ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαδικτυακή εφαρμογή SDR

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ηλίας Κ. Ζέρβας

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος, 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαδικτυακή εφαρμογή SDR

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ηλίας Κ. Ζέρβας

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Ιουνίου 2024.

.....	.....	.....
Ευστάθιος Συκάς Καθηγητής Ε.Μ.Π.	Νικόλαος Μήτρου Καθηγητής Ε.Μ.Π.	Ιωάννα Ρουσσάκη Αναπληρώτρια Καθηγήτρια Ε.Μ.Π

Αθήνα, Ιούνιος, 2024

.....

Ηλίας Κ. Ζέρβας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι συσκευές SDR (Software Defined Radio) είναι πομποδέκτες, που βασίζουν μέρος ή όλη τη λειτουργικότητα τους σε λογισμικό. Με αυτόν τον τρόπο διαθέτουν ευελιξία και μπορούν να μεταδώσουν ή να λάβουν σήματα πολλών διαφορετικών προτύπων. Συνδέονται με υπολογιστή, όπου τρέχει κάποιο πρόγραμμα για SDR. Η εργασία αυτή ασχολήθηκε με την επέκταση του προγράμματος ανοικτού κώδικα OpenWebRX.

Το OpenWebRX επιτρέπει τη λήψη του σήματος από τη συσκευή SDR και τον διαμοιρασμό του, μέσω Internet, σε χρήστες σε όλο τον κόσμο. Στην εργασία προστέθηκε ένας ισοσταθμιστής φορτίου μπροστά από ένα σύνολο OpenWebRX εξυπηρετητών. Έτσι επιτυγχάνεται αύξηση των ταυτόχρονα υποστηριζόμενων χρηστών. Γράφτηκε κώδικας για περιβάλλον Linux, κυρίως στη γλώσσα Python, ενώ περιλαμβάνεται και κώδικας σε C++, HTML, CSS και JavaScript.

Υλοποιήθηκε σύστημα με το νέο λογισμικό, στο εργαστήριο, το οποίο συνεργάζεται με τις συσκευές SDR που υπάρχουν εκεί. Παρέχεται πρόσβαση στα λαμβανόμενα ραδιοφωνικά σήματα FM και τα σήματα της ζώνης των αεροπλάνων (Airband). Στόχος αυτής της εργασίας είναι η αναβάθμιση των υπαρχόντων και η δημιουργία νέων διαδικτυακών δεκτών, από ραδιοερασιτέχνες και μη, που θα υποστηρίζουν μεγάλο αριθμό χρηστών.

Λέξεις κλειδιά: SDR, Web, Πομποδέκτης, Λογισμικό για SDR, OpenWebRX, Ισοσταθμιστής φορτίου, Python, RTL SDR, USRP, Ραδιοφωνική εκπομπή FM, Μπάντα αεροπλάνων

Abstract

SDR (Software Defined Radio) devices are transceivers that base part or whole of their functionality on software. This way, they are flexible and can transmit and receive using many different standards. They are connected to a PC running SDR software. Topic of this thesis is the expansion of the OpenWebRX open source program.

OpenWebRX enables the reception of the signal from the SDR device and distributes it, through Internet, to users around the world. This thesis added a load balancer in front of a pool of OpenWebRX servers. This way, it increased the number of simultaneous supported users. Code haw been written for Linux, mainly in Python language and also C++, HTML, CSS and JavaScript code is included.

A system with the new software was deployed in the lab, using the SDR devices that were available. This system provides access to the received FM Radio Broadcasting signals and Airband signals. Target of this thesis is the upgrade of existing and creation of new Internet receivers, by radio amateurs and others, that will support a large number of users.

Keywords: SDR, Web, Radio, SDR software, OpenWebRX, Load Balancing, Python, RTL SDR, USRP, FM Broadcasting, Airband.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κύριο Ευστάθιο Συκά για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Τον ευχαριστώ ακόμα για την εμπιστοσύνη που μου έδειξε και μου διέθεσε τα μηχανήματα του εργαστηρίου για να προχωρήσω τη διπλωματική. Επίσης, τον ευχαριστώ για τη καθοδήγηση που μου προσέφερε καθ' όλη τη διάρκεια της εργασίας.

Θα ήθελα να ευχαριστήσω και όλους τους καθηγητές της σχολής που μου έκαναν μάθημα και μου μετέδωσαν τις γνώσεις τους και του συμφοιτητές που κάναμε παρέα αυτά τα χρόνια.

Θα ήθελα να ευχαριστήσω και την οικογένεια μου για τη στήριξη της. Αφιερώνω αυτήν την εργασία στη μνήμη του πατέρα μου, που ήταν ηλεκτρονικός και με ενέπνευσε να ασχοληθώ με το αντικείμενο.

Περιεχόμενα

0. Εισαγωγή 12
1. Θεωρία SDR 13
1.1 Εισαγωγή στις συσκευές SDR 13
1.1.1 Ορισμός της συσκευής SDR (Software Defined Radio) 13
1.1.2 Πλεονεκτήματα, μειονεκτήματα και εφαρμογές των συσκευών SDR 14
1.1.3 Ιστορικά στοιχεία και μελλοντικές κατευθύνσεις για τις συσκευές SDR 16
1.2 Θεμελιώδεις αρχές των συσκευών SDR 22
1.2.1 Εισαγωγή 22
1.2.2 Μετατροπή σήματος μεταξύ αναλογικού και ψηφιακού 22
1.2.3 Αρχιτεκτονική πομπού και δέκτη SDR 24
1.2.4 Ψηφιακή επεξεργασία σήματος για τις συσκευές SDR 25
1.3 Υπολογιστικές πλατφόρμες για συσκευές SDR 28
1.4 Λειτουργία των συσκευών SDR 29
2. Λογισμικά για SDR 31
2.1 Λογισμικά για SDR με εφαρμογή 31
2.1.1 SDR++ 31
2.1.2 SDR Console 32
2.1.3 GNU Radio 33
2.1.4 Gqrx 35
2.1.5 HDSDR 37
2.1.6 SDRangel 38
2.2 Λογισμικά για SDR με διαδικτυακή (web) διεπαφή 41
2.2.1 OpenWebRX 41
2.2.2 Receiverbook 43
2.2.3 OpenWebRX+ 44
2.2.4 WebSDR 45
2.2.5 KiwiSDR 47
2.2.6 ShinySDR 48
3. OpenWebRX 49
3.1 Τεχνικές Λεπτομέρειες 49

3.2 Οδηγοί συσκευών στο OpenWebRX 51
3.2.1 Οδηγοί συσκευών (Drivers) 51
3.2.2 Συνδετικά προγράμματα (Connectors) 51
3.2.3 SoapySDR 52
3.2.4 Πολλαπλές συσκευές SDR 54
3.3 Ψηφιακή επεξεργασία σήματος στο OpenWebRX 55
3.3.1 Βιβλιοθήκη libcsdr 55
3.3.2 Διαχειριστής Αλυσίδας επεξεργασίας DspManager 57
3.3.3 FFT για γράφημα καταρράκτη 59
3.3.4 Αλυσίδα επεξεργασίας αναλογικού ήχου 60
3.3.5 Ψηφιακοί Αποκωδικοποιητές 64
3.4 Ο Εξυπηρετητής του OpenWebRX 65
3.4.1 Εξυπηρετητής 65
3.4.2 HTTP 65
3.4.3 WebSocket 67
3.4.4 Ρυθμίσεις 68
3.4.5 Αυθεντικοποίηση 69
3.4.6 Δεδομένα Προγράμματος 69
3.4.7 Άλλες λειτουργίες 70
3.5 Η πλευρά του πελάτη 71
3.5.1 Περιγραφή της διεπαφής χρήστη 71
3.5.2 Αρχεία Υπερκειμένου 72
3.5.3 Openwebrx.js 72
3.5.4 HTML5 72
3.5.5 Χάρτης 73
3.5.6 Ρυθμίσεις 73
4. Load Balancing OpenWebRX 74
4.1 Εισαγωγή στην ισοστάθμιση Φορτίου (Load Balancing) 74
4.1.1 Τρόπος λειτουργίας του ισοσταθμιστή φορτίου 75
4.1.2 Κατανομή φορτίου σε δίκτυα διανομής περιεχομένου 76
4.1.3 Πλεονεκτήματα και μειονεκτήματα χρήσης ισοσταθμιστή φορτίου 78
4.1.4 Αλγόριθμοι για ισοστάθμιση φορτίου 79

4.1.5 Τεχνολογία Ισοστάθμισης φορτίου 81
4.1.6 Το παράδειγμα του Scalelite 82
4.2 Load Balancing OpenWebRX 84
4.2.1 Η ιδέα 84
4.2.2 Τεχνική παρουσίαση 85
4.3 Λειτουργίες του Load Balancing OpenWebRX 87
4.3.1 Καταμερισμός εργασιών 87
4.3.2 Καταγραφή της κατάστασης (state) στους τροποποιημένους OpenWebRX εξυπηρετητές 87
4.3.3 Ενδοεπικοινωνία (Load Balancer – OpenWebRX) 89
4.3.4 Αλγόριθμος Κατανομής 90
4.3.5 Κατεύθυνση του πελάτη 92
4.3.6 Διαχείριση Σφάλματος 94
4.3.7 Διαφορές του Load Balancing OpenWebRX από το αρχικό OpenWebRX 96
4.3.8 Οι προσθήκες του Load Balancing OpenWebRX αναλυτικά 97
5. Πρακτική εφαρμογή στο εργαστήριο 102
5.1 Οι συσκευές SDR του εργαστηρίου 102
5.1.1 Ettus USRP B210 102
5.1.2 Ettus USRP N200 104
5.1.3 Ettus USRP2 107
5.1.4 RTL-SDR 108
5.2 Οι υπολογιστές του εργαστηρίου 109
5.2.1 Intel NUC mini PC 110
5.2.2 Raspberry Pi 111
5.2.3 Virtual Machine (εικονικό μηχάνημα) 112
5.3 Δίκτυο 112
5.4 Λογισμικό που χρησιμοποιήθηκε στο εργαστήριο 113
5.5 Κεραίες του εργαστηρίου 115
5.6 Ρυθμίσεις ζωνών συχνοτήτων και συσκευών στο Load Balancing OpenWebRX 115
5.7 Μετρικές λειτουργίας του Load Balancing OpenWebRX στο εργαστήριο 117
5.8 Στιγμιότυπα οθόνης 117
6. Σύνοψη και μελλοντικές επεκτάσεις 122
7. Βιβλιογραφία 126

Παράρτημα Κώδικα 131

0. Εισαγωγή

Βασικός στόχος αυτής της εργασίας είναι η ανάπτυξη ενός κατανεμητή φορτίου για συσκευές SDR. Αυτός ο κατανεμητής φορτίου συνδυάζεται με πολλούς εξυπηρετητές μιας διαδικτυακής εφαρμογής για SDR. Σε μια διαδικτυακή εφαρμογή για αυτές τις συσκευές, μπορούν να συνδεθούν χρήστες από όλο τον κόσμο και να λάβουν το σήμα της κεραίας της συσκευής. Σε ένα σύστημα μπορούν να είναι διαθέσιμες πολλές συσκευές SDR ταυτόχρονα.

Με την αύξηση της δημοφιλίας αυτών των εφαρμογών, κυρίως μεταξύ ραδιοερασιτεχνών, έχει προκύψει η ανάγκη υποστήριξης μεγαλύτερου αριθμού χρηστών. Με την εισαγωγή ενός κατανεμητή φορτίου σε ένα τέτοιο σύστημα μπορεί να αυξηθεί ο αριθμός των υποστηριζόμενων χρηστών. Αυτό επιτυγχάνεται αξιοποιώντας περισσότερες συσκευές SDR και περισσότερους υπολογιστές – εξυπηρετητές για αυτόν τον σκοπό. Επίσης, έτσι μπορεί να βελτιωθεί η συνύπαρξη πολλών χρηστών και η εμπειρία χρήσης της εφαρμογής.

Η εφαρμογή που επεκτάθηκε είναι το πρόγραμμα ανοικτού κώδικα OpenWebRX. Το OpenWebRX είναι μια διαδικτυακή εφαρμογή για SDR, προσβάσιμη μέσω περιηγητή ιστού. Στη βασική του έκδοση, περιλαμβάνει ένα μόνο εξυπηρετητή, που λαμβάνει το σήμα της SDR συσκευής και το διανέμει μέσω διαδικτύου.

Έχει γίνει και άλλη προσπάθεια επέκτασης του, το OpenWebRX+. Εκεί προστέθηκαν περισσότεροι αποκωδικοποιητές και άλλες λειτουργίες, στο μοναδικό πάλι εξυπηρετητή. Στην παρούσα εργασία προστέθηκε ένας κατανεμητής φορτίου και αξιοποιήθηκε ο βασικός εξυπηρετητής, προσθέτοντας όμως πολλούς κατάλληλα τροποποιημένους εξυπηρετητές, σε ένα ενιαίο σύστημα.

Στο 1^{ο} κεφάλαιο γίνεται η αρχική παρουσίαση των συσκευών SDR και παρατίθενται κάποια θεωρητικά στοιχεία της τεχνολογίας αυτής. Στο 2^{ο} κεφάλαιο αναλύονται διάφορα λογισμικά για SDR, είτε με μορφή ανεξάρτητης εφαρμογής, είτε με διεπαφή περιηγητή ιστού. Στο 3^{ο} κεφάλαιο παρουσιάζεται το πρόγραμμα που βασίστηκε η εργασία, το OpenWebRX και αναλύονται οι λειτουργίες του και τα διάφορα τμήματα της αρχιτεκτονικής του. Στο 4^{ο} κεφάλαιο παρουσιάζεται η επέκταση που δημιουργήθηκε από την παρούσα εργασία, το Load Balancing OpenWebRX. Αρχικά αναλύεται η έννοια της ισοστάθμισης φορτίου και στη συνέχεια αναπτύσσονται οι λειτουργίες που προστέθηκαν. Επίσης, καταγράφονται οι προσθήκες και οι τροποποιήσεις που έγιναν στον κώδικα. Στο 5^{ο} κεφάλαιο παρουσιάζεται η εφαρμογή του νέου λογισμικού στις συσκευές SDR και τους υπολογιστές του εργαστηρίου. Στο 6^{ο} κεφάλαιο βρίσκεται η σύνοψη και οι μελλοντικές επεκτάσεις. Στο 7^{ο} κεφάλαιο υπάρχει εκτενής βιβλιογραφία. Τέλος, παρατίθενται και αποσπάσματα από τον κώδικα που γράφτηκε.

1. Θεωρία SDR

1.1 Εισαγωγή στις συσκευές SDR

1.1.1 Ορισμός της συσκευής SDR (Software Defined Radio)

Ο ορισμός των συσκευών SDR όπως δίνεται στο [1] είναι: «Οι συσκευές SDR είναι ραδιοσυσκευές, στις οποίες κάποια ή όλα τα μέρη των λειτουργιών του φυσικού στρώματος, υλοποιούνται σε λογισμικό.» Ενώ σύμφωνα με το [2], μια ραδιοσυσκευή λαμβάνει και εκπέμπει σήματα, ασύρματα, σε μια περιοχή του ηλεκτρομαγνητικού φάσματος, πραγματοποιώντας μεταφορά πληροφορίας. Τα συστατικά στοιχεία των ραδιοσυσκευών όπως μίκτες, φίλτρα, (απο)διαμορφωτές, ανιχνευτές κ.α., παραδοσιακά υλοποιούνται σε υλικό. Στις συσκευές SDR υλοποιούνται σε λογισμικό, σε έναν υπολογιστή ή ένα ενσωματωμένο σύστημα.

Το [1] αναφέρει ότι οι παραδοσιακές, υλοποιημένες σε υλικό ραδιοσυσκευές, είναι εξειδικευμένες σε μια συγκεκριμένη λειτουργία. Έχουν περιορισμένη διαλειτουργικότητα με άλλα συστήματα ή πρότυπα. Η τροποποίηση τους είναι δύσκολη και απαιτεί φυσική παρέμβαση. Αντίθετα οι συσκευές SDR έχουν μεγάλη ευελιξία και μπορούν να λειτουργήσουν με πολλαπλούς τρόπους και σε πολλαπλές ζώνες συχνοτήτων. Τροποποιούνται και επεκτείνονται οι δυνατότητες τους εύκολα, απλά μέσω αναβάθμισης λογισμικού.

Ένα βασικό SDR σύστημα σύμφωνα με το [3] μπορεί να αποτελείται από έναν υπολογιστή στον οποίο τρέχει λογισμικό για SDR. Αυτός ο υπολογιστής επικοινωνεί με ένα μετατροπέα αναλογικού σήματος σε ψηφιακό (για λήψη) ή ψηφιακού σε αναλογικό (για εκπομπή). Επιπλέον υπάρχει μια εμπροσθοφυλακή ραδιοσυχνοτήτων με ενισχυτές, μίκτες και φίλτρα.

1.1.2 Πλεονεκτήματα, μειονεκτήματα και εφαρμογές των συσκευών SDR

Πλεονεκτήματα

Η υλοποίηση μέρους ή ολόκληρης της λειτουργικότητας των SDR συσκευών σε λογισμικό έχει πολλά πλεονεκτήματα. Σύμφωνα με τα [3,4] αυτά είναι:

- Διαλειτουργικότητα και ευελιξία:
  Είναι δυνατή η λήψη και η εκπομπή σε διάφορες ζώνες συχνοτήτων, με διάφορες διαμορφώσεις και διάφορους τρόπους λειτουργίας.

- Αποδοτική χρήση των πόρων σε διάφορες καταστάσεις - συνθήκες:
  Στο πλαίσιο των ευφυών ραδιοσυστημάτων (Cognitive Radio) είναι δυνατή η προσαρμογή των παραμέτρων λειτουργίας, ανάλογα με τις συνθήκες του περιβάλλοντος, ώστε να βελτιστοποιείται η επικοινωνία. Είναι για παράδειγμα δυνατό, να ανιχνευθούν και να αποφευχθούν παρεμβολές από άλλα κανάλια, επιλέγοντας με αυτόματο τρόπο την κατάλληλη ζώνη συχνοτήτων, όπου πραγματοποιείται η επικοινωνία.

- Αύξηση της διάρκειας ζωής τη συσκευής:
  Σε αντίθεση με τις παραδοσιακές συσκευές που μετά από ένα διάστημα καθίστανται απαρχαιωμένες, οι συσκευές SDR μπορούν να ανταποκρίνονται σε μελλοντικές ανάγκες και να επεκτείνουν τη λειτουργικότητα τους, απλά με μια αναβάθμιση στο λογισμικό τους.

- Λιγότερο αναλογικό υλικό:
  Αυτό συνεπάγεται μείωση του κόστους που θα απαιτούνταν και απλοποίηση της αρχιτεκτονικής.

- Καταλληλότητα για έρευνα και ανάπτυξη:

Οι συσκευές SDR προσφέρουν δυνατότητα για πειραματισμό, π.χ. για ανάπτυξη νέων πρωτοκόλλων επικοινωνίας.

Υπάρχουν και άλλα πλεονεκτήματα που αναφέρονται στο [1]. Συγκεκριμένα, ειδικά για τους κατασκευαστές των συσκευών SDR υπάρχει η δυνατότητα:

- Δημιουργία κοινής πλατφόρμας αρχιτεκτονικής και κοινής πλατφόρμας λογισμικού για μια ολόκληρη οικογένεια συσκευών – προϊόντων. Με επαναχρησιμοποίηση της υπάρχουσας εργασίας και μείωση του χρόνου ανάπτυξης των προϊόντων.

- Διόρθωση σφαλμάτων (bugs) ή επέκταση της λειτουργικότητας μέσω απομακρυσμένης αναβάθμισης, εύκολα και γρήγορα, χωρίς διακοπή της λειτουργίας της συσκευής.

Μειονεκτήματα

Από την άλλη μεριά υπάρχουν και ορισμένα μειονεκτήματα των συσκευών SDR, αυτά αναφέρονται στο [4] και είναι:

- Πιθανώς μεγαλύτερο κόστος, λόγω αυξημένης πολυπλοκότητας, αφού υλοποιούνται περισσότερες λειτουργίες.

- Αυξημένη κατανάλωση ενέργειας, λόγω πολυπλοκότητας της ψηφιακής επεξεργασίας σήματος και του πιθανώς μεγαλύτερου εύρους ζώνης λειτουργίας.

Εφαρμογές

Οι συσκευές SDR σύμφωνα με τα [1,2,4] έχουν εφαρμογή στους ακόλουθους τομείς:

- Στρατιωτικές επικοινωνίες:
  Χρησιμοποιούνται για διαλειτουργικότητα με άλλα συστήματα, ασφάλεια επικοινωνίας μέσω κρυπτογράφησης, ευελιξία και προσαρμογή σε δύσκολες συνθήκες.

- Κινητές επικοινωνίες:

  Χρησιμοποιούνται σε σταθμούς βάσεις για υποστήριξη πολλαπλών ζωνών συχνοτήτων, πολλαπλών πρωτοκόλλων και πολλαπλών προτύπων. Εφαρμόζονται και στα κινητά τηλέφωνα, με προγραμματίσιμους πυρήνες ψηφιακής επεξεργασίας. Εφαρμόζονται και σε δορυφόρους, με ψηφιακή επεξεργασία σε προγραμματίσιμες συσκευές.

- Έρευνα:

  Χρησιμοποιούνται από πανεπιστήμια και ερευνητικά τμήματα εταιριών για ανάπτυξη νέων τεχνολογιών και προϊόντων.

- Ραδιοερασιτέχνες:

  Χρησιμοποιούνται από ραδιοερασιτέχνες για τη μεταξύ τους επικοινωνία και τον πειραματισμό με νέους τρόπους και νέα πρωτόκολλα επικοινωνίας.

1.1.3 Ιστορικά στοιχεία και μελλοντικές κατευθύνσεις για τις συσκευές SDR

Όπως περιγράφεται στο [2], ο όρος Software Radio χρησιμοποιήθηκε πρώτη φορά το 1984 από την εταιρία E-Systems και αναφερόταν σε ένα δέκτη ψηφιακής βασικής ζώνης. Χρησιμοποιήθηκε ξανά το 1991 από τον Joe Mitola, σε σχέδια του για ένα σταθμό βάσης GSM. Ένα από τα πρώτα SDR συστήματα ήταν η στρατιωτική εφαρμογή SpeakEasy. Αυτή ξεκίνησε στην Αμερική τη δεκαετία του 1990 και περιλάμβανε δυο φάσεις ανάπτυξης.

Αρχικά το SpeakEasy είχε στόχο να δημιουργήσει μια συσκευή που να λειτουργεί στις συχνότητες από 2 MHz έως 2 GHz. Στόχος ήταν η διαλειτουργικότητα με άλλα υπάρχοντα στρατιωτικά συστήματα και η δυνατότητα εύκολης υποστήριξης νέων προτύπων κωδικοποίησης και διαμόρφωσης.

Τελικά, δημιουργήθηκε μια συσκευή που κάλυπτε τις συχνότητες από 4 MHz έως 400 MHz και βγήκε στην παραγωγή. Αποτελούνταν από μια επαναρυθμιζόμενη αρχιτεκτονική, με δυνατότητα ελέγχου της συχνότητας, της διαμόρφωσης, κρυπτογράφηση, υποστήριξη

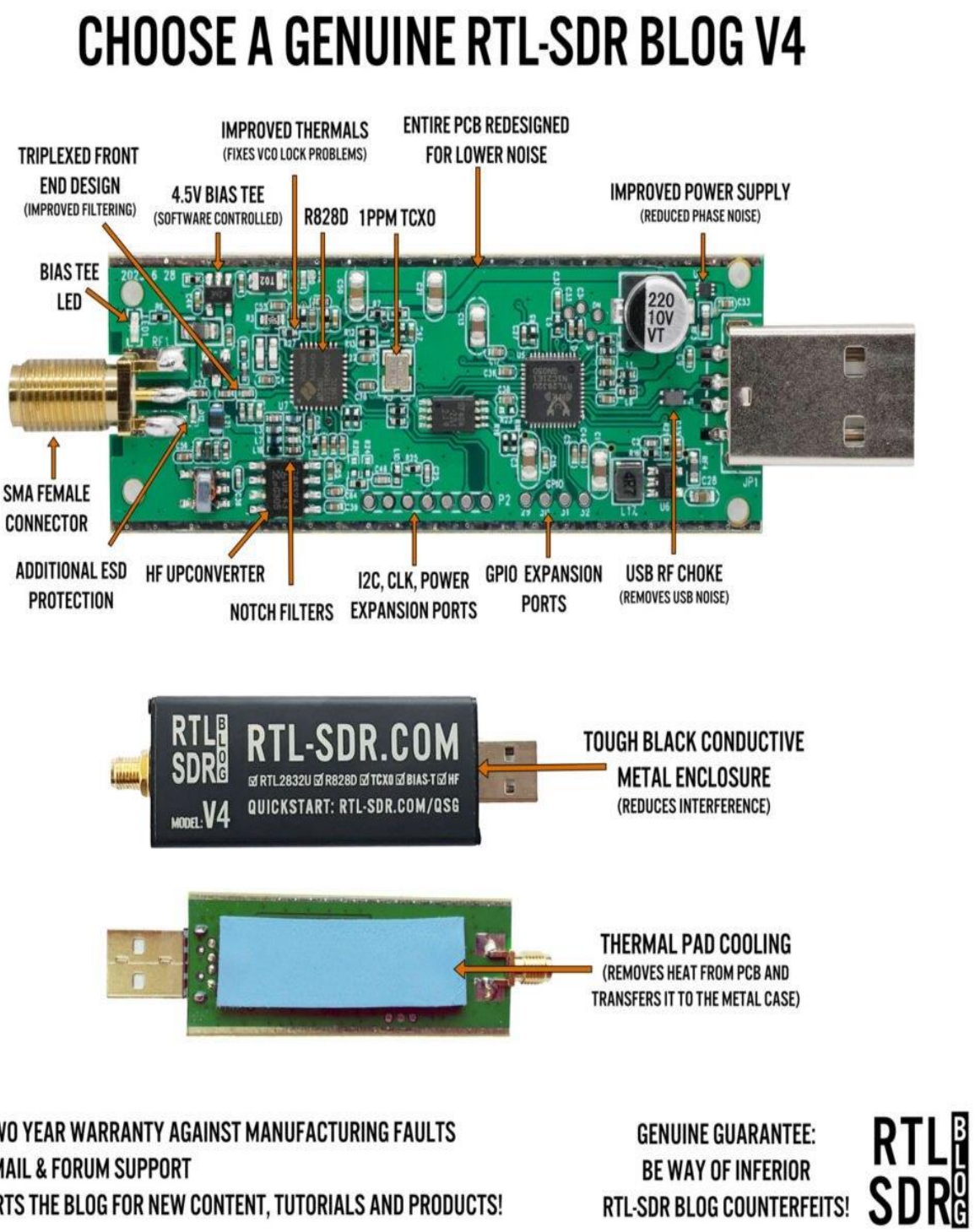
πολλών ταυτόχρονων καναλιών κ.α. Περιλάμβανε και συστοιχία επιτόπια προγραμματιζόμενων πυλών (FPGA).

Ακολούθησαν νέες συσκευές SDR και αναπτύχθηκαν προγράμματα λογισμικού για SDR. Ένα από αυτά είναι το ανοικτού κώδικα GNU Radio. Περισσότερες πληροφορίες για αυτό στο [5]. Αυτό ξεκίνησε να αναπτύσσεται το 2001 και είναι ενεργό μέχρι σήμερα. Τρέχει σε συνηθισμένους υπολογιστές, στους οποίους έχει συνδεθεί μια συσκευή SDR. Διαθέτει γραφική διεπαφή και επιτρέπει στον χρήστη να συνδέσει στοιχειώδεις μονάδες επεξεργασίας, κατασκευάζοντας μια αλυσίδα επεξεργασίας σήματος.

Αργότερα, το 2012, ανακαλύφθηκε η δυνατότητα μιας μαζικά παραγόμενης συσκευής τηλεοπτικού δέκτη, βασισμένη στο ολοκληρωμένο RTL2832U, να λειτουργήσει με τρόπο που δίνει πρόσβαση σε ακατέργαστα δεδομένα δειγμάτων (IQ samples). Έτσι αυτή η συσκευή, το RTL SDR μετατράπηκε σε ένα φτηνό SDR για το ευρύ κοινό. Υποστηρίζεται από το ανοικτού κώδικα έργο Osmocom. Περισσότερα για το RTL SDR στο [6].

Το RTL SDR υποστηρίζει εύρος ζώνης 2.4 MHz και κοστίζει λιγότερο από 50€. Σήμερα υπάρχουν συσκευές με πολύ μεγαλύτερες δυνατότητες. Υποστηρίζονται εύρος ζώνης της τάξης των GigaHertz, πολλά κανάλια λήψης και μετάδοσης και συνδεσιμότητα της τάξης των εκατοντάδων Gigabits, για μεταφορά δεδομένων από και προς τον υπολογιστή. Για παράδειγμα, το τελευταίο μοντέλο της Ettus, το USRP X440 υποστηρίζει 3.2 GHz συνολικό εύρος ζώνης, 8 κανάλια μετάδοσης και 8 κανάλια λήψης, δύο 100 Gigabit QSFP28 διεπαφές και κοστίζει περίπου 30000€. Τα χαρακτηριστικά του είναι δημοσιευμένα στο [7].

Ακολουθούν φωτογραφίες από δύο συσκευές, το RTL SDR v4 και το Ettus USRP X440.



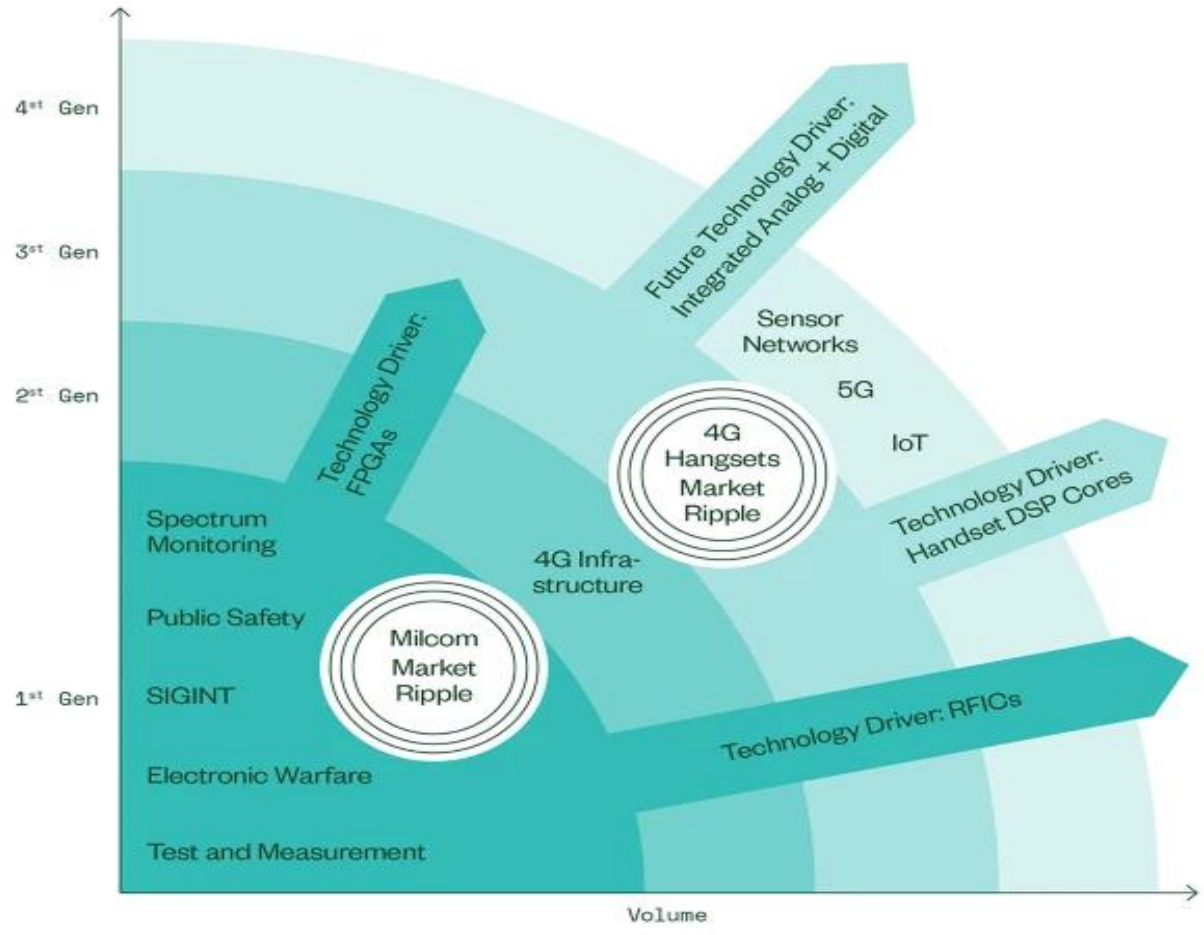
Εικόνα 1.1 *Το τελευταίο μοντέλο (v4) της οικογένειας του δημοφιλούς RTL SDR [8]*

Εικόνα 1.2 *Μπροστινή όψη του Ettus USRP X440 [7]*

Εικόνα 1.3 *Πίσω όψη του Ettus USRP X440 [7]*

Όπως περιγράφεται στο [9], κινητήριος εφαρμογή των συσκευών SDR τα πρώτα χρόνια ήταν οι στρατιωτικές επικοινωνίες. Σήμερα και στο άμεσο μέλλον αναμένεται να είναι τα δίκτυα κινητής τηλεφωνίας. Μεγάλος αριθμός συσκευών θα υλοποιούν μέρος ή ολόκληρη τη λειτουργικότητα τους σε λογισμικό.

Σύμφωνα με το [9], σημαντικές τεχνολογίες στο παρελθόν, ήταν η εισαγωγή των συστοιχιών επιτόπια προγραμματιζόμενων πυλών (FPGAs) και τα ολοκληρωμένα κυκλώματα ραδιοσυχνοτήτων (RFICs). Σήμερα σημαντικοί είναι οι επεξεργαστές ψηφιακού σήματος στα κινητά. Στο άμεσο μέλλον, αναμένεται να είναι τα ολοκληρωμένα κυκλώματα, με συνδυασμό αναλογικού και ψηφιακού υλικού, όπως για παράδειγμα FPGAs μαζί με μετατροπείς αναλογικού σήματος σε ψηφιακό και ψηφιακού σε αναλογικό (ADCs και DACs), όλα στο ίδιο ολοκληρωμένο.



Εικόνα 1.4 *Τεχνολογίες SDR [9]*

Μια πρόκληση για την τεχνολογία SDR σήμερα και στο μέλλον σύμφωνα με το [4], είναι η ικανότητα χειρισμού σημάτων διαφορετικών ζωνών συχνοτήτων, από την μονάδα ραδιοσήματος (RF). Συγκεκριμένα, αυτή τη στιγμή, σε πολλές συσκευές, ο χειρισμός πολλαπλών προτύπων και πολλαπλών ζωνών συχνοτήτων γίνεται με ξεχωριστές μονάδες. Ωστόσο είναι μελλοντικός στόχος η ενοποίηση τους σε μια προσδιοριζόμενη από λογισμικό μονάδα. Η διαθεσιμότητα μονάδων ραδιοσήματος, όπως κεραίες και ενισχυτές με μεγάλο εύρος συχνοτήτων λειτουργίας, θα καθορίσει σε σημαντικό βαθμό την ανάπτυξη και την αποδοχή της τεχνολογίας SDR.

Όπως περιγράφεται στα [1,4,10] μελλοντική εξέλιξη του SDR είναι και τα ευφυή ραδιοσυστήματα (Cognitive Radio). Σε αυτήν την τεχνολογία οι ραδιοσυσκευές ανιχνεύουν το ηλεκτρομαγνητικό περιβάλλον τους και αποκτούν γνώση του. Επιπλέον μπορούν να προσαρμόζονται σε αυτό, λαμβάνοντας αυτόματα αποφάσεις, σχετικά με τη δική τους ηλεκτρομαγνητική συμπεριφορά. Έτσι μπορούν για παράδειγμα να αποφεύγουν την ηλεκτρομαγνητική συμφόρηση και τις παρεμβολές, επικοινωνώντας σε ελεύθερες ζώνες συχνοτήτων. Αυτό μπορεί να ενισχυθεί και με μηχανική μάθηση προκειμένου να εξυπηρετούν καλύτερα τον χρήστη. Για παράδειγμα, να προσαρμόζουν το σχήμα επικοινωνίας στις τρέχουσες συνθήκες απαιτήσεων και περιβάλλοντος. Αυτό μπορεί να γίνει και σε επίπεδο δικτύου, όπου συνδεδεμένες ευφυείς συσκευές θα χρησιμοποιούν το ηλεκτρομαγνητικό φάσμα αποδοτικά.

1.2 Θεμελιώδεις αρχές των συσκευών SDR

1.2.1 Εισαγωγή

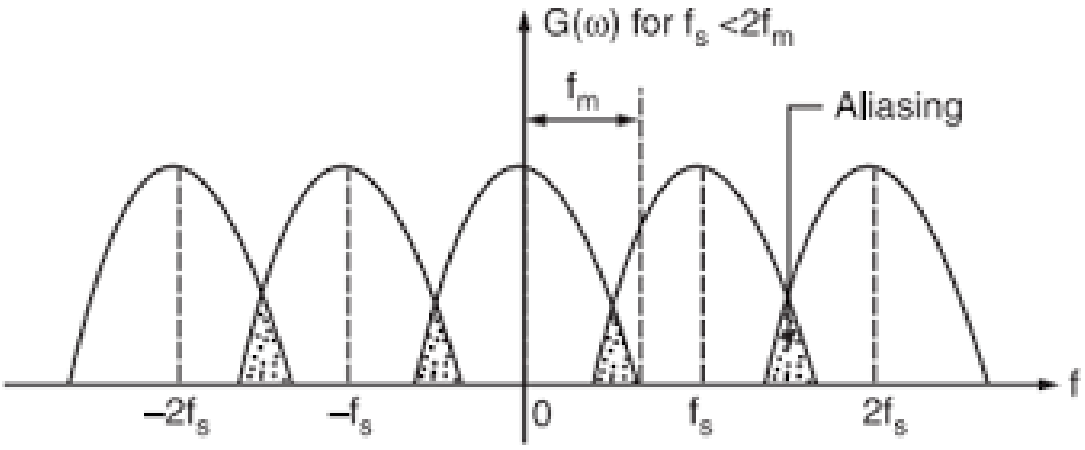
Το ραδιοσήμα που εκπέμπουν οι κεραίες είναι αναλογικό. Οι συσκευές SDR όταν λαμβάνουν, μετατρέπουν το αναλογικό σήμα σε ψηφιακό με τη μέθοδο της δειγματοληψίας (sampling) και του κβαντισμού (quantization). Στη συνέχεια, το σήμα μπορεί να σταλθεί στο FPGA και στον υπολογιστή για να πραγματοποιηθεί ψηφιακή επεξεργασία σήματος (DSP). Εκεί εκτελείται η αποδιαμόρφωση (demodulation) ή και η αποκωδικοποίηση (decoding) και το τελικό σήμα φτάνει στον χρήστη ως ήχος ή δεδομένα. Αντίστοιχα όταν γίνεται μετάδοση, το ψηφιακό σήμα δημιουργείται στον υπολογιστή και από εκεί φτάνει στη συσκευή SDR, όπου και μετατρέπεται σε αναλογικό για να εκπεμφθεί.

1.2.2 Μετατροπή σήματος μεταξύ αναλογικού και ψηφιακού

Όπως περιγράφεται στο [11], η μετατροπή του σήματος από αναλογικό σε ψηφιακό γίνεται στο κύκλωμα μετατροπέα αναλογικού σήματος σε ψηφιακό (ADC) και η μετατροπή από ψηφιακό σε αναλογικό στο μετατροπέα DAC. Αυτοί οι μετατροπείς βρίσκονται στη συσκευή SDR. Υποστηρίζουν μετατροπή με ταχύτητα έως ένα όριο ρυθμού δειγμάτων το δευτερόλεπτο. Το όριο αυτό είναι μεγαλύτερο ή ίσο από το εύρος ζώνης που υποστηρίζει συνολικά η συσκευή SDR. Αυτό συμβαίνει, γιατί το σύστημα συνολικά, μπορεί να περιορίζεται και από άλλους παράγοντες.

Στο μετατροπέα αναλογικού σήματος σε ψηφιακό (ADC) πραγματοποιείται δειγματοληψία στο πεδίο του χρόνου. Δηλαδή ένα συνεχές χρονικά σήμα μετατρέπεται σε διακριτού χρόνου, λαμβάνοντας δείγματα σε ισαπέχουσες χρονικές στιγμές. Στη συνέχεια πραγματοποιείται και κβαντισμός στο πεδίο του πλάτους. Δηλαδή, το πλάτος του σήματος από συνεχές τιμές, μετατρέπεται σε σήμα που λαμβάνει διακριτές τιμές - στάθμες. Βασικές παράμετροι αυτής της διαδικασίας είναι ο ρυθμός δειγματοληψίας (sampling rate) και ο αριθμός των ψηφίων των δειγμάτων (number of bits).

Για τη δειγματοληψία ισχύει ένα βασικό θεώρημα, το οποίο λέει πως ο ρυθμός με τον οποίο γίνεται η δειγματοληψία πρέπει να είναι μεγαλύτερος ή ίσος από το διπλάσιο της μέγιστης συχνότητας του βασικού σήματος. Διαφορετικά, εμφανίζεται το φαινόμενο της αναδίπλωσης (aliasing) και το δειγματοληπτημένο σήμα δεν αντιστοιχεί σωστά με αυτό της εισόδου. Αυτό συμβαίνει επειδή φασματικές συνιστώσες από είδωλα του σήματος εισχωρούν στο φάσμα του βασικού σήματος και το αλλοιώνουν. Ο ρυθμός που είναι ακριβώς διπλάσιος της μέγιστης συχνότητας του βασικού σήματος ονομάζεται ρυθμός Nyquist.



Εικόνα 1.5 *Φαινόμενο αναδίπλωσης [12]*

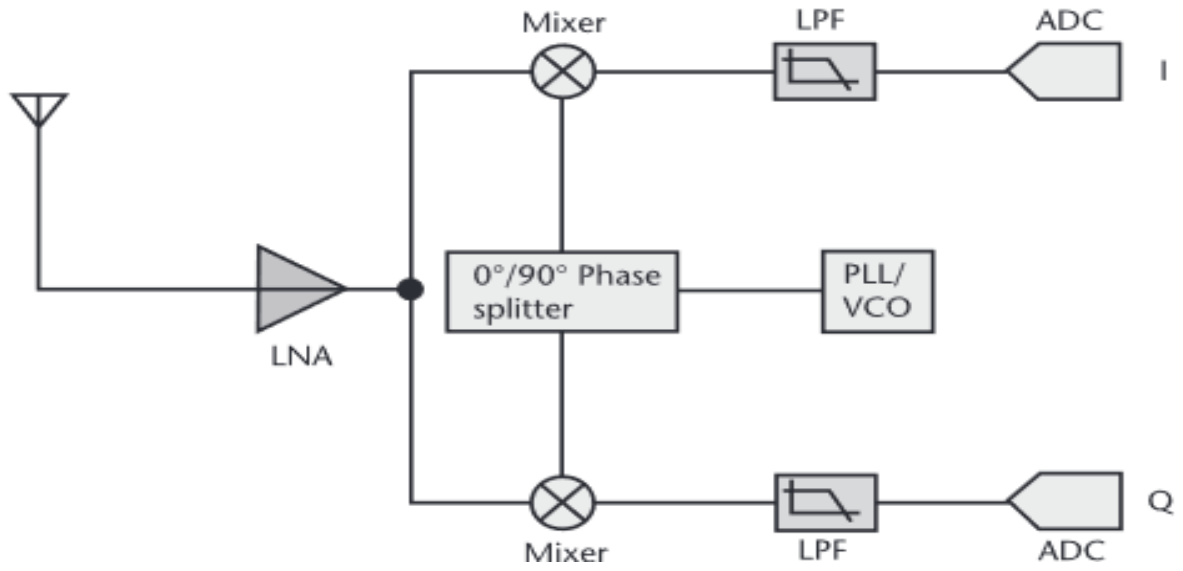
Στον μετατροπέα ψηφιακού σήματος σε αναλογικό (DAC) πραγματοποιείται ανακατασκευή του σήματος, αποκτώντας αναλογική μορφή. Αυτό γίνεται ιδανικά με βαθυπερατό φιλτράρισμα ή πρακτικά με παρεμβολή παλμών. Η αύξηση του ρυθμού δειγματοληψίας με τη μέθοδο της παρεμβολής δειγμάτων είναι πολλές φορές χρήσιμη, προκειμένου να απομακρυνθούν μεταξύ τους τα φασματικά είδωλα και να διευκολυνθεί ο σχεδιασμός του φίλτρου ανακατασκευής.

1.2.3 Αρχιτεκτονική πομπού και δέκτη SDR

Όπως περιγράφεται στα [13,14] η παραδοσιακή αρχιτεκτονική στις ραδιοσυσκευές, είναι η υπερετερόδυνη. Σε αυτή την αρχιτεκτονική πραγματοποιείται μετατροπή του σήματος σε μια ενδιάμεση συχνότητα (IF). Η αρχική ιδέα του υπερετερόδυνου δέκτη, ήταν ο συνδυασμός μιας τοπικής συχνότητας, με τη φέρουσα του σήματος, ώστε να παραχθεί ένα σήμα χαμηλότερης συχνότητας, με πιο εύκολη αποδιαμόρφωση. Επίσης, μπορεί να υλοποιηθεί ευκολότερα ένας ενισχυτής σε μια χαμηλή ζώνη συχνοτήτων, παρά σε μια υψηλή.

Έτσι στην περίπτωση του δέκτη, το σήμα κατεβαίνει μέσω μίξης, από μια υψηλή ζώνη συχνοτήτων σε μια ενδιάμεση ζώνη, όπου και θα ενισχυθεί και θα φιλτραριστεί. Στη συνέχεια θα αποδιαμορφωθεί ως αναλογικό ή θα μετατραπεί σε ψηφιακό για να αποδιαμορφωθεί ή και να αποκωδικοποιηθεί στο ψηφιακό πεδίο. Η επιλογή του σήματος επιτυγχάνεται μέσω ρύθμισης των φίλτρων ενδιάμεσης συχνότητας. Στην αρχιτεκτονική αυτή, είναι απαραίτητος και τουλάχιστον ένας τοπικός ταλαντωτής.

Όπως περιγράφεται στο [15] στις συσκευές SDR χρησιμοποιείται όμως μια άλλη αρχιτεκτονική, αυτή της μηδενικής ενδιάμεσης συχνότητας (zero-IF). Σε αυτή την αρχιτεκτονική έχουμε μόνο μια φάση μίξης, με ορισμό της συχνότητας τοπικού ταλαντωτή απευθείας στην επιθυμητή ζώνη συχνοτήτων. Έτσι γίνεται μετάφραση του σήματος κατευθείαν στη βασική ζώνη, σε ένα ορθογώνιο μιγαδικό IQ σήμα.



Εικόνα 1.6 *Αρχιτεκτονική μηδενικής ενδιάμεσης συχνότητας [15]*

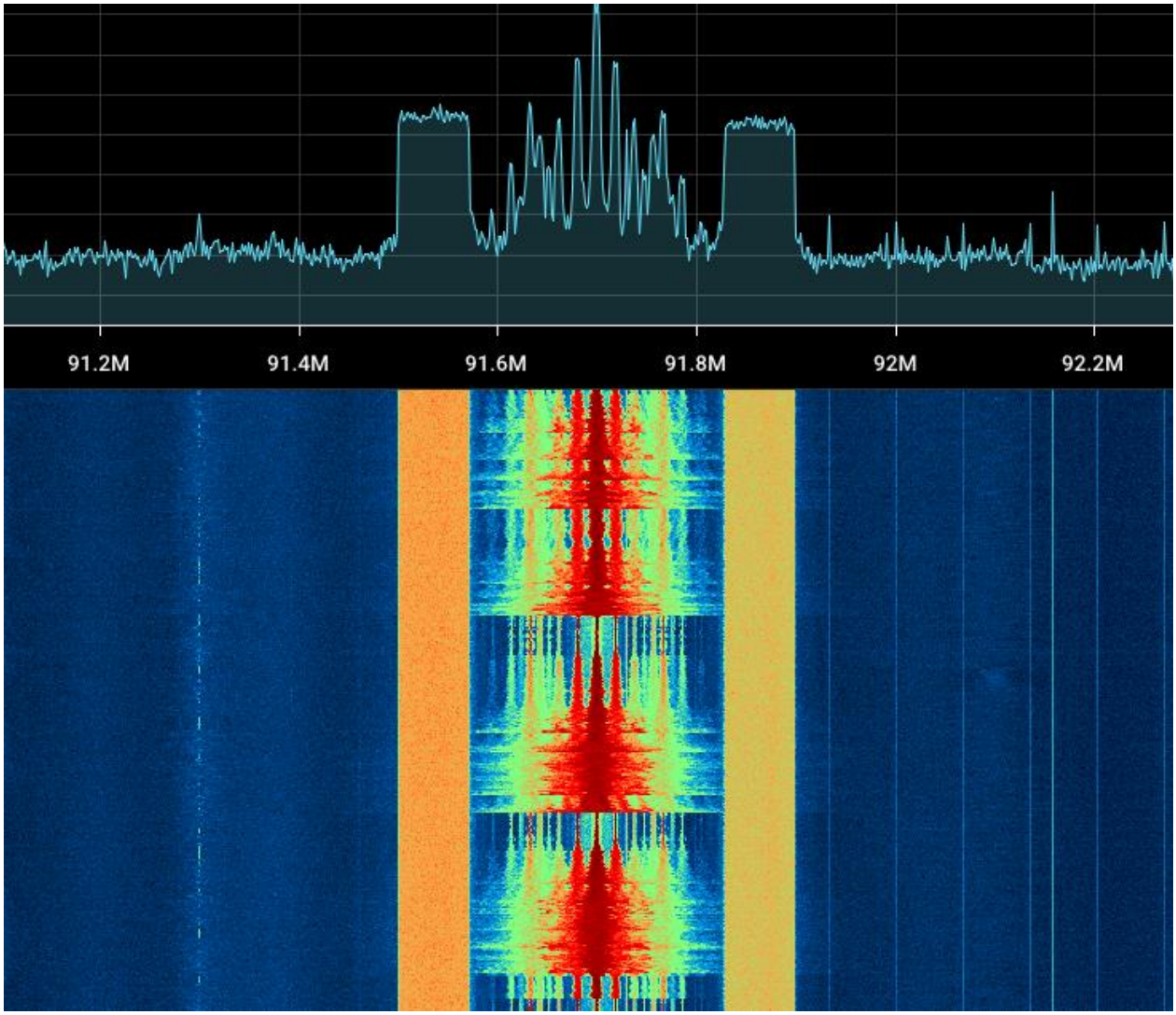
Σύμφωνα με το [15] η απουσία ενδιάμεσης συχνότητας, μειώνει τον απαιτούμενο αριθμό αναλογικών φίλτρων και ο σχεδιασμός τους διευκολύνεται, αφού αυτά υλοποιούνται στη βασική ζώνη. Επίσης, οι ADC και DAC λειτουργούν στη βασική ζώνη, σε σχετικά χαμηλότερο ρυθμό δειγματοληψίας. Με αυτήν την αρχιτεκτονική επιτυγχάνεται μείωση του μεγέθους, του βάρους και της απαιτούμενης ισχύος, αφού απλοποιείται η αναλογική εμπροσθοφυλακή ραδιοσήματος και χρησιμοποιούνται λιγότερα εξαρτήματα. Προβλήματα όπως η διαρροή φέροντος και η εμφάνιση ανισορροπίας των IQ δειγμάτων, σήμερα διορθώνονται. Για αυτούς τους λόγους προτιμάται η συγκεκριμένη αρχιτεκτονική στις συσκευές SDR.

