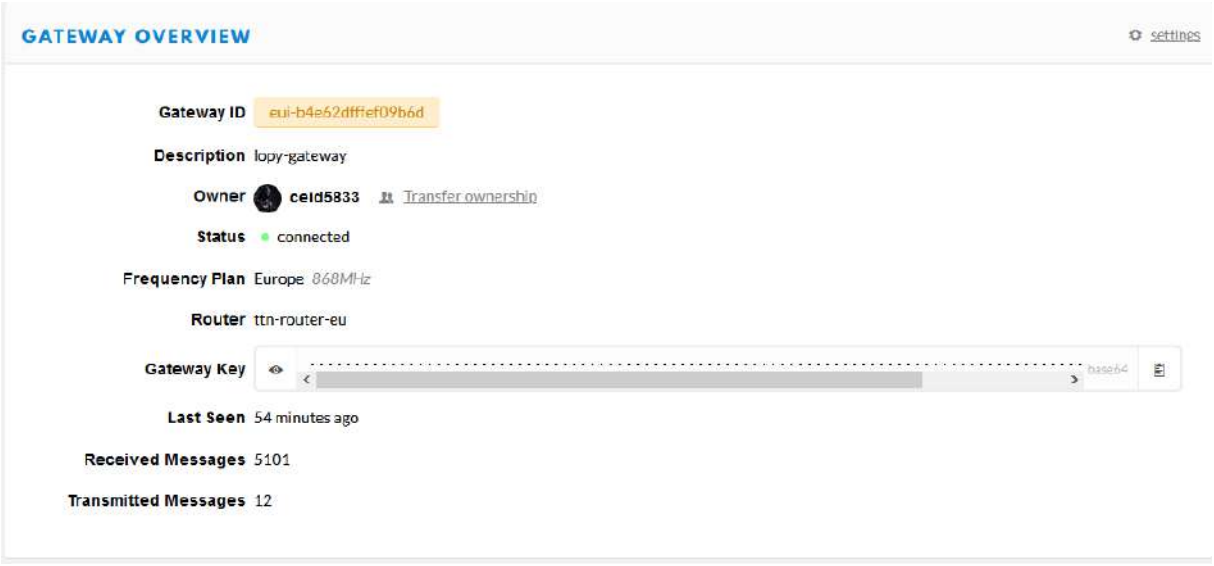
```
self._log('Starting LoRaWAN nano gateway with id: {}', self.id)

# setup WiFi as a station and connect
self.wlan = WLAN(mode=WLAN.STA)
self._connect_to_wifi()

# get a time sync
self._log('Syncing time with {} ...', self.ntp_server)
self.rtc.ntp_sync(self.ntp_server, update_period=self.ntp_period)
while not self.rtc.synced():
    utime.sleep_ms(50)
self._log("RTC NTP sync complete")

# get the server IP and create an UDP socket
self.server_ip = usocket.getaddrinfo(self.server, self.port)[0][-1]
self._log('Opening UDP socket to {} ({}) port {}...', self.server, self.server_ip[0], self.server_ip[1])
self.sock = usocket.socket(usocket.AF_INET, usocket.SOCK_DGRAM, usocket.IPPROTO_UDP)
self.sock.setsockopt(usocket.SOL_SOCKET, usocket.SO_REUSEADDR, 1)
self.sock.setblocking(False)
self._log ('Opened UDP socket!')

# push the first time immediately
self._push_data(self._make_stat_packet())

# create the alarms
self.stat_alarm = Timer.Alarm(handler=lambda t: self._push_data(self._make_stat_packet()), s=60, periodic=True)
self.pull_alarm = Timer.Alarm(handler=lambda u: self._pull_data(), s=25, periodic=True)

# start the UDP receive thread
self.udp_stop = False
_thread.start_new_thread(self._udp_thread, ())

# initialize the LoRa radio in LORA mode
self._log('Setting up the LoRa radio at {} Mhz using {}', self._freq_to_float(self.frequency), self.datarate)
self.lora = LoRa(

    mode=LoRa.LORA, #raw lora is lotta
    frequency=self.frequency,
    bandwidth=self.bw,
    sf=self.sf,
    preamble=8,
    coding_rate=LoRa.CODING_4_5,
    region=LoRa.EU868,
    tx_iq=True
)

# create a raw LoRa socket
self.lora_sock = usocket.socket(usocket.AF_LORA, usocket.SOCK_RAW)
self.lora_sock.setblocking(False)
self.lora_tx_done = False

self.lora.callback(trigger=(LoRa.RX_PACKET_EVENT | LoRa.TX_PACKET_EVENT), handler=self._lora_cb)
self._log("LoRaWAN nano gateway online")
```

Εικόνα Ε27: Στιγμιότυπο Κώδικα nanogateway.py

Σε περίπτωση που δεν λαμβάνονται δεδομένα από κάποιο node, το gateway εμφανίζει στη κονσόλα output το οποίο εναλλάσσεται συνεχώς μεταξύ push και pull ack από τον router, ενώ σε περίπτωση λήψης πακέτου, υπάρχει σχετικό print out που μπορεί να παρατηρήσει ο χρήστης στη κονσόλα της συσκευής.

Εικόνα E28: Έξοδος nanogateway χωρίς (πάνω) και μετά από (κάτω) λήψη LoRa πακέτου

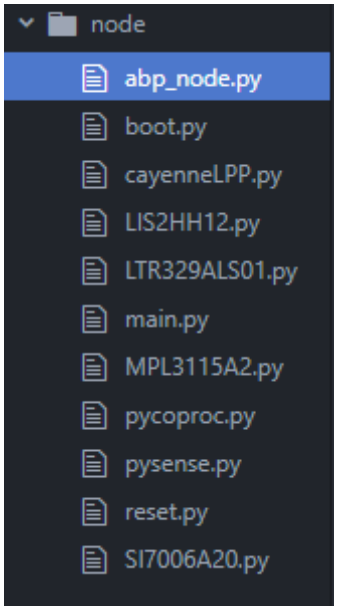
Για να επιβεβαιωθεί η σωστή λειτουργία της πύλης από τη πλευρά του TTN μετά την εισαγωγή στοιχείων, πηγαίνοντας πίσω στην κονσόλα και επιλέγοντας gateways, στην καρτέλα overview θα πρέπει να βλέπουμε την κατάσταση σύνδεσης της πύλης καθώς και τη τελευταία φορά επικοινώνησε επιτυχώς με τον router του TTN



Εικόνα E29: Gateway Overview συνδεδεμένης συσκευής στο TTN console

Όσον αφορά το end device – sensor στο FiPy που χρησιμοποιείται, τα αρχεία κώδικα που το αποτελούν είναι περισσότερα σε πλήθος (9) αλλά τα περισσότερα παρέχουν απλές λειτουργίες όπως reset της συσκευής, αρχικοποίηση των αισθητήρων και ανάγνωση δεδομένων από αυτούς. Κυρίως ενδιαφέρον παρουσιάζουν τα αρχεία **main.py** και **abp_node.py** στα οποία γίνεται και η παραμετροποίηση, αφού το πρώτο επιστρέφει τις

τιμές των αισθητήρων από τους οποίους θέλουμε να λάβουμε τιμές και το δεύτερο είναι υπεύθυνο για τον τρόπο επικοινωνίας με χρήση LoRa.



Εικόνα Ε30: Εκτελέσιμα αρχεία κώδικα node

Μετά το επιτυχές registration της συσκευής (node) στο TTN, τα στοιχεία **Device Address**, **Network Session Key** και **App Session Key** θα πρέπει να εισαχθούν αντίστοιχα στις μεταβλητές **dev_addr**, **nwk_swkey** και **app_swkey** εντός του αρχείου **abp_node.py** αφού παράγονται αυτόματα από την κονσόλα του TTN προκειμένου της ταυτοποίησης της συσκευής και της ασφαλούς επικοινωνίας της με το δίκτυο, μέσω του gateway. Για την σωστή εισαγωγή των χαρακτήρων, χρειάζεται η συνάρτηση python **binascii.unhexlify()** η οποία επιστρέφει τα δυαδικά δεδομένα που αντιπροσωπεύονται από την δεκαεξαδική συμβολοσειρά.

```
# create an ABP authentication params
dev_addr = struct.unpack(">l", binascii.unhexlify('26011F51'))[0]
nwk_swkey = binascii.unhexlify('D174FAC44686447985139240C02C31739')
app_swkey = binascii.unhexlify('3DF8D0D8782046D5D219B07F258C5711')
```

Εικόνα Ε31: Τιμές μεταβλητών dev_addr, nwk_swkey και app_swkey

Επόμενη κίνηση είναι η απενεργοποίηση των μη προεπιλεγμένων καναλιών LoRa και η ρύθμιση των 3 πρώτων καναλιών (0,1,2) στην ίδια συχνότητα. Στην συνέχεια η συσκευή επιχειρεί προσχώρηση στο δίκτυο LoRa της συχνότητας αυτής με την μέθοδο ABP που

κάνουμε χρήση, δημιουργώντας ένα LoRa socket, θέτοντας τον ρυθμό μετάδοσης δεδομένων και ρυθμίζοντας το socket ώστε να είναι non-blocking ώστε να μπορεί ταυτόχρονα να δέχεται αιτήματα υποδοχής δεδομένων ενώ αποστέλλει. Τέλος, διαβάζει τα δεδομένα από τους αισθητήρες, τα μετατρέπει σε strings βάζοντας το καθένα σε διαφορετικό πακέτο πληροφοριών, τα οποία στέλνει χωρισμένα με delimiter τον χαρακτήρα "," ώστε να μπορούν να διαβαστούν, επιστρέφοντας παράλληλα και το κατάλληλο output στη κονσόλα.

```
#what is needed for measurements
mp = MPL3115A2(py,mode=ALTITUDE)
mpp = MPL3115A2(py,mode=PRESSURE)
si = SI7006A20(py)
t_ambient = 24.4
#sending the measurements
for i in range (200):
    temp = str(mp.temperature())
    hum = str(si.humidity())
    pkt1 = temp
    pkt2 = hum
    #stelnoume ta 2 paketa xwrismena me komma gia delimiter wste na mporoun na diavastoun
    pkt = pkt1 + ',' + pkt2
    #print status 1
    print('Sending Temp')
    #print Temperature Values
    print ('Temp: ',temp, 'Celsius')
    #print status 2
    print('Sending Hum')
    #print HUmidity Values
    print ('Hum:',hum,'%')
    #to feel better while reading from console
    print ('**********')
    #steile to paketo meso LoRa sto TTN
    s.send(pkt)
```

Εικόνα Ε32: Στιγμιότυπο κώδικα abp_node.py

```
Sending Temp
Temp:  33.5625 Celsius
Sending Hum
Hum: 38.28482 %
**********
```

Εικόνα Ε33: Στιγμιότυπο εξόδου κονσόλας συσκευής για τη μέτρηση θερμοκρασίας και υγρασίας

Αντίστοιχα, η επικοινωνία αυτή κατοπτρίζεται και στο TTN, καθώς πλοηγόμενοι στη σελίδα “Devices” εντός του Application, στη καρτέλα “Data” φαίνεται το μήνυμα που απεστάλη στο δίκτυο από το node, σε δεκαεξαδική μορφή.

Εικόνα Ε33: Στιγμιότυπο επιτυχής αποστολής δεδομένων στο TTN από τη συσκευή node

Για να πετύχουμε μία μορφή δεδομένων που είναι εύκολα και γρήγορη αναγνωρίσιμη από τον χρήση που παρακολουθεί τις τιμές αυτές, θα πρέπει να προχωρήσουμε σε επεξεργασία των δεδομένων αυτών ώστε να εμφανίζονται σε πιο φιλική μορφή. Για την περίπτωση, το TTN περιέχει, κάτω από τη σελίδα applications, την καρτέλα Payload Formats, στην οποία μπορούμε να αλλάξουμε τον τρόπο εμφάνισης των δεδομένων στην σελίδα TTN, χρησιμοποιώντας τη γλώσσα προγραμματισμού Javascript.

```
function Decoder(bytes, port) {
  // Decode plain text; for testing only
    var string = String.fromCharCode.apply(null, bytes)
    var values = string.split(",");

     return {
      temp: parseFloat(values[0]),
      hum: parseFloat(values[1])
  };
}
```

5.2 Φυσική Σημασία Πειραμάτων

Όπως έχει προαναφερθεί αρκετές φορές στην εργασία αυτή, ο παράγοντας διάδοσης (Spreading Factor – SF) είναι πολύ σημαντικός στην επικοινωνία LoRa, καθώς αποτελεί κύριο παράγοντα σε όλες τις σημαντικές μετρήσεις αυτού του πρωτοκόλλου επικοινωνίας (κατανάλωση ενέργειας, μέγιστη απόσταση δυνατής επικοινωνίας, ποιότητα επικοινωνίας ανάλογα με το περιβάλλον, κ.α.)

Για το λόγο αυτό, στο σημείο αυτό θεωρήσαμε σημαντικό να παρουσιάσουμε τον τρόπο με τον οποίο μπορεί να γίνει όσο το δυνατόν πιο βέλτιστα η επιλογή του παράγοντα διάδοσης – έχοντας υπόψιν συγκεκριμένα δεδομένα όσον αφορά το περιβάλλον στο οποίο λαμβάνει χώρα το πείραμα.

