ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ασύρματη μετάδοση δεδομένων με τη χρήση μηχανισμών έλεγχου ενέργειας μετάδοσης

Συγγραφέας
Ταβουλάρης Νικόλαος
Α.Μ. 4266

Υπεύθυνος Καθηγητής
Χρήστος Μπούρας, Καθηγητής

Επιβλέπων
Κώστας Στάμος

ΠΑΤΡΑ, Σεπτέμβριος 2015

Περιεχόμενα

Κατάλογος Εικόνων 3
Ευχαριστίες 4
Περίληψη 4
Κεφάλαιο 1 Εισαγωγή 7
1.1 Τεχνολογική ανάπτυξη 7
1.2 Φορητότητα 7
1.3 Ενεργειακές ανάγκες 8
1.4 Ασύρματη δικτύωση 8
Κεφάλαιο 2 Η οικογένεια προτύπων 802.11 9
2.1 Εισαγωγή 9
2.2 Συχνότητες και κανάλια 9
2.3 Σχεδιασμός 10
Σταθμοί 10
Βασικό σύνολο υπηρεσιών (Basic Service Set - BSS) 11
Βασική περιοχή υπηρεσιών (Basic Service Area - BSA) 11
Σημείο πρόσβασης (Access Point - AP) 11
2.4 Είδη Δικτύων 12
Σύνολα με υποδομή (Infrastructure BSS) 12
Ανεξάρτητα Σύνολα (Independent BSS) 12
Σύνολα τύπου mesh (mesh Basic Service Set - MBSS) 13
2.5 Έλεγχος Ισχύος Μετάδοσης (Transmit Power Control) 14
2.6 Δυναμική Επιλογή Συχνότητας (Dynamic Frequency Selection) 14
2.7 Διαχείριση ενέργειας 14
Διαχείριση ενέργειας σε Infrastructure BSS 15
Διαχείριση ενέργειας σε Independent BSS 15
Κεφάλαιο 3 Σχετικές εργασίες 17
Κεφάλαιο 4 Αρχιτεκτονική του μηχανισμού 19

4.1 Κεραίες 19
4.2 Μετρικές της ποιότητας ασυρμάτων συνδέσεων 19
Bit Error Rate (BER) 20
Packet Delivery Ratio (PDR) 20
Signal to Noise Ratio (SNR) 21
Received Signal Strength Indicator (RSSI) 22
4.3 Αλλοίωση του σήματος 23
Το πρόβλημα του κρυμμένου τερματικού. 25
4.4 Εξασθένιση του σήματος 26
4.5 Ισχύς εκπομπής 27
4.6 Ανταλλαγή μηνυμάτων 28
Κεφάλαιο 5 Υλοποίηση του μηχανισμού 31
5.1 Εισαγωγή 31
5.2 Το λειτουργικό σύστημα GNU Linux 31
Ιστορική αναδρομή 31
Χρήσεις του Linux 32
server συστήματα 32
Linux ενσωματωμένο σε συσκευές 33
Linux σε κινητές συσκευές 33
5.3 Θέματα πυρήνα 33
5.4 Kernel modules 34
5.5 Προγράμματα οδήγησης (drivers) 34
5.5 Blocking calls 35
5.6 Εικονικά αρχεία 35
5.7 Υλοποίηση σε επίπεδο driver 36
Θέματα κώδικα 37
5.8 Υλοποίηση σε επίπεδο εφαρμογής 42
Ανταλλαγή μηνυμάτων 42

Τύποι μηνυμάτων 42
Threads 44
Κεφάλαιο 6 Πειράματα και αποτελέσματα 46
6.1 Πρώτη σειρά πειραμάτων. 46
6.2 Δεύτερη σειρά πειραμάτων. 48
6.3 Τρίτη σειρά πειραμάτων. 50
Κεφάλαιο 7 Επιπλέον εργασία 54
Κεφάλαιο 8 Επίλογος 57
Αναφορές 58
Παράρτημα - Κώδικες 60

Κατάλογος Εικόνων

Εικονα 1 επικαληψη καναλιων 9
Εικόνα 2 Infrastructure BSS 12
Εικόνα 3 Independent BSS 13
Εικονα 4 Συνολα τυπου mesh 13
Εικονα 5 Το PDR μειωνετε βαση του μεγεθους των πακετων 21
Εικονα 6 Σχεση SNR - PDR 22
Εικονα 7 RSSI σε περιβαλλων με παρεμβολες 23
Εικονα 8 hidden terminal problem 25
Εικονα 9 Πρωτο πειραμα - τοπολογια 47
Εικονα 11 Σχεση εντασης εκπομπης-αποστασης 48
Εικονα 12 Σχεση PDR - αποστασης 49
Εικονα 13 πειραμα 3 - τοπολογια 50
Εικονα 14 σταθμος 1 στα 10μ.- μεση ισχυς εκπομπης 51
Εικονα 15 σταθμος 1 στα 10μ. PDR 51
Εικονα 16 15 σταθμος 1 στα 30μ.- μεση ισχυς εκπομπης 52
Εικονα 17 σταθμος 1 στα 30μ. - PDR 52
Εικονα 18 Σχεση SNR - PDR 54

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τους ανθρώπους που με βοήθησαν να φέρω εις πέρας αυτήν την εργασία.

Τον επιβλέποντα καθηγητή μου Χρήστο Μπούρα για την πολύτιμη καθοδήγηση του.

Τον Κωνσταντίνο Στάμο για την συμβολή του

Τον Σταθόπουλο Νικόλαο για την βοήθεια και την υποστήριξη του.

Περίληψη

Σε αυτή τη διπλωματική εργασία παρουσιάζεται ένας μηχανισμός που προσαρμόζει την ένταση του εκπεμπόμενου σήματος βάσει της ποιότητας της σύνδεσης. Σκοπός του μηχανισμού αυτού είναι να μειώσει την δαπανώμενη ενέργεια κατά την διάρκεια ασύρματων μεταδόσεων διατηρώντας όμως την ποιότητα της σύνδεσης σε ικανοποιητικά επίπεδα. Για τη μέτρηση της ποιότητας της σύνδεσης χρησιμοποιείται ένας δείκτης για την ένταση του ληφθέντος σήματος (received signal strength indicator ή RSSI), ο οποίος υπολογίζεται από τον driver της ασύρματης κάρτας δικτύου. Μια άλλη παρόμοια εργασία είχε παρουσιαστεί από τους Anmol Sheth και Richard Han με τίτλο "*An Implementation of Transmit Power Control in 802.11b Wireless Networks*" [1]. Η εργασία αυτή ρύθμιζε την ένταση του εκπεμπόμενου σήματος μεταξύ μόνο δύο κόμβων. Στην παρούσα διπλωματική ο μηχανισμός υποστηρίζει την ύπαρξη πολλαπλών κόμβων. Πιο συγκεκριμένα υπάρχει ένας σταθμός βάσης πάνω στον οποίο συνδέονται οι κόμβοι. Οι κόμβοι ρυθμίζουν την ενέργεια εκπομπής ώστε να έχουν ικανοποιητική σύνδεση με τη βάση ενώ ο σταθμός βάσης τη ρυθμίζει με τέτοιο τρόπο ώστε να μπορεί να επικοινωνεί με όλους τους συνδεδεμένους σε αυτόν κόμβους.

Γενικά θα μπορούσαμε να θεωρήσουμε ότι ο σταθμός βάσης προσομοιώνει κάποιο router ή κάποια κινητή συσκευή που διαμοιράζει μια σύνδεση (mobile hotspot), ενώ οι κόμβοι είναι κινητές συσκευές όπως κινητά τηλέφωνα, laptop κ.α.

Τέλος για την τεκμηρίωση της αποδοτικότητας του μηχανισμού αυτού διεξήχθησαν πειράματα σε πραγματικό περιβάλλον με τη χρήση laptop. Τα αποτελέσματα των πειραμάτων παρουσιάζονται στο αντίστοιχο κεφάλαιο.

Στα πλαίσια της παρούσας διπλωματικής εργασίας, δημοσιεύθηκε η εργασία με τίτλο:

Feedback-based Adaptation for Improved Power Consumption metrics"

Συγγραφείς: Χρήστος Μπούρας
Βαγγέλης Καπούλας
Κώστας Στάμος
Νικόλαος Ταβουλάρης
Γιώργος Κιουμουρτζής
Νικόλαος Σταθόπουλος

Συνέδριο: 27th IEEE Intenational Conference on Advanced Information Networking and Applications (AINA - 2013), Barcelona, Spain,

Περίληψη:

In this paper we present a feedback-based adaptation mechanism that adjusts the transmission power of a wireless card on commodity PCs depending on the quality of the connection. Our purpose is to manage the available power in order to achieve lower power consumption without negatively affecting the user's perception of connection quality. We based our implementation on an existing theoretical model and focused on resolving problems and removing assumptions which made it inefficient in real life implementation. The initial model manages to minimize the power consumption in networks with exactly two nodes. In this paper, we extend the model to consider the possibility of the existence of a base station, where any number of nodes can be connected. Our objectives for the base station are to

minimize the power consumption and guarantee continuous connectivity for all mobile nodes. We implement the adaptation mechanism for a specific adapter with open sources drivers thus allowing necessary modifications. We conduct a number of real world experiments. The results indicate that power consumption can be significantly reduced for nodes that are either almost stationary or slowly moving (e.g. at walking speed), without any significant increase in packet loss. The results are quite important as nowadays mobile devices with limited battery life time use tethering to become base stations for other devices like in ad-hoc networks

Κεφάλαιο 1

Εισαγωγή

1.1 Τεχνολογική ανάπτυξη

Τα τελευταία χρόνια παρατηρείται μια ραγδαία τεχνολογική ανάπτυξη σε όλες της επιστήμες η οποία έχει επηρεάσει άμεσα την καθημερινότητα του σύγχρονου ανθρώπου. Πρωτοπόρος στις τεχνολογικές αυτές εξελίξεις βρίσκεται η επιστήμη της πληροφορικής. Με την εξέλιξη του υλικού των υπολογιστών (hardware) αυξάνεται συνεχώς η παρεχόμενη υπολογιστική ισχύς, ενώ μειώνεται το μέγεθος των συσκευών. Παράλληλα η ανάπτυξη των τηλεπικοινωνιών δίνει τη δυνατότητα σε όλο και περισσότερους ανθρώπους να αποκτήσουν πρόσβαση στο διαδίκτυο με όλο και ταχύτερες συνδέσεις. Η βελτίωση αυτή των συνδέσεων οδηγεί στην αύξηση του όγκου της παρεχόμενης πληροφορίας καθώς και των διαδικτυακών υπηρεσιών.

1.2 Φορητότητα

Η εξέλιξη στο hardware των υπολογιστών έχει δώσει τη δυνατότητα δημιουργίας πληθώρας φορητών συσκευών όπως laptop, smartphones, tablets κ.α. Κύριο χαρακτηριστικό αυτών των συσκευών είναι ότι ο χρήστης μπορεί να τις μετακινεί ή ακόμα και να τις έχει συνεχώς μαζί του καθώς δεν απαιτείται να βρίσκονται συνεχώς συνδεδεμένες σε κάποια εξωτερική πηγή τροφοδοσίας. Σημαντικός παράγοντας στην εξέλιξη των φορητών συσκευών έπαιξε η ασύρματη δικτύωση που παρέχει την δυνατότητα σύνδεσης στο διαδίκτυο χωρίς την ύπαρξη καλωδίων. Στης μέρες μας οι φορητές συσκευές γίνονται όλο και πιο δημοφιλείς λόγω της ευκολίας και της ευελιξίας που παρέχουν καθώς και της ανάγκης του ανθρώπου να βρίσκεται συνεχώς συνδεδεμένος και να έχει άμεση πρόσβαση σε live πληροφορίες. Η αυτονομία όμως των φορητών συσκευών περιορίζεται από τις ενεργειακές τους ανάγκες και το μέγεθος της μπαταρίας.

1.3 Ενεργειακές ανάγκες

Η τροφοδοσία των φορητών συσκευών γίνεται κυρίως μέσω μπαταρίας. Έτσι ο χρόνος αυτόνομης λειτουργίας τους είναι περιορισμένος. Κύριοι παράγοντες που τον επηρεάζουν είναι η χωρητικότητα της μπαταρίας καθώς και οι ίδιες οι ανάγκες της συσκευής σε ενέργεια. Η αυξημένη υπολογιστική ισχύς, καθώς και η συνεχής σύνδεση των συσκευών στο διαδίκτυο οδηγεί και σε αυξημένες ενεργειακές ανάγκες. Μια λύση στο πρόβλημα θα μπορούσε να είναι η χρήση μπαταριών μεγαλύτερης χωρητικότητας. Δυστυχώς όμως μπαταρίες μεγάλης χωρητικότητας συνήθως οδηγούν σε μεγαλύτερο μέγεθος συσκευής ή σε αυξημένο χρηματικό κόστος. Λόγω του παραπάνω γίνεται σαφές ότι η αποδοτική εκμετάλλευση των πόρων της συσκευής είναι αναγκαία. Πιο συγκεκριμένα όσον άφορα τα κινητά τηλέφωνα (smartphones) ο Muhammad Yasir Malik στην έρευνα του **'**Power Consumption Analysis of a Modern Smartphone**'** [2] υπολόγισε ότι η ασύρματη δικτύωση (WI-FI) είναι ένα από τα πιο δαπανηρά από άποψη ενέργειας μέρη ενός κινητού τηλεφώνου. Ο μηχανισμός που παρουσιάζεται εδώ έχει σκοπό να μειώσει τη δαπάνη ενέργειας κατά την ασύρματη μετάδοση.

1.4 Ασύρματη δικτύωση

Η ασύρματη δικτύωση των συσκευών γίνεται κυρίως με ραδιοσυχνότητες. Οι συσκευές περιλαμβάνουν κεραίες και ειδικό hardware που αναλαμβάνει την επικοινωνία. Για την συνεργασία όμως διαφορετικών συσκευών είναι απαραίτητο ένα πρότυπο που καθορίζει την λειτουργία τους. Στις ασύρματες επικοινωνίες υπάρχει μια οικογένεια προτύπων για αυτό το σκοπό που ονομάζεται 802.11. Η οικογένεια αυτή περιλαμβάνει ένα σύνολο προτύπων κατά κύριο λόγο συμβατά μεταξύ τους όπως το 802.11b, το 802.11g , το 802.11n κ.α. Τα πρότυπα αυτά καθορίζουν δύο κυρίως τρόπους επικοινωνίας. Την άμεση επικοινωνία των κόμβων (ad-hoc δίκτυα), ή την επικοινωνία μέσω κάποιου σταθμού βάσης (infrastructure δίκτυο). Και οι δύο τρόποι είναι διαδεδομένοι στις μέρες μας αλλά πιο συνηθισμένα είναι τα infrastructure δίκτυα όπου ο σταθμός βάσης μπορεί να συνδεθεί ενσύρματα στην τηλεφωνική γραμμή και να αποκτήσει καθώς και να διαμοιράσει πρόσβαση στο διαδίκτυο.

Κεφάλαιο 2

Η οικογένεια προτύπων 802.11

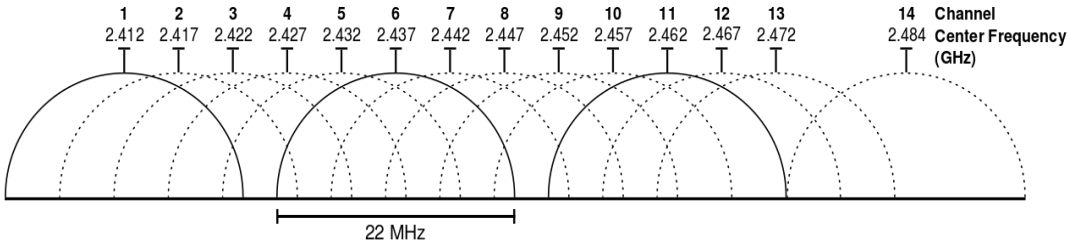
2.1 Εισαγωγή

Το 802.11 αποτελεί μια οικογένεια προτύπων για την ασύρματη δικτύωση υπολογιστών και άλλων συσκευών. Στην αγορά είναι κυρίως γνωστό με την ονομασία wi-fi από τον οργανισμό 'wi-fi aliance' που πιστοποιεί την συμβατότητα τον ασύρματων συσκευών με τα 802.11 πρότυπα. Πριν από το 802.11 οι συσκευές συνδέονταν συνήθως σε τοπικά δίκτυα με χρήση καλωδίων (ενσύρματα). Το πρότυπο που επικρατούσε τότε ήταν το ethernet ή 802.3 το οποίο χρησιμοποιείται και σήμερα για την ενσύρματη δικτύωση. Το 1997 δημοσιεύτηκε το πρώτο πρότυπο για ασύρματη δικτύωση που ονομάστηκε 802.11-1997. Το πρότυπο αυτό διαδέχτηκε μια πληθώρα προτύπων με επεκτάσεις και διορθώσεις. Τα σημαντικότερα από αυτά είναι το 802.11a το οποίο επέκτεινε την μπάντα συχνοτήτων από το 2.4GHz στα 5GHz, το 802.11b που αύξησε το throughput στα 11Mbit/s για την μπάντα των 2.4GHz ,το 802.11g που αύξησε περαιτέρω το throughput καθώς και το 802.11n που συνδύασε τις δυο μπάντες συχνοτήτων.

2.2 Συχνότητες και κανάλια

Το 802.11b καθώς και το 802.11g εκπέμπουν στο εύρος συχνοτήτων από 2.400 έως 2.500 GHz.

Οι συχνότητες αυτές μοιράζονται σε δεκατέσσερα κανάλια εύρους ζώνης 22 MHz. Ηνεικόνα που ακολουθεί περιγράφει σχηματικά τα κανάλια αυτά.



Εικόνα 1 επικάληψη καναλίων

Όπως φαίνεται απ' το σχήμα τα κανάλια αυτά επικαλύπτονται. Ως αποτέλεσμα η εκπομπή σε ένα κανάλι μπορεί να δημιουργήσει παρεμβολές σε κάποιο δίκτυο που χρησιμοποιεί διαφορετικό κανάλι. Η μείωση της έντασης του σήματος εκπομπής μπορεί να μετριάσει της παρεμβολές που δημιουργεί ένας σταθμός σε άλλα δίκτυα.

Κύρια διαφορά του 802.11b από το 802.11g είναι το throughput. Το 802.11b έχει μέγιστο throughput τα 11Mbit/s. Αντίθετα το 802.11g πετυχαίνει σημαντικά καλύτερη απόδοση καθώς το μέγιστο throughput ανέρχεται πλέον στα 54 Mbit/s.

Το 802.11a εκπέμπει στην μπάντα συχνοτήτων των 5GHz. Η μπάντα αυτή αντιμετώπιζε αρκετά προβλήματα καθώς υπήρχε μεγάλη πιθανότητα να δημιουργήσει παρεμβολές σε δορυφόρους, ραντάρ και κρατικά δίκτυα. Για το λόγο αυτό πολλές χώρες απαγόρευαν τη χρήση αυτών των συχνοτήτων από ιδιώτες (Η.Π.Α. και Ε.Ε.). Μετέπειτα πρότυπα έλυσαν αρκετά προβλήματα παρεμβολών και έτσι η χρήση τους τελικά επιτράπηκε.

Τέλος το 802.11n συνδυάζει της δύο μπάντες συχνοτήτων ώστε να είναι συμβατό με όλα τα προηγούμενα πρότυπα. Επίσης βελτιώνει σημαντικά την απόδοση καθώς πετυχαίνει μέγιστο throughput τα 600 Mbit/s.

2.3 Σχεδιασμός

Σταθμοί

Οποιαδήποτε συσκευή συνδεδεμένη σε ένα ασύρματο δίκτυο αποτελεί έναν σταθμό (station). Ο κάθε σταθμός έχει μια μοναδική διεύθυνση σε αυτό το δίκτυο που χαρακτηρίζει τα μηνύματα που στέλνει καθώς και αυτά που λαμβάνει. Σε αντίθεση με τα ενσύρματα δίκτυα στο 802.11 η διεύθυνση αυτή δεν αντιστοιχεί σε κάποια φυσική τοποθεσία στο χώρο καθώς οι σταθμοί έχουν την δυνατότητα κίνησης. Γενικά το πρότυπο ορίζει τρία είδη σταθμών βάσει της κινητικότητας τους.

1. **Σταθεροί σταθμοί (fixed stations)**
   Ένας σταθμός θεωρείται σταθερός όταν δεν κινείται στο χώρο.
2. **Φορητοί σταθμοί (portable stations)**
   Φορητοί λέγονται οι σταθμοί που έχουν την δυνατότητα να αλλάξουν θέση αλλά για όση ώρα είναι συνδεδεμένοι βρίσκονται σε σταθερή τοποθεσία.

3. **Κινητοί σταθμοί (mobile stations)**
   Κινητοί λέγονται οι σταθμοί που κινούνται ελεύθερα στο χώρο ακόμα και κατά την περίοδο που βρίσκονται συνδεδεμένοι. Οι κινητοί σταθμοί συνήθως λειτουργούν με μπαταρία οπότε η αποδοτική διαχείριση της ενέργειας τους είναι απαραίτητη.

Γενικά από τεχνικής απόψεως είναι δύσκολος ο διαχωρισμός του φορητού σταθμού από τον κινητό.

Βασικό σύνολο υπηρεσιών (Basic Service Set - BSS)

Οι σταθμοί μπορούν να συνδέονται μεταξύ τους και να ανταλλάξουν μηνύματα. Μια ομάδα συνδεδεμένων σταθμών αποτελούν ένα δίκτυο που καλείται βασικό σύνολο υπηρεσιών.

Βασική περιοχή υπηρεσιών (Basic Service Area - BSA)

Καθώς η εκπομπή των σταθμών έχει μια εμβέλεια, ένα βασικό σύνολο υπηρεσιών λαμβάνει χώρα σε μια περιορισμένη περιοχή που καλείται βασική περιοχή υπηρεσιών.

Σημείο πρόσβασης (Access Point - AP)

Εκτός από την σύνδεση των σταθμών μεταξύ τους θα θέλαμε να υπάρχει η δυνατότητα επικοινωνίας και με άλλα εξωτερικά δίκτυα. Χαρακτηριστικό παράδειγμα είναι το διαδίκτυο (internet) όπου αποτελεί την ένωση πολλών διαφορετικών δικτύων. Για να επιτευχθεί η πρόσβαση σε διαφορετικού τύπου δίκτυα ένας σταθμός αναλαμβάνει να παίξει των ρόλο του διαμεσολαβητή μεταξύ των δικτύων αυτών. Ο σταθμός αυτός ονομάζεται σημείο πρόσβασης (Access Point - AP). Οι υπόλοιποι σταθμοί συνδέονται στο AP και επικοινωνούν με αυτό όποτε επιθυμούν να στείλουν κάποιο μήνυμα. Το AP αναλαμβάνει να προωθήσει το μήνυμα αυτό ανάλογα με τον τελικό του παραλήπτη.