1.2.4 Ψηφιακή επεξεργασία σήματος για τις συσκευές SDR

Το σήμα που λαμβάνουν οι συσκευές SDR κατευθύνεται σε υπολογιστή, όπου επεξεργάζεται ψηφιακά (DSP). Εκεί υπολογίζεται συνήθως ο Διακριτός Μετασχηματισμός Fourier (DFT), γίνεται το φιλτράρισμα για επιλογή του επιθυμητού σήματος και αυτό αποδιαμορφώνεται ή και αποκωδικοποιείται. Μπορεί να εφαρμόζονται και διάφορες άλλες λειτουργίες ψηφιακής επεξεργασίας, για βελτίωση της λήψης. Αντίστοιχα, όταν οι SDR συσκευές μεταδίδουν, πραγματοποιείται η διαμόρφωση ή και η κωδικοποίηση στον υπολογιστή, πριν το σήμα εκπεμφθεί.

Υπολογίζεται ο Διακριτός Μετασχηματισμός Fourier (DFT), με τη βοήθεια του αλγορίθμου FFT. Έτσι υπολογίζονται οι φασματικές συνιστώσες του σήματος. Τα δεδομένα αυτά συνήθως εμφανίζονται στον χρήστη με τη μορφή του γραφήματος καταρράκτη (waterfall). Το γράφημα καταρράκτη του FFT, όπως αναφέρεται στο [16] εμφανίζει την κατανομή της ισχύος του σήματος, στις διάφορες συχνότητες, συναρτήσει του χρόνου.

Ακολουθεί εικόνα με ένα ενδεικτικό γράφημα καταρράκτη σήματος ραδιοφωνικής εκπομπής FM.



Εικόνα 1.7 *Γράφημα καταρράκτη [16]*

Στη συνέχεια, πραγματοποιείται το φιλτράρισμα για επιλογή του επιθυμητού σήματος. Ανάμεσα σε όλα τα σήματα που λαμβάνονται εντός του εύρους ζώνης γύρω από την κεντρική συχνότητα, επιλέγεται αυτό που μας ενδιαφέρει. Συνήθως υπάρχει επιλογή ρύθμισης του εύρους του φίλτρου, για καλύτερη λήψη. Το φιλτράρισμα γίνεται συνήθως με ψηφιακά FIR φίλτρα, υλοποιημένα σε λογισμικό.

Μετά, το επιλεγμένο σήμα αποδιαμορφώνεται ή και αποκωδικοποιείται από λογισμικό. Επειδή η αποδιαμόρφωση στις συσκευές SDR γίνεται σε λογισμικό, υποστηρίζονται πολλές αναλογικές και ψηφιακές αποδιαμορφώσεις, αρκεί να είναι ενταγμένες στο πρόγραμμα που χρησιμοποιείται. Υποστηρίζονται και αποκωδικοποιήσεις

που εκτελούνται συνήθως από εξειδικευμένα προγράμματα. Το αποτέλεσμα αυτής της διαδικασίας φτάνει στον χρήστη ως ήχος, στην περίπτωση της φωνής ή ως δεδομένα.

Κατά τη λήψη, μπορεί να έχουν ενεργοποιηθεί και λειτουργίες ψηφιακής επεξεργασίας, όπως αυτόματος έλεγχος κέρδους (AGC) και σιγασμός (squelch). Στην πρώτη περίπτωση όπως αναφέρεται στο [17], ρυθμίζεται αυτόματα το κέρδος της ενίσχυσης προκειμένου να διατηρείται ένα επαρκές πλάτος εξόδου. Ενώ στην περίπτωση του σιγασμού όπως αναφέρεται στο [18], εξασφαλίζεται η ομαλή ακρόαση του σήματος, επιβάλλοντας σιγή κατά τα χρονικά διαστήματα, όταν το κανάλι είναι ανενεργό και επικρατεί μόνο θόρυβος.

Αντίστοιχα, στην περίπτωση της μετάδοσης, ο χρήστης επιλέγει στον υπολογιστή του, την πηγή της πληροφορίας, π.χ. μικρόφωνο ή κάποιο αρχείο ή έξοδο κάποιου προγράμματος. Στη συνέχεια, επιλέγει τη διαμόρφωση που θέλει και τη συσκευή SDR που θα χρησιμοποιηθεί. Κατόπιν ρυθμίζει επιλογές όπως η συχνότητα, το εύρος ζώνης και την ισχύ εκπομπής και στέλνει το σήμα στη συσκευή SDR, για να εκπεμφθεί. Όλα αυτά μπορούν να υλοποιηθούν σε κάποιο λογισμικό για SDR.

1.3 Υπολογιστικές πλατφόρμες για συσκευές SDR

Υπάρχει μια ποικιλία υπολογιστικών πλατφόρμων για αξιοποίηση τους από τις συσκευές SDR. Αυτές περιγράφονται στα [15,19]. Αρχικά, είναι οι γενικού σκοπού επεξεργαστές (GPP). Αυτοί εμφανίζουν μεγάλη ευελιξία, είναι εύκολα επαναπρογραμματίσιμοι και η ανάπτυξη λογισμικού σε αυτούς είναι πιο απλή σε σχέση με τις άλλες πλατφόρμες. Ωστόσο, δεν είναι αποδοτικοί ενεργειακά. Τα τελευταία χρόνια σε αυτή την κατηγορία έχουν αναπτυχθεί και οι επεξεργαστές ARM, που λόγω του χαμηλού κόστους, του μικρού μεγέθους και της χαμηλότερης κατανάλωσης ενέργειας, σε συνδυασμό με τις υπολογιστικές τους ικανότητες προτιμώνται σε συσκευές SDR. Οι γενικού σκοπού επεξεργαστές είναι κατάλληλοι για υλοποίηση των πρωτοκόλλων και της δικτυακής στοίβας. Επίσης σε γενικού σκοπού επεξεργαστή τρέχει συνήθως το λογισμικό του συνδεδεμένου υπολογιστή.

Άλλη επιλογή είναι οι επεξεργαστές ψηφιακού σήματος (DSP). Αυτοί βρίσκονται σήμερα στα κινητά τηλέφωνα. Είναι εξειδικευμένοι στους μαθηματικούς υπολογισμούς, ωστόσο ίσως είναι αργοί για κάποιες εφαρμογές.

Ακολουθούν οι συστοιχίες επιτόπια προγραμματιζόμενων πυλών (FPGA). Αυτές είναι ολοκληρωμένα που προγραμματίζονται από τον χρήστη μετά την κατασκευή τους. Είναι συνήθως ενσωματωμένες στη συσκευή SDR. Διαθέτουν χώρο, με λογικές μονάδες, γενικές ή εξειδικευμένες για ψηφιακή επεξεργασία (dedicated DSP slices). Προγραμματίζονται διαφορετικά από τους επεξεργαστές, σε γλώσσα περιγραφής υλικού. Δεν είναι κατάλληλες για πολύπλοκο έλεγχο ροής προγραμμάτων, αλλά χρησιμεύουν σε δύσκολους υπολογισμούς. Είναι λιγότερο γρήγορες και αποδοτικές από τα ολοκληρωμένα ειδικού σκοπού (ASIC).

Τέλος, είναι οι κάρτες γραφικών (GPU). Αυτές μπορεί να είναι διαθέσιμες στον συνδεδεμένο υπολογιστή, όπου γίνεται η επεξεργασία του σήματος. Αξιοποιούνται από τα προγράμματα για SDR, λόγω της μεγάλης υπολογιστικής ικανότητας και ταχύτητας τους. Χρησιμεύουν για παράδειγμα στην επεξεργασία σημάτων μεγάλου εύρους, τα οποία έρχονται από τη συσκευή SDR.

1.4 Λειτουργία των συσκευών SDR

Κάποιες συσκευές SDR υποστηρίζουν μόνο λήψη (RX), ενώ άλλες υποστηρίζουν και λήψη και μετάδοση (TX). Για παράδειγμα η δημοφιλής χαμηλού κόστους συσκευή RTL SDR μπορεί μόνο να λάβει. Ενώ τα Ettus USRP, που χρησιμοποιήθηκαν στο εργαστήριο, μπορούν και να λάβουν και να μεταδώσουν. Μια λίστα με πολλές συσκευές SDR και αρκετά χαρακτηριστικά τους μπορεί να βρεθεί στο: https://en.wikipedia.org/wiki/List_of_software-defined_radios [20]

Για τη λειτουργία των συσκευών SDR είναι απαραίτητος και επιπλέον εξοπλισμός. Απαιτούνται κεραίες για λήψη και μετάδοση, ενισχυτής ισχύος για μετάδοση με ισχύ υψηλότερη από αυτήν του ενσωματωμένου και υπολογιστής για τον χειρισμό της συσκευής και τη διεπαφή χρήστη.

Μπορούν να χρησιμοποιηθούν κεραίες με διαφορετικά χαρακτηριστικά, ανάλογα με την περιοχή συχνοτήτων που λειτουργεί η εφαρμογή. Ενδεικτικά αναφέρονται οι κεραίες από μακρύ σύρμα για τις χαμηλές συχνότητες μέχρι και την HF, κάθετες κεραίες για VHF και UHF, κεραίες μεγάλου εύρους συχνοτήτων όπως η Discone για VHF και UHF και παραβολικό κάτοπτρο για επικοινωνία με δορυφόρους σε UHF και ανώτερες συχνότητες. Δείτε περισσότερες λεπτομέρειες στο [21].

Το εύρος συχνοτήτων λειτουργίας των κεραιών βρίσκεται συνήθως σε μια μικρή σχετικά περιοχή, κάτι που αποτελεί σημαντικό περιοριστικό παράγοντα για τη λειτουργία της συσκευής SDR. Δηλαδή, η συσκευή μπορεί να διαχειριστεί σήματα διαφορετικών συχνοτήτων, όμως αυτά δεν μπορούν να ληφθούν εξαιτίας της κεραίας. Παρόλα αυτά, τα τελευταία χρόνια έχουν πραγματοποιηθεί σημαντικές εξελίξεις και υπάρχουν πλέον επαναρυθμιζόμενες κεραίες, που μπορούν να αλλάξουν τη συχνότητα και άλλες παραμέτρους λειτουργίας τους. Δείτε περισσότερες λεπτομέρειες στο [22].

Οι συσκευές SDR διαθέτουν ενσωματωμένο ενισχυτή, ωστόσο αυτός συχνά δεν είναι αρκετός. Για παράδειγμα από τις διαθέσιμες εμπορικά φορητές συσκευές με χαμηλό ή μέσο κόστος οι περισσότερες δεν ξεπερνάνε τα 10 dBm σε ισχύ εκπομπής, με λίγες εξαιρέσεις όπως το Ettus USRP B210 σύμφωνα με το [23]. Για αυτόν τον λόγο, σε πρακτικές εφαρμογές που απαιτούν μετάδοση, χρειάζεται και ένας ενισχυτής ισχύος που να λειτουργεί στην επιθυμητή ζώνη συχνοτήτων. Το εύρος της περιοχής λειτουργίας του ενισχυτή ισχύος,

αποτελεί και αυτό ένα περιοριστικό παράγοντα, για τη λειτουργία της συσκευής SDR σε πολλαπλές ζώνες συχνοτήτων.

Επιπλέον χρειάζεται και ένας υπολογιστής. Απαιτείται να τρέχει στον υπολογιστή κάποιο λογισμικό για SDR. Μέσω αυτού μπορεί ο χρήστης να χειριστεί τη συσκευή SDR. Επίσης, μπορεί να ακούσει τον ήχο ή να δει τα δεδομένα του σήματος στην περίπτωση της λήψης ή να εισάγει τον ήχο ή τα δεδομένα που θα εκπεμφθούν, στην περίπτωση της μετάδοσης. Επιπρόσθετα, συνήθως στον υπολογιστή γίνεται και η ψηφιακή επεξεργασία του σήματος που έρχεται από τη συσκευή SDR. Ωστόσο υπάρχουν και εξαιρέσεις, όπως το KiwiSDR, το οποίο σύμφωνα με το [24] είναι αυτόνομο και όλη η επεξεργασία γίνεται στο σύστημα της συσκευής SDR.

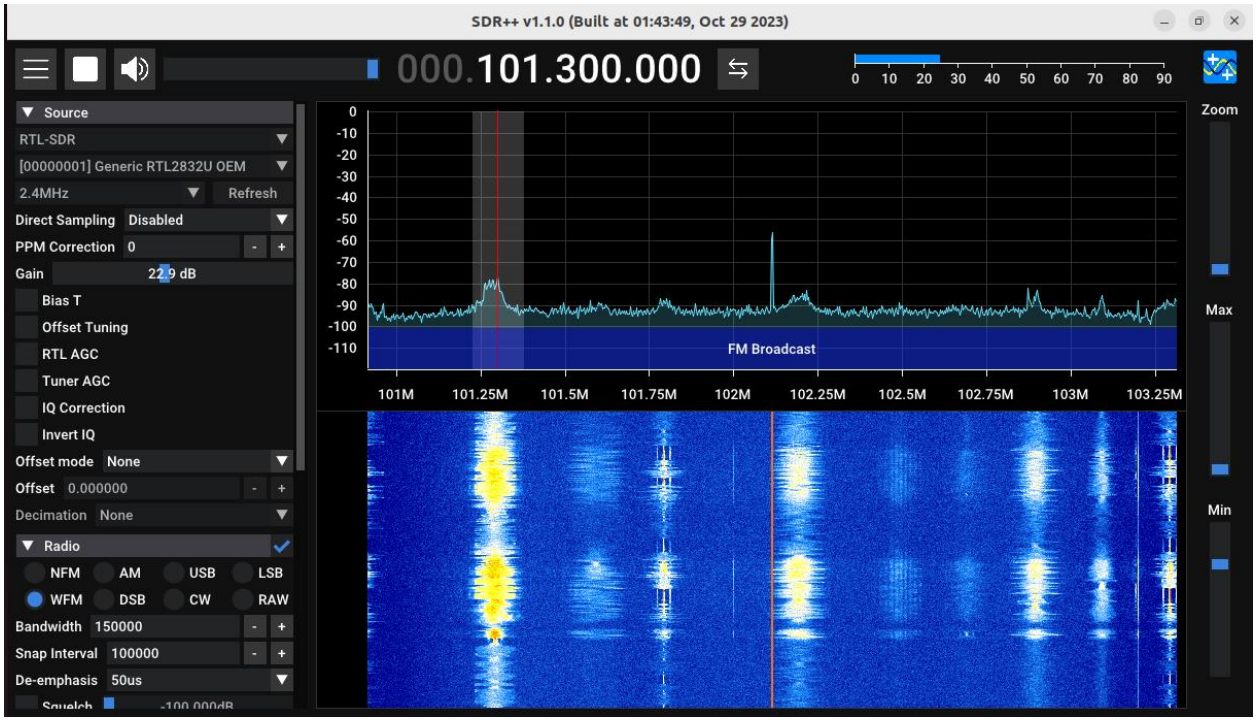
Τα προγράμματα για SDR, που μπορούν να τρέχουν στον συνδεδεμένο με τη συσκευή SDR υπολογιστή, αποτελούν το αντικείμενο της επόμενης ενότητας.

2. Λογισμικά για SDR

2.1 Λογισμικά για SDR με εφαρμογή

2.1.1 SDR++

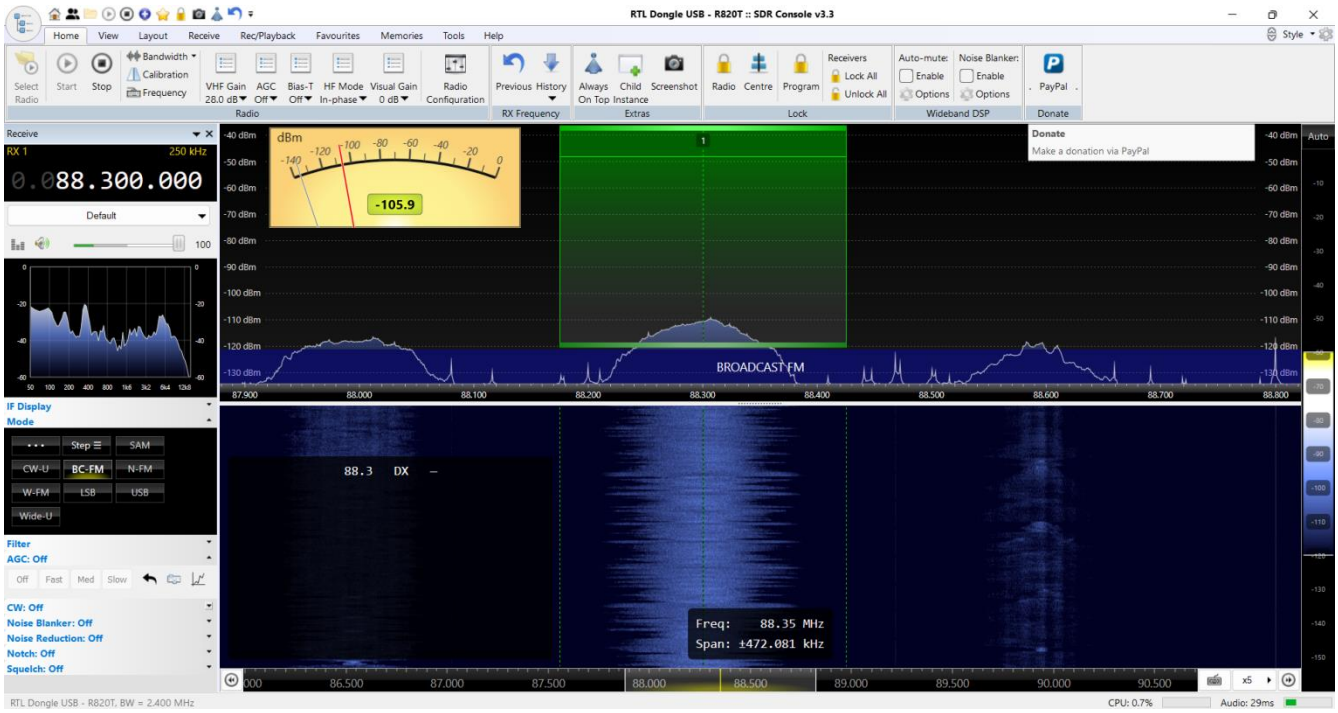
Το SDR++, όπως περιγράφεται στο [25] είναι ένα ανοικτού κώδικα πρόγραμμα για SDR. Τρέχει σε πολλές πλατφόρμες: Windows, Linux, Mac, BSD και Android. Δημιουργός του είναι ο Alexandre Rouma. Είναι εφαρμογή γραμμένη σε C++. Είναι πρόγραμμα εύκολο στον χειρισμό και διαθέτει εκτενή οδηγό χρήσης (manual), ο οποίος βρίσκεται στο [26]. Περιλαμβάνει αναλογικές αποδιαμορφώσεις και επιτρέπει πολλαπλούς ταυτόχρονους αποδιαμορφωτές (VFOs). Επιταχύνει την ψηφιακή επεξεργασία σήματος (DSP) μέσω SIMD, δηλαδή εντολές που εκτελούν τον ίδιο υπολογισμό σε πολλά δεδομένα ταυτόχρονα, πετυχαίνοντας παραλληλία. Υπάρχει η δυνατότητα καταγραφής ήχου και ακατέργαστων δειγμάτων. Υπάρχουν και ρυθμίσεις αυτόματου ελέγχου κέρδους και σιγασμού.



Εικόνα 2.1 *Στιγμιότυπο οθόνης από το SDR++*

Υποστηρίζεται εξ αποστάσεως χειρισμός και λήψη με τον SDR++ server, από εφαρμογή εγκατεστημένη και στην πλευρά του πελάτη. Το SDR++ χρησιμοποιεί το SoapySDR για υποστήριξη των οδηγών των συσκευών (drivers). Υποστηρίζει πολλές συσκευές SDR, μεταξύ των οποίων τα Airspy, HackRF, PlutoSDR, RTL SDR και άλλα. Τα Ettus USRP, βρίσκονται ακόμα σε δοκιμαστική φάση υποστήριξης.

2.1.2 SDR Console

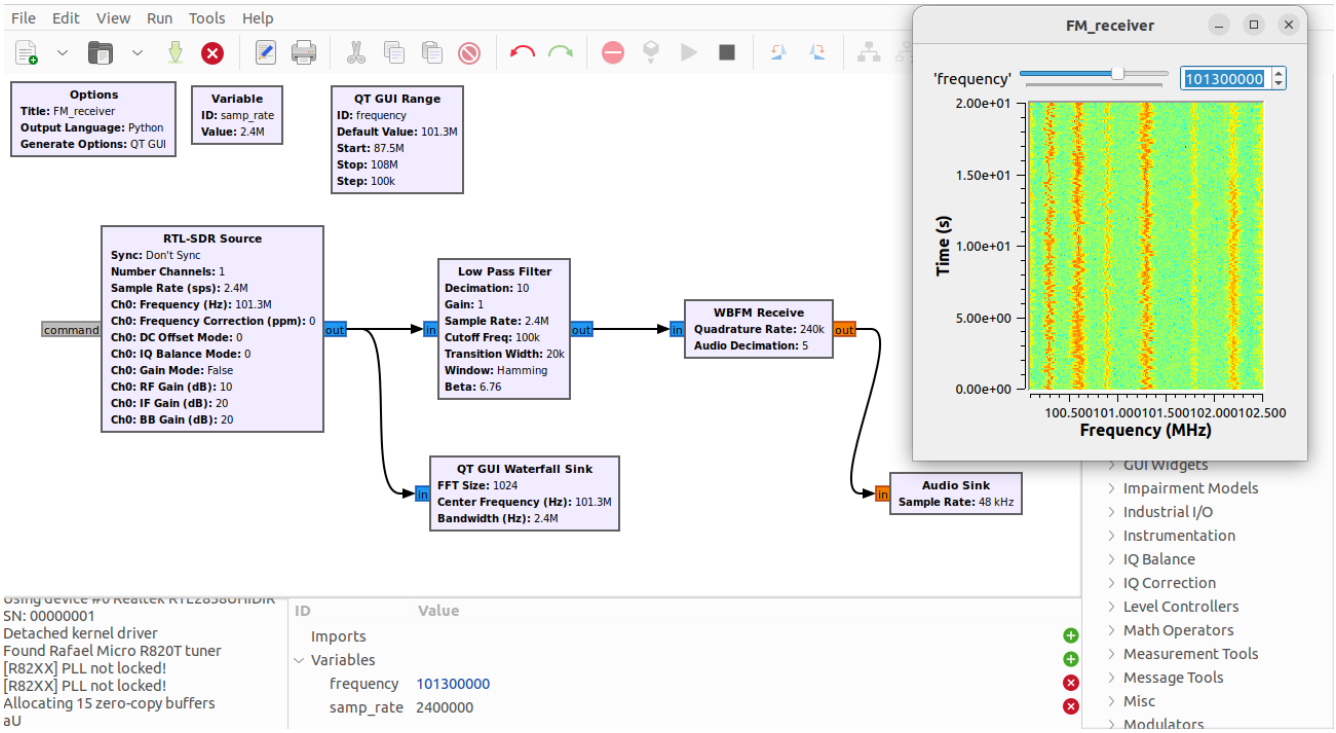


Εικόνα 2.2 *Στιγμιότυπο οθόνης από το SDR Console.*

Το SDR Console, όπως περιγράφεται στο [27] είναι ένα δωρεάν πρόγραμμα SDR για Windows. Καλύπτει τις εκδόσεις Windows 7 μέχρι Windows 11. Είναι κλειστού κώδικα. Έχει γραφτεί από τον Simon Brown (Αγγλία). Γλώσσες συγγραφής του είναι κυρίως η Microsoft Visual C++ και λιγότερο η C και η Assembly. Το πρόγραμμα υποστηρίζει έως 24 ταυτόχρονους δέκτες (VFOs). Υποστηρίζονται αναλογικές διαμορφώσεις όπως AM, SSB, CW, FM, στερεοφωνικό FM. Η λειτουργία μετάδοσης (TX) βρίσκεται ακόμα σε αρχικό στάδιο, υποστηρίζοντας λίγες μόνο συσκευές SDR. Υπάρχει η δυνατότητα καταγραφής ήχου,

IQ δειγμάτων και video, όπως επίσης και η αναπαραγωγή των IQ δειγμάτων από αρχείο εισόδου. Επιπλέον, το γράφημα καταρράκτη έχει δυνατότητα ρύθμισης των χαρακτηριστικών του. Διατηρείται και ιστορικό της ισχύος του σήματος. Μπορούν ακόμα να αποθηκευτούν οι αγαπημένες ρυθμίσεις και συχνότητες. Για καλύτερη απόδοση υποστηρίζεται επεξεργασία και σε κάρτα γραφικών. Υποστηρίζονται πολλές συσκευές SDR, μεταξύ των οποίων τα Airspy, bladeRF, Ettus USRP, FUNcube, HackRF, LimeSDR, PlutoSDR, RTL-SDR, SDRplay. Τέλος, υπάρχει σχεδίαση εξυπηρετητή πελάτη, επιτρέποντας είτε απομακρυσμένη σύνδεση σε άλλο υπολογιστή που τρέχει το ίδιο πρόγραμμα (με λήψη της εξόδου του), είτε δημιουργία δημόσιου εξυπηρετητή, για σύνδεση με κωδικό.

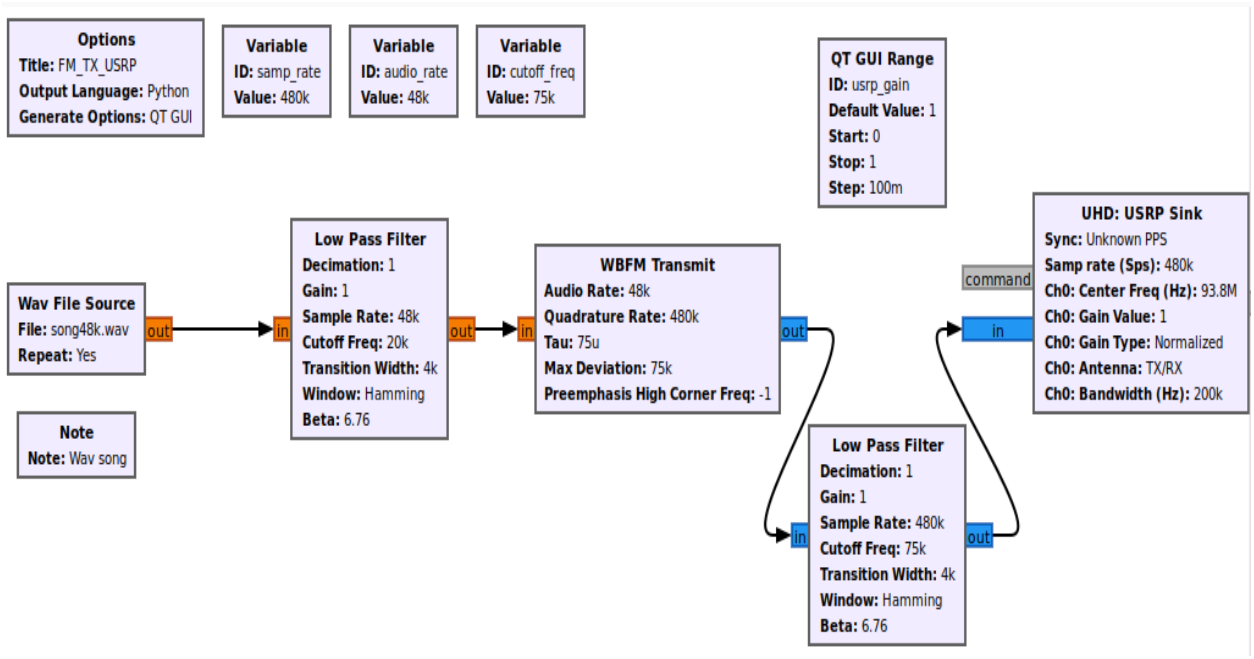
2.1.3 GNU Radio



Εικόνα 2.3 *Στιγμιότυπο οθόνης από το GNU Radio (Φαίνεται ένας αποδιαμορφωτής FM εν λειτουργία και το σχετικό γράφημα καταρράκτη.)*

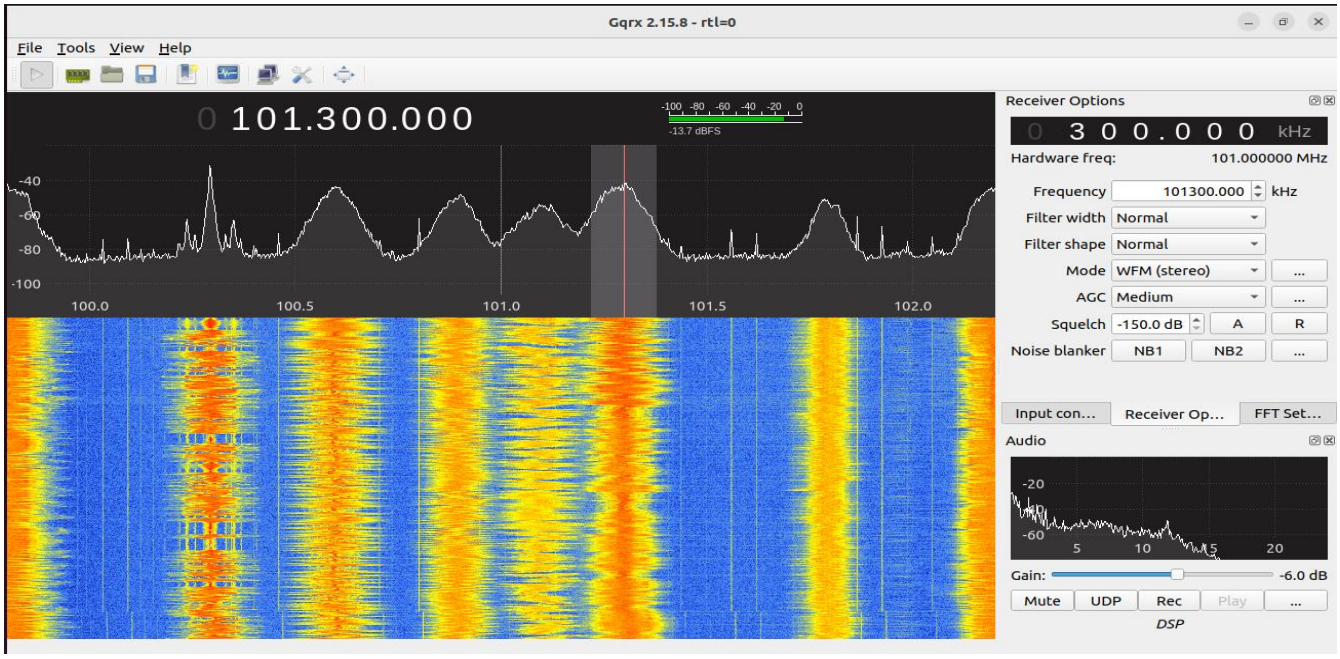
Το GNU Radio, όπως περιγράφεται στο [28] είναι ένα δωρεάν πρόγραμμα ανοιχτού κώδικα για SDR. Είναι διαθέσιμο για πολλές πλατφόρμες, Linux, Mac και Windows. Γλώσσες συγγραφής του είναι η C++ και η Python. Διαθέτει γραφικό περιβάλλον χρήστη, το GNU Radio Companion. Προσφέρει στοιχειώδεις μονάδες επεξεργασίας σήματος, υλοποιημένες σε λογισμικό. Χρησιμοποιείται είτε με υπάρχουσες συσκευές SDR, είτε σε περιβάλλον προσομοίωσης. Μπορεί να χρησιμοποιηθεί τόσο για λήψη (RX), όσο και για μετάδοση (TX). Η επεξεργασία γίνεται σε ένα γράφο ροής, με αρθρωτή σχεδίαση, αποτελούμενο από ένα πλήθος διασυνδεδεμένων μονάδων επεξεργασίας. Έτσι μπορούν για παράδειγμα να υλοποιηθούν κυκλώματα, όπως φίλτρα ή αποδιαμορφωτές, όλα σε λογισμικό, που τρέχει σε έναν απλό υπολογιστή. Υπάρχουν διαθέσιμα και γραφικά σε Qt, όπως γράφημα καταρράκτη ή επιλογείς, π.χ. συχνότητας συντονισμού. Επιτρέπει τον σχεδιασμό, την προσομοίωση και την εφαρμογή στην πράξη των επιθυμητών ραδιοσυστημάτων. Έχει εφαρμογή στην έρευνα, στη βιομηχανία και στους ερασιτέχνες που ασχολούνται με ασύρματες επικοινωνίες. Είναι από τα πρώτα προγράμματα για SDR που φτιάχτηκαν, καθώς ξεκίνησε το 2001. Έχει σημαντική θέση στο οικοσύστημα των προγραμμάτων SDR, κάτι που φαίνεται και στα [5,29], καθώς και πολλά άλλα προγράμματα για SDR βασίζονται σε αυτό (π.χ. το GQRX και το ShinySDR) ή επηρεάστηκαν από αυτό.

Δοκιμάστηκε και η λειτουργία μετάδοσης (TX). Χρησιμοποιήθηκε ένα B210 USRP του εργαστηρίου. Σαν σήματα εισόδου δοκιμάστηκαν: α) ένα αρχείο ήχου ενός τραγουδιού και β) ζωντανή καταγραφή φωνής. Ακολουθεί ο γράφος ροής που κατασκευάστηκε για την περίπτωση του τραγουδιού.



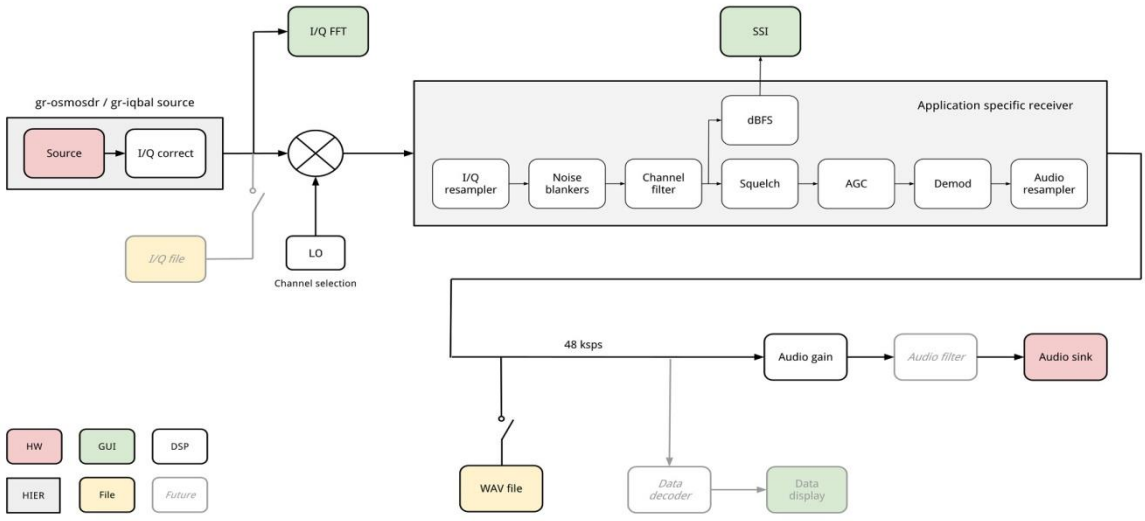
Εικόνα 2.4 *Στιγμιότυπο οθόνης από το GNU Radio, μετάδοση FM σήματος*

2.1.4 Gqrx



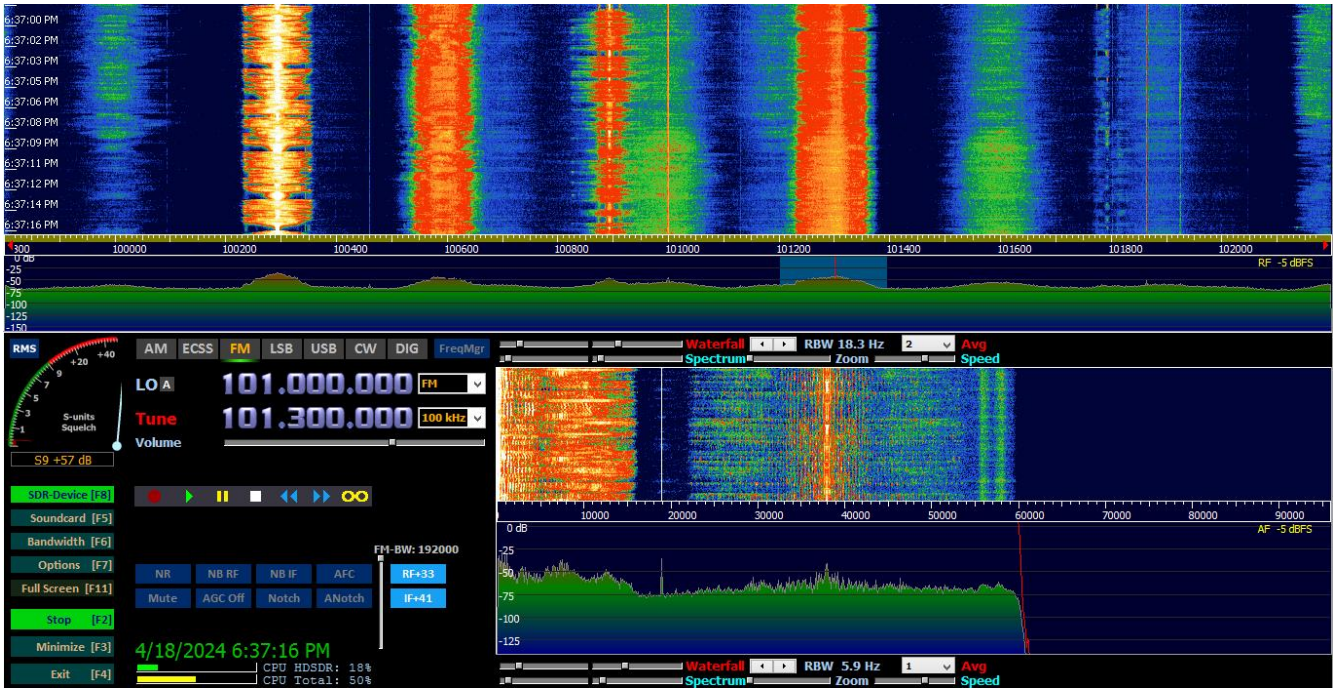
Εικόνα 2.5 *Στιγμιότυπο οθόνης από το Gqrx*

Το Gqrx είναι ένα ανοικτού κώδικα πρόγραμμα, για SDR που τρέχει σε Linux και Mac. Ο κώδικας βρίσκεται στο [30]. Όπως περιγράφεται στο [31], είναι γραμμένο από τον Alexandru Csete (Δανία). Η γλώσσα που γράφτηκε είναι η C++. Βασίζεται στο πρόγραμμα GNU Radio και στο εργαλείο γραφικών διεπαφών QT. Είναι πρόγραμμα απλό στη χρήση. Υποστηρίζει πολλές συσκευές SDR, μεταξύ των οποίων και τα RTL-SDR, Airspy, Funcube, HackRF, Ettus USRP και άλλα, μέσω του SoapySDR. Έχει δυνατότητα αναλογικών αποδιαμορφώσεων AM, SSB, CW, NFM, WFM (μονοφωνική και στερεοφωνική) και FM ειδικά για NOAA APT (δορυφόρους). Διαθέτει αυτόματο έλεγχο κέρδους (AGC), σιγασμό (squelch), μονάδα απαλοιφής θορύβου (noise blanker) και ρυθμιζόμενο βαθυπερατό φίλτρο. Μπορεί να καταγράψει ήχο ή ακατέργαστα δεδομένα σε αρχείο ή να δεχθεί ως είσοδο αποθηκευμένο αρχείο. Διαθέτει και πολύ απλή δυνατότητα ελέγχου από απομακρυσμένο υπολογιστή, μέσω telnet. Επίσης, διαθέτει δυνατότητα αποστολής ροής ήχου μέσω UDP. Αυτό μπορεί να χρησιμοποιηθεί για να προωθηθούν τα δεδομένα σε εξωτερικούς αποκωδικοποιητές. Σε συνδυασμό ο απομακρυσμένος έλεγχος και η αποστολή ροής ήχου, επιτρέπουν την σύνδεση σε υπολογιστή - εξυπηρετητή με συσκευή SDR και την απομακρυσμένη ακρόαση, π.χ. μέσω του VLC. Σε αυτήν την περίπτωση, στέλνεται μόνο ο ήχος, όχι όμως και το γράφημα καταρράκτη.



Εικόνα 2.6 *Αρχιτεκτονική του Gqrx* [32]

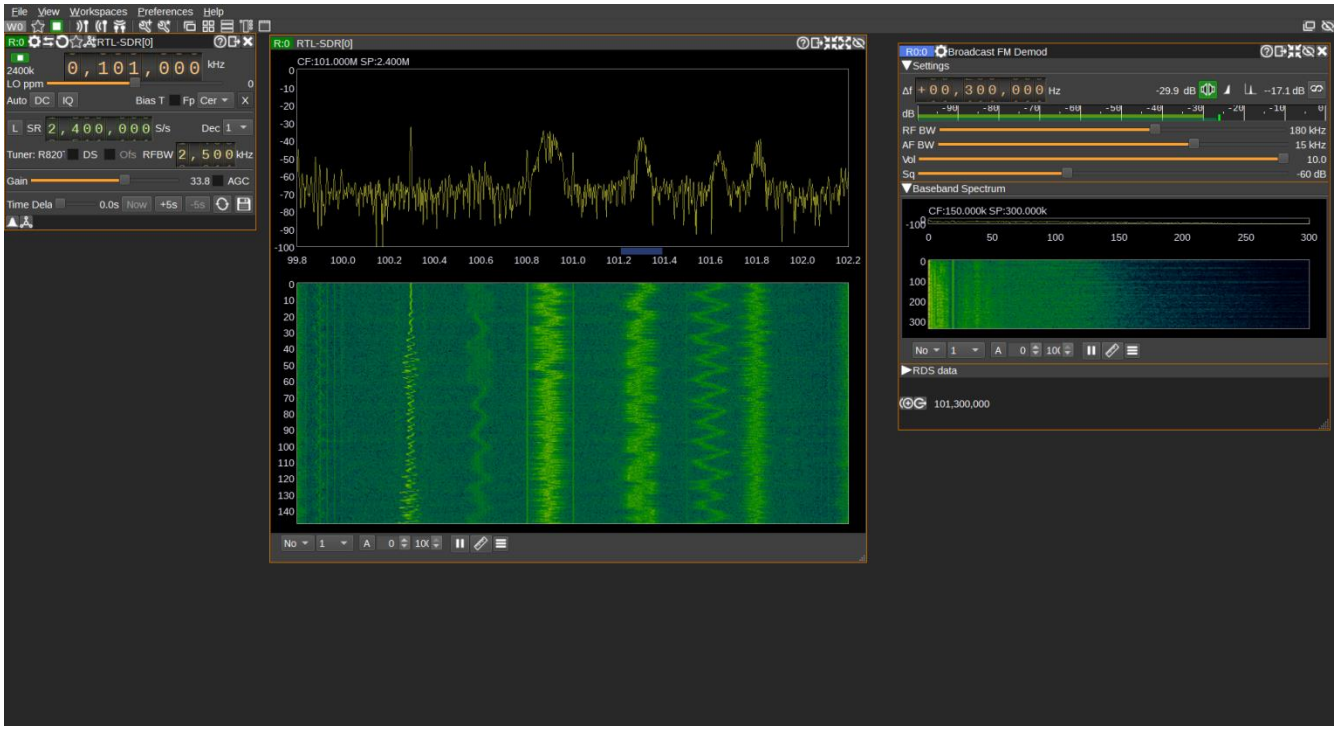
2.1.5 HDSDR



Εικόνα 2.7 *Στιγμιότυπο οθόνης από το HDSDR*

Το HDSDR, όπως περιγράφεται στο [33] είναι ένα πρόγραμμα για SDR αποκλειστικά για Windows. Είναι δωρεάν πρόγραμμα κλειστού κώδικα αν και βασίστηκε στο ανοικτού κώδικα Winrad. Υποστηρίζει αναλογικές αποδιαμορφώσεις όπως AM,FM,SSB,CW, ECSS. Βασικό χαρακτηριστικό του είναι η δυνατότητα μετάδοσης (TX), για όσες συσκευές το υποστηρίζουν. Διαθέτει σιγασμό, μείωση θορύβου, ρυθμιζόμενο βαθυπερατό φίλτρο και φίλτρο μη αναδίπλωσης φάσματος (anti-aliasing). Διαθέτει γραφήματα καταρράκτη τόσο για το ραδιοσήμα, όσο και για το σήμα ήχου. Μπορεί να καταγράψει ραδιοσήμα ή ήχο σε αρχείο αλλά και να αναπαραγάγει δείγματα από είσοδο αρχείου. Υπάρχει και δυνατότητα προγραμματισμού καταγραφής. Επιπλέον, διαθέτει λίστες αποθηκευμένων συχνοτήτων. Υποστηρίζει πολλές συσκευές SDR μέσω δυναμικών βιβλιοθηκών εξωτερικής εισόδου εξόδου (ExtIO DLL). Κάποιες από τις συσκευές που υποστηρίζει είναι τα RTL-SDR, SDRplay, Adalm Pluto, HackRF, Ettus USRP. Η πρώτη έκδοση του προγράμματος δημοσιεύτηκε το 2009, ενώ η τελευταία το 2022.

