

Ατομική Διπλωματική Εργασία

**ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ MINT ΣΕ
ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ**

Παναγιώτα Γιαννή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

**Πειραματική Αξιολόγηση του Αλγορίθμου MINT σε Ασύρματα Δίκτυα
Αισθητήρων**

Παναγιώτα Γιαννή

Επιβλέπων Καθηγητής
Δημήτρης Ζεϊναλιπούρ

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των
απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του
Πανεπιστημίου Κύπρου

Μάιος 2010

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής μου εργασίας θα ήθελα να εκφράσω τις ευχαριστίες μου προς όλα τα άτομα που με βοήθησαν και με στήριξαν όλα αυτά τα χρόνια στο Πανεπιστήμιο Κύπρου.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επιβλέπων καθηγητή μου, κύριο Δημήτρη Ζεϊναλιπούρ, για την πολύτιμη βοήθεια και καθοδήγησή του, για την ολοκλήρωση και συγγραφή της παρούσας διπλωματικής εργασίας.

Επιπρόσθετα, θα ήθελα να εκφράσω τις ευχαριστίες μου προς τον διδακτορικό φοιτητή κύριο Παναγιώτη Αντρέου για τις ουσιαστικές του υποδείξεις, καθώς και για τη δημιουργική συνεργασία που είχαμε.

Θα ήθελα επίσης, να ευχαριστήσω την οικογένεια μου και τους φίλους μου, για την αγάπη, τη ψυχολογική στήριξη και την ενθάρρυνση που μου παρείχαν για όλη τη διάρκεια της φοίτησης μου.

Τελειώνοντας, ιδιαίτερες ευχαριστίες θα ήθελα να απευθύνω στον αδερφό μου, που ήταν πάντοτε δίπλα μου, με στήριζε σε κάθε δυσκολία και στάθηκε ο κύριος αρωγός για να επιτύχω τους στόχους, τις φιλοδοξίες μου καθώς και για να πραγματοποιήσω το παρόν όνειρο.

Περίληψη

Η παρούσα διπλωματική εργασία έχει ως σκοπό την πειραματική αξιολόγηση του αλγόριθμου MINT (Materialized In-Network Top-k Views) στα ασύρματα δίκτυα αισθητήρων.

Ο αλγόριθμος που μελετάμε ασχολείται με τη βελτιστοποίηση της εκτέλεσης ερωτημάτων συνεχούς παρακολούθησης σε δίκτυα αισθητήρων και αποσκοπεί στην μείωση της επικοινωνίας μέσα στο δίκτυο. Συγκεκριμένα, επικεντρώνεται στην αποτελεσματική ανάκτηση των k υψηλότερων σε κατάταξη (ή Top- k) απαντήσεων. Η γενική φιλοσοφία στην οποία βασίζεται ο MINT είναι η επεξεργασία των αποτελεσμάτων μέσα στο δίκτυο, πριν την προώθησή τους προς το υπόλοιπο δίκτυο. Ο απώτερος σκοπός αυτής της ενδοδίκτυας επεξεργασίας που υλοποιεί ο αλγόριθμος, είναι η ελαχιστοποίηση του κόστους της αξιολόγησης του ερωτήματος στο δίκτυο, με αποτέλεσμα τη μείωση της καταναλισκόμενης ενέργειας και κατά συνέπεια τη μεγιστοποίηση του χρόνου ζωής του δικτύου.

Για να αξιολογήσουμε τις αποδόσεις του αλγόριθμου Mint χρειάστηκε να γνωρίζουμε τη δομή των πακέτων που μεταφέρονται καθώς και το πώς γίνεται η επικοινωνία στο λειτουργικό σύστημα TinyOS. Για την πειραματική αξιολόγηση χρησιμοποιήθηκε ο TOSSIM που είναι ένας προσομοιωτής διακριτών συμβάντων (discrete event) ειδικά σχεδιασμένος για το TinyOS μαζί με το TinyViz που είναι το εργαλείο απεικόνισης του TOSSIM. Επιπλέον, για τον υπολογισμό της κατανάλωσης ενέργειας ανά κόμβο έγινε χρήση του PowerTOSSIM που είναι μια επέκταση του προσομοιωτή. Επίσης, σε συνδυασμό με τον προσομοιωτή χρησιμοποιήθηκε το TinyDB, το σύστημα επεξεργασίας επερωτήσεων για την εξαγωγή πληροφοριών από ένα TinyOS δίκτυο αισθητήρων.

Με βάση το πιο πάνω περιβάλλον, αξιολογήσαμε τον αλγόριθμο MINT συγκρίνοντας τον με τρεις άλλους αλγόριθμους για να αποδείξουμε ότι ο αλγόριθμος αυτός μειώνει σημαντικά τα ποσά κατανάλωσης ενέργειας. Οι αλγόριθμοι που χρησιμοποιήθηκαν για σύγκριση είναι ο TAG, ο TINA και ο INT.

Για να αποδείξουμε την αποτελεσματικότητα των αλγορίθμων πραγματοποιήσαμε πέντε πειραματικές σειρές, μέσα από τις οποίες αποκαλύπτεται πως ο αλγόριθμος MINT επιτυγχάνει σημαντική μείωση της καταναλισκόμενης ενέργειας σε σύγκριση με τους υπόλοιπους αλγορίθμους, με αποτέλεσμα να επιμηκύνεται ο χρόνος ζωής του δικτύου.

Περιεχόμενα

Κεφάλαιο 1	Εισαγωγή.....	1
1.1.	Εισαγωγή στο Αντικείμενο Μελέτης.....	1
1.2.	Περιγραφή Αισθητήρα.....	1
1.3.	Ορισμός Ασύρματου Δικτύου Αισθητήρων.....	3
1.4.	Επισκόπηση Κεφαλαίων.....	8
Κεφάλαιο 2	Εισαγωγικό Πλαίσιο Αλγόριθμου Mint.....	9
2.1.	Εισαγωγή στον Αλγόριθμο MINT	9
2.2.	Μοντέλο Συστήματος και Ορολογία.....	14
2.3.	Συσχετιζόμενη Εργασία.....	15
2.4.	Πλαίσιο MINT	16
2.4.1.	Φάση Δημιουργίας (Creation Phase).....	16
2.4.2.	Φάση Κλαδέματος (Pruning Phase).....	18
2.4.3.	Φάση Ενημέρωσης (Update Phase).....	21
Κεφάλαιο 3	Περιβάλλον αξιολόγησης του MINT.....	23
3.1.	Εισαγωγή.....	23
3.2.	Δομή Πακέτων και Επικοινωνία στο TinyOS.....	23
3.2.1.	Γενική Δομή Πακέτων TinyOS.....	27
3.3.	Επικοινωνία στο TinyOS.....	29
3.3.1.	Active Message(AM).....	30
3.4.	TinyDB: Μια δηλωτική Βάση Δεδομένων για το TinyOS.....	35
3.4.1.	Αρχιτεκτονική Λογισμικού TinyDB.....	37
3.5.	TOSSIM [16]: Ένας προσομοιωτής για το TinyOS	39
3.5.1.	Αρχιτεκτονική TOSSIM.....	39
3.5.2.	Μεταγλώττιση και Εκτέλεση TOSSIM.....	44
3.5.3.	Χρήση LossyBuilder για τη Δημιουργία του Αρχείου lossy.....	44
3.5.4.	Γραφικό περιβάλλον (GUI).....	45

3.5.4.1. Μεταγλώττιση και Εκτέλεση TinyViz.....	46
3.6. PowerTOSSIM.....	47
3.6.1. Προσομοίωση της Κατανάλωσης Ισχύος σε Εφαρμογές μεγάλης κλίμακας δικτύου αισθητήρων.....	47
3.6.2. Συσκευές οι οποίες καταναλώνουν ενέργεια.....	48
3.6.3. Η Αρχιτεκτονική του PowerTOSSIM.....	48
3.6.4. Αποτίμηση του PowerTOSSIM.....	51
3.7. Μέτρηση ενέργειας.....	52
Κεφάλαιο 4 Πειραματική Μεθοδολογία Αξιολόγησης του MINT.....	56
4.1. Εισαγωγή.....	56
4.2. Πειραματική Μεθοδολογία	57
4.3. Σύνολα Δεδομένων.....	62
4.4. Συσκευή Αισθητήρα (Sensing Device).....	63
4.5. Τοπολογίες Multi-hop.....	63
4.6. Πρωτόκολλο Επικοινωνίας.....	64
4.7. Σύνταξη Επερώτησης.....	65
Κεφάλαιο 5 Πειραματικά Αποτελέσματα.....	67
5.1. Εισαγωγή.....	67
5.2. Πειραματική Σειρά 1: Κατανάλωση Ενέργειας.....	68
5.3. Πειραματική Σειρά 2: Αποκοπή Πλειάδων.....	74
5.4. Πειραματική Σειρά 3: Ευελιξία σε σχέση με το k.....	75
Πειραματική Σειρά 4: Πληθικότητα με GROUP-BY.....	77
5.5. Πειραματική Σειρά 5: Διάρκεια ζωής Δικτύου.....	79
Κεφάλαιο 6 Συμπεράσματα	82
6.1. Συμπεράσματα.....	82
6.2. Μελλοντική Εργασία.....	82
Βιβλιογραφία	83

Π α ρ ά ρ τ η μ α Α Τεχνικό Υπόβαθρο : Το λειτουργικό σύστημα TinyOS.....Α-30

Π α ρ ά ρ τ η μ α Β Τεχνικό Υπόβαθρο : TinyDB.....Β-14

Π α ρ ά ρ τ η μ α Γ Τεχνικό Υπόβαθρο : KSPOT.....Γ- 9

Π α ρ ά ρ τ η μ α Δ Κώδικας MINT.....Δ-13

Κεφάλαιο 1

Εισαγωγή

- 1.1. Εισαγωγή στο Αντικείμενο Μελέτης
 - 1.2. Περιγραφή Αισθητήρα
 - 1.3. Ορισμός Ασύρματου Δικτύου Αισθητήρων
 - 1.4. Επισκόπηση Κεφαλαίων
-

1.1. Εισαγωγή στο Αντικείμενο Μελέτης

Η πρόσφατη πρόοδος στην τεχνολογία των μικρό-ηλεκτρομηχανικών συστημάτων, στις ασύρματες επικοινωνίες και στα ψηφιακά ηλεκτρονικά συστήματα έχει οδηγήσει σε μια επανάσταση η οποία έχει μεταβάλει τον τρόπο με τον οποίο οι άνθρωποι σκέφτονται και επικοινωνούν. Η σύγχρονη τάση για ολοένα μεγαλύτερη ολοκλήρωση και μείωση του μεγέθους έχει δώσει την δυνατότητα για ανάπτυξη πολλών συσκευών χαμηλού κόστους και ισχύος που υλοποιούν ένα πλήθος εφαρμογών. Μέσα σε αυτή την τάση αναδεικνύεται και η ανάπτυξη των ασύρματων προσωπικών δικτύων και των ασύρματων δικτύων αισθητήρων που δίνουν τη δυνατότητα ανάπτυξης δύσκολων εφαρμογών.

1.2. Περιγραφή Αισθητήρα

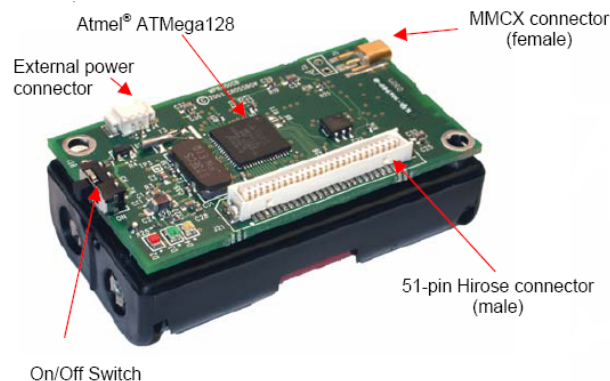
Ένας αισθητήρας είναι ένας ασύρματος υπολογιστής ο οποίος τροφοδοτείται από μπαταρία. Η μεγάλη διαφορά του από ένα τυπικό φορητό υπολογιστή είναι η ισχύς του, λίγες δεκάδες miliwatt σε σχέση με τις δεκάδες των watt του φορητού υπολογιστή καθώς και το φυσικό του μέγεθος που είναι πολύ μικρό, λίγα κυβικά εκατοστά.

Οι αισθητήρες έχουν τη δυνατότητα να συλλέγουν πληροφορίες από μετρήσεις διαφόρων φυσικών μεγεθών όπως η θερμοκρασία, η φωτεινότητα, η υγρασία, και άλλα. Μεγάλο τους πλεονέκτημα αποτελεί το γεγονός ότι μπορούν να φιλτράρουν τα δεδομένα τους ή να τα διαμοιράζονται και να τα συνδυάζουν με άλλους αισθητήρες που βρίσκονται σε κοντινή απόσταση. Επομένως, μπορούν να δημιουργήσουν ένα ασύρματο δίκτυο από αισθητήρες.

Η διαφορά των δικτύων αισθητήρων από τα υπόλοιπα ασύρματα δίκτυα είναι ότι περιέχουν δεκάδες ή εκατοντάδες αυτόνομες συσκευές οι οποίες έχουν την ικανότητα να λειτουργούν χωρίς ανθρώπινη παρέμβαση για εβδομάδες ή και για μήνες.

Ο κάθε κόμβος εκτός από τον αισθητήρα περιλαμβάνει έναν μικροεπεξεργαστή και έναν αναμεταδότη. Ο μικροεπεξεργαστής επεξεργάζεται και αποθηκεύει τα δεδομένα που λαμβάνει από τον αισθητήρα ενώ ο αναμεταδότης μεταδίδει την πληροφορία προς τους άλλους κόμβους του δικτύου.

Οι αισθητήρες χρησιμοποιούνται για να μετρήσουν διάφορες παραμέτρους που μπορούν να μεταβάλλονται στο χώρο και στο χρόνο. Ως συνακόλουθο, απαιτείται ένας μεγάλος αριθμός από αισθητήρες για να λαμβάνουν μετρήσεις σε διαφορετικές θέσεις και χρονικές στιγμές. Οι αισθητήρες πρέπει να οργανώνονται σε δίκτυο, για να διευκολύνεται η μετάδοση των μετρήσεων στο κέντρο συλλογής για την επεξεργασία των πληροφοριών και την εξαγωγή αποτελεσμάτων που θα οδηγήσουν στη λήψη αποφάσεων. Το κέντρο συλλογής επικοινωνεί με τον κόμβο διαχείρισης της εφαρμογής.



Σχήμα 1.2.1 : Συσκευή Κόμβου Αισθητήρα [1]

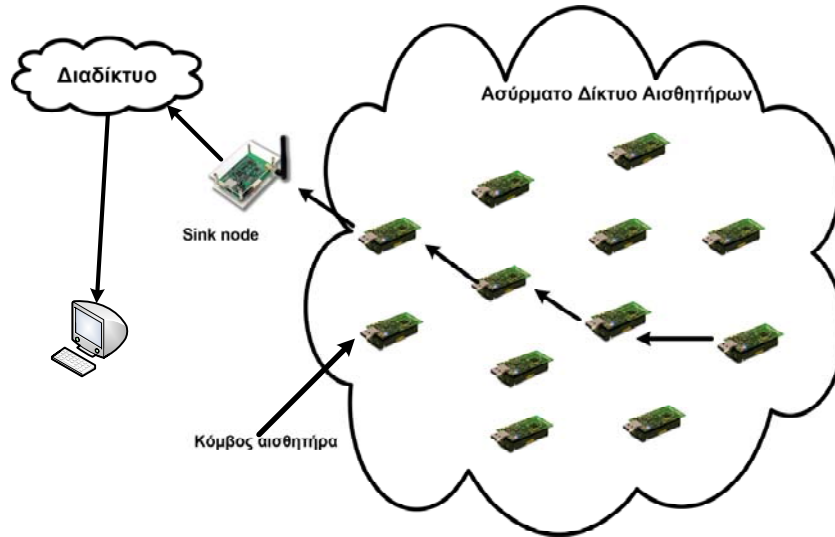
1.3. Ορισμός Ασύρματου Δικτύου Αισθητήρων [2]

Τα ασύρματα δίκτυα αισθητήρων (sensor network) υλοποιούνται με την ανάπτυξη και συνεργασία πολλών αισθητήρων πεπερασμένης διαθέσιμης ενέργειας οι οποίοι είναι καταναμεμημένοι σε μια καθορισμένη γεωγραφική περιοχή χωρίς καμία εκ των προτέρων εγκατεστημένη δικτυακή υποδομή και γνώση της τοπολογίας. Επομένως, οι αισθητήρες μπορούν να τοποθετούνται σε υψηλή πυκνότητα είτε μέσα στο προς παρατήρηση φαινόμενο είτε πολύ κοντά σε αυτό. Η θέση των αισθητήρων δεν είναι απαραίτητο να είναι προκαθορισμένη, επιτρέποντας την τυχαία τοποθέτησή τους έτσι ώστε να μπορούν να εγκατασταθούν σε απρόσιτες τοποθεσίες οι οποίες δεν είναι προσπελάσιμες από τον άνθρωπο.

Εξαιτίας της τυχαίας τοποθέτησής τους, τα πρωτόκολλα και οι αλγόριθμοι των δικτύων αισθητήρων θα πρέπει να παρέχουν στο δίκτυο τη δυνατότητα να οργανώνεται από μόνο του (self-organization). Επιπρόσθετα, ένα σημαντικό χαρακτηριστικό που συναντούμε στα δίκτυα αισθητήρων είναι η συλλογική προσπάθεια των αισθητήρων και η συνεχής επικοινωνία μεταξύ τους. Ο κάθε αισθητήρας έχει προσαρμοσμένο πάνω του έναν επεξεργαστή, τον οποίο χρησιμοποιεί για να επεξεργάζεται μερικώς τα δεδομένα που συλλέγει και δέχεται από τους γειτονικούς αισθητήρες (multi-hop routing), πριν τα μεταδώσει στον τελικό τους προορισμό που είναι το κέντρο συλλογής (sink) ο οποίος έχει απεριόριστο αποθηκευτικό χώρο και ενέργεια.

Είναι σημαντικό να αναφερθεί ότι στα ασύρματα δίκτυα αισθητήρων υπάρχουν κάποιες προκλήσεις όπως είναι η μικρή υπολογιστική δύναμη, η ιδιαίτερα υψηλές απαιτήσεις για συγχρονισμό λόγω του ότι η ροή της πληροφορίας έρχεται από πολλές πηγές και λόγω της μικρής μνήμης όπου δεν μπορεί να γίνει buffering, οπότε πρέπει να επεξεργαστούν γρήγορα τα μηνύματα που δέχονται αλλιώς μπορεί να χαθούν. Επιπρόσθετα, μια άλλη πρόκληση είναι το μικρό μέγεθος του συστήματος καθώς και η σχεδίαση που πρέπει να είναι modular για να μπορούμε να φτιάξουμε γρήγορα και εύκολα εφαρμογές [3]. Κατά τη συλλογή των δεδομένων από τους κόμβους του δικτύου καθοριστικό ρόλο διαδραματίζει η κατανάλωση ενέργειας σε κάθε κόμβο, η οποία πρέπει να είναι η ελάχιστη δυνατή ώστε να μεγιστοποιείται η ζωή του. Οι πηγές ισχύος που χρησιμοποιούνται είναι κυρίως μπαταρίες λιθίου ή αλκαλικές και συνήθως

δεν υπάρχει δυνατότητα επαναφόρτισης τους. Συνεπώς, για τη μεγιστοποίηση του χρόνου ζωής του δικτύου απαιτείται η ελάχιστη κατανάλωση ενέργειας για κάθε λειτουργία του αισθητήρα.



Σχήμα 1.3.1: Διασπορά αισθητήρων σε ένα πεδίο παρακολούθησης και αποστολή δεδομένων σε ένα σταθμό βάσης.

Ένα τυπικό χαρακτηριστικό αυτών των δικτύων είναι ότι οι κόμβοι διατηρούν ή παράγουν μεγάλες ποσότητες δεδομένων. Για παράδειγμα, μια τυπική συσκευή αισθητήρα μπορεί να λάβει δείγμα για περιβαλλοντικές παραμέτρους, όπως η πίεση, η υγρασία ή η θερμοκρασία σε ποσοστά που κυμαίνονται από 2Hz μέχρι 500Hz. Έτσι, το έργο των ερωτήσεων είναι μια συλλογή από αισθητήρες που είναι αντιμέτωπη με τη θεμελιώδη πρόκληση που κατανέμεται σε ένα ερώτημα επεξεργασίας: *"Πώς να βρείτε την αναμενόμενη απάντηση, χωρίς να τηρήσετε όλες τις σχέσεις που κατανέμονται στο σύνολο"*.

Σε προηγούμενη εργασία μελετήθηκε το θέμα της επεξεργασίας επερωτήσεων σε στατικά δίκτυα αισθητήρων με σκοπό την μείωση της επικοινωνίας στο δίκτυο και κατά συνέπεια της κατανάλωσης ισχύος. Στα πλαίσια της εργασίας μελετήθηκε πειραματικά το σύστημα *"KSPOT: Αποδοτική Παρακολούθηση των Κορυφαίων-K Απαντήσεων σε Ασύρματα Δίκτυα Αισθητήρων"*, το οποίο μπορεί να χρησιμοποιηθεί για την αποδοτική παρακολούθηση των Κορυφαίων-K (Top-K) απαντήσεων σε ένα στατικό ασύρματο

δίκτυο αισθητήρων. Το KSPOT αποτελείται από ένα σύστημα κατάταξης και μία γραφική διεπιφάνεια χρήσης. Το σύστημα κατάταξης αξιοποιεί κατανεμημένους αλγόριθμους επεξεργασίας Κορυφαίων-Κ απαντήσεων (MINT αλγόριθμος [4]) έτσι ώστε να ελαχιστοποιήσει την κατανάλωση των περιορισμένων πόρων που έχει ένα ασύρματο δίκτυο αισθητήρων επιτυγχάνοντας έτσι την μακροζωία του. [36]

Η εργασία αυτή επομένως, στοχεύει στην πειραματική αξιολόγηση του αλγορίθμου MINT που χρησιμοποιείται στο σύστημα KSPOT. Όπως γνωρίζουμε μία από τις βασικές προκλήσεις σε αυτή τη νέα εποχή των δικτύων αισθητήρων, είναι η ανάκτηση των αναγνώσεων από έναν αισθητήρα χρησιμοποιώντας έναν αλγόριθμο που να λαμβάνει υπόψη την μείωση της ενέργειας. Στις παραδοσιακές τεχνικές συλλογής δεδομένων, τα δεδομένα από τους αισθητήρες διαβιβάζονται στο sink (σταθμός βάσης ή querying node) αμέσως μετά την απόκτηση τους από το φυσικό κόσμο χωρίς καμία επεξεργασία. Η φιλοσοφία του αλγορίθμου MINT ασχολείται με την επεξεργασία των αποτελεσμάτων μέσα στο δίκτυο πριν την προώθησή τους προς το υπόλοιπο δίκτυο. Με την ιδέα αυτή παρά το γεγονός ότι μέσα στο δίκτυο μειώνεται σημαντικά η κατανάλωση ενέργειας, αγνοείται η διαβίβαση όλων των αποτελεσμάτων των ερωτημάτων από όλους τους αισθητήρες σε κάθε γύρο.

Για την αξιολόγηση του αλγορίθμου MINT, πραγματοποιήθηκαν πολλαπλές προσομοιώσεις με τη χρήση διαφόρων εργαλείων του λειτουργικού TinyOS [30], όπως το TinyDB [13], ο προσομοιωτής TOSSIM [18], το TinyViz [12,17] και το PowerTOSSIM [19], καθώς επίσης χρειάστηκε να γνωρίζουμε τη δομή των πακέτων που μεταφέρονται καθώς και το πώς γίνεται η επικοινωνία στο λειτουργικό σύστημα TinyOS.

Το TinyOS είναι ένα νέο είδος ανοικτού κώδικα λειτουργικό σύστημα που αποτελεί ένα από τα πρώτα περιβάλλοντα λογισμικού που έχουν σχεδιαστεί για να ικανοποιήσουν τις απαιτήσεις των κατανεμημένων ενσωματωμένων συστημάτων που έχουν περιορισμένους πόρους. Σχεδιάστηκε στο Πανεπιστήμιο του Berkeley και από τότε έχει γίνει ένα από τα πιο δημοφιλή λειτουργικά συστήματα για ασύρματα δίκτυα αισθητήρων. Το λειτουργικό αυτό, είναι βασισμένο στην αρχιτεκτονική των components, η οποία κάνει δυνατή τη γρήγορη υλοποίηση των εφαρμογών, ενώ

ταυτόχρονα ελαχιστοποιεί το μέγεθος του κώδικα που απαιτείται καθώς και την κατανάλωση ενέργειας στους αισθητήρες.

Για τη διεξαγωγή των προσομοιώσεων, χρησιμοποιήθηκε ο TOSSIM που είναι ένας προσομοιωτής διακριτών συμβάντων (discrete event) ειδικά σχεδιασμένος για το TinyOS και έχει την δυνατότητα να αντιλαμβάνεται και να αναπαριστά την συμπεριφορά και τις αλληλεπιδράσεις σε δίκτυα χιλιάδων κόμβων (moten), μαζί με το TinyViz που είναι το εργαλείο απεικόνισης του TOSSIM. Η πειραματική αξιολόγηση πραγματοποιήθηκε πάνω σε τρία διαφορετικά datasets. Επιπλέον, για τον υπολογισμό της κατανάλωσης ενέργειας ανά κόμβο έγινε χρήση του PowerTOSSIM που είναι μια επέκταση του προσομοιωτή. Το PowerTOSSIM, είναι ένα εξελικτικό περιβάλλον προσομοίωσης για ασύρματα δίκτυα αισθητήρων που παρέχει μια ακριβή εκτίμηση της κατανάλωσης ενέργειας ανά κόμβο. Μέσω των οργάνων μέτρησης των πραγματικών κόμβων αισθητήρων, έχει αποδειχτεί ότι το PowerTOSSIM παρέχει ακριβή εκτίμηση της κατανάλωσης ενέργειας για ένα ευρύ φάσμα εφαρμογών και πινάκων για την υποστήριξη πολύ μεγάλων προσομοιώσεων. Το PowerTOSSIM έχει δείξει να είναι περισσότερο από 90% ακριβές.

Επίσης, σε συνδυασμό με τον προσομοιωτή χρησιμοποιήθηκε το TinyDB, το σύστημα επεξεργασίας επερωτήσεων για την εξαγωγή πληροφοριών από ένα TinyOS δίκτυο αισθητήρων.

Με βάση το πιο πάνω περιβάλλον, αξιολογήσαμε τον αλγόριθμο MINT συγκρίνοντας τον με τρεις άλλους αλγόριθμους για να αποδείξουμε ότι ο αλγόριθμος αυτός μειώνει σημαντικά τα ποσά κατανάλωσης ενέργειας. Οι αλγόριθμοι που χρησιμοποιήθηκαν για σύγκριση είναι ο TAG, ο TINA και ο INT. Είναι σημαντικό να αναφέρουμε ότι ο αλγόριθμος TAG είναι ήδη υλοποιημένος ως μέρος του πλαισίου TinyDB και χρησιμοποιείται ως βάση για σύγκριση. Οι υπόλοιποι αλγόριθμοι, MINT, TINA και INT έχουν υλοποιηθεί σε γλώσσα nesC, η οποία είναι η γλώσσα προγραμματισμού για το λειτουργικό σύστημα TinyOS, και έχει παρόμοια σύνταξη με τη γλώσσα C. Η nesC είναι βασισμένη στην αρχή της σύνδεσης (wiring) μεταξύ των components για τη δημιουργία μιας εφαρμογής και προορίζεται κυρίως για ενσωματωμένα συστήματα, όπως τα δίκτυα αισθητήρων.

Οι προσομοιώσεις για τους πιο πάνω αλγόριθμους διεξάχθηκαν για τρία σύνολα δεδομένων: Washington State Climate (AtmoMon 32), Great Duck Island (GDI14 2002), Intel Research Berkeley (Intel49).

Για να αποδείξουμε την αποτελεσματικότητα των αλγορίθμων πραγματοποιήσαμε πέντε πειραματικές σειρές. Στην πρώτη πειραματική σειρά έχουμε σύγκριση της κατανάλωσης ενέργειας των αλγορίθμων INT και MINT ως προς τους αλγόριθμους TAG και TINA, δείχνοντας ότι οι αλγόριθμοι INT και MINT επιτυγχάνουν σημαντική εξοικονόμηση ενέργειας σε σύγκριση με τους άλλους δύο. Στη δεύτερη πειραματική σειρά, μελετάμε το μέγεθος των πλειάδων που αποκόπτονται σε κάθε επίπεδο ως προς τον sink, για τους αλγόριθμους INT και MINT. Στην τρίτη πειραματική σειρά, διερευνάται η επεκτασιμότητα της παραμέτρου k δηλαδή μελετάμε την απόδοση του αλγορίθμου για διαφορετικές τιμές του k . Στην τέταρτη πειραματική σειρά διερευνήθηκε η επίδραση της πληθικότητας GROUP-BY. Σε όλα τα σύνολα δεδομένων μοιράζουμε τυχαία με ομοιόμορφο τρόπο τους αισθητήρες σε δωμάτια. Σε αυτό το πείραμα, διανέμουμε τους αισθητήρες σε διάφορες συνθέσεις δωματίων για τη μελέτη της ενεργειακής κατανάλωσης όλων των αλγορίθμων. Τέλος, στην πέμπτη πειραματική σειρά, αξιολογούμε την αποτελεσματικότητα του πλαισίου TinyDB εστιάζοντας στην κατανάλωση ενέργειας και τη διάρκεια ζωής του συστήματος.

Με βάση τα αποτελέσματα των προσομοιώσεων που πραγματοποιήθηκαν αποδεικνύεται πως ο αλγόριθμος MINT επιτυγχάνει σημαντική μείωση της καταναλισκόμενης ενέργειας σε σύγκριση με τους υπόλοιπους αλγορίθμους, με αποτέλεσμα να επιμηκύνεται ο χρόνος ζωής ολόκληρου του δικτύου.

Για την εξαγωγή των αποτελεσμάτων χρειάστηκε να υλοποιηθεί η πειραματική υποδομή, να εκτελεστούν τα πειράματα κάτω από δύσκολες συνθήκες, λόγω προβλημάτων που αντιμετωπίστηκαν εξαιτίας λαθών που υπάρχουν στο λειτουργικό TinyOS και ειδικότερα στον προσομοιωτή TOSSIM και στην γραφική απεικόνισή του, το TinyViz. Επιπλέον, χρειάστηκαν κάποιες μικρές διορθώσεις στο πλαίσιο του αλγορίθμου MINT.

1.4. Επισκόπηση Κεφαλαίων

Η λογική πορεία που ακολουθείται στην εργασία αυτή περιγράφεται παρακάτω.

Στο **Κεφάλαιο 2**, γίνεται λεπτομερής περιγραφή του αλγόριθμου MINT και δίνεται το μοντέλο συστήματος και η ορολογία που χρησιμοποιείται. Επιπλέον, περιγράφονται οι τρεις σημαντικές φάσεις εκτέλεσης του αλγόριθμου οι οποίες αφορούν τη φάση δημιουργίας, κλαδέματος και ενημέρωσης.

Στο **Κεφάλαιο 3**, περιγράφεται το Περιβάλλον Αξιολόγησης του MINT, η βασική δομή των πακέτων που χρησιμοποιούνται στο TinyOs και ειδικότερα στον αλγόριθμο μας καθώς επίσης και ο τρόπος με τον οποίο γίνεται η επικοινωνία των αισθητήρων στο δίκτυο. Δίνεται επίσης αναλυτική περιγραφή των διαφόρων εργαλείων που χρησιμοποιήθηκαν για την αξιολόγηση του αλγόριθμου για να εισάγουμε το περιβάλλον στο οποίο αξιολογήθηκε ο αλγόριθμος.

Στο **Κεφάλαιο 4**, περιγράφεται η πειραματική μεθοδολογία που ακολουθήθηκε. Δίνονται σημαντικές πληροφορίες οι οποίες σχετίζονται με την πειραματική εργασία την οποία εκτελέσαμε καθώς και οι τρεις αλγόριθμοι οι οποίοι έχουν χρησιμοποιηθεί για σύγκριση με τον MINT για την εξαγωγή συμπερασμάτων.

Στο **Κεφάλαιο 5**, επεξηγούνται αναλυτικά τα αποτελέσματα των διαφόρων πειραμάτων για την αξιολόγηση του αλγόριθμου, και αναπαριστούνται μέσω γραφικών παραστάσεων για καλύτερη κατανόηση.

Στο **Κεφάλαιο 6** παρουσιάζονται τα συμπεράσματα τα οποία προκύπτουν από την παρούσα έρευνα και η μελλοντική εργασία η οποία μπορεί να γίνει στο θέμα αυτό.

Κεφάλαιο 2

Εισαγωγικό Πλαίσιο Αλγόριθμου MINT

- 2.1. Αλγόριθμος MINT
 - 2.2. Μοντέλο Συστήματος και Ορολογία
 - 2.3. Συσχετιζόμενη Εργασία
 - 2.4. Πλαίσιο MINT
-

2.1. Εισαγωγή στον Αλγόριθμο MINT [4,5]

Μία από τις βασικές προκλήσεις σε αυτή τη νέα εποχή των δικτύων αισθητήρων είναι η ανάκτηση των αναγνώσεων από έναν αισθητήρα χρησιμοποιώντας έναν αλγόριθμο που να λαμβάνει υπόψη την μείωση της ενέργειας. Στις παραδοσιακές τεχνικές συλλογής δεδομένων, τα δεδομένα από τους αισθητήρες διαβιβάζονται στο sink (σταθμός βάσης ή querying node) αμέσως μετά την απόκτηση τους από το φυσικό κόσμο, χωρίς καμία επεξεργασία. Η φιλοσοφία του αλγόριθμου MINT ασχολείται με την επεξεργασία των αποτελεσμάτων μέσα στο δίκτυο πριν την προώθηση τους προς το υπόλοιπο δίκτυο. Με την ιδέα αυτή, παρά το γεγονός ότι, μέσα στο δίκτυο μειώνεται σημαντικά η κατανάλωση ενέργειας, αγνοείται η διαβίβαση των αποτελεσμάτων όλων των ερωτημάτων από όλους τους αισθητήρες σε κάθε γύρο. Αυτό έχει ως αποτέλεσμα, να μειώνεται σημαντικά η επικοινωνία στο δίκτυο, η οποία αποτελεί το σημαντικότερο παράγοντα κατανάλωσης ενέργειας.

Ο αλγόριθμος *MINT* (*Materialized In-Network Top-k View*), είναι ένα νέο πλαίσιο (framework) για τη βελτιστοποίηση της εκτέλεσης των ερωτημάτων συνεχούς παρακολούθησης σε δίκτυα αισθητήρων. Ένα τυπικό Materialized View V διατηρεί τα πλήρη αποτελέσματα ενός ερωτήματος Q , προκειμένου να ελαχιστοποιηθεί το κόστος των μελλοντικών εκτελέσεων της επερώτησης. Σε ένα δίκτυο αισθητήρων, διατηρείται

η συνοχή μεταξύ των V . Επίσης, η κατανομημένη σχέση βάσης R είναι πολύ ακριβή από την άποψη της επικοινωνίας. Έτσι, η προσέγγισή επικεντρώνεται σε ένα υποσύνολο $V'(\subseteq V)$ το οποίο αποκαλύπτει μόνο τις k υψηλότερες σε κατάταξη απαντήσεις στο sink για κάποιο χρήστη που ορίζει την παράμετρο k . Επιπλέον, παρέχεται μια περιγραφή για την κατασκευή του pruning (αποκοπή περιττών αποτελεσμάτων) και τη διατήρηση των εμφανίσεων που ορίζονται αναδρομικά στο δίκτυο. Η πειραματική μελέτη που παρουσιάζεται στη διατριβή αυτή, με πραγματικά σύνολα δείχνει ότι ο MINT προσφέρει σημαντικές μειώσεις ενέργειας σε σύγκριση με άλλα μοντέλα που υπερσχύουν στην απόκτηση δεδομένων.

Για το μοντέλο ανάκτησης των δεδομένων, έχουμε υπόψη ότι ο χρήστης ενδιαφέρεται μόνο για τις k υψηλότερες σε κατάταξη απαντήσεις και όχι για όλες. Προτάθηκαν MINT Views, για την ελαχιστοποίηση των μηνυμάτων με αποτέλεσμα τη μείωση της κατανάλωσης ενέργειας κατά την εκτέλεση των ερωτημάτων συνεχούς παρακολούθησης. Όπως και σε άλλα πλαίσια (framework), υποστηρίζονται επερωτήματα, αλλά εστιάζεται και στη βελτιστοποίηση των top- k επερωτήσεων πάνω από τις πολλαπλές πλειάδες απαντήσεων. Τέτοιες απαντήσεις είναι πολύ τυπικές για ερωτήματα με GROUP BY ή για μη συνολικά ερωτήματα.

Ένα view V σε σχεσιακές βάσεις δεδομένων είναι ένας εικονικός πίνακας ο οποίος περιέχει τα αποτελέσματα από ένα αυθαίρετο ερώτημα Q το οποίο αξιολογείται κάθε φορά που αναφέρεται. Προκειμένου να αποφευχθεί η άσκοπη εκ νέου εκτέλεση του Q είναι καλύτερη η αποθήκευση του V στη δευτεροβάθμια αποθήκευση. Αυτό εισάγει την έννοια του materialized view (πιο κάτω αναφέρεται ως view). Τα views έχουν καθαρό χώρο σε σχέση με το χρόνο tradeoff: Ένα πλήρως materialized view V απαιτεί περισσότερο χώρο, αλλά και λιγότερο χρόνο για την αξιολόγηση Q . Το materialized view V μπορεί δυννητικά να εξοικονομήσει ενέργεια, όπως η εφαρμογή μπορεί να αποφύγει το κόστος επαναξιολόγησης στο δίκτυο Q . Παρά το γεγονός ότι ένα πλήρως materialized view V διατηρεί τα πλήρη αποτελέσματα ενός ερωτήματος Q , η κατανομημένη φύση του δικτύου αισθητήρων περιβάλλοντος, μαζί με τα ιδιαίτερα χαρακτηριστικά του, επιβάλλει ορισμένους θεμελιώδεις περιορισμούς σε αυτό το μοντέλο:

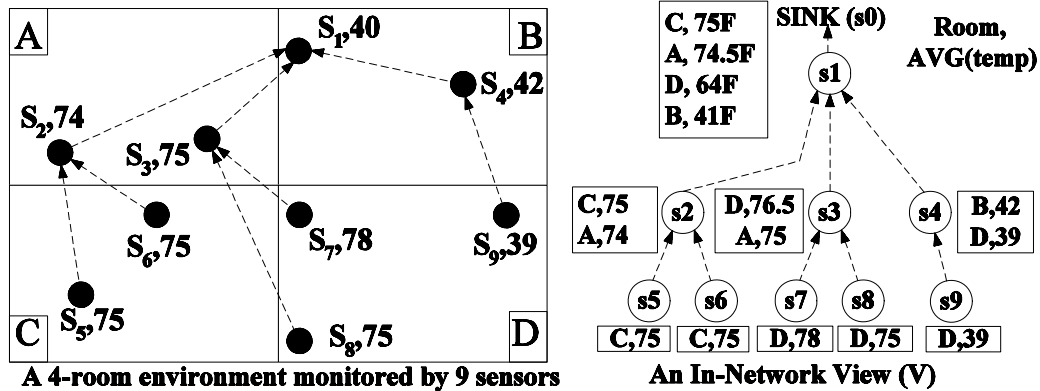
Πρώτο, η διατήρηση της συνοχής μεταξύ των V υποκειμένων και κατανεμημένων βασικών σχέσεων R (που ορίζεται από τις αναγνώσεις του αισθητήρα) είναι πολύ ακριβή σε σχέση με την ενέργεια. Έτσι, εστιάζουμε στη διατήρηση ενός υποσυνόλου $V'(\subseteq V)$ που αποκαλύπτει μόνο τις k υψηλότερες σε κατάταξη απαντήσεις για έναν ορισμένο χρήστη που ορίζει το k .

Δεύτερο, το V' ορίζεται κατ'επανάληψη με τα αποτελέσματα που είναι αποθηκευμένα στα χαμηλότερα επίπεδα των multi-hop δέντρων δρομολόγησης που διασυνδέουν το sink με τις συσκευές αίσθησης (sensing devices). Με αποτέλεσμα να μην είναι όπως τις παραδοσιακές τεχνικές συντήρησης view.

Για ευκολία, ας εξετάσουμε το σενάριο που παρουσιάζεται στο σχήμα 2.1.1 όπου θα επεξηγήσουμε την ανάπτυξη 9 αισθητήρων σε ένα κτίριο τεσσάρων δωματίων. Μας ενδιαφέρει να δοθεί απάντηση στο ερώτημα στο sink (ρίζα ανωτέρω s_1).

Επίσης, ας υποθέσουμε μια συνεχή ερώτηση στιγμιότυπου (snapshot query) που έχει ως στόχο να ανακαλυφθεί το μέσο επίπεδο ήχου (ως ποσοστό) του $k = 1$ δωματίου κάθε ένα λεπτό. Συγκεκριμένα, ένα τέτοιο ερώτημα μπορεί να εκφραστεί σε SQL όπως την ακόλουθη σύνταξη:

```
SELECT TOP 1 roomid, AVERAGE (sound)
FROM sensors
GROUP BY roomid
EPOCH duration 1 min
```



Σχήμα 2.1.1 Το αριστερό σχήμα απεικονίζει έναν sensor network scenario που αποτελείται από 9 αισθητήρες ($s_1, \dots, s_9$) και αναπτύσσεται σε τέσσερις αίθουσες (A, B, Γ, Δ). Η ετικέτα δίπλα σε κάθε αισθητήρα δηλώνει το αναγνωριστικό του κόμβου και τη θερμοκρασία (ως ποσοστό).

Η εικόνα στα δεξιά παρουσιάζει το δέντρο των ενδείξεων για την αντίστοιχη ερώτηση. Η ετικέτα δίπλα σε κάθε κόμβο δείχνει το τοπικό μέσο όρο [4]

Χρησιμοποιώντας τη λογική του TAG, της συσσωρευτικής προσέγγισης στο δίκτυο που χρησιμοποιείται στο σύστημα TinyDB, κάθε κόμβος θα μπορούσε να προωθήσει (διαβιβάσει) πλειάδες (tuples) του τύπου (roomid, sum, count) στον πατέρα του κάθε ένα λεπτό. Θα μπορούσε τότε να εφαρμοστεί εύκολα ένας νέος top-k τελεστής που να ρωτά τον κόμβο από τον οποίο θα αποκομιστούν οι σωστές απαντήσεις με ένα συγκεντρωτικό τρόπο με κλάδεμα (pruning) του χώρου απαντήσεων (αν και κανένας από τα υπάρχοντα συστήματα μέχρι στιγμής δε υποστηρίζει αυτή τη δυνατότητα). Ακόμη και αν μια τέτοια τεχνική ήταν άμεσα διαθέσιμη, δεν είναι οικονομικά αποδοτική, διότι όλες οι πλειάδες (tuples) πρέπει να μεταφερθούν στον κόμβο που έχει κάνει την ερώτηση (querying node).

Μία εναλλακτική προσέγγιση είναι η έννοια του in-network view (V) (Σχήμα 2.1.1 δεξιά). Το V υλοποιεί το αποτέλεσμα της Q και αξιοποιεί τα αποτελέσματα αυτά για να επιταχύνει την επόμενη εκτέλεση της Q. Η απόδοση του V βασίζεται σε μεγάλο βαθμό στην προϋπόθεση της χρονικής συνοχής μεταξύ των διαδοχικά ανακτημένων αναγνώσεων αισθητήρα ως τοπικές αλλαγές που θα επηρεάσουν το ενδιάμεσο view μέχρι τον sink.

Για να βελτιωθεί το σφάλμα εκτέλεσης των in-network views, αποκόπτονται τα τοπικά views που αποθηκεύονται σε κάθε κόμβο που δεν υπάρχει πιθανότητα να βρίσκονται στο αποτέλεσμα και επικεντρώνεται στις k υψηλότερες σε κατάταξη απαντήσεις και όχι σε όλες. Αυτό αποδεικνύεται εξαιρετικά χρήσιμο διότι οι αισθητήρες μπορούν να απορρίψουν ενημερώσεις views που δεν αναφέρονται στις k υψηλότερες σε κατάταξη απαντήσεις. Από την άλλη πλευρά, αυτό επιβάλλει επίσης ένα εξαιρετικά δύσκολο πρόβλημα: *"μια αφελής τοπική άπληστη στρατηγική αποκοπής αποτελεσμάτων (pruning) μπορεί να απορρίψει εύκολα πλειάδες που είναι τελικά μεταξύ των k υψηλότερων σε κατάταξη απαντήσεων"*.

Για να κατανοήσουμε αυτό το πρόβλημα, θεωρούμε ξανά το ερώτημα που δόθηκε πιο πάνω, αλλά υποθέτουμε ότι μας ενδιαφέρει μόνο το top-1 αποτέλεσμα. Ένα τέτοιο ερώτημα θα πρέπει να επιστρέψει την αίθουσα (C, 75F). Υποθέτοντας ότι κάθε κόμβος εξαλείφει αφελώς οτιδήποτε κάτω από την τοπική κορυφή του top-1 το αποτέλεσμα θα μας οδηγήσει στη λανθασμένη απάντηση (D, 76.5F). Ειδικότερα, τα φύλλα (s_5, s_6, s_7, s_8, s_9) θα στείλουν μόνο τις δικές τους πλειάδες στους αντίστοιχους προγόνους τους. Οι γονείς (s_2, s_3, s_4) θα έχουν συνολικά τα αποτελέσματα των παιδιών τους μαζί με το δικό τους αποτέλεσμα και θα διαβιβάσουν αυτό το αποτέλεσμα στο δικό τους πατέρα (δηλαδή, s_1). Συγκεκριμένα, θα στείλει ο s_2 (C, 75F), ο s_3 πλειάδα (D, 76.5F) και ο s_4 πλειάδα (B, 42F). Τώρα είναι εύκολο να δούμε ότι αν ο s_1 έχει τα συγκεντρωτικά αποτελέσματα των παιδιών του (s_2, s_3, s_4) μαζί με το δικό του αποτέλεσμα (B, 40F), τότε αυτό θα αποφέρει $V_0^{wrong} = ((D, 76.5F), (C, 75F), (B, 41F))$, όπου η αίθουσα D είναι η top-1 απάντηση και όχι η αίθουσα C.

Ο αλγόριθμος MINT για να εξαλείψει το πρόβλημα αυτό χρησιμοποιεί ένα έξυπνο πάνω bounding αλγόριθμο και μια τοπική παράμετρο k για να κατασκευάσει ένα υποσύνολο του V , που συμβολίζεται με το k -covered bound-set V' , που θα υλοποιηθεί. Κάθε πλειάδα εκτός του V' μπορεί με ασφάλεια να εξαλειφθεί κατά την εκτέλεση ενός ερωτήματος, επειδή αυτή η πλειάδα δεν μπορεί να είναι μεταξύ των k υψηλότερων σε κατάταξη αποτελεσμάτων.

Η βασική ιδέα του pruning αλγόριθμου MINT είναι να αξιοποιηθεί ένα σύνολο από $|\gamma|$ παραμέτρους ($\gamma = \{\gamma_1, \gamma_2, \dots\}$) προκειμένου να δεσμευθεί πάνω από το σκορ των

πλειάδων που δεν είναι γνωστές σε ένα συγκεκριμένο επίπεδο του δικτύου αισθητήρων. Τα χαρακτηριστικά γ έχουν αναπτυχθεί ειδικά για αυτές τις εφαρμογές, και μπορούν είτε να είναι γνωστά εκ των προτέρων, ή μπορεί να οριστούν πριν από την εκτέλεσης ενός ερωτήματος.

Για παράδειγμα: $\gamma_1 = \text{"η μέγιστη δυνατή τιμή της θερμοκρασίας"}$ και $\gamma_2 = \text{"ο αριθμός των αισθητήρων σε κάθε δωμάτιο"}$. Επίσης, ο αισθητήρας θερμοκρασίας για το Mica Weather Board για παράδειγμα ενδέχεται να καταγράφει μόνο τιμές μεταξύ των -40F ως 250F και η βαρομετρική πίεση μπορεί να μετρήσει μόνο την πίεση στην περιοχή 300mb μέχρι 1100mb. Έτσι, το γ_1 μπορεί να είναι γνωστό εκ των προτέρων. Επιπλέον, η απόκτηση και η διάδοση των γ_2 μπορεί να πραγματοποιηθεί κατά τη διάρκεια της αρχικοποίησης.

2.2. Μοντέλο Συστήματος και Ορολογία

Σύμβολο	Ορισμός
Q	Η Επερώτηση (query)
k	Αριθμός αιτούμενων αποτελεσμάτων
s_i	Αριθμός αισθητήρα i (s_0 είναι ο sink)
n	Αριθμός από αισθητήρες $\{s_1, s_2, \dots, s_n\}$
M	Αριθμός χαρακτηριστικών κάθε αισθητήρα $\{a_1, a_2, \dots, a_m\}$
V_i	Τοπικό View (τα αποτελέσματα της επερώτησης Q) από τον αισθητήρα s_i ($i \leq n$)
V'_i	Κλαδεμένη Όψη (Pruned View) του s_i

Πίνακας 2.2.1: Ορισμοί συστήματος

Θεωρούμε ότι το S απαρτίζει ένα σύνολο από n αισθητήρες $S = (s_1, s_2, \dots, s_n)$. Υποθέτουμε ότι s_i ($i \leq n$) είναι σε θέση να αποκτήσει m φυσικές ιδιότητες $A = (a_1, a_2, \dots, a_m)$ από το περιβάλλον τους σε κάθε ξεχωριστό t χρόνο. Αυτό δημιουργεί πλειάδες του τύπου $(t, a_1, a_2, \dots, a_m)$ σε κάθε αισθητήρα. Σε μια δεδομένη στιγμή, οι προαναφερθείσες αποδόσεις του σεναρίου ενός πίνακα $n \times m$ αναγνώσεων $R = (s_{ij})_{n \times m}$. Αυτός ο πίνακας είναι κατακερματισμένος οριζοντίως μεταξύ των n αισθητήρων, δηλαδή η σειρά i περιέχει τις αναγνώσεις του αισθητήρα s_i και $R = \bigcup_{i \in n} R_i$.

Προκειμένου ο αλγόριθμος να διαδώσει το ερώτημα προς τους n αισθητήρες, θα χρησιμοποιεί ένα τυπικό δέντρο με βάση το ερώτημα μηχανισμού διάδοσης, όπου ο sink κόμβος στέλνει το ερώτημα Q σε έναν αισθητήρα s_1 αναδρομικά, και προωθεί το συγκεκριμένο ερώτημα σε όλους τους γείτονές του, έως ότου όλοι οι n αισθητήρες να έχουν λάβει το συγκεκριμένο ερώτημα. Χωρίς απώλεια της γενικότητας, έχει θεσπιστεί ο μηχανισμό πρόσδεσης των παιδιών (child anchor) που προτείνεται στο άρθρο [6], όπου ο s_i αισθητήρας επιβεβαιώνει σε ακριβώς έναν από τους γείτονές του, ότι ο s_j θέλει να είναι παιδί του. Αυτό παρέχεται στον s_j με μια λίστα με τα παιδιά, ώστε να είναι γνωστό στον s_j όταν όλες οι απαντήσεις από τα παιδιά του έχουν φτάσει. Επίσης, υποστηρίζεται μια TAG πολιτική τοπολογίας συντήρησης που προσαρμόζεται σε μια μετατόπιση του δικτύου για την παρακολούθηση της ποιότητας σε κάθε κόμβο και την αλλαγή των γονιών, εάν η ποιότητα πέφτει κάτω από ένα συγκεκριμένο όριο.