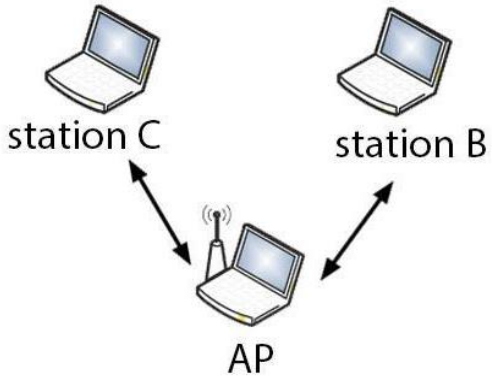
Στο εμπόριο είναι πολύ διαδεδομένα τα λεγόμενα modem routers που συνδέονται ενσύρματα στο internet και διαμοιράζουν την σύνδεση αυτή μέσω ενός τοπικού ασύρματου δικτύου.

2.4 Είδη Δικτύων

Το 802.11 καθορίζει τρία είδη βασικών συνόλων υπηρεσιών ανάλογα με τον τρόπο επικοινωνίας των σταθμών. Αυτά είναι τα Σύνολα με υποδομή, τα Ανεξάρτητα Σύνολα και τα σύνολα τύπου mesh.

Σύνολα με υποδομή (Infrastructure Basic Service Set - Infrastructure BSS)

Τα σύνολα αυτά αποτελούνται από σταθμούς που επικοινωνούν μέσω κάποιου access point. Οι σταθμοί δεν έχουν άμεση επικοινωνία μεταξύ τους αλλά όλα τα μηνύματα περνάνε από το AP. Αυτά είναι τα πιο συχνά απαντώμενα σύνολα και συνήθως οι σταθμοί τους έχουν πρόσβαση και στο διαδίκτυο.

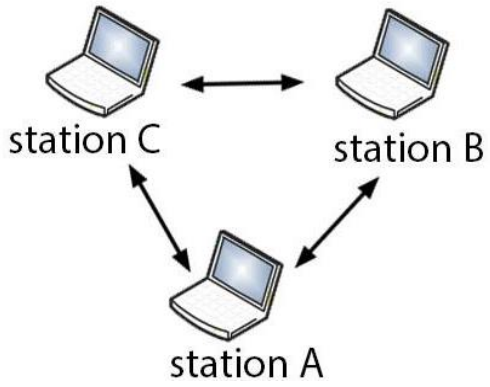


Εικόνα 2 Infrastructure BSS

Ανεξάρτητα Σύνολα (Independent Basic Service Set - Independent BSS)

Ο κάθε σταθμός είναι ανεξάρτητος και μπορεί να επικοινωνήσει άμεσα με όλους τους σταθμούς στο δίκτυο. Σε αυτήν την περίπτωση δεν υπάρχει κάποιο

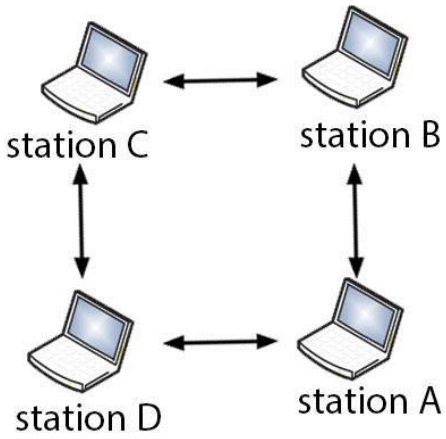
access point. Μια κοινή ονομασία που περιγράφει τα ανεξάρτητα σύνολα είναι ο αγγλικός όρος ad-hoc network.



Εικόνα 3 Independent BSS

Σύνολα τύπου mesh (mesh Basic Service Set - MBSS)

Τα σύνολα τύπου mesh αποτελούνται και αυτά από ανεξάρτητους σταθμούς και όπως και με τα ad-hoc δίκτυα απουσιάζει το Access Point. Σε αυτήν την περίπτωση όμως ο κάθε σταθμός δεν έχει απαραίτητα άμεση επικοινωνία με όλους τους άλλους σταθμούς στο δίκτυο. Έτσι η δρομολόγηση γίνεται αρκετά πιο περίπλοκη και απαιτούνται ειδικοί αλγόριθμοι.



Εικόνα 4 Σύνολα τύπου mesh

2.5 Έλεγχος Ισχύος Μετάδοσης (Transmit Power Control)

Όπως ήδη αναφέρθηκε η μετάδοση στα 5GHz μπορεί να δημιουργήσει παρεμβολές σε ραντάρ και κρατικά δίκτυα. Για την μείωση τέτοιων παρεμβολών το 802.11 ορίζει των έλεγχο της ισχύος μετάδοσης σε αυτή την μπάντα. Βασικός σκοπός του ελέγχου αυτού είναι να δοθεί προτεραιότητα στις κρατικές μεταδόσεις. Έτσι σε περίπτωση παρατήρησης παρεμβολών η ισχύς μειώνεται στο ελάχιστο δυνατό ώστε να διατηρείται η σύνδεση και να ελαχιστοποιηθούν οι παρεμβολές. Ο Έλεγχος Ισχύος Μετάδοσης δεν έχει σκοπό να εξοικονομήσει ενέργεια και δεν εφαρμόζεται ρητά στα 2.4 GHz.

Για τα 2.4 GHz το πρότυπο ορίζει ότι η ισχύς μετάδοσης δεν θα ξεπερνά ένα μέγιστο όριο. Παρόλα αυτά ο κατασκευαστής μπορεί ελεύθερα να εφαρμόσει οποιονδήποτε αλγόριθμο ελέγχου ισχύος επιθυμεί βάσει των δικών του κριτηρίων.

2.6 Δυναμική Επιλογή Συχνότητας (Dynamic Frequency Selection)

Αν παρατηρηθούν παρεμβολές από κάποιο ραντάρ, αντί να εφαρμοστεί ο έλεγχος ισχύος μετάδοσης θα ήταν αποδοτικότερο να αλλάξει το κανάλι που χρησιμοποιείται. Το 802.11 ορίζει την δυναμική επιλογή συχνότητας ώστε στην περίπτωση παρεμβολών το BSS να χρησιμοποιήσει κάποιο άλλο κανάλι. Αρχικά το Access Point ελέγχει τα διαθέσιμα κανάλια και επιλέγει αυτό με τις λιγότερες παρεμβολές από ραντάρ. Έπειτα ενημερώνει τους σταθμούς για το νέο κανάλι.

Η Δυναμική Επιλογή Συχνότητας όπως και ο Έλεγχος Ισχύος Μετάδοσης έχουν στόχο να μειώσουν τις παρεμβολές που δημιουργούν οι ιδιώτες σε κρατικά δίκτυα και όχι να βελτιώσουν την σύνδεση.

2.7 Διαχείριση ενέργειας

Το 802.11 εκτός από τον Έλεγχο Ισχύος Μετάδοσης καθορίζει και τη Διαχείριση και εξοικονόμηση ενέργειας των καρτών δικτύου. Η εξοικονόμηση ενέργειας γίνεται κυρίως με το να απενεργοποιείται η κεραία για κάποιες χρονικές στιγμές. Την περίοδο αυτή λέμε ότι ο σταθμός βρίσκεται σε κατάσταση νάρκης (sleep mode) και δεν καταναλώνει ενέργεια. Παράλληλα όμως δεν έχει την

δυνατότητα να λάβει ή να αποστείλει δεδομένα. Γενικά ανάλογα με τον τύπο του δικτύου η διαχείριση ενέργειας λειτουργεί διαφορετικά.

Διαχείριση ενέργειας σε Infrastructure BSS

Η διαχείριση ενέργειας σε Infrastructure BSS είναι σχετικά απλή. Το Access point στέλνει περιοδικά beacon μηνύματα με πληροφορίες σχετικά με το δίκτυο. Οι υπόλοιποι σταθμοί έχουν τη δυνατότητα να μπουν σε κατάσταση νάρκης αλλά πρέπει να ενεργοποιηθούν πριν περάσει η περίοδος των beacon μηνυμάτων (beacon interval). Κατά την απενεργοποίηση τους ο κάθε σταθμός ενημερώνει το AP το οποίο διατηρεί μια λίστα με τους σταθμούς σε εξοικονόμηση ενέργειας και αποθηκεύει προσωρινά το μηνύματα που προορίζονται γι' αυτούς. Επειδή όλοι οι σταθμοί ενεργοποιούνται πριν την αποστολή των beacon μηνυμάτων τα beacon μηνύματα λαμβάνονται πάντα. Στα μηνύματα αυτά εμφωλεύεται και πληροφορία για ποιούς σταθμούς εκκρεμούν αποθηκευμένα δεδομένα. Οι σταθμοί αυτοί δεν μπαίνουν σε κατάσταση νάρκης για την επόμενη περίοδο και το AP τους αποστέλλει τα πακέτα που εκκρεμούν. Για να μην δημιουργηθούν προβλήματα από την εξοικονόμηση ενέργειας είναι σημαντικό οι σταθμοί να ενεργοποιηθούν εγκαίρως (πριν σταλεί το beacon). Για να καταστεί αυτό εφικτό όλοι οι σταθμοί συγχρονίζονται με ένα κοινό ρολόι. Έτσι το AP αποστέλλει μια τιμή ρολογιού εμφωλευμένη στο beacon.

Διαχείριση ενέργειας σε Independent BSS

Η διαχείριση ενέργειας στα Independent BSS λειτουργεί με παρόμοιο τρόπο. Οι σταθμοί και πάλι ενεργοποιούνται πριν το πέρας της περιόδου των beacon μηνυμάτων. Αυτό που είναι πιο σύνθετο στα Independent BSS είναι ο συγχρονισμός των σταθμών και το κοινό ρολόι. Μετά το πέρας της περιόδου των beacon μηνυμάτων (beacon interval) ένα νέο beacon μήνυμα πρέπει να αποσταλεί. Ο κάθε σταθμός δημιουργεί το δικό του beacon αλλά καθυστερεί την αποστολή του κατά ένα τυχαίο χρονικό διάστημα. Αν μέσα στο διάστημα αυτό λάβει κάποιο beacon από κάποιον άλλο σταθμό τότε ακυρώνει την αποστολή του beacon που ο ίδιος δημιούργησε. Με αυτόν τον τρόπο οι σταθμοί συγχρονίζονται βάσει του ρολογιού του σταθμού που απέστειλε πρώτος beacon. Αφότου ένας σταθμός λάβει ένα beacon μήνυμα περιμένει ένα προκαθορισμένο χρονικό διάστημα πριν μπει σε κατάσταση νάρκης. Το διάστημα αυτό καλείται ATIM

WINDOW και δίνει αρκετό χρόνο στους σταθμούς να στείλουν τα λεγόμενα ATIM μηνύματα. Ένας σταθμός αποστέλλει ένα ATIM μήνυμα σε κάποιο άλλο σταθμό εντός δικτύου που επιθυμεί να επικοινωνήσει μαζί του. Ο δεύτερος σταθμός οφείλει να απαντήσει με ένα ACK και στην περίπτωση αυτή οι δύο σταθμοί δεν εισέρχονται σε κατάσταση νάρκης αλλά συνεχίζουν την επικοινωνία μέχρι το πέρας του beacon interval. Αν ένας σταθμός δεν λάβει κάποιο ATIM μήνυμα, αφότου περάσει το ATIM window μπορεί να μπει σε κατάσταση νάρκης για να εξοικονομήσει ενέργεια.

Τέλος η εξοικονόμηση ενέργειας κατά το 802.11 δεν καθορίζει κάποιον αλγόριθμο για την ρύθμιση της έντασης του σήματος εκπομπής αλλά αφήνει ελεύθερους τους κατασκευαστές να υλοποιήσουν τους δικούς τους αλγορίθμους.

Κεφάλαιο 3

Σχετικές εργασίες

Η εξοικονόμηση ενέργειας όπως και η προσαρμογή της έντασης εκπομπής αποτελεί μια πρόκληση για τους ερευνητές και τους σχεδιαστές δικτύων στις μέρες μας. Η ρύθμιση της έντασης εκπομπής μπορεί να εξοικονομήσει ενέργεια καθώς και να ελαττώσει τις παρεμβολές που προκαλούνται από το δίκτυο. Ως αποτέλεσμα υπάρχει μια πληθώρα αναφορών στην βιβλιογραφία για το συγκεκριμένο πρόβλημα και έχουν προταθεί αρκετοί μηχανισμοί για ρύθμιση της έντασης εκπομπής, ο καθένας εξειδικευμένος σε διαφορετικού τύπου δίκτυα και δεδομένα ενώ λίγοι από αυτούς έχουν δοκιμαστή σε πραγματικά δίκτυα.

Η εργασία [3] παρουσιάζει δύο μηχανισμούς για την προσαρμογή της έντασης εκπομπής ειδικευμένους για τη μεταφορά αρχείων βίντεο. Οι μηχανισμοί αυτοί επιτυγχάνουν μια ισορροπία μεταξύ της δαπανώμενης ενέργειας και της ποιότητας του βίντεο που μεταδίδεται, ενώ ο έλεγχος τους έχει πραγματοποιηθεί με χρήση του προγράμματος εξομοίωσης δικτύων ns2. Αντίστοιχα η εργασία [4] προτείνει ένα αποκεντρωμένο πρωτόκολλο για ρύθμιση της έντασης που πετυχαίνει την αύξηση του throughput. Και αυτό το πρωτόκολλο μελετήθηκε στον ns2 και δεν έχει δοκιμαστεί με πραγματικά πειράματα. Αλγόριθμοι ρύθμισης της έντασης έχουν προταθεί και για δίκτυα κυψελών. Η εργασία [5] ρυθμίζει την ένταση εκπομπής σε δίκτυα κυψέλης με σκοπό την μείωση των παρεμβολών. Επίσης ένας άλλος μηχανισμός ρυθμίζει την ισχύ του εκπεμπόμενου σήματος βάσει του RSSI [1]. Ο μηχανισμός αυτός χρησιμοποιεί το 802.11b πρότυπο και λειτουργεί μεταξύ δυο σταθμών στο επίπεδο μεταφοράς (TCP layer). Εντούτοις δεν υποστηρίζει τοπολογίες με περισσότερους σταθμούς.

Παράλληλα η διαχείριση ενέργειας καθώς και η ισχύς του σήματος κατά την λήψη του (ή το RSSI) παίζουν έναν σημαντικό ρόλο για την δρομολόγηση των πακέτων. Η εργασία “Power Management for Throughput Enhancement in Wireless Ad-Hoc Networks“ [6] δείχνει ότι σε ένα ad-hoc δίκτυο με πολλαπλούς σταθμούς, η κατάλληλη ισχύς εκπομπής μπορεί να μειώσει τις παρεμβολές καθώς και των αριθμό των αναμεταδόσεων (hops). Ως συνέπεια έχουμε την αύξηση του throughput. Επιπλέον η πατέντα [7] παρουσιάζει έναν αποτελεσματικό τρόπο δρομολόγησης βασισμένο στο RSSI. Επίσης η εργασία “On Passive Characterization of Aggregated Traffic in Wireless Networks“ [8] χρησιμοποιεί το

RSSI για την εκτίμηση της αξιοποίησης του καναλιού ώστε να θέσει με αποδοτικό τρόπο τα χαρακτηριστικά του mac layer όπως το μέγεθος του παραθύρου, την πολιτική αναμεταδόσεων κ.α.

Παρόλο που το RSSI αποτελεί ένα πολύ χρήσιμο εργαλείο, η χρήση του έχει κάποιους περιορισμούς. Κύριος περιορισμός είναι ότι δεν περιέχει πληροφορία για το θόρυβο και τις παρεμβολές στο κανάλι. Συνεπώς σε ένα περιβάλλον με υψηλό θόρυβο ή παρεμβολές το RSSI δεν είναι ικανοποιητικός δείκτης τις κατάστασης τις σύνδεσης [9].

Κεφάλαιο 4

Αρχιτεκτονική του μηχανισμού

4.1 Κεραίες

Όπως έχει ήδη αναφερθεί η ασύρματη μετάδοση πραγματοποιείται με ηλεκτρομαγνητικά κύματα μέσω κεραιών Ένα απλουστευμένο θεωρητικό μοντέλο θεωρεί την ύπαρξη ισοτροπικών κεραιών. Μια ισοτροπική κεραία ακτινοβολεί με την ίδια ισχύ σε όλες τις κατευθύνσεις. Τέτοιου είδους κεραίες, παρόλο που δεν μπορούν να κατασκευαστούν στην πράξη, αποτελούν ένα πολύτιμο θεωρητικό μοντέλο για τη μελέτη της εκπομπής του σήματος καθώς και πολλών άλλων φαινομένων. Στη πράξη, αντί για ισοτροπικές κεραίες μπορούμε να κατασκευάσουμε παγκατευθυντικές κεραίες (omnidirectional antennas) που να ακτινοβολούν ομοιόμορφα σε ένα επίπεδο. Καθώς αυξάνεται όμως η γωνία εκπομπής μειώνεται αντίστοιχα και η ισχύς. Τέλος ένα άλλο είδος κεραίας είναι οι κατευθυντικές κεραίες που έχουν αυξημένη ισχύ σε μια συγκεκριμένη κατεύθυνση.

4.2 Μετρικές της ποιότητας ασυρμάτων συνδέσεων

Για να μπορέσουμε να έχουμε έναν μηχανισμό που εγγυάται μια ικανοποιητική σύνδεση θα πρέπει πρώτα να απαντήσουμε στο ερώτημα πότε μια σύνδεση είναι καλή και πότε όχι. Δηλαδή χρειαζόμαστε κάποια μετρική για να βαθμολογεί την ποιότητα μιας σύνδεσης. Για να το πετύχουμε αυτό υπάρχουν διάφορες προσεγγίσεις. Στην εργασία “Wireless Networks: Which is the right metric?” [9] αναλύονται οι περισσότερες από αυτές και σε ποιες περιστάσεις η κάθε μια είναι κατάλληλη.

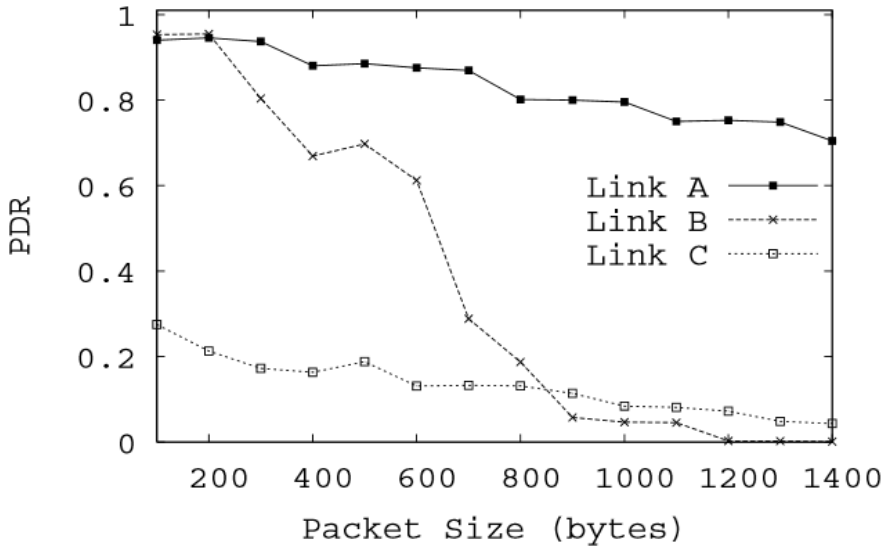
Bit Error Rate (BER)

Μια πρώτη προσέγγιση θα μπορούσε να είναι βάση του bit error rate (BER). Δηλαδή τον αριθμό των bit που ελήφθησαν λανθασμένα δια των συνολικό αριθμό των ληφθέντων bit. Είναι προφανές ότι όσο μεγαλύτερο είναι το BER τόσο χειρότερη είναι η σύνδεση μας. Το BER είναι σημαντικό γιατί μας δείχνει το σφάλμα στα ίδια τα δεδομένα μας, που είναι τελικά και αυτό που επιθυμούμε να ελαττώσουμε. Ο υπολογισμός του BEP όμως είναι σχετικά πολύπλοκος. Η κύρια μέθοδος για τον έλεγχο σφαλμάτων είναι ο κυκλικός έλεγχος πλεονασμού (Cyclic redundancy check - CRC). Εντούτοις ο CRC δεν μπορεί να μας πληροφορήσει για τον ακριβή αριθμό των bit που ελήφθησαν λανθασμένα. Για τον υπολογισμό λοιπόν του BER θα πρέπει να σταλούν κάποιες ήδη γνωστές ακολουθίες bit και να μετρηθούν τα bits που ελήφθησαν λανθασμένα. Κάτι τέτοιο όμως δεν είναι εφικτό σε live δεδομένα.

Packet Delivery Ratio (PDR)

Άλλη μετρική είναι το packet delivery ratio (PDR). Σε αυτήν την μετρική υπολογίζουμε τον λόγο των πακέτων που ελήφθησαν επιτυχώς προς το σύνολο των πακέτων που έστειλε ο αποστολέας. Όταν το PDR φτάσει το 100% τότε έχουμε μηδενικά σφάλματα μετάδοσης ενώ όσο μικρότερο είναι το PDR τόσο περισσότερα είναι και τα πακέτα που ελήφθησαν λανθασμένα.

Παρόλα αυτά το PDR δεν περιέχει πληροφορία για τους λόγους ύπαρξης του σφάλματος και για το πως αυτό σχετίζεται με την ένταση του λαμβανόμενου σήματος. Χαρακτηριστικό είναι ότι PDR εξαρτάται άμεσα από το μέγεθος του πακέτου. Όσο μεγαλύτερα είναι τα πακέτα τόσο μειώνεται και το PDR. Η παρακάτω παράσταση αποτελεί μέρος των πειραμάτων της εργασίας [9] και δείχνει τον συσχετισμό του PDR με το μέγεθος των πακέτων για τρεις διαφορετικές συνδέσεις.