2.1.6 SDRangel

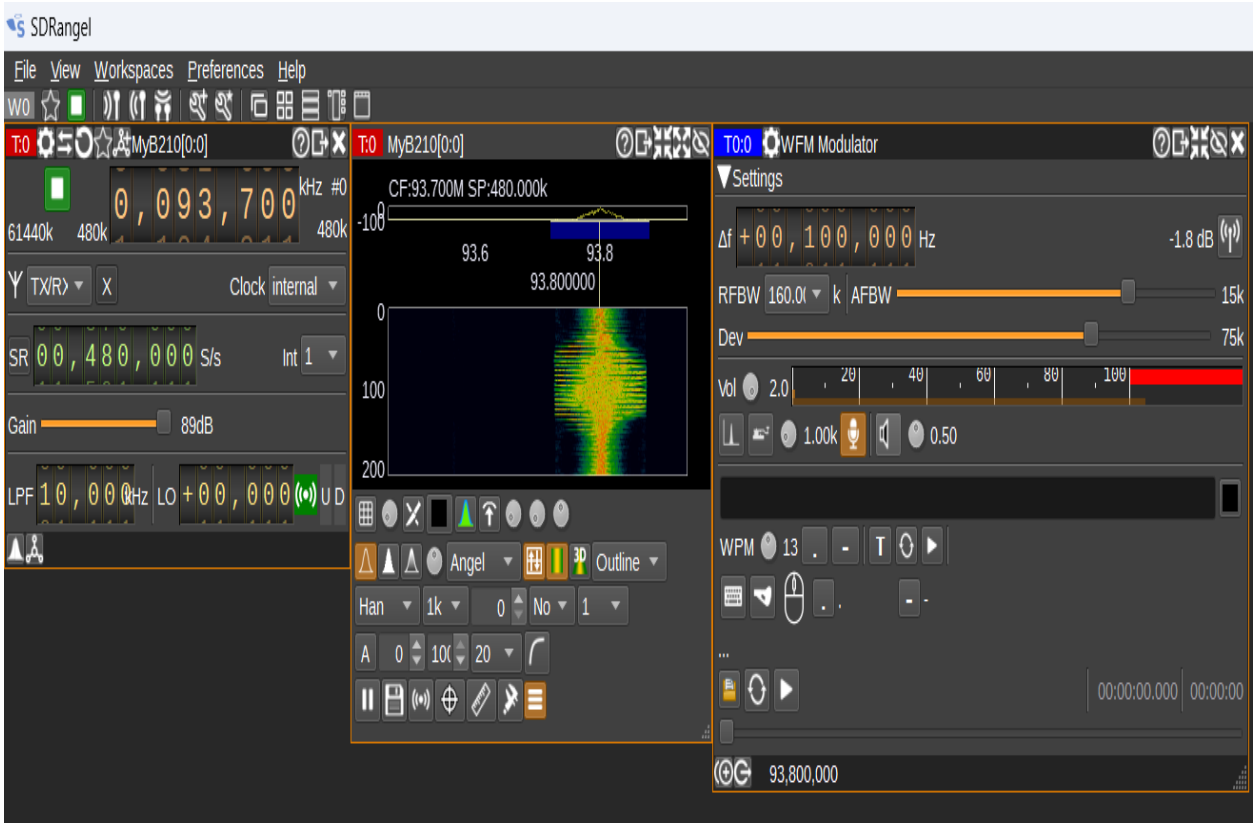


Εικόνα 2.8 *Στιγμιότυπο οθόνης από το SDRangel*

Το SDRangel, όπως περιγράφεται στο [34] είναι ένα ανοικτού κώδικα πρόγραμμα για SDR. Ο κώδικας βρίσκεται στο [35]. Τρέχει σε πολλές πλατφόρμες Windows, Linux, Mac και Android. Έχει γραφτεί από τον Edouard Griffiths (Γαλλία). Βασίστηκε στο παλαιότερο SDRangellove. Είναι γραμμένο σε C++ και χρησιμοποιεί Qt5 και OpenGL3.0+. Το πρόγραμμα διαθέτει αρκετές προχωρημένες λειτουργίες. Αρχικά, εκτός από αναλογικές διαμορφώσεις, υποστηρίζει μια πληθώρα ψηφιακών αποκωδικοποιητών, με δυνατότητα ταυτόχρονης χρήσης πολλών αποδιαμορφωτών. Επίσης σημαντικό χαρακτηριστικό του είναι η υποστήριξη λειτουργίας μετάδοσης (TX). Ακόμα υποστηρίζει MIMO λειτουργία. Υπάρχουν ρυθμίσεις για παραμετροποίηση πολλών λειτουργιών, όπως των αποδιαμορφωτών και του γραφήματος καταρράκτη. Διαθέτει πρόσθετες λειτουργίες μεταξύ των οποίων εντοπισμός δορυφόρων και αστεριών, ο έλεγχος περιστροφέα κεραίας και ραδιοαστρονομικές μετρήσεις. Στο πρόγραμμα περιλαμβάνονται και εργαλεία κεραιών, π.χ. για υπολογισμό του μήκους διπόλου, ανάλογα με τη συχνότητα. Περιλαμβάνονται και

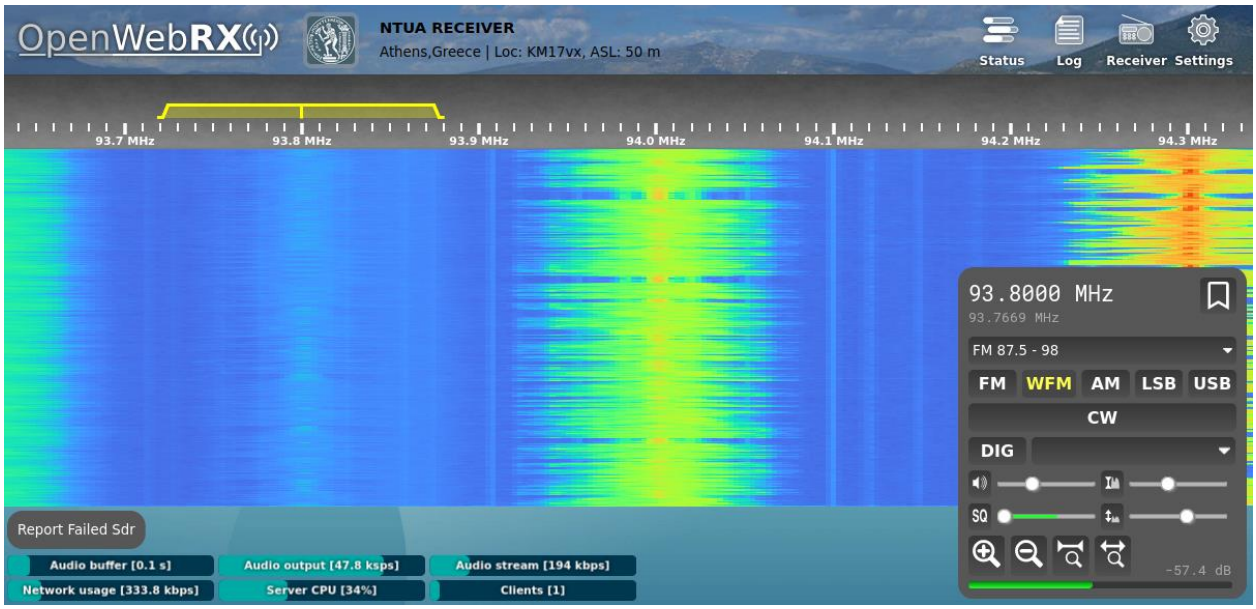
ενσωματωμένοι χάρτες γης και ουρανού. Μπορεί να τρέξει και σε κάρτα γραφικών, αξιοποιώντας τις δυνατότητες της. Διαθέτει και δυνατότητα απομακρυσμένου ελέγχου. Υποστηρίζει πολλές συσκευές SDR, μεταξύ των οποίων τα Airspy, BladeRF, Fun Cube, HackRF, KiwiSDR, LimeSDR, PlutoSDR, RTL SDR, SDRplay, USRP B210 και άλλες μέσω SoapySDR.

Δοκιμάστηκε και η λειτουργία μετάδοσης (TX). Χρησιμοποιήθηκε ένα B210 USRP του εργαστηρίου. Σαν σήματα εισόδου δοκιμάστηκαν: α) ένα αρχείο ήχου ενός τραγουδιού και β) ζωντανή καταγραφή φωνής. Ακολουθεί στιγμιότυπο οθόνης κατά την μετάδοση φωνής FM.



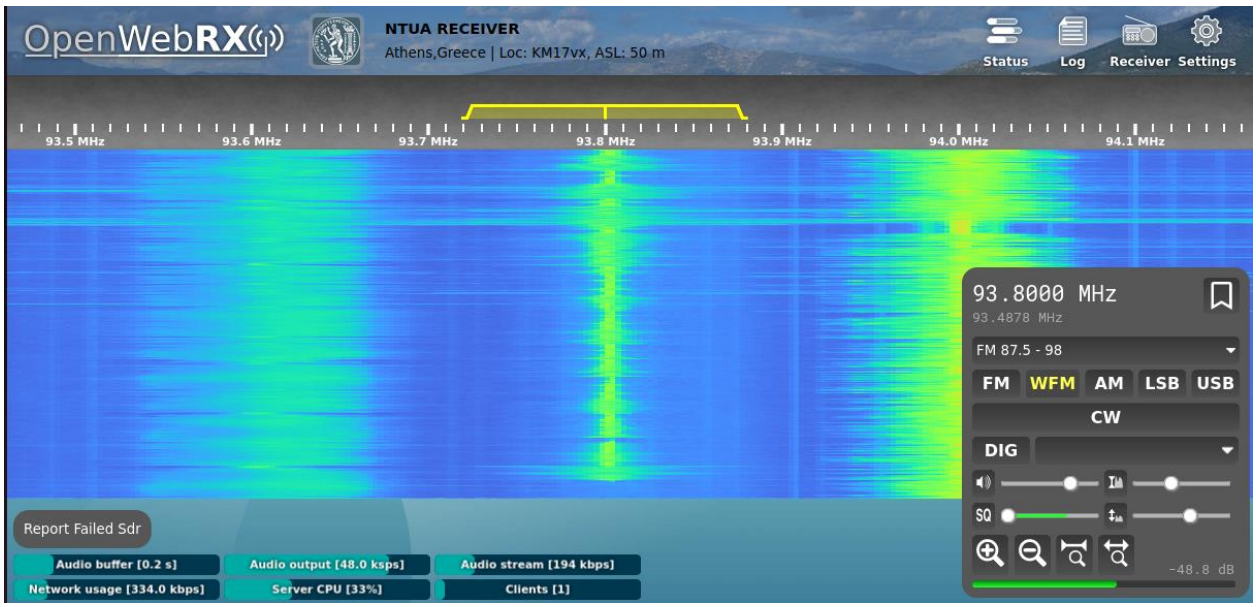
Εικόνα 2.9 *Στιγμιότυπο οθόνης από το SDRangel, μετάδοση φωνής FM.*

Ακολουθεί στιγμιότυπο οθόνης, που λήφθηκε χρησιμοποιώντας τον δέκτη του OpenWebRX. Φαίνεται μια κενή θέση στο φάσμα της μπάντας του ραδιοφώνου FM, γύρω από τη συχνότητα 93.8MHz, πριν από την μετάδοση.



Εικόνα 2.10 *Στιγμιότυπο οθόνης, που δείχνει μια κενή θέση στο φάσμα των FM*

Χρήση του φάσματος, κατά τη μετάδοση τραγουδιού, στη συχνότητα 93.8 MHz.



Εικόνα 2.11 *Στιγμιότυπο οθόνης, κατά τη μετάδοση στα FM.*

2.2 Λογισμικά για SDR με διαδικτυακή (web) διεπαφή

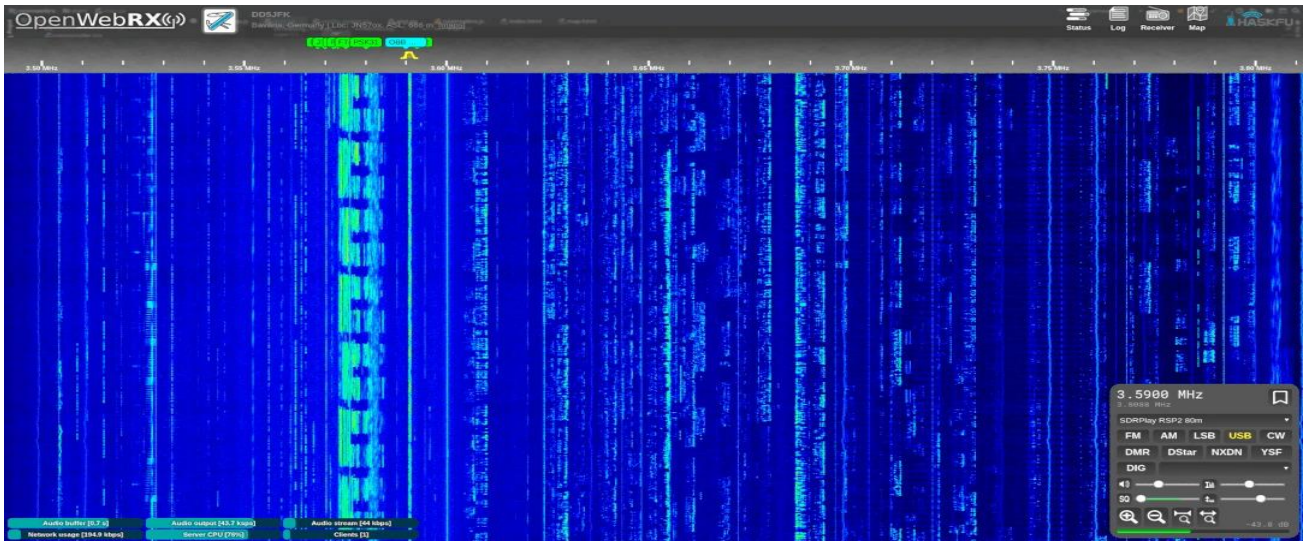
2.2.1 OpenWebRX

Το OpenWebRX είναι λογισμικό για Linux που τρέχει σε υπολογιστή συνδεδεμένο με συσκευές SDR. Επιτρέπει τη λήψη του ραδιοσήματος (RF) και την αποδιαμόρφωση του. Το αποδιαμορφωμένο σήμα και το γράφημα καταρράκτη (waterfall) της ανάλυσης συχνοτήτων Fourier, στέλνονται μέσω Internet στον περιηγητή (browser) των χρηστών, σε όλο τον κόσμο.

Περιλαμβάνει βιβλιοθήκη αποδιαμόρφωσης γραμμένη σε C++, εξυπηρετητή γραμμένο σε Python και διαδικτυακή (web) διεπαφή με HTML5 χαρακτηριστικά. Υποστηρίζει ένα ευρύ φάσμα συσκευών SDR.

Υποστηρίζει αναλογικές διαμορφώσεις όπως AM, FM, SSB, CW και πληθώρα ψηφιακών κωδικοποιήσεων φωνής, δεδομένων και ασθενούς σήματος (WSJT).

Ο διαχειριστής μπορεί να ορίσει με εύκολο τρόπο τις διαθέσιμες συσκευές SDR και τις ζώνες συχνοτήτων που αυτές θα λειτουργούν. Ο χρήστης αφού συνδεθεί στην ιστοσελίδα μπορεί να επιλέξει το σήμα που θα ακούσει/αποκωδικοποιήσει. Υπάρχει υποστήριξη για πολλούς ταυτόχρονους χρήστες.



Εικόνα 2.12 *OpenWebRX [36]*

Περισσότερες Πληροφορίες

Αρχικός δημιουργός του OpenWebRX είναι ο Andras Retzler (HA7ILM). Παρουσίασε την πρώτη έκδοση του προγράμματος στην πτυχιακή του εργασία, το 2014. Τότε το OpenWebRX υποστήριζε μόνο το RTL-SDR. Από τότε έγιναν αρκετές προσθήκες και βελτιώσεις, τόσο από τον ίδιο, όσο και από τον Jakob Ketterl (DD5JFK), ο οποίος είναι και ο τωρινός συντηρητής του.

Σκοπός του OpenWebRX σύμφωνα με την ιστοσελίδα του *https://openwebrx.de* [36] είναι η τοποθέτηση του σε μέρη, όπου υπάρχει καλή λήψη ενδιαφέροντων σημάτων. Για παράδειγμα σε σημεία χαμηλού θορύβου στη HF ή υπερυψωμένα σημεία στις VHF, UHF ή υψηλότερες ζώνες συχνοτήτων. Έτσι θα μπορεί να προσφέρει πρόσβαση στη λήψη ενδιαφέροντων σημάτων σε οποιονδήποτε θέλει, αλλά διαφορετικά δεν θα είχε τη δυνατότητα να τα λάβει από μόνος του.

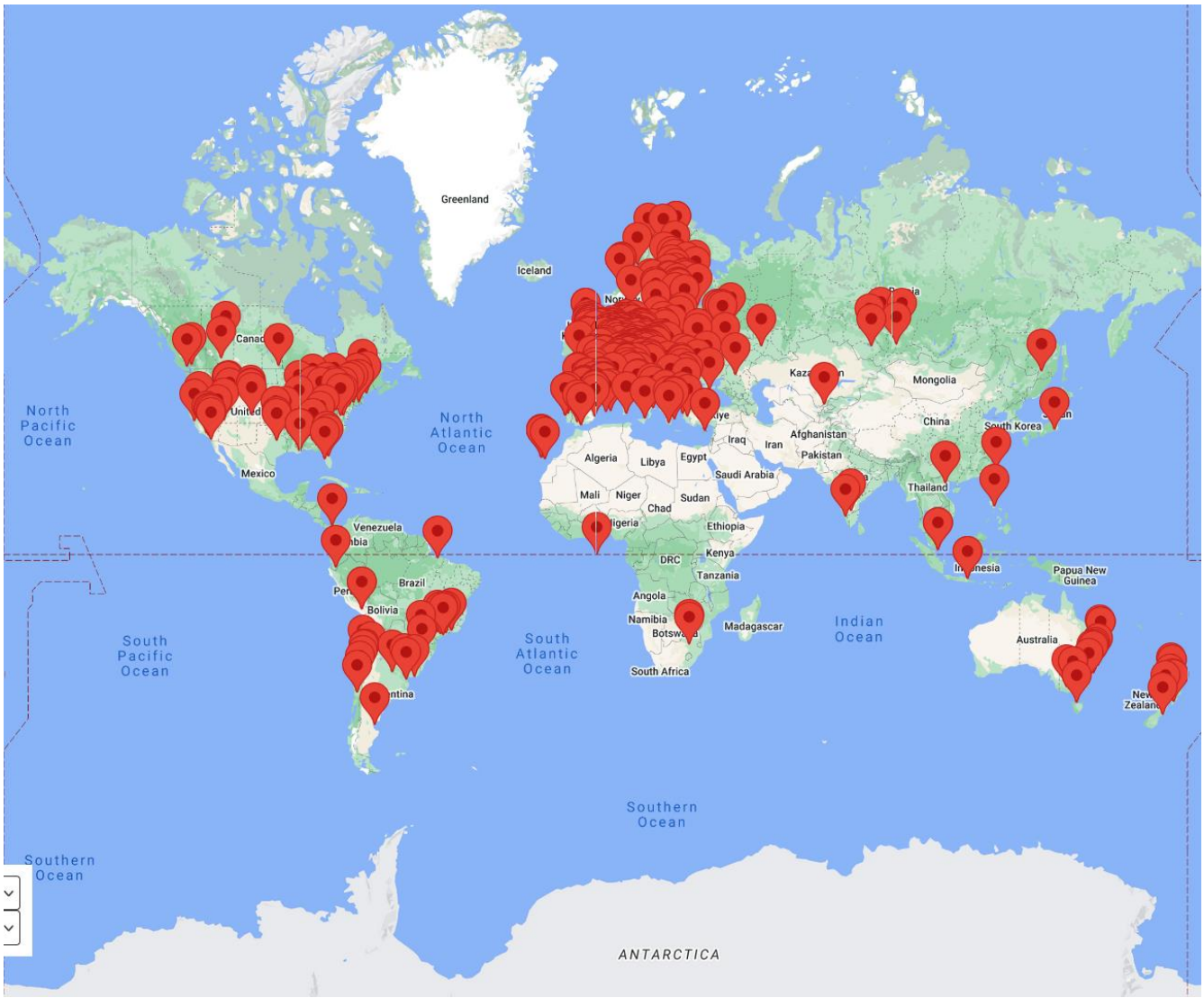
Στόχοι του OpenWebRX είναι:

1) Εστίαση σε δημόσιους και ανοικτούς δέκτες.
2) Ιδιότητες και χαρακτηριστικά που προωθούν και υποστηρίζουν την πρόσβαση από πολλαπλούς χρήστες.
3) Ευκολία στη χρήση από αρχάριους που δεν ξέρουν ακόμα πολλά πράγματα σχετικά με τις ραδιοεπικοινωνίες.
4) Ενσωμάτωση των αποκωδικοποιητών απευθείας στον περιηγητή (browser) ώστε να μην χρειάζονται επιπλέον προγράμματα στον υπολογιστή του χρήστη.
5) Ποιότητα του κώδικα για μεγαλύτερη διάρκεια ζωής του.
6) Ευκολία στην εγκατάσταση από τους χειριστές.
7) Χρήση των ανενεργών πόρων με παρασκηνιακή λειτουργία αποκωδικοποίησης και αναφορά (reporting).

Ο κώδικας του προγράμματος βρίσκεται στο Github: *https://github.com/jketterl/openwebrx* [37]. Όλα του τα κομμάτια είναι διαθέσιμα με ανοικτού κώδικα (open source) άδειες.

2.2.2 Receiverbook

Υπάρχει ένας κατάλογος από διαθέσιμους OpenWebRX εξυπηρετητές (και όχι μόνο) στην ιστοσελίδα *https://receiverbook.de* . [38] Εκεί κάθε ενδιαφερόμενος μπορεί να ακούσει/αποκωδικοποιήσει μια μεγάλη ποικιλία διαθέσιμων σημάτων που λαμβάνονται και διαμοιράζονται από OpenWebRX (και άλλους) σταθμούς σε όλο τον κόσμο.

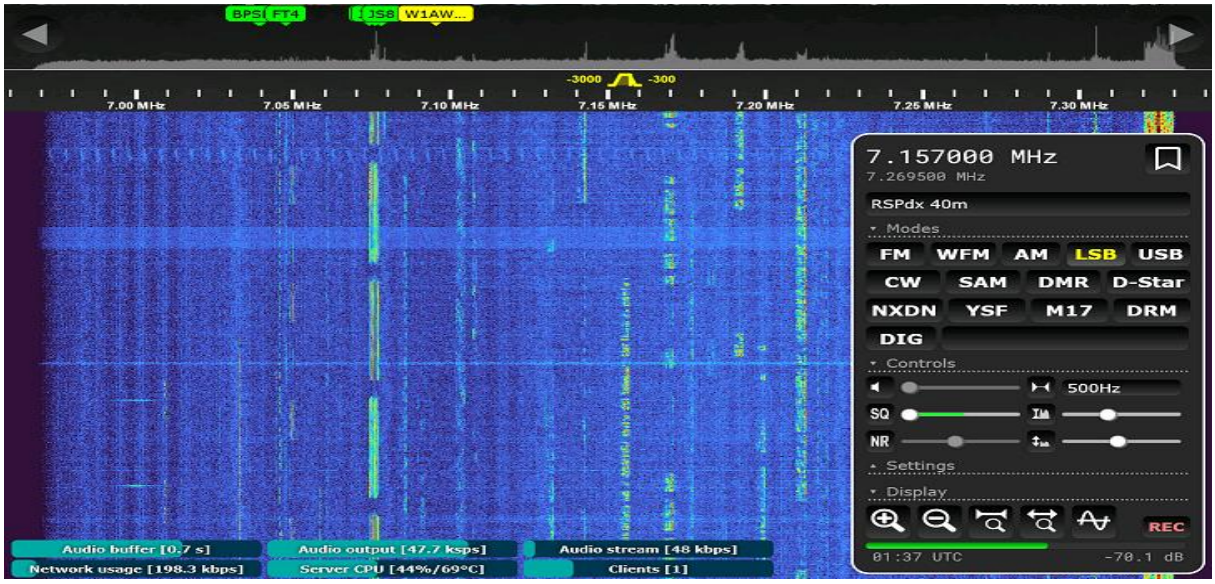


Εικόνα 2.13 *Χάρτης του Receiverbook*

2.2.3 OpenWebRX+

Υπάρχει μια πρόσφατη επέκταση (2023), το OpenWebRX+. Παρουσιάζεται στο [39] και προσθέτει πολλές νέες λειτουργίες όπως:

1) Πολλούς νέους ψηφιακούς αποκωδικοποιητές.
2) Λειτουργία ανίχνευσης συχνοτήτων (scanner) πραγματοποιώντας αναζήτηση σε προεπιλεγμένους σταθμούς (bookmarks).
3) Ενσωματωμένη συνομιλία (chat).
4) Καταγραφή και κατέβασμα αρχείου ήχου.
5) Μια επιπλέον γραφική φάσματος, πάνω από το γράφημα καταρράκτη (waterfall).
6) Επιτρέπει στον διαχειριστή να βλέπει τις συνδέσεις.
7) Συμπεριλαμβάνει ενημερωμένο χάρτη με πληροφορίες όπως δημόσια web SDR, πομπούς βραχέων, επαναλήπτες και θέσεις αεροπλάνων για όλο τον κόσμο.
8) Βελτιώνει την υποστήριξη των SDRPlay συσκευών.
9) Υποστηρίζει ρυθμιζόμενο βήμα συντονισμού.
10) Εστίαση (zoom) για συσκευές οθόνες αφής (touchscreen).
11) Προσθέτει και υποστήριξη HTTPS πρωτοκόλλου για ασφαλή δικτυακή σύνδεση (απαιτείται έκδοση πιστοποιητικού).



Εικόνα 2.14 *OpenWebRX+ [40]*

Αρκετοί OpenWebRX+ εξυπηρετητές παγκοσμίως χρησιμοποιούν αυτήν τη νέα έκδοση. Ένας αναλυτικός οδηγός χρήσης του βρίσκεται στο [40].

2.2.4 WebSDR

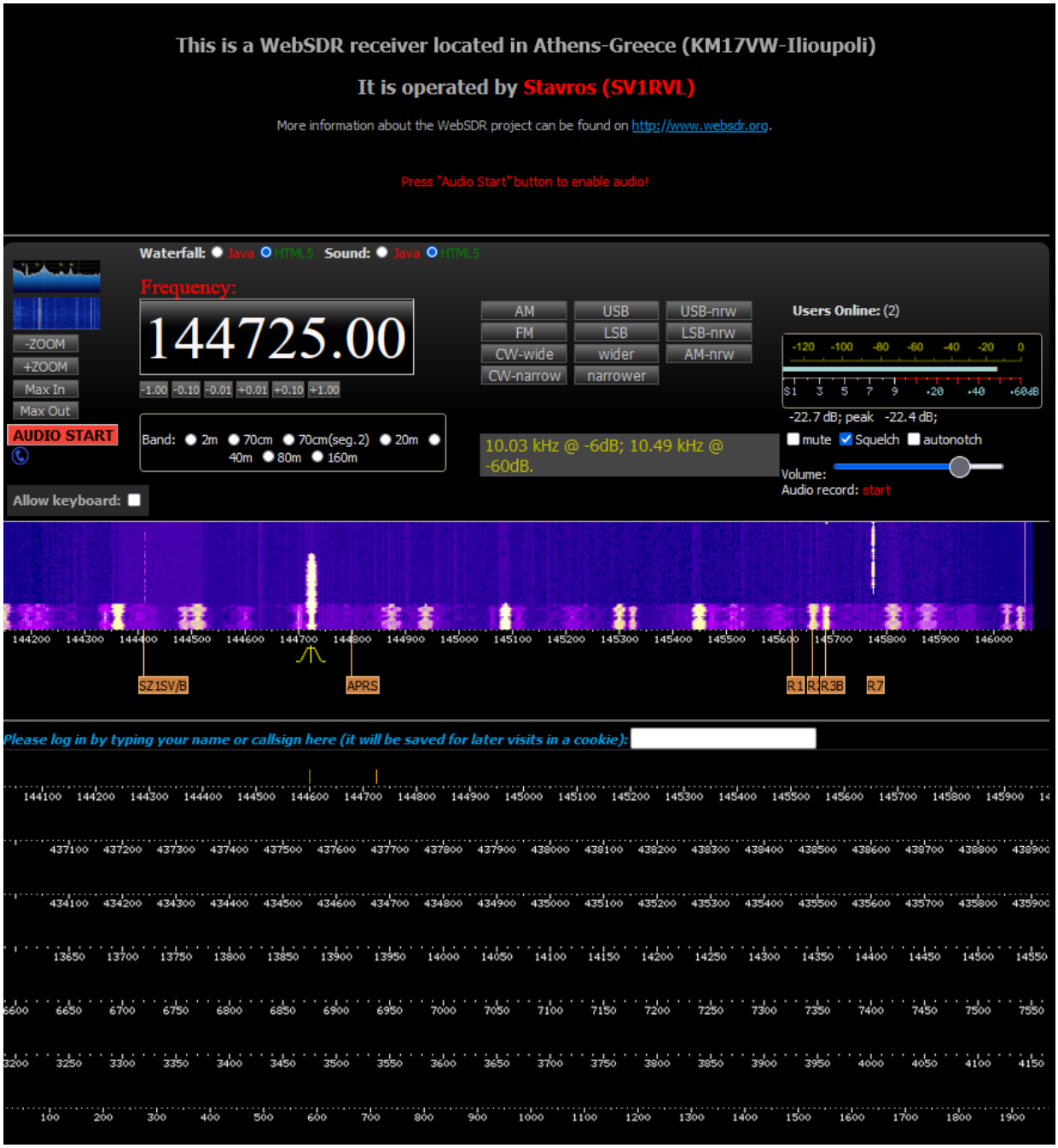
Ένα άλλο προγράμματα σαν το OpenWebRX είναι το WebSDR. Το WebSDR όπως περιγράφεται στο [41] επιτρέπει και αυτό σε πολλούς χρήστες να λαμβάνουν και να ακούν στον περιηγητή τους το σήμα της SDR συσκευής. Υποστηρίζει και αυτό ένα σύνολο από ζώνες συχνοτήτων, μέσα από τις οποίες επιλέγει ο χρήστης το επιθυμητό σήμα. Εμφανίζει γράφημα καταρράκτη και αποδιαμορφώνει το επιλεγμένο σήμα προς ακρόαση. Υποστηρίζει μόνο αναλογικές διαμορφώσεις. Υπάρχει επιλογή για χρήση της εφαρμογής μέσω Java ή HTML5. Υπάρχει δυνατότητα και για καταγραφή του ήχου. Δεν υποστηρίζεται η ζώνη του ραδιοφώνου FM.

Είναι λογισμικό κλειστού κώδικα (proprietary software). Δεν βρίσκεται κάπου διαθέσιμο προς κατέβασμα (download), αλλά διαμοιράζεται δωρεάν κατόπιν συνεννόησης με τον δημιουργό του μέσω ηλεκτρονικής αλληλογραφίας (email).

Ο πρώτος WebSDR σταθμός ξεκίνησε στο πανεπιστήμιο Twente της Ολλανδίας το 2008 και καλύπτει όλο το φάσμα από τα 0 μέχρι τα 30Mhz. Τρέχει μια ειδική έκδοση του WebSDR και χρησιμοποιεί μια ιδιοκατασκευασμένη πλακέτα SDR. Περισσότερες πληροφορίες υπάρχουν στο [42].

Το WebSDR είναι αυτή τη στιγμή πολύ διαδεδομένο στην Ελλάδα και υπάρχουν αρκετοί σταθμοί του σε διάφορα μέρη.

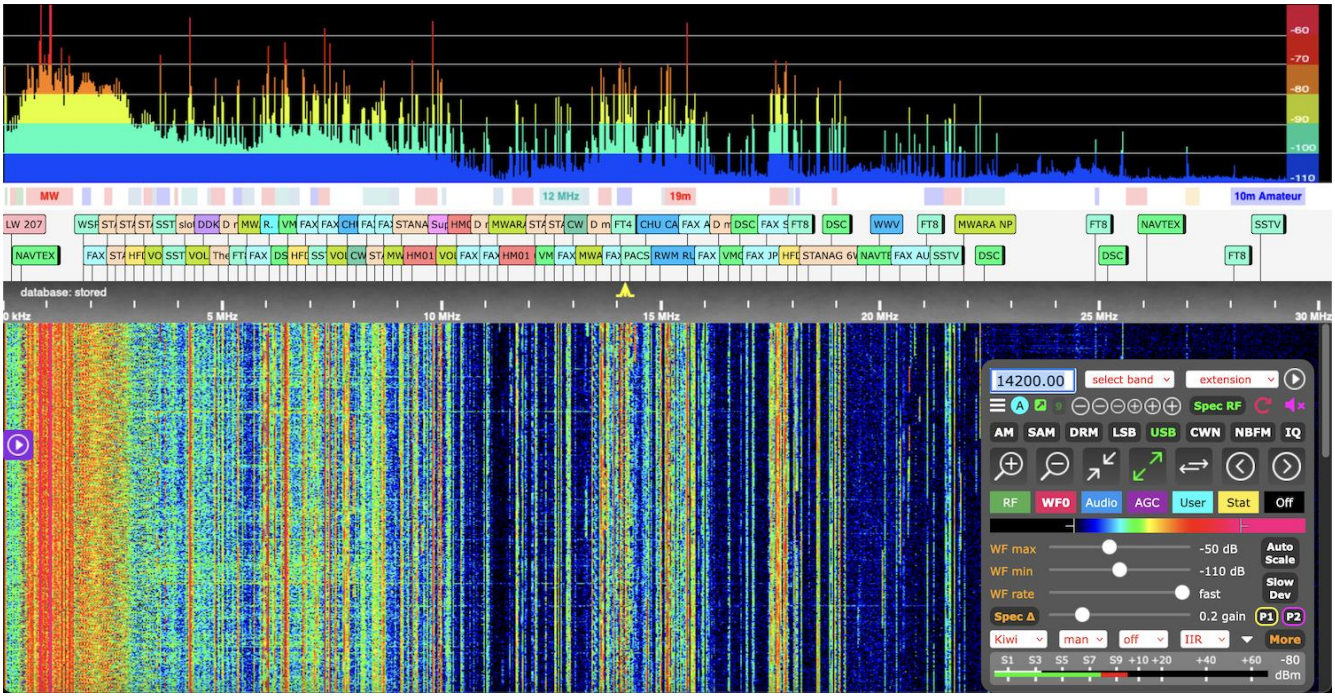
Το παρακάτω στιγμιότυπο οθόνης (screenshot) είναι από εξυπηρετητή WebSDR στην Αθήνα (Ηλιούπολη) που λειτουργεί στις ζώνες 2m, 70cm (2), 20m, 40m, 80m, 160m.



Εικόνα 2.15 *Στιγμιότυπο οθόνης από το WebSDR*

2.2.5 KiwiSDR

Το KiwiSDR, όπως περιγράφεται στο [24] είναι μια SDR συσκευή που συνδέεται στο διαδίκτυο και υποστηρίζει σύνδεση πολλών χρηστών μέσω web. Ο κώδικας που τρέχει περιέχει μια τροποποιημένη εκδοχή του OpenWebRX. Βασίζεται στην πλακέτα Beaglebone, (το Beaglebone είναι σαν το RaspberryPi). Λαμβάνει σήματα ολόκληρης της ζώνης συχνοτήτων 10KHz – 30 MHz, στις μπάντες VLF, LF, MW, HF. Είναι ανεξάρτητη συσκευή και δεν χρειάζεται να συνδεθεί με υπολογιστή. Όλη η επεξεργασία του σήματος γίνεται στο Beaglebone και στο FPGA που διαθέτει. Ο χρήστης συνδέεται μέσω web διεπαφής απευθείας στη συσκευή. Είναι σχετικά φτηνό και ανοικτού κώδικα (open source). Απαιτείται σύνδεση με καλώδιο Ethernet ή USB ασύρματος προσαρμογέας δικτύου (wireless network adapter). Δέχεται και κεραία GPS και διαθέτει GPS Disciplined Oscillator για εξαιρετική σταθερότητα συχνότητας. Υποστηρίζει έως 4 ταυτόχρονους χρήστες ή έως 8 με κάποιους όμως περιορισμούς.

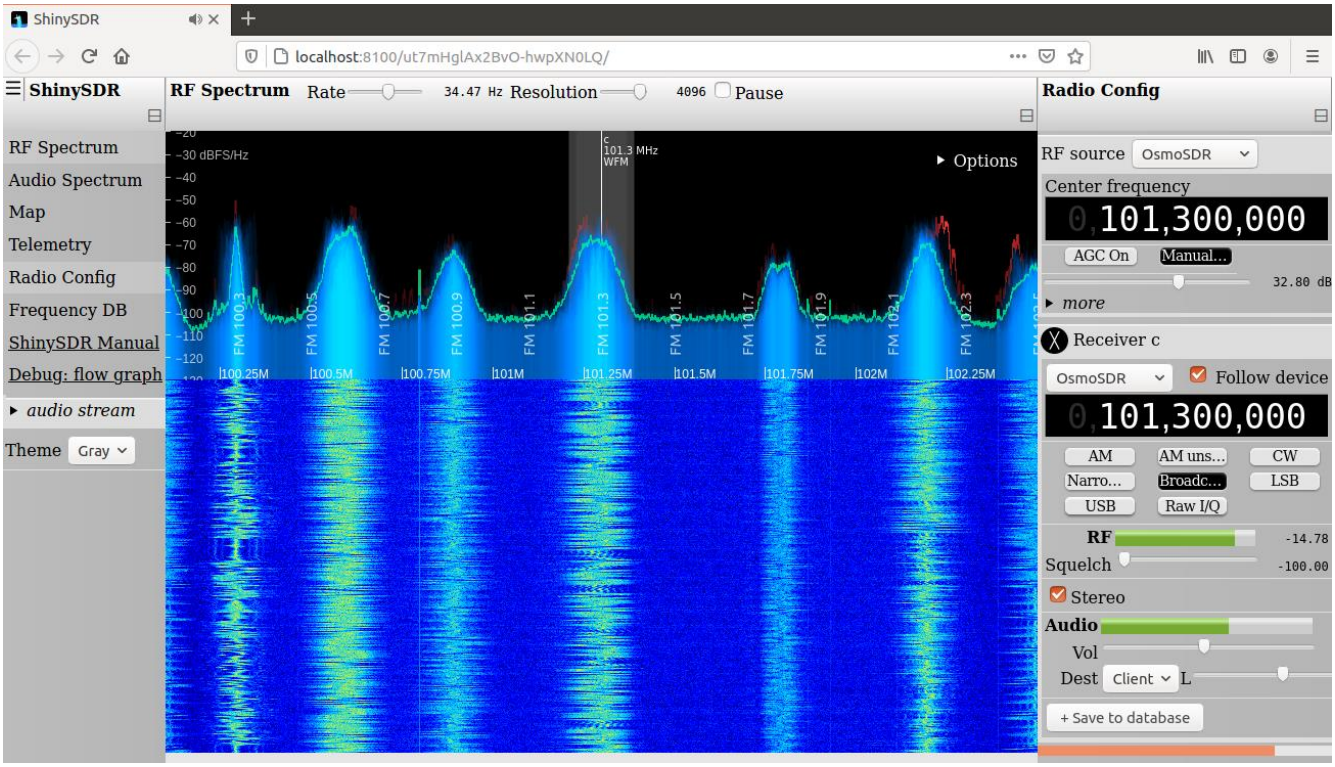


Εικόνα 2.16 *Στιγμιότυπο οθόνης από το KiwiSDR*

Μαζί με τους OpenWebRX, μερικοί WebSDR και μερικοί εξυπηρετητές του KiwiSDR είναι καταχωρημένοι στο *https://receiverbook.de* . [38]

2.2.6 ShinySDR

Το ShinySDR, όπως περιγράφεται στο [43], είναι γραμμένο σε Python και χρησιμοποιεί το GNU Radio για να λάβει και να αποκωδικοποιήσει το σήμα της συσκευής SDR. Υποστηρίζει αναλογικές διαμορφώσεις, ψηφιακό ήχο, κείμενο και κωδικοποιήσεις τηλεμετρίας, μέσω του rtl433. Πολλαπλά σήματα, εντός φάσματος, μπορούν να ληφθούν ταυτόχρονα. Έχει HTML5 διεπαφή και είναι προσβάσιμο από τον περιηγητή (κατά προτίμηση Chrome). Υποστηρίζει συσκευές SDR όπως τα RTL-SDR, HackRF, Ettus USRP. Είναι ανοικτού κώδικα (open source) και έχει GNU General Public άδεια. Δεν υποστηρίζει όμως πολλαπλούς χρήστες και έχει σταματήσει να συντηρείται.



Εικόνα 2.17 *Στιγμιότυπο οθόνης από το ShinySDR*

Έτσι προκύπτει ότι το OpenWebRX είναι το καταλληλότερο πρόγραμμα ανοικτού κώδικα (open source) αυτή τη στιγμή, για διαμοιρασμό σε πολλούς χρήστες, μέσω web, των σημάτων που λαμβάνουν SDR συσκευές.

3. OpenWebRX

3.1 Τεχνικές Λεπτομέρειες

Όπως ήδη αναφέρθηκε (στην παράγραφο 2.2.1) το OpenWebRX είναι λογισμικό για Linux που τρέχει σε υπολογιστή συνδεδεμένο με συσκευές SDR. Επιτρέπει τη λήψη του ραδιοσήματος (RF) και την αποδιαμόρφωση του. Το αποδιαμορφωμένο σήμα και το γράφημα καταρράκτη (waterfall) της ανάλυσης συχνοτήτων Fourier, στέλνονται μέσω Internet στον περιηγητή (browser) των χρηστών, σε όλο τον κόσμο.

Το OpenWebRX αποτελείται από:

1) τη βιβλιοθήκη ψηφιακής επεξεργασίας σήματος
2) τον HTTP εξυπηρετητή
3) τη πλευρά του χρήστη με τη διεπαφή για τον περιηγητή (browser).
4) τμήματα κώδικα που χειρίζονται τις συσκευές SDR.

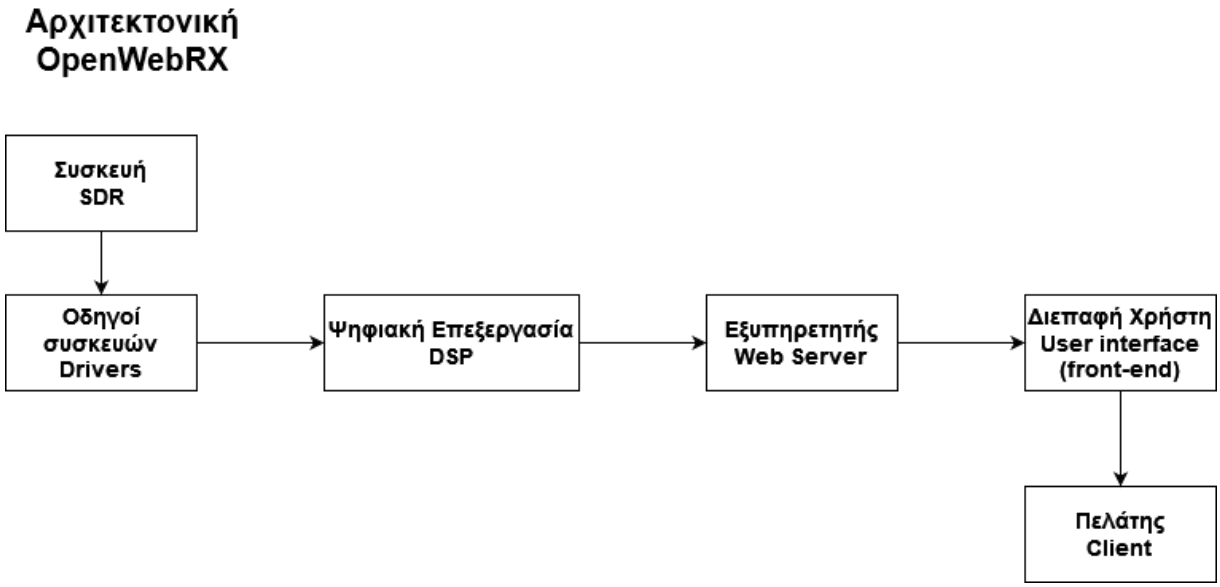
Η επικοινωνία με τις συσκευές SDR και ο χειρισμός τους γίνεται είτε απευθείας μέσω των αντίστοιχων οδηγών συσκευών (drivers), είτε μέσω μιας προγραμματιστικής διεπαφής οδηγού συσκευών (driverAPI), το SoapySDR. Αυτούς του οδηγούς συσκευών αλλά και το SoapySDR ενθυλακώνει (wrap) ο owrx_connector. Υποστηρίζονται πολλές διαφορετικές συσκευές, πολλών διαφορετικών κατασκευαστών.

Η αποδιαμόρφωση είναι υλοποιημένη εξ ολοκλήρου σε λογισμικό. Οι λειτουργίες ψηφιακής επεξεργασίας σήματος (DSP) βρίσκονται στη βιβλιοθήκη libcsdr και είναι γραμμένες σε C++ για απόδοση. Υποστηρίζονται αναλογικές αποδιαμορφώσεις όπως FM, AM, SSB,CW. Υποστηρίζονται ακόμα πολλές ψηφιακές αποκωδικοποιήσεις: φωνής, δεδομένων και αδύναμου σήματος (WSJT).

Ο HTTP εξυπηρετητής είναι γραμμένος σε Python και βασίζεται στο διαθέσιμο τμήμα κώδικα (module) http.server. Χρησιμοποιείται JavaScript και jQuery για τη διαδικτυακή διεπαφή χρήστη και WebSocket για αμφίδρομη επικοινωνία και την αποστολή του γραφήματος καταρράκτη και του ήχου. Η εμφάνιση του γραφήματος καταρράκτη γίνεται μέσω HTML5 Canvas. Υπάρχει και σελίδα χάρτη για εμφάνιση σημειωμένων τοποθεσιών

και σχετικών πληροφοριών. Υποστηρίζεται εντοπισμός και αναφορά σε τρίτες υπηρεσίες (spotting and reporting), χρήσιμα για παράδειγμα στο APRS.

Ο διαχειριστής μπορεί να ορίσει με εύκολο τρόπο τις διαθέσιμες συσκευές SDR και τις ζώνες συχνοτήτων που αυτές θα λειτουργούν. Ο χρήστης αφού συνδεθεί στην ιστοσελίδα της εφαρμογής μπορεί να επιλέξει το σήμα που θα ακούσει/αποκωδικοποιήσει. Υπάρχει υποστήριξη για πολλούς ταυτόχρονους χρήστες, με κάποιους όμως περιορισμούς. Κάτι που έρχεται να λύσει το Load Balancing OpenWebRX.



Σχήμα 3.2 *Αρχιτεκτονική OpenWebRX*

Στα επόμενα κεφάλαια αυτής της ενότητας θα εξεταστούν με λεπτομέρεια τα τμήματα από τα οποία αποτελείται το OpenWebRX.

3.2 Οδηγοί συσκευών στο OpenWebRX

3.2.1 Οδηγοί συσκευών (Drivers)

Το OpenWebRX υποστηρίζει πληθώρα από συσκευές SDR. Ένα βασικό πρόβλημα που αντιμετωπίζει είναι η διαχείριση όλων αυτών των οδηγών (drivers) των συσκευών. Το λύνει χρησιμοποιώντας την βιβλιοθήκη ενθυλάκωσης (wrapper) owrx_connector. Μια βιβλιοθήκη ενθυλάκωσης μεταφράζει μια αρχική διεπαφή, σε μια άλλη πιο βολική διεπαφή. Έτσι το OpenWebRX καλεί το κατάλληλο συνδετικό πρόγραμμα (connector) και όχι απευθείας τον οδηγό της συσκευής.

3.2.2 Συνδετικά προγράμματα (Connectors)

Το OpenWebRX έχει πρόσβαση στις συσκευές SDR μέσω της librtlsdr στην περίπτωση του RTL-SDR ή μέσω του SoapySDR για τις υπόλοιπες συσκευές. Στην πρώτη περίπτωση χρησιμοποιούνται η ενθυλάκωση (wrapper) rtl_connector ή ο παλαιότερος rtl_tcp_connector, ενώ στη δεύτερη η ενθυλάκωση (wrapper) soapy_connector. Οι connectors δέχονται ως ορίσματα:

1) την κεντρική συχνότητα (center frequency)
2) τον ρυθμό δειγματοληψίας (sample rate)
3) το κέρδος ενίσχυσης (gain)
4) την διόρθωση συχνότητας (ppm)
5) την επιλογή εισόδου κεραίας (αν υπάρχουν πολλές στη συσκευή) (antenna)

Και για την περίπτωση του RTL-SDR:

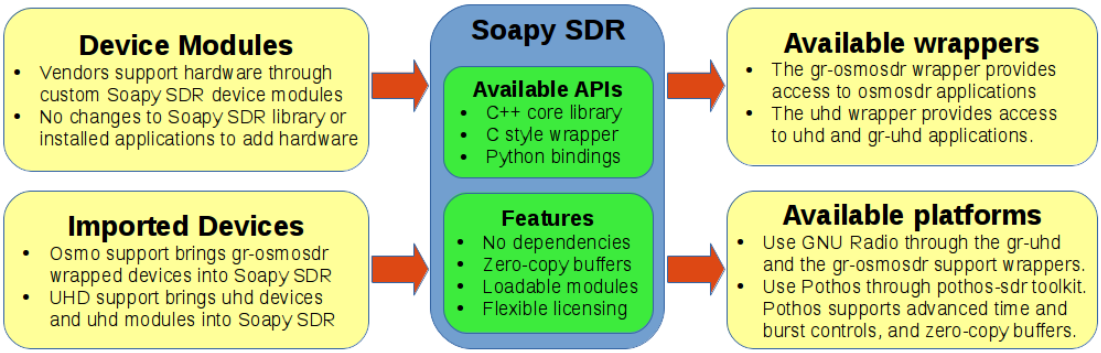
6) απευθείας δειγματοληψία (direct sampling)
7) σταθερή (DC) τάση τροφοδοσίας (bias tee)

Επίσης δέχονται ως ορίσματα και 2 θύρες, μια για δεδομένα και μια για έλεγχο. Στις θύρες αυτές ανοίγουν υποδοχές (sockets). Η υποδοχή δεδομένων χρησιμοποιείται για τη ροή των

μιγαδικών δειγμάτων (IQ stream), ενώ η υποδοχή ελέγχου για αλλαγή των παραμέτρων της συσκευής SDR κατά τη λειτουργία (runtime).
Μετά τον connector, τα IQ δεδομένα διαμοιράζονται μέσω TCP σε πολλαπλούς πελάτες (νήματα του OpenWebRX) για να τα επεξεργαστούν (DSP).

3.2.3 SoapySDR

Επειδή οι υποστηριζόμενες συσκευές είναι πολλές και διαφορετικές, χρειάζεται ένας ενιαίος τρόπος αντιμετώπισης τους. Το SoapySDR, όπως περιγράφεται στο [44] είναι ένα γενικό API και βιβλιοθήκη που ενθυλακώνει τις ιδιαιτερότητες της κάθε SDR συσκευής και προσφέρει ένα γενικό τύπο συσκευής SDR. Έτσι το OpenWebRX αρκεί να καλέσει το SoapySDR και μέσω των τμημάτων κώδικα (modules) του αποκτά πρόσβαση σε όλες τις υποστηριζόμενες συσκευές. Αυτή τη στιγμή υποστηρίζονται 23 module οδηγών συσκευών SDR, καλύπτοντας ένα ευρύ φάσμα κατασκευαστών. Για παράδειγμα τα Ettus USRP του εργαστηρίου, χρησιμοποιούν το driver UHD και υποστηρίζονται από το module SoapyUHD.