Θα χρησιμοποιήσουμε για το σκοπό αυτό Machine Learning (ML) αλγορίθμους. Πιο συγκεκριμένα, το ML προσεγγίζει το στόχο του μέσω της εποπτευόμενης μάθησης (supervised learning), της μη εποπτευόμενης μάθησης (unsupervised learning) και της ενισχυτικής μάθησης (reinforcement learning). Συγκεκριμένα για την εποπτευόμενη μάθηση, καθώς αυτή επιλέγουμε σαν προσέγγιση, αποτελεί τη διαδικασία κατά την οποία η εκμάθηση της μηχανής πραγματοποιείται με τη χρήση δεδομένων για τα οποία γνωρίζουμε εξ’ ολοκλήρου τη κατηγορία – «κλάση» τους. Με άλλα λόγια, αναφερόμαστε σε «σεσημασμένα δεδομένα» (labeled data). Η διαδικασία εκμάθησης αυτή καθαυτή ονόμαζεται εκπαίδευση (training) και το υποσύνολο ή σετ δεδομένων που τροφοδοτούμε τη μηχανή ονομάζεται «training dataset». Προκειμένου να δοκιμάσουμε την απόδοση του αλγορίθμου μετά την εκπαίδευση, χρησιμοποιούμε ένα «άγνωστο» υποσύνολο δεδομένων, στο οποίο και αναφερόμαστε συνήθως ως «testing dataset».

Το πρόβλημα επιλογής του κατάλληλου Spreading Factor, αποτελεί πρόβλημα ταξινόμησης (classification problem) καθώς εμπεριέχεται διακριτές/πεπερασμένες σημάνσεις στα δεδομένα. Αναλυτικότερα, οι τιμές του SF κυμαίνονται από 7 έως 12 άρα έχουμε 6 κλάσεις/κατηγορίες/σετ στο σενάριό μας (multi-class classification problem).

Οι συγκεκριμένοι αλγόριθμοι με τους οποίους θα εργαστούμε είναι, ο k-NN, ο Naive Bayes και τα SVM (Support Vector Machines).

Ο αλγόριθμος k-NN κάνει την υπόθεση ότι υπάρχουν κοντινοί συγκρίσιμοι κόμβοι ή ότι κοντινοί παρόμοιοι κόμβοι μοιράζονται μια κοινή συμπεριφορά. Τα βασικά πλεονεκτήματά του είναι η απλότητα και η ευκολία χρήσης του, καθώς και η μέτρια ακρίβειά του. Μετά την εκμάθηση, τα νέα «άγνωστα» δεδομένα τροφοδοτούνται ειδικά ως σημείο D-διάστασης στον αλγόριθμο k-NN. Στη συνέχεια προσδιορίζεται η ελάχιστη απόσταση μεταξύ του σημείου εισόδου και των εκπαιδευτικών χώρων. Στον αλγόριθμο k-NN, η απόσταση είναι κρίσιμη. Η σωστή επιλογή της μετρικής απόστασης είναι επομένως απαραίτητη.

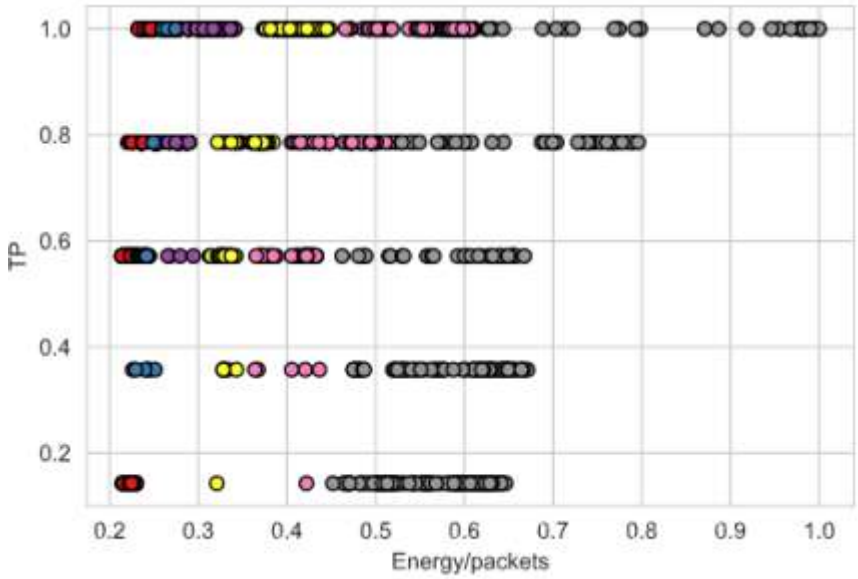
Ως εκ τούτου, για τον υπολογισμό της απόστασης, χρησιμοποιούμε την παρακάτω εξίσωση, εισάγοντας p=2 ώστε να λάβουμε την ευκλείδια απόσταση.

$$D = \left(\sum_{1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Τα SVM διαφέρουν από την οικογένεια των μη πιθανοτικών ταξινομητών γνωστών ως Naive Bayes. Η εύρεση ενός (υπερ)επιπέδου που διακρίνει τις τάξεις του σετ εκπαίδευσης με τον υψηλότερο βαθμό απόκλισης είναι ο κύριος στόχος των SVM. Η πρόβλεψη της «σήμανσης» πραγματοποιείται με βάση την περιοχή του (υπερ)επιπέδου που πέφτει όταν τα νέα μη εκπαιδευμένα δεδομένα παρέχονται στα SVM. Τα SVM πιστεύεται ότι είναι μεταξύ των καλύτερων αλγορίθμων ταξινόμησης, καθώς μπορούν να χειριστούν δυαδικά και πολυκλασικά προβλήματα. Από την αξιολόγηση απώλειας διαδρομής, πολλά δείγματα λαμβάνονται ως μέρος της διαδικασίας ταξινόμησης.

Ο Naive Bayes στη πραγματικότητα δεν είναι ταξινομητής αλλά μία κατηγορία πιθανολογικών ταξινομητών. Η κύρια ιδέα είναι ότι με δεδομένο ένα διάνυσμα εισόδου, το οποίο αντιπροσωπεύει το μη ορατό σημείο δεδομένων, εφαρμόζεται το ομώνυμο θεώρημα υποθέτοντας ανεξαρτησία μεταξύ των χαρακτηριστικών του δεδομένου διανύσματος εισόδου. Σχετικά με τις πιθανότητες εντός του θεωρήματος, μπορούμε να υποθέσουμε ότι ακολουθούν συγκεκριμένα Γκαουσιανή κατανομή (Gaussian distribution).

Έχοντας περιγράψει το πρόβλημα προς επίλυση και τους αλγόριθμους που θα χρησιμοποιήσουμε, μένει να δημιουργήσουμε το αρχικό σετ δεδομένων (dataset). Στην οπτικοποίηση του dataset, βοηθά το παρακάτω γράφημα.



Εικόνα Ε34: Οπτικοποίηση του dataset σε άξονες TP (Transmission Power) και Energy/packets

Στη συνέχεια, εκπαιδεύσαμε τους αλγορίθμους, διαμοιράζοντας το dataset σε 75% training και σε 25% testing, με σκοπό τα καλύτερα αποτελέσματα. Αναλυτικότερα, κάνοντας χρήση τη μέθοδο 10-fold cross validation, ο k-NN έδειξε με μέσο δείκτη επιτυχίας πάνω από 95%, ότι το k=4 είναι η καλύτερη επιλογή. Χρησιμοποιώντας την ίδια μέθοδο για τα SVM, καταλήξαμε ότι οι βέλτιστες παράμετροι είναι c=10 σε συνδυασμό με τη χρήση της γραμμικής συνάρτησης, παράγοντας μέση βαθμολογία επικύρωσης περίπου 0,95.

Μετά τη διαδικασία εκπαίδευσης, προχωρήσαμε στις δοκιμές με το testing dataset όπου προέκυψαν τα δεδομένα της παρακάτω εικόνας όσον αφορά τις βαθμολογίες σε γνωστά μετρήσιμα μεγέθη (accuracy, precision, recall & F1).

Metric	k-NN	Naive Bayes
Accuracy	0,9692	0,8547
Precision	0,9694	0,8678
Recall	0,9696	0,8549
F1	0,9695	0,8671

Εικόνα Ε35: Βαθμολογίες k-NN και Naive Bayes Αλγορίθμου κατά τις δοκιμές με το testing dataset.

Σύμφωνα λοιπόν με τις παραπάνω μετρήσεις που εξάχθηκαν, φαίνεται πως είναι όντως δυνατή η όσο το δυνατόν πιο βέλτιστη επιλογή του SF που κάνει χρήση η LoRa επικοινωνία, εφόσον γνωρίζουμε εκ των προτέρων το σενάριο με το οποίο ασχολούμαστε, είτε έχουμε ως πληροφορία συγκεκριμένα τοπολογικά δεδομένα (π.χ. απόσταση επικοινωνίας που θέλουμε να επιτύχουμε με όσο το δυνατόν λιγότερα λάθη, πλήθος nodes και gateway στο LoRa network καθώς και περιβαλλοντολογικές συνθήκες όπως περιοχή υλοποίησης), είτε σύνολο δεδομένων από δοκιμές με διαφορετικούς παράγοντες διάδοσης κατά την αρχική υλοποίηση του σεναρίου της LoRa επικοινωνίας που θέλουμε να επιτύχουμε.

5.3 Οφέλη

Όπως υποδηλώνει το όνομα, το Lora είναι ένα πρωτόκολλο μεγάλης εμβέλειας. Είναι σε θέση να μεταδίδει δεδομένα σε μεγάλες αποστάσεις. Μια ενιαία πύλη μπορεί να καλύψει εκατό χιλιόμετρα τετραγωνικά της περιοχής. Το μεγάλο εύρος της τεχνολογίας LoRa οφείλεται στον προϋπολογισμό της σύνδεσης και στη διαμόρφωση φάσματος chirp spread που χρησιμοποιεί.

Η LoRa χρησιμοποιεί τεχνική διαμόρφωσης φάσματος chirp spread. Αυτή η τεχνική χρησιμοποιείται στη στρατιωτική και διαστημική επικοινωνία για πάνω από δεκαετίες λόγω της ισχυρής φύσης της και της χωρητικότητας μεγάλης εμβέλειας. Τώρα χρησιμοποιείται εμπορικά στην επικοινωνία LoRa. Παρέχει επίσης ανοσία σε πολλαπλές διαδρομές και ξεθώριασμα. Το φάσμα εξάπλωσης chirp έχει χαμηλή απαίτηση ισχύος μετάδοσης. Το Chirp είναι ένα σήμα του οποίου η συχνότητα αυξάνεται ή μειώνεται με την πάροδο του χρόνου. Έτσι, ένα σήμα chirp μπορεί να είναι up-chirp και down-chip. Στη διαμόρφωση φάσματος εξάπλωσης chirp το επιθυμητό σήμα δεδομένων πολλαπλασιάζεται με το σήμα chirp. Αυτό απλώνει το εύρος ζώνης πέρα από το εύρος ζώνης του αρχικού σήματος δεδομένων. Στο τέλος του δέκτη το λαμβανόμενο σήμα πολλαπλασιάζεται εκ νέου με το τοπικά παραγόμενο αντίγραφο του σήματος chirp.

Τα πιο σημαντικά κριτήρια μιας ενσωματωμένης συσκευής είναι η διάρκεια ζωής της μπαταρίας. Οι περισσότερες από τις ενσωματωμένες συσκευές πρέπει να επικοινωνούν με

άλλες συσκευές κοντά ή μακριά. Αυτό καταναλωτές υψηλής ισχύος. Οι ενσωματωμένες συσκευές λειτουργούν κυρίως με μπαταρία. Έτσι, η βασική απαίτηση αυτών των ενσωματωμένων συσκευών είναι η διάρκεια ζωής της μπαταρίας. Τα περισσότερα από τα πρωτόκολλα ή τις τεχνικές που χρησιμοποιούνται για τη δημιουργία ενσωματωμένης συσκευής IoT σήμερα καταναλώνουν πολύ υψηλή ισχύ μειώνοντας έτσι τη διάρκεια ζωής της μπαταρίας. Το LoRa βελτιστοποιεί την κατανάλωση μπαταρίας σε μια συσκευή και είναι πιο κατάλληλο για ενσωματωμένη συσκευή που λειτουργεί με μπαταρία. Η LoRa καταναλώνει τη λιγότερη ενέργεια σε σύγκριση με όλες τις υπάρχουσες τεχνολογίες.

6

Συμπεράσματα – Μελλοντική Εργασία

Η μεγάλης απόστασης μετάδοση της ασύρματης τεχνολογίας LoRa το καθιστά από τις πρώτες επιλογές για ευρεία χρήση στα έξυπνα κτήρια, αλλά οι ανησυχίες της αξιοπιστίας ψηφιακών σημάτων στα διαφορετικά περιβάλλοντα οικοδόμησης οδηγούν ακόμα σε αμφιβολίες όσον αφορά τη παγίωση της τεχνολογίας αυτής, σε πρακτικό επίπεδο. Σε αυτή τη μελέτη, δύο σημαντικές μετρήσεις, RTT και PDR, εξετάζονται και αξιολογούνται κατά την εκτίμηση της αξιοπιστίας της διάδοσης σήματος LoRa. Το RTT είναι μια βασική παράμετρος για τη μέτρηση της απόδοσης σε πραγματικό χρόνο του συστήματος ελέγχου και το PDR μπορεί να ποσοτικοποιήσει αποτελεσματικά την αξιοπιστία του ασύρματου συστήματος επικοινωνίας.