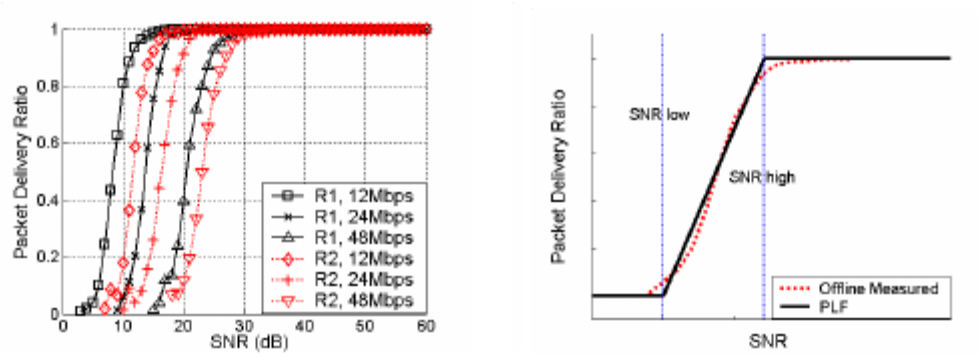
Εικόνα 5 Το PDR μειώνεται βάσει του μεγέθους των πακέτων

Signal to Noise Ratio (SNR)

Μια άλλη προσέγγιση είναι βάσει του λόγου της έντασης του εκπεμπόμενου σήματος δια τον Θόρυβο στο κανάλι (Signal to Noise Ratio ή SNR). Μεγαλύτερο SNR σημαίνει και καλύτερη ποιότητα σύνδεσης. Αυτή η τεχνική μας δείχνει επίσης τον τρόπο που η ένταση του ληφθέντος σήματος και ο θόρυβος επηρεάζει την σύνδεσή μας, οπότε στη συγκεκριμένη περίπτωση είναι καταλληλότερη. Ένα μειονέκτημα αυτής της προσέγγισης είναι ότι δοθείσης μιας συγκεκριμένης τιμής του SNR δεν μπορούμε να κάνουμε μια ακριβή πρόβλεψη για το PDR. Διαφορετικό hardware σημαίνει και διαφορετικές κεραίες με διαφορετική ευαισθησία, οπότε ο συσχετισμός του SNR και του PDR δεν είναι σαφής.

Ο συσχετισμός του SNR με το PDR εξαρτάται επίσης και από το bitrate που εκπέμπουμε τα δεδομένα. Αυτό όμως δεν θα μας απασχολήσει περαιτέρω καθώς θεωρούμε ότι το bitrate διατηρείται σταθερό.

Στην εργασία 'A Practical SNR-Guided Rate Adaptation' [10] μελετήθηκε η σχέση του SNR με το PDR. Οι παρακάτω γραφικές παραστάσεις αναλύουν πως μεταβάλλεται το PDR καθώς αυξάνεται το SNR.

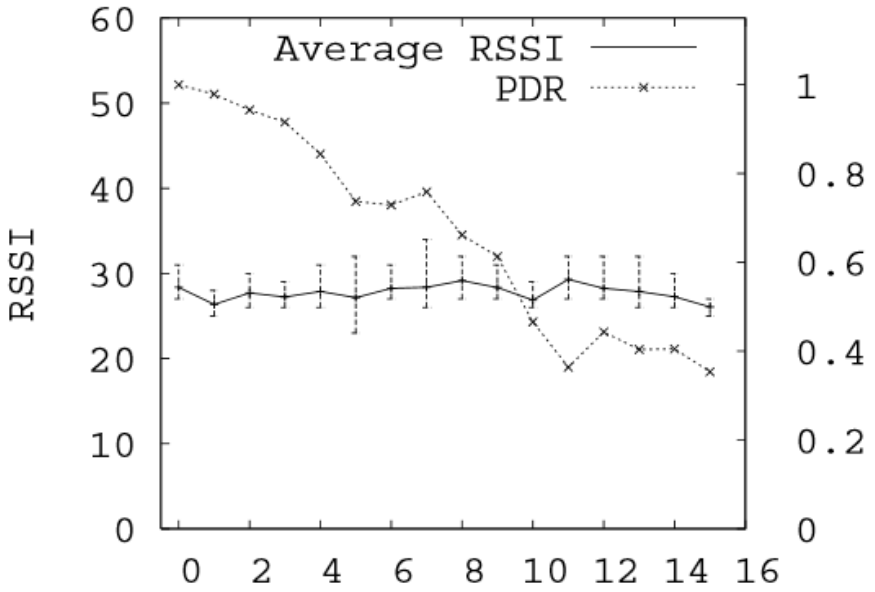


Εικόνα 6 Σχέση SNR - PDR

Από τις γραφικές παραστάσεις παρατηρούμε ότι για μικρές τιμές του SNR (μικρότερες από SNR_{low}) η σύνδεση έχει πλήρως χαθεί καθώς το PDR είναι σχεδόν μηδενικό. Για τιμές του SNR απο SNR_{low} μέχρι SNR_{high} , το PDR αυξάνεται γραμμικά ενώ για SNR μεγαλύτερο από SNR_{high} το PDR παραμένει σταθερό. Το πρόβλημα εδώ είναι ότι δεν γνωρίζουμε εκ των προτέρων τις τιμές του SNR_{low} και του SNR_{high}. Οι τιμές αυτές εκτός από το bitrate εξαρτώνται και από την κεραία.

Received Signal Strength Indicator (RSSI)

Τέλος εκτός από τις παραπάνω μεθόδους ο ίδιος ο driver μας παρέχει πληροφορίες για την ποιότητα της σύνδεσης μέσω του received signal strength indicator (RSSI). Το RSSI είναι ένας δείκτης για την ισχύ του σήματος που λαμβάνει η κεραία. Όσο μεγαλύτερη είναι η τιμή του RSSI τόσο ισχυρότερο είναι και το σήμα αυτό. Παρόλα αυτά δεν υπάρχει κάποιο πρότυπο για το RSSI ή το πως αυτός ο δείκτης υπολογίζεται. Είναι σύνηθες όμως το RSSI να αντιστοιχεί σε decibel χωρίς αυτό να είναι υποχρεωτικό. Άλλο μειονέκτημα του RSSI είναι ότι σε ένα περιβάλλον με μεταβαλλόμενο θόρυβο ή παρεμβολές το RSSI παραμένει σταθερό.



Εικόνα 7 RSSI σε περιβάλλων με παρεμβολές

Η παραπάνω παράσταση είναι από την εργασία "Assessing link quality in IEEE 802.11 Wireless Networks: Which is the right metric?" [9]. Από την γραφική παράσταση φαίνεται η πτώση του PDR λόγω παρεμβολών ενώ το RSSI παραμένει σταθερό.

Ο μηχανισμός που περιγράφεται παρακάτω χρησιμοποιεί σαν μετρική το RSSI το οποίο αντιστοιχεί σε decibel.

4.3 Αλλοίωση του σήματος

Ένα σήμα είναι επιρρεπές σε παράγοντες που το αλλοιώνουν και ενδεχομένως οδηγούν στην λανθασμένη λήψη του. Ένα σύνηθες φαινόμενο είναι να υπάρχει ενέργεια στο κανάλι όπου μεταδίδεται το σήμα η οποία προκαλεί και παραμόρφωσή του. Η ενέργεια αυτή χωρίζεται σε δύο κατηγορίες, τον θόρυβο (noise) και τις παρεμβολές (interference). Ο θόρυβος είτε υπάρχει φυσικά στο χώρο, είτε είναι ανεπιθύμητο αποτέλεσμα ανθρώπινης παρέμβασης. Από φυσικά αίτια παρατηρείται η ύπαρξη λευκού Γκαουσιανού θορύβου (white Gaussian

noise). Ο θόρυβος αυτός έχει ομοιόμορφα κατανεμημένη ενέργεια σε όλο το εύρος συχνοτήτων και ακολουθεί κανονική κατανομή με μηδενική μέση τιμή. Ο θόρυβος που προκαλείται από ανθρώπινη παρέμβαση δημιουργείται κυρίως από τις ηλεκτρικές συσκευές. Η κίνηση των ηλεκτρονίων εντός των συσκευών επιφέρει την εκπομπή ραδιοκυμάτων. Η κατανομή αυτού του θορύβου εξαρτάται από το είδος της συσκευής που τον προκάλεσε. Γενικά ο θόρυβος υποβιβάζει την σύνδεση χωρίς να προσφέρει οφέλη.

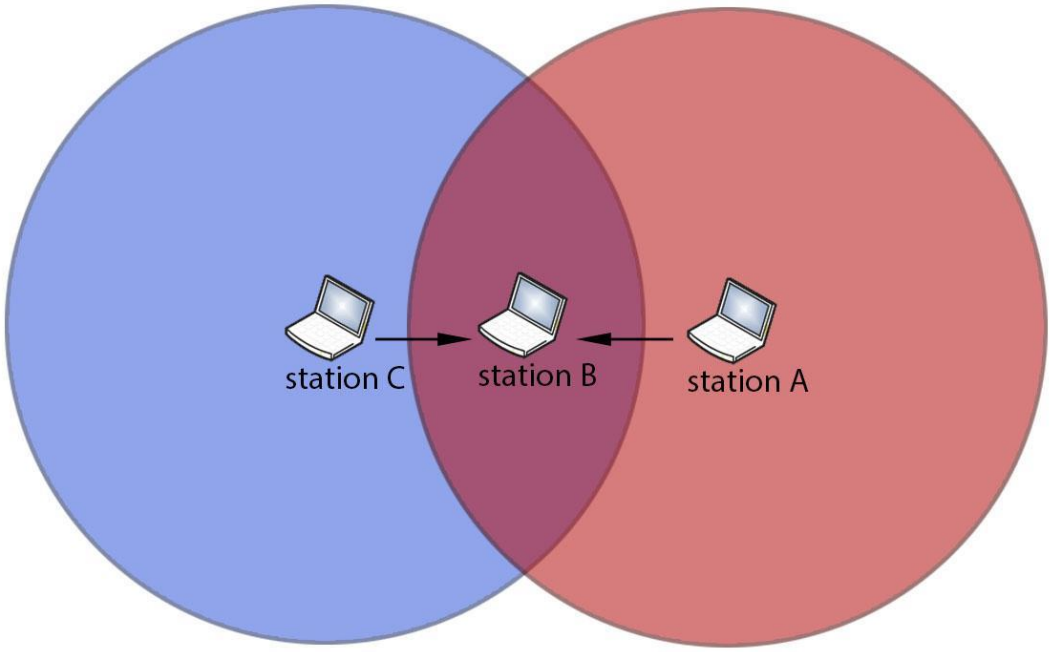
Αντίθετα από τον θόρυβο οι παρεμβολές προξενούνται όταν δυο συσκευές προσπαθούν να επικοινωνήσουν ταυτόχρονα πάνω στο ίδιο κανάλι (ή όταν τα κανάλια επικαλύπτονται). Τέτοιες συσκευές είναι είτε άλλοι σταθμοί που επικοινωνούν με κάποιο από τα πρωτόκολλα της οικογένειας 802.11 είτε τελείως διαφορετικές συσκευές όπως ασύρματα τηλέφωνα bluetooth κ.α.

Γενικά είναι σημαντικό να διαχωρίσουμε τον θόρυβο από τις παρεμβολές και να διαχειριστούμε το κάθε φαινόμενο με διαφορετικό τρόπο. Για παράδειγμα η αύξηση της έντασης εκπομπής σε ένα περιβάλλον με υψηλό θόρυβο θα αύξανε και το SNR και θα βελτίωνε την ποιότητα της σύνδεσης. Αντιθέτως σε ένα περιβάλλον με παρεμβολές κάτι τέτοιο δεν θα ήταν αποδοτικό. Αν αυξάναμε την ένταση στη μία σύνδεση η ποιότητα της θα βελτιώνονταν εις βάρος όμως της άλλης. Αν η ένταση εκπομπής αυξάνονταν παράλληλα και στις δύο συνδέσεις τότε θα αυξάνονταν και οι παρεμβολές και δεν θα υπήρχε ουσιαστικό όφελος. Στην περίπτωση των παρεμβολών οπότε, είναι πιο αποδοτικό να μειωθεί η ένταση εκπομπής ώστε να μειωθούν και οι παρεμβολές που δημιουργεί η μια σύνδεση στην άλλη.

Το ιδανικό βέβαια θα ήταν να μπορούσαμε να ανιχνεύσουμε αν κάποιος τρίτος μεταδίδει στο κανάλι και να αποφύγουμε την ταυτόχρονη μετάδοση. Κάτι τέτοιο θα εκμηδένιζε τελείως την ύπαρξη παρεμβολών. Δυστυχώς αυτό δεν είναι πάντα εφικτό. Το πρόβλημα του κρυμμένου τερματικού συνιστά ένα θεωρητικό σενάριο που περιγράφει μια απλή περίπτωση που η ανίχνευση των παρεμβολών δεν είναι δυνατή.

Το πρόβλημα του κρυμμένου τερματικού.

Το πρόβλημα του κρυμμένου τερματικού (hidden terminal problem) απαντάται πολλές φορές στις τηλεπικοινωνίες και τις ασύρματες συνδέσεις. Βάσει αυτού του προβλήματος ένας σταθμός προσπαθεί να επικοινωνήσει με κάποιον άλλον χωρίς όμως να γνωρίζει ότι ένας τρίτος παρεμβάλει στο κανάλι. Η παρακάτω εικόνα περιγράφει σχηματικά το πρόβλημα.



Εικόνα 8 hidden terminal problem

Στο παραπάνω σχήμα ο σταθμός C μεταδίδει στο κανάλι. Ο σταθμός A είναι εκτός εμβέλειας του C οπότε και δεν ανιχνεύει τη μετάδοση. Νομίζοντας ο A ότι το κανάλι είναι ελεύθερο ξεκινάει επικοινωνία με το σταθμό B. Στο σημείο που βρίσκεται όμως ο B τα σήματα του C και του A παρεμβάλλονται και δεν είναι δυνατή η αποκωδικοποίησή τους.

Η επίλυση αυτού του προβλήματος είναι σχετικά δύσκολη, πολλές λύσεις έχουν προταθεί χωρίς όμως καμία από αυτές να είναι πλήρως αποδοτική [11]. Η πιο απλή λύση είναι η μετακίνηση των σταθμών A και C. Αυτό όμως προϋποθέτει ότι το πρόβλημα έχει είδη ανιχνευτή ενώ πολλές φορές η μετακίνηση των σταθμών δεν είναι εφικτή. Μια άλλη λύση θα ήταν οι σταθμοί A και C να αυξήσουν την

ένταση εκπομπής τους. Στην περίπτωση αυτή είναι πιθανόν να μπορέσουν να εντοπίσουν ο ένας τα σήματα του άλλου. Αυτό όμως δεν είναι απόλυτο και αν παραμείνουν εκτός εμβέλειας οι παρεμβολές τους θα αυξηθούν. Σημαντική είναι επίσης η αποφυγή κατευθυντικών κεραιών καθώς οι σταθμοί που δεν βρίσκονται στην κατεύθυνση της κεραίας δεν μπορούν να ανιχνεύσουν τις μεταδόσεις της. Τέλος διάφορα πρωτόκολλα με βάσει σκυτάλη (token) ή αιτήσεις μεταδόσεων (polling) έχουν προταθεί τα οποία μπορούν να επιλύσουν το πρόβλημα. Ως αποτέλεσμα όμως τέτοια πρωτόκολλα αυξάνουν την καθυστέρηση (latency) και μειώνουν το throughput.

Στα πειράματα μας προσπαθήσαμε να έχουμε όσο το δυνατόν χαμηλότερο noise και να αποφύγουμε οποιεσδήποτε παρεμβολές.

4.4 Εξασθένιση του σήματος

Κατά την μετάδοση ενός σήματος σε ένα κανάλι ο δέκτης λαμβάνει το σήμα με χαμηλότερη ισχύ από αυτήν που το εξέπεμψε ο αποστολέας. Μια πληθώρα παραγόντων επηρεάζουν την ισχύ του σήματος και το εξασθενούν. Οι πιο σημαντικοί παράγοντες είναι η Εξασθένηση ελεύθερου χώρου (free space path loss), η ανάκλαση (reflection) και η περίθλαση (diffraction) καθώς και η απορρόφηση ενέργειας από το περιβάλλον.

Η Εξασθένηση ελεύθερου χώρου έγκειται στο ότι καθώς το σήμα εξαπλώνεται, η ηλεκτρομαγνητική του ενέργεια διαμοιράζεται στον χώρο. Έτσι η πυκνότητα ισχύος του σήματος μειώνεται και το σήμα εξασθενεί.

Η εξίσωση του Friis περιγράφει την Εξασθένηση ελεύθερου χώρου μεταξύ ισοτροπικών κεραιών η οποία είναι η ακόλουθη:

$$L = 20log_{10}(\frac{4\pi d}{\lambda})$$

Όπου το *'d'* είναι η απόσταση μεταξύ των κεραιών και το *'λ'* εκφράζει το μήκος κύματος του σήματος. Εδώ είναι σημαντικό να επισημάνουμε ότι από τον τύπο του Friis γίνεται σαφές ότι η ισχύς εκπομπής δεν επηρεάζει την τιμή της εξασθένησης ελεύθερου χώρου.

Τα φαινόμενα της ανάκλασης και της περίθλασης που επίσης αναφέρθηκαν συμβαίνουν κυρίως όταν το σήμα συναντά κάποιο εμπόδιο. Δυστυχώς είναι πολύ δύσκολο να προβλέψουμε τη διαρρύθμιση του χώρου και τα εμπόδια που μπορεί

να συναντήσει ένα σήμα. Λόγω ανάκλασης ένα σήμα μπορεί να ληφθεί από διαφορετικές διαδρομές με διαφορετική καθυστέρηση. Η μείωση της έντασης εκπομπής μπορεί να αποκλείσει μερικές από της διαδρομές αυτές και ως αποτέλεσμα να αυξήσει την καθυστέρηση.

Για τους παραπάνω λόγους δεν λάβαμε υπόψη μας αυτούς τους παράγοντες αλλά διεξάγαμε τα πειράματά μας σε ανοικτό χώρο.

4.5 Ισχύς εκπομπής

Μεγάλο ενδιαφέρον παρουσιάζει η ισχύς του σήματος κατά την λήψη. Η ισχύς αυτή καθορίζει αν το σήμα μπορεί να αποκωδικοποιηθεί ή όχι. Στην περίπτωση που αυτή είναι πολύ χαμηλή, η αποκωδικοποίηση του σήματος δεν είναι εγγυημένη. Έστω ότι υπάρχει ένα ελάχιστο όριο για την ισχύ που τιμές μεγαλύτερες αυτού μπορούν να εγγυηθούν την αποκωδικοποίηση του σήματος. Ας συμβολίσουμε το όριο αυτό με P_{thr}. Θα θέλαμε στα πλαίσια που αυτό είναι εφικτό, το σήμα να λαμβάνεται με ισχύ το λιγότερο P_{thr}. Έτσι θα προσπαθήσουμε να θέσουμε την ισχύ εκπομπής με τέτοιο τρόπο ώστε η ισχύς λήψης να είναι μεγαλύτερη του P_{thr} αλλά όσο γίνεται πιο κοντά σε αυτό. Σε αυτήν την περίπτωση δαπανάται η λιγότερη δυνατή ενέργεια ώστε να είναι εγγυημένη η σύνδεση.

Θα ξεκινήσουμε υπολογίζοντας το path loss το οποίο μπορεί να υπολογιστεί ως η διαφορά της ισχύος εκπομπής μείον την ισχύ λήψης.

$$Pathloss = P_{tx} - P_{rx} \quad (1)$$

Οπού P_{tx} είναι η ισχύς λήψης και P_{rx} η ισχύς εκπομπής.

Αφού επιθυμούμε η λήψη να γίνεται το λιγότερο με P_{thr} , η ελάχιστη ισχύς εκπομπής είναι:

$$P_{txmin} = PathLoss + P_{thr} \quad (2)$$

Το P_{txmin} όμως δεν είναι ανεκτικό σε κανενός είδους διακυμάνσεις της ισχύος λήψεως. Τέτοιες διακυμάνσεις θα οδηγούσαν σε συνεχή σφάλματα, αύξηση του bit error rate και επαναϋπολογισμό του $PathLoss$. Για να αποφύγουμε αυτά τα προβλήματα θα αυξήσουμε την ισχύ εκπομπής κατά μια σταθερή τιμή (P_c). Έτσι διακύμανση μικρότερη από P_c δεν θα επηρεάσει την σύνδεση.

$$P_{tx} = PathLoss + P_{thr} + P_c \quad (3)$$

Ο κάθε σταθμός με χρήση του παραπάνω τύπου υπολογίζει το πόσο πρέπει να εκπέμψει ώστε να μειώσει την ενέργεια του και να εγγυάται την επικοινωνία. Στην περίπτωση όμως που ο σταθμός βρίσκεται αρκετά μακριά από το access point, τότε η τιμή του P_{tx} μπορεί να είναι πολύ μεγάλη και να μην είναι εφικτή. Τότε ο σταθμός εκπέμπει με το μέγιστο δυνατό που υποστηρίζει το hardware του. Το access point από την άλλη εγγυάται την επικοινωνία με όλους τους συνδεδεμένους σε αυτό σταθμούς. Καθώς όμως οι σταθμοί βρίσκονται σε διαφορετική απόσταση από το access point, ο κάθε σταθμός έχει διαφορετικό path loss. Έτσι το access point θέτει την ισχύ εκπομπής βάσει του τύπου (3) αλλά στην θέση του path loss εισάγεται η μεγαλύτερη τιμή που έχει παρατηρηθεί. Με αυτόν τον τρόπο το access point εγγυάται την σύνδεση με τον πιο απομακρυσμένο συνδεδεμένο σταθμό.

4.6 Ανταλλαγή μηνυμάτων

Στην παραπάνω παράγραφο είδαμε πως υπολογίζεται η επιθυμητή ισχύς εκπομπής στην περίπτωση που είναι γνωστό το path loss. Οι σταθμοί όμως δεν γνωρίζουν το path loss εκ των προτέρων και αναλαμβάνουν να το υπολογίσουν. Για τον υπολογισμό του είναι αναγκαία η τιμή της ισχύος εκπομπής (P_{tx}) καθώς και η τιμή της ισχύος κατά την λήψη (P_{rx} ή RSSI). Αν όμως ένας σταθμός αποστείλει ένα πακέτο, ο ίδιος ο σταθμός δεν γνωρίζει το RSSI του πακέτου που έστειλε. Επομένως ένας μηχανισμός ανταλλαγής μηνυμάτων (feedback messages mechanism) είναι απαραίτητος ώστε ο κάθε αποστολέας να λαμβάνει γνώση του RSSI των πακέτων που αποστέλλει. Η τιμή του RSSI όμως είναι ευμετάβλητη. Περιβαλλοντικοί παράγοντες μπορούν να επηρεάσουν το RSSI ενός συγκεκριμένου πακέτου χωρίς να επηρεάσουν τις μετέπειτα εκπομπές. Τέτοιοι παράγοντες πρέπει να απομονωθούν, ειδάλλως ο μηχανισμός καταλήγει στην συνεχή αποστολή μηνυμάτων και συνεχή αυξομείωση της ισχύος εκπομπής αλλά και σε προβλήματα απόδοσης. Για τον λόγο αυτό αντί να αποστέλλεται η τελευταία τιμή του RSSI ο μηχανισμός υπολογίζει έναν εκθετικό κινητό μέσο όρο (Exponential Weighted Moving Average - EWMA). Ο κινητός αυτός μέσος όρος πολλαπλασιάζει την κάθε τιμή με ένα βάρος. Οι πιο πρόσφατες τιμές έχουν μεγαλύτερο βάρος ενώ τα βάρη μειώνονται εκθετικά. Ένα θετικό στοιχείο είναι ότι ο νέος μέσος όρος μπορεί να υπολογιστεί αναδρομικά βάσει της προηγούμενης

μέσης τιμής μειώνοντας τον αριθμό των πράξεων. Ο παρακάτω τύπος υπολογίζει τον εκθετικό κινητό μέσο όρο αναδρομικά [12].

$$S_1 = Y_1$$
$$Για\ t > 1\ , \qquad S_t\ =\ a \cdot Y_t + (1-a) \cdot S_{t-1}\ \ , \qquad 0 < a\ < 1$$

Το S_t αποτελεί τον κινητό μέσο όρο την χρονική στιγμή t.