Εικόνα 3.1 *Χαρακτηριστικά SoapySDR [44]*

Το SoapySDR είναι γραμμένο σε C++ και δεν έχει εξαρτήσεις (dependencies). Προσφέρει C ενθυλάκωση (wrapper) και συνδετικό κώδικα (bindings) σε γλώσσες, όπως η Python και άλλες. Τα bindings είναι συνδετικός κώδικας που επιτρέπει μέσα στο πρόγραμμα μιας γλώσσας, να καλείται ένα πρόγραμμα μιας άλλης γλώσσας. Το SoapySDR προσφέρει ακόμα ενθυλακώσεις (wrappers) για συμβατότητα με πλατφόρμες όπως το GNU Radio.

Το SoapySDR επιτρέπει ακόμα σε μια συσκευή να ελεγχθεί μέσω δικτύου, για απομακρυσμένη πρόσβαση. Για μεγαλύτερη απόδοση χρησιμοποιούνται αποθηκευτικοί χώροι χωρίς αντιγραφή δεδομένων (zero-copy buffers) κατά τη μεταφορά των δεδομένων από τη συσκευή. Μια χρήσιμη εντολή (από command line) είναι η SoapySDRUtil --find, που τυπώνει τις διαθέσιμες SDR συσκευές. Σε αυτή μπορούν να δοθούν και ορίσματα για να αναζητήσει μια συσκευή με συγκεκριμένα χαρακτηριστικά.

Δημιουργός του SoapySDR είναι η Pothosware. Η ίδια εταιρία κατασκευάζει και το Pothos framework. Το Pothos μοιάζει πολύ με το GNU Radio. Το Pothos επεξεργάζεται δεδομένα μέσω διασυνδεδεμένων τοπολογιών από τμήματα επεξεργασίας (block). Περιλαμβάνει και γραφικό περιβάλλον για στήσιμο της αλυσίδας επεξεργασίας και προβολή των γραφημάτων. Το Pothos φυσικά είναι συμβατό με το SoapySDR, για είσοδο από τις SDR συσκευές, μέσω του SoapySDR και επεξεργασία των σημάτων στο Pothos. Το SoapySDR προσφέρεται από την εταιρία ελεύθερα, με άδεια κατάλληλη για open source αλλά και εμπορικά προϊόντα.

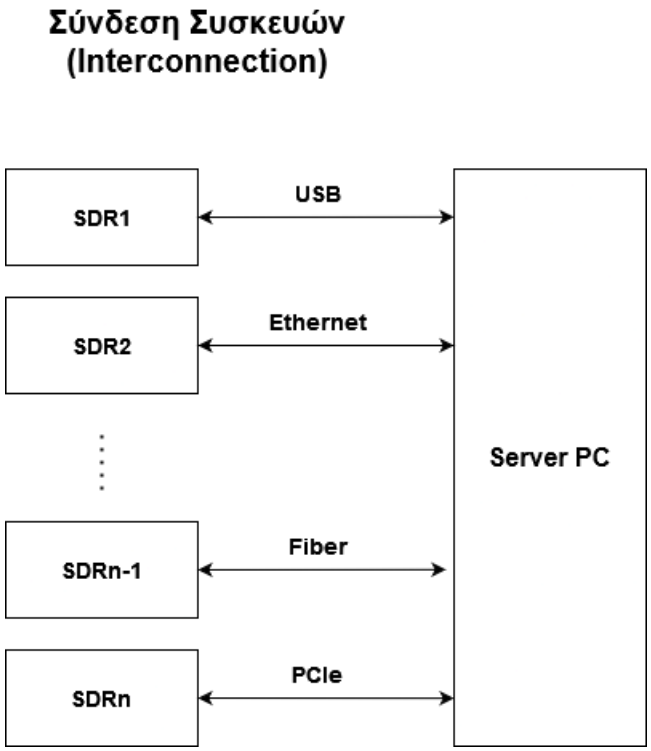
Σχήμα 3.3 *Ιεραρχία κώδικα οδηγών συσκευών*

3.2.4 Πολλαπλές συσκευές SDR

Το OpenWebRX υποστηρίζει πολλαπλές συνδεδεμένες συσκευές SDR ταυτόχρονα. Για κάθε συσκευή τρέχει και ένας connector. Δεν υπάρχει κάποιος θεμελιώδης περιορισμός στο πλήθος των συνδεδεμένων συσκευών. Σίγουρα απαιτούνται περισσότεροι υπολογιστικοί πόροι όταν όλες οι συσκευές είναι ενεργές.

Αυτό όμως που στην πράξη περιορίζει (bottleneck) είναι το μεγάλο εύρος ζώνης κατά την μεταφορά των δεδομένων από τη συσκευή στον φιλοξενών (host) υπολογιστή. Συγκεκριμένα, στην περίπτωση των USB συνδεδεμένων συσκευών, το συνολικό εύρος ζώνης του διαύλου του USB, είναι αυτό που περιορίζει.

Για συσκευές με μεγαλύτερο εύρος ζώνης (bandwidth) χρησιμοποιείται και δικτυακή σύνδεση μέσω καλωδίου Ethernet ή οπτικής ίνας ή και PCIe εξωτερικό καλώδιο.

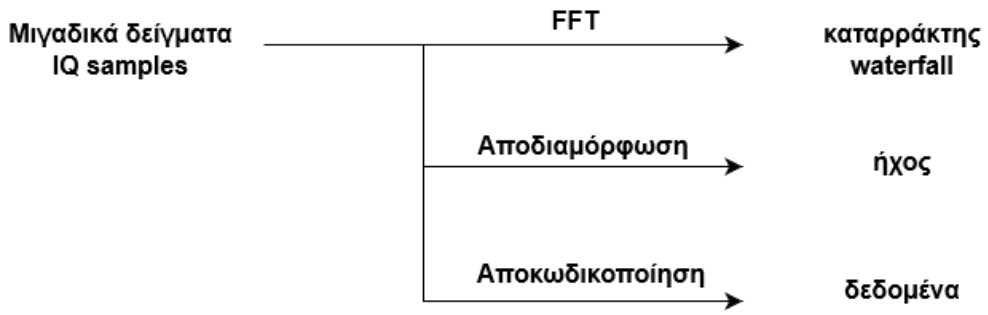


Σχήμα 3.3 *Σύνδεση συσκευών SDR στον υπολογιστή του εξυπηρετητή*

3.3 Ψηφιακή επεξεργασία σήματος στο OpenWebRX

3.3.1 Βιβλιοθήκη libcsdr

Τα μιγαδικά δείγματα (IQ samples) αφού έρθουν από την SDR συσκευή επεξεργάζονται ψηφιακά στον υπολογιστή, προκειμένου να παραχθεί το γράφημα καταρράκτη (waterfall) και να εξαχθεί ο ήχος στην αναλογική περίπτωση ή τα δεδομένα στην ψηφιακή περίπτωση. Αυτό γίνεται με αποδιαμόρφωση ή και με αποκωδικοποίηση αντίστοιχα. Οι λειτουργίες αυτές εκτελούνται αποκλειστικά σε λογισμικό.



Σχήμα 3.4 *DSP λειτουργίες*

Στο OpenWebRX η ψηφιακή επεξεργασία σήματος (DSP) υλοποιείται από την βιβλιοθήκη libcsdr, η οποία βρίσκεται στο [45]. Η libcsdr είναι μια βιβλιοθήκη γραμμένη σε C++, για ψηφιακή επεξεργασία σήματος SDR συσκευών. Φτιάχτηκε για το OpenWebRX αλλά μπορεί να χρησιμοποιηθεί και αλλού. Έχει σχεδιαστεί ώστε να αξιοποιεί την ρύθμιση του μεταγλωττιστή gcc, για αυτόματη επεξεργασία δεδομένων ως διανύσματα (auto-vectorization), πράγμα που την κάνει πιο αποδοτική.

Η libcsdr περιέχει συναρτήσεις που αρχικά χρησιμοποιούσε το πρόγραμμα csdr. Το csdr είναι ένα πρόγραμμα γραμμής εντολών (command line) με το οποίο φτιάχνονται γράφοι ροών (flowgraphs) για ψηφιακή επεξεργασία σήματος SDR. Με το csdr φτιάχνονται οι αλυσίδες αποδιαμόρφωσης (ξεχωριστή για κάθε χρήστη) χρησιμοποιώντας σωληνώσεις (pipes). Έτσι παράγεται για παράδειγμα ο ήχος, που στη συνέχεια στέλνεται στον χρήστη. Σε

κάποιες εντολές περιλαμβάνεται και μια πρόσθετη είσοδος (fifo) για έλεγχο κατά την λειτουργία. Για παράδειγμα, όταν κάποιος χρήστης αλλάξει τον σταθμό FM που ακούει με έναν άλλο εντός της ίδιας ορισμένης ζώνης συχνοτήτων (OpenWebRX profile) τότε θα αλλάξει η ζώνη διέλευσης του ζωνοπερατού φίλτρου δυναμικά.

Τον τελευταίο καιρό έγινε μια ανανέωση της δομής της αλυσίδας επεξεργασίας. Αυτό που άλλαξε είναι ότι δεν χρησιμοποιούνται πλέον ξεχωριστές διεργασίες, από εντολές γραμμής εντολών, όπως επίσης δεν χρησιμοποιούνται πλέον Linux σωληνώσεις για την μεταξύ τους επικοινωνία. Αλλά χρησιμοποιούνται συνεργαζόμενα νήματα, που εκτελούν τμήματα κώδικα (modules) την libcsdr και αναπαριστώνται ως διασυνδεδεμένοι εργάτες. Η επικοινωνία γίνεται μέσω αποθηκευτικών χώρων (buffers), που βρίσκονται μεταξύ διαδοχικών στην αλυσίδα εργατών.

Επίσης, δημιουργήθηκε συνδετικός κώδικας python (bindings) ώστε να καλείται και να δημιουργείται η αλυσίδα επεξεργασίας από την python και όχι με κάποια εντολή (command line). Δημιουργήθηκε και ένα API τμημάτων κώδικα (modules) για να επιτρέψει κατασκευή τμημάτων κώδικα επεξεργασίας από τρίτους. Τέλος τα τμήματα του κώδικα που υλοποιούν την ψηφιακή επεξεργασία παραμένουν σε C++, με το κάθε ένα να τρέχει σε ξεχωριστό νήμα για να αξιοποιείται η παραλληλία σε πολυπύρηνους επεξεργαστές και να επιτυγχάνεται υψηλή απόδοση. Υπολογίζεται πως με τις αλλαγές αυτές, η χρήση του επεξεργαστή μειώθηκε κατά 30%, σύμφωνα με αυτά που αναφέρονται στο [46]

Η αυτόματη επεξεργασία δεδομένων ως διανύσματα (auto-vectorization) είναι μια τεχνική παράλληλης επεξεργασίας που αυξάνει την απόδοση. Υποστηρίζεται από τους σύγχρονους μεταγλωττιστές (compilers), όπως ο gcc. Η τεχνική αυτή μετατρέπει εντολές του επεξεργαστή, που επεξεργάζονται ένα μόνο ζεύγος δεδομένων τη φορά (scalar) σε μια εντολή που επεξεργάζεται πολλά ζεύγη δεδομένων σε μια κλήση. Αυτό γίνεται ομαδοποιώντας τα δεδομένα σε διανύσματα. Στους σύγχρονους υπολογιστές υπάρχει υποστήριξη για αυτό, από το υλικό π.χ. SIMD. Έμφαση δίνεται στις επαναλήψεις (loops) όπου με αυτή την τεχνική μπορούν να επιταχυνθούν σημαντικά. Ο μεταγλωττιστής ελέγχει πρώτα για ύπαρξη εξαρτήσεων δεδομένων πριν κάνει την αλλαγή. Δεν μπορούν να επιταχυνθούν όλες οι επαναλήψεις. Δείτε περισσότερες λεπτομέρειες στο [47].

3.3.2 Διαχειριστής Αλυσίδας επεξεργασίας DspManager

Ο DspManager είναι μια σημαντική κλάση του OpenWebRX και βρίσκεται στο αρχείο owrx/dsp.py. Ο ρόλος του είναι η κατασκευή και η διαχείριση της αλυσίδας επεξεργασίας. Αυτό που γίνεται στο GNURadio με επιλογή και σύνδεση στοιχειωδών μονάδων επεξεργασίας, στο OpenWebRX γίνεται αυτόματα, σύμφωνα με τις επιλογές που κάνει ο χρήστης, κάνοντας απλώς μερικά κλικ στο πρόγραμμα περιήγησης του (browser). Στο τέλος στέλνεται στον χρήστη το αποτέλεσμα αυτής της επεξεργασίας.

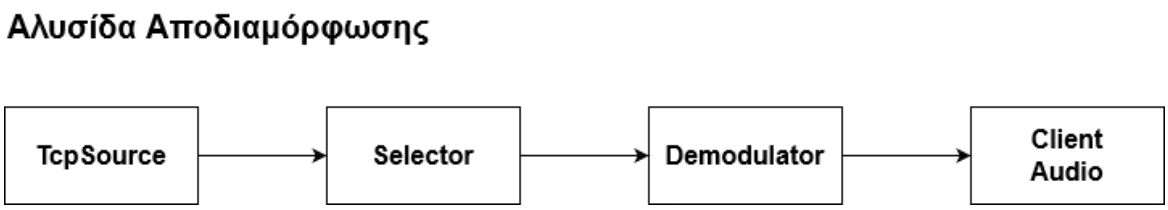
Ο DspManager καλείται από το αρχείο owrx/connection.py από την κλάση OpenWebRxReceiverClient, δηλαδή από την κλάση που αναπαριστά μια αμφίδρομης επικοινωνίας σύνδεση ενός χρήστη. Λαμβάνονται ως ορίσματα οι πληροφορίες της σύνδεσης και οι πληροφορίες που αναπαριστούν την επιλεγμένη συσκευή SDR.

Από την σύνδεση λαμβάνονται ως είσοδος οι τιμές ενός συνόλου παραμέτρων, των localProps και επικυρώνονται. Κάποιες από αυτές είναι: η επιλεγμένη διαμόρφωση, τα όρια του ζωνοπερατού φίλτρου επιλογής σήματος και το επίπεδο του σιγασμού (squelch). Αυτές οι παράμετροι δεν προωθούνται στη συσκευή SDR, αλλά χρησιμοποιούνται τοπικά στην αλυσίδα επεξεργασίας και μπορούν να μεταβάλλονται.

Από τις ρυθμίσεις της SDR συσκευής λαμβάνονται και άλλες παράμετροι όπως: ο ρυθμός δειγματοληψίας, η κεντρική συχνότητα κ.α. Αυτές οι παράμετροι είναι σταθερές κατά τη διάρκεια ζωής της αλυσίδας.

Η αλυσίδα επεξεργασίας αρχικοποιείται με την κατασκευή μιας ClientDemodulatorChain. Αυτή αποτελείται από:

1) πηγή δεδομένων TcpSource
2) επιλογέα σήματος Selector
3) αποδιαμορφωτή Demodulator ή και ψηφιακό αποκωδικοποιητή SecondaryDemodulator
4) αλυσίδα ήχου χρήστη ClientAudioChain



Σχήμα 3.5 *Συστατικά στοιχεία αλυσίδας αποδιαμόρφωσης*

Δημιουργούνται και συνδέονται μεταξύ τους οι κατάλληλοι εργάτες. Κάθε ένας από αυτούς υλοποιεί ένα τμήμα επεξεργασίας (module) της libcsdr βιβλιοθήκης και τρέχει ως ένα C++ νήμα. Μεταξύ των εργατών τοποθετούνται αποθηκευτικοί χώροι (buffers), έτσι ώστε εκεί που γράφεται η έξοδος του ενός, από εκεί να διαβάζεται η είσοδος του επόμενου. Τέλος ενώνονται και η αρχική είσοδος από το SDR μέσω ενός TcpSource και η τελική έξοδος προς τη σύνδεση του χρήστη.

Όταν ο χρήστης αλλάξει, για παράδειγμα το σήμα που ακούει, εντός της ίδιας ζώνης συχνοτήτων, τότε η αλυσίδα ενημερώνεται και τροποποιείται. Όταν όμως ο χρήστης αλλάξει SDR ή profile (βασικά ζώνη συχνοτήτων) η αλυσίδα διαγράφεται και ξαναδημιουργείται νέα.

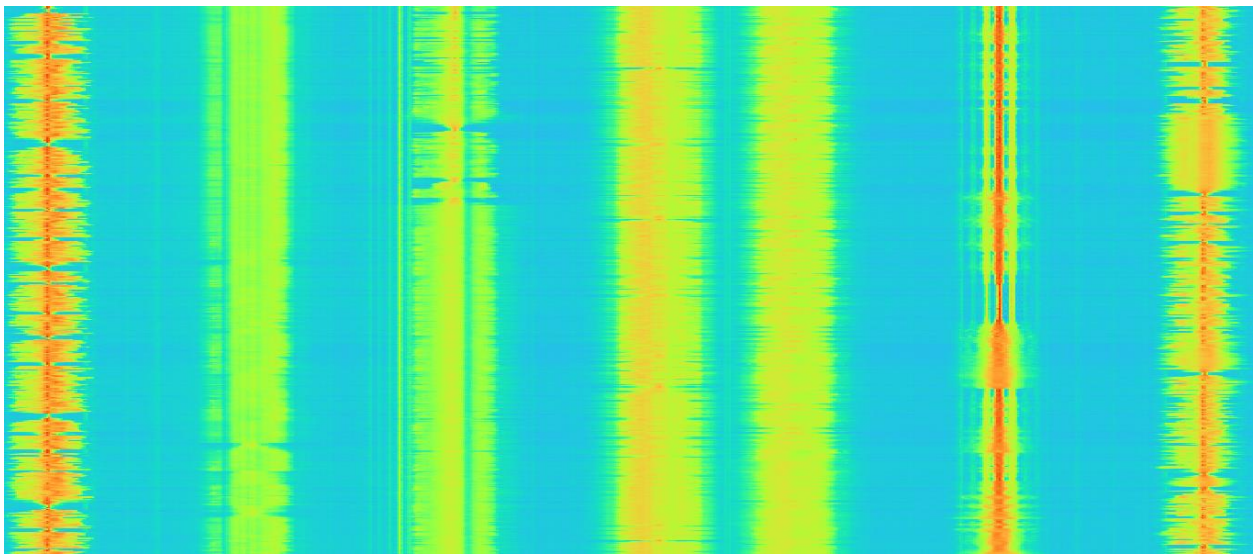
Συνδετικός κώδικας μεταξύ Python και C++

Ο τρόπος με τον οποίο ο DspManager από Python δημιουργεί μια C++ αλυσίδα νημάτων επεξεργασίας, είναι μέσω του pycsdr. Το pycsdr είναι συνδετικός κώδικας (bindings) μεταξύ της Python του OpenWebRX και της C++ βιβλιοθήκης libcsdr. Αποτελεί μια επέκταση της Python με C++ module. Έτσι το pycsdr καθιστά προσβάσιμες όλες τις συναρτήσεις της libcsdr για κλήση από την Python. Το module μεταγλωττίζεται πρώτα και μετά χρησιμοποιείται απλά με εισαγωγή (import). Δείτε περισσότερες λεπτομέρειες στο [48].

3.3.3 FFT για γράφημα καταρράκτη

Από τα αρχικά μιγαδικά δείγματα (IQ samples) υπολογίζεται ο διακριτός μετασχηματισμός Fourier (DFT). Αυτό γίνεται μέσω του αλγορίθμου γρήγορου μετασχηματισμού Fourier (FFT), ο οποίος είναι μια αποδοτική υλοποίηση του DFT. Μετατρέπει μια ακολουθία διακριτών στον χρόνο τιμών (μιγαδικών δειγμάτων στην περίπτωση μας) σε συνιστώσες διακριτών συχνοτήτων (φάσμα). Η πολυπλοκότητα του FFT είναι O(nlogn), σύμφωνα με το [49].

Στη συνέχεια το αποτέλεσμα του μετασχηματισμού απεικονίζεται στον χρήστη ως γράφημα καταρράκτη (waterfall) συναρτήσει του χρόνου. Ο FFT υπολογίζεται μια φορά για κάθε ενεργή συσκευή και στέλνεται ο ίδιος σε όλους τους χρήστες της.



Εικόνα 3.2 *Στιγμιότυπο οθόνης του OpenWebRX με γράφημα καταρράκτη*

Η αλυσίδα επεξεργασίας για τον υπολογισμό των δεδομένων του γραφήματος καταρράκτη αποτελείται από:

1) τον αλγόριθμο FFT
2) υπολογισμό μέσων όρων
3) αντιμετάθεση του διανύσματος του FFT
4) συμπίεση

Ο αλγόριθμος FFT που εφαρμόζεται στο OpenWebRX είναι ο FFTW έκδοση 3. Έχει γραφτεί στο MIT από τους Matteo Frigo και Steven G. Johnson και έχει ελεύθερη GPL άδεια. Είναι μια βιβλιοθήκη γραμμένη σε C και είναι φορητή, καθώς τρέχει σε οποιαδήποτε πλατφόρμα έχει ένα C μεταγλωττιστή. Υποστηρίζει SIMD εντολές για μεγαλύτερη ταχύτητα. Δεν είναι κατάλληλη όμως για εκτέλεση σε κάρτες γραφικών. Δείτε περισσότερες πληροφορίες στο [50].

Εκτός από τα δεδομένα του γραφήματος καταρράκτη, ο FFT χρησιμοποιείται και για τον υπολογισμό των ζωνοπερατών φίλτρων πεπερασμένης κρουστικής απόκρισης (FIR).

Η λήψη του μέσου όρου των τελευταίων μετασχηματισμών FFT γίνεται για να εμφανίζονται στον χρήστη πιο σταθερά και χρήσιμα αποτελέσματα. Η αντιμετάθεση του διανύσματος του FFT γίνεται για να έρθει η σταθερή (DC) συνιστώσα στο κέντρο του διανύσματος, κάτι που αποτελεί σύμβαση του λογισμικού.

Η συμπίεση υλοποιείται με Προσαρμοστική Διαφορική Παλμοκωδική Διαμόρφωση (ADPCM). Δηλαδή, κωδικοποιείται η διαφορά δυο διαδοχικών δειγμάτων με προσαρμοζόμενο βήμα κβαντισμού. Έτσι επιτυγχάνεται σημαντική μείωση του εύρους ζώνης που απαιτείται για τη μεταφορά των δεδομένων του γραφήματος καταρράκτη στον χρήστη. Δείτε περισσότερες λεπτομέρειες στο [51].

3.3.4 Αλυσίδα επεξεργασίας αναλογικού ήχου

Η αλυσίδα επεξεργασίας αναλογικών σημάτων, παράγει τον ήχο του σήματος που θα ακούσει ο χρήστης. Αποτελείται από 4 τμήματα, που βρίσκονται στον κώδικα της βιβλιοθήκης libcsdr.

1) TcpSource, πηγή δεδομένων, μέσω αυτής λαμβάνονται τα δεδομένα (μιγαδικά δείγματα). Εκτελεί τις εντολές (system calls) socket() και connect() προκειμένου να συνδεθεί με τον connector και να γίνει πελάτης του. Στη συνέχεια εκτελεί επαναληπτικά poll() και recv() για να λαμβάνει τα δεδομένα, τα οποία και προωθεί στο επόμενο τμήμα της αλυσίδας.

2) Selector, επιλογέας σήματος, φιλτράρει το λαμβανόμενο σήμα και περνάει μόνο το επιθυμητό. Προωθεί στο επόμενο στάδιο επίσης μιγαδικά δεδομένα. Οι εργάτες του είναι:

   Α) μετατόπιση συχνότητας (Shift) για κεντράρισμα στο επιλεγμένο σήμα.
   Β) φίλτρο πεπερασμένες κρουστικής απόκρισης (FIR), δηλαδή κρουστική απόκριση που μηδενίζεται μετά από κάποιο χρονικό διάστημα. Χρησιμοποιείται για μείωση των δειγμάτων (decimation).
   Γ) ζωνοπερατό φίλτρο με πεπερασμένη κρουστική απόκριση (FIR). Με ρυθμιζόμενα όρια και φιλτράρισμα μέσω μετασχηματισμών FFT και iFFT.
   Δ) λειτουργίες σιγασμού (squelch) και μέτρησης ισχύος (smeter).

3) Demodulator, αποδιαμορφωτής, υποστηρίζονται διάφορες κλάσεις αποδιαμορφωτών, όπως διαμόρφωση πλάτους AM, διαμόρφωση συχνότητας ευρείας ή στενής ζώνης WFM ή NFM, διαμόρφωση πλάτους άνω ή κάτω πλευρικής ζώνης USB ή LSB και διαμόρφωση συνεχούς κύματος CW. (Θα δοθούν παραδείγματα για την AM και την WFM παρακάτω).

4) ClientAudioChain, αλυσίδα ήχου χρήστη, αποτελεί την τελική επεξεργασία του ήχου πριν σταλθεί στον χρήστη. Περιλαμβάνει μετατροπή των μιγαδικών δεδομένων σε κινητής υποδιαστολής, επαναδειγματοληψία (resample) μέσω της <samplerate.h> και προσαρμοστική διαφορική παλμοκωδική διαμόρφωση (ADPCM) για συμπίεση και εξοικονόμηση εύρους ζώνης της σύνδεσης. Σημειώνεται πως στην ADPCM το κάθε δείγμα ήχου κωδικοποιείται σε 4 bit, σύμφωνα με το [52].

Παράδειγμα αλυσίδας επεξεργασίας AM ήχου

Η αλυσίδα επεξεργασίας της AM αποδιαμόρφωσης περιλαμβάνει:

1) AM αποδιαμορφωτή, που απλά υπολογίζει το μέτρο/απόλυτη τιμή των δειγμάτων της εισόδου.

2) φίλτρο σταθερής (DC) συνιστώσας, όπου στο [53] περιγράφονται η εξίσωση διαφορών, η συνάρτηση μεταφοράς, η απόκριση συχνότητας και φάσης, οι πόλοι, η κρουστική απόκριση και η υλοποίηση σε κώδικα.

3) αυτόματο έλεγχο κέρδους (AGC), ώστε μέσω ανάδρασης να προσαρμόζεται στις μεταβολές του πλάτους του σήματος εισόδου, έτσι ώστε όταν το πλάτος εισόδου μειώνεται, να αυξάνεται το κέρδος και το αντίστροφο. Έτσι διατηρείται ένα κατάλληλο πλάτος εξόδου. Για περισσότερες λεπτομέρειες δείτε το [17]

Παράδειγμα αλυσίδας επεξεργασίας FM ήχου

Η αλυσίδα επεξεργασίας της FM αποδιαμόρφωσης περιλαμβάνει:

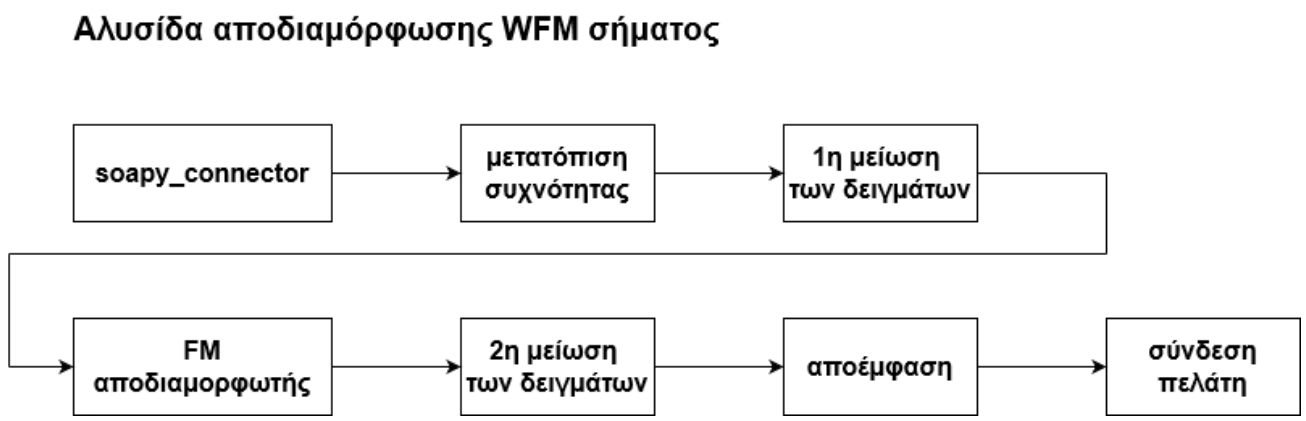
1) FM αποδιαμορφωτή, που χρησιμοποιεί τη γωνία των μιγαδικών δειγμάτων, υπολογίζει τη διαφορά φάσης δυο διαδοχικών, την προσαρμόζει εντός των ορίων [–π,π] και επιστρέφει σε αριθμό κινητής υποδιαστολής τη διαφορά φάσης δια του π. Δείτε περισσότερες λεπτομέρειες στο [54].

2) περιοριστή, για προσαρμογή του πλάτους εντός των μέγιστων ορίων.

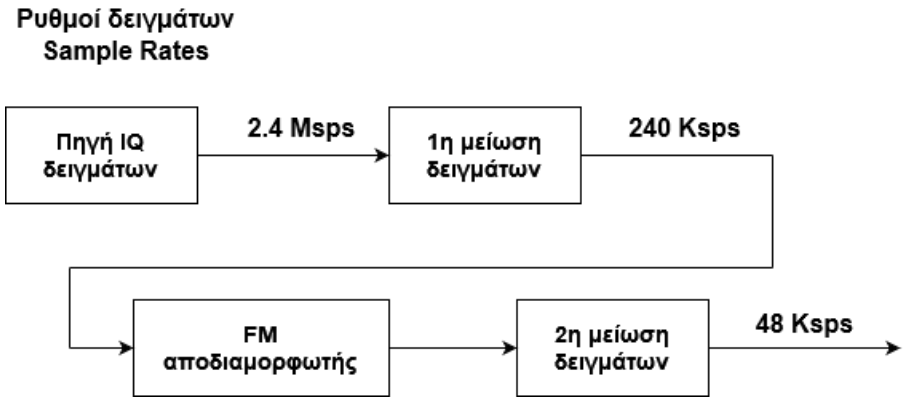
3) αποθηκευτικό χώρο και σημείο λήψης (tap) της ακατέργαστης ροής από τον αποκωδικοποιητή RDS, για εξαγωγή του μηνύματος κειμένου του FM σταθμού.

4) δεύτερο επίπεδο μείωσης των δειγμάτων (decimation), απαιτείται γιατί το WFM σήμα έχει μεγαλύτερο εύρος ζώνης από τον τελικό ήχο.

5) αποέμφαση, για ρύθμιση προς τα κάτω του πλάτους των υψηλότερων συχνοτήτων, διαδικασία αντίστροφη από αυτήν που γίνεται στον FM πομπό. Δείτε περισσότερες λεπτομέρειες στο [55].



Σχήμα 3.6 *Αλυσίδα αποδιαμόρφωσης WFM σήματος, με πηγή των δειγμάτων και προορισμό του ήχου.*



Σχήμα 3.7 *Μεταβολή του ρυθμού δειγμάτων στα διάφορα στάδια της WFM αποδιαμόρφωσης.* [56].

3.3.5 Ψηφιακοί Αποκωδικοποιητές

Το OpenWebRX υποστηρίζει πληθώρα ψηφιακών αποκωδικοποιητών. Στο αρχείο owrx/modes.py στην κλάση Modes φαίνονται όλοι οι υποστηριζόμενοι. Απαιτείται εγκατάσταση επιπλέον λογισμικού στον εξυπηρετητή για να γίνουν διαθέσιμοι. Με την επέκταση OpenWebRX+ υποστηρίζονται ακόμα περισσότεροι.

Με την ψηφιακή αποδιαμόρφωση μετατρέπονται τα δείγματα του σήματος σε δυαδικά ψηφία (bits). Με την αποκωδικοποίηση αυτά τα bits γίνονται χρήσιμα δεδομένα.

Η αποκωδικοποίηση γίνεται είτε απευθείας στα λαμβανόμενα δείγματα του σήματος είτε μετά από κάποια άλλη βασική αποδιαμόρφωση (π.χ. USB). Για κάθε πρωτόκολλο (mode) ρυθμίζεται αυτόματα και το κατάλληλο εύρος ζώνης του ζωνοπερατού φίλτρου.

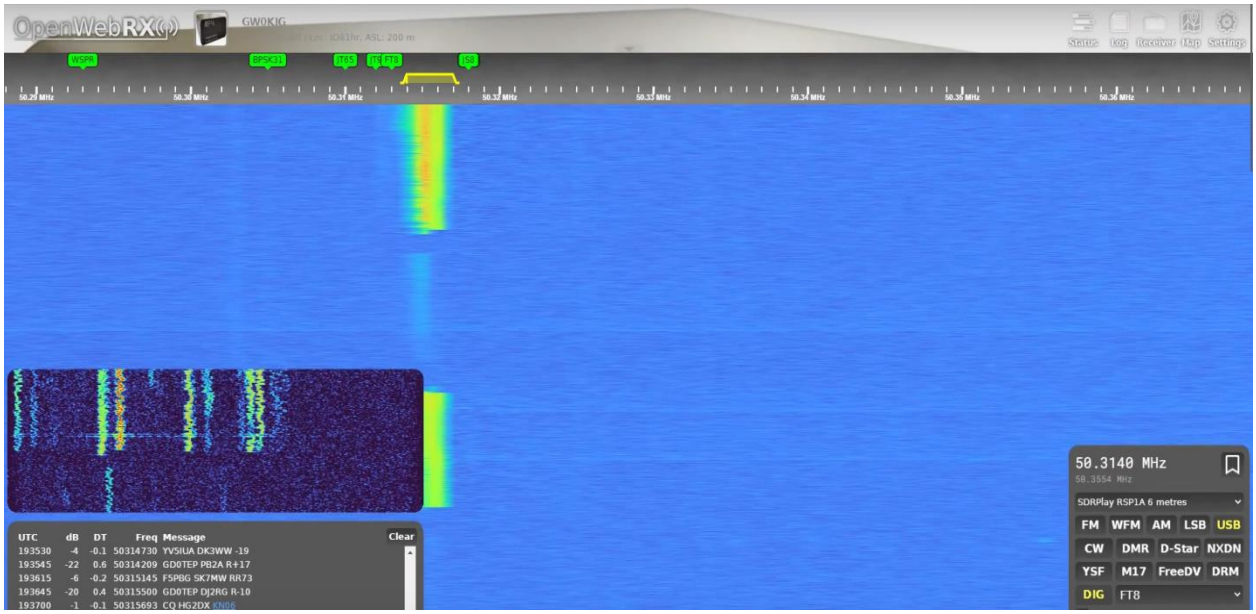
Υποστηρίζονται:

1) ψηφιακή φωνή

2) ψηφιακά δεδομένα

3) αναφορά ψηφιακών δεδομένων (reporting)

4) ψηφιακά δεδομένα αδύναμου σήματος (WSJT)



Εικόνα 3.3 *Στιγμιότυπο οθόνης από WSJT (FT8επαφές) στο OpenWebRX. Περισσότερα στα [57,58]*

3.4 Ο Εξυπηρετητής του OpenWebRX

3.4.1 Εξυπηρετητής

Το OpenWebRX διαμοιράζει το σήμα, που λαμβάνει μια τοπική κεραία μιας συσκευής SDR, σε όλο τον κόσμο, μέσω διαδικτύου. Για να το πετύχει αυτό χρησιμοποιεί έναν εξυπηρετητή (web server). Ο εξυπηρετητής τρέχει στον ίδιο υπολογιστή, όπου γίνεται η ψηφιακή επεξεργασία σήματος. Τα δεδομένα που παράγονται στέλνονται στον υπολογιστή του χρήστη, όπου και εμφανίζονται στον περιηγητή του. Σε αυτό το κεφάλαιο θα αναλυθεί το κομμάτι της εφαρμογής που είναι σχετικό με τις λειτουργίες του εξυπηρετητή.

Ο εξυπηρετητής απαντάει στα HTTP αιτήματα των χρηστών, στέλνοντας τους τα HTML, CSS και JavaScript αρχεία. Αυτά τα αρχεία καθορίζουν το γενικό περίγραμμα και τον τρόπο εμφάνισης της ιστοσελίδας της εφαρμογής από τον περιηγητή. Ωστόσο τα χρήσιμα δεδομένα, όπως το γράφημα καταρράκτη, ο ήχος, αλλά και οι επιλογές του χρήστη, ανταλλάσσονται μέσω του WebSocket. Το WebSocket είναι ένα πρωτόκολλο επικοινωνίας που υποστηρίζεται από τους περιηγητές και επιτρέπει αμφίδρομη επικοινωνία πελάτη – εξυπηρετητή.

Εκτός από τα δυο αυτά πρωτόκολλα που υποστηρίζονται, ο εξυπηρετητής εκτελεί και ένα σύνολο από άλλες λειτουργίες. Στο OpenWebRX υπάρχει πρόβλεψη για τον ρόλο του διαχειριστή, ώστε να κάνει ρυθμίσεις με εύκολο τρόπο μέσω web διεπαφής. Αυτές οι ρυθμίσεις επιτρέπονται μόνο μετά από εισαγωγή κωδικού (authentication). Επίσης, τα δεδομένα του προγράμματος οργανώνονται σε δομές και αποθηκεύονται στο σύστημα αρχείων του υπολογιστή. Τέλος, υπάρχουν και άλλες λειτουργίες, που θα περιγραφούν στη συνέχεια.

3.4.2 HTTP

Το HTTP (Πρωτόκολλο Μεταφοράς Υπερκειμένου) είναι το κύριο πρωτόκολλο επικοινωνίας που χρησιμοποιείται από τους περιηγητές, για την μεταφορά δεδομένων, μέσω διαδικτύου, μεταξύ εξυπηρετητή και πελάτη. Η επικοινωνία γίνεται με HTTP αιτήματα από

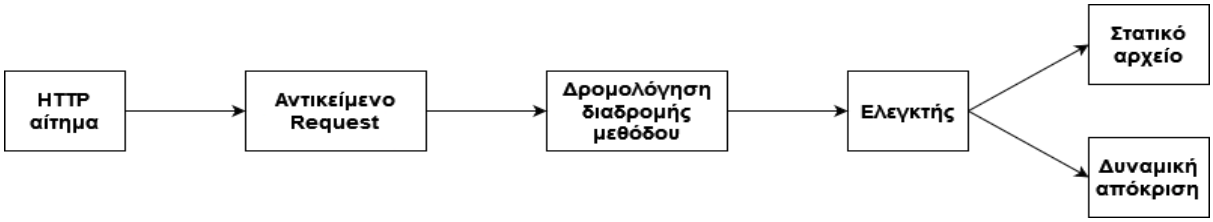
τον πελάτη και HTTP αποκρίσεις από τον εξυπηρετητή. Δείτε περισσότερες λεπτομέρειες για το HTTP στο [59].

Στο OpenWebRX περιλαμβάνεται HTTP εξυπηρετητής. Αυτός είναι γραμμένος σε Python και χρησιμοποιεί το τμήμα κώδικα (module) http.server. Βασίζεται σε υποδοχές (sockets) του πρωτοκόλλου TCP και υποστηρίζει ipv4 ή ipv6. Είναι πολυνηματικός και μπορεί να εξυπηρετήσει ταυτόχρονα πολλούς πελάτες. Το module http.server αποτελεί μια βάση και προσφέρει κλάσεις με μεταβλητές (instance variables), γνωρίσματα (class attributes) και μεθόδους (methods) που επεκτείνονται στη συνέχεια. Δείτε περισσότερες λεπτομέρειες στο [60].

Στο αρχείο http.py κωδικοποιείται η διαδικασία απόκρισης σε ένα HTTP αίτημα. Αρχικά, χτίζεται το αντικείμενο Request, το οποίο περιλαμβάνει πληροφορίες για το αίτημα όπως η διεύθυνση (url), το μονοπάτι (path), η συμβολοσειρά του ερωτήματος (query string), η μέθοδος, οι επικεφαλίδες και τα cookies. Στη συνέχεια με βάση την διαδρομή και τη μέθοδο, γίνεται η δρομολόγηση του αιτήματος και επιλέγεται ο χειρισμός από τον κατάλληλο ελεγκτή (controller). Για να γίνει αυτό έχουν δηλωθεί όλες οι διαθέσιμες διαδρομές και μέθοδοι και η αντιστοιχία τους με τους ελεγκτές.

Ο ελεγκτής (controller) μπορεί να επιστρέψει στον πελάτη ένα στατικό αρχείο (html, css ή js). Για εξοικονόμηση του εύρους ζώνης της σύνδεσης χρησιμοποιείται συμπίεση Gzip και η επικεφαλίδα If-Modified-Since για έλεγχο και χρήση της προσωρινής αποθήκευσης. Μαζί με το κύριο αρχείο html στέλνεται και μια επικεφαλίδα html, που περιλαμβάνει καθορισμένες μεταβλητές, σχετικές με τον δέκτη.

Εναλλακτικά, στην περίπτωση των ρυθμίσεων, υποστηρίζονται λειτουργίες δημιουργίας, ανάγνωσης, τροποποίησης και διαγραφής δυναμικών δεδομένων, που βρίσκονται αποθηκευμένα στη μνήμη της εφαρμογής και στο σύστημα αρχείων. Για παράδειγμα, επιστρέφεται μια λίστα με τις καταχωρημένες SDR συσκευές.



Σχήμα 3.8 *Πορεία HTTP αιτήματος*

3.4.3 WebSocket

Το WebSocket είναι ένα πρωτόκολλο αμφίδρομης επικοινωνίας μεταξύ πελάτη και εξυπηρετητή. Υποστηρίζεται από τους μοντέρνους περιηγητές. Η επικοινωνία πραγματοποιείται πάνω από μια TCP σύνδεση. Είναι συμβατό με το HTTP. Σε αντίθεση με το HTTP δεν χρησιμοποιείται το μοντέλο αιτήματος – απόκρισης και γενικώς δεν γίνονται επαναλαμβανόμενα ερωτήματα (polling). Είναι κατάλληλο για ζωντανού χρόνου (real-time) εφαρμογές. Δείτε περισσότερες λεπτομέρειες στο [61].

Τα βασικά αρχεία για το WebSocket στον εξυπηρετητή του OpenWebRX είναι τα ακόλουθα:

1) websocket.py, όπου περιγράφεται το πρωτόκολλο του WebSocket.
2) connection.py, όπου περιγράφεται η WebSocket σύνδεση ενός πελάτη.
3) websocket controller, όπου φτιάχνεται μια νέα WebSocket σύνδεση και καλείται ο χειρισμός της (μέθοδος handle).

Αρχικά γίνεται αναβάθμιση της σύνδεσης από HTTP σε WebSocket. Ακολουθούν μηνύματα χειραψίας. Στη συνέχεια η σύνδεση μπαίνει σε μια επαναληπτική διαδικασία διαβάσματος, διαβάζοντας μηνύματα κειμένου ή δυαδικά. Επίσης, στέλνει και απαντάει σε δοκιμαστικά μηνύματα ping και pong, περιοδικά, για να βεβαιώνεται ότι διατηρείται η σύνδεση ζωντανή.

Στον πελάτη στέλνονται μηνύματα που τα δεδομένα τους μπορεί να είναι:

1) json κωδικοποιημένα για λεξικά (dictionaries)
2) κωδικοποιημένο UTF-8 κείμενο
3) δυαδικά

Πριν την αποστολή πρέπει να έχει ληφθεί ένα κλείδωμα που εξασφαλίζει ότι τα νήματα στέλνουν ένα - ένα.

Μετά τη χειραψία δημιουργείται μια WebSocket σύνδεση πελάτη.

Μέσω αυτής λαμβάνονται:

1) παράμετροι σχετικοί με την αλυσίδα επεξεργασίας του σήματος (DSP)
2) η επιλεγμένη συσκευή SDR
3) η επιλεγμένη ζώνη συχνοτήτων (profile).

Ενώ στέλνονται στον πελάτη:

1) δεδομένα φάσματος,
2) ήχος ή γενικότερα τα δεδομένα από την αλυσίδα επεξεργασίας
3) η μέτρηση ισχύος
4) μέτρηση χρήσης υπολογιστικής μονάδας (cpu usage)
5) ο συνολικός αριθμός των ενεργών πελατών
6) οι τρέχουσες ρυθμίσεις του SDR και της ζώνης συχνοτήτων (profile), ρυθμίσεις του γραφήματος καταρράκτη και της συμπίεσης
7) οι διαθέσιμες ζώνες συχνοτήτων
8) οι υποστηριζόμενες συσκευές και η διαθεσιμότητα επιπλέον λογισμικού
9) οι αποθηκευμένες συχνότητες γνωστών σημάτων
10) μηνύματα καταγραφής και σφάλματος
11) υποστηριζόμενες αποδιαμορφώσεις – αποκωδικοποιήσεις

3.4.4 Ρυθμίσεις

Μέσω των σελίδων των ρυθμίσεων (settings), ο διαχειριστής μπορεί με ευκολία να ρυθμίσει ολόκληρη την εφαρμογή. Μπορεί να καταχωρίσει συσκευές SDR, να ορίσει τις ζώνες συχνοτήτων που θα λειτουργούν και άλλες εξειδικευμένες ιδιότητες όπως π.χ. το χειροκίνητο κέρδος (manual gain) ή ο αυτόματος έλεγχος κέρδους (AGC) του SDR.

Επίσης, στις γενικές ρυθμίσεις μπορεί να ανεβάσει μια φωτογραφία του δέκτη και να προσθέσει πληροφορίες όπως η τοποθεσία και το αναγνωριστικό (callsign). Ρυθμίζονται ακόμα παράμετροι του γραφήματος καταρράκτη και η συμπίεση.

Πρόσθετες ρυθμίσεις αφορούν την καταχώρηση γνωστών σημάτων (bookmarks), την επισκόπηση υποστηριζόμενων συσκευών και εγκατεστημένων λογισμικών. Επιπλέον υπάρχουν ρυθμίσεις αποκωδικοποίησης και ρυθμίσεις παρασκηνιακών λειτουργιών.

3.4.5 Αυθεντικοποίηση

Πραγματοποιείται αυθεντικοποίηση (authentication) κατά την είσοδο στις ρυθμίσεις της ιστοσελίδας. Έτσι εξασφαλίζεται ότι ο διαχειριστής είναι ο μόνος που έχει την δυνατότητα να ρυθμίσει την εφαρμογή.

Δημιουργείται πρώτα μέσω γραμμής εντολών ένα ζεύγος ονόματος χρήστη – κωδικού και αποθηκεύεται στο αρχείο χρηστών. Στη συνέχεια όταν ο διαχειριστής εισάγει το σωστό κωδικό του στην ιστοσελίδα, αποθηκεύεται στον περιηγητή του ένα cookie. Σε κάθε πρόσβαση σελίδας ρυθμίσεων ελέγχεται αν υπάρχει αυτό το cookie και αν αντιστοιχεί σε ενεργή συνεδρία (session) του δεδομένου χρήστη.

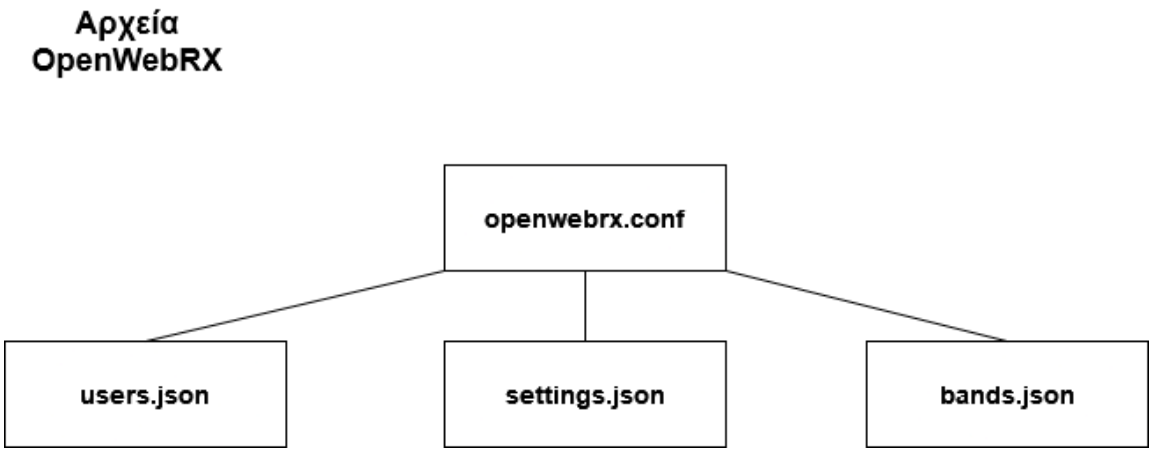
Αυτό υλοποιείται με ένα Mixin, δηλαδή μια κλάση που προσθέτει συμπληρωματική λειτουργία (αυθεντικοποίηση) σε πολλές άλλες κλάσεις (των ρυθμίσεων), μέσω πολλαπλής κληρονομικότητας. Δείτε περισσότερες λεπτομέρειες στο [62].

3.4.6 Δεδομένα Προγράμματος

Τα δεδομένα που προκύπτουν από τις ρυθμίσεις, αποθηκεύονται αρχικά στη μνήμη του προγράμματος, στην κλάση Config (και συγκεκριμένα στο DynamicConfig) και στη συνέχεια με json μορφή στο αρχείο ρυθμίσεων.

Υπάρχει και το CoreConfig όπου αποθηκεύονται τα κυριότερα προκαθορισμένα δεδομένα όπως π.χ. η θέση του καταλόγου με τα αρχεία δεδομένων και η θύρα που ακούει ο εξυπηρετητής. Αυτά σώζονται στο αρχείο openwebrx.conf.

Στο αρχείο bands.json, βρίσκεται ένας κατάλογος συχνοτήτων - διαμορφώσεων γνωστών σημάτων (bookmarks). Αυτά εμφανίζονται στον χρήστη, για ταχύτερη επιλογή σήματος.