2.3. Συσχετιζόμενη Εργασία

Η Top-k επεξεργασία επερωτήσεων έχει μελετηθεί σε μια ποικιλία καταστάσεων, συμπεριλαμβανομένων middleware συστήματα [11,21], web διαθέσιμων βάσεων δεδομένων [22,23], επεξεργαστές stream [24], peer-to-peer [25] και άλλα καταναμεμένα συστήματα [26,27,28]. Για παράδειγμα, το ερώτημα *"Βρείτε το $k = 5$ δωμάτια με τη μεγαλύτερη μέση θερμοκρασία"* επιστρέφει ένα υποσύνολο του συνόλου που περιέχει όλες τις απαντήσεις, ώστε να ελαχιστοποιηθεί το κόστος μετρικών που σχετίζεται με την ανάκτηση της πλήρους δέσμης απαντήσεων. Αυτό το κόστος συνήθως μετριέται από την άποψη της πρόσβασης στο δίσκο ή μεταδόσεις δικτύου, ανάλογα με το πού βρίσκονται φυσικά αποθηκευμένα τα δεδομένα.

Έχει αποδειχθεί σε πολλές μελέτες, ότι η top-k επεξεργασία επερωτήματος έχει νόημα μόνο εάν το υποκείμενο k αναφέρεται σε ένα μικρό υποσύνολο της πλήρους δέσμης απαντήσεων (συνήθως μέχρι - 5%). Για μεγαλύτερες τιμές του k , η βελτίωση του ερωτήματος μπορεί να επιλεγεί για να ανακτηθεί το πλήρες σύνολο απάντησης.

2.4. Πλαίσιο MINT

Σε αυτό το τμήμα περιγράφονται οι βασικοί αλγόριθμοι του MINT View Framework στις ακόλουθες τρεις εννοιολογικές φάσεις:

A. Η φάση Δημιουργίας (Creation Phase): εκτελείται κατά την πρώτη απόκτηση των αναγνώσεων από τους κατανεμημένους αισθητήρες. Αυτή η φάση έχει ως αποτέλεσμα n κατανεμημένα views V_i ($i \leq n$).

B. Η φάση Κλαδέματος (Pruning Phase): κάθε s_i αισθητήρας κάνει τοπική αποκοπή (pruning) V_i και δημιουργεί $V'_i(V_i)$. Ο νέος V'_i πίνακας περιέχει μόνο τις πλειάδες που θα μπορούσαν να βρίσκονται μεταξύ των τελικών αποτελεσμάτων top- k .

Γ. Η Φάση Ενημέρωσης (Update Phase): εκτελείται μία φορά ανά χρονική περίοδο, κατά τη διάρκεια που s_i ενημερώσεις φθάνουν στο κόμβο του πατέρα με V'_i .

Οι παραπάνω εννοιολογικές φάσεις εκτελούνται κατανεμημένα χρησιμοποιώντας το δέντρο με βάση το ερώτημα δρομολόγησης, πρωτόκολλο που θεσπίστηκε από το στρώμα του λειτουργικού συστήματος, μετά που έχει διαδοθεί το ερώτημα στους n αισθητήρες.

2.4.1. Φάση Δημιουργίας (Creation Phase)

Στην πρώτη φάση του αλγορίθμου γίνεται αναδρομική εκτέλεση του Αλγορίθμου 1 (εικόνα 2.4.1) σε όλους τους αισθητήρες σε ένα συγκεκριμένο δίκτυο. Ένας αισθητήρας παράγει μια $(m + 1)$ -πλειάδα του τύπου $v = \{t, a_1, a_2, \dots, a_m\}$ σε κάθε timestamp t . Ένας αισθητήρας ξεκινά από την εκτέλεση της επιλογής σQ που διατηρεί τις πλειάδες που πληρούν το κριτήριο επιλογής (π.χ. θερμοκρασία > 60). Ένας αισθητήρας μπορεί να αποκτήσει ταυτόχρονα πολλές αναγνώσεις, όλες εκ των οποίων δεν μπορούν να έχουν ενδιαφέρον για ένα συγκεκριμένο ερώτημα. Για παράδειγμα, το MICA Weather Board το οποίο χρησιμοποιείται στη μελέτη Great Duck Island, συμπληρώνει τα MICA motes με 14 φυσικές παραμέτρους. Έτσι, έχουμε μόνο τα χαρακτηριστικά του project που σχετίζονται με το Q πριν από την αποθήκευση του αποτελέσματος στην μνήμη buffer V_i (γραμμή 3). Τα ο επόμενο βήμα του αλγορίθμου συγχωνεύει τις πλειάδες που φτάνουν από τα παιδιά του s_i σε V_i (γραμμή 4-13).

Εάν οι διάφορες τιμές σε κάθε κόμβο του δέντρου που απεικονίζεται δεν αλλάζουν σε συνεχή timestamps, τότε τα V μπορούν να προσφέρουν αποτελεσματική απάντηση στη μετέπειτα εκ νέου εκτέλεση του Q . Αντιθέτως, κάθε φορά που έχουμε μια απόκλιση, ή μια αλλαγή, σε μια παράμετρο στο s_i , η αλλαγή αυτή πρέπει να διαχέεται σε όλη τη διαδρομή μέχρι το sink. Μια αλλαγή σε όλους τους αισθητήρες έχει τη χειρότερη περίπτωση πολυπλοκότητας της τάξης $O(n)$ για κάθε χρονική σήμανση της εποχής (epoch), έτσι προσπαθούμε να βελτιστοποιήσουμε τη διαδικασία αυτή με την πρόταση της φάσης pruning.

Algorithm 1 : Construct MINT/INT View

Input: A distributed sensor s_i ($\forall s_i \in S$) that generates m attributes $\{a_1, a_2, \dots, a_m\}$, a query Q , an empty buffer $V_i = \{\}$

Output: A set of n distributed views $V = \{V_1, V_2, \dots, V_n\}$.

```

1: procedure CONSTRUCT_MINT_VIEW( $s_i, Q$ )
2:   // Execute  $Q$  and store the answer in  $V_i$  (takes  $O(1)$  time).
3:    $insert(\pi_Q(\sigma_Q(current\_reading()))), V_i$ ;
4:   for  $j = 1$  to  $|children(s_i)|$  do
5:      $c = child(s_i, j)$ ; //  $c$  is the  $j^{th}$  child of node  $s_i$ 
6:     //  $w$  is a list of tuples returned to query  $Q$ .
7:      $w = Construct\_Mint\_View(c, Q)$ ;
8:     for  $l = 1$  to  $|w|$  do
9:       //  $w_l$  is the  $l^{th}$  entry of table  $w$ .
10:      // Inserts tuple  $w_l$  into local table  $V_i$  in  $O(1)$  time.
11:       $insert(w_l, V_i)$ ;
12:   end for
13: end for
14:    $send(V_i, parent(s_i))$ ;
15: end procedure

```

Σχήμα 2.4.1: Αλγόριθμος δημιουργίας των views[4]

2.4.2. Φάση Κλαδέματος (Pruning Phase)

Ο Αλγόριθμος 1 κατασκευάζει μια ιεραρχία από views, όπου οι προγονικοί κόμβοι στην ιεραρχία δρομολόγησης διατηρούν ένα υπερσύνολο view των απογόνους τους. Πριν να εξηγήσουμε τις λεπτομέρειες της φάσης pruning που ελαχιστοποιεί τα μηνύματα μεταξύ των αισθητήρων θεωρείστε το εξής ερώτημα:

```
Q2 = "SELECT TOP k room, avg (temp) FROM  
SENSORS GROUP BY room EPOCH DURATION 1 min "
```

Το ερώτημα αυτό επιστρέφει τα k δωμάτια με την υψηλότερη μέση θερμοκρασία. Αν το s_i μπορούσε να καθορίσει σε τοπικό επίπεδο τις k -υψηλότερες απαντήσεις σε Q_2 (στο s_0), τότε ο s_i θα μπορούσε να χρησιμοποιήσει αυτές τις πληροφορίες για να κλαδέψει (prune) το τοπικό του view V_i . Ωστόσο, αυτό είναι ένα πρόβλημα που ορίζεται κατ' επανάληψη το οποίο μπορεί να λυθεί μόνο όταν όλες οι πλειάδες φιλτραριστούν μέχρι το sink s_0 . Για να αποφευχθεί αυτό, θα χρησιμοποιείται ένα σύνολο περιγραφικών γ τα οποία χρησιμοποιούνται για να δεσμεύονται τα παραπάνω χαρακτηριστικά σε V_0 και στη συνέχεια επιτρέπει ένα ισχυρό pruning framework.

Ας εξετάσουμε το παράδειγμα του πιο κάτω σχήματος 2.4.2 (αριστερά), όπου θα απεικονίζει το V_i για ένα συγκεκριμένο αισθητήρα. Πριν από την εκτέλεση του Q_2 δηλώνεται ότι $\gamma_1 = \text{"μέγιστη δυνατή τιμή της θερμοκρασίας"} = 120$ και $\gamma_2 = \text{"ο αριθμός των αισθητήρων σε κάθε δωμάτιο"} = 5$. Η εικόνα δείχνει το sum και count για πολλούς αριθμούς δωματίων. Με την παρατήρηση της στήλης 3 (δηλαδή count), καθίσταται σαφές ότι το sum για τα δωμάτια (2, 5, 11, 12, 15) είναι μια μερική αξία του αθροίσματος που επέστρεψε στο sink (αφού $\gamma_2 = 5$). Αντιθέτως, η πλειάδα του δωματίου 6 είναι ήδη στην τελική της μορφή (δηλαδή, 500). Σε αυτό το παράδειγμα το άθροισμα της κάθε γραμμής οριοθετείται ανωτέρω χρησιμοποιώντας τον τύπο $\text{sum}' = \text{sum} + (\gamma_2 - \text{count}) * \gamma_1$ και οριοθετείται στη συνέχεια με βάση το πραγματικό χαρακτηριστικό sum. Αυτό δημιουργεί έξι κατώτατα όρια (lb) και ανώτατα όρια (ub) ζεύγη που δείχνουν ακριβώς το εύρος των πιθανών τιμών για το χαρακτηριστικό sum στο sink.

Έχοντας αυτή τη γνώση σε τοπικό επίπεδο, μπορεί να μας βοηθήσει τώρα να κάνουμε prune τα ζεύγη (lb,ub) που δεν θα είναι στο τελικό top-k αποτέλεσμα. Η διαίσθηση πίσω από τον αλγόριθμο είναι να εντοπιστούν οι k^{th} υψηλότερες κατώτατο όριο (δηλαδή u_k^{lb}) και στη συνέχεια να εξαλείψει όλες τις πλειάδες που έχουν άνω όριο (δηλαδή, u^{ub}) κάτω από u_k^{lb} . Το Σχήμα 2.4.2, απεικονίζει οπτικά αυτή την ιδέα. Αποδείχτηκε ότι με την εφαρμογή σε τοπικό επίπεδο μια τέτοια πράξη απόδοσης έχει στο τέλος τις ορθές top-k πλειάδες στο sink. Για να επιτευχθεί αυτό, καθορίζεται η έννοια του k-Covered bound-Set ως εξής:

Ορισμός: k-Covered bound-Set (V_i'), είναι το υποσύνολο του V_i που ικανοποιεί τον ακόλουθο όρο: Αν υπάρχει κάποιο $u \in V_i'$, τότε $u^{ub} < u_k^{lb}$ όπου u_k^{lb} είναι η k^{th} υψηλότερη κάτω όριο.

Ο αλγόριθμος 2 απεικονίζει το pruning των V_i κάποιου αυθαίρετου κόμβου si και η κατασκευή του υποψήφιου V' . Αυτός ο αλγόριθμος ισχύει τόσο για το MINT view, όσο και για το INT View frameworks. Το πρώτο βήμα του αλγορίθμου (γραμμές 2-6) προσδιορίζει το κατώτατο όριο (threshold) pruning u_k^{lb} . Το όριο αυτό επιτρέπει στον αλγόριθμο να αποκόψει πλειάδες που δεν θα είναι στο αποτέλεσμα. Παρόλο που το V_i βρίσκεται φυσικά στην κύρια μνήμη, θέλουμε την ελαχιστοποίηση του χρόνου λειτουργίας των αλγορίθμων, προκειμένου να αντιμετωπιστούν σπάνιοι προϋπολογισμοί ενέργειας. Συγκεκριμένα, χρησιμοποιείται αλγόριθμος παρόμοιος με το γνωστό αλγόριθμο επιλογής, για k αποθηκευμένα (kBuff) στοιχεία προκειμένου να εντοπιστεί το u_k^{lb} σε γραμμικό χρόνο (δηλαδή $O(k)$ ανά πλειάδα). Η διαδικασία αυτή λαμβάνει χώρα στο εσωτερικό της kHighest λειτουργίας που καταχωρεί u_j^{lb} σε kBuff, αν ο προηγούμενος είναι μεγαλύτερος από το ελάχιστο στοιχείο σε kBuff. Το επόμενο βήμα του αλγορίθμου είναι να εντοπίσει τις πλειάδες που έχουν άνω όριο u^{ub} κάτω από το όριο u_k^{lb} . Με την οπτική εξέταση στο σχήμα, είναι εύκολο να δούμε ότι ένας αποτελεσματικός τρόπος για να γίνει αυτό είναι να δημιουργηθεί μια λίστα των άνω ορίων και στη συνέχεια, να εκτελεστεί μια γραμμική σάρωση κατά φθίνουσα σειρά μέχρι να βρεθεί η πλειάδα $u_j^{lb} (< u_k^{lb})$. Κάθε άνω όριο κάτω ή ίσο με u_j^{lb} μπορεί να εξαλειφθεί με ασφάλεια. Ο διατεταγμένος κατάλογος μπορεί να κατασκευαστεί παράλληλα με την θέση του pruning ορίου u_k^{lb} . Συγκεκριμένα, για τη σάρωση u_k^{lb} , εισάγεται κάθε άνω όριο u_j^{lb} σε ένα νέο sortedUBs πίνακα (γραμμή 5). Αυτό χρειάζεται

Στις γραμμές 8-12, έχουμε εκτελέσει τελικά μια γραμμική σάρωση του πίνακα sortedUBs κατά φθίνουσα τάξη και θα σταματήσει όταν θα βρει μια πλειάδα u_j^{lb} που είναι κάτω από το u_k^{lb} .

Algorithm 2 : Prune MINT/INT View

Output: A locally pruned view V'_i , such that V'_0 can be utilized to answer a top-k query Q .

```

1: procedure PRUNE_MINT_VIEW( $V_i$ )
2:   for  $j = 1$  to  $|V_i|$  do // Identify the pruning threshold  $v_k^{lb}$ .
3:      $v_j = V_i[j]$  //  $v_j = (v_j^{lb}, v_j^{ub})$  pair.
4:      $kHighest(v_j^{lb}, kBuff)$ 
5:      $bucketinsert(v_j^{ub}, sortedUBs)$ 
6:   end for
7:    $v_k^{lb} = min(kBuff)$ ;
8:   for  $j = 1$  to  $|sortedUBs|$  do
9:      $v_j^{ub} = sortedUBs[j]$ 
10:    If  $(v_j^{ub} < v_k^{lb})$  then break; end if
11:     $add\_to\_candidates(v_j, V_i')$ ;
12:   end for
13: end procedure

```

20

2.4.3. Φάση Ενημέρωσης (Update Phase)

Στο προηγούμενο βήμα, με ένα V_i pruning μετατράπηκε σε ένα υποσύνολο V_i' . Τώρα θα περιγράψουμε τον τρόπο σταδιακά και με αναδρομική ενημέρωση του V' . Ας δηλώσουμε το T' το V_i' που λήφθηκε κατά την τελευταία εκτέλεση του Q . Η παρακάτω περιγραφή ισχύει μόνο για το MINT View framework, για τον οποίο το T' είναι διαθέσιμο. Η φάση ενημέρωσης (Update) του framework INT View είναι απλώς μια εκ νέου εκτέλεση του αλγόριθμου 1, ο οποίος κατασκευάζει το V_i' από την αρχή.

Δεδομένου ότι ο στόχος μας είναι να προσδιορίσει τα σωστά αποτελέσματα στο sink, θα χρησιμοποιεί ένα μηχανισμό άμεσης συντήρησης view: Από τη στιγμή που μια νέα πλειάδα παράγεται από τον s_i , αυτή η ενημερωμένη έκδοση αντανακλάται στο V_i' . Προκειμένου να ελαχιστοποιηθεί η επικοινωνία, στο s_i , εκ νέου μεταδίδει το V_i' στον πατέρα του, εάν το V_i' έχει αλλάξει. Επιπλέον, προκειμένου να ελαχιστοποιηθεί η ενέργεια ακόμη περισσότερο, θα επιδιώξει να ελαχιστοποιήσει την κατανάλωση, καθώς και τη μεταποίηση. Ως εκ τούτου, ο στόχος είναι να κατασκευάσει V_i' αποφεύγοντας την εκ νέου εκτέλεση του αλγόριθμου 2.

Ο αλγόριθμος 3 (Σχήμα 2.4.3.1) παρουσιάζει τον αλγόριθμο MINT Update. Ειδικότερα, η γραμμή 3 δείχνει ότι κάθε ενημέρωση πλειάδας x με ένα άνω όριο που φέρει (όπως x^{ub}) λιγότερο από το u_k^{lb} που μπορεί να αγνοηθεί. Σε αντίθετη περίπτωση, προσθέτονται οι πλειάδες x για το σύνολο των υποψηφίων V_i' (γραμμή 4). Τώρα τα υπόλοιπα ερωτήματα είναι κατά πόσον το u_k^{lb} έχει αλλάξει με την προσθήκη αυτή του x . Αν $x^{lb} \leq u_k^{lb}$ είναι αλήθεια τότε ότι u_k^{lb} δεν έχει αλλάξει. Κατά συνέπεια, το s_i μεταδίδει μόνο την ενημερωμένη x προς τον προγονικό του κόμβου, παρά μία πλήρη ενημερωμένη άποψη. Κατά την υλοποίηση γίνονται buffer αυτές οι ενημερωμένες εκδόσεις, έως ότου όλα τα παιδιά να στείλουν ενημερώσεις για τους γονείς τους. Αντίθετα, αν $u_k^{lb} < x^{lb}$ τότε x^{lb} μπορεί να έχει αλλάξει. Ως αποτέλεσμα, το s_i να ανακατασκευάσει το V_i' χρησιμοποιώντας τον αλγόριθμο 2 και διαβιβάζοντας τα πλήρη V_i' στον πατέρα του. Αυτή η διαδικασία της εκ νέου κατασκευής είναι αναγκαία για τη διασφάλιση της ορθότητας του framework. Η ανασυγκρότηση συμβαίνει μόνο για $|V_i'|$ στοιχεία και όχι για όλα τα στοιχεία (δηλαδή, $|V_i|$), εκτελέστηκε ο αλγόριθμος 2 για πρώτη φορά.

Algorithm 3 : Update MINT View

Input: A buffer T' that contains the V'_i of the previous time instance, the v_k^{lb} of T' , a tuple update x from some child.

Output: A locally pruned view V'_i , such that V'_0 can be utilized to answer a top-k query Q .

```
1: procedure UPDATE_MINT_VIEW( $T', v_k^{lb}, x$ )
2:    $V'_i = T'$ ;
3:   if ( $v_k^{lb} \leq x^{ub}$ ) then
4:     add_to_candidates( $x, V'_i$ );
5:     if ( $x^{lb} \leq v_k^{lb}$ ) then
6:       send( $x, parent(s_i)$ ); // Single tuple x update
7:     else //  $x^{lb} > v_k^{lb}$ 
8:       Prune_MINT_View( $V'_i$ ); // Using Algorithm 2
9:       send( $V'_i, parent(s_i)$ ); // Complete  $V'_i$  update
10:    end if
11:  end if
12:   $T' = V'_i$ ;
13: end procedure
```

Σχήμα 2.4.3.1 : Αλγόριθμος Ενημέρωσης[4]

Κεφάλαιο 3

Περιβάλλον Αξιολόγησης του MINT

- 3.1. Εισαγωγή
 - 3.2. Δομή Πακέτων και Επικοινωνία στο TinyOS
 - 3.3. Επικοινωνία στο TinyOS
 - 3.4. TinyDB
 - 3.5. Περιβάλλον Προσομοίωσης TOSSIM και Γραφικό Περιβάλλον TinyViz
 - 3.6. PowerTOSSIM
-

3.1. Εισαγωγή

Για να αξιολογήσουμε τις αποδόσεις του αλγόριθμου Mint χρειάστηκε να γνωρίζουμε τη δομή των πακέτων που μεταφέρονται καθώς και το πώς γίνεται η επικοινωνία στο λειτουργικό σύστημα TinyOS. Για την πειραματική αξιολόγηση χρησιμοποιήθηκε ο TOSSIM [15] που είναι ένας προσομοιωτής διακριτών συμβάντων (discrete event) ειδικά σχεδιασμένος για το TinyOS μαζί με το TinyViz που είναι το εργαλείο απεικόνισης του TOSSIM. Επιπλέον, για τον υπολογισμό της κατανάλωσης ενέργειας ανά κόμβο έγινε χρήση του PowerTossim που είναι μια επέκταση του προσομοιωτή. Επίσης, σε συνδυασμό με τον προσομοιωτή χρησιμοποιήθηκε το TinyDB, το σύστημα επεξεργασίας επερωτήσεων για την εξαγωγή πληροφοριών από ένα TinyOS δίκτυο αισθητήρων.

3.2. Δομή Πακέτων και Επικοινωνία στο TinyOs

Η επικοινωνία αποτελεί μια από τις σημαντικότερες υπηρεσίες του συστήματος TinyOS και γενικότερα των δικτύων αισθητήρων. Η βασική μονάδα επικοινωνίας είναι τα πακέτα, όπως και στα υπόλοιπα δίκτυα υπολογιστών. Όπως είναι ήδη γνωστό, ένα

πακέτο είναι μια σειρά από bits με συγκεκριμένη αρχή και τέλος. Το TinyOS ορίζει μια συγκεκριμένη δομή στα πακέτα που ανταλλάσσονται μεταξύ των αισθητήρων.

Η επικοινωνία μεταξύ των κόμβων παρέχεται μέσω του βασικού στοιχείου του tinyos, το GenericComm. Το GenericComm παρέχει δύο διεπαφές το SendMsg και ReceiveMsg για αποστολή και λήψη δεδομένων αντίστοιχα.

Τα δεδομένα που στέλνονται είναι σε μορφή “raw data” και η δομή τους όπως χρησιμοποιείται από τη συνάρτηση SendMsg.send(), είναι struct TOS_Msg, όπως ορίζεται στο αρχείο AM.h. Το εν λόγω αρχείο περιέχει διάφορα πεδία για τη διεύθυνση αποστολής όπως τον τύπο μηνύματος (AM handler ID), το μήκος, το ωφέλιμο φόρτο δεδομένων (payload data) και άλλα.

Η διασύνδεση (wiring) στο στοιχείο GenericComm γίνεται με τη χρήση παραμετροποιημένης διεπαφής (parameterized interface), λόγω του ότι πολλά στοιχεία μπορεί να μοιράζονται το ίδιο κανάλι ταυτόχρονα αλλά το καθένα ενδιαφέρεται μόνο για τα δικά του δεδομένα. Για αυτό το κάθε πακέτο TinyOS περιέχει ένα πεδίο ‘group ID’.

Το μήκος του μηνύματος καταγράφεται επίσης μέσα σε ξεχωριστό πεδίο του πακέτου ώστε να διευκολύνεται η επικοινωνία. Επιπρόσθετα, το μέγιστο μήκος του ωφέλιμου πεδίου ‘payload’ είναι ορισμένο σε 29 bytes αλλά υπάρχει η δυνατότητα να αυξηθεί και περισσότερο. Για διευκόλυνση ορίζεται μια δομή για κάθε τύπο μηνύματος που θα σταλεί ανάλογα με την εφαρμογή, και τοποθετείται στο πεδίο payload data του πακέτου TOS_Msg. Οπότε, με τον τρόπο αυτό επιτυγχάνεται η καλύτερη διαχείριση των μηνυμάτων που λαμβάνονται, ιδιαίτερα όταν υπάρχει επικοινωνία μεταξύ υπολογιστή και ενός αισθητήρα. Για την εφαρμογή που αξιολογούμε σε αυτή τη μελέτη η δομή του πακέτου είναι τύπου struct PTopKRoomData.

Ο κάθε αισθητήρας που θέλει να στείλει ένα μήνυμα καταχωρεί ένα χώρο ενδιάμεσης μνήμης (buffer) για την αποθήκευση των δεδομένων του μηνύματος πριν το στείλει στο send(). Στη συνέχεια, εάν το στρώμα μηνυμάτων (message layer) έχει χώρο για να αναλάβει τον έλεγχο της ενδιάμεσης μνήμης, τότε κάνει αποδεκτή την αίτηση

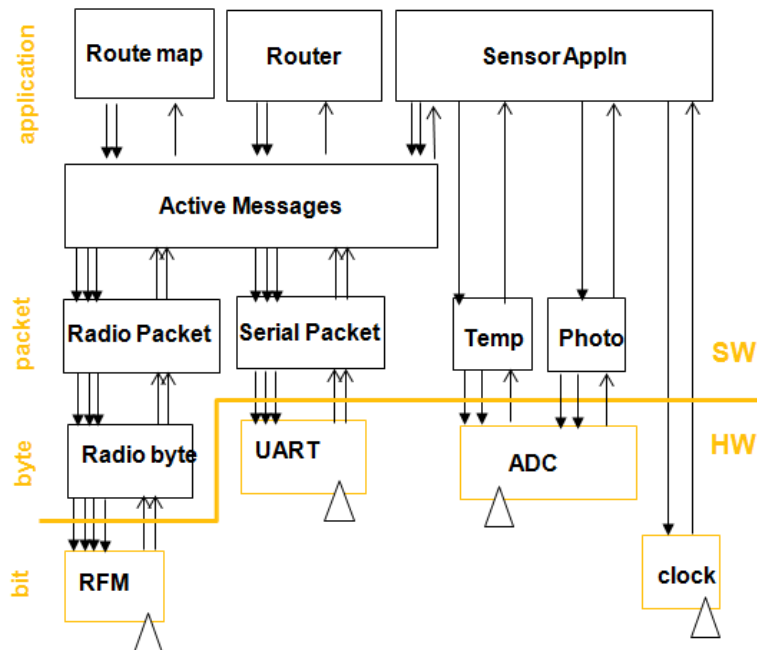
αποστολής, παίρνει τον έλεγχο, ο αισθητήρας που έκανε την αίτηση οφείλει να μην τροποποιήσει το χώρο ενδιάμεσης μνήμης μέχρι την ολοκλήρωση της αποστολής (SendDone). Για το λόγο αυτό, είναι συνηθισμένη τακτική να χρησιμοποιείται κώδικας που κατά κάποιο τρόπο θα κλειδώνει το buffer από κάθε απόπειρα πρόσβασης μέχρι την ολοκλήρωση της αποστολής.

Η διαχείριση της μνήμης για τα εισερχόμενα μηνύματα είναι δυναμική. Όταν το μήνυμα καταφθάνει, τότε συμπληρώνει ένα χώρο ενδιάμεσης μνήμης (buffer). Στη συνέχεια το στρώμα *AM (Active Message layer)* το αποκωδικοποιεί και το στέλνει προς διεκπεραίωση. Μόλις ολοκληρωθεί η επεξεργασία, το στοιχείο της εφαρμογής πρέπει να αποστείλει πίσω στο κατώτερο στρώμα ένα δείκτη που θα δείχνει στην ενδιάμεση μνήμη. Σε αυτή την περίπτωση όπως αυτή όπου η εφαρμογή χρειάζεται να αποθηκεύσει τα περιεχόμενα του μηνύματος προς περαιτέρω χρήση, μπορεί να αντιγράψει το μήνυμα σε ένα νέο ελεύθερο χώρο μνήμης (buffer) για χρήση από το στρώμα δικτύου (buffer swap).

Επιπρόσθετα, υπάρχουν και άλλα στοιχεία που εξασφαλίζουν την αξιόπιστη ασύρματη μετάδοση των πακέτων και τα οποία προσαρτώνται σε πεδία στο πακέτο πριν την αποστολή του. Ένα από αυτά είναι και η χρήση του προοιμίου 'preamble'. Το προοίμιο 'preamble' αποτελεί μια συγκεκριμένη διάταξη από bits που αποστέλλονται πριν τη μετάδοση ενός πακέτου. Το μήνυμα φθάνει στο στρώμα ραδιοσύζευξης και αρχίζει η αποστολή από τον πομπό σύμφωνα με τα bits του μηνύματος και τον τρόπο *Πρόσβασης στο Μέσο (Media Access Control)* το οποίο έχει καθοριστεί. Ένας δέκτης ακούει ένα κανάλι σταθερά και προσπαθεί να ξεχωρίσει αν το επίπεδο της ενέργειας είναι αρκετά υψηλό ή τα χαρακτηριστικά του σήματος είναι αρκετά ώστε να αποτελούν ένα μεταδιδόμενο bit. Εάν διαπιστώσει ότι είναι αρκετά υψηλό, τότε αρχίζει τη δειγματοληψία με τη συχνότητα που έχει οριστεί από το πρότυπο ασύρματης μετάδοσης που είναι σίγουρος ότι θα λάβει ένα πακέτο μηνύματος. Η διάρκεια που ο δέκτης κάνει δειγματοληψία από το κανάλι, πριν σταματήσει, καθορίζεται από το πεδίο που ορίζει το μήκος του πακέτου. Βέβαια, επειδή ο θόρυβος είναι πιθανόν να προκαλέσει σφάλματα κατά την αποστολή του πακέτου, τα οποία πρέπει να διορθωθούν ή να εντοπιστούν πριν το μήνυμα προωθηθεί από το δέκτη σε ανώτερα στρώματα, γίνεται προσάρτηση στο σώμα του πακέτου ενός πεδίου CRC κώδικα. Με

την κατάλληλη χρήση αυτού του πεδίου από το δέκτη είναι δυνατός ο εντοπισμός σφαλμάτων στο πακέτο του μηνύματος που έλαβε.

Για να μειωθεί η πιθανότητα λάθους στη μετάδοση μηνυμάτων και για την αξιόπιστη μετάδοση σε πολλούς κόμβους ταυτόχρονα, έχουν αναπτυχθεί διάφορα πρωτόκολλα *Ελέγχου Πρόσβαση στο Μέσο (MAC Protocols)*. Μερικά από αυτά τα πρωτόκολλα έχουν τη δυνατότητα να καθορίζουν τη μετάδοση σε συγκεκριμένη συχνότητα για τον κάθε κόμβο (FDMA-Frequency Division Multiple Access – Πολλαπλή προσπέλαση με διαίρεση συχνότητας) αλλά μπορεί να ορίζουν στους κόμβους ένα συγκεκριμένο χρονικό διάστημα για να μεταδώσουν, έτσι ώστε να μην μπορεί κανένας να μεταδίδει ταυτόχρονα την ίδια χρονική στιγμή (*TDMA – Time Division Multiple Access – Πολλαπλή προσπέλαση με διαίρεση χρόνου*) ή μπορεί να εφαρμόζουν πολλαπλή πρόσβαση διαίρεσης κώδικα (CDMA). Εκτός από αυτά τα πρωτόκολλα, συχνά γίνεται χρήση του πρωτοκόλλου *CSMA (Carrier Sense Multiple Access)* όπου οι κόμβοι ‘αισθάνονται’ το κανάλι για ένα μικρό χρονικό διάστημα πριν στείλουν το πακέτο, έτσι ώστε να αποφεύγονται πιθανές συγκρούσεις. Αυτός είναι και ο τρόπος πρόσβασης που ακολουθείται από τα περισσότερα ασύρματα δίκτυα αισθητήρων που χρησιμοποιούν το TinyOS ως λειτουργικό σύστημα.



Σχήμα 3.2.1: Επικοινωνία στο TinyOS [11]

Το πιο πάνω σχήμα απεικονίζει την επικοινωνία στο TinyOS. Τα επίπεδα από κάτω προς τα πάνω περιγράφονται ως εξής:

- **Επίπεδο bit (RFM-RF module-μονάδα επικοινωνίας):** σε αυτό το επίπεδο μπορεί να καθορισθεί ο τρόπος λειτουργίας (εκπομπής και λήψης) και ο ρυθμός δειγματοληψίας, να ληφθεί ή να αποσταλεί ένα bit και να υπάρξει κατάλληλη ειδοποίηση για το εάν η εκπομπή ή η λήψη ολοκληρώθηκε, έτσι ώστε να μεταβεί το RFM σε ανενεργή κατάσταση ή γενικά σε κατάσταση μικρής κατανάλωσης ενέργειας.
- **Επίπεδο byte (Radio byte):** αυτό το επίπεδο είναι υπεύθυνο για την κωδικοποίηση των bits μετάδοσης σε επίπεδο byte, την ανίχνευση, τη διόρθωση λαθών και την ανίχνευση του επιπέδου ισχύος του σήματος έτσι ώστε να μπορεί να εντοπιστεί εάν το κανάλι είναι ελεύθερο για μετάδοση δεδομένων, διαφορετικά εισέρχεται η μονάδα επικοινωνίας σε αναμονή για ένα τυχαίο χρονικό διάστημα.
- **Επίπεδο πακέτου (Radio packet):** σε αυτό το επίπεδο γίνεται 16 bit crc έλεγχος και δημιουργία πλεονάζουσας εκπομπής δεδομένων,
- **Επίπεδο μηνύματος (Messaging):** εδώ, οι πληροφορίες ενσωματώνονται σε πακέτα και γίνεται δρομολόγησή τους σύμφωνα με κατάλληλες διευθύνσεις προς τη μονάδα επικοινωνίας (broadcast) ή τη σειριακή θύρα (UART). Μέρος αυτού, αποτελεί και ο AM dispatcher ο οποίος κατευθύνει τα ενεργά πακέτα προς τους αντίστοιχους χειριστές πακέτων (packet handlers).
- **Επίπεδο εφαρμογής (Application):** σε αυτό το επίπεδο αναπτύσσονται τα πρωτόκολλα δρομολόγησης, οι εφαρμογές και γενικά η επεξεργασία δεδομένων που παρέχονται από τα πακέτα μετάδοσης.

3.2.1. Γενική Δομή πακέτων TinyOS [7]

Τα πακέτα TinyOS ακολουθούν μια γενική δομή η οποία διαφοροποιείται, ανάλογα με την εφαρμογή που υλοποιεί ο κάθε χρήστης. Πιο κάτω παραθέτονται κάποια σημαντικά στοιχεία που πρέπει να γνωρίζει κάποιος για να κατανοήσει καλύτερα τη σύνθεση των πακέτων.

- Στο TinyOS ένα πακέτο έχει μέγιστο μήκος 255 bytes.
- Τα ‘πρωτογενή δεδομένα’ (raw data) περιβάλλονται και από τις δυο άκρες από ένα byte πλαισίου συγχρονισμού 0x7E. Αυτό χρησιμοποιείται για τη διαπίστωση της αρχής και του τέλους του κάθε πακέτου στη ροή των δεδομένων

- Το πακέτο ‘raw data’ χρησιμοποιεί ένα byte διαφυγής 0x7D. Αυτό χρειάζεται στην περίπτωση που ένα byte στο πεδίο ωφέλιμου φόρτου (payload data) είναι το ίδιο με ένα δεσμευμένου κώδικα byte, όπως το byte πλαισίου συγχρονισμού 0x7E. Σε αυτή την περίπτωση το byte διαφυγής θα προηγηθεί των δεδομένων του payload data και τα ίδια θα μπουν αλλά μετά την πράξη τους με ‘ΑΠΟΚΛΕΙΣΤΙΚΟ Ή’ με το 0x20. Για παράδειγμα ένα byte πεδίου ωφέλιμων δεδομένων 0x7E θα εμφανιστεί στο πακέτο δεδομένων ως 0x7D 0x5E.
- Τα δεδομένα των bytes στέλνονται και παρουσιάζονται σε little-endian μορφή. Για παράδειγμα, το πεδίο διεύθυνσης UART των 2 bytes: 0x007E θα εμφανιστεί σαν ‘7E 00’ στο πακέτο δεδομένων. Δηλαδή MSB: 0x00 και LSB: 0x7E.

Η δομή του Payload Data στο raw data πακέτο είναι struct TOS_Msg το οποίο περιέχει πεδία για τη διεύθυνση προορισμού, το είδος του μηνύματος (τον χειριστή AM ID (AM handler ID)), το μήκος, το ωφέλιμο φορτίο, κλπ. Το μέγιστο μέγεθος του ωφέλιμου φορτίου είναι TOSH_DATA_LENGTH και είναι καθορισμένο σε 29 bytes. Το πακέτο TOS_Msg θα επεξηγηθεί στη συνέχεια.

Raw Data Πακέτα [7]

Τα πιο κάτω διαγράμματα και πίνακες περιγράφουν το raw data πακέτο

Δομή raw data packet

	SYNC_BYTE	Packet Type	Payload Data	SYNC_BYTE
Byte#	0	1	2...n-1	n

Επεξήγηση raw data Πακέτου

Byte#	Πεδίο	Περιγραφή
0	SYNC_BYTE	Είναι πάντοτε 0x7E
1	Packet Type	Υπάρχουν 5 γνωστοί τύποι πακέτων: • P_PACKET_NO_ACK (0x42) – πακέτο χρήστη χωρίς απαίτηση ACK. • P_PACKET_ACK (0x41) – Πακέτο χρήστη. Απαιτείται ACK. Περιλαμβάνει ένα byte πρόθεμα. Παραλήπτης πρέπει να στείλει P_ACK απάντηση με πρόθεμα byte για περιεχόμενο. • P_ACK (0x40) - Η ACK απόκριση σε ένα πακέτο P_PACKET_ACK. Περιλαμβάνει το byte πρόθεμα στα περιεχόμενα. • P_UNKNOWN (0xFF) - Άγνωστος τύπος πακέτου.
2...n-1	Payload Data	Στις περισσότερες περιπτώσεις είναι ένα TinyOS μήνυμα ποικίλου μήκους.
n	SYNC_BYTE	Είναι πάντοτε 0x7E

3.3. Επικοινωνία στο TinyOS

Η επικοινωνία (Radio communication) στο TinyOS ακολουθεί το μοντέλο Active Message (AM), το οποίο είναι κατάλληλο για την ελαχιστοποίηση του κόστους επικοινωνίας. Στο μοντέλο αυτό κάθε πακέτο στο δίκτυο ορίζει ένα handler ID το οποίο συμπεριλαμβάνεται στους κόμβους των παραληπτών. Το handler ID είναι ένας ακέραιος αριθμός ή ένα "port number" ο οποίος μεταφέρεται από την κεφαλίδα (header) του μηνύματος. Όταν ένα μήνυμα παραληφθεί, τότε ενεργοποιείται το συμβάν λήψης (receive event) που σχετίζεται με το handler ID. Τα διαφορετικά motes μπορούν να σχετίζονται με διαφορετικά συμβάντα λήψης (receive events) με το ίδιο handler ID. Σε κάθε στρώμα μηνυμάτων, υπάρχουν 5 παράγοντες που συνεισφέρουν στην επιτυχή επικοινωνία:

1. Ο προσδιορισμός των δεδομένων για να σταλεί το μήνυμα.
2. Ο προσδιορισμός του κόμβου που πρόκειται να λάβει το μήνυμα.
3. Ο προσδιορισμός του χρόνου στον οποίο η μνήμη μπορεί να λάβει το μήνυμα.
4. Προσωρινή αποθήκευση(buffering) του εισερχόμενου μηνύματος.

5. Επεξεργασία του μηνύματος για την υποδοχή.
6. Στο Tiny Active Messages, η διαχείριση μνήμης είναι πολύ περιορισμένη.

3.3.1. Active Message (AM)[9]

Στις παραδοσιακές αρχιτεκτονικές message passing, οι διεργασίες επικοινωνούν με την αποστολή ενός αιτήματος από έναν επεξεργαστή, και τη λήψη του αιτήματος από έναν άλλο. Η αποστολή εμφανίζεται στιγμιαία στο πρόγραμμα χρήστη, αλλά το message layer buffers περιέχει το μήνυμα μέχρι το network port να είναι διαθέσιμο, τότε το μήνυμα μεταδίδεται, όπου και πάλι αποθηκεύεται στο buffer του παραλήπτη μέχρι ο παραλήπτης να το εκτελέσει. Αν και είναι απλό στην εφαρμογή, είναι αναποτελεσματική μέθοδος.

Τα Active Messages είναι ένας μηχανισμός ασύγχρονης επικοινωνίας που προορίζονται για να αποκαλύψουν το πλήρες υλικό ευελιξίας και την απόδοση των σύγχρονων δικτύων διασύνδεσης. Η βασική ιδέα είναι απλή: κάθε μήνυμα περιέχει στην κεφαλίδα του (header), τη διεύθυνση του επιπέδου του χρήστη (user-level handler) το οποίο εκτελείται κατά την άφιξή του μηνύματος με το σώμα του μηνύματος ως παράμετρο. Ο ρόλος του χειριστή είναι να πάρει το μήνυμα από το δίκτυο και να το βάλει στον υπολογισμό του κόμβου επεξεργασίας. Το κλειδί στη βελτιστοποίηση Active Messages σε σύγκριση με την αποστολή και λήψη είναι η εξάλειψη του buffering.

Το AM μοντέλο έχει σχεδιαστεί αρχικά για μεγάλης κλίμακας πολυεπεξεργαστές (multiprocessors), αλλά η περιορισμένη ισχύς, μνήμη και υπολογιστική ικανότητα του αισθητήρα το καθιστούν κατάλληλο για εφαρμογές TinyOS.

Στο TinyOS 1.x, ο message buffer είναι τύπου TOS_Msg. Ο buffer περιέχει ένα πακέτο ενεργού μηνύματος (active message-AM), καθώς και ένα πακέτο με μεταδεδομένα, όπως τα timestamps, τα bits για αναγνώριση (acknowledgement bits), και την ισχύ του σήματος αν το πακέτο ελήφθη. Το TOS_Msg είναι μια σταθερή δομή της οποίας το μέγεθος καθορίζεται από το μέγιστο μήκος του ωφέλιμου φορτίου AM (η προεπιλογή είναι 29 bytes).

Ένα ζήτημα που προκύπτει είναι αυτό που καθορίζει τη δομή TOS_Msg, όπου διαφορετικά στρώματα σύνδεσης μπορεί να απαιτούν διαφορετικές διατάξεις. Για παράδειγμα, το radio hardware 802.15.4 (όπως το CC2420, που χρησιμοποιούνται στις πλατφόρμες Telos και micaZ) μπορεί να απαιτούν 802.15.4 κεφαλίδες(headers), το CC1000, που χρησιμοποιείται για την πλατφόρμα mica2 μπορεί να προσδιορίσει τη δική του μορφή πακέτων. Αυτό σημαίνει ότι TOS_Msg μπορεί να είναι διαφορετικό σε διαφορετικές πλατφόρμες.

Στο TinyOS, το αρχικό πακέτο Active message είναι TOS_Msg και όλα τα άλλα είδη μηνυμάτων είναι έγκλειστα σε αυτό. Επίσης το Payload Data του raw data είναι struct TOS_Msg.

Η δομή του TOS_Msg έχει ως εξής: [8, 10,12]

```
typedef struct TOS_Msg {
    // The following fields are transmitted/received on the radio.
    uint16_t addr;
    uint8_t type;
    uint8_t group;
    uint8_t length;
    int8_t data[TOSH_DATA_LENGTH];
    uint16_t crc;
    uint16_t strength;
    uint8_t ack;
    uint16_t time;
    uint8_t sendSecurityMode;
    uint8_t receiveSecurityMode;
} TOS_Msg;
```

Δομή TOS_Msg πακέτου

	Address		Message Type	Group ID	Data Length	Data	CRC	
Byte#	0	1	2	3	4	5...n-2	n-1	n

Περιγραφή TOS_Msg πακέτου

Byte#	Πεδίο	Περιγραφή
0-1	Address	Καθορίζει τη διεύθυνση προορισμού (moteID ή η διεύθυνση μετάδοσης). Ένας απ'τους 3 τύπους τιμών: • Broadcast Address (0xFFFF) – μήνυμα σε όλους τους κόμβους. • UART Address (0x007e)- μήνυμα από ένα κόμβο σε μια σειριακή θύρα. Όλα τα εισερχόμενα μηνύματα θα έχουν αυτή τη διεύθυνση. • Node Address – Η μοναδική ταυτότητα(ID) ενός κόμβου για τη λήψη μηνύματος.
2	Message Type	Ορίζει ποιος handler θα κληθεί σε επίπεδο AM όταν λάβει ένα πακέτο. Active Message (AM) μοναδικό αναγνωριστικό για τον τύπο μηνύματος. Τυπικά κάθε εφαρμογή έχει το δικό του τύπο μηνύματος. Παραδείγματα: • AMTYPE_XUART = 0x00 • AMTYPE_MHOP_DEBUG = 0x03 • AMTYPE_SURGE_MSG = 0x11 • AMTYPE_XSENSOR = 0x32 • AMTYPE_XMULTIHOP = 0x33 • AMTYPE_MHOP_MSG = 0xFA
3	Group ID	Καθορίζει ένα κανάλι για motes σε ένα δίκτυο. Αν ένα mote λάβει ένα πακέτο με ένα διαφορετικό αναγνωριστικό ομάδας, το πακέτο χάνεται. Μοναδικά αναγνωρίσιμο για την ομάδα των κόμβων που συμμετέχουν στο δίκτυο. Η εξ' ορισμού τιμή είναι: 125 (0x7d). Μόνο τα motes με το ίδιο groupID θα επικοινωνούν μεταξύ τους
4	Data Length	Καθορίζει το μήκος του τμήματος δεδομένων του TOS_Msg. Αυτό δεν περιλαμβάνει τον CRC ή bytes συγχρονισμού πλαισίου. Τα πακέτα έχουν μέγιστο ωφέλιμο φορτίο 29 bytes.
5...n-2	Payload data	Καθορίζει το πραγματικό περιεχόμενο του μηνύματος και είναι το μήκος του τμήματος δεδομένων του TOS_Msg. Τα πακέτα Το τμήμα δεδομένων αποτελείται από μια σειρά από 29 bytes (όπως καθορίζεται από το TOSH_DATA_LENGTH). Τα δεδομένα βρίσκονται στα byte 5 έως byte 5 σύν το μήκος των δεδομένων. Τα δεδομένα είναι ανάλογα με τον τύπο μηνύματος.
n-1,n	CRC	Κώδικας των 2 byte ο οποίος διασφαλίζει την ακεραιότητα του μηνύματος. Ο CRC περιέχει τον Packet Type συν ολόκληρο το unescaped TinyOS μήνυμα. Κατά την αποστολή, το CRC υπολογίζεται κλιμακωτά όπως κάθε byte από το πακέτο που μεταδίδεται. Τα τελευταία τρία πεδία του TOS_Msg είναι μετα-δεδομένα σχετικά με το πακέτο και δεν μεταδίδονται. Το ack αποστέλλεται από το δέκτη, και προσδιορίζονται από τον αποστολέα. Αυτός είναι ο μηχανισμός που μπορεί να παρέχει αξιοπιστία στην στοίβα.

Το μέγιστο μήκος ενός πακέτου TOS_Msg είναι 36 bytes:

addr (2 bytes) + type (1 bytes) + group (1 bytes) + length(1 bytes) + data(29 bytes) +
crc(2 bytes) = 36 bytes.

Multihop Μήνυμα

Σε πολλές περιπτώσεις, ειδικότερα σε εφαρμογές που αξιοποιούν ad hoc δικτύωση βρόχου, η εφαρμογή χρησιμοποιεί το πρωτόκολλο Multihop μηνύματος. Ο ορισμός του μηνύματος Multihop φαίνεται παρακάτω:

```
typedef struct MultihopMsg {  
    uint16_t sourceaddr;  
    uint16_t originaddr;  
    int16_t seqno;  
    uint8_t hopcount;  
    uint8_t data[(TOSH DATA LENGTH - 7)];  
}
```

Η μορφή και η περιγραφή του πακέτου φαίνονται παρακάτω:

Δομή MultihopMsg πακέτου

	Source Address		Origin Address		Sequence Number		Hop Count	Application Data
Byte#	0	1	2	3	4	5	6	7...n

Περιγραφή MultihopMsg πακέτου

Byte#	Πεδίο	Περιγραφή
0,1	Source Address	Η διεύθυνση του κόμβου που έκανε την προώθηση.
2,3	Origin Address	Η διεύθυνση του κόμβου που δημιούργησε το μήνυμα
4,5	Sequence Number	Χρησιμοποιείται για τον προσδιορισμό της διαδρομής και τον υπολογισμό χαμένων πακέτων
6	Hop Count	Χρησιμοποιείται για τον αριθμητικό υπολογισμό της διαδρομής. Αριθμός διασχιζόμενων κόμβων.
7...n	Application Data	Τα επακριβή δεδομένα. Ανάλογα με την εφαρμογή

Τέλος, τα δεδομένα μέσα στο πεδίο application data του Multihop Message είναι raw data ειδικά ορισμένα για να ικανοποιούν την εφαρμογή του τελικού χρήστη. Ο τύπος των δεδομένων προσδιορίζεται από το πεδίο Message Type στο 2^ο byte του TinyOS μηνύματος.

Τα ακριβή δεδομένα για την εφαρμογή TinyDB που βασίζεται στο σύστημα KSPOT, για την αποδοτική παρακολούθηση των Κορυφαίων-K (Top-K) απαντήσεων σε μία επερώτηση Q σε ένα ασύρματο δίκτυο αισθητήρων ανήκουν στη δομή struct PTopkRoomData.

PTopkRoomData struct

```
typedef struct {
    uint8_t sameAsPrevious:1;
    uint8_t vals:3;
    uint8_t hopcount:4;
    uint32_t count:22;
    uint8_t junk2:2;
    uint32_t room:22;
    uint8_t junk3:2;
    uint16_t sum[roomCount];
} __attribute__((__packed__)) PTopkRoomData;
```

Περιγραφή PTopkRoomData

Πεδίο	Περιγραφή
Value	Ο αριθμός των τιμών που υπάρχουν στο PTopkRoomData πακέτο μέχρι στιγμής. Στο τελευταίο βήμα όπου το πακέτο θα αποσταλεί στο sink, πεδίο αυτό θα ισούται με την παράμετρο k, που δίνεται από την επερώτηση.
Same As Previous	Το πεδίο αυτό παίρνει τιμές 0 ή 1. Εάν η τιμή είναι 0 τότε η προηγούμενη κατάσταση είναι η ίδια με την τρέχουσα κατάσταση. Εάν η τιμή είναι 1 τότε οι δύο καταστάσεις είναι διαφορετικές.
Sum[roomCount]	Ανάλογα με τον αριθμό των δωματίων που υπάρχουν δημιουργείται ένας πίνακας ίσος με το μέγεθος των δωματίων και περιέχει το άθροισμα των μετρήσεων των αισθητήρων του κάθε δωματίου.
Count	Κρατά για το κάθε δωμάτιο τον αριθμό των αισθητήρων από τους οποίους διάβασε τιμές μέχρι στιγμής
Room	Κρατά τον αριθμό των αισθητήρων για το κάθε δωμάτιο
Hopcount	Χρησιμοποιείται για τον αριθμητικό υπολογισμό της διαδρομής. Αριθμός διασχιζόμενων κόμβων.