Ωστόσο, το LoRaWAN είναι ένα ανοιχτό πρότυπο που δημιουργήθηκε για να περιορίσει την ενεργειακή κατανάλωση και να ενισχύσει αποτελεσματικά ένα δίκτυο. Οι πολιτικές TX και RX εισάγονται από τις τάξεις LoRaWAN για τη ρύθμιση της χρήσης ενέργειας τελικού κόμβου και της μέσης πρόσβασης. Το LoRaWAN έχει πλεονεκτήματα σε σχέση με άλλες τεχνολογίες επικοινωνίας, καθώς είναι ένα ανοιχτό πρότυπο με ενσωματωμένη ασφάλεια και κωδικοποίηση από άκρο σε άκρο. Παρέχει επίσης σημαντικά πλεονεκτήματα όσον αφορά την δυνατότητα επικοινωνίας μεγάλης εμβέλειας, τη χαμηλή κατανάλωση ενέργειας και τις εναλλακτικές λύσεις για την ιδίο-μόρφωση ερασιτεχνών ή λοιπών ενδιαφερομένων, προσφέροντας αισθητά χαμηλότερο κόστος αρχικής προμήθειας εξοπλισμού - με ανοιχτό πηγαίο κώδικα. Τα δίκτυα LoRaWAN περιορίζονται για εφαρμογές σε πραγματικό χρόνο

λόγω περιορισμών που έχουν τεθεί από την ΕΕ, σχετικά με το ποσοστό του κύκλου λειτουργίας τους. Επιπλέον, θα υποστηρίζαμε ότι το LoRaWAN είναι κατάλληλο για καταστάσεις όπου οι μεταδόσεις δεδομένων είναι σπάνιες (μερικά πακέτα την ημέρα) και το μέγεθος ωφέλιμου φορτίου είναι μεταξύ 10 και 50 byte. Το LoRaWAN μπορεί να έχει μεγάλο αντίκτυπο στο έξυπνο δίκτυο, την έξυπνη πόλη, την έξυπνη γεωργία και τα συστήματα απομακρυσμένης παρακολούθησης.

7

Βιβλιογραφία

[1] "Kevin Ashton Wikipedia," Wikimedia Foundation, 24 October 2022. [Online]. Available: https://en.wikipedia.org/wiki/Kevin_Ashton.

[2] «IoT and Security,» Worldwide HQ, 2022 April 2022. [Ηλεκτρονικό]. Available: https://www.insiderintelligence.com/insights/iot-security-privacy/.

[3] «Science.Org,» American Association for the Advancement of Science, 9 February 2015. [Ηλεκτρονικό]. Available: https://www.science.org/content/article/could-wireless-pacemaker-let-hackers-take-control-your-heart.

[4] «A Brief History of the Internet,» University System of Georgia, [Ηλεκτρονικό]. Available: https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml.

[5] «Tim Berners-Lee,» Wikimedia Foundation Inc, 10 December 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Tim_Berners-Lee.

[6] «IoT, Wiki,» Wikimedia Foundation, Inc., 12 December 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Internet_of_things.

[7] «Ubiquitous Computing Wiki,» Wikimedia Foundation Inc., 19 November 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Ubiquitous_computing.

[8] «ITU - IoT,» ITU, 15 06 2012. [Ηλεκτρονικό]. Available: https://handle.itu.int/11.1002/1000/11559 .

[9] «IoT History,» PostScapes, 11 December 2019. [Ηλεκτρονικό]. Available: https://www.postscapes.com/iot-history/.

[10] «Mobility,BigData,Cloud,» Harbringer Systems, 28 March 2014. [Ηλεκτρονικό]. Available: https://harbinger-systems.com/blog/2014/03/role-of-mobility-big-data-and-cloud-in-internet-of-things/.

[11] «Gartner-2013-IoT Devices-Forecast,» Gartner, 18 November 2013. [Ηλεκτρονικό]. Available: https://www.gartner.com/en/documents/2625419.

[12] «Morgan-Stanley-Analysis-IoT-by2025,» 2013 October 2013. [Ηλεκτρονικό]. Available: https://www.businessinsider.com/75-billion-devices-will-be-connected-to-the-internet-by-2020-2013-10.

[13] Huawei, «Huawei Suport Site,» Huawei, 08 04 2022. [Ηλεκτρονικό]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1000113315/bd9557ad/80211-

protocols. [Πρόσβαση 2022].

[14] Wikipedia Project Bluetooth LE, «Bluetooth Low Energy From Wikipedia, the free encyclopedia,» Wikimedia Foundation Inc, 17 12 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy.

[15] Wikimedia Foundation, Inc., «GSM-Wikipedia,» Wikimedia Foundation Inc, 16 12 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/GSM.

[16] W. Contributors, «Narrowband IoT,» Wikipedia Contributors, 11 10 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Narrowband_IoT.

[17] W. Contributors, «Near-field communication,» Wikipedia Contributors, 16 12 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Near-field_communication.

[18] W. Contributors, «Weightless (wireless communications),» Wikipedia Contributors, 6 12 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Weightless_(wireless_communications).

[19] W. Contributors, «Z-Wave,» Wikipedia Contributors, 7 12 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Z-Wave.

[20] W. Contributors, «Zigbee,» Wikipedia Contributors, 21 12 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Zigbee.

[21] L. Slats, «A Brief History of LoRa®: Three Inventors Share Their Personal Story at The Things Conference,» 08 01 2020. [Ηλεκτρονικό]. Available: https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference.

[22] R. Wenner, «LoRa CHIRP,» Youtube, 17 11 2017. [Ηλεκτρονικό]. Available: https://www.youtube.com/watch?v=dxYY097QNs0&ab_channel=RichardWenner. [Πρόσβαση 2022].

[23] J.-F. D. J.-J. C. R. J. G. A. Taoufik Bouguera, «Energy Consumption Model for Sensor Nodes Based on LoRa and LoRaWAN,» *MDPI,* p. 23, 2018.

[24] L. Alliance, «What is LoRaWAN,» 11 2015. [Ηλεκτρονικό]. Available: https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2020/11/what-is-lorawan.pdf?time=1672677007.

[25] Semtech, «Lora Developer Portal,» Lora Developer Portal - LoRa® and LoRaWAN Documentation Official, [Ηλεκτρονικό]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan.

[26] T. T. Network, «End Device Activation,» The Things Network, [Ηλεκτρονικό]. Available: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/.

[27] L. A. T. Committee, «LoRaWAN 1.1 Specification,» LoRa Alliance Technical Committee, 11 10 2017. [Ηλεκτρονικό]. Available: https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf?time=1672677007.

[28] «PySense Datasheet,» Pycom, 6 2 2020. [Ηλεκτρονικό]. Available: https://docs.pycom.io/gitbook/assets/pysense-specsheet.pdf.

[29] «FiPy Specifications,» PyCom, [Ηλεκτρονικό]. Available: https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_FiPy_v2.pdf.

[30] K. Iyer, «Semtech UDP Packet Forwarder,» The Things Stack, 14 10 2022. [Ηλεκτρονικό]. Available: https://www.thethingsindustries.com/docs/gateways/concepts/udp/.

[31] G. Damien, «MicroPython Official WebSite,» [Ηλεκτρονικό]. Available: http://www.micropython.org/.

8

Παράρτημα Κώδικα

Παράρτημα 1: Config

```
#!/usr/bin/env python
#
# Copyright (c) 2019, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#

""" LoPy LoRaWAN Nano Gateway configuration options """

import machine
import ubinascii

WIFI_MAC = ubinascii.hexlify(machine.unique_id()).upper()
# Set  the Gateway ID to be the first 3 bytes of MAC address + 'FFFE' +
last 3 bytes of MAC address
GATEWAY_ID = WIFI_MAC[:6] + "FFFE" + WIFI_MAC[6:12]

SERVER = 'router.eu.thethings.network'
PORT = 1700

NTP = "0.gr.pool.ntp.org"
NTP_PERIOD_S = 3600

WIFI_SSID = 'Obi-Wlan-Kenobi'
WIFI_PASS = 'k2w1p6tt45'

# for EU868
LORA_FREQUENCY = 868100000
LORA_GW_DR = "SF7BW125" # DR_5
LORA_NODE_DR = 5

# for US915
# LORA_FREQUENCY = 903900000
# LORA_GW_DR = "SF10BW125" # DR_0
# LORA_NODE_DR = 0
```

Παράρτημα 2: Main

```
#!/usr/bin/env python
#
# Copyright (c) 2019, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#

""" LoPy LoRaWAN Nano Gateway example usage """

import pycom
import config
from nanogateway import NanoGateway
from network import LoRa
import time
import math

pycom.heartbeat(False)
pycom.rgbled(0xFF0000) # red

#pybytes.activate("eyJhIjoiZTBlY2NhMDQtOTMzNS00MjMxLTg0ZWEtNTZkNzNmNzg5YW
IxIiwicyI6IkxFTk9WT1gyNTAgNDkyMCIsInAiOiJrMncxcDZ0dDQ1In0=")

if __name__ == '__main__':
    nanogw = NanoGateway(
        id=config.GATEWAY_ID,
        frequency=config.LORA_FREQUENCY,
        datarate=config.LORA_GW_DR,
        ssid=config.WIFI_SSID,
        password=config.WIFI_PASS,
        server=config.SERVER,
        port=config.PORT,
        ntp_server=config.NTP,
        ntp_period=config.NTP_PERIOD_S
        )

    nanogw.start()
    nanogw._log('You may now press ENTER to enter the REPL')

    input()
```