Σχήμα 3.9 *Χρησιμοποιούμενα αρχεία*

3.4.7 Άλλες λειτουργίες

Λαμβάνονται επίσης, περιοδικά μετρήσεις χρήσης της υπολογιστικής μονάδας (cpu usage) και ο συνολικός αριθμός των ενεργών πελατών και εμφανίζονται στην οθόνη του χρήστη. Σχεδιάζεται ακόμα, χάρτης με σημάδια και δεδομένα από αποκωδικοποιήσεις. Προαιρετικά, μπορεί να λειτουργεί αυτόματη παρασκηνιακή αποκωδικοποίηση για αξιοποίηση ανενεργών πόρων. Αυτό συνδυάζεται και με αναφορά σε δίκτυο τρίτων υπηρεσιών π.χ. για το APRS. Δείτε περισσότερες λεπτομέρειες για το APRS στο [63]. Ακόμα, υπάρχει δυνατότητα για ρυθμίσεις και μέσω γραμμής εντολών. Τέλος, γίνεται καταγραφή των σημαντικότερων ενδείξεων λειτουργίας στην τυπική έξοδο (stdout).

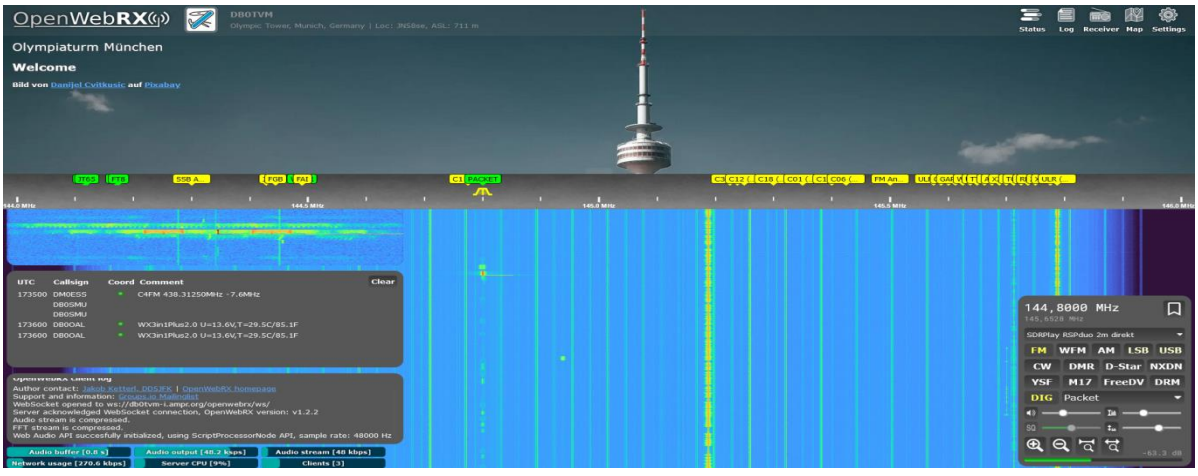
3.5 Η πλευρά του πελάτη

3.5.1 Περιγραφή της διεπαφής χρήστη

Η διεπαφή του χρήστη με το OpenWebRX είναι μια ιστοσελίδα που εμφανίζεται στον περιηγητή του. Εκεί ο χρήστης επιλέγει κάποιο σήμα και το ακούει. Μόλις πατηθεί το κουμπί αναπαραγωγής (play) ακούγεται ο αποδιαμορφωμένος ήχος του σήματος.

Η σελίδα περιλαμβάνει ένα μεγάλο γράφημα καταρράκτη που ανανεώνεται συνεχώς. Πατώντας πάνω σε αυτό μετακινείται η συχνότητα. Εναλλακτικά η συχνότητα πληκτρολογείται στον πίνακα ελέγχου. Από εκεί επιλέγεται και η συσκευή SDR και η ζώνη συχνοτήτων. Εκεί βρίσκονται και κουμπιά επιλογής για κάθε διαθέσιμη διαμόρφωση, ένδειξη ισχύος, ρύθμιση σιγασμού (squelch) και ρυθμίσεις του γραφήματος καταρράκτη.

Υπάρχει ακόμα μια περιοχή όπου εμφανίζονται τα μηνύματα καταγραφής πληροφοριών ή σφάλματος. Υπάρχουν επίσης, μπάρες ένδειξης της τρέχουσας κατάστασης: της χρήσης της υπολογιστικής μονάδας, του ήχου, του αριθμού των πελατών και της χρήσης δικτύου. Επιπλέον στην περίπτωση των ψηφιακών αποκωδικοποιητών εμφανίζεται και δεύτερο γράφημα καταρράκτη και τα παραγόμενα δεδομένα. Τέλος, στο πάνω μέρος της οθόνης εμφανίζεται μια επικεφαλίδα, που προεκτείνεται όταν πατηθεί και περιέχει μια φωτογραφία και πληροφορίες για τον δέκτη.



Εικόνα 3.4 *Στιγμιότυπο οθόνης με OpenWebRX UI*

3.5.2 Αρχεία Υπερκειμένου

Στον φάκελο htdocs περιέχονται όλα τα στατικά αρχεία που καθορίζουν την εμφάνιση της ιστοσελίδας. Αυτά είναι HTML, CSS και JavaScript αρχεία, καθώς και εικόνες, γραφικά και γραμματοσειρές. Περιλαμβάνονται και βιβλιοθήκες όπως για παράδειγμα η css, js bootstrap και η js jquery. Όλες αποθηκεύονται στον εξυπηρετητή. Το πρώτο αρχείο που στέλνεται είναι το index.html.

3.5.3 Openwebrx.js

Αυτό το JavaScript αρχείο, αρχικοποιεί την εμφάνιση της σελίδας και ρυθμίζει τη λειτουργία της. Συγκεκριμένα, αρχικοποιεί το γράφημα καταρράκτη, τον πίνακα ελέγχου και τον ήχο. Επίσης, ανοίγει μια WebSocket σύνδεση με τον εξυπηρετητή και χειρίζεται τη λήψη των δεδομένων μέσω αυτής. Ακόμα, χειρίζεται την αλλαγή SDR συσκευής και ζώνης συχνοτήτων.

3.5.4 HTML5

Δυο τεχνολογίες μέρος της HTML5 που υποστηρίζονται από τους περιηγητές και χρησιμοποιεί το OpenWebRX είναι το Canvas και το Web Audio API.

Με το Canvas δίνεται ένας τρόπος να ζωγραφιστούν γραφικά μέσω JavaScript και να εισαχθούν σε μια ετικέτα HTML. Δείτε περισσότερες λεπτομέρειες στο [64].

Στο OpenWebRX το Canvas χρησιμοποιείται για εμφάνιση των δεδομένων του γραφήματος καταρράκτη. Ειδικότερα, εφαρμόζεται μεταχείριση των εικονοστοιχείων (pixel) με το αντικείμενο ImageData. Το γράφημα φτιάχνεται γραμμή προς γραμμή και μετακινείται προς τα κάτω. Υποστηρίζεται και εστίαση (zoom) στον καταρράκτη.

Με το Web Audio API γίνεται η διαχείριση του ήχου στον περιηγητή. Στο OpenWebRX πραγματοποιείται αποσυμπίεση των ADPCM δεδομένων που λαμβάνονται, επαναδειγματοληψία και προσωρινή αποθήκευση (buffering) πριν ακουστούν.

3.5.5 Χάρτης

Το OpenWebRX υποστηρίζει εμφάνιση δεδομένων σε χάρτη. Ο χάρτης ανοίγει σε ξεχωριστή σελίδα. Βασίζεται στους χάρτες της Google και απαιτείται κλειδί (Google Maps API key) για να εμφανιστεί. Πάνω στον χάρτη σημειώνονται σύμβολα και πληροφορίες, όπως για παράδειγμα στο APRS.

Εικόνα 3.5 *Στιγμιότυπο οθόνης με APRS στον χάρτη του OpenWebRX*

3.5.6 Ρυθμίσεις

Τέλος, υπάρχουν και οι σελίδες ρυθμίσεων, όπου ο διαχειριστής συμπληρώνει δεδομένα σε φόρμες ή του παρουσιάζονται τα καταχωρημένα δεδομένα. Εκεί είναι που χρησιμοποιείται η βιβλιοθήκη css και js του bootstrap.

4. Load Balancing OpenWebRX

4.1 Εισαγωγή στην ισοστάθμιση Φορτίου (Load Balancing)

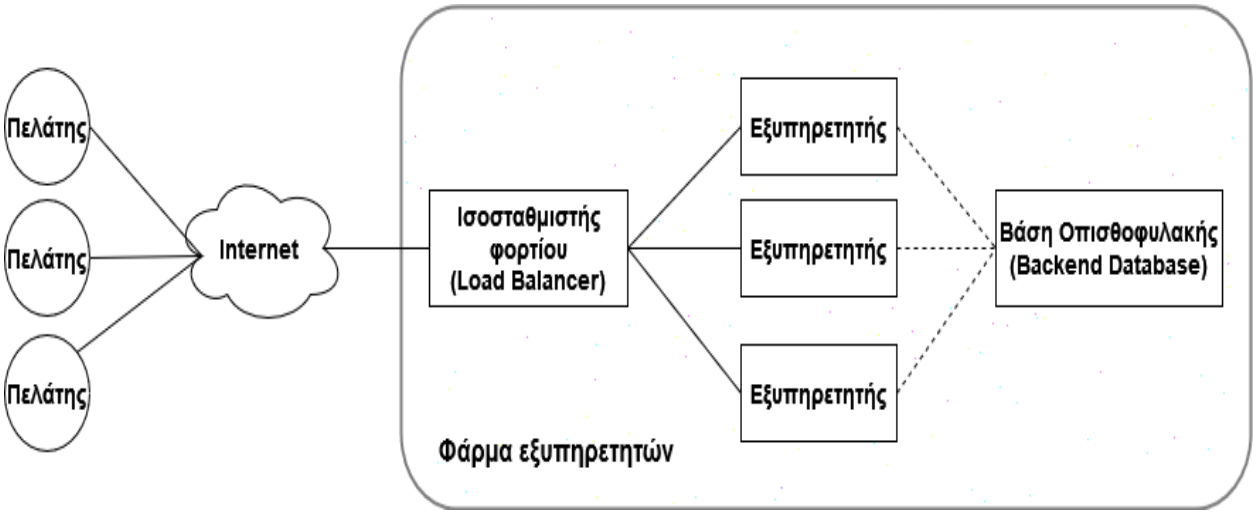
Όπως περιγράφεται στο [65] μια εφαρμογή – ιστοσελίδα για να είναι διαθέσιμη μέσω διαδικτύου χρειάζεται έναν εξυπηρετητή και μια ζεύξη σύνδεσης, επαρκούς εύρους ζώνης. Εάν έχει πολλούς πελάτες ή είναι υπολογιστικά απαιτητική, ανεξάρτητα του εύρους ζώνης που απαιτείται, πρέπει να χειριστεί ένα μεγάλο υπολογιστικό φορτίο. Η προτιμώμενη λύση είναι να χρησιμοποιηθούν πολλοί υπολογιστές μαζί, ως μια φάρμα εξυπηρετητών. Μια φάρμα εξυπηρετητών είναι μια συστοιχία υπολογιστών που λειτουργούν ενιαία, με αυξημένες συνολικές επιδόσεις.

Απαραίτητο στοιχείο μιας φάρμας εξυπηρετητών είναι ο ισοσταθμιστής φορτίου (load balancer). Ο ισοσταθμιστής φορτίου κατανέμει αποδοτικά τις αιτήσεις στους υπολογιστές, ισοσταθμίζοντας το φορτίο τους. Η ισοστάθμιση μπορεί να γίνεται με βάση το τρέχον φορτίο τους.

Κύριο πεδίο εφαρμογής της ισοστάθμισης φορτίου σύμφωνα με τα [65,66] είναι οι φάρμες εξυπηρετητών, από τις οποίες αποτελούνται τα κέντρα δεδομένων (data centers). Εκεί φιλοξενούνται εφαρμογές νέφους (cloud). Στον δικτυακό εξοπλισμό των κέντρων δεδομένων περιλαμβάνονται εκτός από μεταγωγείς και δρομολογητές και ισοσταθμιστές φορτίου.

Όλοι οι υπολογιστές που εξυπηρετούν μια εφαρμογή συνήθως αποτελούν μια λογική οντότητα. Μπορεί να έχουν μια δημόσια διεύθυνση διαδικτύου (IP) ή και περισσότερες. Συνήθως γίνεται η υπόθεση ότι οποιοσδήποτε εξυπηρετητής μπορεί να εξυπηρετήσει οποιονδήποτε πελάτη. Για να συμβεί αυτό, πρέπει όλοι οι εξυπηρετητές να είναι συνδεδεμένοι σε μια κοινή βάση οπισθοφυλακής ή να έχουν ένα αντίγραφο της ιστοσελίδας.

Ακολουθεί σχήμα που δείχνει την αρχιτεκτονική μιας φάρμας εξυπηρετητών.



Σχήμα 4.1 *Φάρμα εξυπηρετητών, από το [66]*

Η αίτηση ενός πελάτη θα προωθηθεί από τον ισοσταθμιστή φορτίου σε ένα υπολογιστή που χειρίζεται την εφαρμογή. Η απόκριση μπορεί να περάσει μέσα από τον ισοσταθμιστή φορτίου και να επανεκπεμφθεί προς τον πελάτη. Εναλλακτικά, μπορεί να υπάρχει απευθείας σύνδεση πελάτη – εξυπηρετητή. Στην περίπτωση που περνάει όλη η κίνηση μέσα από τον ισοσταθμιστή φορτίου, πραγματοποιείται και μετάφραση της εξωτερικής διεύθυνσης IP στην εσωτερική IP του υπολογιστή (NAT). Έτσι δεν υπάρχει άμεση επαφή πελάτη – εξυπηρετητή και αποκρύπτεται η εσωτερική δομή του δικτύου. Επιτυγχάνεται με αυτόν τον τρόπο, αυξημένη ασφάλεια, σύμφωνα με το [65].

4.1.1 Τρόπος λειτουργίας του ισοσταθμιστή φορτίου

Όπως περιγράφεται στα [66,67] ο ισοσταθμιστής φορτίου (load balancer) μπορεί να λειτουργεί είτε στο επίπεδο 4 (μεταφοράς) του μοντέλου αναφοράς OSI, είτε στο επίπεδο 7 (εφαρμογής). Στην πρώτη περίπτωση διαβάζει και χρησιμοποιεί εκτός από την επικεφαλίδα του IP και τη θύρα προορισμού. Αλλιώς, εξετάζει όλες τις επικεφαλίδες μέχρι και την HTTP, όπου μπορεί για παράδειγμα να διαβάσει το ζητούμενο url, τα cookies και πληροφορίες του SSL.

Όταν λειτουργεί στο επίπεδο 4, είναι πιο γρήγορος, αλλά χάνει σημαντική πληροφορία που θα τον βοηθούσε να κάνει καλύτερη κατανομή του φορτίου στου υπολογιστές. Ενώ όταν λειτουργεί στο επίπεδο 7, μπορεί για παράδειγμα να ανιχνεύσει μια

σύνδεση, μέσω των cookies και να στείλει όλες τις αιτήσεις ενός χρήστη στον ίδιο εξυπηρετητή. Έτσι κερδίζει σε ταχύτητα επεξεργασίας, μιας και τα δεδομένα του χρήστη θα προσκομιστούν μια φορά και μετά θα αποθηκευτούν στην προσωρινή μνήμη. Υποστηρίζεται επίσης εκφόρτωση του υπολογισμού του πρωτοκόλλου ασφαλείας SSL (SSL offloading), κάτι που απλοποιεί και επιταχύνει τη μετέπειτα επεξεργασία των εξυπηρετητών.

Σύμφωνα με τα [65,66] μια τέτοια συσκευή, σαν τον ισοσταθμιστή φορτίου, που εισάγεται στο δίκτυο και περιλαμβάνει επιπρόσθετες λειτουργίες σε σχέση με το δρομολογητή και εξετάζει επικεφαλίδες ανώτερων επιπέδων, ονομάζεται ενδιάμεσο κουτί (middlebox). Άλλα παραδείγματα είναι: το κουτί NAT για εξοικονόμηση διευθύνσεων, τα Firewalls για ασφάλεια, κουτιά για βελτίωση της απόδοσης με συμπίεση ή και προσωρινή αποθήκευση (caching). Βέβαια όλα αυτά παραβιάζουν τη σχεδιαστική αρχή των διαστρωματωμένων πρωτοκόλλων, όπου κάθε επίπεδο μπορεί να χρησιμοποιεί για λόγους ελέγχου μόνο την επικεφαλίδα του και όχι να κρυφοκοιτάζει πληροφορίες ανώτερων επιπέδων. Η αρχή προτείνεται, γιατί διαφορετικά, μια αλλαγή σε ανώτερο επίπεδο μπορεί να οδηγήσει σε κατάρρευση ολόκληρο το σύστημα. Πάντως τα ενδιάμεσα κουτιά είναι χρήσιμα στην πράξη και γι' αυτό υπάρχουν.

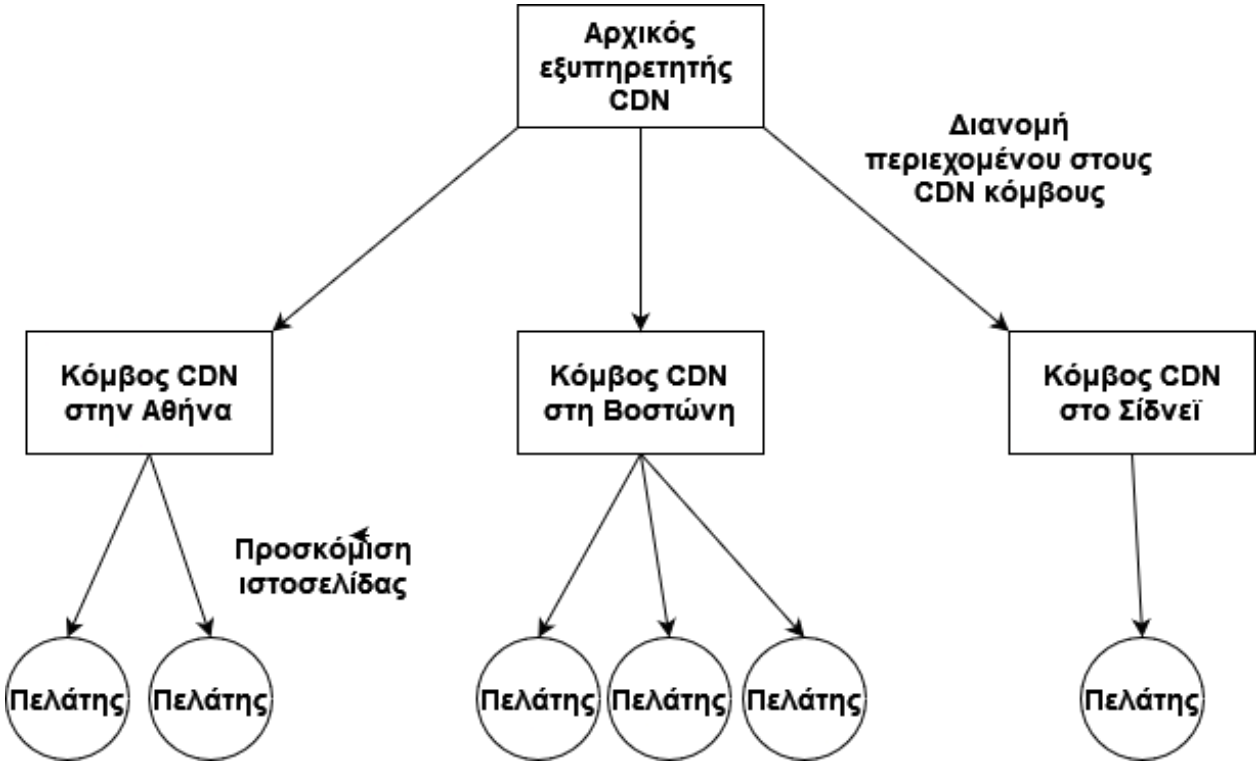
Εναλλακτικά, η κατανομή του φορτίου γίνεται χωρίς να μεσολαβεί κάποια ενδιάμεση συσκευή, αλλά μέσω του συστήματος ονοματοδοσίας διαδικτύου (DNS). Εκεί ο αρμόδιος εξυπηρετητής ονομάτων (authoritative name server) επιστρέφει μια εναλλασσόμενη λίστα (με διαφορετική σειρά κάθε φορά), με εξυπηρετητές που χειρίζονται την εφαρμογή. Έτσι οι πελάτες κατευθύνονται σε διαφορετικούς εξυπηρετητές, ανάλογα με το ποιος είναι πρώτος στη λίστα τους. Η μέθοδος ανακατεύθυνσης μέσω DNS εφαρμόζεται κατά κόρον και παρακάτω, στα δίκτυα διανομής περιεχομένου (CDN).

4.1.2 Κατανομή φορτίου σε δίκτυα διανομής περιεχομένου

Ακόμα ένας τρόπος κατανομής του φορτίου, είναι η γεωγραφική κατανομή του σε κέντρα δεδομένων σε όλο τον κόσμο. Αυτό εφαρμόζεται στα δίκτυα διανομής περιεχομένου (CDN). Τα CDN εγκαθιστούν κέντρα δεδομένων μέσα σε παρόχους διαδικτύου (ISP) ή στα σημεία συνένωσης τους τα χρησιμοποιούν για να εξυπηρετούν εφαρμογές. Και επειδή οι εξυπηρετητές βρίσκονται πολύ πιο κοντά στον χρήστη, ο χρόνος απόκρισης είναι μειωμένος.

Η κατεύθυνση του πελάτη στο προτιμότερο (συνήθως το κοντινότερο) κέντρο δεδομένων γίνεται μέσω του συστήματος ονοματοδοσίας διαδικτύου (DNS). Συγκεκριμένα, εξετάζεται η IP διεύθυνση του χρήστη και ανάλογα με την απόσταση αλλά και τη χωρητικότητα της σύνδεσης του με το κέντρο δεδομένων, επιλέγεται το καταλληλότερο και ανακατευθύνεται εκεί.

Όπως περιγράφεται στο [66] στα δίκτυα διανομής περιεχομένου, το περιεχόμενο της ιστοσελίδας μοιράζεται από τον αρχικό εξυπηρετητή στα τοπικά κέντρα δεδομένων, σε τοπικούς εξυπηρετητές της εφαρμογής. Έτσι δημιουργείται ένα δέντρο διανομής. Κατά την εξυπηρέτηση των πελατών, το μεγαλύτερο μέρος της κίνησης μεταφέρεται από το CDN, έτσι ο αρχικός εξυπηρετητής δεν υπερφορτώνεται. Τελικά, αυξάνεται η διεκπεραιωτική ικανότητα της εφαρμογής και ανταπεξέρχεται ακόμα και σε απότομα ξεσπάσματα κίνησης. Επίσης, εξοικονομείται εύρος ζώνης συνολικά στο δίκτυο, γιατί η κίνηση περνάει μια φορά από κάθε τμήμα του.



Σχήμα 4.2 *Δέντρο διανομής CDN [66]*

Στο παραπάνω σχήμα φαίνεται ένα CDN με κόμβους σε τρία μέρη στον κόσμο. Ο κάθε πελάτης ανάλογα με την τοποθεσία του, εξυπηρετείται από το κοντινότερο σε αυτόν κέντρο.

Συμπερασματικά, στα CDN η ισοστάθμιση φορτίου εκτελείται σε 2 επίπεδα: ένα γεωγραφικό ανάμεσα στα διάφορα τοπικά κέντρα δεδομένων και ένα εντός του τοπικού κέντρου δεδομένων ανάμεσα στους διάφορους εξυπηρετητές της εφαρμογής.

Εναλλακτικά η κατανομή του φορτίου μπορεί να γίνει και χειροκίνητα από τον χρήστη, επιλέγοντας την προτιμότερη για αυτόν τοποθεσία καθρέφτη (mirror site) ανάμεσα σε γεωγραφικά διαφοροποιημένες επιλογές.

4.1.3 Πλεονεκτήματα και μειονεκτήματα χρήσης ισοσταθμιστή φορτίου

Όπως περιγράφεται στα [68,69] η χρήση ισοσταθμιστή φορτίου είναι σημαντική σε φάρμες εξυπηρετητών για τους ακόλουθους λόγους:

1) Κλιμακωσιμότητα:
   επιτρέπει την εξυπηρέτηση περισσότερων αιτήσεων και περισσότερων πελατών. Αποφεύγει τους περιορισμούς (bottleneck) του μοναδικού εξυπηρετητή.

2) Ευελιξία:
   μπορούν εύκολα να προστεθούν ή να αφαιρεθούν εξυπηρετητές και να προσαρμοσθεί το σύστημα γρήγορα σε μεταβολές του φορτίου. Ειδικότερα σε απότομες αυξήσεις της ζήτησης της ιστοσελίδας προστίθενται απλά νέοι εξυπηρετητές. Αυτό είναι γίνεται εύκολα στην περίπτωση που υπάρχουν και άλλοι διαθέσιμοι εξυπηρετητές ή η εφαρμογή φιλοξενείται στο νέφος.

3) Αποδοτικότητα:
   με μια αποδοτική κατανομή φορτίου αξιοποιείται η υπολογιστική ικανότητα όλων των διαθέσιμων εξυπηρετητών, χωρίς να υπερφορτώνεται κανένας.

4) Πλεονασμός και διαχείριση σφάλματος:
   μπορεί ένας εξυπηρετητής με σφάλμα απλώς να αφαιρεθεί και να πάρει άλλος τη θέση του. Έτσι υπάρχει αδιάλειπτη διαθεσιμότητα.

5) Ασφάλεια:
   στην περίπτωση που λειτουργεί και ως μεταφραστής διευθύνσεων (NAT) αποφεύγεται η άμεση επαφή πελάτη – εξυπηρετητή. Επίσης, οι επιθέσεις κατανεμημένης άρνησης υπηρεσίας είναι πιο δύσκολο να κάμψουν ένα σύστημα με πολλούς εξυπηρετητές και πιθανώς κεντρικό φιλτράρισμα πριν την κατανομή του φορτίου.

Βέβαια υπάρχουν και κάποια μειονεκτήματα από τη χρήση ισοσταθμιστή φορτίου, αυτά σύμφωνα με το [70] είναι:

1) Πολυπλοκότητα:
   προστίθεται ένα ακόμα στοιχείο στο σύστημα. Απαιτείται πρόσθετη εργασία για τη διαχείριση και συντήρηση του.

2) Μοναδικό σημείο αποτυχίας:
   εάν συμβεί κάποιο σφάλμα στον ισοσταθμιστή φορτίου, τότε όλο το σύστημα μπορεί να σταματήσει να εξυπηρετεί.

3) Κόστος:
   ο υλικός εξοπλισμός και η λειτουργία του λογισμικού συνεπάγονται κόστη.

4.1.4 Αλγόριθμοι για ισοστάθμιση φορτίου

Όπως περιγράφεται στο [68] υπάρχουν δυο είδη αλγορίθμων για ισοστάθμιση του φορτίου στους εξυπηρετητές, οι στατικοί και οι δυναμικοί αλγόριθμοι. Αυτοί είναι:

Α) Οι στατικοί, που εφαρμόζουν τον ίδιο κανόνα κατανομής κάθε φορά, ανεξάρτητα της τρέχουσας κατάστασης των εξυπηρετητών. Διακρίνονται σε:

- Κυκλικής εναλλαγής (round-robin), μπορούν να υλοποιηθούν μέσω του συστήματος ονοματοδοσίας (DNS). Μπορούν να τροποποιηθούν ώστε να περιέχουν και βάρη, στέλνοντας περισσότερη ή λιγότερη κίνηση σε εξυπηρετητές ανάλογα με τις υπολογιστικές ικανότητες τους.

- Κατακερματισμού (hash) της IP διεύθυνσης προέλευσης (του πελάτη). Υλοποιείται στον ισοσταθμιστή φορτίου, με υπολογισμό κρυπτογραφικής συνάρτησης.

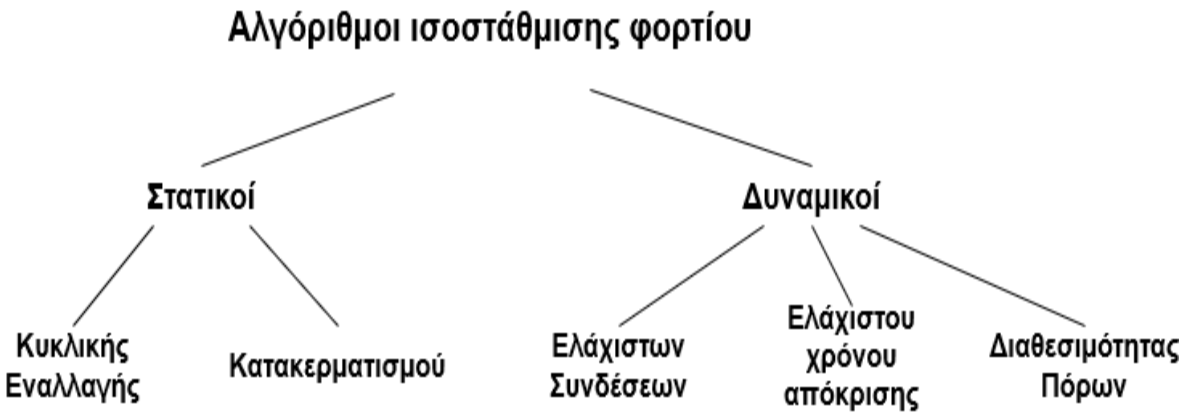
Β) Οι δυναμικοί, που εξετάζουν την τρέχουσα κατάσταση των εξυπηρετητών, πριν αποφασίσουν που θα στείλουν τον πελάτη.

- Ελάχιστων συνδέσεων. Είναι δυναμική μέθοδος που λαμβάνει υπ' όψη της τον αριθμό των τρεχουσών συνδέσεων που διατηρούνται στον κάθε εξυπηρετητή. Επιλέγεται αυτός με τις λιγότερες. Μπορεί να τροποποιηθεί ώστε να περιέχει και βάρη, στέλνοντας περισσότερη ή λιγότερη κίνηση σε εξυπηρετητές ανάλογα με τις υπολογιστικές ικανότητες τους.

- Ελάχιστου χρόνου απόκρισης, όπου λαμβάνεται υπ' όψη ο χρόνος απόκρισης του εξυπηρετητή και ο αριθμός των συνδέσεων. Στοχεύει στην ελαχιστοποίηση του χρόνου απάντησης στον πελάτη.

- Βασισμένων σε διαθεσιμότητα πόρων. Συνδυάζονται με λογισμικό που εκτελείται στους εξυπηρετητές και ενημερώνει τον ισοσταθμιστή φορτίου για τη χρήση πόρων, όπως της υπολογιστικής μονάδας και της μνήμης. Έτσι ελέγχεται εάν υπάρχουν αρκετοί πόροι, πριν γίνει επιπρόσθετη ανάθεση φορτίου.

Ακολουθεί σχήμα με τους αλγόριθμους ισοστάθμισης φορτίου.



Σχήμα 4.3 *Αλγόριθμοι ισοστάθμισης φορτίου*

4.1.5 Τεχνολογία Ισοστάθμισης φορτίου

Σύμφωνα με το [68] υπάρχουν δυο είδη υλοποίησης ισοσταθμιστών φορτίου. Το ένα είναι σε υλικό και το άλλο σε λογισμικό. Το κάθε ένα με τα δικά του πλεονεκτήματα και μειονεκτήματα.

Η υλοποίηση σε υλικό είναι αποδοτική και υποστηρίζει περισσότερη κίνηση, κάτι που είναι το βασικό της πλεονέκτημα. Επίσης, δίνεται και η δυνατότητα εικονικοποίησης του υλικού και η τμηματοποίηση του σε πολλαπλούς ισοσταθμιστές φορτίου για ταυτόχρονη εξυπηρέτηση πολλαπλών εφαρμογών. Απαιτείται βέβαια μια ακριβή αρχική επένδυση, ρύθμιση και συντήρηση. Επίσης, πρέπει να ληφθεί υπ' όψη η μέγιστη κίνηση που μπορεί να χρειαστεί να εξυπηρετηθεί και να αποκτηθεί με βάση αυτή την προδιαγραφή. Αυτό θα έχει ως αποτέλεσμα μικρή χρησιμοποίηση υπό κανονικές συνθήκες. Διαφορετικά, μπορεί να χρειαστεί σύντομη αντικατάσταση του, με μεγαλύτερο, εάν αυξηθεί η ζήτηση της εφαρμογής.

Η υλοποίηση σε λογισμικό, στην κατεύθυνση της εικονικοποίησης δικτυακών λειτουργιών (NFV) προσφέρει μεγαλύτερη ευελιξία, καθώς μπορεί να εκτελείται σε έναν απλό εξυπηρετητή. Κλιμακώνει εύκολα προς τα πάνω ή κάτω και μπορεί να έχει μικρότερο κόστος. Είναι κατάλληλη και για περιβάλλον νέφους, όπου μπορεί να προσφέρεται και ως υπηρεσία (managed service).

4.1.6 Το παράδειγμα του Scalelite

Το Scalelite, που παρουσιάζεται στο [71] είναι ένας ανοικτού κώδικα ισοσταθμιστής φορτίου για την εφαρμογή BigBlueButton. Το BigBlueButton (BBB), που παρουσιάζεται στο [72] είναι μια ανοικτού κώδικα εφαρμογή τηλεδιάσκεψης για τηλεκπαίδευση. Χρησιμοποιείται από πολλά πανεπιστήμια και από το Εθνικό Μετσόβιο Πολυτεχνείο. Στο BigBlueButton ο χρήστης μπορεί να συμμετέχει σε μια συνάντηση (meeting) και να λάβει και να στείλει video, ήχο και μηνύματα από τον περιηγητή του, χωρίς καμία εγκατάσταση λογισμικού. Ένας τυπικός BigBlueButton εξυπηρετητής μπορεί να υποστηρίξει περίπου 200 άτομα.

Το Scalelite είναι ένας ισοσταθμιστής φορτίου που κατανέμει τις συναντήσεις σε διάφορους BBB εξυπηρετητές. Έτσι αυξάνονται ο αριθμός των παράλληλων συναντήσεων και των συμμετεχόντων χρηστών που υποστηρίζονται. Το Scalelite επικοινωνεί περιοδικά με τους BBB εξυπηρετητές και λαμβάνει την κατάσταση τους και το φορτίο τους, όπως για παράδειγμα αν είναι συνδεδεμένοι και τον αριθμό των χρηστών και των συναντήσεων που φιλοξενούν. Όταν έρθει αίτημα για νέα συνάντηση, την ξεκινάει στο λιγότερο φορτωμένο BBB εξυπηρετητή. Όταν ένας χρήστης θέλει να συνδεθεί σε μια συνάντηση ανακατευθύνεται στον κατάλληλο BBB εξυπηρετητή.

Το Scalelite σύμφωνα με το [73] είναι γραμμένο στη γλώσσα Ruby on Rails και η αρχιτεκτονική του ισοσταθμιστή φορτίου, αποτελείται από:

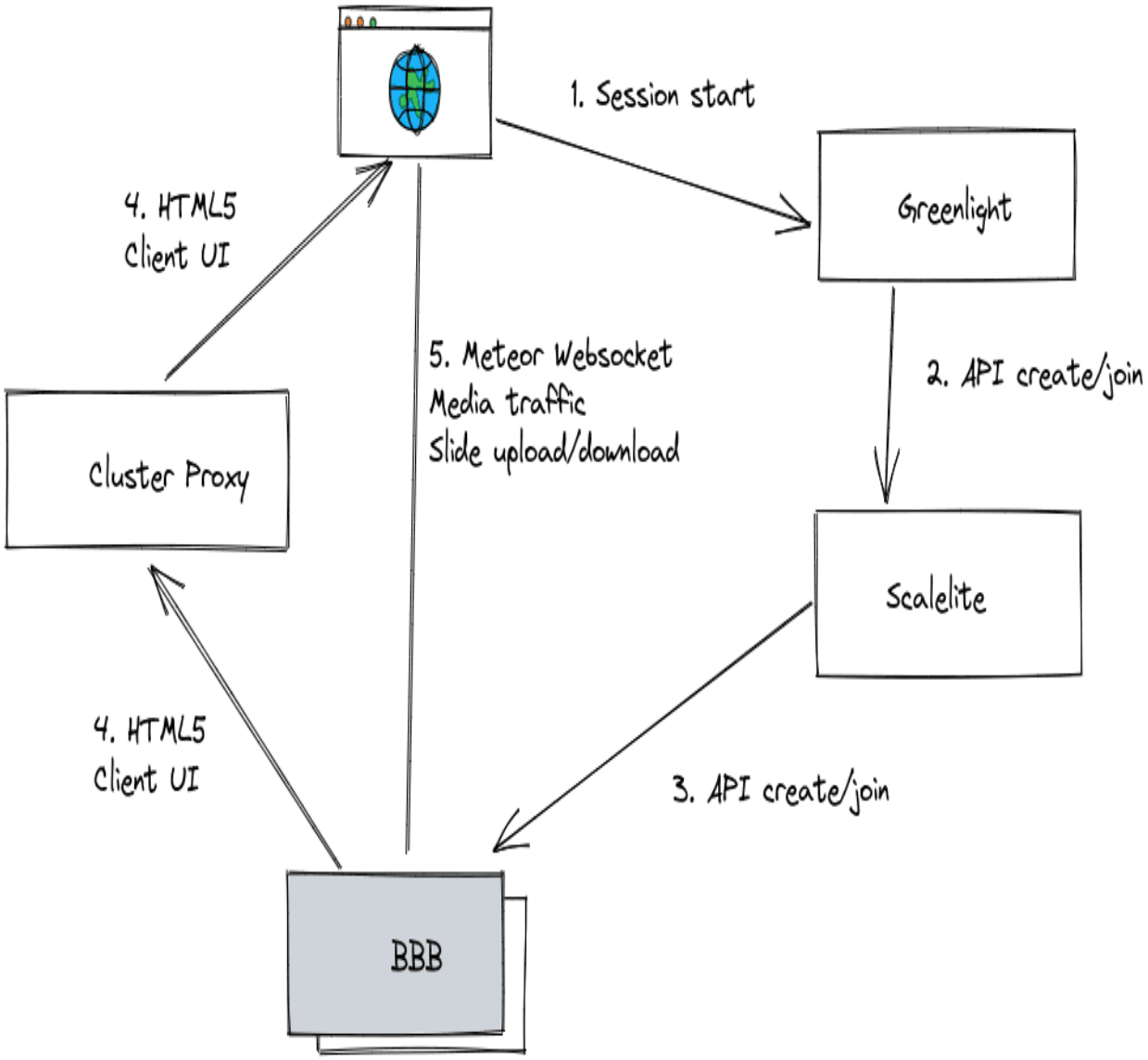
1) έναν τροποποιημένο NGINX διαμεσολαβητή,
2) το τμήμα όπου λαμβάνει την απόφαση κατανομής και στέλνει αιτήματα στο BBB API
3) ένα κομμάτι κώδικα (script) που ρωτάει περιοδικά (poll) τους BBB εξυπηρετητές για την κατάσταση τους

Περιλαμβάνονται ακόμα: ένα τμήμα που ασχολείται με τις καταγραφές των συναντήσεων, βάση δεδομένων, προσωρινή μνήμη και αποθηκευτικός χώρος καταγραφών.

Υπάρχει μια επέκταση του Scalelite, που εισάγει ένα νέο διαμεσολαβητή (cluster proxy). Εδώ ο διαμεσολαβητής χρησιμοποιείται για να στέλνει το HTML και τα στατικά αρχεία της σελίδας, ενώ το video, ο ήχος, η σηματοδοσία και η συνομιλία ανταλλάσσονται απευθείας με

τον BBB εξυπηρετητή. Αυτό επιτρέπει οι άδειες που δίνονται από τον χρήστη, αλλά και οι ρυθμίσεις να γίνονται μόνο μια φορά, ανεξάρτητα από τον εξυπηρετητή που συνδέεται. Επίσης αποφορτώνεται και ο εξυπηρετητής, αφού τα στατικά αρχεία εξυπηρετούνται από την προσωρινή μνήμη του διαμεσολαβητή. Γίνεται να τοποθετηθεί και δεύτερος διαμεσολαβητής, για πλεονασμό σε περίπτωση σφάλματος, πίσω από έναν κλασσικό ισοσταθμιστή φορτίου. Αυτό παρουσιάζεται στο [74].

Ακολουθεί σχέδιο με την πορεία μιας αίτησης στο BBB. Το Greenlight ή το Moodle είναι εφαρμογές που συνεργάζονται με το BBB και του στέλνουν αιτήσεις συμμετοχής ή δημιουργίας συνάντησης.



Εικόνα 4.1 Επέκταση Scalelite με διαμεσολαβητή (cluster proxy) [75]

4.2 Load Balancing OpenWebRX

4.2.1 Η ιδέα

Το Load Balancing OpenWebRX αποτελεί μια επέκταση του προγράμματος OpenWebRX. Σε αυτή την επέκταση υλοποιήθηκε ένας "έξυπνος" ισοσταθμιστής φορτίου, ειδικά για αυτή την εφαρμογή. Συνδυάζει πολλούς τροποποιημένους OpenWebRX εξυπηρετητές με ένα ισοσταθμιστή φορτίου σε μια ενιαία οντότητα.

Κύρια ανάγκη που έρχεται να λύσει το Load Balancing OpenWebRX είναι η αύξηση του αριθμού των ταυτόχρονων υποστηριζόμενων χρηστών. Επιτρέπει ακόμα την αύξηση του αριθμού των συνδεδεμένων συσκευών SDR. Βελτιώνει παράλληλα και την εμπειρία χρήσης, με τρόπους που προσφέρουν αδιάλειπτη και ανεμπόδιστη λειτουργία.

Τα πλεονεκτήματα που έχει το Load Balancing OpenWebRX είναι:

1) Κλιμάκωση του αριθμού των ταυτόχρονων πελατών:
   Με την εισαγωγή νέων OpenWebRX εξυπηρετητών, αυξάνεται η συνολική υπολογιστική ικανότητα της εφαρμογής. Η υπολογιστικά απαιτητική επεξεργασία σήματος περιορίζει τον αριθμό των χρηστών σε κάθε μηχάνημα. Με περισσότερους εξυπηρετητές, μπορούν να εξυπηρετηθούν περισσότεροι χρήστες.

2) Κλιμάκωση του αριθμού των συνδεδεμένων συσκευών SDR:
   Ο αριθμός των συσκευών SDR που μπορούν να συνδεθούν σε ένα υπολογιστή που τρέχει ένας OpenWebRX εξυπηρετητής είναι περιορισμένος. Αυτό συμβαίνει λόγω του μεγάλου όγκου δεδομένων που στέλνονται στη σύνδεση, μεταξύ συσκευής SDR και υπολογιστή. Για παράδειγμα το εύρος ζώνης του USB ελεγκτή περιορίζει τον αριθμό των USB συσκευών SDR που μπορούν να συνδεθούν. Σε περισσότερους εξυπηρετητές, μπορούν να συνδεθούν περισσότερες συσκευές SDR. Αυτό έχει ως συνέπεια να υπάρχουν περισσότερες επιλογές για τους πελάτες και να υποστηρίζονται περισσότεροι ταυτόχρονα.

3) Αρμονική συνύπαρξη χρηστών:

   Μια συσκευή SDR μπορεί να λειτουργεί κάθε στιγμή σε μια συγκεκριμένη ζώνη συχνοτήτων. Στο απλό OpenWebRX όταν ένας χρήστης άκουγε ένα σήμα και συνδεόταν και ένας άλλος στην ίδια συσκευή, αλλά σε άλλη ζώνη συχνοτήτων, τότε άλλαζε η ζώνη συχνοτήτων ταυτόχρονα και στους δύο. Σαν αποτέλεσμα, άλλαζε το σήμα και στον πρώτο χρήστη και η ακρόαση του χάλαγε. Τώρα λαμβάνεται υπ' όψη από τον ισοσταθμιστή φορτίου η ζώνη συχνοτήτων που λειτουργεί μια ενεργή SDR συσκευή και ο επόμενος χρήστης κατευθύνεται σε άλλη συσκευή. Έτσι όλοι ακούν το σήμα τους ανεμπόδιστα.

4) Διαχείριση σφάλματος και πλεονασμός:

   Η διαθεσιμότητα πολλών εξυπηρετητών και συσκευών SDR, επιτρέπει σε περίπτωση σφάλματος, την ανακατεύθυνση του χρήστη σε μια άλλη συσκευή, που δουλεύει καλά. Έχει υλοποιηθεί σύστημα που ανιχνεύει κάποια είδη βλάβης και ανακατευθύνει αυτόματα τον χρήστη. Επίσης, δίνεται και η δυνατότητα χειροκίνητης αλλαγής συσκευής SDR από τον χρήστη, σε περίπτωση σφάλματος.

Σημειώνεται ότι στο Load Balancing OpenWebRX η κατεύθυνση των χρηστών σε κατάλληλο εξυπηρετητή και συσκευή SDR γίνεται αυτόματα. Επίσης, όλοι οι OpenWebRX εξυπηρετητές λειτουργούν ως μια ενιαία εφαρμογή.

Το Load Balancing OpenWebRX προορίζεται για χρήση από OpenWebRX εξυπηρετητές παγκοσμίως, που έχουν μεγάλο αριθμό πελατών προς εξυπηρέτηση.

4.2.2 Τεχνική παρουσίαση

Ο ισοσταθμιστής φορτίου υλοποιήθηκε σε λογισμικό, στη γλώσσα Python. Λειτουργεί στο επίπεδο εφαρμογής και εξετάζει την HTTP επικεφαλίδα για να αποφασίσει την κατανομή. Δρα ως διαμεσολαβητής για τα στατικά αρχεία που μεταφέρονται μέσω HTTP, όπως στην επέκταση cluster proxy του Scalelite. Στο ίδιο πνεύμα, συνδέεται ο πελάτης με τον OpenWebRX εξυπηρετητή απευθείας, μέσω WebSocket, όπου στέλνονται ο ήχος, τα δεδομένα του καταρράκτη και οι επιλογές του χρήστη.

Ο OpenWebRX εξυπηρετητής έχει τροποποιηθεί ώστε να καταγράφει χρήσιμες πληροφορίες για τη λειτουργία του και την τρέχουσα κατάσταση του (state). Ο ισοσταθμιστής φορτίου επικοινωνεί περιοδικά, αλλά και όποτε χρειάζεται, με τους OpenWebRX εξυπηρετητές και λαμβάνει τις πληροφορίες κατάστασης. Ο αλγόριθμος κατανομής γίνεται με βάση αυτές τις πληροφορίες και συγκεκριμένα: τους διαθέσιμους πόρους των εξυπηρετητών, τις διαθέσιμες δυνατότητες των συνδεδεμένων συσκευών SDR, την τρέχουσα κατάσταση εξυπηρέτησης και προτεραιότητες που έχουν οριστεί από τον διαχειριστή. Ένας εξυπηρετητής μπορεί να εισέλθει στο σύστημα και να αποχωρήσει οποτεδήποτε.

Η κατεύθυνση του πελάτη σε εξυπηρετητή γίνεται κατόπιν αίτησης στον ισοσταθμιστή φορτίου και σε συνεργασία με τον κώδικα JavaScript στην πλευρά του πελάτη. Ειδικότερα, ανοίγεται μια WebSocket σύνδεση από τον πελάτη με τον αποφασισμένο εξυπηρετητή. Στον πελάτη υλοποιείται και η διαχείριση σφάλματος, με δυνατότητα επανασύνδεσης σε άλλη συσκευή SDR ή εξυπηρετητή.

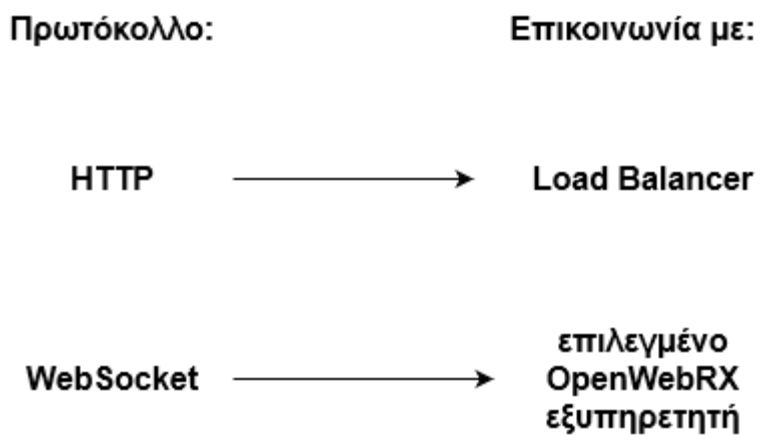
Επιπρόσθετα, υποστηρίζεται ο ρόλος του διαχειριστή, για ρυθμίσεις μέσω web. Υπάρχουν σελίδες γενικών ρυθμίσεων, ρυθμίσεων ζωνών συχνοτήτων και εξυπηρετητών. Τέλος, υπάρχουν αντίστοιχες σελίδες επισκόπησης της λειτουργίας του συστήματος.

Θα ακολουθήσει μια αναλυτική περιγραφή των διάφορων λειτουργιών του Load Balancing OpenWebRX.

4.3 Λειτουργίες του Load Balancing OpenWebRX

4.3.1 Καταμερισμός εργασιών

Τα αιτήματα του πρωτοκόλλου HTTP αναλαμβάνονται από τον ισοσταθμιστή φορτίου, ο οποίος λειτουργεί και ως διαμεσολαβητής. Όπως και στο cluster proxy Scalelite εξυπηρετεί τα στατικά αρχεία HTML, CSS και JavaScript. Μέσα στο αρχείο JavaScript (openwebrx.js), του πελάτη, περιλαμβάνεται κώδικας που ανοίγει μια WebSocket σύνδεση με κάποιο OpenWebRX εξυπηρετητή. Το πρωτόκολλο WebSocket εξυπηρετείται αποκλειστικά από τον επιλεγμένο OpenWebRX εξυπηρετητή. Η αποστολή ήχου, δεδομένων του γραφήματος καταρράκτη και των επιλογών χρήστη γίνεται μέσω του WebSocket, απευθείας μεταξύ πελάτη και εξυπηρετητή.