3.4. TinyDB: Μια δηλωτική Βάση Δεδομένων για το TinyOS [13]

Τα ασύρματα δίκτυα αισθητήρων όπως έχω αναφέρει ήδη, είναι ένας νέος χώρος στην πληροφορική που ασχολείται με τη συλλογή τεράστιων ποσοτήτων δεδομένων από ένα θεωρητικά απεριόριστο δίκτυο ασύρματων αισθητήρων, που είναι γνωστά ως Motes. Τα Motes παρακολουθούν το φυσικό περιβάλλον, επιτρέποντας στο χρήστη να εκτελέσει οποιοδήποτε αριθμό διαφορετικών αναλυτικών εργασιών με τα δεδομένα. Για να γίνει αυτό τα δεδομένα πρέπει να αποθηκεύονται κάπου που να είναι εύκολα προσβάσιμα από ένα πρόγραμμα χρήστη. Υπάρχουν πολλοί τρόποι για την αποθήκευση δεδομένων σε ασύρματα δίκτυα αισθητήρων. Δύο πιθανές μέθοδοι αποθήκευσης είναι:

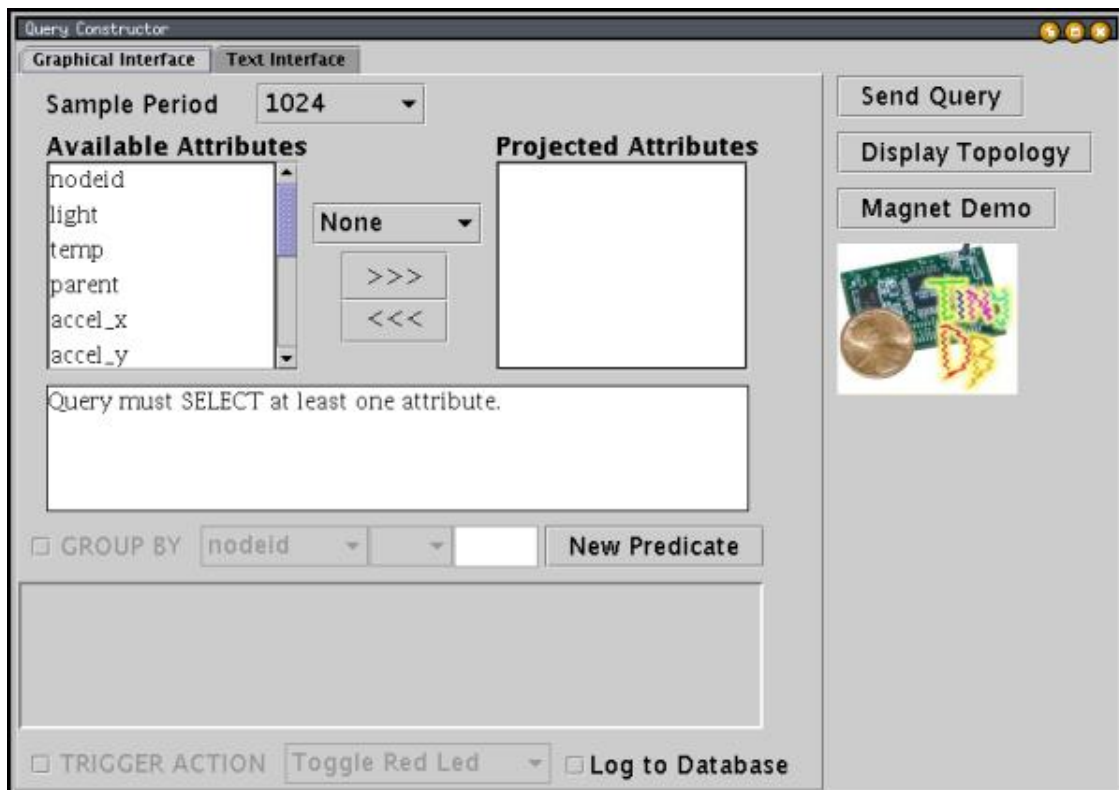
1. Συγκεντρωτική αρχιτεκτονική - σε κεντρικό εξυπηρετητή SQL
2. Κατανεμημένη αρχιτεκτονική - Στη μνήμη του κάθε Mote

Όταν γίνεται αποθήκευση των δεδομένων σε τοπικό επίπεδο, χρησιμοποιούνται επερωτήματα των χρηστών του δικτύου, προκειμένου να ανακτήσουν τις απαιτούμενες πληροφορίες. Αυτή η κατανεμημένη αποθήκευση πραγματοποιείται από ένα πρόγραμμα που ονομάζεται TinyDB. Το TinyDB διαχειρίζεται το υποκείμενο δίκτυο και παρέχει μια απλή διεπαφή επερωτήσεων, αφαιρώντας την ανάγκη για το χαμηλό επίπεδο κώδικα προγράμματος. Αντίθετα με τις υπάρχουσες λύσεις για την επεξεργασία των δεδομένων όπως το TinyOS, το σύστημα TinyDB δεν απαιτεί να γράψουμε ενσωματωμένο C κώδικα για αισθητήρες. Αντίθετα, παρέχει μια απλή διεπαφή, όπως την SQL για να καθορίσει τα στοιχεία που θέλουμε να εξάγουμε, μαζί με πρόσθετες παραμέτρους, όπως και το ρυθμό με τον οποίο τα δεδομένα θα πρέπει να ανανεώνονται, όσο θέτουμε ερωτήματα με βάση μια παραδοσιακή βάση δεδομένων. Δοσμένου ενός ερωτήματος το TinyDB συλλέγει δεδομένα από το περιβάλλον των motes, τα φιλτράρει, κάνει τη συνάθροιση μεταξύ τους, και τα δρομολογεί προς τον υπολογιστή. Για να κάνει αυτές τις λειτουργίες το TinyDB χρησιμοποιεί αποτελεσματικούς αλγορίθμους επεξεργασίας δεδομένων στο δίκτυο, ένα από τους οποίους είναι και ο αλγόριθμος που μελετάμε, ο MINT.

Πρωταρχικός στόχος του TinyDB είναι να κάνει τη ζωή του προγραμματιστή ευκολότερη, καθώς τον απαλλάσσει από το βάρος της γραφής κώδικα χαμηλού

επιπέδου για τις συσκευές αισθητήρα, συμπεριλαμβανομένων των διασυνδέσεων του δικτύου αισθητήρων.

Για να χρησιμοποιηθεί το TinyDB, χρειάζεται εγκατάσταση των TinyOS components πάνω σε κάθε Mote στο δίκτυο αισθητήρων. Στην παρούσα μελέτη δεν χρειάζεται η εγκατάσταση στον κάθε κόμβο λόγω του ότι χρησιμοποιούμε προσομοιωτή. Το TinyDB παρέχει μια απλή Java API για την σύνταξη των εφαρμογών των ερωτήσεων και την εξαγωγή δεδομένων από το δίκτυο.



Σχήμα 3.3.1: TinyDB Query Constructor[12]

Μερικά από τα χαρακτηριστικά του TinyDB περιλαμβάνουν:

- Διαχείριση Μεταδεδομένων: παρέχει ένα κατάλογο με μετα-δεδομένα για να περιγράψει το είδος των μετρήσεων του αισθητήρα που είναι διαθέσιμες στο δίκτυο αισθητήρων.
- Υψηλού Επιπέδου Επερωτήσεις: χρησιμοποιεί μια δηλωτική γλώσσα ερωτημάτων που επιτρέπει να περιγραφούν τα δεδομένα που θέλουμε να εξάγουμε χωρίς να απαιτείται να του πούμε πως να το εξάγει. Αυτό μας βοηθά να γράφουμε πιο εύκολα

εφαρμογές, και εξασφαλίζει ότι οι εφαρμογές μας θα συνεχίσουν να λειτουργούν αποδοτικά, καθώς το δίκτυο αισθητήρων αλλάζει τοπολογία.

- Τοπολογία Δικτύου: διαχειρίζεται το ασύρματο δίκτυο με τους γείτονες, διατηρώντας πίνακες δρομολόγησης, και διασφαλίζει ότι κάθε Mote στο δίκτυο μπορεί αποτελεσματικά να εκδίδει αξιόπιστα δεδομένα προς το χρήστη.
- Πολλαπλά Queries: επιτρέπει πολλαπλές επερωτήσεις που πρέπει να λειτουργούν με το ίδιο σύνολο motes ταυτόχρονα. Τα ερωτήματα θα μπορούν να έχουν διαφορετικούς ρυθμούς δειγματοληψίας και πρόσβαση σε διαφορετικούς τύπους αισθητήρων. Το TinyDB κάνει αποτελεσματική κατανομή της εργασίας μεταξύ των ερωτημάτων όταν αυτό είναι δυνατόν.
- Ανάπτυξη μέσω Query Sharing: Για την επέκταση του δικτύου αισθητήρων στο TinyDB, μπορούμε απλά να κατεβάσουμε το πρότυπο του κώδικα TinyDB σε νέα motes, και το TinyDB κάνει τα υπόλοιπα. Τα motes που τρέχουν ερωτήσεις μοιράζονται τα αποτελέσματα μεταξύ τους: όταν ένα mote ακούει ένα μήνυμα από το δίκτυο για ένα ερώτημα που δεν τρέχει ακόμη, ζητά αυτόματα από τον αποστολέα των εν λόγω δεδομένων ένα αντίγραφο του ερωτήματος, και αρχίζει να τρέχει. [14]

3.4.1. Αρχιτεκτονική Λογισμικού TinyDB [14,15]

Το TinyDB έχει σχεδιαστεί έτσι ώστε να είναι προσιτό στους χρήστες οι οποίοι δεν ενδιαφέρονται για τις τεχνικές λεπτομέρειες της εφαρμογής του συστήματος.

Πιο κάτω δίνεται ένα παράδειγμα της χρήσης του TinyDB. Με ένα απλό ερώτημα SQL μπορεί να εντοπίσει πολύ εύκολα τα δωμάτια που βρίσκονται σε χρήση:

```
SELECT roomno, AVG(light), AVG(volume)
FROM sensors
GROUP BY roomno
HAVING AVG(light) > 1 AND AVG(volume) > v
EPOCH DURATION 5min
```

Το TinyDB μεταφράζει αυτό το ερώτημα σε ένα αποτελεσματικό σχέδιο εκτέλεσης που παρέχει το σύνολο των δωματίων που βρίσκονται σε χρήση, κάθε 5 λεπτά.

Το σύστημα μπορεί να κατηγοριοποιηθεί σε γενικές γραμμές σε δύο υποσυστήματα:

1. Sensor Network λογισμικό: Αυτή είναι η καρδιά του TinyDB, αν και οι περισσότεροι χρήστες του συστήματος δεν πρέπει να χρειαστεί να τροποποιήσουν αυτό το κώδικα. Τρέχει σε κάθε Mote στο δίκτυο, και αποτελείται από πολλά σημαντικά κομμάτια:

- Κατάλογος αισθητήρα και Schema Manager: Ο κατάλογος είναι υπεύθυνος για την παρακολούθηση του συνόλου των χαρακτηριστικών, ή τους τύπους των μετρήσεων (π.χ. φως, ήχο, τάση) και τις ιδιότητες (π.χ. μητρική δικτύου, κόμβου) που διατίθενται για κάθε αισθητήρα. Σε γενικές γραμμές, ο κατάλογος αυτός δεν είναι ίδιος για κάθε αισθητήρα: τα δίκτυα μπορούν να αποτελούνται από ετερογενείς συλλογές συσκευών, και μπορεί να είναι σε θέση να αναφέρουν διαφορετικές ιδιότητες.
- Query Processor: Το κύριο συστατικό του TinyDB αποτελείται από ένα μικρό επεξεργαστή ερωτήματος. Ο Query Processor χρησιμοποιεί τη λίστα για να πάρει τις τιμές των τοπικών χαρακτηριστικών, λαμβάνει τις μετρήσεις αισθητήρα από γειτονικούς κόμβους πάνω από το radio, συνδυάζει και συναθροίζει τις τιμές αυτές μεταξύ τους, φιλτράρει για να μην πάρει τις ανεπιθύμητες τιμές και στέλνει τις τιμές στον κόμβο του πατέρα.
- Διαχείριση μνήμης: το TinyDB φθάνει στο TinyOS με μια μικρή handle-base δυναμική διαχείριση μνήμης.
- Δίκτυο Τοπολογία Manager: το TinyDB διαχειρίζεται τη συνδεσιμότητα των motes στο δίκτυο, για την αποτελεσματική δρομολόγηση των δεδομένων και τα υπο-αποτελέσματα των επερωτήσεων μέσω του δικτύου.

2. Java-based Client Interface: Ένα δίκτυο TinyDB motes είναι προσβάσιμο από ένα συνδεδεμένο υπολογιστή μέσω της διεπαφής πελάτη TinyDB, το οποίο αποτελείται από ένα σύνολο από κλάσεις Java και εφαρμογές. Οι κατηγορίες αυτές αποθηκεύονται στο `tinyos-1.x/tools/java/tinyos/tinydb` πακέτο στο δέντρο πηγής. Οι βασικές κατηγορίες περιλαμβάνουν:

- Μια κλάση διασύνδεσης δικτύου που επιτρέπει στις εφαρμογές να στείλουν ερωτήματα και να ακούσουν τα αποτελέσματα.
- Κλάσεις για την κατασκευή και μετάδοση των επερωτήσεων.
- Μια κλάση για να λαμβάνει και να αναλύει τα αποτελέσματα του ερωτήματος.

- Μια κλάση για την εξαγωγή πληροφοριών σχετικά με τα χαρακτηριστικά και τις δυνατότητες των συσκευών.
- Ένα γραφικό περιβάλλον για την κατασκευή επερωτήσεων.
- Ένα διάγραμμα και πίνακα GUI για να εμφανιστούν τα μεμονωμένα αποτελέσματα αισθητήρα.
- Ένα GUI για να απεικονίσει δυναμικά τις τοπολογίες δικτύου.
- Μια εφαρμογή που χρησιμοποιεί ερωτήματα ως διεπαφή πάνω από ένα δίκτυο αισθητήρων.

3.5. TOSSIM [16]: Ένας προσομοιωτής για το TinyOS

Ο TOSSIM [15] είναι ένας προσομοιωτής διακριτών συμβάντων (discrete event) ειδικά σχεδιασμένος για το TinyOS. Έχει την δυνατότητα να αντιλαμβάνεται και να αναπαριστά την συμπεριφορά και τις αλληλεπιδράσεις σε δίκτυα χιλιάδων κόμβων (moten).

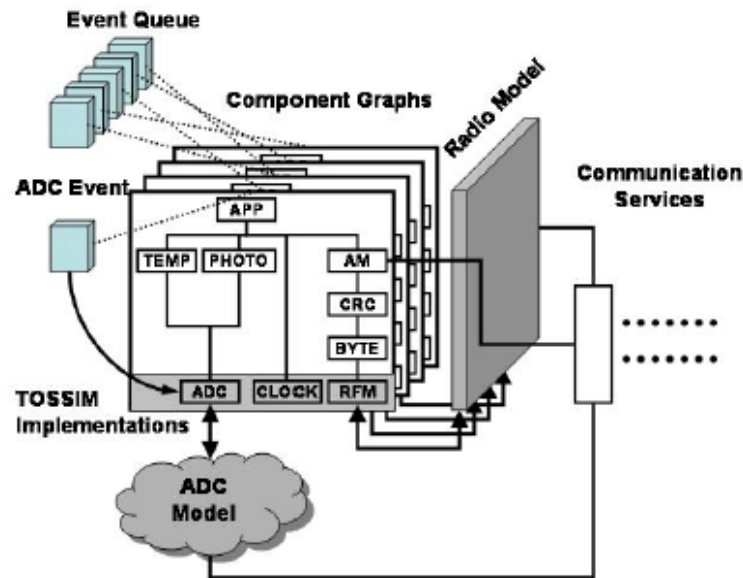
3.5.1. Αρχιτεκτονική TOSSIM [16,18]

Η αρχιτεκτονική του TOSSIM αποτελείται από 5 τμήματα:

- α)Υποστήριξη για το μεταγλωττισμό των συστατικών του TinyOS στη δομή της προσομοίωσης.
- β)Μια ουρά διακριτών συμβάντων (discrete event queue).
- γ)Ένα μικρό αριθμό από υλοποιημένα συστατικά υλικού του TinyOS.
- δ)Μηχανισμούς για επεκτάσιμα μοντέλα ασυρμάτου (radio) και αναλογικό-ψηφιακού μετατροπέα (ADC).
- ε)Υπηρεσίες επικοινωνιών ώστε εξωτερικά προγράμματα να μπορούν να αλληλεπιδρούν με τον προσομοιωτή.

Ο TOSSIM εκτελεί τον ίδιο κώδικα που εκτελείται και στους κόμβους. Έτσι είναι δυνατό γράφοντας κώδικα για τον προσομοιωτή να τον χρησιμοποιήσουμε στο πραγματικό δίκτυο ή και το αντίστροφο.

Στο Σχήμα 3.5.1.1 φαίνονται τα συστατικά του υλικού TinyOS των κόμβων που αντικαθίστανται από το TOSSIM.



Σχήμα 3.5.1.1: Αρχιτεκτονική TOSSIM[18]

Επιπλέον ο προσομοιωτής χρησιμοποιεί μια πολύ απλή αλλά πολύ δυνατή αφαίρεση (abstraction) για το σχεδιασμό του ασύρματου δικτύου. Το δίκτυο αναπαρίσταται σαν ένας άμεσος γράφος στον οποίο κάθε κορυφή είναι ένας κόμβος και κάθε ακμή έχει μια πιθανότητα λάθους bit (bit error).

- **TOSSIM: Μοντέλο Εκτέλεσης [18]:**

Στον πυρήνα του TOSSIM βρίσκεται μια ουρά συμβάντων. Οι διακοπές μοντελοποιούνται μέσω συμβάντων του προσομοιωτή, τα οποία είναι διαφορετικά από τα συμβάντα στο TinyOS. Ένα συμβάν του προσομοιωτή καλεί ένα χειριστή διακοπών (interrupt handler) από τα συστατικά του υλικού. Αυτός ο χειριστής στέλνει σήματα συμβάντων και καλεί εντολές του TinyOS όπως ακριβώς θα συνέβαινε και στην πραγματικότητα σε ένα κόμβο.

Ο TOSSIM τηρεί ένα επίπεδο λεπτομέρειας χρόνου 4MHz, όπως ακριβώς και το ρολόι οδηγίων του κάθε κόμβου. Όλα τα συμβάντα προσομοιωτή σημειώνονται με ένα συγκεκριμένο χρόνο και επεξεργάζονται βάσει αυτού του χρόνου. Τα συμβάντα

εκτελούνται στιγμιαία όταν έρθει η σειρά εκτέλεσης τους που καθορίζεται από τον χειριστή τους. Έτσι υπάρχει μια σειρά στην εκτέλεση τους. Κάθε συμβάν στον προσομοιωτή συνδέεται με ένα συγκεκριμένο κόμβο. Μόλις εκτελεστεί ένα τέτοιο συμβάν στον προσομοιωτή, ο χρόνο-προγραμματιστής εκτελεί τις διεργασίες (tasks) στην ουρά αναμονής διεργασιών αυτού του κόμβου ακολουθώντας το μοντέλο FIFO του κανονικού χρόνο-προγραμματιστή του TinyOS. Εδώ υπάρχει και η διαφορά με το TinyOS, ότι οι διεργασίες εκτελούνται με την σειρά χωρίς να υπάρχει δυνατότητα κάποια να αλλάξει τη σειρά εκτέλεσης της. Αυτό οφείλεται στο γεγονός ότι ο χρόνος εκτέλεσης στον προσομοιωτή είναι στιγμιαίος και δεν λαμβάνεται υπόψη ο χρόνος επεξεργασίας από το CPU.

- **TOSSIM: Προσομοίωση Υλικού:**

Ο TOSSIM προσομοιώνει τη συμπεριφορά του υλικού του TinyOS, προσομοιώνοντας μερικά από τα συστατικά του υλικού. Αυτά είναι ο αναλογικό-ψηφιακός μετατροπέας (ADC), το ρολόι, το ποτενσιόμετρο εκπομπής μεταβλητής ισχύος, η EEPROM, το συστατικό σειράς εκκίνησης (boot sequence component) και αρκετά συστατικά της στοίβας του ασυρμάτου.

- Τα μοντέλα RF και ο μετατροπέας αναλογικού σε ψηφιακό (ADC) χρησιμοποιούνται για να μοντελοποιήσουν τα χαρακτηριστικά των TinyOS motes.

Τα RF μοντέλα ορίζουν τα χαρακτηριστικά που αφορούν τη μετάδοση από κόμβο σε κόμβο. Ένα ελάττωμα στο RF μοντέλο του TOSSIM είναι ότι η απόσταση δεν επηρεάζει την ισχύ του σήματος δημιουργώντας παρεμβολές χειρότερες από τις πραγματικές συμπεριφορές. Υπάρχουν δύο είδη RF μοντέλων:

- Το απλό(simple) και αυτό που προκαλεί απώλειες ενέργειας, το lossy. Το απλό μοντέλο RF τοποθετεί όλους τους κόμβους σε ένα μοναδικό κελί (cell), όπου κάθε bit που μεταδίδεται, λαμβάνεται χωρίς καθόλου σφάλμα. Παρόλα αυτά όμως, ενώ κανένα bit δεν καταστρέφεται λόγω σφάλματος, δύο motes που εκπέμπουν ταυτόχρονα οδηγούν στο πρόβλημα της επικάλυψης των σημάτων, αλλά η πιθανότητα να εκπέμπουν δύο motes είναι πολύ χαμηλή εξαιτίας του πρωτοκόλλου CSMA του TinyOS.
- Το lossy μοντέλο RF τοποθετεί τους κόμβους σε έναν κατευθυνόμενο γράφο και κάθε ακμή (a,b) του γράφου σημαίνει ότι το σήμα του κόμβου a μπορεί να ακουστεί στον κόμβο b. Κάθε ακμή έχει μια τιμή που αναπαριστά την

πιθανότητα ένα bit που αποστέλλεται από τον κόμβο a να είναι κατεστραμμένο, όταν ο b το ακούσει.

Το γράφημα του μοντέλου lossy μπορεί να καθορίζεται κατά την εκκίνηση του προσομοιωτή TOSSIM από ένα αρχείο. Ο TOSSIM αναζητά το αρχείο "lossy.nss" από προεπιλογή, αλλά ένα εναλλακτικό αρχείο μπορεί να καθοριστεί με την παράμετρο -rf.

Το αρχείο έχει την ακόλουθη μορφή:

```
<mote ID>:<mote ID>:bit error rate  
<mote ID>:<mote ID>:bit error rate  
<mote ID>:<mote ID>:bit error rate
```

Για παράδειγμα,
0:1:0.012333
1:0:0.009112
1:2:0.013196

Διευκρινίζει ότι το mote 1 ακούει το mote 0 με ποσοστό σφάλματος bit του 1,2%, το mote 0 ακούει το mote 1, με ποσοστό σφάλματος bit του 0,9%, και το mote 2 ακούει το mote 1, με ποσοστό σφάλματος bit της τάξης του 1,3%.

Για την αξιολόγηση του αλγορίθμου MINT χρησιμοποιήθηκε το μοντέλο lossy καθώς χρειαζόμαστε μια multihop επικοινωνία μεταξύ των Motes.

- Επιπλέον, ο TOSSIM παρέχει δύο ADC μοντέλα : το τυχαίο (random) και το γενικής χρήσης (generic). Ο ADC έχει διάφορα κανάλια που μπορούν να δειγματοληφθούν. Στο τυχαίο μοντέλο, όποτε ένα κανάλι δειγματοληπτείται επιστρέφει μια τυχαία 10-bit τιμή. Για τους σκοπούς της μελέτης μας σε αυτό το σημείο αντί να επιστρέφει μια τυχαία 10-bit τιμή επιστρέφεται η τιμή από τα dataset που χρησιμοποιούμε για τη συγκεκριμένη χρονική στιγμή.

Το γενικό μοντέλο παρέχει τις τυχαίες τιμές. Επιπλέον, παρέχει την πιθανότητα να ενεργοποιηθεί από εξωτερικές εφαρμογές, θέτοντας την τιμή για οποιαδήποτε ADC θύρα σε οποιοδήποτε mote.

- **TOSSIM: Μοντέλο Ασυρμάτου:**

Το TOSSIM παρέχει μηχανισμούς ώστε να είναι δυνατή η επιλογή της ακρίβειας και της πολυπλοκότητας του μοντέλου του ασυρμάτου που χρησιμοποιείται στις προσομοιώσεις. Τα μοντέλα είναι εξωτερικά του προσομοιωτή, ο οποίος παραμένει απλός. Όπως αναφέρθηκε και παραπάνω το TOSSIM αποτελείται από ένα γράφο κόμβων που οι συνδέσεις μεταξύ τους αναπαριστούν μια πιθανότητα λάθους bit (bit error). Ανάλογα με την πιθανότητα λάθους της μετάδοσης, ο κόμβος που λαμβάνει μεταδίδει αυτά που έλαβε στα συστατικά του TinyOS. Για παράδειγμα έστω ότι ο κόμβος A μεταδίδεται προς τον κόμβο B σε ένα κανάλι χωρίς λάθη. Σε κάθε συμβάν αποστολής ενός bit μεταβάλλεται η εσωτερική κατάσταση του B, αναπαριστώντας τι ακούει από το κανάλι. Σε κάθε bit που λαμβάνει ο B διαβάζει αυτή την κατάσταση και στέλνει το bit προς το ανάλογο συστατικό του TinyOS.

- **TOSSIM: Επίπεδο ζεύξης δεδομένων:**

Το πιο πολύπλοκο σύστημα του TinyOS είναι η στοίβα δικτύου. Αποτελείται από 12 συστατικά και χρησιμοποιεί CSMA, μονό έλεγχο και διπλή ανίχνευση λαθών καθώς και κωδικοποίηση δεδομένων με ένα πλήρες πακέτο CRC. Στο TinyOS χρησιμοποιούνται 3 διαφορετικοί ρυθμοί αποστολής και λήψης των πακέτων: 40Kbps για δεδομένα, 20Kbps για λήψη ενός συμβόλου αρχής και 10Kbps για αποστολή ενός συμβόλου αρχής. Στο TOSSIM οι ρυθμίσεις στο ρυθμό μετάδοσης γίνονται αλλάζοντας την περίοδο μεταξύ του ρολογιού συμβάντων του ασυρμάτου.

- **TOSSIM: Υπηρεσίες επικοινωνιών:**

Το TOSSIM παρέχει μηχανισμούς που επιτρέπουν στις εφαρμογές να χειρίζονται και να παρακολουθούν προσομοιώσεις επικοινωνώντας με τον προσομοιωτή μέσω TCP/IP. Αν δούμε την επικοινωνία αυτή από την πλευρά του TinyOS μπορούμε να πούμε ότι το πρωτόκολλο προσομοιωτή -εφαρμογής είναι σαν μια διεπαφή εντολής-συμβάντος. Το πρωτόκολλο επικοινωνίας είναι γενικό επιτρέποντας έτσι στους σχεδιαστές να γράψουν τα δικά τους προγράμματα όπου αυτό είναι αναγκαίο.

3.5.2. Μεταγλώττιση και Εκτέλεση TOSSIM [12,17]

Ο προσομοιωτής TOSSIM μεταγλωττίζεται χρησιμοποιώντας την εντολή **make pc** στον κατάλογο που βρίσκεται η εφαρμογή που θέλουμε να εκτελέσουμε. Το εκτελέσιμο που δημιουργείται με την εντολή `make pc` είναι το `build/pc/main.exe`. Πληκτρολογώντας `build/pc/main.exe --help` μπορούμε να δούμε τι παράμετροι μπορούν να χρησιμοποιηθούν. Το TOSSIM έχει μια παράμετρο για να δώσουμε τον αριθμό των κόμβων που θέλουμε να προσομοιώσουμε.

Πληκτρολογώντας `build/pc/main.exe 1` ο προσομοιωτής τρέχει για ένα κόμβο. Η προσομοίωση τερματίζεται με το `ctrl-C`.

3.5.3. Χρήση LossyBuilder για τη Δημιουργία του Αρχείου lossy[16]

Ο LossyBuilder μπορεί να διαβάσει ή να δημιουργήσει φυσική τοπολογία (x, y) συντεταγμένων και να δημιουργήσει τοπολογίες με απώλεια που προέρχονται από φυσικές τοπολογίες με δειγματοληψία από το μοντέλο της εμπειρικής κατανομής.

Μπορούμε να δημιουργήσουμε το lossy αρχείο με την εντολή:

java net.tinyos.sim.LossyBuilder [παράμετροι]

Παράμετροι:

- t grid: Topology (grid only and default)
- d <m> <n>: Grid size (m by n) (default: 10 x 10)
- s <scale>: Spacing factor (default: 5.0)
- o <file>: Output file
- i <file>: Input file of positions
- p: Generate positions, not error rates

Για παράδειγμα:

```
java net.tinyos.sim.LossyBuilder-d 7 2 -s 20 -o 7x2_20.nss
```

θα εμφανίσει μια τοπολογία 7 επί 2 πλέγμα, με είκοσι διαστήματα, και θα γραφτούν στο αρχείο "7x2-20.nss".

3.5.4. Γραφικό Περιβάλλον (GUI) [12,17]

Το TinyViz είναι το εργαλείο απεικόνισης του TOSSIM. Μέσω αυτού είναι δυνατόν να απεικονιστούν οι υπηρεσίες επικοινωνιών του TOSSIM. Βασίζεται στην Java και επιτρέπει την απεικόνιση, τον έλεγχο και την ανάλυση των προσομοιώσεων. Προσφέρει μια plugin διεπαφή επιτρέποντας στους σχεδιαστές να υλοποιήσουν τα δικά τους γραφικά περιβάλλοντα και κώδικα ώστε να μελετήσουν συγκεκριμένες εφαρμογές. Ουσιαστικά, το εργαλείο TinyViz από μόνο του δεν διαθέτει πλούσια λειτουργικότητα, εκτός από τη διαχείριση της διεπαφής συμβάντος/ εντολής (event/command interface) του TOSSIM. Οι χρήστες αλληλεπιδρούν με τον προσομοιωτή κάνοντας χρήση συγκεκριμένων plugins που προσφέρουν την επιθυμητή λειτουργικότητα. Η μηχανή του TinyViz στέλνει τα συμβάντα του TOSSIM στα plugins. Έτσι είναι δυνατόν ένα plugin δικτύου να απεικονίσει την κίνηση του δικτύου καθώς διακινούνται μηνύματα μέσα σε αυτό. Τα plugins μπορούν επίσης να στέλνουν εντολές προς το TOSSIM ξεκινώντας μια προσομοίωση. Για παράδειγμα αν ένας χρήστης θέσει στο εργαλείο απεικόνισης ένα κόμβο εκτός λειτουργίας, τότε το plugin ελέγχου στέλνει μια αντίστοιχη εντολή προς το TOSSIM.

Το TinyViz έχει ένα σύνολο θεμελιωδών plugins που προσφέρουν τις βασικές δυνατότητες ανάλυσης και λειτουργίας. Μερικά ένα από αυτά είναι το plugin του του ασυρμάτου που αλλάζει την κατάσταση της ασύρματης ζεύξης ανάλογα με την απόσταση καθώς και την πιθανότητα λαθών στη μετάδοση. Χρησιμοποιώντας αυτά τα απλά μοντέλα που παρέχονται από το TinyViz είναι δυνατή η μοντελοποίηση πολύπλοκων καταστάσεων. Για παράδειγμα είναι δυνατόν αλλάζοντας το ρυθμό των λαθών στη μετάδοση να προσομοιωθούν εμπόδια μεταξύ των αισθητήρων. Επίσης μπορούν να εξομοιωθούν αστοχίες των κόμβων χρησιμοποιώντας προγράμματα που θα θέτουν τυχαία κάποιους κόμβους εκτός λειτουργίας σε τυχαίες χρονικές στιγμές.

3.5.4.1. Μεταγλώττιση και Εκτέλεση TinyViz [12]

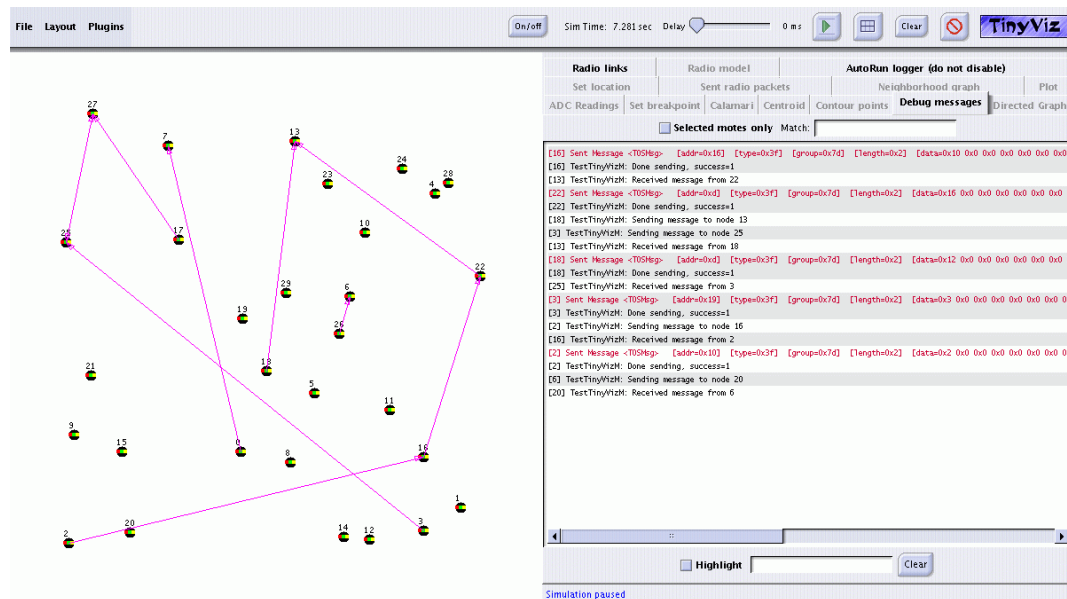
Για να ξεκινήσουμε το tinyViz, μεταφερόμαστε στον κατάλογο της εφαρμογής apps/TestTinyViz, η οποία περιοδικά στέλνει ένα μήνυμα σε ένα τυχαίο γείτονα των motes και εκτελούμε την εντολή make pc.

Για να μεταγλωττίσουμε το TinyViz, μεταφερόμαστε στον tools/java/net/tinyos/sim και πληκτρολογούμε make. Αυτό θα χτίσει το πρόγραμμα TinyOS tinyviz.jar, ένα αυτόνομο Java JAR αρχείο με το οποίο μπορούμε να εκτελέσουμε το tinyviz script, που βρέθηκε σε αυτόν τον κατάλογο. Τοποθετούμε αυτό το script στο PATH και θα είμαστε σε θέση να τρέχουμε το tinyViz απευθείας από την γραμμή εντολών.

Ξεκινούμε το TinyViz με την TestTinyViz εφαρμογή πληκτρολογώντας τις εντολές:

```
$ export DBG=usr1
```

```
$ tinyviz -run build/pc/main.exe 30
```



Σχήμα 3.5.4.1.1: TinyViz [12]

3.6. PowerTOSSIM [19]:

3.6.1. Προσομοίωση της Κατανάλωσης Ισχύος σε Εφαρμογές Μεγάλης Κλίμακας Δικτύου Αισθητήρων [19]

Μια σειρά από περιβάλλοντα προσομοίωσης έχουν αναπτυχθεί που παρέχουν διαφορετικούς βαθμούς επεκτασιμότητας, ρεαλισμό και λεπτομέρεια για την κατανόηση της συμπεριφοράς των δικτύων αισθητήρων. Πριν το PowerTOSSIM, κανένα από αυτά τα εργαλεία δεν είχε αντιμετωπίσει μια από τις πιο σημαντικές πτυχές του σχεδιασμού εφαρμογής αισθητήρα, εκείνη της κατανάλωσης ενέργειας. Ενώ η απλή προσέγγιση της συνολικής χρήσης της ενέργειας μπορεί να προκύψει από τις εκτιμήσεις του κύκλου του κόμβου και τα ποσοστά της επικοινωνίας, οι τεχνικές αυτές συχνά αποτυγχάνουν να συλλάβουν τις λεπτομερή, χαμηλού επιπέδου ενεργειακές απαιτήσεις των CPU, του radio, των αισθητήρων, και των άλλων περιφερειακών συσκευών.

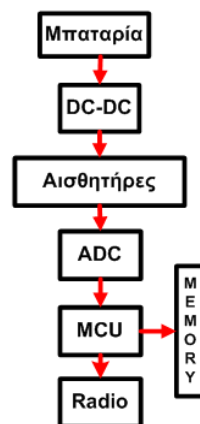
Η κατανόηση της κατανάλωσης ενέργειας είναι ιδιαίτερα κρίσιμη για τα δίκτυα αισθητήρων που λειτουργούν σε περιορισμένα αποθέματα ενέργειας, όπως μπαταρίες ή ηλιακές κυψέλες. Οι σχεδιαστές δικτύου αισθητήρων πρέπει να μπορούν να λαμβάνουν ακριβή και αξιόπιστα στοιχεία για την κατανάλωση ηλεκτρικής ενέργειας για να συντονίσουν τις αιτήσεις τους πριν από την ανάπτυξη σε πραγματικό περιβάλλον. Εκτός από την συνολική κατανάλωση ηλεκτρικής ενέργειας στην πάροδο του χρόνου, ο τρόπος διεξαγωγής του φορτίου ηλεκτρικής ενέργειας είναι σημαντικό να λαμβάνεται υπόψη, καθώς αυτό επηρεάζει την ικανότητα της πηγής ενέργειας για να εξασφαλίσει επαρκή ενέργεια με το χρόνο.

Το PowerTOSSIM, είναι ένα εξελικτικό περιβάλλον προσομοίωσης για ασύρματα δίκτυα αισθητήρων που παρέχει μια ακριβή εκτίμηση της κατανάλωσης ενέργειας ανά κόμβο. Αποτελεί μια επέκταση του TOSSIM, ένα event-driven περιβάλλον προσομοίωσης για TinyOS εφαρμογές. Στο PowerTOSSIM, τα components του TinyOS που αντιστοιχούν σε συγκεκριμένα περιφερειακά υλικού (όπως το radio, EEPROM, τα LED, και ούτω καθεξής) είναι συσκευές για να λάβουν ένα trace της δραστηριότητας κάθε συσκευής κατά την εκτέλεση της προσομοίωσης. Το PowerTOSSIM χρησιμοποιεί μια καινοτόμο τεχνική κώδικα μετασχηματισμού για να

υπολογίσει τον αριθμός των κύκλων CPU που εκτελείται από κάθε κόμβο, εξαλείφοντας την ανάγκη για δαπανηρές προσομοιώσεις σε επίπεδο εντολών των κόμβων αισθητήρων. Το PowerTOSSIM περιλαμβάνει ένα λεπτομερές μοντέλο της ενεργειακής κατανάλωσης υλικού με βάση τις διάφορες πλατφόρμες. Μέσω των οργάνων μέτρησης των πραγματικών κόμβων αισθητήρων, έχει αποδειχτεί ότι το PowerTOSSIM παρέχει ακριβή εκτίμηση της κατανάλωσης ενέργειας για ένα ευρύ φάσμα εφαρμογών και πινάκων για την υποστήριξη πολύ μεγάλων προσομοιώσεων.

3.6.2. Συσκευές οι οποίες καταναλώνουν ενέργεια

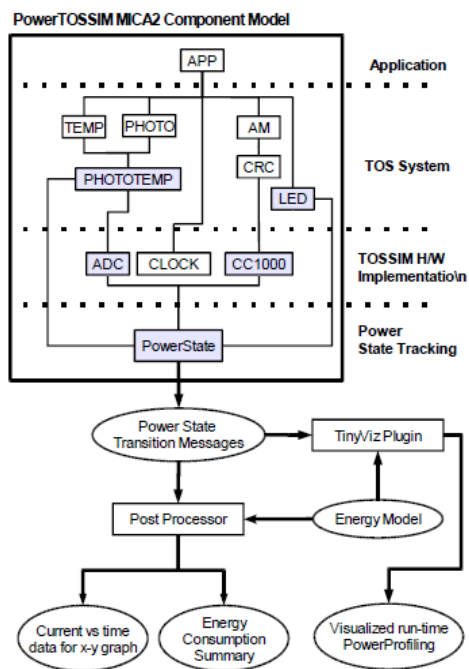
- CPU
- Radio
- Sensor devices
- LEDs
- ADC
- EEPROM



Σχήμα 3.6.2.1: Συσκευές οι οποίες καταναλώνουν ενέργεια

3.6.3. Η Αρχιτεκτονική του PowerTOSSIM [19]

Ο PowerTOSSIM παρακολουθεί την ενεργειακή κατάσταση κάθε συστατικού (component) του υλικού των προσομοιωμένων motes, παράγοντας συγκεκριμένα μηνύματα μεταβάσεων από μία κατάσταση σε μία άλλη, τα οποία καταγράφονται κατά την εκτέλεση της προσομοίωσης. Αυτό επιτυγχάνεται με την τροποποίηση των εικονικών συστατικών υλικού του TOSSIM μέσω κλήσεων σε ένα νέο συστατικό, το PowerState, το οποίο παρακολουθεί την ενεργειακή κατάσταση του κάθε mote και την καταγράφει σε ένα αρχείο κατά τη διάρκεια του τρεξίματος. Η εκτίμηση της χρήσης του CPU είναι κάπως πιο πολύπλοκη : αφού ο PowerTOSSIM τρέχει το λογισμικό του mote σε ένα τοπικό πρόγραμμα στη host μηχανή, δεν παρέχει πληροφορίες για το χρόνο που ένα mote χρησιμοποιεί τη CPU. Έτσι, η μοντελοποίηση της επιτυγχάνεται με χαρτογράφηση των βασικών κομματιών (blocks) της προσομοίωσης σε μετρήσεις κύκλων στο αντίστοιχο πρόγραμμα του mote. Το πιο κάτω σχήμα δείχνει την αρχιτεκτονική του PowerTOSSIM.



Σχήμα 3.6.3.1: Αρχιτεκτονική του PowerTOSSIM: Τα components της εξομοίωσης (RF, sensors, LEDs, κτλ) καλούν το module PowerState και αυτό εκπέμπει μηνύματα αλλαγής ενεργειακής κατάστασης για το καθένα. Αυτά τα μηνύματα μπορούν να συνδυαστούν με ένα ενεργειακό μοντέλο για την παραγωγή λεπτομερών δεδομένων κατανάλωσης ενέργειας ή οπτικοποιήσεων [19]

Τα δεδομένα της ενεργειακής κατάστασης που δημιουργεί ο PowerTOSSIM μπορούν να συνδυαστούν με ένα ενεργειακό μοντέλο, για να αποφασιστεί η ανά mote και ανά component ενεργειακή κατανάλωση. Αυτή η επεξεργασία μπορεί να πραγματοποιηθεί χρησιμοποιώντας εργαλεία για να αποκτηθούν λεπτομερή traces της κατανάλωσης ενέργειας του κάθε συστατικού του κάθε mote ή να περαστεί στο πρόγραμμα οπτικοποίησης TinyViz για να φανεί η κατανάλωση ενέργειας σε πραγματικό χρόνο.

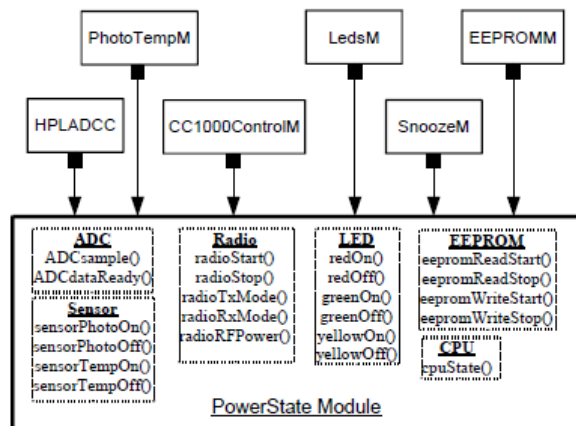
- **PowerTOSSIM Αποδοτικότητα:** Ένα από τα σημαντικά χαρακτηριστικά που ο PowerTOSSIM κληρονομεί από τον TOSSIM είναι η πολύ υψηλή αποδοτικότητά του στην εξομοίωση μεγάλων δικτύων που κλιμακώνονται σε χιλιάδες κόμβων. Για τη διατήρηση αυτής είναι σημαντικό να αποφεύγονται υψηλά overheads στην ίδια προσομοίωση. Ομοίως, με την εκτέλεση της προσομοίωσης ως τοπικό πρόγραμμα αποφεύγουμε το overhead της προσομοίωσης σε επίπεδο εντολής.

- **PowerTOSSIM Ελαστικότητα:** είναι επίσης σημαντικό να παρέχεται υψηλός βαθμός ελαστικότητας για την καταμέτρηση και την μοντελοποίηση της ενεργειακής κατάστασης του mote. Δεν υποστηρίζεται η χρήση μιας πλατφόρμας, αφού συνέχεια χρησιμοποιούνται νέα σχέδια. Με την αποσυνδεδεμένη σχεδίαση είναι δυνατό να εκτιμηθεί η ενεργειακή αποδοτικότητα πιθανών σχεδιάσεων υλικού, μόνο βάζοντας ένα νέο ενεργειακό μοντέλο στα εργαλεία ανάλυσης. Η προσομοίωση δε χρειάζεται να εκτελεστεί ξανά.

- **PowerTOSSIM Η μονάδα (module) PowerState**

Για την αποφυγή της ύπαρξης διασκορπισμένου κώδικα για την παρακολούθηση της ενέργειας σε ολόκληρο τον προσομοίωση, δημιουργήθηκε ένα μόνο module του TinyOS, το PowerState, το οποίο καλούν τα υπόλοιπα συστατικά του TinyOS για να καταγράψουν τις αλλαγές στην ενεργειακή τους κατάσταση. Το πιο κάτω σχήμα απεικονίζει το module PowerState και τις συνδέσεις του με άλλα συστατικά: αποτελείται από μία μοναδική διεπαφή (interface) με μία εντολή για κάθε πιθανή ενεργειακή μετάβαση. Κάθε συνάρτηση ελέγχει αν είναι ενεργοποιημένο το power profiling και, αν ναι, εκπέμπει ένα μήνυμα καταγραφής (log message) που περιέχει τον αριθμό του mote, τη συγκεκριμένη ενεργειακή μετάβαση και το χρόνο της προσομοίωσης. Ένα απόσπασμα αυτού δίνεται παρακάτω:

```
0: POWER: Mote 0 LED_STATE RED_OFF at 18677335
0: POWER: Mote 0 LED_STATE YELLOW_OFF at 18677335
0: POWER: Mote 0 ADC_SAMPLE RSSI_PORT at 18990479
0: POWER: Mote 0 ADC_DATA_READY at 18990679
0: POWER: Mote 0 RADIO_STATE TX at 18993551
```



Σχήμα 3.6.3.2: Η διασύνδεση (wiring) του PowerState module με τα υπόλοιπα components [19]

Απαιτούνται πολύ λίγες τροποποιήσεις στον αυθεντικό κώδικα του TOSSIM για την παρακολούθηση της ενεργειακής κατάστασης των διαφόρων συσκευών. Συνολικά, 46 γραμμές κώδικα άλλαξαν ή προστέθηκαν σε υπάρχοντα συστατικά του TinyOS. Αυτές οι γραμμές ήταν απλά διασυνδέσεις και κλήσεις συναρτήσεων σε εντολές του module PowerState. Στον Πίνακα δίνεται μια περίληψη των γραμμών που επηρεάστηκαν σε κάθε component. Ολόκληρο το PowerState αποτελείται από περίπου 400 γραμμές για τη διαμόρφωση και εκτύπωση μηνυμάτων κατάστασης.

Category	TinyOS Component	Message generated	Files involved	Number of lines added for state tracking
System	LedsC	LED_STATE	LedsC.nc, LedsM.nc	9
	Main	CPU_CYCLES	Main.nc	2
Platform-specific	ADCC	ADC	ADCC.nc, HPLADCC.nc	6
	CC1000RadioC	RADIO_STATE	CC1000RadioC, CC1000ControlM	8
	EEPROM	EEPROM	EEPROM.nc, EEPROMM.nc	7
	SnoozeC	CPU_STATE	SnoozeC.nc, SnoozeM.nc	8
Sensorboard	PhotoTemp	SENSOR_STATE	PhotoTemp.nc, PhotoTempM.nc	6
Total				46

Πίνακας 3.6.3.1: Ο αριθμός των γραμμών που τροποποιήθηκαν σε διάφορα TinyOS components [19]

3.6.4. Αποτίμηση του PowerTOSSIM

Όπως αναφέρθηκε, για την αξιολόγηση του προσομοιωτή PowerTOSSIM, εφαρμόστηκε σε έναν αριθμό εφαρμογών του TinyOS και έπειτα συγκρίθηκαν τα αποτελέσματα με δεδομένα που προέκυψαν από πραγματικές μετρήσεις. Ο Πίνακας δείχνει ότι ο PowerTOSSIM επιτυγχάνει πολύ καλή ακρίβεια, με μέσο σφάλμα 4.7% και μέγιστο 13% σε σχέση με το πραγματικό mote. Ένα μέρος της διαφοράς ανάμεσα στην προσποιούμενη και μετρούμενη ενέργεια μπορεί να αποδοθεί σε διακυμάνσεις τάσης, θόρυβο και σφάλματα στρογγυλοποίησης στο στήσιμο του πειράματος. Άλλες διαφορές μπορεί να οφείλονται σε ανακρίβειες στο ενεργειακό μοντέλο ή στη μέτρηση των κύκλων της CPU.

Σύμφωνα, λοιπόν με τις έρευνες που έχουν γίνει το PowerTOSSIM θεωρείται ένα πολύ αξιόπιστο εργαλείο για την μέτρηση της ενέργειας και για το λόγο αυτό το έχουμε προτιμήσει για την αξιολόγηση του αλγορίθμου μας.

3.7. Μέτρηση Ενέργειας [20]

Αρχικά, θα επεξηγηθεί ο τρόπος με τον οποίο λαμβάνεται ένα trace αρχείο μέσα από τη λειτουργία debugging του PowerTOSSIM, και στη συνέχεια ο τρόπος να μεταφρασθεί αυτό το αρχείο.

- **Παραγωγή trace file – λειτουργία debugging**

Η λειτουργία debugging του TinyOS δίνει τη δυνατότητα στο χρήστη να ελέγχει την ορθή λειτουργία μιας εφαρμογής, παράγοντας κάποιο αρχείο εξόδου (trace file), στο οποίο ο TOSSIM δίνει πληροφορίες π.χ. για τη λειτουργία του ADC, για την αποστολή κάποιου μηνύματος, για το άναμμα/σβήσιμο κάποιου LED, κτλ. Στα πλαίσια της ανάπτυξης του προσομοιωτή ενέργειας PowerTOSSIM, δόθηκε η δυνατότητα παρακολούθησης της ενεργειακής κατάστασης του mote μέσω τέτοιων αρχείων.