Παράρτημα 3: Nano gateway

```
#!/usr/bin/env python
#
# Copyright (c) 2019, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
""" LoPy LoRaWAN Nano Gateway. Can be used for both EU868 and
US915. """
import errno
import machine
import ubinascii
import ujson
import uos
import usocket
import utime
import _thread
from micropython import const
from network import LoRa
from network import WLAN
from machine import Timer
PROTOCOL_VERSION = const(2)
PUSH_DATA = const(0)
PUSH_ACK = const(1)
PULL_DATA = const(2)
PULL_ACK = const(4)
PULL_RESP = const(3)
TX_ERR_NONE = 'NONE'
TX_ERR_TOO_LATE = 'TOO_LATE'
TX_ERR_TOO_EARLY = 'TOO_EARLY'
TX_ERR_COLLISION_PACKET = 'COLLISION_PACKET'
TX_ERR_COLLISION_BEACON = 'COLLISION_BEACON'
TX_ERR_TX_FREQ = 'TX_FREQ'
TX_ERR_TX_POWER = 'TX_POWER'
TX_ERR_GPS_UNLOCKED = 'GPS_UNLOCKED'
UDP_THREAD_CYCLE_MS = const(20)
STAT_PK = {
    'stat': {
        'time': '',
        'lati': 0,
        'long': 0,
        'alti': 0,
        'rxnb': 0,
        'rxok': 0,
        'rxfw': 0,
        'ackr': 100.0,
        'dwnb': 0,
        'txnb': 0
    }
}
RX_PK = {
    'rxpk': [{
        'time': '',
        'tmst': 0,
        'chan': 0,
```

```
            'rfch': 0,
            'freq': 0,
            'stat': 1,
            'modu': 'LORA',
            'datr': '',
            'codr': '4/5',
            'rssi': 0,
            'lsnr': 0,
            'size': 0,
            'data': ''
        }]
}
TX_ACK_PK = {
    'txpk_ack': {
        'error': ''
    }
}
class NanoGateway:
    """
    Nano gateway class, set up by default for use with TTN, but
can be configured
    for any other network supporting the Semtech Packet
Forwarder.
    Only required configuration is wifi_ssid and wifi_password
which are used for
    connecting to the Internet.
    """
    def __init__(self, id, frequency, datarate, ssid, password,
server, port, ntp_server='pool.ntp.org', ntp_period=3600):
        self.id = id
        self.server = server
        self.port = port
        self.frequency = frequency
        self.datarate = datarate
        self.ssid = ssid
        self.password = password
        self.ntp_server = ntp_server
        self.ntp_period = ntp_period
        self.server_ip = None
        self.rxnb = 0
        self.rxok = 0
        self.rxfw = 0
        self.dwnb = 0
        self.txnb = 0
        self.sf = self._dr_to_sf(self.datarate)
        self.bw = self._dr_to_bw(self.datarate)
        self.stat_alarm = None
        self.pull_alarm = None
        self.uplink_alarm = None
        self.wlan = None
        self.sock = None
        self.udp_stop = False
        self.udp_lock = _thread.allocate_lock()
        self.lora = None
        self.lora_sock = None
        self.rtc = machine.RTC()
    def start(self):
        """
```

```
        Starts the LoRaWAN nano gateway.
        """
        self._log('Starting LoRaWAN nano gateway with id: {}',
self.id)
        # setup WiFi as a station and connect
        self.wlan = WLAN(mode=WLAN.STA)
        self._connect_to_wifi()
        # get a time sync
        self._log('Syncing time with {} ...', self.ntp_server)
        self.rtc.ntp_sync(self.ntp_server,
update_period=self.ntp_period)
        while not self.rtc.synced():
            utime.sleep_ms(50)
        self._log("RTC NTP sync complete")
        # get the server IP and create an UDP socket
        self.server_ip = usocket.getaddrinfo(self.server,
self.port)[0][-1]
        self._log('Opening UDP socket to {} ({}) port {}...',
self.server, self.server_ip[0], self.server_ip[1])
        self.sock = usocket.socket(usocket.AF_INET,
usocket.SOCK_DGRAM, usocket.IPPROTO_UDP)
        self.sock.setsockopt(usocket.SOL_SOCKET,
usocket.SO_REUSEADDR, 1)
        self.sock.setblocking(False)
        self._log ('Opened UDP socket!')
        # push the first time immediatelly
        self._push_data(self._make_stat_packet())
        # create the alarms
        self.stat_alarm = Timer.Alarm(handler=lambda t:
self._push_data(self._make_stat_packet()), s=60, periodic=True)
        self.pull_alarm = Timer.Alarm(handler=lambda u:
self._pull_data(), s=25, periodic=True)
        # start the UDP receive thread
        self.udp_stop = False
        _thread.start_new_thread(self._udp_thread, ())
        # initialize the LoRa radio in LORA mode
        self._log('Setting up the LoRa radio at {} Mhz using {}',
self._freq_to_float(self.frequency), self.datarate)
        self.lora = LoRa(
            mode=LoRa.LORA, #edw itan to lathos
            frequency=self.frequency,
            bandwidth=self.bw,
            sf=self.sf,
            preamble=8,
            coding_rate=LoRa.CODING_4_5,
            region=LoRa.EU868,
            tx_iq=True
        )
        # create a raw LoRa socket
        self.lora_sock = usocket.socket(usocket.AF_LORA,
usocket.SOCK_RAW)
        self.lora_sock.setblocking(False)
        self.lora_tx_done = False
        self.lora.callback(trigger=(LoRa.RX_PACKET_EVENT |
LoRa.TX_PACKET_EVENT), handler=self._lora_cb)
        self._log('LoRaWAN nano gateway online')
    def stop(self):
        """
```

```
            Stops the LoRaWAN nano gateway.
            """
            self._log('Stopping...')
            # send the LoRa radio to sleep
            self.lora.callback(trigger=None, handler=None)
            self.lora.power_mode(LoRa.SLEEP)
            # stop the NTP sync
            self.rtc.ntp_sync(None)
            # cancel all the alarms
            self.stat_alarm.cancel()
            self.pull_alarm.cancel()
            # signal the UDP thread to stop
            self.udp_stop = True
            while self.udp_stop:
                utime.sleep_ms(50)
            # disable WLAN
            self.wlan.disconnect()
            self.wlan.deinit()
        def _connect_to_wifi(self):
            self.wlan.connect(self.ssid, auth=(None, self.password))
            while not self.wlan.isconnected():
                utime.sleep_ms(5)
            self._log('WiFi connected to: {}', self.ssid)
        def _dr_to_sf(self, dr):
            sf = dr[2:4]
            if sf[1] not in '0123456789':
                sf = sf[:1]
            return int(sf)
        def _dr_to_bw(self, dr):
            bw = dr[-5:]
            if bw == 'BW125':
                return LoRa.BW_125KHZ
            elif bw == 'BW250':
                return LoRa.BW_250KHZ
            else:
                return LoRa.BW_500KHZ
        def _sf_bw_to_dr(self, sf, bw):
            dr = 'SF' + str(sf)
            if bw == LoRa.BW_125KHZ:
                return dr + 'BW125'
            elif bw == LoRa.BW_250KHZ:
                return dr + 'BW250'
            else:
                return dr + 'BW500'
        def _lora_cb(self, lora):
            """
            LoRa radio events callback handler.
            """
            events = lora.events()
            if events & LoRa.RX_PACKET_EVENT:
                self.rxnb += 1
                self.rxok += 1
                rx_data = self.lora_sock.recv(256)
                stats = lora.stats()
                packet = self._make_node_packet(rx_data,
self.rtc.now(), stats.rx_timestamp, stats.sfrx, self.bw,
stats.rssi, stats.snr)
```

```
            packet = self.frequency_rounding_fix(packet,
self.frequency)
            self._push_data(packet)
            self._log('Received packet: {}', packet)
            self.rxfw += 1
        if events & LoRa.TX_PACKET_EVENT:
            self.txnb += 1
            lora.init(
                mode=LoRa.LORA,
                frequency=self.frequency,
                bandwidth=self.bw,
                sf=self.sf,
                preamble=8,
                coding_rate=LoRa.CODING_4_5,
                region=LoRa.EU868,
                tx_iq=True
                )
    def _freq_to_float(self, frequency):
        """
        MicroPython has some inprecision when doing large float
division.
        To counter this, this method first does integer division
until we
        reach the decimal breaking point. This doesn't completely
elimate
        the issue in all cases, but it does help for a number of
commonly
        used frequencies.
        """
        divider = 6
        while divider > 0 and frequency % 10 == 0:
            frequency = frequency // 10
            divider -= 1
        if divider > 0:
            frequency = frequency / (10 ** divider)
        return frequency
    def frequency_rounding_fix(self, packet, frequency):
        freq = str(frequency)[0:3] + '.' + str(frequency)[3]
        start = packet.find("freq\":")
        end = packet.find(",", start)
        packet = packet[:start + 7] + freq + packet[end:]
        return packet
    def _make_stat_packet(self):
        now = self.rtc.now()
        STAT_PK["stat"]["time"] = "%d-%02d-%02d %02d:%02d:%02d
GMT" % (now[0], now[1], now[2], now[3], now[4], now[5])
        STAT_PK["stat"]["rxnb"] = self.rxnb
        STAT_PK["stat"]["rxok"] = self.rxok
        STAT_PK["stat"]["rxfw"] = self.rxfw
        STAT_PK["stat"]["dwnb"] = self.dwnb
        STAT_PK["stat"]["txnb"] = self.txnb
        return ujson.dumps(STAT_PK)
    def _make_node_packet(self, rx_data, rx_time, tmst, sf, bw,
rssi, snr):
        RX_PK["rxpk"][0]["time"] = "%d-%02d-%02dT%02d:%02d:%02d.
%dZ" % (rx_time[0], rx_time[1], rx_time[2], rx_time[3],
rx_time[4], rx_time[5], rx_time[6])
        RX_PK["rxpk"][0]["tmst"] = tmst
```

```
        RX_PK["rxpk"][0]["freq"] =
self._freq_to_float(self.frequency)
        RX_PK["rxpk"][0]["datr"] = self._sf_bw_to_dr(sf, bw)
        RX_PK["rxpk"][0]["rssi"] = rssi
        RX_PK["rxpk"][0]["lsnr"] = snr
        RX_PK["rxpk"][0]["data"] = ubinascii.b2a_base64(rx_data)
[:-1]
        RX_PK["rxpk"][0]["size"] = len(rx_data)
        return ujson.dumps(RX_PK)
    def _push_data(self, data):
        token = uos.urandom(2)
        packet = bytes([PROTOCOL_VERSION]) + token +
bytes([PUSH_DATA]) + ubinascii.unhexlify(self.id) + data
        with self.udp_lock:
            try:
                self.sock.sendto(packet, self.server_ip)
            except Exception as ex:
                self._log('Failed to push uplink packet to
server: {}', ex)
    def _pull_data(self):
        token = uos.urandom(2)
        packet = bytes([PROTOCOL_VERSION]) + token +
bytes([PULL_DATA]) + ubinascii.unhexlify(self.id)
        with self.udp_lock:
            try:
                self.sock.sendto(packet, self.server_ip)
            except Exception as ex:
                self._log('Failed to pull downlink packets from
server: {}', ex)
    def _ack_pull_rsp(self, token, error):
        TX_ACK_PK["txpk_ack"]["error"] = error
        resp = ujson.dumps(TX_ACK_PK)
        packet = bytes([PROTOCOL_VERSION]) + token +
bytes([PULL_ACK]) + ubinascii.unhexlify(self.id) + resp
        with self.udp_lock:
            try:
                self.sock.sendto(packet, self.server_ip)
            except Exception as ex:
                self._log('PULL RSP ACK exception: {}', ex)
    def _send_down_link(self, data, tmst, datarate, frequency):
        """
        Transmits a downlink message over LoRa.
        """
        self.lora.init(
            mode=LoRa.LORA,
            frequency=frequency,
            bandwidth=self._dr_to_bw(datarate),
            sf=self._dr_to_sf(datarate),
            preamble=8,
            coding_rate=LoRa.CODING_4_5,
            region=LoRa.EU868,
            tx_iq=True
            )
        #while utime.ticks_cpu() < tmst:
        #    pass
        self.lora_sock.send(data)
        self._log(
```

```
            'Sent downlink packet scheduled on {:.3f}, at {:.3f}
Mhz using {}: {}',
            tmst / 1000000,
            self._freq_to_float(frequency),
            datarate,
            data
        )
    def _send_down_link_class_c(self, data, datarate, frequency):
        self.lora.init(
            mode=LoRa.LORA,
            frequency=frequency,
            bandwidth=self._dr_to_bw(datarate),
            sf=self._dr_to_sf(datarate),
            preamble=8,
            coding_rate=LoRa.CODING_4_5,
            tx_iq=True,
            device_class=LoRa.CLASS_C
            )
        self.lora_sock.send(data)
        self._log(
            'Sent downlink packet scheduled on {:.3f}, at {:.3f}
Mhz using {}: {}',
            utime.time(),
            self._freq_to_float(frequency),
            datarate,
            data
        )
    def _udp_thread(self):
        """
        UDP thread, reads data from the server and handles it.
        """
        while not self.udp_stop:
            try:
                data, src = self.sock.recvfrom(1024)
                _token = data[1:3]
                _type = data[3]
                if _type == PUSH_ACK:
                    self._log("Push ack")
                elif _type == PULL_ACK:
                    self._log("Pull ack")
                elif _type == PULL_RESP:
                    self.dwnb += 1
                    ack_error = TX_ERR_NONE
                    tx_pk = ujson.loads(data[4:])
                    if "tmst" in data:
                        tmst = tx_pk["txpk"]["tmst"]
                        t_us = tmst - utime.ticks_cpu() - 15000
                        if t_us < 0:
                            t_us += 0xFFFFFFFF
                        if t_us < 20000000:
                                 self.uplink_alarm = Timer.Alarm(
                                handler=lambda x:
self._send_down_link(

ubinascii.a2b_base64(tx_pk["txpk"]["data"]),
                                    tx_pk["txpk"]["tmst"] - 50,
tx_pk["txpk"]["datr"],
```

```
                                    int(tx_pk["txpk"]["freq"] *
1000) * 1000
                                ),
                                us=t_us
                            )
                        else:
                            ack_error = TX_ERR_TOO_LATE
                            self._log('Downlink timestamp error!,
t_us: {}', t_us)
                    else:
                        self.uplink_alarm = Timer.Alarm(
                            handler=lambda x:
self._send_down_link_class_c(

ubinascii.a2b_base64(tx_pk["txpk"]["data"]),
                                tx_pk["txpk"]["datr"],
                                int(tx_pk["txpk"]["freq"] * 1000)
* 1000
                            ),
                            us=50
                        )
                    self._ack_pull_rsp(_token, ack_error)
                    self._log("Pull rsp")
            except usocket.timeout:
                pass
            except OSError as ex:
                if ex.args[0] != errno.EAGAIN:
                    self._log('UDP recv OSError Exception: {}',
ex)
            except Exception as ex:
                self._log('UDP recv Exception: {}', ex)
            # wait before trying to receive again
            utime.sleep_ms(UDP_THREAD_CYCLE_MS)
        # we are to close the socket
        self.sock.close()
        self.udp_stop = False
        self._log('UDP thread stopped')
    def _log(self, message, *args):
        """
        Outputs a log message to stdout.
        """
        print('[{:>10.3f}] {}'.format(
            utime.ticks_ms() / 1000,
            str(message).format(*args)
            ))
```

Παράρτημα 4: Pymakr

```
{
    "address": "COM3",
    "username": "micro",
    "password": "python",
    "sync_folder": "",
    "open_on_start": true,
    "safe_boot_on_upload": false,
    "py_ignore": [
        "pymakr.conf",
        ".vscode",
        ".gitignore",
        ".git",
        "project.pymakr",
        "env",
        "venv"
    ],
    "fast_upload": false
}
```