Σχήμα 4.4 *Καταμερισμός εργασιών των πρωτοκόλλων*

4.3.2 Καταγραφή της κατάστασης (state) στους τροποποιημένους OpenWebRX εξυπηρετητές

Ο κώδικας του OpenWebRX εξυπηρετητή τροποποιήθηκε κατάλληλα ώστε να καταγράφεται η κατάσταση του (state) και να διατίθεται μέσω μιας προγραμματιστικής

διεπαφής (API). Προστέθηκε αρχείο με κώδικα ελεγκτή (controller) στην τοποθεσία controllers/state.py. Συγκεκριμένα το API περιλαμβάνει δυο προσβάσιμα μέσω HTTP άκρα (endpoints): το initstate.json και το curstate.json. Τα δεδομένα της κατάστασης (state) χωρίζονται:

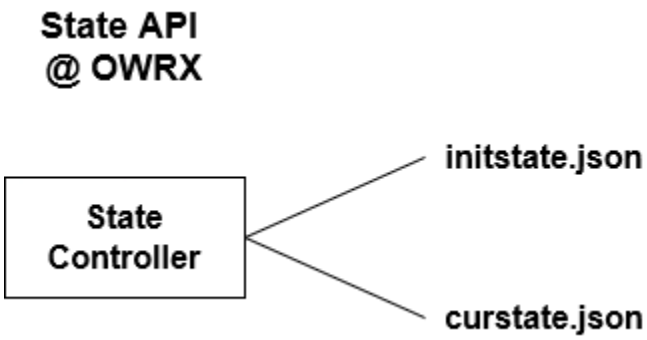
Α) σε αυτά που αλλάζουν σπάνια και στέλνονται μια φορά στην αρχή. Επίσης, ξαναστέλνονται όταν αλλάξουν. Αυτά είναι τα δεδομένα της αρχικής κατάστασης (initstate).

Β) σε αυτά που συχνά μεταβάλλονται και στέλνονται περιοδικά ή κατ' απαίτηση. Αυτά είναι τα δεδομένα της τρέχουσας κατάστασης (curstate).

Τα δεδομένα της αρχικής κατάστασης αποτελούνται από πληροφορίες σχετικές με τις συνδεδεμένες συσκευές SDR, τις ρυθμισμένες ζώνες συχνοτήτων και μια ετικέτα κωδικού κατακερματισμού (hash) που χρησιμοποιείται για να γίνονται αντιληπτές οι αλλαγές σε αυτά τα δεδομένα.

Τα δεδομένα της τρέχουσας κατάστασης αποτελούνται από τη μέση πρόσφατη χρήση της υπολογιστικής μονάδας (latest average cpu usage), την πληροφορία για το ποιες συσκευές SDR είναι ενεργές, τον αριθμό των συνδεδεμένων πελατών σε κάθε συσκευή SDR, τη ζώνη συχνοτήτων που είναι ενεργή τη δεδομένη χρονική στιγμή και την ετικέτα που περιγράφηκε παραπάνω, για άμεση ανίχνευση αλλαγών στα αρχικά δεδομένα.

Τα δεδομένα αποθηκεύονται σε δομή λεξικού (dictionary) και μετατρέπονται σε μορφή JSON για να σταλθούν.



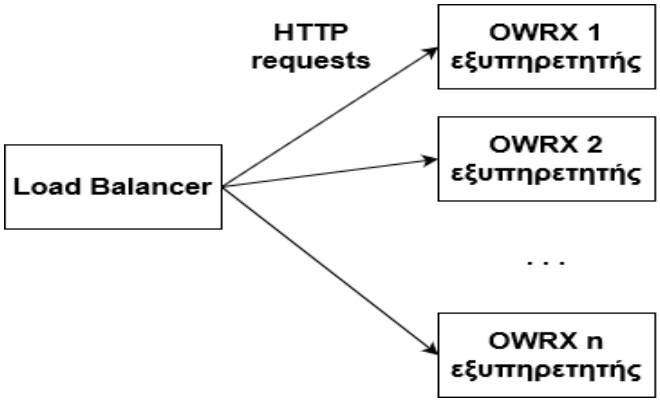
Σχήμα 4.5 *Προγραμματιστική διεπαφή για το state, στον τροποποιημένο OpenWebRX εξυπηρετητή.*

4.3.3 Ενδοεπικοινωνία (Load Balancer – OpenWebRX)

Ο ισοσταθμιστής φορτίου χρησιμοποιεί το State API, ρωτώντας όλους τους εγγεγραμμένους OpenWebRX εξυπηρετητές, για να λάβει την κατάσταση τους. Αυτό υλοποιείται στο αρχείο receivestate.py του load balancer.

Εκεί υπάρχει ένα εξειδικευμένο νήμα για τη λήψη της κατάστασης των εξυπηρετητών. Αυτό ξυπνάει περιοδικά και κατ' απαίτηση, όταν έρχεται νέος πελάτης ή εμφανίζονται πληροφορίες της κατάστασης στην επισκόπηση. Ο λόγος που επιλέχθηκε και κατ' απαίτηση ενεργοποίηση του, είναι η αποδοτική και γρήγορη αλλαγή συσκευής SDR, όπου απαιτούνται τα πιο πρόσφατα δεδομένα. Στην περίπτωση αυτή είναι σημαντικό να ενημερωθεί άμεσα ο ισοσταθμιστής φορτίου, για την αποχώρηση του πελάτη από τον προηγούμενο εξυπηρετητή (που τώρα μένει ελεύθερος) πριν τον κατανείμει στον επόμενο εξυπηρετητή (που μπορεί να παραμείνει και ο ίδιος). Επίσης, έχει υλοποιηθεί σχήμα συγχρονισμού, ώστε να καλείται η διαδικασία ενημέρωσης, από άλλο μέρος του κώδικα και να επιστρέφει μόλις έχει ολοκληρωθεί η επικαιροποίηση των δεδομένων της κατάστασης.

Λαμβάνονται τόσο η αρχική όσο και η τρέχουσα κατάσταση του κάθε εξυπηρετητή, Η λήψη της αρχικής κατάστασης αποτελεί ειδική περίπτωση, καθώς εκτελείται μόνο όταν δεν υπάρχουν διαθέσιμα ή έχουν αλλάξει τα σχετικά δεδομένα. Η μεταφορά των δεδομένων γίνεται με HTTP αιτήματα που δημιουργούνται από το Python module requests. Τέλος, τα δεδομένα που λαμβάνονται, αποθηκεύονται σε μια δομή, η οποία περιλαμβάνει δεδομένα κατάστασης για όλους τους εξυπηρετητές και ορίζεται στο αρχείο distribution.py.



Λήψη:	Πότε?
initstate	στην αρχή και στην αλλαγή
curstate	περιοδικά και κατ' απαίτηση

Σχήμα 4.6 *Ενδοεπικοινωνία για λήψη του state*

4.3.4 Αλγόριθμος Κατανομής

Ο αλγόριθμος κατανομής των πελατών σε εξυπηρετητές και συσκευές SDR έχει υλοποιηθεί στο αρχείο distribution.py, του ισοσταθμιστή φορτίου. Εκεί, αφού τα δεδομένα της κατάστασης έρθουν από τους OpenWebRX εξυπηρετητές, αποθηκεύονται σε μια δομή δεδομένων, τη servers_state_db, ως ένα Python λεξικό. Αυτή η δομή έχει δενδρική μορφή και περιέχει πληροφορίες για τους εξυπηρετητές, τις συσκευές SDR και τις διαθέσιμες ζώνες συχνοτήτων. Διατρέχεται όταν έρθει ένας νέος πελάτης, προκειμένου να βρεθεί η βέλτιστη κατανομή. Περιλαμβάνονται ακόμα μέθοδοι διαχείρισης των δεδομένων και ένα κλείδωμα για συγχρονισμό. Αυτό χρειάζεται γιατί η δομή ταυτόχρονα, από τη μια ενημερώνεται από το νήμα λήψης της κατάστασης των εξυπηρετητών και από την άλλη χρησιμοποιείται στην απόφαση κατανομής.

Η είσοδος του αλγορίθμου προέρχεται από τον πελάτη και περιλαμβάνει την επιλεγμένη γενική ζώνη συχνοτήτων και μια λίστα από συσκευές SDR που προσωρινά αποκλείονται, επειδή έχει αναφερθεί σε αυτές κάποιο σφάλμα. Η έξοδος του, είναι ένας συνδυασμός ταυτότητας (id) εξυπηρετητή, ταυτότητας συσκευής SDR και ταυτότητας ζώνης συχνοτήτων. Αυτά θα επιστραφούν στον πελάτη, αφού πρώτα αντιστοιχισθεί η διεύθυνση του εξυπηρετητή.

Ο αλγόριθμος εξετάζει ένα πλήθος παραμέτρων προκειμένου να αποφασίσει τη βέλτιστη κατανομή του πελάτη. Κάποιες παράμετροι δρουν προσθετικά, καθορίζοντας μια βαθμολογία προτίμησης. Όλες οι παράμετροι είναι:

1) Διαθεσιμότητα του εξυπηρετητή. Αρχικά εξετάζεται αν ο εξυπηρετητής είναι συνδεδεμένος και απαντάει στα αιτήματα που στέλνονται για τη λήψη της κατάστασης του.

2) Το φορτίο του εξυπηρετητή. Ελέγχεται ώστε να μην ξεπερνάει ένα όριο και υπερφορτωθεί, τότε αποκλείεται. Αν έχει αρχίσει να φορτώνεται αρκετά, τότε μειώνεται η βαθμολογία του και ίσως προτιμηθεί άλλος.

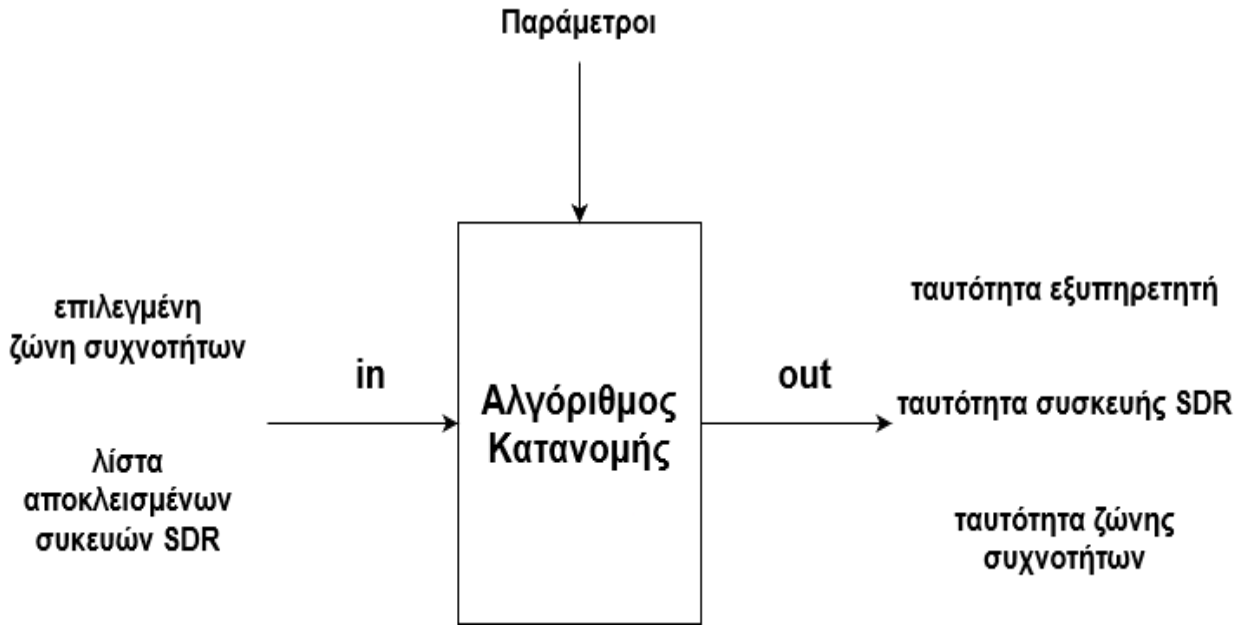
3) Η τρέχουσα κατάσταση εξυπηρέτησης της συσκευής SDR. Ελέγχεται αν είναι κατειλημμένη από άλλο χρήστη και αν ναι σε ποια ζώνη συχνοτήτων λειτουργεί. Αν η ζητούμενη ζώνη συχνοτήτων με την τρέχουσα ταυτίζονται, τότε έχουμε μια επιθυμητή

σύμπτωση και προτιμάται. Αν όχι, αποκλείεται, γιατί διαφορετικά θα χάλαγε η λήψη του αρχικού χρήστη.

4) Ταίριασμα των ζωνών συχνοτήτων, της αιτούμενης και των διαθέσιμων της συσκευής SDR. Ελέγχεται αν η συσκευή SDR έχει ρυθμιστεί και μπορεί να λειτουργήσει στην αιτούμενη ζώνη συχνοτήτων.

5) Προτεραιότητες. Υποστηρίζεται εισαγωγή προτεραιοτήτων από τον διαχειριστή. Επηρεάζουν τη βαθμολογία του συνδυασμού κατανομής.

6) Εξοικονόμηση πόρων. Σε περίπτωση ισοβαθμίας δυο ή περισσότερων συνδυασμών κατανομής που ικανοποιούν όλα τα κριτήρια, τότε επιλέγεται αυτός με το ελάχιστο εύρος ζώνης. Έτσι επιτυγχάνεται εξοικονόμηση υπολογιστικής ισχύος.



Σχήμα 4.7 *Αλγόριθμος Κατανομής*

4.3.5 Κατεύθυνση του πελάτη

Εδώ περιγράφεται ο τρόπος με τον οποίο κατευθύνεται ο πελάτης στο συγκεκριμένο OpenWebRX εξυπηρετητή και στη συσκευή SDR που θα τον εξυπηρετήσει. Ουσιαστικά πρόκειται για την εφαρμογή της απόφασης κατανομής του ισοσταθμιστή φορτίου. Συνοπτικά τα βήματα που ακολουθούνται είναι:

1) Αρχικά ο πελάτης στέλνει στον ισοσταθμιστή φορτίου την επιλογή του σχετικά με τη γενική ζώνη συχνοτήτων, δηλαδή μια ζώνη που είναι ορισμένη ως διαθέσιμη στο σύστημα.
2) Λαμβάνει την απόφαση κατανομής από τον ισοσταθμιστή φορτίου, η οποία περιέχει τη διεύθυνση του εξυπηρετητή, την ταυτότητα της συσκευής SDR και την ταυτότητα της ζώνης συχνοτήτων στη συγκεκριμένη συσκευή.
3) Στη συνέχεια ο περιηγητής του πελάτη ανοίγει μια WebSocket σύνδεση με τον αποφασισμένο OpenWebRX εξυπηρετητή.
4) Κατόπιν, του στέλνει το μήνυμα sdrselection, το οποίο περιέχει τις ταυτότητες της αποφασισμένης συσκευής SDR και της ζώνης συχνοτήτων.
5) Τώρα ο OpenWebRX εξυπηρετητής ξεκινάει την αποφασισμένη συσκευή SDR (αν δεν λειτουργεί ήδη).
6) Αφού γίνουν όλα αυτά, πάνω από την WebSocket σύνδεση που έχει εγκατασταθεί, πραγματοποιείται η εξυπηρέτηση. Στέλνονται: ο ήχος, τα δεδομένα του γραφήματος καταρράκτη και πρόσθετες επιλογές του χρήστη.

Για να επιτευχθεί αυτό, έγιναν αλλαγές:

Α) στον κώδικα JavaScript του πελάτη, στο αρχείο openwebrx.js.

Β) στον κώδικα του OpenWebRX εξυπηρετητή στο αρχείο connection.py.

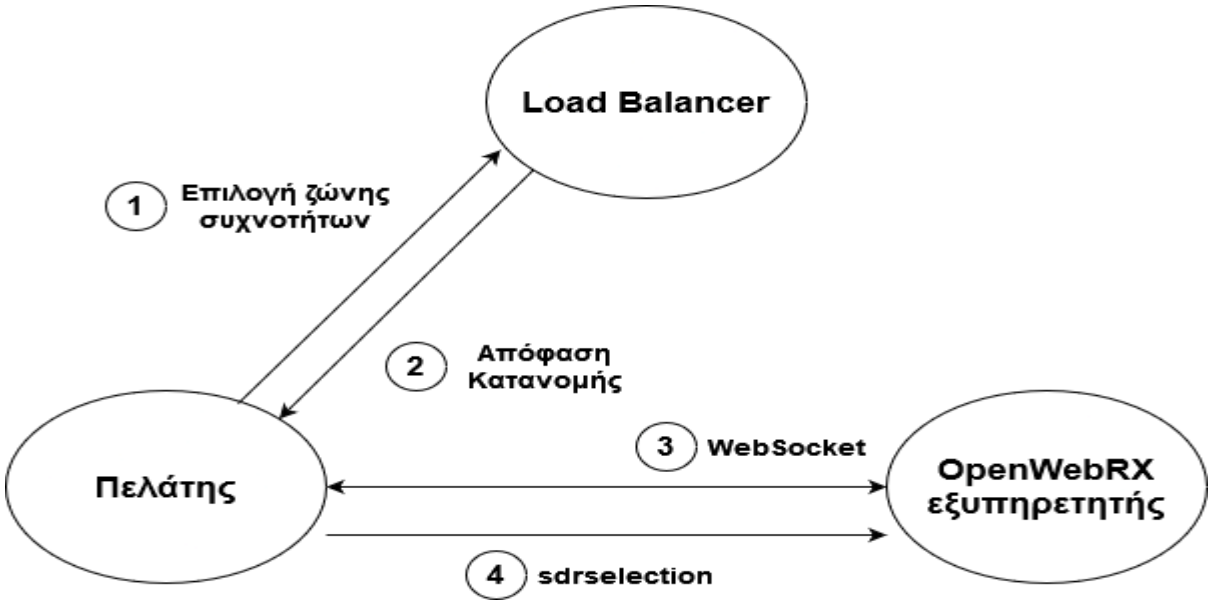
Αναλυτικά, στον κώδικα του πελάτη πραγματοποιείται ένα GET αίτημα στον ισοσταθμιστή φορτίου, μέσω jQuery και με παράμετρο την επιλεγμένη ζώνη συχνοτήτων. Ζητείται το δυναμικό αρχείο JavaScript websocket-vars.js, που περιέχει μεταβλητές, εκ των οποίων είναι και οι μεταβλητές της απόφασης κατανομής. Ο κώδικας του πελάτη αναμένει την απάντηση, πριν προχωρήσει. Στη συνέχεια εκτελείται η συνάρτηση open_websocket(), η οποία ανοίγει τη σύνδεση με τον αποφασισμένο εξυπηρετητή. Τέλος, μέσω αυτής της

WebSocket σύνδεσης στέλνεται το μήνυμα sdrselection που περιέχει την αποφασισμένη ταυτότητα συσκευής SDR και την ταυτότητα της ζώνης συχνοτήτων.

Επίσης, για τη λειτουργία της αλλαγής εξυπηρετητή και συσκευής SDR, έχει προστεθεί στη συνάρτηση sdr_profile_changed() κώδικας, που περιμένει να κλείσει η προηγούμενη σύνδεση με OpenWebRX εξυπηρετητή και στη συνέχεια ανοίγει καινούργια με τον νέο αποφασισμένο εξυπηρετητή.

Στον κώδικα του OpenWebRX εξυπηρετητή έχει αφαιρεθεί ο προεπιλεγμένος ορισμός και έναρξη της συσκευής SDR. Η συσκευή SDR ορίζεται και ξεκινάει μόλις ληφθεί το αντίστοιχο αίτημα. Επίσης, έχει απενεργοποιηθεί η απευθείας αλλαγή συσκευής SDR, καθώς αυτό γίνεται τώρα μέσω του ισοσταθμιστή φορτίου. Έχει αφαιρεθεί ακόμα, η αποστολή των διαθέσιμων ζωνών συχνοτήτων, μιας και αυτές ρυθμίζονται στο σύστημα του ισοσταθμιστή φορτίου.

Προστέθηκε ο χειρισμός του μηνύματος sdrselection που λαμβάνεται μέσω της WebSocket σύνδεσης. Συγκεκριμένα, καλείται η συνάρτηση setSdr() για να οριστεί η επιλεγμένη συσκευή SDR και η συνάρτηση addClient() με την οποία ξεκινάει το συνδετικό πρόγραμμα (connector), το οποίο καλεί τον οδηγό της συσκευής, εκτελεί τον χειρισμό και λαμβάνει τα δεδομένα της.



Σχήμα 4.8 *Επικοινωνία για την κατεύθυνση του πελάτη σε εξυπηρετητή και συσκευή SDR*

4.3.6 Διαχείριση Σφάλματος

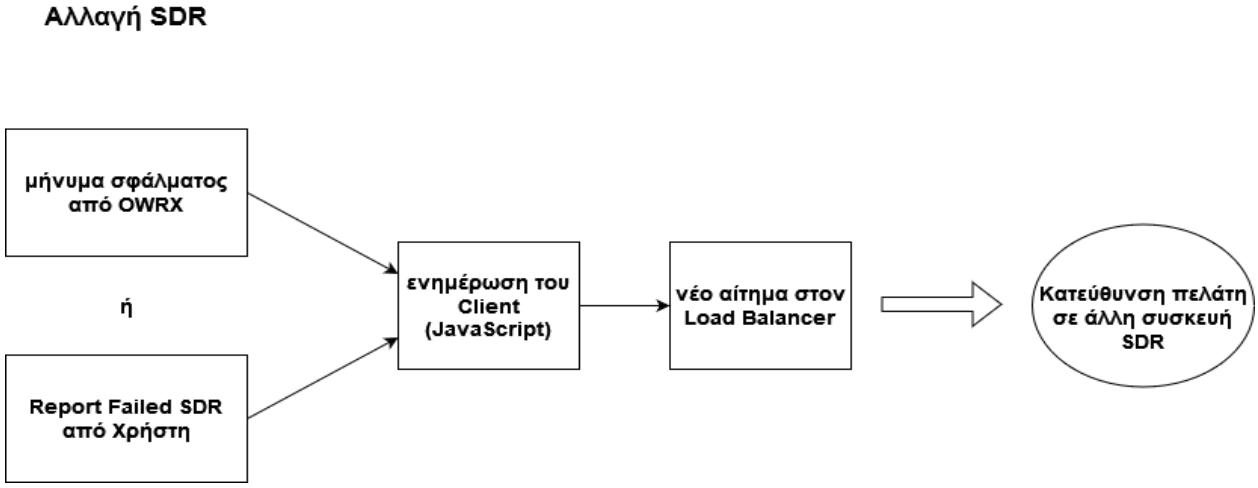
Έχει υλοποιηθεί μηχανισμός διαχείρισης πιθανού σφάλματος, κατά τη λειτουργία μιας συσκευής SDR. Αυτός ο μηχανισμός βρίσκεται στην πλευρά του πελάτη, δηλαδή στον κώδικα της JavaScript. Επίσης, έχει τροποποιηθεί η αντιμετώπιση ενός σφάλματος στον OpenWebRX εξυπηρετητή.

Η διαχείριση ενός σφάλματος στον πελάτη γίνεται είτε αυτόματα είτε χειροκίνητα. Στην πρώτη περίπτωση αξιοποιείται το μήνυμα σφάλματος μιας συσκευής SDR, που παράγει ο OpenWebRX εξυπηρετητής, το οποίο φτάνει μέσω WebSocket στον πελάτη. Ο κώδικας του πελάτη ενημερώνει με μήνυμα στην οθόνη, τον χρήστη και δρομολογεί αυτόματα την αλλαγή συσκευής SDR (μπορεί να αλλάξει και ο εξυπηρετητής). Για να γίνει αυτό, δημιουργείται ένα cookie, το οποίο ονομάζεται failed_sdrs και περιέχει την ταυτότητα της αποτυχημένης συσκευής SDR. Εάν το cookie υπάρχει από πριν, απλά προστίθεται στο περιεχόμενο του και η νέα ταυτότητα συσκευής, φτιάχνοντας μια λίστα. Το cookie αυτό προωθείται με ένα νέο αίτημα στον ισοσταθμιστή φορτίου. Εκεί η λίστα λαμβάνεται υπ' όψη από τον αλγόριθμο κατανομής και ο πελάτης κατευθύνεται σε άλλη συσκευή SDR.

Επειδή δεν αναγνωρίζονται αυτόματα όλα τα πιθανά σφάλματα, υπάρχει και η χειροκίνητη διαχείριση σφάλματος. Σε αυτή την περίπτωση, ο χρήστης πρέπει να πατήσει το κουμπί Report Failed SDR. Αυτό βρίσκεται στην οθόνη του, κάτω από τον χώρο που εμφανίζονται τα μηνύματα καταγραφής. Αυτή η ενέργεια ισοδυναμεί με τη λήψη μηνύματος σφάλματος, όπως παραπάνω. Δηλαδή, ο κώδικας του πελάτη ενημερώνεται και εκτελεί τις ίδιες ενέργειες χρησιμοποιώντας το cookie failed_sdrs. Έτσι καταλήγει ο χρήστης να αλλάζει συσκευή SDR. Σημειώνεται ότι το cookie αυτό έχει περιορισμένο διάστημα ζωής και σύντομα διαγράφεται, επιτρέποντας στον χρήστη να επιστρέψει σε μια συσκευή SDR, μετά από λίγο, σε περίπτωση που διορθώθηκε το πρόβλημα ή πατήθηκε το κουμπί κατά λάθος. Σε καμία περίπτωση η σήμανση ενός SDR ως αποτυχημένου, δεν επηρεάζει άλλο χρήστη, πέρα από αυτόν που το επισήμανε.

Ο OpenWebRX εξυπηρετητής τροποποιήθηκε στο αρχείο owrx/source/__init__.py στη συνάρτηση isFailed() και στο sdr.py στη συνάρτηση onFail() ώστε να μην επιστρέφει ποτέ ένδειξη μόνιμης αποτυχίας μιας συσκευής SDR. Από εμπειρία με τη χρήση των συσκευών του εργαστηρίου, συνήθως τα λάθη που προκύπτουν είναι προσωρινά και μετά

από λίγο η συσκευή δουλεύει κανονικά. Εάν μια συσκευή SDR δηλωθεί ως αποτυχημένη, τότε στο υπάρχον OpenWebRX, θα έπρεπε να γίνει επανεκκίνηση του εξυπηρετητή, προκειμένου η συσκευή να ανατεθεί σε πελάτη και να λειτουργήσει ξανά. Για αυτόν τον λόγο, αποφεύγεται η δήλωση αποτυχίας και δίνεται πάντα μια ευκαιρία. Αυτό πραγματοποιείται σε συνδυασμό με τη λογική που περιγράφηκε για τη διαχείριση στην πλευρά του πελάτη.



Σχήμα 4.9 *Διαχείριση σφάλματος συσκευής SDR*

Σύνοψη λειτουργιών που προστέθηκαν στο Load Balancing OpenWebRX:

1) Καταμερισμός εργασιών

2) Καταγραφή της κατάστασης (state) στους τροποποιημένους OpenWebRX εξυπηρετητές

3) Ενδοεπικοινωνία (Load Balancer – OpenWebRX)

4) Αλγόριθμος Κατανομής

5) Κατεύθυνση του πελάτη

6) Διαχείριση Σφάλματος

Σχήμα 4.10 *Σύνοψη λειτουργιών που προστέθηκαν στο Load Balancing OpenWebRX*

4.3.7 Διαφορές του Load Balancing OpenWebRX από το αρχικό OpenWebRX

Όλες οι λειτουργίες που αναφέρθηκαν προηγουμένως δεν υπήρχαν στο αρχικό OpenWebRX και προστέθηκαν από το Load Balancing OpenWebRX. Αυτό έγινε είτε δημιουργώντας νέο κώδικα στον ισοσταθμιστή φορτίου είτε τροποποιώντας τον κώδικα του εξυπηρετητή ή και του πελάτη. Υπάρχει ωστόσο και μια λειτουργία που ήταν διαθέσιμη στο αρχικό OpenWebRX, αυτή του χάρτη, αλλά δεν υποστηρίζεται σε αυτή την επέκταση. Αυτό συμβαίνει επειδή λόγω έλλειψης του κατάλληλου εξοπλισμού (εξωτερικού συστήματος κεραίας) δεν είναι εφικτή η λήψη σημάτων από πρωτόκολλα που απεικονίζουν τα δεδομένα τους σε χάρτη, όπως για παράδειγμα το APRS. Έτσι δεν υπήρχε η δυνατότητα να δοκιμαστεί η λειτουργία χάρτη στην πράξη.

Μελλοντικά, πέρα από τους Google Maps με JavaScript API που χρησιμοποιούνται τώρα, θα μπορούσε να προστεθεί και ο χάρτης ανοικτών δεδομένων του OpenStreetMap, που παρουσιάζεται στο [76]. Αυτό γίνεται ήδη σε άλλη επέκταση, στο OpenWebRX+, σύμφωνα με το [77].

Οι λειτουργίες των ρυθμίσεων και της επισκόπησης μέσω διαδικτυακής διεπαφής (web interface) υπάρχουν και στο OpenWebRX, ωστόσο εδώ προσαρμόστηκαν ώστε να καλύπτουν τις ανάγκες αυτής της επέκτασης. Συνεχίζει να υποστηρίζεται η δημιουργία ζεύγους ονόματος χρήστη – κωδικού, για τον ρόλο του διαχειριστή, μέσω γραμμής εντολών. Ο διαχειριστής στη συνέχεια μπορεί να ρυθμίσει όλη την εφαρμογή μέσω διαδικτυακής διεπαφής. Οι νέες ρυθμίσεις διαφέρουν στα ακόλουθα:

- Οι γενικές ρυθμίσεις, όπου καταχωρούνται γενικές πληροφορίες για τον δέκτη, όπως το όνομα, η τοποθεσία και μια φωτογραφία του, πλέον γίνονται μια φορά για όλο το σύστημα στον ισοσταθμιστή φορτίου.

- Οι ρυθμίσεις γενικών ζωνών συχνοτήτων, όπου καταχωρούνται τα όρια των διαθέσιμων ζωνών ολόκληρου του συστήματος, γίνονται στον ισοσταθμιστή φορτίου. Παράλληλα ρυθμίζονται οι ζώνες συχνοτήτων λειτουργίας της κάθε συσκευής SDR στον OpenWebRX εξυπηρετητή όπου αυτή είναι συνδεδεμένη. Το εύρος ζώνης των γενικών ζωνών συχνοτήτων πρέπει να είναι μικρότερο ή ίσο από το αντί-

στοιχο ρυθμισμένο στις συσκευές SDR των εξυπηρετητών, για να γίνει σωστά το ταίριασμα τους. Οι γενικές ζώνες συχνοτήτων είναι αυτές που εμφανίζονται για επιλογή στον χρήστη.

- Ρυθμίζονται και οι εξυπηρετητές και καταχωρούνται για κάθε ένα από αυτούς το όνομα, η διεύθυνση IP, η θύρα λειτουργίας και μια προαιρετική προτεραιότητα, που λαμβάνεται υπ' όψη από τον αλγόριθμο κατανομής. Επίσης προτεραιότητες εισάγονται προαιρετικά και στους OpenWebRX εξυπηρετητές, σε κάθε συσκευή SDR.

Στη λειτουργία της επισκόπησης, εκτός των συνδεδεμένων συσκευών SDR και των διαθέσιμων ζωνών συχνοτήτων, προστέθηκαν και οι εξυπηρετητές. Εμφανίζονται δεδομένα από τις ρυθμίσεις αλλά και δεδομένα που λαμβάνονται από την κατάσταση (state) τους. Εμφανίζονται για παράδειγμα οι ενεργές συνδέσεις ανά εξυπηρετητή, το υπολογιστικό φορτίο του και ο αριθμός των συνδεδεμένων συσκευών SDR.

4.3.8 Οι προσθήκες του Load Balancing OpenWebRX αναλυτικά

Προσθήκες ή και τροποποιήσεις έγιναν τόσο στον κώδικα του ισοσταθμιστή φορτίου, που είναι καινούριος, όσο και στον OpenWebRX εξυπηρετητή και τον κώδικα του πελάτη.
Στον ισοσταθμιστή φορτίου προστέθηκαν:

1) Αλγόριθμος κατανομής, που γράφτηκε από την αρχή, στο αρχείο distribution.py. Εκεί, στην κλάση Distribution βρίσκεται η συνάρτηση decision_algorithm() που αποφασίζει την κατανομή του πελάτη. Επίσης, βρίσκεται και μια δομή δεδομένων η servers_state_db, μαζί με τις μεθόδους διαχείρισης της και χρησιμοποιείται από τον αλγόριθμο κατανομής.

2) Νήμα λήψης της κατάστασης (state) των εξυπηρετητών μέσω ενδοεπικοινωνίας, που γράφτηκε από την αρχή, στο αρχείο receivestate.py. Αυτό το νήμα της κλάσης

ReceiveServersStateThread επικοινωνεί με τους εξυπηρετητές και λαμβάνει τα δεδομένα της αρχικής (initstate) και της τρέχουσας (curstate) κατάστασης τους.

3) Αποστολή της απόφασης κατανομής στον πελάτη, που γράφτηκε σε τμήμα του controller assets.py (CompiledAssetsControllerVars). Εκεί συμπληρώνονται δυναμικά οι τιμές των μεταβλητών του αρχείου websocket-vars.js. Αυτό το αρχείο με τις μεταβλητές λαμβάνεται από τον πελάτη και καθορίζει την κατανομή του σε εξυπηρετητή και συσκευή SDR.

4) HTTP εξυπηρετητής, γράφτηκε στο πνεύμα του εξυπηρετητή του αρχικού OpenWebRX. Προσαρμόστηκαν τα αρχεία __main__.py και httproutes.py.

5) Ρυθμίσεις μέσω διαδικτυακής διεπαφής (web interface), με βάση τον τρόπο που είχαν γραφεί στο αρχικό πρόγραμμα. Προσαρμόστηκαν το αρχείο settings.html, οι controllers των settings __init__.py, general.py, profiles.py, sdrs.py και ο νέος servers.py, το αρχείο details.py και τα config core.py και dynamic.py.

Στον κώδικα της πλευράς του πελάτη προστέθηκαν και τροποποιήθηκαν:

1) Στον φάκελο htdocs, τροποποιήθηκε το αρχείο openwebrx.js και δημιουργήθηκε το websocket-vars.js. Στο openwebrx.js έγιναν αλλαγές σχετικά με:

   - Την λήψη και εμφάνιση των διαθέσιμων ζωνών συχνοτήτων και συσκευών SDR. Αυτά δεν λαμβάνονται πλέον από τον OpenWebRX εξυπηρετητή. Οι διαθέσιμες ζώνες συχνοτήτων (profiles) λαμβάνονται από το αρχείο websocket-vars.js στη μεταβλητή available_profiles, που καθορίζεται από τον ισοσταθμιστή φορτίου. Εμφανίζονται από τη συνάρτηση show_available_profiles() σε έναν επιλογέα.

   - Το άνοιγμα της WebSocket σύνδεσης με τον αποφασισμένο εξυπηρετητή decided_server, που γίνεται στη συνάρτηση open_websocket(). Επίσης, προστέθηκε η αποστολή του κατάλληλου μηνύματος επιλογών sdrselection μέσω

της σύνδεσης, στη συνάρτηση on_ws_opened(), με παραμέτρους τα decided_sdrid και decided_profileid.

- Την αλλαγή ζώνης συχνοτήτων και ίσως συσκευής SDR και εξυπηρετητή, μετά από επιλογή του χρήστη, στη συνάρτηση sdr_profile_changed(). Εκεί αρχικά αναμένεται το κλείσιμο της τρέχουσας WebSocket σύνδεσης (αν υπάρχει) με τη συνάρτηση wait_ws_to_close() και μετά ζητείται με ασύγχρονο GET αίτημα από τον ισοσταθμιστή φορτίου να κατανείμει τον πελάτη σε νέο εξυπηρετητή και συσκευή. Τέλος, ανοίγεται νέα σύνδεση με κλήση της open_websocket().

2) Τροποποιήθηκαν και τα αρχεία index.html, προκειμένου να ανανεωθεί η λίστα των συγγραφέων και το openwebrx-header.css, για να αλλάξει το χρώμα και να γίνει ευδιάκριτο το κείμενο της επικεφαλίδας.

Στον κώδικα του OpenWebRX εξυπηρετητή προστέθηκαν και τροποποιήθηκαν:

1) Η καταγραφή της κατάστασης (state) του εξυπηρετητή φτιάχτηκε από την αρχή στο αρχείο controllers/state.py. Επίσης, αυτή η κατάσταση έγινε διαθέσιμη στον ισοσταθμιστή φορτίου μέσα από δύο νέα HTTP άκρα initstate.json και curstate.json, στο αρχείο http.py. Φτιάχτηκαν δύο είδη State controller, το InitialStateController και το CurrentStateController. Αυτά λαμβάνουν τις πληροφορίες τους από το αρχείο Config, τις ρυθμίσεις των συσκευών SDR SdrService και το νήμα που μετράει τη χρήση της υπολογιστικής μονάδας CpuUsageThread.

2) Τροποποιήθηκε ο χειρισμός της WebSocket σύνδεσης στο αρχείο connection.py:

- Προστέθηκε νέο μήνυμα στη συνάρτηση handleTextMessage(), το sdrselection, με το οποίο επιλέγεται η συσκευή SDR και η ζώνη συχνοτήτων που αυτή θα λειτουργήσει.

- Αλλαγές έγιναν στη συνάρτηση SetSdr(), ώστε να μην ορίζεται προεπιλεγμένη συσκευή SDR, αλλά να χρησιμοποιείται αυτή που αποφασίζεται από τον ισοσταθμιστή.

- Η αλλαγή συσκευής SDR γίνεται αποκλειστικά μέσω του ισοσταθμιστή.

- Οι πληροφορίες του δέκτη και οι διαθέσιμες ζώνες συχνοτήτων στέλνονται μόνο από τον ισοσταθμιστή φορτίου.

3) Προστέθηκε μεταβλητή προτεραιότητας για τις συσκευές SDR, για συνυπολογισμό της, από τον αλγόριθμο κατανομής. Τροποποιήθηκε το αρχείο controller/settings/sdr.py, όπου στην κλάση NewSdrDeviceController προστέθηκε αριθμητική είσοδος για την προτεραιότητα. Επίσης, η προτεραιότητα προστέθηκε και στο αρχείο source/__init__.py στις μεθόδους getDeviceInputs() και getDeviceMandatoryKeys().

4) Τροποποιήθηκε το νήμα που καταγράφει τη χρήση της υπολογιστικής μονάδας, στο αρχείο cpu.py. Εκεί λαμβάνεται ο μέσος όρος των 3 τελευταίων μετρήσεων για πιο σταθερό και χρήσιμο αποτέλεσμα. Προστέθηκε και κλείδωμα συγχρονισμού και η μέθοδος getLatestCpu() για λήψη των μετρήσεων από τον ελεγκτή της κατάστασης του εξυπηρετητή (state controller).

5) Τροποποιήθηκε η εξυπηρέτηση του HTTP άκρου της αρχικής σελίδας «/», ώστε να ανακατευθύνει ο IndexController στη σελίδα των ρυθμίσεων. Αυτό έγινε έτσι, επειδή η εξυπηρέτηση των HTTP αιτημάτων για στατικά αρχεία, γίνεται αποκλειστικά από τον ισοσταθμιστή φορτίου. Ενώ οι ρυθμίσεις εξακολουθούν να γίνονται διαδικτυακά και στον κάθε εξυπηρετητή.

6) Τροποποιήθηκε η πολιτική σχετικά με το μόνιμο σφάλμα μιας συσκευής SDR. Τώρα δίνεται ευκαιρία το σφάλμα να διορθωθεί και να ξαναχρησιμοποιηθεί η συσκευή. Αυτό γίνεται παράλληλα με τη διαχείριση σφάλματος στην πλευρά του πελάτη. Άλλαξαν

τα αρχεία source/__init__.py στη συνάρτηση fail() και το sdr.py, στη συνάρτηση onFail().

5. Πρακτική εφαρμογή στο εργαστήριο

5.1 Οι συσκευές SDR του εργαστηρίου

Στο Εργαστήριο Δικτύων του Εθνικού Μετσόβιου Πολυτεχνείου υπάρχουν πολλές διαθέσιμες συσκευές SDR. Αυτές αξιοποιήθηκαν από αυτήν την εργασία, για να στηθεί ένας δέκτης OpenWebRX. Συνολικά, χρησιμοποιούνται 7 συσκευές, οι οποίες είναι:

- Ettus USRP B210 (x2)
- Ettus USRP N200 (x2)
- Ettus USRP2 (x2)
- RTL-SDR (x1)

Στη συνέχεια θα παρουσιαστούν τα χαρακτηριστικά αυτών των συσκευών:

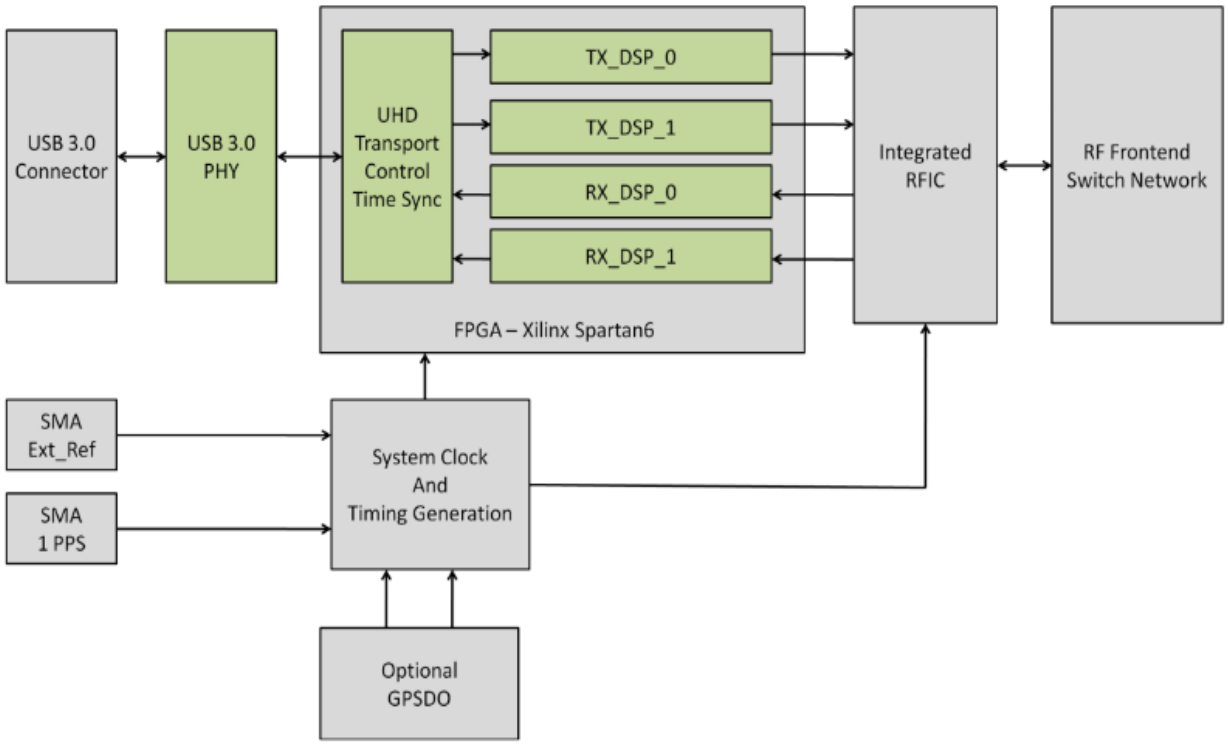
5.1.1 Ettus USRP B210

Εικόνα 5.1 *Η πλακέτα του B210* [78]

Το Ettus USRP B210 διαθέτει τα ακόλουθα χαρακτηριστικά, σύμφωνα με το [78]:

1) Είναι ένας πομποδέκτης SDR με δυο κανάλια, δηλαδή 2 εξόδους εκπομπής και 2 εισόδους λήψης, με δυνατότητα ταυτόχρονης λειτουργίας και στις δυο κατευθύνσεις.
2) Αποτελείται από μια αυτοτελή πλακέτα και δουλεύει στο εύρος συχνοτήτων από 70 MHz έως 6 GHz.
3) Έχει δυνατότητα να χειριστεί σήματα με εύρος ζώνης έως 56 MHz.
4) Συνδέεται σε υπολογιστή μέσω USB 3.
5) Διαθέτει προγραμματίσιμο FPGA.
6) Ο μετατροπέας αναλογικού σε ψηφιακό σήμα (ADC) και ο μετατροπέας ψηφιακού σε αναλογικό (DAC), έχουν ανάλυση 12 bit για κάθε δείγμα.
7) Διαθέτει ακροδέκτες GPIO.
8) Υποστηρίζει σύμφωνη MIMO λειτουργία.
9) Μπορεί να λάβει εξωτερικό σήμα ρολογιού.
10) Για εφαρμογές που απαιτούν ακρίβεια στο συγχρονισμό, διατίθεται και προαιρετικό σύστημα ταλαντωτή συγχρονισμένου μέσω δορυφορικού σήματος GPS (GPSDO).
11) Επικοινωνεί με τον υπολογιστή μέσω του οδηγού συσκευών UHD.

Στη συνέχεια φαίνεται το διάγραμμα με την αρχιτεκτονική του *B210 SDR*.



Εικόνα 5.2 *Αρχιτεκτονική του B210 [23]*

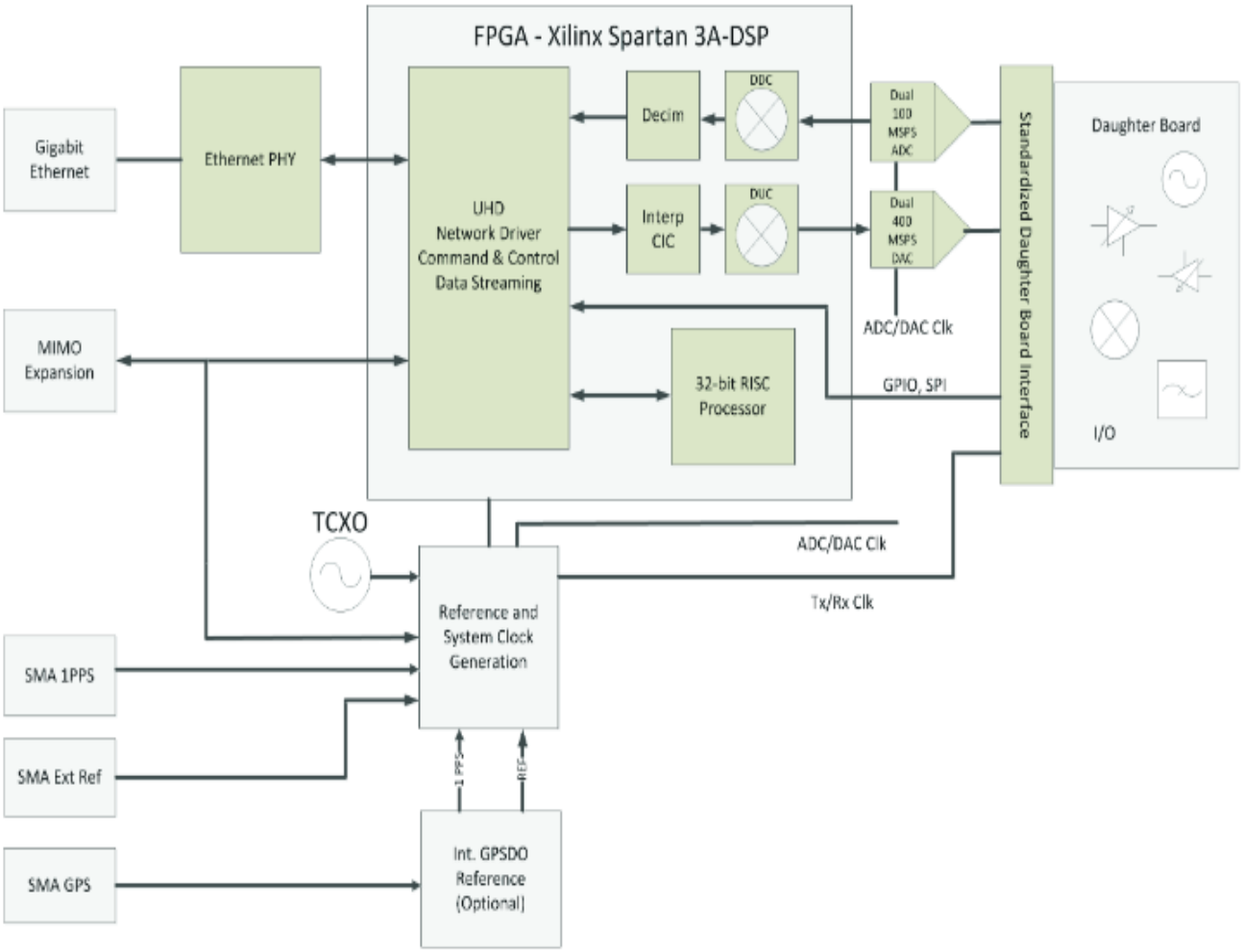
5.1.2 Ettus USRP N200

Εικόνα 5.3 *Φωτογραφία του N200 [79]*

Το Ettus USRP N200 διαθέτει τα ακόλουθα χαρακτηριστικά, σύμφωνα με το [79]:

1) Είναι ένας πομποδέκτης SDR με αρθρωτή σχεδίαση. Συνδυάζεται με μια θυγατρική κάρτα εμπροσθοφυλακής (RF front end). Ανάλογα με τη θυγατρική κάρτα λειτουργεί και σε διαφορετική ζώνη συχνοτήτων.
2) Γενικά μπορεί να λειτουργήσει σε εύρος συχνοτήτων από τη DC μέχρι 6 GHz.
3) Στο εργαστήριο έχει εγκατασταθεί η WBX κάρτα, για εκπομπή και λήψη σε συχνότητες από 50 έως 2200 MHz.
4) Η σύνδεση με τον υπολογιστή γίνεται μέσω καλωδίου Ethernet σε ταχύτητα έως 1 Gbps.
5) Το εύρος ζώνης των σημάτων που μπορεί να χειριστεί εξαρτάται από πολλούς παράγοντες. Ο μετατροπέας αναλογικού σε ψηφιακό σήμα (ADC) υποστηρίζει δειγματοληψία έως και 100 Msps. Η θυγατρική κάρτα υποστηρίζει μέχρι 40 MHz. Όμως αυτό που τελικά περιορίζει, είναι η σύνδεση με τον υπολογιστή. Εάν χρησιμοποιηθούν δείγματα I/Q μεγέθους 16 bit, τότε μπορούν να μεταφερθούν δείγματα με ρυθμό 25Msps. Τελικά μπορεί να χειριστεί σήματα μέχρι περίπου 20 - 25 MHz εύρος, όπως αναφέρεται στο [80].
6) Έχει καλό δυναμικό εύρος (dynamic range) στον διπλό μετατροπέα αναλογικού σε ψηφιακό σήμα (ADC), χρησιμοποιώντας 14 bit για κάθε δείγμα. Στον διπλό μετατροπέα ψηφιακού σε αναλογικό (DAC) χρησιμοποιούνται δείγματα μεγέθους 16 bit.
7) Διαθέτει προγραμματίσιμο FPGA και επεξεργαστή.
8) Υποστηρίζει λειτουργία MIMO.
9) Μπορεί να λάβει εξωτερικό σήμα ρολογιού.
10) Για εφαρμογές που απαιτούν ακρίβεια στον συγχρονισμό, διατίθεται και προαιρετικό σύστημα ταλαντωτή συγχρονισμένου μέσω δορυφορικού σήματος GPS (GPSDO).
11) Επικοινωνεί με τον υπολογιστή μέσω του οδηγού συσκευών UHD.