Στη γραμμή εντολών του Cygwin, η εντολή που θα συνδέσει το debugging (DBG) με την ενεργειακή παρακολούθηση είναι η εξής : **\$ export DBG=power**

Έπειτα ακολουθούμε τα πιο κάτω βήματα:

1. Μεταγλώττιση της TinyDB εφαρμογής

- Μεταφερόμαστε στο κατάλογο `/opt/tinyos-1.x/apps/TinyDBApp`
- **\$ export DBG=power**
- Εκτελούμε την εντολή **\$ make -k -f MakePC pc**

2. Run TinyDBApp client

- Έπειτα, τρέχουμε το εκτελέσιμο που παράχθηκε με την ακόλουθη εντολή, με την οποία διευκρινίζεται το διάστημα κατά το οποίο τα motes ξεκινούν(επιλογή-b),το χρόνο προσομοίωσης(-t), το radio μοντέλο που χρησιμοποιείται (-r), input file για το lossy μοντέλο (-rf), DBG (-p), ο αριθμός των mote (14) και το όνομα του trace αρχείου (mintGdiMica2.trace).

```
$ ./build/pc/main.exe -b=10 -seed=10 -t=1000 -r=lossy -rf= 7x2-20.nss -p 14 > mintGdi.trace
```

3. Χρήση TinyViz plugin

Το plugin TinyViz μας επιτρέπει να δούμε την κατάσταση ισχύος της εφαρμογής μας καθώς τρέχει εκτελώντας τις πιο κάτω εντολές

```
$ cd /opt/tinyos-1.x/tools/java/net/tinyos/sim/
```

```
$ java net/tinyos.sim.TinyViz
```

4. Μεταγλώττιση του TinyDB

a. Μεταφερόμαστε στο κατάλογο /opt/tinyos-1.x/tools/java/net/tinyos/tinydb/

b. **\$ make clean**

c. **\$ make -f MakePC**

4. Εκτέλεση TinyDB σε προσομοιωτή

a. Μεταφερόμαστε στο κατάλογο opt/tinyos-1.x/tools/java/

b. **\$ java net/tinyos.tinydb.TinyDBMain -sim**

- Το Query επιτρέπει στο χρήστη να καθορίσει συνολικά (AVG, MIN και MAX) και μη συνολικά όπως SQL queries είτε γραφικά ή με το χέρι. Το ερώτημα είναι κατασκευασμένο και συνεπώς προωθείται συνέχεια σε ένα τροποποιημένο παράδειγμα TinyDB. Για την μελέτη αυτή το query που χρησιμοποιήθηκε είναι:

```
SELECT TOP 1 cluster, AVG (temp)
```

```
FROM sensors
```

```
GROUP BY cluster
```

Με την παραπάνω διαδικασία γίνεται η παραγωγή του trace αρχείου που θα περιέχει όλες τις αλλαγές της ενεργειακής κατάστασης του mote για την εκάστοτε εφαρμογή.

Ένα παράδειγμα τέτοιου αρχείου για την TinyOS εφαρμογή TinyDBApp για τον αλγόριθμο MINT δίνεται παρακάτω:

```
1: POWER: Mote 1 RADIO_STATE TX at 1917534
```

```
0: POWER: Mote 0 RADIO_STATE RX at 1926450
```

```
0: POWER: Mote 0 RADIO_STATE TX at 1975953
```

Στο κομμάτι αυτό βλέπουμε ότι ένα αρχείο trace μας δίνει πληροφορίες για τον αριθμό του mote, για τη συσκευή της οποίας η κατάσταση αλλάζει, καθώς και τη χρονική στιγμή της προσομοίωσης που γίνεται αυτό. Στόχος είναι η μετατροπή αυτού του

αρχείου κειμένου σε μια γραφική παράσταση που να δίνει την κατανάλωση ενέργειας / ρεύματος ως προς το χρόνο προσομοίωσης.

Έπειτα, τρέχουμε `postprocess.py` πάνω στο resulting trace. Το πρόγραμμα της python με την πιο κάτω εντολή επεξεργάζεται το trace αρχείο για να πάρουμε πληροφορίες για την κατανάλωση ενέργειας στις διάφορες συσκευές ανά χρονική στιγμή της προσομοίωσης:

```
$TOSROOT/tools/scripts/PowerTOSSIM/postprocess.py --detail --sb=1 --em  
$TOSROOT/tools/scripts/PowerTOSSIM /telos_energy_model.txt mintGdi.trace >  
mintTotalEnergy.txt
```

Η παράμετρος `--detail` δημιουργεί ένα αρχείο για κάθε mote και δίνει για κάθε χρονική στιγμή της προσομοίωσης τη συνολική ενέργεια και την ενέργεια που καταναλώνεται για κάθε συσκευή.

Η παράμετρος `--sb` καθορίζει το αν θα θεωρήσει ότι τα motes έχουν ενσωματωμένο sensor board. Η παράμετρος `--em` καθορίζει το ενεργειακό μοντέλο.

Εάν τρέξουμε το `postprocess.py` με την παράμετρο `--help` μπορούμε να δούμε τι παραμέτρους επιτρέπεται να χρησιμοποιήσουμε.

Εδώ βλέπουμε το output που παίρνουμε για το trace της εφαρμογής TinyDBApp για τον αλγόριθμο MINT:

```
Mote 0, cpu total: 84241.904222  
Mote 0, radio total: 143704.278377  
Mote 0, adc total: 0.000000  
Mote 0, leds total: 67727.042587  
Mote 0, sensor total: 14074.313296  
Mote 0, eeprom total: 0.000000  
Mote 0, cpu_cycle total: 0.000000  
Mote 0, Total energy: 309747.538483
```

Πιο κάτω βλέπουμε ένα απόσπασμα από το output που δίνει η παράμετρος `--detail` για το mote 0,

# time	total	cpu	radio	adc	leds	sensor	eeprom
1968940.000000	0.744500	0.054500	0.000000	0.000000	0.000000	0.690000	0.000000
2120740.000000	2.944500	0.054500	0.000000	0.000000	2.200000	0.690000	0.000000
2179940.000000	2.944500	0.054500	0.000000	0.000000	2.200000	0.690000	0.000000

Έπειτα περνούμε τα αρχεία των motes που δημιουργούνται από την παράμετρο --detail και τα επεξεργαζόμαστε με ένα κώδικα c (totalEnergy.c) για να ομαδοποιήσουμε την ενέργεια ανά χρονική στιγμή. Ο κώδικας C με την κλήση ενός perl script (totalEnergy.pl) παίρνει την συνολική ενέργεια και τη χρονική στιγμή (ms) και ομαδοποιεί τα δεδομένα με βάση τον αριθμό των γραμμών του αρχείου και των συνολικών αναγνώσεων του αισθητήρα και διαιρεί με 1000 για να δώσει την ενέργεια σε joules. Με το πέρας της εκτέλεσης του script γίνεται ομαδοποίηση της ενέργειας ανά χρονική στιγμή.

Τέλος παίρνουμε τα δεδομένα που παράγονται από την εκτέλεση του totalEnergy.c από το αρχείο data.txt και με τη χρήση του gnuplot δημιουργούμε τις γραφικές παραστάσεις.

Κεφάλαιο 4

Πειραματική Μεθοδολογία Αξιολόγησης του MINT

- 4.1. Εισαγωγή
 - 4.2. Πειραματική Μεθοδολογία
 - 4.3. Σύνολα Δεδομένων
 - 4.4. Συσκευή Αισθητήρα (Sensing Device)
 - 4.5. Τοπολογίες Multi-hop
 - 4.6. Πρωτόκολλο Επικοινωνίας
 - 4.7. Σύνταξη Επερώτησης
-

4.1. Εισαγωγή

Σε αυτή την ενότητα θα περιγράψουμε την πειραματική μεθοδολογία που ακολουθήσαμε, η οποία περιλαμβάνει πολλαπλές προσομοιώσεις πάνω σε πραγματικά σύνολα δεδομένων που προέρχονται, από το Τμήμα Επιστημών της Ατμόσφαιρας του Πανεπιστημίου της Ουάσιγκτον, από το Intel Research Berkeley και UC Berkeley.

Αξιολογήσαμε τον αλγόριθμο MINT συγκρίνοντας τον με τρεις άλλους αλγόριθμους για να αποδείξουμε ότι ο αλγόριθμος αυτός μειώνει σημαντικά τα ποσά κατανάλωσης ενέργειας. Οι αλγόριθμοι που χρησιμοποιήθηκαν για σύγκριση είναι ο TAG, ο TINA και ο INT οι οποίοι θα περιγραφούν στη συνέχεια. [4,5]

4.2. Πειραματική Μεθοδολογία (Experimental Methodology)

Αλγόριθμοι που χρησιμοποιήθηκαν:

1. **TAG:** βασίζεται στη συνάθροιση μέσα στο δίκτυο, για την ελαχιστοποίηση της επικοινωνίας. Κάθε κόμβος συλλέγει τις τοπικές συναθροίσεις από τα παιδιά του και συνδυάζει τα συγκεντρωτικά στοιχεία μαζί με τις δικές του αναγνώσεις αισθητήρα, και τα προωθεί στον πατέρα του.
2. **TINA:** αναπτύσσει ομαδοποίηση δικτύου συνεχόμενων τιμών εάν αυτές δεν αλλάζουν. Δηλαδή στέλλεται μια ανάγνωση από έναν αισθητήρα μόνο εάν η ανάγνωση διαφέρει από την τελευταία καταγραφή.
3. **Διαφορές μεταξύ MINT και INT:** Οι διαφορές των δύο αλγορίθμων συνοψίζονται ως εξής: 1) ο MINT εκμεταλλεύεται μια χρονική συνέπεια, προκειμένου να σταματήσει τα update view τα οποία δεν αλλάζουν, μεταξύ των διαδοχικών περιπτώσεων χρόνου, ενώ ο INT έχει να στείλει ξανά αυτές τις ενημερωμένες εκδόσεις, επειδή δεν ανήκουν σε αυτό το χώρο. 2) Στον MINT, έχουμε μόνο να ενημερώσουμε το V_i χρησιμοποιώντας τον αλγόριθμο 3 (σε $O(|V_i'|)$ time), ενώ στον INT έχουμε την κατασκευή από την αρχή κάθε φορά, σε χρόνο $O(|V_i|)$, χρησιμοποιώντας τον αλγόριθμο 2. 3) ο INT έχει το πλεονέκτημα ότι δεν απαιτείται καμία πρόσθετη αποθήκευση έτσι είναι πιο κατάλληλος για τους αισθητήρες όπου η αποθήκευση κοστίζει.

Είναι σημαντικό να αναφέρουμε ότι ο αλγόριθμος TAG είναι ήδη υλοποιημένος ως μέρος του πλαισίου TinyDB και χρησιμοποιείται ως βάση για σύγκριση. Οι υπόλοιποι αλγόριθμοι, MINT, TINA και INT έχουν υλοποιηθεί σε γλώσσα nesC [29], η οποία είναι η γλώσσα προγραμματισμού για το λειτουργικό σύστημα TinyOS [30], και έχει παρόμοια σύνταξη με τη γλώσσα C. Η nesC [29] είναι βασισμένη στην αρχή της σύνδεσης (wiring) μεταξύ των συστατικών για τη δημιουργία μιας εφαρμογής και προορίζεται κυρίως για τα ενσωματωμένα συστήματα, όπως τα δίκτυα αισθητήρων.

Το TinyOS είναι ένα νέο είδος ανοικτού κώδικα λειτουργικό σύστημα που αποτελεί ένα από τα πρώτα περιβάλλοντα λογισμικού που έχουν σχεδιαστεί για να ικανοποιήσουν τις απαιτήσεις των κατανεμημένων ενσωματωμένων συστημάτων που έχουν περιορισμένους πόρους. Σχεδιάστηκε στο Πανεπιστήμιο του Berkeley και από τότε έχει γίνει ένα από τα πιο δημοφιλή λειτουργικά συστήματα για ασύρματα δίκτυα αισθητήρων. Το λειτουργικό αυτό, είναι βασισμένο στην αρχιτεκτονική των συστατικών, η οποία κάνει δυνατή τη γρήγορη υλοποίηση των εφαρμογών, ενώ ταυτόχρονα ελαχιστοποιεί το μέγεθος του κώδικα που απαιτείται καθώς και την κατανάλωση ενέργειας στους αισθητήρες.

Για τη διεξαγωγή των προσομοιώσεων, χρησιμοποιήθηκε ο TOSSIM [18] που είναι ένας προσομοιωτής διακριτών συμβάντων (discrete event) ειδικά σχεδιασμένος για το TinyOS και έχει την δυνατότητα να αντιλαμβάνεται και να αναπαριστά την συμπεριφορά και τις αλληλεπιδράσεις σε δίκτυα χιλιάδων κόμβων (moten), μαζί με το TinyViz που είναι το εργαλείο απεικόνισης του TOSSIM. Επιπλέον, για τον υπολογισμό της κατανάλωσης ενέργειας ανά κόμβο έγινε χρήση του PowerTOSSIM που είναι μια επέκταση του προσομοιωτή. Το PowerTOSSIM, είναι ένα εξελικτικό περιβάλλον προσομοίωσης για ασύρματα δίκτυα αισθητήρων που παρέχει μια ακριβή εκτίμηση της κατανάλωσης ενέργειας ανά κόμβο. Μέσω των οργάνων μέτρησης των πραγματικών κόμβων αισθητήρων, έχει αποδειχτεί ότι το PowerTOSSIM παρέχει ακριβή εκτίμηση της κατανάλωσης ενέργειας για ένα ευρύ φάσμα εφαρμογών και πινάκων για την υποστήριξη πολύ μεγάλων προσομοιώσεων. Το PowerTOSSIM έχει δείξει να είναι περισσότερο από 90% ακριβές. [19] Ειδικότερα, οι συγγραφείς του [19], μετρούν την ενέργεια για την εκτέλεση των παραδειγμάτων επίδειξης με TinyOS τόσο χρησιμοποιώντας PowerTOSSIM όσο και σε πραγματικούς αισθητήρες (που μετράται με πολύμετρο). Οι συγγραφείς δείχνουν ότι το PowerTOSSIM απέδωσε κατά μέσο όρο σφάλμα μόνο 4,7%. Παρόμοιες παρατηρήσεις ισχύουν και για πιο σύνθετες εφαρμογές όπως TinyDB και Surge που είχαν δείξει ότι έχει κατά μέσο όρο λάθος 9,5%. Ως εκ τούτου, αναμένουμε ότι η ακρίβεια θα παραμείνει στα ίδια υψηλά επίπεδα και με το μοντέλο ηλεκτρικής ενέργειας TelosB.

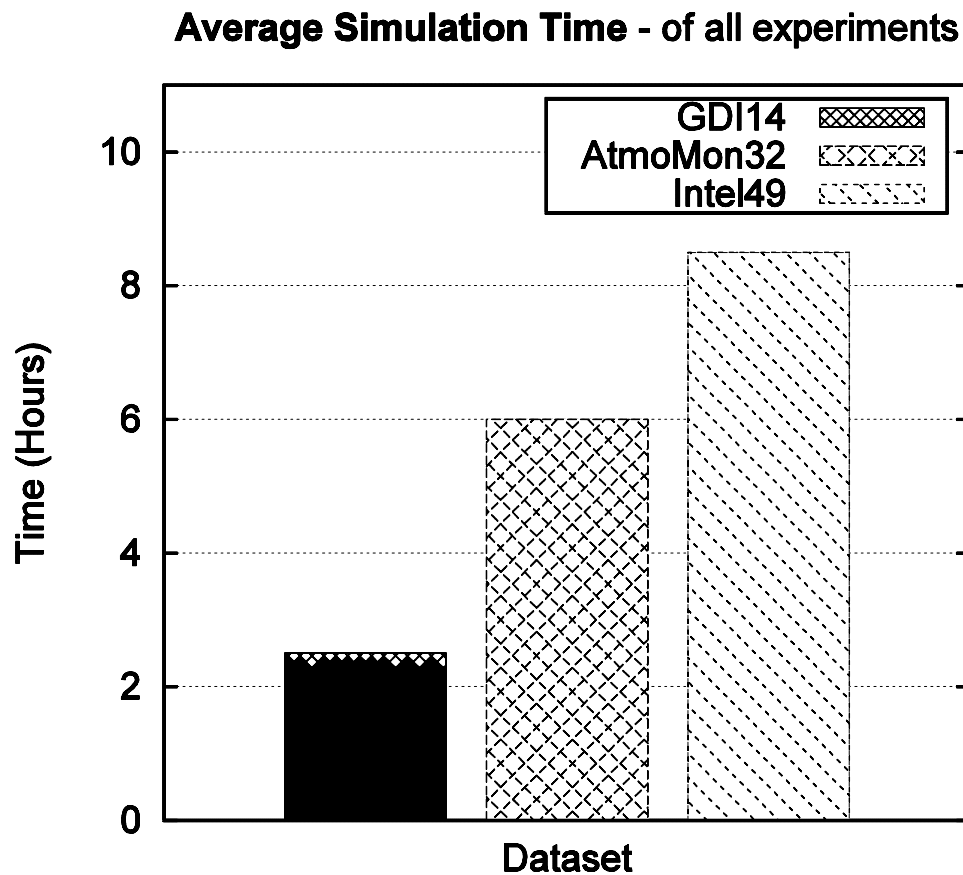
Το Σχήμα 4.2.1, δείχνει ένα παράδειγμα της διαδικασίας που χρησιμοποιήθηκε για για τη συλλογή στατιστικών ενέργειας για τα πειράματά μας. Στο πρώτο στάδιο, δημιουργούμε ένα lossy μοντέλο για την τοπολογία και το αποθηκεύσαμε σε ένα .nss αρχείο. Στη συνέχεια, εκτελούμε το πείραμα για καθορισμένο χρονικό διάστημα (π.χ., 1000s (-t = 1000)) και συλλέγουμε τα ενεργειακά στατιστικά σε ένα .trace αρχείο. Ένα παράδειγμα του PowerTOSSIM trace αρχείου απεικονίζεται στην εικόνα 4.2.2. Τέλος, επεξεργαζόμαστε τα ενεργειακά στατιστικά στοιχεία, προκειμένου να παράγουμε τα αποτελέσματα της ενέργειας για κάθε αισθητήρα.

Βήμα 1: Δημιουργία μοντέλου lossy radio <pre>java net.tinyos.sim.LossyBuilder -d 7 2 -s 20 -o 7x2_20.nss</pre> Βήμα 2: Εκτέλεση πειράματος με το TOSSIM και συλλογή ενεργειακών στατιστικών <pre>DBG=power ./build/pc/main.exe -b=10 -seed=10 -t=1000 -r=lossy -rf= 7x2-20.nss -p 14 > mintGdi.trace</pre> Βήμα 3: Παραγωγή αποτελεσμάτων ενέργειας για κάθε αισθητήρα <pre>\$TOSROOT/tools/scripts/PowerTOSSIM/postprocess.py --detail --sb=1 --em \$TOSROOT/tools/scripts/PowerTOSSIM /telos_energy_model.txt mintGdi.trace > mintTotalEnergy.txt</pre>	<pre>... 38: POWER: Mote 38 ADC ON at 2741220 38: POWER: Mote 38 ADC ON at 2741220 38: POWER: Mote 38 ADC ON at 2741220 38: POWER: Mote 38 RADIO_STATE ON at 2741220 8: POWER: Mote 8 RADIO_STATE TX at 2791414 38: POWER: Mote 38 RADIO_STATE RX at 2842220 8: POWER: Mote 8 RADIO_STATE RX at 2850614 8: POWER: Mote 8 RADIO_STATE RX at 2851464 8: POWER: Mote 8 RADIO_STATE RX at 2852264 8: POWER: Mote 8 RADIO_STATE RX at 2853064 8: POWER: Mote 8 RADIO_STATE RX at 2853864 8: POWER: Mote 8 RADIO_STATE RX at 2853864 38: POWER: Mote 38 RADIO_STATE TX at 2862733 38: POWER: Mote 38 RADIO_STATE TX at 2863533 ...</pre>
	Σχήμα 4.2.2 : Trace αρχείου του PowerTOSSIM

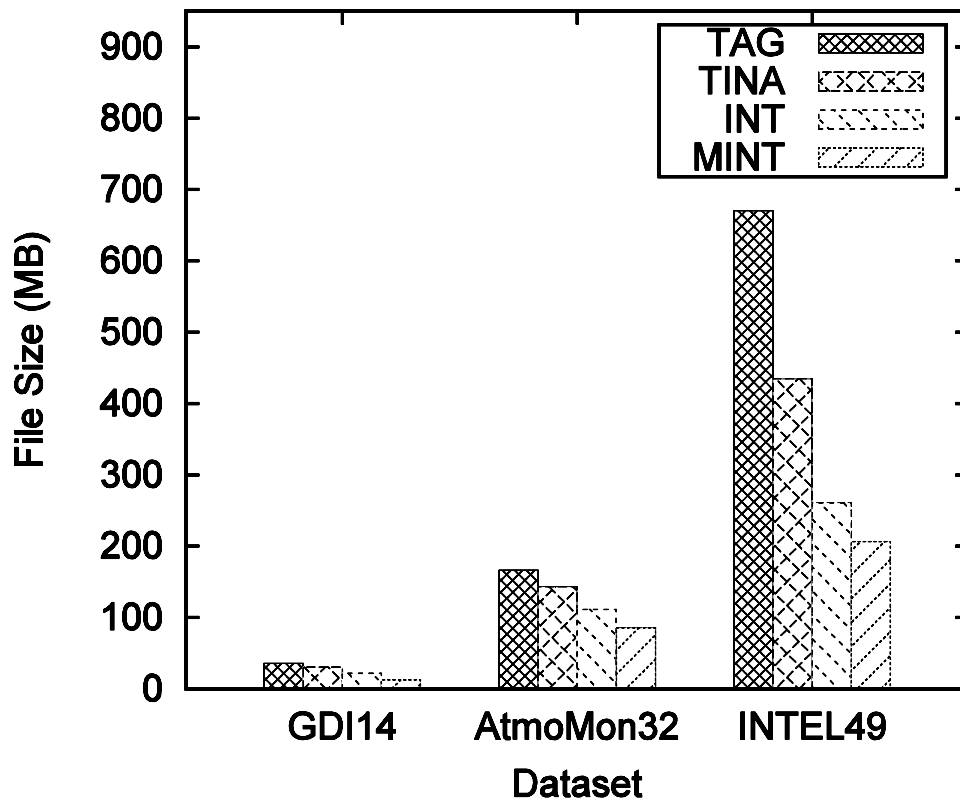
Σχήμα 4.2.1 : Παράδειγμα εκτέλεσης σεναρίου για τον αλγόριθμο MINT πάνω στο σύνολο δεδομένων GDI

Τα πειράματα των προσομοιώσεων μας έγιναν σε ένα Lenovo Thinkpad T61p PC με επεξεργαστή Intel Core 2 Duo CPU που τρέχει στα 2,4 GHz και έχει 2,0 GB μνήμη RAM. Για να μπορέσουν να συλλεχθούν ρεαλιστικά αποτελέσματα σε ένα μεγάλο χρονικό διάστημα, συλλέγουμε στατιστικά στοιχεία για 1000 χρονικές στιγμές της προσομοίωσης, σε κάθε πείραμα. Για να αυξήσουμε την αξιοπιστία των μετρήσεων επαναλαμβάνουμε το κάθε πείραμα 5 φορές και πήραμε το μέσο όρο κατανάλωση

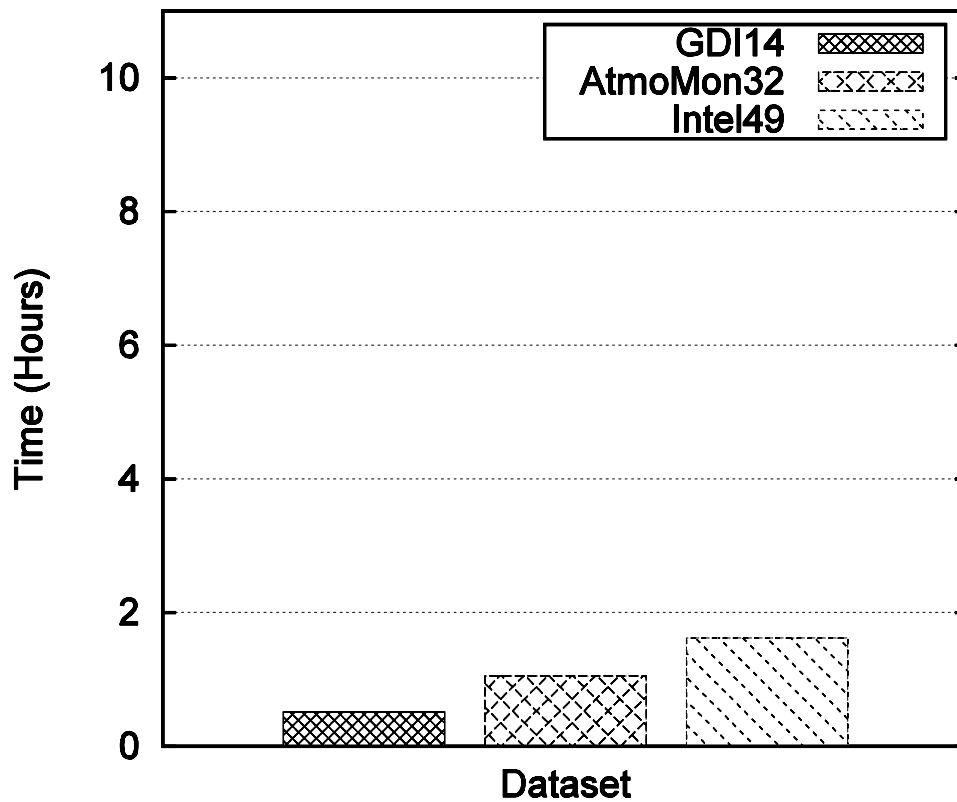
ενέργειας για κάθε γραφική. Η παραπάνω διαδικασία, οδήγησε σε μεγάλης χρονικής διάρκειας προσομοιώσεων για κάθε τύπο γραφικών και ο χρόνος που απαιτείται για την προσομοίωση ενός πειράματος για να παραχθεί το trace αρχείο, κυμάνθηκε από 2,5 έως 8,5 ώρες. Επιπλέον, το μέγεθος του trace αρχείο που παράγεται κυμαίνεται από 20-250MB, ανάλογα με το σύνολο δεδομένων. Αυτό οδήγησε σε μια πρόσθετη επιβάρυνση του χρόνου από 30-100 λεπτά για την επεξεργασία του κάθε αρχείου trace, προκειμένου να συλλεχθούν τα αποτελέσματα της ενέργειας. Τα στατιστικά των προσομοιώσεων παρουσιάζονται στις πιο κάτω γραφικές παραστάσεις.



Average Trace File Size - for each algorithm per simulat



Average Processing Time - for each trace file



4.3. Σύνολα Δεδομένων:

Οι προσομοιώσεις για τους πιο πάνω αλγόριθμους διεξάχθηκαν για τρία σύνολα δεδομένων: Washington State Climate (AtmoMon 32), Great Duck Island (GDI14 2002), Intel Research Berkeley (Intel49).

1. Great Duck Island (GDI 14): Αυτό είναι ένα πραγματικό σύνολο δεδομένων από το σχέδιο παρακολούθησης οικοτόπων που αναπτύχθηκε το 2002 για τη Great Duck Island που είναι 15 χιλιόμετρα στα ανοικτά των ακτών της Maine [31], ΗΠΑ. Χρησιμοποιούμε αναγνώσεις από 14 αισθητήρες που έχουν το μεγαλύτερο ποσοστό των τοπικών αναγνώσεων. Το σύνολο δεδομένων GDI περιλαμβάνει αναγνώσεις, όπως: φως, θερμοκρασία, thermopile, θερμίστορ, υγρασία και τάση.

2. Washington State Climate(AtmoMon32): Αυτό είναι ένα πραγματικό σύνολο στοιχείων των ατμοσφαιρικών δεδομένων που συλλέγονται από 32 αισθητήρες στην Ουάσιγκτον και Oregon states, από το Τμήμα Επιστημών της Ατμόσφαιρας στο Πανεπιστήμιο της Ουάσιγκτον [32]. Πιο συγκεκριμένα, κάθε ένας από τους 32 αισθητήρες διατηρεί τη μέση θερμοκρασία και την ταχύτητα ανέμου σε ωριαία βάση για 208 ημέρες μεταξύ Ιουνίου 2003 και Ιουνίου 2004 (δηλαδή, 4.990 στιγμές του χρόνου).

3. Intel Research Berkeley (Intel49): Αυτό είναι ένα πραγματικό σύνολο δεδομένων που συλλέγεται από 58 αισθητήρες που είναι εγκατεστημένοι στις εγκαταστάσεις της Intel Research Berkeley [33] μεταξύ 28 Φεβρουαρίου και 5 Απριλίου, 2004. Οι αισθητήρες που αξιοποιήθηκαν για την ανάπτυξη ήταν εξοπλισμένοι με πλακέτες για τον καιρό (weather boards) και λάμβαναν σε χρονικά διαστήματα πληροφορίες τοπολογίας μαζί με την υγρασία, τη θερμοκρασία, το φως και τιμές τάσης μία φορά κάθε 31 δευτερόλεπτα. Το σύνολο δεδομένων περιλαμβάνει 2,3 εκατομμύρια αναγνώσεις που συλλέγονται από τους αισθητήρες αυτούς. Χρησιμοποιούμε αναγνώσεις από τους 49 αισθητήρες που είχε το μεγαλύτερο ποσοστό των τοπικών αναγνώσεων, δεδομένου ότι ορισμένοι από αυτούς είχαν πολλές τιμές που έλειπαν.

4.4. Συσκευή Αισθητήρα (Sensing Device)

Χρησιμοποιούμε το ενεργειακό μοντέλο TelosB [1] της Crossbow (Σχήμα 4.4.1). Το TelosB είναι μια εξαιρετικά χαμηλής ισχύος ασύρματος αισθητήρας που είναι εξοπλισμένος με έναν πυρήνα 8 MHz MSP430, 1MB εξωτερικής αποθήκευσης flash, και 250Kbps Chipcon CC2420 RF Transceiver που καταναλώνει 23mA όταν λαμβάνει δεδομένα (Rx), 19.5mA όταν μεταδίδει δεδομένα (Tx), 7.8mA όταν είναι ενεργοποιημένο (MCU active) με το radio απενεργοποιημένο και 5.1mA όταν βρίσκεται σε sleep mode. Μέτρηση της απόδοσης είναι η ενέργεια, σε joules και ο χρόνος για την επίλυση του ερωτήματος που απαιτείται σε κάθε ξεχωριστή περίπτωση.

Το TelosB δεν αποτελεί μέρος της ενότητας του PowerTOSSIM, οπότε έπρεπε να επεκταθεί το PowerTOSSIM με την ενσωμάτωση ενός νέου ενεργειακού μοντέλου για την υποστήριξη συσκευών TelosB.

Η εξίσωση για τον υπολογισμό της καταναλισκόμενης ενέργειας είναι :

Ενέργεια(joules) = Volts x Amperes x Seconds.

Για παράδειγμα για την μετάδοση 30 bytes με 1.8 V η ενέργεια είναι : $1.8V \times 23 \times 10^{-3} A \times 30 \times 8\text{bits}/250\text{kbps} = 39\text{mJ}$.



Σχήμα 4.4.1: Crossbow's TelosBMote

4.5. Τοπολογίες Multi-hop

Προκειμένου να δημιουργηθεί μια τοπολογία δικτύου πολλαπλών hops, έχουμε χρησιμοποιήσει το συστατικό Lossy-Builder του TOSSIM. Το LossyBuilder επιτρέπει τη δημιουργία απωλειών (lossy) radio μοντέλων για τοπολογίες που χρησιμοποιούνται σε όλα τα σύνολα δεδομένων. Αυτά τα μοντέλα δημιουργούν απώλειες θέσης στους αισθητήρες σε διάφορες αποστάσεις από τον κόμβο sink και δημιουργούν μια Gaussian

κατανομή πιθανοτήτων αποτυχίας των πακέτων για κάθε απόσταση. Ο TOSSIM στη συνέχεια δημιουργεί ποσοστά απώλειας πακέτων για κάθε ζεύγος αισθητήρων με δειγματοληψία αυτών των διανομών και μετάφραση τους σε ανεξάρτητα ποσοστά σφάλματος. Ένα παράδειγμα εξόδου του LossyBuilder απεικονίζεται στο Σχήμα 4.5.1 όπου δίνεται μια λίστα με μερικά από τα σφάλματα bit για τον αισθητήρα (με id = 0) σε μια τοπολογία 10 κόμβων. Για παράδειγμα, η γραμμή 3 (0 2 0,5) του Σχήμα 4.5.1 αναφέρει ότι το Mote 0 ακούει το Mote 2, με ποσοστό σφάλματος bit 50%. Αυτή η διεργασία επιτρέπει τη δημιουργία Multi-hop τοπολογιών δικτύου που απαιτούνται για το σύνολο των πειραμάτων μας.

```

<mote id> <mote id> <error rate>
...
0 0 0.0
0 1 8.99E-4
0 2 0.5
0 3 0.5
0 4 8.99E-4
0 5 0.012694
0 6 0.5
0 7 0.002147
0 8 0.5
0 9 0.5
0 10 0.00965
...

```

Σχήμα 4.5.1: Έξοδος Lossy-Builder

4.6. Πρωτόκολλο Επικοινωνίας

Το πρωτόκολλο επικοινωνίας βασίζεται στο πανταχού παρών, για τα δίκτυα αισθητήρων πρότυπο IEEE 802.15.4 (η βάση για το ZigBee [35] προδιαγραφές που χρησιμοποιούνται από συσκευές αισθητήρα συμπεριλαμβανομένου και της συσκευής αισθητήρα TelosB). Το ZigBee χρησιμοποιεί το CSMA/CA σύστημα αποφυγής σύγκρουσης, όπου ένας κόμβος χρησιμοποιεί ένα τυχαίο εκθετικό back-off αλγόριθμο που περιμένει για ένα τυχαίο χρονικό διάστημα 0.25-0.5s πριν από την αναμετάδοση.

Το πλαίσιο μηνυμάτων του TinyDB διαρθρώνεται ως εξής [18]: Κάθε μήνυμα σχετίζεται με μια κεφαλίδα TinyDB εφαρμογής 7 Bytes που περιλαμβάνει: το αναγνωριστικό προέλευσης (2B), το αναγνωριστικό ερωτήματος προέλευσης (2B), το

sequence number (2B) και το μετρητή αλμάτων (hop) (1B). Στις υπόλοιπες θέσεις βρίσκεται το ωφέλιμο φορτίο (29B) σύμφωνα με το ερώτημα θα εκτελέσει:

1. Για τη δομή δεδομένων TopkData: διαθέτουμε 1 bit για τον προσδιορισμό αν η τρέχουσα κατάσταση είναι η ίδια με την προηγούμενη κατάσταση, 3 bits για τον προσδιορισμό του αριθμός των πλειάδων στην τρέχουσα κατάσταση και 2B για την τιμή.

2. Για τη δομή δεδομένων TopkRoomData: η οποία χρησιμοποιείται για Top-k GROUP-BY ερωτήματα, διαθέτουμε μια σειρά μεταβλητών για την αποθήκευση των αποτελεσμάτων για κάθε δωμάτιο. Προκειμένου να μεγιστοποιηθεί ο αριθμός των δωματίων που μπορούν να υποστηριχθούν από αυτό το ερώτημα χρησιμοποιείται μια δομή δεδομένων που αποτελείται από τα ακόλουθα στοιχεία: 1) sameAsPrevious (1 bit): είναι ένα bit που δείχνει κατά πόσο το τρέχον αποτέλεσμα είναι το ίδιο με το προηγούμενο αποτέλεσμα και ως εκ τούτου δεν θα πρέπει να διαβιβάζεται η πλειάδα εάν είναι το ίδιο με το προηγούμενο, 2) vals (3 bits): ο αριθμός των τιμών στο αποτέλεσμα topk. Σημειώστε ότι ο αριθμός των τιμών είναι ο ίδιος με τον αριθμό των δωματίων που ανέφεραν τις τιμές τους. Ο μέγιστος αριθμός των τιμών (δηλαδή, κατ' ανώτατο όριο αριθμός δωματίων είναι 7), 3) count (22 bits): Αυτό το χαρακτηριστικό είναι ο αριθμός των αποτελεσμάτων για κάθε δωμάτιο χρησιμοποιώντας ένα 3-bit μετρητή για κάθε δωμάτιο (Μέγιστος αριθμός δωματίων = $7 \times 3\text{-bits} = 21$), 4) room (22 bits): Αυτό το χαρακτηριστικό γνώρισμα καταγράφει το id του κάθε δωματίου, και 5) sum (16 bits x μέγιστος αριθμός δωματίων): αποθηκεύει το άθροισμα του αποτελέσματος των αναγνώσεων του αισθητήρα για κάθε δωμάτιο. Δεδομένου ότι ο μέγιστος αριθμός των διαθέσιμων bytes στο μήνυμα TinyDB για ωφέλιμο φορτίο είναι 25 bytes, η δομή δεδομένων μας υποστηρίζει μέχρι 7 δωμάτια.

4.7. Σύνταξη Επερώτησης

Χρησιμοποιήσαμε την ακόλουθη σύνταξη για την επερώτηση:

```
SELECT TOP K attribute [,aggregate]
FROM sensors
[WHERE filter]
[GROUP BY attribute]
[ORDER BY [attribute|aggregate] [ASC|DESC]]
[SAMPLE PERIOD time (ms)]
```

Η δήλωση του χαρακτηριστικού (attribute) που αναφέρεται στην σύνταξη της επερώτησης αναφέρεται σε όλες τις μετρήσεις που μπορεί να ανακτηθούν από το sensor board καθώς και τις μεταβλητές που αποθηκεύονται τοπικά σε κάθε κόμβο αισθητήρα. Στο πλαίσιο του TinyDB, όταν το TOP k χαρακτηριστικό εκτελείται μέσω του δικτύου επιστρέφονται μόνο τα k υψηλότερα αποτελέσματα για το εν λόγω χαρακτηριστικό, εάν δεν χρησιμοποιείται το ORDER BY. Ωστόσο θα μπορούσαν να έχουν επιστρέψει εύκολα τα k χαμηλότερα αποτελέσματα με παρόμοιο τρόπο. Η συγκεντρωτική (aggregate) κατάσταση που αναφέρεται στο ερώτημα υπάγεται στην κατηγορία που αναφέρεται σε όλα τα μεγέθη που υποστηρίζονται από το πλαίσιο TinyDB. Τα εν λόγω μεγέθη μπορούν να διακριθούν σε:

1) distributive aggregates, όπου οι εγγραφές μπορούν να ομαδοποιηθούν σε δίκτυο χωρίς να διακυβεύεται η ορθότητα τους (π.χ., Max, Min, Sum, Count), και 2) holistic aggregates, όπου σε δίκτυο συνάθροισης μπορεί να διακυβεύσει το αποτέλεσμα ορθά (Π.χ., Median), οπότε όλες οι πλειάδες πρέπει να διαβιβάζονται στο sink πριν το ερώτημα να εκτελεστεί. Εάν ένα ερώτημα GROUP-BY έχει αναρτηθεί στο δίκτυο, τα αποτελέσματα ομαδοποιούνται από τη δήλωση του χαρακτηριστικού (attribute) και των συναθροιστικών (aggregates) και υπολογίζονται για κάθε ομάδα ξεχωριστά. Στα πειράματά μας, χρησιμοποιούμε το ακόλουθο ερώτημα:

```
SELECT TOP K room, AVG(temp)
FROM sensors
GROUP BY room
ORDER BY AVG(temp) [DESC]
SAMPLE PERIOD 4096
```

όπου k έχει ρυθμιστεί ως το 5% του πλήρους συνόλου απαντήσεων. Επίσης, η χρήση του ίδιου χρονικού διαστήματος για όλα τα πειράματα μας, προσδιορίζει το ποσό του χρόνου που οι αισθητήρες πρέπει να περιμένουν πριν από την επαναπιστοποίηση του υπολογισμού. Πιο συγκεκριμένα θέσαμε τη διάρκεια να είναι ίση με 4096ms.

Κεφάλαιο 5

Πειραματικά Αποτελέσματα

- 5.1. Εισαγωγή
 - 5.2. Πειραματική Σειρά 1: Κατανάλωση Ενέργειας
 - 5.3. Πειραματική Σειρά 2: Αποκοπή Πλειάδων
 - 5.4. Πειραματική Σειρά 3: Ευελιξία σε σχέση με το k
 - 5.5. Πειραματική Σειρά 4: Πληθικότητα με GROUP-BY
 - 5.6. Πειραματική Σειρά 5: Διάρκεια ζωής Δικτύου
-

5.1. Εισαγωγή

Για να αξιολογηθεί η αποτελεσματικότητα των αλγορίθμων που παρουσιάζονται στην παρούσα εργασία έχουν διεκπεραιωθεί πέντε πειραματικές σειρές. Στην πρώτη πειραματική σειρά έχουμε σύγκριση της κατανάλωσης ενέργειας των αλγορίθμων INT και MINT ως προς τους αλγόριθμους TAG και TINA, δείχνοντας ότι οι αλγόριθμοι INT και MINT επιτυγχάνουν σημαντική εξοικονόμηση ενέργειας σε σύγκριση με τους άλλους δύο. Στη δεύτερη πειραματική σειρά, μελετάμε το μέγεθος των πλειάδων που αποκόπτονται σε κάθε επίπεδο ως προς τον sink, για τους αλγόριθμους INT και MINT, από το k -Covered Bound-Set V_i . Στην τρίτη πειραματική σειρά, διερευνάτε η επεκτασιμότητα της παραμέτρου k δηλαδή μελετάμε την απόδοση του αλγορίθμου για διαφορετικές τιμές του k . Στην τέταρτη πειραματική σειρά διερευνήθηκε η επίδραση της πληθικότητας GROUP-BY. Σε όλα τα σύνολα δεδομένων μοιράζουμε τυχαία με ομοιόμορφο τρόπο τους αισθητήρες σε δωμάτια. Σε αυτό το πείραμα, διανέμουμε τους αισθητήρες σε διάφορες συνθέσεις δωματίων για τη μελέτη της ενεργειακής κατανάλωσης όλων των αλγορίθμων. Τέλος, στην πέμπτη πειραματική σειρά, αξιολογούμε την αποτελεσματικότητα του πλαισίου TinyDB εστιάζοντας στην κατανάλωση ενέργειας και τη διάρκεια ζωής του συστήματος.

Ο πίνακας 2 συνοψίζει τις παραμέτρους διαμόρφωσης για όλα τα πειράματα που αναφέρονται στα επόμενα τμήματα.

Υποκεφάλαιο	Σκοπός	Σύνολο Δεδομένων	k	Αριθμός Δωματίων (R)
5.2.	Energy Consumption	GDI14,AtmoMon32 Intel49	5%	4-7
5.3.	Pruning Magnitude	AtmoMon32	5%	7
5.4.	Scalability of k	GDI14	5%- 100%	4
5.5.	GROUP-BY cardinality	GDI14	5%	1-7
5.6.	Network lifetime	GDI14	5%	4

Πίνακας 5.1.1 : Παράμετροι διαμόρφωσης για όλες τις πειραματικές σειρές

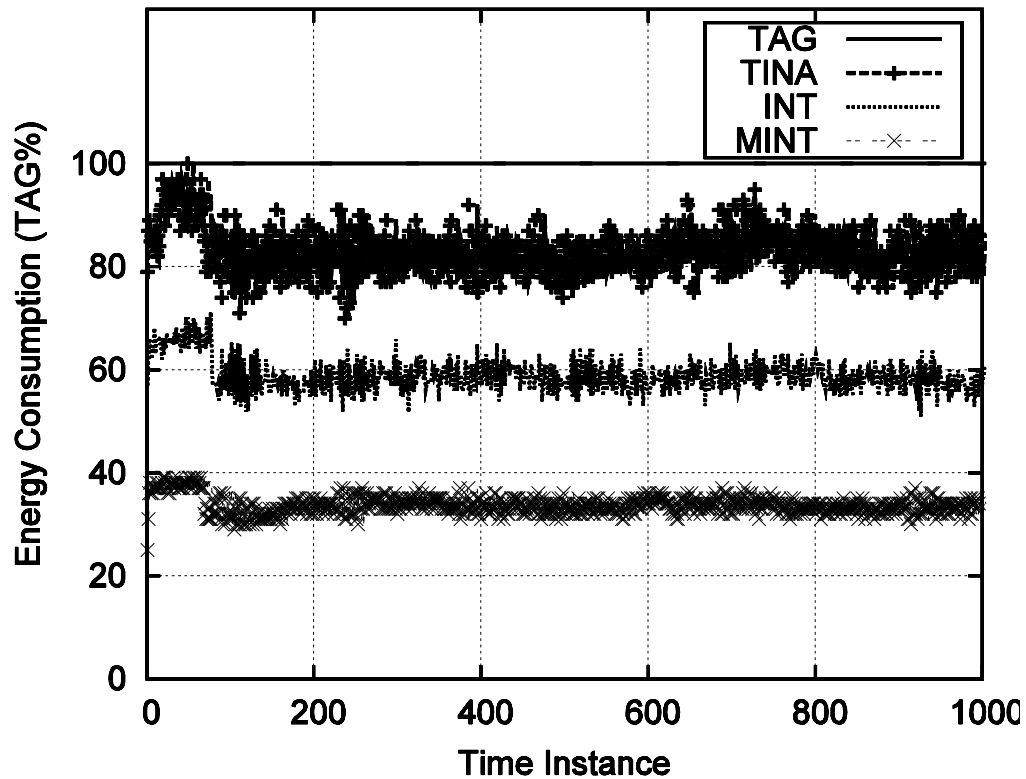
5.2. Πειραματική Σειρά 1: Κατανάλωση Ενέργειας

Στην πρώτη πειραματική σειρά έχουμε σύγκριση της κατανάλωσης ενέργειας των αλγορίθμων INT και MINT ως προς τους αλγόριθμους TAG και TINA. Έχουμε εκτελέσει το ερώτημα Q στα τρία σύνολα δεδομένων και μετρήσαμε την κατανάλωση ενέργειας ξεχωριστά για κάθε σύνολο δεδομένων.

Στο Σχήμα 5.2.1, παρουσιάζεται η κατανάλωση ενέργειας από τους τέσσερις αλγόριθμους (MINT, INT, TINA και TAG) χρησιμοποιώντας το σύνολο δεδομένων GDI14. Προκαταρκτικά, να αναφέρω ότι η κλίμακα της ενέργειας μεταξύ των διαδοχικών στιγμών κυμαίνεται σε μεγάλο βαθμό. Αυτό συμβαίνει λόγω της αυθαιρεσίας που υπάρχει, όταν συμβεί αποκοπή πλειάδων, οι οποίες δεν υπάρχει περίπτωση να βρίσκονται στο τελικό αποτέλεσμα, οπότε αποκόπονται και όταν λαμβάνει χώρα μια χρονική συνοχή, δηλαδή όταν η πλειάδα είναι η ίδια με της προηγούμενης χρονικής στιγμής, οπότε δεν χρειάζεται να διαβιβαστεί και σε αυτή τη

χρονική στιγμή. Για να διορθωθεί η κατάσταση αυτή κατά τα επόμενα διαγράμματα, εφαρμόσαμε spline παρεμβολή εξομάλυνσης μεταξύ των διαδοχικών σημείων των δεδομένων.

**Energy Consumption (Average for all n sensors) (NO INTERPOLA
(Algorithm(s)=All Dataset=GDI14, n=14, network=250Kbps)**



Σχήμα 5.2.1: Κατανάλωση Ενέργειας αλγορίθμων για το σύνολο δεδομένων GDI14 χωρίς παρεμβολές στα δεδομένα

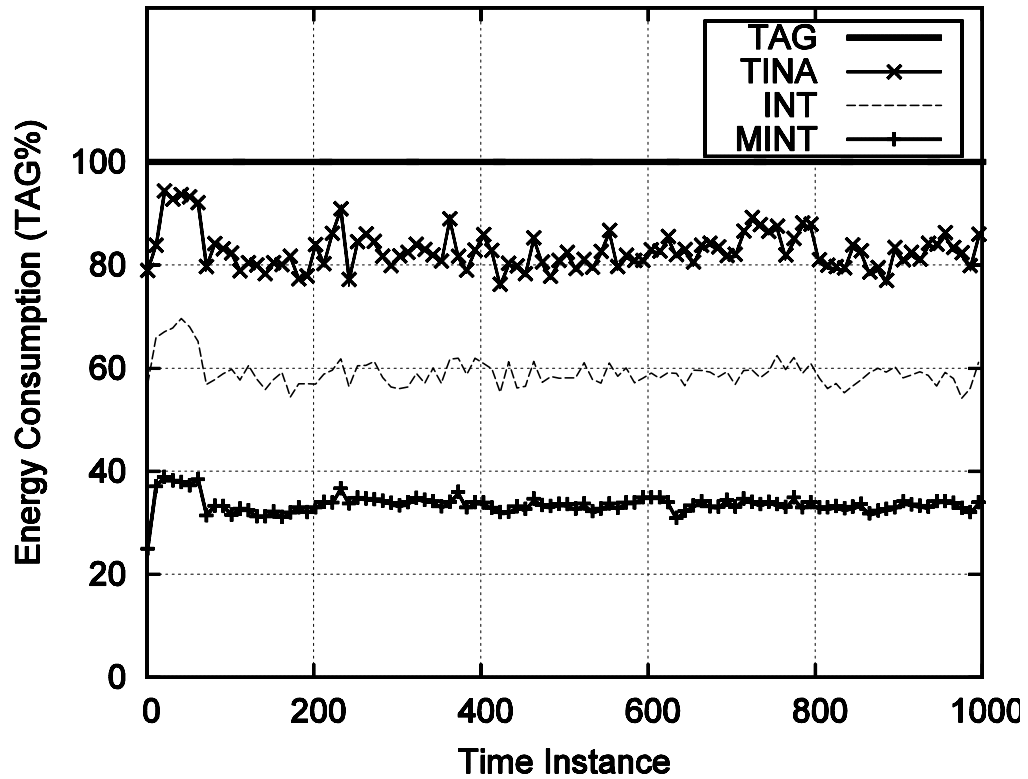
Στο Σχήμα 5.2.2, παρουσιάζονται τα αποτελέσματα χρησιμοποιώντας το σύνολο δεδομένων GDI14. Χρησιμοποιούμε τον αλγόριθμο TAG ως βάση σύγκρισης και έχει πάντα τιμή 100%. Με βάση τα πραγματικά δεδομένα των προσομοιώσεων, πριν υπολογίσουμε τα ποσοστά ως προς 100%, ο TAG έχει μέσο όρο απόδοσης περίπου $57 \pm 2.52\text{J}$ για τους 14 κόμβους του δικτύου. Να υπενθυμίσω ότι στον αλγόριθμο TAG, ένας αισθητήρας μεταδίδει πάντα όλες τις πλειάδες στο sink. Αν και ο αλγόριθμος TINA επιστρέφει ακόμα όλες τις απαντήσεις στο sink, παίρνει μέση κατανάλωση ενέργειας μέχρι $48 \pm 1.57\text{J}$. Το γεγονός αυτό επικυρώνει ότι η εκμετάλλευση της χρονικής συνοχής μπορεί να είναι ευεργετική στις περισσότερες περιπτώσεις. Από την

άλλη πλευρά, ο αλγόριθμος INT παρέχει στο δίκτυο την αποκοπή των αποτελεσμάτων η οποία μειώνει την κατανάλωση ενέργειας στο $34 \pm 1J$ (δηλαδή ≈ 41 % λιγότερο από το TAG).

Τέλος, ο αλγόριθμος MINT εκτός από την αποκοπή πλειάδων, εκμεταλλεύεται τη χρονική συνοχή και καταναλώνει ενέργεια κατά μέσο όρο $19 \pm 0.56J$ η οποία αντιστοιχεί σε μείωση της ενέργειας κατά 66% από το TAG, το 49% μείωση της κατανάλωσης ενέργειας από TINA και 25% από το INT. Ο λόγος για τον οποίο οι αλγόριθμοι TINA και MINT ένα παρόμοιο μοτίβο είναι διότι και στις δύο καμπύλες, η μείωση της κατανάλωσης ενέργειας κυριαρχείται από την εξοικονόμηση δαπανών που οφείλονται στη χρονική συνοχή μεταξύ των διαδοχικών χρονικών σημείων.