Y_t είναι η νέα τιμή που εισέρχεται την στιγμή t και a είναι μια σταθερά εξομάλυνσης.

Η χρήση του EWMA εξασφαλίζει πιο αντιπροσωπευτικές τιμές για το RSSI (οπότε και για το path loss) και πιο ομαλές γραφικές παραστάσεις. Επίσης αξίζει να σημειωθεί ότι εάν δεν υπάρξει μια σημαντική μεταβολή στην μέση τιμή του RSSI, ο σταθμός αποφεύγει να στείλει κάποιο feedback μήνυμα και ο αριθμός των εναλλασσόμενων μηνυμάτων μειώνεται δραματικά.

Ο παρακάτω ψευδοκώδικας περιγράφει την συμπεριφορά ενός σταθμού όταν λάβει ένα πακέτο.

```
message_received(packet)
{
   rssi=extract_rssi(packet);
   rssi_avg=average(rssi,mac)
   previous_rssi=get_previous_rssi(packet.mac)

   if( abs(rssi_avg-previous_rssi) > MAX_CHANGE )
   {
      send_rssi_message(rssi_avg,mac)
      set_previous_rssi(packet.mac,rssi_avg)
   }
}
```

Όταν ένας σταθμός λάβει ένα feedback μήνυμα με πληροφορία για το RSSI τρέχει κώδικα ανάλογο του παρακάτω.

```
get_rssi_information(rssi)
{
   Path_loss=calculate_path_loss(rssi).
   Ptx=calculate_ptx(path_loss)
   set_transmition_power(Ptx)
}
```

Με παρόμοιο τρόπο λειτουργεί και το access point. Στην περίπτωση όμως που το access point λάβει κάποιο feedback με πληροφορία για το RSSI ελέγχει μια λίστα με το RSSI από όλους τους συνδεδεμένους σταθμούς. Ο πιο μακρινός σταθμός (αυτός με το μικρότερο RSSI) είναι αυτός που καθορίζει και την ισχύ εκπομπής του access point.

Ένα άλλο πρόβλημα αποτελεί η περίπτωση που η σύνδεση μείνει ανενεργή για μεγάλο χρονικό διάστημα. Τότε ένας σταθμός μπορεί να έχει απομακρυνθεί εκτός εμβέλειας του access point. Μια λύση αυτού του προβλήματος είναι η αποστολή περιοδικών hello μηνυμάτων. Η συχνότητα των hello μηνυμάτων καθορίζει και την μέγιστη ταχύτητα που μπορούν να κινηθούν οι σταθμοί χωρίς να υποβιβαστεί η σύνδεση. Η αποστολή των hello μηνυμάτων όμως αυξάνει τη κατανάλωση ενέργειας, και αυτό κάνει τον μηχανισμό μας πιο αποδοτικό σε συνδέσεις με σχετικά υψηλό φόρτο.

Κεφάλαιο 5

Υλοποίηση του μηχανισμού

5.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται ο τρόπος λειτουργίας του μηχανισμού και η υλοποίησή του.

Ο μηχανισμός υλοποιήθηκε σε περιβάλλον linux και χωρίζεται σε δύο μέρη. Ένα μέρος υλοποιήθηκε εσωτερικά στο πρόγραμμα οδηγού της ασύρματης κάρτας δικτύου και ένα μέρος υλοποιήθηκε σαν ξεχωριστή εφαρμογή. Γενικά η υλοποίηση σε επίπεδο εφαρμογής στο χώρο διευθύνσεων του χρήστη (user space) είναι πιο εύκολα διαχειρίσιμη και παρέχει μεγαλύτερη ευελιξία. Κατά συνέπεια ο κυρίως μηχανισμός υλοποιήθηκε σε επίπεδο εφαρμογής και το πρόγραμμα οδηγού τροποποιήθηκε ούτως ώστε να εξάγει τις απαραίτητες πληροφορίες στην εφαρμογή αυτή. Τέτοιες πληροφορίες αποτελούν το RSSI του κάθε πακέτου και η διεύθυνση mac του αποστολέα. ΟΙ πληροφορίες αυτές εξάγονται σε ένα εικονικό αρχείο στον κατάλογο proc. Η εφαρμογή διαβάζει αυτές της πληροφορίες και υπολογίζει τον εκθετικό κινητό μέσο όρο του RSSI για κάθε σταθμό όπως αναλύθηκε στο κεφάλαιο 4. Αν παρατηρηθεί σημαντική μεταβολή στο RSSI τότε στέλνονται τα αντίστοιχα feedback μηνύματα μέσω UDP πρωτοκόλλου. Όταν η εφαρμογή λάβει ένα μήνυμα που περιέχει το RSSI τότε υπολογίζει εκ νέου το path loss και την προτεινόμενη ισχύ εκπομπής. Τέλος η ισχύς εκπομπής τίθεται μέσω εντολών συστήματος.

5.2 Το λειτουργικό σύστημα GNU Linux

Ιστορική αναδρομή

Ένα από τα πιο διαδεδομένα λειτουργικά συστήματα στις μέρες μας είναι το λειτουργικό σύστημα gnu-linux ή απλώς linux. Ιστορικά το linux ξεκίνησε ως ένα ελευθέρου λογισμικού αντίγραφο του unix. Πρώτος ο Richard Stallman

διαμόρφωσε την ιδέα του ελεύθερου λογισμικού το οποίο όρισε ως το λογισμικό που σέβεται τις ελευθερίες του χρήστη και της κοινωνίας γενικότερα. Στη συνέχεια έθεσε τις τέσσερις βασικές αρχές που πρέπει να υπακούει το λογισμικό ώστε να μπορεί να θεωρηθεί ελεύθερο [13], οι οποίες περιγράφονται παρακάτω.

0. Ο καθένας είναι ελεύθερος να χρησιμοποιήσει το λογισμικό για όποιο σκοπό επιθυμεί.
1. Ο καθένας είναι ελεύθερος να μελετήσει τον τρόπο λειτουργίας ενός λογισμικού και να το τροποποιήσει ώστε να καλύπτει τις ανάγκες του (η αρχή αυτή επιτρέπει την πρόσβαση στον πηγαίο κώδικα του λογισμικού).
2. Ο καθένας είναι ελεύθερος να αναδιανείμει αντίγραφα του λογισμικού ώστε να βοηθήσει τους γύρω του.
3. Ο καθένας είναι ελεύθερος να διανείμει τροποποιημένες εκδόσεις ενός λογισμικού ώστε αυτές οι τροποποιήσεις να ωφελήσουν όλη την κοινωνία.

Το 1983 ο Stallman ανακοίνωσε την ίδρυση του GNU project με σκοπό την υλοποίηση ενός λειτουργικού συστήματος βασισμένο σε αυτές τις αρχές. Το GNU project ανέπτυξε μια πληθώρα εργαλείων (όπως ο μεταφραστής gcc κ.α.) αλλά η υλοποίηση του πυρήνα του λειτουργικού παρουσίαζε μεγάλη καθυστέρηση. Τελικά το 1992 ο Linus Torvalds ήταν αυτός που δημιούργησε των πυρήνα και έδωσε στο νέο αυτό λειτουργικό σύστημα το όνομά του.

Χρήσεις του Linux

Οι αρχές 2 και 3 του ελεύθερου λογισμικού επιτρέπουν την διανομή του χωρίς περιορισμούς.

Ως αποτέλεσμα μια πληθώρα από εταιρίες και οργανισμούς αναπτύσσουν τις δικές τους διανομές όπου η κάθε μια έχει τα δικά της χαρακτηριστικά. Το γεγονός αυτό οδηγεί σε μια πολυμορφία του λειτουργικού συστήματος και το καθιστά κατάλληλο για διαφορετικές χρήσεις.

server συστήματα

Το Linux έχει σημαντικό μερίδιο στην αγορά server συστημάτων καθώς είναι πιο οικονομικό από άλλα λειτουργικά συστήματα. Επίσης το γεγονός ότι είναι ανοιχτού κώδικα αυξάνει την εμπιστοσύνη στο linux σε θέματα ασφάλειας.

Linux ενσωματωμένο σε συσκευές

Καθώς το linux είναι αρκετά παραμετροποιήσιμο είναι εύκολο να δημιουργηθούν εκδόσεις του κατάλληλες να ενσωματωθούν σε συσκευές. Τέτοιες συσκευές αποτελούν οι έξυπνες τηλεοράσεις, συσκευές αισθητήρων κ.α. Μεγάλο όμως ενδιαφέρον παρουσιάζει το γεγονός ότι το Linux ενσωματώνεται και σε πολλές συσκευές access point.

Linux σε κινητές συσκευές

Ραγδαία ανάπτυξη παρουσιάζει επίσης το linux και σε κινητές συσκευές όπως κινητά τηλέφωνα και tablets. Πολλές εταιρίες βγάζουν τα δικά τους λειτουργικά συστήματα για τέτοιες συσκευές και τα περισσότερα από αυτά είναι βασισμένα σε Linux όπως για παράδειγμα το *android* της google, το *ubuntu phone* της canonical, το *firefox os* της mozila κ.α.

5.3 Θέματα πυρήνα

Ο πυρήνας του Linux ανήκει στην κατηγορία των μονολιθικών πυρήνων.

Σε μία μονολιθική αρχιτεκτονική, οι βασικές λειτουργίες του λειτουργικού συστήματος υλοποιούνται σε επίπεδο πυρήνα και διαμοιράζονται τον ίδιο χώρο διευθύνσεων. Αντίθετα σε αρχιτεκτονική μικροπυρήνων ο πυρήνας του λειτουργικού υλοποιεί μόνο πολύ βασικές λειτουργίες όπως την επικοινωνία μεταξύ διεργασιών ενώ οι υπόλοιπες λειτουργίες υλοποιούνται σε επίπεδο ξεχωριστών εξυπηρετητών (services). Σε αυτήν την αρχιτεκτονική το κάθε service έχει τον δικό του χώρο διευθύνσεων.

Γενικά ένας μονολιθικός πυρήνας είναι πιο εύκολος στην υλοποίηση του, καταλαμβάνει λιγότερο χώρο και σε επίπεδο πηγαίου αλλά και τελικού κώδικα ενώ μπορεί να χαρακτηριστεί πιο αποδοτικός καθώς δεν σπαταλάει πόρους για την επικοινωνία των εξυπηρετητών μεταξύ τους. Αντιθέτως ένας μεγάλος μονολιθικός πυρήνας είναι πιο δύσκολος στην συντήρηση του και ένα οποιοδήποτε πρόβλημα (bug) σε ένα κομμάτι του δημιουργεί αστάθεια σε όλο το σύστημα.

Από την άλλη οι μικροπυρήνες συντηρούνται πιο εύκολα και ένα bug σε κάποιον εξυπηρετητή δεν επηρεάζει την απόδοση στους υπόλοιπους.

5.4 Kernel modules

Επειδή όπως ήδη έχουμε αναφέρει το Linux ακολουθεί μονολιθική αρχιτεκτονική, οποιαδήποτε λειτουργία επιθυμούμε να προσθέσουμε στον πυρήνα δεν μπορούμε να την προσθέσουμε με κάποιον εξυπηρετητή. Για τον λόγο αυτόν φορτώνουμε κάποιο module. Τα modules αποτελούν κομμάτια κώδικα που μεταγλωττίζονται ξεχωριστά και φορτώνονται στον πυρήνα κατ' απαίτηση (on demand) κατά την διάρκεια λειτουργίας του λειτουργικού και χωρίς να απαιτείται επανεκκίνηση του συστήματος. Ένα module είναι άμεσα συνδεδεμένο με μια συγκεκριμένη έκδοση του πυρήνα. Αυτό έχει ως αποτέλεσμα τα modules που επιθυμούμε να φορτωθούν σε άλλες εκδόσεις του πυρήνα να απαιτείται να μεταγλωττιστούν ξανά. Τέλος τα modules και ο πυρήνας διαμοιράζονται τον ίδιο χώρο διευθύνσεων.

5.5 Προγράμματα οδήγησης (drivers)

Κάθε περιφερειακή συσκευή χρειάζεται ένα πρόγραμμα οδήγησης που να επιτρέπει την αναγνώρισή της και την επικοινωνία της με το υπόλοιπο σύστημα. Τέτοια προγράμματα οδηγών φορτώνονται στον πυρήνα ως modules. Έτσι ο πυρήνας μπορεί να κρατηθεί σχετικά μικρός φορτώνοντας μόνο τους οδηγούς για τις συσκευές που είναι διαθέσιμες. Ένα πρόβλημα αποτελεί το γεγονός ότι για κάθε έκδοση του πυρήνα τα modules πρέπει να μεταγλωττιστούν ξανά. Πολλές εταιρείες δεν δημοσιοποιούν τον πηγαίο κώδικα από τους οδηγούς τους οπότε η επαναμεταγλώττισή τους δεν είναι εφικτή. Άλλες φορές μικρές αλλαγές στον πυρήνα μπορούν να οδηγήσουν σε εκτεταμένες τροποποιήσεις στο πρόγραμμα οδήγησης. Για την επίλυση αυτού του προβλήματος τα προγράμματα οδήγησης διαχωρίζονται περαιτέρω σε δύο υπομέρη, το module και το firmware. Το firmware υλοποιεί τις βασικές λειτουργίες του οδηγού ενώ το module φορτώνεται στον πυρήνα και αναλαμβάνει την επικοινωνία με το firmware. Για κάθε νέα έκδοση του πυρήνα το module μπορεί να τροποποιηθεί και να επαναμεταγλωττιστεί ενώ το firmware παραμένει αμετάβλητο. Σε αυτό το σημείο πρέπει να αναφέρουμε ότι ο όρος firmware δεν είναι δόκιμος καθώς το firmware μπορεί να εκτελεστεί στον κεντρικό επεξεργαστή του συστήματος και όχι απαραίτητα στον μικροεπεξεργαστή του περιφερειακού.

5.5 Blocking calls

Στο Linux ,για τις γλώσσες προγραμματισμού c και c++ οι κύριες συναρτήσεις για ανάγνωση και εγγραφή αρχείων είναι οι *'read'* και *'write'*. Οι συναρτήσεις αυτές εσωτερικά επικοινωνούν με τον πυρήνα μέσω των αντίστοιχων κλήσεων συστήματος. Έτσι ο πυρήνας είναι αυτός που τελικά αναλαμβάνει να επικοινωνήσει με το hardware και να ανακτήσει ή να αποθηκεύσει την πληροφορία.

Οι κλήσεις συστήματος για τις εντολές read και write αποτελούν blocking call. Αυτό σημαίνει ότι η εκτέλεση της διεργασίας (ή του thread) που τις κάλεσε μπορεί να ανασταλεί έως ότου τα ζητούμενα δεδομένα να είναι διαθέσιμα. Σε αυτήν την περίπτωση κάποια άλλη διεργασία θα εκτελεστεί στη θέση της.

Ένα χαρακτηριστικό παράδειγμα είναι όταν μια διεργασία θέλει να διαβάσει δεδομένα από ένα αρχείο στον σκληρό δίσκο. Σε αυτήν την περίπτωση η διεργασία θα καλέσει την αντίστοιχη κλήση συστήματος της εντολής read. Ο πυρήνας θα αναστείλει την εκτέλεση της διεργασίας και θα επικοινωνήσει με τον δίσκο ζητώντας του τα δεδομένα. Όταν ο δίσκος θα έχει πλέον ανακτήσει τα δεδομένα, ο πυρήνας θα τα μεταφέρει στην κύρια μνήμη και η εκτέλεση της διεργασίας που τα ζήτησε θα συνεχιστεί.

5.6 Εικονικά αρχεία

Στο παράδειγμα της παραπάνω παραγράφου αναφερθήκαμε σε ένα αρχείο στον σκληρό δίσκο. Στο linux όμως όταν αναφερόμαστε σε ένα αρχείο δεν συνεπάγεται αυτόματα ότι το αρχείο αυτό βρίσκεται αποθηκευμένο σε κάποιο μόνιμο μέσο αποθήκευσης (όπως π. χ. ο δίσκος). Το λειτουργικό χρησιμοποιεί μια πληθώρα εικονικών αρχείων για να διεκπεραιώσει διάφορες λειτουργίες του. Ένα χαρακτηριστικό παράδειγμα είναι ότι κάθε περιφερειακή συσκευή συνδεδεμένη στον υπολογιστή αναπαρίσταται ως ένα εικονικό αρχείο. Η ανάγνωση και η εγγραφή σε αυτά τα αρχεία συσκευών αποτελεί ένα τρόπο άμεσης επικοινωνίας με το hardware. Γενικά τα εικονικά αρχεία υποστηρίζουν πλήρως την διεπαφή των συνηθισμένων αρχείων. Αντί όμως η εγγραφή τους να αποθηκεύει δεδομένα συνήθως χρησιμοποιείται για να διαβιβαστούν εντολές στον πυρήνα του λειτουργικού. Αντίστοιχα η ανάγνωση από τα εικονικά αρχεία χρησιμοποιείται για να μεταβιβαστούν πληροφορίες από τον πυρήνα σε εφαρμογές του χρήστη. Ένα

τέτοιο σύστημα εικονικών αρχείων αποτελεί και το proc filesystem [14]. Το proc filesystem προτάθηκε το 1984 για το unix ως ένα σύστημα όπου κάθε διεργασία θα αναπαρίσταται ως ένα εικονικό αρχείο. Η ανάγνωση από αυτά τα αρχεία θα επιτρέπει την ανάκτηση πληροφοριών για τη συγκεκριμένη διεργασία. Από τότε το proc filesystem έγινε ένα σημαντικό χαρακτηριστικό του λειτουργικού συστήματος. Πλέον εκτός από τα αρχεία των διεργασιών στο proc filesystem έχει προστεθεί και μια πληθώρα άλλων εικονικών αρχείων που επιτρέπουν την ανάγνωση γενικών πληροφοριών. Τέτοιες πληροφορίες είναι ο τύπος του επεξεργαστή, η κατάσταση του δικτύου κ.α. Επίσης για τη διευκόλυνση των προγραμματιστών ο πυρήνας του λειτουργικού περιέχει κατάλληλες συναρτήσεις για την εύκολη δημιουργία νέων αρχείων σε αυτό το filesystem.

5.7 Υλοποίηση σε επίπεδο driver

Η ανάκτηση του RSSI καθώς και η ρύθμιση της έντασης εκπομπής δεν υποστηρίζεται από όλες τις κάρτες δικτύου και όλους τους διαθέσιμους drivers. Για την υλοποίηση του μηχανισμού που παρουσιάζεται χρησιμοποιήθηκαν κάρτες δικτύου της εταιρίας Cisco με chipset atheros.

Πιο συγκεκριμένα οι κάρτες αυτές είναι οι 'Cisco Aironet cb21ag carbus' [15].

Όσον αφορά το πρόγραμμα οδηγού φορτώθηκε ο driver ath5k από το σύνολο οδηγών 'compat wireless' [16]. Ο οδηγός αυτός είναι ανοιχτού κώδικα και ήταν εύκολη η τροποποίηση του. Επίσης υποστήριζε την εξαγωγή του RSSI καθώς και την προσαρμογή της έντασης εκπομπής από τα 5dBm έως τα 20 dBm. Εν τούτοις η ανάγνωση του RSSI αντιμετώπιζε αρκετά προβλήματα. Χαρακτηριστικό είναι ότι ενώ ο driver υπολόγιζε το RSSI για κάθε πακέτο ξεχωριστά, οι τιμές αυτές δεν ήταν διαθέσιμες σε εξωτερικές εφαρμογές. Αντιθέτως ο driver εξήγαγε έναν μέσο όρο του RSSI από τα τελευταία πακέτα που ελήφθησαν. Το πρόβλημα αυτό γινόταν ακόμα μεγαλύτερο όταν λαμβάνονταν πακέτα από περισσότερους από έναν σταθμούς. Σε αυτήν την περίπτωση το RSSI δεν είχε αντιπροσωπευτική τιμή για κανέναν σταθμό.

Ένα άλλο πρόβλημα συνιστούσε το γεγονός ότι θα επιθυμούσαμε να γνωρίζουμε πότε λήφθηκε ένα πακέτο και να διαβάζαμε το RSSI την συγκεκριμένη χρονική στιγμή. Έτσι θα αποφεύγαμε να ελέγχουμε περιοδικά για

νέες τιμές, κάτι που θα οδηγούσε σε σπατάλη πόρων. Κάτι τέτοιο όμως δεν ήταν άμεσα εφικτό.

Για την επίλυση των παραπάνω προβλημάτων επιλέχτηκε η τροποποίηση του driver. Μέτα την τροποποίηση ο driver δημιουργεί πλέον ένα εικονικό αρχείο στο proc filesystem. Η ανάγνωση από αυτό το αρχείο αποτελεί ένα blocking call όπως περιγράφηκε παραπάνω. Μια εφαρμογή (ή ένα thread) που επιθυμεί να διαβάσει δεδομένα σταματάει την εκτέλεση της μέχρι την έλευση ενός νέου πακέτου. Μετά την λήψη κάποιου πακέτου, ο driver θα επιστρέψει στην εφαρμογή την τιμή του RSSI για το ληφθέν πακέτο καθώς και την διεύθυνση mac του αποστολέα. Με τον τρόπο αυτόν μια εφαρμογή μπορεί να γνωρίζει την χρονική στιγμή που ελήφθη ένα πακέτο, την ακριβή τιμή του RSSI του συγκεκριμένου πακέτου αλλά και τον αποστολέα του. Έτσι γίνεται δυνατή η υλοποίηση του κυρίως μηχανισμού εκτός του driver.