Ακολουθεί σχήμα που φαίνεται η αρχιτεκτονική του N200. Σημειώνεται ότι απαιτείται σύνδεση με θυγατρική κάρτα, από την οποία λαμβάνεται το ραδιοσήμα (RF signal).



Εικόνα 5.4 *Αρχιτεκτονική του N200 [81]*

5.1.3 Ettus USRP2

Εικόνα 5.5 *USRP2 [82]*

Το USRP2 είναι παλαιότερο μοντέλο και δεν κατασκευάζονται πλέον καινούρια κομμάτια.

Το Ettus USRP2 διαθέτει τα ακόλουθα χαρακτηριστικά, σύμφωνα με το [82]*:*

1) Είναι ένας πομποδέκτης SDR με αρθρωτή σχεδίαση. Συνδυάζεται με μια θυγατρική κάρτα εμπροσθοφυλακής (RF front end). Ανάλογα με τη θυγατρική κάρτα λειτουργεί και σε διαφορετική ζώνη συχνοτήτων.
2) Γενικά μπορεί να λειτουργήσει σε εύρος συχνοτήτων από τη DC μέχρι 6 GHz.
3) Στο εργαστήριο έχει εγκατασταθεί η WBX κάρτα, για εκπομπή και λήψη σε συχνότητες από 50 έως 2200 MHz.
4) Η σύνδεση με τον υπολογιστή γίνεται μέσω καλωδίου Ethernet σε ταχύτητα έως 1 Gbps.
5) Όπως και στην περίπτωση του N200 τελικά μπορεί να χειριστεί σήματα εύρους 20 – 25 MHz.

6) Στον μετατροπέα αναλογικού σε ψηφιακό σήμα (ADC) χρησιμοποιούνται 14 bit για κάθε δείγμα. Στον μετατροπέα ψηφιακού σε αναλογικό (DAC) χρησιμοποιούνται δείγματα μεγέθους 16 bit.
7) Διαθέτει προγραμματίσιμο FPGA.
8) Υποστηρίζει λειτουργία MIMO.
9) Μπορεί να λάβει εξωτερικό σήμα ρολογιού.
10) Επικοινωνεί με τον υπολογιστή μέσω του οδηγού συσκευών UHD.

5.1.4 RTL-SDR

Εικόνα 5.6 *RTL-SDR [83]*

Υπάρχουν πολλά διαφορετικά μοντέλα RTL-SDR με μικρές διαφορές μεταξύ τους. Αυτό που χρησιμοποιείται στο εργαστήριο βασίζεται στο ολοκληρωμένο R820T.

Το RTL-SDR διαθέτει τα ακόλουθα χαρακτηριστικά, σύμφωνα με το [6]:

1) Λειτουργεί σε συχνότητες από 24 μέχρι 1766 MHz.
2) Έχει εύρος ζώνης 2.4 MHz.
3) Μπορεί μόνο να λάβει σήματα και δεν στέλνει.
4) Συνδέεται στον υπολογιστή μέσω USB.
5) Χρησιμοποιεί 8 bits για κάθε δείγμα.
6) Είναι χαμηλού κόστους.
7) Επικοινωνεί με τον υπολογιστή μέσα από δικό του οδηγό συσκευής.

5.2 Οι υπολογιστές του εργαστηρίου

Η εφαρμογή Load Balancing OpenWebRX αξιοποιεί αρκετούς υπολογιστές του εργαστηρίου, όπου τρέχουν οι OpenWebRX εξυπηρετητές και ο Load Balancer. Συγκεκριμένα χρησιμοποιούνται:

- Intel NUC mini PC (x2)
- Desktop PC
- Virtual Machine (εικονικό μηχάνημα)
- Raspberry Pi

Ο Load Balancer τρέχει στο εικονικό μηχάνημα, όλοι οι άλλοι υπολογιστές χρησιμοποιούνται για τους εξυπηρετητές.

Πέρα από τον σταθερό υπολογιστή, οι υπόλοιποι είναι ειδικοί υπολογιστές μικρού μεγέθους. Θα παρουσιαστούν παρακάτω τα χαρακτηριστικά τους.

5.2.1 Intel NUC mini PC

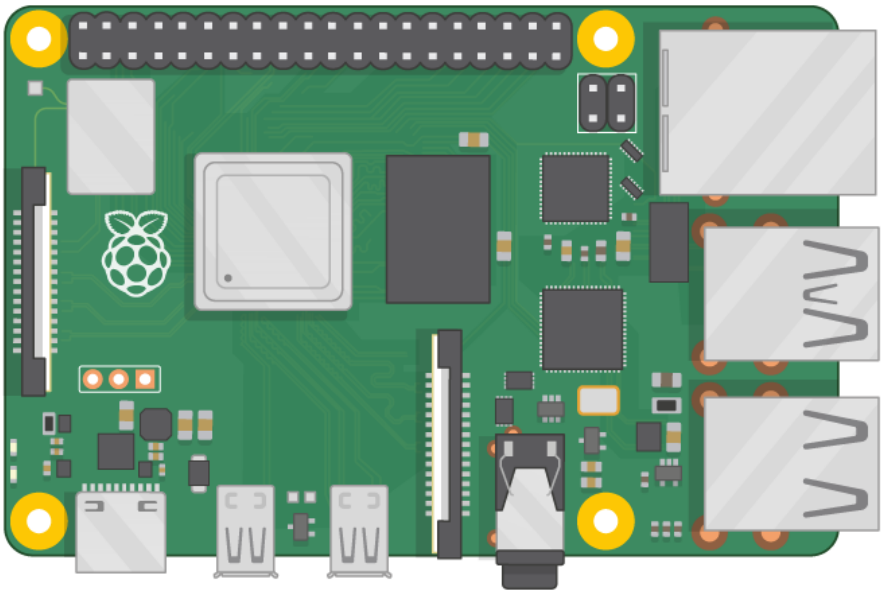
Εικόνα 5.7 *Intel NUC mini PC [84]*

Το Intel NUC mini PC διαθέτει τα ακόλουθα χαρακτηριστικά σύμφωνα με τα [85,86]:

1) Είναι ένας μικρού μεγέθους υπολογιστής.
2) Αγοράζεται με τα απολύτως απαραίτητα: μητρική πλακέτα με θήκη και ανεμιστήρα και ενδεχομένως με επεξεργαστή.
3) Έχει ικανοποιητικές υπολογιστικές επιδόσεις.
4) Προσφέρει ευελιξία, καθώς ο καταναλωτής του προσθέτει δίσκο SSD, μνήμη και λειτουργικό σύστημα, σύμφωνα με τις ανάγκες του.
5) Τα τελευταία μοντέλα διαθέτουν Ethernet, Wifi, Bluetooth, GPU, θύρες Display, έως 8 θύρες USB και Thunderbolt.
6) Η παραγωγή της πρώτης γενιάς, ξεκίνησε το 2013.
7) Το 2023 βγήκε ανακοίνωση από την Intel ότι σταματάει την παραγωγή τους.

Χρησιμοποιούνται δυο Intel NUC mini PC του εργαστηρίου.

5.2.2 Raspberry Pi



Εικόνα 5.8 *Raspberry Pi [87]*

Το Raspberry Pi διαθέτει τα ακόλουθα χαρακτηριστικά σύμφωνα με τα [88,89]:

1) Είναι μικρός υπολογιστής χαμηλού κόστους.
2) Κατασκευάζεται από το Raspberry Pi Foundation, στην Αγγλία (UK).
3) Διαθέτει πολυπύρηνο επεξεργαστή, επιλογή μεγέθους μνήμης και ενσωματωμένη GPU.
4) Διαθέτει GPIO ακροδέκτες, Ethernet, WiFi, Bluetooth, 4 θύρες USB, HDMI.
5) Το καινούριο μοντέλο RPi 5 υποστηρίζει και απευθείας σύνδεση με M.2 SSD δίσκο.
6) Διατίθενται διάφορες θήκες και πλακέτες επέκτασης.
7) Χρησιμοποιείται συχνά ως εξυπηρετητής ή και σε εργασίες ρομποτικής, μιας και υποστηρίζει και κάμερα.

Το RPi που χρησιμοποιείται, είναι έκδοσης 4 και ανήκει στην προσωπική μου συλλογή.

5.2.3 Virtual Machine (εικονικό μηχάνημα)

Εικόνα 5.9 *VMware virtualization [90]*

Χρησιμοποιείται ένα εικονικό μηχάνημα για να τρέχει εκεί ο Load Balancer.

Το εικονικό μηχάνημα διαθέτει τα ακόλουθα χαρακτηριστικά σύμφωνα με τα [91,92]:

1) Βασίζεται σε λογισμικό που τρέχει σε ένα φυσικό μηχάνημα.
2) Το λογισμικό αυτό, ο επόπτης, είναι στην περίπτωση αυτή το vmware.
3) Ανατίθενται πόροι στο εικονικό μηχάνημα, όπως επεξεργαστές και μνήμη, σύμφωνα με τις ανάγκες.
4) Διαθέτει το δικό του λειτουργικό σύστημα.
5) Τρέχει λίγο πιο αργά σε σχέση με ένα φυσικό μηχάνημα.
6) Η ίδια τεχνολογία χρησιμοποιείται και στο νέφος (cloud).

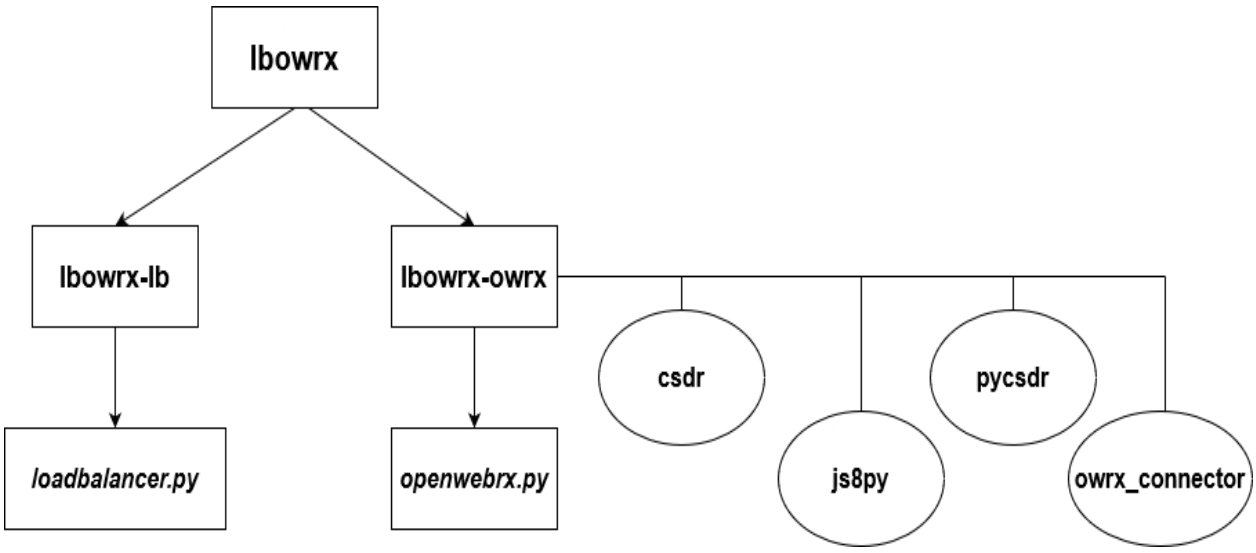
5.3 Δίκτυο

Όλα τα μηχανήματα εκτός από το Raspberry Pi είναι συνδεδεμένα σε μεταγωγέα του εργαστηρίου, στο ίδιο υποδίκτυο με τις δικτυακές συσκευές SDR. Διαθέτουν δημόσιες διευθύνσεις IP για να επικοινωνούν μαζί τους οι πελάτες. Έχουν ρυθμιστεί και τοίχοι προστασίας (firewalls).

5.4 Λογισμικό που χρησιμοποιήθηκε στο εργαστήριο

Όλοι οι OpenWebRX εξυπηρετητές τρέχουν σε υπολογιστές με λειτουργικό σύστημα Ubuntu 22, την πιο πρόσφατη μακροχρόνια υποστηριζόμενη έκδοση (LTS) αυτή τη στιγμή, σύμφωνα με το [93]. Το πρόγραμμα OpenWebRX τρέχει αποκλειστικά σε Linux. Υπάρχουν διαθέσιμα: αποθετήριο πακέτων για Debian συστήματα, εικόνα για κάρτα μνήμης Raspberry Pi, εικόνα για Docker. Ωστόσο στο εργαστήριο το πρόγραμμα χτίστηκε από τον πηγαίο κώδικα, μαζί με τις αλλαγές του Load Balancing OpenWebRX. Για την εγκατάσταση απαιτείται Python με έκδοση >= 3.7 και κάποια άλλα προαπαιτούμενα πακέτα που αναφέρονται στη σελίδα Manual installation στο Github, που βρίσκεται στο [94]. Απαιτείται επίσης εγκατάσταση των σχετικών με το OpenWebRX προγραμμάτων csdr, pycsdr, js8py και owrx_connector.

Στον κώδικα του Load Balancing OpenWebRX, κάτω από το lbowrx περιλαμβάνονται δύο φάκελοι: ο lbowrx-lb για τον ισοσταθμιστή φορτίου και ο lbowrx-owrx για τον τροποποιημένο OpenWebRX εξυπηρετητή. Στους φακέλους αυτούς περιλαμβάνονται και τα αρχεία που εκτελούνται, δηλαδή το loadbalancer.py και το openwebrx.py αντίστοιχα. Επίσης στον φάκελο του τροποποιημένου OpenWebRX, περιλαμβάνεται και ο κώδικας των σχετικών προγραμμάτων csdr, pycsdr, js8py και owrx_connector.



Σχήμα 5.2 *Φάκελοι του Load Balancing OpenWebRX*

Για τον χειρισμό των διάφορων συσκευών SDR πέρα από το RTL-SDR, απαιτείται και το SoapySdr, που αναφέρθηκε στο [44]. Στο εργαστήριο εγκαταστάθηκε η soapysdr0.8 έκδοση του. Το SoapySdr περιλαμβάνει και module της UHD, δηλαδή του οδηγού των Ettus USRP συσκευών SDR, που βρίσκονται στο εργαστήριο και περιγράφηκαν παραπάνω.

Ειδικά στο UHD, που παρουσιάζεται στο [95], κατά τη λειτουργία των δικτυακά συνδεδεμένων συσκευών SDR, εμφανίζεται ορισμένες φορές ένα σφάλμα λογισμικού. Αυτό είναι το «RuntimeError: fifo ctrl timed out looking for acks». Για να λυθεί, δοκιμάστηκε να εγκατασταθεί παλαιότερη έκδοση, με χτίσιμο όλων των απαιτούμενων από τον πηγαίο κώδικα. Παρόλα αυτά, το σφάλμα συνέχισε να εμφανίζεται μερικές φορές. Με τη διαχείριση σφάλματος που προστέθηκε στο Load Balancing OpenWebRX οι συνέπειες του σφάλματος ελαχιστοποιούνται.

Ακόμα, για την έναρξη, τη λειτουργία και τον τερματισμό των προγραμμάτων του Load Balancer και του OpenWebRX χρησιμοποιείται το systemd. Όπως περιγράφεται στο [96], το systemd διαχειρίζεται υπηρεσίες (services) στο Linux. Σε κατάλληλη τοποθεσία έχουν δημιουργηθεί αρχεία systemd services για τον ισοσταθμιστή φορτίου και τον OpenWebRX εξυπηρετητή. Με την εντολή systemctl έχουν ενεργοποιηθεί αυτές οι υπηρεσίες και τρέχουν τα προγράμματα. Έτσι επιτυγχάνεται η λειτουργία των προγραμμάτων και μετά που θα έχει κλείσει η ssh σύνδεση με το μηχάνημα. Επίσης, έχει ρυθμιστεί αυτόματη εκκίνηση των προγραμμάτων μετά την εκκίνηση ή επανεκκίνηση του υπολογιστή όπου τρέχει η εφαρμογή, σύμφωνα με τον οδηγό του [97].

Επιπρόσθετα, για τον εξυπηρετητή που τρέχει στο Raspberry Pi και βρίσκεται σε οικιακό περιβάλλον χρησιμοποιήθηκε η υπηρεσία δυναμικού συστήματος ονοματοδοσίας (DDNS) no-ip, που βρίσκεται στο [98]. Με αυτόν τον τρόπο, χωρίς σταθερή διεύθυνση IP, ο εξυπηρετητής έγινε προσβάσιμος από το διαδίκτυο και μπορούν να συνδέονται σε αυτόν οι πελάτες, αλλά και να επικοινωνεί με τον ισοσταθμιστή φορτίου. Για να λειτουργήσει αυτό, χρειάστηκε και μια διεύθυνση IP, χωρίς μετάφραση διευθύνσεων από τον πάροχο (εκτός του CGNAT).

5.5 Κεραίες του εργαστηρίου

Στο εργαστήριο χρησιμοποιούνται μικρές κεραίες, απευθείας πάνω σε κάθε συσκευή SDR, κατάλληλες για τις μπάντες VHF και UHF. Οι συσκευές SDR βρίσκονται σε εσωτερικό χώρο. Λαμβάνονται επιτυχώς, ισχυρά σήματα στην VHF, όπως η ραδιοφωνία FM. Όμως, κάποια αδύναμα σήματα στην VHF δεν είναι δυνατό να ληφθούν. Επίσης, δεν μπορούν να ληφθούν σήματα στην HF. Μελλοντικά, θα ήταν χρήσιμη μια εξωτερική εγκατάσταση κεραίας.

5.6 Ρυθμίσεις ζωνών συχνοτήτων και συσκευών στο Load Balancing OpenWebRX

Στο Load Balancing OpenWebRX πραγματοποιήθηκε η ρύθμιση των διαθέσιμων, για τον χρήστη, ζωνών συχνοτήτων και των συσκευών SDR που χρησιμοποιούνται. Επιλέχθηκαν δύο ζώνες λειτουργίας: η ραδιοφωνική εκπομπή FM και η Airband (μπάντα σημάτων επικοινωνίας των αεροπλάνων με τον πύργο ελέγχου). Ρυθμίστηκαν και οι συσκευές SDR ώστε να καλύπτουν αυτές τις ζώνες αποδοτικά.

Η ραδιοφωνική ζώνη εκπομπής FM χρησιμοποιεί την ομώνυμη αναλογική διαμόρφωση συχνότητας, με μεγάλο εύρος (WFM). Τοποθετείται στη ζώνη VHF, μεταξύ 87.5 και 108 MHz, στις ίδιες συχνότητες σχεδόν σε όλο τον κόσμο. Είναι κατάλληλη για εκπομπή μουσικής και στερεοφωνικού ήχου. Μπορεί να περιλαμβάνεται και σύντομο κείμενο μέσω του RDS. Το εύρος ζώνης ενός σταθμού είναι περίπου 200 KHz. Οι σταθμοί μπορεί να εκπέμπουν πολύ ισχυρό σήμα, ακόμα και της τάξης των δεκάδων χιλιάδων Watt. Η εμβέλεια ενός σταθμού εξαρτάται εκτός από την ισχύ του και από την τοποθεσία του σημείου εκπομπής. Ένα ψηλό σημείο με ορίζοντα χωρίς εμπόδια, επιτρέπει στο σήμα να φτάσει μακριά. Η λήψη είναι εύκολη, καθώς αρκεί μια απλή ραδιοφωνική συσκευή, που σήμερα μπορεί να έχει μέγεθος τσέπης.

Ιστορικά η εκπομπή FM ακολούθησε την εκπομπή AM. Η εκπομπή AM άρχισε πειραματικά στις αρχές του 20^{ου} αιώνα. Η εκπομπή FM ξεκίνησε δοκιμαστικά τη δεκαετία του 1930. Στην Ελλάδα η πρώτη ραδιοφωνική AM εκπομπή έγινε τη δεκαετία του 1920. Οι

FM ραδιοφωνικοί σταθμοί στην Ελλάδα ξεκίνησαν αργότερα, μετά την εμφάνιση της τηλεόρασης. Οι πρώτοι πειρατικοί σταθμοί, άρχισαν να εκπέμπουν στα FM, μαζί με το Εθνικό Ίδρυμα Ραδιοφωνίας Τηλεόρασης, τη δεκαετία του 1970. Το 1987 αυτή η ζώνη συχνοτήτων ανατέθηκε και σε ιδιωτικούς εμπορικούς και δημόσιους σταθμούς. Δείτε περισσότερες λεπτομέρειες στα [55,99].

Η ραδιοφωνική ζώνη εκπομπής FM προτιμήθηκε αρχικά, λόγω της δημοφιλίας της. Ακόμα, λόγω του ισχυρού σήματος και της εύκολης λήψης των σταθμών της. Επίσης, οι συσκευές SDR του εργαστηρίου έχουν τη δυνατότητα λήψης ολόκληρης της ζώνης, εύρους 20.5 MHz.

Η άλλη μπάντα που λαμβάνεται από τις συσκευές του εργαστηρίου, είναι η Airband. Είναι δηλαδή η μπάντα σημάτων επικοινωνίας των αεροπλάνων με τον πύργο ελέγχου των αεροδρομίων. Αυτή τοποθετείται στην VHF μπάντα μεταξύ 108 – 137 MHz. Το κάτω τμήμα της 108 – 118 MHz, που συνορεύει με τη μπάντα του ραδιοφώνου FM, χρησιμοποιείται για ραδιοπλοήγηση. Το άνω τμήμα της 118 – 137 MHz χρησιμοποιείται για έλεγχο εναέριας κυκλοφορίας. Η διαμόρφωση που χρησιμοποιείται είναι διαμόρφωση πλάτους AM. Το σήμα είναι πολύ ισχυρό και η εμβέλεια του φτάνει έως και 300 χιλιόμετρα περίπου, με καλό καιρό. Δείτε περισσότερες λεπτομέρειες στο [100].

Κάθε αεροδρόμιο έχει τις δικές του συχνότητες επικοινωνίας με τα αεροπλάνα. Πραγματοποιήθηκε αναζήτηση των σημάτων με μια συσκευή SDR προκειμένου να βρεθούν οι συχνότητες του αεροδρομίου της Αθήνας, Ελευθέριος Βενιζέλος (ATH). Στις συχνότητες που βρέθηκαν, σχετικές με τον έλεγχο εναέριας κυκλοφορίας, ακούγεται κατά διαστήματα συνομιλία του πύργου ελέγχου με τους πιλότους των αεροπλάνων, με χρήση του φωνητικού αλφαβήτου. Ο χρήστης της εφαρμογής έχει τη δυνατότητα σε συνδυασμό με την ακρόαση της ομιλίας στην Airband, να παρακολουθεί και την κίνηση των σχετικών αεροπλάνων σε ζωντανή μετάδοση από μια άλλη εφαρμογή, όπως το https://www.flightradar24.com [101].

Οι συσκευές SDR ρυθμίστηκαν ώστε να καλύπτουν τις παραπάνω μπάντες. Η μπάντα συχνοτήτων των FM καλύφθηκε σε δύο τμήματα, από 87.5 – 98 MHz και 98 – 108 MHz. Αυτό έγινε ρυθμίζοντας στις παραπάνω συσκευές SDR, της Ettus, ρυθμό δειγματοληψίας στα 12.5 Msps, προκειμένου να επιτευχθεί οικονομία υπολογιστικών πόρων. Αυτό προτιμήθηκε γιατί αν μια συσκευή κάλυπτε ολόκληρη την FM μπάντα, όπως θα μπορούσε, θα απαιτούνταν πολλαπλάσιοι υπολογισμοί για την αποδιαμόρφωση, με αποτέλεσμα να εξυπηρετούσε λιγότερους πελάτες. Χρησιμοποιώντας περισσότερες συσκευές SDR, με

μικρότερο εύρος ζώνης μεγιστοποιείται ο αριθμός των πελατών που μπορούν να εξυπηρετηθούν.

Η συσκευή που εξυπηρετεί τη μπάντα της Airband είναι το RTL-SDR. Έχουν οριστεί τρεις ζώνες συχνοτήτων λειτουργίας, που αντιστοιχούν στις συχνότητες των σημάτων που βρέθηκαν. Το ρυθμισμένο εύρος κάθε ζώνης είναι 2.4 MHz.

5.7 Μετρικές λειτουργίας του Load Balancing OpenWebRX στο εργαστήριο

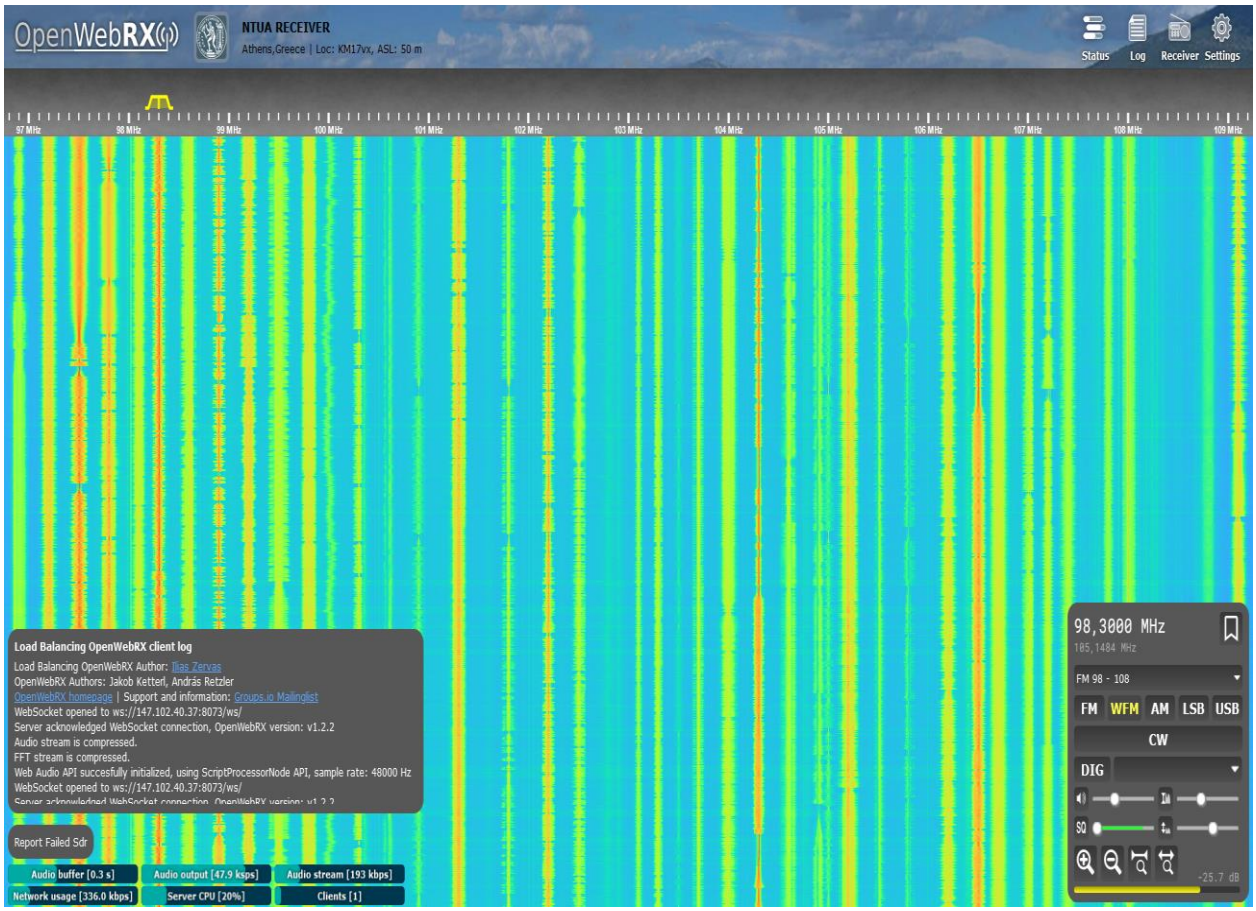
Μετρήθηκαν δύο σημαντικοί παράμετροι του συστήματος. Η χρήση του δικτύου και ο μέγιστος αριθμός των υποστηριζόμενων πελατών. Σχετικά με τη χρήση του δικτύου έχουμε κίνηση προς τον OpenWebRX εξυπηρετητή και κίνηση προς τον πελάτη. Κίνηση προς τον εξυπηρετητή έχουμε στην περίπτωση των δικτυακών συσκευών SDR, όπως το N200 και το USRP2, που στέλνουν τα IQ δείγματα. Επειδή ο ρυθμός δειγματοληψίας έχει ρυθμιστεί στα 12.5 Msps, η κίνηση ανέρχεται στα 400 Mbps, κάτι που υποστηρίζεται από την Gigabit Ethernet σύνδεση. Όσον αφορά την κίνηση προς τον πελάτη, στην περίπτωση του στερεοφωνικού ήχου FM και του καταρράκτη, απαιτούνται περίπου 300Kbps για κάθε πελάτη.

Ο μέγιστος αριθμός πελατών που μπορεί να εξυπηρετηθεί από το σύστημα του εργαστηρίου αυτή τη στιγμή είναι περίπου 30 πελάτες. Αν αναλογιστούμε ότι ένας μοναδικός εξυπηρετητής (από αυτούς που υπάρχουν στο εργαστήριο) μπορεί να εξυπηρετήσει λιγότερους από 10 πελάτες, όταν λειτουργεί σε πλήρεις δυνατότητες, η βελτίωση είναι μεγάλη (περισσότεροι από τριπλάσιοι πελάτες). Με την προσθήκη και άλλων υπολογιστών - εξυπηρετητών, ο αριθμός των πελατών αυξάνεται και άλλο.

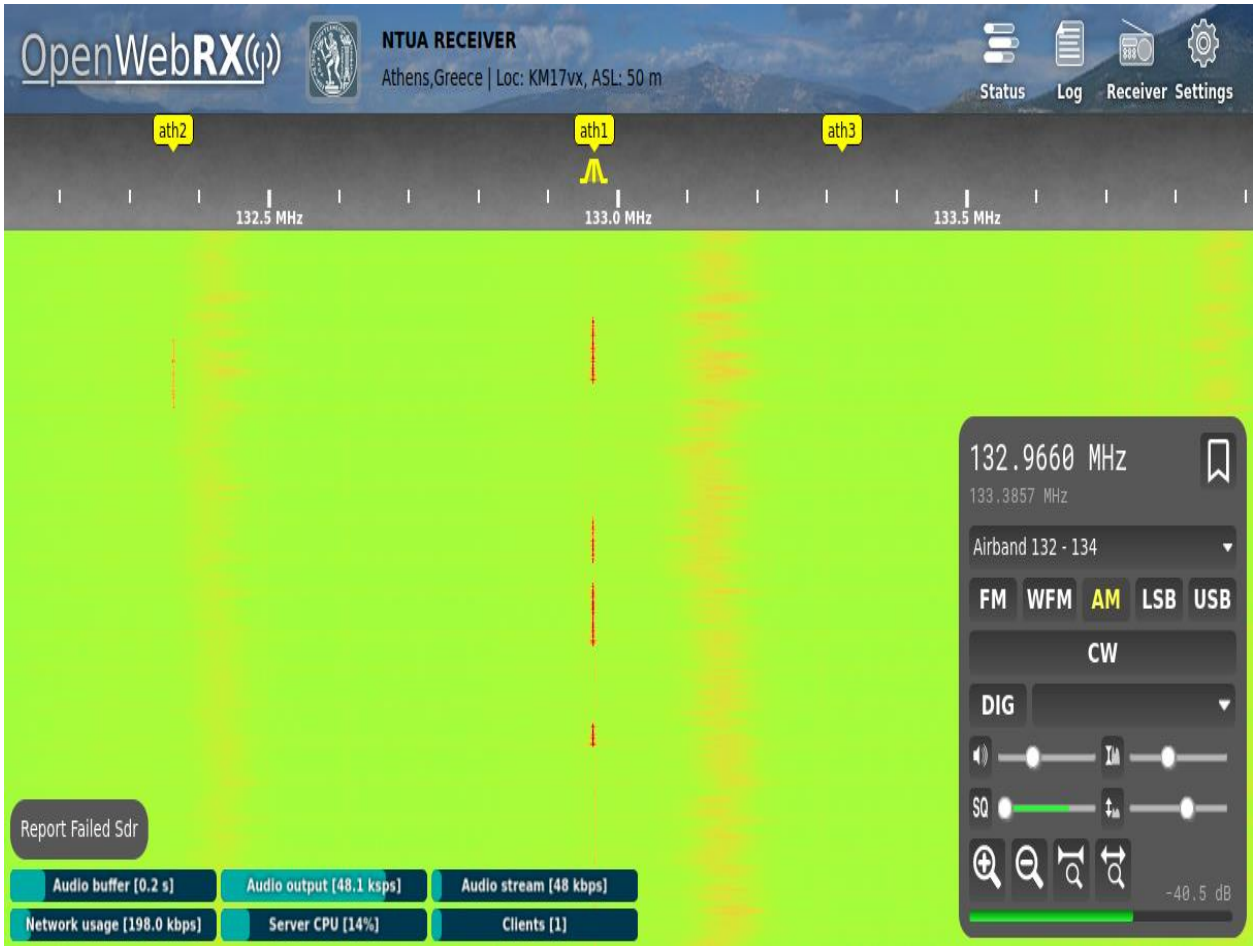
5.8 Στιγμιότυπα οθόνης

Ακολουθούν στιγμιότυπα οθόνης από την εφαρμογή του Load Balancing OpenWebRX. Συγκεκριμένα, συμπεριλαμβάνονται η οθόνη προβολής του καταρράκτη και

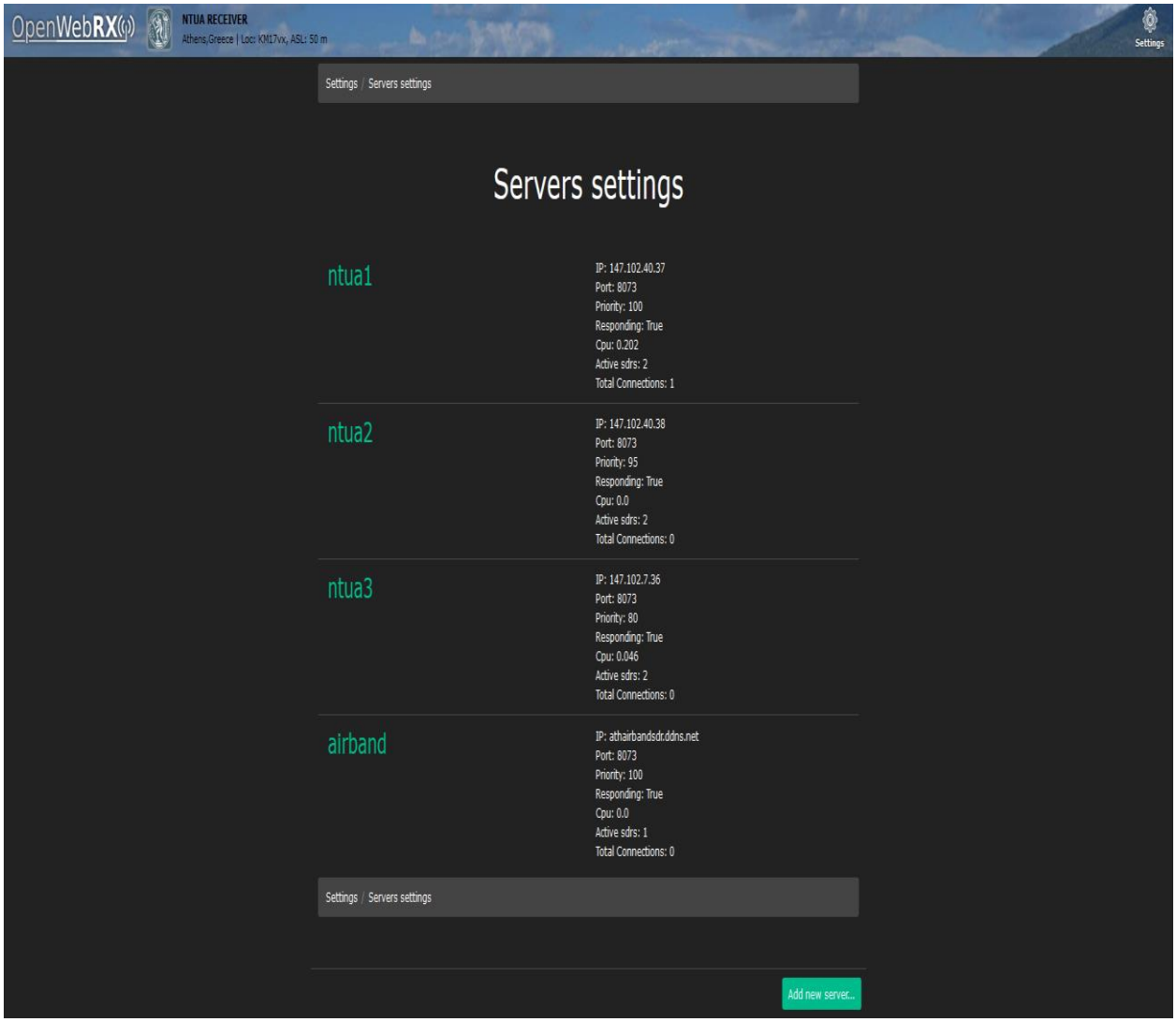
επιλογής σήματος ακρόασης στη ραδιοφωνική ζώνη FM και στην Airband και οι ρυθμίσεις των εξυπηρετητών και των συσκευών SDR.



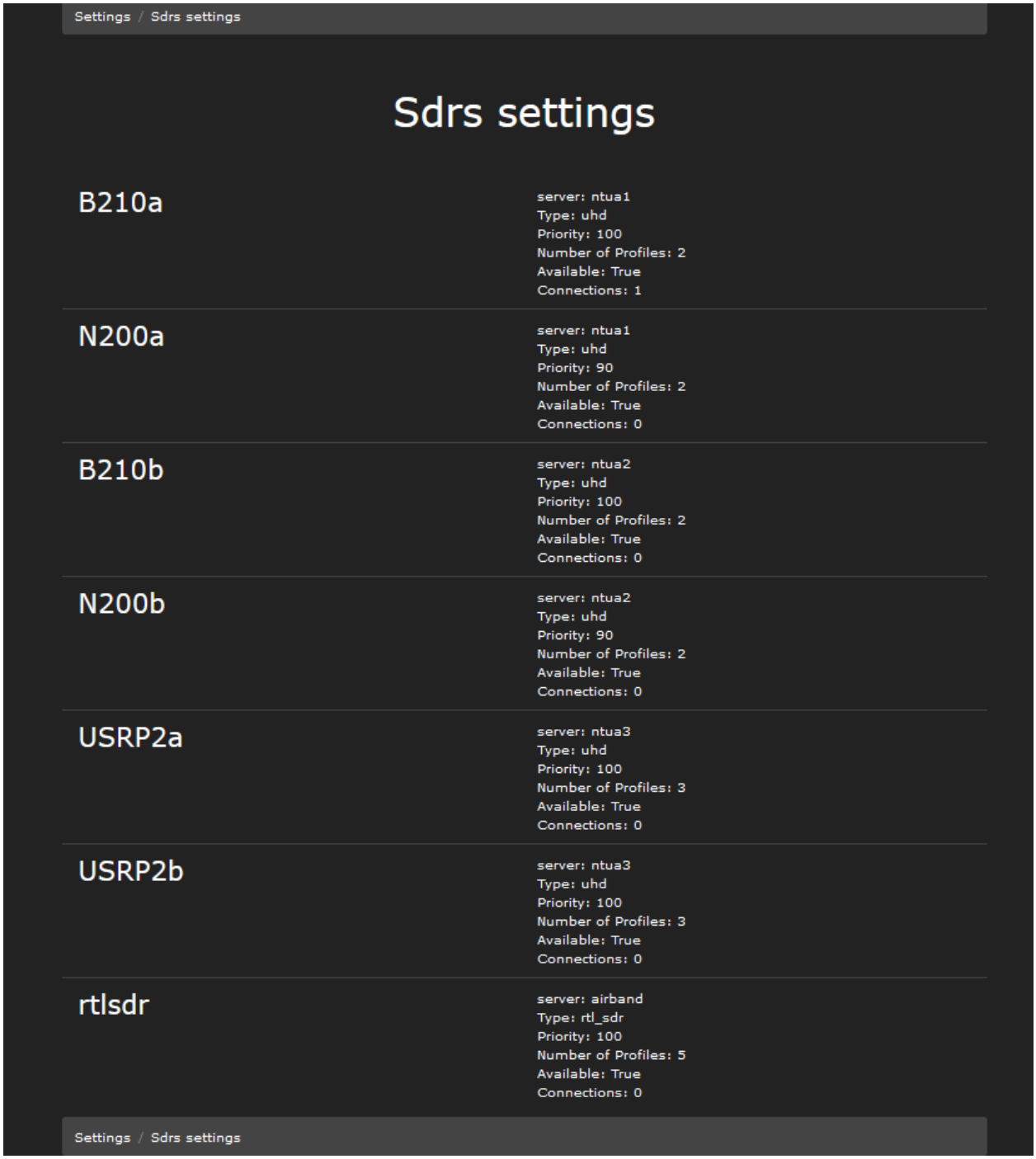
Εικόνα 5.11 *Στιγμιότυπο οθόνης του Load Balancing OpenWebRX (λήψη ραδιοφωνικής εκπομπής FM)*



Εικόνα 5.12 *Στιγμιότυπο οθόνης του Load Balancing OpenWebRX (λήψη Airband)*



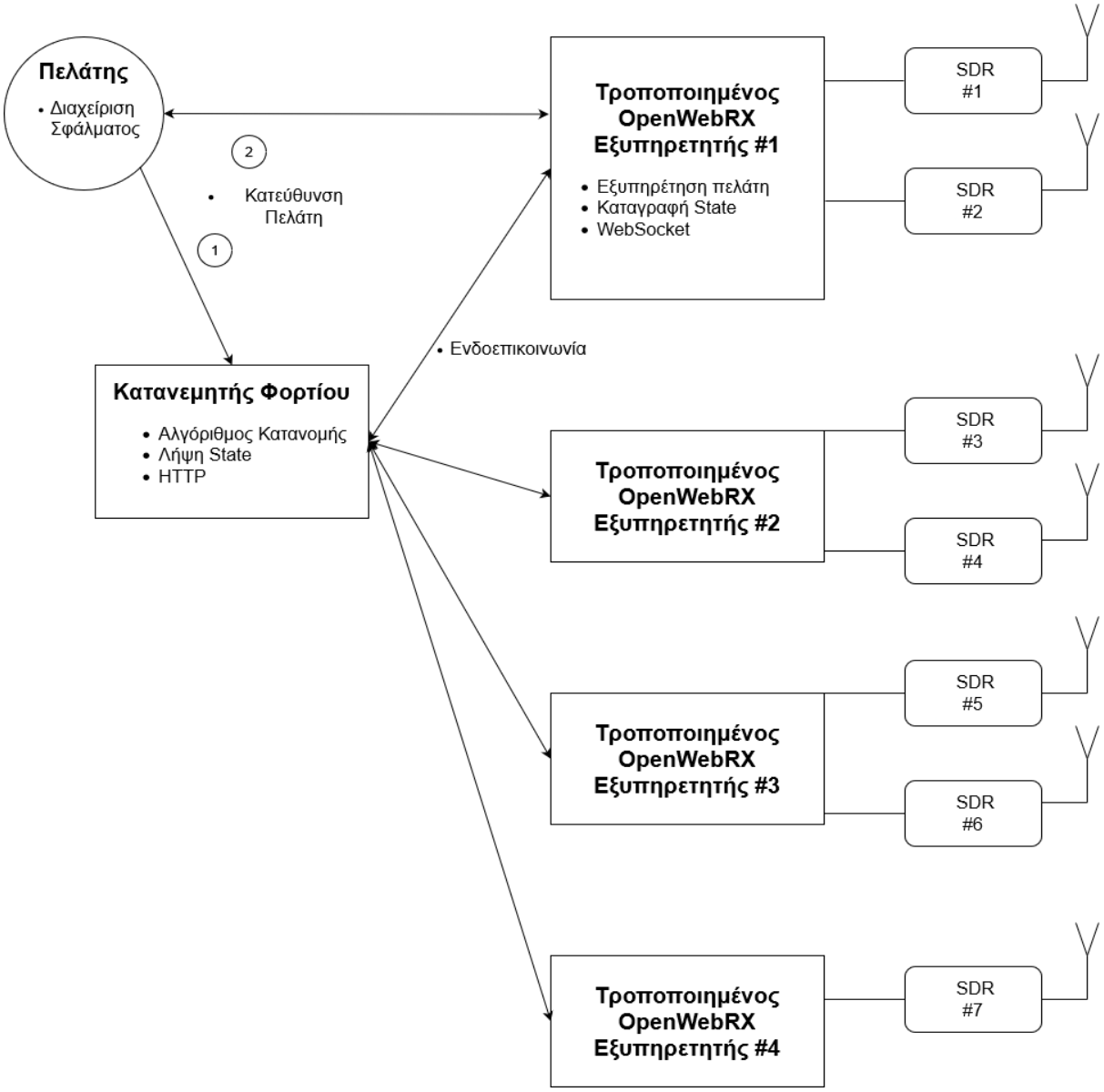
Εικόνα 5.13 *Στιγμιότυπο οθόνης του Load Balancing OpenWebRX (ρυθμίσεις των εξυπηρετητών)*



Εικόνα 5.14 *Στιγμιότυπο οθόνης του Load Balancing OpenWebRX (ρυθμίσεις των SDR συσκευών)*

6. Σύνοψη και μελλοντικές επεκτάσεις

Το Load Balancing OpenWebRX εισάγει έναν κατανεμητή φορτίου μπροστά από ένα σύνολο OpenWebRX εξυπηρετητών. Έτσι επιτρέπει μεταξύ άλλων την αύξηση του αριθμού των ταυτόχρονα υποστηριζόμενων χρηστών. Το σύστημα με το νέο λογισμικό εφαρμόστηκε στο εργαστήριο. Αυτή τη στιγμή παρέχει στους χρήστες πρόσβαση στα σήματα της ραδιοφωνικής εκπομπής FM και στα σήματα της μπάντας των αεροπλάνων (Airband).



Σχήμα 6.1 *Τοπολογία Συστήματος*

Σύστημα

Όπως φαίνεται και στο παραπάνω σχήμα το σύστημα που υλοποιήθηκε αποτελείται από έναν κατανεμητή φορτίου, ο οποίος συνδέεται με πολλούς τροποποιημένους OpenWebRX εξυπηρετητές, πάνω στους οποίους συνδέονται πολλές συσκευές SDR. Όταν έρχεται ένας πελάτης αρχικά απευθύνει ένα HTTP αίτημα στον κατανεμητή φορτίου που προστέθηκε και στη συνέχεια ανάλογα με την απάντηση κατευθύνεται σε ένα τροποποιημένο OpenWebRX εξυπηρετητή.

Εκεί εξυπηρετείται πάνω από το πρωτόκολλο αμφίδρομης επικοινωνίας WebSocket, με τον ίδιο τρόπο που θα εξυπηρετούνταν από το αρχικό OpenWebRX. Συγκεκριμένα, συνδέεται σε μια συσκευή SDR και λαμβάνει τον ήχο ή τα δεδομένα και το γράφημα καταρράκτη του σήματος της, ενώ στέλνει και τις δικές του ρυθμίσεις.

Παράλληλα, στους τροποποιημένους OpenWebRX εξυπηρετητές προστέθηκε καταγραφή της τρέχουσας κατάστασης και των αρχικών ρυθμίσεων τους (state). Αυτή λαμβάνεται μέσω της ενδοεπικοινωνίας, περιοδικά και κατ’ απαίτηση, από τον κατανεμητή φορτίου, προκειμένου να ληφθεί υπ’ όψη από τον αλγόριθμο κατανομής.