Στο σχήμα αυτό επίσης, εξετάζουμε τις αποκλίσεις για τους αλγόριθμους TINA, INT και MINT για όλα τα πειράματα. Στην περίπτωση του αλγορίθμου TINA, οι αποκλίσεις αποδίδονται στο γεγονός ότι, σε ορισμένα χρονικά διαστήματα, οι αισθητήρες εκμεταλλεύονται τη χρονική συνοχή και δεν αναφέρουν τα αποτελέσματά τους στους γονείς τους. Αυτό μειώνει τη συνολική κατανάλωση ενέργειας του δικτύου. Στην περίπτωση του αλγορίθμου INT, οι αποκλίσεις συσχετίζονται με το γεγονός ότι, σε ορισμένα χρονικά διαστήματα, το ποσό των αποτελεσμάτων μειώνεται ή αυξάνεται από το V_i , λόγω των τιμών στο σύνολο δεδομένων. Αυτό είναι μια ένδειξη ότι η top-k απάντηση έχει αλλάξει κατά τη συγκεκριμένη χρονική σήμανση και ότι αυτό έχει επιφέρει κάποια αύξηση στην κατανάλωση ενέργειας, έως ότου διαδοθεί στο sink. Από την άλλη πλευρά, οι αποκλίσεις του αλγορίθμου MINT συσχετίζονται με τα δύο ανωτέρω χαρακτηριστικά, καθώς ο MINT εκμεταλλεύεται τόσο τη χρονική συνοχή όσο και την αποκοπή πλειάδων top-k.

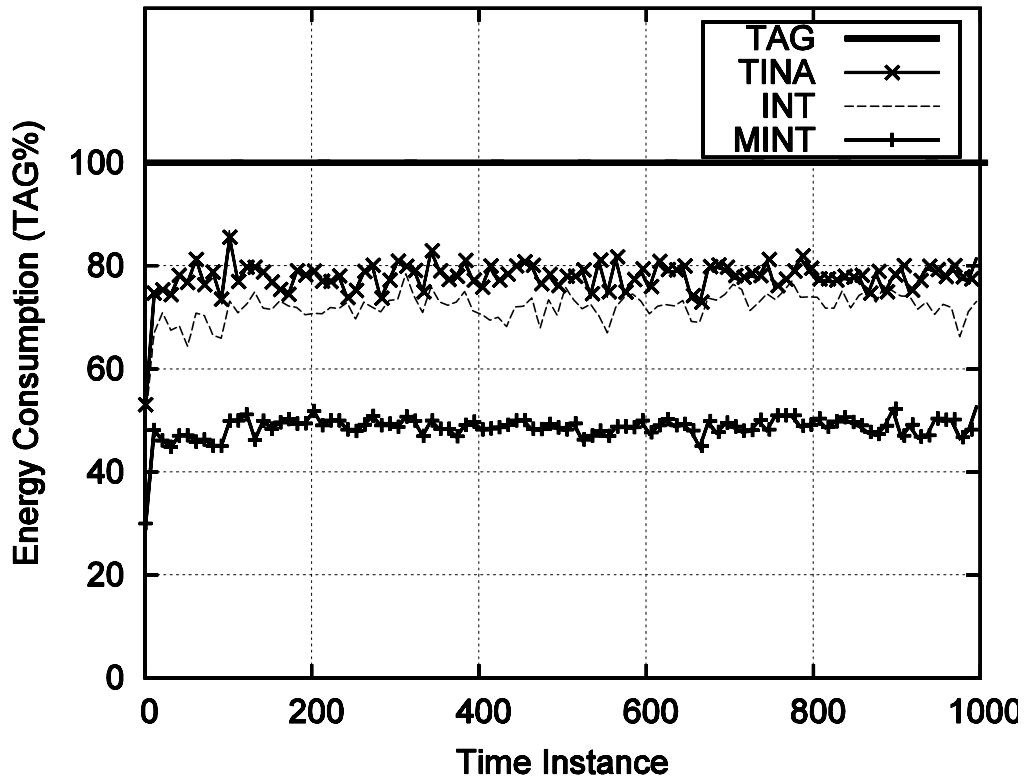
Energy Consumption (Average for all n sensors)
 (Algorithm(s)=All Dataset=GDI14, n=14, network=250Kbps)



Σχήμα 5.2.2: Κατανάλωση Ενέργειας αλγορίθμων για το σύνολο δεδομένων GDI14

Επαναλαμβάνοντας το ίδιο πείραμα για το σύνολο δεδομένων AtmoMon32, παρατηρούμε από το σχήμα 5.2.3, ότι ο MINT διατηρεί συνεχώς ένα ανταγωνιστικό πλεονέκτημα έναντι των TAG και TINA. Ειδικότερα, παρατηρούμε ότι ο MINT καταναλώνει 50% λιγότερη ενέργεια από τον TAG (δηλαδή $115 \pm 4J$ έναντι $234 \pm 2J$). Το ίδιο συμπέρασμα ισχύει και για τον αλγόριθμο INT με τη διαφορά ότι παρατηρούμε ότι οι επιδόσεις σε σύγκριση με τον TINA έχουν μειωθεί. Αυτό συμβαίνει επειδή στο σύνολο δεδομένων AtmoMon32, οι τιμές της θερμοκρασίας δεν αλλάζουν συχνά και αυτό επιτρέπει τη χρονική συνοχή για τη σημαντική μείωση του αριθμού των πλειάδων που μεταδίδονται μέσω του δικτύου. Ωστόσο, η αποκοπή των πλειάδων top-k που εφαρμόζει ο αλγόριθμος INT έχει καταφέρει να μειώσει σημαντικά το μέγεθος των πακέτων που μεταδίδονται μέσω του δικτύου διατηρώντας έτσι ένα πλεονέκτημα σε σχέση με τον TINA.

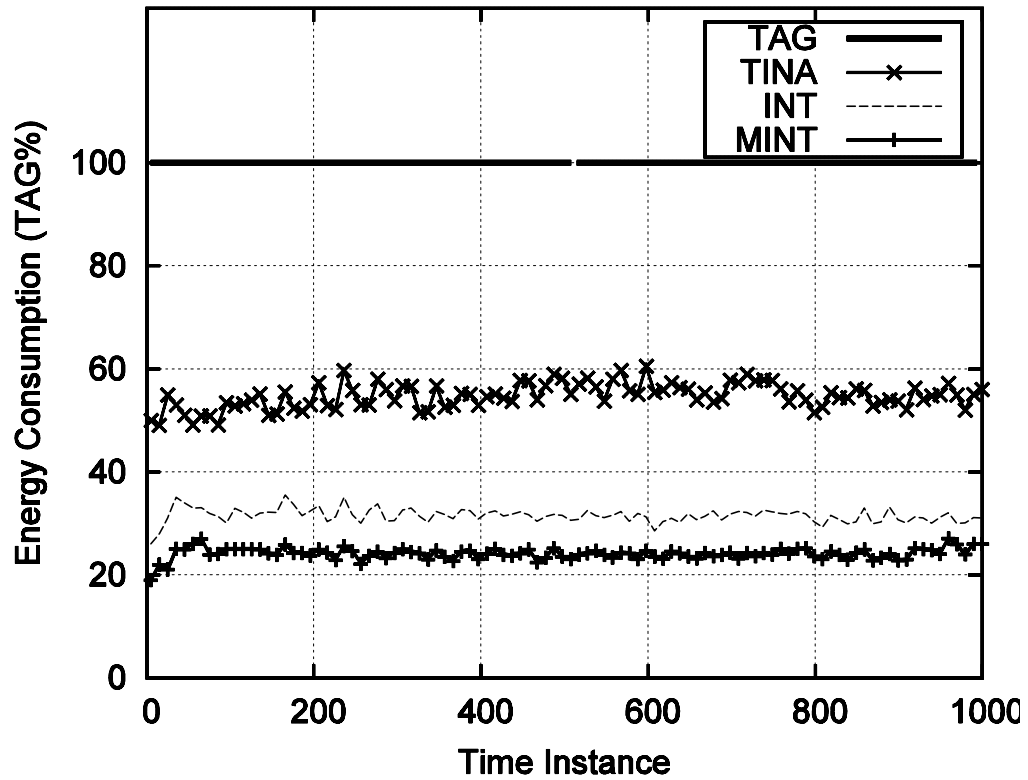
Energy Consumption (Average for all n sensors)
(Algorithm(s)=All Dataset=AtmoMon32, n=32, network=250Kb)



**Σχήμα 5.2.3: Κατανάλωση Ενέργειας αλγορίθμων για το σύνολο δεδομένων
AtmoMon32**

Στο τελευταίο σύνολο δεδομένων, Intel49 (Σχήμα 5.2.4.) παρατηρούμε ότι όλοι οι αλγόριθμοι συμπεριφέρονται κατά τρόπο παρόμοιο με τα προηγούμενα πειράματα. Η διαφορά είναι ότι η ενεργειακή απόδοση όλων των αλγορίθμων έχει μειωθεί σε σύγκριση με το TAG. Ένας λόγος που συμβαίνει αυτό, είναι το γεγονός ότι, όπως στο σύνολο δεδομένων AtmoMon32, οι τιμές της θερμοκρασίας του Intel49 δεν αλλάζουν συχνά, οπότε αξιοποιείται η χρονική συνοχή των αλγορίθμων TINA και MINT. Από την άλλη πλευρά, ο αλγόριθμος INT που δεν εφαρμόζει χρονική συνοχή, ξεπερνά σημαντικά τους αλγόριθμους TAG και TINA. Αυτό σημαίνει ότι ο μηχανισμός αποκοπής πλειάδων του INT, μειώνει σημαντικά το μέγεθος των πακέτων, ελαχιστοποιώντας έτσι την κατανάλωση ενέργειας που συνδέεται με τη μετάδοση. Τα αποτελέσματα για όλα τα πειράματα συνοψίζονται στον Πίνακα 5.2.1.

Energy Consumption (Average for all n sensors)
 (Algorithm(s)=All Dataset=Intel49, n=49, network=250Kbps)



Σχήμα 5.2.4: Κατανάλωση Ενέργειας αλγορίθμων για το σύνολο δεδομένων Intel49

Algorithm \ Dataset	GDI14	AtmoMon32	Intel49
TAG	57±2.52J	234±2J	523±22J
TINA	48±1.57J	183±6J	289±15J
INT	34±1.01J	170±7J	187±08J
MINT	19±0.56J	115±4J	139±06J

Πίνακας 5.2.1: Μέση κατανάλωση ενέργειας για όλους τους αισθητήρες, πειραματική σειρά 1: Αξιολόγηση των αλγορίθμων TAG, TINA, INT και MINT πάνω σε διάφορα σύνολα δεδομένων.

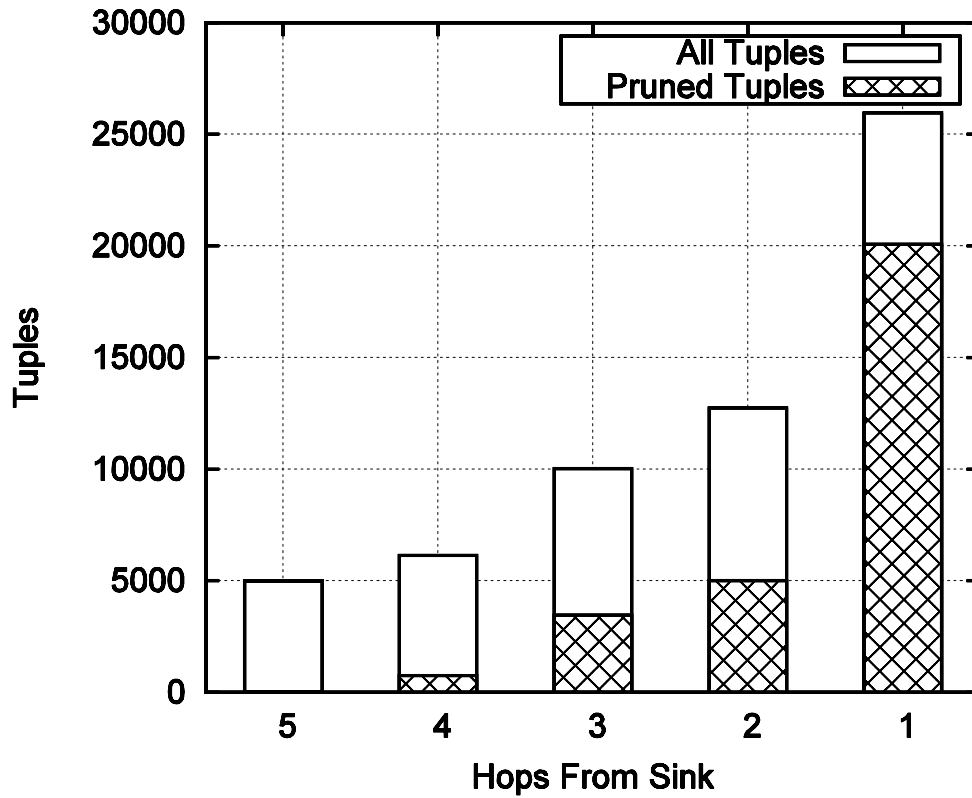
5.3. Πειραματική Σειρά 2: Αποκοπή Πλειάδων

Στη δεύτερη πειραματική σειρά, μελετάμε το μέγεθος των πλειάδων που αποκόπτονται σε κάθε επίπεδο ως προς τον sink, για τους αλγόριθμους INT και MINT, από το k-Covered Bound-Set V_i , χρησιμοποιώντας το σύνολο δεδομένων AtmoMon32. Στο Σχήμα 5.3.1 παρουσιάζουμε με ένα άσπρο κουτί το μέσο όρο αριθμού των πλειάδων σε κάθε επίπεδο της τοπολογίας (για όλες τις 1000 χρονικές στιγμές). Επίσης, στη γραφική παράσταση αναπαριστούμε με διακεκομμένο πλαίσιο το συνολικό αριθμό των πλειάδων που αποκόπτονται με τη χρήση του Αλγόριθμου 2 που επεξηγήθηκε στο τρίτο κεφάλαιο.

Παρατηρούμε ότι όσο πιο κοντά οδεύουμε προς το sink, η αποκοπή των πλειάδων στο πλαίσιο μας αυξάνεται εκθετικά. Αυτό οφείλεται στο γεγονός ότι η πληθικότητα του V_i μπορεί στη χειρότερη περίπτωση να αυξηθεί με γεωμετρική πρόοδο (δηλαδή, κάθε αισθητήρας βρίσκεται σε διαφορετικό αριθμό δωματίου). Ειδικότερα, παρατηρούμε ότι η αποκοπή πλειάδων στο επίπεδο πέντε κυμαίνεται από 0%, σε 39% στο επίπεδο δύο και 77% στο επίπεδο ένα. Είναι σημαντικό να υπογραμμιστεί το γεγονός ότι μια τέτοια περικοπή παρουσιάζει μια μείωση άνω των 20.000 πλειάδων.

Μια τελευταία παρατήρηση είναι ότι τα αποτελέσματα αυτά ισχύουν τόσο για το MINT όσο και για τον INT, καθώς οι δύο αυτοί αλγόριθμοι διαφέρουν μόνο στο πώς το V_i διατηρείται και όχι με το τελικό περιεχόμενο του εν λόγω δικτύου.

Pruning Magnitude (Average for all n sensors)
 (Algorithm(s)=MINT Dataset=AtmoMon32, n=32, network=256)



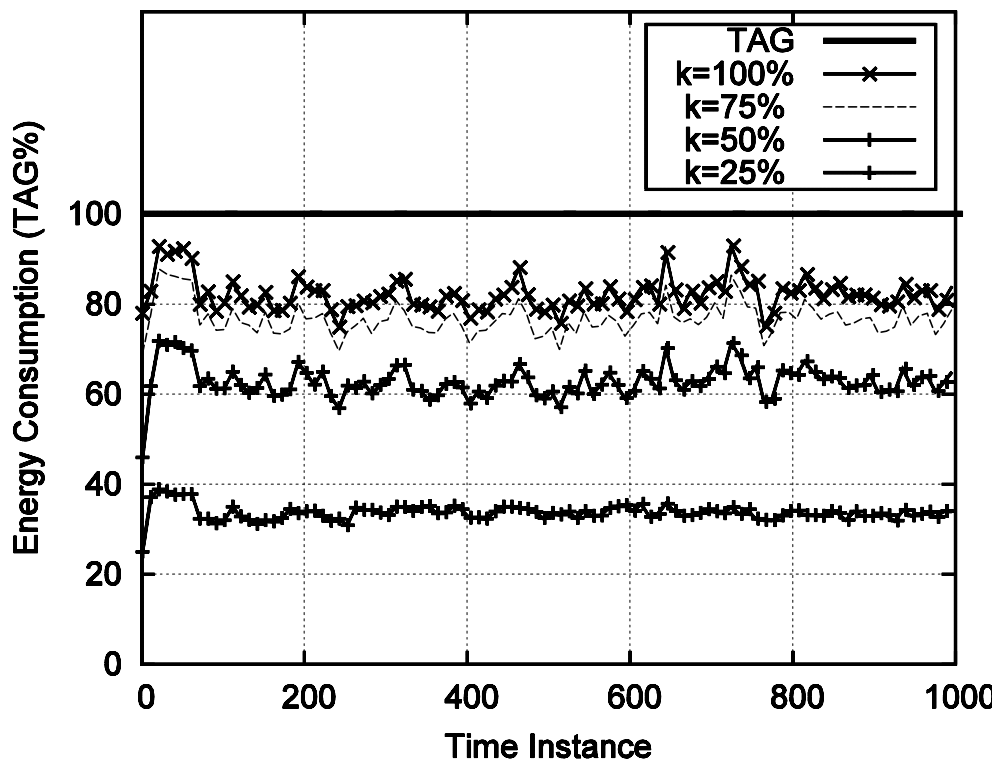
Σχήμα 5.3.1: Αποκοπή πλειάδων ως προς τον sink για το σύνολο δεδομένων Atmomon32

5.4. Πειραματική Σειρά 3: Ευελιξία σε σχέση με το k

Στην τρίτη πειραματική σειρά, αξιολογήσαμε την αποτελεσματικότητα του αλγορίθμου MINT σε σχέση με την παράμετρο k. Πιο συγκεκριμένα, αυξήσαμε την παράμετρο k, διατηρώντας παράλληλα την ίδια τοπολογία δικτύου. Αναμένουμε ότι με την αύξηση της παραμέτρου k, το μέγεθος των πακέτων θα αυξηθεί, επίσης θα διαγράφονται λιγότερα πακέτα από το V_i . Έχουμε αξιοποιήσει το σύνολο δεδομένων GDI14, για αυτό το πείραμα, και μετρούμε τη μέση κατανάλωση ενέργειας για όλους τους κόμβους αισθητήρων. Ωστόσο, να αναφέρουμε ότι παρόμοιες παρατηρήσεις, ισχύουν και για τα υπόλοιπα σύνολα δεδομένων.

Το σχήμα 5.4.1 δείχνει τα αποτελέσματα του πειράματος μας. Για τη χαμηλότερη τιμή του k ($k = 25\%$ της απάντησης), η συνολική κατανάλωση ενέργειας είναι 66% λιγότερο από ό, τι στο TAG ($19 \pm 0.5J$). Παρατηρούμε ότι, καθώς η τιμή της παραμέτρου k αυξάνεται, το κέρδος απόδοση είναι μειωμένο. Ιδιαίτερα, για το $k = 50\%$ και $k = 75\%$ της ενέργειας σε σύγκριση με το TAG φθάνει 36,5% ($36 \pm 1.6J$) και 23,4% ($44 \pm 1.5J$) αντίστοιχα. Αυτό είναι αναμενόμενο, όπως μεταδίδεται ο αριθμός των αποτελεσμάτων από κάθε κόμβο αισθητήρα συσχετίζεται με την παράμετρο k (δηλαδή, οι υψηλότερες τιμές του k μείωση του αριθμού των πλειάδων που αποβάλλονται από V_i). Όταν η παράμετρος k φθάνει το 100% (δηλαδή, όλοι οι κόμβοι αισθητήρων διαβιβάζουν το σύνολο των αποτελεσμάτων τους), τότε ο αλγόριθμος MINT συμπεριφέρεται παρόμοια με τον αλγόριθμο TINA. Δεδομένου ότι δεν γίνεται αποκοπή πλειάδων για τους αισθητήρες, ο αλγόριθμος εκμεταλλεύεται μόνο τη χρονική συνοχή ακριβώς όπως ο αλγόριθμος TINA. Εντούτοις, όπως ο TINA έτσι και ο MINT εξακολουθεί να διατηρεί το ανταγωνιστικό πλεονέκτημα του 18% ($47 \pm 1.59J$) μειωμένη κατανάλωση ενέργειας σε σχέση με το TAG ($57 \pm 2.52J$).

Scalability with respect to k (Average for all n sensors)
(Algorithm(s)=MINT Dataset=GDI14, $n=14$, network=250Kbp)



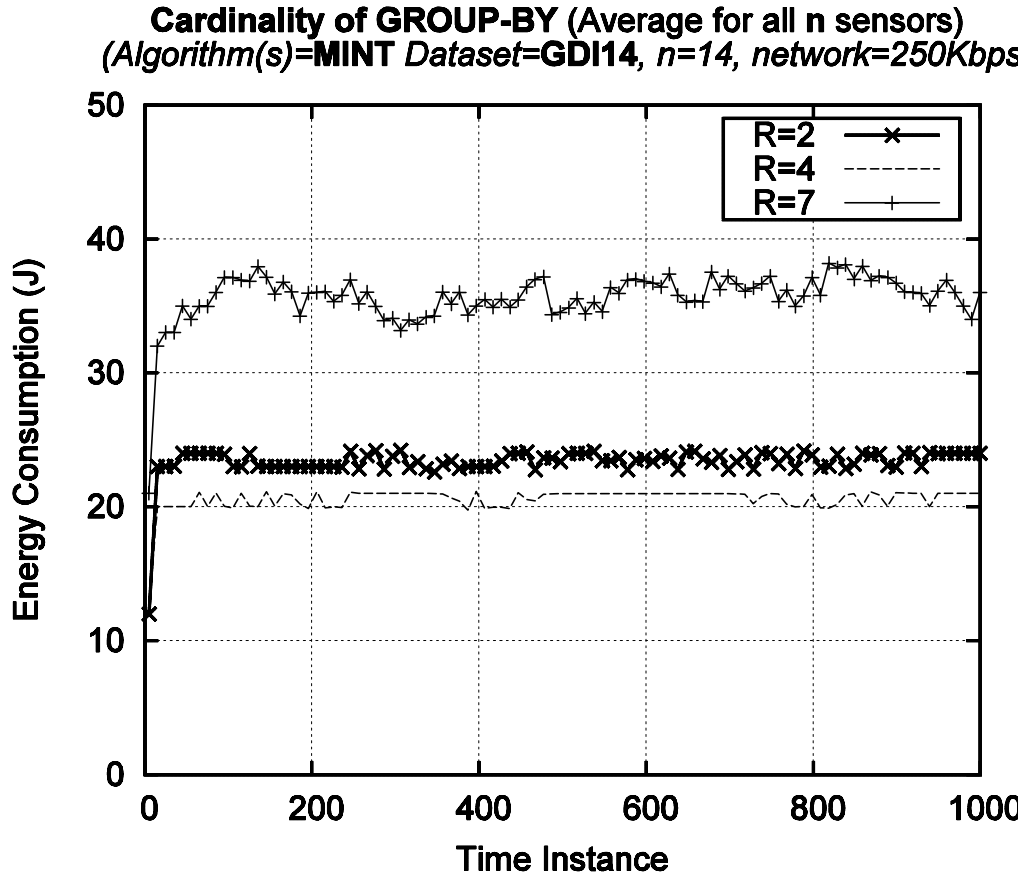
Σχήμα 5.4.1: Ευελιξία σε σχέση με το k

5.5. Πειραματική Σειρά 4: Πληθικότητα με GROUP-BY

Στην τέταρτη πειραματική σειρά διερευνήθηκε η επίδραση της πληθικότητας GROUP-BY. Σε όλα τα σύνολα δεδομένων μοιράζουμε τυχαία με ομοιόμορφο τρόπο τους αισθητήρες σε δωμάτια. Σε αυτό το πείραμα, διανέμουμε τους αισθητήρες σε διάφορες συνθέσεις δωματίων για τη μελέτη της ενεργειακής κατανάλωσης όλων των αλγορίθμων. Πιο συγκεκριμένα, ορίζουμε τον αριθμό των δωματίων (R) σε 2, 4 και 7 για το σύνολο δεδομένων GDI14 και διανέμουμε ομοιόμορφα τους αισθητήρες σε κάθε δωμάτιο, αντίστοιχα. Μετράμε την μέση κατανάλωση ενέργειας για τους 14 κόμβους αισθητήρων. Υπάρχουν δύο σημαντικοί παράμετροι που επηρεάζουν την πληθικότητα. Πρώτον, όταν το R αυξάνεται, αυξάνεται και το μέγεθος του πακέτου, καθώς η δομή των δεδομένων TopKRoom διαθέτει χώρο για να αποθηκευτούν τα R αποτελέσματα. Από την άλλη πλευρά, δεδομένου ότι ένας μικρότερος αριθμός αισθητήρων διανέμεται σε κάθε δωμάτιο, στο χαμηλότερο επίπεδο των κόμβων μπορεί να υπολογίσει γρήγορα το ακριβές αποτέλεσμα στο δωμάτιο, έτσι το μέγεθος αποκοπής πλειάδων είναι αυξημένο. Δεύτερον, όταν μειώνεται το R, το μέγεθος του πακέτου μειώνεται για τους ίδιους παραπάνω λόγους. Ωστόσο, σε αυτή την περίπτωση το μέγεθος αποκοπής πλειάδων μειώνεται γρήγορα, καθώς στο ανώτερο επίπεδο οι κόμβοι αισθητήρα έχουν μια πλήρη εικόνα για τα ακριβή αποτελέσματα ενός δωματίου.

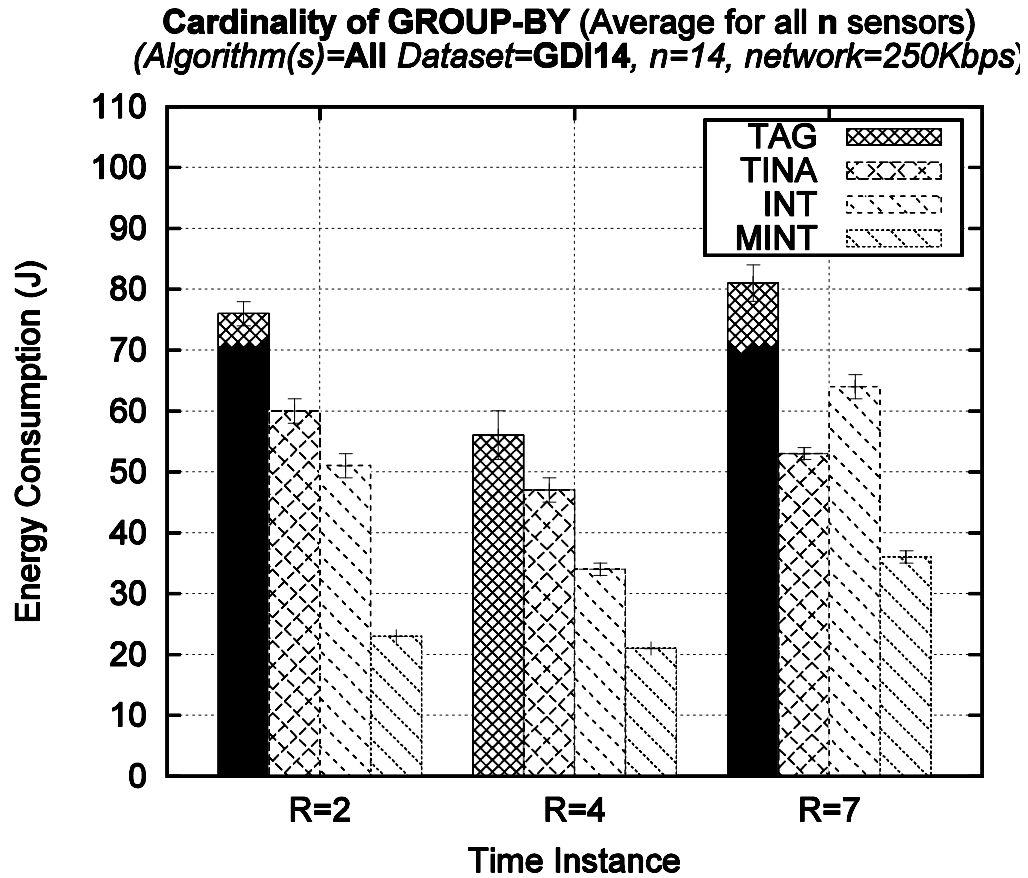
Το Σχήμα 5.5.1, δείχνει το πρώτο αποτέλεσμα του πειράματος. Παρατηρούμε ότι η καλύτερη ενεργειακή απόδοση εμφανίζεται όταν ο αριθμός των δωματίων R είναι ίσος με 4 (δηλαδή, $21 \pm 0.7J$). Στην περίπτωση που τα δωμάτια είναι λιγότερα, (δηλαδή, $R = 2$) παρατηρούμε ότι η κατανάλωση ενέργειας αυξήθηκε ελαφρά αν και στην περίπτωση αυτή τα στοιχεία ωφέλιμου φορτίου του MINT μειώνονται σχεδόν στο μισό μέγεθος. Ο λόγος για την αύξηση αυτή, είναι το γεγονός ότι ο MINT, σχεδόν ποτέ δεν κάνει αποκοπή πλειάδων από το V_i σε κόμβους αισθητήρα που έχουν άλμα μεγαλύτερο από 1 (δηλαδή, τα αποτελέσματα πρέπει να φθάσουν σε κόμβους πολύ κοντά στο sink ώστε ένας κόμβος να είναι σε θέση να εξαλείψει πλειάδες). Από την άλλη πλευρά, όταν το $R = 7$, παρατηρούμε μια σημαντική αύξηση της ενεργειακής κατανάλωσης. Αυτό οφείλεται στο γεγονός ότι το ωφέλιμο φορτίο των στοιχείων έχει πλέον διαμορφωθεί σε 7 πλειάδες σε κάθε αισθητήρα που απαιτεί σχεδόν τη διπλάσια

συνολική ενέργεια μετάδοσης. Ωστόσο, στην προκειμένη περίπτωση ο μηχανισμός αποκοπής πλειάδων εξαλείφει πλειάδες στα χαμηλότερα επίπεδα από την τοπολογία του δικτύου και αυτός είναι ο λόγος για τον οποίο υπάρχει η τυπική απόκλιση αυτής της αύξησης στο πείραμα (δηλαδή, $36 \pm 1,68$ J).



Σχήμα 5.5.1: Πληθικότητα με GROUP-BY για τον αλγόριθμο MINT

Στο Σχήμα 5.5.2, παρουσιάζουμε τα αποτελέσματα όλων των αλγορίθμων για το σύνολο δεδομένων GDI14 με διαφορετικά cardinalities. Έχουμε διαπιστώσει ότι ο MINT διατηρεί πάντα ένα πλεονέκτημα έναντι των ανταγωνιστών του σε όλα τα σενάρια. Στην περίπτωση όπου $R = 7$, έχουμε παρατηρήσει ότι ο TINA παρουσιάζει καλύτερες επιδόσεις από τον INT. Η εξέλιξη αυτή αποδίδεται στο γεγονός ότι ο TINA καταστέλλει πολλά αποτελέσματα από το να διαβιβάζονται στο δίκτυο, λόγω της χρονικής συνοχής του. Από την άλλη πλευρά, ο MINT εφαρμόζει τόσο την αποκοπή πλειάδων top-k όσο και τη χρονική συνοχή και για αυτό το λόγο υπερτερεί όλων των αλγορίθμων.



Σχήμα 5.5.2: Πληθικότητα με GROUP-BY για όλους τους αλγόριθμους

5.6. Πειραματική Σειρά 5: Διάρκεια ζωής Δικτύου

Στην πέμπτη πειραματική σειρά, αξιολογούμε την αποτελεσματικότητα του πλαισίου TinyDB εστιάζοντας στην κατανάλωση ενέργειας και τη διάρκεια ζωής του συστήματος. Ορίζουμε χρόνο ζωής του δικτύου, τη μέση ποσότητα ενέργειας στο δίκτυο. Ειδικότερα, το πιο κάτω άθροισμα δηλώνει την ποσότητα ενέργειας που είναι διαθέσιμη κατά το χρόνο t σε ένα δίκτυο n αισθητήρων:

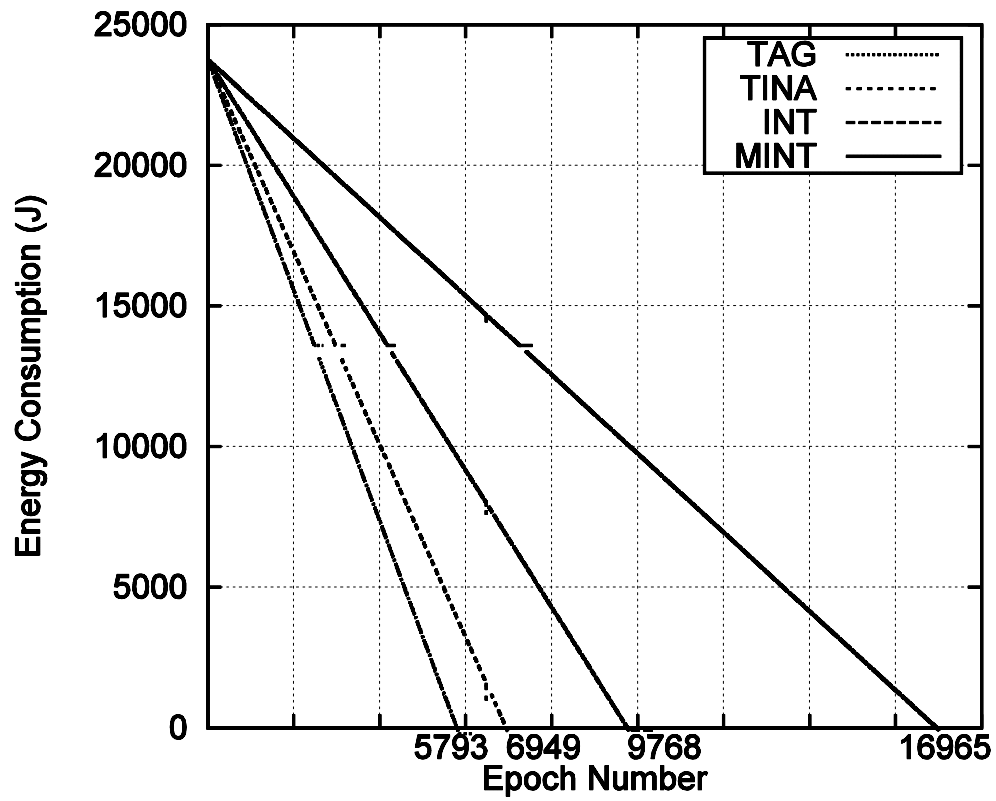
$$\text{Ενέργεια (t)} = \sum_{i=1}^n \text{διαθέσιμη ενέργεια (s}_i, t) / n$$

Η διαθέσιμη ενέργεια (s_i, t) δηλώνει την ενέργεια που είναι διαθέσιμη στον s_i αισθητήρα ($i \leq n$) κατά τη χρονική στιγμή t . Έχουμε καθορίσει τη διάρκεια ζωής του δικτύου, όπως στο [34], στη χρονική στιγμή t' κατά την οποία η Ενέργειας $(t') = 0$.

Σημειώστε ότι αυτό ισχύει μόνο για το περίπτωση κατά την οποία οι αισθητήρες λειτουργούν χρησιμοποιώντας μπαταρίες. Διπλές μπαταρίες (AA), που χρησιμοποιούνται σε πολλά τρέχοντα σχέδια αισθητήρων (συμπεριλαμβανομένου του αισθητήρα TelosB) λειτουργούν σε 3V τάση και παροχή ρεύματος 2.500 mAh (milliAmpere ανά ώρα). Υποθέτοντας όπως το [31], ότι μόνο 2.200 mAh είναι διαθέσιμο, μπορούμε να υπολογίσουμε ότι 2xAA μπαταρίες προσφέρουν 23, 760J ($2, 200\text{mAh} \times 60\text{min} \times 60 \times 3\text{V}$). Ξεκινάμε με αυτή την αρχική ενέργεια και αφαιρούμε σε κάθε χρονική στιγμή για κάθε αισθητήρα την απαιτούμενη ενέργεια για την επεξεργασία του top-k ερωτήματος. Σταματούμε την πιο πάνω επανάληψη όταν δεν υπάρχει ενέργεια.

Το Σχήμα 5.6.1, απεικονίζει το μέσο όρο ενεργειακής κατάστασης του δικτύου αισθητήρων, σε κάθε χρονική στιγμή, κατά την εκτέλεση ενός ερωτήματος χρησιμοποιώντας το σύνολο δεδομένων GDI14. Παρατηρούμε ότι η διαθέσιμη ενέργεια των αισθητήρων για τον αλγόριθμο TAG καταναλώνεται πολύ πιο γρήγορα από τους υπόλοιπους αλγορίθμους, που οδηγεί σε μια δικτυακή ζωή μόλις 5.793 χρονικών στιγμών (δηλαδή, 193 λεπτά). Ο αλγόριθμος TINA βρίσκεται στην τρίτη θέση με 6.949 χρονικές στιγμές (δηλαδή, 231 λεπτά) και ο INT δεύτερος με 9.768 χρονικές στιγμές (δηλαδή, 325 λεπτά). Τέλος, ο MINT καταναλώνει τη διαθέσιμη ενέργεια πολύ πιο αργά στη χρονική στιγμή 43.824 (δηλαδή, 565 λεπτά), και αυτό μεταφράζεται σε αύξηση 292% της διάρκειας ζωής του δικτύου.

Network Lifetime (Average energy consumption for all n sens
(Algorithm(s)=All Dataset=GDI14, n=14, network=250Kbp



Σχήμα 5.6.1: Διάρκεια ζωής Δικτύου για τους διάφορους αλγόριθμους

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1. Συμπεράσματα

6.2. Μελλοντική Εργασία

6.1. Συμπεράσματα

Στην παρούσα εργασία, παρουσιάζουμε ένα καινοτόμο πλαίσιο για την αποτελεσματική παρακολούθηση ασύρματων δικτύων αισθητήρων, για να επιτευχθεί η αποδοτικότητα και η ποιότητα του αποτελέσματος ενός ερωτήματος.

Για να δείξουμε την αποτελεσματικότητα του αλγόριθμου MINT, έχουμε εφαρμόσει μια σειρά πειραμάτων πάνω σε τρία σύνολα δεδομένων, και συγκρίνουμε τον αλγόριθμο με άλλους τρεις. Ως αποτέλεσμα, αποδεικνύουμε ένα ποσοστό μέχρι 66% εξοικονόμησης ενέργειας σε σύγκριση με τον αλγόριθμο TAG που είναι ο βασικός αλγόριθμος του TinyDB. Ο MINT ελαχιστοποιεί τόσο το μέγεθος όσο και τον αριθμό των πακέτων που διαβιβάζονται στο δίκτυο (μέχρι 77%), παρατείνοντας με αυτόν τον τρόπο τη διάρκεια ζωής ενός ασύρματου δικτύου αισθητήρων. Ο αλγόριθμος MINT επιτυγχάνει σημαντική μείωση της καταναλισκόμενης ενέργειας σε σύγκριση με τους υπόλοιπους αλγορίθμους, με αποτέλεσμα να επιμηκύνεται ο χρόνος ζωής του δικτύου.

6.2. Μελλοντική Εργασία

Στο μέλλον, θα μπορούσε να διερευνηθεί η δυνατότητα της εφαρμογής αυτών των ιδεών πάνω σε Κινητά Δίκτυα Αισθητήρων και Δίκτυα Συσκευών Smartphone.

Βιβλιογραφία

- [1] MPR – MIB Series User Manual and data sheet. [Online] www.xbow.com, June 2006.
- [2] P. Rentala, R. Musunuri, S. Gandham, U. Saxena. "Survey on Sensor Networks, University of Texas at Dallas". *Survey Paper submitted as per the requirements of Mobile Computing (CS 6392) Course, Proc. of International Conference on Mobile Computing and Networking*, 2001
- [3] Wikipedia. [Online] Τμήμα Μηχανικών Η/Υ & Πληροφορικής Πανεπιστήμιο Πατρών .
http://www.ceid.upatras.gr/courses/katanemhmena/wiki/index.php/TinyOS:Έκδοση_1.1.0.
- [4] D. Zeinalipour-Yazti, P. Andreou, P. Chrysanthis and G. Samaras. "MINT Views: Materialized In-Network Top-k Views in Sensor Networks". *The IEEE 8th International Conference on Mobile Data Management (MDM'07), Mannheim, Germany*. May 7 - 11.
- [5] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos and V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, D. Srivastava. "Finding the K Highest-Ranked Answers in a Distributed Network". *Computer Networks (ComNet)*. January 2009.
- [6] Zeinalipour-Yazti D., Vagena Z., Gunopulos D., Kalogeraki V., Tsotras V., Vlachos M., Koudas N., Srivastava D. "The Threshold Join Algorithm for Top-K Queries in Distributed Sensor Networks". *In VLDB's DMSN*. 2005, pp.61-66 .
- [7] Thorn, Jeff. "Deciphering TinyOS Serial Packets". [Online]
http://www.capsule.hu/local/sensor/documentation/deciphering_tinyOS_serial_packets.pdf
- [8] Brian Carter, Rammohan Ragade. "Message Transformation Services for Wireless Sensor Networks". *Engineering and Compute Science University of Louisville Louisvill*. Vols. (MTS-WSN) .
- [9] SHAO Ming-kai, HU Liang, HAO Xiao-qi. "Radio Communication Mechanism in TinyOS". *School of Computer Wuhan University, Wuhan, Hubei, China*. [Online]
<http://www.gs.whu.edu.cn/journal/nature/200404/20040425.pdf>

- [10] COMMON-Sense Net Working draft for MICA2 motes (Phase 1). *CEDT*. [Online] November 17, 2004.
<http://commonsense.epfl.ch/Admin/Documents/Mica2TechnicalOverview.pdf>.
- [11] Arora, Anish. "Introduction to Wireless Sensor Networks". [Online] www.cse.ohio-state.edu/~anish/788Notes/Lecture2.ppt.
- [12] TinyOS Documentation . [Online] <http://www.tinyos.net/tinyos-1.x/doc>.
- [13] Rutter, Paul William. "*Data Gathering in Wireless Sensor Networks: Dissertation*". May 2007.
- [14] Sam Madden, Joe Hellerstein, Wei Hong. "TinyDB: In-Network Query Processing in TinyOS". September, 2003, Vol. Version 0.4 .
- [15] Samuel R.,Madden Massachusetts,Michael J. Franklin,Joseph M. Hellerstrein . "TinyDB: An Acquisitional Query Processing System for Sensor Networks¹". *UC Berkeley And WEI HONG Intel Research*, 2003
- [16] Philip Levis,Nelson Lee . "TOSSIM: A Simulator for TinyOS Networks". [Online] <http://www.cs.berkeley.edu/~pal/pubs/nido.pdf>.
- [17] TOSSIM .[Online] <http://www.cs.berkeley.edu/~pal/research/tossim.html>.
- [18] P. Levis, N. Lee, M. Welsh, D. Culler. "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications". [Online] In Proceedings of the 1st international conference on Embedded networked sensor systems, Los Angeles, California . pp. 126-137.
- [19] V. Shnayder et al. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proc. of Sen- Sys*, Nov. 2004.
- [20] Installation instructions for PowerTOSSIM [Online] <http://www.eecs.harvard.edu/~shnayder/ptossim/install.html>.
- [21] Fagin R, Lotem A., Naor M. "Optimal Aggregation Algorithms For Middleware". In *ACM PODS*. [Online] pp.102-113, 2001.
- [22] Bruno N., Gravano L., Marian A. "Evaluating Top-K Queries Over Web Accessible Databases". [Online] In ICDE. pp.319-362, 2004.Marian A., Gravano L., Bruno N., "Evaluating Top-k Queries over Web-Accessible Databases", In *TODS*, Vol.29, No.2, pp.319-362, 2004.
- [23] Marian A., Gravano L., Bruno N. "Evaluating Top-k Queries over Web-Accessible Databases". In *TODS*. No.2, 2004, Vol. Vol.29, pp.319-362.

- [24] Babcock B., Olston C., "Distributed Top-K Monitoring". In *ACM SIGMOD*. 2003, pp.28-39
- [25] Balke W.-T., Nejdl W., Siberski W., Thaden U. "Progressive Distributed Top-K Retrieval in Peer-to-Peer Networks". In *ICDE*. pp.174- 185, 2005.
- [26] Cao P. , Wang Z. . "Efficient Top-K Query Calculation in Distributed Networks". In *ACM PODC* . 2004, pp.206-215.
- [27] Zeinalipour-Yazti D., Vagena Z., Gunopulos D., Kalogeraki V., Tsotras V., Vlachos M., Koudas N., Srivastava D., "The Threshold Join Algorithm for Top-K Queries in Distributed Sensor Networks". In *VLDB's DMSN*. pp.61-66, 2005.
- [28] Zeinalipour-Yazti D., Lin S., Gunopulos D. "Distributed Spatio- Temporal Similarity Search". In *ACM CIKM*. 2006, pp. 14-23.
- [29] D. Gay, P. Levis, R. Von Behren, M.Welsh, E. Brewer, D. Culler. "The nesC Language: A Holistic Approach to Networked Embedded Systems". In *ACM PLDI*. 2003., pp.1-11.
- [30] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister. "System Architecture Directions for Networked Sensors" . In *ACM SIGOPS Operating Systems Review*. No.5, 2000, Vol. Vol.34, pp.93-104.
- [31] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, D. Culler. "An Analysis of aLarge Scale Habitat Monitoring Application". In *ACM SenSys* . 2004.
- [32] Earth Climate and Weather. *University of Washington*. [Online] <http://www.k12.atmos.washington.edu/k12/grayskies/>.
- [33] Intel Lab Data . [Online] <http://db.csail.mit.edu/labdata/labdata.html>.
- [34] H. Thomas, S. Yi, H.D. Sherali. "Rate allocation in Wireless Sensor Networks with Network Lifetime Requirement". In *The ACM International Symposium on Mobile Ad Hoc Networking and Computing, Tokyo, Japan*. [Online] 24-26, 2004, pp. 67-77.
- [35] J. Polastre, R. Szewczyk, D.E. Culler. "TELOS: Enabling Ultra-low Power Wireless Research". In *Fourth International Symposium on Information Processing in Sensor Networks*. [Online] Los Angeles, CA, USA, 1, Apr 25-27, 2005. pp. 364-369.
- [36] Andreou, D. Zeinalipour-Yazti, M. Vassiliadou, P. K. Chrysanthis, G. Samaras "KSpot: Effectively Monitoring the K Most Important Events in a Wireless Sensor

Network", 25th International Conference on Data Engineering March (ICDE'09)
(demo), Shanghai, China, May 29 - April 4, 2009

Παράρτημα Α

Τεχνικό Υπόβαθρο : Το λειτουργικό σύστημα TinyOS

Τα ασύρματα δίκτυα αισθητήρων συνδυάζουν δυνατότητες επεξεργασίας, αίσθησης (sensing) και επικοινωνίας, σε ενσωματωμένες συσκευές. Στη συνέχεια τα πρωτόκολλα επικοινωνίας συνδυάζουν κατάλληλα τις ανεξάρτητες συσκευές, για τη δημιουργία ενός διασυνδεδεμένου δικτύου (mesh network), όπου τα δεδομένα δρομολογούνται ανάμεσα σε όλους τους κόμβους. Στο κεφάλαιο αυτό παρουσιάζεται η πλατφόρμα που χρησιμοποιείται για αυτή την εφαρμογή δικτύου αισθητήρων, όπου οι συσκευές έχουν διαστάσεις από μερικά χιλιοστά μέχρι το μέγεθος ενός υπολογιστή παλάμης και έπειτα γίνεται ανάλυση του λειτουργικού συστήματος που χρησιμοποιήθηκε.

Όπως είναι γνωστό τα πρώτα ασύρματα ενσωματωμένα συστήματα αισθητήρων έτρεχαν πάνω σε προσωπικούς υπολογιστές και χρησιμοποιούσαν κυρίως προγράμματα Linux. Όταν η ανάπτυξη αυτών των δικτύων πέρασε από τους μικροεπεξεργαστές (microprocessors) στους μικροελεγκτές (microcontrollers), το Linux δεν αποτελούσε πλέον την κατάλληλη επιλογή. Οι εφαρμογές των συστημάτων της εποχής εκείνης αναπτύσσονταν κυρίως σε τυπική γλώσσα C ή κατευθείαν σε γλώσσα assembly. Ο προγραμματισμός όμως σε αυτή τη γλώσσα είναι δύσκολο να αναλυθεί και επίσης μπορεί εύκολα να καταλήξει εκτός ελέγχου όταν αυξηθεί η πολυπλοκότητα της εφαρμογής.

Οπότε, οι ερευνητές του Πανεπιστημίου UC Berkeley ανέπτυξαν τη γλώσσα NesC, η οποία αντιπροσωπεύει ένα νέο πεδίο για τους σχεδιαστές εφαρμογών. Είναι κατάλληλα σχεδιασμένη για ενσωματωμένα συστήματα δικτύων και υποστηρίζει ένα προγραμματιστικό μοντέλο που ενσωματώνει αντιδραστικότητα με το περιβάλλον, ταυτοχρονισμό και δυνατότητα επικοινωνίας. Οι εφαρμογές των αισθητήρων (motes) είναι βαθιά συνδεδεμένες στο υλικό και κάθε μονάδα τρέχει μια εφαρμογή κάθε φορά. Αυτή η προσέγγιση αποφέρει τρεις σημαντικές ιδιότητες. Πρώτον, όλοι οι πόροι θεωρούνται στατικοί. Δεύτερον, αντί να χρησιμοποιείται μιας γενικής εξυπηρέτησης λειτουργικό σύστημα, οι εφαρμογές κατασκευάζονται από ένα σύνολο στοιχείων

συστήματος συσχετισμένων με συγκεκριμένο κώδικα. Τρίτον, τα όρια υλικού/λογισμικού εξαρτώνται από την εφαρμογή και την πλατφόρμα υλικού που χρησιμοποιείται και είναι σημαντικό να σχεδιάζονται για ευέλικτη αποδόμηση.

Στο τέλος αυτής της ενότητας δίνεται αναλυτική περιγραφή του τρόπου εγκατάστασης του λειτουργικού και οι απαραίτητες ρυθμίσεις που γίνονται τόσο στο υλικό όσο και στο λογισμικό.

Υλικό που χρησιμοποιήθηκε [7][8][9]

- **Processor and Radio Platform (MPR400)**

Το MPR400 βασίζεται στο Atmel ATmega128L. Το ATmega128L είναι χαμηλής ισχύος μικροελεγκτής ο οποίος τρέχει MoteWorks από την εσωτερική μνήμη flash.

- **Mica2 Mote (MPR400CB)**

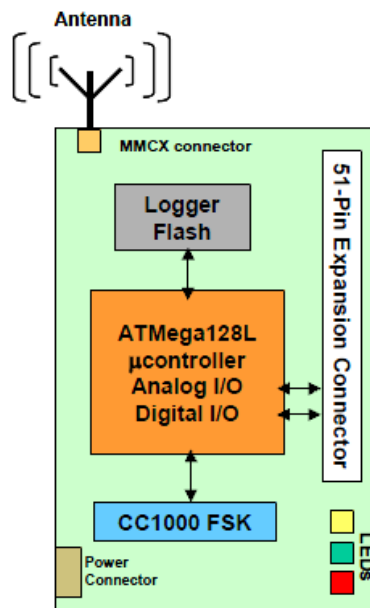


Συσκευή κόμβου αισθητήρα

Το Mica2 mote είναι 3ης γενιάς, tiny ασύρματη πλατφόρμα που έχει σχεδιαστεί ειδικά για ενσωματωμένα δίκτυα αισθητήρων. Η διάρκεια ζωής της μπαταρίας του φθάνει μέχρι το 1 έτος.

