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

Αλγόριθμοι Ελέγχου Ισχύος σε Αυτόνομα Ενεργειακά Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Α. Αρμενιάκος

Επιβλέπων : Γεώργιος Στασινόπουλος

Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2011

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

Αλγόριθμοι Ελέγχου Ισχύος σε Αυτόνομα Ενεργειακά Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Α. Αρμενιάκος

Επιβλέπων : Γεώργιος Στασινόπουλος

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 01η Ιουνίου 2011 .

.....	.....	.....
Γ. Στασινόπουλος	Ε. Συκάς	Μ. Θεολόγου

Αθήνα, Ιούνιος 2011

.....

Γεώργιος Α. Αρμενιάκος

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

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η εφαρμογή αλγόριθμων για την διαχείριση της ενέργειας σε αυτόνομα ενεργειακά συστήματα. Πιο συγκεκριμένα χρησιμοποιήσαμε το eZ430-RF2500 της Texas Instruments. Το eZ430-RF2500 περιλαμβάνει ένα σένσορα που του επιτρέπει να μετράει την θερμοκρασία του περιβάλλοντος. Επίσης τα modules αυτά έχουν την δυνατότητα να επικοινωνούν ασύρματα μεταξύ τους και να ανταλλάσουν πληροφορίες. Χρησιμοποιώντας ένα module σε λειτουργία Access Point, τροφοδοτούμενο μέσω της USB του Η/Υ, και ένα (ή περισσότερα) modules σε λειτουργία πελάτη, τροφοδοτούμενα μέσω δύο AA μπαταριών, εφαρμόζοντας διάφορους αλγόριθμους, εξομοιώσαμε πως αυτά θα συμπεριφέρονταν αν τα τελευταία δεν τροφοδοτούνταν μέσω των μπαταριών, αλλά μέσω κάποιας πηγής ενέργειας που θα τα καθιστά αυτόνομα ενεργειακά συστήματα. Αυτή η πηγή ενέργειας μπορεί να είναι για παράδειγμα ηλιακή ή κινητική. Οι πελάτες θα στέλνουν τη θερμοκρασία του περιβάλλοντος στην οποία βρίσκονται ασύρματα στο Access Point. Στόχος είναι να επιτύχουμε την υψηλότερη βιώσιμη ρυθμοαπόδοση και παράλληλα να μικρύνουμε την καθυστέρηση των μεταδόσεων.

Λέξεις Κλειδιά

Texas Instruments, eZ430-RF2500, embedded programming, αυτόνομα ενεργειακά συστήματα, αλγόριθμοι ελέγχου, σένσορες, θερμοκρασία, ασύρματα

Abstract

The scope of this thesis is the application of algorithms in order to manage the energy in autonomous energy systems. More precisely we used the eZ430-RF2500 of Texas Instruments. The eZ430-RF2500 includes a sensor which allows it to measure the temperature of the environment. In addition these modules are able to wirelessly communicate with each other and exchange information. Setting a module in Access Point mode, while powered through the USB port of the PC, and one (or more) modules in client mode, powered through two AA batteries, by applying several algorithms, we simulated how these would react if the latter weren't powered through the batteries, but through some energy source which would transform them into autonomous energy systems. This energy source may be for example solar or kinetic. The clients will wirelessly transmit to the Access Point the temperature of the environment in which they stand. Our goal is to achieve the highest sustainable throughput and to minimize the delay in message transmission.

Key Words

Texas Instruments, eZ430-RF2500, embedded programming, autonomous energy systems, control algorithms, sensors, temperature, wireless

Ευχαριστίες

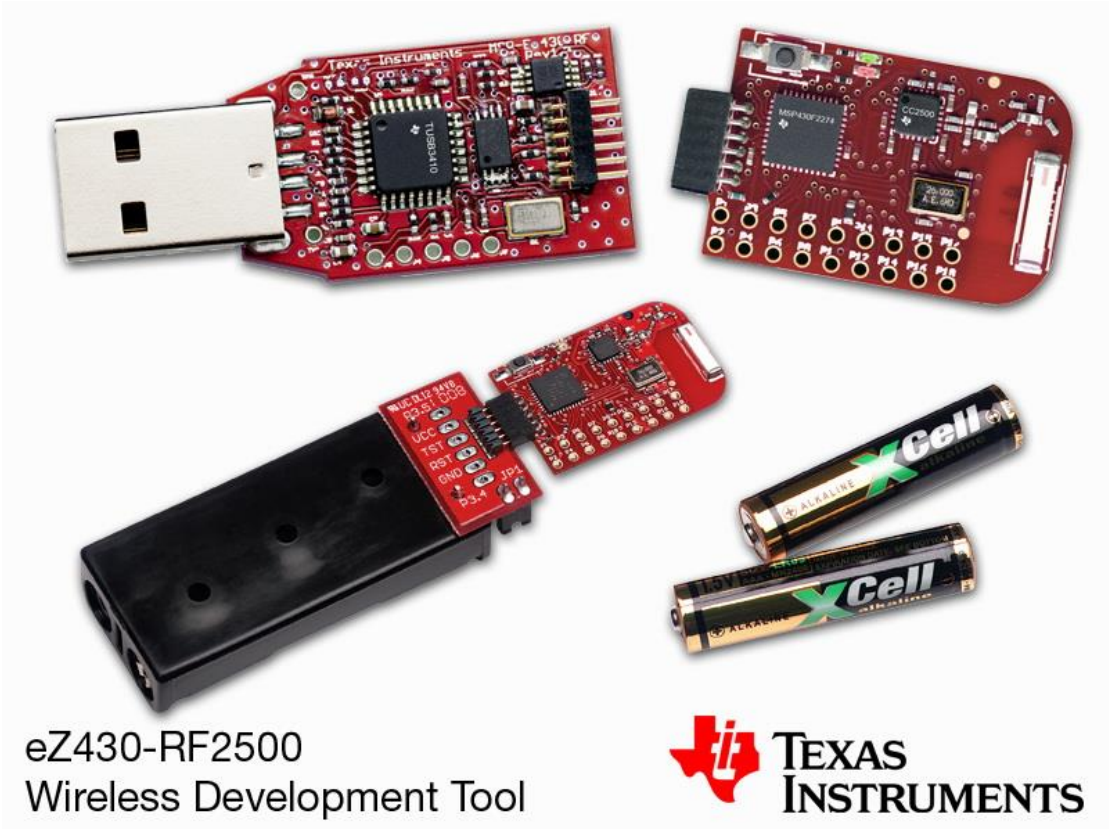
Θα ήθελα να ευχαριστήσω τον κ. Γεώργιο Στασινόπουλο καθηγητή του Εθνικού Μετσόβιου Πολυτεχνείου καθώς επίσης και τον κύριο Δημήτριο Βέργαδο για την πολύτιμη βοήθειά τους.

Περιεχόμενα

1. Εισαγωγή

Το eZ430-RF2500 της Texas Instruments είναι ένα ασύρματο module το οποίο με το κατάλληλο software μπορεί να επιτεύξει την επικοινωνία δύο ή περισσοτέρων eZ430-RF2500T, και μέσω της επικοινωνίας αυτής να μεταφέρονται διάφορες πληροφορίες (όπως η θερμοκρασία του περιβάλλοντος και άλλα). Η επικοινωνία αυτή επιτυγχάνεται θέτοντας το ένα eZ430-RF2500 σαν Access Point και τα υπόλοιπα σαν «πελάτες» (End Devices) αυτού. Το εργοστασιακό λογισμικό υποστηρίζει μέχρι οχτώ πελάτες σε ένα access point.

Ένα eZ430-RF2500 έχει μνήμη τύπου Flash (MSP430F2274) μεγέθους 32kb (όπου εκεί υπάρχει το λειτουργικό του σύστημα), 1kb μνήμης RAM και μπορεί να επικοινωνεί με ένα άλλο eZ430-RF2500 wireless μέσω της RF συχνότητας, ή με έναν υπολογιστή συνδεόμενο στην USB μέσω της UART (Universal Asynchronous Receiver/Transmitter). Επίσης φέρει επάνω του ένα push button και δύο LED, το ένα χρώματος πράσινου και το άλλο κόκκινου.



Το eZ430-RF2500 όταν λειτουργεί σαν Access Point τροφοδοτείται από την USB θύρα του υπολογιστή που είναι συνδεδεμένο, ενώ όταν λειτουργεί σαν End Device τροφοδοτείται από δύο AA μπαταρίες των 1,5Volt. Στην παρούσα αναφορά θα μελετήσουμε τρόπους λειτουργίας της παραπάνω συσκευής αν σε όλα τα End Devices έχουμε συνδέσει κάποια συσκευή συλλογής ενέργειας, ώστε να έχουμε όσο το δυνατόν πιο συχνές μεταδόσεις από τα End Devices στο Access Point ανάλογα με την αποθηκευμένη ενέργεια που θα υπάρχει στις μπαταρίες των End Devices. Η συσκευή συλλογής ενέργειας μπορεί να είναι μια συσκευή που να μετατρέπει για παράδειγμα την ηλιακή ενέργεια ή την κινητική σε ηλεκτρική.

2. Λίγα Λόγια για τον Κώδικα

Παρακάτω παρουσιάζονται τα πιο κρίσιμα σημεία του κώδικα του Access Point και του End Device.

2.1 Ο κώδικας του Access Point.

Ξεκινώντας στην main() γίνονται οι απαραίτητες αρχικοποιήσεις και αμέσως μετά το πρόγραμμα πέφτει σε ένα άπειρο loop. Σε αυτό γίνονται τα εξής:

1. **if** (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS))
   Ελέγχουμε να δούμε αν κάποιο End Device θέλει να συνδεθεί στο Access Point. Αν αυτό είναι αληθές τρέχει ο κώδικας μέσα στην if για να επιτευχθεί η επικοινωνία. Επίσης αν τα ήδη συνδεδεμένα End Devices γίνουν ίσα ή μεγαλύτερα του μέγιστου αριθμού που έχουμε θέσει ως μέγιστο αριθμό συνδέσεων τότε δεν αφήνουμε άλλο End Device να συνδεθεί.
2. **if**(sSelfMeasureSem)
   Αν η τιμή sSelfMeasureSem είναι διάφορη του μηδενός αυτό σημαίνει πως ήρθε η ώρα το Access Point να πάρει τις απαραίτητες μετρήσεις και να τις διαβιβάσει στην USB μέσω της συνάρτησης transmitDataString. Η τιμή sSelfMeasureSem γίνεται 1 χρησιμοποιώντας interrupts που καθορίζονται από ένα timer. Αυτό γίνεται μέσω της παρακάτω συνάρτησης.
   ```
   #pragma vector=TIMERB0_VECTOR
   __interrupt void Timer_B (void){
      sSelfMeasureSem = 1;
   }
   ```
3. **if** (sPeerFrameSem)
   Ελέγχουμε αν έχουμε παραλάβει κάποιο «πακέτο» από κάποιο End Device. Αν αυτό είναι αληθές επεξεργαζόμαστε το πακέτο για να το στείλουμε στην USB μέσω της συνάρτησης transmitData.

2.2 Ο κώδικας του End Device.

Ξεκινώντας στην main() γίνονται οι απαραίτητες αρχικοποιήσεις. Στο τέλος της main() καλείται η linkTo() όπου προσπαθεί να συνδεθεί το End Device σε κάποιο Access Point μέσω της συνάρτησης SMPL_Link. Αφού το επιτύχει το πρόγραμμα πέφτει σε ένα άπειρο loop. Σε αυτό γίνονται τα εξής:

1. Μετράται η θερμοκρασία του περιβάλλοντος καθώς επίσης και το voltage της μπαταρίας και στέλνεται το μήνυμα στο Access Point μέσω της συνάρτησης SMPL_Send.
2. Το End Device πέφτει σε sleep mode για ορισμένο χρόνο μέσω της συνάρτησης transmit_time_delay. Μόλις περάσει αυτός ο χρόνος η διαδικασία επαναλαμβάνεται.

Αξίζει να σημειώσουμε πως η συχνότητα των μεταδόσεων μεταβάλλεται πατώντας το push button. Πατώντας αυτό το κουμπί δημιουργείται ένα interrupt στον κώδικα μέσω της συνάρτησης #pragma vector=PORT1_VECTOR __interrupt void Port_1 η οποία αλλάζει την μεταβλητή timer_state που με τη σειρά της αλλάζει την μεταβλητή change_mode μέσω της συνάρτησης display_mode.

3. Θεωρητικό Υπόβαθρο

Η συλλογή ενέργειας στους ασύρματους sensors αναμένεται να βελτιώσει την περιβαλλοντική επίδραση που δημιουργούν οι sensors μειώνοντας την ρύπανση του περιβάλλοντος από την ανάγκη χρήσης και αντικατάστασης μπαταριών μέσω της αυτόνομης λειτουργίας. Εντούτοις, η χρήση των συσκευών συλλογής ενέργειας ως πηγές ενέργειας επιβάλλει περιορισμούς στην κατανάλωση ενέργειας των αισθητήρων. Θα παρουσιάσουμε έναν μηχανισμό όπου θα μεταβάλλουμε δυναμικά το duty cycle των μεταδόσεων, έτσι ώστε η ενέργεια που εισέρχεται στο σύστημα μέσω της συσκευής συλλογής ενέργειας να μπορεί να χρησιμοποιηθεί σε όσο δυνατόν πιο υψηλό επίπεδο.

3.1 Ιδέες και Μέθοδοι

Παρακάτω παρουσιάζονται τεχνικές για να ελέγξουμε το χρόνο των μεταδόσεων σε αυτόνομα ενεργειακά συστήματα τέτοιου τύπου χρησιμοποιώντας μια συσκευή συλλογής κινητικής ενέργειας (που θα μετατρέπεται σε ηλεκτρική).

1. Μπορούμε να χρησιμοποιήσουμε ένα κύκλωμα που θα αποτελείται από ένα πιεζο-ηλεκτρικό module, που χρησιμοποιείται για την εξαγωγή της ενέργειας διαμέσου των δονήσεων, ένα πυκνωτής, και ένα control-module. Το control-module θα μετράει συνεχώς την τάση στον πυκνωτή και μόλις αυτή περάσει μια μέγιστη τιμή Vmax, θα ανοίγει ο διακόπτης και θα τροφοδοτείται το φορτίο μέχρι η τάση να γίνει λιγότερη από μια τιμή Vmin όπου τότε η παροχή της ενέργειας θα διακόπτεται έως ότου πάλι ο πυκνωτής θα έχει πάλι τάση μεγαλύτερη της τιμής Vmax.
2. Εναλλακτική αυτού είναι να συλλέγεται ενέργεια παρουσία μπαταριών ιόντων Λιθίου (Li-Ion) που προορίζονται για να βελτιώσουν το χρόνο φόρτισης τω μπαταριών τέτοιου είδους. Χρειάζεται προσοχή στην τάση και το ρεύμα φόρτισης των μπαταριών για να αποφύγουμε την καταστροφή των κελιών της μπαταρίας. Συγκεκριμένα η τάση και το ρεύμα φόρτισης πρέπει να είναι μικρά όταν η τάση της μπαταρίας είναι μικρή. Στο τελικό στάδιο φόρτισης πρέπει η τάση να είναι μεγαλύτερη της τάσης της μπαταρίας λόγω της εσωτερικής σύνθετης αντίστασης της μπαταρίας. Επομένως οι προδιαγραφές του κυκλώματος φόρτισης της μπαταρίας πρέπει να είναι τέτοιες ώστε να αποφύγουμε την καταστροφή της μπαταρίας.
3. Μια εναλλακτική για να αποφύγουμε τα προβλήματα του προηγούμενου παραδείγματος, είναι να χρησιμοποιήσουμε δύο μπαταρίες, όπου η μία θα χρησιμοποιείται για να τροφοδοτεί το φορτίο και η άλλη να φορτίζει από τη συσκευή συλλογής ενέργειας.
4. Μια όμοια προσέγγιση του προηγούμενου παραδείγματος είναι αντί για δύο μπαταρίες να χρησιμοποιήσουμε μια μπαταρία και έναν πυκνωτή. Ο πυκνωτής βοηθάει στο να εξομαληνθεί το ρεύμα φορτίσεως, ενώ η μπαταρία μεγαλώνει τη «ζωή» του συστήματος τις «σκοτεινες περιόδους» (περίοδοι που δεν έχουμε εισερχόμενη ενέργεια από την συσκευή συλλογής ενέργειας).
5. Άλλη μέθοδος είναι να εκτιμίσουμε τους αλγόριθμους διαχείρησης ενέργειας, προτείνοντας έναν προσαρμοστικό αλγόριθμο για τον έλεγχο του duty-cycle των αυτόνομων αισθητήρων. Αυτός ο αλγόριθμος θα χρησιμοποιεί προηγούμενα δεδομένα για να υπολογίσει το βέλτιστο duty-cycle, εξετάζοντας την ενέργεια και ένα max και ένα min όριο του duty-cycle. Ο αλγόριθμος θα συγκρίνει την εκτιμόμενη στάθμη ενέργειας με την μετρούμενη για να διορθώσει τα εκτιμόμενα λάθη και να διατηρήσει την ενεργειακή ουδέτερη λειτουργία.
6. Μια απλούστερη μέθοδος προσαρμογής του duty-cycle είναι να κρατήσουμε τη στάθμη της μπαταρίας όσο πιο κοντά γίνεται στην αρχική της στάθμη μετά από κάθε κύκλο. Αν αυτή η υπόθεση επιτευχθεί η λειτουργία θα είναι «ενεργειακά ουδέτερη», αφού δεν θα

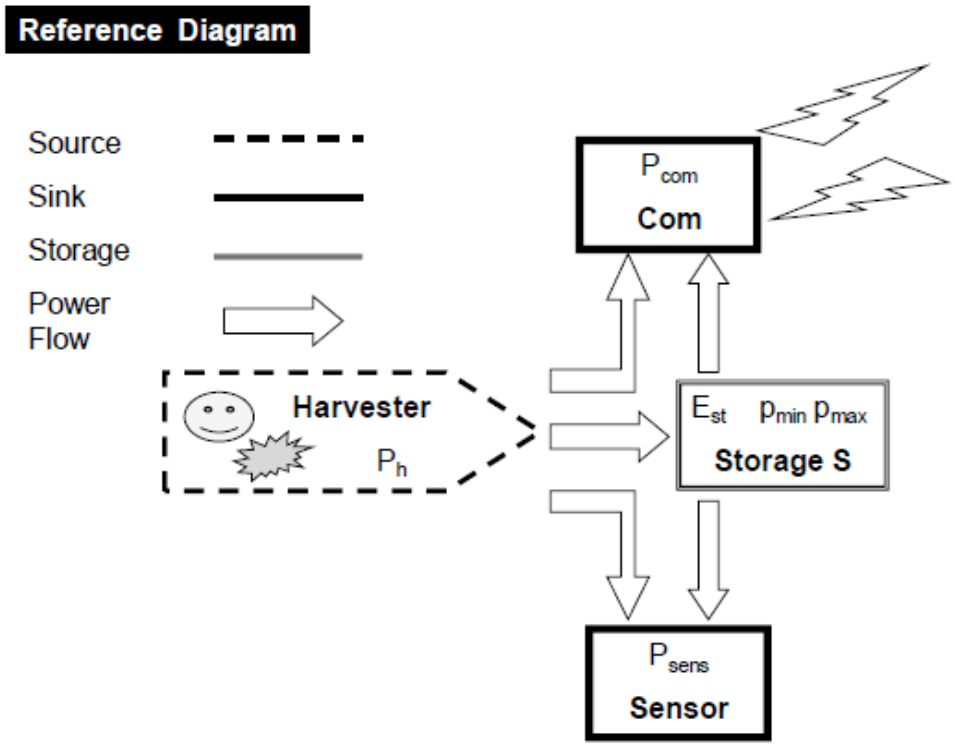
ξοδεύεται περισσότερη ενέργεια από αυτή που συλλέχθηκε, και η μέγιστη απόδοση έχει επιτευχθεί αφού η ενεργειακή στάθμη δεν μπορεί να αυξηθεί.

7. Μια διαφορετική προσέγγιση είναι ο αισθητήρας να αποθηκεύει την ενέργεια που συλλέγεται, και να κάνει την μετάδοση όταν η ενέργεια είναι ικανοποιητική.

Στόχος είναι να επιτύχουμε την υψηλότερη βιώσιμη ρυθμοαπόδοση και παράλληλα να μικρύνουμε την καθυστέρηση των μεταδόσεων.