Θέματα κώδικα

Κατά την έλευση ενός πακέτου οι τιμές του RSSI και η διεύθυνση mac αποθηκεύονται σε μια συνδεδεμένη λίστα (rssi_list). Όταν μια διεργασία επιθυμεί να αναγνώσει πληροφορίες από το εικονικό αρχείο επιστρέφεται πάντα η πρώτη τιμή της λίστας. Αν η λίστα είναι κενή η διεργασία περιμένει σε μια ουρά αναμονής (waitqueue).

```
struct rssi_list
{
     s8 rssi;
    u8 mac[6];
    struct rssi_list *next;
};

struct rssi_list *head;
struct rssi_list *tail;
```

Για την δημιουργία του εικονικού αρχείου αρχικά ορίζουμε τις συναρτήσεις που αναλαμβάνουν να διεκπεραιώσουν το άνοιγμα και το κλείσιμο του αρχείου καθώς και την ανάγνωση και την έγγραφή σε αυτό. Οι συναρτήσεις αυτές είναι οι *proc_open*, *proc_close*, *proc_read* και *proc_write* αντίστοιχα.

```
static struct file_operations file_op =
{
      .read = proc_read,  /* "read" from the file */
      .write = proc_write,
      .open = proc_open, /* called when the /proc file is opened */
      .release = proc_close,          /* called when it's closed */
};
```

```
static int proc_open(struct inode *inode, struct file *file)
 {
      try_module_get(THIS_MODULE);
      opened_files++;
      file->private_data=0;
      printk("RSSI FILE OPENED\n");
      return 0;
 }
```

```
static int proc_close(struct inode *inode, struct file *file)
{
   module_put(THIS_MODULE);
   printk("RSSI FILE CLOSED\n");
   opened_files--;
   if(opened_files==0)
   {
       clean_list();
    }
    return 0;
 }
```

```
static ssize_t proc_write(struct file *file, const char *buf,size_t length,loff_t * offset)
{
   printk("writting is not supported\n");
   return 0;
}
```

```
static ssize_t proc_read(struct file *file, char *buf,size_t len,loff_t * offset)
{
   if ((file->f_flags & O_NONBLOCK) )
   {
      return -EAGAIN;
   }
   while(head==0)
  {
     wait_event_interruptible(WaitQ, head);
  }

  char message[32];
  memset (message,'\0',32);
  int k;
  if(head->rssi>999 || head->rssi<-99)
  {
      k=-1;
   }
   else
   {
       k=head->rssi;
   }
   int err=copy_to_user(buf,message,32);
   if(err!=0)
   {
       printk("ERROR copping to users %d\n",err);
       return 0;
    }
   struct rssi_list *l=head;
   head=head->next;
   kfree(l);
   return k;
}
```

Η συνάρτηση proc_read επιστρέφει τις ζητούμενες τιμές στην εφαρμογή. Αν η rssi_list είναι κενή ο κώδικας σταματάει την εκτέλεση του κατά το wait_event_interruptible περιμένοντας κάποιο interrupt για να συνεχίσει. Με αυτόν τον τρόπο υλοποιείται το blocking call που αναφέρθηκε.

Το εικονικό αρχείο δημιουργείται κατά το φόρτωμα του προγράμματος οδηγού από τη συνάρτηση *txpower_init.*

```
int txpower_init(void)
{
   printk("txpower init");
   proc_File = create_proc_entry(PROC_ENTRY_FILENAME, 0444, NULL);
   opened_files=0;
   head=0;
   tail=0;
   if (proc_File == NULL)
   {
      remove_proc_entry(PROC_ENTRY_FILENAME, proc_File);
      printk(KERN_ALERT "Error: Could not initialize %s\n",PROC_ENTRY_FILENAME);
      return ENOMEM;
   }
   proc_File->owner = THIS_MODULE;
   proc_File->proc_iops = 0;
   proc_File->proc_fops = &file_op;
   proc_File->mode = S_IFREG | S_IRUGO | S_IWUSR;
   proc_File->uid = 0;
   proc_File->gid = 0;
   proc_File->size = 0;
   printk(KERN_INFO "proc file %s created\n",PROC_ENTRY_FILENAME);

   return 0;
}
```

Η συνάρτηση *update_txpower* καλείται κατά τη λήψη ενός πακέτου και αποθηκεύει τις τιμές του RSSI και τη διεύθυνση mac στη λίστα.

```
void update_txpower(struct ath5k_hw *ah, struct sk_buff *skb, struct ath5k_rx_status *rs)
{
   //nobady listens. we do not keep the rssi values
   if(opened_files==0)
   {
      return ;
   }
   struct rssi_list *l=(struct rssi_list*)kmalloc(sizeof(struct ssi_list), GFP_ATOMIC);
   if(l==0)
   {
     printk("can not allocate memmory\n");
     return ;
   }
   l->rssi=rs->rs_rssi;
   l->next=0;
   struct ieee80211_mgmt *mgmt= (struct ieee80211_mgmt *)skb->data;
   memcpy(l->mac,mgmt->sa,6); //copy mac address
   if(head==0)
   {
      head=l;
      tail=l;
    }
    else
    {
        tail->next=l;
        tail=l;
    }
    wake_up(&WaitQ);
}
```

Μεγάλη σημασία εδώ έχει η τελευταία γραμμή κώδικα (*wake_up(&WaitQ)*). Σε αυτό το σημείο ενημερώνεται η ουρά αναμονής ότι υπάρχουν νέα δεδομένα στην λίστα.

5.8 Υλοποίηση σε επίπεδο εφαρμογής

Εκτός από τον driver ο υπόλοιπος μηχανισμός υλοποιείται από ένα service που τρέχει στο background. Αφού δημιουργηθεί το εικονικό αρχείο, το service διαβάζει τις τιμές του RSSI και επικοινωνεί με τους σταθμούς μέσω ενός προκαθορισμένου UDP port.

Τα μηνύματα που αποστέλλονται αναλύονται παρακάτω.

Ανταλλαγή μηνυμάτων

Παρόλο που τα πειράματά μας λάβανε χώρα σε ένα ad-hoc δίκτυο, ένας κεντρικός σταθμός προσομοίωνε ένα access point. Οι σταθμοί έστελναν μηνύματα μόνο στο access point και λάμβαναν μηνύματα μόνο από αυτό.

Ο μηχανισμός αποστέλλει μηνύματα των 3 bytes (σε επίπεδο εφαρμογής) μέσω ενός UDP port. Καθώς στο UDP port μπορεί να υπάρξει απώλεια πακέτων κάθε μήνυμα απαντάται με ένα acknowledgment (ACK).

Ο γενικός τύπος μηνύματος είναι της παρακάτω μορφής.

```
struct mechanism_message
{
   char    mechanism;
   char    type;
   int8_t  value;
};
```

Το πεδίο *mechanism* παίρνει πάντα μια προκαθορισμένη τιμή και συμβολίζει ότι το μήνυμα αυτό αναφέρεται στον συγκεκριμένο μηχανισμό.

Το πεδίο *type* καθορίζει τον τύπο του μηνύματος. Γενικά για την επικοινωνία των σταθμών ορίζονται πέντε τύποι μηνυμάτων που αναλύονται παρακάτω. Τέλος στο πεδίο *value* καταχωρούνται οι τιμές του RSSI.

Τύποι μηνυμάτων

Για την επικοινωνία των σταθμών ορίζονται πέντε τύποι μηνυμάτων.

- **HNDSH_R**
  Κατά την εκκίνηση του service ο σταθμός στέλνει ένα HNDSH_R (handshake request) μήνυμα για να ειδοποιήσει ότι υποστηρίζει τον συγκεκριμένο μηχανισμό.

- **HNDSH_A**
  Αν το access point λάβει ένα HNDSH_R απαντάει με ένα HNDSH_A (handshake ack). Στην περίπτωση που χαθεί είτε το HNDSH_R είτε το HNDSH_A ο σταθμός που έστειλε το HNDSH θα το στείλει ξανά.

- **RSSI_M**
  Ένα RSSI_M μήνυμα περιέχει στο πεδίο του value την μέση τιμή του RSSI.

- **RSSI_ACK**
  Ένα RSSI_ACK μήνυμα στέλνεται ως απάντηση στο RSSI_M. Σε ένα UDP port είναι δύσκολο να εγγυηθούμε σε ποιο ακριβώς μήνυμα απαντάει το RSSI_ACK. Για παράδειγμα μέχρι να σταλεί το RSSI_ACK ένα άλλο RSSI_M στάλθηκε αλλά χάθηκε στην πορεία. Ο παραλήπτης του RSSI_ACK δεν μπορεί να γνωρίζει σε ποιο μήνυμα αναφέρεται το ACK. Στην πραγματικότητα όμως δεν μας ενδιαφέρει να ληφθούν όλα το μηνύματα. Αυτό που μας ενδιαφέρει είναι ο κάθε σταθμός να έχει γνώση του πιο πρόσφατου RSSI. Για τον λόγο αυτόν στο πεδίο value του RSSI_ACK περιλαμβάνεται το RSSI από το τελευταίο RSSI_M που ελήφθη (αυτό που απαντάει το RSSI_ACK). Έτσι αν υπάρξει διαφοροποίηση των τιμών του RSSI_M και RSSI_ACK το RSSI_M ξαναστέλνεται (ομοίως και αν το RSSI_ACK δεν ληφθεί καθόλου).

- **HELLO**
  Ανά τακτά χρονικά διαστήματα στέλνονται hello μηνύματα ενημερώνοντας το access point ότι ο σταθμός εξακολουθεί και τρέχει το μηχανισμό. Αν ένας σταθμός απομακρυνθεί από το access point σε μια περίοδο που δεν αποστέλλονται μηνύματα υπάρχει περίπτωση να βγει εκτός εμβέλειας και η σύνδεση να χαθεί. Για τον λόγο αυτό στέλνονται περιοδικά HELLO μηνύματα. Η περίοδος των μηνυμάτων αυτών καθορίζει και την μέγιστη ταχύτητα που είναι ασφαλές να κινούνται οι σταθμοί.

Threads

Το service αποτελείται από δύο threads. Το πρώτο thread αποτελεί ένα thread αποστολέα το οποίο διαβάζει τις τιμές του RSSI από το εικονικό αρχείο και στέλνει τα αντίστοιχα RSSI_M μηνύματα**.** Το δεύτερο thread αποτελεί ένα thread παραλήπτη που ακούει το UDP port, λαμβάνει τα μηνύματα και θέτει την ένταση εκπομπής μέσω εντολών συστήματος.

Και τα δύο threads εκτελούν τις λειτουργείες τους μέσω ενός ατέρμονου βρόχου.

Ο κύριος βρόχος του thread αποστολέα είναι ο ακόλουθος:

```
int sender::run()
{
    input.open(RSSI_FILE,std::ios::in);
    while(1)
    {
        struct packet_info packet =readRssi();
        string mac=string(packet.mac_addr);
        int rssiAvg=-1;
        nMap()->lock();
        node *n=nMap()->nodeFromMac(mac);
        nMap()->unlock();
        if(n==0)
        {
            continue;
        }
        n->addRssi(packet.rssi);
        rssiAvg=n->rssiAvg();

        nMap()->lock();
        if(n->needSend() )
        {
            nMap()->unlock();
            soc->sendRssiMessage(n->ip(),rssiAvg);
        }
        else
        {
            nMap()->unlock();
        }
    }
    return 0;
}
```

Ο αποστολέας διαβάζει τις τιμές του RSSI και την διεύθυνση mac κατά το `input.open`. Έπειτα ανανεώνει το μέσο RSSI στην λίστα των σταθμών και στην περίπτωση που είναι αναγκαίο ενημερώνει τον κόμβο για την νέα μέση τιμή.

Ακολουθεί ο βρόχος του thread παραλήπτη:

```
int receiver::run()
{
    int k=soc->bindSocket();
    if(k!=0)
    {
       return 0;
    }
    while(1)
    {
        struct mechMes m=soc->received();
        if(m.type==HELLOW )
        {
          node *n = nMap()->nodeFromIp(m.ip);
          n->helloInc();
          continue;
        }
        else if(m.type==RSSI_M)
        {
            rssiReceived(m.ip,m.value );
            soc->sendAck(m.ip,m.value);
        }
        else if(m.type==ACK)
        {
            nMap()->lock();
            node *n=nMap()->nodeFromIp(m.ip);
            if(n!=0)
            {
                n->ackReceived(m.value);
            }
            nMap()->unlock();
        }
        else if(m.type==HNDSH_R)
        {
            handShR(m.ip);
        }
        else if(m.type==HNDSH_A)//ack to handshake
        {
            nMap()->lock();
            nMap()->setHandSh(true);
            nMap()->unlock();
        }
    }
    return 0;
}
```

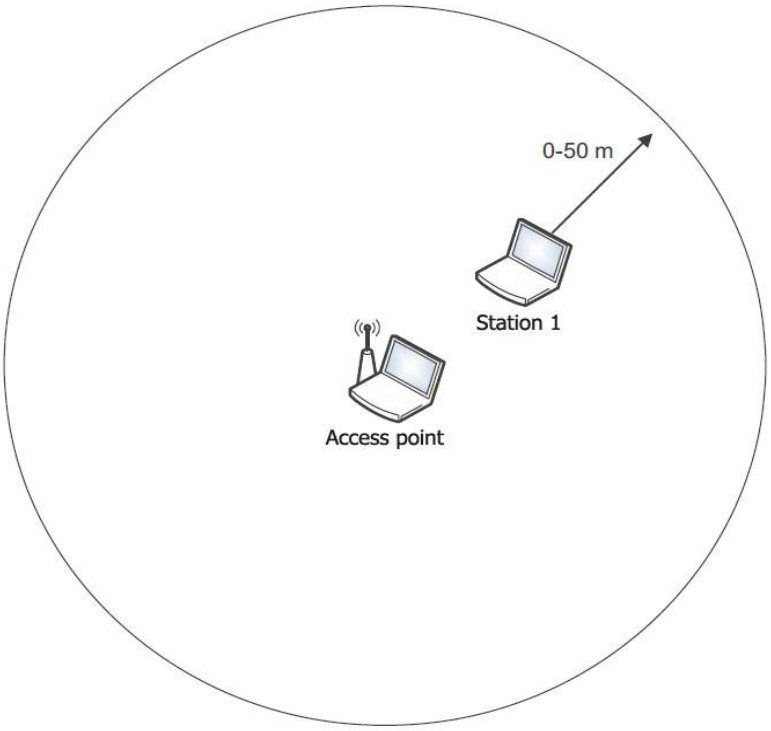
Κεφάλαιο 6

Πειράματα και αποτελέσματα

Για την αξιολόγηση του μηχανισμού που παρουσιάζουμε πραγματοποιήσαμε διαφορετικές σειρές πειραμάτων. Τα πειράματα αυτά διεξήχθησαν σε εξωτερικό χώρο ώστε να ελαττώσουμε όσο είναι δυνατόν θέματα ανάκλασης και απορρόφησης από τους τοίχους. Σε κάθε πείραμα μετρήσαμε την κατανάλωση ενέργειας καθώς και την ποιότητα της σύνδεσης. Για να μετρήσουμε την ποιότητα της σύνδεσης χρησιμοποιήθηκε το εργαλείο MTR (my traceroute). Το MTR συνιστά μια εφαρμογή που φορτώνεται σε όλους τους σταθμούς του δικτύου και παίρνει στατιστικές μετρήσεις για τη σύνδεση. Τέτοιες μετρήσεις αποτελούν ο μέσος χρόνος αναμονής (average latency), ο μέγιστος χρόνος αναμονής (maximum latency), το ποσοστό των πακέτων που χάθηκαν (packet loss) κ. α. Για να πάρει αυτές τις μετρήσεις το MTR εγκαθιδρύει συνδέσεις μεταξύ των σταθμών μέσω ICMP ή UDP πρωτοκόλλου. Όσον αφορά την κατανάλωση ενέργειας μετρήσαμε την μέση ισχύ εκπομπής σε decibel σε κάθε πείραμα. Τέλος δώσαμε ιδιαίτερη έμφαση στο packet loss καθώς στην πλειοψηφία των περιπτώσεων το packet loss προκαλεί είτε απώλεια δεδομένων είτε αναμεταδόσεις οι οποίες οδηγούν σε αυξημένη χρήση του bandwidth και υποβάθμιση της σύνδεσης.

6.1 Πρώτη σειρά πειραμάτων.

Στην πρώτη σειρά πειραμάτων χρησιμοποιήσαμε μόνο δύο σταθμούς που αποτελούνταν από δύο φορητούς υπολογιστές. Ο πρώτος φορητός υπολογιστής προσομοίωνε ένα access point και στήθηκε σε ένα σταθερό σημείο. Ο δεύτερος τοποθετήθηκε σε αντιπροσωπευτικά σημεία εντός εμβέλειας του access point. Τα σημεία αυτά επιλέχτηκαν κυρίως από την απόσταση μεταξύ των σταθμών και της διαφοράς τους στην ισχύ εκπομπής. Το πείραμα αυτό επαναλήφθηκε δύο φορές, μια με χρήση του μηχανισμού που υλοποιήσαμε και μια χωρίς αυτόν. Ο σκοπός αυτών των πειραμάτων είναι να μελετήσουμε την συμπεριφορά του μηχανισμού προσαρμογής της έντασης και το κατά πόσο αυτός επηρεάζει την ποιότητα της σύνδεσης. Για αυτό το λόγο συγκρίναμε την σύνδεση με χρήση του μηχανισμού και χωρίς αυτόν.



Εικόνα 9 Πρώτο πείραμα - τοπολογία

Ο παρακάτω πίνακας παρουσιάζει τα αποτελέσματα των μετρήσεων της ισχύος εκπομπής και του packet loss από το πρώτο σύνολο πειραμάτων. Εκτός από αυτές τις τιμές μετρήθηκε επίσης και ο μέσος χρόνος αναμονής (average latency) ο οποίος δεν παρουσίαζε καμία διαφοροποίηση. Αυτό οφείλεται στο γεγονός ότι τα πακέτα μεταδίδονταν κατευθείαν μεταξύ των σταθμών χωρίς να μεσολαβούν τρίτοι (one hop).

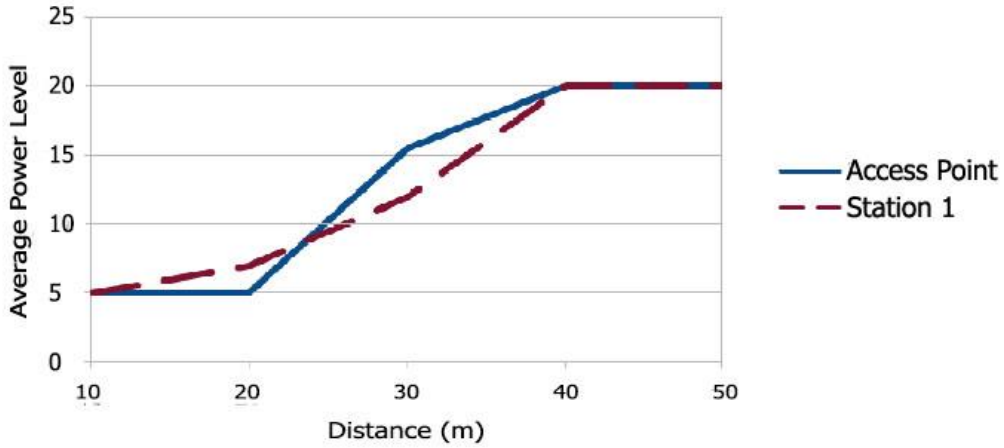
No adaptation		**Using adaptation**	
Tx power (avg)	Packet loss	Tx power (avg)	Packet loss
20	1.6%	19.5	2.6%
20	0.2%	13.5	3.6%
20	0.0%	5	0.1%
20	0.0%	5	0.0%

Βάσει των αποτελεσμάτων δεν υπάρχει ουσιαστική μεταβολή του packet loss οπότε ο μηχανισμός δεν επηρεάζει άμεσα την ποιότητα της σύνδεσης. Εκεί όμως που παρατηρείται σημαντική διαφορά είναι στην κατανάλωση ενέργειας κατά την εκπομπή. Η κατανάλωση μειώθηκε κατά 75% στις κοντινές αποστάσεις και κατά 2% σε πιο απομακρυσμένα σημεία. Είναι προφανές ότι ο μηχανισμός εξοικονόμησης ενέργειας είναι πιο αποδοτικός όταν οι σταθμοί έχουν σχετικά μικρή απόσταση μεταξύ τους. Σε μεγάλες αποστάσεις η εξοικονόμηση ενέργειας μπορεί να θεωρηθεί αμελητέα.

6.2 Δεύτερη σειρά πειραμάτων.

Στην δεύτερη σειρά πειραμάτων χρησιμοποιήσαμε πάλι δυο σταθμούς όπως και παραπάνω. Αυτήν την φορά όμως αντί να τοποθετήσουμε τον κινητό σταθμό σε διαφορετικά σημεία, τον μετακινήσαμε με τέτοιο τρόπο ώστε να απομακρύνεται από το access point και με ταχύτητα αντίστοιχη του βηματισμού ενός ανθρώπου. Σκοπός αυτού του πειράματος είναι να αξιολογήσουμε τον μηχανισμό σε ένα περιβάλλον που η σύνδεση συνεχώς υποβαθμίζεται και το κατά πόσο καλά και άμεσα καταφέρνει να ανταποκριθεί.

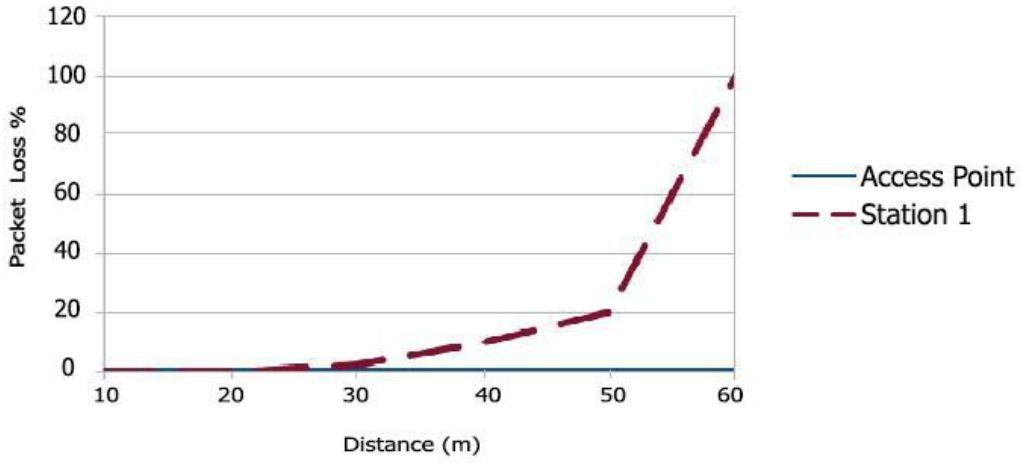
Η παρακάτω γραφική παράσταση περιγράφει την μεταβολή της έντασης εκπομπής στον σταθμό και στο Access Point σε σχέση με την μεταξύ τους απόσταση. Υπενθυμίζεται ότι χωρίς τον μηχανισμό η ισχύς εκπομπής είναι σταθερή στα 20dBm.