Το MICA2 Mote διαθέτει πολλές νέες βελτιώσεις σε σχέση με το αρχική MICA Mote. Τα ακόλουθα χαρακτηριστικά καθιστούν το MICA2 το καταλληλότερο για εμπορική ανάπτυξη: παρέχεται ασύρματη επικοινωνία και διαθέτει από 868 μέχρι 916 MHz συχνότητα για Multi-Channel πομποδέκτη με εκτεταμένο φάσμα. Επιπρόσθετα, υπάρχει επέκταση Connector για Φως, Θερμοκρασία, RH, Βαρομετρική Πίεση,

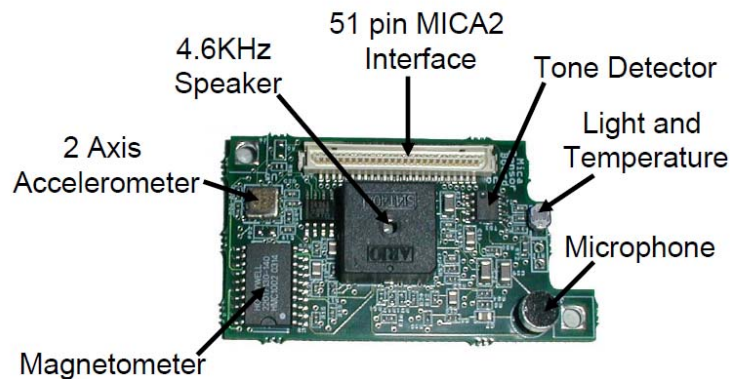
Επιτάχυνση / Σεισμική, Ακουστικές, Μαγνητικό και άλλα Crossbow Sensor Boards. Και τέλος, υποστηρίζει ασύρματο απομακρυσμένο επαναπρογραμματισμό.



MPR400 Block Diagram[1]

Το MICA2 διαθέτει επεξεργαστή χαμηλής ισχύος και ένα radio module που λειτουργεί σε 868/ MPR400 (915 MHz), MPR410 (433 MHz) ή MPR420 (315 MHz) για multi-channel πομποδέκτη που επιτρέπει τη διαβίβαση των στοιχείων με 38.4Kbits/s μέχρι 150 μέτρα. Τα Motes χρησιμοποιούν το Chipcon CC1000, FSK modulated radio.[1] Το MICA2 υποστηρίζεται από το λειτουργικό σύστημα TinyOS που επιτρέπει στους χρήστες να αναπτύξουν εφαρμογές που χρησιμοποιούν τη γλώσσα NESC που χρησιμοποιήθηκε για την ανάπτυξη του KSPOT client. Επιπλέον, το MICA2 υποστηρίζει σύνδεση επέκτασης για την τοποθέτηση των διαφόρων τμημάτων του αισθητήρα(sensors boards). Χρησιμοποιούμε το MTS310 sensor board που υποστηρίζει μια πλακέτα αισθητήρα με 2-Axis Accelerometer(Επιταχυνσιόμετρο), 2-Axis Magnetometer, φως, θερμοκρασία, μικρόφωνο, και speaker.

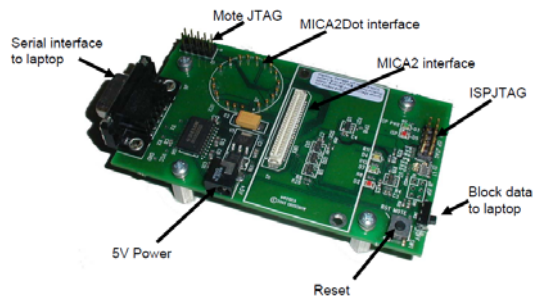
- **MTS300CA Sensor Board[6]**



Περιγραφή MTS310 sensor board[6]

Το Crossbow προσφέρει μια ποικιλία αισθητήρων και boards για την απόκτηση δεδομένων για το MICA2 Mote. Όλα αυτά τα boards συνδέονται με το MICA2 μέσω των 51-pin συνδέσεων. Το MICA2 51-pin υποδοχή (connector) υποστηρίζει Analog Inputs, Digital I / O, I2C, SPI και UART διασυνδέσεις. Αυτές οι διεπαφές καθιστούν πιο εύκολη τη σύνδεση με μια ευρεία ποικιλία εξωτερικών περιφερειακών.

- **Programming Board (MIB510)[6]**



Περιγραφή MIB510 Programming Board[6]

Ένα base station επιτρέπει την ομαδοποίηση των δεδομένων του δικτύου αισθητήρων σε ένα PC ή άλλη πλατφόρμα υπολογιστή. Κάθε MICA2 Mote μπορεί να λειτουργήσει ως σταθμός βάσης, όταν είναι συνδεδεμένο με μια τυπική διασύνδεση υπολογιστή ή το gateway board. Το MIB510/MIB520 Programming Board παρέχει ένα σειριακό-USB interface για τον προγραμματισμό και την επικοινωνία δεδομένων.

Λειτουργικό Σύστημα

Όπως αναφέρθηκε τα βασικά υλικά συστατικά των motes είναι οι αισθητήρες, ο μικροελεγκτής, η μνήμη, ο πομποδέκτης και η μπαταρία. Οπότε, το υλικό τους μπορεί να οργανωθεί ώστε να λειτουργεί σωστά και αποδοτικά για μια πληθώρα εφαρμογών. Για να λειτουργήσει όμως, ο κάθε κόμβος χρειάζεται ένα λειτουργικό σύστημα που να μπορεί να ελέγχει το υλικό, και να γεφυρώνει την επικοινωνία ανάμεσα στις εφαρμογές και το υλικό. Το TinyOS είναι το λειτουργικό σύστημα που είναι ειδικά σχεδιασμένο για ασύρματα ενσωματωμένα συστήματα και αυτό χρησιμοποιήθηκε στη συγκεκριμένη εφαρμογή.

Απαιτήσεις στη σχεδίαση Λειτουργικών Συστημάτων για ασύρματα δίκτυα αισθητήρων

Ο στόχος των λειτουργικών συστημάτων για ασύρματα δίκτυα αισθητήρων είναι η διαχείριση των διεργασιών, της μνήμης, του χρόνου του επεξεργαστή, του συστήματος αρχείων και των συσκευών. Συνήθως, αυτό επιτυγχάνεται με τον διαχωρισμό σε πολλά επίπεδα, όμως, τα ασύρματα δίκτυα αισθητήρων λόγω της ιδιομορφίας των εφαρμογών που εκτελούν και των περιορισμένων πόρων συστήματος, απαιτούν μια διαφορετική προσέγγιση στη σχεδίαση του λειτουργικού τους συστήματος.

Σε ένα κανονικό λειτουργικό σύστημα η εκτέλεση πολλαπλών διεργασιών επιτυγχάνεται μέσω της κατανομής της μνήμης (στοίβα) για κάθε διεργασία ξεχωριστά. Σε έναν ασύρματο κόμβο το περιορισμένο μέγεθος μνήμης δεν επαρκεί για μια τέτοια διαδικασία καθώς και η εναλλαγή στην εκτέλεση των διεργασιών⁶ δεν ευνοεί την εκτέλεση κάποιων εφαρμογών πραγματικού χρόνου (real-time) που εμφανίζονται σε ένα ασύρματο δίκτυο αισθητήρων, με αποτέλεσμα να χάνονται σημαντικά δεδομένα. Επιπρόσθετα, ένα κανονικό λειτουργικό σύστημα αξιοποιεί το σύνολο των δυνατοτήτων του υλικού για την καλύτερη εκτέλεση των προγραμμάτων και αυτό αποτελεί μια λογική προσέγγιση για έναν υπολογιστή με τροφοδοσία από το δίκτυο, για ένα σύστημα που τροφοδοτείται από δυο μικρές μπαταρίες το οποίο καταναλώνει πολλή ενέργεια .

Οπότε ένα λειτουργικό σύστημα για ασύρματα δίκτυα πρέπει να ικανοποιεί τις πιο κάτω προϋποθέσεις:

- 1) Μικρή έκταση κώδικα εξαιτίας της περιορισμένης μνήμης ενός κόμβου. Συνεπώς, ο πυρήνας του λειτουργικού πρέπει να υλοποιείται από τον ελάχιστο δυνατό κώδικα.
- 2) Διαχείριση των πόρων του συστήματος και της ενεργείας. Ο χρόνος στον επεξεργαστή και στη μνήμη πρέπει να κατανέμεται κατάλληλα για την ομαλή λειτουργία και τη τήρηση της προτεραιότητας στην εκτέλεση των διεργασιών. Η διαχείριση της ενέργειας πρέπει να αποσκοπεί στην μεγιστοποίηση της ζωής ενός κόμβου, κάνοντας σωστή χρήση των ενεργειακών καταστάσεων του επεξεργαστή (π.χ. τοποθέτηση σε idle ή sleep κατάσταση όταν αυτό είναι δυνατό).
- 3) Παροχή διεπαφής προγράμματος εφαρμογής (Application Program Interface).
- 4) Δυνατότητα επαναπρογραμματισμού. Σε ένα WSN είναι συχνό φαινόμενο είναι ο επαναπροσδιορισμός της λειτουργίας ενός κόμβου, είτε όσον αφορά τη συλλογή δεδομένων και τη συμπεριφορά του στο δίκτυο, είτε όσον αφορά την προσαρμογή της κατανάλωσης ενέργειας.
- 5) Η σχεδίαση πρέπει να αποτελείται από υπομονάδες (modular) για να μπορούμε να φτιάξουμε γρήγορα και εύκολα εφαρμογές.

TinyOS 1.x [5]

Το TinyOS 1.x είναι ένα νέο είδος λειτουργικού συστήματος που αποτελεί ένα από τα πρώτα περιβάλλοντα λογισμικού που έχουν σχεδιαστεί για να ικανοποιήσουν τις απαιτήσεις των κατανεμημένων ενσωματωμένων συστημάτων που έχουν περιορισμένους πόρους. Σχεδιάστηκε στο Πανεπιστήμιο του Berkeley και από τότε έχει γίνει ένα από τα πιο δημοφιλέστερα λειτουργικά συστήματα για ασύρματα δίκτυα αισθητήρων. Μέχρι στιγμής, υποστηρίζει διάφορα είδη πλατφόρμων υλικού, εμπορικές και μη, με επεξεργαστές από 8 bit αρχιτεκτονική και μόλις 2 KB RAM, μέχρι επεξεργαστές αρχιτεκτονικής 32-bit και μνήμης άνω των 32 MB. Μερικές από τις πλατφόρμες είναι το mica2 και MicaZ της εταιρίας Crossbow, το iMote της Intel, το Eyes από κοινοπραξία Ευρωπαϊκών πανεπιστημίων και εταιριών.

Το TinyOS είναι ανοικτού κώδικα (open source) λειτουργικό σύστημα και αυτό δίνει πολλές δυνατότητες στο χρήστη όσον αφορά τον έλεγχο και το χειρισμό του υλικού.

Είναι βασισμένο στην αρχιτεκτονική των components, η οποία κάνει δυνατή τη γρήγορη υλοποίηση των εφαρμογών ενώ ταυτόχρονα ελαχιστοποιεί το μέγεθος του κώδικα που απαιτείται.

Τα components του TinyOS περιέχουν πρωτόκολλα δικτύου, καταναεμημένες υπηρεσίες, οδηγούς αισθητήρων και εργαλεία ανάκτησης δεδομένων τα οποία μπορούν να χρησιμοποιηθούν ως έχει ή μπορούν να τροποποιηθούν.

Επιπρόσθετα, προσφέρει ευκολία και ευελιξία στη δημιουργία εφαρμογών ασύρματων δικτύων αισθητήρων παρά τις περιορισμένες δυνατότητες των motes. Γενικά παρέχει ένα σύνολο από βασικές υπηρεσίες οι οποίες μπορούν να μας φανούν χρήσιμες για την υλοποίηση καταναεμημένων αλγορίθμων. Το μοντέλο εκτέλεσης των διεργασιών είναι event-driven και εξαιτίας αυτού έχουμε τη δυνατότητα να επιτύχουμε καλύτερη διαχείριση της ενέργειας μέσω της ευελιξίας που παρέχεται στον χρονοπρογραμματισμό (scheduling) των διεργασιών.

Το TinyOs λοιπόν, έχει πλέον εισαχθεί σε αρκετές πλατφόρμες και σε αμέτρητες πλακέτες αίσθησης (sensor boards). Επιπλέον, χρησιμοποιείται και σε εξομοιωτές όπως είναι ο TOSSIM, για τη δημιουργία και δοκιμή των αλγορίθμων και πρωτοκόλλων.

Αρχικά το σύστημα TinyOS είχε γραφτεί σε γλώσσα C όμως δεν μπορούσε να υποστηρίξει αποτελεσματικά τις εφαρμογές των δικτύων αισθητήρων. Οπότε το TinyOS, οι βιβλιοθήκες, και οι εφαρμογές του είναι γραμμένες σε NESC [2][3][4], μια νέα γλώσσα προγραμματισμού δομημένη για component-based εφαρμογές. Ο μεταγλωττιστής της nesC καθώς και τα απαραίτητα εργαλεία Atmel AVR binutils είναι γραμμένα σε C. Το λειτουργικό περιέχει επίσης ενσωματωμένες κάποιες βοηθητικές εφαρμογές σε Java. Η εγκατάσταση του γίνεται σε ένα host PC, (με λειτουργικό Linux ή Windows XP με την προσθήκη του Unix προσομοιωτή, Cygwin), μέσω της σειριακής ή της USB θύρας του οποίου, γίνεται ο προγραμματισμός των κόμβων και η συλλογή των δεδομένων.

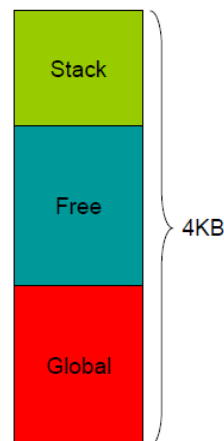
Η NESC είναι βασισμένη στην αρχή της σύνδεσης (wiring) μεταξύ των components για τη δημιουργία μιας εφαρμογής. Η γλώσσα NESC προορίζεται κυρίως για τα ενσωματωμένα συστήματα, όπως τα δίκτυα αισθητήρων. Η NESC έχει παρόμοια σύνταξη με τη γλώσσα C. Το TinyOS ορίζει ορισμένες σημαντικές έννοιες που εκφράζονται σε NESC. Πρώτο, οι εφαρμογές σε NESC έχουν κατασκευαστεί από συστατικά (components) με καθορισμένες διεπαφές (interfaces) διπλής κατεύθυνσης. Και δεύτερο, ο NESC συγχρονισμός ορίζει ένα μοντέλο, με βάση τα tasks και τα hardware event handlers, και ανακαλύπτει τα data races κατά τη μεταγλώττιση.

Κατά τη διάρκεια εκτέλεσης μιας διαδικασίας σε κάθε λειτουργικό σύστημα, η μνήμη για τη στοίβα κατανέμεται για την αποθήκευση των ενεργών εγγράφων και των τοπικών μεταβλητών. Αυτό γίνεται για κάθε μια ξεχωριστή διεργασία. Ωστόσο, από τη στιγμή που οι ασύρματες συσκευές έχουν περιορισμένο μέγεθος μνήμης, οι σχεδιαστές του TinyOs έχουν υιοθετήσει ένα νέο μοντέλο εκτέλεσης για την ελαχιστοποίηση του μεγέθους της μνήμης που χρησιμοποιείται για την εκτέλεση. Οι εφαρμογές στο TinyOs απαρτίζονται από πολλαπλές μικρές διεργασίες οι οποίες μοιράζονται όλες μια στοίβα. Λόγω αυτού του σχεδιασμού, μια διεργασία θα πρέπει να εκτελείται μέχρι να τελειώσει (run-to-completion) πριν δοθεί ο επεξεργαστής και η στοίβα σε μια άλλη διεργασία. Οι διεργασίες αυτές μπορούν να διακοπούν από διαχειριστές γεγονότων υλικού, οι οποίοι και αυτοί τρέχουν μέχρι να τελειώσουν, αλλά πριν δώσουν πίσω τη στοίβα πρέπει να σώσουν τις μεταβλητές κατάστασης στη γενική μνήμη.

Τώρα όταν λέμε ότι ένα mote τρέχει TinyOs εννοούμε ότι έχει εγκατεστημένο στη flash μνήμη του ένα binary εκτελέσιμο image με τις βιβλιοθήκες του TinyOs που χρειάζονται και είναι συνδεδεμένες με την εφαρμογή που θέλουμε να εκτελέσουμε.

Μοντέλο μνήμης δεδομένων[6]

- Στατική δέσμευση μνήμης
- Καθολικές μεταβλητές
- Τοπικές μεταβλητές



Το πλαίσιο μνήμης είναι πάντα σταθερού μεγέθους και κατανέμεται στατικά. Ορίζει τις απαιτήσεις σε μνήμη ενός συστατικού (component) κατά τη διάρκεια της μεταγλώττισης και έτσι εξασφαλίζει ότι δεν θα υπάρχουν overheads που είναι χαρακτηριστικό πρόβλημα της δυναμικής κατανομής μνήμης.

Αρχιτεκτονική TinyOS

Η αρχιτεκτονική του TinyOS περιλαμβάνει ένα χρονοπρογραμματιστή (scheduler) και ένα σύνολο από συστατικά (components) που αποτελούν το λογισμικό. Επίσης, στο TinyOS όπως έχουμε αναφέρει δεν υπάρχει καθόλου η έννοια της διεργασίας (process) αλλά του συστατικού (component). Δεν υπάρχει άρα η έννοια του πυρήνα (kernel) οπότε πρέπει να γίνει απευθείας διαχείριση του υλικού καθώς υπάρχει μόνο μια διαδικασία στο σύστημα. Συνεπώς, υπάρχει γραμμικός χώρος διευθύνσεων και η μνήμη ανατίθεται στατικά την ώρα που μεταγλωττίζεται η εφαρμογή στον υπολογιστή, οπότε το μέγεθος του TinyOs είναι μικρό και δεν ξεπερνά τα 3400 bytes.

Γενικά, το TinyOs από μόνο του δεν εκτελεί κάποια ιδιαίτερη λειτουργία και ούτε έχει κάποιο user interface (πχ shell στο unix) οπότε δεν έχει κανένα νόημα να το εγκαταστήσουμε μόνο του σε ένα mote.

Εργασίες και ταυτοχρονισμός βασισμένος σε γεγονότα (Tasks and event-based concurrency)

Το λειτουργικό σύστημα TinyOS εκτελεί μόνο ένα πρόγραμμα αποτελούμενο από επιλεγμένα στοιχεία συστήματος και στοιχεία κατασκευασμένα για την εφαρμογή που απαιτείται.

Υπάρχουν δυο μηχανισμοί εκτέλεσης: οι εργασίες (tasks) και οι χειριστές συμβάντων υλικού (hardware event handlers). Οι εργασίες είναι λειτουργίες των οποίων η εκτέλεση μετατίθεται χρονικά. Εκτελούνται μέχρι την ολοκλήρωσή τους και δεν προηγείται η μια της άλλης. Τα στοιχεία (components) μιας εφαρμογής μπορούν να θέσουν εργασίες προς εκτέλεση. Η χρήση αυτή των εργασιών από τα στοιχεία είναι

επιθυμητή και λογική όταν οι χρονικές απαιτήσεις δεν είναι τόσο αυστηρές. Αυτό περιλαμβάνει ουσιαστικά τις περισσότερες λειτουργίες εκτός από την επικοινωνία σε χαμηλό επίπεδο (low-level). Για να εξασφαλίσουμε χαμηλή καθυστέρηση κατά την εκτέλεση εργασιών, όλες οι ανεξάρτητες εργασίες πρέπει να είναι σχετικά σύντομες. Οι πολύπλοκες λειτουργίες θα έπρεπε να διασπώνται σε πολλαπλές μικρές εργασίες. Άλλωστε, οι απαιτήσεις των δικτύων αισθητήρων για μεγάλη χρονική διάρκεια ζωής απαγορεύουν βαρύ υπολογιστικό φορτίο. Οι χειριστές συμβάντων υλικού εκτελούνται σε απάντηση μιας διακοπής υλικού (hardware interrupt) και επίσης εκτελούνται μέχρι ολοκλήρωσης, αλλά μπορούν να προηγηθούν της εκτέλεσης μιας εργασίας ή ενός άλλου χειριστή συμβάντος υλικού. Λέγοντας διακοπή υλικού ονομάζουμε ένα εξωτερικό συμβάν το οποίο προκαλεί τον επεξεργαστή να ξεκινήσει να εκτελεί μια προκαθορισμένη λειτουργία. Ο επεξεργαστής διαθέτει ένα πίνακα με διευθύνσεις λειτουργιών, μια για κάθε πιθανή διακοπή και αναμένει τον μεταγλωττιστή να συμπληρώσει κώδικα στην αρχή του μεταγλωττισμένου προγράμματος, ο οποίος θα συμπληρώνει κατάλληλα αυτόν τον πίνακα ανάλογα με το πρόγραμμα. Τα συμβάντα υποδηλώνουν είτε ολοκλήρωση μιας λειτουργίας διαχωρισμένης σε φάσεις είτε ένα άλλο συμβάν από το περιβάλλον (όπως η λήψη ενός μηνύματος ή η παρέλευση ενός χρονικού διαστήματος) .

Λειτουργίες διαχωρισμένες σε φάσεις (Split-phase operations)

Επειδή οι εργασίες εκτελούνται χωρίς τη δυνατότητα η μια να προηγηθεί της άλλης, το TinyOS δεν έχει λειτουργίες φραγμού. Όλες οι λειτουργίες με μεγάλη καθυστέρηση εκτελούνται σε φάσεις: η αίτηση μιας λειτουργίας και η ολοκλήρωση της εκτελούνται ξεχωριστά. Οι εντολές (commands) είναι τυπικά αιτήσεις για εκτέλεση μιας λειτουργίας. Εάν η λειτουργία αυτή είναι διαχωρισμένη σε φάσεις, τότε η εντολή απαντά αμέσως, ενώ η ολοκλήρωσή της θα σηματοδοτήσει ένα συμβάν. Οι λειτουργίες που δε διαχωρίζονται σε φάσεις, όπως για παράδειγμα το άναμμα της λάμπας (led), δεν έχουν συμβάν ολοκλήρωσης. Ένα τυπικό παράδειγμα μιας λειτουργίας διαχωρισμένης σε φάσεις είναι η ασύρματη αποστολή ενός πακέτου. Κάποιο στοιχείο της εφαρμογής μπορεί να καλέσει την εντολή αποστολής (send) ώστε να ξεκινήσει την ασύρματη μετάδοση ενός μηνύματος, ενώ το στοιχείο που υλοποιεί την επικοινωνία σηματοδοτεί το συμβάν ολοκλήρωση αποστολής (sendDone) μόλις η μετάδοση του μηνύματος έχει

ολοκληρωθεί. Κάθε στοιχείο υλοποιεί το μισό της συνολικής διφασικής λειτουργίας και καλεί το άλλο. Η διασύνδεση (wiring) συνδέει τόσο τις εντολές όσο και τα συμβάντα με κατάλληλο τρόπο ώστε να δράσουν από κοινού και ολοκληρωμένα.

Ο ανταγωνισμός των πόρων διαχειρίζεται μέσω απόρριψης σύγχρονων αιτήσεων. Στο παραπάνω παράδειγμα, εάν το στοιχείο που υλοποιεί την επικοινωνία δεν καταφέρει να χειριστεί πολλαπλές ταυτόχρονες λειτουργίες αποστολής, τότε σηματοδοτεί ένα μήνυμα λάθους όταν επιχειρηθεί μια ταυτόχρονη αποστολή. Εναλλακτικά, το στοιχείο που υλοποιεί την επικοινωνία θα μπορούσε να θέσει σε αναμονή την αίτηση για αποστολή προς μελλοντική επεξεργασία. Το απλό μοντέλο ταυτοχρονισμού του TinyOS επιτρέπει υψηλό ταυτοχρονισμό με μικρό φόρτο σε αντίθεση με το μοντέλο ταυτοχρονισμού βασισμένο σε νήματα στο οποίο οι στοίβες νημάτων καταναλώνουν πολύτιμη μνήμη. Παρόλα αυτά, όπως και κάθε ταυτόχρονο σύστημα, ο ταυτοχρονισμός και ο μη ντετερμινισμός μπορεί να είναι η αιτία πολύπλοκων προβλημάτων (bugs), συμπεριλαμβανομένου και του αδιεξόδου (deadlock).

Προγραμματισμός στο TinyOS [2][3][4][6]

Η αρχική ιδέα των σχεδιαστών του TinyOS, ήταν να χρησιμοποιηθεί η C ως γλώσσα προγραμματισμού. Στην πορεία όμως προέκυψε ότι για τη σωστή υποστήριξη του μοντέλου λειτουργίας του TinyOS, απαραίτητος ήταν ο σχεδιασμός μιας νέας γλώσσας, η οποία ονομάστηκε nesC (network embedded systems C). Η nesC αποτελεί διάλεκτο της C. Η ουσιαστική τους διαφορά δεν έγκειται στο πως συντάσσεται ο κώδικας ενός λογισμικού συστατικού (software component), αλλά στο πως συνδέεται με αλλά ώστε να δημιουργηθεί μια λειτουργική εφαρμογή, χρησιμοποιώντας μια διαδικασία που αποκαλείται διασύνδεση (wiring). Ας δούμε όμως ποια είναι τα συστατικά και οι έννοιες που συνθέτουν ένα πρόγραμμα στη nesC.

Commands

Οι εντολές(commands) είναι αιτήσεις (requests) προς τα συστατικά (components) χαμηλού επιπέδου. Η εκτέλεση τους είναι άμεση και χωρίς δυνατότητα φραγής, για αυτό και ολοκληρώνεται σε πολύ μικρό χρονικό διάστημα. Έχουν τη δυνατότητα να καλούν άλλα commands χαμηλού επιπέδου, tasks, αλλά δεν μπορούν να σηματοδοτούν

συμβάντα (events). Μαζί με τα events αποτελούν τον μηχανισμό επικοινωνίας μεταξύ των συστατικών (components).

Components

Η χρήση των συστατικών (components) για τη δημιουργία των εφαρμογών, είναι ένα ακόμη ξεχωριστό κομμάτι του Tinyos, έναντι της ανάπτυξης βιβλιοθηκών με συναρτήσεις οι οποίες μπορούν να κληθούν από τα προγράμματα του χρήστη. Τα συστατικά είναι κομμάτια κώδικα, τα οποία έχουν ορισμένες διεπαφές για είσοδο και έξοδο. Για να μπορεί να παρέχεται μια διεπαφή, πρέπει στο συστατικό να υλοποιηθεί μια σειρά από εντολές από τις διεπαφές. Για να μπορέσει όμως να χρησιμοποιηθεί μια διεπαφή, ένα συστατικό πρέπει να υλοποιεί μια σειρά από συναρτήσεις, τα λεγόμενα συμβάντα (events). Τα συστατικά πρέπει να είναι μικρά έτσι ώστε να έχουμε γρήγορη εκτέλεση χωρίς απώλειες.

Events

Τα συμβάντα (events) χρησιμοποιούνται για την επικοινωνία μεταξύ των συστατικών (components), κάτι σαν διακοπή λογισμικού (software interrupt) δηλαδή οι αντίστοιχοι χειριστές (handlers) που υπάρχουν μέσα στα συστατικά μεταβάλλουν την εσωτερική κατάσταση των συστατικών τους. Υπάρχουν δύο είδη:

- (1) Εξωτερικά: προκαλούνται από διακοπές υλικού (hardware interrupts), από το χρονοδιακόπτη (timer) και τον πομποδέκτη.
- (2) Εσωτερικά: προκαλούνται από χειριστές συμβάντων (event handlers) μέσα στα συστατικά τα οποία αρχικά ξύπνησαν από κάποιο εξωτερικό συμβάν και στη συνέχεια έστειλαν ένα συμβάν σε κάποιο άλλο συστατικό.

Η εφαρμογή συνδέει μόνο τα συστατικά τα οποία χρειάζονται για να λειτουργήσει και έτσι στην απεικόνιση (image) που φορτώνουμε στη flash περιέχονται μόνο αυτά και όχι όλα τα συστατικά του συστήματος.

Όπως αναφέρθηκε πιο πάνω οι εντολές (commands) ορίζονται ως αιτήσεις προς τα συστατικά για την έναρξη μιας λειτουργίας ενώ τα αντίστοιχα συμβάντα σηματοδοτούν τη λήξη τους. Για παράδειγμα, μια εντολή μπορεί να ζητήσει από έναν αισθητήρα την έναρξη συλλογής δεδομένων. Ένα συμβάν θα προκληθεί όταν ο αισθητήρας ολοκληρώσει τη συλλογή μετρήσεων και είναι έτοιμος να τις στείλει στις εισόδους που

είναι συνδεδεμένος. Επίσης, ένα συμβάν μπορεί να εμφανιστεί και ασύγχρονα εξαιτίας μιας διακοπής (interrupt) του υλικού. Τα συμβάντα όπως και οι εντολές δεν υπόκεινται σε φραγή εκτέλεσης.

Με τη χρήση εντολών και συμβάντων, επιτυγχάνεται η εκτέλεση των λειτουργιών του λογισμικού σε δυο φάσεις (split phase operation), όπως ακριβώς συμβαίνει και με το υλικό. Μετά από την αίτηση εκτέλεσης μιας λειτουργίας η εντολή επιστρέφει αμέσως και το αντίστοιχο συμβάν σηματοδοτεί το τέλος της λειτουργίας μετά τη μεσολάβηση κάποιου χρονικού διαστήματος. Τέλος τα συμβάντα έχουν δυνατότητα να καλέσουν χαμηλού επιπέδου εντολές, εργασίες καθώς και να σηματοδοτήσουν άλλα ανώτερου επιπέδου συμβάντα.

Tasks

Οι εργασίες (tasks) είναι non-preemptive. Αυτό σημαίνει ότι μόνο μια τρέχει οποιαδήποτε στιγμή, και το TinyOS δεν διακόπτει μια εργασία για να τρέξει άλλο. Εξαιτίας της ιδιότητας να μην διακόπτονται, άρα να είναι δεσμευτικά για τους πόρους του συστήματος, ο υπολογιστικός φόρτος που ανατίθεται στις εντολές και στα συμβάντα πρέπει να είναι ο ελάχιστος δυνατός. Σκοπός της ύπαρξης τους άλλωστε είναι να επιτελούν απαραίτητες λειτουργίες για το σύστημα. Όλα τα υπόλοιπα εμπίπτουν στη δικαιοδοσία των εργασιών, που αποτελούν μεγάλο κομμάτι των συστατικών. Οι εργασίες καλούνται μέσα από τα συστατικά ή τα συμβάντα, για να εκτελέσουν λειτουργίες χαμηλότερης προτεραιότητας σε κάποια στιγμή μετά την ολοκλήρωση των τρεχόντων συστατικών και συμβάντων. Μια εργασία μπορεί στο σώμα της να καλέσει μια χαμηλού επιπέδου εντολή, να σηματοδοτήσει ένα συμβάν ή ακόμα και μια καινούρια εργασία. Η εκτέλεση της είναι δυνατόν να διακοπεί από κάποιο ασύγχρονο συμβάν, αλλά όχι από άλλη εργασία ή συγχρονισμένο συμβάν. Οι εργασίες αντιπροσωπεύουν την δυνατότητα ενός είδους παράλληλης επεξεργασίας μέσα σε ένα συστατικό. Ως εκ τούτου, έχουν πρόσβαση μόνο σε δεδομένα που αφορούν ένα συγκεκριμένο συστατικό, στο οποίο ανήκουν. Μια δήλωση μιας εργασίας δίνεται πιο κάτω:

```
task void readDoneTask();
```

Ένα συστατικό (component) τροχοδομεί μια εργασία στον TinyOS χρονοπρογραμματιστή (scheduler) με τη δεσμευμένη λέξη post:
post readDoneTask();

Interfaces

Οι διεπαφές (interfaces) αποτελούν αναπόσπαστο κομμάτι ενός προγράμματος. Χρησιμεύουν στην απόδοση ενός αφηρημένου ορισμού της αλληλεπίδρασης μεταξύ των συστατικών. Οι διεπαφές μπορούν να παρέχονται ή να χρησιμοποιούνται από ένα συστατικό. Οι παρεχόμενες (provider), διεπαφές αντιπροσωπεύουν τις λειτουργίες που παρέχει ένα συστατικό ενώ τα χρησιμοποιούμενα (user), τις λειτουργίες που απαιτεί ένα συστατικό να του δοθούν για την εκτέλεση της λειτουργίας που του έχει ανατεθεί. Όπως συμβαίνει και στην Java, έτσι και εδώ μια διεπαφή απαγορεύεται να περιέχει κώδικα για εκτέλεση υπολογισμών. Το μόνο που μπορεί να περιέχει είναι απλές δηλώσεις συναρτήσεων (commands) που πρέπει να υλοποιούνται από το συστατικό που παρέχει (provides) τη διεπαφή και δηλώσεις συναρτήσεων (events) που πρέπει να υλοποιηθούν από το συστατικό που χρησιμοποιεί (uses) τη διεπαφή. Αυτός ο τρόπος χρήσης των διεπαφών, τους δίνει την ιδιότητα της διπλής κατεύθυνσης, που σχετίζεται με ότι αναφέρθηκε για το split-phase operation. Πιο συγκεκριμένα, ένα συστατικό δεν μπορεί να καλέσει μια εντολή αν δεν παρέχει μια υλοποίηση για τα συμβάντα που αυτή θα προκαλέσει στο τέλος της εκτέλεσης της. Έχουμε για παράδειγμα το παρακάτω interface :

```
interface Send{  
    command error_t send(message_t* msg,uint8_t len);  
    command void* getBuffer(TOS_MsgPtr msg, uint16_t* length);  
    event void sendDone(message_t* msg,error_t error);  
}
```

Το όνομα της διεπαφής είναι Send και σε αυτό δηλώνονται δυο εντολές (send, getBuffer) και ένα συμβάν(sendDone). Ένα συστατικό που παρέχει τη διεπαφή πρέπει οπωσδήποτε να περιέχει υλοποιήσεις των send και getBuffer, ενώ ένα συστατικό που θα κάνει χρήση του Send, πρέπει να υλοποιεί το συμβάν sendDone. Μια διεπαφή μπορεί να δέχεται παράμετρο. Αυτό σημαίνει ότι επιτρέπει σε ένα συστατικό να παρέχει πολλαπλά στιγμιότυπα της διεπαφής. Για παράδειγμα αν σε ένα συστατικό

υπάρχει η δήλωση ‘provides interface Timer[uint8_t id]’, σημαίνει ότι παρέχει 256 στιγμιότυπα του Timer, ένα για κάθε uint8_t τιμή.

Modules

Το module είναι ένα συστατικό που υλοποιεί ή χρησιμοποιεί μια ή και περισσότερες διεπαφές. Μέσα στον κώδικα του module βρίσκεται ο πυρήνας μιας εφαρμογής TinyOS. Εδώ καθορίζεται η λειτουργία που θέλουμε να εκτελέσει το πρόγραμμα μας.

Η δομή ενός τυπικού module έχει ως εξής :

```
module BlinkM {
  provides {
    interface StdControl as Control;
  }

  uses {
    interface Timer;
    interface Leds;
  }

  implementation {
    Bool state;
    command result_t Control.init() {
      state = FALSE;
      call Leds.init();
      return SUCCESS;
    }
    command result_t Control.start() {
      return call Timer.start(TIMER_REPEAT, 1000);
    }
    command result_t Control.stop() {
      return call Timer.stop();
    }
    event result_t Timer.fired()
    {
      state = !state;
      post processing();
      return SUCCESS;
    }
    task void processing()
    {
      if (state)
        call Leds.redOn();
      else
        call Leds.redOff();
    }
  }
}
```

Βλέπουμε ότι για τη διεπαφή Control που παρέχει το module BlinkM, υπάρχουν και οι αντίστοιχες υλοποιήσεις των εντολών που αυτό ορίζει. Από τη στιγμή που υπάρχουν στο module οι εντολές `init()`, `start()` και `stop()`, καταλαβαίνουμε χωρίς να έχουμε δει τη διεπαφή Control, ότι αυτές οι εντολές είναι δηλωμένες σε αυτό. Στο σώμα του `Control.init()`, υπάρχει η εντολή ‘call `Leds.init()`;’, η οποία σημαίνει ότι καλείται η εντολή `init()`, που δηλώνεται στη διεπαφή Leds και εκτελεί μια λειτουργία. Για αυτό και λέμε ότι το BlinkM χρησιμοποιεί τη διεπαφή Leds. Προφανώς, η `Leds.init()`, υλοποιείται σε κάποιο άλλο module που παρέχει τη διεπαφή Leds. Το BlinkM χρησιμοποιεί άλλη μια διεπαφή, το Timer. Το Timer περιέχει τη συνάρτηση συμβάντος `fired()`, για αυτό και στο BlinkM υπάρχει η αντίστοιχη υλοποίηση της. Επίσης στο BlinkM ορίζεται η εργασία `processing()`, η οποία όπως βλέπουμε καλείται μέσα στο συμβάν `Timer.fired()` με την εντολή `post`. Τέλος, να σημειώσουμε ότι σε ένα module εκτός από την κλήση εντολών με την εντολή `call`, μπορούν να προκληθούν τεχνητά συμβάντα με την εντολή `signal`. Για παράδειγμα, η `signal Timer.fired()`, θα προκαλούσε τεχνητά μια διακοπή του Timer.

Configurations

Τα modules είναι οι βασικές δομικές μονάδες μιας TinyOS εφαρμογής, όπου υλοποιείται η εκτελέσιμη λογική της εφαρμογής. Οι συναρτήσεις και οι μεταβλητές που μπορεί να χρησιμοποιήσει ένα module, είναι μόνο αυτές που έχουν οριστεί στο δικό του τοπικό χώρο ονομάτων (namespace). Με λίγα λόγια, ένα module από μόνο του δεν διαθέτει τρόπο να καλέσει κάποιο άλλο. Για να συμβεί κάτι τέτοιο πρέπει να αποτυπωθεί το σύνολο ονομάτων ενός συστατικού module μέσω μιας διασύνδεσης, στο σύνολο ονομάτων ενός άλλου συστατικού. Η διαδικασία σύνδεσης των συστατικών μεταξύ τους αποκαλείται διασύνδεση (wiring) όπως έχουμε αναφέρει και πιο πριν και υλοποιείται από ένα ξεχωριστό συστατικό που ονομάζεται διαμόρφωση (configuration). Ένα configuration όσον αφορά τη βασική δομή μοιάζει με ένα module. Χωρίζεται σε πεδίο για δήλωση (specification) και σε πεδίο για υλοποίηση (implementation). Συντακτικά τα configurations είναι πολύ απλά. Διαθέτουν μόλις τρεις τελεστές. Δεξί βέλος ($\rightarrow$), αριστερό βέλος ($\leftarrow$) και ίσον ($=$). Οι δυο πρώτοι εξυπηρετούν τη βασική διασύνδεση (wiring), η οποία είναι ουσιαστικά μια διαδικασία αντιστοίχισης διεπαφής ενός συστατικού σε διεπαφή ενός άλλου συστατικού. Η κατεύθυνση είναι πάντα από την πλευρά του χρήστη του συστατικού προς αυτή του παροχέα.

```
user.interface -> provider.interface  
provider.interface <- user.interface
```

Με αυτόν τον τρόπο ένα module που χρησιμοποιεί μια διεπαφή, βρίσκει το module στο οποίο υλοποιείται η διεπαφή μέσω της διασύνδεσης στο configuration. Πιο κάτω δίνεται ένα παράδειγμα για το configuration του module BlinkM.

```
configuration Blink {  
}  
implementation {  
  components Main, BlinkM, TimerC as Timer, LedsC;  
  Main.StdControl -> BlinkM.Control;  
  
  Main.StdControl -> Timer;  
  BlinkM.Timer -> Timer;  
  BlinkM.Leds -> LedsC;  
}
```

Στην πρώτη γραμμή απλά δηλώνεται το όνομα του configuration. Το πεδίο για specification είναι άδειο για την συγκεκριμένη εφαρμογή, οπότε ,αμέσως ακολουθεί το πεδίο υλοποίησης, το οποίο ξεκινά πάντα με την δεσμευμένη λέξη implementation. Η πρώτη σειρά της υλοποίησης ορίζει ποια components διασυνδέει το configuration. Αυτά είναι τα : Main, BlinkM, TimerC, LedsC. Η έκφραση TimerC as Timer, σημαίνει ότι από εδώ και πέρα, σε οποιοδήποτε σημείο του κώδικα, το component TimerC θα αναφέρεται ως Timer. Το TimerC περιγράφει τη λειτουργία ενός timer. Επειδή όμως, στο TinyOS δεν υπάρχει μόνο ένα είδος timer, αν θέλουμε η εφαρμογή να χρησιμοποιήσει κάποιον άλλο (π.χ. τον SingleTimer), η μόνη τροποποίηση που θα γίνει θα είναι μόνο σε αυτό το σημείο και όχι σε όλο τον κώδικα. Στις επόμενες σειρές γίνεται χρήση του τελεστή ->. Η μετάφραση των εκφράσεων έχει ως εξής: το component Main χρησιμοποιεί το interface StdControl που υλοποιείται και του παρέχεται ως Control, από το component BlinkM. Αξίζει να αναφέρουμε ότι το component Main, συνδέεται μέσω μιας διαδρομής με τον scheduler του TinyOS, για αυτό είναι και το πρώτο που καλείται σε κάθε εφαρμογή. Για την ακρίβεια, η συνάρτηση Main.StdControl.init() είναι η πρώτη που εκτελείται και την ακολουθεί η Main.StdControl.start(). Το interface StdControl χρησιμοποιείται για αρχικοποίηση των components που το παρέχουν, κάτι σαν reset. Έτσι γίνεται φανερό ότι τα components BlinkM και Timer που παρέχουν το StdControl στο Main, θα αρχικοποιηθούν με την έναρξη της εφαρμογής. Στη συνέχεια, ορίζεται ότι το BlinkM

χρησιμοποιεί τη διεπαφή Timer που παρέχεται από το Timer8, και το interface Leds που παρέχεται από το LedsC. Η έκφραση 'BlinkM.Leds -> LedsC' υπονοεί την 'BlinkM.Leds -> LedsC.Leds'. Με βάση τα όσα έχουν αναφερθεί θα περίμενε κανείς ότι το component TimerC είναι και αυτό module που υλοποιεί το Interface Timer. Εξετάζοντας όμως τον κώδικα του TimerC, παρατηρούμε ότι πρόκειται για configuration.

```
configuration TimerC {  
  provides {  
    interface StdControl;
```

Το ένα Timer είναι interface, ενώ το άλλο component. Τυχαίνει να έχουν το ίδιο όνομα, αλλά είναι τελείως διαφορετικές οντότητες.

```
interface Timer;  
}  
} implementation {  
  components TimerM, HWClock;  
  StdControl = TimerM.Stdcontrol;  
  Timer = TimerM.Timer;  
  TimerM.Clk -> HWClock.Clock;  
}
```

Πως όμως ένα configuration μπορεί και παρέχει διεπαφές. Σε αντίθεση με το προηγούμενο configuration, το πεδίο για specification του TimerC δεν είναι κενό, αλλά περιέχει δηλώσεις ότι παρέχει δυο διεπαφών (Timer και StdControl). Παρακάτω στην υλοποίηση δηλώνεται ότι το TimerC, διασυνδέει τα modules TimerM και HWClock μέσω του interface Clock.

Παρατηρούμε ότι στις δύο πρώτες δηλώσεις αντί για το βέλος, χρησιμοποιείται ο τρίτος τελεστής, το ίσον. Αυτό γίνεται επειδή θέλουμε να εξισώσουμε τις διεπαφές που παρέχει το TimerC στις διεπαφές που παρέχουν το TimerM. Δεν παρεμβάλλεται κάποιο συστατικό μεταξύ τους, οπότε είναι ουσιαστικά οι ίδιες διεπαφές. Με βάση όλα τα παραπάνω ας δούμε βήμα προς βήμα πως θα εξελιχθεί η εκτέλεση της εφαρμογής.

- Καλείται η συνάρτηση Main.StdControl.init()
- Μέσω της προηγούμενης καλούνται οι BlinkM.Control.init() και TimerC.Timer.init(). Μέσα από την πρώτη καλείται η LedsC.Leds.init(), οπότε όλα τα συστατικά έχουν αρχικοποιηθεί.
- Καλείται η Main.StdControl.start()

- Μέσω αυτής καλείται η `BlinkM.StdControl.start()`.
- Η προηγούμενη καλεί την `TimerC.Timer.start(TIMER_REPEAT,100)`. Ο `Timer` ξεκινά και κάθε 1000ms προκαλεί μια διακοπή.
- Μόλις συμβεί η διακοπή εκτελείται η `BlinkM.Timer.fired()`
- Μέσα στην προηγούμενη καλείται η εργασία `processing()`
- Το `processing()` καλεί την `LedsC.Leds.redOn()` ή την `LedsC.Leds.redOff`, ανάλογα με την τιμή της μεταβλητής `state`. Ανάβει ή σβήνει το κόκκινο led.
- Το `processing()` επιστρέφει και εκτελείται ο υπόλοιπος κώδικας της `BlinkM.Timer.fired()`, η οποία απλά επιστρέφει μια σταθερά.

Async συναρτήσεις και atomic statements

Πιο πάνω είδαμε ότι η εκτέλεση των εργασιών είναι non preemptive εκτός των περιπτώσεων που έχουμε εμφάνιση ασύγχρονων events, λόγω διακοπής που προέρχεται απευθείας από το υλικό (hardware interrupt handlers). Σε αυτή την περίπτωση η τρέχουσα εργασία διακόπτεται και εκτελείται το συμβάν. Οι συναρτήσεις που σχετίζονται με interrupt handlers του υλικού δηλώνονται με τη λέξη `async` μπροστά. Για παράδειγμα το interface `ADC`, που περιγράφει τον A/D Converter ορίζεται ως εξής:

```
interface ADC {
  async command result_t getData();
  async command result_t getContinuousData();
  async event result_t dataReady(uint16_t data);
}
```

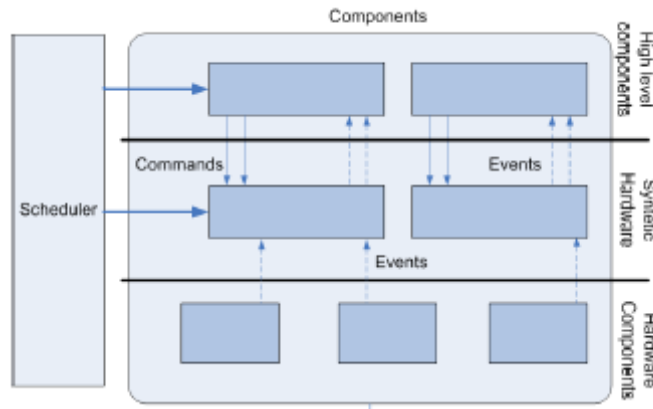
Καθώς η δειγματοληψία είναι μια καθαρά ασύγχρονη διαδικασία. Αυτού του είδους η ασύγχρονη εκτέλεση, μπορεί να προκαλέσει προβλήματα σε τμήματα κώδικα, όπου υπάρχουν εντολές που επεξεργάζονται μεταβλητές ελέγχου ή εντολές που καταχωρούν σε μεταβλητές ληφθέντα δεδομένα. Ας υποθέσουμε ότι εκτελείται ο παρακάτω κώδικας:

```
async event dataReady(uint16_t data){
  LightReadings=data;
  post Store();
  return SUCCESS ;
}
```

Το συμβάν `dataReady` σηματοδοτεί ότι στο A/D υπάρχουν δεδομένα (`data`), τα οποία λαμβάνονται και καταχωρούνται στην μεταβλητή `LightReadings`. Στη συνέχεια, καλείται μια εργασία που αποθηκεύει την τιμή στην EEPROM. Αν για κάποιο λόγο ένα καινούριο συμβάν `dataReady` εμφανιστεί πριν γίνει η καταχώρηση στην `LightReadings`, η τιμή που είχε μετρηθεί πρώτη δεν θα αποθηκευτεί ποτέ στην EEPROM καθώς θα καλυφθεί από την καινούρια. Τη λύση σε τέτοιες περιπτώσεις δίνει η χρήση της δήλωσης `atomic` πριν από κάθε εντολή που θέλουμε να τρέξει `non-preemptively`. Έτσι αν γράψουμε `atomic LightReadings=data`, η εντολή θα εκτελεστεί ότι και αν συμβεί. Τα `atomic statements` πρέπει να χρησιμοποιούνται προσεκτικά γιατί κατά την εκτέλεση τους απενεργοποιούνται οι `interrupts` οπότε η άσκοπη χρήση τους προκαλεί χάσιμο κύκλων του επεξεργαστή.

Scheduler

Ο χρονοπρογραμματιστής οργανώνει την λειτουργία των συστατικών. Χρησιμοποιεί έναν απλό `fifo` (`first in first out`) μηχανισμό και η φιλοσοφία λειτουργίας του γενικά είναι `non-preemptive`. Κάθε εργασία που καλείται τοποθετείται στη στοίβα του χρονοπρογραμματιστή. Αν δεν τρέχει κάποιο συμβάν, επιτρέπει την εκτέλεση των εργασιών που έχουν καταφτάσει, ένα κάθε φορά, διασφαλίζοντας την ολοκλήρωση τους. Ένα ασύγχρονο συμβάν έχει μεγαλύτερη σημασία από ένα `task`, οπότε αν υπάρχουν μόνο εργασίες στη στοίβα και ξαφνικά έρθει ένα συμβάν, μπορεί να διακοπεί η εκτέλεση της εργασίας και να αρχίσει η εκτέλεση του συμβάντος. Με άλλα λόγια, μια εργασία μπορεί να γίνει `preempted` από ένα συμβάν, δεν μπορεί να γίνει όμως το αντίθετο. Η εκτέλεση των συμβάντων είναι ατομική μεταξύ τους, όπως ατομική είναι και η εκτέλεση μεταξύ διαφορετικών εργασιών. Όταν η στοίβα αδειάσει ο χρονοπρογραμματιστής δίνει εντολή στον επεξεργαστή να μπει σε κατάσταση `sleep`. Τα περιφερειακά όμως παραμένουν ενεργά, οπότε μια διακοπή μπορεί να επανεκκινήσει την ίδια διαδικασία.



Δομή εφαρμογής στο TinyOS

Με τη χρήση αυτής της δομής, ένα συστατικό που θέλει να χρησιμοποιήσει τις εντολές μιας συγκεκριμένης διεπαφής, πρέπει να υλοποιήσει τα συμβάντα αυτής της εφαρμογής. Από τη στιγμή που μια σειρά από συστατικά έχει αναπτυχθεί, θα πρέπει να οργανωθούν έτσι ώστε να υλοποιηθεί η λειτουργία μιας εφαρμογής. Η διασύνδεση των συστατικών μεταξύ τους γίνεται με τη χρήση συστατικών διαμόρφωσης (configuration). Όταν ολοκληρωθεί, μια εφαρμογή του TinyOS μπορεί να αναπαρασταθεί ως ένα γράφημα στο οποίο οι ενώσεις των εντολών και των συμβάντων μεταξύ των συστατικών που υποδηλώνουν τα όρια του γράφου. Η δομή αυτή μπορεί να φανεί ως ένα δέντρο, όπου οι ρίζα είναι μια απλή διεργασία που εκτελείται και τα φύλλα είναι οι διαχειριστές των γεγονότων του υλικού.