3.2 Μακροσκοπικοί Περιορισμοί

Στο παρακάτω σχήμα βλέπουμε την εκτίμηση του φορτίου που αποτελείται από τους αισθητήρες και τα τμήματα επικοινωνίας.



Το «αισθητήριο» μέρος (μέτρηση θερμοκρασίας) πρέπει να εκτελείται περιοδικά αλλά είναι σχετικά χαμηλής ισχύος Psens. Διαβιβάζοντας τα αισθητήρια αποτελέσματα, είτε ακατέργαστα είτε με κάποια τοπική επεξεργασία καταναλώνουμε σχετικά υψηλή ενέργεια Pcom. Η βασική ιδέα είναι να προσαρμόσουμε –δοθέντος συγκεκριμένης αστάθειας της εισερχόμενης ενέργειας Ph– το duty-cycle του «Com» μέρους λαμβάνοντας υπόψη τη στάθμη της αποθηκευμένης ενέργειας.

Ο δείκτης της απόδοσης έχει δύο συστατικά. Το ένα είναι ότι πρέπει να ελαχιστοποιηθούν τα χρονικά διαστήματα όπου μεταδόσεις δεν μπορούν να ικανοποιηθούν. Επίσης η κατανάλωση πρέπει να επισπευτεί προκειμένου να μειωθεί η αποθήκευση και να υπάρχει έτσι η ικανότητα συλλογής ενέργειας στις αναμενόμενες εισερχόμενες εκρήξεις εισερχόμενης ενέργειας. Διαφορετικά όταν ασταθώς εμφανιστεί η ευκαιρία να συγκομιστεί η ενέργεια θα είναι αδύνατο. Κατά συνέπεια ο δείκτης της απόδοσης έχει και τις δύο πτυχές.

Μερικοί πρώτοι μακροσκοπικοί περιορισμοί που αφορούν το Ph, Psense και Pcom, με την πάροδο του χρόνου οδηγούν σε χρήσιμους ορισμούς. Το σύστημα είναι «περιορισμένος συλλέκτης ενέργειας» αν σε οποιαδήποτε στιγμή Ph > Psense + Pcom, μια χωρίς ενδιαφέρον περίπτωση όπου καμία αποθήκευση δεν απαιτείται. Αν επιτρέψουμε στο

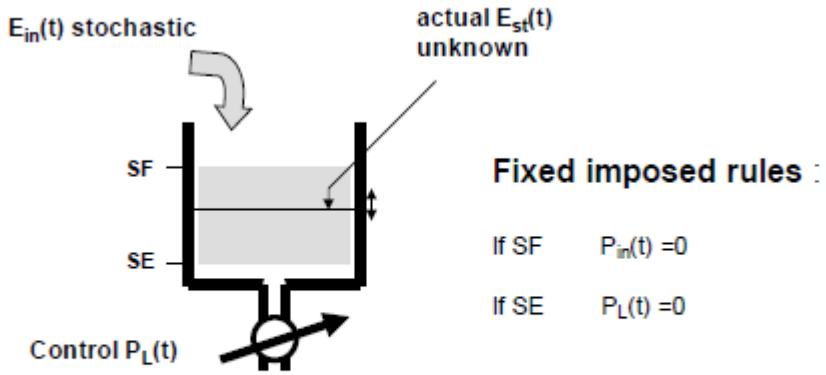
σύστημα να κρατήσει ενέργεια αλλά όχι πάνω από ένα περιορισμένο μικρό μέσο διάστημα, τότε αυτό οδηγεί μαζί με την ύπαρξη μιας σχετικά μικρής αποθηκευτικής ενέργειας Est σε ένα «περιορισμένης Ισχύος» σύστημα. Σε αυτή την περίπτωση ένας πυκνωτής θα ήταν προτιμότερος από την χρήση ηλεκτροχημικών μπαταριών για δύο λόγους.

1. Ιδανικός για αποθήκευση περιορισμένης και βραχυπρόθεσμης ενέργειας, και
2. Πρακτικά απεριόριστους κύκλους φόρτισης/εκφόρτισης

Με σημαντική μεγάλη χωρητικότητα βασισμένη στην ηλεκτροχημική αποθήκευση, φθάνουμε σε ένα σύστημα «περιορισμένης ενέργειας», όπου σε μεγάλα χρονικά διαστήματα Ph > Psense + Pcom, αλλά μετά από μερικά σύντομα διαστήματα Ph << Psense + Pcom. Τελικά μια ικανοποιητική περίπτωση, θα ήταν η «μικτή περίπτωση». Εδώ έχουμε μια «Περιορισμένη Ισχύ» σχέση μεταξύ της συσκευής συλλογής ενέργειας και των αισθητήρων και μια σχέση «περιορισμένης ενέργειας» μεταξύ της συσκευής συλλογής ενέργειας και της COM μόνο. Με άλλα λόγια η συσκευή συλλογής ενέργειας συνεχώς ικανοποιεί τις ανάγκες των αισθητήρων και, όταν μπορεί φυλάσσει ενέργεια για την COM.

3.3 Σημεία Μέτρησης και Ελέγχου

Ένα πρόβλημα ελέγχου της στάθμης μίας δεξαμενής έχει να κάνει με τους περιορισμούς του εκάστοτε προβλήματος. Η στάθμη της δεξαμενής του παρακάτω σχήματος αναπαριστά την στάθμη της ενέργειας των μπαταριών.



What can we measure:

time interval Δt between transitions SF / SE

Η δεξαμενή τροφοδοτείται με μία στοχαστική διαδικασία της ενέργειας που συλλέγεται και αδειάζει από την ντετερμινιστική συνάρτηση χρόνου της ενέργειας PL που καταναλώνεται από το φορτίο. Στο PL συνυπολογίζουμε το Psense και το Pcom, ή μόνο το Pcom αφού Pcom>>Psense. Η χρονική συνάρτηση Pin της εισερχόμενης ροής ενέργειας είναι ακανόνιστη αφού αυτή προέρχεται ακανόνιστα από την φύση.

Για την Pin θεωρούμε δύο εκδοχές.

Εκδοχή 1: Υποθέτουμε πως δεν γνωρίζουμε τίποτα όσο αν αφορά την Pin. Πλήρη άγνοια του πότε και σε ποια ποσότητα θα μας έρθει αυτή η ενέργεια.

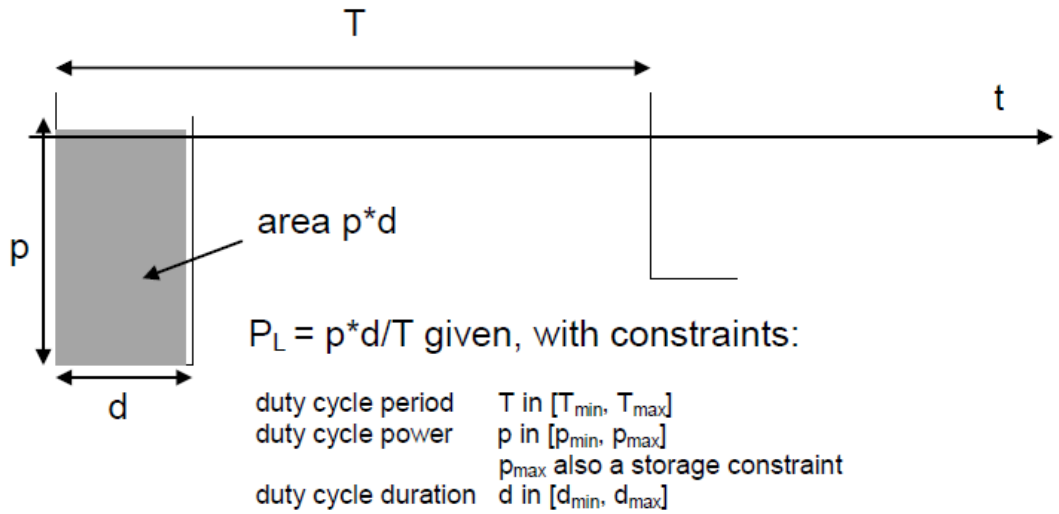
Εκδοχή 2: Υποθέτουμε περιορισμένη εκτίμηση του $\int P_{in} tdt$ σε ένα διάστημα Δt στην μορφή kE_b, όπου E_b είναι ένα κβάντο της ενέργειας που μεταφέρεται από το συλλέκτη ενέργειας στις μπαταρίες. Αυτό το κβάντο δεν είναι σταθερό με τον χρόνο. Ο ακέραιος k αντιστοιχεί στον χρόνο μεταξύ κάθε μεταφοράς εισερχόμενης ενέργειας.

Στην περίπτωση του πυκνωτή η στάθμη Est μπορεί να εκτιμηθεί μετρώντας την τάση στα άκρα του. Αυτό δεν μπορεί να χρησιμοποιηθεί σε περίπτωση μπαταριών. Σε κάθε περίπτωση υποθέτουμε πως μπορούμε να εντοπίσουμε τα χρονικά στιγμιότυπα όπου η στάθμη ενέργειας είναι full (SF) ή empty (SE). Και οι δύο καταστάσεις είναι ανεπιθύμητες,

(SE) γιατί το φορτίο δεν μπορεί να εξυπηρετηθεί, (SF) γιατί δεν υπάρχει χώρος να συλλέξουμε περεταίρω ενέργεια σε περίπτωση που έχουμε μια έκρηξη ενέργειας.

Από την πλευρά του φορτίου το πρόβλημα είναι να καταφέρουμε να ταιριάξουμε την κατανάλωση με την τυχαία εισερχόμενη ενέργεια που αποθηκεύεται. Αυτό μπορεί να επιτευχθεί μεγαλώνοντας τον χρόνο που θα γίνει η κατανάλωση τόσο όσο οι απαιτήσεις της εφαρμογής μας το επιτρέπουν.

Στο παρακάτω σχήμα φαίνεται ένα γενικό αποδεκτό σχέδιο.



Ο χρονισμός του microsensor μπορεί να λάβει την τιμή T όπου αυτή θα ανήκει σε ένα διάστημα [Tmin,Tmax]. Η ενέργεια που καταναλώνεται σε κάθε περίοδο έχει μία μέση τιμή ίση με PL=p*d/T .

3.4 Πρόβλημα Ελέγχου

Υποθέτουμε πως η εισερχόμενη ενέργεια είναι μια στοχαστική μεταβλητή που αλλάζει τυχαία υπό τον έλεγχό μας. Κατά συνέπεια, προκειμένου να εξασφαλιστεί συνεχής λειτουργία, πρέπει να ελέγξουμε το ποσοστό κατανάλωσης ενέργειας σε σχέση με τον συλλέκτη ενέργειας. Εντούτοις, μια τέλεια αντιστοιχία μεταξύ του συλλέκτη ενέργειας και της κατανάλωσης δεν είναι πάντα επιθυμητή. Παρουσιάζουμε παρακάτω τρείς διαφορετικές στρατηγικές:

3.4.1 Βέλτιστη Συλλογή Ενέργειας

Η βέλτιστη στρατηγική συλλογής ενέργειας στοχεύει στην ελαχιστοποίηση του ποσού ενέργειας που χάνεται λόγω της ανεπαρκούς μεγάλης χωρητικότητας. Σε αυτήν την στρατηγική, το ποσοστό κατανάλωσης ενέργειας ακολουθεί το ποσοστό συλλογής ενέργειας, και η αποθηκευμένη ενέργεια παραμένει, όσο το δυνατόν περισσότερο, σταθερή. Η εφαρμογή αυτής της στρατηγικής είναι απλή:
Ένας συγκριτής τάσης ελέγχει το ενεργειακό επίπεδο στις μπαταρίες και όταν αυτό υπερβεί μίας προκαθορισμένης τιμής, ξεκινάει ένας κύκλος κατανάλωσης. Η κατανάλωση μειώνει το ποσό ενέργειας στις μπαταρίες, και ο κύκλος επαναλαμβάνεται. Δεδομένου ότι το επίπεδο αποθήκευσης δεν υπερβαίνει ποτέ αυτό το προκαθορισμένο επίπεδο, δεν υπάρχει καμία ενεργειακή απώλεια λόγω της πεπερασμένης χωρητικότητας της μπαταρίας. Επιπλέον, δεδομένου ότι όλες οι καταναλώσεις αρχίζουν σε υψηλής ενέργειας επίπεδο, δεν υπάρχει καμία απραγματοποίητη κατανάλωση. Αυτή η στρατηγική κατανάλωσης οδηγεί στα απρόβλεπτα διαστήματα μεταξύ των καταναλώσεων ενέργειας, και μια υψηλή

μεταβλητότητα. Αν έχουμε μειωμένη συλλογή ενέργειας, δεν μπορούμε να διατηρήσουμε ούτε μια κυμαινόμενη περιοδικότητα. Αυτή η στρατηγική χρησιμοποιείται μόνο ως κριτήριο για άλλα τα δύο.

3.4.2 Βέλτιστη Κατανάλωση

Η βέλτιστη στρατηγική κατανάλωσης στοχεύει στη διατήρηση του duty-cycle όσο το δυνατόν πιο σταθερού. Το duty-cycle υπολογίζεται βασισμένο στη μέση ενέργεια που συλλέγεται μακροπρόθεσμα. Αυτό είναι μια ιδανική περίπτωση υπό την έννοια ότι η μελλοντική ενέργεια που συλλέγεται δεν μπορεί να προβλεφθεί πάντα ακριβώς. Αφ' ενός αντιπροσωπεύει ένα σταθερό σενάριο duty-cycle. Το ποσοστό της ενέργειας που συλλέγεται ποικίλλει με το χρόνο. Έτσι, κατά τη διάρκεια των χαμηλών περιόδων συλλογής ενέργειας, μερικές προγραμματισμένες καταναλώσεις μπορεί να μην είναι σε θέση να πραγματοποιηθούν, λόγω της ανεπαρκούς ενέργειας. Επιπλέον, κατά τη διάρκεια των υψηλών περιόδων συλλογής ενέργειας, η ενέργεια θα χαθεί λόγω της πεπερασμένης χωρητικότητας της μπαταρίας.

3.4.3 Προσαρμοστική Κατανάλωση

Η προσαρμοστική στρατηγική κατανάλωσης είναι μια εναλλακτική λύση της ιδανικής βέλτιστης κατανάλωσης. Ο στόχος της στρατηγικής είναι να κρατηθεί το duty-cycle εύλογα σταθερό, ενώ συνεχώς διορθώνουμε την εκτίμηση του μέσου ποσοστού συλλογής ενέργειας. Ο στόχος είναι να σταθεροποιηθεί το duty-cycle χωρίς να έχουμε μεγάλα διαστήματα όπου το επίπεδο αποθήκευσης είναι πλήρες ή άδειο. Επομένως η ενέργεια που συλλέγεται πρέπει να αλλάζει σύμφωνα με την κατανάλωση. Δεδομένου ότι η ενέργεια που συλλέγεται αλλάζει συνεχώς, το duty-cycle πρέπει επίσης να αλλάξει, αλλά όσο το δυνατόν σε ένα χαμηλότερο το ποσοστό.

Σε κάθε διάστημα Δt, έχουμε ότι $\Delta E_{st} = \int P_{in} dt - \int P_L dt$ λόγω της διατήρησης φορτίου, όπου ΔE_{st} είναι η διαφορά της αποθηκευμένης ενέργειας, Pin είναι η συνάρτηση χρόνου της εισερχόμενης ενέργειας, PL είναι η συνάρτηση χρόνου της καταναλισκόμενης ενέργειας και Δt το διάστημα της ολοκλήρωσης. Προσοχή: Το Pin δεν είναι πάντα ίσο με την εισερχόμενη ενέργεια που προέρχεται από τη συλλογή ενέργειας λόγω της πεπερασμένης χωρητικότητας της μπαταρίας. Αν η ενέργεια που συλλέγεται Ph είναι σταθερή σε ένα διάστημα Δt, τότε $\int P_{in} dt = a_h P_h \Delta t$, όπου a_h είναι το ποσοστό του χρόνου του Δt που οι μπαταρίες δεν είναι γεμάτες.

Η ενέργεια που καταναλώνεται είναι $\int P_L dt = nE_L$, όπου $E_L=P_LT=pd$ και $n \approx \Delta t/T$, όπου n είναι ο αριθμός των καταναλώσεων που πραγματοποιήθηκαν στον διάστημα Δt. Συνδυάζοντας τις παραπάνω εξισώσεις έχουμε ότι $\Delta E_{st}/E_L \approx (a_h P_h/E_L)\Delta t - n$.

Η βέλτιστη λειτουργία επιτυγχάνεται όταν $a_h=1$, δηλαδή όταν οι μπαταρίες δεν είναι ποτέ γεμάτες, και $T \approx E_L/P_h = T_h$. Κατά συνέπεια, το σημείο κλειδί στη μέθοδό μας είναι να καθοριστεί βέλτιστα ένα μεταβλητό T_h, που αντιστοιχεί στο σταθερό ιδανικό duty-cycle περιόδου T. Σύμφωνα με τη δραστηριότητας συλλογής ενέργειας, ένα κυμαινόμενο T_h θα ικανοποιήσει τα κριτήρια της λειτουργίας «ουδέτερης ενέργειας». Κατά συνέπεια προτείνεται η ακόλουθη επαναληπτική διαδικασία:

- Μετά από ένα Storage Full (SF), ή Storage Empty (SE) γεγονός, υπολογίζουμε το T_h^* $T_h^* = a_h \Delta t/(\Delta E_{st}/E_L + n)$ και
- $T_h^{(n)} = \alpha\, T_h^* + (1-\alpha) T_h^{(n-1)}$, όπου η παράμετρος α καθορίζει πόσο γρήγορα προσαρμοζόμαστε στους μεταβαλλόμενους όρους.

Προκειμένου να εφαρμοστεί ο ανωτέρω μηχανισμός, πρέπει να υπολογίσουμε την αναλογία $\Delta E_{st}/E_L$. Αυτό μπορεί να επιτευχθεί εύκολα χωρίς απευθείας μέτρηση της αποθηκευμένης ενέργειας, ή της ενέργειας που καταναλώθηκε, κάνοντας εκτιμήσεις όταν οι στάθμες ενέργειας είναι γνωστές. Οι πιο κατάλληλες στάθμες είναι αυτές όπου έχουμε Storage Full (SF) και Storage Empty (SE). Για παράδειγμα αν έχουμε κάποια μετάβαση από

SF σε SF ή SE σε SE έχουμε $\Delta E_{st}/E_L=0$ αφού η αρχική και τελική κατάσταση έχουν την ίδια ενέργεια. Επιπλέον για μεταβάσεις από SF σε SE ή SE σε SF έχουμε $\Delta E_{st}/E_L=\pm S/E_L$, όπου S είναι η χωρητικότητα του μέσου αποθήκευσης ενέργειας. Ο λόγος S/E_L αλλάζει αργά με το χρόνο σε περίπτωση μπαταριών και είναι πρακτικά σταθερός στην περίπτωση πυκνωτή. Αν είναι απαραίτητο μπορεί να εκτιμηθεί συγκρίνοντας τις τιμές του T_h* από διαφορετικές διαδοχές γεγονότων.

4. Εφαρμογή στο eZ430-RF2500

Θα εφαρμόσουμε τον αλγόριθμο της παραγράφου 3.4.3 στο eZ430-RF500. Η κατανομή της εισερχόμενης ενέργειας είναι άγνωστη αφού αυτή προέρχεται ακανόνιστα από τη φύση. Αρχικά θα υποθέσουμε πως η κατανομή αυτή είναι μία ομοιόμορφη κατανομή στο διάστημα [0,2]. Λόγω της συγκεκριμένης κατανομής κατά μέση τιμή αυτή θα έχει τιμή 1. Επίσης όσο αν αφορά την κατανάλωση υποθέτουμε πως σε κάθε κατανάλωση ξοδεύουμε ενέργεια ίση με 1. Θα δημιουργήσουμε μια εικονική μπαταρία με χωρητικότητα 20. Άρα σε κάθε μετάδοση που πραγματοποιούμε θα καταναλώνουμε ενέργεια ίση με 1 ενώ παράλληλα θα δεχόμαστε ενέργεια η οποία θα συλλέγεται. Ο τύπος της εισερχόμενης ενέργειας θα έχει την μορφή Eh=λxT, όπου λ μία μεταβλητή που καθορίζει το ποσοστό της ενέργειας που συλλέγεται στη μονάδα του χρόνου, x η τυχαία μεταβλητή ομοιόμορφης κατανομής που λαμβάνει τιμές στο διάστημα [0,2] και T το χρονικό διάστημα που θεωρούμε πως συλλέγεται ενέργεια. Παρακάτω παρουσιάζουμε τον κώδικα για την συγκεκριμένη εφαρμογή. Αρχικά σε μία απλοποιημένη μορφή και μετά σε μια πιο σύνθετη και ρεαλιστική υλοποίηση.