Παράρτημα 5: Abp Node

```
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey,
app_swkey))
# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
# creating Cayenne LPP packet
lpp = cayenneLPP.CayenneLPP(size = 100, sock = s)
# set the LoRaWAN data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, LORA_NODE_DR)
# make the socket non-blocking
s.setblocking(False)
#what is needed for measurements
mp = MPL3115A2(py,mode=ALTITUDE)
mpp = MPL3115A2(py,mode=PRESSURE)
si = SI7006A20(py)
t_ambient = 24.4
#sending the measurements
for i in range (200):
    temp = str(mp.temperature())
    hum = str(si.humidity())
    pkt1 = temp
    pkt2 = hum
    #stelnoume ta 2 paketa xwrismena me komma gia delimiter wste
na mporoun na diavastoun
    pkt = pkt1 + ',' + pkt2
    #print status 1
    print('Sending Temp')
    #print Temperature Values
    print ('Temp: ',temp, 'Celsius')
    #print status 2
    print('Sending Hum')
    #print HUmidity Values
    print ('Hum:',hum,'%')
    #to feel better while reading from console
    print ('**********************')
    #steile to paketo meso LoRa sto TTN
    s.send(pkt)
    # lpp.add_temperature(-11.0)
    # temp=float(temp)
    # lpp.add_temperature(int(temp), channel = 118)
    #steile paketo sto Cayenne gia temperature
    # lpp.send(reset_payload = True)
    # lpp.add_relative_humidity(100.0)
    # lpp.add_relative_humidity(0.0, channel = 119)
    # hum=float(hum)
    # lpp.add_relative_humidity(int(hum), channel = 120)
    #steile paketo sto Cayenne gia Humidity
    # lpp.send(reset_payload = True)
    #wait 4 seconds gia downlink
    time.sleep(4)
    rx, port = s.recvfrom(256)
    if rx:
    #se periptosi downlink tupwse status
        print('Received: {}, on port: {}'.format(rx, port))
    #wait 56 seconds for next value --> sunolo 1 x values ana
lepto
    time.sleep(10)
    #sleep gia sunolo mia wra
    #py.setup_sleep(3590)
```

```
#!/usr/bin/env python
#
# Copyright (c) 2019, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
from network import LoRa
import socket
import binascii
import struct
import time
import pycom
import math
from pysense import Pysense
import ubinascii
import cayenneLPP
#import config
from LIS2HH12 import LIS2HH12
from SI7006A20 import SI7006A20
from LTR329ALS01 import LTR329ALS01
from MPL3115A2 import MPL3115A2,ALTITUDE,PRESSURE
py = Pysense()
LORA_FREQUENCY = 868100000
LORA_NODE_DR = 5
COLOUR_WHITE = 0xFFFFFF
COLOUR_BLACK = 0x000000
COLOUR_RED   = 0xFF0000
COLOUR_GREEN = 0x00FF00
COLOUR_BLUE  = 0x0000FF
pycom.heartbeat(False)
pycom.rgbled(COLOUR_GREEN)
# initialize LoRa in LORAWAN mode.
# Please pick the region that matches where you are using the
device:
# Asia = LoRa.AS923
# Australia = LoRa.AU915
# Europe = LoRa.EU868
# United States = LoRa.US915
lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
# create an ABP authentication params
dev_addr = struct.unpack(">l", binascii.unhexlify('26011F51'))[0]
nwk_swkey =
binascii.unhexlify('D174FAC4468644798513924C02C31739')
app_swkey =
binascii.unhexlify('3DF8D0D8782046D5D219B07F258C5711')
# remove all the non-default channels
for i in range(3, 16):
    lora.remove_channel(i)
# set the 3 default channels to the same frequency
lora.add_channel(0, frequency=LORA_FREQUENCY, dr_min=0, dr_max=5)
lora.add_channel(1, frequency=LORA_FREQUENCY, dr_min=0, dr_max=5)
lora.add_channel(2, frequency=LORA_FREQUENCY, dr_min=0, dr_max=5)
# join a network using ABP (Activation By Personalization)
```

```
#py.go_to_sleep()
```

Παράρτημα 6: Boot

```
# boot.py -- run on boot-up
```

Παράρτημα 7: CayenneLPP

```
        """
        Adding an gyrometer reading to the payload

        Resolution: 0.01 deg / sec for each axis, signed.

        Args:
            channel: The channel of the payload.
            value_x: The angular speed on the x axis.
            value_y: The angular speed on the y axis.
            value_z: The angular speed on the z axis.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(GYROMETER[1]):
            value_x = int(value_x * 100) # precision is 0.01 per
axis
            value_y = int(value_y * 100)
            value_z = int(value_z * 100)
            self.payload = (self.payload +
                            bytes([channel]) +
                            GYROMETER[0] +
                            struct.pack('>h', value_x) +
                            struct.pack('>h', value_y) +
                            struct.pack('>h', value_z))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_gps(self, lat, lon, alt, channel = 12):
        """
        Adding an GPS reading to the payload

        Resolution:
            0.0001 deg for the latitude and longitute, signed.
            0.01 meters for the altitude, signed.

        Args:
            channel: The channel of the payload.
            lat: The latitude.
            lon: The longitute.
            alt: The altitude.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(GPS[1]):
```

```
            value_x: The acceleration value on the x axis.
            value_y: The acceleration value on the y axis.
            value_z: The acceleration value on the z axis.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(ACCELEROMETER[1]):
            value_x = int(value_x * 1000) # precision is 0.001
per axis
            value_y = int(value_y * 1000)
            value_z = int(value_z * 1000)
            self.payload = (self.payload +
                            bytes([channel]) +
                            ACCELEROMETER[0] +
                            struct.pack('>h', value_x) +
                            struct.pack('>h', value_y) +
                            struct.pack('>h', value_z))

        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_barometric_pressure(self, value, channel = 10):
        """
        Adding an barometric pressure reading to the payload

        Resolution: 0.1 hPa, unsigned.

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(BAROMETER[1]):
            value = int(value * 10) # precision is 0.1
            self.payload = (self.payload +
                            bytes([channel]) +
                            BAROMETER[0] +
                            struct.pack('>H', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_gyrometer(self, value_x, value_y, value_z, channel =
11):
```

```
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(TEMPERATURE_SENSOR[1]):
            value = int(value * 10) # precision is 0.1
            self.payload = (self.payload +
                            bytes([channel]) +
                            TEMPERATURE_SENSOR[0] +
                            struct.pack('>h', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_relative_humidity(self, value, channel = 8):
        """
        Adding an humidty reading to the payload

        Resolution: 0.5 %, signed.

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(HUMIDITY_SENSOR[1]):
            value = int(value * 2) # precision is 0.5
            self.payload = (self.payload +
                            bytes([channel]) +
                            HUMIDITY_SENSOR[0] +
                            struct.pack('>B', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_accelerometer(self, value_x, value_y, value_z,
channel = 9):
        """
        Adding an accelerometer reading to the payload

        Resolution: 0.001 G per axis, signed.

        Args:
            channel: The channel of the payload.
```

```
        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(ILLUMINANCE_SENSOR[1]):
            value = int(value) # precision is 1
            self.payload = (self.payload +
                            bytes([channel]) +
                            ILLUMINANCE_SENSOR[0] +
                            struct.pack('>H', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_presence(self, value, channel = 6):
        """
        Adding a presence reading to the payload

        Resolution: 1.

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(PRESENCE_SENSOR[1]):
            value = int(value) # precision is 1
            self.payload = (self.payload +
                            bytes([channel]) +
                            PRESENCE_SENSOR[0] +
                            struct.pack('>B', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_temperature(self, value, channel = 7):
        """
        Adding a temperature reading to the payload

        Resolution: 0.1 degrees Celsius, signed.

        Args:
```

```
        Resolution: 0.01, signed.

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(ANALOG_INPUT[1]):
            value = int(value * 100)  # precision is 0.01
            self.payload = (self.payload +
                            bytes([channel]) +
                            ANALOG_INPUT[0] +
                            struct.pack('>h', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_analog_output(self, value, channel = 4):
        """
        Adding an analog output to the payload

        Resolution: 0.01, signed.

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(ANALOG_OUTPUT[1]):
            value = int(value * 100)  # precision is 0.01
            self.payload = (self.payload +
                            bytes([channel]) +
                            ANALOG_OUTPUT[0] +
                            struct.pack('>h', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_luminosity(self, value, channel = 5):
        """
        Adding a luminosity reading to the payload

        Resolution: 1 lux, unsigned.
```

```
        """
        Adding a digital input to the payload

        Resolution: 1, unsigned

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(DIGITAL_INPUT[1]):
            value = int(value) # precision is 1
            self.payload = (self.payload +
                            bytes([channel]) +
                            DIGITAL_INPUT[0] +
                            struct.pack('>B', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_digital_output(self, value, channel = 2):
        """
        Adding a digital output to the payload

        Resolution: 1, unsigned.

        Args:
            channel: The channel of the payload.
            value: The value of the sensor to be converted.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(DIGITAL_OUTPUT[1]):
            value = int(value) # precision is 1
            self.payload = (self.payload +
                            bytes([channel]) +
                            DIGITAL_OUTPUT[0] +
                            struct.pack('>B', value))
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_analog_input(self, value, channel = 3):
        """
        Adding an analog input to the payload
```

```
    def change_size(self, a_size):
        """
        Changing the size

        Args:
            a_size: The new maximum size of the payload.
        """

        self.size = a_size

    def get_size(self):
        """Return the size (number of bits) of the payload"""

        return len(self.payload)

    def get_payload(self):
        """Returning the payload"""

        return self.payload

    def set_socket(self, a_socket):
        """
        Setting the socket

        Args:
            a_socket: A socket.
        """

        self.socket = a_socket;

    def send(self, reset_payload = False):
        """
        Sending the payload via the socket

        Args:
            reset_payload: Indicates whether the payload must be
reset after
                           the transmission (i.e. if a socket is
defined).

        Returns:
            True if a socket is defined, False otherwise. If a
socket is
            defined, then the payload is transmitted using it.
Additionnaly
            the payload of the object may be reset if requested.
        """

        if self.socket is None:
            return False
        else:
            self.socket.send(self.payload)
            if reset_payload:
                self.reset_payload()
            return True

    def add_digital_input(self, value, channel = 1):
```

```
    You can send the payload via the socket using the 'send'
method. The socket
    can be set using the 'set_socket' method.

    To add the data from a sensor, the methods 'add_sensor_name'
are provided.

    Author: Johan Barthelemy
    """

    def __init__(self, size = 11, sock = None):
        """
        Constructor

        Args:
            size: The maximum size (in bytes) for the payload.
Default = 11. If
                the size is lower than 3, then it is set to 3.
            sock: A socket that can be used by the send method to
transmit the
                payload (optional).
        """

        if size < 3:
            size = 3

        self.size = size
        self.payload = bytes()
        self.socket = sock

    def is_within_size_limit(self, a_size):
        """
        Check if adding data will result in a payload size below
size

        The actual size increase is given by a_size + 2 for the
channel and
        sensor type.

        Args:
            a_size: The size of the data to be added to the
payload (in bytes).

        Returns:
            True if the current size of the payload + a_size is
lower than size,
            False otherwise.
        """

        if (len(self.payload) + a_size + 2) <= self.size:
            return True
        else:
            return False

    def reset_payload(self):
        """Reset the payload"""

        self.payload = bytes()
```

```
GYROMETER           = (bytes([134]), 6) # 0.01 deg/sec signed msb
per axis
GPS                 = (bytes([136]), 9) # latitude:  0.0001 degree
signed MSB
                                        # longiture: 0.0001 degree
signed MSB
                                        # altitude:  0.01 meter
signed MSB

class CayenneLPP:
    """
    Class for packing data in the Cayenne LPP format

    The class contains the methods to pack data from sensors in a
Cayenne LPP
    format. The payload structure for the Cayenne LPP format is
data frame of
    the form: [SENSOR_1, SENSOR_2, ... SENSOR_N], where the
format for one
    sensor is defined by: [CHANNEL, SENSOR TYPE, DATA].

    The channel is an unique identifier for each sensor in the
data frame.

    The type of sensors compatible with this class are:
    - digital input/output;
    - analog input/output;
    - luminosity (or illuminance) sensor;
    - presence sensor;
    - temperature sensor;
    - humidity sensor;
    - accelerometer;
    - barometer;
    - gyrometer;
    - gps.

    An object of this class has 3 attributes:
    - payload: the data from one or more senors formatted with
the Cayenne LPP
               format;
    - size: the maximum size of the payload (depends on the
network
            on which the data will be send to);
    - socket: a socket via which we can send the payload.

    The constructor will generate an object with an empty payload
and with a
    maximum size.

    It is possible to reset the payload with the 'reset' method
and change the
    maximum size with the 'change_size' method.

    The current payload and maximum size can be obtained with the
methods
    'get_payload' and 'get_size' methods.
```

```
"""
CayenneLPP module.

A module for the Cayenne Low Power Packet format.

It aims to facilate the conversion of values typically read from
sensors to a
sequence of bits (the payload) that can be send over a network
using the
Cayenne Low Power Packet format. This format is particularly
suited for LPWAN
networks such as LoRaWAN.

The payload can then be send for instance to an application of
The Things
Network, a LoRaWAN-based community network, which will then
forward the data to
a Cayenne application thanks to its Cayenne integration.

The module consists of constants defining the different sensors
and their size
and one class CayenneLPP containing the methods to build a
payload.

The constants have the format NAME_SENSOR = (LPP id, Data size)
where LPP id
is the IPSO id - 3200 and Data size is the number of bytes that
must be used
to encode the reading from the sensor.

More info here:
https://mydevices.com/cayenne/docs/lora/#lora-cayenne-low-power-
payload-overview

Use of this source code is governed by the MIT license that can
be found in the
LICENSE file.
"""

__version__ = '0.5'
__author__  = 'Johan Barthelemy'

import struct

# Some constants of the form
# NAME_SENSOR = (LPP id = IPSO id - 3200, Data size in bytes)
DIGITAL_INPUT      = (bytes([0]),   1) # 1 unsigned (True/False)
DIGITAL_OUTPUT     = (bytes([1]),   1) # 1 unsigned (True/False)
ANALOG_INPUT       = (bytes([2]),   2) # 0.01 signed
ANALOG_OUTPUT      = (bytes([3]),   2) # 0.01 signed
ILLUMINANCE_SENSOR = (bytes([101]), 2) # 1 lux unsigned MSB
PRESENCE_SENSOR    = (bytes([102]), 1) # 1 unsigned (True/False)
TEMPERATURE_SENSOR = (bytes([103]), 2) # 0.1 deg Celcius signed
MSB
HUMIDITY_SENSOR    = (bytes([104]), 1) # 0.5 unsigned
ACCELEROMETER      = (bytes([113]), 6) # 0.001 G signed MSB per
axis
BAROMETER          = (bytes([115]), 2) # 0.1 hPa unsigned MSB
```

```
            lat = int(lat * 10000) # precision is 0.0001 for lat
and lon
            lon = int(lon * 10000)
            alt = int(alt * 100)   # precision is 0.01 for
altitude
            self.payload = (self.payload +
                            bytes([channel]) +
                            GPS[0] +
                            struct.pack('>l', lat)[1:4] +
                            struct.pack('>l', lon)[1:4] +
                            struct.pack('>l', alt)[1:4])
        else:
            raise Exception('payload too big: size exceeds the
limit!')

    def add_generic(self, lpp_id, values, channel = 13, data_size
= 1,
                    is_signed = True, precision = 1):
        """
        Adding an generic sensor reading to the payload

        Resolution:
            Defined by the 'precision' argument (see below). See
also the
            resolution of the other methods.

        Args:
            channel: The channel of the payload.
            lpp_id: The LPP id of the sensor (IPSO id - 3200).
            data_size: The total number of bytes for the payload.
            is_signed: Boolean indicating whether we need to use
signed (True)
                       or unsigned (False) encoding.
            precision: The precision of the sensor reading (e.g.
0.01, 1, 0.5).
            values: The data to be encoded, either a scalar or a
list.

        Raises:
            Exception: raises an exception when the data can't be
added to the
                       payload because the resulting size would
exceeds the
                       maximum.
        """

        if self.is_within_size_limit(data_size):

            # determining the encoding
            enc = ''
            if is_signed:
                enc = '>l'
            else:
                enc = '>L'

            # updating the payload
            self.payload = self.payload + bytes([channel]) +
bytes([lpp_id])
```

```
            if isinstance(values, list):
                values = [int(v / precision) for v in values]
                for v in values:
                    self.payload = self.payload +
struct.pack(enc, v)[-data_size:]
            else:
                values = int(values / precision)
                self.payload = self.payload + struct.pack(enc,
values)[-data_size:]

        else:
            raise Exception('payload too big: size exceeds the
limit!')
```