Εγκατάσταση TinyOS και απαραίτητες Ρυθμίσεις (Installation and Configuration) [5]

Εγκατάσταση σε Windows XP

- Εγκατάσταση VMware-workstation-6.0
- Εγκατάσταση Windows XP στο VMware
- Εγκατάσταση cygwin <http://www.cygwin.com>
- Εγκατάσταση tinynos-1.1.11-3is.exe <http://www.tinyos.net/dist-1.1.0/tinyos/windows>
- Εγκατάσταση text editor

Setup environmental variables

```
cd /etc/profile.d/tinyos.sh
```

```
# script for profile.d for bash shells, adjusted for each users
# installation by substituting /opt for the actual tinyos tree
# installation point.
TOSROOT="/opt/tinyos-1.x"
export TOSROOT
TOSDIR="$TOSROOT/tos"
export TOSDIR
CLASSPATH=`$TOSROOT/tools/java/javapath`
export CLASSPATH="$CLASSPATH;./cygdrive/c/Program
Files/UCB/jdk1.4.1_02/j2sdk1.4.1_02/jre/lib/ext/comm.jar"
MAKERULES="$TOSROOT/tools/make/Makerules"
export MAKERULES
# Extend path for java
type java >/dev/null 2>/dev/null || PATH=`/usr/local/bin/locate-jre --java`:$PATH
type javac >/dev/null 2>/dev/null || PATH=`/usr/local/bin/locate-jre --javac`:$PATH
echo $PATH | grep -q /usr/local/bin || PATH=/usr/local/bin:$PATH
• Στο .bash_profile:
PATH=/opt/tinyos-1.x/tools/java/net/tinyos/sim:${PATH}:.
tools/java/net/tinyos/sim/Makefile
add row → net/tinyos/message/avrmote/*.class
```

• Στο tools/Make δημιουργώ ένα Makelocal file και δίνει groupID και συχνότητα για να μπορούν τα motes να επικοινωνούν.

```
# Makelocal File
PFLAGS += -I%/T/./beta/MyBetaCode
DEFAULT_LOCAL_GROUP = 0x33
PFLAGS += -DCC1K_DEF_FREQ=916700000
```

Σε αυτό το makefile θέτουμε το προκαθορισμένο Group ID με (0x33) και ρυθμίζουμε τη συχνότητα σε (916.7 MHz) για τα mica2 motes.

• Επιπρόσθετα για να χρησιμοποιήσουμε το sensor board στο makefile της εφαρμογής μας τοποθετούμε πριν από τη δήλωση #include Makerules, SENSORBOARD=micasb.

•

• Τέλος στο terminal εκτελούμε την εντολή tocheck για να ελέγξουμε εάν η εγκατάσταση έχει γίνει επιτυχώς και οι μεταβλητές περιβάλλοντος έχουν ρυθμιστεί κατάλληλα.

Εγκατάσταση εφαρμογής στα motes και λήψη δεδομένων στον υπολογιστή

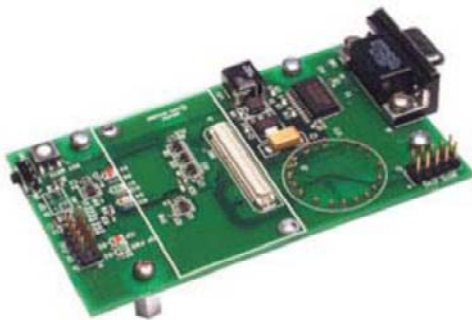
Εγκατάσταση εφαρμογής στα motes

Για να προγραμματίσουμε τα motes, και να κάνουμε κάτι χρήσιμο με τα δεδομένα που συγκεντρώνονται, πρέπει να υπάρχει μια διαδρομή από το δίκτυο αισθητήρων προς τον πραγματικό υπολογιστή. Για αυτή την εφαρμογή το crossbow mib510 programming board, θα λειτουργεί ως πύλη μεταξύ του δικτύου και του υπολογιστή. Η προγραμματιστική πλακέτα (programming board) συνδέεται με τον υπολογιστή μέσω ενός καλωδίου που επισυνάπτεται στην σειριακή θύρα του. Στη συνέχεια ένα Mote είναι συνδεδεμένο, μέσω των 51-pin connector στο programming board, και επιτρέπει την επικοινωνία μεταξύ των δύο συσκευών.

Αφού συνδέσαμε το αντίστοιχο mote, η μεταγλώττιση της εφαρμογής σε mica2 πλατφόρμα γίνεται μέσα από το παράθυρο εντολών του Cygwin. Μεταφερόμαστε στο φάκελο που περιέχει την εφαρμογή και εκτελούμε την εντολή “**make mica2**”.

Όταν ένα πρόγραμμα χρειάζεται να φορτωθεί στο Mote, αυτό γίνεται με την πληκτρολόγηση «**make mica2 [re]install mib510,serial(πχ dev/tty/S2)**» στη γραμμή εντολών, στον κατάλογο της εφαρμογής. Το πρόγραμμα στη συνέχεια μεταγλωττίζεται και ανεβαίνει στο Mote.

Με το reinstall γίνεται set η διεύθυνση και η προγραμματιστική συσκευή (program device) δεν κάνει ξανά μεταγλώττιση οπότε είναι πιο γρήγορο.



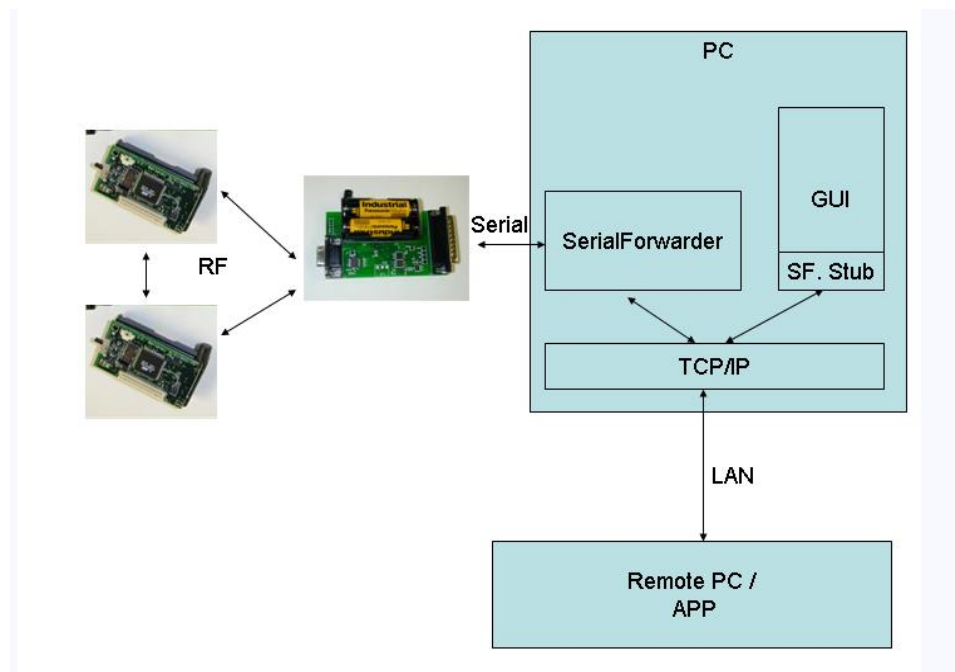
Crossbow MIB510



**Mica mote συνδεδεμένο στο
programming board**

Λήψη δεδομένων στον υπολογιστή[10]

Για να πάρουμε πίσω τα πακέτα δεδομένων στον ηλεκτρονικό υπολογιστή, ένα πρόγραμμα που ονομάζεται TOSBase πρέπει να εγκατασταθεί σε ένα Mote συνδεδεμένο με την προγραμματιστική πλακέτα (programming board). Το TOSBase ενεργεί ως γέφυρα μεταξύ του Motes radio, και τον υπολογιστή. Κάθε μήνυμα που λαμβάνει το TOSBase στέλνεται μέσω της σειριακής σύνδεσης (serial link) στον υπολογιστή. Στον υπολογιστή, ένα άλλο πρόγραμμα που ονομάζεται SerialForwarder σταματά το μήνυμα. Είναι δυνατό να γράψουμε μια εφαρμογή Java που να συνδέεται άμεσα με τη σειριακή θύρα για να συλλέξει τα εισερχόμενα πακέτα. Το πρόβλημα με αυτόν τον τρόπο ωστόσο, είναι ότι μόνο ένα πρόγραμμα μπορεί να ακουστεί στο port σε μια στιγμή. Ο SerialForwarder ξεπερνά αυτό το πρόβλημα λαμβάνοντας τα εισερχόμενα πακέτα και τη διάθεσή τους στην υποδοχή TCP 9001. Αυτό σημαίνει τώρα ότι πολλά προγράμματα μπορεί να συνδεθούν με το socket του SerialForwarder και όλα να λαμβάνουν τα εισερχόμενα πακέτα.



Java interaction με το δίκτυο αισθητήρων[10]

Απεικόνιση δεδομένων στον υπολογιστή

Η απεικόνιση των δεδομένων στον υπολογιστή έγινε με τη χρήση συγκεκριμένων εργαλείων σε Java που μας παρέχει το TinyOS. Τα εργαλεία αυτά μας βοηθούν τόσο στο να απεικονίσουμε τα πακέτα στη μορφή ‘raw data’ με την οποία καταφθάνουν στο mote, όσο και να παραστήσουμε γραφικά τις τιμές των μετρήσεων που έχουμε πάρει.

Το εργαλείο Listen – Μορφή πακέτων raw data

Το εργαλείο Listen χρησιμοποιείται για να απεικονίσουμε στη οθόνη τα πακέτα raw data όπως αυτά καταφθάνουν στη σειριακή θύρα του υπολογιστή μέσω της εφαρμογής TOSBase. Μεταφερόμαστε στο κατάλογο tools/java και πληκτρολογούμε τις εντολές :

\$make

\$export MOTECOM=serial@serialport:baudrate

όπου serialport είναι η σειριακή θύρα στην οποία έχουμε συνδέσει το mica2 στον υπολογιστή, καθοδηγούμε το εργαλείο Listen ώστε να ακούει το συγκεκριμένο mote.

Παράδειγμα για το mica2: **\$ export MOTECOM=serial@COM3:mica2**

Εκτελώντας στη συνέχεια την εντολή “**java net.tinyos.tools.Listen**” παρατηρούμε στην οθόνη μια συνεχόμενη ροή δεδομένων που αντιστοιχούν στα πακέτα “raw data” που λαμβάνουμε:

```
% java net.tinyos.tools.Listen
```

```
serial@COM1:19200: resynchronising
```

```
7e 00 0a 7d 1a 01 00 0a 00 01 00 46 03 8e 03 96 03 96 03 96 03 97 03 97 03 97 03 97 03
```

```
7e 00 0a 7d 1a 01 00 14 00 01 00 96 03 97 03 97 03 98 03 97 03 96 03 97 03 96 03 96 03 96 03
```

```
7e 00 0a 7d 1a 01 00 1e 00 01 00 98 03 98 03 96 03 97 03 97 03 98 03 96 03 97 03 97 03 97 03
```

Κάθε πακέτο που φθάνει από το mote αποτελείται από ένα σύνολο πεδίων. Κάποια από αυτά αντιστοιχούν στα αντίστοιχα πεδία που ορίζονται στη δομή του πακέτου TOS_Msg, ενώ στο πεδίο data payload του μηνύματος περιέχεται το μήνυμα της εφαρμογής μας. Πρέπει εδώ να αναφέρουμε ξανά ότι, τα δεδομένα μέσα στα πακέτα παριστάνονται σε μορφή little-endian, οπότε και μια τιμή ‘6F 08’ (2 bytes) παριστάνει

μια μέτρηση με το περισσότερο σημαντικό στοιχείο 0x08 και το λιγότερο σημαντικό ψηφίο 0x6F, το οποίο τελικά μας δίνει την τιμή 0x086F ή 2159 σε δεκαδική μορφή.

dest addr	handlerID	groupID	msg len	source addr	counter	channel	readings
7e 00	0a	7d	1a	01 00	14 00	01 00	96 03 97 03 97 03 98 03 97 03 96 03 97 03 96 03 96 03 96 03

Η μορφή του πακέτου TOS_Msg που στέλνεται ασύρματα αποτελείται από τα επόμενα πεδία:

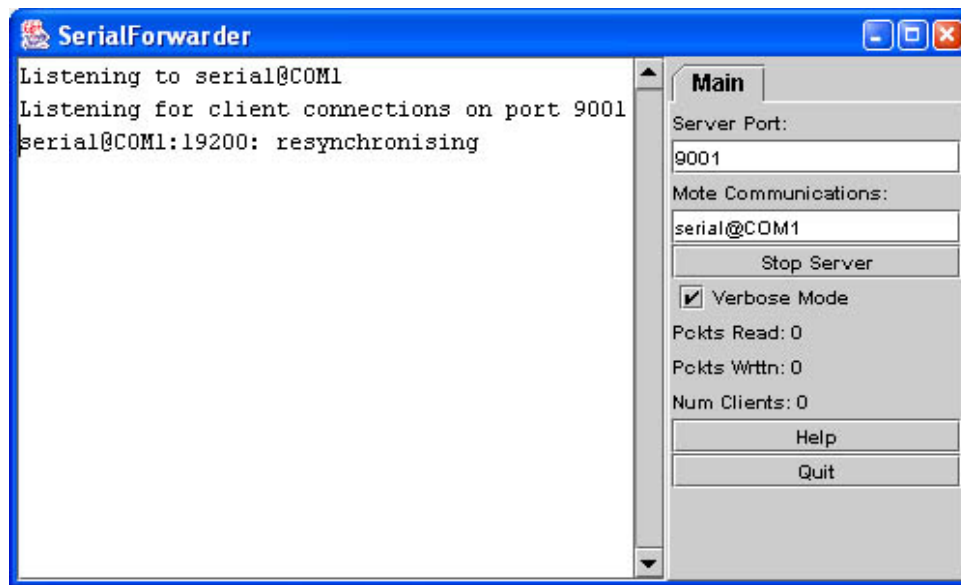
- **Destination address (2 bytes)**
- **Active Message handler ID (1 byte)**
- **Group ID (1 byte)**
- **Message length (1 byte)**
- **Payload (up to 29 bytes)**
- **CRC**

Η εφαρμογή SerialForwarder

Με το εργαλείο Listen δεν μπορούμε να απεικονίσουμε άμεσα και εύκολα τα δεδομένα που έρχονται μέσω της σειριακής θύρας του υπολογιστή. Για το λόγο αυτό, υπάρχουν μερικά άλλα εργαλεία που μπορούν να βοηθήσουν προς την κατεύθυνση αυτή. Ο Serial Forwarder είναι ένα πρόγραμμα που χρησιμοποιείται για να διαβάζει πακέτα από μια σειριακή θύρα και να τα προωθεί μέσα από μια TCP/IP σύνδεση, έτσι ώστε και άλλα προγράμματα να μπορούν να επικοινωνούν με το δίκτυο αισθητήρων μέσα από το δίκτυο αυτό. Το εργαλείο SerialForwarder τρέχει εισάγοντας την παρακάτω εντολή στο παράθυρο εντολών του Cygwin, έχοντας βέβαια αρχικά συνδέσει μια μονάδα mote στη θύρα USB του υπολογιστή:

```
java net.tinyos.sf.SerialForwarder -comm serial@COM1:<baud rate>
```

Παράδειγμα για το mica2: **java net.tinyos.sf.SerialForwarder -comm serial@COM4:mica2**



Το παράθυρο του SerialForwarder

Το όρισμα `-comm` λέει στον SerialForwarder να επικοινωνήσει με τη θύρα COM4. Επίσης, προσδιορίζει την προέλευση των πακέτων που προωθούνται μέσω του εργαλείου αυτού, χρησιμοποιώντας την ίδια σύνταξη όπως και η μεταβλητή MOTECOM που είδαμε προηγουμένως. Σε αντίθεση όμως με τα περισσότερα προγράμματα, ο SerialForwarder δεν κάνει χρήση της μεταβλητής MOTECOM αλλά χρησιμοποιεί το όρισμα `-comm` για την πηγή προέλευσης του πακέτου.

Το όρισμα `baud rate` λέει στον SerialForwarder να επικοινωνήσει με την ταχύτητα που υποστηρίζει η πλατφόρμα mica2, δηλαδή στην περίπτωσή μας τον ρυθμό 916700000 baud.

Ο SerialForwarder δεν απεικονίζει το ίδιο το πακέτο, αλλά ανανεώνει τον μετρητή πακέτων στην δεξιά γωνία του παραθύρου. Καθώς τρέχει, ακούει για συνδέσεις πελάτη δικτύου σε μία δοσμένη θύρα (port) – 9001 είναι η προκαθορισμένη θύρα και απλά προωθεί πακέτα TinyOS από τη σειριακή θύρα στη σύνδεση πελάτη και αντίστροφα. Εδώ αξίζει να αναφέρουμε ότι στο

SerialForwarder, μπορούν να συνδεθούν ταυτόχρονα περισσότερες της μια εφαρμογές και όλες θα λαμβάνουν ένα αντίγραφο του μηνύματος από το δίκτυο αισθητήρων.

Το εργαλείο γραφικής απεικόνισης (πχ για την εφαρμογή Oscilloscope)

Για την γραφική απεικόνιση των δεδομένων χρησιμοποιούμε το Javaεργαλείο Oscilloscope. Το εργαλείο αυτό δίνει επίσης τη δυνατότητα για αποθήκευση των μετρήσεων , έτσι ώστε να είναι δυνατή στη συνέχεια η περαιτέρω επεξεργασία τους σε εφαρμογές όπως το Matlab. Αφήνοντας τον SerialForwarder ανοικτό να τρέχει, πληκτρολογούμε την παρακάτω εντολή:

```
$ java net.tinyos.oscope.oscilloscope
```

Η εντολή αυτή ανοίγει το παράθυρο γραφικής απεικόνισης του Oscilloscope. Το εργαλείο αυτό συνδέεται με τον serialForwarder μέσω του δικτύου, ανακτά τα πακέτα δεδομένων που του προωθεί, αναλύει τις τιμές των μετρήσεων που στάλθηκαν ασύρματα μέσω του mote και τις απεικονίζει γραφικά. Έχοντας συνδεδεμένα όλα τα επιμέρους τμήματα της εφαρμογής και εκτελώντας την παραπάνω εντολή βλέπουμε το επόμενο παράθυρο:



Ο οριζόντιος άξονας x του γραφήματος παριστάνει τον αριθμό του μετρητή πακέτων και ο κάθετος άξονας y παριστάνει την τιμή μέτρησης του ADC. Πολλές φορές όταν δεν είναι άμεσα ορατή η κυματομορφή, είναι απαραίτητη η ρύθμιση του zoom με πάτημα των κουμπιών 'Zoom In ' ή 'Zoom Out ' για τον οριζόντιο ή τον κατακόρυφο άξονα. Επίσης, όταν ο αριθμός των δειγμάτων του μετρητή είναι εκτός του πεδίου απεικόνισης, η ενεργοποίηση της επιλογής 'Scrolling' μας οδηγεί απευθείας στις πιο πρόσφατες εικονιζόμενες τιμές.

Τέλος, αποεπιλέγοντας το πεδίο 'Connect Data points' μπορούμε να δούμε τα αντίστοιχα σημεία (τιμές) που αποτελούν τα δείγματα του ADC που στάλθηκαν ασύρματα. Τα δεδομένα μπορούν ταυτόχρονα να αποθηκευτούν σε αρχείο .txt πατώντας το πλήκτρο 'Save Data' και επιλέγοντας το φάκελο αποθήκευσης. Το αρχείο αυτό στη συνέχεια μπορεί να 'διαβαστεί' από το Matlab με τη δημιουργία κατάλληλου κώδικα, να απεικονιστεί η κυματομορφή και να γίνει η επεξεργασία της, σύμφωνα με τις ανάγκες μας.

Βιβλιογραφία Παραρτήματος Α

- [1] MPR – MIB Series User Manual and data sheet ,Revision B, June 2006 PN: 7430-0021-07
- [2] David Gay, Philip Levis, David Culler, Eric Brewer May 2003, nesC 1.1 “Language Reference Manual” (<http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf>)
- [3] David Gay, Phil Levis, Robert von Behren, Matt Welsh, Eric Brewer and David Culler. Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and implementation (2003), “The NesC Language: A Holistic Approach to Networked Embedded Systems”.
- [4] Philip Levis October 27, 2006, “TinyOS Programming”
- [5] TinyOS Documentation - <http://www.tinyos.net/tinyos-1.x/doc/>
- [6] Chien-Liang Fok. CS521 Fall 2004, TinyoOS Tutorial (www.princeton.edu/~wolf/EECS579/imotes/tos_tutorial.pdf)
- [7] Rev. A, September 2005, Document 7430-0022-07 Crossbow “Getting Started Guide”
- [8] Crossbow MPR-MIB Users Manual
- [9] Crossbow MICA2 WIRELESS MEASUREMENT SYSTEM
- [10] Paul William Rutter May 2007, Dissertation “Data Gathering in Wireless Sensor Networks”

Παράρτημα Β

Τεχνικό Υπόβαθρο : TinyDB

Εγκατάσταση και Απαιτήσεις

Το TinyDB απαιτεί να είναι εγκατεστημένο το TinyOS και να δουλεύει και η Java.

Ο ακόλουθος πίνακας παρουσιάζει τις απαιτήσεις του λογισμικού TinyDB:

Required Software	Notes
avr-gcc	Version 3.3 or later
Java SDK	Version 1.4 or later
nesC compiler	From the TinyOS 1.1 release
javax.comm tools	Version 1.3 or later, from IBM
TinyOS	From SourceForge CVS

Για να βεβαιωθούμε ότι η εγκατάσταση λειτουργεί σωστά, κάνουμε τα εξής:

1. Μεταγλωττίζουμε και εγκαθιστούμε την εφαρμογή TinyDB στο mote. Για να το γίνει αυτό, συνδέουμε το mote στο programming board και στη συνέχεια, πληκτρολογούμε τις πιο κάτω εντολές:

- **\$ cd tinyos-1.x/apps/TinyDBApp/**
- **\$ make mica**
- **\$ make mica install mib510,/dev/tty/COMx**

2. Μεταγλωττίζουμε και τρέχουμε το TinyDBMain java classes. Για να γίνει αυτό πληκτρολογούμε τις πιο κάτω εντολές:

- **\$ cd tinyos-1.x/tools/java/net/tinyos/tinydb**
- **\$ make**
- **\$ cd tinyos-1.x/tools/java**
- **\$ java net/tinyos.tinydb.TinyDBMain**

Θα πρέπει να βεβαιωθούμε ότι τα παρακάτω αρχεία jar υπάρχουν στο classpath (θα πρέπει όλα να είναι διαθέσιμα στο tinyos-1.x/tools/java/jars):

- JLex.jar
- cup.jar
- plot.jar
- xercesImpl.jar
- xmlParserAPIs.jar

Πρέπει επίσης στον κατάλογο tinyos-1.x/tools/java να υπάρχει στο CLASSPATH.

Το CLASSPATH θα πρέπει να είναι κάπως έτσι:

```
./path/to/java/jre/lib/rt.jar:path/to/java/lib/dt.jar:
```

```
/path/to/java/lib/tools.jar:/opt/IBMJava2-13/jre/lib/ext/comm.jar:
```

```
tinyos-1.x/tools/java/jars/plot.jar:tinyos-1.x/tools/java/jars/cup.jar:
```

```
tinyos-1.x/tools/java/jars/xercesImpl.jar:tinyos-1.x/tools/java/jars/xmlParserAPIs.jar:
```

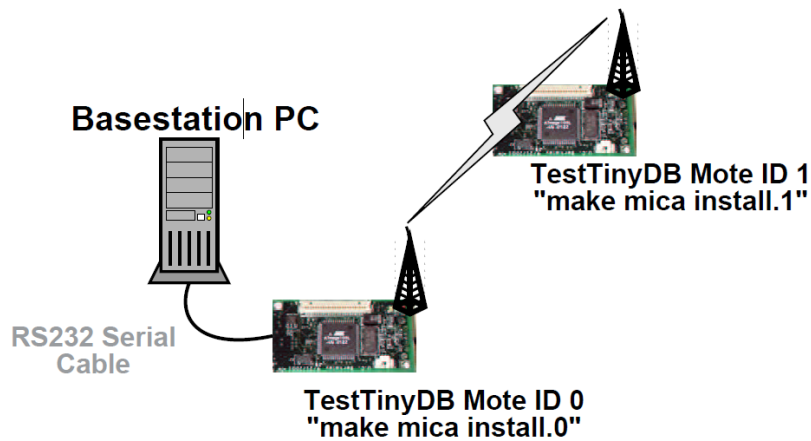
```
tinyos-1.x/tools/java/jars/JLex.jar:tinyos-1.x/tools/java
```

Δημιουργία ενός δικτύου από TinyDB motes

Το πρώτο βήμα είναι να προγραμματίσουμε μια σειρά από motes με το λογισμικό TinyDB. Καθένα από αυτά τα motes πρέπει να έχει ένα μοναδικό αναγνωριστικό. Σε TinyOS, μπορούμε να ρυθμίσουμε το ID ενός Mote στην εγκατάσταση της εφαρμογής στο mote, για παράδειγμα, να προγραμματίσω ένα mote TinyDB στο ID 2:

- cd tinyos-1.x/apps/TinyDBApp/
- make mica install.2 mib510,/dev/tty/COMx

Για να εκτελέσουμε την εφαρμογή TinyDB, θα χρειαστούν τουλάχιστον δύο motes: ένα για να λειτουργεί ως Mote basestation, και ένα ή περισσότερα για τη διανομή και την εκτέλεση των ερωτημάτων. Όλα τα motes, συμπεριλαμβανομένου του basestation, εκτελούν την ίδια εφαρμογή TinyDBApp, ωστόσο, το Mote basestation πρέπει να οριστεί σε 0 αναγνωριστικό. Μετά τον προγραμματισμό των motes, συνδέουμε τη προγραμματιστική πλακέτα (programming board) στον υπολογιστή μέσω της σειριακής θύρας (USB), και τοποθετούμε το Mote basestation στο programming board.



Τα Leds

Τα LED για TinyDB motes χρησιμοποιούνται για να δείχνει την τρέχουσα κατάσταση της συσκευής. Κάθε LED χρησιμοποιείται για να επικοινωνούν ένα από τα πολλά κομμάτια των πληροφοριών, ανάλογα με την τρέχουσα κατάσταση του Mote. Ένα απλό ερώτημα επιλογής θα πρέπει να προκαλέσει το Mote να αναβοσβήνει το κίτρινο LED μία φορά ανά epoch, και το κόκκινο LED του μία φορά ανά τιμή δεδομένου διαβιβάζονται μέχρι το δέντρο. Ο πίνακας που ακολουθεί συνοψίζει τις δράσεις των LED στα Mote:

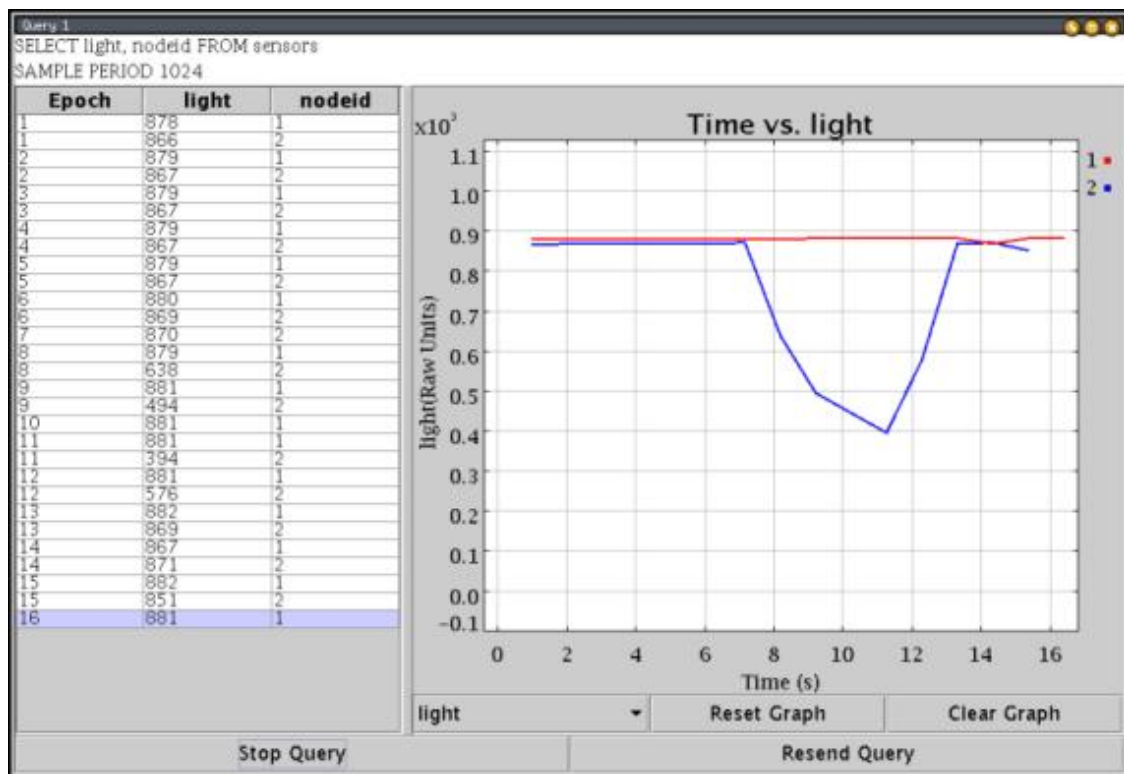
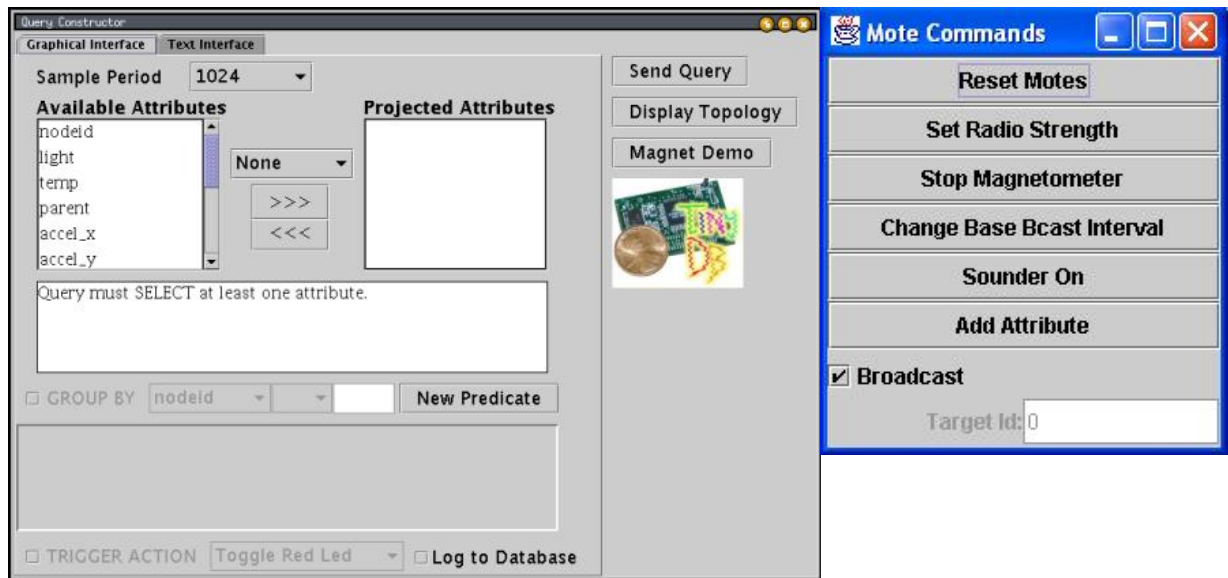
Mote Status	LED Response	Notes
Sent Message Received Aggregate From Child End of Epoch Aborted Query Reset Mote	Red Toggle Green Toggle Yellow Toggle All LEDs On All LEDs Flash	Once per query

Running the TinyDBMain GUI [1]

Η TinyDBMain Java εφαρμογή παρέχει μια γραφική διεπαφή για τη διανομή ερωτημάτων πάνω σε motes και τη συλλογή στοιχείων από αυτά. Για να εκτελέσουμε αυτή την εφαρμογή, πληκτρολογούμε:

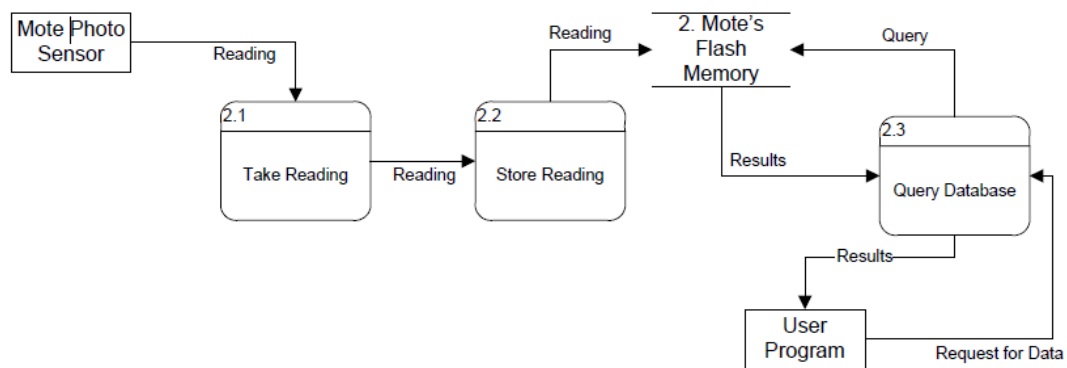
- `cd tinyos-1.x/tools/java/net/tinyos/tinydb`
- `make`
- `cd tinyos-1.x/tools/java`
- `java net/tinyos.tinydb.TinyDBMain`

Δύο παράθυρα πρέπει να εμφανιστούν. Ένα, το παράθυρο εντολών που επιτρέπει να στείλουμε μια ποικιλία εντολών ελέγχου στα motes. Το άλλο, το query window, επιτρέπει να οικοδομήσουμε και να στείλουμε τα ερωτήματα στο δίκτυο.



Υλοποίηση με καταναμημένη βάση δεδομένων[2]

Η καταναμημένη βάση δεδομένων θα υλοποιηθεί με τη χρήση του προγράμματος TinyDB που τρέχει μέσω των Motes για να παράγει ένα δίκτυο επεξεργασίες επερωτήσεων. Θα αποτελείται από τουλάχιστο δύο Motes, όπου κάθε ένα θα τρέχει την εφαρμογή TinyDBApp.nc. Ένα ή περισσότερα Motes θα χρησιμοποιηθούν για τη συλλογή των αναγνώσεων από τους αισθητήρες και το άλλο θα είναι συνδεδεμένο με το programming board για την παροχή του συνδετικού κρίκου μεταξύ του δικτύου και του υπολογιστή. Ο πίνακας θα δημιουργηθεί στη μνήμη των Motes που θα χρησιμοποιηθούν για να καταγράψουν τις αναγνώσεις. Στη συνέχεια θα μπορούν να εκδίδονται ερωτήματα από τον υπολογιστή προς το δίκτυο.



TinyDB Αρχιτεκτονική Components

Αυτό το διάγραμμα δείχνει πώς οι ροές δεδομένων διανέμονται μέσω της αρχιτεκτονικής. Οι μετρήσεις που αποθηκεύονται στα Motes στην εσωτερική μνήμη flash, ώστε ένα πρόγραμμα χρήστη χρησιμοποιεί το ερώτημα αυτό, προκειμένου να ανακτήσει τις αναγνώσεις.

Χρόνος για να γράψεις στο TinyDB

Ο τρόπος της καταναμημένης αρχιτεκτονικής TinyDB που δημιουργήθηκε σημαίνει ότι ο χρόνος για να γράψουμε στη βάση δεδομένων πρέπει να είναι σχετικά γρήγορος. Αυτό συμβαίνει διότι, σε αυτή την αρχιτεκτονική, κάθε ανάγνωση αποθηκεύεται τοπικά

στο Mote. Ως εκ τούτου το μόνο overhead θα είναι ο χρόνος που απαιτείται για να γράψουμε στην τοπική μνήμη flash. Το TinyDB δεν περιλαμβάνει καμία μέθοδο χρονοσήμανσης μετρήσεων προκειμένου να υπολογιστεί αυτή η τιμή.

Το πρόγραμμα SenseLightToLog, το οποίο περιλαμβάνεται με το TinyOS, τροποποιήθηκε προκειμένου να δημιουργήσει το πρόγραμμα LogTest. Το πρόγραμμα λαμβάνει μετρήσεις από τον αισθητήρα ανά δευτερόλεπτο και τις γράφει σε ένα buffer. Όταν ο buffer είναι πλήρης, γράφεται στο flash. Η ανάγνωση από το στοιχείο SimpleTime.nc λαμβάνεται αμέσως πριν από την έναρξή της εγγραφής και αμέσως μετά από το τέλος. Αυτές οι δύο μετρήσεις κατασκευάζονται στη συνέχεια σε ένα μήνυμα και να αποστέλλονται μέσω του radio πίσω στον υπολογιστή όπου έχουν διακοπεί από το πρόγραμμα WriteTime.java. Αυτό το πρόγραμμα είναι η έναρξη και λήξη τιμών και αφαιρεί το ένα από το άλλο για να δώσει τη διαφορά ώρας σε χιλιοστά του δευτερολέπτου.

Χρόνος για να διαβάσεις από το TinyDB

Σε αυτή την εφαρμογή, κάθε κόμβος του δικτύου είναι υπεύθυνος για τη διατήρηση της δικής του τοπικής βάσης δεδομένων των αποτελεσμάτων. Όταν ένα ερώτημα έχει εκδοθεί από τον υπολογιστή, εναπόκειται στο κάθε κόμβο για την επεξεργασία του ερωτήματος, και επιστρέφει τα κατάλληλα αποτελέσματα. Στο TinyDB, η ιδέα είναι ότι τα ερωτήματα θα μπορούν να αποστέλλονται απευθείας στα motes που τρέχουν την εφαρμογή TinyDB, η οποία το διακόπτει και το εκτελεί. Αυτό γίνεται χρησιμοποιώντας το περιβάλλον κειμένου στην εφαρμογή TinyDB Java. Για την αλληλεπίδραση με τη βάση δεδομένων, έχει δημιουργηθεί ένα πρόγραμμα QueryDB Java. Αυτό το πρόγραμμα αναπτύχθηκε από το πρόγραμμα DemoApp που περιλαμβάνεται στο TinyDB. Το πρόγραμμα λαμβάνει το TinySQL ερώτημα, το ερμηνεύει και το διοχετεύει στο δίκτυο. Στη συνέχεια, χρησιμοποιεί τη διασύνδεση ResultListener για να ακούσουν τα αποτελέσματα που έρχονται πίσω από το δίκτυο.

Το TinyDB έχει την ικανότητα να εκτελεί ερωτήματα αφού συνδεθεί με τη Flash μνήμη των motes. Για να ενεργοποιήσουμε αυτή τη δυνατότητα, θα πρέπει να ενεργοποιήσουμε την επιλογή kMATCHBOX στο αρχείο κεφαλίδας CompileDefines.h.

Ερωτήματα πάνω στο flash είναι προσβάσιμα μόνο μέσω της διεπαφής κειμένου του TinyDB GUI. Το TinyDB περιλαμβάνει εντολές για τη δημιουργία πινάκων που αποθηκεύονται στο flash, για τη λειτουργία ερωτημάτων που εισάγονται σε αυτούς τους πίνακες, τα ερωτήματα για τη λειτουργία της ανάκτησης από τους πίνακες αυτούς, και για τη διαγραφή αυτών των πινάκων. Η σύνταξη για τη δημιουργία ενός πίνακα μοιάζει πολύ με τη σύνταξη για τη δημιουργία ενός πίνακα στο παραδοσιακό SQL:

```
CREATE BUFFER name
```

```
SIZE x
```

```
( field1 type,
```

```
field2 type,
```

```
... )
```

Αυτό δημιουργεί έναν πίνακα με το όνομα name, με χώρο για x γραμμές, όπου κάθε σειρά έχει τα πεδία που αναφέρονται στο τέλος του ερωτήματος. Τα πεδία και τα ονόματα των πινάκων μπορεί να είναι μέχρι 8 χαρακτήρες, και οι τύποι πεδίου πρέπει να είναι ένας από uint8, uint16, int8, int16 ή Int32, όπου το u αναφέρει μη προσημασμένα πεδία και ο αριθμός στο τέλος του τύπου δείχνει τον αριθμό των bits στο πεδίο. Για να προσθέσουμε δεδομένα σε ένα συγκεκριμένο πίνακα, τρέχουμε ένα ερώτημα, όπως:

```
SELECT field1, field2, ...
```

```
FROM sensors
```

```
SAMPLE PERIOD x
```

```
INTO name
```

Ο αριθμός και το είδος των πεδίων που εμφανίζονται σε αυτό το ερώτημα πρέπει να ταιριάζει με τον αριθμό και το είδος των πεδίων στην CREATE BUFFER δήλωση παραπάνω. Ο ταχύτερος ρυθμός που διατίθενται για ερωτήματα είναι 1 δείγμα ανά 64 ms (τιμές κάτω των 64 δεν θα λειτουργήσει). Μόνο ένα ερώτημα μπορεί να συνδεθεί στο buffer σε μια στιγμή, και ότι τα νέα ερωτήματα θα αντικαταστήσουν τα στοιχεία που είχαν συνδεθεί στο παρελθόν στον πίνακα. Τέλος, για να επιλέξουμε από πίνακα, εκτελούμε ένα ερώτημα, όπως:

```
SELECT field1, field2, ...
```

```
SAMPLE PERIOD y
```

```
FROM name
```

Όπου τα ονόματα των πεδίων είναι τα ίδια με τα ονόματα των πεδίων που ορίστηκαν στην CREATE BUFFER δήλωση. Αυτό θα φέρει τα αποτελέσματα που καταγράφονται από τον αναφερόμενο buffer, ξεκινώντας από την αρχή του πίνακα και σαρώνει προς τα εμπρός.

Επί του παρόντος, σε TinyDB, δεν μπορούμε να εκτελέσουμε ένα ερώτημα που επιλέγει από ένα πίνακα Flash την ίδια στιγμή που ένα ερώτημα γράφει στον ίδιο πίνακα και βρίσκεται σε λειτουργία. Πρέπει να σταματήσει το πρώτο ερώτημα, χρησιμοποιώντας το "Stop Query" κουμπί για το ερώτημα, ή χρησιμοποιώντας το Stop Query από τη γραμμή εντολών.

Τέλος, μπορείτε να διαγράψετε όλους τους πίνακες από τη flash και να επαναφέρετε την εσωτερική κατάσταση που συνδέονται με αυτά εκτελώντας το ερώτημα:

DROP ALL

Γενικά το TinyDB είναι ένα χρήσιμο πρόγραμμα το οποίο είναι εύκολο στη χρήση, σε σύγκριση με τη σύνταξη μιας συλλογής δεδομένων αίτηση από το μηδέν. Η διεπαφή επιτρέπει στο χρήστη ένα μεγάλο μέρος του ελέγχου επί ποια δεδομένα συλλέγονται και πόσο συχνά συμβαίνει αυτό. Ως εκ τούτου, ένας αρχάριος χρήστης πολύ γρήγορα θα μπορούσε να δημιουργήσει ένα ασύρματο δίκτυο και να ξεκινήσει τη λήψη δεδομένων με πολύ λίγη προσπάθεια.

Πιο κάτω θα δίνονται κάποιες περεταίρω λεπτομέρειες για την εφαρμογή TinyDB

Στο κατάλογο /opt/tinyos-1.x/apps/TinyDBApp βρίσκεται η εφαρμογή..

Η εφαρμογή TinyDBApp αποτελείται από μόνο ένα configuration και όλα τα components modules βρίσκονται αποθηκευμένα σε βιβλιοθήκες.

```
includes CompileDefines;
```

```
configuration TinyDBApp {  
}
```

```
implementation {
```

```
components Main, TupleRouter, AttrRoom, AttrAccel, AttrMic, AttrTemp,  
AttrTime, AttrVoltage, AttrHumidity, AttrHama, AttrNewAccel, AttrPhoto,  
AttrPot, AttrTimestamp;
```

```
Main.StdControl -> TupleRouter; // \tos\lib\TinyDB\TupleRouterM.nc  
Main.StdControl -> AttrRoom; // \tos\lib\Attributes\AttrRoom.nc  
Main.StdControl -> AttrAccel;  
Main.StdControl -> AttrMic;  
Main.StdControl -> AttrTime;  
Main.StdControl -> AttrVoltage;  
Main.StdControl -> AttrHumidity;  
Main.StdControl -> AttrHama;  
Main.StdControl -> AttrNewAccel;  
Main.StdControl -> AttrPhoto;  
Main.StdControl -> AttrPot;  
Main.StdControl -> AttrTemp;  
Main.StdControl -> AttrTimestamp;
```

```
}
```

Εσωτερικά το TinyDB [3]

Attr και Command

Τα Attr και command συστατικά περιέχουν τον κώδικα για τη διαχείριση του σχήματος του Mote. Το σχήμα αποτελείται από τους πίνακες από τους τύπους των χαρακτηριστικών, τις εντολές, και των συναφών ρουτινών για την ενημέρωση και την αναζήτηση αυτών των πινάκων.

TinyDBAttr

Αυτό το απλό συστατικό είναι το κομβικό σημείο για όλα τα Builtin χαρακτηριστικά του TinyDB. Συνδέει όλα τα στοιχεία που εφαρμόζουν Builtin χαρακτηριστικά μαζί. Αυτό το στοιχείο πρέπει να ενημερωθεί εάν προσθέσουμε ένα νέο στοιχείο που υλοποιεί νέα χαρακτηριστικά για το TinyDB.

TinyDBCommand

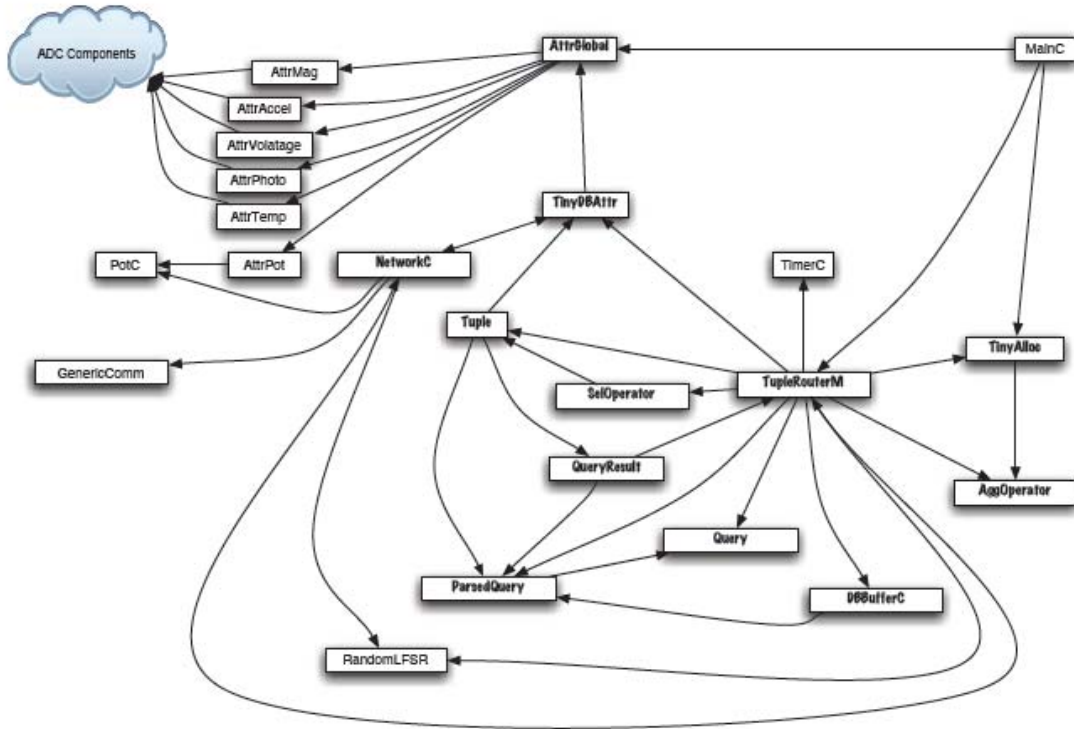
Αυτό το απλό συστατικό είναι το κομβικό σημείο για όλες τις Builtin εντολές (command) του TinyDB. Συνδέει όλα τα στοιχεία που εφαρμόζουν Builtin εντολές μαζί. Αυτό το στοιχείο πρέπει να ενημερωθεί εάν προσθέσουμε ένα νέο στοιχείο που υλοποιεί νέα εντολή για το TinyDB.

Tuple

Αυτό το στοιχείο παρέχει αρκετά απλή κοινή ωφελείας για διαχείριση της Tuple δομής δεδομένων, όπως ορίζεται στο TinyDB.h.

QueryResult

Το στοιχείο αυτό μετατρέπει μεταξύ Tuples, QueryResults, και byte-strings. Αυτές οι δομές δεδομένων ορίζονται στο `tinyos-1.x/tos/lib/TinyDB/TinyDB.h`. Εν συντομία, ένα Tuple είναι ένας πίνακας τιμών (typed vector value), όπως είναι ο SQL. Ένα QueryResult κρατά την πλειάδα (tuple) και ορισμένα μεταδεδομένα, συμπεριλαμβανομένης και της αναζήτησης ταυτότητας της επερώτησης, ένα δείκτη στο σύνολο των αποτελεσμάτων, και ένα epoch αριθμό.



Διάγραμμα συστατικών TinyDB [3]

TinyDB Query Processing Operators

TupleRouter

Το TupleRouter αποτελεί τον πυρήνα του συστήματος TinyDB και με την κλήση συναρτήσεων του Network μπορεί να διεκπεραιώσει τις πιο κάτω λειτουργίες:

- Λαμβάνει ερωτήματα από το δίκτυο,
- Δημιουργεί την τοπική κατάσταση
- Συλλέγει τα αποτελέσματα από τους τοπικούς αισθητήρες και τους γειτονικούς κόμβους
- Τροφοδοτεί τους τοπικούς αισθητήρες και τους γειτονικούς κόμβους με τις τοπικές επερωτήσεις.

Τα ερωτήματα επιλέγονται και συναθροίζονται. Τα αποτελέσματα από τις επερωτήσεις χωρίς συνάθροιση μπορούν απλά να προωθηθούν στη ρίζα του δέντρου και χειρίζονται από τον επεξεργαστή επερωτήσεων. Τα ερωτήματα με συνάθροιση επεξεργάζονται σύμφωνα με την προσέγγιση TAG: όπου κάθε κόμβος συλλέγει τις τοπικές συναθροίσεις από τα παιδιά του και συνδυάζει τα συγκεντρωτικά στοιχεία μαζί με τις δικές του αναγνώσεις αισθητήρα, και τα προωθεί στον πατέρα του.

Υπάρχουν τρεις κύριες διαδρομές εκτέλεσης εντός του TUPLE_ROUTER:

Ένα για αποδοχή νέων επερωτήσεων, ένα για την αποδοχή των αποτελεσμάτων από τους γειτονικούς κόμβους, και ένα για τη δημιουργία τοπικών αποτελεσμάτων και την παροχή δεδομένων στον κόμβο του πατέρα.

Query Arrive

- 1) Τα νέα ερωτήματα φθάνουν στο TUPLE_ROUTER_QUERY_MESSAGE. Κάθε ερώτημα θεωρείται ότι θα πρέπει να προσδιορίζεται από μια παγκοσμίως μοναδική ταυτότητα. Τα query messages περιλαμβάνουν ένα μέρος της επερώτησης. Όλα τα QUERY_MESSAGES περιγράφουν έναν απλό ερώτημα που πρέπει να φτάσει πριν ο router να αρχίσει τη δρομολόγηση των πλειάδων για το συγκεκριμένο ερώτημα.