4.1 Απλοποιημένη Μορφή

Στην απλοποιημένη μορφή τα End Devices θα παίρνουν μετρήσεις για το level της μπαταρίας αμέσως πριν από κάθε μετάδοση, και θα «μαντεύουμε» στο περίπου πότε θα είχαμε το level της μπαταρίας γεμάτο. Επίσης δεν θα περιμένουμε για κάποιο Storage Full ή Storage Empty event και το duty cycle θα προσαρμόζεται αμέσως πριν από κάποια μετάδοση και θα είναι διαφορετικό κάθε φορά.

4.1.1 Ο Κώδικας του Access Point

Οι αλλαγές που πρέπει να γίνουν στον κώδικα στου Access Point δεν είναι πολλές. Αφορούν την διαφορετική επεξεργασία του πακέτου που παραλαμβάνει το Access Point από τα End Devices. Πιο συγκεκριμένα, τα End Devices θα στέλνουν μια παραπάνω πληροφορία στο Access Point που αυτή θα είναι το level της virtual μπαταρίας τους. Η μέγιστη χωρητικότητα της μπαταρίας όπως προείπαμε θα είναι ίση με 20. Θα χρησιμοποιήσουμε και δύο δεκαδικά ψηφία για μεγαλύτερη ακρίβεια. Επομένως προγραμματιστικά θεωρούμε το level της μπαταρίας ίσο με 2000 (όπου διαιρούμενο με το 100 θα βλέπουμε το πραγματικό level της μπαταρίας). Η συνάρτηση που πρέπει να αλλάξουμε στον κώδικα του Access Point είναι η συνάρτηση transmitDataString όπου δέχεται σαν όρισμά της το υπό επεξεργασία πακέτο, το επεξεργάζεται και στέλνει τα δεδομένα στην USB του υπολογιστή. Η τροποποιημένη συνάρτηση παρουσιάζεται παρακάτω:

```
void transmitDataString(char addr[4],char rssi[3], char
msg[MESSAGE_LENGTH] )
{
  /*GArm Start*/
  char temp_level[]= {"XXXX"};
  char temp_ah[]= {"XXX"};
  /*GArm End*/
```

```
  char temp_string[] = {" XX.XC"};

  int temp = msg[0] + (msg[1]<<8);
  int tcnt=msg[4] + (msg[5]<<8);
  if( !degCMode )
  {
    temp = (int) ((((float)temp)*1.8)+320);
    temp_string[5] = 'F';
  }
  if( temp < 0 )
  {
    temp_string[0] = '-';
    temp = temp * -1;
  }
  else if( ((temp/1000)%10) != 0 )
  {
    temp_string[0] = '0'+((temp/1000)%10);
  }
  temp_string[4] = '0'+(temp%10);
  temp_string[2] = '0'+((temp/10)%10);
  temp_string[1] = '0'+((temp/100)%10);

  /*GArm Start*/
  temp = msg[7] + (msg[8]<<8);
  temp_level[3] = '0'+(temp%10);
  temp_level[2] = '0'+((temp/10)%10);
  temp_level[1] = '0'+((temp/100)%10);
  temp_level[0] = '0'+((temp/1000)%10);

  temp = msg[9]+1;
  temp_ah[2] = '0'+(temp%10);
  temp_ah[1] = '0'+((temp/10)%10);
  temp_ah[0] = '0'+((temp/100)%10);
  /*GArm End*/
  if( verboseMode )
  {
      /*GArm Start*/
    char output_verbose[] = {"\r\nNode:XXXX,Temp:-
XX.XC,Battery:X.XV,Strength:XXX%,RE:XXXXXXX,Lvl:XX.XX,ah=X.XX"};
      /*GArm End*/
    output_verbose[46] = rssi[2];
    output_verbose[47] = rssi[1];
    output_verbose[48] = rssi[0];

    output_verbose[17] = temp_string[0];
    output_verbose[18] = temp_string[1];
    output_verbose[19] = temp_string[2];
    output_verbose[20] = temp_string[3];
    output_verbose[21] = temp_string[4];
    output_verbose[22] = temp_string[5];

    output_verbose[32] = '0'+(msg[2]/10)%10;
    output_verbose[34] = '0'+(msg[2]%10);
    output_verbose[7] = addr[0];
    output_verbose[8] = addr[1];
    output_verbose[9] = addr[2];
    output_verbose[10] = addr[3];
    output_verbose[54]= '0'+(msg[3]/100)%10;
    output_verbose[55] = '0'+(msg[3]/10)%10;
    output_verbose[56] = '0'+(msg[3]%10);
    output_verbose[57] = '0'+(msg[6]%10);
```

```
    output_verbose[58] = '0'+((tcnt/100)%10);
    output_verbose[59] = '0'+((tcnt/10)%10);
      output_verbose[60] = '0'+(tcnt%10);
      /*GArm Start*/
      output_verbose[66] = temp_level[0];
      output_verbose[67] = temp_level[1];
      output_verbose[69] = temp_level[2];
      output_verbose[70] = temp_level[3];

      output_verbose[75] = temp_ah[0];
      output_verbose[77] = temp_ah[1];
      output_verbose[78] = temp_ah[2];
    /*GArm End*/
    TXString(output_verbose, sizeof output_verbose );
  }
  else
  {
    char output_short[] = {"\r\n$ADDR,-XX.XC,V.C,RSI,N#"};

    output_short[19] = rssi[2];
    output_short[20] = rssi[1];
    output_short[21] = rssi[0];

    output_short[8] = temp_string[0];
    output_short[9] = temp_string[1];
    output_short[10] = temp_string[2];
    output_short[11] = temp_string[3];
    output_short[12] = temp_string[4];
    output_short[13] = temp_string[5];

    output_short[15] = '0'+(msg[2]/10)%10;
    output_short[17] = '0'+(msg[2]%10);
    output_short[3] = addr[0];
    output_short[4] = addr[1];
    output_short[5] = addr[2];
    output_short[6] = addr[3];

    TXString(output_short, sizeof output_short );
  }
}
```

Επίσης επειδή η ίδια συνάρτηση χρησιμοποιείται για να προωθήσει το Access Point και το δικό του μήνυμα που παράγει μέσα στην main κάνουμε δύο συμβάσεις, ότι το virtual level της υποτιθέμενης μπαταρίας του Access Point είναι πάντα 20, και ότι το ποσοστό του χρόνου που φορτίζει η μπαταρία είναι ίσο με 1, και προσθέτουμε τις εξής γραμμές στην σύνθεση του μηνύματος του Access Point που βρίσκεται μέσα στην main και μέσα στην συνθήκη if(sSelfMeasureSem).

```
msg[7]=level&0xFF;
msg[8]=(level>>8)&0xFF;
msg[9]=ah;
```

Εννοείται πως πιο πάνω έχουμε δηλώσει και αρχικοποιήσει τις μεταβλητές level και ah (που είναι τύπου int). Αρχικοποιούμε την μεταβλητή level=2000. Όσο αν αφορά την μεταβλητή ah, επειδή μπορούμε να στείλουμε από τον εκάστοτε πελάτη στο Access Point ακόμα έναν διψήφιο αριθμό ορίζουμε (και μέσω του κώδικα του End Device) το εύρος του ah στο διάστημα [0,99] και λόγω του ότι προσθέτουμε στο msg[9] μια μονάδα αυτό διαμορφώνεται στο εύρος [1,100]. Δηλαδή το ποσοστιαίο εύρος θα λαμβάνει τιμές (χωρίς ιδιαίτερο σφάλμα λόγω του ότι αγνοούμε το μηδέν) στο διάστημα [0.01 , 1.00].

4.1.2 Ο Κώδικας του End Device

Σε αυτό το σημείο θα διαμορφώσουμε τον κώδικα του End Device ώστε να παράγει τα απαραίτητα δεδομένα και να συνεργάζεται με το Access Point. Θα χρειαστεί να δηλώσουμε κάποιες μεταβλητές. Αυτές είναι οι εξής:

```
int level=2000;   //Level of the battery 20 is represented as 2000
float l=0.1;
float a=0.005;
int sec=3;
int henergy=0;
float ah=0;
int prev_level;
#define lowerlevel 500
```

Στη μεταβλητή level θα καταχωρείται το τρέχον επίπεδο της εικονικής μπαταρίας, η μεταβλητή l είναι μία μεταβλητή που καθορίζει το ποσοστό της ενέργειας που συλλέγεται στη μονάδα του χρόνου και θα χρησιμοποιηθεί στην συνάρτηση που θα υπολογίζει την ενέργεια που συλλέχθηκε, στη μεταβλητή a θα καταχωρούμε το ποσοστό που θα λαμβάνουμε υπόψη μας για τον υπολογισμό του καινούριου χρόνου της μετάδοσης της τιμής που υπολογίστηκε και το υπόλοιπο αυτού του ποσοστού θα προέρχεται από το χρόνο της προηγούμενης μετάδοσης.

Ο υπολογισμός της ενέργειας που συλλέχθηκε όπως προείπαμε θα υπολογίζεται από τον τύπο Eh=λxT, όπου λ μία μεταβλητή που καθορίζει το ποσοστό της ενέργειας που συλλέγεται στη μονάδα του χρόνου, x η τυχαία μεταβλητή ομοιόμορφης κατανομής που λαμβάνει τιμές στο διάστημα [0,2] και T το χρονικό διάστημα που πέρασε από την προηγούμενη μετάδοση. Προγραμματιστικά το παραπάνω υπολογίζεται από την παρακάτω συνάρτηση:

```
int HarvestEnergy(){
	unsigned int temp;
	//harvest energy range should be for 0 to 200
	temp=rand()%201;
	return (int)(l*(float)temp*(float)sec);
}
```

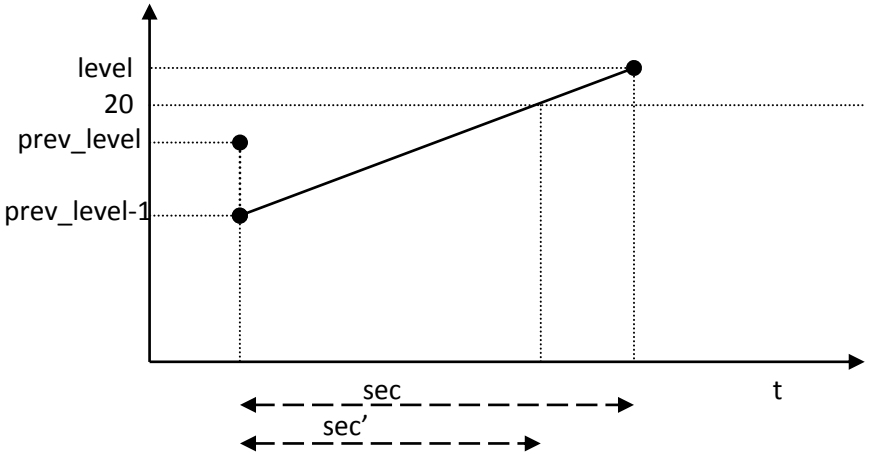
Επίσης ο υπολογισμός της επόμενης μετάδοσης όπως είπαμε στην παράγραφο 3.4.3 θα δίνεται από τους εξής τύπους: $T_h^*=a_h\Delta t/(\Delta E_{st}/E_L+n)$ και $T_h^{(n)}= \alpha T_h^* + (1-\alpha)T_h^{(n-1)}$. Έδω να σημειώσουμε πως στην συγκεκριμένη υλοποίηση δεν θα περιμένουμε να συναντήσουμε ένα Storage Empty ή Storage Full event άλλα η διαδικασία θα επαναλαμβάνεται σε κάθε μετάδοση. Κατά συνέπεια το Δt θα ταυτίζεται με το T και το n θα είναι ίσο με 1 αφού κατά το διάστημα T=Δt θα έχουμε μόνο μία μετάδοση. Επίσης θέτουμε σαν ελάχιστη τιμή του χρόνου της νέας μετάδοσης το 1 δευτερόλεπτο και μέγιστη τα 99 δευτερόλεπτα. Η συνάρτηση που θα υπολογίζει το χρόνο της νέας μετάδοσης θα είναι η παρακάτω:

```
void calc_new_trassmition(void){
	float Th_temp, temp;
	if(level>=2000){
		ah=(float)(2000-(prev_level-100))/(float)henergy;
		level=2000;
	}
	else{
		ah=1.0;
	}
	if(level<=0){
		level=0;
	}
	temp=(float)(henergy-100)/100.0;
	if(temp==-1.0) temp=-0.99;

	Th_temp=ah*(float)sec/(temp + 1.0);
```

```
        temp= ((float)a*Th_temp + (1.0-(float)a)*(float)sec);
        //strogilopoiisi
        if((float)((int)temp)+0.5 > temp)
              sec=(int)temp+1;
        else
              sec=(int)temp;
        /*In case the new transmit has to be right now i.e. sec=0
         * then all of the next transmitions will be zero in our
simulation.
         * If this happens, harvest energy will be zero all the time.
         * So we set the minimum transmit delay = 1
         */
        if(sec<1) sec=1;
        //Maximum transmit time 99sec
        if(sec>99) sec=99;
}
```

Παραπάνω το ah στην περίπτωση που η μπαταρία δεν φορτίζει καθ όλη την διάρκεια του χρόνου μεταξύ των δύο μεταδόσεων υπολογίζεται ως εξής. Έστω ότι λίγο πριν την μετάδοση το level της μπαταρίας έχει την τιμή prev_level και αμέσως πριν την επόμενη μετάδοση έχει τιμή level η οποία είναι μεγαλύτερη του 20. Επίσης αμέσως μετά την πρώτη μετάδοση το level της μπαταρίας γίνεται prev_level-1 λόγω της κατανάλωσης της ενέργειας που χρειάζεται για να γίνει η μετάδοση. Θεωρούμε λόγω του ότι δεν γνωρίζουμε στο ενδιάμεσο διάστημα την κατανομή της εισερχόμενης ενέργειας, πως η εισερχόμενη ενέργεια είναι γραμμική ως προς τον χρόνο.



Επίσης έχουμε πως η εισερχόμενη ενέργεια (henergy) ισούται με henergy=level-(prev_level-1). Επομένως αφού σε χρόνο sec έχουμε εισερχόμενη ενέργεια ίση με henergy, υπολογίζουμε τον χρόνο που χρειάζεται για να έχουμε εισερχόμενη ενέργεια ίση με 20-(prev-1). Ο χρόνος αυτός θα ισούται με $sec' = sec\frac{20-(prev\_level-1)}{hevergy}$. Άρα το ποσοστό του χρόνου φόρτισης θα ισούται με $ah = \frac{20-(prev\_level-1)}{hevergy}$.

Διάφορες άλλες αλλαγές που πρέπει να γίνουν είναι οι παρακάτω.
Η συνάρτηση transmit_time_delay πρέπει να τροποποιηθεί όπως παρακάτω:

```
void transmit_time_delay(void)
{
  int i = 0;
  in_delay = 1;
  for(i=0;i<sec;i++){
     delay(sec1);
```

```
  }
  in_delay = 0;
  battery_full_timer += sec;
}
```
Όπου ο χρόνος είναι κβαντισμένος με ελάχιστη τιμή το ένα δευτερόλεπτο.

Η συνάρτηση display_mode:
```
void display_mode(void)
{
	change_mode=sec;

}
```
Όπου απλά θα αλλάζει την τιμή της μεταβλητής change_mode.

Επίσης σβήνουμε την συνάρτηση #pragma vector=PORT1_VECTOR __interrupt void Port_1() μιας και αυτή η συνάρτηση χρησιμοποιούταν για να αλλάζει από το μπουτόν χειροκίνητα το χρόνο της επόμενης μετάδοσης.

Οι αλλαγές που μένουν είναι στο άπειρο loop που πέφτει το πρόγραμμα μέσα στην συνάρτηση LinkTo. Συγκεκριμένα πρέπει να προσθέσουμε την πληροφορία της στάθμης της εικονικής μπαταρίας καθώς επίσης και του ah στο μήνυμα που θα μεταδοθεί στο Access Point. Αυτό γίνεται με την προσθήκη των παρακάτω γραμμών (που θα μπουν πριν να μεταδοθεί το μήνυμα):
```
msg[7]=level&0xFF;
msg[8]=(level>>8)&0xFF;
msg[9]=--ah_int;
```

Επίσης αμέσως μετά το transmit_time_delay που πρέπει να υπολογίσουμε την καινούρια στάθμη της εικονικής μπαταρίας και τον καινούριο χρόνο της επόμενης μετάδοσης προσθέτουμε τις παρακάτω γραμμές:
```
prev_level=level;
if(level>=lowerlevel){
	henergy=HarvestEnergy();
	level=level + henergy;
	//every transmit cost 1 unit of our virtual battery level
	level-=100;
	calc_new_trassmition();

}
else{//Not enough power to send information
	henergy=HarvestEnergy();
	level=level + henergy;
	calc_new_trassmition();
}
ah_int=(int)(ah*100);
if(ah_int==0) ah_int++;
display_mode();
```

Όπου διαχωρίζονται και οι καταστάσεις για το αν το σύστημα ήταν σε θέση να στείλει τα δεδομένα (όταν το επίπεδο της μπαταρίας είναι μεγαλύτερο ή ίσο του 5) οπότε και λειτουργεί ομαλά, ή αν δεν ήταν σε θέση επομένως σε αυτή την περίπτωση δεν θα έχουμε κατανάλωση αφού μετάδοση δεν έγινε.

Σημείωση: Όταν από το terminal δούμε μετάδοση με επίπεδο μπαταρίας μικρότερο του 5 αυτό σημαίνει πως η μετάδοση ΔΕΝ πραγματοποιήθηκε.

Στην επόμενη παρουσιάζεται μια τυπική έξοδος του προγράμματος, καθώς επίσης και διάφορα στατιστικά για το σύστημα.