Εικόνα 10 Σχέση έντασης εκπομπής-απόστασης

Από την γραφική παράσταση γίνεται σαφές ότι καθώς ο σταθμός απομακρύνεται η ένταση εκπομπής αυξάνεται όπως είναι και το αναμενόμενο. Σε απόσταση σαράντα μέτρων η ισχύς έχει φτάσει το μέγιστο δυνατόν και έπειτα παραμένει σταθερή. Όταν ο κινούμενος σταθμός απομακρύνθηκε κατά πενήντα μέτρα η σύνδεση χάθηκε. Επίσης παρατηρείται μια μικρή διαφοροποίηση στο μοτίβο μεταξύ του κινητού σταθμού και του σταθμού που παρέμενε ακίνητος

Η επόμενη γραφική παράσταση αναπαριστά την μεταβολή του packet loss κατά την διάρκεια του ίδιου πειράματος.

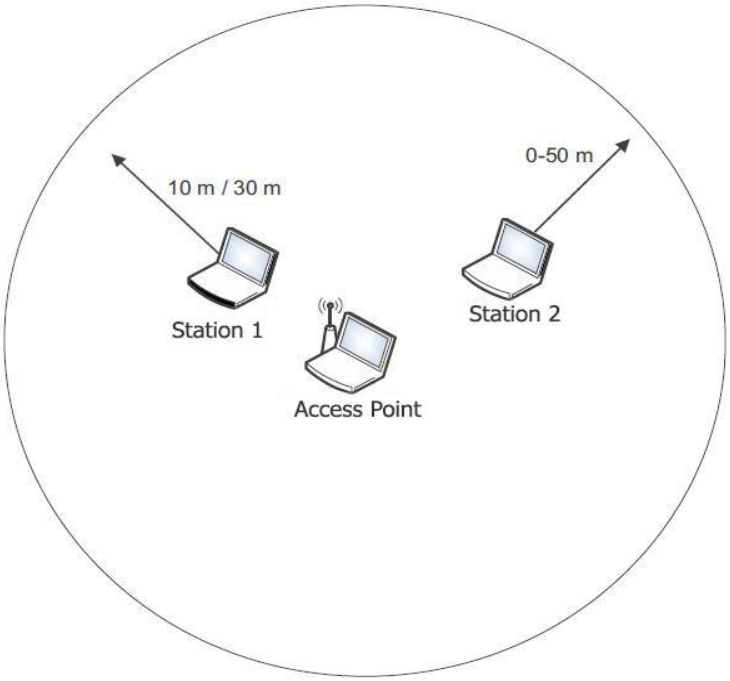


Εικόνα 11 Σχέση PDR - απόστασης

Όπως φαίνεται από την παραπάνω γραφική παράσταση, μέχρι τα σαράντα μέτρα (όπου η ισχύς εκπομπής έχει φτάσει το μέγιστο) το ποσοστό του packet loss παραμένει σε χαμηλά επίπεδα. Ξεπερνώντας τα πενήντα μέτρα το packet loss αυξάνεται ραγδαία και η σύνδεση χάνεται λόγω απόστασης.

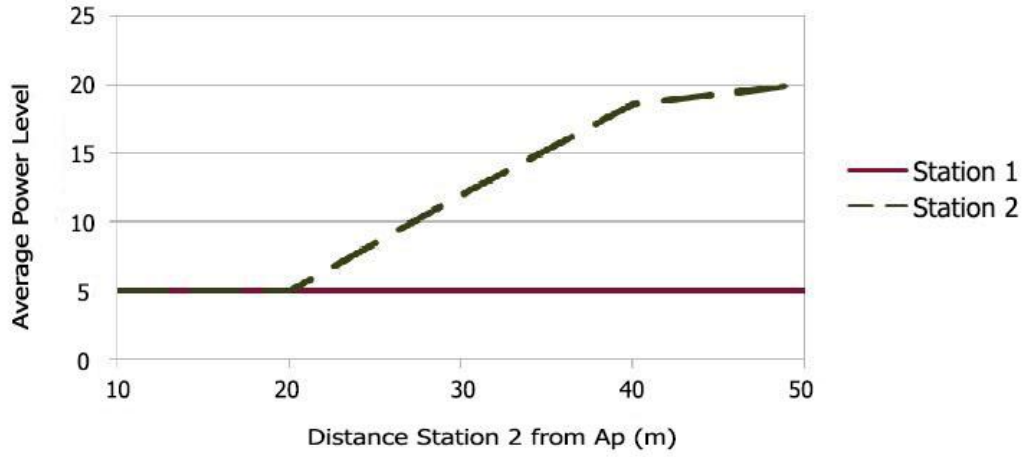
6.3 Τρίτη σειρά πειραμάτων.

Σε αυτό το σετ πειραμάτων χρησιμοποιήσαμε τρεις σταθμούς. Ένας σταθμός, όπως και παραπάνω, προσομοίωνε ένα access point και παρέμενε σταθερός. Οι υπόλοιποι σταθμοί μετακινούνταν γύρω του. Αρχικά ο πρώτος σταθμός (σταθμός 1) παρέμενε σε απόσταση δέκα μέτρων από το access point ενώ ο δεύτερος (σταθμός 2) απομακρύνονταν από το access point με ταχύτητα ίση με τον μέσο βηματισμό ενός ατόμου. Στη συνέχεια ο πρώτος σταθμός μετακινήθηκε στα τριάντα μέτρα ενώ ο δεύτερος ακολούθησε την ίδια κίνηση με προηγουμένως.

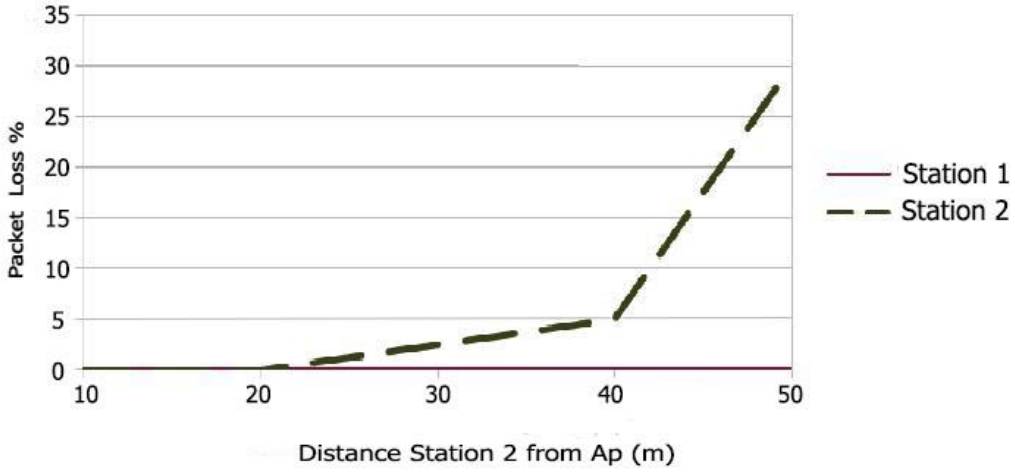


Εικόνα 12 πείραμα 3 - τοπολογία

Τα διαγράμματα παρακάτω δείχνουν πως εξελίχθηκε η ισχύς εκπομπής και το packet loss τις περιόδους που ο σταθμός 1 βρισκόταν σε απόσταση δέκα μέτρων από το access point.



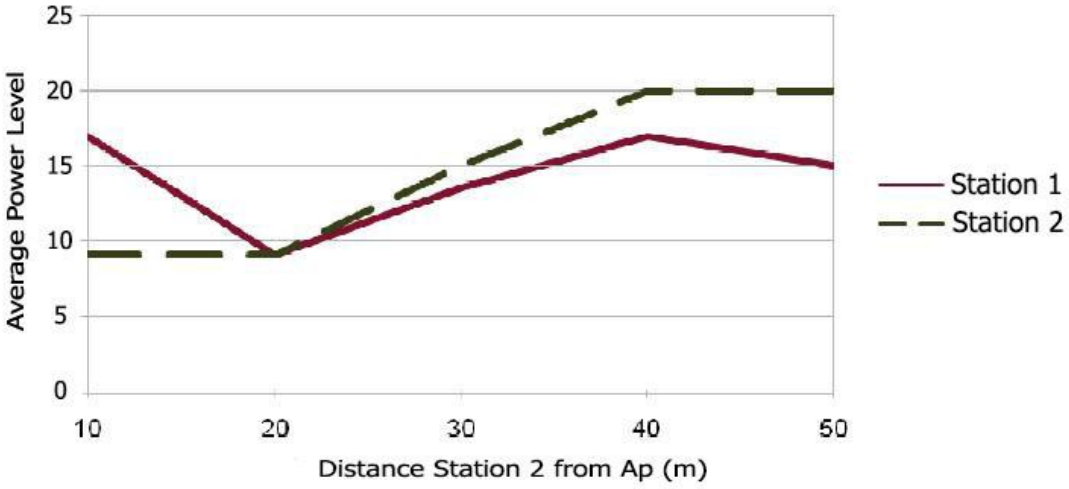
Εικόνα 13 σταθμός 1 στα 10μ.- μέση ισχύς εκπομπής



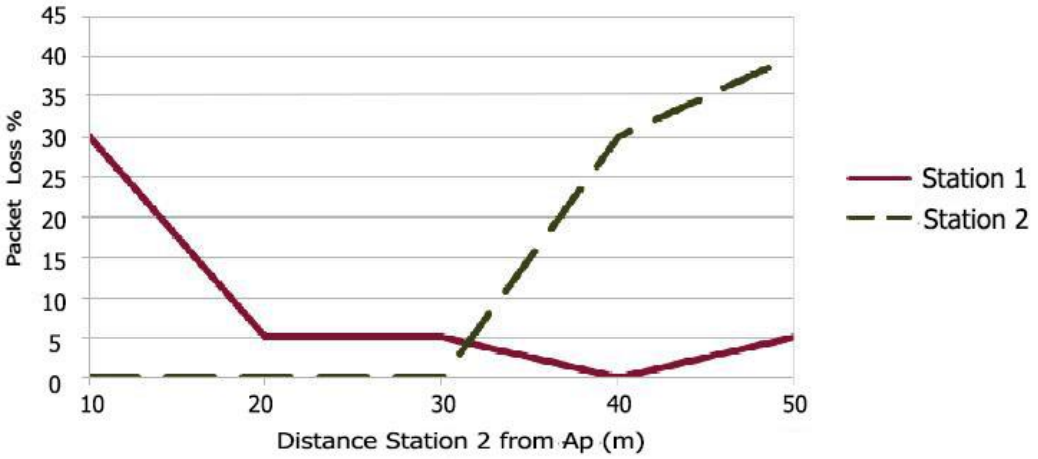
Εικόνα 14 σταθμός 1 στα 10μ. PDR

Καθ' όλη τη διάρκεια που ο σταθμός 1 βρισκόταν σε απόσταση 10 μέτρων από το access point η ισχύς εκπομπής παρέμενε στο χαμηλότερο δυνατό επίπεδο (5dBm). Παράλληλα ο σταθμός 2 αύξανε σταδιακά την ισχύ όσο απομακρυνόταν. Το packet loss επίσης φαίνεται να παραμένει σε χαμηλά επίπεδα κάτω του 5%. Εξαίρεση αποτελεί η περίοδος που ο σταθμός 2 έχει ξεπεράσει τα 40 μέτρα και σύντομα βγαίνει εκτός εμβέλειας του access point.

Ακολουθούν τα διαγράμματα μετά από την μετακίνηση του σταθμού 1 στα 30 μέτρα.



Εικόνα 15 σταθμός 1 στα 30μ.- μέση ισχύς εκπομπής



Εικόνα 16 σταθμός 1 στα 30μ. - PDR

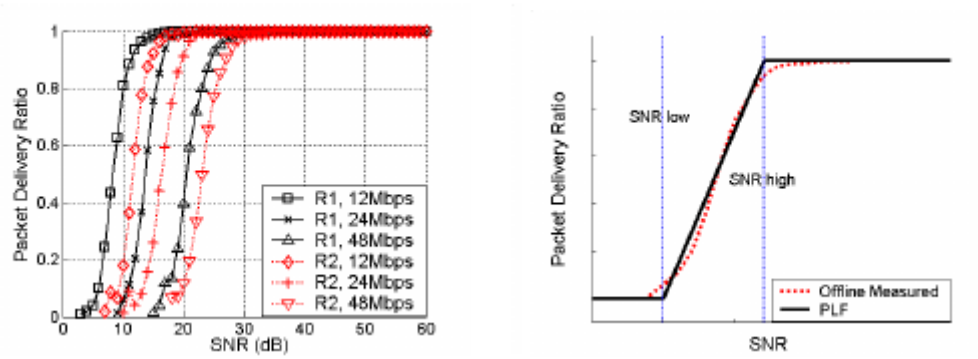
Όταν ο σταθμός 1 μετακινήθηκε στα 30 μέτρα παρατηρείται μια αύξηση στο packet loss η οποία σύντομα εξομαλύνεται. Ο μηχανισμός ρυθμίζει την ένταση ώστε το packet loss να παραμείνει σε χαμηλά επίπεδα. Η αύξηση λοιπόν που παρατηρείται προκαλείται λόγω της καθυστέρησης του μηχανισμού να προσαρμόσει την ένταση εκπομπής έπειτα από την μετακίνηση του σταθμού 1.

Γενικά ο μηχανισμός είναι καταλληλότερος για σταθμούς που κινούνται σχετικά αργά, για παράδειγμα με ταχύτητα βαδίσματος. Αν οι σταθμοί κινούνται πιο γρήγορα (π.χ. αυτοκίνητα) ο μηχανισμός αποτυχαίνει να ρυθμίσει εγκαίρως την ένταση εκπομπής. Άλλες προσεγγίσεις θα υποστηρίζανε μεγαλύτερες ταχύτητες. Για παράδειγμα θα μπορούσε να αυξηθεί η συχνότητα των 'hello' μηνυμάτων ή κατά τον υπολογισμό του κινητού μέσου του RSSI να δοθεί περισσότερο βάρος στο τελευταίο πακέτο. Αυτές οι προσεγγίσεις όμως θα οδηγούσαν σε αύξηση των μηνυμάτων που αποστέλλονται για τους σκοπούς του μηχανισμού οπότε και σε επιβάρυνση στην σύνδεση και στην κατανάλωση ενέργειας.

Κεφάλαιο 7

Επιπλέον εργασία

Όπως είδη αναφέραμε το RSSI δεν είναι κατάλληλη μετρική για περιβάλλον με υψηλό θόρυβο. Σε ένα τέτοιο περιβάλλον θα ήταν πιο κατάλληλη η ρύθμιση της έντασης εκπομπής βάσει του SNR. Για την διευκόλυνση του αναγνώστη παραθέτονται ξανά οι γραφικές παραστάσεις του κεφαλαίου 4 που περιγράφουν την σχέση του SNR με το PDR.



Εικόνα 17 Σχέση SNR - PDR

Όταν το SNR έχει τιμή ίση η μεγαλύτερη από SNR_{hight} το PDR ξεπερνάει το 90% και περαιτέρω αύξηση του SNR δεν βελτιώνει την σύνδεση. Για αυτό το λόγο θα επιθυμούσαμε να μειώσουμε την ένταση εκπομπής όποτε αυτό είναι δυνατόν ώστε το SNR να μην υπερβαίνει κατά πολύ το SNR_{high}.

Αν κάνουμε τους υπολογισμούς μας με decibel τότε έχουμε τις παρακάτω μαθηματικές εκφράσεις:

$$SNR = P_{rx} - N \quad (1), \quad \text{οπού } P_{rx} \text{ η ισχύς λήψης του σήματος και } N \text{ ο θόρυβος.}$$

Τώρα μπορούμε να ορίσουμε την ένταση εκπομπής προσαυξημένη με το κέρδος της κεραίας.

$$P_{tx} = P_t + G - L$$

Όπου P_t είναι η ένταση πριν την ενίσχυση, G το κέρδος και L η απώλεια εσωτερικά στην συσκευή. Αν θεωρίσουμε ότι το L είναι πολύ μικρό (αμελητέο) έχουμε:

$$P_{tx} = P_t + G \quad (2)$$

Πλέον το path loss ορίζεται ως εξής.

$$PathLoss = P_{tx} - P_{rx} \Rightarrow$$
$$PathLoss = P_t + G - P_{rx}$$

Αντίστοιχα:

$$P_{rx} = P_t + G - PathLoss$$

Από (1):

$$P_{rx} = SNR + N$$

Αν θέλουμε να επιτύχουμε το SNR να έχει τιμή SNR_{high} έχουμε.

$$P_{rx} = SNR_{hight} + N \Rightarrow$$
$$P_t + G - PathLoss = SNR_{hight} + N \Rightarrow$$
$$P_t = SNR_{hight} + N + PathLoss - G \quad (3)$$

Με τον τύπο (3) μπορούμε να θέσουμε την έντασή εκπομπής ώστε να επιτύχουμε το SNR_{high}. Ο τύπος αυτός όμως είναι πολύ αυστηρός καθώς δεν επιτρέπει καμία διακύμανση του SNR. Επίσης μικρές αλλαγές του θορύβου θα χρειαστούν άμεσα αλλαγή της έντασης εκπομπής. Έτσι για να είμαστε πιο ασφαλείς στις εκτιμήσεις μας θα αυξήσουμε το P_t κατά μια αυθαίρετη σταθερά C. Ο τύπος (3) πλέον έχει ως εξής.

$$P_t = SNR_{hight} + N + PathLoss - G + C \quad (4)$$

Όπως και με το RSSI έτσι και ένας μηχανισμός βασισμένος στο SNR θα ανταλλάσσει μηνύματα με την ισχύ του σήματος κατά την λήψη του. Ένας

σταθμός καθώς θα γνωρίζει την ένταση κατά την αποστολή αλλά και την λήψη του σήματος θα μπορεί να υπολογίσει το path loss. Αν παρατηρηθεί ραγδαία αλλαγή στο path loss (π.χ. λόγω κίνησης του σταθμού) ή στον θόρυβο, ο σταθμός θα επαναυπολογίζει και θα θέτει την ένταση εκπομπής από τον τύπο (4).

Ένας μηχανισμός ρύθμισης της ισχύος εκπομπής βάσει το SNR όπως αναλύθηκε παραπάνω δημοσιεύτηκε με τον τίτλο:

“Power management for wireless adapters using multiple feedback metrics”

Από του Συγγραφείς:

Χρήστος Μπούρας
Βαγγέλης Καπούλας
Κώστας Στάμος
Νικόλαος Σταθόπουλος
Νικόλαος Ταβουλάρης

Στο συνέδριο:

International Wireless Communications & Mobile Computing Conference 2014 – IWCMC 2014, Λευκωσία,Κύπρος

Κεφάλαιο 8

Επίλογος.

Η εξάπλωση των κινητών συσκευών καθιστά την εξοικονόμηση ενέργειας και την αύξηση του χρόνου αυτονομίας τους ως ένα από τα μείζονα προβλήματα του σχεδιασμού του σύγχρονου υλικού και λογισμικού. Καθώς οι χρήστες του διαδικτύου πολλαπλασιάζονται και οι διαδικτυακές υπηρεσίες συνεχώς κερδίζουν έδαφος η εξοικονόμηση ενέργειας στις ασύρματες επικοινωνίες γίνεται όλο και πιο αναγκαία.

Ο μηχανισμός που προτείνεται σε αυτήν την διπλωματική καταφέρνει να επιτύχει μια σημαντική μείωση της δαπανώμενης ενέργειας στις ασύρματες μεταδόσεις και αποτελεί ένα σημαντικό βήμα για την μελέτη του προβλήματος. Εντούτοις παρουσιάζει αρκετές ελλείψεις και ατέλειες που αποτρέπουν την άμεση εφαρμογή του. Η πιο σοβαρή από αυτές είναι ότι εξαρτάται άμεσα από το υλικό λόγω της τροποποίησης του driver. Αυτό εμποδίζει το μηχανισμό να εφαρμοστεί σε συσκευές με διαφορετικές κάρτες δικτύου. Άλλο μειονέκτημα είναι το ότι δεν έχει μελετηθεί επαρκώς η συμπεριφορά του σε περιβάλλον με υψηλό θόρυβο ή παρεμβολές. Σε αυτήν την περίπτωση η προσέγγιση βάσει του SNR , όπως περιγράφτηκε στο κεφάλαιο 7, θα ήταν προτιμότερη καθώς θα μπορούσε να αναπροσαρμόσει την ισχύ εκπομπής αναλόγως του περιβάλλοντος. Επιπλέον μελέτη χρήζει επίσης και το κατά πόσον η ενέργεια που εξοικονομήθηκε αύξησε πραγματικά την αυτονομία της συσκευής. Ο μηχανισμός κατανάλωσε ενέργεια κατά την διάρκεια υπολογισμών στον κεντρικό επεξεργαστή. Η ενέργεια αυτή δεν έχει μετρηθεί καθώς δεν υπήρχαν τα μέσα για κάτι τέτοιο. Ακόμα όμως και αν είχαμε τα μέσα να την μετρήσουμε, αυτή θα διέφερε ανάλογα με την κάθε κινητή συσκευή. Τέλος αξίζει να σημειωθεί ότι για μπορέσει να χρησιμοποιηθεί ευρέως οποιοσδήποτε μηχανισμός για την ρύθμιση της ισχύος εκπομπής θα πρέπει να υποστηριχθεί και από τους κατασκευαστές του υλικού και των προγραμμάτων οδηγών.

Αναφορές

1] A. S. a. R. Han., "An Implementation of Transmit Power Control in 802.11b Wireless Networks.," in *Techhnical report, Dept. of Comp.Science*, 2002.

2] M. Y. Malik, "Power Consumption Analysis of a Modern Smartphone," in *arXiv*, 2012.

3] C.Bouras, V. Papapanagiotoy, K. Stamos and G. Zaoudis, "“Efficient Power Management Adaptation for Video Transmission over TFRC”," in *AICT*, Barcelana, Spain, 2010.

4] J. Monks, V. Bharghavan and W.-M. Hwu, "A power controlled multiple access protocol for wireless packet networks," in *INFOCOM. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2001, pp. 219-228.

5] E. L. Aguilera and J. Casademont, "Transmission Power Control Mechanisms in IEEE 802.11 Cellular Networks," in *Proc. of the International Conference on Communications and Mobile Computing (IWCMC)*, 2006, pp. 731 - 736.

6] T. ElBatt, S. Krishnamurthy, D. Connors and S. Dao, "Power management for throughput enhancement in wireless ad-hoc networks," in *Communications. IEEE International Conference*, vol. 3, 2000, pp. 1506-1513.

7] William Vann Hasty, P. Jr. and J. Stanforth, "System and method for using per-packet receive signal strenght indication and transmit power levels to compute path loss for a link for use in layer 2 routing in a wireless communication network". U.S. Patent Patent 7,672,246,, 23 Feb 2006.

8] A. Chaltseva and E. Osipo, "On passive characterization of aggregated traffic in wireless networks," in *WWIC Proceeding of 10th international conference on Wired/Wireless Internet Communication*, 2012, pp. 282-289.