- 2) Όταν όλα τα QUERY_MESSAGESs έχουν φθάσει, ο δρομολογητής καλεί τον `parseQuery ()` για να δημιουργήσει μια συμπαγές εκπροσώπηση του ερωτήματος όπου τα ονόματα των πεδίων αντικαθίστανται με την ταυτότητα `id` η οποία μπορεί να χρησιμοποιηθεί ως `offsets` μέσα στον τοπικό κατάλογο των αισθητήρων.
- 3) Δοσμένου ενός `parsedQuery`, ο δρομολογητής της πλειάδας διαθέτει χώρο στο τέλος του ερωτήματος για να κρατήσει το ερώτημα μοναδικό, `in-flight tuple` για το συγκεκριμένο ερώτημα – η πλειάδα θα συμπληρώνει τα κατάλληλα πεδία δεδομένων, κατά την εκτέλεση του ερωτήματος.
- 4) Το `TupleRouter` καλεί το `setSampleRate ()` για να ξεκινήσει (ή επανεκκίνηση) το 32kHz ρολόι `Mote` και για να προσδιορίσει το `rate` για την αποστολή των δεδομένων για όλες τις ερωτήσεις του συστήματος. Εάν υπάρχει μόνο ένα ερώτημα, θα ενεργοποιείται μια φορά ανά `epoch` ενώ αν υπάρχουν πολλαπλά ερωτήματα, θα ενεργοποιείται το `GCD` παράδοση όλων των ερωτημάτων.

Delivery Tuple

Όταν ένα `clock event` εμφανίζεται (`TUPLE_ROUTER_TIMER_EVENT`), ο δρομολογητής πρέπει να εκτελεί τέσσερις δράσεις:

- α) Παράδοση των πλειάδων που ολοκληρώθηκαν στην προηγούμενη περίπτωση `clock event` (`deliverTuplesTask`). Αν το ερώτημα περιέχει συνάθροιση θα παραδώσει συγκεντρωτικά δεδομένα από τον `aggregate operator`, εάν δεν περιέχει `aggregate` θα παραδώσει την πλειάδα που συμπληρώθηκε κατά τη διάρκεια της τελευταίας επανάληψης. Μηδενίζει τους μετρητές.
- β) Μειώνει τους μετρητές για όλα τα ερωτήματα. Οι ερωτήσεις που έχουν μετρητή 0 πρέπει να παραδώσουν τα δεδομένα.

γ)Φέρνει τα πεδία δεδομένων για κάθε ερώτημα που ενεργοποιούνται σε αυτό το epoch. Εκτελείται βρόγχος σε όλα τα πεδία όλων των ερωτημάτων, ανάκτηση και συμπλήρωση των πλειάδων με τις κατάλληλες τιμές των σχετικών ερωτήσεων.

δ) Η διαδρομή συμπληρώνει τις πλειάδες. Πρώτα δρομολογούνται οι επιλογές και έπειτα οι συναθροίσεις εάν υπάρχουν. Εάν μια επιλογή απορρίψει την πλειάδα, σταματά η δρομολόγηση.

Neighbor Result Arrival

Όταν ένα αποτέλεσμα φθάσει από ένα γείτονα `TUPLE_ROUTER_RESULT_MESSAGE`, πρέπει να ενσωματωθούν στις συναθροισμένες τιμές που υπολογίζονται σε τοπικό επίπεδο. Αν το αποτέλεσμα αντιστοιχεί σε ένα συνολικό ερώτημα, το αποτέλεσμα διαβιβάζεται στο `AGG_OPERATOR` component, αλλιώς διαβιβάζονται προς τα μέχρι τη ρίζα.

SelOperator

Ο `SelOperator` είναι υπεύθυνος για τη σχεσιακή επιλογή.

AggOperator

Το στοιχείο αυτό επιτελεί δύο χαρακτηριστικά SQL: `GROUP BY` και την ομαδοποίηση.

Run TinyDB στο TOSSIM Simulator

Το `TinyDB` περιλαμβάνει υποστήριξη για τον προσομοιωτή `TinyOS`, `TOSSIM`. Για να τρέξει στο `TOSSIM` απαιτεί `recompilation` του πηγαίου κώδικα `Java`, καθώς και του `Mote` κώδικα.

Τα βασικά βήματα για τη δημιουργία του `TinyDB` για το `TOSSIM` έχουν ως εξής:

- `recompile` τον κώδικα `java` πληκτρολογώντας `make clean`; Στον κατάλογο `tinyos-1.x/tools/java/net/tinyos/` κάνουμε `make -f MakePC`. Με αυτό τον τρόπο θα ανοικοδομηθεί ο πηγαίος κώδικας `TinyDB`, διασφαλίζοντας ότι οι τάξεις μηνυμάτων έχουν ρυθμιστεί για να μιλήσουν με τον προσομοιωτή.

- Στον κατάλογο `tinynos-1.x/apps/TinyDBApp` πληκτρολογούμε `make-f MakePC pc`.
- Ρυθμίζουμε κατάλληλα τη μεταβλητή `DBG string`. Για παράδειγμα `DBG = USR1`.
- Ξεκινούμε το δυαδικό υπολογιστή με `n motes` για προσομοίωση στο κατάλογο `tinynos-1.x/apps/TinyDBApp` πληκτρολογώντας:
`/build / pc / main.exe n`.
- Θέτουμε το `java GUI` προσομοιωτή σε λειτουργία πληκτρολογώντας στον κατάλογο `tinynos-1.x/tools/java` (σε ένα διαφορετικό παράθυρο κέλυφος) `java net.tinynos.tinydb.TinyDBMain-sim`.
- Υποθέτοντας ότι έχετε ορίσει το `string DBG` όπως παραπάνω, θα πρέπει να μπορούμε να κάνουμε κλικ στην "Επαναφορά Motes" (Reset Motes).

Βιβλιογραφία Παραρτήματος Β

- [1] TinyOS Documentation - <http://www.tinynos.net/tinynos-1.x/doc/>
- [2] Paul William Rutter May 2007, "Data Gathering in Wireless Sensor Networks: Dissertation"
- [3] Sam Madden, Joe Hellerstein, and Wei Hong Version 0.4 September, 2003, "TinyDB: In-Network Query Processing in TinyOS"

Παράρτημα Γ

Τεχνικό Υπόβαθρο : KSPOT

KSPOT: Αποτελεσματική παρακολούθηση των K πιο σημαντικών γεγονότων σε ένα Wireless Sensor Network [1]

Το KSPOT, είναι ένα σύστημα το οποίο μπορεί να χρησιμοποιηθεί για την αποδοτική παρακολούθηση των Κορυφαίων-K (Top-K) απαντήσεων σε μία ερωτήση Q σε ένα ασύρματο δίκτυο αισθητήρων, το οποίο αποτελείται από ένα σύστημα κατάταξης και μία γραφική διεπιφάνεια χρήσης. Το σύστημα κατάταξης αξιοποιεί κατανεμημένους αλγόριθμους επεξεργασίας Κορυφαίων-K απαντήσεων έτσι ώστε να ελαχιστοποιήσει την κατανάλωση των περιορισμένων πόρων που έχει ένα ασύρματο δίκτυο αισθητήρων επιτυγχάνοντας έτσι την μακροζωία του.

Το σύστημα αναπτύσσει την κατάσταση of-the-art διανέμοντας Top-K ερωτήσεις επεξεργάζοντας αλγόριθμους, προκειμένου να πραγματοποιήσει δύο ερωτήματα, ερωτήματα στιγμιότυπου (snapshot query) και ιστορικά ερωτήματα (history query) ελαχιστοποιώντας την κατανάλωση των πόρων του συστήματος και την παράταση της διάρκειας ζωής του αναπτυσσόμενου δικτύου αισθητήρων. Επιπρόσθετα, το KSPOT είναι φιλικό και προσαρμοστικό προς το χρήστη παρέχοντας ένα σύστημα διαπροσωπείας όπου ο χρήστης ορίζει με δηλωτικό τρόπο ερωτήσεις τύπου SQL παραλαμβάνοντας τα αποτελέσματα γραφικά αντί υπό μορφή πίνακα.

Η επεξεργασία ερωτήσεων (query) σε ad-hoc περιβάλλοντα είναι ένα δύσκολο έργο, λόγω της πολυπλοκότητας που επιβάλλεται από τις εγγενείς δυσχέρειες της ενέργειας και της επικοινωνίας. Για το σκοπό αυτό, η ερευνητική κοινότητα έχει προτείνει να λαμβάνονται υπόψη καθορισμένες παράμετροι από το χρήστη προκειμένου να προκύψουν οι πιο σχετικές K (ή Top-K) απαντήσεις γρήγορα και αποτελεσματικά. Ένα Top-K ερώτημα επιστρέφει το υποσύνολο των πιο σχετικών απαντήσεων, σε αντικατάσταση του συνόλου όλων των απαντήσεων, για δύο λόγους:

1. για την ελαχιστοποίηση του κόστους των μετρικών που σχετίζεται με την ανάκτηση όλων των απαντήσεων και
2. για να βελτιωθεί η ανάκληση και η ακρίβεια του συνόλου των απαντήσεων, έτσι ώστε ο χρήστης να μην κατακλύζεται με άσχετα αποτελέσματα. Οι Top-K ερωτήσεις(Queries) μπορεί να υποστηρίξουν ένα ευρύ φάσμα εφαρμογών στις παρακάτω κατηγορίες:

1. Στιγμιότυπο Top-K ερωτήσεων (Snapshot Top-K Queries): Αυτές αναφέρονται στις τρέχουσες αναγνώσεις των συσκευών αισθητήρα (sense device). Ένα παράδειγμα αυτής της κατηγορίας είναι ένα ερώτημα του τύπου: *"Βρείτε τους κόμβους K με την υψηλότερη θερμοκρασία "*, ή , ερώτημα στιγμιότυπου του τύπου: *"Βρείτε τους πλησιέστερους K κόμβους σε απόσταση από τη φωτιά "*, ή τέλος, μια συνεχής αναζήτηση ερώτησης στιγμιότυπο του τύπου: *"Για την επόμενη μία ώρα να δίνεται σε συνεχή αναφορά τα K δωμάτια με την υψηλότερη μέση θερμοκρασία "*.

2. Ιστορικές Top-K ερωτήσεις (Historic Top-K Queries): Αυτές αναφέρονται σε ιστορικά δεδομένα τα οποία είναι προσωρινά αποθηκευμένα σε κάθε αισθητήρα. Δεδομένου ότι ένας αισθητήρας έχει περιορισμένο χώρο αποθήκευσης, μπορεί να πραγματοποιήσει την αποθήκευση (buffering) σε ένα συρόμενο παράθυρο (sliding window), είτε στη κύρια μνήμη, για συσκευές με επαρκή SRAM, όπως τα IMote2, ή σε δευτερογενείς μνήμη. Ένα παράδειγμα σε αυτή την κατηγορία είναι ένα ερώτημα του τύπου: *"Βρείτε το χρόνο K περιπτώσεων με την υψηλότερη μέση θερμοκρασία κατά τη διάρκεια των 3 τελευταίων μηνών"*, ή ένα χωρο-χρονικό ερώτημα του τύπου: *"Βρείτε τις K ζέβρες με τις πιο παρόμοιες ρίγες με τη ζέβρα X "*.

Το σύστημα KSPOT παρουσιάζεται να εφαρμόζει διάφορους top-k αλγόριθμους επεξεργασίας, προκειμένου να ξεπεράσει τις προκλήσεις κατάταξης τόσο για τις ερωτήσεις στιγμιότυπου όσο και για τις ιστορικές ερωτήσεις.

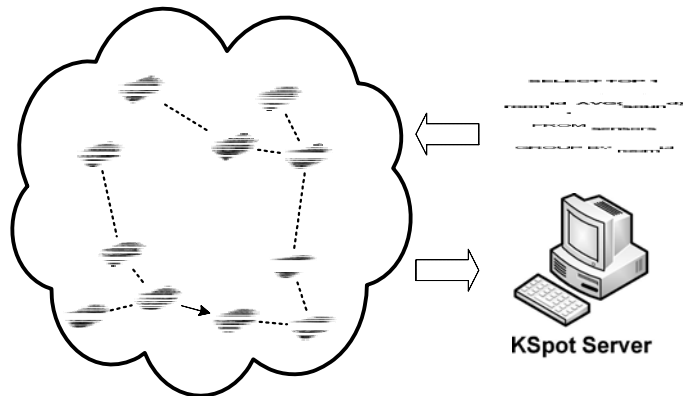
Το σύστημά αποτελείται από δύο υποσυστήματα του λογισμικού: KSPOT εξυπηρετητή (server) και KSPOT πελάτη (client). Το λογισμικό εξυπηρετητή KSPOT είναι γραμμένο σε JAVA και ενσωματώνει τη διανομή των top-k ερωτήσεων επεξεργασίας από το TinyDB server και διαθέτει επίσης μια γραφική διεπαφή χρήστη που εμφανίζει τα αιτήματα των χρηστών. Αυτό είναι ουσιαστικά ο σταθμός βάσης μέσω του οποίου οι αιτήσεις των χρηστών διαδίδονται με το χρησιμοποιούμενο δίκτυο αισθητήρων.

Από την άλλη πλευρά το λογισμικό πελάτη KSPOT είναι γραμμένο σε NESC και η τοπική μονάδα ενσωματώνει τις μεθόδους πρόσβασης και των top-k λειτουργιών επεξεργασίας σε μια TinyOS συσκευή (mote). Αυτό επομένως είναι το λογισμικό που εκτελείται σε κάθε κόμβο αισθητήρα KSPOT.

Αρχιτεκτονική συστήματος

Το KSPOT διαθέτει δύο βαθμίδες αρχιτεκτονικής που απεικονίζονται στη διπλανή εικόνα.

Η πρώτη βαθμίδα αποτελείται από έναν αριθμό ασύρματων κόμβων αισθητήρων που είναι τοποθετημένοι σε



προκαθορισμένες περιοχές (για παράδειγμα αίθουσες συνεδριάσεων). Οι συσκευές είναι φορτωμένες με το λογισμικό πελάτη KSPOT που τρέχει το TinyOS λειτουργικό σύστημα το οποίο επιτρέπει στους αισθητήρες να μεταδίδουν συνεχώς τα δεδομένα τους στον κόμβο sink (querying node).

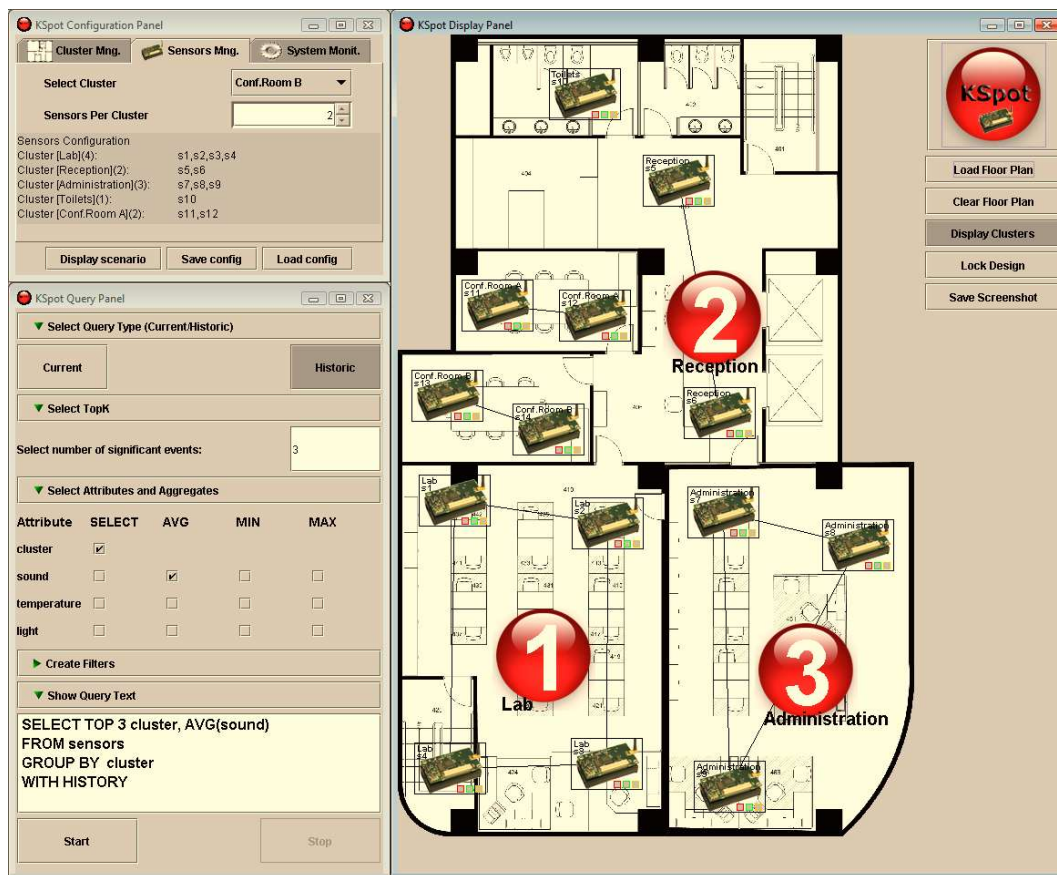
Η διασύνδεση του δικτύου (network interface) του πελάτη KSPOT κατευθύνει τις οδηγίες (δηλαδή, ερωτήσεις και άλλες εντολές) από το KSPOT server στον πελάτη KSPOT. Η τοπική ανάλυση ερωτήματος του πελάτη KSPOT εφαρμόζει εσωτερικά ένα ερώτημα δρομολογητή που βασικά διαδίδει SELECT και GROUP-BY ερωτήματα προς την υπάρχουσα τοπική μηχανή επεξεργασίας επερωτήσεων ενώ τα top-k ερωτήματα κατευθύνονται προς ένα εξειδικευμένο top-k φορέα ερωτήματος που είναι διασυνδεδεμένος άμεσα με το parser ερωτήματος του πελάτη KSPOT.

Η δεύτερη βαθμίδα είναι μια γραφική διεπαφή χρήστη (GUI KSPOT), η οποία χρησιμοποιείται από τους χρήστες για να δημοσιεύουν νέα ερωτήματα και για την εμφάνιση των αποτελεσμάτων των ερωτήσεων με τρόπο που αναδεικνύει τις ιδιότητες κατάταξης του ερωτήματος που εκτελείται. Ειδικότερα, το KSPOT GUI αποτελείται από τρία τμήματα:

1. Η Διαμόρφωση Panel (Conference Panel) (στην πιο κάτω εικόνα, πάνω-αριστερά), το οποίο επιτρέπει στον χρήστη να φορτώσει ένα νέο σενάριο από ένα αρχείο διαμόρφωσης (configuration file) ή να δημιουργήσει ένα νέο σενάριο που μπορεί να αποθηκευτεί σε ένα αρχείο ρυθμίσεων (configuration file). Μέσω αυτού του panel, ο χρήστης μπορεί να καθορίσει ποιοι κόμβοι ανήκουν (οι ομάδες (clustered)) στην ίδια φυσική περιοχή (π.χ., Auditorium, συνεδριακούς χώρους, coffee stations, κλπ.)

2. Το Query Panel (εικόνα, από κάτω προς τα αριστερά), το οποίο επιτρέπει στο χρήστη να καθορίσει συνολικά (AVG, MIN και MAX) και μη συνολικά όπως SQL queries είτε γραφικά ή με το χέρι. Το ερώτημα είναι κατασκευασμένο και συνεπώς προωθείται συνέχεια σε ένα τροποποιημένο παράδειγμα TinyDB, το οποίο μπορεί να επεκταθεί με Top-k φορείς για στιγμιότυπο και ιστορικά ερωτήματα.

3. Display Panel (εικόνα, δεξιά), το οποίο επιτρέπει στον χρήστη να φορτώσει μια αναπαράσταση εικόνας JPG η οποία θα αποτελεί το χάρτη του σεναρίου. Στη συνέχεια, ο χρήστης μπορεί να κάνει drag-and-drop τις συσκευές αισθητήρων για τις αντίστοιχες θέσεις στο χάρτη. Το σύστημα επιτρέπει στο χρήστη να επιλέξει ανάμεσα σε μια ευρεία γκάμα συσκευών αισθητήρα που έρχονται σε διάφορα σχήματα και μεγέθη, προκειμένου να φιλοξενήσει πολυσύχναστες διαμορφώσεις χάρτη. Επίσης, στο Display Panel οι κόμβοι που ανήκουν στο ίδιο cluster χρησιμοποιούν μια μαύρη γραμμή. Επιπλέον, το panel τονίζει την K-υψηλότερη κατάταξη cluster χρησιμοποιώντας μια κόκκινη κουκκίδα, που ονομάζεται KSPOT Bullet, όπου το project δείχνει την κατάταξη του cluster κάθε δεδομένη χρονική στιγμή. Στη συνέχεια, το KSPOT συνεχώς υπολογίζει εκ νέου την κατάταξη έτσι ώστε ο χρήστης να ενημερώνεται στιγμιαία με ένα Bullet σχετικά με την υψηλότερη κατάταξης K απαντήσεις.



Γραφική KSPOT του User Interface (GUI) που επιτρέπει στους χρήστες να διαχειρίζονται την εκτέλεση του Top-K Queries μέσα από μια διαισθητική και δηλωτική διεπαφή χρήστη. Το παραπάνω σενάριο διεξάγει ένα από τα Top-3 ερωτήματα σε μια 14-κόμβους δικτύου αισθητήρων οργανωμένες σε 6 clusters. Στο Display Panel (δεξιά), απεικονίζει τα τρία KSPOT-Bullet για τους τρεις υψηλότερης κατάταξης αισθητήρα clusters[1]

TOP-K επεξεργασία επερωτήσεων στο KSPOT

Η επεξεργασία Top-k επερωτήσεων (Top-k Query Processing) έχει μελετηθεί μέσα σε μια ποικιλία από περιεχόμενα, συμπεριλαμβανομένων των συστημάτων middleware, web accessible βάσεις δεδομένων, επεξεργαστές stream, peer-to-peer και άλλα καταναμημένα συστήματα. Το κύμα του κεντρικού ερωτήματος top- k αλγορίθμων επεξεργασίας έγινε επιτυχία στην πρόσφατη βιβλιογραφία και διανέμονται μέσα στο δίκτυο αλγορίθμων για ιστορικά ερωτήματα (δηλαδή TPUT, Tja[3], TPAT) και για ερωτήματα στιγμιότυπου (δηλαδή MINT, FILA και BABOLS).

Το σύστημα KSPOT ενσωματώνει αλγόριθμους και από τις δύο κατηγορίες ερωτημάτων. Το σκεπτικό πίσω από αυτό το διαχωρισμό είναι το γεγονός ότι δεν υπάρχει ένας καθολικός αλγόριθμος που να έχει βελτιστοποιηθεί και για τις δύο κατηγορίες ερωτημάτων, αλλά υπάρχει μια πισίνα (pool) αλγορίθμων επεξεργασίας δεδομένων για κάθε κατηγορία. Το KSPOT εκμεταλλεύεται έξυπνα αυτή την πισίνα αλγορίθμων εκτελώντας ένα διαφορετικό αλγόριθμο επεξεργασίας ερωτημάτων με βάση το ερώτημα σημασιολογίας. Πιο συγκεκριμένα, εφαρμόζει τον αλγόριθμο TJA[3] για την πραγματοποίηση ιστορικών ερωτημάτων και τον αλγόριθμο MINT για την πραγματοποίηση ερωτημάτων στιγμιότυπου. Οι δύο κατηγορίες και οι στρατηγικές της εκτέλεσής τους περιγράφονται αναλυτικότερα στο επόμενη τμήμα.

A. Snapshot Top-K Query Processing

Ένα ερώτημα στιγμιότυπου αναφέρεται στις τρέχουσες αναγνώσεις της συσκευής αισθητήρα. Η παρακάτω ερώτηση αποτελεί παράδειγμα ενός τέτοιου ερωτήματος:

```
SELECT TOP K roomid, AVERAGE (sound)
FROM sensors
GROUP BY roomid
```

Το σύστημα KSPOT χρησιμοποιεί τον αλγόριθμο MINT για αποτελεσματική αντιμετώπιση αυτής της κατηγορίας των ερωτημάτων με την κατασκευή ενός ιεραρχικού δικτύου απόψεων, όπου οι προγονικοί κόμβοι με ιεραρχική δρομολόγηση διατηρούν ένα υπερσύνολο άποψης των απογόνων τους.

B. Historic Top-K Query Processing

Ένα ιστορικό ερώτημα αναφέρεται στην περίπτωση όπου κάθε αισθητήρας μπορεί να αποθηκεύσει (buffer) τοπικά τις αναγνώσεις του αισθητήρα σε ένα συρόμενο παράθυρο (είτε στην κύρια μνήμη ή σε flash). Όταν το ερώτημα αναφέρεται σε οριζόντιο τεμαχισμό δεδομένων, στην περίπτωση κατά την οποία το κλάδεμα(pruning) μπορεί να διεξαχθεί σε τοπικό επίπεδο, το σύστημα KSPOT επεκτείνει το ερώτημα στιγμιότυπου που περιγράφεται στο παρελθόν με τη διεξαγωγή μιας τοπικής αναζήτησης και φιλτραρίσματος στο αντίστοιχο παράθυρο του ιστορικού πριν από τη διαβίβαση των

αποτελεσμάτων προς τα πάνω στο ιεραρχικό δέντρο ερωτημάτων. Το παρακάτω ερώτημα παρέχει ένα παράδειγμα, πως είναι το ερώτημα:

```
SELECT TOP K roomid, AVERAGE (sound)
FROM sensors
GROUP BY roomid
WITH HISTORY {interval}
```

Αντίθετα, όταν το ερώτημα αναφέρεται σε δεδομένα που είναι τεμαχισμένα κάθετα πάνω από τους n αισθητήρες, στην περίπτωση όπου το κλάδεμα μπορεί να γίνεται αποκλειστικά κατά την ανάγνωση όλων των n αισθητήρων συνδυασμένων, το σύστημα KSPOT υλοποιεί τον αλγόριθμό TJA. Ένα παράδειγμα ενός τέτοιου ερωτήματος είναι το ακόλουθο: *"Βρείτε τις K περιπτώσεις χρόνου με την υψηλότερη μέση θερμοκρασία κατά τη διάρκεια των 3 τελευταίων μηνών."*

Εγκατάσταση KSPOT

1.Εγκατάσταση TinyOS. Οι οδηγίες δίνονται στην προηγούμενη ενότητα.

2. Οδηγίες διαμόρφωσης KSPOT

Για να υποβληθούν ερωτήματα με <GROUP BY cluster> θα πρέπει να προσαρμοστεί ο αριθμός των clusters στο TinyDB/KSpot configuration. Για να γίνει αυτό:

α.Επεξεργαζόμαστε το

```
/opt/tinyos1.x/apps/tools/java/net/tinyos/tinydb/PTopKRoomReader.java
```

και πληκτρολογούμε τον αριθμό των δωματίων π.χ `int roomCount = 3;`

β. Επεξεργαζόμαστε το `/opt/tinyos1.x/tos/lib/TinyDB/TinyDB.h` και εισάγουμε τις ρυθμίσεις

```
#define roomCount R
```

```
uint8_t room_sensors[roomCount]={r1,r2,r3};
```

π.χ. αν σκοπεύετε να χρησιμοποιήσετε 3 clusters με 2 αισθητήρες στο καθένα

```
#define roomCount 3
```

```
uint8_t room_sensors[roomCount]={2,2,2};
```

γ. Εγκατάσταση KSPOT

α. Unzip `kspot.zip` σε ένα προσωρινό κατάλογο `%TEMP%`.

Θα δημιουργηθεί ο κατάλογος `%TEMP%/tinyos-1.x`

- β. Αντιγραφή και αντικατάσταση όλων των αρχείων στο tinyos-1.x folder.
- Θα εγκατασταθούν όλα τα αρχεία που απαιτούνται για το KSPOT (server+client)
- γ. Στο αρχείο /opt/tinyos1.x/tools/java/KSpot/tinydb.conf πρέπει να γίνει αλλαγή στο comm. string:serial@COM5\$57600
- δ. Αλλαγή του COM5 στο COM αριθμό που είναι εγκατεστημένο στη μηχανή

3. KSPOT οδηγίες εκτέλεσης

A. Εκτέλεση με προσομοιωτή

1. Μεταγλώττιση του TinyDB-KSpotclient
 - α. Μεταφερόμαστε στο κατάλογο /opt/tinyos1.x/apps/TinyDBApp/
 - β. Εκτελούμε **\$ make -k -f MakePC pc**
2. Run TinyDB-KSPOT client
 - α. Εκτελούμε `DBG=<usr1,usr2,...> ./build/pc/main.exe N`
 N=αριθμός αισθητήρων που έχουν δηλωθεί στο TinyDB.h
3. Μεταγλώττιση του TinyDB
 - α. Μεταφερόμαστε στο κατάλογο /opt/tinyos1.x/tools/java/net/tinyos/tinydb και εκτελούμε
 - β. **\$ make clean**
 - γ. **\$ make -f MakePC**
4. Run KSPOT in simulation mode
 - α. Μεταφερόμαστε στο κατάλογο %TINYOS%\tools\java\KSPOT και εκτελούμε
 - β. **\$ make sim**

B. Εκτέλεση σε πραγματικό περιβάλλον

1. Μεταγλώττιση του TinyDB-KSpotclient
 - α. Μεταφερόμαστε στο κατάλογο /opt/tinyos1.x/apps/TinyDBApp και εκτελούμε
 - β. **\$ make micaz**
2. Run TinyDB client
 - α. Εγκατάσταση του TinyDB-KSPOT client σε κάθε αισθητήρα
 Από το command line εκτελούμε:
\$ make micaz
\$ make micaz reinstall.<SENSOR_ID> mib510,/dev/ttyS<COM_ID-1>
3. Μεταγλώττιση του TinyDB

α. Μεταφερόμαστε στο κατάλογο `/opt/tinyos1.x/tools/java/net/tinyos/tinydb` και εκτελούμε

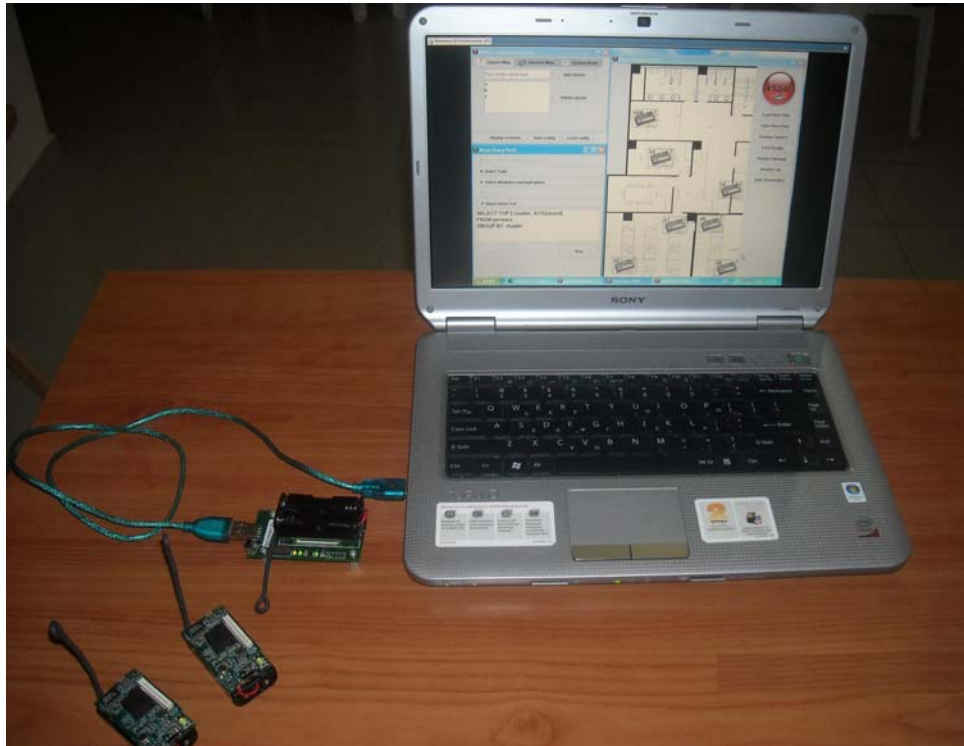
β. **\$ make clean**

γ. **\$ make**

4. Run KSPOT in runtime mode

α. Μεταφερόμαστε στο κατάλογο `/opt/tinyos1.x/tools/java/KSpot` και εκτελούμε

β. **\$ make**



Βιβλιογραφία Παραρτήματος Γ

[1] Andreou, D. Zeinalipour-Yazti, M. Vassiliadou, P. K. Chrysanthis, G. Samaras "KSpot: Effectively Monitoring the K Most Important Events in a Wireless Sensor Network", 25th International Conference on Data Engineering March (ICDE'09) (demo), Shanghai, China, May 29 - April 4, 2009

Παράρτημα Δ

Κώδικας MINT

```
includes Aggregates;

module PTopkRoomM{
provides{
  interface Aggregate;
  command uint8_t getRoomIDfromNodeID();
  command void printState(char *data);
  command void printPreviousState();
  command void sortStateDesc(char *data);
  command uint8_t getTopkAttribute(ParamVals *v);
  command uint8_t getTopkValue(ParamVals *v);
  command void generateUpperBounds(char *data, uint8_t attr);
  command void createKCoveredBoundSet(char *data, uint8_t kVal, uint16_t kHLB);
  command uint8_t IsSameState(PTopkRoomData* s1, PTopkRoomData* s2);

  command uint32_t getCount(PTopkRoomData* s, uint8_t bitindex);
  command void setCount(PTopkRoomData* s, uint8_t bitindex, uint32_t value);
  command uint32_t getRoom(PTopkRoomData* s, uint8_t bitindex);
  command void setRoom(PTopkRoomData* s, uint8_t bitindex, uint32_t value);
}

implementation{
  command uint32_t getCount(PTopkRoomData* s, uint8_t bitindex){
    uint32_t result=s->count;
    result = result<<(32-((bitindex+1)*3));
    result = result>>29;
    return result;
  }

  command uint32_t getRoom(PTopkRoomData* s, uint8_t bitindex){
    uint32_t result=s->room;
    result = result<<(32-((bitindex+1)*3));
    result = result>>29;
    return result;
  }

  command void setCount(PTopkRoomData* s, uint8_t bitindex, uint32_t value){
    uint32_t right,left,result;
    left=0;
    right=0;
    result=0;
```

```

        right = s->count;
        if(bitindex>0){
            right<<=(32-(bitindex*3));
            right>>=(32-(bitindex*3));
        }
        else{
            right=0;
        }

        left = s->count;
        left>>=((bitindex+1)*3);
        left<<=((bitindex+1)*3);

        result = (value<<(bitindex*3))+left+right;
        s->count = result;
    }

command void setRoom(PTopkRoomData* s, uint8_t bitindex, uint32_t value){
    uint32_t right,left,result;
    left=0;
    right=0;
    result=0;

        right = s->room;
        if(bitindex>0){
            right<<=(32-(bitindex*3));
            right>>=(32-(bitindex*3));
        }
        else{
            right=0;
        }

        left = s->room;
        left>>=((bitindex+1)*3);
        left<<=((bitindex+1)*3);

        result = (value<<(bitindex*3))+left+right;
        s->room = result;
    }

```

```

/*
* PTopkRoomM Aggregate - operation outline
* 1. init:      initializes the state of the aggregate
* 2. update:    is called when a new value is generated
*               locally at the sensor
* 3. merge:     is called when new results arrive from the
*               child nodes of the current sensor
* 4. hasdata:   is called when the results are ready
*               to be output
*               hasData performs the following operations
*               A. Retrieve the current state
*               B. Sort the current state in DESC order
*               C. Retrieve the topk and attribute parameters
*               D. Generate upper bounds
*               E. Find the k highest lower bound
*               F. Eliminate Tuples not in k-Covered-Bound set
*               G. Finalize state
*               H. Check if generated state is the same as
*                  as the previous one. If it is do not send
*                  anything.
*               I. Copy current state to the previous one
*/

```

```

/*
* Declaration of upper and lower limits for
* each attribute. column(0) specifies the min
* limit and column(1) specifies the max limit.
* Each row index corresponds to a specific
* attribute used by the query. Below is the
* list of attributes for each index
* 0 - light
* 1 - temp
* 2 - rawmic
*/
uint16_t limits[3][2] = {{0,1000},{0,600},{0,800}};

```

```

/*
* Table sum_aux will hold the upper bounds for each
* epoch
*/
uint16_t sumUpperBound[roomCount];

```

```

/*
* prevState stores the previous state of the sensor
*/

```

```

PTopkRoomData prevState;
/*
 * Return the size of the state
 * struct PTopkRoomData (declared in Aggregates.h)
 */
command uint16_t Aggregate.stateSize(ParamList *params, ParamVals
*paramValues) {return sizeof(PTopkRoomData);}

/*
 * No optimization details required
 */
command AggregateProperties Aggregate.getProperties() {return 0;}

/* getRoomIDfromNodeID returns the roomID of the current node
 * This information is retrieved from roomCount and
 * room_sensors array (TinyDB.h).
 * Ideally this information should be stored using a create
 * storage buffer query
 */
command uint8_t getRoomIDfromNodeID(){
    uint8_t i, result=0, totalSensors=0;

    if(TOS_LOCAL_ADDRESS!=0){
        for(i=0; i<roomCount; i++){
            totalSensors+=room_sensors[i];
            if(TOS_LOCAL_ADDRESS<=totalSensors){
                result=i;
                break;
            }
        }
    }

    return result;
}

/*
 * Retrieves the Top [k] parameter from the query
 */
command uint8_t getTopkValue(ParamVals *v) {
    return *(uint8_t *) (v->paramDataPtr[0]);
}

```

```

/*
 * Retrieves the attribute (light,temp,rawmic) from
 * the query. It is used in conjunction with uint16_t limits[3][2]
 * to generate the upper bounds
 */
command uint8_t getTopkAttribute(ParamVals *v) {
    return *(uint8_t *) (v->paramDataPtr[1]);
}

/*
 * Prints the (char *data) state of the sensor
 */
command void printState(char *data){
    uint8_t i, roomID;
    PTopkRoomData *dest = (PTopkRoomData *) data;
    roomID = call getRoomIDfromNodeID();
    dbg(DBG_TEMP, "Printing State of NodeID=%d\tRoomID=%d\tVals:%d\n",
TOS_LOCAL_ADDRESS, roomID, dest->vals);
    dbg(DBG_TEMP, "COUNT [%d]\tROOM[%d]\n", dest->count, dest->room);
    for(i=0; i<dest->vals; i++){
        dbg(    DBG_TEMP,
            "sum:%d\tcount:%d\troom:%d\tsum':%d\n",
            dest->sum[i],
            call getCount(dest, i),
            call getRoom(dest, i),
            sumUpperBound[i]);
    }
}

/*
 * Prints the previous state of the sensor
 */
command void printPreviousState(){
    uint8_t i;
    dbg(DBG_TEMP, "Printing Previous State of NodeID=%d\n",
TOS_LOCAL_ADDRESS);
    for(i=0; i<prevState.vals; i++){
        dbg(    DBG_TEMP,
            "sum:%d\tcount:%d\troom:%d\n",
            prevState.sum[i],
            call getCount(&prevState, i),
            call getRoom(&prevState,i));
    }
}

```

```

/*
 * Sorts the (char *data) state of the sensor in desc order
 */
command void sortStateDesc(char *data){
    uint8_t i, j;
    uint16_t sum;
    uint32_t count,room;

    PTopkRoomData *dest = (PTopkRoomData *) data;
    //Do a bubble sort
    for(i=0; i<dest->vals-1; i++){
        for(j=0; j<dest->vals-1; j++){
            if( (dest->sum[j]/(call getCount(dest,j))) < (dest-
>sum[j+1]/(call getCount(dest,(j+1)))) ) {
                //swap values in posistion j, (j+1)
                sum = dest->sum[j];
                count = call getCount(dest,j);
                room = call getRoom(dest,j);

                dest->sum[j] = dest->sum[j+1];

                call setCount(dest,j, (call getCount(dest, (j+1)))) );
                call setRoom(dest,j, (call getRoom(dest, (j+1)))) );

                dest->sum[j+1] = sum;

                call setCount(dest, (j+1), count);
                call setRoom(dest, (j+1), room);
            }
        }
    }
}

/*
 * Generates the [sum] upper bounds utilized
 * by the MINT algorithm
 */
command void generateUpperBounds(char *data, uint8_t attr){
    uint8_t i;
    PTopkRoomData *dest = (PTopkRoomData *) data;
    for(i=0; i<dest->vals; i++){
        /*
        if(TOS_LOCAL_ADDRESS==0) {dbg(DBG_TEMP,
        "Room:%d\tCount:%d\tCount':%d\tCount(-):%d\tLimit:%d\n",
        dest->room[i],
        dest->count[i],
        room_sensors[dest->room[i]],
        (room_sensors[dest->room[i]] - dest->count[i]),

```

```

limits[attr][1]);}

*/
sumUpperBound[i] = ( (room_sensors[(call getRoom(dest,i))] - (call getCount(dest,i)))
* limits[attr][1]) + dest->sum[i];
}
}

/*
* This is the MINT algorithm that generates the k-covered
* bound set by eliminating tuples that do not satisfy the
* k highest lower bound constraint.
* The kVal and k-Highest-Lower-Bound (kHLB) are passed as
* arguments (calculated at hasData).
* The sumUpperBound table that stores the upperbounds is
* calculated in a previous step.
* The k-Highest-Lower-Bound is an average value sum/count.
* The operation of this method eliminates all tuples in the
* following way:
* Loop through all tuples following the ones under the kHLB.
* (Note that the tuples prior to kHLB all satisfy the
* constraint because the table is sorted.
* If a tuple satisfies the constraint then simply move to
* the next one.
* If a tuple does not satisfy the constraint (i.e., its
* upperbound average (calculated by dividing the sumUpperBound
* by the total number of sensors in the room) is below kHLB
* then the tuple is eliminated by shifting all next tuples one
* position up and decreasing the number of values.
*/
command void createKCoveredBoundSet(char *data, uint8_t kVal, uint16_t kHLB){
    uint8_t i, j, destVals, count;
    char level[4];
    FILE *fp;
    PTopkRoomData *dest = (PTopkRoomData *) data;

    sprintf(level, "%d.dat", dest->hopcount);
    fp=fopen(level, "a");

    //printf("node id %d hop %d\n", TOS_LOCAL_ADDRESS, dest->hopcount);
    //dest->vals cannot be used in the following for{} because
    //it is volatile because (it will change
    //according to which tuples are going to be eliminated)
    destVals = dest->vals;

    //Eliminate all tuples not in the k-Covered-Bound set
    //Start from position [kVal] since all tuples prior to
    //this position satisfy the >=kHLB since the table is
    //sorted in descending order
    count=0;

```



```

        for(i=kVal; i<destVals; i++){
        if( (sumUpperBound[i]/room_sensors[(call getRoom(dest,i))]) < kHLB ){
            //just shift all values in the table
            //one position upwards EXCEPT if we
            //are at the last position
            //dbg(DBG_TEMP, "ELIMINATED TUPLE s:%d\tr:%d\tr:%d\n",
                //dest->sum[i], dest->count[i],dest->room[i]);

            //if we are at the last position then just
            //decreasing the number of values does the trick
            dest->vals--;
            count+=1;

            //if we are not at the last position then
            //shift all values one position up
            if(i!=(destVals-1)){
                for(j=i; j<destVals-1; j++){
                    dest->sum[j] = dest->sum[j+1];
                    call setCount(dest, j, (call getCount(dest, (j+1))) );
                    call setRoom(dest, j, (call getRoom(dest, (j+1))) );
                    sumUpperBound[j] = sumUpperBound[j+1];
                }
            }
        }
    }

    fprintf(fp, "%d: hop %d destVals %d count %d\n",TOS_LOCAL_ADDRESS,dest-
    >hopcount,destVals,count);

    fclose(fp);
}

/*
 * Checks of two states are the same.
 * Note that two states may hold the same values but in different
 * order. If this holds, they still represent the same state.
 */
command uint8_t IsSameState(PTopkRoomData* s1, PTopkRoomData* s2){
    uint8_t i, j, result = 1;
    bool found;

    if(s1 && s2){
        if(s1->vals == s2->vals){
            dbg(DBG_TEMP, "[%d] -- %d -- %d\n",TOS_LOCAL_ADDRESS,s1-
            >vals,s2->vals);
            for(i=0; i<s1->vals; i++){
                found = TRUE;
                for(j=0; j<s2->vals; j++){
                    if( (s1->sum[i]==s2->sum[j]) &&
                        ( (call getCount(s1,i))==(call getCount(s2,j)) ) &&

```

```

        ( (call getRoom(s1,i))==(call getRoom(s2,j)) ) ){
        found = FALSE;
        break;
    }
}
if(!found){
    result=0;
    break;
}
}
}
}

return result;
}

/*
 * Initialize aggregate
 * Sum and count of each room is set to 0 vals is set to 0
 */
command result_t Aggregate.init(char *data, ParamList *params, ParamVals
*paramValues, bool isFirstTime){
    uint8_t i;

    PTopkRoomData *dest = (PTopkRoomData *) data;

    //dbg(DBG_TEMP, "%d INIT\n", TOS_LOCAL_ADDRESS);
    if(dest){
        for(i=0; i<roomCount; i++){
            dest->sum[i]=0;
        }
        dest->count=0;
        dest->room=0;
        dest->vals=0;
        dest->hopcount=0;
        dest->junk2=0;
        dest->junk3=0;
    }

    return SUCCESS;
}

```

```

/*
 * Update aggregate result
 * update is called with the current aggregate state (char *destdata)
 * and the current sensor reading (char* value) as arguments
 */
command result_t Aggregate.update(char *destdata, char* value, ParamList
*params, ParamVals *paramValues) {
    PTopkRoomData *dest;
    uint8_t roomID;
    uint16_t val;

    //dbg(DBG_TEMP, "%d UPDATE\n", TOS_LOCAL_ADDRESS);

    //current state (destdata)
    dest = (PTopkRoomData *) destdata;

    //retrieve current reading
    val = *(uint16_t *) value;
    //check if new values are received ok
    //dbg(DBG_TEMP, "NodeID=%d Value=%d\n",
TOS_LOCAL_ADDRESS, val);

    //update the current room's sum and count
    dest->sum[0] += val;
    //dest->count[0]++;
    call setCount(dest, 0, 1);
    //get roomID from TOS_LOCAL_ADDRESS
    roomID = call getRoomIDfromNodeID();
    //dest->room[0]=roomID;
    call setRoom(dest,0,roomID);
    dest->vals = 1;

    //call printState(destdata);

    return SUCCESS;
}

/*
 * Merge aggregate result
 * merge is called with two aggregate states as arguments
 * Its task is to update the node's aggregate state (char *destdata)
 * with the data from another node (char *mergedata)
 * For each room update the sum and count
 */
command result_t Aggregate.merge(char *destdata, char *mergedata, ParamList
*params, ParamVals *paramValues){

```

```

uint8_t i, j;
bool foundRoom;

//current state (destdata)
PTopkRoomData *dest = (PTopkRoomData *) destdata;

//new data (mergedata)
PTopkRoomData *merge = (PTopkRoomData *) mergedata;

//Check the hop count of the merge message and set your own (dest)
if(merge->hopcount>=dest->hopcount) {
    dest->hopcount = merge->hopcount+1;
}

//dbg(DBG_TEMP, "%d MERGE\n", TOS_LOCAL_ADDRESS);

/*if(TOS_LOCAL_ADDRESS==0){
    dbg(DBG_TEMP, "\n\nBEFORE MERGE - CURRENT STATE\n");
    call printState(destdata);
    dbg(DBG_TEMP, "BEFORE MERGE - CHILD STATE\n");
    call printState(mergedata);
}*/
for(i=0; i<merge->vals; i++){
    //Search if room in merge already exists in dest
    foundRoom = FALSE;
    for(j=0; j<dest->vals; j++){
        if( (call getRoom(merge,i))==(call getRoom(dest,j))
    ){foundRoom=TRUE;break;}
    }
    if(foundRoom){
        dest->sum[j] += merge->sum[i];
        call setCount( dest, j, ( (call getCount(dest,j))+(call
getCount(merge,i)) ) );
    }
    else{
        dest->sum[dest->vals] = merge->sum[i];

        call setCount(dest, dest->vals, (call getCount(merge,i)) );

        call setRoom(dest, dest->vals, (call getRoom(merge,i)) );

        dest->vals++;
    }
}
/*if(TOS_LOCAL_ADDRESS==0){
    dbg(DBG_TEMP, "AFTER MERGE - CURRENT STATE\n");
    call printState(destdata);
}*/
return SUCCESS;

```

```

/*
 * hasData always returns true since this is not a temporal aggregate.
 */
command bool Aggregate.hasData(char *data, ParamList *params, ParamVals
*paramValues) {
    uint8_t i,kVal, attrVal;
    uint16_t kHighestLowerBound;

    //A. Retrieve the current state
    PTopkRoomData *dest = (PTopkRoomData *) data;
    //dbg(DBG_TEMP, "node id %d hop
%d\n",TOS_LOCAL_ADDRESS,dest->hopcount);
    //B. Sort the current state in DESC order
    call sortStateDesc(data);

    /*if(TOS_LOCAL_ADDRESS==0){
        dbg(DBG_TEMP, "CURRENT STATE\n");
        call printState(data);
    }*/
    //dbg(DBG_TEMP, "[%d] CURRENT
STATE\n",TOS_LOCAL_ADDRESS);
    //call printState(data);

    //C. Retrieve the topk and attribute parameters
    // on topk and attribute parameters
    //C1. Retrieve the topk value
    kVal = call getTopkValue(paramValues);
    //C2. Retrieve the attribute value
    attrVal = call getTopkAttribute(paramValues);
    //if(TOS_LOCAL_ADDRESS==0){dbg(DBG_TEMP, "\n\nkVal:%d\t
attrVal:%d\n\n", kVal, attrVal);}

    //NOTICE: THE FOLLOWING STEPS ARE ONLY PERFORMED
    //IF THE NUMBER OF TUPLES ARE GREATER THAN THE
    //TOP-K PARAMETER
    //G. Finalize state if sink

    if(TOS_LOCAL_ADDRESS==0) {dest->vals = kVal;}
    else if(dest->vals>kVal){
        //dbg(DBG_TEMP, "ENTERING ELIMINATION STEP\n");
        //D. Generate upper bounds
        call generateUpperBounds(data, attrVal);
        //dbg(DBG_TEMP, "%d HASDATA 6\n",
TOS_LOCAL_ADDRESS);

        //E. Find the k highest lower bound
        //Since the state is sorted then the k highest

```

```

        //lower bound is the sum value at position k
        kHighestLowerBound = dest->sum[kVal-1]/(call
getCount(dest,(kVal-1)));
        //dbg(DBG_TEMP, "kHighestLowerBound=%d\n",
kHighestLowerBound);

        //F. Eliminate Tuples not in k-Covered-Bound set
        call createKCoveredBoundSet(data, kVal,
kHighestLowerBound);
    }

    //H. Check if generated state is the same as
    //as the previous one. If it is, do not send
    //anything.
    dest->sameAsPrevious = call IsSameState(dest, &prevState);

    /*if(TOS_LOCAL_ADDRESS==0){
        call printState(data);
    }*/

    //I. Copy current state to previous state
    prevState.vals = dest->vals;
    prevState.count = dest->count;
    prevState.room = dest->room;
    for(i=0; i<dest->vals; i++){
        prevState.sum[i] = dest->sum[i];
    }

    if(dest->sameAsPrevious==0)
    {
        //dbg(DBG_TEMP, "[%d] is Same As
Previous\n",TOS_LOCAL_ADDRESS);

        return FALSE;

        return TRUE;
    }

    /*
    * Finalize is not currently called in TinyDB
    * The finalization is performed in the application layer
    * in PTopkRoomReader.java
    */
    command TinyDBError Aggregate.finalize(char *data, char *result_buf,
ParamList *params, ParamVals *paramValues) {return err_NoError;}
}

```