4.1.3 Μετρήσεις και Στατιστικά

Για να δούμε την έξοδο του προγράμματος συνδεόμαστε μέσω terminal στο κατάλληλο port:

Τυπική έξοδος του προγράμματος:

Node:HUB0,Temp: 88.1F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:0001,Temp: 72.3F,Battery:3.1V,Strength:025%,RE:0131400,Lvl:20.00,ah=0.21
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.1F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.1F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.1F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.8F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 89.0F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 89.0F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:0001,Temp: 72.3F,Battery:3.1V,Strength:026%,RE:0121400,Lvl:20.00,ah=0.49
Node:HUB0,Temp: 89.0F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 89.0F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.8F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.7F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.7F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.7F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.7F,Battery:3.6V,Strength:000%,RE:2080000,Lvl:20.00,ah=1.00
Node:0001,Temp: 71.6F,Battery:3.1V,Strength:026%,RE:0131400,Lvl:19.84,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.8F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.8F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.8F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.8F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.1F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.3F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.5F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00
Node:HUB0,Temp: 88.7F,Battery:3.6V,Strength:000%,RE:1920000,Lvl:20.00,ah=1.00

Επεξήγηση:

Node:0001,Temp: 72.3F,Battery:3.1V,Strength:025%,RE:0131400,Lvl:20.00,ah=0.21

Από τη παραπάνω γραμμή παίρνουμε τις εξής πληροφορίες:

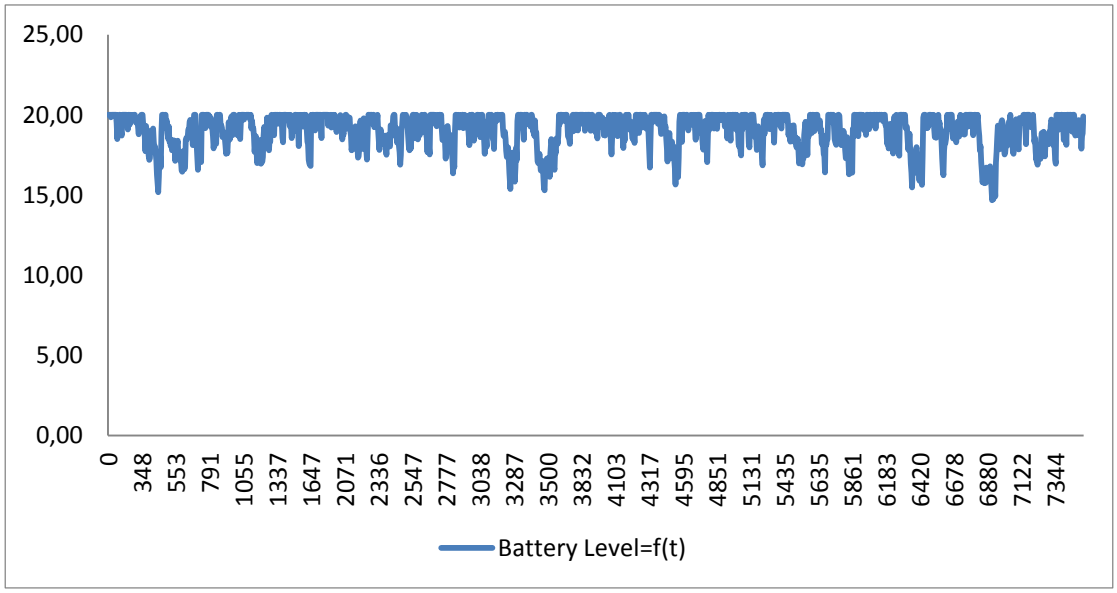
- Node:0001 - Ο Id αριθμός του End Device
- Temp: 72.3F – Η θερμοκρασία του περιβάλλοντος που βρίσκεται το εν λόγω End Device
- Battery:3.1V – Η τάση που παρέχουν οι μπαταρίες στο εν λόγω End Device
- Strength:025% – Η Ισχύς του σήματος
- RE:0131400 – Στο εν λόγω πεδίο μας ενδιαφέρει το δεύτερο και τρίτο ψηφίο που μας δίνει την πληροφορία του σε πόσα δευτερόλεπτα θα πραγματοποιηθεί η επόμενη μετάδοση
- Lvl:20.00 – Το level της μπαταρίας
- ah=0.21 – Το ποσοστό του χρόνου που η μπαταρία φορτίζει από την προηγούμενη προσαρμογή του duty cycle. (Σε αυτήν την περίπτωση ο χρόνος αυτός ταυτίζεται με το ποσοστό του χρόνου που η μπαταρία φορτίζει από την προηγούμενη μετάδοση).

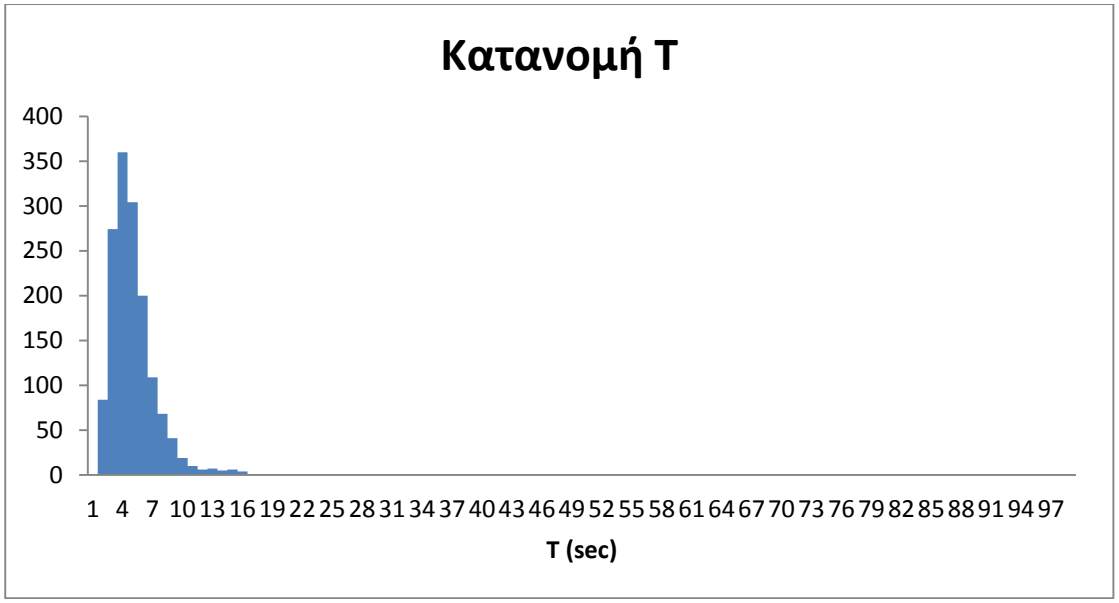
Για να πάρουμε στατιστικά για το κατά πόσο λειτουργεί αποδοτικά το σύστημα απομονώνουμε από τις μετρήσεις μας τις πληροφορίες που παίρνουμε μόνο από το node 0001. Για διάφορες τιμές έχουμε τα παρακάτω.

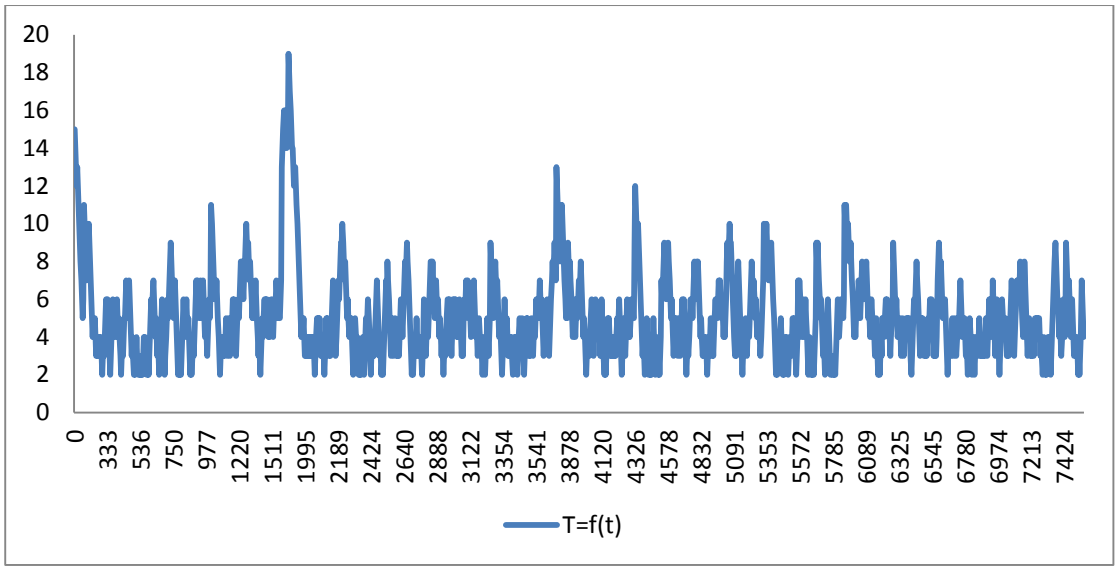
Για α=0.01 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

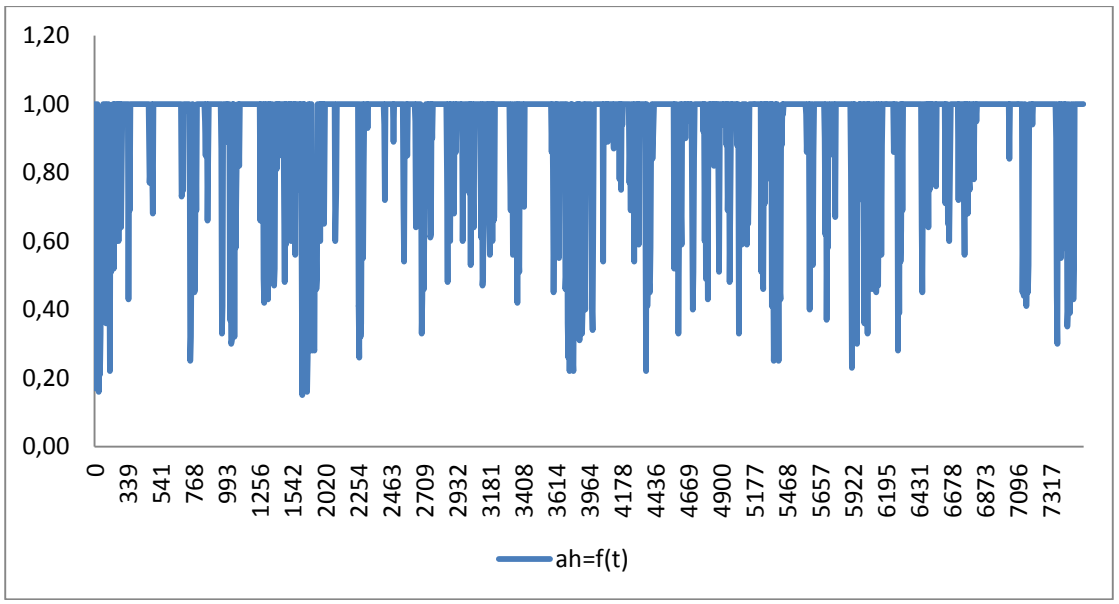
Μέση τιμή level μπαταρίας	19,0161
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	5,044667
Χαμένα πακέτα	0
Ποσοστό χαμένων πακέτων	0
Μέση τιμή ah	0,90

*Σημείωση: Λόγω του λ=0,25 θα περίμενε κανείς ότι ο μέσος χρόνος μετάδοσης θα ισούταν με 4, αφού η μέση τιμή της τυχαίας μεταβλητής θα είναι ίση με 1 και αφού 0,25*4=1 (όσο δηλαδή θα χρειαζόταν για να ικανοποιηθεί η κατανάλωση λόγω της μετάδοσης, που ισούται με 1). Ωστόσο αυτό δεν συμβαίνει και θα είναι πάντοτε μεγαλύτερη αυτής της τιμής που υπολογίζουμε αφού υπάρχουν διαστήματα που δεν μπορούμε να λάβουμε ενέργεια λόγω του κορεσμού της μπαταρίας.*



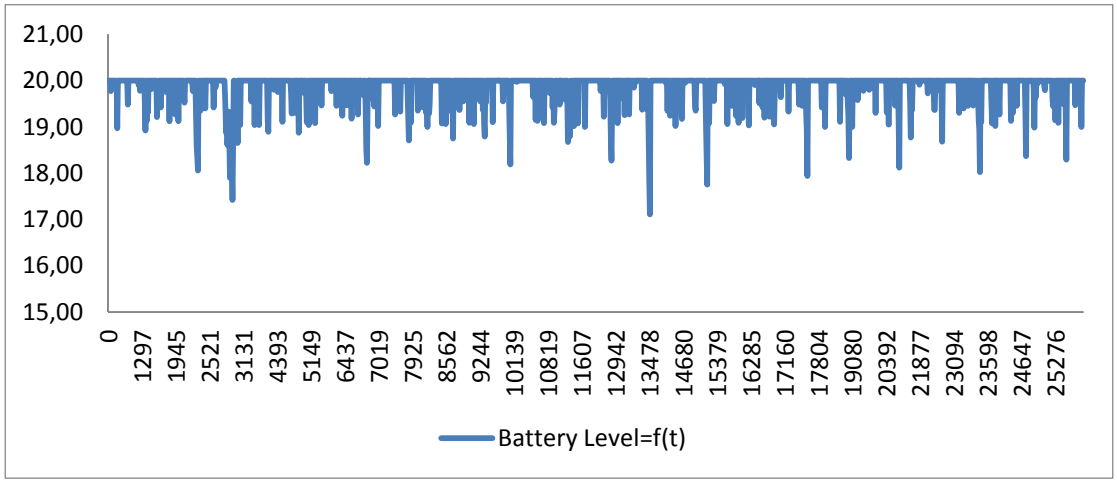

Κατανομή T
400
350
300
250
200
150
100
50
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97
T (sec)

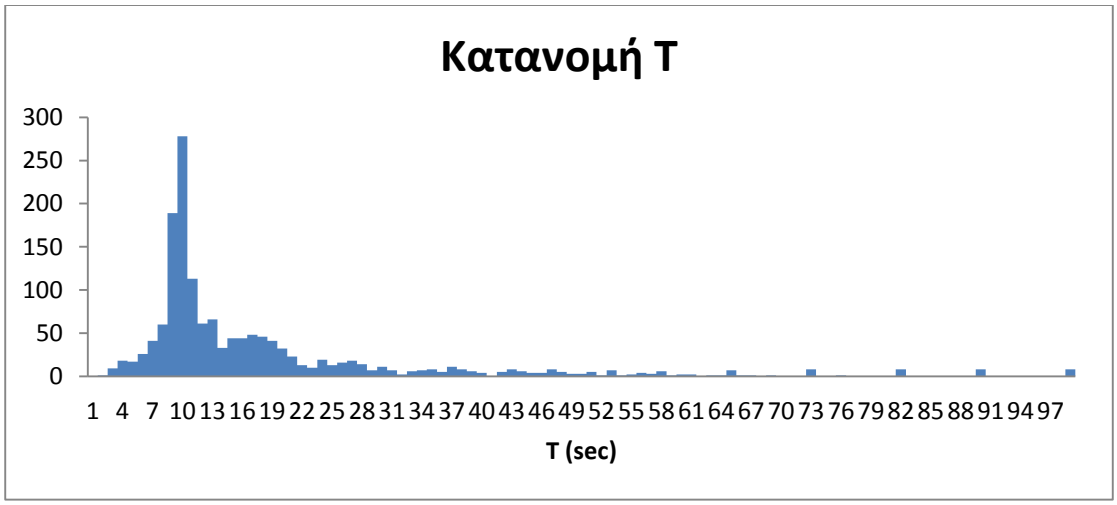

20
18
16
14
12
10
8
6
4
2
0
0 333 536 750 977 1220 1511 1995 2189 2424 2640 2888 3122 3354 3541 3878 4120 4326 4578 4832 5091 5353 5572 5785 6089 6325 6545 6780 6974 7213 7424
T=f(t)

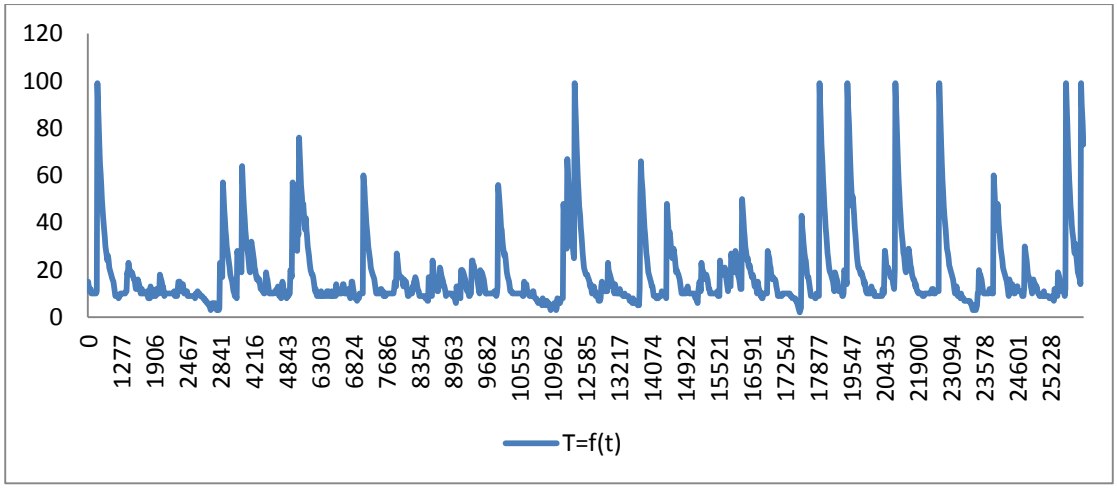

1,20
1,00
0,80
0,60
0,40
0,20
0,00
0 339 541 768 993 1256 1542 2020 2254 2463 2709 2932 3181 3408 3614 3964 4178 4436 4669 4900 5177 5468 5657 5922 6195 6431 6678 6873 7096 7317
ah=f(t)

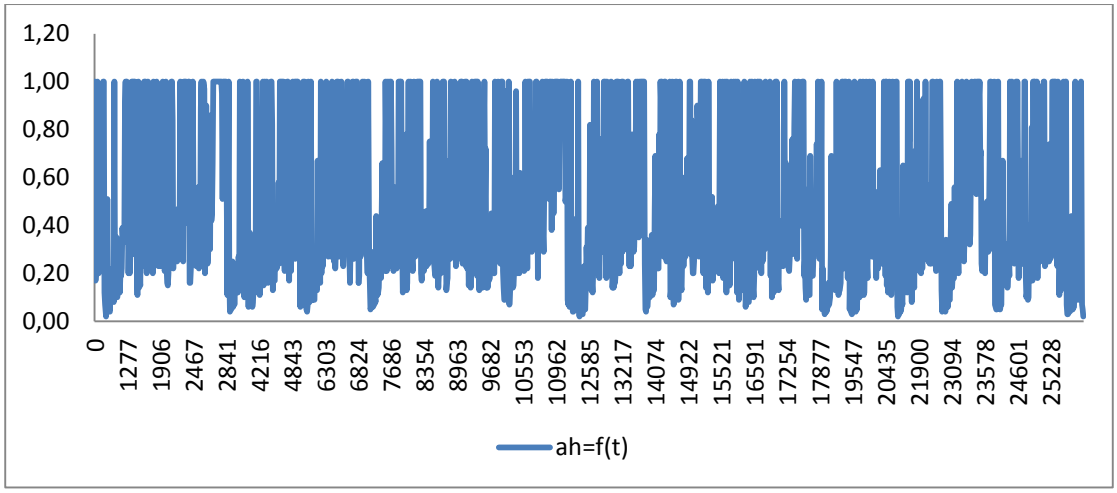
Για α=0. 1 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

Μέση τιμή level μπαταρίας	19,86819
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	17,86267
Χαμένα πακέτα	0
Ποσοστό χαμένων πακέτων	0
Μέση τιμή ah	0,47