9] A. Vlavianos, L. Law, I. Broustis, S. Krishnamurthy and M. Faloutsos, "Assessing link quality in IEEE 802.11 Wireless Networks: Which is the right metric?," in *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium*, 2008, pp. 1-6.

J. Zhang, K. Tan, J. Zhao, H. Wu and Y. Zhang, "A Practical SNR-

10] Guided Rate Adaptation," in *INFOCOM. The 27th Conference on Computer Communications. IEEE*, 2008.

11] V. V. Kapadia, S. N. Patel and R. H. Jhaveri, "Comparative study of hidden node problem and solution using different techniques and protocols," in *Journal of computing*, vol. 2, 2010, pp. 65-67.

12] M. Kendall and A. Stuart, "The Advanced Theory of Statistics," *Griffin,* vol. 3, 1983.

13] [Online]. Available: http://www.gnu.org/philosophy/free-sw.htm. [Accessed September 2015].

14] T. Killian, "Processes as File," in *USENIX Summer Conf. Proc*, Salt Lake City, 1984.

15] Cisco Systems Inc., *Cisco Aironet 802.11A/B/G Wireless CardBus Adapter.*

16] "Ath5k, Atheros Linux wireless driver," [Online]. Available: http://wireless.kernel.org/en/users/Drivers/ath5k. [Accessed September 2015].

Παράρτημα

Κώδικες

Στο παράρτημα αυτό βρίσκονται οι κώδικες από το μηχανισμό σε επίπεδο εφαρμογής καθώς και το patch για την τροποποίηση του προγράμματος οδηγού.

Κώδικες εφαρμογής

Αρχείο: defs.h

```
#ifndef DEFS_H
#define DEFS_H

#define TX_MIN 5
#define TX_CONST 2
#define SPORT 7000
#define PERF_RSSI 10
#define TX_MAX 20
#define TX_DEF TX_MAX
#define WLAN "ah"

#endif
```

Αρχείο: message.h

```
#ifndef MESSAGE_H
#define MESSAGE_H
#define MECH    'M'
#define HELLOW  'h'
#define RSSI_M  'r'
#define ACK     'A'
#define HNDSH_R 'H'
#define HNDSH_A 'a'
#endif
```

Αρχείο: thread.h

```
#ifndef THREAD_H
#define THREAD_H
#include<pthread.h>

class Thread
{
    public:
        Thread(){};
        virtual ~Thread(){};

        void start()
        {
            pthread_create(&thr,NULL,thr_run,this);
        }

        virtual int cancel()
        {
            return pthread_cancel(thr);
        }

        int join()
        {
            return pthread_join(thr, NULL);
        }

    protected:
        virtual int run()=0;
    private:
        static void* thr_run(void *r)
        {

            int ret=( (Thread *)r)->run();
            pthread_exit(0);
        }
        pthread_t thr;
};
#endif
```

Αρχείο: average.h

```
#ifndef AVERAGE_H
#define AVERAGE_H
#include<stdio.h>
struct ewma {
        unsigned long internal;
        unsigned long factor;
        unsigned long weight;
};

extern void ewma_init(struct ewma *avg, unsigned long factor,
                      unsigned long weight);

extern struct ewma *ewma_add(struct ewma *avg, unsigned long val);

/**
 * ewma_read() - Get average value
 * @avg: Average structure
 *
 * Returns the average value held in @avg.
 */
static inline unsigned long ewma_read(const struct ewma *avg)
{
        return avg->internal >> avg->factor;
}
#endif
```

Αρχείο: node.h

```
#ifndef NODE_H
#define NODE_H

#include"average.h"
#include"iostream"
#include<string>
class node
{
    public:
        node(std::string mac,std::string ip) :_mac(mac),_ip(ip)
        {
            count=0;
            hello_messages = 1;
            ackRssi=-1;
            _mySignal=20;
            ewma_init(&avg,2,2);
        }
        node(){}
```

```
    bool addRssi(char rssi);
    void clear()
    {
        count=0;
        ewma_init(&avg,2,2);
    }
    char rssiAvg()
    {
        return (char)ewma_read(&avg);
    }
    std::string mac()
    {
        return _mac;
    }
    std::string ip()
    {
        return _ip;
    }
    char mySignal()
    {
        return _mySignal;
    }
    short int getHelloMessages()
    {
        return hello_messages;
    }
    void setMySignal(char s)
    {
        _mySignal=s;
    }
    void helloInc()
    {
        hello_messages++;
    }
    void resetHelloMessages()
    {
        hello_messages = 0;
    }
    bool needSend();

    void ackReceived(char rssi);

    static node* nodeFromMac(std::string mac);
    static node* nodeFromIp(std::string ip);

private:
    struct ewma avg;
    std::string _mac;
    std::string _ip;
```

```
        short int hello_messages;
        char ackRssi;
        char _mySignal;
        short count;
};
#endif
```

Αρχείο: nodeMap.h

```
#ifndef NODE_MAP_H
#define NODE_MAP_H

#include<string>
#include<map>
#include"node.h"
#include<pthread.h>

class nodeMap
{
    public:
        friend nodeMap* nMap();
        nodeMap()
        {
            handsh=false;
            pthread_mutex_init(&mutex, NULL);
        }
        ~nodeMap()
        {
            pthread_mutex_destroy(&mutex);
        }
        node* addNode(node* n);
        node* nodeFromIp(std::string ip);
        node* nodeFromMac(std::string mac);

        bool  hasHandSh()
        {
            return handsh;
        }
        void setHandSh(bool b)
        {
            handsh=b;
        }
        int size()
        {
            return ipMap.size();
        }

        typedef std::map<std::string,node*>::iterator nodeIter;
```

```
        nodeIter ipBegin()
        {
            return ipMap.begin();
        }
        nodeIter ipEnd()
        {
            return ipMap.end();
        }
        node* nodeFromIt(nodeIter &it)
        {
            return it->second;
        }
        void lock()
        {
            pthread_mutex_lock(&mutex);
        }

        void unlock()
        {
            pthread_mutex_unlock(&mutex);
        }
        static void init()
        {
            _nmap=new nodeMap();
        }
        void delete_node(node* n)
        {
            ipMap.erase(n->ip());
            macMap.erase(n->mac());
        }
        static void clear()
        {
            delete _nmap;
            _nmap=0;
        }

    private:
        pthread_mutex_t mutex;
        bool handsh;
        std::map<std::string,node*> ipMap;
        std::map<std::string,node*> macMap;
        static nodeMap* _nmap;
};
nodeMap* nMap();
#endif
```

Αρχείο: hello.h

```
#ifndef HELLO_H
#define HELLO_H
#include "thread.h"
#include <string.h>
#include "socketC.h"
class helloMessage :public Thread
{
    public:
        helloMessage(std::string ip) :Thread(),bIp(ip){};
        void setSocket(socketC *s)
        {
            soc=s;
        }

        int run();

    private:
        std::string bIp;
        socketC *soc;
};
#endif
```

Αρχείο: timeout.h

```
#ifndef TIMEOUT_H
#define TIMEOUT_H

#include "thread.h"
#include "base_receiver.h"
#define TIMEOUT 3


class timeoutChecker : public Thread
{
  public:
    timeoutChecker(baseReceiver *b):Thread(){base = b;};
    int run();

  private:
    baseReceiver *base;

};
#endif
```

Αρχείο: base_receiver.h

```
#ifndef BASE_RECEIVER_H
#define BASE_RECEIVER_H
#include"receiver.h"
#include<map>
#include <list>
#include<pthread.h>

class baseReceiver :public receiver
{
    public:
        baseReceiver():receiver()
        {
          pthread_mutex_init(&mutex, NULL);
          low_rssi=255;//just a very big value
        }
        void checkForTimeouts();

    protected:
        void rssiReceived(std::string ip,char rssi);
        void handShR(std::string ip);

    private:
        std::list <std::string> low_nodes;
        char low_rssi;
        char find_lowest();
        void clear();
        void lock()
        {
            pthread_mutex_lock(&mutex);
        }

        void unlock()
        {
            pthread_mutex_unlock(&mutex);
        }
      pthread_mutex_t mutex;
};
#endif
```

Αρχείο: node_receiver.h

```
#ifndef NODE_RECEIVER_H
#define NODE_RECEIVER_H

#include"receiver.h"
#include"defs.h"
#include <cmath>
#include<string>

class nodeReceiver :public receiver
{
    public:
        nodeReceiver(std::string s):receiver(),bIp(s){}

    protected:
        std::string bIp;
        virtual void rssiReceived(std::string ip,char rssi)
        {
            if(ip==bIp && std::abs(PERF_RSSI - rssi) > TX_CONST)
            {
                setPower(rssi);
            }
        }
        virtual void handShR(std::string ip)
        {
            return ;
        }
};
#endif
```

Αρχείο: receiver.h

```
#ifndef RECEIVER_H
#define RECEIVER_H

#include <pthread.h>
#include "socketC.h"
#include <fstream>
#include "thread.h"
#define LOG_F "receiver.log"
#define TX_LOG "tx.log"
#define DB(STRING) cout<<#STRING<<endl;
```

```
class receiver :public Thread
{
    public:
        receiver();
        virtual ~receiver()
        {
            log.close();
        };
        int setPower(char rssi);
        void setSocket(socketC *s)
        {
            soc=s;
        }
        int run();
    protected:
        virtual void rssiReceived(std::string ip,char rssi)=0;
        void sendRssiAck(std::string ip,char rssi);
        virtual void handShR(std::string ip)=0;
        int setAbsPower(char tx_new);
        std::ofstream log;
        socketC *soc;
    private:
        std::ofstream txLog;
        char curr_tx;
};
#endif
```

Αρχείο: sender.h

```
#ifndef SENDER_H
#define SENDER_H
#include"node.h"
#include <iostream>
#include <fstream>
#include"nodeMap.h"
#include"socketC.h"
#include "thread.h"

struct packet_info
{
  int rssi;
  char mac_addr[18];
};
```

```
class sender :public Thread
{
    public:
        sender();
        ~sender()
        {
            soc->closeFd();
            input.close();
        }
        int run();
        void setSocket(socketC *s)
        {
            soc=s;
        }

    private:
        std::ifstream input;
        std::ofstream log;
        struct packet_info readRssi();
        socketC *soc;
};
#endif
```

Αρχείο: socketC.h

```
#ifndef SOCKETC_H
#define SOCKETC_H
#include<string>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#define BUFF_S 3

struct mechMes
{
    std::string ip;
    char value;
    char type;
};

class socketC
{
    public:
        socketC(){}
        ~socketC()
        {
            closeFd();
        }
```

```
        void init();
        int bindSocket();
        struct mechMes received();
        int sendRssiMessage(std::string ip,char rssi);
        int sendHellow(std::string ip);
        int sendAck(std::string ip,char rssi);
        int sendHsh(std::string ip);
        int sendHshAck(std::string ip);
        void closeFd()
        {
            close(sockfd);
        }
    protected:
        int sendMessage(std::string ip, char*buff, int size);
    private:
        struct sockaddr_in peer, bindSa;
        socklen_t len;
        int sockfd;
        char buff[BUFF_S+1];
};
#endif
```

Αρχείο: average.c

```
#include"average.h"

void ewma_init(struct ewma *avg, unsigned long factor, unsigned long weight)
{

    avg->weight = weight;
    avg->factor = factor;
    avg->internal = 0;
}

struct ewma *ewma_add(struct ewma *avg, unsigned long val)
{
    avg->internal = avg->internal  ?
        (
            ( (avg->internal << avg->weight)
               - avg->internal ) +
            (val << avg->factor)
        ) >> avg->weight
        :
                (val << avg->factor);
    return avg;
 }
```

Αρχείο: base_receiver.cpp

```
#include"base_receiver.h"
#include"defs.h"
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <fstream>
#include <list>
#include <string>
#include "nodeMap.h"

using namespace std;
typedef std::map<std::string,node*>::iterator nodeIter;

#define DB(STRING) cout<<#STRING<<endl;

void baseReceiver::rssiReceived(string ip,char value)
{
    if(value<0)
    {
        return ;
    }

    lock();
    nMap()->lock();
    node *n=nMap()->nodeFromIp(ip);
    nMap()->unlock();

    if(n==0)
    {
        unlock();
        return ;
    }

    n->setMySignal(value);

    if(low_nodes.empty() )
    {
        low_rssi=value;
        low_nodes.push_front(ip);
        log<<"\tnew lowest "<<(int)low_rssi<<" "<<ip<<endl;
    }
    else
    {
```

```
        if(value < low_rssi)
        {
            low_rssi = value;
            low_nodes.clear();
            low_nodes.push_front(ip);
            log<<"\tnew lowest "<<(int)low_rssi<<" "<<ip<<endl;
        }
        else if(value > low_rssi)
        {
            std::list <std::string>::iterator it;
            for(it = low_nodes.begin(); it != low_nodes.end() ; it++)
            {
                if( ip == *it)
                {
                    low_nodes.erase(it);
                    break;
                }
            }

            if(low_nodes.empty())
            {
                low_rssi=find_lowest();
                log<<"new lowest "<<(int)low_rssi<<endl;
            }
        }
        else if(value == low_rssi)
        {
            std::list <std::string>::iterator it;
            for(it = low_nodes.begin(); it != low_nodes.end() ; it++)
            {
                if ( ip == *it)
                {
                    break;
                }
            }
            if (it == low_nodes.end())
                low_nodes.push_front(ip);
        }
    }

    if(abs(PERF_RSSI - low_rssi) > TX_CONST)
    {
        setPower(low_rssi);
        clear();
    }
    unlock();
}
```

```
void baseReceiver::clear()
{
    nMap()->lock();
    nodeMap::nodeIter it=nMap()->ipBegin();

    for(; it != nMap()->ipEnd(); it ++)
    {
        nMap()->nodeFromIt(it)->clear();
    }
    nMap()->unlock();
}

char baseReceiver::find_lowest()
{
    nMap()->lock();
    nodeMap::nodeIter it=nMap()->ipBegin();

    if(it==nMap()->ipEnd() )
    {
        nMap()->unlock();
        return -1;
    }

    char lowest = nMap()->nodeFromIt(it)->mySignal();
    it ++;

    for(; it != nMap()->ipEnd(); it ++)
    {
        char signal=nMap()->nodeFromIt(it)->mySignal();
        if( signal< lowest)
            lowest = signal;
    }

    for(it=nMap()->ipBegin(); it != nMap()->ipEnd(); it ++)
    {
        char signal=nMap()->nodeFromIt(it)->mySignal();
        if(signal == lowest)
            low_nodes.push_front(nMap()->nodeFromIt(it)->ip());
    }
    nMap()->unlock();

    return lowest;
}

void baseReceiver::handShR(string ip)
{
    node *n=node::nodeFromIp(ip);
    if(n!=0)
    {
```

```
        nMap()->lock();
        nMap()->addNode(n);
        nMap()->unlock();
        cout<<"sendAck"<<endl;
        soc->sendHshAck(ip);
    }
}


void baseReceiver::checkForTimeouts()
{

    lock();
    nMap()->lock();

    nodeIter cnode = nMap()->ipBegin();
    while(cnode!=nMap()->ipEnd() )
    {
        if(nMap()->nodeFromIt(cnode) -> getHelloMessages() == 0)
        {
            if(low_rssi== nMap()->nodeFromIt(cnode)->mySignal())
            {
                low_nodes.remove( nMap()->nodeFromIt(cnode)->ip() );
            }
            nodeIter delI=cnode;
            cnode++;
            nMap()->delete_node(nMap()->nodeFromIt(delI));
        }
        else
        {
            nMap()->nodeFromIt(cnode) -> resetHelloMessages();
            cnode++;
        }
    }


    nMap()->unlock();


    low_rssi=find_lowest();

    if(low_rssi==-1)
    {
        low_rssi=PERF_RSSI;
        setAbsPower(low_rssi);
    }
    else if(abs(PERF_RSSI - low_rssi) > TX_CONST)
    {
```

```
        setPower(low_rssi);
        clear();
    }
    unlock();
}
```

Αρχείο: hello.cpp

```
#include"hello.h"
#include "nodeMap.h"
#define TIME 1//in seconts

int helloMessage::run()
{
    while(1)
    {
        nMap()->lock();
        if(!nMap()->hasHandSh() )
        {
            nMap()->unlock();
            soc->sendHsh(bIp);
            sleep(TIME);
        }
        else
        {
            nMap()->unlock();
            break ;
        }
    }
    while(1)
    {
        soc->sendHellow(bIp);
        sleep(TIME);
    }
    return 0;
}
```

Αρχείο: node.cpp

```
#include"node.h"
#include"defs.h"
#include <cmath>
#include<stdlib.h>
#define MAX_PACK 5
using namespace std;
```

```
bool node::addRssi(char rssi)
{
    if(rssi<0 )
    {
        return false;
    }
    count++;
    ewma_add(&avg,rssi);
    return true;
}

void node::ackReceived(char rssi)
{
    ackRssi=rssi;
}

bool node::needSend()
{
    if(count<MAX_PACK)
    {
        return false;
    }
    int rssi=rssiAvg();
    //already knows
    if(ackRssi>0 && (abs(ackRssi - rssi) < TX_CONST) )
    {
        return false;
    }
    if( rssi> 0  &&  abs(PERF_RSSI - rssi) > TX_CONST)
    {
        return true;
    }
    return false;
}

node* node::nodeFromIp(string ip)
{
    char mac_add[18];
    string cmd = "arp -n -i ";
    cmd.append(WLAN);
    cmd.append("|grep -i ");
    cmd.append(ip);
    cmd.append("|awk '{print $3}'");
    FILE *fp = popen(cmd.c_str(),"r");
    if(fp==NULL)
    {
        return 0;
    }
```

```
    if(fgets(mac_add,18,fp)==NULL)
    {
        pclose(fp);
        return 0;
    }
    pclose(fp);
    int i=0;
    while (mac_add[i])
    {
        mac_add[i]=(char) ( toupper(mac_add[i]) );
        i++;
    }
    string mac=string(mac_add);
    return new node(mac,ip);
}

node* node::nodeFromMac(string mac)
{
    char ip_add[18];

    string cmd = "arp -n -i ";
    cmd.append(WLAN);
    cmd.append("|grep -i ");
    cmd.append(mac);
    cmd.append("|awk '{print $1}'");

    FILE *fp = popen(cmd.c_str(),"r");
    if(fp==NULL)
    {
        return 0;
    }
    if(fgets(ip_add,18,fp)==NULL)
    {
        pclose(fp);
        return 0;
    }
    pclose(fp);
    string ip=string(ip_add);
    return new node(mac,ip);
}
```

Αρχείο: nodeMap.cpp

```
#include"nodeMap.h"
using namespace std;
node* nodeMap::addNode(node* n)
{
    ipMap[n->ip()]=n;
    macMap[n->mac()]=n;

    return n;
}

node* nodeMap::nodeFromIp(string ip)
{
    map<string,node*>::iterator it=ipMap.find(ip);
    if(it==ipMap.end() )
    {
        return 0;
    }
    return it->second;
}

node* nodeMap::nodeFromMac(string mac)
{
    map<string,node*>::iterator it=macMap.find(mac);
    if(it==macMap.end() )
    {
        return 0;
    }
    return it->second;
}

nodeMap* nMap()
{
    return nodeMap::_nmap;
}

nodeMap* nodeMap::_nmap=0;
```

Αρχείο: receiver.cpp

```
#include <stdio.h>
#include <string.h>
#include<iostream>
#include <stdlib.h>
#include <time.h>
#include"receiver.h"
#include"message.h"
```

```
#include"defs.h"
#include "nodeMap.h"
#include <signal.h>
#include <unistd.h>
#define BUFF_S 3
using namespace std;

receiver::receiver() :Thread(),log(LOG_F)
{
    curr_tx=TX_DEF;
}

int receiver::run()
{
    log<<"Starting"<<endl;
    int k=soc->bindSocket();
    if(k!=0)
    {
        log<<"could not bind socket"<<endl;
        log<<"ERR "<<k<<endl;
        return 0;
    }

    while(1)
    {
        struct mechMes m=soc->received();
        log<<"[RECEIVED]"<<m.ip<<endl;
        if(m.type==HELLOW )
        {
            log<<'\t'<<"hello"<<endl;
            node *n = nMap()->nodeFromIp(m.ip);
            n->helloInc();
            continue;
        }
        else if(m.type==RSSI_M)
        {
            log<<'\t'<<"rssi "<<(int)m.value<<endl;
            rssiReceived(m.ip,m.value );
            soc->sendAck(m.ip,m.value);
        }
        else if(m.type==ACK)
        {
            log<<'\t'<<"ack "<<(int)m.value<<endl;
            nMap()->lock();
            node *n=nMap()->nodeFromIp(m.ip);
            if(n!=0)
            {
                n->ackReceived(m.value);
            }
```

```
            nMap()->unlock();
        }
        else if(m.type==HNDSH_R)
        {
            log<<'\t'<<"handshake req"<<endl;
            handShR(m.ip);
        }
        else if(m.type==HNDSH_A)//ack to handshake
        {
            log<<'\t'<<"handshake ack "<<endl;
            nMap()->lock();
            nMap()->setHandSh(true);
            nMap()->unlock();
        }
    }
    return 0;
}

int receiver::setPower(char rssi)
{
    if(rssi<0 )
    {
        return -1;
    }
    char path_loss = curr_tx - rssi;
    char tx_new;
    tx_new=PERF_RSSI+path_loss;

    return setAbsPower(tx_new);
}

int receiver::setAbsPower(char tx_new)
{
    if(tx_new <TX_MIN)
        tx_new=TX_MIN;

    if(tx_new >TX_MAX)
        tx_new=TX_MAX;

    if(tx_new!=curr_tx)
    {
        char foo[3];
        string cmd = "iwconfig ";
        cmd+=WLAN;
        cmd+=" txpower ";
        sprintf(foo,"%d",tx_new);
        cmd.append(foo);
        time_t t=time(NULL);
```

```
        if(system(cmd.c_str()) !=0)
        {
            cout<<"error on system command"<<endl;
            return -1;
        }

        txLog.open(TX_LOG,std::ios::out | std::ios::app);
        int txl=curr_tx;
        txLog<<t<<'\t'<<txl;
        txl=tx_new;
        txLog<<'\t'<<txl<<endl;
        curr_tx=tx_new;
        txLog.close();

        log<<'\t'<<cmd<<endl;
    }
    return 0;
}
```