Παράρτημα 8: LIS2HH12

```
        if duration > 255 * 1000 * 8 / self.ODRS[self.odr]:
            error = "duration %d exceeds max possible value %d" %
(duration, 255 * 1000 * 8 / self.ODRS[self.odr])
            print(error)
            raise ValueError(error)
        if duration < 1000 * 8 / self.ODRS[self.odr]:
            error = "duration %d below resolution %d" %
(duration, 1000 * 8 / self.ODRS[self.odr])
            print(error)
            raise ValueError(error)
        _ths = int(127 * threshold /
self.SCALES[self.full_scale]) & 0x7F
        _dur = int((duration * self.ODRS[self.odr]) / 1000 / 8)
        self.i2c.writeto_mem(ACC_I2CADDR, ACT_THS, _ths)
        self.i2c.writeto_mem(ACC_I2CADDR, ACT_DUR, _dur)
        # enable the activity/inactivity interrupt
        self.set_register(CTRL3_REG, 1, 5, 1)
        self._user_handler = handler
        self.int_pin = Pin('P13', mode=Pin.IN)
        self.int_pin.callback(trigger=Pin.IRQ_FALLING |
Pin.IRQ_RISING, handler=self._int_handler)
        # return actual used threshold and duration
        return (_ths * self.SCALES[self.full_scale] / 128, _dur *
8 * 1000 / self.ODRS[self.odr])
    def activity(self):
        if not self.debounced:
            time.sleep_ms(self.act_dur)
            self.debounced = True
        if self.int_pin():
            return True
        return False
    def _int_handler(self, pin_o):
        if self._user_handler is not None:
            self._user_handler(pin_o)
        else:
            if pin_o():
                print('Activity interrupt')
            else:
                print('Inactivity interrupt'
)
```

```
        whoami = self.i2c.readfrom_mem(ACC_I2CADDR ,
PRODUCTID_REG, 1)
        if (whoami[0] != 0x41):
            raise ValueError("LIS2HH12 not found")
        # enable acceleration readings at 50Hz
        self.set_odr(ODR_50_HZ)
        # change the full-scale to 4g
        self.set_full_scale(FULL_SCALE_4G)
        # set the interrupt pin as active low and open drain
        self.set_register(CTRL5_REG, 3, 0, 3)
        # make a first read
        self.acceleration()
    def acceleration(self):
        x = self.i2c.readfrom_mem(ACC_I2CADDR , ACC_X_L_REG, 2)
        self.x = struct.unpack('<h', x)
        y = self.i2c.readfrom_mem(ACC_I2CADDR , ACC_Y_L_REG, 2)
        self.y = struct.unpack('<h', y)
        z = self.i2c.readfrom_mem(ACC_I2CADDR , ACC_Z_L_REG, 2)
        self.z = struct.unpack('<h', z)
        _mult = self.SCALES[self.full_scale] / ACC_G_DIV
        return (self.x[0] * _mult, self.y[0] * _mult, self.z[0] *
_mult)
    def roll(self):
        x,y,z = self.acceleration()
        rad = math.atan2(-x, z)
        return (180 / math.pi) * rad
    def pitch(self):
        x,y,z = self.acceleration()
        rad = -math.atan2(y, (math.sqrt(x*x + z*z)))
        return (180 / math.pi) * rad
    def set_register(self, register, value, offset, mask):
        reg = bytearray(self.i2c.readfrom_mem(ACC_I2CADDR,
register, 1))
        reg[0] &= ~(mask << offset)
        reg[0] |= ((value & mask) << offset)
        self.i2c.writeto_mem(ACC_I2CADDR, register, reg)
    def set_full_scale(self, scale):
        self.set_register(CTRL4_REG, scale, 4, 3)
        self.full_scale = scale
    def set_odr(self, odr):
        self.set_register(CTRL1_REG, odr, 4, 7)
        self.odr = odr
    def set_high_pass(self, hp):
        self.set_register(CTRL2_REG, 1 if hp else 0, 2, 1)
    def enable_activity_interrupt(self, threshold, duration,
handler=None):
        # Threshold is in mg, duration is ms
        self.act_dur = duration
        if threshold > self.SCALES[self.full_scale]:
            error = "threshold %d exceeds full scale %d" %
(threshold, self.SCALES[self.full_scale])
            print(error)
            raise ValueError(error)
        if threshold < self.SCALES[self.full_scale] / 128:
            error = "threshold %d below resolution %d" %
(threshold, self.SCALES[self.full_scale]/128)
            print(error)
            raise ValueError(error)
```

```
#!/usr/bin/env python
#
# Copyright (c) 2020, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
import math
import time
import struct
from machine import Pin
FULL_SCALE_2G = const(0)
FULL_SCALE_4G = const(2)
FULL_SCALE_8G = const(3)
ODR_POWER_DOWN = const(0)
ODR_10_HZ = const(1)
ODR_50_HZ = const(2)
ODR_100_HZ = const(3)
ODR_200_HZ = const(4)
ODR_400_HZ = const(5)
ODR_800_HZ = const(6)
ACC_G_DIV = 1000 * 65536
class LIS2HH12:
    ACC_I2CADDR = const(30)
    PRODUCTID_REG = const(0x0F)
    CTRL1_REG = const(0x20)
    CTRL2_REG = const(0x21)
    CTRL3_REG = const(0x22)
    CTRL4_REG = const(0x23)
    CTRL5_REG = const(0x24)
    ACC_X_L_REG = const(0x28)
    ACC_X_H_REG = const(0x29)
    ACC_Y_L_REG = const(0x2A)
    ACC_Y_H_REG = const(0x2B)
    ACC_Z_L_REG = const(0x2C)
    ACC_Z_H_REG = const(0x2D)
    ACT_THS = const(0x1E)
    ACT_DUR = const(0x1F)
    SCALES = {FULL_SCALE_2G: 4000, FULL_SCALE_4G: 8000,
FULL_SCALE_8G: 16000}
    ODRS = [0, 10, 50, 100, 200, 400, 800]
    def __init__(self, pysense = None, sda = 'P22', scl = 'P21'):
        if pysense is not None:
            self.i2c = pysense.i2c
        else:
            from machine import I2C
            self.i2c = I2C(0, mode=I2C.MASTER, pins=(sda, scl))
        self.odr = 0
        self.full_scale = 0
        self.x = 0
        self.y = 0
        self.z = 0
        self.int_pin = None
        self.act_dur = 0
        self.debounced = False
```

Παράρτημα 9: LTR329ALS01

```
#!/usr/bin/env python
#
# Copyright (c) 2020, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
import time
from machine import I2C
class LTR329ALS01:
    ALS_I2CADDR = const(0x29) # The device's I2C address
    ALS_CONTR_REG = const(0x80)
    ALS_MEAS_RATE_REG = const(0x85)
    ALS_DATA_CH1_LOW = const(0x88)
    ALS_DATA_CH1_HIGH = const(0x89)
    ALS_DATA_CH0_LOW = const(0x8A)
    ALS_DATA_CH0_HIGH = const(0x8B)
    ALS_GAIN_1X = const(0x00)
    ALS_GAIN_2X = const(0x01)
    ALS_GAIN_4X = const(0x02)
    ALS_GAIN_8X = const(0x03)
    ALS_GAIN_48X = const(0x06)
    ALS_GAIN_96X = const(0x07)
    ALS_INT_50 = const(0x01)
    ALS_INT_100 = const(0x00)
    ALS_INT_150 = const(0x04)
    ALS_INT_200 = const(0x02)
    ALS_INT_250 = const(0x05)
    ALS_INT_300 = const(0x06)
    ALS_INT_350 = const(0x07)
    ALS_INT_400 = const(0x03)
    ALS_RATE_50 = const(0x00)
    ALS_RATE_100 = const(0x01)
    ALS_RATE_200 = const(0x02)
    ALS_RATE_500 = const(0x03)
    ALS_RATE_1000 = const(0x04)
    ALS_RATE_2000 = const(0x05)
    def __init__(self, pysense = None, sda = 'P22', scl = 'P21',
gain = ALS_GAIN_1X, integration = ALS_INT_100, rate =
ALS_RATE_500):
        if pysense is not None:
            self.i2c = pysense.i2c
        else:
            self.i2c = I2C(0, mode=I2C.MASTER, pins=(sda, scl))
        contr = self._getContr(gain)
        self.i2c.writeto_mem(ALS_I2CADDR, ALS_CONTR_REG,
bytearray([contr]))
        measrate = self._getMeasRate(integration, rate)
        self.i2c.writeto_mem(ALS_I2CADDR, ALS_MEAS_RATE_REG,
bytearray([measrate]))
        time.sleep(0.01)
    def _getContr(self, gain):
        return ((gain & 0x07) << 2) + 0x01
    def _getMeasRate(self, integration, rate):
        return ((integration & 0x07) << 3) + (rate & 0x07)
```

```
    def _getWord(self, high, low):
        return ((high & 0xFF) << 8) + (low & 0xFF)
    def light(self):
        ch1low = self.i2c.readfrom_mem(ALS_I2CADDR ,
ALS_DATA_CH1_LOW, 1)
        ch1high = self.i2c.readfrom_mem(ALS_I2CADDR ,
ALS_DATA_CH1_HIGH, 1)
        data1 = int(self._getWord(ch1high[0], ch1low[0]))
        ch0low = self.i2c.readfrom_mem(ALS_I2CADDR ,
ALS_DATA_CH0_LOW, 1)
        ch0high = self.i2c.readfrom_mem(ALS_I2CADDR ,
ALS_DATA_CH0_HIGH, 1)
        data0 = int(self._getWord(ch0high[0], ch0low[0]))
        return (data0, data1)
```