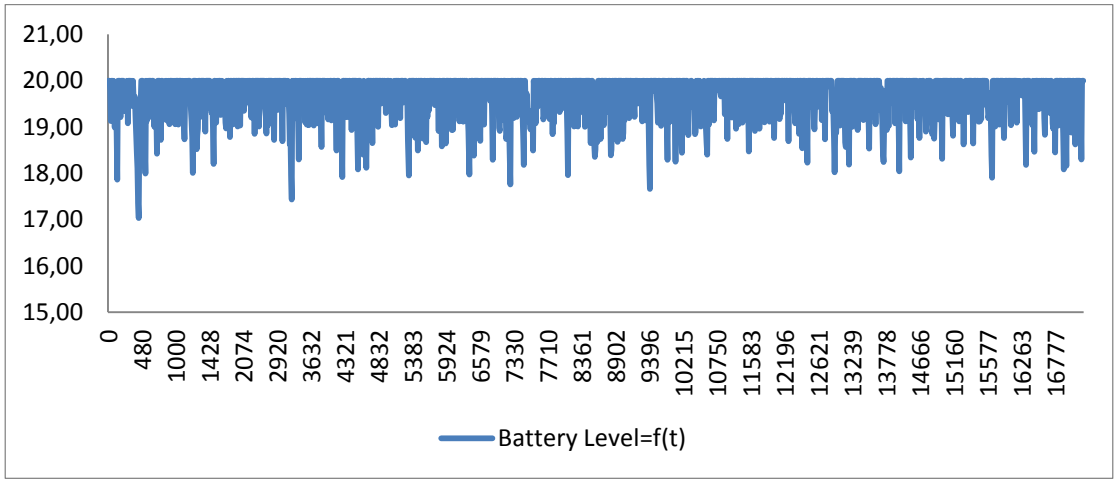


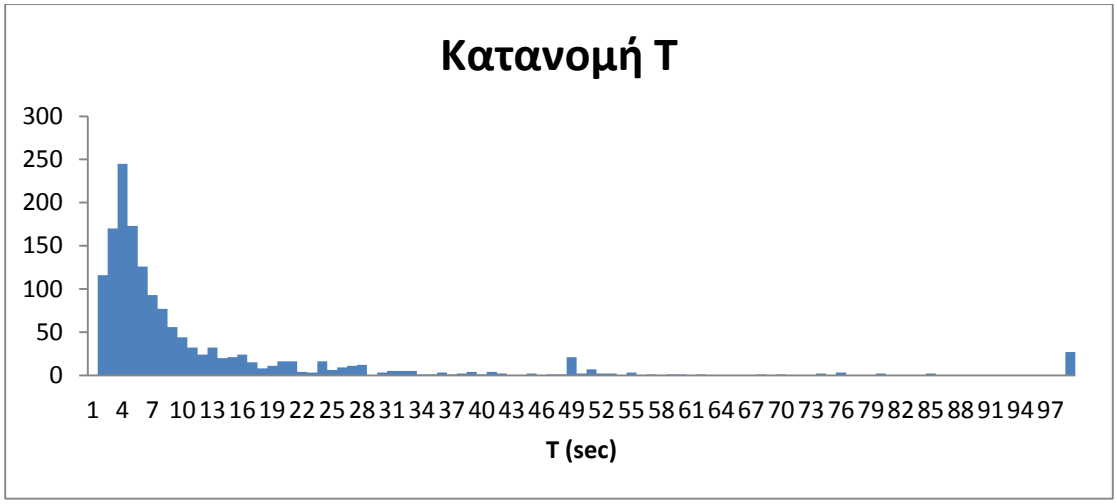


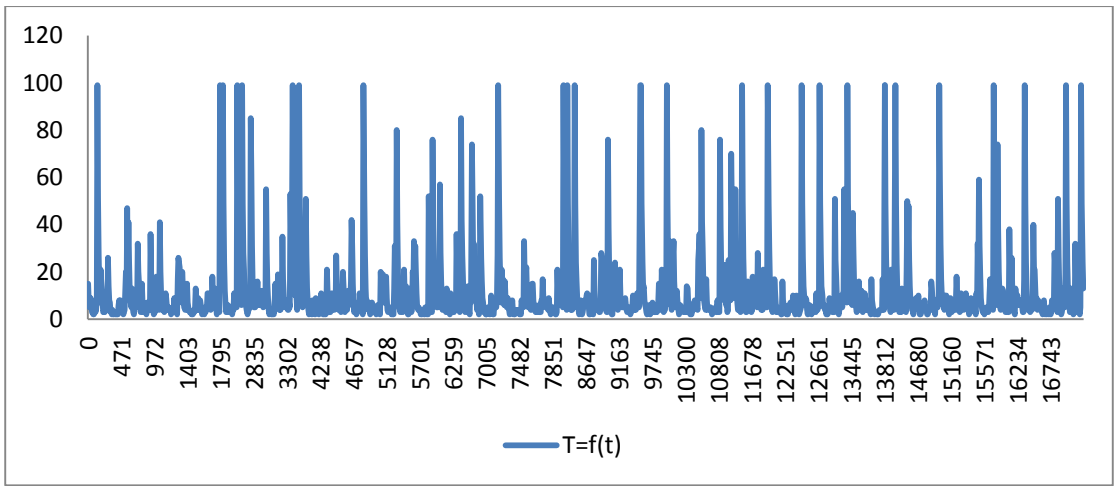


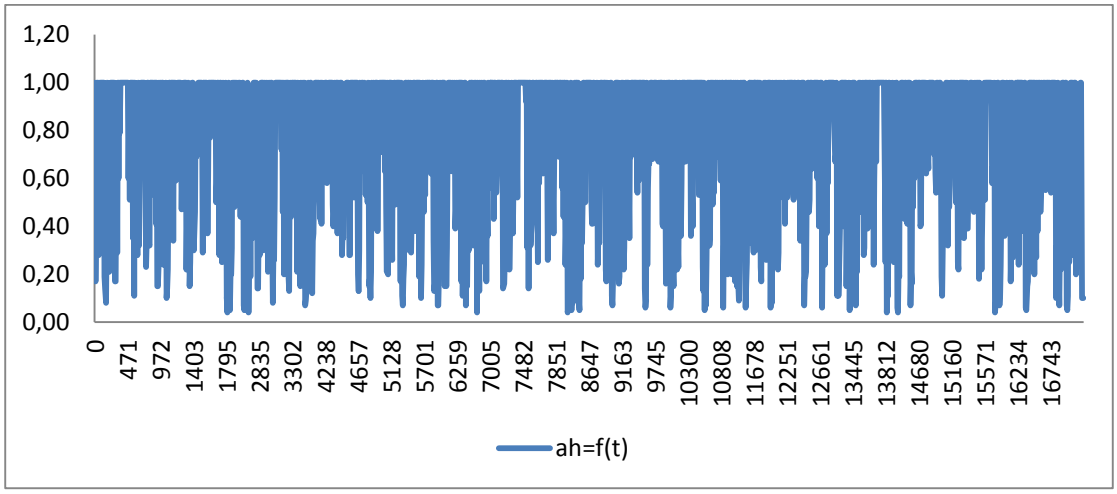
Για α=0. 5 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

Μέση τιμή level μπαταρίας	19,67359
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	11,67733
Χαμένα πακέτα	0
Ποσοστό χαμένων πακέτων	0
Μέση τιμή ah	0,72



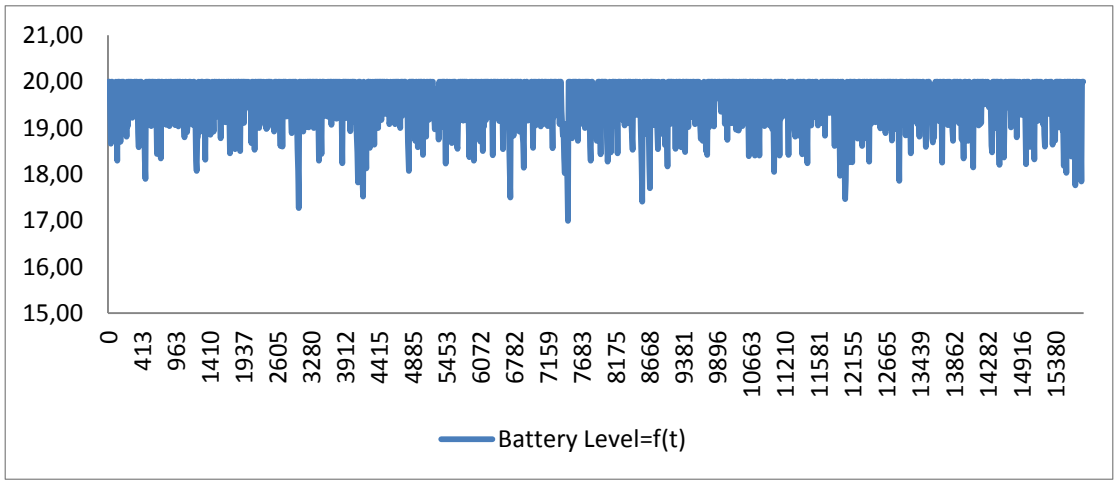


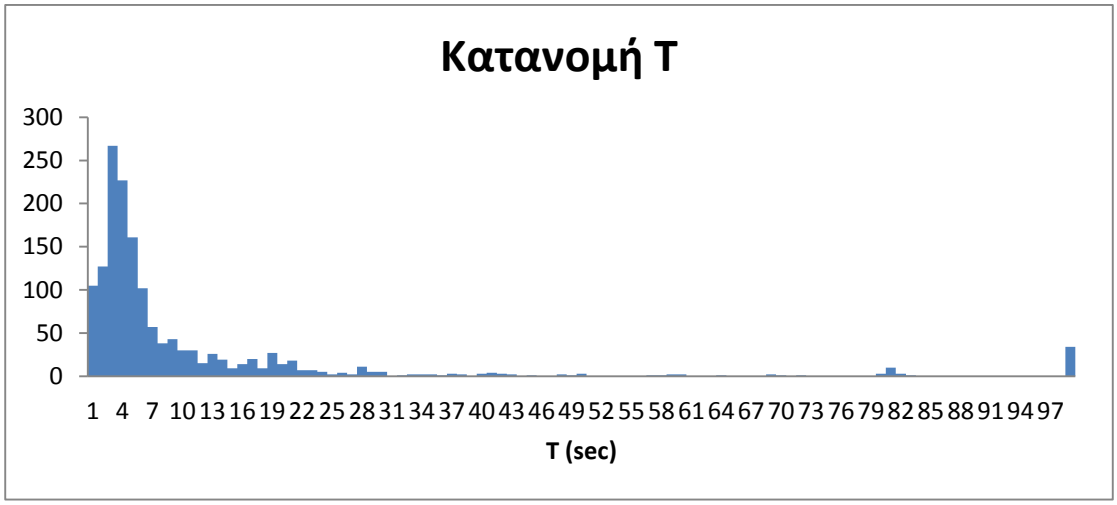


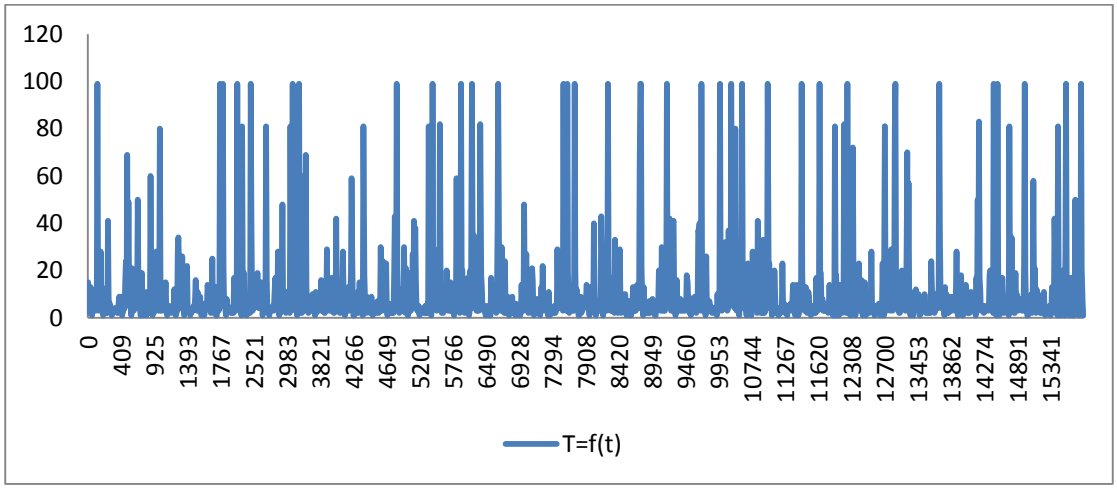


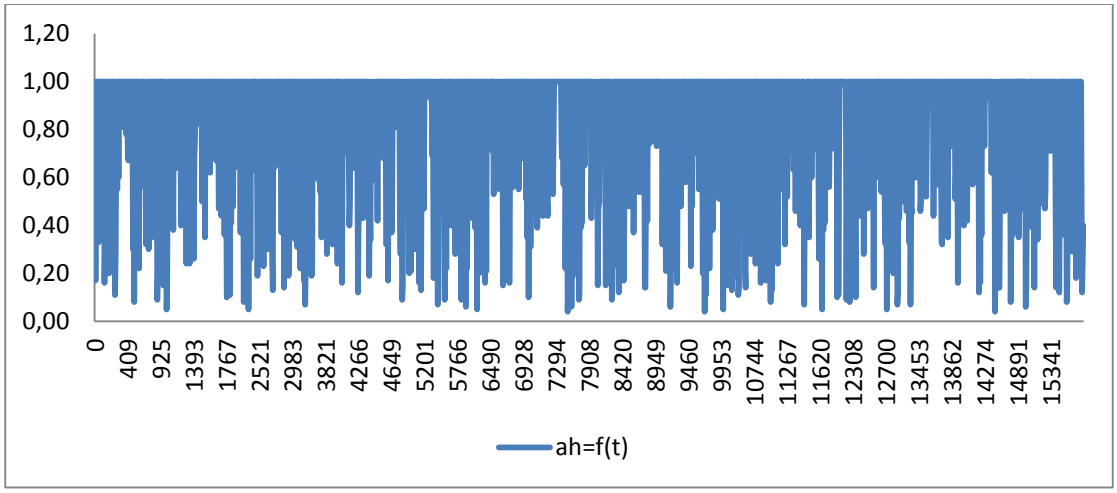
Για α=0. 8 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

Μέση τιμή level μπαταρίας	19,56811
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	10,66933
Χαμένα πακέτα	0
Ποσοστό χαμένων πακέτων	0
Μέση τιμή ah	0,79


21,00
20,00
19,00
18,00
17,00
16,00
15,00
0
413
963
1410
1937
2605
3280
3912
4415
4885
5453
6072
6782
7159
7683
8175
8668
9381
9896
10663
11210
11581
12155
12665
13439
13862
14282
14916
15380
Battery Level=f(t)


Κατανομή Τ
300
250
200
150
100
50
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97
T (sec)


120
100
80
60
40
20
0
0
409
925
1393
1767
2521
2983
3821
4266
4649
5201
5766
6490
6928
7294
7908
8420
8949
9460
9953
10744
11267
11620
12308
12700
13453
13862
14274
14891
15341
T=f(t)



4.1.4 Αποτελέσματα Μετρήσεων

Παρότι δεν μπορούμε να βγάλουμε «καθαρά» συμπεράσματα από τις μετρήσεις που κάναμε διαπιστώνουμε τα εξής:

Όσο πιο μικρή είναι η τιμή του α τόσο πιο σταθερό τείνει να είναι το σύστημα. Αυτό γιατί ιδιαίτερα για πολύ μικρές τιμές το ah τείνει στην μονάδα. Επίσης όσο πιο μικρό είναι το α τόσο μικρότερη διασπορά τείνουμε να έχουμε στην κατανομή του T. Χαμένα πακέτα δεν έχουμε σε καμία των περιπτώσεων και γενικά είναι πολύ δύσκολο να δημιουργηθούν αφού σε όλες τις περιπτώσεις το level της μπαταρίας τείνει στο 20.

Γενικά όταν το α λάβει πολύ χαμηλές τιμές το σύστημα δουλεύει ομαλά με πολύ μικρή διασπορά στην κατανομή του T.

4.2 Βελτίωση Απλοποιημένης Μεθόδου

Στην ενότητα 4.1 μελετήσαμε μια απλοποιημένη μέθοδο για την δυναμική μεταβολή του duty cycle. Σε αυτή την ενότητα θα βελτιώσουμε λιγάκι αυτή τη μέθοδο. Η βελτίωση που θα κάνουμε θα είναι η εξής. Λόγω του ότι είχαμε μεγάλη διασπορά της κατανομής του χρόνου T στην προηγούμενη περίπτωση, τώρα για τον υπολογισμό της επόμενης μετάδοσης θα περιοριστούμε σε κάποια όρια ±num του χρόνου αναμονής που είχαμε για την πραγματοποίηση της τρέχουσας μετάδοσης, όπου num ένας αριθμός τύπου integer που θα τον κάνουμε define και θα είναι ίσος με 4. Με αυτόν τον τρόπο πετυχαίνουμε γρήγορες αποκρίσεις του συστήματος στις μεταβολές της εισερχόμενης ενέργειας μεταξύ όμως κάποιων ορίων, με στόχο να περιορίσουμε την διασπορά της κατανομής του χρόνου αναμονής που θα γίνει η επόμενη μετάδοση.

4.2.1 Προγραμματιστικές Αλλαγές

Οι προγραμματιστικές αλλαγές που θα γίνουν αφορούν μόνο τον κώδικα του End Device. Πιο συγκεκριμένα οι αλλαγές αυτές θα γίνουν στην συνάρτηση calc_new_trassmition όπου υπολογίζουμε τον χρόνο αναμονής για την επόμενη μετάδοση. Στο τέλος της συνάρτησης προσθέτουμε τις παρακάτω γραμμές:

```
if(sec<prev_sec - num)
     sec=prev_sec - num;
if(sec>prev_sec + num)
       sec=prev_sec + num)
```

Βεβαίως θα πρέπει να έχουμε δηλώσει στην αρχή του κώδικα την σταθερά num προσθέτοντας την παρακάτω γραμμή:

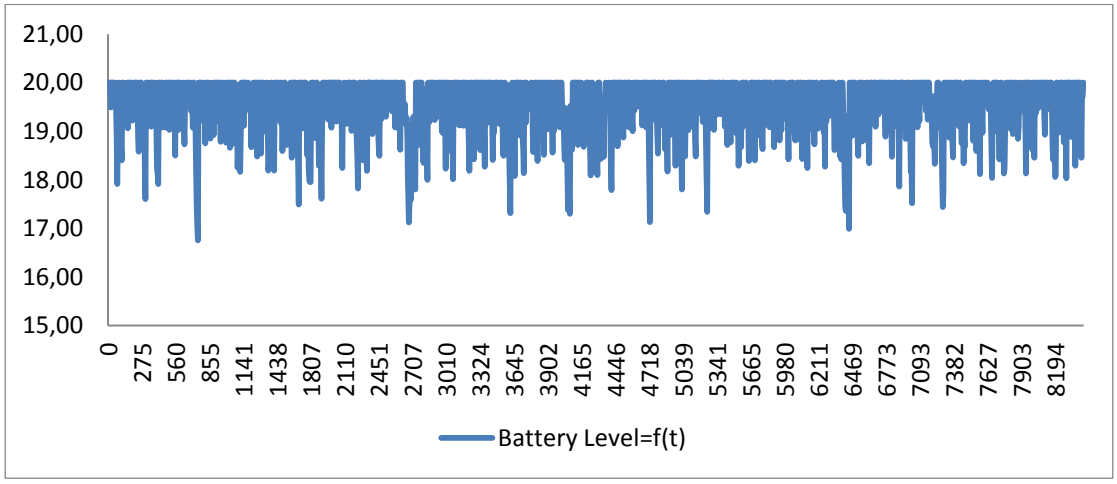
```
#define num 5
```

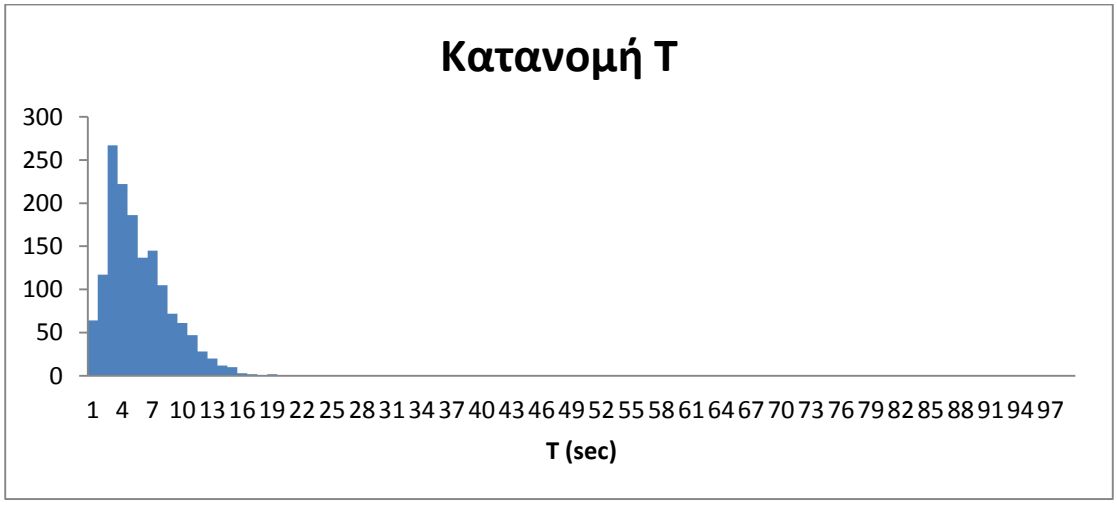
4.2.2 Μετρήσεις και Στατιστικά

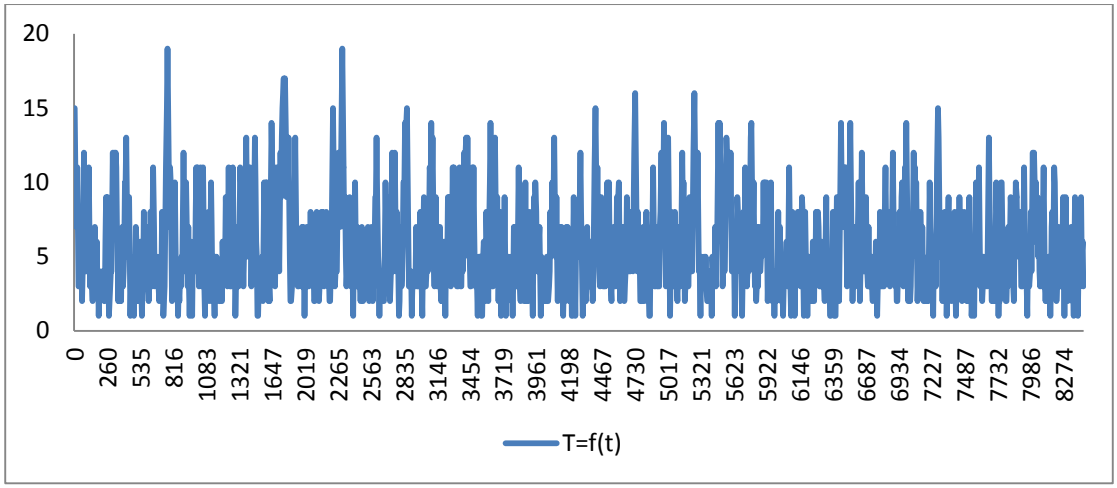
Όμοια με την παράγραφο 4.1.3 παίρνουμε αντίστοιχες όμοιες μετρήσεις (για να μπορέσουμε να τις συγκρίνουμε) για τις δύο τελευταίες περιπτώσεις όπου εμφανίζουν την μεγαλύτερη διασπορά.