Αρχείο: sender.cpp

```
#include"sender.h"

#include <cstdlib>
#include <cstring>
#include <cmath>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include"defs.h"
#include"message.h"

#define RSSI_FILE "/proc/rssi"
#define LOG_F "sender.log"
using namespace std;

sender::sender()
    :log(LOG_F){}

int sender::run()
{
    input.open(RSSI_FILE,std::ios::in);

    while(1)
    {
        struct packet_info packet =readRssi();
```

```
        if(packet.rssi<0)
        {
            continue ;
        }
        log<<"RSSI_READ: "<<packet.mac_addr<<" "<<packet.rssi<<endl;
        string mac=string(packet.mac_addr);
        int rssiAvg=-1;
        nMap()->lock();
        node *n=nMap()->nodeFromMac(mac);
        nMap()->unlock();

        if(n==0)
        {
            log<<"\tUNKOWN MAC "<<packet.mac_addr<<endl;
            continue;
        }

        n->addRssi(packet.rssi);
        rssiAvg=n->rssiAvg();

        nMap()->lock();
        if(n->needSend() )
        {
            nMap()->unlock();
            log<<"\tsend message"<<packet.mac_addr<<" "<<packet.rssi<<endl;
            soc->sendRssiMessage(n->ip(),rssiAvg);
        }
        else
        {
            nMap()->unlock();
        }
    }
    return 0;
}

struct packet_info sender::readRssi()
{
    struct packet_info packet;
    char info[35];
    char rssiC[4];

    memset(packet.mac_addr,'\0',18);
    memset(rssiC,'\0',4);
    input.getline(info,35);
    char *p=info+5;
    int i=0;
```

```
    while(*p!=' ')
    {
        rssiC[i]=*p;
        p++;
        i++;
        if(i==3)
        {
            packet.rssi = -1;
            return packet;
        }
    }
    packet.rssi = atoi(rssiC);
    if(packet.rssi<0)
    {
        cout<<"ERR:negative rssi\n"<<endl;
    }
    p+=5;
    strncpy(packet.mac_addr,p,18);
    return packet;
}
```

Αρχείο: socketC.cpp

```
#include"socketC.h"
#include<stdlib.h>
#include<cstring>
#include<iostream>
using namespace std;
#include <arpa/inet.h>

#include"defs.h"
#include"message.h"

void socketC::init()
{
    sockfd = socket(PF_INET,SOCK_DGRAM,IPPROTO_UDP);
}

int socketC::bindSocket()
{
    memset(&bindSa, 0, sizeof bindSa);
    bindSa.sin_family = PF_INET;
    bindSa.sin_port =htons(SPORT);
    bindSa.sin_addr.s_addr = INADDR_ANY;
    socklen_t len = sizeof(peer);

    return bind(sockfd,(struct sockaddr *)&bindSa,sizeof(bindSa) );
}
```

```
struct mechMes socketC::received()
{
    while(1)
    {
        std::cout<<"waiting message"<<std::endl;
        memset(buff,0,BUFF_S+1);
        if(recvfrom(sockfd,buff,BUFF_S,0,(struct sockaddr *)&peer,&len) > 0)
        {
            if(buff[0]==MECH )
            {
                break ;
            }
        }
    }
    struct mechMes message;
    message.type=buff[1];
    message.value=buff[2];
    message.ip=std::string(inet_ntoa(peer.sin_addr) );
    cout<<"message received "<<message.type<<endl;
    return message;
}

int socketC::sendHsh(string ip)
{
    char buff[3];
    buff[0]=MECH;
    buff[1]=HNDSH_R;
    buff[2]='\0';
    return sendMessage(ip,buff,3);
}

int socketC::sendAck(string ip, char rssi )
{
    char buff[3];
    buff[0]=MECH;
    buff[1]=ACK;
    buff[2]=rssi;
    return sendMessage(ip,buff,3);
}

int socketC::sendHshAck(string ip)
{
    char buff[3];
    buff[0]=MECH;
    buff[1]=HNDSH_A;
    buff[2]=0;
    return sendMessage(ip,buff,3);
}
```

```
int socketC::sendRssiMessage(string ip,char rssi)
{
    char buff[3];
    buff[0]=MECH;
    buff[1]=RSSI_M;
    buff[2]=rssi;
    return sendMessage(ip,buff,3);
}

int socketC::sendHellow(string ip)
{
    char buff[3];
    buff[0]=MECH;
    buff[1]=HELLOW;
    buff[2]='\0';
    return sendMessage(ip,buff,3);
}

int socketC::sendMessage(string ip,char* buff, int size)
{
    struct sockaddr_in sa;
    memset(&sa, 0, sizeof sa);
    sa.sin_family = PF_INET;
    sa.sin_port = htons(SPORT);
    sa.sin_addr.s_addr = inet_addr(ip.c_str() );
    sendto(sockfd,buff,size,0,(struct sockaddr *)&sa,sizeof(sa));
    return 0;
}
```

Αρχείο: timeout.cpp

```
#include "timeout.h"
#include "nodeMap.h"
#include "base_receiver.h"

int timeoutChecker :: run()
{
  while(1)
  {
    sleep(TIMEOUT);
    base->checkForTimeouts();
  }
  return 0;
}
```

Αρχείο: nodeRun.cpp

```
#include"node_receiver.h"
#include"sender.h"
#include"nodeMap.h"
#include"socketC.h"
#include"hello.h"
#include<pthread.h>
#include"nodeMap.h"
#include <signal.h>

helloMessage *hl;
sender *Send;
nodeReceiver *Rec;
socketC *SocR;
socketC *SocS;
socketC *SocH;
void leave(int sig);

int main()
{
    (void) signal(SIGINT,leave);
    (void) signal(SIGTERM,leave);
    nodeMap::init();
    std::string ip("10.0.0.2");
    node *n=node::nodeFromIp(ip);
    if(n==0)
    {
        std::cout<<"error initialising the node"<<std::endl;
        return 1;
    }
    std::cout<<n->mac()<<" "<<n->ip()<<std::endl;
    nMap()->addNode(n);
    SocR =new socketC();
    SocS =new socketC();
    SocH =new socketC();
    Send =new sender();
    Rec =new nodeReceiver(ip);
    hl=new helloMessage(ip);
    SocR->init();
    SocH->init();
    SocS->init();
    Send->setSocket(SocS);
    Rec->setSocket(SocR);
    hl->setSocket(SocH);
    Rec->start();
    hl->start();
    Send->start();
```

```
    hl->join();
    return 0;
}

void leave(int sig)
{
    Rec->cancel();
    Send->cancel();
    hl->cancel();

    delete hl;
    delete Rec;
    delete Send;
    nodeMap::clear();
    delete SocR;
    delete SocH;
    delete SocS;
}
```

Αρχείο: baseRun.cpp

```
#include"base_receiver.h"
#include"sender.h"
#include"nodeMap.h"
#include"socketC.h"
#include"timeout.h"
#include <signal.h>
#include <cstdlib>
#include <unistd.h>

sender *Send;
baseReceiver *Rec;
socketC *Soc;
timeoutChecker *Chk;

void leave(int sig);



int main()
{
    (void) signal(SIGINT,leave);
    (void) signal(SIGTERM,leave);
    nodeMap::init();
    Soc =new socketC();
    Send =new sender();
    Rec =new baseReceiver();
```

```
    Chk = new timeoutChecker(Rec);
    Soc->init();
    Send->setSocket(Soc);
    Rec->setSocket(Soc);
    Rec->start();
    Chk->start();
    Send->start();
    Rec->join();
}

void leave(int sig)
{
    Rec->cancel();
    Send->cancel();
    delete Rec;
    delete Send;
    nodeMap::clear();
    delete Soc;
    delete Chk;
    exit (0);
}
```

Αρχείο: makefile

```
CXX      =  g++
CXXFLAGS = -ggdb
TARGET   = ../build


all: mktarget baseRun nodeRun txaver

mktarget:
      mkdir -p $(TARGET)

txaver: aver.cpp
      $(CXX) $(CXXFLAGS) $^ -o $(TARGET)/$@

baseRun: baseRun.cpp $(TARGET)/base_receiver.o $(TARGET)/sender.o
$(TARGET)/timeout.o $(TARGET)/average.o $(TARGET)/receiver.o $(TARGET)/node.o
$(TARGET)/nodeMap.o $(TARGET)/socketC.o
      $(CXX) $(CXXFLAGS)  -lpthread -pthread $^ -o $(TARGET)/$@

nodeRun: nodeRun.cpp $(TARGET)/base_receiver.o $(TARGET)/hello.o
$(TARGET)/sender.o  $(TARGET)/average.o $(TARGET)/receiver.o $(TARGET)/node.o
$(TARGET)/nodeMap.o $(TARGET)/socketC.o
      $(CXX) $(CXXFLAGS)  -lpthread -pthread $^ -o $(TARGET)/$@
```

```
$(TARGET)/sender.o: sender.cpp  $(TARGET)/node.o
	$(CXX) $(CXXFLAGS) sender.cpp  -c -o $@

$(TARGET)/node.o: $(TARGET)/average.o node.cpp
	$(CXX) $(CXXFLAGS)   node.cpp  -c -o $@

$(TARGET)/average.o: average.c
	$(CXX) $(CXXFLAGS) average.c -c -o $@

$(TARGET)/base_receiver.o: base_receiver.cpp $(TARGET)/receiver.o
	$(CXX) $(CXXFLAGS) base_receiver.cpp -c -o $@

$(TARGET)/receiver.o: receiver.cpp
	$(CXX) $(CXXFLAGS) -lpthread -pthread receiver.cpp -c -o $@

$(TARGET)/hello.o: hello.cpp $(TARGET)/nodeMap.o
	$(CXX) $(CXXFLAGS) -lpthread -pthread hello.cpp -c -o $@

$(TARGET)/timeout.o:timeout.cpp
	$(CXX) $(CXXFLAGS) -lpthread -pthread timeout.cpp -c -o $@

$(TARGET)/nodeMap.o: nodeMap.cpp
	$(CXX) $(CXXFLAGS) $^  -c -o $@

$(TARGET)/socketC.o: socketC.cpp
	$(CXX) $(CXXFLAGS) $^ -c -o $@

clean:
	rm -f *.o sender base_receiver
	rm -rf $(TARGET)
```

Τροποποίηση προγράμματος οδηγού

Ακολουθεί το patch για το σύνολο οδηγών compat wireless με έκδοση compat-wireless-2012-01-08.

Patch

```
diff -crBN compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/base.c
compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/base.c
*** compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/base.c    2012-
01-08 23:10:27.000000000 +0200
--- compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/base.c    2012-
03-16 14:54:00.000000000 +0200
***************
*** 67,72 ****
--- 67,74 ----

  #define CREATE_TRACE_POINTS
  #include "trace.h"
+ #include"txpower.h"
+ #include"linux/skbuff.h"

  bool ath5k_modparam_nohwcrypt;
  module_param_named(nohwcrypt, ath5k_modparam_nohwcrypt, bool, S_IRUGO);
***************
*** 1312,1317 ****
--- 1314,1320 ----
  ath5k_receive_frame(struct ath5k_hw *ah, struct sk_buff *skb,
                struct ath5k_rx_status *rs)
  {
+ //  update_txpower(ah, skb, rs);
      struct ieee80211_rx_status *rxs;

      ath5k_remove_padding(skb);
***************
*** 1366,1378 ****

      trace_ath5k_rx(ah, skb);

      ath5k_update_beacon_rssi(ah, skb, rs->rs_rssi);

      /* check beacons in IBSS mode */
      if (ah->opmode == NL80211_IFTYPE_ADHOC)
            ath5k_check_ibss_tsf(ah, skb, rxs);

!     ieee80211_rx(ah->hw, skb);
```

```
  }

  /** ath5k_frame_receive_ok() - Do we want to receive this frame or not?
--- 1369,1383 ----

      trace_ath5k_rx(ah, skb);

+     update_txpower(ah, skb, rs);
      ath5k_update_beacon_rssi(ah, skb, rs->rs_rssi);
+

      /* check beacons in IBSS mode */
      if (ah->opmode == NL80211_IFTYPE_ADHOC)
            ath5k_check_ibss_tsf(ah, skb, rxs);

!     ieee80211_rx(ah->hw, skb);
  }

  /** ath5k_frame_receive_ok() - Do we want to receive this frame or not?
***************
*** 1480,1486 ****
                  ATH5K_ERR(ah, "error in processing rx descriptor\n");
                  ah->stats.rxerr_proc++;
                  break;
!           }

            if (ath5k_receive_frame_ok(ah, &rs)) {
                  next_skb = ath5k_rx_skb_alloc(ah, &next_skb_addr);
--- 1485,1491 ----
                  ATH5K_ERR(ah, "error in processing rx descriptor\n");
                  ah->stats.rxerr_proc++;
                  break;
!           }

            if (ath5k_receive_frame_ok(ah, &rs)) {
                  next_skb = ath5k_rx_skb_alloc(ah, &next_skb_addr);
***************
*** 2559,2564 ****
--- 2563,2570 ----
      /* ready to process interrupts */
      __clear_bit(ATH_STAT_INVALID, ah->status);

+     txpower_init();
+
      return 0;
  err_ah:
      ath5k_hw_deinit(ah);
***************
*** 2985,2991 ****
```

```
--- 2991,2999 ----
  void
  ath5k_deinit_ah(struct ath5k_hw *ah)
  {
+     txpower_clean();
      struct ieee80211_hw *hw = ah->hw;
+

      /*
       * NB: the order of these is important:
diff -crBN compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/Makefile
compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/Makefile
*** compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/Makefile  2012-
01-08 23:10:27.000000000 +0200
--- compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/Makefile  2012-
01-18 14:31:40.000000000 +0200
***************
*** 15,20 ****
--- 15,21 ----
  ath5k-y                     += ani.o
  ath5k-y                     += sysfs.o
  ath5k-y                     += mac80211-ops.o
+ ath5k-y                     += txpower.o
  ath5k-$(CONFIG_ATH5K_DEBUG) += debug.o
  ath5k-$(CONFIG_ATH5K_AHB)   += ahb.o
  ath5k-$(CONFIG_ATH5K_PCI)   += pci.o
diff -crBN compat-wireless-2012-01-
08/drivers/net/wireless/ath/ath5k/txpower.c compat-wireless-2012-01-
08/drivers/net/wireless/ath/ath5k/txpower.c
*** compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/txpower.c 1970-
01-01 02:00:00.000000000 +0200
--- compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/txpower.c 2012-
09-03 16:30:08.000000000 +0300
***************
*** 0 ****
--- 1,245 ----
+ #include"txpower.h"
+ #include <linux/average.h>
+ //#include "phy.c"
+
+ #include <linux/module.h>
+ #include <linux/moduleparam.h>
+ #include <linux/kernel.h>
+ #include <linux/init.h>
+ #include <linux/stat.h>
+ #include <linux/skbuff.h>
+ #include <linux/ieee80211.h>
+
+ #define PROC_ENTRY_FILENAME "rssi"
```

```
+ #define MAC_LEN 2
+
+ static struct proc_dir_entry *proc_File;
+ static int proc_open(struct inode *inode, struct file *file);
+ static int proc_close(struct inode *inode, struct file *file);
+ static ssize_t proc_read(struct file *file, char *buf,size_t len,loff_t *
offset);
+ static ssize_t proc_write(struct file *file, const char *buf,size_t
length,loff_t * offset);
+
+ DECLARE_WAIT_QUEUE_HEAD(WaitQ);
+ // module_param(myint, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
+
+
+ // int calc_path_loss(int power,int rssi)
+ // {
+ //     return power - rssi;
+ // }
+ /*
+ void update_txpower(struct ath5k_hw *ah, struct sk_buff *skb,
ath5k_rx_status *rs)
+ {
+     int rssi=rs->rs_rssi;
+     printk("RSSI %d\n",rssi);
+ //     int k=ath5k_hw_set_txpower_limit(ah,12);
+ //     struct sk_buff *skb_sent=malloc(sizeof(struct sk_buff));
+ //     memset(skb_sent, 0, sizeof(struct sk_buff));
+ //     skb_sent->data=malloc(sizeof(int));
+ //     memset(skb_sent->data,rssi,sizeof(char));
+ //     skb_sent->data_len=sizeof(char);
+ //     memcpy(skb_sent->mac.raw,skb->mac.raw,skb->mac_len);
+ //     skb_sent->mac_len=skb->mac_len;
+ //     int k=ath5k_tx_queue(ah->hw,skb_sent,&ah->txqs[1]);
+ //     printk("SENT %d\n",k);
+
+     return ;
+
+     struct txpower  txp;
+
+    // struct ieee80211_mgmt *mgmt = (struct ieee80211_mgmt *)skb->data;
+
+     int rssi_average=(int)ewma_read(&ah->ah_beacon_rssi_avg);
+     int path_loss=calc_path_loss(ah->ah_txpower.txp_cur_pwr,rssi_average);
+
+     int power=9;
+     if(path_loss >txp.max_path_loss)
+     {
+     //minimase the txpower
+            u8 mode=AR5K_EEPROM_MODE_11B;
```

```
+      struct ieee80211_channel *channel = &ah->ah_current_channel;
+      //ath5k_hw_txpower(ah,channel,power);
+     }
+     else if(path_loss<txp.min_path_loss)
+     {
+     //increase the txpower
+     u8 mode=AR5K_EEPROM_MODE_11B;
+     struct ieee80211_channel *channel = &ah->ah_current_channel;
+     //ath5k_hw_txpower(ah,channel,power);
+     }
+ }
+ */
+
+ struct rssi_list
+ {
+     s8 rssi;
+     u8 mac[6];
+     struct rssi_list *next;
+ };
+
+ int opened_files;
+ struct rssi_list *head;
+ struct rssi_list *tail;
+
+ static void clean_list(void)
+ {
+     struct rssi_list *l=head;
+     head=0;
+     tail=0;
+     while(l!=0)
+     {
+     struct rssi_list *tmp=l->next;
+     kfree(l);
+     l=tmp;
+     }
+
+ }
+
+ static struct file_operations file_op =
+ {
+         .read = proc_read,  /* "read" from the file */
+     .write = proc_write,
+         .open = proc_open,    /* called when the /proc file is opened */
+         .release = proc_close,        /* called when it's closed */
+ };
+
+ static ssize_t proc_write(struct file *file, const char *buf,size_t
length,loff_t * offset)
+ {
```

```
+     printk("writting is not supported\n");
+     return 0;
+ }
+
+
+ static ssize_t proc_read(struct file *file, char *buf,size_t len,loff_t *
offset)
+ {
+     if ((file->f_flags & O_NONBLOCK) )
+     {
+     return -EAGAIN;
+     }
+
+     while(head==0)
+     {
+     wait_event_interruptible(WaitQ, head);
+     }
+
+     char message[32];
+     memset (message,'\0',32);
+     int k;
+     if(head->rssi>999 || head->rssi<-99)
+     {
+         k=-1;
+     }
+     else
+     {
+         k=head->rssi;
+     }
+
+     k=sprintf(message, "rssi=%d
mac=%02X:%02X:%02X:%02X:%02X:%02X\n",k,head->mac[0],head->mac[1],head-
>mac[2],head->mac[3],head->mac[4],head->mac[5]);
+     int err=copy_to_user(buf,message,32);
+
+     if(err!=0)
+     {
+     printk("ERROR copping to users %d\n",err);
+     return 0;
+     }
+
+     struct rssi_list *l=head;
+     head=head->next;
+     kfree(l);
+
+     return k;
+ }
+
+ static int proc_open(struct inode *inode, struct file *file)
```

```
+ {
+     try_module_get(THIS_MODULE);
+     opened_files++;
+     file->private_data=0;
+     printk("RSSI FILE OPENED\n");
+     return 0;
+ }
+
+ static int proc_close(struct inode *inode, struct file *file)
+ {
+     module_put(THIS_MODULE);
+     printk("RSSI FILE CLOSED\n");
+     opened_files--;
+     if(opened_files==0)
+     {
+         clean_list();
+     }
+     return 0;
+ }
+
+
+ int txpower_init(void)
+ {
+     printk("txpower init");
+     proc_File = create_proc_entry(PROC_ENTRY_FILENAME, 0444, NULL);
+     opened_files=0;
+     head=0;
+     tail=0;
+     if (proc_File == NULL)
+     {
+     remove_proc_entry(PROC_ENTRY_FILENAME, proc_File);
+     printk(KERN_ALERT "Error: Could not initialize
%s\n",PROC_ENTRY_FILENAME);
+     return ENOMEM;
+     }
+
+ //     proc_File->owner = THIS_MODULE;
+     proc_File->proc_iops = 0;
+     proc_File->proc_fops = &file_op;
+     proc_File->mode = S_IFREG | S_IRUGO | S_IWUSR;
+     proc_File->uid = 0;
+     proc_File->gid = 0;
+     proc_File->size = 0;
+     printk(KERN_INFO "proc file %s created\n",PROC_ENTRY_FILENAME);
+
+     return 0;
+ }
+
```

```
+ void update_txpower(struct ath5k_hw *ah, struct sk_buff *skb, struct
ath5k_rx_status *rs)
+ {
+     //nobady listens. we do not keep the rssi values
+
+     if(opened_files==0)
+     {
+         return ;
+     }
+
+     struct rssi_list *l=(struct rssi_list*)kmalloc(sizeof(struct
rssi_list), GFP_ATOMIC);
+     if(l==0)
+     {
+     printk("can not allocate memmory\n");
+     return ;
+     }
+
+     l->rssi=rs->rs_rssi;
+     l->next=0;
+     struct ieee80211_mgmt *mgmt= (struct ieee80211_mgmt *)skb->data;
+     memcpy(l->mac,mgmt->sa,6);
+
+     if(head==0)
+     {
+     head=l;
+     tail=l;
+     }
+     else
+     {
+     tail->next=l;
+     tail=l;
+     }
+     wake_up(&WaitQ);
+     return ;
+ }
+
+ void txpower_clean(void)
+ {
+     remove_proc_entry(PROC_ENTRY_FILENAME, proc_File);
+ //     clean_list();
+     printk(KERN_INFO "proc file %s removed\n",PROC_ENTRY_FILENAME);
+     printk("txpower cleared");
+ }
+
+
+
```

```
diff -crBN compat-wireless-2012-01-
08/drivers/net/wireless/ath/ath5k/txpower.h compat-wireless-2012-01-
08/drivers/net/wireless/ath/ath5k/txpower.h
*** compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/txpower.h 1970-
01-01 02:00:00.000000000 +0200
--- compat-wireless-2012-01-08/drivers/net/wireless/ath/ath5k/txpower.h 2012-
03-07 13:56:52.000000000 +0200
***************
*** 0 ****
--- 1,27 ----
+ #ifndef TXPOWER_H
+ #define TXPOWER_H
+ //#include"phy.h"
+ #include "ath5k.h"
+ #include <linux/proc_fs.h>      /* Necessary because we use proc fs */
+ #include <linux/sched.h>        /* For putting processes to sleep and
+                                    waking them up */
+ #include <asm/uaccess.h>        /* for get_user and put_user */
+
+
+ struct txpower
+ {
+     int max_path_loss;
+     int min_path_loss;
+ };
+
+
+
+ // int calc_path_loss(int power,int rssi);
+
+ int txpower_init(void);
+ void txpower_clean(void);
+
+ void update_txpower(struct ath5k_hw *ah, struct sk_buff *skb, struct
ath5k_rx_status *rs);
+
+
+ #endif
```