Παράρτημα 10: Main

```
#!/usr/bin/env python
#
# Copyright (c) 2020, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
# See https://docs.pycom.io for more information regarding
library specifics
import time
import math
import pycom
#import config
import abp_node
from pysense import Pysense
import machine
import ubinascii
import binascii
from network import LoRa
from LIS2HH12 import LIS2HH12
from SI7006A20 import SI7006A20
from LTR329ALS01 import LTR329ALS01
from MPL3115A2 import MPL3115A2,ALTITUDE,PRESSURE
# pycom.heartbeat(False)
# pycom.rgbled(0x800080) # purple
py = Pysense()
#
pybytes.activate("eyJhIjoiMzM0NTYzY2QtNGQwMi00ZmE1LWE5Y2QtN2YyNmR
kNGJiZTcwIiwicyI6IkxFTk9WTlgyNTAgNDkyMCIsInAiOiJrMncxcDZ0dDQ1In0=
")
mp = MPL3115A2(py,mode=ALTITUDE) # Returns height in meters. Mode
may also be set to PRESSURE, returning a value in Pascals
print("MPL3115A2 temperature: " + str(mp.temperature()))
print("Altitude: " + str(mp.altitude()))
mpp = MPL3115A2(py,mode=PRESSURE) # Returns pressure in Pa. Mode
may also be set to ALTITUDE, returning a value in meters
print("Pressure: " + str(mpp.pressure()))
si = SI7006A20(py)
print("Temperature: " + str(si.temperature())+ " deg C and
Relative Humidity: " + str(si.humidity()) + " %RH")
print("Dew point: "+ str(si.dew_point()) + " deg C")
t_ambient = 24.4
print("Humidity Ambient for " + str(t_ambient) + " deg C is " +
str(si.humid_ambient(t_ambient)) + "%RH")
lt = LTR329ALS01(py)
print("Light (channel Blue lux, channel Red lux): " +
str(lt.light()))
li = LIS2HH12(py)
print("Acceleration: " + str(li.acceleration()))
print("Roll: " + str(li.roll()))
print("Pitch: " + str(li.pitch()))
#print("Battery voltage: " + str(py.read_battery_voltage()))
#print Device EUI
DevEUI = ubinascii.hexlify(machine.unique_id()).decode('ascii')
lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
```

```
print(binascii.hexlify(lora.mac()).upper().decode('utf-8'))
# while True:
    # for i  in range (0,3):
    #     print ('Loop Begin')
    #     pybytes.send_signal(1,str(si.temperature()))
    #     print('sent temp {}'.format(i))
    #     print("MPL3115A2 temperature: " +
str(mp.temperature()))
    #     pybytes.send_signal(2,str(si.humidity()))
    #     print('sent hum {}'.format(i))
    #     print("Temperature: " + str(si.temperature())+ " deg C
and Relative Humidity: " + str(si.humidity()) + " %RH")
    #     pybytes.send_signal(3,str(py.read_battery_voltage()))
    #     print('sent Batt Voltage {}'.format(i))
    #     print ('*****Loop END*****')
    #     print ('See you in 10 seconds....')
    #     time.sleep(10)
    #     py.setup_sleep(20)
    #     py.go_to_sleep()
#time.sleep(3)
```

Παράρτημα 11: MPL3115A2

```
#!/usr/bin/env python
#
# Copyright (c) 2020, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
import time
from machine import I2C
ALTITUDE = const(0)
PRESSURE = const(1)
class MPL3115A2exception(Exception):
    pass
class MPL3115A2:
    MPL3115_I2CADDR = const(0x60)
    MPL3115_STATUS = const(0x00)
    MPL3115_PRESSURE_DATA_MSB = const(0x01)
    MPL3115_PRESSURE_DATA_CSB = const(0x02)
    MPL3115_PRESSURE_DATA_LSB = const(0x03)
    MPL3115_TEMP_DATA_MSB = const(0x04)
    MPL3115_TEMP_DATA_LSB = const(0x05)
    MPL3115_DR_STATUS = const(0x06)
    MPL3115_DELTA_DATA = const(0x07)
    MPL3115_WHO_AM_I = const(0x0c)
    MPL3115_FIFO_STATUS = const(0x0d)
    MPL3115_FIFO_DATA = const(0x0e)
    MPL3115_FIFO_SETUP = const(0x0e)
    MPL3115_TIME_DELAY = const(0x10)
    MPL3115_SYS_MODE = const(0x11)
    MPL3115_INT_SORCE = const(0x12)
    MPL3115_PT_DATA_CFG = const(0x13)
    MPL3115_BAR_IN_MSB = const(0x14)
    MPL3115_P_ARLARM_MSB = const(0x16)
    MPL3115_T_ARLARM = const(0x18)
    MPL3115_P_ARLARM_WND_MSB = const(0x19)
    MPL3115_T_ARLARM_WND = const(0x1b)
    MPL3115_P_MIN_DATA = const(0x1c)
    MPL3115_T_MIN_DATA = const(0x1f)
    MPL3115_P_MAX_DATA = const(0x21)
    MPL3115_T_MAX_DATA = const(0x24)
    MPL3115_CTRL_REG1 = const(0x26)
    MPL3115_CTRL_REG2 = const(0x27)
    MPL3115_CTRL_REG3 = const(0x28)
    MPL3115_CTRL_REG4 = const(0x29)
    MPL3115_CTRL_REG5 = const(0x2a)
    MPL3115_OFFSET_P = const(0x2b)
    MPL3115_OFFSET_T = const(0x2c)
    MPL3115_OFFSET_H = const(0x2d)
    def __init__(self, pysense = None, sda = 'P22', scl = 'P21',
mode = PRESSURE):
        if pysense is not None:
            self.i2c = pysense.i2c
        else:
            self.i2c = I2C(0, mode=I2C.MASTER, pins=(sda, scl))
        self.STA_reg = bytearray(1)
```

```
        self.mode = mode
        if self.mode is PRESSURE:
            self.i2c.writeto_mem(MPL3115_I2CADDR,
MPL3115_CTRL_REG1, bytes([0x38])) # barometer mode, not raw,
oversampling 128, minimum time 512 ms
            self.i2c.writeto_mem(MPL3115_I2CADDR,
MPL3115_PT_DATA_CFG, bytes([0x07])) # no events detected
            self.i2c.writeto_mem(MPL3115_I2CADDR,
MPL3115_CTRL_REG1, bytes([0x39])) # active
        elif self.mode is ALTITUDE:
            self.i2c.writeto_mem(MPL3115_I2CADDR,
MPL3115_CTRL_REG1, bytes([0xB8])) # altitude mode, not raw,
oversampling 128, minimum time 512 ms
            self.i2c.writeto_mem(MPL3115_I2CADDR,
MPL3115_PT_DATA_CFG, bytes([0x07])) # no events detected
            self.i2c.writeto_mem(MPL3115_I2CADDR,
MPL3115_CTRL_REG1, bytes([0xB9])) # active
        else:
            raise MPL3115A2exception("Invalid Mode MPL3115A2")
        if self._read_status():
            pass
        else:
            raise MPL3115A2exception("Error with MPL3115A2")
    def _read_status(self):
        while True:
            self.i2c.readfrom_mem_into(MPL3115_I2CADDR,
MPL3115_STATUS, self.STA_reg)
            if(self.STA_reg[0] == 0):
                time.sleep(0.01)
                pass
            elif(self.STA_reg[0] & 0x04) == 4:
                return True
            else:
                return False
    def pressure(self):
        if self.mode == ALTITUDE:
            raise MPL3115A2exception("Incorrect Measurement Mode
MPL3115A2")
        OUT_P_MSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_PRESSURE_DATA_MSB,1)
        OUT_P_CSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_PRESSURE_DATA_CSB,1)
        OUT_P_LSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_PRESSURE_DATA_LSB,1)
        return float((OUT_P_MSB[0] << 10) + (OUT_P_CSB[0] << 2) +
((OUT_P_LSB[0] >> 6) & 0x03) + ((OUT_P_LSB[0] >> 4) & 0x03) /
4.0)
    def altitude(self):
        if self.mode == PRESSURE:
            raise MPL3115A2exception("Incorrect Measurement Mode
MPL3115A2")
        OUT_P_MSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_PRESSURE_DATA_MSB,1)
        OUT_P_CSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_PRESSURE_DATA_CSB,1)
        OUT_P_LSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_PRESSURE_DATA_LSB,1)
        alt_int = (OUT_P_MSB[0] << 8) + (OUT_P_CSB[0])
```

```
        alt_frac = ((OUT_P_LSB[0] >> 4) & 0x0F)
        if alt_int > 32767:
            alt_int -= 65536
        return float(alt_int + alt_frac / 16.0)
    def temperature(self):
        OUT_T_MSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_TEMP_DATA_MSB,1)
        OUT_T_LSB = self.i2c.readfrom_mem(MPL3115_I2CADDR,
MPL3115_TEMP_DATA_LSB,1)
        temp_int = OUT_T_MSB[0]
        temp_frac = OUT_T_LSB[0]
        if temp_int > 127:
            temp_int -= 256
        return float(temp_int + temp_frac / 256.0)
```