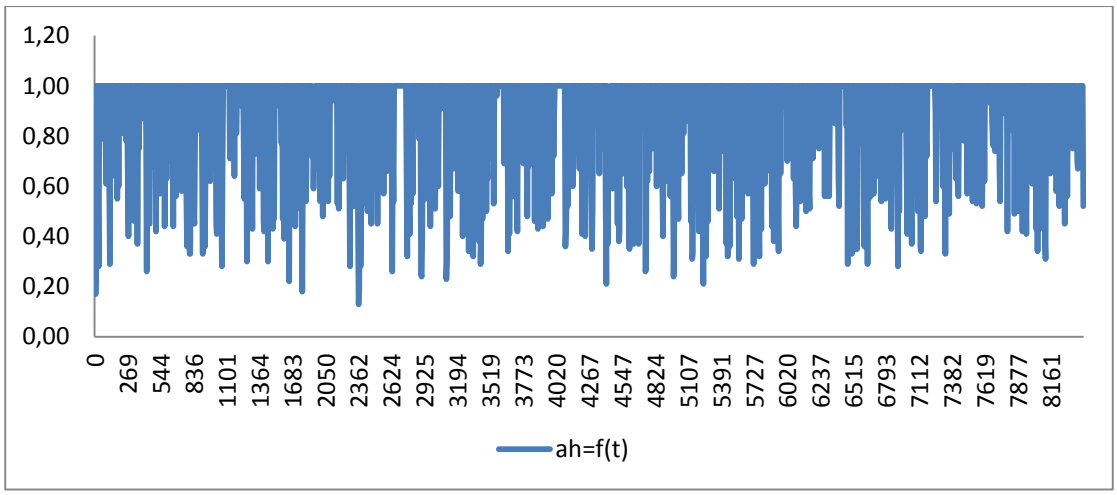
Για α=0. 5 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

Μέση τιμή level μπαταρίας	19,54226
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	5,764
Χαμένα πακέτα	0
Ποσοστό χαμένων πακέτων	0
Μέση τιμή ah	0,85



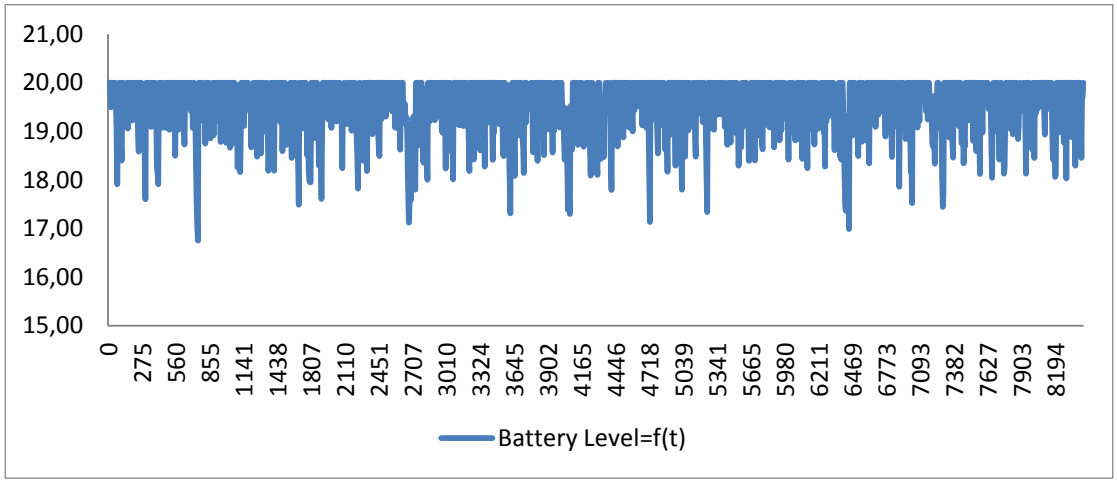


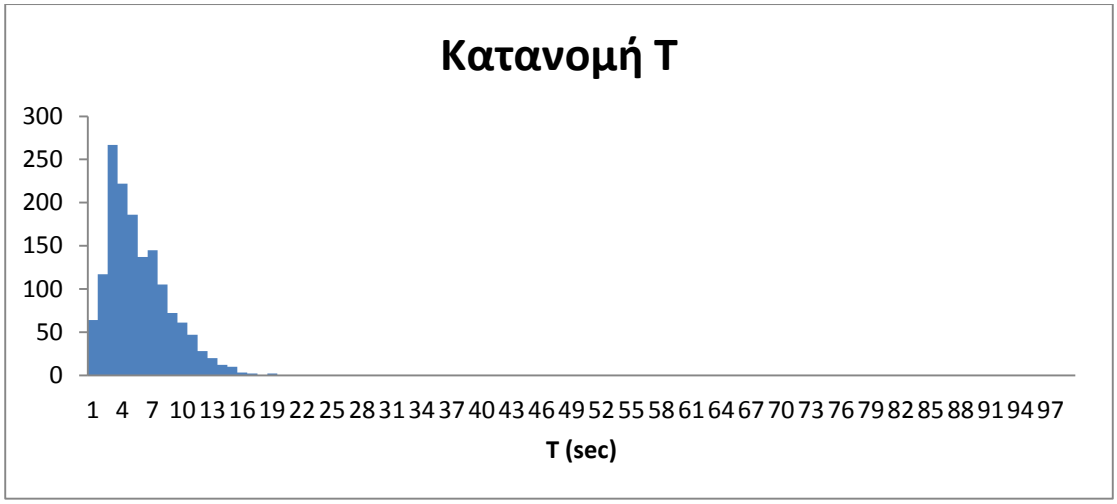


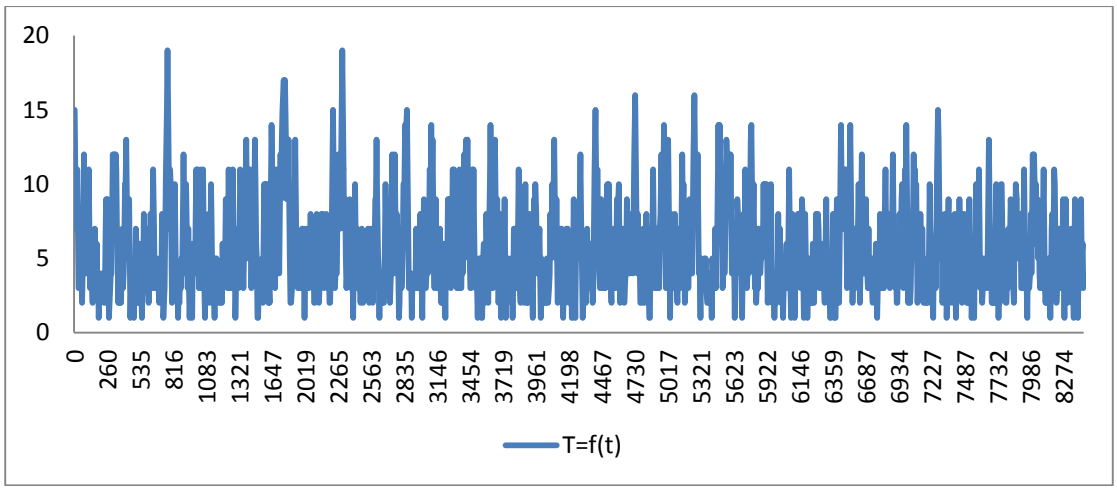


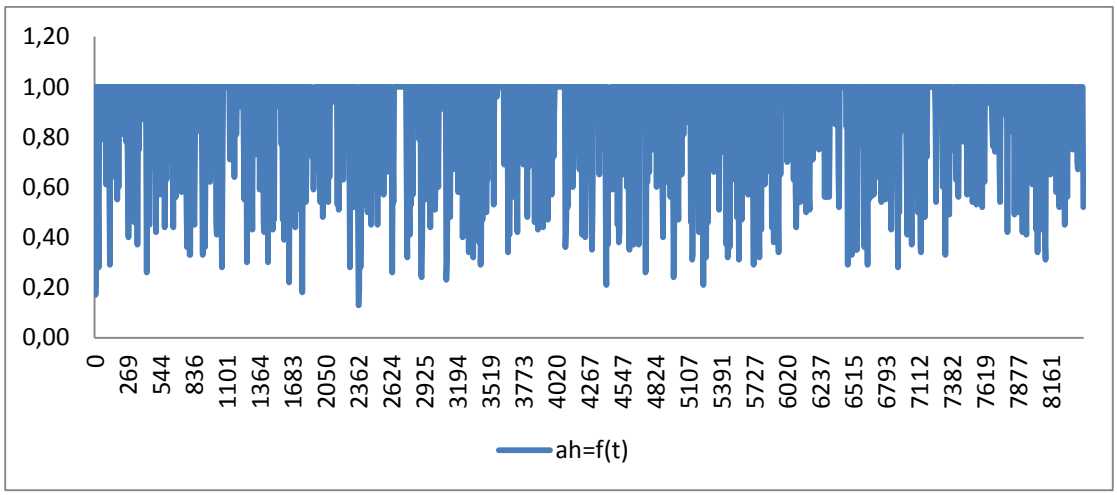
Για α=0. 8 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

Μέση τιμή level μπαταρίας	19,49939
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	5,602
Χαμένα πακέτα	0
Ποσοστό χαμένων πακέτων	0
Μέση τιμή ah	0,87









4.2.3 Αποτελέσματα Μετρήσεων

Συγκρινόμενα τα αποτελέσματα με αυτά της παραγράφου 4.1.3 συμπεραίνουμε πως έχουμε μία καλύτερη υλοποίηση του συστήματος. Αυτό φαίνεται από τις γραφικές παραστάσεις της κατανομής Τα όπου έχουμε πολύ μικρότερη διασπορά καθώς επίσης και από τις αυξημένες τιμές της μέσης τιμής του ah στις αντίστοιχες περιπτώσεις.

λ=0,25		Απλοποιημένη Μορφή	Απλοποιημένη Μορφή (Extended)
Μέση τιμή ah	α=0.5	0,72	0,85
	α=0.8	0,79	0,87

Αποτυχία αποστολής κάποιου πακέτου δεν έχουμε σε καμία των περιπτώσεων που μελετήσαμε και γενικά είναι πολύ δύσκολο να έχουμε τέτοια περίπτωση. Σε αυτή την μέθοδο έχουμε γρήγορες αποκρίσεις στις μεταβολές της εισερχόμενης ενέργειας και ταυτόχρονα μικρότερη διασπορά της κατανομής Τ λόγω του περιορισμού υπολογισμού του επόμενου Τ εντός ορίων που εξαρτάται από το τρέχον Τ.

4.3 Ρεαλιστική Μέθοδος

Οι δύο προηγούμενες μέθοδοι είναι γενικά δύσκολο να εφαρμοστούν σε ένα πραγματικό σύστημα. Αυτό γιατί είναι δύσκολο να γνωρίζουμε το level της εκάστοτε μπαταρίας που ο χρήστης του συστήματος θα χρησιμοποιεί (διαφορετικές μπαταρίες έχουν διαφορετικές χαρακτηριστικές καμπύλες από τις οποίες μπορούμε να βρούμε το level και επίσης η τάση των μπαταριών δεν είναι αυτό καθεαυτό ικανό κριτήριο για την ανεύρεση του level). Αυτό που μπορούμε να μετρήσουμε σχετικά εύκολα είναι πότε μια μπαταρία είναι πλήρως φορτισμένη και πότε είναι άδεια. Στην προκειμένη περίπτωση δεν μας ενδιαφέρει ακριβώς πότε είναι άδεια η μπαταρία αφού το πιθανότερο είναι η μπαταρία να έχει την ενέργεια να τρέχει τον αλγόριθμο αλλά να μην έχει την ενέργεια να κάνει την μετάδοση. Σε αυτήν την παράγραφο θα εφαρμόσουμε επακριβώς τον αλγόριθμο της παραγράφου 3.4.3. Σε κάθε δευτερόλεπτο θα γίνεται έλεγχος αν η μπαταρία είναι γεμάτη. Αν αυτό ισχύει η επόμενη μετάδοση θα προγραμματιστεί να γίνει σύμφωνα με τα νέα δεδομένα. Για να καταλάβουμε αν η μπαταρία είναι άδεια υποθέτουμε πως μια προγραμματισμένη μετάδοση δεν πραγματοποιήθηκε. Αυτό μπορεί να το καταλάβει το End Device αν (με την ενέργεια που του έμεινε και συνεχίζει να τρέχει ο αλγόριθμος) δεν λάβει confirmation από το Access Point ότι έλαβε το πακέτο. Αν αυτό συμβεί σημαίνει πως η ενέργεια στις μπαταρίες είναι τόση ώστε να τρέχει ο αλγόριθμος αλλά όχι αρκετή για να πραγματοποιηθεί η μετάδοση, και κατά συνέπεια αμέσως πριν την επόμενη μετάδοση πρέπει να υπολογιστεί ο καινούριος χρόνος του Τ. Σε αυτήν την υλοποίηση αναγκαστικά θα έχουμε κάποια χαμένα πακέτα. Παρακάτω θα δούμε το source code του End Device τις μετρήσεις και τα στατιστικά καθώς επίσης και τα συμπεράσματα που βγάζουμε.

4.3.1 Πηγαίος Κώδικας – Εφαρμογή στο eZ430-RF500

Ο κώδικας του Access Point θα παραμείνει ίδιος και πάλι ακριβώς όπως στην παράγραφο 4.1.1. Αυτό που αλλάζει πάλι είναι ο κώδικας του End Device. Παρακάτω παρουσιάζουμε τις αλλαγές που πρέπει να γίνουν στον κώδικα ξεκινώντας από τον εργοστασιακό κώδικα.

Δηλώσεις βοηθητικών μεταβλητών σταθερών και συναρτήσεων:

```
float HarvestEnergy(void);
void calc_new_trassmition(void);
//initial values
float level=2000;   //Level of the battery 20 is represented as 2000
float l=0.25;
float a=0.8;
int sec=15;

int prev_sec;
```

```
int charging_time=0,dt=0,n=0;
int henergy=0;
float ah=1;
int prev_event=1;
int event=0;
#define lowerlevel 500
#define num 4
#define SF 1
#define SE 2
```

Η συνάρτηση HarvestEnergy θα μας δίνει το ποσό της ενέργειας (τυχαία μεταβλητή) που εισέρχεται στο σύστημα σε διάρκεια ενός δευτερολέπτου.

```
float HarvestEnergy(){
      unsigned int temp;
      //harvest energy range should be for 0 to 200
      temp=rand()%201;
      return (l*(float)temp);
}
```

Αντικαθιστούμε την συνάρτηση transmit_time_delay με την time_delay η οποία απλώς εισάγει μία καθυστέρηση του ενός δευτερολέπτου.

```
void time_delay(void)
{

  in_delay = 1;
  //for(i=0;i<sec;i++){
     delay(sec1);
  //}
  in_delay = 0;
  battery_full_timer += sec;
}
```

Εισάγουμε την συνάρτηση calc_new_trassmition η οποία θα υπολογίζει το νέο T κάθε φορά που έχουμε ένα Storage Full ή Storage Empty event.

```
void calc_new_trassmition(void){
      int de;
      float Th_temp,temp;
      prev_sec=sec;
      //calculation of ah
      if(event == SF){
            if(dt!=0)
                  ah=(float)charging_time/(float)dt;
            else
                  ah=0.5;
      }
      if(event == SE)
            ah=1.0;
      //calculation of ΔE
      if(event==prev_event){
            de=0;
      }
      else{
            if(event==SF && prev_event==SE)
                  de=2000.0-lowerlevel;
            else
                  de=lowerlevel-2000.0;
```

```
      }
      temp=((float)de/100.0) +(float)n;
      if(temp==0) temp=0.01;
      //calculation of Th*
      Th_temp=ah*((float)dt)/temp;
      //Finally the new T
      sec= a*Th_temp + ((float)sec)*(1.0-a);

      prev_event=event;
      event=0;
    dt=0;
    n=0;
    charging_time=0;
      //T should be within [1,99] range
      if(sec<1) sec=1;
      //Maximum transmit time 99sec
      if(sec>99) sec=99;
      //Limits of the next T related to the previous T
      if(sec<prev_sec - num)
           sec=prev_sec - num;
      if(sec>prev_sec + num)
             sec=prev_sec + num;
}
```