Ο αλγόριθμος κατανομής που σχεδιάστηκε, χρησιμοποιεί ένα σύνολο από παραμέτρους σχετικές με τις δυνατότητες και τη διαθεσιμότητα πόρων στους συνδεδεμένους OpenWebRX εξυπηρετητές, για να αποφασίσει που θα σταλθεί ο πελάτης. Έτσι όταν έρθει ένας νέος πελάτης θα κατευθυνθεί αυτόματα σε έναν κατάλληλο εξυπηρετητή και συσκευή SDR.

Τέλος, η πλευρά του πελάτη μπορεί να ενημερωθεί από τους τροποποιημένους OpenWebRX εξυπηρετητές για πιθανό σφάλμα σε συσκευή SDR ή να το αναφέρει ο ίδιος ο χρήστης. Τότε ενεργοποιείται η διαχείριση σφάλματος και ο πελάτης κατευθύνεται σε άλλη συσκευή SDR.

Κώδικας

Ο κώδικας του προγράμματος αποτελείται από 2 μέρη, τον κατανεμητή φορτίου (load balancer) και τον τροποποιημένο OpenWebRX εξυπηρετητή. Όλος ο κώδικας που γράφτηκε, είναι ανοικτό λογισμικό. Έχει δημοσιευθεί στο Github, ο σύνδεσμος βρίσκεται στο https://github.com/iliasz-ECE/lbowrx [102]. Επίσης κάποια ενδεικτικά αρχεία βρίσκονται στο παράρτημα κώδικα που ακολουθεί στο τέλος αυτής της εργασίας.

Μελλοντικές επεκτάσεις

Μελλοντικά θα μπορούσε να διευρυνθεί η λίστα των παρεχόμενων σημάτων, εισάγοντας σήματα και από άλλες ζώνες συχνοτήτων. Αυτό θα μπορούσε να γίνει με μια εγκατάσταση κεραίας. Συγκεκριμένα, για καλύτερη λήψη σημάτων των ζωνών της Airband, των 2m, των 70cm και των 33cm της ISM, καλή επιλογή θα ήταν μια εξωτερικού χώρου Discone κεραία. Αυτό το είδος κεραίας προσφέρει ευρεία λήψη στις μπάντες VHF και UHF. Περισσότερες πληροφορίες στο [103]. Για παράδειγμα, μια κεραία Discone όπως η [104], μπορεί να λειτουργεί από τα 25MHZ μέχρι τα 3GHz. Τέτοιου είδους κεραίες είναι κατάλληλες για συσκευές SDR, λόγω του μεγαλύτερου εύρους συχνοτήτων λειτουργίας τους, σε σχέση με τις απλές κάθετες κεραίες, όπως αναλύεται στο [21]. Για λήψη στην HF και στα MW, θα χρειαζόταν ένα μακρύ σύρμα σε εξωτερικό χώρο, για παράδειγμα πάνω στην ταράτσα του κτηρίου. Όσον αφορά σήματα ανώτερων συχνοτήτων από δορυφόρους, θα μπορούσαν να ληφθούν με μια κατευθυντική παραβολική κεραία στην ταράτσα.

Εικόνα 6.1 *Discone κεραία [104]*

Επίσης, θα μπορούσε να υποστηριχθεί και στο Load Balancing OpenWebRX η λειτουργία χάρτη. Μέχρι στιγμής, αυτή η λειτουργία λόγω έλλειψης κατάλληλης κεραίας και διαθεσιμότητας των σχετικών σημάτων, δεν μπόρεσε να δοκιμαστεί. Με την ύπαρξη κεραίας και τη λήψη των σχετικών σημάτων θα μπορούσε να ενταχθεί και ο χάρτης στο πρόγραμμα. Ακόμα, θα μπορούσαν να προστεθούν εκτός από τους χάρτες της Google και οι ανοικτοί χάρτες OpenStreetMap, για μεγαλύτερη ποικιλία επιλογών.

Τέλος, θα ήταν χρήσιμο να ενταχθούν στο πρόγραμμα και βελτιώσεις που υπάρχουν ήδη σε μια άλλη επέκταση, το OpenWebRX+. Τέτοιες βελτιώσεις είναι περισσότεροι αποκωδικοποιητές και λειτουργία ασφαλούς σύνδεσης HTTPS.

7. Βιβλιογραφία

(Όλες οι πηγές ελέγχθηκαν τον 5/2024.)

[1] "What is Software Defined Radio" https://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf

[2] "Software-defined radio" https://en.wikipedia.org/wiki/Software-defined_radio

[3] "Why SDR (Software Defined Radio)?" https://content.redpitaya.com/blog/why-sdr-software-defined-radio

[4] "SDR Lectures" http://martian.radio.pub.ro/wp-content/uploads/2017/05/Lecture1.pdf

[5] "GNU Radio" https://en.wikipedia.org/wiki/GNU_Radio

[6] "About RTL-SDR" https://www.rtl-sdr.com/about-rtl-sdr/

[7] "NI Ettus USRP X440" https://www.ettus.com/all-products/usrp-x440/

[8] "RTL-SDR Blog V4 Dongle" https://www.rtl-sdr.com/rtl-sdr-blog-v4-dongle-initial-release/

[9] "Software Defined Radio: Past, Present, and Future" https://www.ni.com/en/perspectives/software-defined-radio-past-present-future.html

[10] "Cognitive radio" https://en.wikipedia.org/wiki/Cognitive_radio

[11] "ΣΥΓΧΡΟΝΕΣ ΑΝΑΛΟΓΙΚΕΣ ΚΑΙ ΨΗΦΙΑΚΕΣ ΕΠΙΚΟΙΝΩΝΙΕΣ" Εκδόσεις ΤΖΙΟΛΑ, B.P. Lathi, Zhi Ding

[12] "What is Aliasing and How It Is Reduced" https://electronicspost.com/what-is-aliasing-and-how-it-is-reduced/

[13] "Superheterodyne Receiver" https://www.analog.com/en/resources/glossary/superheterodyne_receiver.html

[14] "Wiki - Superheterodyne receiver" https://en.wikipedia.org/wiki/Superheterodyne_receiver

[15] "SOFTWARE DEFINED RADIO FOR ENGINEERS" Artech House, Travis F. Collins, Robin Getz, Di Pu, Alexander M. Wyglinksi

[16] "Waterfall plot" https://en.wikipedia.org/wiki/Waterfall_plot

[17] "Automatic gain control "https://en.wikipedia.org/wiki/Automatic_gain_control

[18] "Squelch" https://en.wikipedia.org/wiki/Squelch

[19] "SDR Lectures" http://martian.radio.pub.ro/wp-content/uploads/2017/05/Lecture5.pdf

[20] "List of Software Defined Radios" https://en.wikipedia.org/wiki/List_of_software-defined_radios

[21] "SDR antenna suggestions" https://www.sdrplay.com/antennasuggestions/

[22] "Reconfigurable antenna" https://en.wikipedia.org/wiki/Reconfigurable_antenna

[23] "Ettus USRP B210 Datasheet" https://www.ettus.com/wp-content/uploads/2019/01/b200-b210_spec_sheet.pdf

[24] "KiwiSDR" http://kiwisdr.com/

[25] "SDR++ code" *https://github.com/AlexandreRouma/SDRPlusPlus*

[26] "SDR++ manual" https://www.sdrpp.org/manual.pdf

[27] "SDR Radio Console" https://www.sdr-radio.com/console

[28] "GNU Radio" *https://www.gnuradio.org/*

[29] "What is GNU Radio" *https://wiki.gnuradio.org/index.php?title=What_Is_GNU_Radio*

[30] "Gqrx code" https://github.com/gqrx-sdr/gqrx

[31] "Gqrx" https://www.gqrx.dk/

[32] "Gqrx architecture" https://www.gqrx.dk/doc/gqrx-design

[33] "HDSDR" https://www.hdsdr.de/

[34] "SDRangel" *https://www.sdrangel.org/*

[35] *"SDRangel code" https://github.com/f4exb/sdrangel*

[36] "OpenWebRX" https://www.openwebrx.de/

[37] "OpenWebRX code" *https://github.com/jketterl/openwebrx*

[38] "Receiverbook" *https://receiverbook.de*

[39] "OpenWebRX+ code" *https://github.com/luarvique/openwebrx*

[40] "OpenWebRX+ manual" https://fms.komkon.org/OWRX/

[41] "WebSDR" *http://websdr.org/*

[42] "Wide Band WebSDR" *http://websdr.ewi.utwente.nl:8901/*

[43] "ShinySDR code" *https://github.com/kpreid/shinysdr/*

[44] "SoapySDR code" https://github.com/pothosware/SoapySDR/wiki

[45] "Csdr code" https://github.com/jketterl/csdr

[46] "Video: SDRA'23 - Jakob Ketterl, DD5JFK: OpenWebRX: Recent State of Development" https://www.youtube.com/watch?v=cd7PyyxnjYo

[47] “Single Instruction Multiple Data”
https://en.wikipedia.org/wiki/Single_instruction,_multiple_data
[48] “Python C extension module”
https://realpython.com/build-python-c-extension-module
[49] “Fast Fourier Transform”
https://en.wikipedia.org/wiki/Fast_Fourier_transform
[50] “FFTW” https://fftw.org
[51] “PCM” https://testbook.com/physics/pulse-code-modulation
[52] “ADPCM” https://en.wikipedia.org/wiki/Dialogic_ADPCM
[53] “DC blocker” https://www.dsprelated.com/freebooks/filters/DC_Blocker.html
[54] “FM modulation”
https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-demodulation/quadrature-frequency-and-phase-demodulation
[55] “FM broadcasting” https://en.wikipedia.org/wiki/FM_broadcasting
[56] “FM demodulation chain”
https://github.com/jketterl/csdr?tab=readme-ov-file#demodulate-wfm-advanced
[57] “Video: FT8 on 6 metres with OpenwebRX”
https://www.youtube.com/watch?v=UZfqL00kQWQ
[58] “Άρθρο για το FT8”
https://raag.org/wp-content/uploads/2021/01/2018-%CE%96%CE%97%CE%A3%CE%A4%CE%95-%CE%A4%CE%97%CE%9D-%CE%95%CE%A0%CE%91%CE%9D%CE%91%CE%A3%CE%A4%CE%91%CE%A3%CE%97-%CE%A4%CE%9F%CE%A5-FT8-SVNEA-168-MAR-APR-2018.pdf
[59] “HTTP protocol” https://en.wikipedia.org/wiki/HTTP
[60] “Python HTTP server” https://docs.python.org/3/library/http.server.html
[61] “WebSocket protocol” https://en.wikipedia.org/wiki/WebSocket
[62] “Mixin”
https://stackoverflow.com/questions/533631/what-is-a-mixin-and-why-is-it-useful
[63] “APRS” https://aprs.fi
[64] “Canvas API” https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

[65] “ΔΙΚΤΥΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ - 8η Έκδοση” Εκδόσεις Μ. Γκιούρδας, James F. Kurose, Keith W. Ross

[66] "ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ – 6η έκδοση”
Εκδόσεις Κλειδάριθμος, Andrew Tanenbaum, David Wetherall, Nick Feamster

[67] “SSL Load Balancer”
https://www.nginx.com/resources/glossary/ssl-load-balancer

[68] “AWS - What is Load Balancing?”
https://aws.amazon.com/what-is/load-balancing

[69] “NGINX - Load Balancing”
https://www.nginx.com/resources/glossary/load-balancing

[70] “Baeldung - Load Balancer” https://www.baeldung.com/cs/load-balancer

[71] “Scalelite” https://github.com/blindsidenetworks/scalelite

[72] “BigBlueButton” https://bigbluebutton.org/

[73] “Scalelite Deployment”
https://jffederico.medium.com/scalelite-lazy-deployment-745a7be849f6

[74] “Video: Hiding Multiple BigBlueButton Servers Behind a Single Hostname”
https://www.youtube.com/watch?v=X2X9hOMaCBM

[75] “Cluster Proxy Scalelite”
https://docs.bigbluebutton.org/administration/cluster-proxy

[76] “OpenStreetMap” https://www.openstreetmap.org

[77] “OpenWebRX+ Map” https://fms.komkon.org/OWRX/#HOW-CONFIG-MAP

[78] “Ettus USRP B210” https://www.ettus.com/all-products/ub210-kit/

[79] “Ettus USRP N200” https://www.ettus.com/all-products/un200-kit/

[80] “USRP Info” https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates

[81] “Ettus USRP N200 Datasheet”
https://www.ettus.com/wp-content/uploads/2019/01/07495_Ettus_N200-210_DS_Flyer_HR.pdf

[82] “Ettus USRP 2” https://bmisurplus.com/product/ettus-research-usrp2

[83] “RTL SDR photo” https://www.passion-radio.com/668-large_default/rtl-sdr-r820t2.jpg

[84] “Intel Nuc” https://rutronik-tec.com/intel-nuc-13-pro-built-for-business/

[85] “NUC” https://www.dialogic.com/glossary/next-unit-of-computing-nuc-

[86] “Wiki - NUC” https://en.wikipedia.org/wiki/Next_Unit_of_Computing

[87] “RPi photo” https://projects-static.raspberrypi.org/projects/raspberry-pi-getting-started/c66d8a79f813edd87ee9626e86f5cfbffafbf3e6/en/images/raspberry-pi.png

[88] “RPi” https://en.wikipedia.org/wiki/Raspberry_Pi

[89] “RPi 5” https://www.raspberrypi.com/products/raspberry-pi-5/

[90] “VMware logo” https://upload.wikimedia.org/wikipedia/commons/thumb/1/11/VMware_logo.svg/2560px-VMware_logo.svg.png

[91] “Virtual Machine” https://www.vmware.com/topics/glossary/content/virtual-machine.html

[92] “Wiki- Virtual Machine” https://en.wikipedia.org/wiki/Virtual_machine

[93] “Ubuntu Jammy” https://releases.ubuntu.com/jammy/

[94] “OpenWebRX Installation Manual” https://github.com/jketterl/openwebrx/wiki/Manual-Package-installation-(including-digital-voice)

[95] “Ettus UHD” https://github.com/EttusResearch/uhd

[96] “Systemd” https://en.wikipedia.org/wiki/Systemd

[97] “Systemd Tutorial” https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units

[98] “NO-IP” https://www.noip.com/

[99] “Το Ραδιόφωνο στην Ελλάδα” https://dspace.lib.uom.gr/bitstream/2159/26777/4/PitsoudiParaskeviMsc2022.pdf

[100] “Airband” https://en.wikipedia.org/wiki/Airband

[101] “FlightRadar24” https://www.flightradar24.com

[102] “Load Balancing OpenWebRX” https://github.com/iliasz-ECE/lbowrx

[103] “Discone antenna” https://en.wikipedia.org/wiki/Discone_antenna

[104] “Diamond Discone antenna” https://www.diamondantenna.net/d3000n.html

Παράρτημα Κώδικα

Παρουσιάζονται ενδεικτικά αρχεία κώδικα από την εργασία.
Ολόκληρος ο κώδικας βρίσκεται στο:
https://github.com/iliasz-ECE/lbowrx .

lbowrx-lb/lbowrx_lb/distribution.py

```
1  import threading
2  import logging
3
4  logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(name)s -
%(levelname)s - %(message)s")
5  logger = logging.getLogger(__name__)
6
7  from lbowrx_lb.config.dynamic import DynamicConfig
8
9  class ServerInfo():
10     def __init__(self, server_id):
11         server_info =
DynamicConfig.get_servers_object().get_server_info(server_id)
12         self.name = server_info.name
13         self.ip = server_info.ip
14         self.port = server_info.port
15         self.priority = server_info.priority
16         self.initial_response = False
17         self.init_state_tag = None
18         self.responding = False
19         self.cpu = None
20         self.sdrs = {}
21
22     def update_cur(self,server_dict):
23         self.responding = server_dict["responding"]
24         if self.responding:
25             self.cpu = server_dict["latest_avg_cpu_usage"]
26             return self.init_state_tag != server_dict["init_state_tag"]
27         else:
28             return False
29
30 class SdrInfo():
31     def __init__(self):
32         self.name = None
33         self.type = None
34         self.priority = None
35         self.usage = None
36         self.failed = None
```

```
37          self.disabled = None
38          self.has_users = None
39          self.cur_profile_id = None
40          self.num_of_connections = None
41          self.profiles = {}
42
43      def update_init(self,sdr_dict):
44          self.name = sdr_dict["name"]
45          self.type = sdr_dict["type"]
46
47          if "enabled" in sdr_dict:
48              self.disabled = not sdr_dict["enabled"]
49          else:
50              self.disabled = False
51          if "priority" in sdr_dict:
52              self.priority = sdr_dict["priority"]
53          else:
54              self.priority = 100
55          if "usage" in sdr_dict:
56              self.usage = sdr_dict["usage"]
57          else:
58              self.usage = "exclusive"
59
60      def update_cur(self,sdr_dict):
61          self.failed = sdr_dict["failed"]
62          self.has_users = sdr_dict["has_users"]
63          self.num_of_connections = sdr_dict["number_of_connections"]
64          self.cur_profile_id = sdr_dict["current_profile"]["profile_id"]
65
66  class ProfileInfo():
67      def __init__(self, center_freq, sample_rate):
68          self.center_freq = center_freq
69          self.sample_rate = sample_rate
70
71  class Distribution():
72      servers_state_db = {}
73      state_lock = threading.Lock()
74
75
76      @classmethod
77      def create_init_state(cls, loglevel=None):
78
79          if loglevel:
80              logger.setLevel(loglevel)
81
82          cls.state_lock.acquire()
83          cls.servers_state_db = {}
84
85          for server_id in DynamicConfig.get_servers().keys():
86              server_info = ServerInfo(server_id)
87              cls.servers_state_db.update({server_id: server_info})
88
89          cls.state_lock.release()
90          logger.debug("Created initial state")
91
92      @classmethod
93      def add_init_state(cls, init_dict):
94          cls.state_lock.acquire()
```

```
95
96          for server_id, server_dict in init_dict.items():
97              server_info = ServerInfo(server_id)
98              server_info.initial_response =
server_dict["initial_response"]
99
100             if server_dict["initial_response"]:
101                 del server_dict["initial_response"]
102
103                 server_info.init_state_tag =
server_dict["init_state_tag"]
104                 del  server_dict["init_state_tag"]
105
106                 for sdr_id, sdr_dict in server_dict.items():
107                     sdr_info = SdrInfo()
108                     sdr_info.update_init(sdr_dict)
109                     server_info.sdrs.update({sdr_id: sdr_info})
110
111                     for profile_id, profile_dict in
sdr_dict["profiles"].items():
112                         profile_info =
ProfileInfo(profile_dict["center_freq"], profile_dict["samp_rate"])
113                         sdr_info.profiles.update({profile_id:
profile_info})
114
115             cls.servers_state_db.update({server_id: server_info})
116
117         cls.state_lock.release()
118         logger.debug("Added initial state")
119
120     @classmethod
121     def update_cur_state(cls, cur_dict):
122         cls.state_lock.acquire()
123
124         ret = []
125         for server_id, server_dict in cur_dict.items():
126             if cls.servers_state_db[server_id].update_cur(server_dict):
127                 logger.debug("change in server: %s init state",
cls.servers_state_db[server_id].name)
128                 ret.append(server_id)
129                 continue
130
131             if server_dict["responding"]:
132                 del server_dict["latest_avg_cpu_usage"]
133                 del server_dict["responding"]
134                 del server_dict["init_state_tag"]
135
136                 for sdr_id, sdr_dict in server_dict.items():
137
cls.servers_state_db[server_id].sdrs[sdr_id].update_cur(sdr_dict)
138
139         cls.state_lock.release()
140         if not ret:
141             logger.debug("Updated current state")
142         # ret is a list of server id when init state is needed
143         return ret
144
145
```

```
146     @classmethod
147     def get_servers(cls):
148         all_servers_dict = {}
149         for server_id, server in cls.servers_state_db.items():
150             server_dict = {}
151             server_dict.update({"name": server.name})
152             server_dict.update({"ip": server.ip})
153             server_dict.update({"port": server.port})
154             server_dict.update({"priority": server.priority})
155             server_dict.update({"responding": server.responding})
156             if server.responding:
157                 server_dict.update({"cpu": server.cpu})
158                 num_of_active_sdrs = 0
159                 total_connections = 0
160 
161                 for sdr in server.sdrs.values():
162                     if (not sdr.disabled) and (not sdr.failed):
163                         num_of_active_sdrs += 1
164                         total_connections += sdr.num_of_connections
165 
166                 server_dict.update({"active_sdrs": num_of_active_sdrs})
167                 server_dict.update({"connections": total_connections})
168 
169             all_servers_dict.update({server_id: server_dict})
170 
171         return all_servers_dict
172 
173     @classmethod
174     def add_server(cls, server_id):
175         server_info = ServerInfo(server_id)
176         cls.servers_state_db.update({server_id: server_info})
177 
178     @classmethod
179     def delete_server(cls, server_id):
180         del cls.servers_state_db[server_id]
181 
182     @classmethod
183     def modify_server(cls, server_id):
184         del cls.servers_state_db[server_id]
185         server_info = ServerInfo(server_id)
186         cls.servers_state_db.update({server_id: server_info})
187 
188 
189     @classmethod
190     def get_sdrs(cls):
191         all_sdrs_dict = {}
192         for server in cls.servers_state_db.values():
193             if server.responding:
194                 for sdr_id, sdr in server.sdrs.items():
195                     sdr_dict = {}
196                     sdr_dict.update({"name": sdr.name})
197                     sdr_dict.update({"server": server.name})
198                     sdr_dict.update({"type":sdr.type})
199                     sdr_dict.update({"priority": sdr.priority})
200                     sdr_dict.update({"profiles_num": len(sdr.profiles)})
201                     sdr_dict.update({"available": (not sdr.failed) and
(not sdr.disabled)})
```

```
202                     sdr_dict.update({"connections":
sdr.num_of_connections})
203                     all_sdrs_dict.update({sdr_id: sdr_dict})
204
205         return all_sdrs_dict
206
207     @classmethod
208     def get_server_info(cls,server_id):
209         return cls.servers_state_db[server_id]
210
211     @classmethod
212     def check_profiles_match(cls,checked_profile,selected_profile):
213         start_freq = checked_profile.center_freq -
checked_profile.sample_rate/2
214         end_freq = checked_profile.center_freq +
checked_profile.sample_rate/2
215
216         return start_freq <= selected_profile.start_freq and end_freq >=
selected_profile.end_freq
217
218     @staticmethod
219     def score_sort(candidate_tuple):
220         # negation for reversing sort order.
221         return (candidate_tuple[3], -candidate_tuple[4])
222
223     @classmethod
224     def print_match_list(cls,match_list):
225         servers = cls.servers_state_db
226         for x in match_list:
227             server_id = x[0]
228             server_name = servers[server_id].name
229             sdr_id = x[1]
230             sdr_name = servers[server_id].sdrs[sdr_id].name
231             profile_id = x[2]
232             profile_freq =
servers[server_id].sdrs[sdr_id].profiles[profile_id].center_freq
233             logger.debug((server_name,sdr_name,profile_freq,x[3],x[4]))
234
235     @classmethod
236     def decision_algorithm(cls, selected_profile_id, failed_sdrs_list):
237         cls.state_lock.acquire()
238
239         selected_profile_info =
DynamicConfig.get_profiles_object().get_profile_info(selected_profile_id)
240
241         match_list = []
242         score = 0
243         for server_id, server_info in cls.servers_state_db.items():
244             server_available = server_info.responding
245
246             if server_info.priority > 0:
247                 server_score = server_info.priority
248             else:
249                 server_available = False
250
251             if server_available:
252                 if server_info.cpu > 0.5 and server_info.cpu <= 0.8:
253                     server_score -= 20
```

```
254                 if server_info.cpu > 0.8:
255                     server_available = False
256 
257             if server_available:
258                 for sdr_id, sdr_info in server_info.sdrs.items():
259                     sdr_available = (not sdr_info.failed) and (not
sdr_info.disabled) and (not sdr_id in failed_sdrs_list)
260                     sdr_score = sdr_info.priority
261 
262                     if sdr_available:
263                         if sdr_info.has_users:
264                             if
cls.check_profiles_match(sdr_info.profiles[sdr_info.cur_profile_id],selecte
d_profile_info):
265                                 sdr_score += 10
266                                 profile_id = sdr_info.cur_profile_id
267                                 score = server_score + sdr_score
268                                 sample_rate =
sdr_info.profiles[sdr_info.cur_profile_id].sample_rate
269 
match_list.append((server_id,sdr_id,profile_id,score,sample_rate))
270                             else:
271                                 sdr_available = False
272                                 continue
273 
274                         else:
275                             for profile_id, profile_info in
sdr_info.profiles.items():
276                                 if
cls.check_profiles_match(profile_info, selected_profile_info):
277                                     score = server_score + sdr_score
278                                     sample_rate =
profile_info.sample_rate
279 
match_list.append((server_id,sdr_id,profile_id,score,sample_rate))
280         cls.state_lock.release()
281 
282         match_list.sort(reverse = True, key=cls.score_sort)
283         cls.print_match_list(match_list)
284 
285         best_match = max(match_list, key=cls.score_sort, default=None)
286         if best_match:
287             return (best_match[0],best_match[1],best_match[2])
288         else:
289             logger.debug("No Decision - No Sdr match")
290             return None
291 
292 
```

lbowrx-lb/lbowrx_lb/receivestate.py

```
1  import threading
2  import requests
3  import time
4  import logging
5
6  logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(name)s -
%(levelname)s - %(message)s")
7  logger = logging.getLogger(__name__)
8
9  from lbowrx_lb.config.dynamic import DynamicConfig
10 from lbowrx_lb.distribution import Distribution
11
12 class ReceiveServersStateThread(threading.Thread):
13     shared_instance = None
14
15     def __init__(self, loglevel=None):
16         self.loglevel = loglevel
17         if loglevel:
18             logger.setLevel(loglevel)
19
20         self.doRun = True
21         self.wakeEvent = threading.Event()
22         self.wakeEvent.clear()
23         self.workedEvent = threading.Event()
24         self.workedEvent.clear()
25
26         super().__init__()
27         ReceiveServersStateThread.shared_instance = self
28
29     @classmethod
30     def wake_and_wait_until_thread_works(cls):
31         if cls.shared_instance is None:
32             cls.shared_instance = cls()
33
34         cls.shared_instance.wakeEvent.set()
35         cls.shared_instance.workedEvent.wait()
36         cls.shared_instance.workedEvent.clear()
37
38     def run(self):
39         logger.info("Receive State Thread is running")
40
41         Distribution.create_init_state(self.loglevel)
42         self.receive_cur_servers_state()
43
44         while self.doRun:
45             self.wakeEvent.wait(30)
46
47             if self.doRun == False:
48                 break
49
```

```
50              self.receive_cur_servers_state()
51              if self.wakeEvent.is_set():
52                  self.wakeEvent.clear()
53                  self.workedEvent.set()
54
55
56      def shutdown(self):
57          self.doRun = False
58          self.wakeEvent.set()
59
60      def receive_init_servers_state(self, servers_target_list):
61          target_servers_dict = {}
62
63          servers = DynamicConfig.get_servers()
64
65          for server_id in servers_target_list:
66              server_info = servers[server_id]
67
68              ip = server_info.ip
69              port = server_info.port
70
71              server_dict = {}
72
73              url = "http://" + ip + ':' + str(port) + "/initstate.json"
74
75              try:
76                  r = requests.get(url, timeout=(2,2))
77                  server_dict = r.json()
78                  server_dict.update({"initial_response": True})
79              except:
80                  server_dict.update({"initial_response": False})
81
82              target_servers_dict.update({server_id: server_dict})
83
84          Distribution.add_init_state(target_servers_dict)
85
86      def receive_cur_servers_state(self):
87          all_servers_dict = {}
88          servers = DynamicConfig.get_servers()
89          for server_id, server_info in servers.items():
90              ip = server_info.ip
91              port = server_info.port
92
93              server_dict = {}
94
95              url = "http://" + ip + ':' + str(port) + "/curstate.json"
96
97              try:
98                  r = requests.get(url, timeout=(1,1))
99                  server_dict = r.json()
100                 server_dict.update({"responding": True})
101             except:
102                 server_dict.update({"responding": False})
103                 logger.debug(server_info.ip + " is not responding")
104
105             all_servers_dict.update({server_id: server_dict})
106
107         ret = Distribution.update_cur_state(all_servers_dict)
```

```
108
109         # If there is no initial state receive it.
110         if ret:
111             self.receive_init_servers_state(ret)
112             self.receive_cur_servers_state()
113
```

lbowrx-lb/lbowrx_lb/controllers/assets.py

```
1  from datetime import datetime, timezone
2  import mimetypes
3  import os
4  import pkg_resources
5  from abc import ABCMeta, abstractmethod
6  import gzip
7  from string import Template
8  import logging
9  logger = logging.getLogger(__name__)
10
11 from lbowrx_lb.controllers import Controller
12 from lbowrx_lb.config.core import CoreConfig
13 from lbowrx_lb.config.dynamic import DynamicConfig
14
15 from lbowrx_lb.distribution import Distribution
16 from lbowrx_lb.receivestate import ReceiveServersStateThread
17
18
19 class GzipMixin(object):
20     def send_response(self, content, code=200, headers=None,
content_type="text/html", *args, **kwargs):
21         if self.zipable(content_type) and "accept-encoding" in
self.request.headers:
22             accepted = [s.strip().lower() for s in
self.request.headers["accept-encoding"].split(",")]
23             if "gzip" in accepted:
24                 if type(content) == str:
25                     content = content.encode()
26                 content = self.gzip(content)
27                 if headers is None:
28                     headers = {}
29                 headers["Content-Encoding"] = "gzip"
30         super().send_response(content, code, headers=headers,
content_type=content_type, *args, **kwargs)
31
32     def zipable(self, content_type):
33         types = ["text/javascript", "application/javascript",
"text/css", "text/html", "image/svg+xml"]
34         return content_type in types
35
36     def gzip(self, content):
37         return gzip.compress(content)
38
39
40 class ModificationAwareController(Controller, metaclass=ABCMeta):
41     @abstractmethod
42     def getModified(self, file):
43         pass
44
45     def wasModified(self, file):
46         try:
```

```
47              modified = self.getModified(file).replace(microsecond=0)
48
49              if modified is not None and "If-Modified-Since" in
self.handler.headers:
50                  client_modified = datetime.strptime(
51                      self.handler.headers["If-Modified-Since"], "%a, %d
%b %Y %H:%M:%S %Z"
52                  ).replace(tzinfo=timezone.utc)
53                  if modified <= client_modified:
54                      return False
55          except FileNotFoundError:
56              pass
57
58          return True
59
60
61  class AssetsController(GzipMixin, ModificationAwareController,
metaclass=ABCMeta):
62      def getModified(self, file):
63          return
datetime.fromtimestamp(os.path.getmtime(self.getFilePath(file)),
timezone.utc)
64
65      def openFile(self, file):
66          return open(self.getFilePath(file), "rb")
67
68      @abstractmethod
69      def getFilePath(self, file):
70          pass
71
72      def serve_file(self, file, content_type=None):
73          try:
74              modified = self.getModified(file)
75
76              if not self.wasModified(file):
77                  self.send_response("", code=304)
78                  return
79
80              f = self.openFile(file)
81              data = f.read()
82              f.close()
83
84              if content_type is None:
85                  (content_type, encoding) =
mimetypes.guess_type(self.getFilePath(file))
86              self.send_response(data, content_type=content_type,
last_modified=modified, max_age=3600)
87          except FileNotFoundError:
88              self.send_response("file not found", code=404)
89
90      def indexAction(self):
91          filename = self.request.matches.group(1)
92          self.serve_file(filename)
93
94
95  class OwrxAssetsController(AssetsController):
96      def getFilePath(self, file):
97          mappedFiles = {
```

```
98              "gfx/openwebrx-avatar.png": "receiver_avatar",
99              "gfx/openwebrx-top-photo.jpg": "receiver_top_photo",
100         }
101         if file in mappedFiles and ("mapped" not in self.request.query
or self.request.query["mapped"][0] != "false"):
102             config = CoreConfig.get_shared_config()
103             for ext in ["png", "jpg", "webp"]:
104                 user_file =
"{}/{}.{}".format(config.get_data_directory(), mappedFiles[file], ext)
105                 if os.path.exists(user_file) and
os.path.isfile(user_file):
106                     return user_file
107         return pkg_resources.resource_filename("htdocs", file)
108
109
110class AprsSymbolsController(AssetsController):
111     def __init__(self, handler, request, options):
112         path = CoreConfig.get_shared_config().get_aprs_symbols_path()
113         if not path.endswith("/"):
114             path += "/"
115         self.path = path
116         super().__init__(handler, request, options)
117
118     def getFilePath(self, file):
119         return self.path + file
120
121class CompiledAssetsControllerLibs(GzipMixin,
ModificationAwareController):
122     profiles = {
123         "receiver.js": [
124             "lib/chroma.min.js",
125             "openwebrx.js",
126             "lib/jquery-3.2.1.min.js",
127             "lib/jquery.nanoscroller.min.js",
128             "lib/Header.js",
129             "lib/Demodulator.js",
130             "lib/DemodulatorPanel.js",
131             "lib/BookmarkLocalStorage.js",
132             "lib/BookmarkBar.js",
133             "lib/BookmarkDialog.js",
134             "lib/AudioEngine.js",
135             "lib/ProgressBar.js",
136             "lib/Measurement.js",
137             "lib/FrequencyDisplay.js",
138             "lib/MessagePanel.js",
139             "lib/Js8Threads.js",
140             "lib/Modes.js",
141             "lib/MetaPanel.js",
142         ],
143         "settings.js": [
144             "lib/jquery-3.2.1.min.js",
145             "lib/bootstrap.bundle.min.js",
146             "lib/location-picker.min.js",
147             "lib/Header.js",
148             "lib/settings/MapInput.js",
149             "lib/settings/ImageUpload.js",
150             "lib/BookmarkLocalStorage.js",
151             "lib/settings/BookmarkTable.js",
```

```
152             "lib/settings/WsjtDecodingDepthsInput.js",
153             "lib/settings/WaterfallDropdown.js",
154             "lib/settings/GainInput.js",
155             "lib/settings/OptionalSection.js",
156             "lib/settings/SchedulerInput.js",
157             "lib/settings/ExponentialInput.js",
158             "settings.js",
159         ],
160     }
161 
162     def indexAction(self):
163         profileName = self.request.matches.group(1)
164         if profileName not in self.profiles:
165             self.send_response("profile not found", code=404)
166             return
167 
168         files = self.profiles[profileName]
169         files = [pkg_resources.resource_filename("htdocs", f) for f in
files]
170 
171         modified = self.getModified(files)
172 
173         if not self.wasModified(files):
174             self.send_response("", code=304)
175             return
176 
177         contents = [self.getContents(f) for f in files]
178 
179         (content_type, encoding) = mimetypes.guess_type(profileName)
180         self.send_response("\n".join(contents),
content_type=content_type, last_modified=modified, max_age=3600)
181 
182     def getContents(self, file):
183         with open(file) as f:
184             return f.read()
185 
186     def getModified(self, files):
187         modified = [os.path.getmtime(f) for f in files]
188         return datetime.fromtimestamp(max(*modified), timezone.utc)
189 
190class CompiledAssetsControllerVars(GzipMixin, Controller):
191     def indexAction(self):
192 
193         # Always update current state before deciding, so that newest
data are available, especially at profile change.
194         ReceiveServersStateThread.wake_and_wait_until_thread_works()
195 
196         profiles_id_list = DynamicConfig.get_profiles_id_list()
197 
198         selected_profile = None
199         decision = None
200 
201         if self.request.query:
202             if "profileid" in self.request.query:
203                 selected_profile = self.request.query["profileid"][0]
204 
205                 if selected_profile not in profiles_id_list:
206                     # default profile
```

```
207                     selected_profile =
DynamicConfig.get_profiles_id_list()[0]
208         else:
209             # default profile
210             selected_profile = DynamicConfig.get_profiles_id_list()[0]
211 
212 
213         all_sdrs = Distribution.get_sdrs()
214         failed_sdrs_list = []
215 
216         if "failed_sdrs" in self.request.cookies:
217             failed_sdrs_cookie =
self.request.cookies["failed_sdrs"].value
218             for sdr_id in failed_sdrs_cookie.split(','):
219                 if sdr_id in all_sdrs:
220                     failed_sdrs_list.append(sdr_id)
221         else:
222             failed_sdrs_list = []
223 
224         decision = Distribution.decision_algorithm(selected_profile,
failed_sdrs_list)
225 
226         if not decision:
227             decision_vars = {"decision": "false","server": "null",
"sdrid": "null", "profileid": "null",
228                              "current_profile": "'"+selected_profile+"'",
"profiles": DynamicConfig.get_profiles_id_name_list()}
229         else:
230             servers = DynamicConfig.get_servers()
231 
232             decided_server = decision[0]
233             decided_ip = servers[decided_server].ip
234             decided_port = servers[decided_server].port
235 
236             decided_sdr = decision[1]
237 
238             decided_profile = decision[2]
239 
240             decision_vars = {"decision": "true", "server":
"'"+decided_ip+":"+str(decided_port)+"'",
241             "sdrid": "'"+decided_sdr+"'", "profileid":
"'"+decided_profile+"'", "current_profile":
242             "'"+selected_profile+"'", "profiles":
DynamicConfig.get_profiles_id_name_list()}
243 
244 
245         file_content = pkg_resources.resource_string("htdocs",
"websocket-vars.js").decode("utf-8")
246         template = Template(file_content)
247         content = template.safe_substitute(**decision_vars)
248 
249         self.send_response(content, content_type="text/javascript",
max_age=0)
250 
```

lbowrx-lb/lbowrx_lb/__main__.py

```
1  import logging
2
3  logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(name)s -
%(levelname)s - %(message)s")
4  logger = logging.getLogger(__name__)
5
6  from http.server import HTTPServer
7  from socketserver import ThreadingMixIn
8  import signal
9  import argparse
10 import socket
11
12 from lbowrx_lb.httproutes import RequestHandler
13
14 from lbowrx_lb.config.core import CoreConfig
15 from lbowrx_lb.config.dynamic import DynamicConfig
16
17 from lbowrx_lb.receivestate import ReceiveServersStateThread
18 from lbowrx_lb.distribution import Distribution
19
20 from lbowrx_lb.admin import add_admin_parser, run_admin_action
21
22 class ThreadedHttpServer(ThreadingMixIn, HTTPServer):
23     def __init__(self, web_port, RequestHandlerClass, use_ipv6):
24         bind_address = "0.0.0.0"
25         if use_ipv6:
26             self.address_family = socket.AF_INET6
27             bind_address = "::"
28         super().__init__((bind_address, web_port), RequestHandlerClass)
29
30
31 class SignalException(Exception):
32     pass
33
34
35 def handleSignal(sig, frame):
36     raise SignalException("Received Signal {sig}".format(sig=sig))
37
38
39 def main():
40
41     parser = argparse.ArgumentParser(description="OpenWebRX - Open
Source SDR Web App for Everyone!")
42     parser.add_argument("--debug", action="store_true", help="Set
loglevel to DEBUG")
43
44     moduleparser = parser.add_subparsers(title="Modules", dest="module")
45     adminparser = moduleparser.add_parser("admin", help="Administration
actions")
46     add_admin_parser(adminparser)
47
48     args = parser.parse_args()
49
```

```
50      if args.debug:
51         logger.setLevel(logging.DEBUG)
52
53      if args.module == "admin":
54          return run_admin_action(adminparser, args)
55
56      CoreConfig()
57
58      return start_receiver(loglevel=logging.DEBUG if args.debug else
None)
59
60  def start_receiver(loglevel=None):
61      print(
62          """
63
64  Load Balancing OpenWebRX - Open Source SDR Web App for Everyone!
65  for license see LICENSE file in the package
66
```

```
67
68  Load Balancing OpenWebRX Author:    Ilias Zervas
69  OpenWebRX Authors: Jakob Ketterl, András Retzler
70  Documentation:          https://github.com/jketterl/openwebrx/wiki
71  Support and info:       https://groups.io/g/openwebrx
72
73          """,
74          flush=True
75      )
76
77
78      for sig in [signal.SIGINT, signal.SIGTERM]:
79          signal.signal(sig, handleSignal)
80
81      coreConfig = CoreConfig.get_shared_config()
82      logger.debug("Port: " + str(coreConfig.get_web_port()))
83      logger.debug(coreConfig.get_data_directory())
84
85      dynamicConfig = DynamicConfig()
86
87      receive_servers_state_thread = ReceiveServersStateThread(loglevel)
88      receive_servers_state_thread.start()
89
90      try:
91          server = ThreadedHttpServer(coreConfig.get_web_port(),
RequestHandler, coreConfig.get_web_ipv6())
92          logger.info("Ready to serve requests.")
93          server.serve_forever()
94      except SignalException:
95          pass
96
97
98      receive_servers_state_thread.shutdown()
99
100     return 0
101
102
103 import sys
```

```
104if __name__ == "__main__":
105    sys.exit(main())
106
```

lbowrx-lb/lbowrx_lb/httproutes.py

```
1  from http.server import BaseHTTPRequestHandler
2  from urllib.parse import urlparse, parse_qs
3  import re
4  from abc import ABC, abstractmethod
5  import logging
6
7  from http.cookies import SimpleCookie
8
9  from lbowrx_lb.controllers.template import IndexController
10 from lbowrx_lb.controllers.assets import OwrxAssetsController,
AprsSymbolsController, CompiledAssetsControllerLibs,
CompiledAssetsControllerVars
11
12 from lbowrx_lb.controllers.settings import SettingsController
13 from lbowrx_lb.controllers.settings.general import
GeneralSettingsController
14 from lbowrx_lb.controllers.settings.servers import
ServersListController, NewServerController, ServerController
15 from lbowrx_lb.controllers.settings.profiles import
ProfilesListController, NewProfileController, ProfileController
16 from lbowrx_lb.controllers.settings.sdrs import SdrsListController
17
18 from lbowrx_lb.controllers.imageupload import ImageUploadController
19 from lbowrx_lb.controllers.session import SessionController
20 from lbowrx_lb.controllers.pwc import PasswordChangeController
21
22
23 logger = logging.getLogger(__name__)
24 logger.setLevel(logging.INFO)
25
26
27 class Request(object):
28     def __init__(self, url, method, headers):
29         parsed_url = urlparse(url)
30         self.path = parsed_url.path
31         self.query = parse_qs(parsed_url.query)
32         self.matches = None
33         self.method = method
34         self.headers = headers
35         self.cookies = SimpleCookie()
36         if "Cookie" in headers:
37             self.cookies.load(headers["Cookie"])
38
39     def setMatches(self, matches):
40         self.matches = matches
41
42
43 class Route(ABC):
44     def __init__(self, controller, method="GET", options=None):
45         self.controller = controller
46         self.controllerOptions = options if options is not None else {}
```

```
47          self.method = method
48
49      @abstractmethod
50      def matches(self, request):
51          pass
52
53
54  class StaticRoute(Route):
55      def __init__(self, route, controller, method="GET", options=None):
56          self.route = route
57          super().__init__(controller, method, options)
58
59      def matches(self, request):
60          return request.path == self.route and self.method ==
request.method
61
62
63  class RegexRoute(Route):
64      def __init__(self, regex, controller, method="GET", options=None):
65          self.regex = re.compile(regex)
66          super().__init__(controller, method, options)
67
68      def matches(self, request):
69          matches = self.regex.match(request.path)
70          # this is probably not the cleanest way to do it...
71          request.setMatches(matches)
72          return matches is not None and self.method == request.method
73
74
75  class Router(object):
76      def __init__(self):
77          self.routes = [
78              StaticRoute("/", IndexController),
79              RegexRoute("^(/favicon.ico)$", OwrxAssetsController),
80              RegexRoute("^/static/(.+)$", OwrxAssetsController),
81              StaticRoute("/compiled/websocket-vars.js",
CompiledAssetsControllerVars),
82              RegexRoute("^/compiled/(.+)$",
CompiledAssetsControllerLibs),
83              StaticRoute("/settings", SettingsController),
84
85              StaticRoute("/settings/general", GeneralSettingsController),
86              StaticRoute("/settings/general", GeneralSettingsController,
method="POST", options={"action": "processFormData"}),
87
88              StaticRoute("/settings/servers", ServersListController),
89              StaticRoute("/settings/newserver", NewServerController),
90              StaticRoute("/settings/newserver", NewServerController,
method="POST", options={"action": "processFormData"}),
91
92              RegexRoute("^/settings/server/([^/]+)$", ServerController),
93              RegexRoute("^/settings/server/([^/]+)$", ServerController,
method="POST", options={"action": "processFormData"}),
94              RegexRoute("^/settings/deleteserver/([^/]+)$",
ServerController, options={"action": "deleteServer"}),
95
96              StaticRoute("/settings/profiles", ProfilesListController),
97              StaticRoute("/settings/newprofile", NewProfileController),
```

```
98              StaticRoute("/settings/newprofile", NewProfileController,
method="POST", options={"action": "processFormData"}),
99
100             RegexRoute("^/settings/profile/([^/]+)$",
ProfileController),
101             RegexRoute("^/settings/profile/([^/]+)$", ProfileController,
method="POST", options={"action": "processFormData"}),
102             RegexRoute("^/settings/deleteprofile/([^/]+)$",
ProfileController, options={"action": "deleteProfile"}),
103
104             StaticRoute("/settings/sdrs", SdrsListController),
105
106             StaticRoute("/imageupload", ImageUploadController),
107             StaticRoute("/imageupload", ImageUploadController,
method="POST", options={"action": "processImage"}),
108
109             StaticRoute("/login", SessionController, options={"action":
"loginAction"}),
110             StaticRoute("/login", SessionController, method="POST",
options={"action": "processLoginAction"}),
111             StaticRoute("/pwchange", PasswordChangeController),
112             StaticRoute("/pwchange", PasswordChangeController,
method="POST", options={"action": "processPwChange"})
113
114         ]
115
116     def find_route(self, request):
117         for r in self.routes:
118             if r.matches(request):
119                 return r
120
121     def route(self, handler, request):
122         route = self.find_route(request)
123         if route is not None:
124             controller = route.controller
125             controller(handler, request,
route.controllerOptions).handle_request()
126         else:
127             handler.send_error(404, "Not Found", "The page you requested
could not be found.")
128
129
130class RequestHandler(BaseHTTPRequestHandler):
131     timeout = 30
132     router = Router()
133
134
135     def log_message(self, format, *args):
136         # fix default implementation
137         pass
138
139     def do_GET(self):
140         self.router.route(self, self._build_request("GET"))
141
142     def do_POST(self):
143         self.router.route(self, self._build_request("POST"))
144
145     def do_DELETE(self):
```

```
        self.router.route(self, self._build_request("DELETE"))

    def _build_request(self, method):
        return Request(self.path, method, self.headers)

```

lbowrx-owrx/lbowrx_owrx/owrx/controllers/state.py

```
1 import json
2 from uuid import uuid4
3
4 from owrx.controllers import Controller
5 from owrx.sdr import SdrService
6 from owrx.config import Config
7 from owrx.connection import OpenWebRxReceiverClient
8 from owrx.source import SdrClientClass
9 from owrx.cpu import CpuUsageThread
10
11
12
13class StateController(Controller):
14    init_state_tag = str(uuid4())
15
16    @classmethod
17    def update_state_tag(cls):
18        cls.init_state_tag = str(uuid4())
19
20# Sends all sdrs and profiles of the server. Sends once at beggining.
21class InitialStateController(StateController):
22    def indexAction(self):
23        sdrs_dict = {}
24        for sdrid,sdr in Config.get()["sdrs"].items():
25            sdr_full_dict = {}
26            sdr_full_dict.update({"name": sdr["name"]})
27            if "type" in sdr.keys():
28                sdr_full_dict.update({"type": sdr["type"]})
29            if "device" in sdr.keys():
30                sdr_full_dict.update({"device": sdr["device"]})
31            if "enabled" in sdr.keys():
32                sdr_full_dict.update({"enabled": sdr["enabled"]})
33            if "priority" in sdr.keys():
34                sdr_full_dict.update({"priority": sdr["priority"]})
35            else:
36                print("\nError: You need to specify priority in settings
of {}\n".format(sdr["name"]))
37
38            sdr_profiles_dict = {}
39            for profileid,profile in sdr["profiles"].items():
40                profile_dict = {}
41                for (key,value) in profile.items():
42                    profile_dict.update({key:value})
43                sdr_profiles_dict.update({profileid: profile_dict})
44
45            sdr_full_dict.update({"profiles": sdr_profiles_dict})
46
47            sdrs_dict.update({sdrid: sdr_full_dict})
48
49
50        sdrs_dict.update({"init_state_tag": self.init_state_tag})
```

```
51
52          json_state = json.dumps(sdrs_dict)
53          self.send_response(json_state, content_type="application/json")
54
55# Sends periodically and on demand the part of state that changes.
56class CurrentStateController(StateController):
57      def indexAction(self):
58          current_state = {}
59          current_state.update({"latest_avg_cpu_usage":
CpuUsageThread.getLatestCpu()})
60          sources = SdrService.getAllSources()
61
62          for sdrid,source in sources.items():
63              sdr_info_dict = {}
64              sdr_name = (Config.get()["sdrs"][sources[sdrid].id]["name"])
65              sdr_info_dict.update({"sdr_name": sdr_name})
66
67              sdr_info_dict.update({"has_users":
source.hasClients(SdrClientClass.USER)})
68              connections = []
69              clients = source.getClients()
70              for client in clients:
71                  if isinstance(client, OpenWebRxReceiverClient):
72                      connections.append(client)
73              sdr_info_dict.update({"number_of_connections":
len(connections)})
74
75              profiles = source.getProfiles()
76              current_profile = profiles[source.getProfileId()]
77
78              sdr_info_dict.update({"current_profile": {"profile_id":
source.getProfileId()}})
79
sdr_info_dict["current_profile"].update(current_profile.__dict__())
80
81              sdr_info_dict.update({"failed": source.isFailed()})
82
83              current_state.update({sdrid: sdr_info_dict})
84
85
86          current_state.update({"init_state_tag": self.init_state_tag})
87
88          json_state = json.dumps(current_state)
89          self.send_response(json_state, content_type="application/json")
90
```