Παράρτημα 12: Pycoproc

```
        # and then there is a binary prescaler, e.g., 1, 2, 4 ...
512, 1024 ms
        # hence the need for the constant
        self._write(bytes([CMD_CALIBRATE]), wait=False)
        self.i2c.deinit()
        Pin('P21', mode=Pin.IN)
        pulses = pycom.pulses_get('P21', 100)
        self.i2c.init(mode=I2C.MASTER, pins=(self.sda, self.scl))
        idx = 0
        for i in range(len(pulses)):
            if pulses[i][1] > EXP_RTC_PERIOD:
                idx = i
                break
        try:
            period = pulses[idx][1] - pulses[(idx - 1)][1]
        except:
            period = 0
        if period > 0:
            self.clk_cal_factor = (EXP_RTC_PERIOD / period) *
(1000 / 1024)
        if self.clk_cal_factor > 1.25 or self.clk_cal_factor <
0.75:
            self.clk_cal_factor = 1
    def button_pressed(self):
        button = self.peek_memory(PORTA_ADDR) & (1 << 3)
        return not button
    def read_battery_voltage(self):
        self.set_bits_in_memory(ADCON0_ADDR,
_ADCON0_GO_nDONE_MASK)
        time.sleep_us(50)
        while self.peek_memory(ADCON0_ADDR) &
_ADCON0_GO_nDONE_MASK:
            time.sleep_us(100)
        adc_val = (self.peek_memory(ADRESH_ADDR) << 2) +
(self.peek_memory(ADRESL_ADDR) >> 6)
        return (((adc_val * 3.3 * 280) / 1023) / 180) + 0.01    #
add 10mV to compensate for the drop in the FET
    def setup_int_wake_up(self, rising, falling):
        """ rising is for activity detection, falling for
inactivity """
        wake_int = False
        if rising:
            self.set_bits_in_memory(IOCAP_ADDR, 1 << 5)
            wake_int = True
        else:
            self.mask_bits_in_memory(IOCAP_ADDR, ~(1 << 5))
        if falling:
            self.set_bits_in_memory(IOCAN_ADDR, 1 << 5)
            wake_int = True
        else:
            self.mask_bits_in_memory(IOCAN_ADDR, ~(1 << 5))
        self.wake_int = wake_int
    def setup_int_pin_wake_up(self, rising_edge = True):
        """ allows wakeup to be made by the INT pin (PIC -RC1)
"""
        self.wake_int_pin = True
        self.wake_int_pin_rising_edge = rising_edge
```

```
        return time_s
    def setup_sleep(self, time_s):
        try:
            self.calibrate_rtc()
        except Exception:
            pass
        time_s = int((time_s * self.clk_cal_factor) + 0.5)  #
round to the nearest integer
        if time_s >= 2**(8*3):
            time_s = 2**(8*3)-1
        self._write(bytes([CMD_SETUP_SLEEP, time_s & 0xFF,
(time_s >> 8) & 0xFF, (time_s >> 16) & 0xFF]))
    def go_to_sleep(self, gps=True):
        # if we have a Pytrack then enable or disable back-up
power to the GPS receiver
        if self.board_type == self.PYTRACK and gps:
            # disable GPS only if Pytrack
            self.set_bits_in_memory(PORTC_ADDR, 1 << 7)
        else:
            # Pysense or Pyscan or no GPS
            self.mask_bits_in_memory(PORTC_ADDR, ~(1 << 7))
        # disable the ADC
        self.poke_memory(ADCON0_ADDR, 0)
        if self.wake_int:
            # Don't touch RA3, RA5 or RC1 so that interrupt wake-
up works
            self.poke_memory(ANSELA_ADDR, ~((1 << 3) | (1 << 5)))
            self.poke_memory(ANSELC_ADDR, ~((1 << 6) | (1 << 7) |
(1 << 1)))
        else:
            # disable power to the accelerometer, and don't touch
RA3 so that button wake-up works
            self.poke_memory(ANSELA_ADDR, ~(1 << 3))
            self.poke_memory(ANSELC_ADDR, ~(1 << 7))
        self.poke_memory(ANSELB_ADDR, 0xFF)
        # check if INT pin (PIC RC1), should be used for wakeup
        if self.wake_int_pin:
            if self.wake_int_pin_rising_edge:
                self.set_bits_in_memory(OPTION_REG_ADDR, 1 << 6)
# rising edge of INT pin
            else:
                self.mask_bits_in_memory(OPTION_REG_ADDR, ~(1 <<
6)) # falling edge of INT pin
            self.mask_bits_in_memory(ANSELC_ADDR, ~(1 << 1)) #
disable analog function for RC1 pin
            self.set_bits_in_memory(TRISC_ADDR, 1 << 1) # make
RC1 input pin
            self.mask_bits_in_memory(INTCON_ADDR, ~(1 << 1)) #
clear INTF
            self.set_bits_in_memory(INTCON_ADDR, 1 << 4) # enable
interrupt; set INTE)
        self._write(bytes([CMD_GO_SLEEP]), wait=False)
        # kill the run pin
        Pin('P3', mode=Pin.OUT, value=0)
    def calibrate_rtc(self):
        # the 1.024 factor is because the PIC LF operates at 31
KHz
        # WDT has a frequency divider to generate 1 ms
```

```
    def _wait(self):
        count = 0
        time.sleep_us(10)
        while self.i2c.readfrom(I2C_SLAVE_ADDR, 1)[0] != 0xFF:
            time.sleep_us(100)
            count += 1
            if (count > 500):  # timeout after 50ms
                raise Exception('Board timeout')
    def _send_cmd(self, cmd):
        self._write(bytes([cmd]))
    def read_hw_version(self):
        self._send_cmd(CMD_HW_VER)
        d = self._read(2)
        return (d[1] << 8) + d[0]
    def read_fw_version(self):
        self._send_cmd(CMD_FW_VER)
        d = self._read(2)
        return (d[1] << 8) + d[0]
    def read_product_id(self):
        self._send_cmd(CMD_PROD_ID)
        d = self._read(2)
        return (d[1] << 8) + d[0]
    def peek_memory(self, addr):
        self._write(bytes([CMD_PEEK, addr & 0xFF, (addr >> 8) &
0xFF]))
        return self._read(1)[0]
    def poke_memory(self, addr, value):
        self._write(bytes([CMD_POKE, addr & 0xFF, (addr >> 8) &
0xFF, value & 0xFF]))
    def magic_write_read(self, addr, _and=0xFF, _or=0, _xor=0):
        self._write(bytes([CMD_MAGIC, addr & 0xFF, (addr >> 8) &
0xFF, _and & 0xFF, _or & 0xFF, _xor & 0xFF]))
        return self._read(1)[0]
    def toggle_bits_in_memory(self, addr, bits):
        self.magic_write_read(addr, _xor=bits)
    def mask_bits_in_memory(self, addr, mask):
        self.magic_write_read(addr, _and=mask)
    def set_bits_in_memory(self, addr, bits):
        self.magic_write_read(addr, _or=bits)
    def get_wake_reason(self):
        """ returns the wakeup reason, a value out of constants
WAKE_REASON_* """
        return self.peek_memory(WAKE_REASON_ADDR)
    def get_sleep_remaining(self):
        """ returns the remaining time from sleep, as an
interrupt (wakeup source) might have triggered """
        c3 = self.peek_memory(WAKE_REASON_ADDR + 3)
        c2 = self.peek_memory(WAKE_REASON_ADDR + 2)
        c1 = self.peek_memory(WAKE_REASON_ADDR + 1)
        time_device_s = (c3 << 16) + (c2 << 8) + c1
        # this time is from PIC internal oscilator, so it needs
to be adjusted with the calibration value
        try:
            self.calibrate_rtc()
        except Exception:
            pass
        time_s = int((time_device_s / self.clk_cal_factor) + 0.5)
# 0.5 used for round
```

```
    _ADCON0_GO_nDONE_MASK = const(0x02)
    ADRESL_ADDR = const(0x09B)
    ADRESH_ADDR = const(0x09C)
    TRISC_ADDR = const(0x08E)
    PORTA_ADDR = const(0x00C)
    PORTC_ADDR = const(0x00E)
    WPUA_ADDR = const(0x20C)
    WAKE_REASON_ADDR = const(0x064C)
    MEMORY_BANK_ADDR = const(0x0620)
    PCON_ADDR = const(0x096)
    STATUS_ADDR = const(0x083)
    EXP_RTC_PERIOD = const(7000)
    def __init__(self, board_type, i2c=None, sda='P22',
scl='P21'):
        if i2c is not None:
            self.i2c = i2c
        else:
            self.i2c = I2C(0, mode=I2C.MASTER, pins=(sda, scl))
        if board_type not in self.BOARD_TYPE_SET:
            raise Exception('Board type not in the set
{}'.format(self.BOARD_TYPE_SET))
        self.sda = sda
        self.scl = scl
        self.board_type = board_type
        self.clk_cal_factor = 1
        self.reg = bytearray(6)
        self.wake_int = False
        self.wake_int_pin = False
        self.wake_int_pin_rising_edge = True
        # Make sure we are inserted into the
        # correct board and can talk to the PIC
        try:
            self.read_fw_version()
        except Exception as e:
            raise Exception('Board not detected: {}'.format(e))
        # init the ADC for the battery measurements
        self.poke_memory(ANSELC_ADDR, 1 << 2)
        self.poke_memory(ADCON0_ADDR, (0x06 << _ADCON0_CHS_POSN)
| _ADCON0_ADON_MASK)
        self.poke_memory(ADCON1_ADDR, (0x06 <<
_ADCON1_ADCS_POSN))
        # enable the pull-up on RA3
        self.poke_memory(WPUA_ADDR, (1 << 3))
        # make RC5 an input
        self.set_bits_in_memory(TRISC_ADDR, 1 << 5)
        # set RC6 and RC7 as outputs and enable power to the
sensors and the GPS
        self.mask_bits_in_memory(TRISC_ADDR, ~(1 << 6))
        self.mask_bits_in_memory(TRISC_ADDR, ~(1 << 7))
#        if self.read_fw_version() < 6:
#            raise ValueError('Firmware out of date')
    def _write(self, data, wait=True):
        self.i2c.writeto(I2C_SLAVE_ADDR, data)
        if wait:
            self._wait()
    def _read(self, size):
        return self.i2c.readfrom(I2C_SLAVE_ADDR, size + 1)[1:
(size + 1)]
```

```
#!/usr/bin/env python
#
# Copyright (c) 2020, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
# See https://docs.pycom.io for more information regarding
library specifics
from machine import Pin
from machine import I2C
import time
import pycom
__version__ = '0.0.2'
""" PIC MCU wakeup reason types """
WAKE_REASON_ACCELEROMETER = 1
WAKE_REASON_PUSH_BUTTON = 2
WAKE_REASON_TIMER = 4
WAKE_REASON_INT_PIN = 8
class Pycoproc:
    """ class for handling the interaction with PIC MCU """
    I2C_SLAVE_ADDR = const(8)
    PYSENSE = const(1)
    PYTRACK = const(2)
    PYSCAN = const(3)
    BOARD_TYPE_SET = (PYSENSE, PYTRACK, PYSCAN)
    CMD_PEEK = const(0x0)
    CMD_POKE = const(0x01)
    CMD_MAGIC = const(0x02)
    CMD_HW_VER = const(0x10)
    CMD_FW_VER = const(0x11)
    CMD_PROD_ID = const(0x12)
    CMD_SETUP_SLEEP = const(0x20)
    CMD_GO_SLEEP = const(0x21)
    CMD_CALIBRATE = const(0x22)
    CMD_BAUD_CHANGE = const(0x30)
    CMD_DFU = const(0x31)
    REG_CMD = const(0)
    REG_ADDRL = const(1)
    REG_ADDRH = const(2)
    REG_AND = const(3)
    REG_OR = const(4)
    REG_XOR = const(5)
    ANSELA_ADDR = const(0x18C)
    ANSELB_ADDR = const(0x18D)
    ANSELC_ADDR = const(0x18E)
    ADCON0_ADDR = const(0x9D)
    ADCON1_ADDR = const(0x9E)
    IOCAP_ADDR = const(0x391)
    IOCAN_ADDR = const(0x392)
    INTCON_ADDR = const(0x0B)
    OPTION_REG_ADDR = const(0x95)
    _ADCON0_CHS_POSN = const(0x02)
    _ADCON0_ADON_MASK = const(0x01)
    _ADCON1_ADCS_POSN = const(0x04)
```

Παράρτημα 13: Pymacr

```
{
    "address": "COM9",
    "username": "micro",
    "password": "python",
    "sync_folder": "",
    "open_on_start": true,
    "safe_boot_on_upload": false,
    "py_ignore": [
        "pymakr.conf",
        ".vscode",
        ".gitignore",
        ".git",
        "project.pymakr",
        "env",
        "venv"
    ],
    "fast_upload": false
}
```

Παράρτημα 14: Pysense

```
#!/usr/bin/env python
#
# Copyright (c) 2020, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
# See https://docs.pycom.io for more information regarding
library specifics
from pycoproc import Pycoproc
__version__ = '1.4.1'
class Pysense(Pycoproc):
    def __init__(self, i2c=None, sda='P22', scl='P21'):
        Pycoproc.__init__(self, Pycoproc.PYSENSE, i2c, sda, scl)
```

Παράρτημα 15: Reset

```
import machine
machine.reset()
```

Παράρτημα 16:

```
        """ reading the heater configuration register """
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0x11]))
        time.sleep(0.5)
        data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 1)
        return data[0]
    def read_electronic_id(self):
        """ reading electronic identifier """
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xFA]) +
bytearray([0x0F]))
        time.sleep(0.5)
        sna = self.i2c.readfrom(SI7006A20_I2C_ADDR, 4)
        time.sleep(0.1)
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xFC]) +
bytearray([0xC9]))
        time.sleep(0.5)
        snb = self.i2c.readfrom(SI7006A20_I2C_ADDR, 4)
        return [sna[0], sna[1], sna[2], sna[3], snb[0], snb[1],
snb[2], snb[3]]
    def read_firmware(self):
        """ reading firmware version """
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0x84])+
bytearray([0xB8]))
        time.sleep(0.5)
        fw = self.i2c.readfrom(SI7006A20_I2C_ADDR, 1)
        return fw[0]
    def read_reg(self, reg_addr):
        """ reading a register """
        self.i2c.writeto(SI7006A20_I2C_ADDR,
bytearray([reg_addr]))
        time.sleep(0.5)
        data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 1)
        return data[0]
    def write_reg(self, reg_addr, value):
        """ writing a register """
        self.i2c.writeto(SI7006A20_I2C_ADDR,
bytearray([reg_addr])+bytearray([value]))
        time.sleep(0.1)
    def dew_point(self):
        """ computing the dew pointe temperature (deg C) for the
current Temperature and Humidity measured pair
            at dew-point temperature the relative humidity is
100% """
        temp = self.temperature()
        humid = self.humidity()
        h = (math.log(humid, 10) - 2) / 0.4343 + (17.62 * temp) /
(243.12 + temp)
        dew_p = 243.12 * h / (17.62 - h)
        return dew_p
    def humid_ambient(self, t_ambient, dew_p = None):
        """ returns the relative humidity compensated for the
current Ambient temperature
            for ex: T-Ambient is 24.4 degC, but sensor indicates
Temperature = 31.65 degC and Humidity = 47.3%
                -> then the actual Relative Humidity is 72.2%
            this is computed because the dew-point should be the
same """
        if dew_p is None:
            dew_p = self.dew_point()
```

```
#!/usr/bin/env python
#
# Copyright (c) 2019, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more
information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
import time
from machine import I2C
import math
__version__ = '0.0.2'
class SI7006A20:
    """ class for handling the temperature sensor SI7006-A20
        +/- 1 deg C error for temperature
        +/- 5% error for relative humidity
        datasheet available at
https://www.silabs.com/documents/public/data-sheets/Si7006-
A20.pdf """
    SI7006A20_I2C_ADDR = const(0x40)
    TEMP_NOHOLDMASTER = const(0xF3)
    HUMD_NOHOLDMASTER = const(0xF5)
    def __init__(self, pysense = None, sda = 'P22', scl = 'P21'):
        if pysense is not None:
            self.i2c = pysense.i2c
        else:
            self.i2c = I2C(0, mode=I2C.MASTER, pins=(sda, scl))
    def _getWord(self, high, low):
        return ((high & 0xFF) << 8) + (low & 0xFF)
    def temperature(self):
        """ obtaining the temperature(degrees Celsius) measured
by sensor """
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xF3]))
        time.sleep(0.5)
        data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 3)
        #print("CRC Raw temp data: " + hex(data[0]*65536 +
data[1]*256 + data[2]))
        data = self._getWord(data[0], data[1])
        temp = ((175.72 * data) / 65536.0) - 46.85
        return temp
    def humidity(self):
        """ obtaining the relative humidity(%) measured by sensor
"""
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xF5]))
        time.sleep(0.5)
        data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 2)
        data = self._getWord(data[0], data[1])
        humidity = ((125.0 * data) / 65536.0) - 6.0
        return humidity
    def read_user_reg(self):
        """ reading the user configuration register """
        self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xE7]))
        time.sleep(0.5)
        data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 1)
        return data[0]
    def read_heater_reg(self):
```

```
        h = 17.62 * dew_p / (243.12 + dew_p)
        h_ambient = math.pow(10, (h - (17.62 * t_ambient) /
(243.12 + t_ambient)) * 0.4343 + 2)
        return h_ambient
```