Τέλος αλλάζουμε την συνάρτηση linkTo (όπου εκεί τρέχει το άπειρο loop):

```
void linkTo(void)
{
  linkID_t linkID1;
  uint8_t  msg[10];
  int  ah_int;
  unsigned int *tempOffset;                 // Initialize temperature
offset
  tempOffset = (unsigned int *)0x10F4;      // coefficient
  number_transmits = xmt_count;             // Initialize to max
transmit #

  // keep trying to link... Uses Timer B to wake up periodically
  while (SMPL_SUCCESS != SMPL_Link(&linkID1))
  {
    __bis_SR_register(LPM3_bits + GIE);     // LPM3 with interrupts
enabled
  }

  // put radio to sleep once a successfull connection has been
established
  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );
  prev_sec=sec;
  while(1)
  {
    if(dt%change_mode==0){
        volatile long temp;
      int degC, volt;
      int results[2];

            if(event!=SF) ah=1;
            //if we have a Storage Full event
            //we calculate the new T for the next transmission
            if(event==SF){
            calc_new_trassmition();
            display_mode();
```

```
      }
      //if level<lowerlevel the package can't be send
      //and we have a Storage Empty event
      if(level<lowerlevel){
            event=SE;
      }
      //This will be executed after we calculate the
      //new T after a Storage Empty event
            if(change_mode<sec){
                  charging_time=0;
                  dt=0;
                  display_mode();
            }
            // If battery charging, go back to sleep  (if P3.5 = 1,
Sleep)
      P3REN &= ~0x20;                           // turn off pulldown
resistor
      delay(port_delay);

      // Measure Temperature
      ADC10CTL1 = INCH_10 + ADC10DIV_4;       // Temp Sensor
ADC10CLK/5
      ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE +
ADC10SR;
      __delay_cycles(350);                    // delay to allow
reference to settle
      ADC10CTL0 |= ENC + ADC10SC;             // Sampling and
conversion start
      __bis_SR_register(LPM0_bits + GIE);     // LPM0 with interrupts
enabled
      results[0] = ADC10MEM;
      ADC10CTL0 &= ~ENC;

      // Measure Battery Voltage
      ADC10CTL1 = INCH_11;                    // AVcc/2
      ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE +
REF2_5V;
      __delay_cycles(350);                    // delay to allow
reference to settle
      ADC10CTL0 |= ENC + ADC10SC;             // Sampling and
conversion start
      __bis_SR_register(LPM0_bits + GIE);     // LPM0 with interrupts
enabled
      results[1] = ADC10MEM;
      ADC10CTL0 &= ~ENC;
      ADC10CTL0 &= ~(REFON + ADC10ON);        // turn off A/D to save
power

      // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 -
278
      // the temperature is transmitted as an integer where 32.1 =
321
      // hence 4230 instead of 423
      temp = results[0];
      degC = ((temp - 673) * 4230) / 1024;
      if( *tempOffset != 0xFFFF )
      {
        degC += *tempOffset;
      }

      /* message format,  UB = upper Byte, LB = lower Byte
```

```
      ------------------------------------------------------------
-------------
      |degC LB | degC UB |  volt LB | Mode  | # transmit LB |#
transmit UB |  ?  |
      ------------------------------------------------------------
-------------
         0         1          2          3          4               5
6
      */

      // Wake radio-up
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, "" );

          temp = results[1];
      volt = (temp*25)/512;
      msg[0] = degC&0xFF;
      msg[1] = (degC>>8)&0xFF;
      msg[2] = (int)volt;

      // If using Solar & not first time through, set battery as
ready
      if(((P3IN & 0x20) == 0x00) && (ftt_flag == 0) && (battery_ready
== 0))
      {
        battery_ready = 1;
      }
      // If battery is ready or first time through, transmit packets
      if(battery_ready == 1 || ftt_flag == 1){
            if((P3IN & 0x20))                     // If P3.5 = 1,
then
            {                                     // running on
battery
            msg[3] = (change_mode + running_on_battery); // +100 is
for GUI to know
                                                            // is
on battery
                  if( number_transmits != 0)
                  number_transmits--;               // # transmit
countdown
                  }
                  else
                  {                                    // else using
solar cells
                  msg[3] = change_mode;
                  if(number_transmits != xmt_count)
                  {
                        number_transmits++ ;
                        if(number_transmits >= xmt_count) // If max #
of transmits achieved,
                        {                                 // Reset
counter to 400.
                              number_transmits = xmt_count;
                        }
                  }
                  if(battery_full_flag == 1)          // If battery
is fully charged,
                  {                                   // reset
counter to 400
                        number_transmits = xmt_count;
                  }
                  }
```

```
                  msg[4] = number_transmits&0xFF;
                  msg[5] = (number_transmits>>8)&0xFF;

                  // used as a spare bit to indicate on and off
                  if(P3IN & 0x01)
                  {
                  msg[6] = ON;
                  }
                  else
                  {
                  msg[6] = OFF;
                  }
                  /*GArm Start*/
                  //Battery Level
                  msg[7]=((int)level)&0xFF;
                  msg[8]=(((int)level)>>8)&0xFF;

                  ah_int=(int)(ah*100);
                  if(ah_int==0) ah_int++;
                  msg[9]=--ah_int;
                  /*GArm End*/

                  // if end of battery, turn off battery
                  if(number_transmits == 0)
                  {
                  in_delay = 1;
                  while((P3IN & 0x20))                // Continue
sleeping if still on
                  {                                   // battery
                        __bis_SR_register(LPM4_bits);
                  }
                  in_delay = 0;
                  }

                  // Send message
                  if (SMPL_SUCCESS == SMPL_Send(linkID1, msg,
sizeof(msg)))
                  {
                  delay(port_delay);

                  if(P3IN & 0x20)                     // Using
Battery, Blink Red
                  {
                        status_indicator(status_one, 1);
                  }
                  else                                // Using Solar
& Blink Green
                  {
                        status_indicator(status_one, 2);
                  }
                  }
                  else                                  // Blink both
LED if transmission
                  {                                     // failed
                  status_indicator(status_one, 1);
                  status_indicator(status_one, 2);
                  }
      }

      ftt_flag = 0;                           // first time thru the
program flag
```

```
      check_bat_full();
      status = status_six;
      P3REN |= 0x20;                          // Set /Charge pulldown
resistor
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );

      //every transmit cost 1 unit of our virtual battery level
      level-=100;
      //n = number of transmissions within dt time
      n++;
      //if the package fail to transmit there is no
      //loss in the level of the battery
      //and we calculate the new T
      if(event==SE){
            level+=100;
            n--;
            calc_new_trassmition();
      }

    }//end of if(dt%change_mode==0)

    //from this point everything will be executed every 1 sec
    time_delay(); // sleep time

    dt+=1;
    //new level of the battery
    level+=HarvestEnergy();
    //In order to calculate ah or find a Storage full event
    if(level<2000){
      charging_time+=1;
    }
    else{//battery fully charged (SF event)
      level=2000;
      event = SF;
    }
  }
}
```

Τυχόν επεξηγήσεις φαίνονται στα σχόλια του κώδικα.

<u>Σημείωση:</u> *Λόγω του ότι η υλοποίηση είναι εικονική εντοπίζουμε ένα Storage Empty event με τις παρακάτω γραμμές:*

```
if(level<lowerlevel){
            event=SE;
      }
```

Αυτό σε μία πραγματική υλοποίηση μπορεί να εντοπιστεί αν δεν λάβουμε confirmation από το Access Point πως έλαβε το πακέτο προσθέτοντας το event=SE μέσα στο else στο παρακάτω σημείο:

```
if (SMPL_SUCCESS == SMPL_Send(linkID1, msg, sizeof(msg)))
                  {
                  delay(port_delay);

                  if(P3IN & 0x20)                     // Using
Battery, Blink Red
                  {
                        status_indicator(status_one, 1);
                  }
                  else                                // Using Solar
& Blink Green
                  {
```

```
                    status_indicator(status_one, 2);
                }
                }
                else                                    // Blink both
LED if transmission
                {                                       // failed
                event=SE;
                status_indicator(status_one, 1);
                status_indicator(status_one, 2);
                }
```

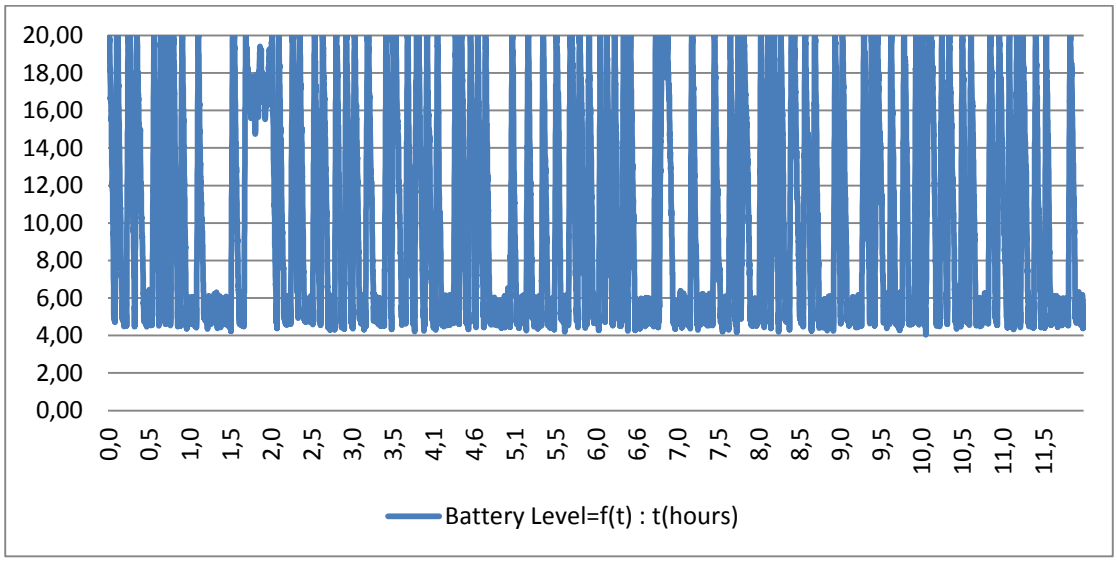
4.3.2 Μετρήσεις και Στατιστικά

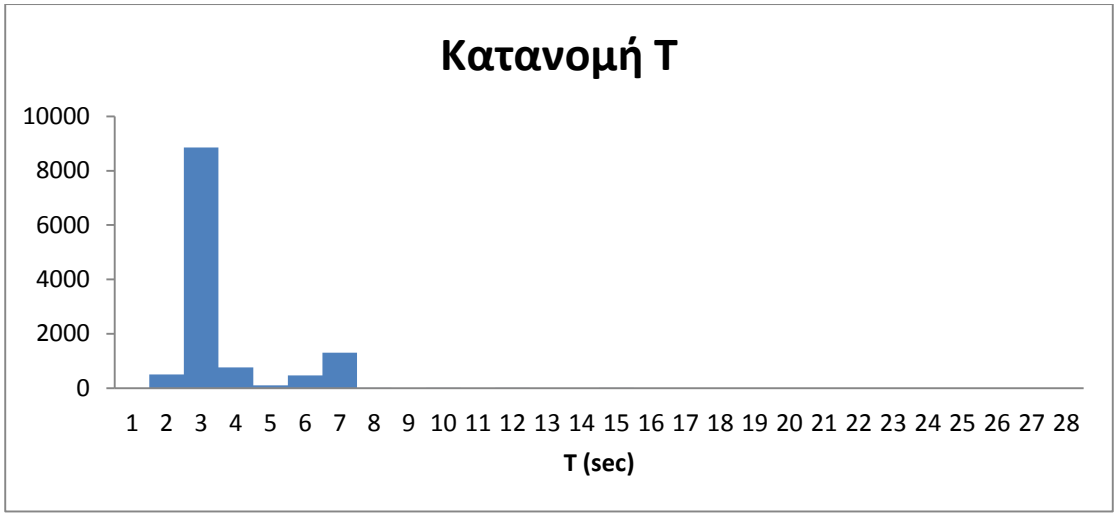
Όμοια με τις προηγούμενες περιπτώσεις παίρνοντας τις μετρήσεις φτιάχνουμε τις αντίστοιχες γραφικές παραστάσεις. Επιπλέον μετράμε πόσες φορές στη σειρά έχουμε χαμένα πακέτα λόγω αδυναμίας του συστήματος να τα στείλει λόγω έλλειψης ενέργειας. Τις μετρήσεις της πήραμε σε όλες τις περιπτώσεις για τις πρώτες 12.000 μεταδόσεις.

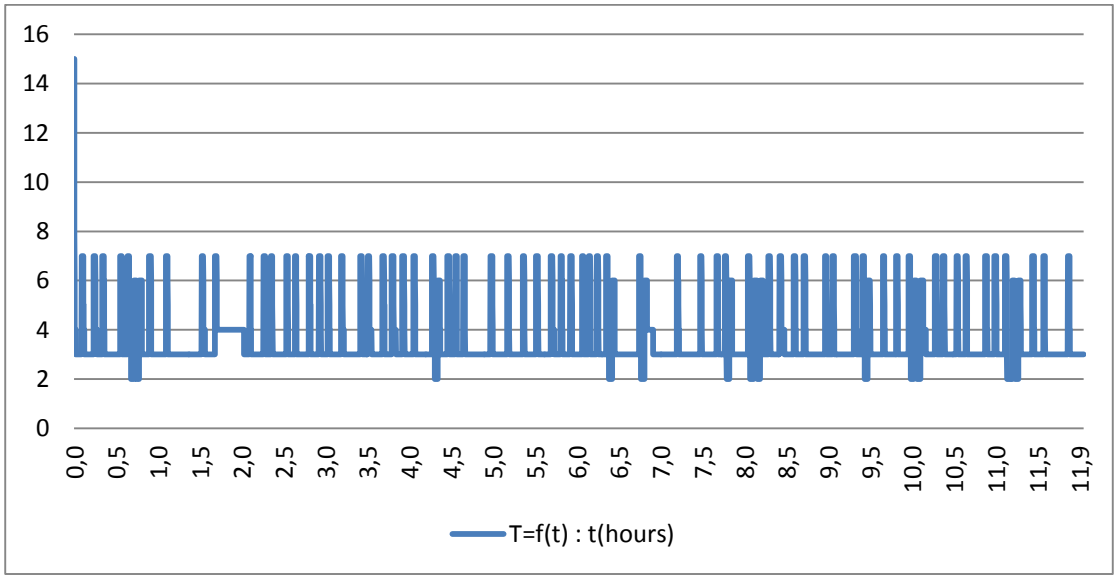
Για α=0. 2 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

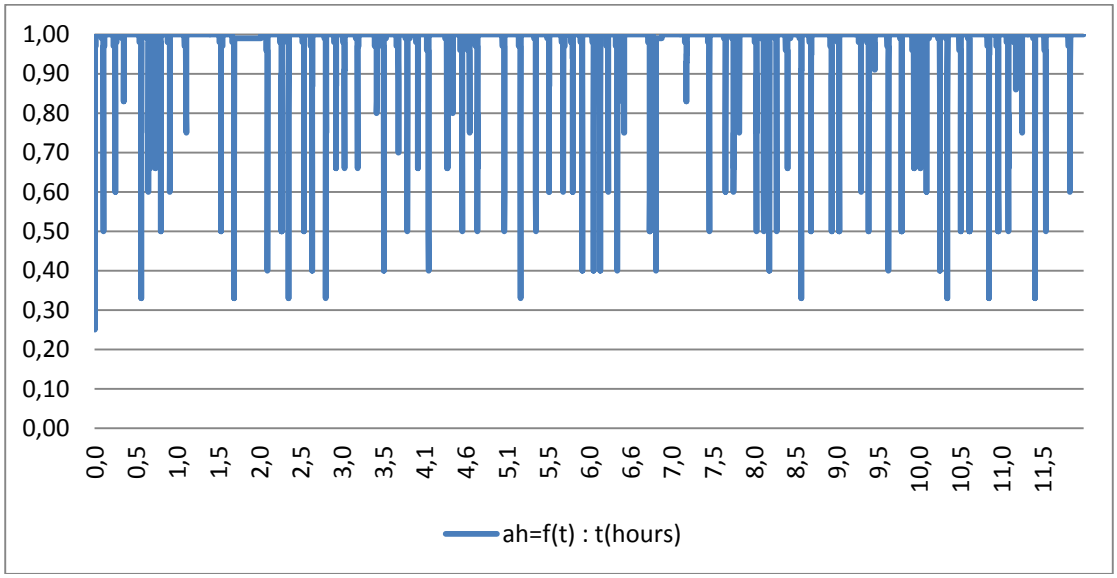
Μέση τιμή level μπαταρίας	10,13
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	4,03507
Χαμένα πακέτα	1307
Ποσοστό χαμένων πακέτων	0,108917
Μέση τιμή ah	0,98914

Χαμένα πακέτα			
2 στη σειρά	3 στη σειρά	4 στη σειρά	5 στη σειρά
82	3	0	0





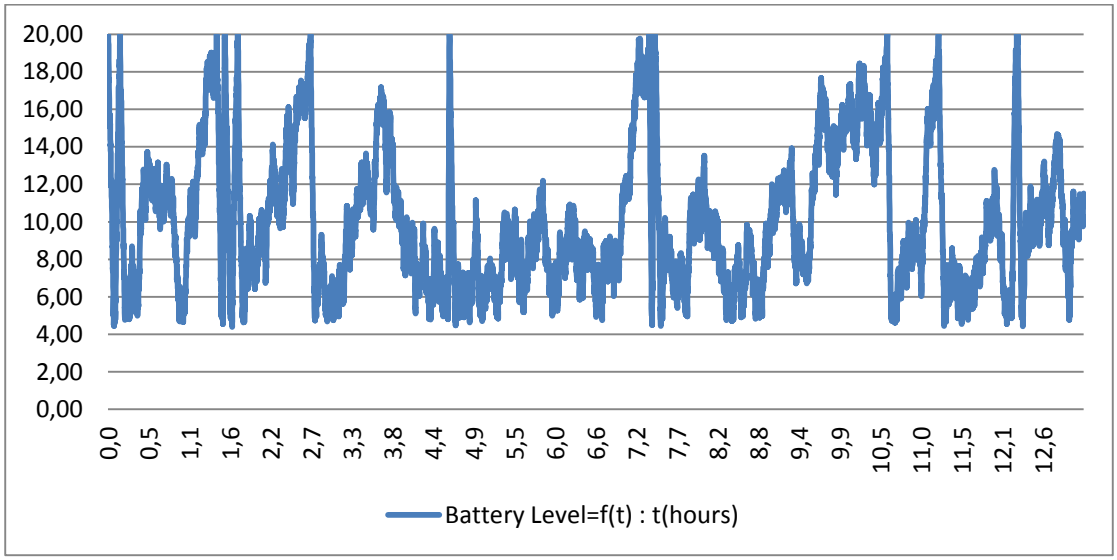


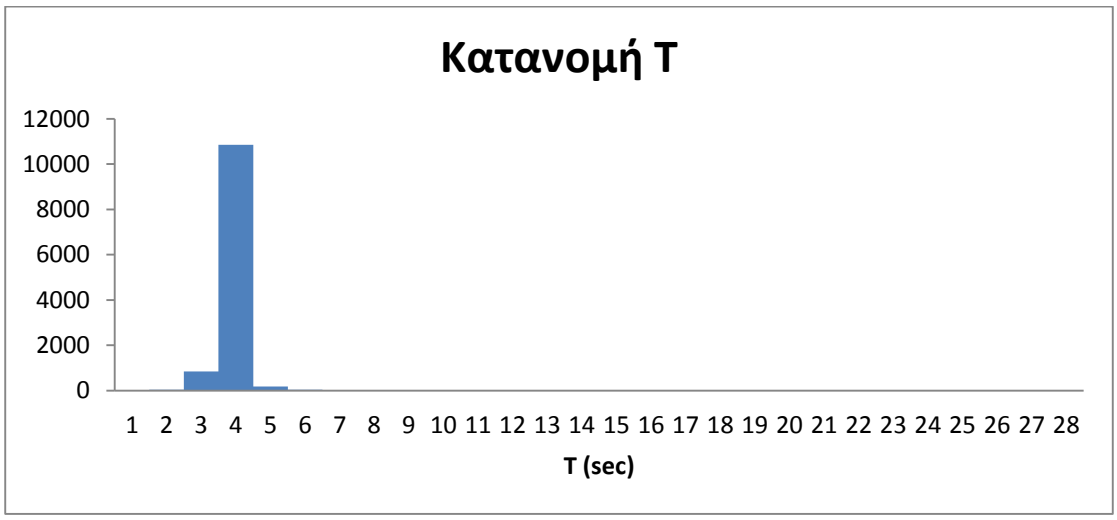


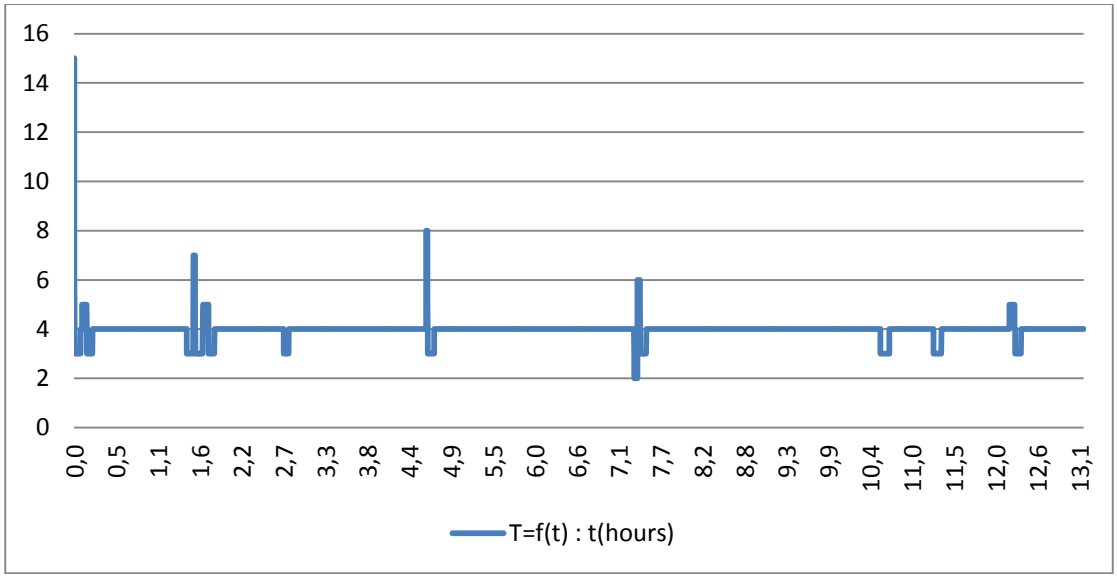
Για α=0. 4 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

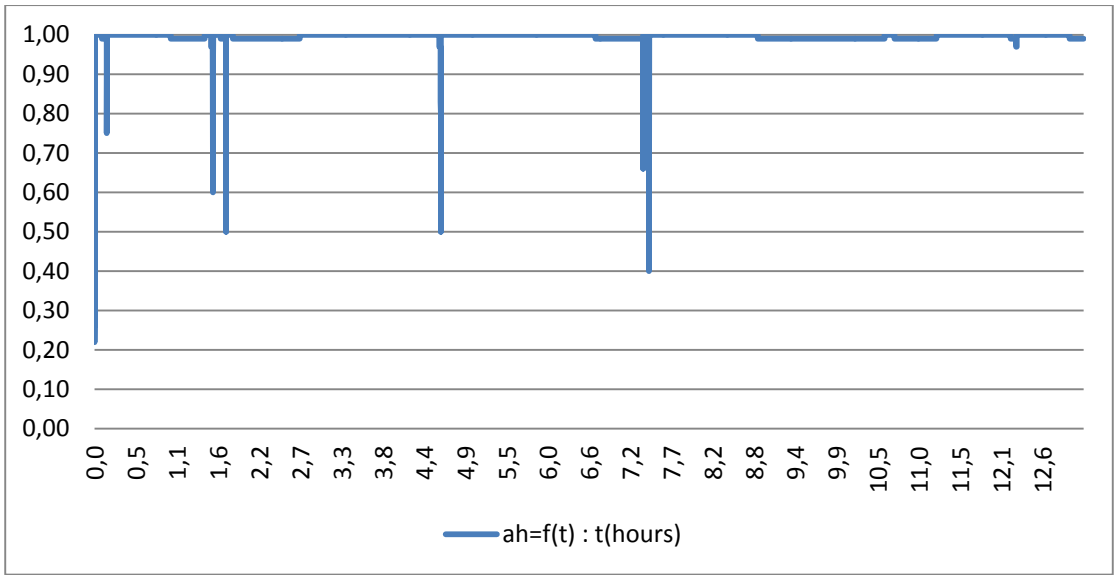
Μέση τιμή level μπαταρίας	10,04
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	3,998232
Χαμένα πακέτα	125
Ποσοστό χαμένων πακέτων	0,010417
Μέση τιμή ah	0,99583

Χαμένα πακέτα			
2 στη σειρά	3 στη σειρά	4 στη σειρά	5 στη σειρά
3	0	0	0





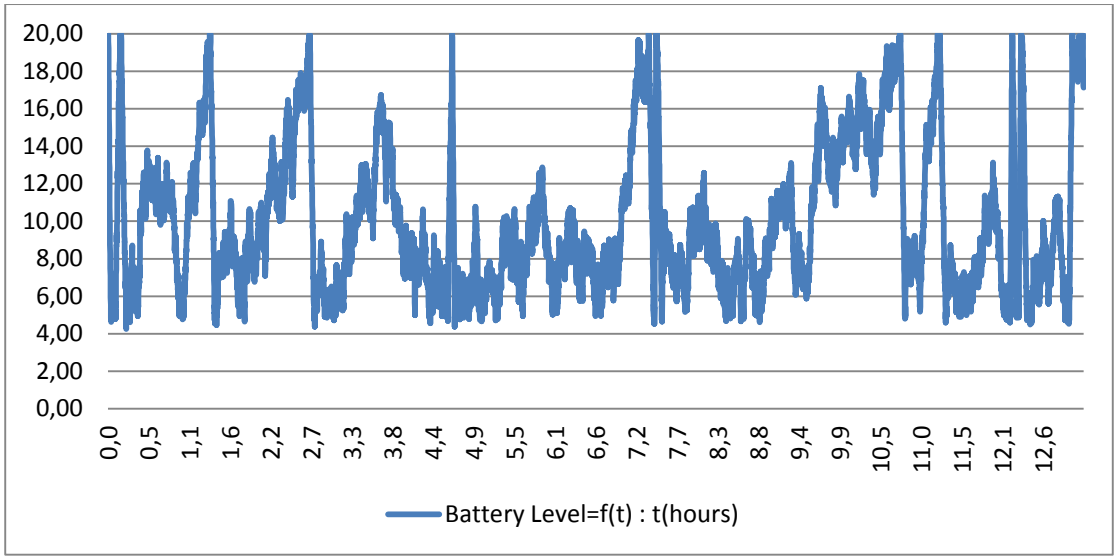


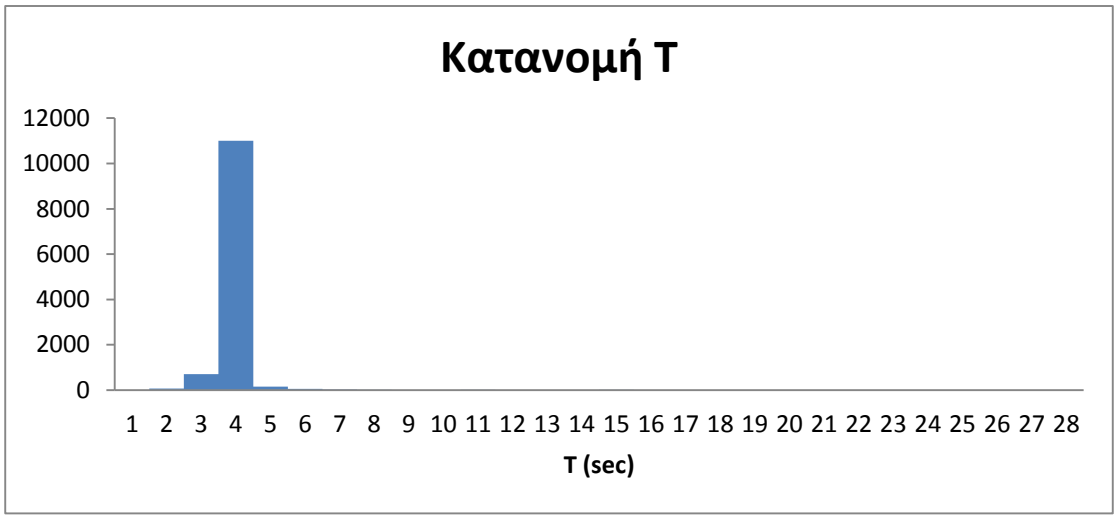


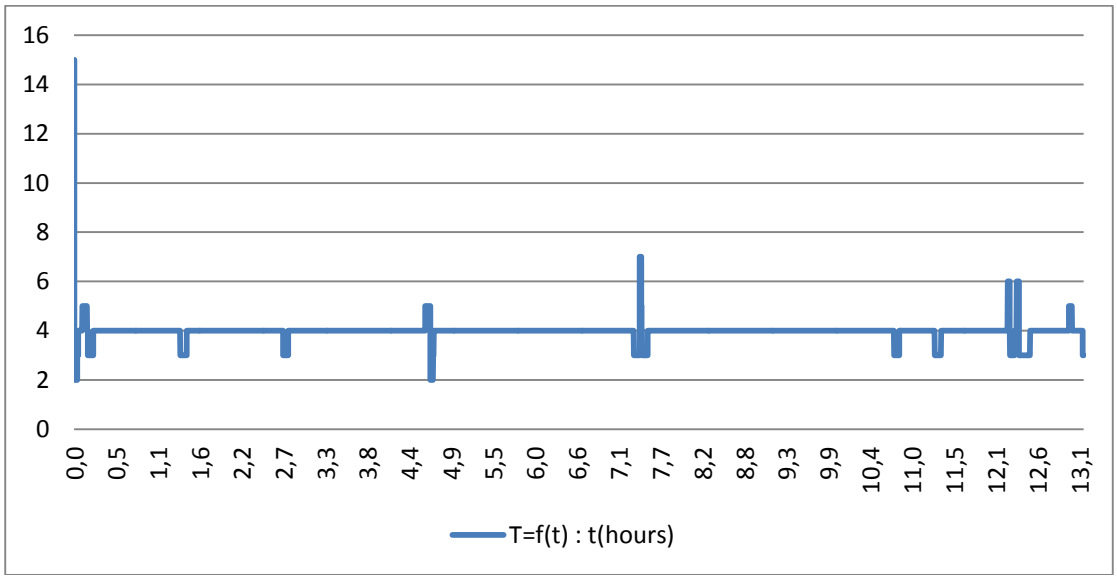
Για α=0. 6 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

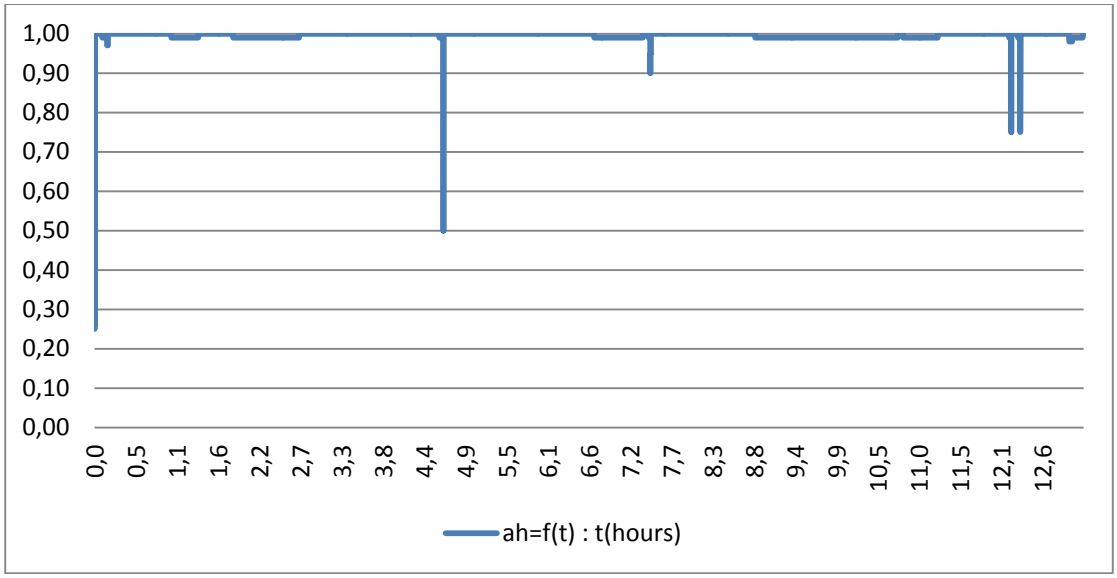
Μέση τιμή level μπαταρίας	9,97
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	3,999495
Χαμένα πακέτα	119
Ποσοστό χαμένων πακέτων	0,009917
Μέση τιμή ah	0,99600

Χαμένα πακέτα			
2 στη σειρά	3 στη σειρά	4 στη σειρά	5 στη σειρά
2	0	0	0


20,00
18,00
16,00
14,00
12,00
10,00
8,00
6,00
4,00
2,00
0,00
0,0 0,5 1,1 1,6 2,2 2,7 3,3 3,8 4,4 4,9 5,5 6,1 6,6 7,2 7,7 8,3 8,8 9,4 9,9 10,5 11,0 11,5 12,1 12,6
Battery Level=f(t) : t(hours)


Κατανομή T
12000
10000
8000
6000
4000
2000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
T (sec)

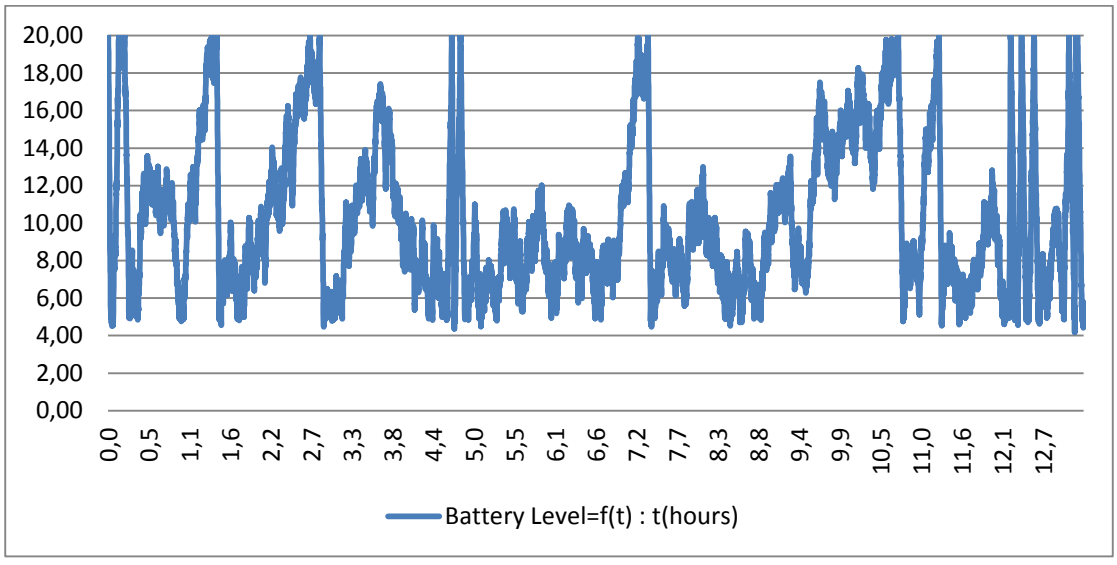

16
14
12
10
8
6
4
2
0
0,0 0,5 1,1 1,6 2,2 2,7 3,3 3,8 4,4 4,9 5,5 6,0 6,6 7,1 7,7 8,2 8,8 9,3 9,9 10,4 11,0 11,5 12,1 12,6 13,1
T=f(t) : t(hours)



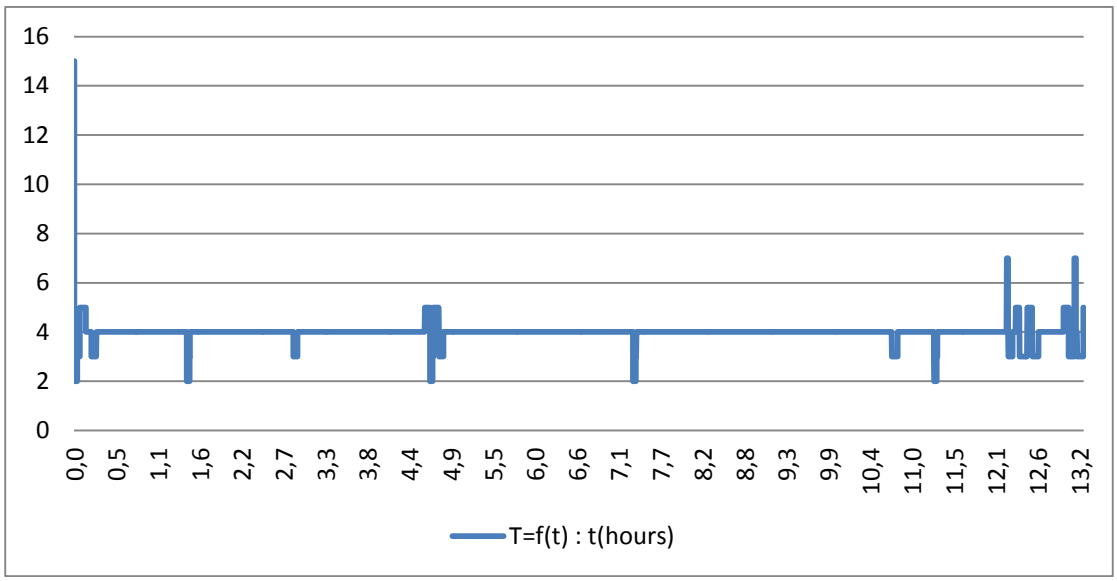
Για α=0. 8 και λ=0.25 και αρχική τιμή του T=15sec έχουμε:

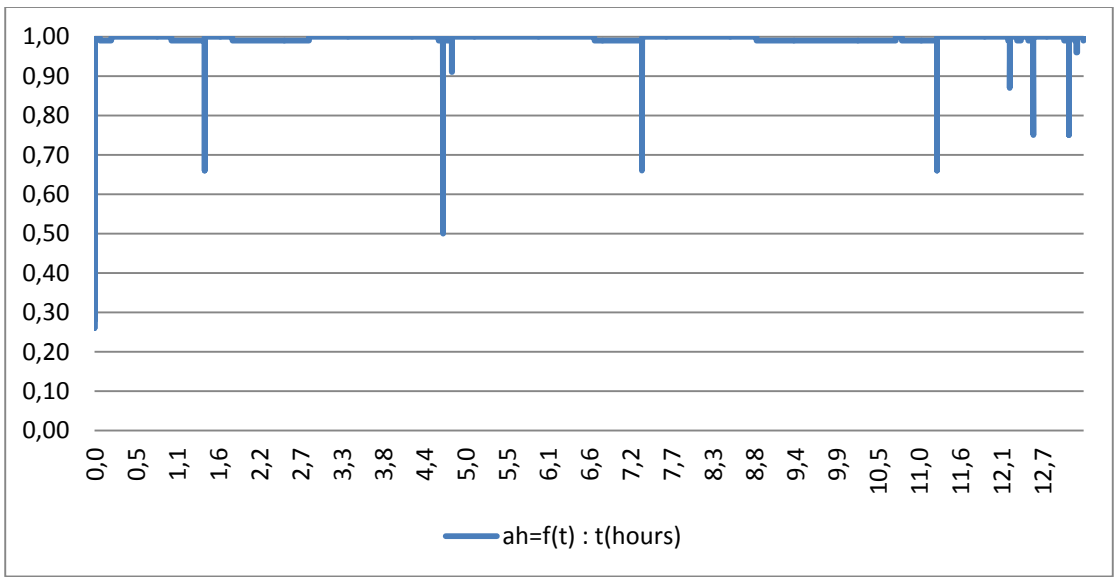
Μέση τιμή level μπαταρίας	10,24
Μέση τιμή μεταξύ των χρόνων μετάδοσης T	3,995711
Χαμένα πακέτα	109
Ποσοστό χαμένων πακέτων	0,009083
Μέση τιμή ah	0,99581

Χαμένα πακέτα			
2 στη σειρά	3 στη σειρά	4 στη σειρά	5 στη σειρά
2	0	0	0




Κατανομή T
12000
10000
8000
6000
4000
2000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
T (sec)


16
14
12
10
8
6
4
2
0
0,0 0,5 1,1 1,6 2,2 2,7 3,3 3,8 4,4 4,9 5,5 6,0 6,6 7,1 7,7 8,2 8,8 9,3 9,9 10,4 11,0 11,5 12,1 12,6 13,2
T=f(t) : t(hours)


1,00
0,90
0,80
0,70
0,60
0,50
0,40
0,30
0,20
0,10
0,00
0,0 0,5 1,1 1,6 2,2 2,7 3,3 3,8 4,4 5,0 5,5 6,1 6,6 7,2 7,7 8,3 8,8 9,4 9,9 10,5 11,0 11,6 12,1 12,7
ah=f(t) : t(hours)

4.3.3 Αποτελέσματα Μετρήσεων

Παρατηρούμε ότι όσο αυξάνει το α τόσο πιο σταθερό γίνεται το σύστημα και έχουμε λιγότερο αριθμό χαμένων πακέτων. Βέβαια από κάποια τιμή του α και πάνω το σύστημα τείνει να γίνει λιγότερο ευσταθές πάλι. Η μέση τιμή του ah για σχετικά μικρές τιμές του α είναι πιο μικρή σχετικά με μεγαλύτερες τιμές του α αλλά ακόμα και τότε πολύ ικανοποιητική. Η μέση τιμή μεταξύ των χρόνων μετάδοσης τείνει να είναι πολύ κοντά στο 4 σε όλες τις περιπτώσεις πράγμα αναμενόμενο λόγω του ότι το λ ισούται με 0,25 επομένως χρειάζονται 4 δευτερόλεπτα για την ικανοποίηση της κατανάλωσης της μετάδοσης η οποία ισούται με 1. Η κατανομή του Τ έχει μικρότερη διασπορά για μεγαλύτερες τιμές του α. Το level της μπαταρίας, χρονικά, γεμίζει και αδειάζει και οι μεταβολές αυτές είναι πιο γρήγορες όσο πιο μικρό α έχουμε. Από τις μετρήσεις φαίνεται ότι η τιμή 0.6 και 0.8 για το α είναι μια άριστη επιλογή.

5. Βιβλιογραφία

1. The C Programming Language by Brian Kernighan and Dennis Ritchie
2. eZ430-RF2500 Development Tool User's Guide
   http://www.ti.com/lit/ug/slau227e/slau227e.pdf
3. Texas Instruments official site: http://www.ti.com και πιο συγκεκριμένα
   http://www.ti.com/tool/ez430-rf2500